Myriad: A Distributed Machine Vision Application Framework

PhD Thesis

Luke Thomas Chatburn, BSc

ProQuest LLC
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106-1346

Abstract Of Thesis

Candidate's Surname: Chatburn
Candidate's Forenames: Luke Thomas
Candidate for the Degree of: PhD

Institution/College at which study pursued:
University of Wales, Cardiff

Full title of thesis:

Myriad: A distributed Machine Vision application framework

Abstract:

This thesis examines the potential for the application of distributed computing frameworks to industrial and also lightweight consumer-level Machine Vision (MV) applications.

Traditional, stand-alone MV systems have many benefits in well-defined, tightly-controlled industrial settings, but expose limitations in interactive, de-localised and small-task applications that seek to utilise vision techniques. In these situations, single-computer solutions fail to suffice and greater flexibility in terms of system construction, interactivity and localisation are required. Network-connected and distributed vision systems are proposed as a remedy to these problems, providing dynamic, componentised systems that may optionally be independent of location, or take advantage of networked computing tools and techniques, such as web servers, databases, proxies, wireless networking, secure connectivity, distributed computing clusters, web services and load balancing.

The thesis discusses a system named Myriad, a distributed computing framework for Machine Vision applications. Myriad is composed components, such as image processing engines and equipment controllers, which behave as enhanced web servers and communicate using simple HTTP requests. The roles of HTTP-based distributed computing servers in simplifying rapid development of networked applications and integrating those applications with existing networked tools and business processes are explored. Prototypes of Myriad components, written in Java, along with supporting PHP, Perl and Prolog scripts and user interfaces in C#, Java, VB and C++/Qt are examined.

Each component includes a scripting language named MCS, enabling remote clients (or other Myriad components) to issue single commands or execute sequences of commands locally to the component in a sustained session. The advantages of server-side scripting in this manner for distributed computing tasks are outlined with emphasis on Machine Vision applications, as a means to overcome network connection issues and address problems where consistent processing is required. Furthermore, the opportunities to utilise scripting to form complex distributed computing network topologies and fully-autonomous federated networked applications are described, and examples given on how to achieve functionality such as clusters of image processing nodes.

Through the medium of experimentation involving the remote control of a model train set, cameras and lights, the ability of Myriad to perform traditional roles of fixed, stand-alone Machine Vision systems is supported, along with discussion of opportunities to incorporate these elements into network-based dynamic collaborative inspection applications. In an example of 2D packing of remotely-acquired shapes, distributed computing extensions to Machine Vision tasks are explored, along with integration into larger business processes.

Finally, the thesis examines the use of Machine Vision techniques and Myriad components to construct distributed computing applications with the addition of vision capabilities, leading to a new class of image-data-driven applications that exploit mobile computing and Pervasive Computing trends.

TABLE OF CONTENTS

Table of Figures

1 Introduction

1.1 Definition of Machine Vision

One core definition of Machine Vision was described by the Automated Vision Association [AVA]:

> *"The use of devices for optical, non-contact sensing to automatically receive and interpret an image of a real scene in order to obtain information and/or control machines or processes"*

Machine Vision is a broad discipline, encompassing a range of subjects that straddle engineering and computing; but at the core of MV, this definition is central.

1.2 Introducing the Problem

Machine Vision as an industry is growing rapidly, most especially in markets such as quality control systems and monitoring of industrial processes [OCO]. However, there are also extensive opportunities for diversification into other applications. Requirements of security, forensic science, robotics, general purpose monitoring and other areas of interest continually increase with the accelerating proliferation of mobile computing devices, networking, wireless systems and data-aware devices. As this process ensues, the need for Vision-based solutions increases, while the number of Vision Engineers and those experienced in Image Processing and system integration increases at a far reduced rate.

As a consequence, present workers in the field of Machine Vision require tools and methods to:

- Train more Vision engineers
- Enable existing Vision Engineers to assess situations and problems more quickly
- Intuitively examine products more quickly
- Develop prototype solutions and migrate to final deployment easily

One of the most common requirements of Machine Vision solutions for industry is an understanding of the problem, the objects involved in the inspection and the environment

within which the inspection will take place. For the products, objects and equipment being inspected, samples can typically be removed from the industrial site and examined in a lab (it is often the case that businesses supply a researcher with a palette of the appropriate goods to test); for the environment itself, however, the researching developer needs to acquire a detailed understanding of the situation in the factory, such as:

- Lighting levels (ambient, directional)
- Personnel movements and access
- Dust
- Noise and vibration levels
- Type of lighting (fluorescent strobing must be considered, for example)
- Physical confinements in the deployment area

While simpler tools can assist in the rapid development of an MV solution, such as interactive Image Processors, site visits become a constantly time- and money-consuming feature of MV solution creation.

The importance of the vision engineer being able to experience the site and the problem must not be underestimated for the discontinuity between theory and the physical scene is the difference between Computer Vision and Machine Vision. This difference is explored later with the description and example of what MV adds to the Computer Vision process and the additional concerns that it introduces to problem solution. For emphasis, the following anecdote is apt:

A vision engineer was commissioned by a large food processing company, who were producing bags of peanuts. Their vision problem was that the nuts entering their factory inevitably were not always perfect and some were covered in a fungus. Unfortunately, this need a human being to examine every single nut; a time-consuming process that was not efficient as it added a high per-nut cost which translated into a substantially higher price for a bag including hundreds of peanuts. The vision engineer, working in a laboratory was provided with a number of boxes of samples, both of good and fungal peanuts and set about trying to develop a system which would visually detect the bad peanuts. Once the system was ready, the engineer organised a demonstration of the system to the company. The system utterly failed to detect the fungus.

The reason for the failure was that the samples of bad nuts were not representative of what entered the production system. The mould was a white colour and the engineer had developed a solution that was capable of identifying that colour of fungus. It turned out that the fungus on the production line peanuts was actually green in colour and changed to white after a few days (which was the time between the peanuts being picked, boxed and sent to the engineer). The matter was resolved reasonably through changes to the detection algorithms and lighting, but if the engineer had been able to personally see the production line, he would have realised that this change takes place.

The engineer in the anecdote was fortunate, in that his system could be easily changed to correct for the difference in colour of the fungus. In other cases, however, this type of change is not always possible and causes the failure of an entire project. Taking the discussion of problem constraint in J. R. Park's seminal paper [PARK], industry is ripe for the use of Vision technology, because the problem and environment may be understood and subsequently controlled, but this relies on a good understanding of the environment in the first instance.

As a consequence, the step of the Vision Engineer visiting the site of the problem is essential to the Machine Vision process and must not be removed, lest the project fail through the application of bad assumptions.

As other areas of the development process become streamlined, this physical step (especially considering long-distance travel) becomes a larger element of the time consumed in proportion to other factors, while being an essential element that cannot be removed or skipped.

1.3 A Solution: Networked Vision Systems

In order to alleviate the time-consuming necessity for a Vision Engineer to visit a site to create an understanding of the environment and the problem, and to facilitate testing of vision system prototypes, a remote control and administration system needs to be created. The development of network-connected camera and equipment systems, presents a ready basis for remote control vision systems with local or global range. The vision engineer has the same degree of control if he/she is on another continent to a site, as if they were in an adjacent room or building. Less obviously, remote access to an environment can prove more effective in

diagnosing a situation, since the vision engineer can observe, examine and test the problem scene from the comfort of their conventional workplace. This provides increased access to tools, knowledge bases, articles, books and information resources that would not be available if the vision engineer was in the midst of a factory floor.

Simply planting a camera in the midst of a factory is insufficient to achieve a good understanding of the scene, however. A single camera image does not:

- Give a complete view of the scene
- Allow elements to be focused upon and examined at close range
- Give an impression of the level of dust/floating debris
- Allow the engineer to explore the scene, equipment, process and products simultaneously, to understand the interaction of scene elements
- Enable the engineer to monitor the scene throughout the day, see variations in lighting and attempt to compensate through the activation of temporary lighting rigs.

The idea of distributing Machine Vision systems across a network is, of itself, not a new idea and indeed, it becomes the next logical step beyond adding basic image processing operations to embedded chipsets onboard networked camera devices. The first discussion of networked MV systems occurred in the 1991 paper *"Distributed Industrial Vision Systems"* by P. Neve. In that paper, Neve defined a network-distributed Vision system as a collection of 'intelligent' cameras with onboard framegrabbers and image processing capabilities which operate on a network and are controlled and monitored by a single controller, connected to the same local area network. The system described in that paper was designed to meet the objective of distributing image processing tasks around a network to the cameras involved in a problem, with the intention of pre-processing images to perform IP operations on DSPs, which were far more effective for that type of operation than personal computers contemporary to the paper. Forcing the host computer to perform the image processing tasks for a large number of cameras would be exceedingly slow, while each camera could process the image in the same manner before transmission in less time than the controller.

The UK-based company, Image Inspection Ltd., for whom Neve worked continued with the idea of intelligent cameras, marketing their products with the somewhat unimaginative title of *'Intelligent Camera'*. This network layout of embedding IP commands in camera systems was highly effective, and derivatives of this approach are now widely used in products from

companies such as Cognex and Axis, or in consumer electronics such as hand-held video recorders and in-built cellular telephone cameras.

In his paper, Neve also takes the time to outline the advantages of networked MV systems:

- Minimal equipment needs to be installed next to the production line, saving cost and space, exposing less equipment to hazardous conditions
- The complete system can be monitored using just one network-connected computer
- The monitoring computer can be freely located away from the production line, in a convenient place
- Wiring between the camera and IP system is minimised, reducing signal noise
- The system can be expanded at any time with the addition of a new camera to the network
- If a camera unit fails, the remainder of the system continues to operate
- If the controller fails, the system can continue to act autonomously

Neve also mentions a number of advantages that are unique to his system, such as reduced connection distances between the IPE and hardware interfaces. In most modern digital systems, interference is less of an issue due to integrity control and resending provisions included in protocols such as TCP/IP.

In the concept outlined in the paper, and subsequent papers (such as *"Distributed Vision System: A Perceptual Information Infrastructure for Robot Navigation"* by Ishiguro in 1997 and *"A Fault-Tolerant Distributed Vision System Architecture for Object Tracking in a Smart Room"* by Karuppiah et al. in 2001), the definition is implied that networked and distributed MV systems are synonymous with networked and distributed image processing systems. As a consequence of this, applications of distributed Machine Vision have been primarily limited to:

- Motion tracking
- Security
- Fixed, encapsulated quality control camera systems

1.4 Distributed Vision Application Frameworks

This thesis seeks to first and foremost to examine the opportunities for distributed computing methods for Machine Vision tasks. Where Neve saw the benefits of connecting multiple uniform cameras to a network, to support distributed acquisition and pre-processing of images, this thesis expounds the benefits of extending this concept to include the entirety of MV tools:

- Equipment/lighting controllers
- Intelligent controllers
- Knowledge bases & Intelligent advisory systems
- Networked information stores (databases, web servers, LDAP servers)
- Mobile computing devices

While this extension may seem a simple logically successor to Neve's outline, it is also radically different and more complex, in that it no longer deals with a homogeneous network of cameras with a controller. Including a diverse range of components poses questions as to how these elements communicate with each other and operators. Do networks need to be 1 Controller to Many Single Devices? Are non-hierarchical networks supported? Multiple controllers? Neve never envisioning use with WANs or the Internet; how should networked vision systems operate at global scales? How do networked vision systems integrate with businesses that are increasingly completely networked? Can diverse networked vision systems address new, original vision problems that fixed systems cannot?

Allowing any element of an MV system to be distributed provides the ability to:

- Take advantage of other networked tools and services
- Add MV techniques and facilities easily to any network-connected computing device or task
- Design flexible MV systems that can add and remove components as they operate, compensating automatically as the problem changes
- Support locational independence – If a problem needs a component, processor or information that is not available locally, it can find one from anywhere around the world
- Control an MV system from any location connected to the network; also anywhere around the world for (S)WANs or the Internet

This thesis aims to contribute to the field of Machine Vision through promoting consideration of how distributed computing might be used to:

- Expand existing typical Machine Vision techniques to include distributed components, providing dynamically-allocated resources and taking advantage of networked resources, such as remote knowledge bases or image processing engine clusters (see Train example).
- Enable distributed cameras, processing nodes and controllers to be utilised to provide Machine Vision solutions where localised conditions inhibit access (e.g. controlling cameras and equipment remotely in hazardous environments) (see Linpacker example).
- Incorporate Machine Vision systems into larger networked systems, such as manufacturing control and monitoring systems (see Car Park example) or e-commerce systems.
- Provide opportunities to create wide-area Machine Vision applications, combining the functionality of multiple smaller Machine Vision systems (e.g. managing multiple production lines (see Car Park example).

Most importantly, this thesis seeks to extend Machine Vision discourse to include less rigid vision problems than the traditional factory fixed-system inspection tasks. There exist a range of problems that can benefit from Machine Vision tools and techniques, such as the Non-Hospitalised Dermalogical Testing example, where small or flexible use of vision can actively open up new tasks and solutions. With the advent of Pervasive Computing, wireless hotspots, cellular data networks and Metropolitan Area Wireless Networks, combined with small form-factor computing devices and integrated cameras, the opportunities for acquiring imagery are great, and the necessity for the use of Machine Vision to support that acquisition and use those images as operable networked data rather than just snapshot photographs is great.

Conversely, this thesis also intends to contribute to the field of distributed computing by proposing the use of Machine Vision with distributed elements. While considering how Machine Vision might be affected by distributing computing components, the reverse situation is also critical; in an increasingly networked or Pervasive Computing environment, how might visual data and vision techniques affect established tasks and offer opportunities for the development of augmented networked applications based around the acquisition of visual data?

Through the Myriad system, this thesis details:

- How vision tools may be integrated into web-based distributed computing systems
- The means of bridging distributed computing systems with problems that require highly physically-localised elements (e.g. cameras, equipment controllers)
- How scripted distributed computing components may be used to overcome network difficulties that pose obstacles to the use of networked elements in situations where consistent control is required

1.5 Introducing Myriad

Myriad is a framework for distributed vision systems, based around internet-connected servers that perform specific functions, such as Image Processing Engines, equipment controllers, web servers, databases and intelligent controllers.

Myriad components are based around a web server template, communicating using HTTP requests and responses.

HTTP-based components support widely-used and well-understood tools such as firewalls and proxy servers, integrating with existing networked systems which have been built around web servers for the past decade. Not only may Myriad components utilise existing web technologies, but the ubiquitous nature of the HTTP interface enables the integration of machine vision tools into networked tasks without reprogramming of interfaces or creation of protocol handlers.

The objectives of Myriad are:

- Locational freedom – Operator and components may be located anywhere, connected to the internet, without physical limitation
- Implicit integration of existing web tools (firewalls, web servers, proxies, VPNs) and network features (multiplexing, load-balancing, secure communication, rerouting to redundant servers on failure)
- Access for low-performance and mobile devices (PDAs, cellular telephones, embedded systems)
- Simple interface for small task solution, extended with scripting for complex tasks

- Support for all modern programming and scripting languages which include HTTP libraries
- Granting machine vision tools access to all networked information systems and enabling those services to access MV tools to manipulate their data, acquire visual data and provide physical output
- Dynamic connection of components at runtime to accommodate problem and objective changes

In this thesis, Myriad will be further explained, along with details of component technologies, how Myriad components may be constructed, how the MCS scripting language may be used to enhance networked vision systems, and how to interact with Myriad components and systems from a variety of modern programming languages.

Finally, it shall be demonstrated in a part-by-part manner that Myriad enables a vision engineer to inspect an environment, allows for equipment to be controlled remotely for testing of an environment, that pan and tilt cameras may be supported for a more complete and controllable perspective on a scene and that this entire process is able to be managed by a vision engineer connected to a network at an arbitrary distance from the scene (see fig. 1).

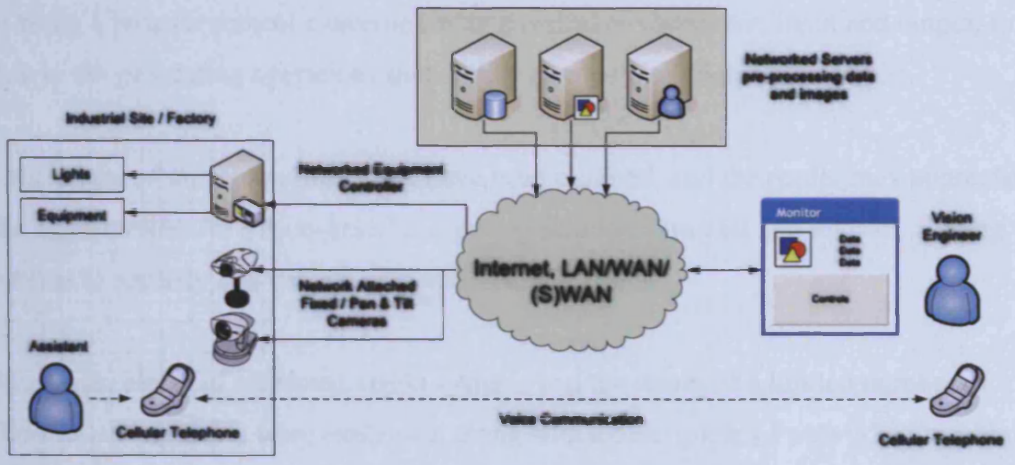


Figure 1. Utilising Myriad network-connected servers and a local assistant on site to provide a remote Vision Engineer the means to inspect a factory environment and establish the situation for the development of a vision system

Moreover, it shall also be demonstrated that network components may be changed dynamically, that security elements can be added to the system, that the system can support a wide range of interfaces and controllers including mobile computing devices (such as PDAs and cellular telephones) and that the system is able to take advantage of web server and database-stored data, and also supply vision-generated data to other software.

In this first chapter, we have examined vision as a concept; how it occurs in the natural world, and the uses to which it is put. It has been established that vision is an important tool for human beings and animals, solving problems that uniquely require visual sensing. From there, artificial vision was introduced, and the fields of computer and machine vision, showing that vision can be useful for computing tasks, frequently mimicking human operation. However, computer-based vision is not constructed in the same way as human vision, and while it does offer a number of benefits thanks to this difference, it should also be understood that one should not assume that a human vision task can be completed by a computer, nor in the same manner.

The difference between computer vision and machine vision has been defined, with machine vision being a broader subject concerned with physical environment, input and output, in addition to the processing operations that are the emphasis of computer vision.

Demonstrations of machine vision tasks have been outlined, and the reader may appreciate that the opportunities for vision-based computing solutions are vast and diverse, ranging from agricultural to security and vehicle automation applications.

Finally, the problems of restricted vision systems and the issues of a limited number of qualified vision engineers were explained, along with a description of networked machine vision systems, and how their abilities may be enlisted to support improved efficiency of vision engineers, enhanced by the freedom of non-localised developmental systems.

---

2 Vision

This chapter outlines Vision as a concept in living, describing the tasks to which it is applied, the essential nature of Vision in human society, and the opportunities it offers for discovery, interaction and manipulation.

From the fundamental understanding of visual tasks, the chapter examines the construction and functionality of human vision, in order to establish how those visual tasks are conventionally performed. Once natural vision has been discussed, the reader is invited to explore and contrast assisted and artificial vision, completed with the subjects of Computer Vision and Machine Vision, concerned with semi- and fully-automated vision.

Examples are subsequently provided to illustrate avenues of development and applications of automated vision systems.

Finally, the chapter outlines the problem addressed in this thesis and briefly describes networked vision systems as a solution together with a description of Myriad, an implementation of this solution.

2.1 Benefits of Vision

Of all senses in the natural world, vision is quite unique. Along with smell and hearing, vision is non-tactile, and most crucially, in comparison to the olfactory and auditory senses, it is highly specific and perfectly directional. Moreover, vision is also one of the most descriptive of the natural senses, able to discern fine details, such as textures, colours and forms, even at significant distance. For human beings, vision is possibly the most essential sense, evident in the capability of people to adapt to deafness and nasal problems (anyone with the misfortune of suffering from hay fever will express discomfort and irritation, but the majority will be able to function, albeit with a stock of anti-histamine tablets and tissues), still leading relatively normal lives. The losses of touch and taste are more extreme in most cases, and thankfully rare enough that they need not be considered except in the cases of neurological injury and

leprosy. Vision, however, is an essential component of society, with blindness being a significant handicap, and a massively isolating disability. To take a most rudimentary case, the reader's visual acuity is being tested now, in the reading of this sentence. Advances do mean that a computerised version of this text might be made audible through the use of screen-reading software, but the locating of the document alone is a visually intensive task, whether it arrived in the reader's possession by disc, e-mail or direct file browsing; these are once again, visual tasks in order to locate and process information.

In historical terms, human vision has a wide range of benefits to life:

- Hunting animal prey
- Acquiring plant-type foods (distinguishing by visual cues, such as size, shape and colour, preventing the person from eating poisonous or limited nutrition foods)
- Recognition and avoidance of hazards
- Recognition of known individuals and developing and understanding of new persons
- Aiding manipulation of objects
- Determining security, particularly prior to rest (e.g. checking that doors are locked/gas elements are off before sleep)
- Assessing direction and distance
- Monitoring and judging time and periods of time

Beyond the application of Vision to tasks of survival, contemporary society also provides careers and work with a large number of visual elements.

- Transportation driving for people and goods
- Word processing
- Creation of visual advertising and relational materials (posters, brochures, booklets, branded objects)
- Management and transformation of data (accounting, stock control, surveying, data monitoring)
- Restaurants and catering (precision and navigation are needed to manage/avoid hot and dangerous objects or transport them to customers and locations)
- Quality control in production environments

Vision therefore enables what most people would consider to be more fulfilling aspects of modern life in westernised societies. The development of technological assistance, tools such

as screen readers, scanning pens, brail keyboards and labelling attempt to reduce the burden of access to conventional living to visually disabled persons, but there are areas of inaccessibility that still exist, and the workplace is still bound to visual tasks. It is hardly a coincidence that the majority of computing activity (to the frustration of new media designers and next-generation programmers) concerns tasks analogous to physical activity; typing has become word processing, letter reading and writing has become e-mail, magazine/newspaper/book reading has become web browsing, typesetting has become desktop publishing and canvas artwork has become digital design. Accessibility tools in a computerised world along with empowered employment laws have broken down many of the barriers to working that would otherwise be present, but in less technological nations, disenfranchisement of visually disabled people is clearly evident. The effort required to overcome a lack of sight illustrates the essential nature of vision to the human condition and the performance of a life considered normal.

2.2 Natural Vision

Within the natural world, there exist a variety of different forms of Vision system, from the single dynamic optic, high inference system of the human eye, to the multi-optic combinatorial system present in many flying insects, to photo-sensitive single-cell organisms and other microscopic flora and fauna. The forms of natural vision are diverse, from dual-sensor combinatorial human vision to massively multi-sensor insect vision, also compensating for a range of circumstances and needs encompassing low-light optics (nocturnal animals), sub-marine vision (fish and general amphibians), high-intensity correction (desert creatures), motion tracking (many top predators) and spectra beyond the human visual norm. Careful assessments tend to place the number of unique visual forms at around 23-26 types, although recent discoveries such as creatures existing around hydrothermal vents in deep sea situations lead to further discussion that could recognise further forms.

The key advantages of vision for natural applications are articulated by Prof. B. Batchelor and Prof. Fred Waltz in their book "Intelligent Machine Vision : Techniques, Implementations and Applications" [BAT2]:

> *"Vision bestows great advantages on an organism. Looking for food, rather than chasing it blindly, is very efficient in terms of the energy expended. Animals that look into a crevice in a rock to check that it is safe to go in are more likely to*

survive than those that do not do so. Animals that use vision to identify a suitable mate are more likely to find a fit, healthy one than those that ignore the appearance of a possible partner. Animals that can see a predator approaching are more likely to be able to escape capture than those that cannot."

This understanding outlines four principle areas of life where Vision is advantageous:

- Effective food acquisition
- Enhanced Safety
- Improved reproductive opportunity
- Improved protection from predation

In many aspects, human vision follows much the same requirements as that of natural vision in other animal species, retaining the same objectives of safety, food discovery, sexual assessment and protection from predatory danger. Beyond these requirements, however, human beings have developed complex social infrastructures with the specific objectives of eliminating these natural constraints, imposing a new set of Vision requirements on existence. These artificial demands include tasks such as:

- Recognising social groups and hierarchies (e.g. identifying police uniforms, uniform logos, styles of dress, etc.)
- Reading instructions and educational materials
- Visually understanding signs and safety notices
- Categorising objects (cutlery & crockery)
- Distinguishing physically similar products (e.g. bottles of shampoo vs. bottles of concentrated household bleach).
- Controlling equipment (drills, cars, computing devices)
- Enjoying visual and interactive entertainment (films, video games, sports)

An ideal representation of this replacement of natural demands for artificial is a trivial example encountered recently by the author. An elderly lady whose first language was not English had relied on her husband for assistance in reading complex script for many years, until he passed away. As a consequence, she found herself unable to read the text on a number of household products, most particularly a bottle of bleach and one of shampoo, which were deceptively packaged in the same colour plastic with almost identical colouring of label. Whilst confusing the shampoo for bleach when cleaning the bathroom might have been

unfortunate, the opposite reversal would have produced far more grave results were the wrong bottle to be used in washing her hair. This left her with an unsafe choice to make, all due to poor visual cues provided by the manufacturers of those products. Thankfully, the lady in question had assistance from her family, who diligently relabelled all products in her house in her native language.

This problem, however, illustrates some of the visual dependencies that exist within modern society through the inconsiderate use of visual information. A wiser choice for the producers of those products would have been to apply warning labels with clear imagery (such as the 'corrosive' icon, mandated by European law). In this case, however the lady would have been unable to appreciate such additions to the packaging, since she did not possess suitable social context to recognise such markings. Specific colourings of bottles might also have helped (e.g. Milk cartons typically use blue for whole milk and green for semi-skimmed, although this knowledge is also contextual, as the American standard is the precise reverse of this). Relying on natural analogues would also fail to provide satisfactory remedy, as bright red and yellow colouring used in the wild to indicate danger and many other cues have been cunningly repurposed by companies realising that exciting natural responses can give their products improved potential recognition (e.g. Royal Mail emblematic colours are red with yellow trim, clearly identifiable and eye-catching. Are postmen naturally dangerous?).

Human vision in contemporary society involves a range of high-level inference and recognition systems, developed to satisfy and overcome the complexities introduced by society and life with many artificial elements. This application of intelligence, however, also rests atop an advanced optics and recognition system that composes human biological vision elements.

Human optics are similar to many other high-cognitive animals, in that they are directed to form a system whose characteristics are:

- Limited image quality
    - Narrow region of visual acuity surrounding the fovea
    - Limited response sensors
    - Low granularity signal differentiation
- High movement
- Situation compensation

- High degree of inference from resultant images

As discussed by Brian Wandell [WAN], the quality of image encoding is distinctly limited due to the quality of the photoreceptors and the optics leading to them. See fig. 2 for a diagram of the human eye. Any given photon reaching a rod or cone will pass through the cornea, distortable lens and the transparent vitreous (complete with floating cell matter) before reaching the optic disc.

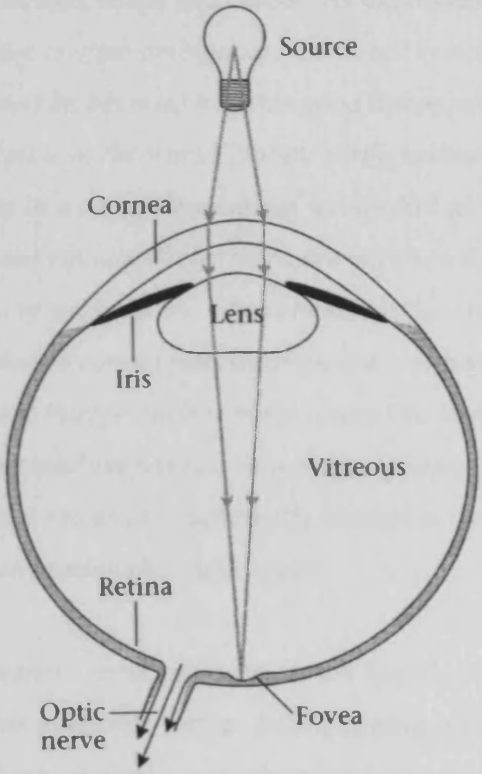


Figure 2. : Path of optics through the eye to the optic disc.

While this means of image acquisition is not nearly as accurate as might be expected from a modern digital CCD and optics, it does respond far more effectively to variable conditions, such as temperature variations (cornea and lens compensate), impact (vitreous shields the retina from most simple damage), high and low light (iris and lens distortion) and general movement both of the target and the viewer. This is, of course, not to mention the focal range provided by the lens when examining ranged and close targets. Therefore, whilst the eye is not able to gain a high quality image in all conditions, it does succeed in obtaining a sufficient

image in the majority of situations, often a more valuable trait. These features also mean that minor damage inflicted to any part of the eye, or loss of degrees of function due to age, illness or malformation are correctable, and are less severe than they might be in a high precision system with more restricted range of operation.

The optics of human vision are, however, only part of the overall solution that constitutes vision. The counterpart of this element is the inference system provided in the neurons of the brain and the specialised visual streams formed in the optic nerve, physically encoding recognition elements, such as size, shape and colour. As expressed by Wandell:

> *"Since the retinal image is often ambiguous, the visual system's success in interpreting images must be because it makes good assumptions about the likely properties of objects in the world. Not all configurations of objects are equally likely; we exist in a three-dimensional world. Not all surface-reflectance functions are equally likely; there are regularities in the wavelength properties of surfaces and illuminants. Not all types of motion are equally likely; hard objects cannot pass through one another. The unequal probabilities of different interpretations make it possible to make informed guesses about the color, motion position and shape of objects. The probabilities of different events are sufficiently skewed so that the visual system succeeds in interpreting the image data"*

What Wandell does not comment upon, of course, is the degree to which human beings customise their environments to encode further distinguishing information, through the application of writing, marking, general manipulation and wearing that lend further opportunities for differentiation of stimuli. In summation, the human visual system is designed for highly active and diverse use, with the expectation of damage to be overcome, and taking maximal advantage of a large inference system, able to compensate for the limitations formed from the implementation of such a generalised system.

2.3 Artificial Vision (AV)

Beyond natural vision, the creation of scientific imaging devices has enabled human beings, animals and computers to see beyond their natural abilities (of which computers, technically have none). We will examine two types of artificial vision in an effort to establish where computers fit into a visual world. Artificial Vision can be seen as the opposite to natural

human vision in many ways; it is precise, fixed, high response, limited dynamic range and minimally inferred. The use of such systems, to complement human vision in cases where acuity it desirable, therefore, is hardly surprising, and enables a user to examine a far more complete set of problems than those to be inspected by the naked eye alone.

Computer Vision

Most definitely, computers with cameras and other super-visual acquisition apparatus (such as X-ray detectors) may be classed as artificial vision examples.

Augmented Natural Vision

The first stage in artificial vision, is the attempt to expand the range of natural vision, beyond that which is in fact, natural. Even prior to the development of computing devices and their ability to capture and process imagery, efforts were completed to enable human beings, with the assistance of cameras and film of special types, to visualise:

- X-rays
- Infra-red
- Ultra-violet
- Alpha/Beta/Gamma radiation

We may consider these tools to be the first attempts at artificial vision, and it is not coincidental that these tools then became the same tools, once equipped with analogue-digital capture equipment to facilitate computer vision, through the use of digital cameras and frame grabbing hardware.

The line between what is natural and what is artificial is somewhat blurred by this definition, as while it might be obvious that a seeing computer is artificial, the question arises as to exactly whether artificial retinal implants are artificial.

2.4 Computer Vision

In enabling computers to see, there are two discrete areas of development:

- Capture of the image
- Analysis of the image

Computer Vision (CV) is concerned, most primarily with the creation of techniques of processing images to attempt to understand whatever is being visualised, or to distil the image down in particular ways to perform a deduction (objects being observed, quality of the image, how to respond to the image).

The objectives of a Computer Vision system (see fig. 3), therefore, are based around Image Processing and the processing of additional data, such as the use of knowledge bases to assist in the analysis of an image (e.g. recognition of object silhouettes[OTT]). CV tasks are non-localised problems based in the realm of computing; they do not deal with the physical world directly, but assume that the images and data being used are already captured, and that resulting deductions may be acted upon by a party such as a human operator to affect the world, or deal with other computing resources.

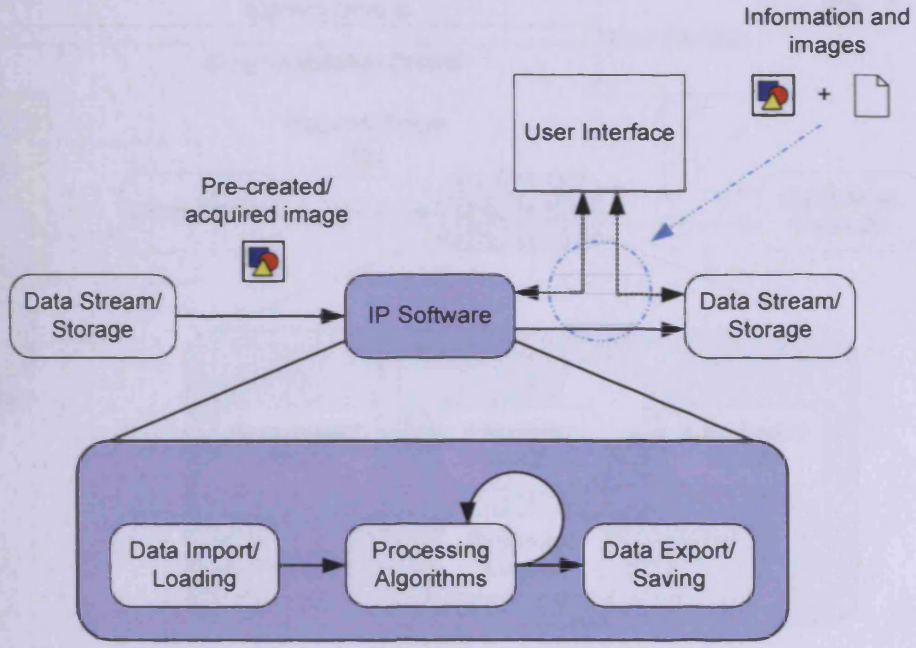


Figure 3. : Conceptual Computer Vision system and interactions

While this thesis makes detailed use of CV techniques, and particularly Image Processing operations and the concepts of data integration from databases and other sources, it also deals extensively with physical capture of images, object manipulation in the real world, and problems of locality and hardware/connectivity. As such, we deal with the wider subject of Machine Vision.

2.5 Machine Vision

While Computer Vision deals with the processing of image data and other data, it exists in isolation from physical activity; it has nether control of the images being captured, nor is it able to directly affect the world in response to understandings that are developed from a captured image (e.g. a Computer Vision system can understand that an image is dark, but it cannot capture the image from a camera, realise that it is dark, and thus turn on additional lights, seeing the results and then being able to capture and process the much improved image of the scene). Machine Vision can be seen as a superset of Computer Vision, as it deals with system input and output as well as the processing step (see fig.4).

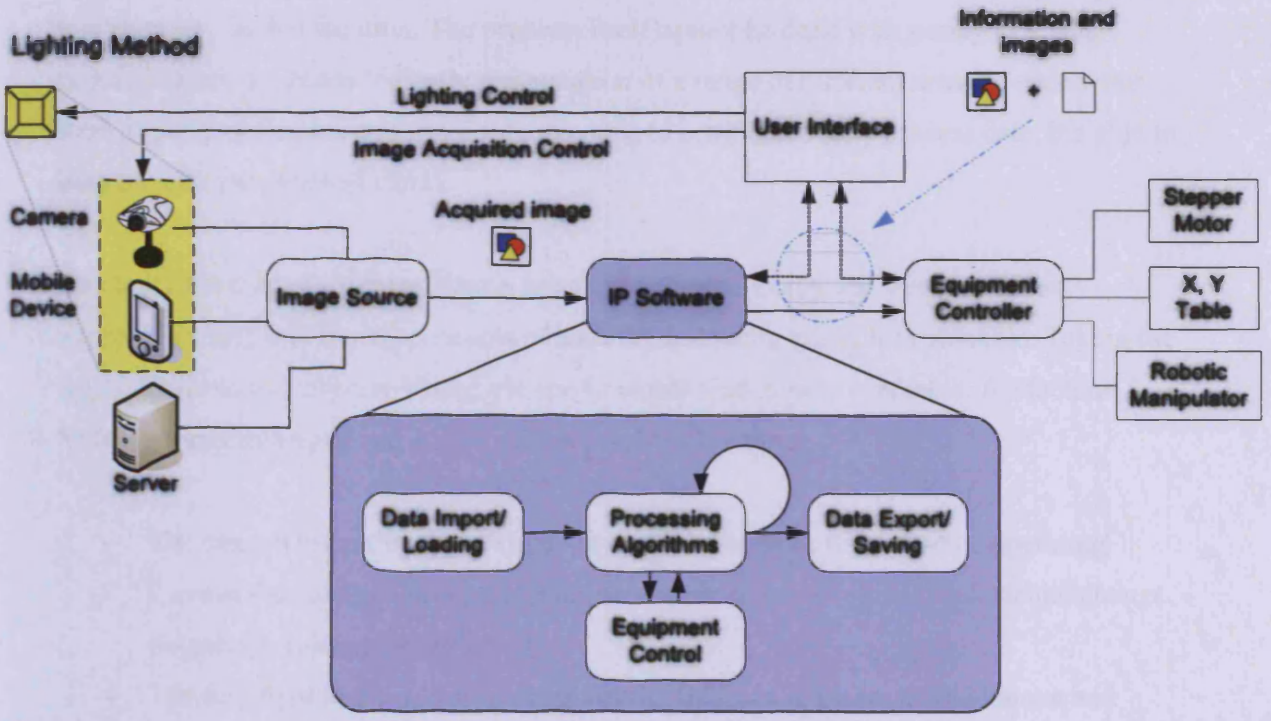


Figure 4. : Conceptual Machine Vision system and interactions

This is not mentioned to dismiss the validity of the work done in Computer Vision, as it does form the core of both endeavours, and while Machine Vision research dedicates a portion of time to data processing, CV researchers are able to spend far more of their time on the subject, while MV researchers deal with the wider concerns of the system into which the data processing operations will be incorporated. In this way, Machine Vision and Computer Vision

are in partnership and together, provide high-quality solutions to the problem of helping computers to see.

It is important, however, to carefully mark the difference between the two disciplines, as problems of seeing computers in the physical world, most often deal specifically with issues that contain physically-constraining elements. For example, when dealing with examination of a factory-based production line, an image must be captured in a specific place, with specific lighting; a solution that assumes that a high-definition image is already ready for processing, and that does not take account of what to do with the conclusion that a product being visualised is bad, is only half complete. Regardless of the system being used to process the image data, an image must be captured in a given location, and the quality of the lighting and environment must be controlled in that location, and equipment must be able to respond to a situation, in that location. The problem itself cannot be dealt with purely as a data-processing puzzle, but is logically a summation of a range of different areas of endeavour working in coordination to form a solution; able to computationally process data, but able to interact with the physical world.

To clarify the role of Machine Vision as an expansion of Computer Vision, with an appreciation of the differing concerns of task, the following example is valuable. Taking the figure 5 below, a Computer Vision viewpoint would find it quite agreeable. A Machine Vision perspective however, would raise a number of issues:

- The camera mounting is not rigid – it could move or be moved while operating
- Camera focus/iris controls are freely accessible to any workers or accidental change, potentially ruining image quality
- The ring light in front of the lens is not rigidly fixed in place; it could move and obscure the view, alter the lighting, or harm the camera
- The camera and lighting system as a whole is not protected from physical injury which may occur in an industrial environment
- The camera and lighting system is not protected in an way from the leaking chemicals it is intended to detect
- The metallic bars containing the products are actively reflecting light back towards the camera and partially block the ring light's field
- Ambient lighting is not controlled in any way

- There is no uniform background behind the products to ensure a good silhouette, and indeed, the rear wall is reflecting a partial image of the bottles back toward the camera
- The operator's monitor is on the opposing side of the production line to the camera, not only introducing light into the scene, but also preventing the operator from managing the camera and monitor at the same time.
- The operator must face away from the unprotected production line in order to see the monitor, with leaking chemicals behind their back
- Video cabling has exposed inputs and plugs hang out at a distance from the terminal, risking damage, and potentially introducing a hazard to workers passing by.

From this list, the reader should appreciate that the role of Machine Vision is to examine a system as a whole, rather than the steps taken within a computer, after an image has been acquired. The combination of good Computer Vision and careful physical consideration of a problem provides a far easier task to solve and more reliable solutions than Computer Vision alone.

Figure 5. A sample machine vision quality testing of chemical bottles system, image used with the kind permission of Omron Electronics LLC.

2.6 Applications of Automated Vision

2.6.1 Industrial

One of the primary applications of Automated Vision, is that of quality control in industrial settings. Manufacturing industry supplies a range of products that require assessment, for safety and overall visual quality.

Such products include:

- Components for vehicles (e.g. brake blocks, turbine blades, engine cylinders)
- Medical apparatus (e.g. syringes, high pressure liquid drug containers, heart valves, artificial joints)
- Food products (e.g. jars & bottles (defects and foreign objects), boned meats, highly visually variable products (fruit, bread, cakes)

In these cases, the two requirements being satisfied by the application of a Machine Vision system are:

- Safety
- Quality for the objective of saleability

A variety of goods are also inhibitive in the manner in which they may be inspected, necessitating the use of vision; such as food. In food and medical consumables, it is often the case that the use of tactile sensing and probing will damage the product (examining a loaf of bread for foreign objects would require poking a very large number of holes in the loaf from multiple angles) or expose it to contamination from the probe (searching for objects in medical vials). Therefore, Machine Vision is uniquely suited to supporting manufacturing industry, where other investigative methods are incapable of providing the level of safety and quality control required to ensure the integrity of the product.

These systems often perform tasks that are already being performed by human operators [BAT], albeit with a reduced rate of error. This can have advantages:

- The problem is already apparently visually solvable
- The problem is well understood
- The process of manufacture is readily set up to allow for a quality control stage

There can also be drawbacks:

- Jobs may be lost [BAT]
- Very diverse problems may still be cheaper to solve in the short and medium terms through the application of human beings

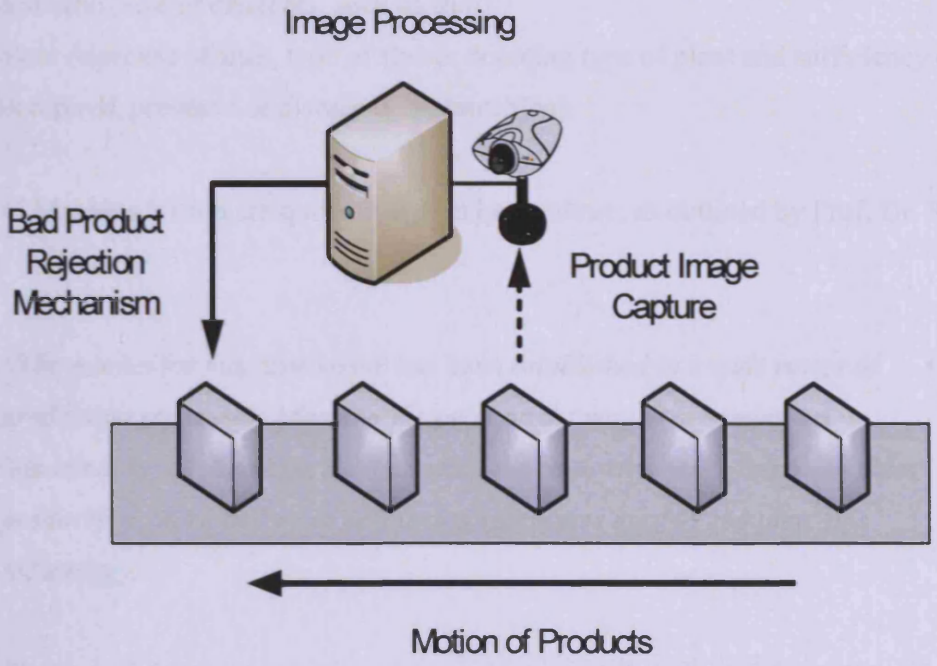


Figure 6. : Classical industrial quality control MV system design

A common Machine Vision system in a production setting consists (see fig.6) of a stage in the manufacture where products pass a certain point (typically on a conveyer), are visually acquired and inspected, and then are allowed to continue if acceptable, or are rejected into a spoil hopper if defective.

Such a system as outline above is dependent on a number of factors:

- Clear understanding of the products being investigated
- Understanding of faults which can occur, and their recognition
- Control of the position of products during inspection
- Appreciation of the environment of the workplace, especially with relation to dust, movement of components and personnel and ambient lighting
- Control of lighting and intrusion into the inspection space
- Reliable image processing functionality and recognition algorithms
- Reliable manipulation and ejection mechanisms

2.6.2 Horticulture

Horticulture is an intriguing area for the application of Machine Vision, principally because visual examination and classification is the only means of inspecting the state of a plant or fruit while it continues to grow. Metrics used are threefold:

- Numerical (number of fruit, leaves, etc.)
- Geometric (size of elements, such as fruit)
- Colour (ripeness of fruit, type of flower denoting type of plant and sufficiency of chlorophyll, presence of disease or malnutrition)

The uses of Machine Vision are quite diverse in horticulture, as outlined by Prof. Dr. Rath [HANV]:

> *"The market for machine vision has been established in a wide range of production processes. Machine Vision is on the way to becoming an important key technology, safeguarding the competiveness of users in plant production. More and more colour information is used to complete this technology.*
>
> *The task of this project is to analyse and interpret the colour information of horticultural scenes* in a biometrical or statistical way. The goal is a system which can select coloured agricultural objects (plants, weeds, cut flowers, etc.) in an optimal way using the information of the different colour channels.*
>
> **Examples of horticultural scenes: Weed detection in open field situation, discrimination of plant parts (e.g. leaves from stems) for plant propagation, analysis of the nutrient and health status of plants using the colour of the leaves"*

In contrast with most other Machine Vision tasks, horticultural use very much focuses on using MV for analysis of the scene and subsequent statistical activity. In addition, the primary operators for examination are colour-based, rather than greyscale presence/no presence methods that are the core of industrial inspection.

There are, however, a number of developments in the field of robotics and horticulture, relating to the management of robotic equipment to manipulate plants, automatically prune them, collect fruit and perform complex propagation tasks, as discussed by Kondo, Nishitsuji, Ling & Ting in their 1996 work *"Visual feedback guided robotic cherry-tomato harvesting."* [KON]. These systems have a great deal in common with activities such as Automated Guided Vehicles and regular vision-targeted robotics. This is, however, a unique vision problem, in that it deals with:

- Non-uniform lighting at the plant level, due to repetitive occlusion.
- Small-object analysis
- Randomised object motion (swaying stems, leaves, etc.)
- Non-man-made targets
- Variability of colour across elements (leaves, petals, fruit, etc.) and more widely across numerous plants, requiring tolerance systems

Development is also ongoing in the creation of systems to respond to analysis of plants, such as systems to judge the correct amount of required chemicals to apply to food crops as shown by Alchanatis, Hetzroni and Edan [ALC].

2.6.3 Medical

The use of Machine Vision and Artificial Vision in medicine is one of the earliest areas of development for associated technologies, as outlined by Lasse Jyrkinen's discussion of Radiology as a field of endeavour:

> *"Radiology is the field of medicine that uses various forms of acoustic and electromagnetic energy for producing diagnostic images of the human body. Since the discovery of X-rays by Rontgen in 1895, X-ray imaging has dominated the history of radiology and it has only been during the last three decades that alternative imaging methods have begun to emerge. The first of these was ultrasound, which used acoustic waves in forming an image. Shortly afterwards computed tomography revolutionised the world of medicine when it made it possible to acquire cross-section images from a living human body. One drawback, however, was that computed tomography still used X-rays, which are harmful to living organs.*

In the late 1970s magnetic resonance imaging (MRI) was introduced. MRI is able to produce two- or three-dimensional images of great quality, showing cross-sections through body parts at regular intervals. The images are so precise that radiologists are often able to get as much information from a scan as from looking at the tissue directly."

Non-visual-spectrum investigative methods now include such techniques as:

- Magnetic Resonance Imaging (MRI)
- Computed Tomography (CT) scanning
- Infra-red examination
- Ultra-violet exposure of injuries
- Internal organ and systemic visualisation using Beta and Gamma radiation sources
- Ultrasound visualisation

The use of vision-based methods and tools can be highly effective and advantageous as a means of diagnosis of the human body, since palpation and fluid analysis can only reveal a limited number of conditions. In order to extend the range of diagnosis invasive procedures must be performed to gain an understanding of the internal situation. Not only are such procedures costly and requiring the skills of a wide range of professionals, but they also expose the patient to risks of complications arising from:

- General or local anaesthesia (to which a number of patients react adversely)
- Collateral damage from invasion
- Loss of blood and non-clotting bleeding
- Post-operative infection

This is not to mention that the amount of work required to maintain dressings and stitches and provide outpatient care to the people whose abdominal pain is irresolvable and required a number of investigative surgical operations, would be prohibitive in its own right.

Developments in medical vision extend beyond the limits of 2-Dimensional visualisation and diagnosis, however. There are a range of services and projects designed to produce 3-Dimensional visualisation of organs for exploration; one such example would be the inspection of the pancreatobiliary tract as discussed in [ROB] for the recognition of bile fluids

and their placement internally. The benefits of 3D visualisation of internal structures of bodily organs are primarily in the field of surgery, where the use of such tools assist the operating team in planning operative procedures and gauging the potential outcomes.

Medical applications also benefit from the use of Machine Vision in the fields of keyhole (laparoscopic) surgery and Interventional Radiology.

Interventional Radiology is defined [SIR] by the Society for Interventional Radiology (US) as:

> *"Interventional Radiology (IR) -- minimally invasive targeted treatments performed under guided imaging -- in the mid-1970s. Interventional radiology procedures are a major advance in medicine that do not require large incisions, and offer less risk, less pain and shorter recovery times compared to surgery."*

The use of IR in surgical procedures provides visualisation and instrument tracking in 'closed' surgery, allowing surgeons to follow the path of fibre-optic cameras and instruments topologically as they progress internally and administer directed injections and tools (e.g. angioplasty balloons) to the tracked site [JYR]. Image processing techniques also provide a range of tools for augmented visualisation, pre-surgical simulation and Medical Teleoperation [PASS].

2.6.4 AGVs

Automated Guided Vehicles (AGVs) are robotic systems that operate by tracking visual cues. In the simplest case, a truck (with a computing device and camera onboard) designed to carry stock might follow the path of a line painted on the floor and stop when the line ends, transporting the load from one place to another.

In most implementations, AGVs are designed as 'pullers', carriers or manipulators.

'Pullers' or 'tuggers' are vehicles produced to pull other objects behind them, along their course. One example would be a small vehicle able to tug aeroplanes and loaded wagons along the lines painted on the tarmac of runways and paved airport areas. Specifically

coloured and modified markings already exist on runways and parkways to enable pilots to follow to their allocated gate or station. The function of a tugging vehicle is therefore essentially the same as that already provided by the pilots and manned tugs, the drivers of which are providing a visually-cued sequence of movements.

In his 1981 paper [THO] to the 1st International Conference on AGVs, Brigadier R Thornton describes the use of 'driverless tractors' into the Central Ordnance Depot at Donnington, one of the largest storage depots in Europe, at the time of his writing. The AGVs in use were provided with both visual and magnetic guidance measures, following white painted lines on the floor, or embedded wires below the floor. The AGVs themselves were able to be operated manually or automatically, had been ruggedised for outside use (including waterproofing) and were powered by rechargeable batteries with a range of included safety measures (undoubtedly beneficial in a situation where munitions were being transported and stored). Brigadier Thornton reports a number of acknowledged benefits of the system:

1. They are reliable: downtime is negligible
2. They save manpower.
3. They provide a certain discipline in the operation of the warehouse
4. They are simple to install and need little maintenance
5. They are ideal for repetitive work
6. They will work easily outside their normal location
7. They cause less accidents than many conventional movement systems
8. In the event of power failure they will continue to operate for up to 24 hours

The origins of the AGV can be seen in the industrial revolution, with the introduction of small-scale railway systems in factories and between critical sites. Embedded rail designs ensued, with transport systems being the primary cause of deployment. With the addition of electronic systems in the mid/late 1960s, however, these systems can be firmly classed as AGVs. The inclusion of visual tracking in the mid-1970s which was present in Brig. Thornton's paper marks a clear change in that it introduces computing elements into such systems, and Brig. Thornton himself speculates on potential future developments with connection to the depot computing systems for high level control and advanced programmed tasks. Much of this progress has now taken place, and such systems commonly participate in the working of large warehouse environments, following visual markings and

loading/unloading stock autonomously, often by examining product barcodes visually and ascertaining the desired action.

Carriers operate in much the same way as 'pullers', except that they provide a set of carriages or customised loading volume which can be enjoyed by appropriate objects. These systems are frequently employed in large industrial environments, where large quantities of products are required to be transported from one area of the site to another. Conventionally, this process would be performed by manually-guided vehicles or forklift trucks, but this requires multiple trips and the permanent retention of drivers. In addition, it also exposes the site to health and safety risks that are introduced with driven vehicles, such as crashes, fumes from combustion engines, noise and catastrophic failure (e.g. brake failure, toppling loads). For highly repetitive operations along well-defined paths, therefore, it makes sense to use an AGV.

In addition to the benefits of AGVs as enumerated above, there are also similar considerations for safety that require the implementation of non-human operators. In industrial environments including the use of hazardous chemicals and radioactive agents, there are considerable risks to the regular use of human workers for transportation, particularly when driver error might result in the spilling of a dangerous load or damage of safety equipment. The same risks are true in the cases of operation requiring manipulation. One of the first areas of deployment of robotics is the use of assembly, welding and painting robots in automobile production. Aerosol paints, sealants, fixers, preparation chemicals and cleaning products are very frequently hazardous, both on physical contact, and when breathed by human workers. Lifting, moving and manipulating heavy elements such as modern reinforced car doors can also be physically detrimental on a regular basis to workers, producing muscle damage and chronic conditions, such as back pain. Finally, exposure to arc-welding equipment and high temperature systems in factory environments is a constant source of danger that poses a threat to workers; while this may not be a problem for fully-aware and well-trained personnel, conditions such as distraction, tiredness or illness may produce fatal accidents. As a result, automated systems are frequently used to replace human beings in production of large-sized and complex products.

2.6.5 Astronomy

In much the same vein as Landsat, astronomical use of Machine Vision is concerned firstly with the analysis of high-resolution imagery. In contrast to Landsat, of course, astronomical progress is outward looking, rather than being Earth-examining and contains a far wider range of objects being inspected, and with far more potential data to be acquired. The total number of objects in the universe is so massive, that present estimates are highly vague by many orders of magnitude. Considering only stars, estimates range from $2x10^{11}$ to $3x10^{18}$ stars by many parties, with NASA going so far as to officially guesstimate over a zillion stars in the universe as a whole [NASA]. Apart from stars, the goal of finding extra-solar planets (those outside our solar system, forming other solar systems) requires sensitive noise-reduction and measurement techniques on both visual and radio telescope data, as described with the discovery of a planet circling the star HD 209458 [HUBB]. The number of potential planets in the universe is even less precisely known, as too few have been discovered to establish grounds for extrapolation, however, the number in almost all rough guides starts in the billion figures and escalates to several times the number of existing stars, depending on the data source.

Moreover, astronomical data is of special significance, because it includes three types of data (see fig.7):

- Earth-based
- Orbital satellite imagery
- Travelling probe imagery

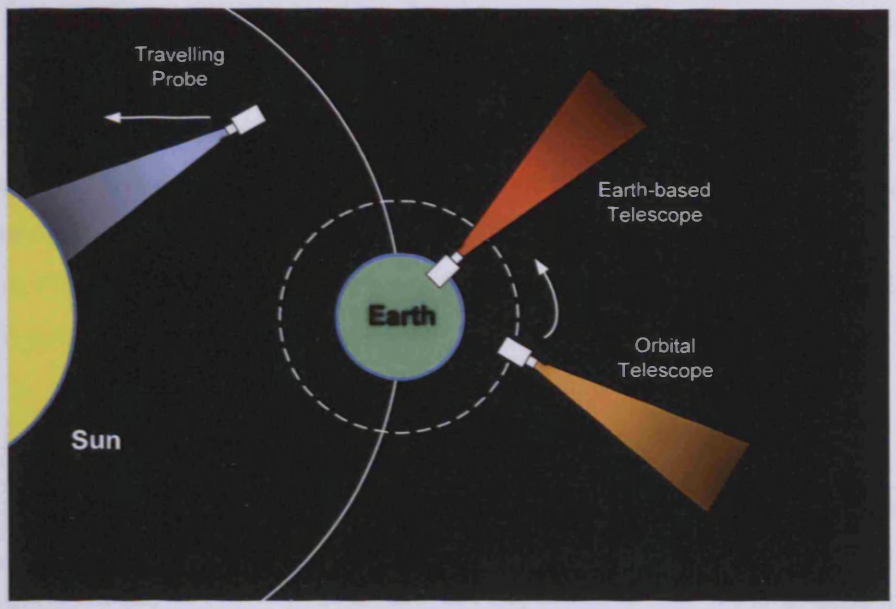


Figure 7 : Differing classes of telescopes and space image sources

Due to this range of sources of data, astronomy is also a more diverse field in terms of processing than earth imagery. The diversity of subjects covered and the expectations for data, from mapping and enumerating extra-planetary bodies, to exploring more abstract spatial relationships in attempts to gain insight into universal processes, require a range of analytical methods and skills.

Some of the bodies under study are:

- Stars (binary, singular), pulsars, quasars
- Galaxies
- Solar system planets and moons
- Extra-solar planets and systems
- Black holes
- Novae
- Nebulae
- Galactic dust clouds
- Comets and asteroids
- Dwarf stars
- Star-birthing clouds (M16 et al.)

The most marked difference between astronomy and Earth-specific inspection, is the type of sensing which is done. The majority of Earth-based exploration is done with the visible and near-visible spectrum, such as IR wavelengths. At a galactic scale, however, photons are actively scattered and impeded by the objects between the viewer and the body in question; inter-stellar dust particles are especially problematic in this regard; not only this, but due to the distance of objects, the actual paths being followed by the photons are perpendicular to the viewer and also exceptionally fine (see fig.8).

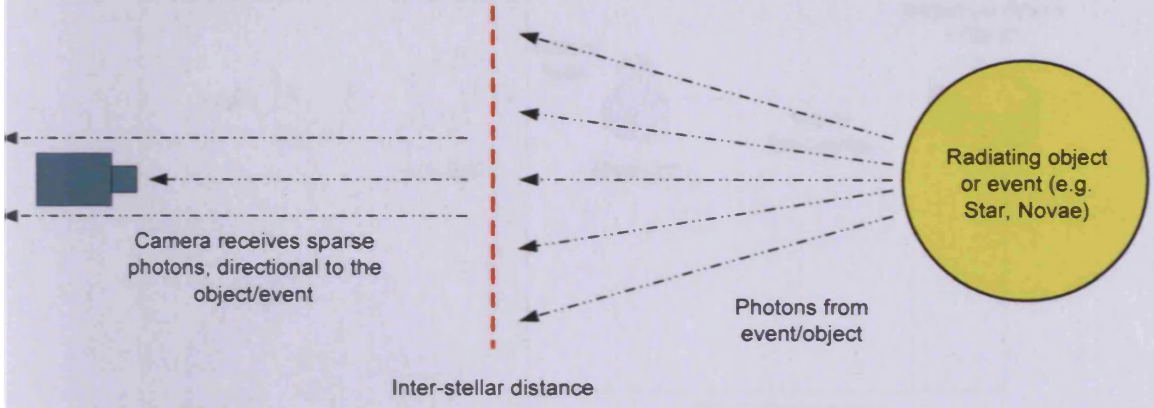


Figure 8 : Capture of light rays from remote stellar objects

There exist very few optical sensors capable of identifying light from truly distant stars, and when these sensors are on Earth, they suffer from ambient light from habitations, and atmospheric scattering effects. For these reasons, visual inspection is used for near-Earth bodies, but radio telescopes are the most common means of examining remote stellar objects.

In the case of astronomy, therefore, most Vision-based activities are done not on visual data, but on interpreted radio data, from which images are constructed and then processed. Aside from the difference in the EM wavelengths being used, utilising radio for inspection is remarkably similar to the visual approach. For example, the examination of a star using visual wavelengths and a diffraction grating would distinguish the spectral fingerprint of the star and identify the molecules present in the internal fusion reaction. Were radio wavelengths used, however, subtly different wavelengths will indicate the active composition of the star and the nature of the fusion. Using radio telescopes, a computer may be used to collate the data from the incoming waves and build a profile of the star, coming to a similar conclusion to that of a visible example.

Once an image is acquired (either visually or radiologically), however, processing may begin in much the same manner as with other inspection. Apart from Earth-facing processing, astronomical image processing tends to be more diverse, and as such, demands intelligent automated processing systems, or interactive processing.

2.6.6 Robotics

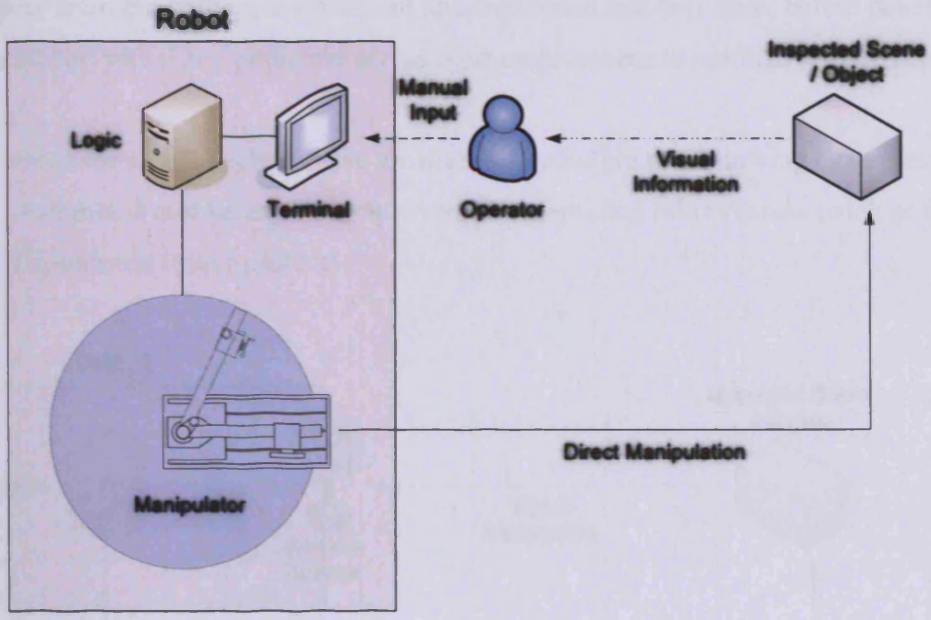


Figure 9 : An operator administering a robotic system, acting as in intermediary who translates and enters visual data

The application of Machine Vision to produce a genre of 'seeing robots' is a significant aspiration of robotics as a whole. As mentioned previously, Machine Vision is a helpful tool in that it is a non-tactile, highly data-dense and provides a bridge between computing systems and the physical world, both in terms of data acquisition and active response (see fig.9). Robotics as a subject is specifically concerned with the production of systems to perform the latter half of that activity, through the control of equipment, most typically for manipulation. Including Machine Vision elements in robotic endeavours provides the former activity and enables the construction of systems that are able to work without a person acting as an intermediary for data entry or initiation (see fig.10).

Seeing robotics for hard-wired automated tasks may be applied to a range of problems, such as:

- Auto-targeting welders
- Barcode-reading automated stock collectors

Beyond these heavily controlled applications, developments are ongoing into seeing robots with the ability to actively learn, through the use of genetic computing methods and neural networks. Principle objectives for the first generation of such systems, are mobile robots, able to actively learn about their environment and the terrain that they face, before developing sets of rules to best travel and path-find across their environment to perform a specific task.

Applications for robots such as these are diverse, including the following examples:

- Automated mobile exploration rovers for examining other planets (such as the Mars Exploration Rover project)

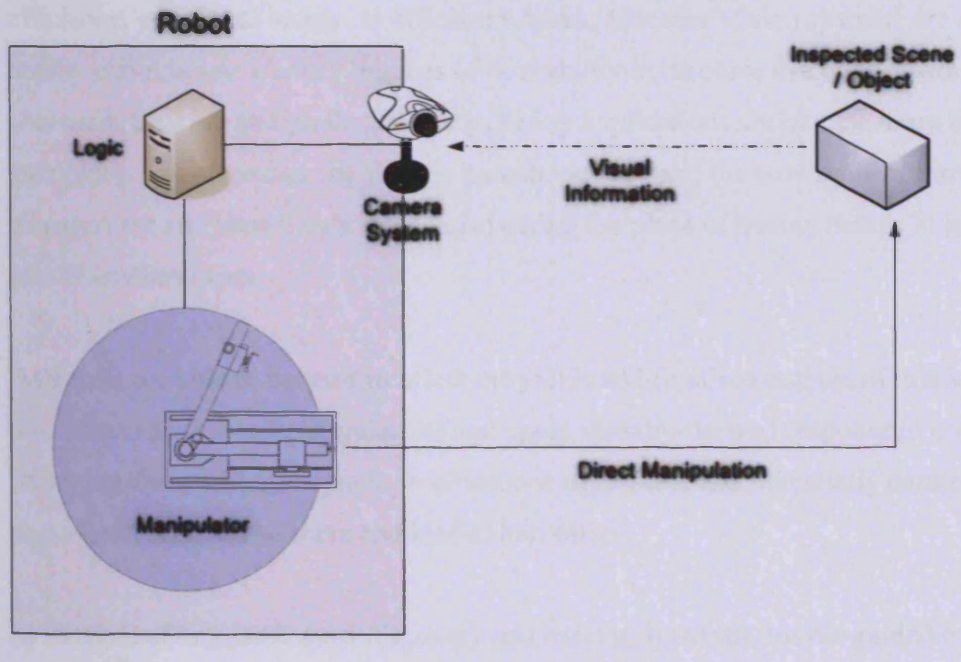


Figure 10 : Solution utilising image processing and Machine Vision techniques to compose a 'seeing' robotic system

The two subjects of Robotics and Machine Vision are very closely entwined, despite their often frequent delineation for research reasons. In the other areas of development mentioned here, it should be clear that the two are very much used hand in hand for tasks such as horticultural management, warehousing and astronomical telescope control.

2.6.7 Commerce

Use of Machine Vision in commercial environments is one of the distinct areas of growth for new MV activity. The greater industry may be broken down in much the way that economists, geographers and social scientists might espouse:

- Primary industry (initial manufacture, mining, growing)
- Secondary industry (processing, product manufacture)
- Tertiary industry (presentation of services and products to the consumer; sales)

Examining the uses and progress of Machine Vision in these discrete areas provides reasonable categorisation for our purposes.

Primary Industry

The uses of Machine Vision in the acquisition of resources are integral to industrial efficiency, but also to safety. In efficiency terms, Machine Vision systems are able to provide quality controls and identify degrees of desirability in resource discovery, with tools such as coal seam tracking and grade discovery. Safety applications are of a far more broad range, attempting to both reduce the risks to human workers and the environment through risk management and identification, and also taking the place of human beings in high-risk or hostile environments.

MV tools are able to be used to affect the yields and finalised outputs of this sector. They are also able to handle agricultural tasks and apply visually-derived response to conditions, removing the need for people to troubleshoot large-scale and potentially dangerous equipment, such as threshers and bladed harvesters.

In the field of inorganic asset discovery and harvest, however, vision-guided robotic systems and analysis tools can be valuable in automating highly repetitive and strenuous tasks, such as mining. Not only are robots able to inspect and excavate smaller spaces than human beings, but there are reduced consequences in the event of mine collapses, digging equipment failure (including fire and loose blades when damaged). Automated mining systems, of course, have applications in the mining of other planets, the Moon and comets/asteroids; techniques that will be essential in initial manned missions to Mars and beyond, and the establishment of a

long-term presence in space beyond the 200km orbit currently playing host to the International Space Station.

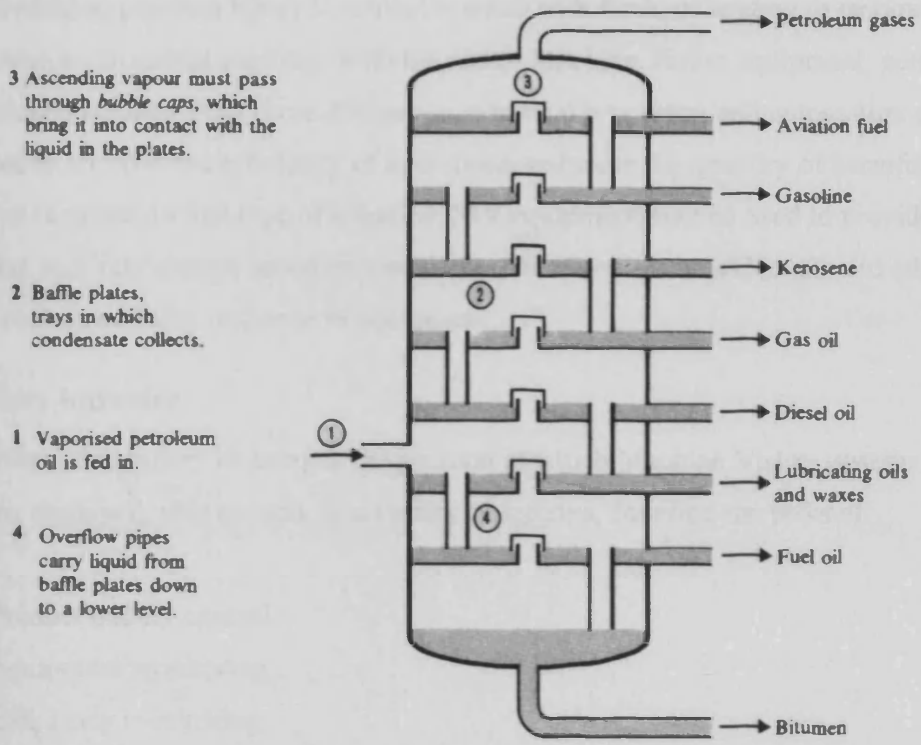


Figure 11 : Fractional distillation of crude oil into products ("Chemistry" 2nd Edition, Ramsden et al.)

Once an inorganic resource is discovered and acquired, however, the majority of cases require purification and refinement of the product in question. In the case of oil, it must be graded and then refined into differing forms (see fig.11) prior to further manufacturing processes. The actual required products vary depending on the objectives of the company involved in the refinement, but common outputs are:

- Petroleum elements for plastics
- Automotive fuel
- Lubricants
- Aerospace fuel
- Petroleum gas
- Waxes

In other cases, raw materials are processed in order to extract metals and other products, such as the conversion of bauxite to constituent un-oxidised aluminium. These processes normally

involve the use of concentrated oxidation and reduction agents, such as concentrated sulphuric acid. Clearly, the use of these chemicals may be intensely harmful around human beings, leading to physical injury if contact is made with flesh, or leading to respiratory damage such as bronchial scarring. With the use of Machine Vision equipment, personnel may be clearly isolated from these dangerous industrial processes, and automation may be introduced to improve the efficiency of operations, reducing the quantity of harmful substances required. In this type of situation, MV equipment may be used to provide monitoring and intervention based on visual cues, measured chemical levels and other metrics, such as auditory response of equipment.

Secondary Industry

In most cases, Secondary Industry is the position at which Machine Vision systems have been classically deployed, able to work in a variety of settings, covering the tasks of:

- Product quality control
- Equipment monitoring
- Efficiency monitoring

Inter-industry Transport and Distribution

In addition to the use of Machine Vision equipment with Secondary Industry, there is a class of inter-industry services, based around distribution, which makes extensive use of MV techniques.

The use of AGVs in factory and warehousing environments has already been discussed, and is one application of MV systems, but that also goes hand in hand with robotic systems designed to perform picking operations in large storage spaces, reading the barcodes on boxes of stock, picking the desired box and placing it on the AGV ready for transport to packing and shipping areas of the business, which might similarly employ robotic equipment. As a result, an operator may enter into their desktop computer, an order which they received over the telephone or e-mail, or approved in their web-based business-to-business ('b2b') ordering service; noting products, quantities and the destination. The automated warehousing system proceeds to:

- Move the AGV to the appropriate area of the warehouse, where the products may be found

- Use a camera on the robotic picker on the AGV to discover all the locations of the barcodes on the visible boxes
- When recognising the correct barcodes, lifting the boxes from the shelves and placing them on the AGV, to the quantity required by the customer.
- Move the AGV to the dispatch loading dock
- Inform the distribution workers that boxes are ready to be loaded
- Print tracking numbers for each box, ready for the workers to apply them to the boxes as they load the vehicle (or applied automatically)
- Store the tracking numbers in a database and inform the distribution company of the barcodes

Upon delivery, the driver uses a mobile computing device with a barcode reader to record the barcode-marked items delivered, and these can then be checked off against the database records, ensuring complete delivery, prior to invoicing.

The majority of companies with very large warehouses are presently instituting such systems, with integration with internal distribution and courier companies for transport to customers, such as retail premises. One such example of this is Amazon Corp. the pioneer of web-based commerce; more than most, their infrastructure has been built in recent years, enabling them to take advantage of new technologies in their first implementation, rather than having to replace existing systems and consider the cost benefits of replacement.

Tertiary Industry

Concerned with the sales and distribution of products and services to customers and end users, the Tertiary Industry is most commonly associated with the retail space, but includes:

- Shop retail
- Internet retail
- Service provision
- Support services (call centres, automated systems (b2b ordering services, etc.))

As a result, the range of Machine Vision opportunities is very different to manufacturing industry. Quality control aspects are typically not a key point of investigation, but more immediate elements, such as visual sales registering, stock management and customer monitoring are more essential.

Examples:

- Visual sales registering
    - Barcode readers
    - Full-item recognition systems
- Stock management
    - Identifying damaged goods
    - Visual tracking of stock objects around a warehouse/retail space
    - Visually-assisted high-speed stock taking systems
    - AGV systems for transfer of goods, and automated picking for packing and dispatch
    - Barcode-based stock confirmation and data addition systems
- Customer monitoring
    - Visual customer identification and privileged security authorisation
    - Tracking of personnel and customers around privileged locations
    - Motion tracking security systems
    - Visually-assisted CRM (Customer Relationship Management) applications

In the majority of Tertiary Industrial environments, however, the use of Machine Vision tools and techniques is firmly restricted to the reading of barcodes and dot matrix codes for retail and warehouse picking. Likely next steps of implementation include detailed barcode stock tracking, visual security measures and assisted CRM tools (*"Welcome Mr Philips, we've just had a few new products in stock that you might be interested in... "*).

2.6.8 Forensic Science

Vision techniques are a modern and increasingly frequent tool in forensic science; the ability to examine evidence, deduce additional information, and yet leave the evidence itself untainted by interaction is highly valuable in criminal investigation.

There are two discreet areas of endeavour that should be addressed in this field:

- Image processing for examination of physical evidence
- Video processing and correction

The first covers more publicly-understood tasks, such as:

- Fingerprint identification
- Bullet/cartridge striation identification
- Facial reconstruction
- Photographic analysis
- Handwriting recognition, comparison and indentation analysis

Geradts and Bijhold [GER] outline the following common tasks for image processing of video images in forensics:

- Image enhancement
- Visualization of images for the court
- Image comparison
- Video tape integrity verification
- Camera identification

While the first set of tasks are those which may be carried out by skilled forensic scientists, albeit with limited speed and potentially reduced deductive features, the second set are uniquely computational operations.

In an increasingly automated world, the advance of the close circuit television camera has been hailed by specialists and lamented by civil rights activists; while the concerns of privacy in public places and work spaces are worthy if serious discussion, the notion often persists that such facilities are perfect and able to distinguish people and objects, label them and manipulate this data. In reality however, the images produced from wall/column-mounted CCTV cameras, especially first and second generation digital CCD cameras are far less sufficient to the task than is perceived. The use of such evidence in criminal investigations frequently requires that a judgement be made as to whether a grainy image identifies a reported scene or person sufficiently, and while much of this information is presented to magistrates for criminal prosecution (90% of minor cases), more serious cases are presented to a Crown Court and jury [HAM]:

> *"So, it is the jury not the judge which reaches a verdict on the guilt or innocence of the defendant. In criminal cases, the prosecution has the burden of proof - it must prove guilt, rather than the defendant having to prove innocence. The standard (= level) of proof is heavy - guilt must be proven beyond reasonable doubt."*

In these cases, a noisy CCTV image will clearly not suffice to identify the defendant in an action or location. In these situations, clearly reproducible and scientifically-based image enhancement techniques are an important tool to provide unambiguous higher-quality imaging.

With contemporary developments in computing, image manipulation and digital photography and video, however, the opposing question of the veracity of imaging evidence presents itself; if an image might be manipulated and enhanced to highlight elements for criminal prosecution, might not standard images be altered to misrepresent information to protect criminal activity, masqueraded as fact? Photographic manipulation has always been a present problem in the verification of evidence, but microscopic analysis has typically sufficed to identify forging; in the case of digital imaging, however, pixel-level accuracy is possible in manipulation, making an edited image perceptibly similar to an original. These problems make the verification of taint-able image evidence a key hurdle to legal acceptance.

Machine Vision and Image Processing techniques capable of examining data in ways beyond conventional linear continuity checking are able to offer competent scientific means to ensure the veracity of digital and analogue imaging for evidential purposes. Methods such as artefact examination, lighting discontinuity checking and perspective/distortion compensation are able to provide high-level tools for discovering tampering within image data, through video sequences and ranges of images [GER]. While it is still insufficient to ensure that evidence is completely trustworthy, it does provide the degree of trust that was previously true of microscopic examination of print photography.

In many ways, the provenance of data is one of the most problematic questions of the networked digital age, in that documents may be easily copied and forged, and yet there exist few tools capable of providing secure and complete chains of trust. Means such as public/private key cryptographic systems based on factorisation of large primes are able to

help as a short-term measure, but quantum computing is capable of ruining such techniques, and attaching a true key to an unencrypted forged document is still an issue unless rolling and interference key generation is introduced.

2.6.9 Personal Identification

Associated in many ways with the use of Machine Vision in surveillance, is the task of visual verification and personal identification. This can be instituted as part of a number of discrete scenarios:

- Searching for people within crowds
- Identifying people in photographic images
- Logged and verifying people at security/access checkpoints
- Corroboration with documents (e.g. checking against passport images)

These tasks are traditionally those performed by a security officer, but frequently the circumstances in which such tasks are performed have outgrown the capabilities of basic human inspection. A suitable example of such a dilemma of examining the photographic ID cards of employees at the gate of a large technology firm's main campus. With the advent of information-based industries, employees working in front of computers in cubicles and small spaces, the density of personnel and their shear number in the case of multi-national corporations means that up to 10,000 people might work at any one site during a single day. Institutions such as universities and governmental organisations can often raise this number to 25,000 or higher (the approximate number of students at Cardiff University, for example). Clearly, this number of faces is beyond the capability of a single security officer, or a group of officers to validate. In addition, supposing only simple validation against a photo ID card, lacks a number of safeguards:

- It fails to log entries and exits
- It might confirm that a face matches a photo, but it does not confirm that the card is valid
- It similarly does not raise an alert when a false photograph is forged onto a valid ID card
- It does not support the ability of security officers to examine behavioural change and take advantage of their prior knowledge of the subject

In large-scale cases, therefore, some system of databasing and collective knowledge must be built to ensure a higher level of security. In point of fact, while US airports have had significant issues in producing systems able to identify faces of people deemed to be security threats in the midst of crowds, there has been more advancement in the second generation of systems produced for this security industry, with the development of carefully lit individual-specific checkpoints at the US Immigration and Naturalisation Service (INS) desks at the aforementioned airports. As of October 2004, foreign nationals entering the United States are subject to fingerprinting (automated and compared against international databases) and having their picture taken by a webcam-style camera, subsequently compared to a similar national database with international links to legal agencies. This along with passport details and inclusion of previous flight data supplied by the airline companies enables detailed tracking of persons travelling to and from the United States. Later improvements to the system promise to include retinal imaging.

More broadly, the United Kingdom government is attempting to introduce a system of personal ID cards, identifying an individual and including encoded data for verification, such as:

- Finger prints
- Voice prints
- Iris scan
- Facial thermal images
- Palm prints
- Thermal hand images

The intention of the ID card is to provide biometric verifiable data to confirm an individual's identity and also to unify presently disparate forms of identification (birth certificates, passports, credit cards, photo cards) into a single ubiquitous card which may be carried by the person. The person may then be assured that any company, organisation or individual wishing to identify them may use the ID card, rather than some other form of verification or ID. Prototypes of such cards mimic the technologies presently in use for the new biometric passport style, although the complete national networked database and connectivity systems have yet to be developed, leading dissenters to argue that the somewhat nebulous nature of the proposal and the ill-defined scope of the project may lead to failure.

A universal ID card, however, has other potential benefits, as a means of identification for both person-to-person challenges and also electronic testing. It does not take a great deal of imagination to envision automobiles capable of being unlocked (and perhaps started) either by key or granting access to authorise persons based on their ID card (indeed, cars already unlock using simple codes transmitted from electronic key fobs). Moreover, if the price of biometric scanners and card readers fell sufficiently into the realm of commodity computing (and finger-print reading mice are readily available at present), regular computer tasks could validate the user at a terminal and either act upon that data directly, or use it to unlock a personal password collection, able to grant access to web sites, programs, databases and documents.

In an increasingly security and trust-focused information environment, there are regular needs to ensure authenticity of a person and their works. As a result, computer users and more general workers find themselves constantly challenged for passwords and identification.

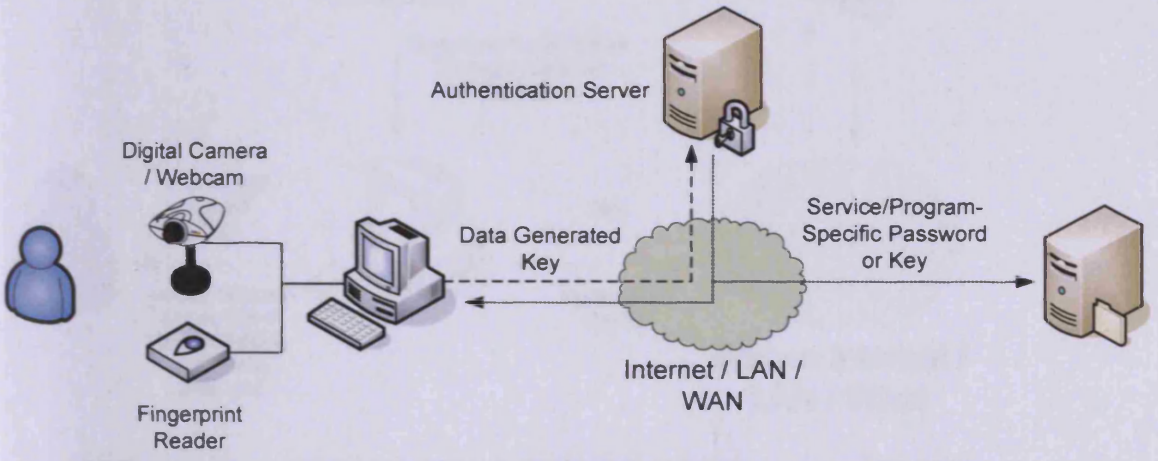


Figure 12 : Establishing authentication through mutually trusted third party networked systems

In short, security measures, already overwhelming for security personnel begin to become overwhelming for those being examined as well. Methods for aggregating security details together and producing one manageable form are ongoing in computing environments, stored as local and networked collections with strong encryption against single passwords. Examples are such as Microsoft Corp.'s Passport initiative and KDE's KWallet system. The means for authenticating and opening the collection, however, are becoming ever more sophisticated, including passwords and increasingly, biometric analysis. Presently efforts are underway to

provide systems across a network, using authentication such as fingerprint analysis [NOV] through mice or dedicated peripherals and accessible as a network service (see fig.12).

Systems of this type are presently in production with Novell, although Microsoft's Passport mechanism has stalled due to customer resistance.

Reliable personal identification has more computing-based uses in addition to resource access; the increasing need for document provenance poses problems with respect to true validation, where keystroke logging software and password insecurities are insufficient to legally ensure that a document was authored or handled by a named individual. Heightened awareness of unsolicited email and email-bound worms and

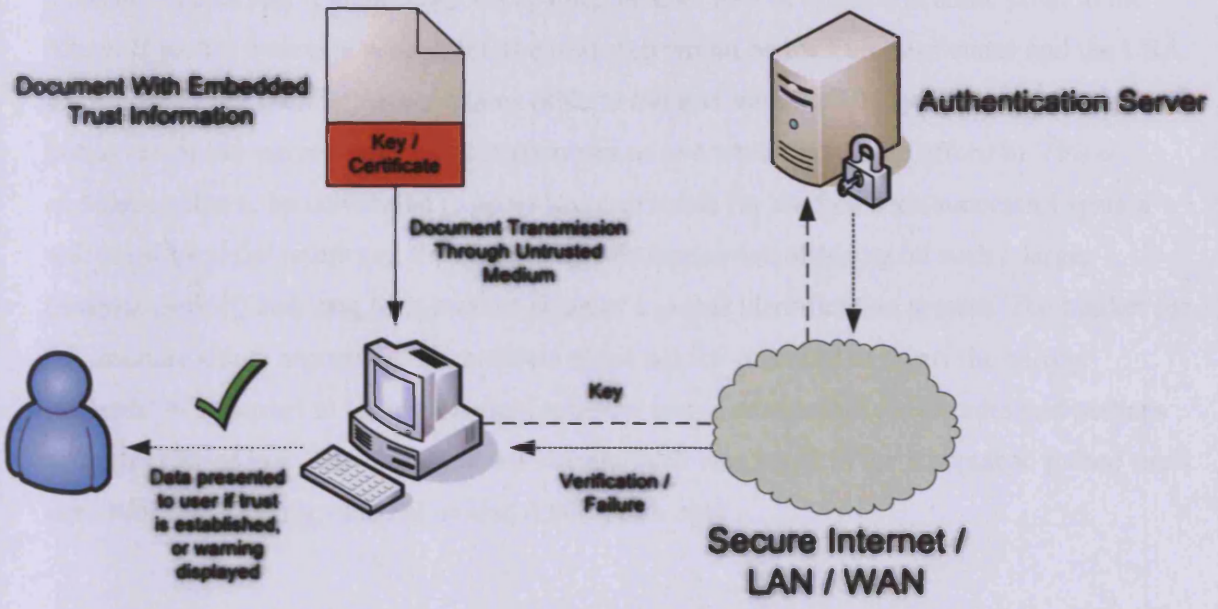


Figure 13 : Utilising trusted 3rd parties to verify document/data origin and integrity

Trojan horse software expands the needs for the use of security certificates, chains of trust and biometric validation to mark the origins of messages and establish their level of acceptability (see fig.13).

The problem of personal identification both physically and electronically is increasingly massive in the modern interconnected, security-conscious, data-driven world. Biometric identification provides a general mechanism for individual verification through an association of identity with the physical actuality of the person's own body, the most definitive method of

assuring identity short of full DNA comparison (which is not feasible for instantaneous tests at this point in time). Combining biometric information contained within a single ID card object, a national authoritative database capable of validating the veracity of that ID card (checksum + certificate) and a system of shared secrets will provide the capability to physically identify a person at a checkpoint or electronically for access provision, or even for electronic access to computing devices. Because of the scale of such a task, only a government or a limited number of large companies have the capacity to undertake the project and provide the system validation, security and scaling required for a national network able to deal with billions of transactions per day. Whether the present government is able to achieve such a lofty goal as a single, ubiquitous biometric identification system is debatable, but the desirability of such a system for the wide range of security tasks in everyday living alone tends to indicate that if technology is capable, the goal will be realised at some point in the future. If such a system is successful, the next step would be for European states and the USA to introduce their own implementations (able to transact with the UK system), and then global initiatives would see roll-outs to most countries as and when they could afford it. Vision companies able to be contracted to provide components for the first such successful system will see substantial return (on the admittedly substantial risk of taking on such a large, complex project) and long term market-share of a global identification system. The market for the machine vision aspects of this problem alone has the potential to dwarf the current industrial MV market in terms of annual turnover and is sustainable for decades and perhaps centuries (for context, the oldest known key and lock was found in the Khorsabad palace ruins near Nineveh, Assyria, dated at around 4,000 years old).

3 Existing MV Systems

This chapter provides an outline with advantages and limitations of present Machine Vision (MV) systems for use in prototyping tasks, their application to five types of problem classes found in exploratory MV:

- Examining & measuring products
- Monitoring & controlled acquisition
- Management control
- Prototyping
- Education & Training

After considering present MV solutions, the ability of these systems to be networked and provide efficient networked vision operation is examined through the use of remote control mechanisms such as VNC. The limitations and overheads of these methods are illustrated, displaying the need for more direct and remotely controllable vision systems for confederated and uniform solutions across large distances.

Existing MV systems can be defined as being of one of four key classes in terms of their intended operation. While there do exist systems which do not fall within these categories, at the current time the majority of MV systems do conform to one of these. It is using these cases that we may form the basis of comparison of Myriad to contemporary solutions.

As well as considering Vision systems to be categorised by the form of task they perform, we also separate them into Target and Prototype systems, reflecting the stage of development of the system itself.

3.1 Classes

3.1.1 Examining & Measuring Products

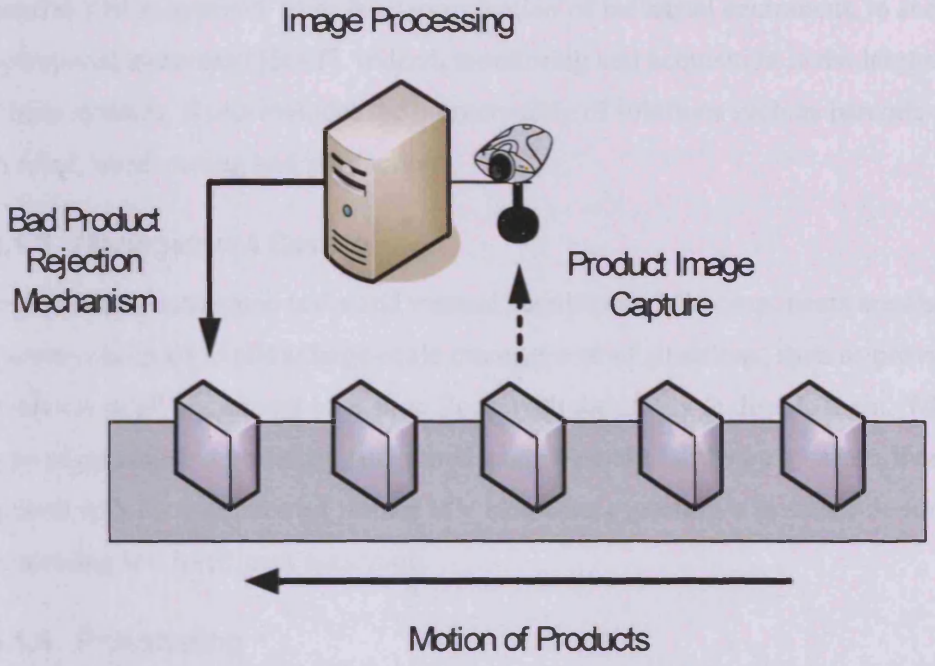


Figure 14 : Standard quality control visually-acquired accept-reject system

The process of examining objects for quality, unique measurement and uniformity is the most significant current class of MV systems supplied and operated in industrial situations. Most often, these systems form part of a direct manufacturing system and examine objects as they pass along a production line, attempting to search for defects, maintain manufacturing quality by rejection (see fig.14) or feedback mechanism, or to repetitively quantify a given object metric [BAT2].

In many industries, there is high potential for complex products to become flawed in manufacture, with also extremely hazardous results [BAT3]. Often, MV systems are used to replace human operators and line examiners, who, while skilled at their jobs, are incapable of perfect accuracy and assurance. MV systems are often used to replace such jobs, with the promise that while systems may not be perfect, they are better. The merits of such an argument depend on circumstance and implementation, however.

3.1.2 Monitoring & Controlled Acquisition

Where measurement may constitute the majority of deployed MV systems in manufacturing scenarios, monitoring tasks contain the largest variety of operations and form the majority of general Vision systems, from fault examination of industrial equipment, to security and operational awareness [BAT]. Indeed, monitoring and acquisition is the largest category of Vision systems, if one includes the commonality of solutions such as barcode-reading systems in retail, warehousing and production.

3.1.3 Management Control

In addition to automated tasks and manual acquisition, MVcomponents are also used in systems designed to allow large-scale management of situations, such as providing an overview of all equipment on a shop floor, with the ability to disable them. While most of this type of operation is presently performed using bespoke hardware systems, there is room for growth with the inclusion of further MV elements, especially a greater role for image processing and intelligent reasoning.

3.1.4 Prototyping

Prototyping systems are those that are designed to operate in a more interactive manner than typically automated MV systems. With a highly interactive and exploratory objective, these systems are intended to assist in such tasks as evaluating new product designs, visually troubleshooting equipment on-site and analysing unknown objects (e.g. searching for faults in samples of a given product).

Due to the unique nature of these situations, prototyping systems often consist of:

- User interfaces with fine granularity of control (typically command line control of single instructions).
- Support for multiple video streams or multiple image loading.
- Image store and instruction/image history
- Macro/script creation and testing

3.1.5 Education & Training

The uses of MV in educational environments are various and involve two forms:

- Demonstration of MV systems
- Demonstration of equipment and objects through remote or controlled interaction

One of the first areas of application of MV technologies, is the use of MV tools in teaching MV and Computer Vision techniques at a Higher Education level, or as part of a professional development and training programme. Being a experimentally-driven endeavour, MV system creation, algorithms and equipment control systems are best taught with the assistance of interactive image processors and equipment control systems, such as LabVIEW, JIFIC or custom-made interfaces. Since MV problems can very rarely be solved through the pure use of theory without experimental application, at least by modular parts, it is insufficient to teach persons new to image processing and Vision techniques in a purely paper-based manner. The complexity of MV command sequences is a product of the variability of the data involved, and the understanding that a means of identifying elements in images, and working with them may not be satisfactory in all cases and that this type of failure can only be discovered through the testing of sample data interactively.

Beyond the teaching of Vision itself, there is also a need to be able to demonstrate equipments and objects in a manner than students are otherwise deprived of, be that through space constraints, safety, of the simple inappropriate nature of certain materials to be viewed out of their context, in a classroom environment. For example, a 5 tonne production line pumping system, which fills bottles and caps them could not normally be moved to a classroom and shown in operation. Indeed, because the pump works through interaction with other industrial systems and a number of conveyor belts, it does not make sense to educate someone as to the equipment and its regular operation without being able to see it working within the context of the production site. Using remote cameras and equipment control, a teacher is able to articulate to students the operating procedures involved in the equipment, show it working, with an appreciation of the movement, noise, environmental impact and is able to potentially interact with the machine to the authorised extent. As a result a fuller impression of the pump and its activity are available to students.

3.2 Prototype Systems

In terms of the lifecycle of a Vision system, most solutions begin with a developmental stage in which the Vision Engineer is attempting to logically understand and solve the given problem, and there is a very high degree of interactive processing involved. These systems are usually as a result of consultancy and include both feasibility samples and actual development of the final system.

In the prototype stage, MV system engineers place more emphasis on software design, where they attempt to form the appropriate algorithms and methods for the problem, with extensive logging and data processing. In this stage, it is not uncommon for the majority of a project to be software-bound, using equipment other than the final hardware, whilst that is being developed, with the two being merged to produce a target system. Prototyping and Education & Training systems may be considered to be perpetually prototype systems, for the most part, as they are usually under continuous development and are deployed and used in their developmental state.

3.3 Target Systems

Once a prototype system is complete, it usually undergoes a transition to a more hardware-based version of itself, with algorithms added to embedded systems and custom logic, producing durable hardware forms of the software model, intended to operate in harsh environments, such as those encountered in industrial settings.

The Myriad demonstrations provided here do not attempt to be target systems; rather they form a prototype system where components may be successively replaced by hardware components to produce a target system. It is worth noting, however, that with the quality of modern operating systems (such as Linux) and server applications (such as Apache and proFTPd), high reliability may be achieved with software, to the point of comparability with hardware systems. With the advent of small, low powered computing devices with no moving parts and embedded fully programmable hardware, there is very little difference in hardware reliability between a custom logic board and a generic x86-compatible host. With the added cost of custom hardware and the difficulty of correcting errors in custom logic, it could be argued that modern embedded generic systems are preferable; although this varies, depending on the situation. The fundamental question becomes *"If I write a custom system for a PIC or other programmable chip, am I more confident of that code than if I wrote a program to run atop a modern (for example) Linux kernel?"*

3.4 Existing Solutions

3.4.1 PIP

Prolog Image Processing (PIP), is a development of Professor Batchelor of Cardiff University. Building upon past image processing systems, PIP was designed to be an interactive processing system for use in prototyping tasks. Using a command line as the primary means of interface, with three letter mnemonics representing instructions (thr for threshold, neg for negate, etc.) and Prolog as a means of creating scripts, with intelligent reasoning, PIP still finds use as a primary exploratory tool for consultancy.

While PIP has been superseded by CIP and others as a primary development area, the combination of Prolog and image processing has not been adequately recreated since. The desire to include aspects of PIP into Myriad may be seen by the continuing use of the existing mnemonics associated with operations duplicated from CIP, which in turn were previously derived from PIP. In addition, there is a effort to develop a Prolog component for the Myriad sample server set, to continue the level of functionality enjoyed in PIP.

3.4.2 CIP

Developed by Mr. George Karantalis as a student at Cardiff University, CIP is an Image Processing package, written in Java. Designed for teaching and prototyping, CIP provides a GUI with an embedded command line interface for interaction and windows for image display, file browsing, histograms representation and visible image storage.

CIP works around the same image/alternate image construction as PIP and inherits the same command acronyms and method of operation. Essentially, CIP may be seen as PIP rebuilt for operation on a Java Virtual Machine, without the Prolog scripting abilities.

Through the efforts of successive students and members of staff, CIP gained further operations that extended the repertoire of commands beyond that of PIP. Examples of these functions are: histogram analysis, additional blob functions and dimensional integration.

Due to the nature of the Java code, wrappers have been written to interface CIP commands with Myriad::IP, the Myriad Image Processing server initially written in Java. This was the

most expedient way to gain functionality and actively demonstrate use of the Myriad framework.

CIP/Prolog

Further to the initial capabilities of CIP, the ability to script the application using Prolog was added through the introduction of CKI Prolog, a Java-based Prolog interpreter written by Sieuwert van Otterloo of Utrecht University. This added much-needed scripting, macro creation and intelligent control facilities to the system, which were otherwise lacking and preventing the solution of complex problems. Good examples of Prolog-requiring operations are:

- Packing of 2D shapes
- Examining validity of table place settings
- Deducing playing card suit and value

3.4.3 WIP

WIP is a reimplementation of PIP/CIP in the form of a Windows DLL library by Miss Melanie Lewis, student of Cardiff University, able to be used directly by COM-based applications. Produced in Visual C++, WIP shared the mnemonics of CIP and earlier programs of the same family. WIP may be readily understood in the role of an image processing library for use with Windows-based GUIs and applications. It does not seek to provide a UI itself, but rather, to be incorporated into larger solutions and bespoke software for MV tasks.

3.4.4 Neatvision

Created as an image processing system with broad developmental and prototyping abilities, NeatVision is a package created, also using Java by the Vision research group at Dublin City University, Eire. Taking a different direction from established systems with command line interactive interfaces, NeatVision uses a visual canvas space to connect functional blocks. Each functional block represents a given operation, with defined numbers and types of inputs and outputs, which are connected together by the user. Once the construction is complete, the sequence may then be run through and results inspected at each stage of the process.

It is perhaps most valuable to consider NeatVision to be a script development environment for image processing tasks. While there are benefits to this view of development, there are

limitations, principally in that the user does not achieve immediate results of changes until the entire system is run, limiting speed of response and user appreciation of changes and flow of operations. One of the key advantages of NeatVision, is the extensibility of the system, with the user able to design their own Java classes to perform functions and use them directly in their NeatVision work area. In this sense, NeatVision can conceptually develop into a generic visual scripting environment for any given computational task, and indeed, a progressed form of such a system may be found in Cardiff University's own Triana[1] package, or also in LabVIEW's visual constructor.

During a visit to Dublin in 2003, the author assisted the development of Neatvision with the introduction of HTTP download and control mechanisms. With these developments, Neatvision is now able to control Myriad components and complete systems, although the design of Neatvision currently precludes any form of scripting, remote or autonomous control, preventing it from being controlled by Myriad components.

3.4.5 LabVIEW

LabVIEW is a collection of tools designed to both control and manage data flow between devices, especially in research and manufacturing contexts. Beginning as a tool for creating control GUIs for local devices and for local automated control as reported by Johnson and Jennings [LAB], it has grown to include data reporting to databases and web sites. Most recently, it has developed the ability to control networked devices, albeit through custom driver modules, which must be incorporated for each individual device.

LabVIEW operates as an environment for connecting together and processing data from what it terms 'Virtual Instruments' (VI). Each VI is a custom-written driver for a piece of hardware or software, which mimics the interface and/or capabilities of the device. Johnson and Jennings describe VIs as:

> *"The objective in virtual instrumentation is to use a general-purpose computer to mimic real instruments with their dedicated controls and displays, but with the added versatility that comes with software."*

[1] Triana: http://www.trianacode.org/ (March 2005)

The primary development environment for LabVIEW is the RAD (Rapid Application Development) user interface designer, where VIs are connected together with each other and additional operations added to process their data, building a complete program visually. Underlying the visual programming environment is a programming language named 'G', which can be accessed manually to add non-visual logic. Using the RAD designer of LabVIEW in comparison to C for device control has enabled significant increases in the field of rapid prototyping of device control systems, up to 5 times the relative productivity [WELL2].

More recently, LabVIEW has added compilation and bindings for development subsequent to the creation of the UI and basic device connections, using Microsoft Corporation's Visual Basic or Visual C++ to add required custom logic.

As a result of this development process, LabVIEW is ideal for the market of user interface development for fixed equipment and integration with existing custom business software, written for Windows. One of the key advantages to such use, is the opportunity to reuse skills required for the latter, producing rapid development processes for experienced developers and the creation of relatively light-weight interfaces.

LabVIEW presently operates on Windows, MacOS, Solaris, HP-UX and under virtual machines in Linux.

3.4.6 Vision Pro

The Cognex Corporation began, as mentioned in Chapter 2, as a firm dedicated to developing systems for Optical Character Recognition tasks. As the field of industrial MV developed, they began to produce cameras, framegrabbers and embedded image processing engines on logic boards. Over time, these systems became integrated into a number of their camera products, providing 'Intelligent Camera' systems, able to capture images and perform image processing tasks autonomously. While intelligent cameras are highly useful in producing secure and reliable target systems, there are also benefits from using a camera with a host computer to perform processing:

- Processing power of commodity computing devices may not match the abilities of DSPs for specific linear processing needs, but they do outperform DSPs for complex logic with other I/O and data dependencies

- Using a host computing device, image processing and camera control may be integrated into a GUI application for ease of use for operators
- From a host computer, sharing data with other local applications may be more easily programmed and managed. A vision system that monitors a scene but is unable to affect that scene or tell any other person of device what it is seeing is redundant

Therefore, Cognex offers a range of framegrabbers and digital cameras, along with a development suite for vision software applications, named VisionPro. From the brochure for the product:

> *"Cognex ® VisionPro® systems combine the world's leading machine vision technology with the most flexible and powerful PC-based vision application development. VisionPro makes it easier than ever to rapidly build integrated PC solutions for the most challenging machine vision applications. The extensive Cognex vision tool library provides extremely robust, reliable, and repeatable performance while the VisionPro acquisition engine supports a wide variety of Cognex framegrabbers and industrial camera options."*

VisionPro operates under Windows (9x and NT/2000/XP) and provides a range of features:

- Rapid creation of generic applications through interface creation wizards
- Programmer documentation with example solutions for a wide range of vision applications (inspection, location, identification, geometry reconstruction, etc.)
- ActiveX controls for:
    - Equipment interfaces
    - Image processing operation packs (pattern matching, targeting, geometry, image analysis)

Using VisionPro, a programmer can develop an application using the in-built tools and Microsoft's visual development software, such as Visual Studio: C#. Components are referenced and imported, and integrated with conventional libraries, third party components and GUI elements, such as forms, buttons and menus.

In many ways, VisionPro mirrors the development of LabVIEW, except that rather than focus on ActiveX device control components, it provides control for Cognex's own equipment, and

then expands the repertoire of image processing and vision tools for image analysis; it emphasises vision analysis functions, rather than hardware control.

Comments on the use of VisionPro as a tool for networked MV tasks are highly similar to those of LabVIEW. It provides a basis for the programmer of a solution to take development as far as they like within the construction facilities of Microsoft's toolbox. It does assume that the framegrabber is local to the computing device, which must be an x86 personal computer running the Windows operating system. For these reasons, VisionPro is ideal for the creation of vision solutions with image processing needs close to the scene of inspection; e.g. for the stand-alone computer next to the production line for manufactured products monitoring quality of output. It is not ill-suited to the task of remote vision systems, but it does not assist the programmer in any substantial way in overcoming concerns of networking or communication; that is up to the developer themselves to implement.

3.4.7 Matlab

As described in chapter 3, Matlab is a mathematical development and prototyping environment created by MathWorks Inc. centred around a procedural programming language with C/C++/Java bindings with an emphasis on the use of functions to support the development of non-linear scripts.

Matlab supports a wide range of expansion 'Toolkits' which provide targeted operation libraries for particular tasks, such as:

- Image Processing
- Drawing 2D/3D
- FPGA design and testing
- Fluid system modelling
- GPS data analysis
- Firewire image acquisition

The number of MathWorks and third party toolkits numbers in the hundreds, including visualisation, device control and complex modelling. Many of these are built as compiled binary libraries with Matlab bindings for speed, rather than use Matlab's own interpreter for performance-critical applications.

As a consequence of these toolkits, Matlab is highly extensible to the user and enables the development of prototype applications within the interpreted environment. Matlab code (stored as text files with the .m file extension) is able to be compiled into C or directly into binaries for the given computing platform, provided the developer has a suitable C compiler installed. Unfortunately, the compiled code:

- Is much slower than the equivalent written in C (the M file -> C meta-compiler tends not to be that well optimised and provides generic solutions, incorporating most of the Matlab core libraries, even when they are unused)
- Requires a unique Matlab license to be paid for in the case of each deployed computing device
- Requires the installation of the Matlab core libraries on each deployed computing device
- Is resource (CPU/memory/disk) heavy in operation

Prof. Batchelor at Cardiff University has ported the CIP image processing command set to Matlab, adding a number of features such as Internet image downloading; this library of functions has been named 'QT' by Prof. Batchelor, but to prevent confusion with Trolltech's Qt (lower-case 't') toolkit for C++, we shall henceforth refer to this as Matlab-CIP.

Mr Simon Caton of Cardiff University has continued this work to include a small Java application which acts as a front-end to Matlab and presents an HTTP interface. This enhancement allows Matlab to be sent data as HTTP requests and return results. The Java acts as an interface and passes formatted data between Matlab and the client. Mr Caton's work occurred after the bulk of the Myriad demonstration software was produced, as a way to replace the Myriad image processing server with Matlab as an IPE. While it does not use identical syntax to the Myriad server, it is sufficiently similar that it may be replaced in a networked vision system with minimal code alterations and therefore may be considered to be an acceptable part of a Myriad solution and compliant with the framework. Thanks to Mr. Caton's contributions, Matlab should not be considered to be at odds with the objectives of Myriad, but rather a valuable extension to the concept which allows for the inclusion of complex engineering prototyping tasks into networked systems, especially those focused on MV. It should be noted however, that this networked Matlab/CIP is not multi-threaded, and can only deal with one user at a time, hence is not suitable for use as a universal networked image processing server for multiple concurrent tasks.

3.4.8 Super Server

In addition to his work on Matlab, Mr. Caton is also presently developing a piece of software named 'Super Server' (SS), which acts as a central pass-thru, indexing and load-balancing server for multiple Matlab/CIP IPEs, residing on different machines. To best illustrate the intention of the Super Server, an example is helpful.

When the Super Server daemon is running, it listens for client network connections, but also for new Matlab IPEs starting up across the network. When these start, they notify the SS daemon and it maintains a list of currently available Matlab/CIP instances. Any user wishing to utilise the services of a Matlab IPE, therefore, connects to the Super Server, which then allocates the user a particular instance from the stored list of servers and passes data back and forth between the user and that server. The user communicates with the SS daemon, which communicates with the Matlab/CIP server and when data is returned, it passes to the SS daemon, which feeds it back to the user. This process is necessary, due to the single-threaded nature of the Matlab IPE, allowing many users to connect to a single computer running the Super Server, and each of them still having full access to an image processor, without having to be queued.

Aside from differences in scripting language therefore, the combination of a set of Matlab/CIP IPEs and a Super Server is identical in a network sense to a Myriad multi-threaded image processing server.

3.5 Limitations

Fundamentally, LabVIEW, Matlab and Neatvision do not have direct limitations, in that built programs can be as extravagant and far reaching as the developer wishes (thanks to the inclusion of Visual Basic/C++ or Java, limited only by the skills of the programmer in those development systems).

The freedom of relying on others' programming skills to create complex applications, is however, merely a process of moving responsibility for implementation from the tool to developer. While this is useful for small tasks, this obligation becomes less beneficial and more arduous for the programmer as the scale of a project increases; as this occurs, the benefits conferred by the toolkit fall as a ratio of complete development time (the definition of

a toolkit being that it actively attempts to comprise of code that the developer would otherwise be forced to write themselves).

LabVIEW

Since LabVIEW can, in many ways be seen to be a competitor to Myriad for networked MV problems, it is essential to examine the ways in which both systems tackle the problem of remote inspection of a potential MV installation.

Taking the case of a system with a remote device, LabVIEW has two options:

1. If the device is network controllable, and a LabVIEW VI exists (see fig.15)

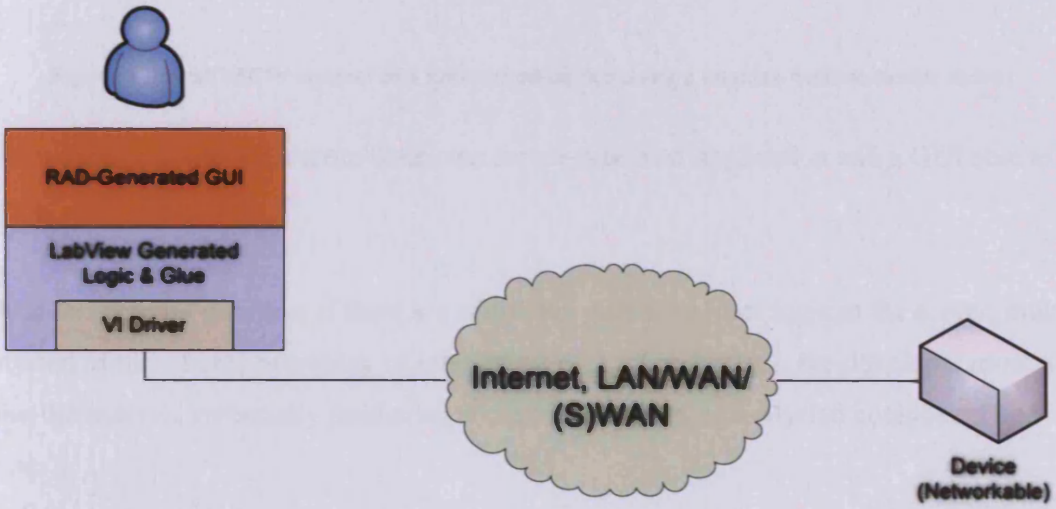


Figure 15 : Simple control of a remote networked device

2. If a device is not network controllable, or a VI does not exist (see fig.16 and fig.17)

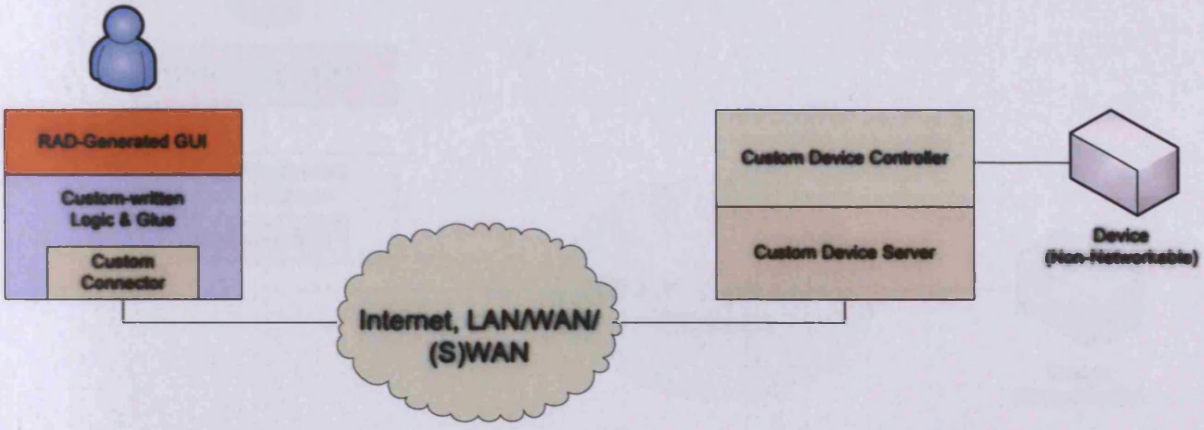


Figure 16 : LabVIEW control of a non-networked device using a local custom-written network host

Or:

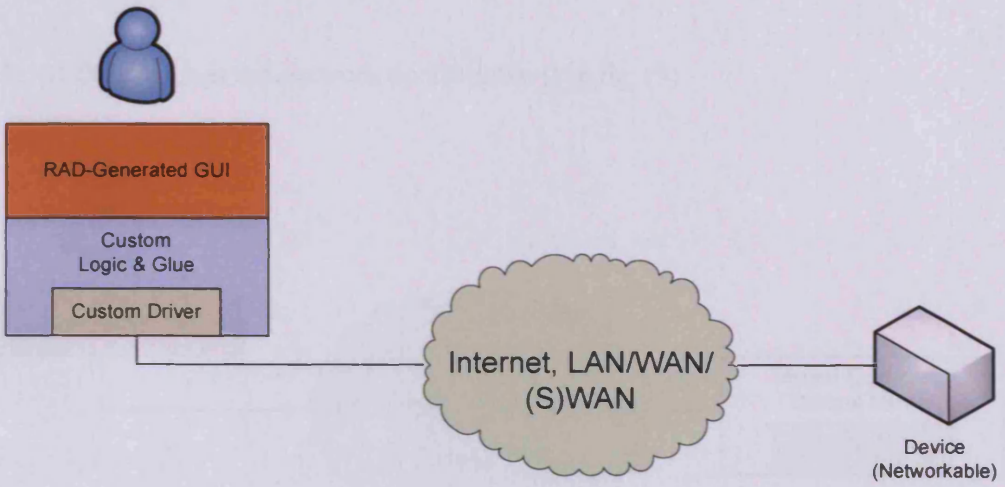


Figure 17 : LabVIEW control of a networked device using a custom-written device driver

Where the developer must write their own server-type host application and a GUI able to engage and operate this.

While this is quite possible, if there are additional needs for local logic in the server, multi-threaded (multi-client) operation, or integration with other systems, the developer must write these themselves, essentially producing something resembling a Myriad component and client.

Meanwhile, using a Myriad system for the same operational situations would result in the following solutions to the same problems:

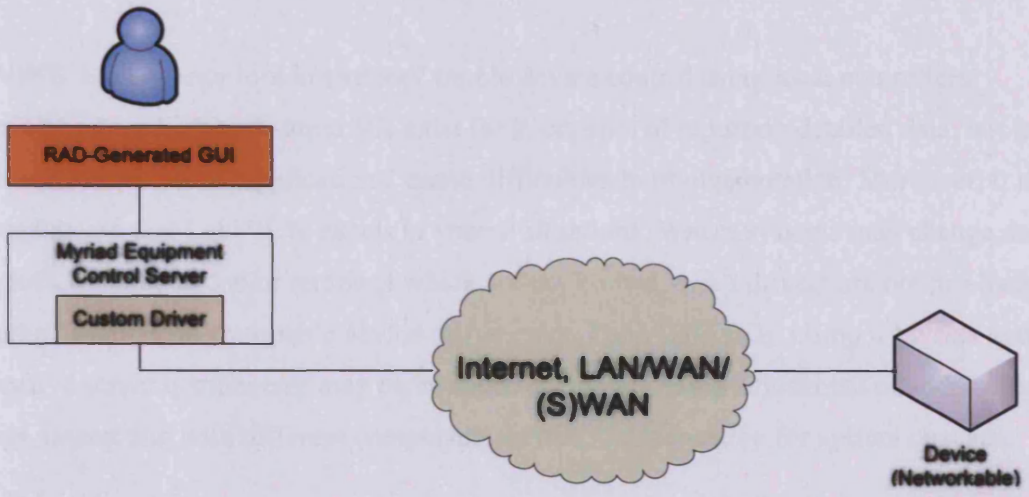


Figure 18 : Myriad control of a networked device using a custom-written device driver incorporated into a standard control server

1. If the device is network controllable (see fig.18)

2. If the device is not network controllable (see fig.19)

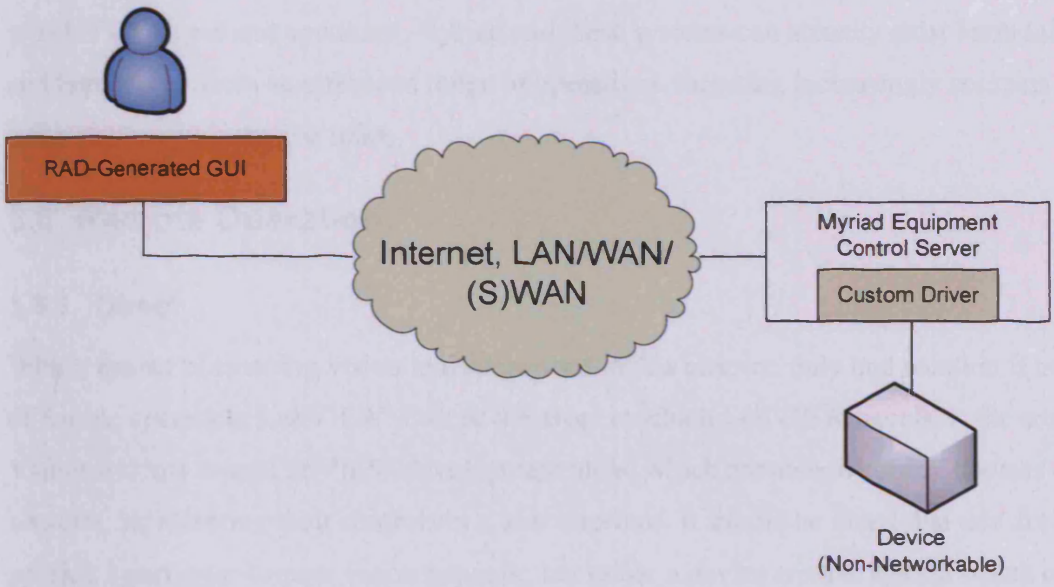


Figure 19 : Myriad control of a non-networked device using a custom-written driver incorporated into a standard control server located locally to the device

Comparing the two, it is clear that LabVIEW is more elegant in the first case, although Myriad is functional and capable of solving the problem. In the second case, however, the ability of LabVIEW to solve the problem is entirely dependent on the skills of the application programmer (and few application developers are able to write complex server applications).

LabVIEW is a superior tool in terms of simple device control using local controllers, especially where highly-featured VIs exist for it, capable of reporting detailed data; but large-scale networked 'Meta-Applications' cause difficulties in implementation. Moreover, it must be emphasised that LabVIEW excels in known situations. Where systems may change during use, devices being added or removed which are not known, or VI drivers are not pre-loaded, or part of a universal swappable device architecture, LabVIEW fails. Using a Myriad system, alternative server components may be swapped at runtime using adjustment of addressing text strings, interacting with different component servers, compensating for system changes.

Matlab & Neatvision

In comparison to Myriad, Neatvision and Matlab are not true competing concepts, as they attempt to address different problems, through the creation of generic development environments. As was expressed with discussion of Matlab and Caton's work to enhance it with HTTP access and operators, Myriad and these systems can actually exist harmoniously and together perform an enhanced range of operations, targeting increasingly complex and computationally-intensive tasks.

3.6 Remote Operation

3.6.1 Direct

Within the set of exisiting vision tools described in this chapter, only one solution is capable of remote operation; LabVIEW. One of the areas in which LabVIEW excels, is the creation of Vision systems using LabVIEW development tools, which are able to control devices across a network, representing their controls in a user interface. It should be noted that this does not provide a pure peer-to-peer vision network, but rather a device control system which remotely controls devices by a single controller node on the network. In many ways, this activity mimics the intelligent camera system discussed in Neve's 1991 paper [NEVE], except that intelligent cameras are replaced by hardware devices. Indeed, it may be considered that this form of development is the natural progression from a system with a controller and devices connected to a single bus system (e.g. RS485); the hardware connections are replaced by ethernet cable, and addressing is replaced with IP-based identification. When a user manipulates an element of a GUI, LabVIEW-developed applications are then able to call upon the VI component with the designated IP address of the hardware device, connect across a network to the given device and issue commands based upon that input, or additional processing. The limitation of this method, however, is that the devices being controlled, by their nature, must be known while the system is being constructed. In this way, LabVIEW represents a strongly-planned, hierarchical model for the management and control of devices across a known network, a set of devices and their controller. Myriad aspires to multiple controllers, peer communication between devices, components that are IPEs, databases, intelligent controllers and more, and most especially the option to control and script controllers to build larger networks and meshes.

3.6.2 VNC / RDP / X Windows

As an alternative to systems of direct control of peripherals across networks, there exists a method of control that allows a remote user to operate a copy of local computer's desktop environment within a window. This will, for example, allow a LabVIEW GUI controller that is restricted to the same network as the devices that it controls to be utilised from across the Internet. PIP, CIP, NeatVision, VisionPro and Matlab may also be transported and controlled in this manner. This form of network interaction is descended from the X Windows specification, which allows for network-transparent windowing, where windows may be transmitted in raw data form to a remote, logged-in user. The original intent of the system was to provide capabilities for the local distribution of interfaces to thin clients and dumb terminals, but still serves as a useful administration tool as windows may be displayed through Telnet and SSH connections, giving the administrator the impression of being at the local terminal. More recently, the limitations of X's raw window transmission have become apparent, as users have begun to use the protocol across wider area networks, such as the internet, or secure connections, where the encryption overhead per data packet is high (sending the entire 24bit-per-pixel framebuffer of a 1280x1024 desktop environment across a network requires a great deal of bandwidth and more so to update it frequently enough to be usable and responsive to the user). As an alternative to this, Symantec Corporation produced a product named PC Anywhere, which operated by transmitting a copy of the entire desktop as captured bitmaps to the remote machine, but with increased data efficiency through image compression. Microsoft's Windows 2000 product saw the introduction of the PC Anywhere code base into Windows itself, renamed Remote Desktop Protocol (RDP). With RDP being a closed standard and Windows XP's licensing requiring that only Windows itself be able to connect legally to an RDP server, a more open standard was promoted by Unix and non-Windows developers, VNC (Virtual Network Computing).

Originally developed in 1992 by Tristan Richardson and James Weatherall at Cambridge University's AT&T labs, VNC's development was outlined in their 1994 article [RICH], under the title 'Teleporting', describing a remote operating system window on a working system being 'teleported' to the screen of another computer to operate it, before being 'teleported' back. VNC operates under X Windows environments by providing a proxy server (typically residing on the same machine as the X windows server) which serves to manage interactions with clients, translates and compresses X windows for transmission. The proxy developed by Richardson et al. was named 'tpproxy' (teleporting proxy) and later became the

basis for RealVNC and a template for VNC servers for non-X environments, such as Windows NT. Subsequent to this, VNC was made an open standard, and has been continually enhanced by reimplementations of clients and servers since that time. VNC exists in a number of versions, with clients and servers for all major computing platforms, allowing for differing degrees of responsiveness, quality of image and security.

Limitations

While these networked systems are able to perform operations remotely, both by direct control and by remote desktop viewing as a portal to control, they fail to scale to larger systems and more general applications.

LabVIEW's use of driver objects allows it to interact with a very high degree of specificity with equipment, even over a network; however, it does this by addressing components directly, with full knowledge of what they are and where they are. It is not dynamic in a manner that allows new components to be added at runtime, nor does it allow for very large numbers of components to be used in a reasonable way. This is not to say that using LabVIEW's Visual Basic and Visual C++ components, such a system could not be created, or that an equivalent to Myriad could not be developed using LabVIEW's components as a basis. The result of such a process, however, would essentially be Myriad created in another language, rather than an enhanced LabVIEW.

Similarly, using a remote desktop system to control networked computers and use interfaces on those machines to operate equipment or do processing is also a poor solution, in that it exhibits even worse scaling for complex tasks. Using a remote desktop system logically requires a given amount of screen space per remote window, if not the complete screen, moreover, it becomes unwieldy with large numbers of connections (> 1). This impairs the user from responding in good time to multiple environments and makes the performance of large, multi-host operations problematic. Moreover, it may give rise to instances of the user interacting with the wrong remote host for an operation, due to confusion over which window identifies which host and what task each is supposed to be performing. Beyond the mechanics of the user attempting to deal with multiple large user interfaces, there are latencies created by the introduction of a user interface and the user's attempts to navigate the interface in situations where immediate response is required. This is exacerbated by the fact that each remote desktop stream requires a very large amount of bandwidth between the two machines

to operate, and it is very easy to saturate the network connection of a controller, inducing high latencies on top of the lag between updates from the remote host, which are substantial on uncompressed (X Windows) or semi-compressed (RDP, VNC) protocols.

Most of all, both of these mechanisms for remote operation of Vision components fail to satisfy in another key regard; they are non-scriptable. Once an interface is created in LabVIEW and compiled, it must be used to control the system and be the top controller. Similarly, PIP, CIP, VisionPro and NeatVision do not have any form of server-daemon always-on scripting functionality for remote control. Unless the developer deliberately creates a scripting system with a network-aware server attached to the user interface, the system may not be controlled by a scripting language, another application, or be integrated into a larger organisational system at a later date. Once again, if the developer were to create such a system, they would have to design a language and protocol for interaction, in essence creating a Myriad component. Indeed, Caton performed this exact task when he added an HTTP server front-end to Matlab (although Matlab already has a scripting language in place), thus enabling Myriad networks to use Matlab, and also be utilised by it.

In this chapter, four different types of prototyping machine MV system were described:

- Monitoring & Acquisition control
- Management control
- Education & Training
- Prototyping

Existing interactive image processors and software systems were discussed, including: PIP, CIP, WIP, JIFIC, NeatVision, VisionPro, Super Server/Matlab and LabVIEW.

The options for remote operation and connection of these systems were outlined (direct connection and VNC-style teleporting), with the limitations of these techniques described:

- High latency
- Limited automation
- Difficulty of interfacing with other software systems
- Difficulties control multiple systems simultaneously

3.7 XML RPC

SOAP, as described in Appendix F.10.1 is a messaging system for web services (web-server hosted applications that communicate using XML). Prior to the development of SOAP, however, Microsoft Corp. developed a basic XML messaging schema for use with early XML web applications, called XML RPC. As the name describes, XML RPC was envisioned as a means of controlling remote applications through a Remote Procedure Call (RPC) using XML as a format for the calling message. XML RPC supported a basic range of data types (integers, doubles, strings, arrays, base64 encoded binary blobs), but with a very loose set of rules that failed to assist in the development of rigid standards for interaction that would be required for the creation of homogeneous web service environments. Microsoft thus increased the complexity of XML RPC, ultimately producing SOAP, while the original development was abandoned.

In 2004, however, interest in a basic XML messaging standard was rekindled, through the necessity of improving bandwidth requirements for large internet sites, and developing highly-interactive meta-applications spanning web servers and web clients transparently. As a consequence, XML RPC began to be reused for small tasks (instantly updating shopping baskets on e-commerce sites, without reloading complete pages; synchronising small applications; transparent application updates). Since this use was typically in association client-side JavaScript interpretation, the association grew to the creation of lightweight, event-driven web applications which were supported by large numbers of small messaging threads, performing updates and responding to user input. The result of this development is a web application paradigm, named AJAX (Asynchronous JavaScript And XML), where a web client uses JavaScript to provide event-driven interfaces, generating XML RPC requests, which are sent to a web server (using PHP, Perl, JSP or ASP as a scripting language), tasks performed and data returned, to be merged with the operation of the JavaScript application. As a result, therefore, XML RPC as a system may be considered to be a messaging mechanism between a highly user-driven interface and server-side scripting.

As it relates to Myriad, while it does deal with messaging between a client and a server in a regimented fashion with respect to the format of messaging and the necessity for a web server, XML RPC presents much the same issues for creating MV frameworks that caused

Microsoft Corp. to abandon it during development in favour of SOAP. XML RPC benefits greatly from simplicity, but it lacks any form of specification over the procedure being called; it does not outline standard procedures that might be called from a server, nor does it provide any means of interrogating a server application to deduce what procedures are available and what they syntax might be. That is not to say that the author of a server application cannot create such an announcing method, but while the lack of specification does afford a great deal of freedom and rapid development for very small tasks, where the client and server applications are being developed concurrently by the same team, it forces larger application environments to be created by hand.

In addition to the need to develop features for XML RPC, rather than have the messaging format provide them, there is another point of divergence from Myriad, in that XML RPC is a messaging mechanism that relies on a client-driven model, rather than enabling server processes to be initiated and run autonomously. One of the key benefits of Myriad is that components may be scripted to overcome network connectivity problems, to ensure consistent processing, and to enable components to interact with each other to produce complex network topologies; XML RPC, being driven by client events, does not implicitly provide this. Once again, it is possible that such functionality may be developed for use with XML RPC, but in order to do so, a server-side application would have to be developed in PHP, Perl, Ruby or another scripting language that could implement a meta-interpreter. By creating a meta-interpreter, commands embedded in an XML RPC message could be utilised to execute server command autonomously. Session management, interfaces to specialist command sets, interfaces to external programs and inter-server communication mechanisms would also have to be developed; essentially, the developer would have to mimic the features of Myriad in whichever meta-interpreter had been constructed. The eventual conclusion of such a process, would be a larger server application with lightweight clients, communicating through the medium of XML, rather than HTTP GET requests; essentially identical to Myriad, but produced using a web server and scripting language, rather than as a monolithic server as in the Myriad Java prototypes. If the detail of means of implementation is removed from the discussion, it would be Myriad, using XML as a means of communication.

4 Designing Industrial Systems

This chapter examines the subject of industrial vision systems, the two primary types of system during development, the target systems, the final solution which is deployed into the industrial setting and the prototyping system (used to create and prototype vision systems, determining whether a system can be built, and how it may appropriately be done). Time is taken to explore the tasks to which prototyping systems are applied, and the modes of use employed in their operation.

Having outlined the use of prototyping systems, the '3M Problem' is discussed, where emerging vision task development requirements are greater than the ability of the industry to fulfil them. In order to overcome this problem, a range of tools are available, to expedite the development process for vision systems. These tools are summarised and interactive image processors are considered in greater detail, with examination of the key features of such applications.

Finally, the utility of networked vision for prototyping systems, and particularly interactive image processors is examined, informing the reader of useful aspects of such systems.

Machine Vision systems for use in industrial environments come in a number of forms, and typically involve several discreet stages:

- Conception
- Feasibility study
- Task/product examination
- Prototype System
- Testing
- Target System
- Installation & Maintenance

While the majority of this sequence will be familiar to the reader as a normal engineering development programme, some less familiar with the process might question the use of a

prototype system as well as a target system. Due to the nature of Machine Vision as a subject, tasks are not simply mechanical, nor software, but a combination of disciplines that develop complex interactions in many cases. Additionally, image processing components in deployed MV systems must be rigorously tested, lighting designed and controlled, equipment set up and operated for test data, and products tested in a range of environments to ensure that unforeseen changes between the laboratory and the industrial setting (such as the factory floor) do not change the task beyond compensation. A prototype system, therefore, is meant to mimic the resultant (Target) system, but with a wider range of capabilities and tolerances, to enable a range of testing.

4.1 Target Systems

A Target System (henceforth referred to as the TS) is the designed, completed Machine Vision system deployed in the industrial environment to perform a task. While it might very often be the case that a TS contains simply computing hardware, a camera and software, it is also possible that it might include:

- Customised lighting systems
- Ambient lighting regulation hardware (e.g. cowling to cover the system, altered bulbs or directional hoods for existing light fixtures)
- Specialised product manipulation equipment designed for use with the production line
- Physical protection equipment (e.g. lockable cages preventing workers from interfering with running equipment)
- Workers and adjusted worker roles and duties

This list is admittedly hardly exhaustive, but it does serve to illustrate that a deployed Machine Vision system is rarely a piece of computing hardware and a camera, isolated from their environment.

A Target System is classed as a federation of tools and mechanisms (including cameras and computing equipment) that provides the complete solution to the industrial problem.

4.1.1 Role

Target Systems are produced as a finalised version of a Prototyping System, with the intention of being deployed in the industrial environment and operating autonomously, or with minimal operator intervention.

Typically, while Prototyping Systems are slower, with a more diverse range of response and interactive potential, TSs are hardware-based and produced with a very specific range of operations in mind, especially in the case of Image Processing algorithms.

The following degree of contrast between the two is frequently the case:

- Prototyping System: Software-based IP operations, 1 operation/second
- Target System: Hardware (DSP, PIC) IP, 100 operations/second

It might be the case that the Prototyping System is able to examine one image per second and perform a resultant action. With a Target System, however, a full 25fps (or higher for a high speed camera) of video might be reasonably processed. Therefore, objects are set up in a laboratory, and the Prototyping System is used to discover how images of the objects might and ought to be inspected; then this process is rebuilt as a Target System with increased speed to operate in the factory situation.

4.1.2 Requirements

Target Systems frequently involve custom-built hardware for mounting and structure, with customised manipulation equipment for elements where the system must interact with the environment or products.

Specifically, however, the majority of Target Systems:

- Operate autonomously or semi-autonomously
- Are resistant to dust, vibration, noise and temperatures expected in the scenario
- Contain minimal moving parts (preferably none)
- Have a high Mean Time Before Failure (MTBF)

The precise requirements for tolerance will vary based upon the environment within which the equipment is meant to operate, but rugged custom-built chassis are not uncommon. Often, Machine Vision systems in industrial settings are designed to replace human workers, or

operate in situations where there are health and safety dangers. In these cases, therefore, the amount of operator interaction is intentionally low, although the degree of isolation varies and monitoring/input terminals frequently appear. A fundamental requirement of a TS, however, is to perform a repeatable function consistently.

4.2 Prototyping Systems

As mentioned previously, a Prototyping System (PS) is the initial developmental model which is produced in the analysis and solution of a problem, normally in a laboratory environment. It need not be quick, entirely robust, nor necessarily easy for the use of a lay person. The qualities essential in a PS, however are:

- Wide range of operations
- High level of interactivity
- Ease of use
- Scripting and macro facilities (software)
- Open access to all functions
- Step-through operation

The fundamental objective of a PS is to explore the problem, and through that experience, to devise a solution.

As a Machine Vision researcher, the author is not in the business of producing complete MV target systems; that job falls to commercial entities and companies such as Cognex. The author is, however, involved the investigation of new MV problems, many of which are unexpected and beyond the present range of deployed and well-understood MV solutions. For such work, a range of diverse and exploratory tools, without presumption as to the form of the problem are essential and significantly improve effectiveness.

A Prototyping System typically includes elements such as:

- Interactive Image Processing applications
- Functional and declarative programming tools (Prolog, Haskell, Lisp)
- Databases
- Generalised equipment interfacing tools
- Lighting advisor
- LabVIEW

- MATLab
- Range of lighting types and rigs
- Range of cameras

4.3 How Prototyping Systems are Used

The use of a Machine Vision Prototyping System is twofold:

- To understand and explore the problem
- To build a solution to the problem using generalised components

It is the second of these two uses that forms the basis for the specification and design of the Target System.

In normal Vision problems, the Vision Engineer (VE) will initially examine the situation and products, proposing a set of possible options based upon prior experience. For example, a task of examining the profile of a loaf of bread would lead the VE to consider using a laser stripe generator to take sections with an offset camera. At this point, the VE might set up an adjustable rig (such as the Flexible Inspection Cell (FIC) [BAT94]) with lighting and camera positions around the object, simulating the position of the bread on the production line. The construction process is guided by experience, but experimentation must take place with feedback from the camera(s) to devise the most ideal setup for the given product and situation. This setup would also include the addition of additional light sources to represent immutable ambient light sources in the factory, or screens that would have to be placed to remove these light sources from the finished system.

At this point, tuition and experience have enabled the VE to establish the fundamental elements of the TS. Beyond this, however, the VE begins a process of iterative improvement in the system both physically and in the software processing of the images acquired by the camera.

It might at times, be necessary to completely rethink the system, based upon the images received and the results of initial processing. As a result of this, the components used in the Prototyping System must be sufficiently broad in their abilities and tolerances that they are able to cope with the drastic changes that sometimes take place, or provide the required functionality to build the ideal processing chain for the imagery. Hopefully a suitably

successful combination of physical components and software operations will be discovered that form a solution that may be distilled into a form that can be copied by a Target System.

Another requirement of a PS that ought not to be underestimated is the need to provide a demonstrable solution to the issue at hand, in a way that is clear and convincing. In most cases, a Prototyping System will be called upon to provide demonstrations to decision-makers that rapidly evidence the feasibility of the project at an early stage, and later prove that the process does indeed work to a satisfactory level. Based on these demonstrations, the project will be continued and a Target System will be approved, respectively.

4.4 3M Business Model Problem

The 3M Corporation is one of the world's largest manufacturers of consumables, equipment and general products. Prior to recent changes in company direction, their product catalogue was so diverse that it included over 60,000 items at any given time, and employed 67,000 personnel worldwide in manufacturing and development.

In order to ensure the future of the company, an ambitious corporate policy was created:

50% of all sales must come from products created within the last five years

This goal is one of sales, marketing and development, hard to achieve, but enabling the company to constantly reinvent the market it addresses and provide new and diverse products.

The issue with such an ethos, of course, is that it places a great strain on the research and manufacturing abilities of the corporation and organisations that it develops partnerships with, such as universities and think tanks.

To ensure redevelopment of 50% of 60,000 products over 5 years, requires 17 new products created daily. Assuming that only 10% of products necessitate the use of a Machine Vision system to control their production to a sufficiently high quality, 11 new MV systems are required to be completed and installed each week. While this is a conservative rough calculation, variable based on limited days of work and inconsistent performance of differing products in the marketplace, it does serve to illustrate the need for a large number of skilled MV engineers to service such a rate of development. In reality, since top selling products tend

to continue to be sold over many years, the rate of development is actually far higher than this; although not publicised nor verified, an estimate would put the need for MV systems at around 5 new installations a day.

To fulfil this requirement for new Machine Vision systems, the efficiency and effectiveness of MV development must be improved to keep pace with a rapidly expanding market.

4.5 Not enough Vision Engineers

Assuming a development time for a complex MV system of 3 months with two engineers, 3M would need to employ 900 Vision Engineers (VEs). With a requirement for many tasks of a lead VE with a Masters-level or higher qualification and substantial experience in the field, the problem escalates, due to the lack of sufficient employees in the workforce. With such a situation duplicated across multiple multi-national corporations and only a limited number of educational institutions teaching Machine Vision, the problem becomes even more severe. Lastly, in the majority of cases, the VE needs to travel to the installation site to inspect the situation as part of the problem-solving process and feasibility discussion. This process consumes an excessive amount of time and involves national and/or international transportation.

Example: Let us suppose that a production line system in a factory is responsible for making bottles for wine. Their normal quality control involves a person standing beside the conveyor belt visually examining the bottles as they pass by. Unfortunately, it is discovered that objects appear in roughly 1/2000 of the bottles, and of those, around 80% are caught by the monitoring personnel. The foremost factor in the cases of the missed 20% of the objects, is the effect of operator error. In fairness, the members of staff are doing well at finding the glass fragments, since the recognition of a small distortion in the liquid after examining 1,999 correct bottles is quite a feat, thanks to tiredness, distraction and work inertia.

Hence, one in 10,000 bottles of wine produced potentially contains a shard of glass, posing a threat to the customer. Producing 5,000 bottles of wine a day the company is understandably worried that they might be endangering a customer's life every two days. As a result, the company engages the services of a Vision Engineer to provide an MV solution. In many

ways, this is the same process that occurs within development teams at large companies with active research divisions such as 3M.

The VE (a person with extensive previous experience inspecting errant objects in food jars) discusses the problem with the company over the telephone and realises rightly that the problem centres around the need for a good lighting method, since the distinction between the bottle, wine and foreign objects is minimal and refraction and reflectance will be a key obstacle in obtaining good imagery. As a result, a detailed understanding of the environment in which the system will operate is essential; ambient lighting, personnel movements and the flow of the production line through the checkpoint will all markedly alter the images for processing.

The engineer resolves to visit the factory and examine the situation first-hand to best assess the requirements of the project. Unfortunately, the VE is located in the United States (West Coast (GMT-8)) and the factory is on the west coast of Australia (GMT+10). The flight time between the two is a little over 12 hours, and hence two days of the visit must be taken up with travelling. Additionally, the time zone difference of 6 hours is sufficient that the engineer suffers from jetlag, reducing their effectiveness and needs a day after each journey to acclimatise again. For a 2 day visit to the factory and personal discussion with the staff, therefore, the VE must take 6 days of time. The same is duplicated when the system is being fine-tuned in development. As a result, two weeks of billable time is taken up with the Vision Engineer travelling to and from the factory, with only four days of that time being actually spent on site performing meaningful work. In addition to this burden are the costs of trans-continental flights, accommodation and food.

In summation, the investigative process requires a large amount of money and time to be spent for both the wine company and the VE. Once feasibility is established and a system may then be produced using this information and product samples supplied by the company. The VE is able to alter the laboratory environment to simulate the lighting within the factory confines and develop both hardware and software mechanisms to perform the quality control task. The system is then built and installed in the factory (once again, with required visits). The complete process takes many months from the start of investigation to deployment.

As is evident from this example, the development of industrial vision systems is time consuming and requires the significant attentions of a VE to build solutions which are highly customised to the task and the environment. In the cases of large companies such as 3M, with a wide range of regular new MV problems, the personnel requirement is substantial.

The solutions to this problem are manifold and must be used in combination in the sector as a whole:

- Increase the number of trained Vision Engineers and experts
- Reduce the development time of MV prototype systems, due to improved investigative tools and rapid development methods
- Reduce the obligations of on-site investigation visits while retaining effectiveness

4.6 Improved Tools

While the training of new Vision Engineers is to the most benefit of Machine Vision as an industry in the long term, the current rate of rapid expansion in the field requires that the existing workforce become more effective; able to solve more problems, concurrently and more quickly. To this end, the production of tools to design prototype systems is a key area of development, enabling the improvements in productivity that have been seen in the software world with the introduction of Rapid Application Development (RAD) Interactive Development Environments (IDEs), advanced debuggers, tracers and construction toolkits.

The specific intention is that the developer of the tools spends additional time sufficient that once complete, the users will be able to save a degree of that time in each system they build, subsequently. This process places additional load on the workforce for several initial years, but pays dividends from that point onwards, and continues to assist when more VEs become trained.

4.6.1 Lighting Advisor

The Lighting Advisor database is a product of the work of Professor B. G. Batchelor of Cardiff University and students under his tuition. It seeks to catalogue as completely as is reasonable, different forms of lighting arrangements and uses that are utilised in Machine Vision problems. With the assistance of the database, a Vision Engineer can examine their problem and quickly discover the most appropriate lighting method to build a solution.

Initially the database began as a set of documents in the Apple Macintosh HyperCard software, a hypertext-connected set of information 'postcards'. Since inception, the collection has grown to include close to 200 unique means of lighting objects and scenes, with associated images descriptions, directions for use, and suggestions of suitable problems to which the methods would be the most valuable. Work is ongoing to translate the information stored in the system into an SQL-based relational database for integration into networked Vision systems. The potential avenues for further development include:

- Providing advanced GUI and web-based interfaces to the Lighting Advisor data
- Including Lighting Advisor functions into interactive image processing systems
- Superimpose Prolog and functional language facilities to provide deductive assistance to users, suggesting solutions and reducing the range of possibilities as a result of input
- Including the data in neural network software to automatically adjust scenes and lighting as situations change, enabling the development of compensating Machine Vision systems in highly variable real world problems

4.6.2 Optical design Programs

In addition to concerns of lighting, equipment control and image processing techniques, there is a need within Machine Vision to consider the form of the lenses used on cameras themselves, and as components of larger multi-lens visualisation systems. While many laboratories may have stocks of limited numbers of lenses for use with the cameras, mountings and conditions that are immediately reproducible, final Target Systems for use in deployed industrial environments require that clarity of image come from optics, rather than later correction in software. Not only are software transformations of this type time-consuming when operating on each captured image, where a more appropriate lens would solve the problem, but transformations such as barrel correction introduce variations into the information density and veracity of images in transformed regions, and also may fail or introduce artefacts which are undesirable.

In order to discover the most suitable lens, a Vision Engineer must have a good grasp of the theory of optics involved in the creation of lenses, light paths around the system being created, and must form a mathematical model of the Target System. In most cases, this is a time-consuming process, and a highly specialised task for Vision Engineers, who most often

act as generalised systems design experts. In order to respond to this need, a number of Optical Design Programs (ODPs) exist; software able to model the characteristics of different lenses, consolidate this information against the reality of the physical problem and call upon a database of unique lens types, finding the most suitable setup and lens to fulfil the requirements of the system. Focal lengths, customised cross-sections and unique curvatures are taken into account, multi-lens systems are able to be modelled, and the VE is able to visually assemble the optical system most suited to his or her problem. One of the most effective benefits of such systems, is their ability to assess the tolerances of optical combinations and enable a Vision Engineer to understand which conditions are optimal for the model which they have resolved upon, and the constraints upon images and setups in order to ensure high quality imagery and system integrity.

In addition to informing the choice of lenses, a number of more advanced ODPs also include detailed facilities to aid in the creation of lighting solutions with simulation of light sources, reflectors and laser collimaters, It is also not unusual to find functional demonstration and template lighting and optical solutions provided by ODPs to enable VEs to immediately implement solutions to basic problems.

Two of the current leaders in the field of ODPs are Zemax by the Zemax Development Corporation [ZEM] and OSLO by Sinclair Optics [SIN].

4.6.3 LabVIEW

LabVIEW is a collection of tools designed by National Instuments to both control and manage data flow between devices, especially in research and manufacturing contexts. Beginning as a tool for creating control GUIs for local devices and for local automated control as reported by Johnson and Jennings [LAB], it has grown to include data reporting to databases and web sites. Most recently, it has developed the ability to control networked devices, albeit through custom driver modules, which must be incorporated for each individual device.

LabVIEW operates as an environment for connecting together and processing data from what it terms 'Virtual Instruments' (VI). Each VI is a custom-written driver for a piece of hardware or software, which mimics the interface and/or capabilities of the device. Johnson and Jennings describe VIs as:

"The objective in virtual instrumentation is to use a general-purpose computer to mimic real instruments with their dedicated controls and displays, but with the added versatility that comes with software."

The primary development environment for LabVIEW is the RAD (Rapid Application Development) user interface designer, where VIs are connected together with each other and additional operations added to process their data, building a complete program visually. Underlying the visual programming environment is a programming language named 'G', which can be accessed manually to add non-visual logic. Using the RAD designer of LabVIEW in comparison to C for device control has enabled significant increases in the field of rapid prototyping of device control systems, up to 5 times the relative productivity [WELL2].

More recently, LabVIEW has added compilation and bindings for development subsequent to the creation of the UI and basic device connections, using Microsoft Corporation's Visual Basic or Visual C++ to add required custom logic.

As a result of this development process, LabVIEW is ideal for the user interface development for fixed equipment and integration with existing custom business software, written for Windows. One of the key advantages, is the opportunity to reuse skills required for the latter, producing rapid development processes for experienced developers and the creation of relatively light-weight interfaces.

LabVIEW presently operates on Windows, MacOS, Solaris, HP-UX and under virtual machines in Linux.

4.6.4 Interactive Image Processing

An interactive Image Processor is a computing application whose purpose is to process images, but in a semi-automated manner. This is to say, that it provides tools for running sequences of Image Processing operations, and also the opportunity to issue commands for individual operations, examining the response to the image, proceeding with further actions, or regressing to an earlier stage in order to attempt a different means of processing. The

intention is to provide a user-driven tool to gradually construct a processing sequence to extract the required data from the input image.

This is a highly investigative task which is typically performed in a laboratory by a qualified and experienced Vision Engineer or Computer Vision expert, who can quickly identify reasonable chains of operations.

The primary operating distinction between an interactive processing application and an automated image processor, is that the latter is completely scripted; an initial image and a script are supplied, and the results can only be displayed on completion. At this point, a failure in the processing, or a bad choice of operation for a given image will present cumulative results. Clearly this leads to each iteration of the processing chain consuming a full script running time (see fig.20) and makes the resolution of bugs and general failures harder to perform.

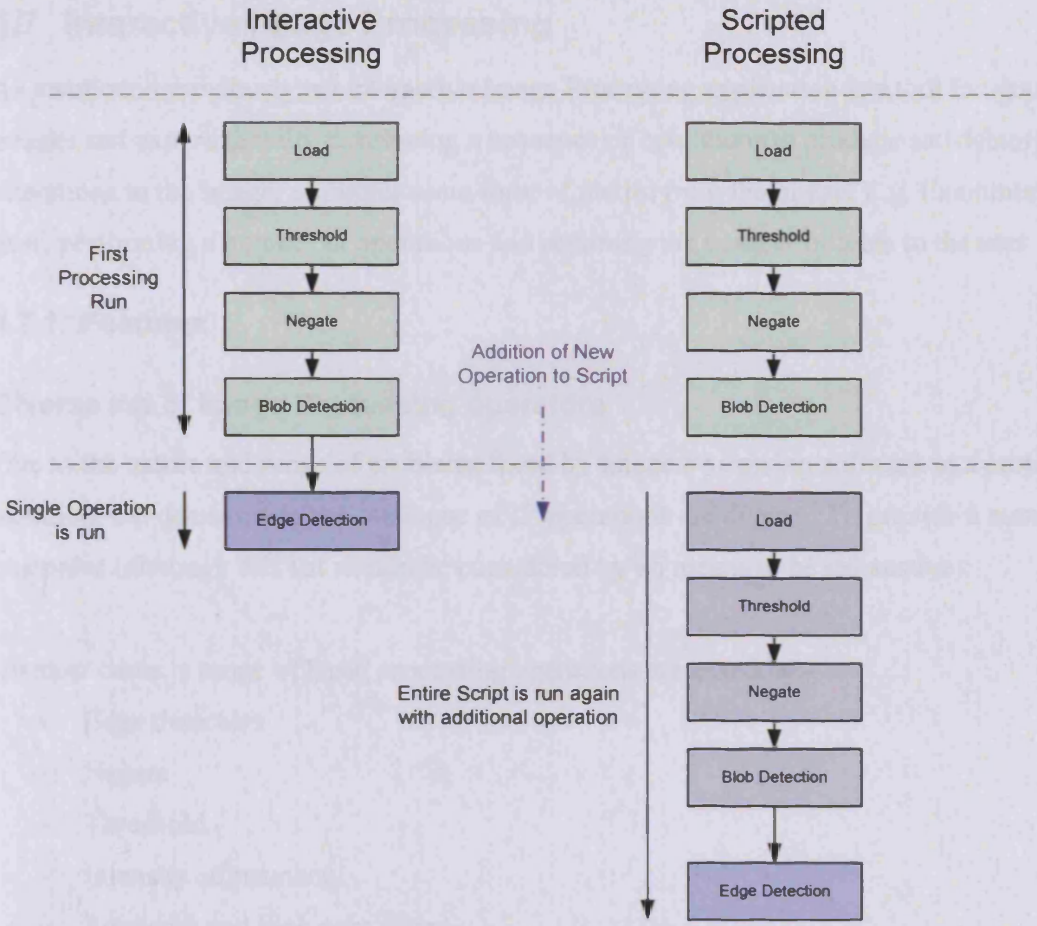


Figure 20 : Interactive and Scripted processing methods and their running times during the task of testing and adding a single operation to an image processing chain

According to Batchelor in [BAT], the use of visual assessment and theoretical, non-experimental means to develop a processing chain has increasingly been recognised as being incapable of solving industrial problems and complex Vision problems in general. As a result, interactive image processing has become an essential feature of Machine Vision project developments, even when non-interactive systems are used iteratively to provide slow interactive investigation. Thus, as experimental investigation is an established requirement of Vision solution creation, comprehensive, easy to use interactive IP environments for solution prototyping become an increasingly important part of improving project time and effort profiles.

4.7 Interactive Image Processing

As mentioned previously, an interactive Image Processing application is a tool for examining images and experimentally developing a sequence of operations to produce satisfactory alterations to the image, or output some form of metric from the image. E.g. Examining a gear, performing a number of operations and returning the number of teeth to the user.

4.7.1 Features

Diverse set of Image Processing operators

Due to the nature and range of problems faced by Image Processing software in a prototyping situation, the demands on the catalogue of IP operations are diverse. To provide a number of examples (although this list should be considered by no means to be exhaustive):

In most cases, a range of basic processing operations are essential:-

- Edge detectors
- Negate
- Threshold
- Intensity adjustment
- Low pass and high pass filters
- Etc.

In addition to these, a set of more advanced processors form the core of most identification tasks:

- Blob functions
- Convex hull
- Erosion & dilation
- Fast Fourier Transform
- Hough Transform

Sets of additional drawing and overlay tools are also typical:

- Centroid drawing
- Marking of line of minimal moment
- Skeleton illustration
- Minimum area rectangle overlay

- Primitive shape drawing
- Text addition

Finally, analysis tools are added:

- Blob counting
- Pixel counting (at given intensity)
- Euler number
- Centroid location and angle of line of minimal moment

Classically, such tools are used on black/white or greyscale images. With the advent of cheap colour cameras and greater performance of image processing software solutions and hardware, colour processing is also performed, albeit as processing of the independent RGB channels.

Command-line operation / visual construction

Interactive image processing systems require an active means of user interaction beyond being able to enter a script and execute it, or point the application at a script file to run. As a result, three types of interfaces have developed:

- Command line operation
- GUI-widget operation (clicking named GUI buttons)
- Visual program design

Command line interaction is perhaps the simplest and most obvious of the three modes of operation. In principle, an IP application has a repertoire of operations that may be performed upon an image. Using a command line or a GUI text box, command names may be entered and those are then used to initiate the named operation on the image. The processed image is then optionally displayed and the application returns to a waiting state until the user enters another command name to perform (see fig.21).

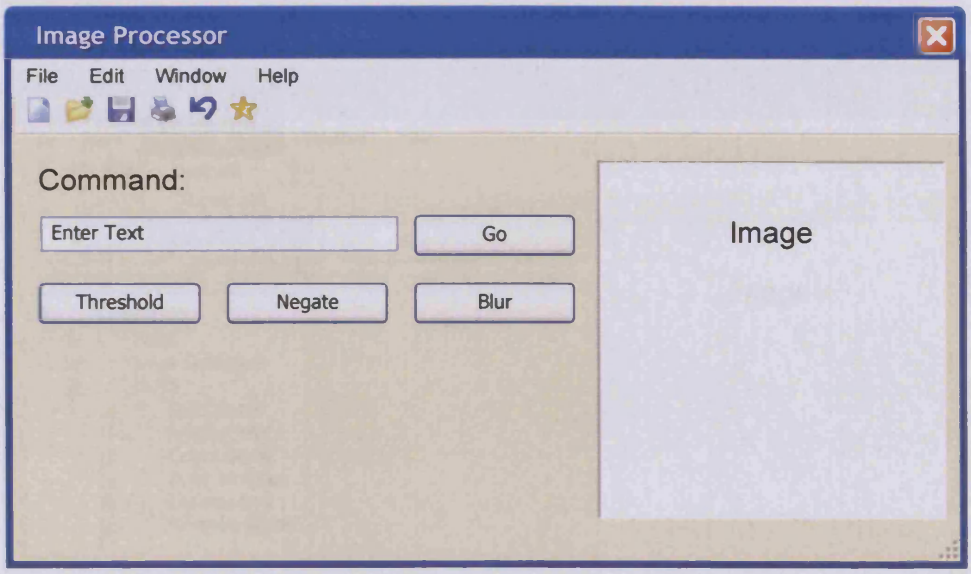


Figure 21 : A simple image processing application, using a command line or single-operation buttons for operation request

GUI operation is a product of more modern programming practices and usability considerations. Essentially, while command-line interaction can be effective, it presupposes that the user has a list of commands for the image processor and knows how to enter them in the appropriate manner to a blank input space without any assistance. Clearly, this may not be the case for all users, and as such, GUIs with menus of possible commands or sets of buttons for each operation are provided. The user need only locate the button with the label that they require and click it to initiate the operation.

The limitations of such a system are inherent in all systems with a large range of options with equal desirability and probability of use; how best to arrange the GUI widgets to enable the user to efficiently find the widget they require. Grouping of similar operations, menus, tabs, drop-down lists and treeview widgets are common to many applications in this field (see fig.22), although discussions persist between users as to which is more useful and manageable, as it does presuppose that the user groups and orders commands in the same manner as the designer of the GUI. More advanced users frequently find command-line operation and a simple printed list of commands to be more effective than a GUI, while new users may prefer the presence of widgets and definitive and limited options on their input to the system [PRE].

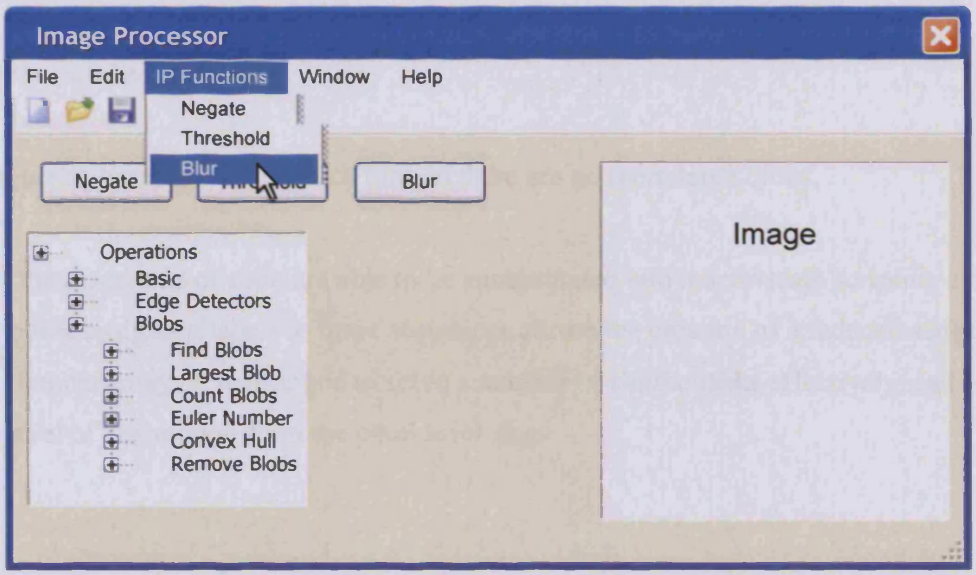


Figure 22 : A simple IP application, offering menus and operation trees as alternative execution mechanisms

Macro-scripting abilities

While image processing sequences may benefit from the single-command execution interfaces of interactive applications, many tasks involve the ability to recite a number of operations in sequence as components of a larger command sequence. In most cases, these small sequences are consistently used and need not be altered for each use. A suitable example of this behaviour is the task of examining a set of simple light objects on a dark background, with the intention to isolate them and count the number of pixels used in their representation.

In this case, the commands are as follows:

Open Image File
Threshold
Negate
Find blobs
Remove blobs smaller than X pixels in area

Then a sequence follows:
Find the largest blob

Isolate it
Count the number of pixels used
Output the number of pixels

This sequence is repeated until such time as there are no more large blobs.

Both of these sections of code are able to be encapsulated into macro-scale scripting elements. At this point, assigning labels to these sequences allows the creation of a reduced script, whose elements may be rearranged to solve a number of similar tasks effectively, and with a higher level of abstraction from the pixel level. E.g.

Setup
While (more large blobs exist in image)
    Find next largest blob
    Find the number of pixels in blob
    Repeat

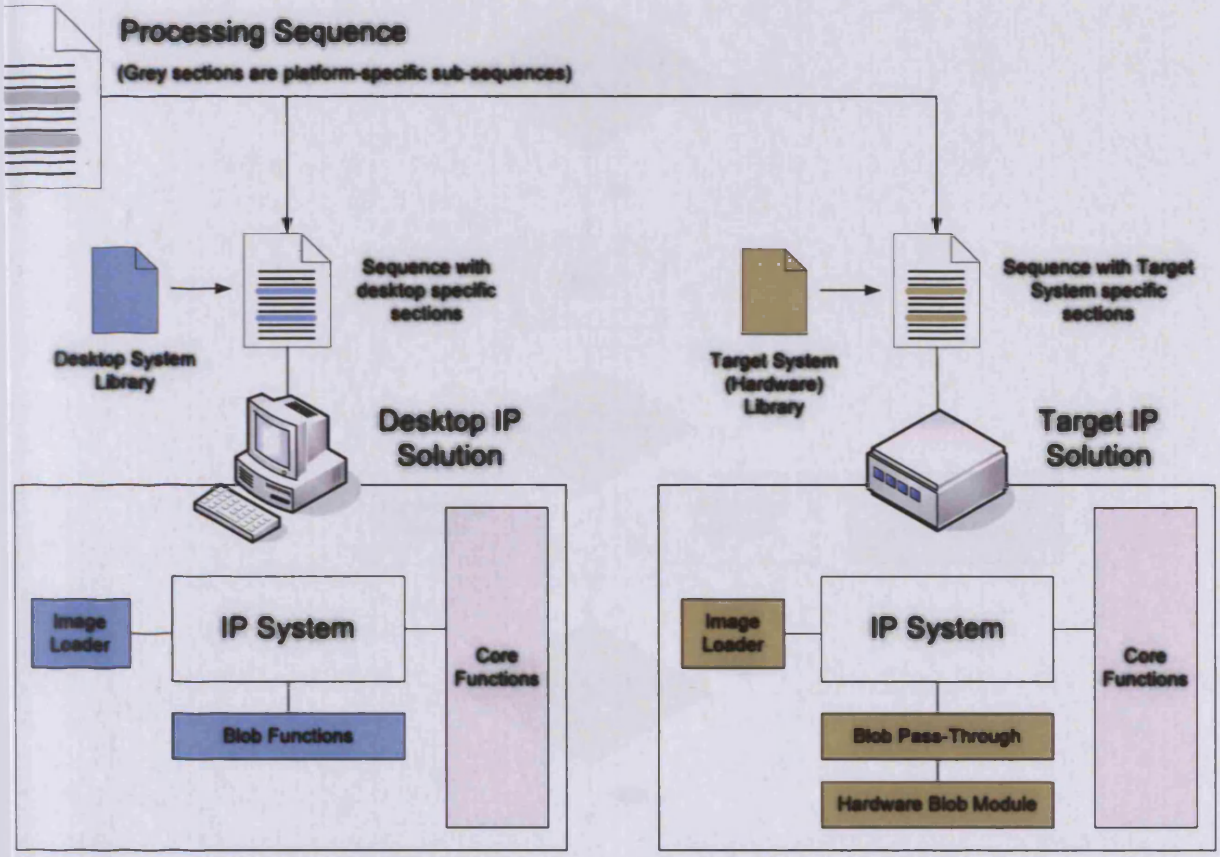
This ability to develop macros and scripts for inclusion into interactive processing is highly beneficial to the user of such a system, as it reduces the potential for typographical errors when replicating the elements of the macro repeatedly within the same overall task. This process makes development of new operations faster and less error-prone. In addition, however, it enables the user to deal with their problem and solution with an added degree of abstraction. To develop the system mentioned above, for example, most developers would find it more expedient to utilise the three-line construction mentioned than the ten line version that the use of basic image-level commands would necessitate.

As a result, the benefits of a macro-enabled system may be outlined as:

- Increased abstraction
- Enhanced speed of development
- Reduced potential for error
- Increased clarity of final command sequences (provided macros are clearly labelled)
- Ease of repetition of command sequences
- Ability to develop large scripts based around building block macros

Interface construction

Beyond these established improvements to workflow, the use of macros and scripts to provide abstraction produces a high-level user-defined programming interface that is isolated from the pixel-level manipulation. This enables portability (see fig.23) of essential task logic and the ability to replace operations within a command sequence based on the platform upon which the logic is running.



gure 23 : MV systems using libraries to provide target-specific command implementations, enabling portability solutions to multiple architectures

Long history / 2 images + saving

Interactive Image Processing tasks are exploratory by nature, and as such, steps taken and operations executed are not always correct nor ideal. In such situations, there is a need for systems to provide an ability to return to a previous state; to undo a change or sequence of changes made to an image (see fig.24).

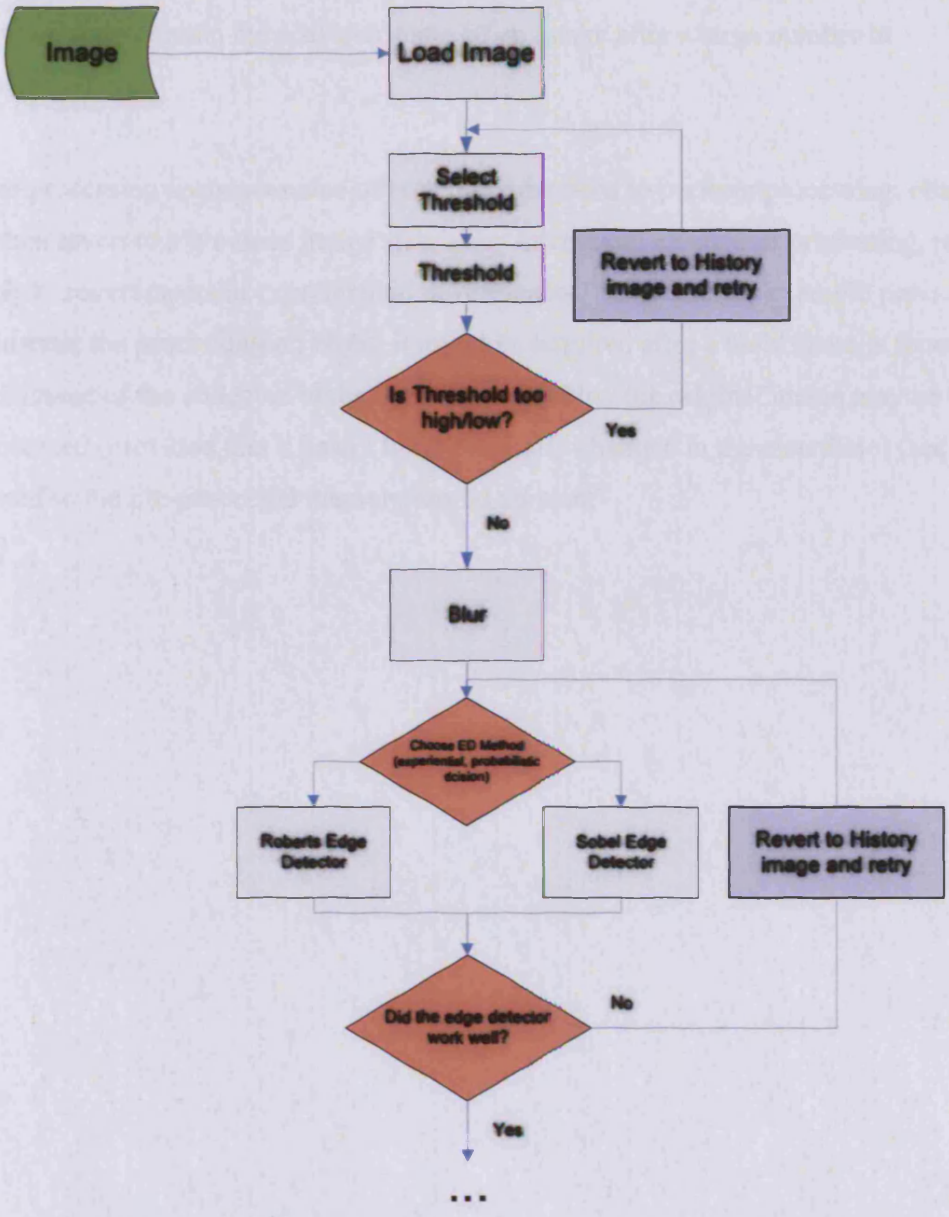


Figure 24 : Use of processing history to provide rollback functionality with interactive processing of images for Machine Vision tasks

This type of feature is generally produced in two forms:

- Lengthy history of each step
- Limited number of retained steps and user-initiated saving of states

While it might seem intuitive and advantageous to maintain a full per-operation reversible history for an interactive image processing application, this is typically not possible with long image sequences due to the limitations of storage retention of all intermediary images. With the necessity of protecting the integrity of images during processing, in-memory compression

techniques can be somewhat limited in effectiveness and workstation or desktop systems are often unable to contain the previous states of an image after a large number of transformations.

Image processing sequences also often require the need to perform processing, obtain results and then revert to a previous image state as an intentional element of processing, rather than simply to revert errors in experimental development. To extend the example previously used to illustrate the processing on blobs, it might be required after a blob's area is found, to revert to the image of the collected blobs. At this point, either the original image may be loaded and reprocessed (provided that it hasn't been externally changed in the meantime) (see fig.25) or reverted to the pre-processed memory-stored version.

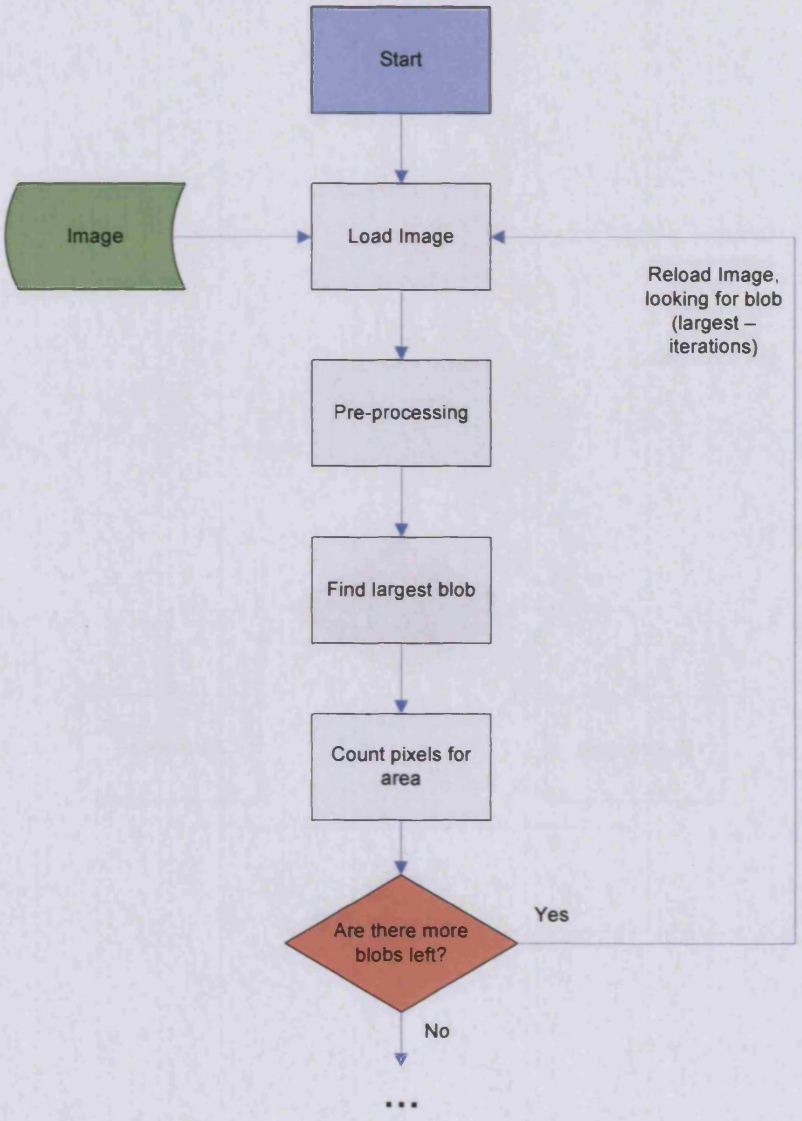


Figure 25 : Reloading and reprocessing rather than utilising a history system for discovering blobs in a scene

It should be clear, however, that this process is identical to the Vision Engineer testing a number of IP sequences, realising that a sequence is not working, and reverting to an earlier stage of processing (see fig.26).

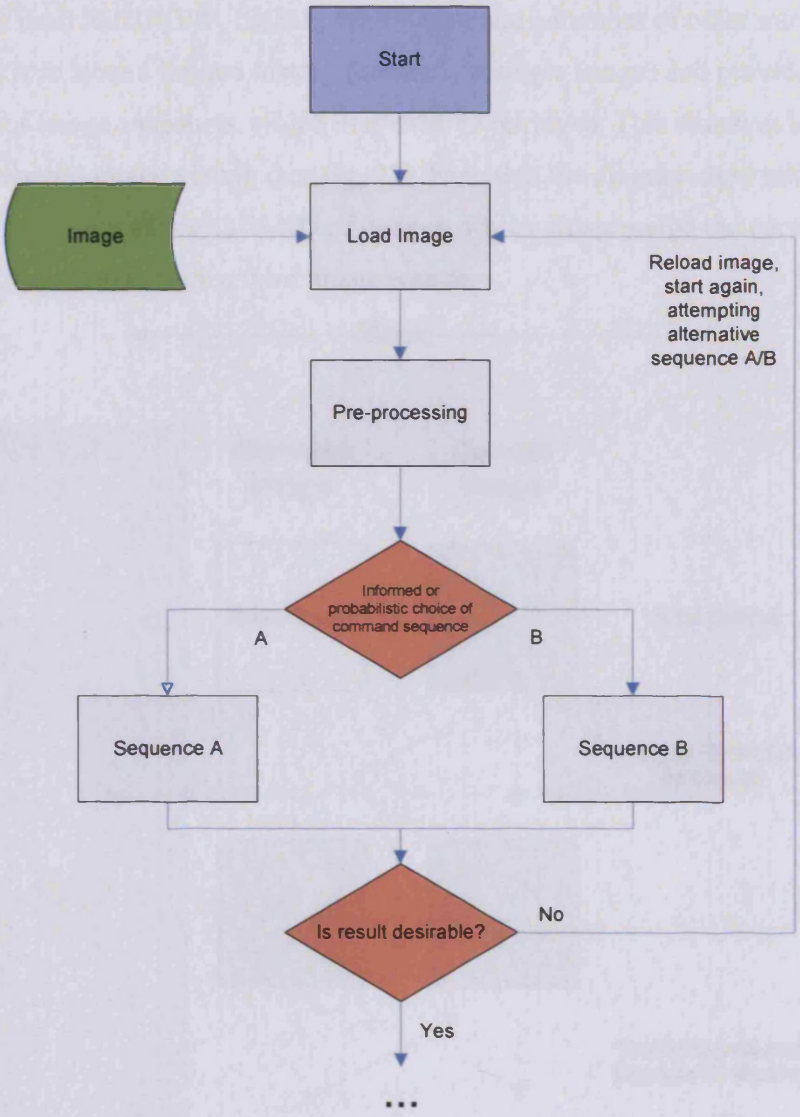


Figure 26 : Reloading an image from input in order to undo a change introduced through incorrect choice of processing direction in a forking situation

Since the history functionality of the interactive application itself cannot be readily separated from the functionality required in constructing solutions, constraints are imposed upon the history functionality of interactive applications, since they must match and yet not exceed those of the potential Target System hardware. Finite memory limitations are one of the most pressing of these constraints. As a consequence, an unlimited or pseudo-unlimited history system is not normally practical to implement in a Prototyping System, as it creates an unnecessary divergence against the Target System.

The response used in CIP, PIP, SUSIE, NeatVision and a number of other interactive applications, is to store a limited history (normally a single image) and provide commands for user storage of image snapshots, which may then be retrieved. This situation is expressed as a 'Current/Alternate' image system (see fig. 27). Note that the diagram does not describe dyadic operators, such as logical AND of images, which either merge the current and alternate image, or introduce data from a third image source.

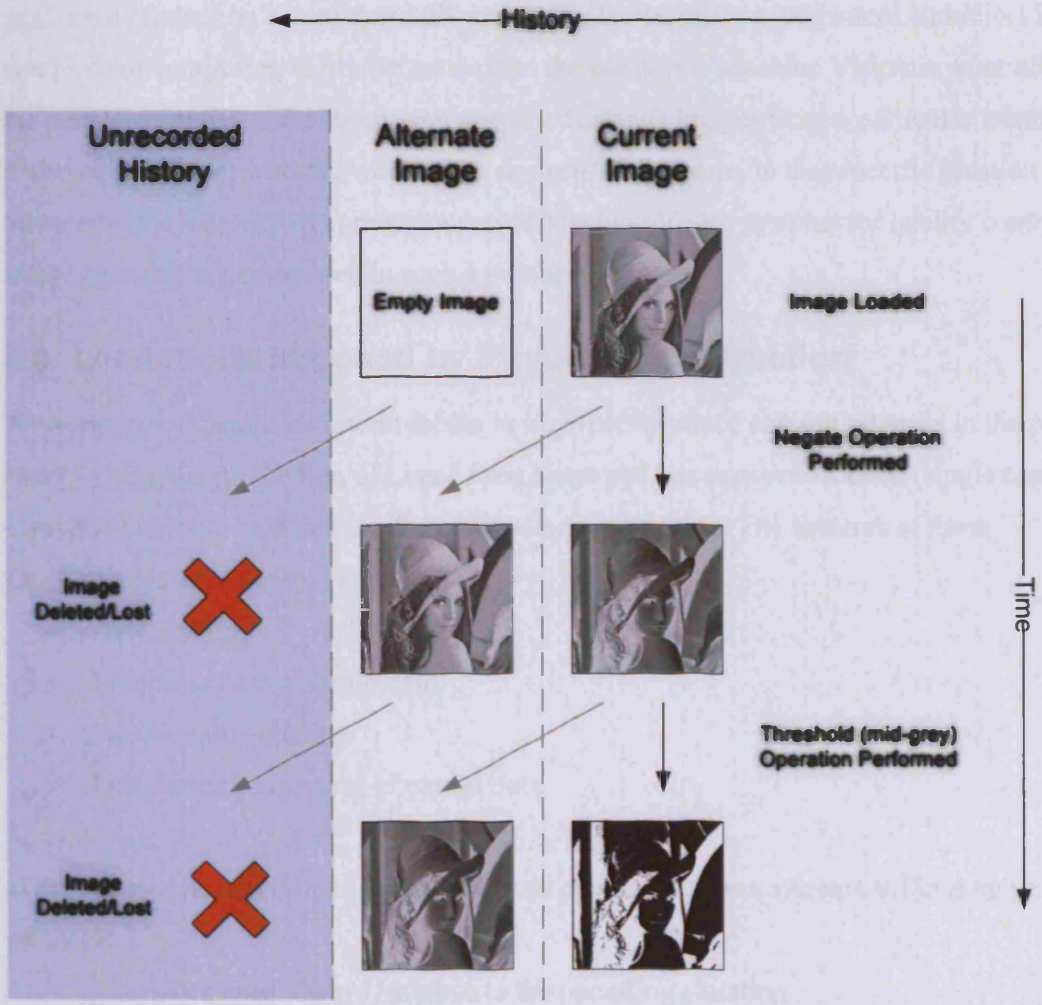


Figure 27 : A 2-image Current/Alternate history image processing in use through a processing sequence, with the current image transferred to the alternate, and the alternate destroyed at the completion of each operation

4.7.2 Physical Integrity

One of the most significant properties of prior interactive Image Processing systems, is that the entire system from camera/framegrabber, image processor, equipment controller, intelligent controller to user interface have been resident in a single location and normally

within the same device. Moreover, many systems are constructed in such a way as to integrate the components mentioned into a single piece of software.

The design of such systems originates in the era of single computing resources, mainframes and single-tasking environments. In many ways, this design also mirrors that of implementation of target systems into hardware and deployment. It is quite reasonable to examine a product on a conveyer belt, search for faults, make a judgement and reject it within one piece of computing hardware sat next to the conveyer. Machine Vision is, after all, one of the most physically-based computing tasks; it demands images from a particular location, and it also must interact with the world once analysis is complete, in that specific location. For this reason the majority of current industrial Machine Vision systems for quality control and safety purposes are composed in such a manner.

4.8 Limitations Imposed by Physical Construction

Networking has been a persistent theme in high-performance computing tasks in the past twenty years; the production of Local Area Networks has exposed isolated (single computer) computing tasks to examination for areas of new expansion. The features of these developments have centred around:

- Data sharing
- Remote access of equipment
- Functionality sharing
- Distributed processing of partial data

Without network capabilities, non-networked combined vision systems suffer a range of problems:

- Operators must always be close to the operating situation
    - Exposure to X-rays, noise, dust, chemicals, strobe lighting
    - Operators affecting the lighting in the scene
    - Operators must be available whenever the system is running; it cannot run alone if there are any cases under which it might require intervention or tuning
    - Routine system alterations (switching tolerances when ambient lights are turned off at night) must be performed manually
- If an element of the system fails, the system as a whole fails until it can be replaced

- Maintenance must be performed on-site, leading to engineers and technical support staff travelling, especially for solutions deployed on multiple sites
- Any proprietary elements of the system must be encapsulated at each site and are only protected by local security facilities. They cannot be provided from one highly-secure centralised location.
- Advanced methods and tools must be provided at the local site and duplicated to all systems, rather than a single instance
- All updates must be physically introduced at the system's location (introducing the same issues of operator safety/interference)
- Systems cannot take advantage of data from other sites to perform more informed processing and organisation-level coordination of tasks
- Data from MV systems cannot easily be integrated into other business tasks
- Sensors located across a wide areas, such as cameras are difficult to integrate without expensive cabling and are prone to interference

In most of the other considerations, the primary issue is inconvenience. If an operator can have access to a system, so long as the operation is not highly time-sensitive, data input, output and system modification can be reasonably performed. The problem of exposing operators to adverse conditions is, however, a significant consideration in the development of Machine Vision systems when using non-visual or strobe illumination. [BAT2]

In cases where the operator may not safely approach the equipment, some means of remote interaction becomes essential, as the remaining issues of isolated devices as mentioned above become insurmountable without a person to act as a bridge to other systems and processes.

With the development of Machine Vision systems in industrial environments as isolated mechanisms, questions must be raised as to whether they can be made more effective or open up new areas of endeavour using networking tools.
Businesses might benefit from (following the themes above):

- Wider access to data from MV systems beyond the attached terminal monitor
- Being able to manipulate MV systems from a distance
- Sharing the facilities (e.g. cameras, lighting, processing) that MV provide with other business tasks

These are output tasks, where the MV tools contribute to wider business processes by supplying data and functionality. There are also input tasks, which seek to improve the abilities of the MV system to perform a task through the inclusion of wider data and interaction:

- Enabling MV systems to perform more educated functions through the inclusion of history and other data from various sources
- MV systems able to wait on remote control in ambiguous situations, or to liaise with other MV systems to perform a combined task
- The ability to include remote processing tools, such as databases, intelligent processors and black-box network data services in processing

4.9 Networked Systems

Implementing Machine Vision systems using network technologies requires the isolation of the components that form a Vision system, and connecting them to a network in their own right.

Examples of such components are:

- Cameras
- Image Processors
- User Interfaces
- Equipment controllers
- Databases
- Intelligent controllers/interpreters

Once this process is complete, programs and scripts can call upon these applications to build networked systems (see fig.28) as and when needed, or they can interact with each other autonomously across the network (producing non-hierarchical topologies without central controllers, referred to as 'meshes').

4.9.1 Block Diagram

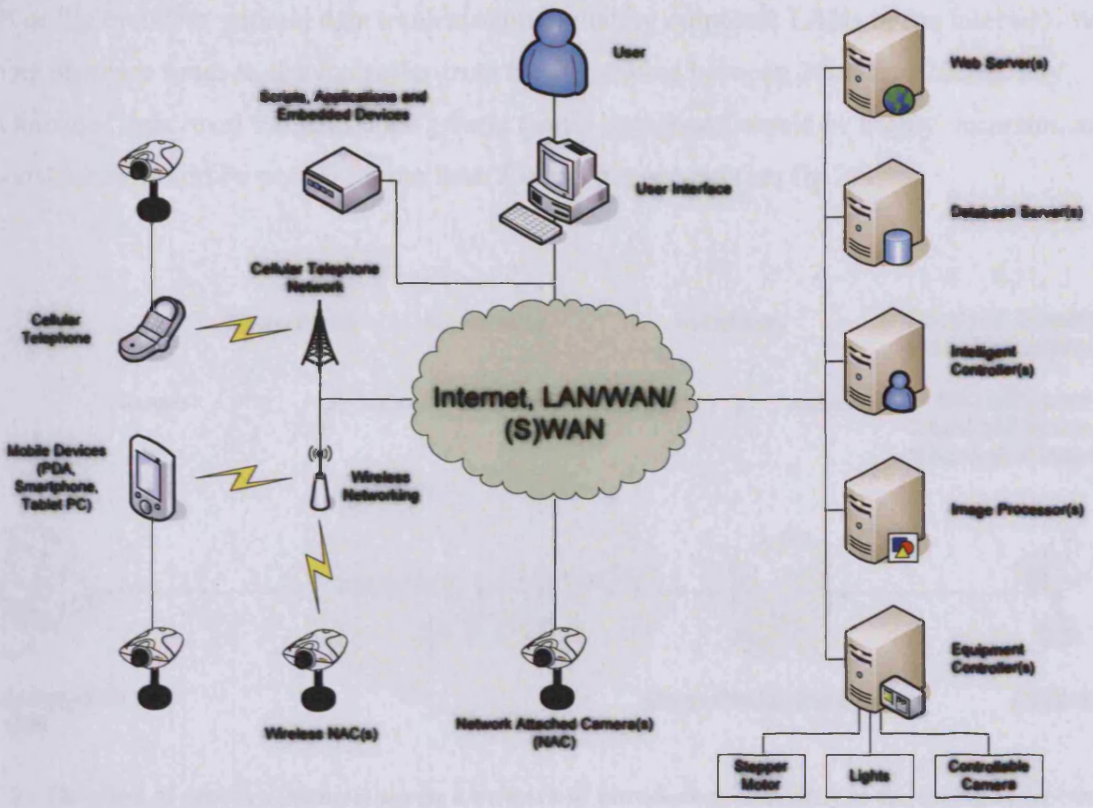


Figure 28 : Generalised Myriad framework, with the inclusion of all possible elements, linked through network connectivity

4.9.2 Modus Operandi

In the case of Myriad, the framework and implemented systems produced for this thesis, Machine Vision components are designed as web servers, with an incorporated scripting language (named Myriad Component Script (MCS)) and enhanced capabilities, such as Image Processing commands.

4.9.3 Network Traffic

The principal limitation of a networked Machine Vision system is the heightened dependence on network bandwidth and particularly, latency, in performing time-sensitive tasks, such as responding to a change in light levels in a high-speed production system. Using a lightly coupled networked application couple, a measured light level would be sent to the controller every division of a second that was required.

However, sending instructions in this manner poses timing issues when network latencies are included, and more so when they are variable. Suppose that the couple exist on two networks

connected via a bridge and one network is highly noisy (inconsistent high-bandwidth traffic from file transfers, general data transmission, imitating corporate LANs or the Internet). With ping response times to the controller from the IPE taking between 20ms and 200ms, any number of light level transmissions greater than 5 per second would be highly uncertain, and consistency would be poor for more than 2 levels per second (see fig.29).

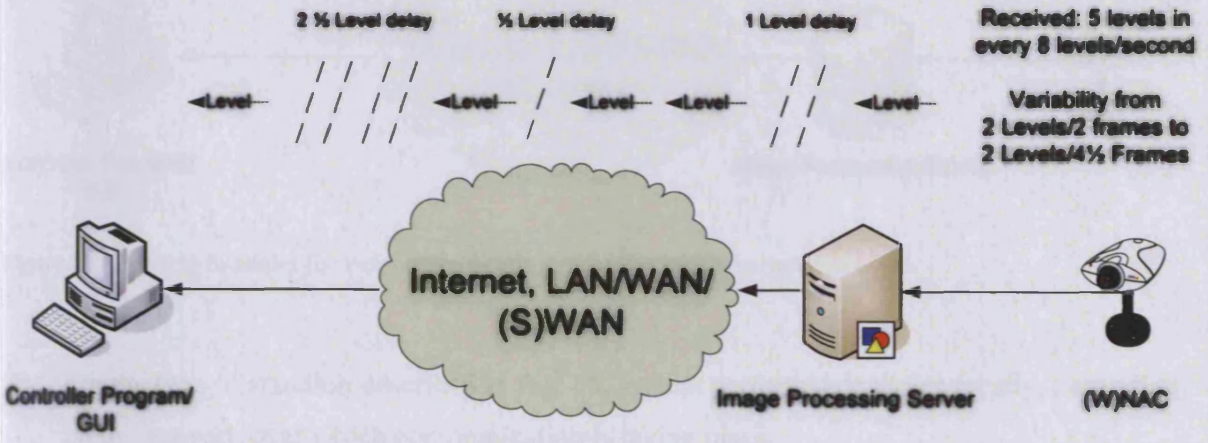


Figure 29 : The effect of network latencies across a network in introducing variability to the timing of repetitive light level transmission

This situation is exacerbated by data processing to assess the requirement for altering the light output of the hardware to compensate for changes in ambient levels. This amplifies latency problems by effectively doubling the time taken by the network issues, due to the time taken for the return trip of the instruction to alter output (see fig.30).

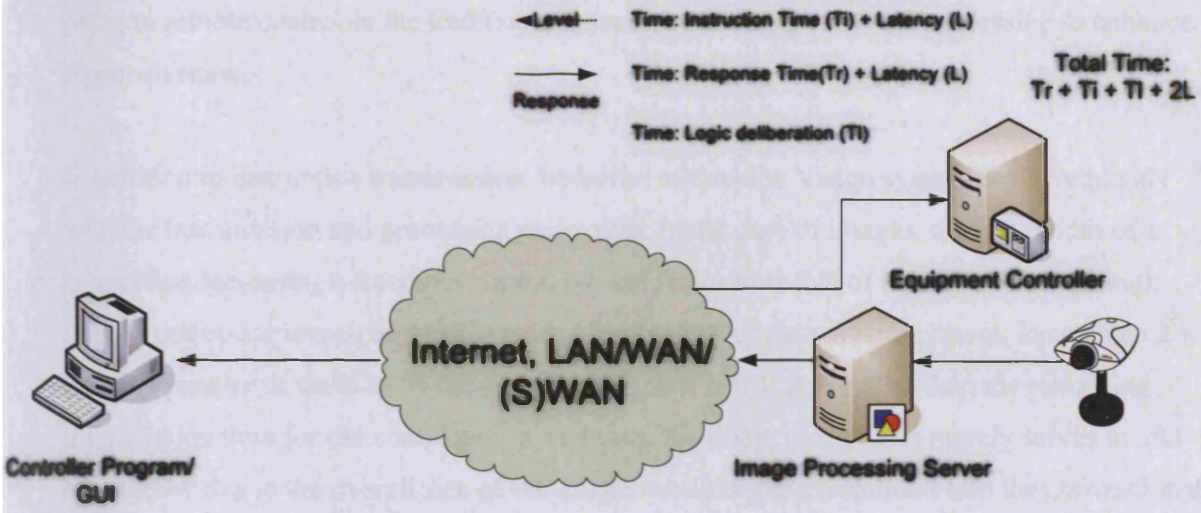


Figure 30 : Per-trip latencies for responsive lighting control across a variable network

For the type of situation described in Fig. 36, system performance varies greatly, depending on the network over which communication is taking place.

Low latency, high bandwidth networks as are typical with intra-national communication across leased lines or cable/DSL tend Ti, L and Tr to 0 and total time becomes synonymous with the time taken for the remote controller to decide which operation to undertake. However, this is the bast case for such a system.

Low bandwidth, high latency systems are the worst case and incorporate wireless technologies, cellular telephone connections and dial-up modem connections across a traditional landed telephone line. In these cases, Ti and Tr become large, as does L, exacerbated by the two trips required in the problem to send the level data and send a response command. It should be noted that some situations such as GSM data and satellite data transmission feature massively high latency values that become the dominant factor in system performance, but while GSM bandwidth is additionally poor, satellite data, once begun, provides high bandwidth.

The key benefits to the use of locally-scriptable networked vision systems in the context of this example are that the effects of latency and bandwidth are restricted to the local machine or local network and escape the variability of large-scale wide-area networks, such as the

Internet. Utilising Myriad components in this scenario allows the system operator to either perform remote control in the traditional manner, or to engage in local processing to enhance responsiveness.

In addition to instruction transmission, however, a Machine Vision system deals frequently with the transmission and processing on images. In the case of images, the bandwidth of a connection increasingly becomes a more critical factor than that of latency. Although high latency does pose issues, especially when a bad packet of data must be resent, incurring a 2 x Latency penalty, it tends to be the case that this sum is still far smaller than the remaining transmission time for the complete job, and thus, the duplicated packet merely serves to add one packet size to the overall size of the image, while being streamlined into the conventional data stream where an opportunity presents itself.

In transferring images between servers, or servers and clients, there are a number of factors that may significantly affect data transmission times:

- Sender bandwidth
- Receiver bandwidth
- Image size & compression

In most cases, users of systems will tailor data size to the type of network connection and bandwidth availability that is involved in the problem. For example, a camera attached to a 64kbps (or less) dial-up/ISDN connection will typically be configured to minimise image file size at the expense of image quality, or retain high quality through reduced compression and accepted increased data transmission times. In most cases, however, the type of network access deployed in a solution will be chosen within a set of constraints imposed by the problem, such as:

- Required image quality
- Acceptable data transmission times
- Locational freedom
- Available forms of network connectivity
- Running costs
- Per-byte data costs

In this chapter, machine vision systems have been split into two categories: target systems (solutions to problems, complete and ready to be deployed) and prototyping systems (developmental and exploratory systems, used to explore problems before the creation of target systems). Examples were provided of how prototyping systems were used, and the features that are advantageous to tools of this type.

The 3M problem was then introduced: companies with high research and development requirements, attempting to overcome product churn frequently need machine vision systems to assist in manufacture or development of new products. The limited number of skilled vision engineers worldwide constrains development, and those engineers need to be made more effective through new tools and methods. These tools include: Intelligent software advisors, optical design programs, visual development tools and interactive image processors. Since interactive image processors are of special interest, time was taken to outline salient features, such as diverse instruction sets, multiple interface methods, macros and scripting and reversible operation histories.

Examining the further broadening of prototyping systems, the issue of restricted location was introduced, highlighting how the logistics of transporting a limited number of vision engineers to different sites further reduces efficiency. Hence, the desirability of systems able to allow a vision engineer to assess problems across large distances was discussed. If a vision engineer does not have to spend time travelling, they may spend more time working to solve a problem. An outline of a networked vision system and a description of the manner of its operation were provided.

5 Myriad Concept

This chapter describes Myriad as an implementation of a framework for networked and distributed vision systems. The chapter begins with an outline of the broad concepts of a network-connected group of HTTP-based server components working together dynamically to solve problems, regardless of their location in relation to each other. The defining features of Myriad are discussed, along with the implications of this functionality and how it enables Myriad systems to operate.

After considering the principal details of Myriad, the next section compares it to the limitations discovered in previously-mentioned vision systems, and the degree to which it resolves these issues.

Finally, the limitations of Myriad as an implemented framework are discussed, most specifically with the issues of communication across unstable, insecure networks and the latencies encountered when transmitting data across the large distances involved with global networked solutions for vision problems.

Myriad is a framework for distributed and networked Machine Vision (MV) systems, defining the structure of components connected by a network and the manner in which they interact.

Myriad converts components of MV systems, which would typically reside on the same host computer, or on equipment interconnected by customised protocols and hardware and disperses them around a TCP/IP-based network. Once this is achieved and protocols for interaction are defined, MV systems may be built using any resource on a network. Using the Internet and Wide Area Networks (not to mention wireless LAN connectivity), resources may be anywhere around the world, even if just in another room or building owned by the same organisation.

Networking of MV components also enables implicit integration of existing Internet-based systems, from web servers to network-attached cameras, high level scripting languages and databases.

Finally, provided a reasonable protocol for relating with Myriad networked components is created and adhered to, other networked systems are given the opportunity to incorporate Machine Vision operations and methods into their tasks with minimal effort. This process expands the potential applications for MV technologies and exposes new avenues of development and research.

5.1 Myriad Outline

The principal design of Myriad systems develops from a simple question:

What happens when different sections of a Machine Vision system are networked together, possibly over large distances?

Myriad, separates traditional components of MV systems, such as cameras, image processors, equipment controllers and interfaces and enables them to communicate over a network, through standardised interfaces, and augmented with scripting capabilities.

5.1.1 Component Classes

Elements in Myriad systems have key characteristics:
-Whether they can be controlled
-Whether they can control other systems
-Whether they supply data

In a system designed to solve a problem, all components effectively reduce to four classes:

Class	Abilities	Example
Controller	Controls others only	GUIs, Scripts, Applications
Server	Controls others and is controllable	Image Processing servers, Equipment Controllers, Natural Language interpreters

Data Source	Supplies data and may be controlled as response to a request for data	Networked cameras, databases
Operator	May only be controlled. Supplies no data.	Output devices, equipment controlling hardware (which returns no confirmation data/interaction)

In the majority of situations, the *Operator* class will not be encountered directly, as most recent hardware and software systems designed to perform a task are able to return data, even if only for confirmation purposes. It is typical that these final result systems are isolated from the remainder of the Myriad system through the use of a Server component as a gateway. A good example of this would be a piece of custom-built equipment controlling hardware, with a specialised interaction language (Myriad Component Script, *see section 7.4*). In this situation, an equipment control *Server* component is developed which serves to manage and translate instructions to the hardware (see fig.31), while providing a universal interface to the network as a whole.

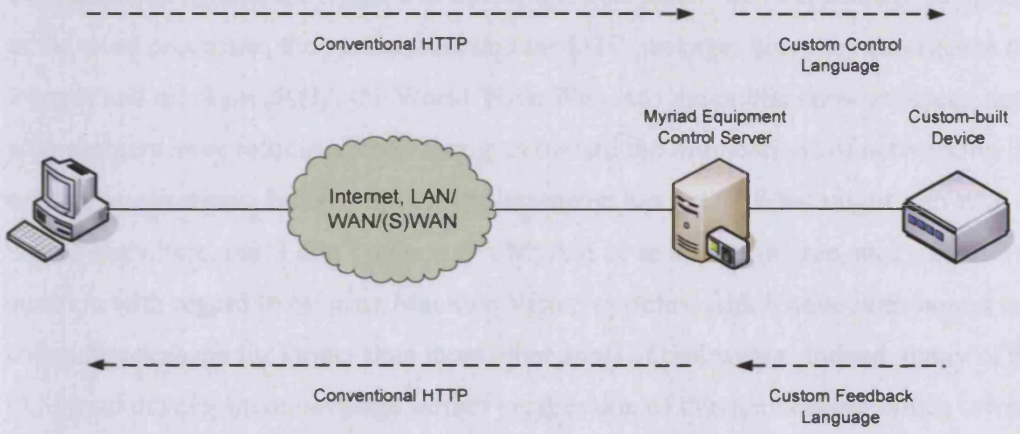


Figure 31 : Using Myriad EC servers to provide higher-level command language interfaces to physical devices

The *Server* components form the heart of any Myriad network, and in augmenting these systems with scripting abilities, basic reasoning and the ability to communicate and control each other, networks of components are able to interact in complex manners, without a

Controller managing every step. Moreover, it also enables decision-making to be managed in a site-local way even in a network-based system, providing high speed response and network stability (see *section 7.4.1*).

All of these components are connected to a shared network, and are capable of interacting using an HTTP-based protocol. Should a new hardware or software system not be capable of doing so, a wrapper Server may be used to integrate such elements into the framework.

5.1.2 The Importance of Control for Networked Systems

Over the past decade, networked computing systems have become a clear area of growth within business environments, from the multi-national corporations, who began networking in the 1970s and 80s with terminal-based applications, developing to intelligent networked systems, with data sharing, CRM (Customer Relationship Management) tools and collaborative working to SMEs (Small/Medium Enterprises) using internet connectivity, document sharing between desktop systems, groupware services and network printing.

Examining most of the innovations in computing technologies since the emergence at the end of the 1970s of personal computing, application development was, for the first 10-15 years, the frontier; programs for single users on single computers drove consumer computing, such as the word processor, the spreadsheet and the DTP package. Since the emergence of the Internet and more precisely, the World Wide Web into the public consciousness, application programmers have refocused their thoughts toward the implications of networking those existing applications. Indeed, the overriding theme has been "What might I do with data that is held elsewhere, that I don't own yet?". Myriad is, to a large degree, an exploration of that question with regard to existing Machine Vision systems, which have been bound to singular computers perhaps for longer than most other areas of endeavour. Indeed, many of the themes of Myriad development envisage further progression of this networking effort, toward a vision of Pervasive Computing, which databases, secure connections, data sharing and wireless connectivity seem intent to provide within the next decade.

One of the key features of the modern networked business, however, is the need for scripted systems across assorted areas of operation. As data becomes easier to access and manipulate, roles of work change from data processing towards data management. Perhaps the most profound example of this is the job description of the secretary, changed from typing, filing

and relationship management (organising meetings, liaising with others) toward the personal assistant, a facilitator and data manager, summarising salient points of incoming data, gathering data for future work and ensuring data integrity, whether it be product schedules or whether given people will be at certain meetings. In cases such as these, traditional secretarial tasks are taken up by voice transcribers (typing), laser printers (duplication and printing), database tools (filing and data gathering) and groupware/CRM applications (organisational, relationship/personal information management).

While discussion of changing roles in the workplace might seem a digression, it does in fact become increasingly pertinent when examining future tools for business, including Machine Vision systems. While roles may change, the abilities of the person filling the role must also change, and increasingly the personal assistant is imbued with talents more common in a working programmer, creating small scripts and tools for managing, moving and transforming data, or at the very least, working closely with a staff programmer to create such tools. Adverts for products such as Microsoft Office, OpenOffice and KDE's tools emphasise scripting capabilities as being essential to the modern workplace, saving time and if used on the correct data, offering a competitive advantage.

In the scripted business environment, not only are isolated Machine Vision systems distinct, but where they are networked, they often fail to allow for scripting control. For Vision systems to expand their role beyond production line monitors and basic controllers, there is a need for scriptable control of the MV systems. Also adjacent to this in thinking is the idea of data integration from Vision systems, enabling data from Vision systems to be used within presentations, databases, documents, and to inform actions in other areas of the business. So, a personal assistant might be asked to manage a factory system, ordered to produce parts A, B and C on Wednesday and parts D and E for the rest of the week. Scripting allows this to be done with a few lines of code and easily changed, in a near-English (or other natural language) terms. It might also be important to gather together all of the production statistics each week into a presentation for management; again, a controllable Vision system, able to export and import data is able to full this role.

It is, however, essential to understand that while the use of scripting may still be selective in business environments, high level languages and natural language processing, along with access to more data through the Internet and database use will increase and necessitate this. In

these situations, a non-scriptable GUI to a Machine Vision system is not acceptable in the long term, and as such, remote desktop and single control GUIs produced by LabVIEW cannot be considered to be enduring products or a firm basis for development for the long term.

5.1.3 Components

The present Myriad prototype systems include a number of different components that may be connected together in order to create solutions to MV tasks. These components consist of:

Myriad Server Daemons

- IP
    - Java
    - C++/QT early prototype
- EC
    - Java
    - J2ME early prototype

Myriad Gateway Servers

- Prolog
- Databases

Myriad Data Resources

- Web Servers
- Networked Cameras

Myriad Control Programs

- Control Scripts
- HTML User Interfaces
    - Desktop
    - Palmtop devices
- C++/QT User Interfaces
- Visual Basic User Interfaces
- C# User Interfaces

5.1.4 External Application Component Interface

Myriad in the current state, supports the introduction of external applications and code in order to construct Myriad components with specific functionality. The Myriad server container provides the server aspects of the functionality, along with MCS and regular I/O operations, to which additional functionality may be added, which defines the component's purpose (e.g. image processing functionality to create a Myriad Image Processing Engine). Currently, this container is provided in 1.5-compliant Java, with a developmental C++/Qt version also created, but incomplete. Myriad prototypes include hard-coded image processing and equipment control functions, which is a product of their progressive development. These features are incorporated directly into a command processing loop in the Java source, where command strings are directly compared against a known list of commands in a series of if-else statements. If the incoming parsed string matches a known command, the contents of the if block are executed, until the list of commands is exhausted. At that point, the parser gives up, notes an error in the log and continues with the next line of input.

Lines are of the form:

command param1 param2 ...

e.g. thr 180 210

The processing loop is as follows:

```
Routine: Process Commands
{
        Read Input Line from Lines Vector according to Line Pointer
        Break Line into CommandParts array

        If (CommandParts[0] matches "command1")
        {
                Do command routine for command1
                Read parameters are held in CommandParts[1+]
                Call function for command1 with parameters
        }
```

```
        else if (CommandParts[0]  matches "command2")
        {
                Do command routine for command2
                Read parameters are held in CommandParts[1+]
                Call function for command2 with parameters
        }
        else
        {
        Note error in log
        }

        Increment Input Line Pointer
}
```

While the code is not very elegant, it was produced in this manner for a number of reasons:

- It is the fastest way to write a command interpreter
- It is clear for students to read and easy to produce small extensions for
- It is very easy for even simple compilers to optimise, since it is a set of linear conditions and comparisons
- It is easy to test for creation of a proof-of-concept Myriad prototype, providing stability when it is more important to spend time testing applications with Myriad components, rather than debugging the component itself

In order to connect an external application or piece of Java code to this command loop, therefore, the programmer is required to add a command matching entry to the list, break out the parameters and either call their included Java class or use the java.lang.Runtime class' exec() method, to execute applications through the system command line interface.

On a Unix system, once again, access to wget or curl on the command line is sufficient to access web-based resources and contact other Myriad components, if necessary. In a KDE-based environment, this method may also be used to establish connection to the DCOP server and use that to launch and control applications. On any system where scripting languages are available, this is also able to execute scripts as a local user with the permissions of the user level of the Java Virtual Machine.

Clearly, this method requires recompilation of the server component in order to take advantage of the functionality that the programmer may want to add. This is not particularly efficient, and neither is it very agreeable to require modification of the Myriad source code. In future work, this should be overcome with the inclusion of a plug-in architecture with XML definition documents for the commands and parameters that external plug-ins support. The command processing loop would need to be rebuilt to iterate across a dynamically-sized data structure (a Java Vector is sufficient). At this point, the running Myriad component would be able to parse the XML document and inject the commands described in the document into the known command list. Command could then either call dynamically-loaded bytecode third-party components, or use the command line to access applications.

It should be noted, however, that another of the reasons for not implementing such a system in the Myriad prototype servers, is that the intention has always been to use them solely as prototypes and encourage re-implemention in languages such as C/C++ for performance reasons (indeed, the C++/Qt container is partially developed). As such, it would be premature to invest large amounts on time in developing and testing a plug-in architecture, and somewhat misleading to the reader to present such an architecture as a finished work that should be targeted by other applications.

5.2 Essential Feature Summary

5.2.1 TCP/IP based connectivity

The first and most essential attribute of Myriad components, is that they communicate using the TCP protocol. This allows Myriad to operate over the range of current TCP/IP based networks, which are currently in existence, such as the Internet and Intranets. This definition allows the use of all current key networks, network devices and interaction with connected systems such as databases, servers, network attached cameras and conventional desktop computers. Critically, the majority of contemporary programming languages are able to create TCP/IP sockets and transmit data over them as the default machine-to-machine communication method. This feature opens up Myriad development and interaction to programmers in those languages without additional libraries or creation of low-level protocol writers and handlers.

5.2.2 Web Server base

Myriad Server components are developed from a basic web server design, operating using a well-defined asymmetric client-server model, able to serve multiple users.

A typical conversation is of the form:

1. Client requests file
2. Server parses request
3. Request variables are processed and action taken on their basis
4. File is returned to the client

This design has a number of implications, both positive and negative:

Negative:

- Due to the asymmetric nature of the client-server relationship (see fig.32), the server may not initiate communication with the client. All data is only supplied as a response to a request. Having the client regularly request data, and comparing responses to identify change may circumvent this. Moreover, should the client be able to act similarly as a server, the primary server may return data through the act of emission of a request.

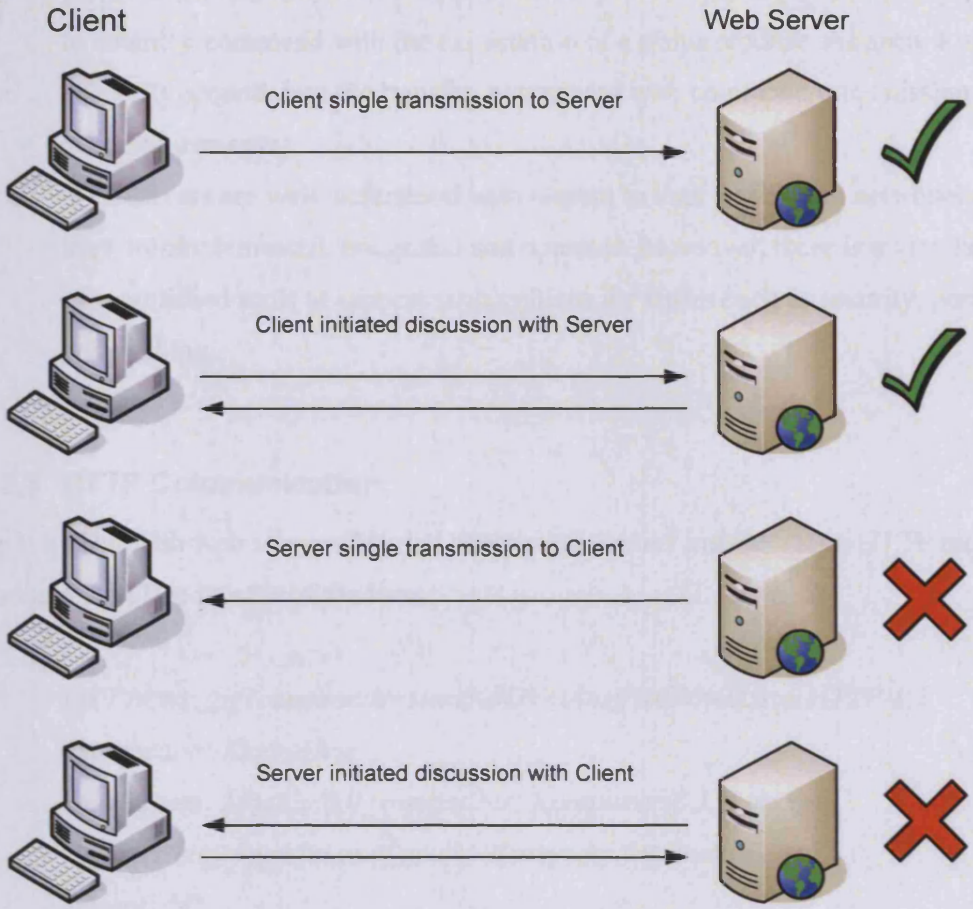


Figure 32 : The asymmetry of HTTP-based client-server relationships

Positive:

- Web server implementation is very well understood, and easy to explain and develop solutions for.
- A web server implementation provides an easy base for rapid development from existing freely available simple server code sets.
- Synchronous data transfer and interaction are generally rare in solutions to real problems and poor network conditions work to impede this method of communication. Local logic is able to mitigate against this to a great extent, and therefore, an asynchronous architecture with local scripting is often more desirable than a synchronous one without scripting.
- Using a web server design, component servers are able to be integrated with existing networked systems, already suited to the operation of conventional web servers.

- Data file transfer is typically desired when systems interact, either to acquire data, or to submit a command with the expectation of a status reponse. As such, a model based primarily around data file transfer, augmented with command transmission fulfils all task requirements.
- Web servers are well understood with respect to their position in networks and how they are implemented, integrated and operated. Moreover, there is a very large range of established tools to support such systems for issues such as security, performance and caching.

5.2.3 HTTP Communication

As is typical with web servers, Myriad component servers interact using HTTP requests. A standard HTTP request is of the form:

GET /temp.jpg?commands=start%0D%0Aneg%0D%0Astop HTTP/1.1
Connection: Keep-Alive
User-Agent: Mozilla/5.0 (compatible; Konqueror/3.1; Linux)
Referer: http://gemini.cs.cf.ac.uk/~dawnrider/interactive.php
*Accept: */**
Accept-Encoding: x-gzip, x-deflate, gzip, deflate, identity
*Accept-Charset: iso-8859-1, utf-8;q=0.5, *;q=0.5*
Accept-Language: en
Host: localhost:3080

The key functions of an HTTP request may be expressed as a combined entity known as a *URL (Uniform Resource Locator)*, formed from a combination of the protocol identifier, the server identifier and the request itself. This combined form is the type shown in the address bar of a web browser and is sufficient to identify:

- The server to which the request is being made
- The data file being requested and its location
- Additional arguments that may change the output data file

As with all web servers, a Myriad Server component is capable of supplying data files without any further modification, on the basis of the HTTP request, provided such a file is known to the server.

In addition to simple file serving, however, most modern web servers are able to use the arguments appended to the end of a URL (GET method) or contained in client variables (POST method) to transmit data to influence the resulting data file or to ask the server to perform an operation. A normal web server passes the arguments of a request to the file being requested, such as script file, which is free to process these as it so wishes. This only applies to files designed to be executable and to parse and process arguments.

E.g.

http://www.myserver.com/myimage.jpg?negate=true

fails to alter output on a conventional web server, as images are not executable, nor able to process arguments. A Myriad image processing server, however does not pass arguments through to the file; instead, it provides its own scripting abilities and processes those arguments itself. If those commands produce a revision of the file being requested, once the operations are complete, the file is returned with those modifications in place, but the file does not provide processing abilities.

It would be possible to integrate Myriad scripting abilities into a conventional web server, such as Apache or Internet Information Server, to provide both methods of processing, but this has not been attempted. Principally, this is due to being quite able to operate a conventional web server on the same computer as the Myriad server, and being able to interact between the two using HTTP requests in a two-way manner. This provides the functionality of a fully integrated server, without the effort of designing and maintaining server plug-ins for each web server and for each Myriad Server component. For example, an Image Processing server component and an Equipment Controller server component together would require 2 x (Number of web servers supported) plug-ins. These plug-ins would have to be written uniquely for each web server architecture, developed in the programming language of the server (or supported language bindings) and potentially recompiled with each

distribution of the server. Using HTTP as an interface allows this replication and high maintenance requirement to be bypassed.

5.2.4 Scripting

Myriad incorporates a scripting language named MCS (Myriad Component Script), a procedural language based around a BASIC-like syntax with extensions for the purpose of the server component. E.g. the image processing server, Myriad::IP provides a standard range of commands, combined with additional image processing operations. Myriad::EC, the equipment controller includes the same generalised commands, with equipment control directives added in order to fulfil the tasks for which it is deployed.

MCS provides local autonomy to component servers, so that they may perform tasks and examine logic directly, as opposed to relying on a controller to manage their operation. As a consequence, time-sensitive operations may be performed on the basis of data which is owned by the server, with immediacy, rather than having to refer the data to a remote controller for decisions to be made at that level. This design protects the overall system from the effects of network latency and failure that might otherwise prevent action from being taken, and minimises the number of interacting systems for rudimentary tasks (e.g."if the image intensity is low, turn on lights").

MCS scripts are passed to the server either as command strings attached to the end of client HTTP requests, or as URLs which are similarly attached to those HTTP requests, which direct the server to where a file with complete script content may be downloaded.

5.3 Components

There are a number of currently developed Myriad components, with more in development to provide classifiers and intelligent controllers, however the present list of components follows.

5.3.1 Image Processing Engine

The image processing server was the primary component developed for the Myriad framework, and comes in two varieties, the Java version, developed first, and the C++ version, which is still in development. The intent of these systems is to provide basic image processing functionality with a structure designed for ease of understanding for future students and researchers.

The Java version of the IPE server is designed to run on Java 1.3 and higher (ideally 1.4) and may be run across all platforms supporting the fully 1.3 JVM functionality. Testing has been performed on both Windows 98/XP and Linux, without error.

The C++ version of the image processor is written using standard C++ with Qt as the primary library. Image processing is currently provided by Matlab through the C++ interface provided by the shipping version of that product. This server is still highly experimental, with a limited range of image processing functions and is produced as a proof of concept and speed benchmark, rather than a final platform. Due to the Qt dependence, this server is capable of operating across Windows, MacOS, Linux and other Unices with a simple recompile, but requires Matlab to be in place for those functions and is as yet, untested.

5.3.2 Equipment Controller

The Equipment Controller (EC) is a Myriad server derived from the Java IPE server. Essentially the same code base as the IPE server, it has the image processing functions removed and replaced with serial port control functions, along with options for selection of the serial port to use on start up. The serial communications are managed by the Java Comm framework, and as such, rely on the server computer running a JVM and an appropriate Comm library. There are additional issues in the way that Windows assigns serial ports that are dealt with in a different manner on other operating systems that makes the current code non-portable to other platforms, but adaptations should be reasonable to perform if required. The current version of the EC server code includes customised commands for the Pan & Tilt camera and J43 Design's M1 and M2 control boards.

A variant edition of the EC server also exists, to operate on the Ajile Systems Inc.'s AJ100 embedded chipset, which runs Java natively in the form of J2ME, a subset of Java 2 (versions 1.2, 1.3 and 1.4). This branch is not being actively maintained, in favour of versions for conventional operating systems, as embedded chipsets supporting Linux and Windows CE become prevalent. One of the key issues for this decision is the wish not to maintain multiple codebases for each server type, one for the embedded system and one for the desktop/server and the limitations that the J2ME implementation imposes, which severely curtail complex operations such as image processing or fine-grained motor control.

The AJ100 test system proved, in tests of repetitive double-looped simple assignment from an array (see below), to perform at roughly 1/60 the speed of desktop systems. As a result, it is more economical in development time and deployment cost to use an x86-derivative embedded system, with significant performance benefits.

Example program:

Loop 1 (counter i)
    Loop 2 (counter j)
        Assignment (x = array(i,j))
    Close Loop 2
Close Loop 1

Tests were carried out for loop values of 500 for both loops, across a 2-dimensional array of random integer values. To aid in accuracy of timing, this process was repeated 100,000 times. Array initialisation and random number generation was performed prior to the timed start, and as such, does not affect results (arrays can be considered to already be initialised and ready).

Desktop PC Specification:
1.2GHz Processor (AMD Duron)
512MB Memory
Mandrake Linux 9.0

Desktop PC Sun Java JVM 1.4.1:
42 Seconds

Desktop PC Compiled C (gcc –O2):
18 Seconds

AJ100:
48 Minutes

Discussions with Ajile Systems suggest that these issues may be due to nested loops, array and object use. This is relatively consistent with Java implementations on the desktop, where

the use of objects, most especially instantiation and less so for method invocation takes considerable amounts of time (once again, benchmarks purporting to show equivalence between compiled C code and Java execution often avoid the use of objects). Unfortunately, for any operation relating to the processing of images, it is essential to use arrays or complex data structures of this type. This difference between what the AJ100 chipset finds most beneficial and what is required for image processing tools is intractable, and therefore, rules out deployment of an AJ100 system for image processing problems.

5.3.3 Prolog

The use of Prolog and intelligent reasoning with Image Processing and Vision tools can assist in a variety of tasks where linear and iterative reasoning is inappropriate or does not provide satisfactory results. There are two options when there is a requirement to make a program or resource network-aware. Firstly, a server may be written in the given programming language, able to accept connections and process data; alternatively, the program may reside on the same machine as a web server and a server script may act as a gateway, passing data back and forth (see fig.33). A number of commercial Prolog implementations are capable of operating as servers, such as LPA's MacProlog and WinProlog.

In the case of Myriad's development system, SWI Prolog is used on the same machine as a web server, using a PHP script as a gateway to interact with the Prolog program at the command line. SWI Prolog is an suitable choice for the core of a Prolog component, as it supports the full Edinburgh Prolog standard, and also has the ability to create and execute HTTP requests, allowing it to interact with other elements of a Myriad network as part of normal operation.

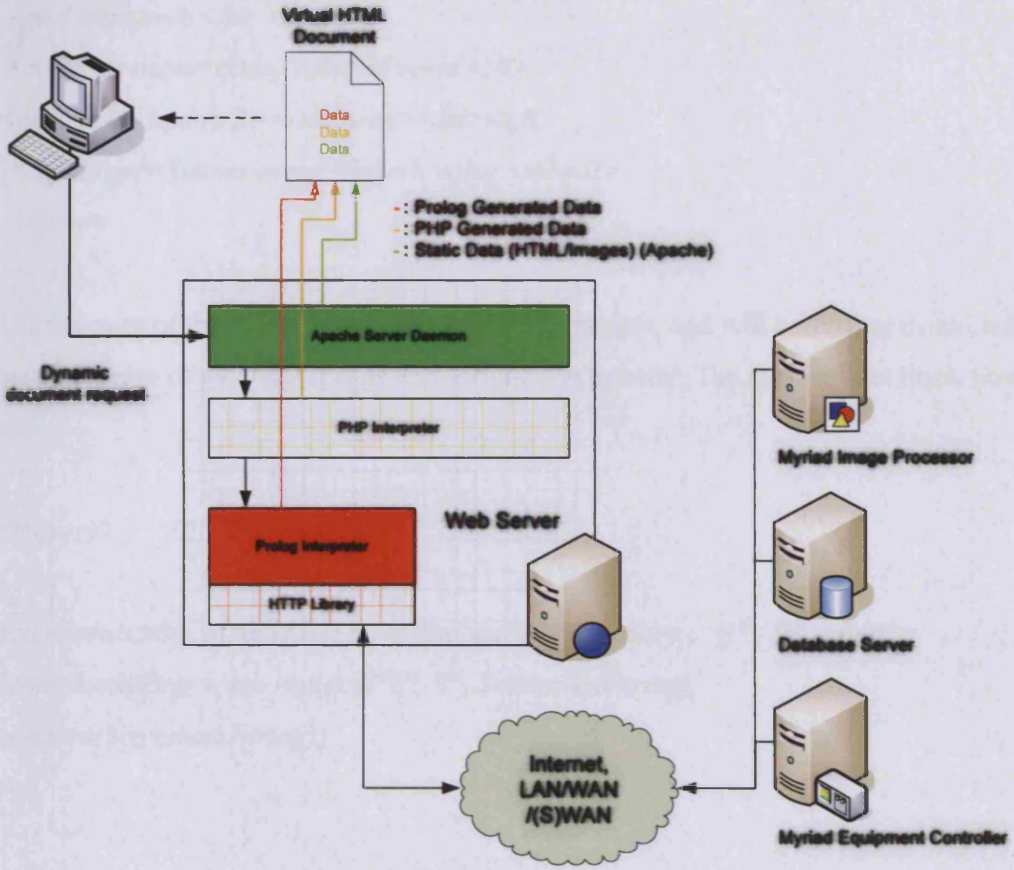


Figure 33 : Using PHP and a Prolog interpreter to expose an intelligent gateway for Myriad component control

The PHP gateway script consists of two sections; one which provides an HTML form which the user can use to enter commands and submit them to the server, and a processing section, which accepts the commands and passes them to the Prolog interpreter.

```
<?php
if($query)
{
$commandstring = "pl -f test.pl -g \"solve(" . "(" . $query . "))" . "\" -t halt";
$commandstring = str_replace("\'", "'", $commandstring);
passthru($commandstring);
}
$query = str_replace("\'", "'", $query);
?>
 <form method=GET action="test_prolog.php">
```

```
<p>Commands</p>
<textarea name=query cols=80 rows=10>
<?php echo $query ?></textarea><br><bR>
<input type=Submit name=Submit value=submit>
</form>
```

The majority of this script is generic to gateway scripts, and will be further examined later, in the discussion of gateway scripts and scripting in general. The most salient lines, however, are:

```
if($query)
{
$commandstring = "pl -f test.pl -g \"solve(" . "(" . $query . "))" . "\" -t halt";
$commandstring = str_replace("\"", "'", $commandstring);
passthru($commandstring);
}
```

While the syntax may seem a little complex to conventional OO-language programmers without Web programming backgrounds, it involves:

```
if($query)
{
...
}
```

This is a simple condition which will be satisfied when *$query* has a value. Examination of the script may seem to reveal a flaw, in that the *$query* variable is never instantiated prior to this test; the reason for this, is that PHP automatically instantiates all web form key-value pairs into variables, when they are found. So using a web browser, calling upon the script *test_prolog.php* as:

http://www.myserver.com/test_prolog.php

will mean that *$query* is not instantiated, and therefore, the test will fail, and no Prolog processing will be done. The following URL is different, however:

http://www.myserver.com/test_prolog.php?query=test

Requesting this URL will start execution of the script, except that a variable named *$query* will be automatically instantiated ready for the test, which will succeed, as the value of the variable *$query* is the string *'test'*. It should also be noted that PHP variables are weakly typed, and thus get converted to floating point numbers, integers, strings or other common types implicitly when used as such.

Another interesting aspect of this script, is that after the end of the test, there is embedded HTML that creates a form in the web browser with one open text area for the user to type their Prolog commands (named *'query'*, which is how *?query=test* is added to the URL automatically when *'test'* is typed in the text area). This form is shown, regardless of whether Prolog commands are sent to the script or not and the input is sent to *test_prolog.php* itself; so the script is the interface and the processing stage wrapped into one script, sending information back to itself. A final point to note on the subject, is that the text area also has the text *<?php echo $query ?>* between the opening tag and the closing one; meaning that when commands are sent to the script through the query value, they are also injected back into the text area when the script finishes executing and produces the web form again, so that the user can iteratively improve their Prolog program.

The next line:

$commandstring = "pl -f test.pl -g \"solve(" . "(" . $query . "))" . "\" -t halt";

may seem complex, mainly due to the extensive use of quotation marks. In essence, however, this line is creating a variable containing a string value. The value of this string is the command line command needed to execute the Prolog interpreter. The results of this, with the value of *$query* mentioned previously are as follows:

pl –f test.pl –g "solve((test))" –t halt

The executable program is named 'pl', and the options are:

-f test.pl	Consult the file test.pl on startup
-g "solve((test))"	Test the goal *solve((test))* where *'solve(A)'* is a predicate in the file test.pl
-t halt	On termination, run the command *'halt'*, which terminates the 'pl' program entirely

The quotation marks around the *solve((test))* are there to ensure that the goal name is passed as a complete option, including spaces, so that *–g solve((this is a test))* will not attempt to solve the goal *'solve((this'*.

The subsequent line is in much the same vein as the use of quotation marks. Using UTF-8 encoding for the web form data such as the Prolog commands from the user through their web browser, apostrophes are translated into \', an escape sequence to make sure that they are not evaluated inappropriately; unfortunately, this means that the slash must be removed to restore the commands that the user intended, prior to sending it to the Prolog interpreter. *Str_replace* simply replaces all occurrences of \' with a single apostrophe ' in the variable *$commandstring*, and then puts the corrected string back where it came.

```
$commandstring = str_replace("\'", "'", $commandstring);
```

The final line of the section is the important:

```
passthru($commandstring);
```

This command passes the contents of the variable given to it within the brackets to the Unix command line for execution and when there is anything printed from the program to the command line, it passes it through to the user without change; hence the name of the command. Therefore, if the Prolog program outputs any information or messages to the command line during normal operation, this data will be sent to the web browser of the user, and displayed as normal text. This also means that the Prolog program can output HTML as text and have the formatting display nicely in the web browser.

The objective of this PHP script, is therefore to take the 'query' data from a web form, filter it, pass it to the Prolog command line interpreter and relay the interpreter output back to the web browser.

5.3.4 Databases

Myriad demonstration systems currently use MySQL as their DBMS, with the option to add drivers directly for different databases, provided those exist for the language that is being used for the server, or a gateway script may be written to wrap around access to the database. This is primarily due to ease of installation of client drivers in Java, detailed acquaintance of the author with MySQL and automatic installation and set-up of MySQL in most modern Linux distributions, presenting a fast means of testing SQL functionality. Moreover, MySQL disables complex functionality such as transaction support by default (it can be enabled), but is optimised for simple select, insert and update queries, making it ideal for the basic testing that the demonstration requires.

5.3.5 Web Servers

In addition to Myriad servers providing basic web serving, it is also advantageous to have a contemporary complete web server to perform more large-scale data serving and allow for the use of web scripting languages such a Perl and PHP. For this reason, Apache is used extensively in the Myriad test systems, with a view to providing HTML-based user interfaces, gateway systems to databases and other resources and also general monitoring. With a wide range of scripting languages and modules that enable secure communications, URL rewriting and extensive logging, Apache is ideal for the purposes of scripting, testing, analysing and providing a front end to Myriad solutions.

5.3.6 Networked Cameras

The present Myriad configuration uses Axis 2100 network cameras as the basis of networked image capture and image availability. These cameras attach directly to a local area network, running the Apache web server atop embedded Linux. Using the web server, remote users are able to download images, configure the camera and redirect images to alternative servers using FTP to upload. Presently, images are uploaded to a single web server to aggregate the outputs of multiple cameras for ease of access and processing.

5.4 How Myriad addresses Limiting Problems

First and foremost, Myriad is a framework for the creation of network-connected Machine Vision elements, designed to operate at a local level and across federated networked devices located at distance from each up (even on other continents) to solve MV problems.

5.4.1 Location

By using conventional HTTP interaction over TCP/IP, enabling the use of existing LAN,(S)WAN and Internet connectivity, Myriad allows differing MV components to be connected wherever there is network (and especially Internet) connectivity.

With the advent and propagation of wireless networks and high speed data over cellular telephone networks, Machine Vision equipment may be added in unwired or roaming locations, with minimal cost. While installing MV systems in unwired areas is not unusual for disconnected classical systems, the ability to have these systems feed data to, and utilise data from other networked systems is innovative. Moreover, Myriad allows components of MV systems, or normal applications, to take advantage of MV tools and techniques which may be available only in remote locations.

In short, Myriad breaks down the requirement that all data, Machine Vision elements and tools for processing the output be located with each other, most specifically when they are required to reside on the same computing device.

5.4.2 System/Data Integration

Developed from the constraint of having all data processing elements in the same location, is the problem that this prevents data from being used within other business systems; because these are so rarely located on the same computer as the Vision system. Similarly, the Vision system itself is unable to take advantage of data from other systems to inform operation.

A good example of integration change is the manufacture of components with a given degree of variability where bad components are ejected from the processing line by the Vision system. A disconnected Vision System would rely on internal logic to decide the threshold of acceptability and if changes needed to be made to the process to improve a given metric, the rejected parts would have to be counted and analysed by hand. A network-connected Vision System, however, would be able to enter the number and type of rejections into a company

database. This data could then be analysed by another program, the manufacturing process could be altered by another networked Vision System and process change could occur in response to problems in real time. Not only this, but the original system could be informed when requirements for quality were to change, altering the threshold for rejection. This type of situation is common in areas of industry such as computer processor construction, where poorer-quality parts are sometimes acceptable as slower-running products, where the degree of acceptability depends on the sizes of performance markets that change regularly. Further, the data of yield, as monitored by the Vision System could be used immediately to alert international management staff of manufacturing problems and to balance the quality of output and stock across multiple factories, thanks to networking of the database system.

This type of data integration of MV systems with each other and wider systems, offers a range of benefits for manufacturing industry, to allow operations to be more responsive to requirements and changing conditions.

5.4.3 Dynamic Systems, Problems and Situations

Conventional Machine Vision systems operate most effectively on well-defined, static problems in a static, predictable environment. Indeed, one of the key requirements of a Vision solution is the facility to control the environment and problem far enough to constrain the problem until it is solvable.

There do exist, however, two other forms of problems; the dynamic solvable problem (where conditions or the problem change, but the application of further tools, such as lighting changes or additional processing can make it solvable) and exploratory problems (where the intent is not to solve a problem per se, but to investigate the problem (often used for feasibility studies, this type of problem makes use of human operators and interactive processing and control mechanisms)).

Examples of both types of problem are as follows:

Dynamic Solvable Problem

Suppose that a Machine Vision system were developed to analyse a particular type of component and compare it to a 3D template produced by the design of the aforementioned

component. During a particular period of time, the designer alters the design, and so the parts being analysed change. A static solution would fail in this case, but a dynamic system would potentially update the 3D template, alter any lighting as required (through interaction with another MV system, connected to the lights through an equipment controller hardware component) and call upon additional processing facilities that it was incapable of, thanks to network-connected remote processing units. Using Myriad components and MCS, the Myriad scripting language, this type of response may be quickly developed and altered, without local operator interaction or extensive reprogramming and extensive system downtime.

Exploratory Problems

While there exist many products, techniques and solutions for fixed Machine Vision problems, situations such as remote inspection of equipment and environments are not so easily catered for.

Suppose that a highly-skilled Machine Vision engineer is employed to inspect a factory for the possible installation of a Machine Vision system (to monitor and quality control the manufacturing process), but the cost of transporting the engineer to the site, hosting them and performing the inspection is prohibitive, or physical circumstances make this impossible (e.g. the Vision engineer is unable to fly, but the factory is on another continent to the engineer's home, or factory access is restricted for security concerns).

To solve this problem, a local assistant takes the place of the engineer and wears a wearable computer with attached cameras, sensors and wireless Internet access. The Vision engineer is then able to direct the assistant to act for him on the local site, collecting data (light levels, noise, dust) and camera imagery to build a detailed impression of the factory environment. From the incoming data and images, the Vision engineer would be able to monitor the data directly, or call upon Myriad components to process the data, include database data and references, presenting a complete final result for human analysis (see fig.34).

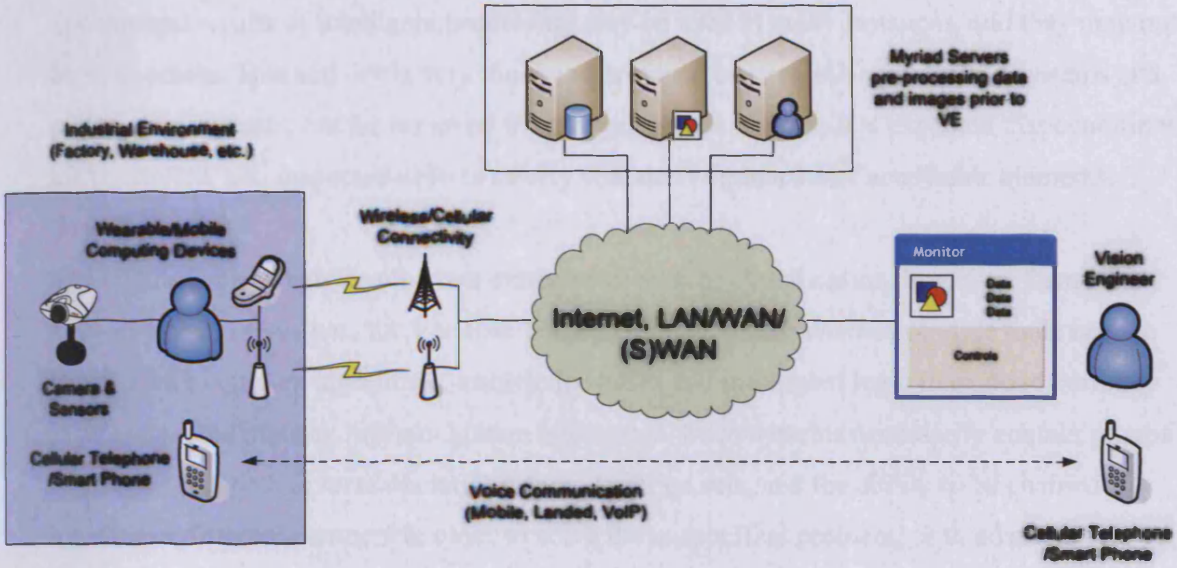


Figure 34 : Vision Engineer and local operator utilising network-connected communication and devices with Myriad components to inspect remote sites for Vision System installation

In point of fact, being able to take advantage of the networked capabilities of Myriad components and access to knowledge bases, with the ability to highlight points of interest in the images and data being received, and presenting inferred information to the Vision engineer, allows such a system to perform a task that is better classified as Machine Assisted Vision (MAV).

5.4.4 Functional Flexibility

One of the principle benefits of Myriad systems in use, is that solutions are non-static in nature, and can be extended or altered in order to add flexibility, tolerance and operational variability to the system.

While there is a clear delineation between MV prototyping systems and target systems in terms of flexibility (prototyping systems being interactive and functionally diverse, intended to tackle unknown problems, target systems are specific and operate within tight constraints), there are also a class of problems which require flexibility in the tools being used to solve them. In terms of the remote inspection problem, there is a high degree of variability in the situations in which the equipment will find itself, with processing of images being highly interactive and potentially assisted by the ability to control lighting in certain circumstances. This is not to say that lighting will always be required; merely that it might be useful in particular cases. Similarly, access to data, such as corporate databases, lighting advisor

systems and results of intelligent processing may be used in many instances, and they may not be so in others. This activity is very much the province of interactive processing systems and prototyping systems, but far removed from target systems, where it is expected that conditions are controlled, and inspected objects clearly contains within a set of acceptable elements.

The middle-ground between the two extremes of system classification, however, forms the basis of a class of systems for Variable Target Systems, where interactive-style tools need to be used with complex algorithms, knowledge bases and intelligent logic to respond correctly in an automated manner, without human interaction. Such systems necessarily contain groups of components with diverse operations, large function sets, and the ability to be chained together in different manners in order to solve the unspecified problem, or to adapt the specified problem and solution to changing conditions.

Not only can Myriad applied systems be adapted through the re-organisation of the component data chain, but adjustments to the system may be automatically performed at runtime based on internal logic. E.g. An image being processed might suffer from poor lighting, to which an image processing server might engage an equipment controller to turn on further lights, or it might request an intelligent controller to work out the most beneficial change in the current lighting conditions. Depending on the inspection task being performed, one option might be preferable to the other, or both equally sufficient. Being able to examine this at runtime and make a judgement as to which method to use enables the system to be most efficient and provide the best output. Such a degree of interconnection and variability in execution is not desirable in rigidly-installed and controlled target systems, but similarly, would not be possible for interactive systems without human intervention; thus, flexibility of this kind promotes a new class of semi-intelligent flexible solutions that are able to constantly compensate for alterations in constraints.

5.5 Limitations of Myriad

5.5.1 Current

The Myriad prototype servers, as outlined here, are significantly limited in many aspects. First and foremost, in that they are the product of a single author over the course of a year and a half, along with other attendant duties. As a consequence, the quality of the source code involved is highly variable, and it should be expected that limit conditions and exceptional

cases may cause failures, even in common functions such as conditions, or unforeseen transformations of image processing functions. The Myriad prototypes are intended as a proof of concept, not as production-grade systems, and they must not be used to perform functions that are mission or safety critical. Aside from the limitations of the quality of Myriad code, the prototypes are built using Java, the license for which clearly states that Sun Microsystems does not distributed Java JVMs or technologies for critical use.

Beyond questions of reliability, current Myriad components lack a proper classifier to perform intelligent assessments of data. A provisional classifier that is able to be adapted to utilise Myriad components has been developed by Mr. Mike Dicks of Cardiff University; however, it does not perform as a server component in its own right, but rather is an application that is able to acquire data from Myriad Equipment Control and Image Processing Servers. Developing a classifier for use in this manner would be useful in application to complex, dynamic problems, where conditions may change and generalised perception of data attributes is essential.

As a consequence of these factors, the applications of Myriad in the present form are restricted to Prototyping Systems for image processing and equipment control, with databases, intelligent deduction and web incorporation. All of these aspects are able to be manifested in tasks which require interactive control for non-critical, but exploratory problems.

Myriad prototypes also provide multiple user interface applications, both generalised and specialised for Windows, Linux, Palmtop devices and scripts, using Java, VB, C#, HTML & PHP, Perl, Prolog and C++/QT.

In terms of performance, Myriad prototypes are developed with reasonable programmatical competence in order to ensure performance free from common failings, such as instantiation within loops. That said, the Java server components are intrinsically speed-limited due to the execution of the Java Virtual Machine and the degree to which that fails to achieve the performance level of natively-compiled source code. Reimplementation of the prototype servers into C++ and QT would improve speed of execution, as would the implementation of high-speed image processing operators, such as Fred Waltz's SKIPSM techniques [WALT], which perform operations using Finite State Automata.

5.5.2 Conceptual

No Network Indexing/Listing System

At the present time, the Myriad prototypes do not include any sort of networked list system with announcements to allow a person to visit a single site and see a range of currently available Myriad servers. This is moderately constraining, as it requires the user to be aware of the URL of any server that they wish to use. This list service is not present for a number of reasons:

- It would add additional complexity and length to this thesis
- It would not be valuable with only a few test servers available
- Implementing a simple list service is easy; implementing a good one is hard. Issues such as:
    - Load balancing
    - Locational preferencing for latency and bandwidth optimisation
    - Command set classification
    - Security
    - Anonymity
    - Data routing vs. connection hand off
    - User profile support and collaboration
    - High-performance list management and searching
- A number of good solutions are close to being ready for consumer use, such as Zero Configuration Networking and LDAP-powered DNS service announcement for peer-based small network applications

In light of these considerations, it would be premature at this point to produce a listing service and define it as the official manner to list and manage large numbers of Myriad servers. This is not to say, however, that such a listing system is not desirable; it is very much so desired and would form the basis of large-scale collaborative Myriad server networks with easily-accessible functionality.

Slow socket performance for local processing

While Myriad servers are able to perform functions both as local and remote servers, this type of operation does pose a primary difficulty, in that the use of TCP/IP sockets for interaction restricts data transfer performance and exposes the communication to intermediate filtering

systems, such as Quality of Service agents and local proxies. The degradation to performance in contrast to direct RPC mechanisms is not insignificant and the latencies introduced have lead to a number of similar server-based systems adopting special methods of operation for local communication. The most applicable example of this is the X Windows implementation provided by XFree86, which replaces the TCP/IP transport layer for remote interaction with UNIX sockets for local connections. This switch of communication type is transparent to the user of the system, however, and provides for low latency response to requests for action within the same computing device. In Myriad terms, this type of fix could be adequately produced with the addition of a second listening thread for connections, which listened to Unix ports or RPC mechanisms for the running system, rather than TCP/IP HTTP ports. Using a shared handler collection, any connection to the server, regardless of the transport would be dealt with in the same manner, and with session management and persisting handlers, connecting clients could switch between different transports while retaining their session.

No Plug-In Architecture & Live updates

One of the primary restrictions of the Myriad prototype systems from the perspective of a programmer, is the lack of any ability to incorporate new modules to give a server a new purpose, most especially while it is running. This is a limitation for equipment control, not being able to load new drivers at runtime, or at start-up, in order to work with new equipment. A common development in C and C++ applications, this feature is not present in the prototypes, due to the lack of pointers in Java posing restrictions on the naming of and reference to variables, objects and classes to those known at compilation time. It should be stated that the Java is limited by the lack of pointers and the use of Virtual Machines, in that the code being produced from source is neither interpreted (preventing direct substitution of source code into the command array), nor is it compiled with pointers and a stack so that previously compiled pieces of code may be placed specifically in memory and executed from that point as binary, compiled, executable elements. There are methods for building module systems in Java, but these revolve around the inclusion of pre-compiled byte-code modules and inventive use of the Virtual Machine in such a manner that mimics the use of pointers and execution stacks.

Clearly the production of a module system, allowing a single Myriad server component code base to incorporate multiple operational modules that turn the generic server into an Image

Processor or an Equipment Controller would be advantageous, but in terms of proving the desirability of a networked machine vision tool, it confers limited benefit to this project as it stands at present. It would be expected, however, for a production Myriad specification to introduce some form of module system specifically for repurposing of servers, enhancement, patching and the introduction of new processing and control abilities, such as additional drivers for new equipment which was previously unknown.

In this chapter, networked vision systems were examined, with illustration through the medium of Myriad, the HTTP-based networked machine vision framework implementation. Considering the problems imposed to vision system creation across large distances and difficulties in remote control and aggregation of vision data and tools in one location, Myriad's potential for addressing these problems was introduced.

Using networked machine vision systems, data and resources become portable and more readily usable from remote location, so a vision engineer can gather together the tools and output from sensors that are required to study a problem, and remotely operate equipment to explore problem solutions. Solutions may also be developed utilising tools that the vision engineer does not consistently have to hand (such as processing clusters and database-backed knowledge bases), which can be coupled together and decoupled as the vision engineer works on a problem. Not only this, but the vision engineer has freedom of physical movement that allows work to be done whilst travelling, or multiple tasks to be addressed simultaneously.

5.6 Security

The security of Myriad components is not explicitly defined within the initial prototypes, the concept, nor the generalised specification. Security is not a feature of the system itself, but of the environment in which it operates. As Myriad components may be considered enhanced HTTP servers, which are commonly deployed in networked situations, and for which high quality access security systems and techniques already exist, there is little necessity to develop new security specifically for Myriad, but rather, to leverage these existing systems and provide guidelines as to their maximal use.

The reasons for this decision are manifold:

- High quality security systems already exist for HTTP servers, such as firewalls, VPNs, proxies, SPI (Stateful Packet Inspection) firewalls and IDSs (Intrusion Detection Systems).
- Producing a new security environment is counter-productive:
    - Equivalents of existing systems must be created.
    - Where existing systems have resolved bugs and circumventions, new systems must tackle these once again.
    - Administration of Myriad security would be isolated from existing methods, making management harder and increasing the risk of failure.
    - In cases of cross-site security, such as VPNs and trusted networks, Myriad servers would only be able to interact with other Myriad systems, producing the type of isolation that networked Myriad components are designed to break down.
    - Producing new security measures to utilise the full range of modern security and access control systems would take far more effort than producing the Myriad servers themselves.

Finally, it must be admitted, that a competent security regime that encompasses a full range of requirements and mimics the strengths and interaction of contemporary networked systems is far beyond the skills of the author, and indeed, would need the attentions of a large number of dedicated security experts and veteran security programmers. Due to the nature of Myriad components, potentially exposing dangerous equipment and important information sources to a network, whether open to the world at large, or restricted, it is essential that risks of intrusion and subversion be minimised, most especially through programmatical error. In this case, inclusion into existing security frameworks is the only reasonable means of deployment, as it would take years to bring a bespoke system to an equivalent level of competence (at which point, existing systems would be further progressed, regardless).

5.6.1 Granularity of Control

There are three primary degrees of control over Myriad components and systems, each which defines a different level of access to the entire system, and control capabilities.

1. Access control
2. Command filtering
3. Resource control

First amongst these, is access control; the process of deciding whether or not a given user should have access to the Myriad system at all. This is produced through the limiting nature of network access and authentication controls, such as firewalls and intelligent gateway systems (normally tied to user databases, ACLs or LDAP/NT domain systems). Using these systems, requests to access a Myriad server can be blocked prior to reaching in the server itself, producing a hard lock on access.

The use of intermediary gateway systems (see fig.35), however, has broader implications. Not only can discrete levels of access (Access/No Access) be defined through additional information, but more subtle degrees of access can be applied. Taking an example of a database-backed user authentication system; it would be possible to also store the range of commands that a given user is able to access, and then strip those commands using a gateway system, prior to the request reaching the Myriad system, preventing the user from executing commands that they should not. This can be thought of as command filtering using a filter constructed from database information.

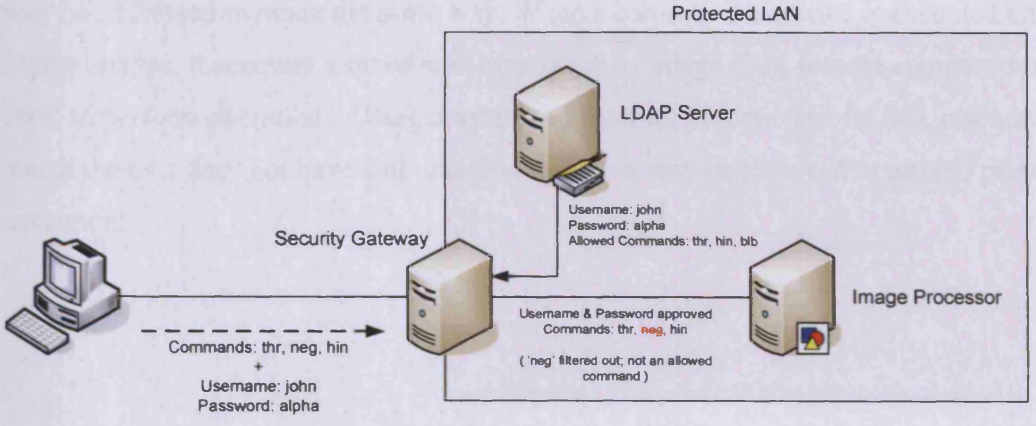


Figure 35. Using a security gateway server to control access to an IPE within an internal network, utilising data from an LDAP server to authenticate users and filter command strings

Operations of this type can commonly be performed using pattern matching systems, such as those found in languages like Perl, PHP and Python, which are able to examine command sequences and requests line by line and replace/remove non-corresponding strings.

```
<?php
// We need to split the $commands string up, process them and merge them
$command_bits = split("%0D%0A", $commands);
$new_commands = "";

// We need to examine each command in $command_bits
foreach ($command in $command_bits)
{
// Do a regular expression replacement, ignoring case
$command = eregi_replace("neg", "negate", $command);
// Join the new command with the new command list
$new_commands = $new_commands . "%0D%0A" . $command;
}
?>
```

The final level of control; resource control, is in part, a subset of command filtering, in that it may be addressed in much the same way. When a command sequence is executed on a Myriad server, it accesses a set of resources (such as image files, remote computers and data files) to perform operations. Using a command filtering process (see fig.36), resources to which the user does not have authorisation to access may be removed or altered prior to execution.

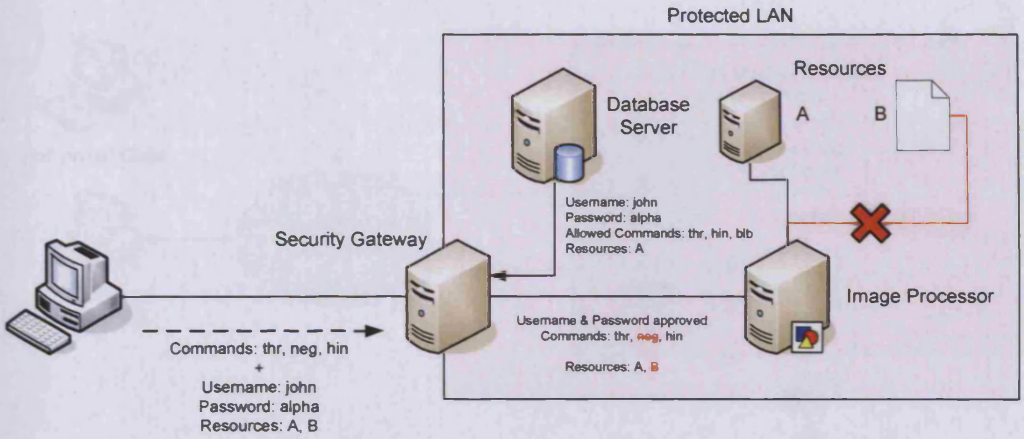


Figure 36. Using a security gateway to control access to an internal network IPE and alternative resources, with a database providing authentication and resource permission data. Commands and resource requests are filtered by the gateway on the basis of this data

In addition to this method of control, however, there is a second method of resource control, which is far more strict. In this method, multiple Myriad servers may be run on the same computer, each listening to different port numbers, waiting for users to connect. These systems have their own access control provided by a gateway system and are run as different system users. Programs run on Unix-type (and progressively, Windows as well) operating systems have groups of users with different access privileges to data resources, devices and connections. These are formed from Unix permission lists, or more recently, Access Control Lists (ACLs) and LDAP plug-ins, which use these data sources for more fine-grained user access information (also easier to administer using hierarchical tools and exception management). All programs running as a given user are provided with the access privileges of that user and no more, unless explicitly granted by a higher-level user. Using several Myriad servers, each running on different user accounts (see fig.37), resources can effectively be securely restricted using conventional kernel security measures and quickly administered.

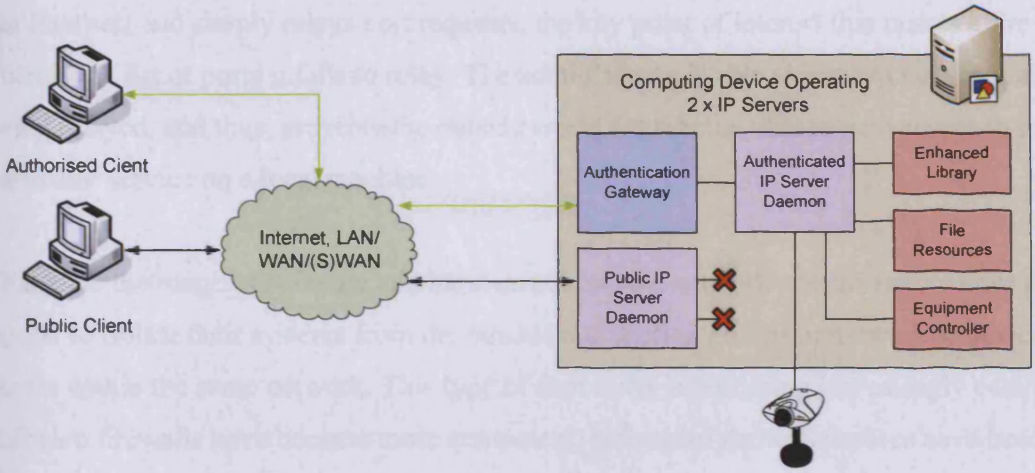


Figure 37. Two image processing servers operating on one networked computing device; one for public use, with limited resource access, and another relayed by a security gateway with access to a wide range of resources for authorised clients

To do this with Perl, see Appendix D.1

Using these three access control methods, we have control of:

- The servers users can connect to
- The commands users may run
- The resources that users may access in the running of their commands

5.6.2 Network Integration

While Myriad as a framework does abdicate responsibility for security to existing networked systems, this may also be seen as one of the key strengths of the framework, as it is perpetually up to date and may take full advantage of new innovations, while integrating smoothly with established structures and allowing for unified administration.

Firewalls

By far the most common form of network security, the modern firewall is a very powerful tool in securing networked systems. To the uninitiated, networked computing devices communicate to each other using TCP/IP using distinct port numbers which indicate the type of service that is being used (for example, 21 is FTP) and server programs listen on the connected-to device for a connection on their port. A firewall is a device (software or embedded hardware) which sits between a computing device and a network (most commonly

the Internet) and simply relays port requests; the key point of interest that makes a firewall potent, is a list of ports it fails to relay. The administrator is able to prevent certain ports from being relayed, and thus, prevents the outside world from being able to gain access to a particular service on a local machine.

Thanks to the range of software and hardware firewalls, network administrators have the option to isolate their systems from the outside and section groups of networked devices from others within the same network. This type of formation is becoming increasingly common as software firewalls have become more competent, embedded firewall devices have become cheaper and more available (thanks to the advent of large-deployment home networking required by the growth of broadband Internet access) and better administration methods have developed, such as high-quality HTML interfaces to embedded systems.

Beyond the normal port-blocking methodology, however, firewall systems are becoming increasingly intelligent and complex, allowing for port filtering, redirection, logging and intrusion detection, allowing them to form the basis for network shaping and advanced security.

Port Filtering

Most modern firewalls do classic firewalling through the judicious application of port filtering, essentially customised port blocking. While this may seem little different from blanket refusal to relay port activity, there are additional options for restricting blocking to only certain internal IP addresses and allowing full access to other IPs, allowing for fine granularity of control per-machine.

Not only can filtering isolate local computers and ports based on IP address, but it can also operate in a reverse manner when filtering and restrict access to given groups of external IP addresses, allowing access to services to be limited to prescribed remote computers (see fig.38).

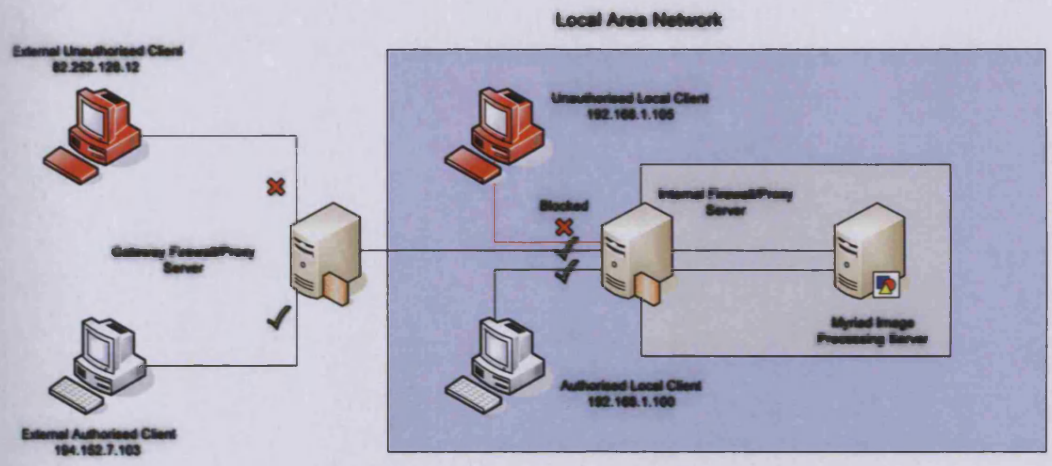


Figure 38. Firewalls and proxies being used for both local area network connections and wide area network connections, to provide access control to authorised clients locally and remotely to IPE servers

In addition, traffic can be filtered uniquely based on the protocol being utilised, allowing for different settings for UDP than TCP; while this option may not be used frequently, it does offer the ability for administrators to control access to rapid access systems and especially streaming video and audio systems, for which UDP is a far better protocol choice. Advanced port filtering is likely to be a common feature in secure Myriad solutions, with the intention of limiting access to networked devices, such as streaming cameras or allowing control of Myriad components only by certain ranges of IP addresses that denote partners, other organisational offices and computers, or authorised users (see fig.39). Using IP-based filtering is an easy way to set up leased computing systems for large-scale Myriad systems, such as image processing clusters or large knowledge-base analysis systems.

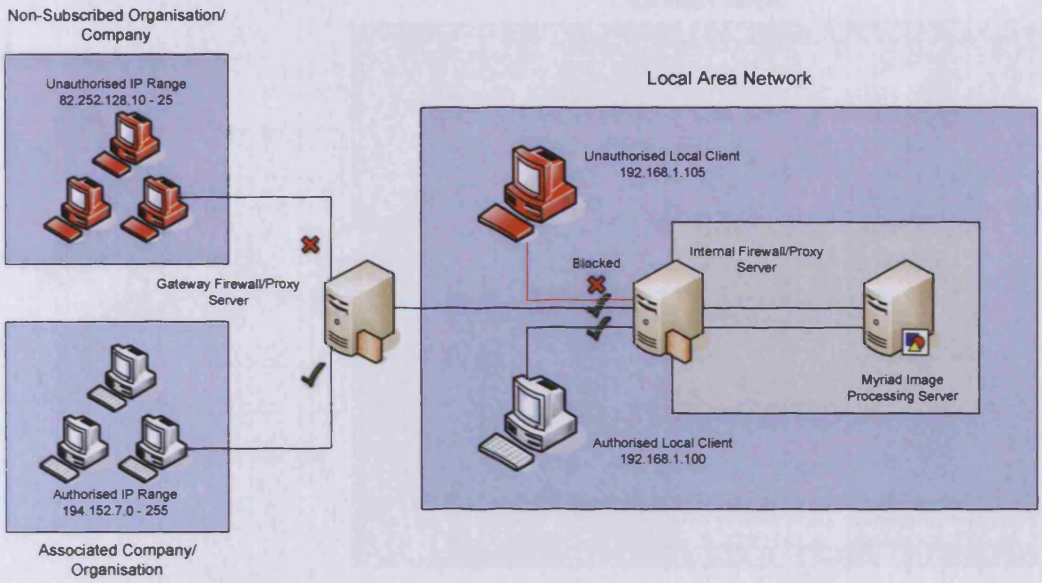


Figure 39. Using firewalls to manage traffic between a local area network and remote customer networks based upon IP address ranges

Port Forwarding

Limiting access to certain resources and restricting the computers that may connect to those resources is a useful way of managing networked devices, but often there are requirements to actively redirect access from one device to a different computer on a network, transparently. This becomes essential if there are large numbers of users and resources, which need to be matched, often on the basis of advanced rules, such as task priority, financial incentive (group Z has paid more, and hence, when they need access to resources, they get directed to higher performance or rich resources) and departmental access.

Using port forwarding, it is possible to manage connections from multiple sites. For example, if six companies leased processing time from organisation X, which employed three processing clusters, using controlled filtering, the users could be segmented into three groups and two users be passed to each cluster on a regular basis (see fig.40), in much the same way that subscribers to small web site hosting share servers.

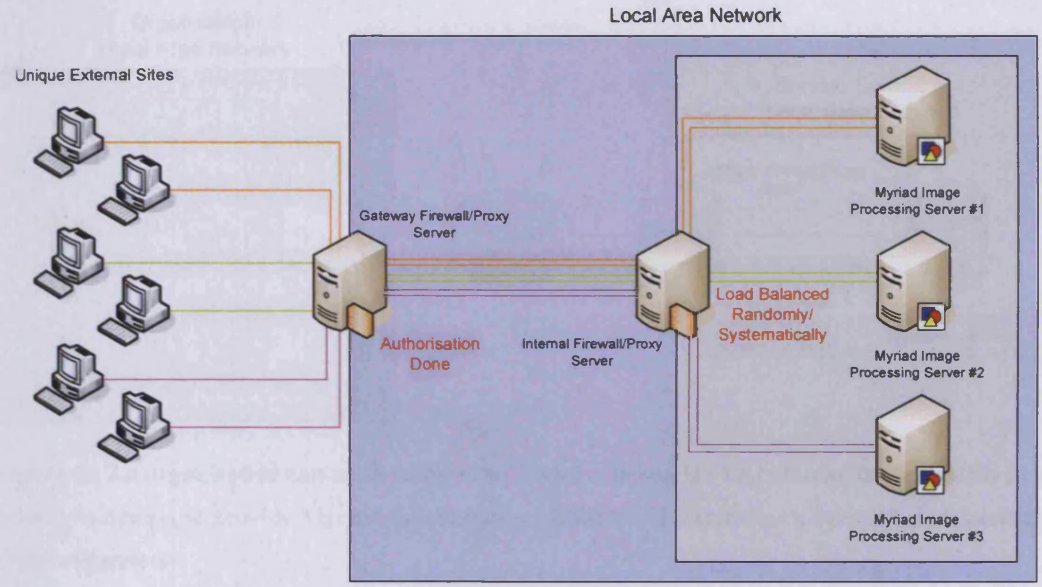


Figure 40. Firewalls may be used in combination with load-balancing server gateways, to direct client traffic to specific Myriad servers, to distribute load and to direct connections from certain clients to designated server resources

Interestingly, it is also possible to use firewalls with port forwarding and redirection to act as an aliasing mechanism for computational resources. For example, if a company decided to institute an image processing service on a local network, providing an IP address and a port number for users to access, it might use a local computer or redirect to an external leased service (see fig.41), which would do the processing, acting to the local user as if the processing was done on the same network (excepting, of course, the potential increase in network latencies of using an external service).

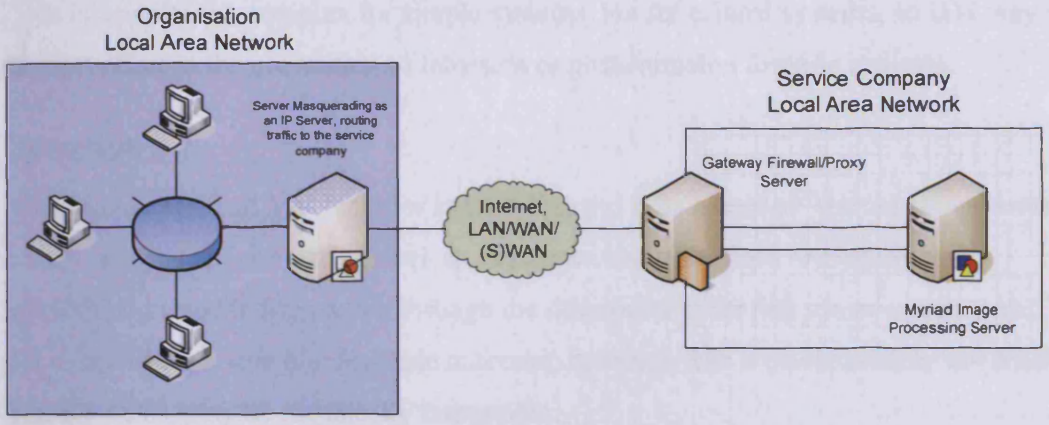


Figure 41. An organisation can use a computing device relaying HTTP commands and results to a remote service company, to provide Myriad functionality within the Organisation's network, transparently to client computers

Logging & IDSs

One of the most essential features of a modern firewall system, is the ability to keep accurate logs of access, to analyse these logs, and to ultimately realise potential security failures, and alert the administrator to both resolve the cause of the failure (the exploit) and also to end the intrusion.

For the most part, firewall logging systems operate by recording incoming and outgoing connections, in formats that make them suitable for processing by a language such as Perl later, with the option of storing the log on the firewall device itself, or exporting to a remote computing device, such as a core server, for analysis.

Analysing log files from a firewall requires a scripting language or system able to do basic string manipulation and, most especially, regular expression matching (most commonly used to match IP ranges and exploitative requests that denote attacks, e.g. "192.168.1.*" to classify all local IPs in the 192.168.1.0-255 local subnet). As the requirements for analysis and the risks of attack have increased, as have the types of system developed to analyse the logs, progressing to what are now known as Intrusion Detection Systems (IDSs), the most common of which is *Snort*, which also aggregates all the log files from major server applications, such as Apache and provides system-wide monitoring, response and administration. The use of

IDSs is, admittedly, complex for simple systems, but for critical systems, an IDS may be used to great effect in the prevention of intrusion or post-intrusion forensic analysis.

Gateways

While several critical MV facilities exist within the demonstration Myriad components, clearly, all types of computing devices, resources and operations are not presently incorporated into the framework through the development for full server components. This is not to say whether this is a desirable outcome; however, this is most certainly not feasible and in many cases, may be technically impossible.

Despite this being true, there are systems such as databases, which are highly beneficial to large-scale networked solutions. Building a complete server around an existing piece of software (such as a database), merely to add connectivity or MCS access is frequently excessive and time consuming to implement. However, because those systems are not normally built to act as core programmable network components, but as data resources and basic operators, bridging software may be written to expose them to a network for operation. The Myriad framework terms these pieces of software *'Gateways'*. Most often, these systems simply retransmit instructions or data to different protocols, or rewrite them slightly to provide a uniform interface.

To take the example of a database server, the easiest language to construct a *Gateway* from, is PHP (assuming that we are dealing with a networked database server running on a computer with a web server also operational). The remote system meaning to access the database simply requests a PHP document from the web server. The PHP passes on any arguments sent to it in the request to the database, and if data is returned by the database, this data is passed back to the remote system (see fig.42).

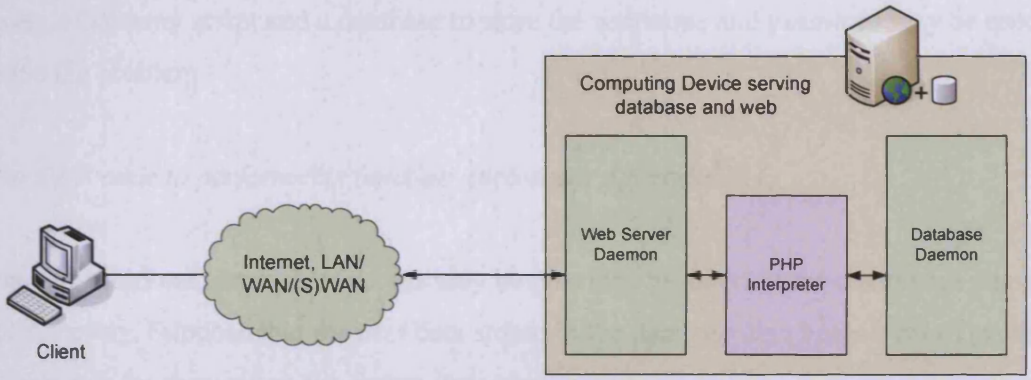


Figure 42. Diagram of the operation of a simple database access gateway script using PHP atop a standard web server (e.g. Apache) to interact with a database across a network

For the PHP code to operate this type of gateway, please see Appendix D.2.

Security Gateways

Astute system administrators will realise that a Gateway as described above will entirely expose the example database to the outside world, without any sort of access control. In certain circumstances this type of free access is useful, but in most it is potentially problematic and dangerous. Using a database, of course, the account used to access the database may have abilities granted to it solely to read data and not write or delete; this is a partial solution, but hardly complete, and useless in the cases of other less-developed software resources.

As a result of these failings, Gateways may be further developed to include access controls. Using pre-existing LDAP, database, system security and directory modules in Perl, PHP and other web scripting languages, facilities may be built to authenticate users.

In the case of the database server example, suppose that the database holds a table of users and passwords. Using extra 'user' and 'password' parameters to the PHP script, authentication details can be processed and accepted or rejected prior to command execution.

For PHP code to perform this function, please see Appendix D.3.

Taking the database example, logically, this type of Gateway may be used to control access to other Myriad components. Suppose that an Image Processing component needs to be secured;

using a Gateway script and a database to store the username and password may be used to solve the problem.

For PHP code to perform this function, please see Appendix D.4.

Finally, additional controls of access may be provided by filtering the commands passed to the Gateway. Suppose that the user data stored in the database also holds a list of prohibited commands for the user on the Image Processing server; operations they are not allowed to perform Using this list to inject values into a regular expression, command lines starting with *%0D%0A<command>* and followed by anything else may be removed and replaced by *%0D%0A*, a carriage return in UTF-8. In this way, any commands that the user tries to perform, which are not allowed, are automatically stripped from their request before it is executed.

For PHP code to perform this function, please see Appendix D.5.

Gateways & Firewalls

Since connections to the IP server component are created using TCP/IP sockets, it is possible to bypass a Gateway and connect directly to the server application, if you know the IP address server name and port number. Obviously this presents a problem for security, if the only means of forcing the user to authenticate through a Gateway is by not revealing the address of the actual server. The benefit of using TCP/IP and HTTP however, is that a range of security systems already exist to restrict access for systems such as web servers and personal computers; one of which, is the humble firewall.

Using a software firewall, the IP server can have all connections to it blocked, aside from those from the same computer (e.g. those produced by the Gateway). Using a hardware firewall, on a Local Area Network, stationed between the server and the incoming connection, all ports aside from those of the Gateway may be blocked, protecting the IP server from direct access by a remote user (see fig.43).

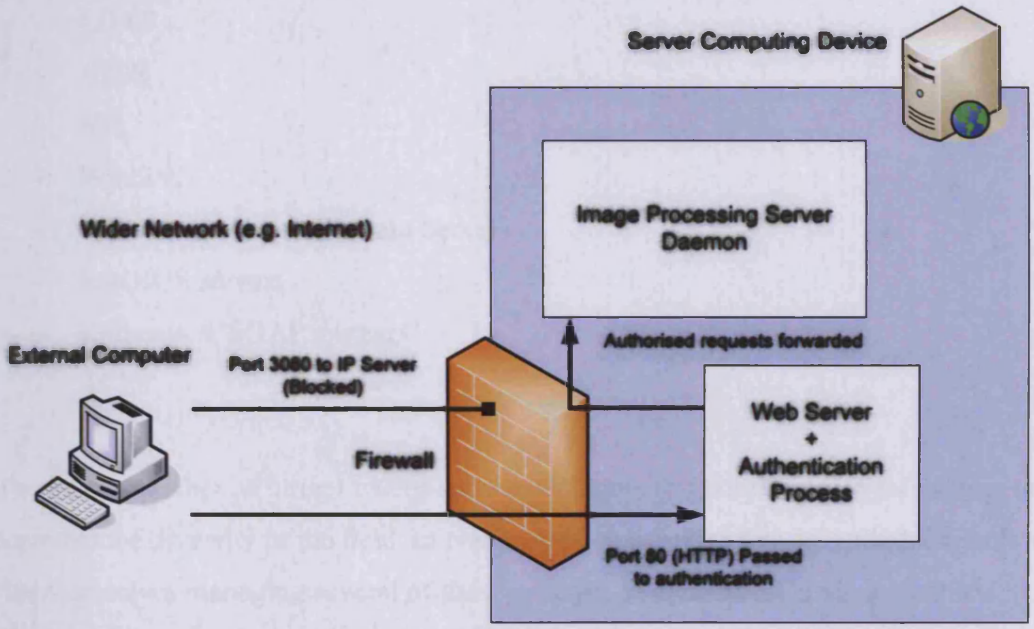


Figure 43. Using a firewall to control access to resources on a server on a port-wise basis. The firewall blocks direct access to the IPE, but a client may authenticate with a script on the web server, which can redirect input and output to the IPE

The ability to employ trusted network technologies such as firewalls to secure Myriad components ensures strong intrusion prevention and uniformity of maintenance, using security measures which administrators are familiar with, and integrating with existing security precautions which are presently maintained.

LDAP/NIS/NT Domains/ACLs

There are a variety of different types of access-control systems throughout networks and stand-alone computing systems. In the most essential examination, however, such systems involve a database of users, passwords and permissions, with an interface, able to administer access to that data, and define operations to be performed. Therefore, the most interesting element of the system is the manner in which the data is stored, and outlines of how to interact with the system as a whole, whether it be over a network, or directly. For convenience, these systems shall be named Authentication Data Sources (ADSs).

Notable examples of these systems include:

- Plaintext files
- Databases

- LDAP
- NDIS
- NIS
- Winbind
- Windows Network Domain Servers
- RADIUS servers
- Kerberos & SOAP systems
- ACLs

There are a number of further examples and divergent systems, however, this serves to represent the diversity of the field. In many cases, network and system administrators will find themselves managing several of these systems, in order to maintain a complete system or site. There is increasing evidence, however, that the use of Linux in enterprise environments and the improvements that are provided in terms of elements of those Linux systems being able to interact with multiple ADSs is allowing administrators to consolidate their user data systems and improve management with single sets of tools. In many cases the preferred ADS is LDAP, with tools providing transparent bridges to other authentication mechanisms, such as SAMBA, the Open Source re-implementation of the Windows SMB file and print sharing protocol, which is able to act as a Windows Domain Controller, authenticating as a Windows server on the client side, while storing data in LDAP form at the back end. This allows Windows workstations to authenticate using UNIX network user details, stored in one place and merely reinterpreted, depending on the system requiring access.

This chapter has described Myriad as a framework to the reader, beginning with an initial examination of how Myriad is designed at a high level, explaining the different types of component: Controller, Server, Data Source and Operator. The network hierarchy and the opportunity for the creation of non-hierarchical networks was explained.

Myriad components communicate using HTTP over TCP/IP in the present implementation, using web server functionality as the basis for intelligent Server components, providing a programmable logic component able to liaise between Controllers and non-programmable data resources and Operators.

6 Myriad Component Script

This chapter describes the need for a scripting language in networked vision servers, in order to provide functionality such as autonomous operation, local intelligence to reduce the impact of network latencies/failures, and to simplify the creation of client programs.

Myriad Component Script is introduced to the user, with details of syntax and sample construction.

Myriad controllable components implement a scripting language (Myriad Component Script (MCS)) to enable them to perform complex operations locally, in an autonomous or semi-autonomous manner.

6.1 Necessity of Scripting for Networked Systems

Myriad provides a scripting language for a number of key reasons:

- Producing scripting abilities in a client's user interface requires making that element of the system larger.
    - This places the requirement on the application developer, who frequently is less experienced than the server programmer.
    - This applies a profound limitation for the creation of exceptionally lightweight interfaces, which would have to produce additional code to handle complex operations.
    - Finally, user interface applications are typically more numerous than servers (evidence can be seen in comparing the sheer range of web-connected applications to the limited number of web servers) and there is less societal code duplication if the server provides such functionality.
- Most importantly, network connections over mediums such as the Internet are inherently unreliable. TCP/IP is designed with the assumption that there will be network failures, and that these should be routed around. The Internet is derived from ARPANet, a United States Department of Defence project to create a decentralised, fault recoverable network to connect nuclear installations.

  - Without local logic on server components, they are dependent for every operation on the receipt of a command from a controller, either locally or over a network. Local controllers pose no problems, but critical failures are created when network connectivity is lost for remote controllers.
  - Even when network connections do not fail, the latency between components and their controllers can vary and sometimes be very large. This prevents fast response to local conditions and consistent timing of operations, precluding the use of non-scripted networked Vision systems for repetitive operations such as stepper motor control, most especially in mission- and safety-critical environments.
- Without scriptability, systems cannot operate autonomously; they require a constant controller, even to perform repetitive operations. This limits systems to multi-threaded hardware, or an embedded system with an additional host controller networked to it, adding to deployment costs and maintenance overheads.

6.2 Language Concept

MCS is built primarily to be a grammatically simple, procedural, basic scripting language, with hints toward the syntax of BASIC. The intention is to provide a language which is able to service most rudimentary local processing needs, but not to enable development of large applications, user interfaces or servers. MCS is a way of programming fast response to local conditions and input, providing programmatical glue for a system.

While it would be possible to outfit a Myriad server with a full object-oriented, complex programming language, the amount of time taken to develop such a language, to design and develop the interpreter, to maintain and bugfix issues would be far beyond the means of the current development team (to take an example, Java has been continually developed by hundreds of programmers for more than a decade now). There is potential to integrate a scripting language such as Python into Myriad components, or Trolltech's QT Scripting for Applications (QSA), but this would be part of further progression towards a second specification.

6.3 Command List & Lexical Documentation of MCS

For a list of MCS commands, their descriptions and use, please see Appendix A.

6.4 Session Types

Myriad servers provide three methods of access: Single Command (SC), Interactive Session (IS) and Scripted Session (SS). These provide differing input options and state retention for the completion of large long-duration tasks.

Single Command

SC mode is the default for Myriad operation, and involves one or more commands attached to the end of a URL as the 'commands' variable to initiate operations. An example request would be as follows:

http://www.myserver.com/?commands=test

This would run the 'test' operation and then terminate. While this is adequate for one-shot commands, or short sequences ('Turn on lights'), the server destroys the handler that is interacting with the user once complete, effectively forgetting the conversation. In situations where it is necessary to perform a number of requests over a time period and retain data (eg. "Turn on lights 2 & 5, measure the light level", "If light level is below 50%, turn on lights 3 & 4 as well"), SC mode is inadequate.

Interactive Session

IS mode provides the retaining of state that SC mode is unable to do. To start an interactive session, send the command 'start', followed by the commands that are required, and then terminate the session with 'stop' or 'end'.

For example:

http://www.myserver.com/?commands=start
http://www.myserver.com/?commands=test
http://www.myserver.com/?commands=test2
http://www.myserver.com/?commands=stop

Between the commands 'start' and 'stop', the user handler is kept alive by the server, managing variables and resources, so that they are available for use upon receipt of the next command. Once 'stop' or 'end' is received, the handler is terminated.

This method of interacting with a Myriad server is likely the most useful in the majority of complex situations, but it does require larger amounts of resources for a longer period of time, since the handler remains active. In using a system capable of such operation, the administrator must carefully choose a good TTL (Time To Live) for idle handlers, based on the capabilities of the given server and expected load.

Scripted Session

Scripted sessions can be seen as being similar to SC mode interactions, in that they run a series of operations and then terminate. The difference, however, is that the commands are not provided to the server through the HTTP request, but rather, the server is directed to obtain input from a text file based on the local system, or remotely. The contents of that file are then executed as if they were appended to the URL supplied to the server.

Example:

http://www.myserver.com/?script=test.ips

This will open the local file, named test.ips and execute it. If the contents were:

Test.ips

Test

Test2

It would be the same as if the client had requested:

http://www.myserver.com/commands=test%0D%0Atest2

SS mode is principally used in combination with other scripting languages, which are able to change the contents of the text file on a regular basis, apart from the user connecting to the server. It is also used where the script is sufficiently long as to necessitate abbreviation and to ensure that the typographical errors do not occur. Finally, the use of a script file instead of commands allows for the hiding of script contents from the user; in scenarios where security is essential, this is highly advantageous; the user can execute the script, but not know what it

is doing, nor be able to see critical details, such as database passwords, processing algorithms or confidential data.

6.5 Inline Script

In addition to the session types previously described, it is possible to merge a script into a command series. This is done with the 'include' command and executes a script in position as if the user had input it themselves.

Example:

http://www.myserver.com/?commands=negate%0D%0Ainclude%20test.ips%0D%0Athr

Would ask the server to perform:

negate
include test.ips
thr

Which would incorporate the script test.ips to provide:

negate
test
test2
thr

6.6 String Interpolation

At the point of processing all strings in Myriad servers, strings are searched for variables and these are interpolated into the string prior to further execution of the command. In terms of string operations, interpolation is a process of evaluating and resolving variable names to their values. A good example would be:

$x = 5
Print "This is the $x th time I have done this."

Result: *This is the 5 th time I have done this.*

Clearly there are issues when variables are neighboured by other characters, such as using *$xth* to result *'5th'* in the example above. For this reason, there is a special syntax for forced isolation of variable names, using braces around the name. Hence:

$x = 5

Print "This is the ${x}th time I have done this."

Result: *This is the 5th time I have done this.*

There are also problems when we consider variables embedded within words, such as:

Print "This is the 2${x}th time I have done this."

As a result, MCS has two types of interpolation, 'Strong' and 'Weak'. In weak interpolation, variables are not evaluated when they are not preceded by a space. This was designed primarily as a transitional case during development, but is useful for partial evaluations:

$x = 5
Print "The value of '$x' is $x"

Result: *The value of '$x' is 5*

Strong interpolation actively searches for '$' symbols in all strings and interpolates them in place:

$x = 5
Print "This is the 2${x}th time I have done this."

Result: *This is the 25th time I have done this.*

In these cases, variables are evaluated through the normal equation/term evaluation routine, so they may contain other strings, equations which are resolved in place and the names of other variables, which are also interpolated in a recursive manner.

$x = 3
$y = "$x + 5"
$z = "The value of '$y' is $y"

Print "${z}, you see."

Result: *The value of '$y' is 8, you see.*

6.7 HTTP Formation

One of the key constituents of the MCS language is seemingly minor; the ability to create and execute HTTP requests. In reality however, this enables Myriad components to interact, share data, control each other, elicit the help of other HTTP-based services (such as web servers, web services, databases and Myriad resources wrapped in gateway code) and form complex networks. Most importantly, the capability to control other elements of a Myriad network frees the entire network from requiring a constant controller. While a GUI/Script controller might ask a number of components to perform a given task, unless they can interact with each other, they always rely on logic back at the controller to provide their next directive. Moreover, this forces the controller software to be much larger in terms of code designed to manage the components than it might otherwise be, and it also limits activity of the system, should the controller disconnect.

Using HTTP as a transport protocol, however, does pose a particular problem: HTTP is asynchronous. A client may connect to a server and request a data object, but once the conversation is complete and the connection is broken, there is no way for the server to contact the client again; nor can the server request further information to inform the data object transmission, short of declaring a lack of information and failing a request.

The advantage of Myriad modelling key components as web servers, however, is that in a conversation, a server is able to take on the role of a client in a two-way process (most commonly after a conversation has terminated).

Example: An equipment controller wants to be able to take advantage of an image processor, in order to perform regular checks as to whether a physical process it is controlling is failing (see fig.44).

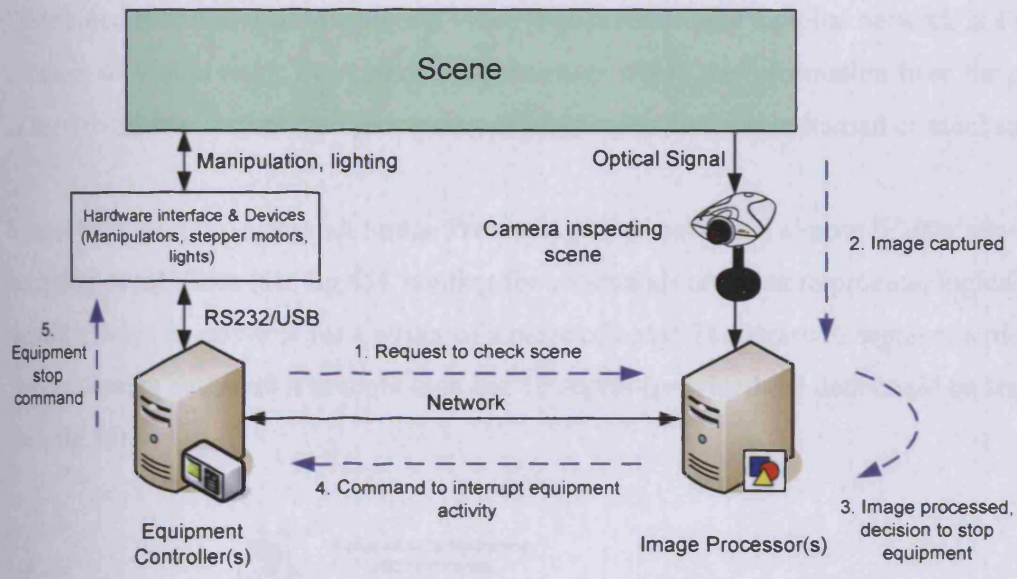


Figure 44. Using Myriad IPE and EC servers to build a system using visual feedback to control scene manipulation

While this situation is not always the most ideal, setting all components up as web servers does allow for good two-way communication, and alleviates the issues of asynchronous communications that the use of HTTP produces.

6.7.1 Complex Networks

HTTP request creation is primarily performed by simple creation of URLs as strings and then executing those using a generic HTTP get procedure, such as *wget* or in the case of Java, most conveniently, a web image download method. The majority of use for this feature is for simple control, requesting an action, rather than transmission of data; for these cases, data request is very easy, and the returned data does not need analysis. In the examples examined later, we will see a range of control GUIs, which are primarily concerned with operations, rather than data transfer.

Clusters

Using HTTP as a communications protocol with intelligent nodes makes the creation of complex networks very simple; a matter of string manipulation in rewriting URLs and writing new scripts, which can be passed to new nodes.

Distributed processing of images and video sequences around a global network is a tempting offshoot of Vision tasks, even more so in the cases where the information from the processing is used to initiate further data processing or physical action, either human or mechanical.

Since, by definition, a Myriad Image Processing component is a simple IP data server with scripting capabilities (see fig.45), waiting for commands and data to process, logically, it would always be possible for a writer of a piece of control software to segment a piece of data for processing and send it to more than one IP server (provided the data could be segmented) (see fig.46).

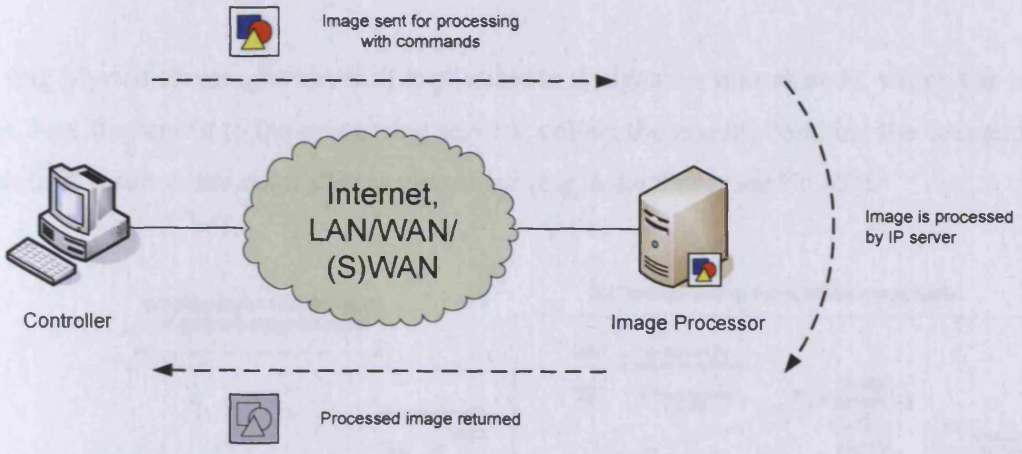


Figure 45. Utilising a single network-connected image processing server to convert an image to greyscale and return the result

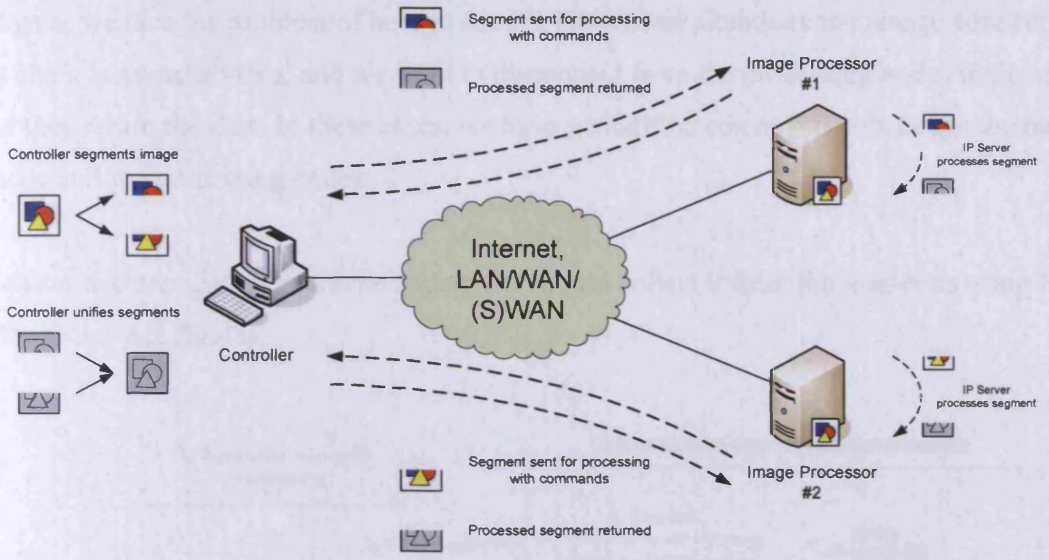


Figure 46. Utilising two networked-connected IPEs to process an image, with the client segmenting the image and recombining the resultant segments from each processing server

The difficulty with this system, however, is that it forces the controller to administer every single communication and deal with the data segmentation.

Using Myriad servers, however, it is possible to designate a master node, which can segment the data, transmit it to the processing servers, collect the results, combine the data and return the final result to the controller or elsewhere (e.g. a database (see fig.47)).

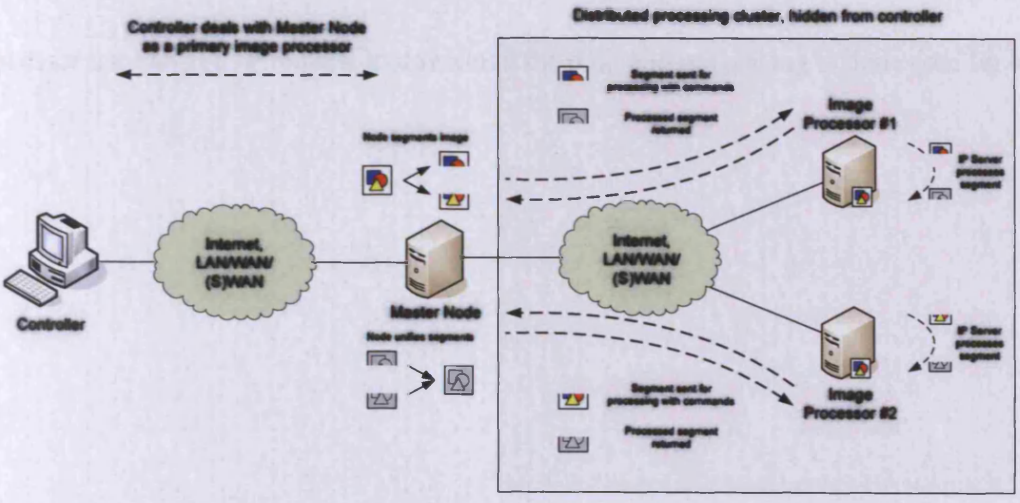


Figure 47. Utilising an intermediary Myriad server to act as a master node to a cluster of two slave nodes, appearing as one IPE to the client, across the network

Again, we face the problem of how to use HTTP in these situations to manage connections, when it is asynchronous, and we want to disconnect from the processing nodes until such time as they return the data. In these cases, we have a modified conversation between the master node and the processing nodes:

Master declares data is available and slaves should collect it from the master as using HTTP download (see fig.48).

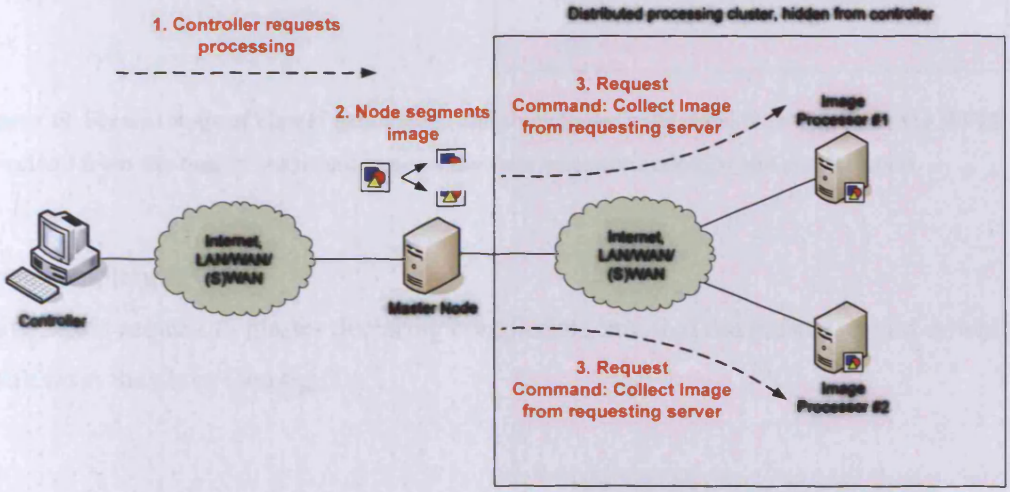


Figure 48. First operation of a cluster-processing task; the client requests an operation, the master node segments the data and commands each slave to retrieve the appropriate data segment

Processor uses an HTTP request to download the data and processing is done (see fig.49).

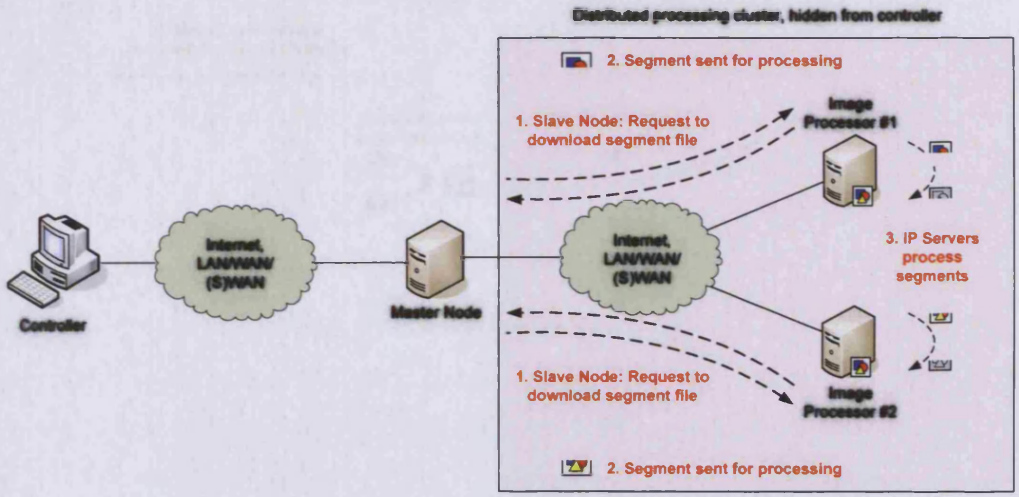


Figure 49. Second stage of cluster processing; the slave nodes collect the data segments via HTTP download from the master node and process the data as commanded by the master node

Upon completion:

Slave sends request to master declaring completion, and that the master should download the result from the slave (see fig.50).

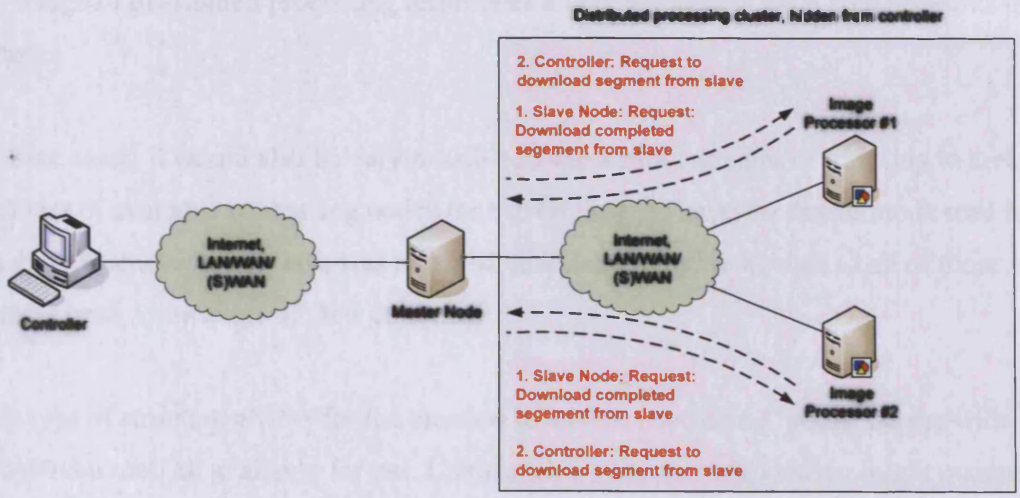


Figure 50. Third stage of cluster processing; slaves announce to the master when their processing is complete, and the master uses HTTP to request the data segments from the slave nodes

Master uses an HTTP request to download the data and reintegrates it (see fig.51).

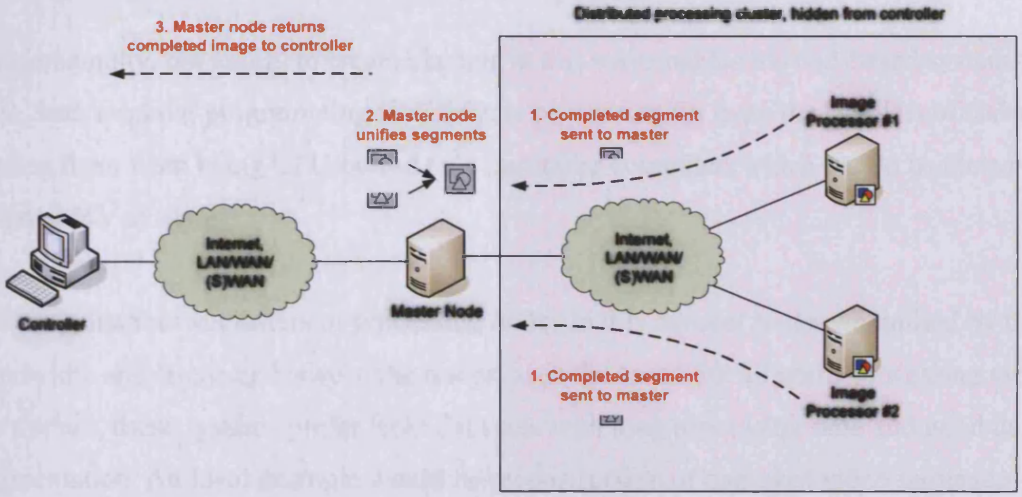


Figure 51. Final step of cluster processing; the master node downloads the data segments via HTTP from the slave nodes, integrates the resultant data and returns the completed image to the remote client as a response to the first HTTP request

Using the 'include' command to incorporate scripts to mask this sort of complexity, the actual programmatical requirements on a GUI are very low. Using simple string processing for control makes this process similarly lightweight, allowing any image processing task to take advantage of distributed processing techniques and power across wide area networks or the Internet.

In these cases, it would also be very possible to use a small amount of scripting to create a database of available processing nodes for a given task, to have the master node read that database whenever a new task was received, and then to distribute data to all of those nodes, without prior knowledge of their existence.

This type of structure allows for the creation of ad-hoc processing 'pools' across wide areas and the Internet, all available for use. Using such a system, a corporation might maintain a list of nodes on the corporate intranet server, which desktop systems announce themselves to when they boot up. Then, whenever a user from anywhere in the corporation's world-wide network required more processing power for an Image Processing task, they could send it to a master node, which would collect the list, segment the data and distribute it to the participants on the list. Such a system would require a few hours to program and take advantage of idle processing cycles on all the computing resources of the corporation, creating a very powerful tool for MV use within the organisation.

Fundamentally, the ability to create clusters in this way enables Myriad-based systems to be able, with minimal programming to distribute processing for even the smallest of tasks, freeing them from being CPU-bound on a particular computer, which would be the case with isolated MV systems.

Creating distributed clusters of processing nodes in this manner is always limited by the bandwidth and latencies between the nodes, as is the case with all multi-processing systems. As a result, these systems prefer large data sets with long processing time and good data segmentation. An ideal example would be reconstruction of damaged video sequences from old film or television, where frames must be processed uniquely and logic must be used to decide which IP operations are most suitable for each frame. In this situation, each processing node could collect a frame from the master, take several minutes or hours performing the appropriate operations, and then return the resulting frame, ready for reintegration. The more complex and lengthy the task, the less the network latencies affect the duration.

Meshes

Beyond the use of clusters of Myriad components to perform tasks, there is also the ability to create non-hierarchical networks of components, which manage themselves in operation, without the assistance of a controller to manage operations. This form of system is widely known as a 'mesh' network.

The use of MCS on Myriad components allows for logic in operations, conditions and response to data. The inclusion of HTTP creation in MCS allows components to communicate with each other and control in a short term manner, or engage in peer-to-peer associations to collectively solve a problem. With the addition of Prolog component servers for intelligent reasoning and classifiers/neural networks, it is possible to create Myriad solutions to problems which are capable of acting in an intelligent manner, without a permanent controller or human intervention.

The requirements for a mesh are as follows:

- The ability to control and be controlled
- Logic and reasoning at a node level
- The ability to transmit data between all nodes to inform logic/reasoning

Using MCS to manage such systems, most appropriately using script files which are executed/included, all that is required is an initial impulse to start a mesh processing, after which it may be left to its own devices. This may be from a script, executed by a user, periodically or through a web site, or a GUI, or simply from a programmable embedded system which is part of the mesh, which executes a script automatically once it is booted. Using these methods, it is possible to create autonomous meshes without intervention for tasks such as maintenance and repetitive mechanical tasks.

In this chapter, Myriad Component Script was described; explaining the necessity of a scripting language for networked machine vision systems, supporting local intelligence, operator-less automation, increasing performance and protecting against network failure. Scripting also provides the opportunity to incorporate interesting network topologies such as ad-hoc clusters and meshes, add macros and ease integration with other software (e.g. database-backed Point of Sale software, administrative monitoring and analysis packages, etc.)

---

7 Design & Implementation

This chapter discusses the detailed implementation of the Myriad prototype server components and gateway systems, starting with their threaded design, control input acceptance, processing, parsing and execution and the I/O abilities included in the MCS demonstrations.

Once the generic details have been established, task-specific features are examined, including the image processing and hardware control functions of the Image Processing Engine (IPE) and Equipment Control (EC) servers.

Lastly, the construction of the Prolog Gateway component is detailed, expressing how gateway systems wrap around conventional software to network their functions, and how these fit into the Myriad framework as a whole.

7.1 Server Design

Myriad servers follow the model of the majority of modern server applications, in that they have a small kernel, which manages the user connections and starts a handler for each user, but does little more (see fig.52). The user handlers are then much larger, complex pieces of programming, which interact with the users and perform tasks. This model poses issues, since handlers do not share the same memory space for much of their code, requiring far more memory than a shared-memory server, but conversely, handlers are also protected from each other and problems in any handler are only able to affect that handler and not others, nor the core of the server (see fig.53). This architecture, therefore, is far more stable than a shared-memory, large-kernel model (see fig.54) where errors can potentially damage the core and subsequently, affect all server users (see fig.55). Good examples of such architectures are the Apache web server and the VS FTP server.

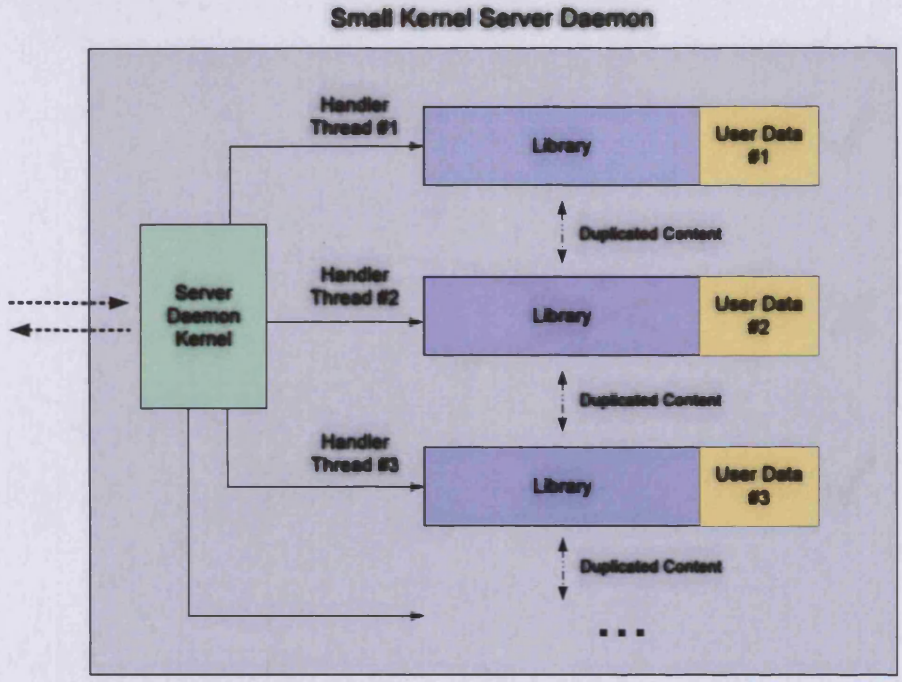


Figure 52. Diagram of server design convention, where a small core kernel manages user administration and connection, while spawning unique threads and large handler processes to interact with each client. Each handler is self-contained.

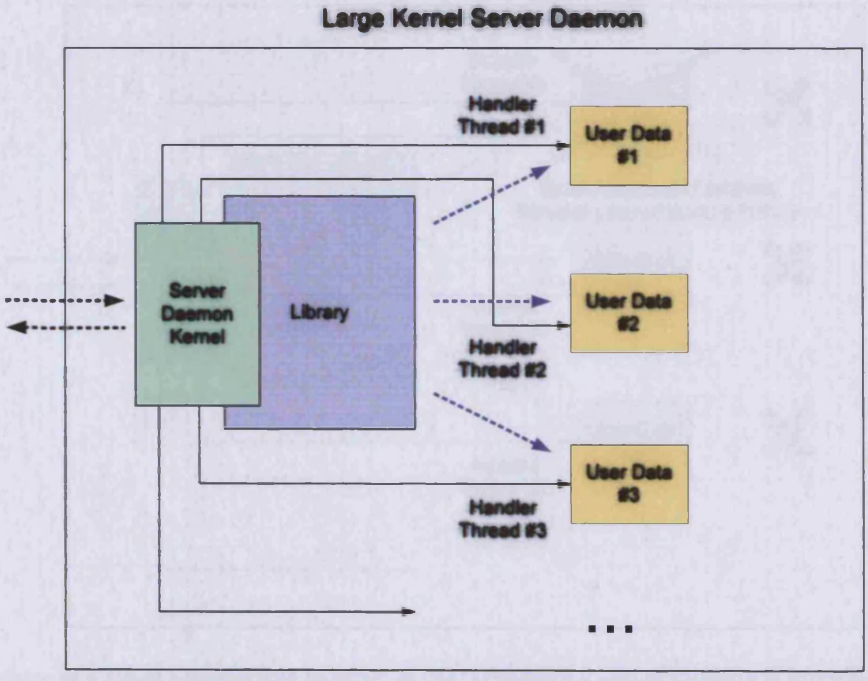


Figure 53. A large-kernel daemon provides a unique thread for each connecting client, but with a small handler process, which relies on a shared library of functionality

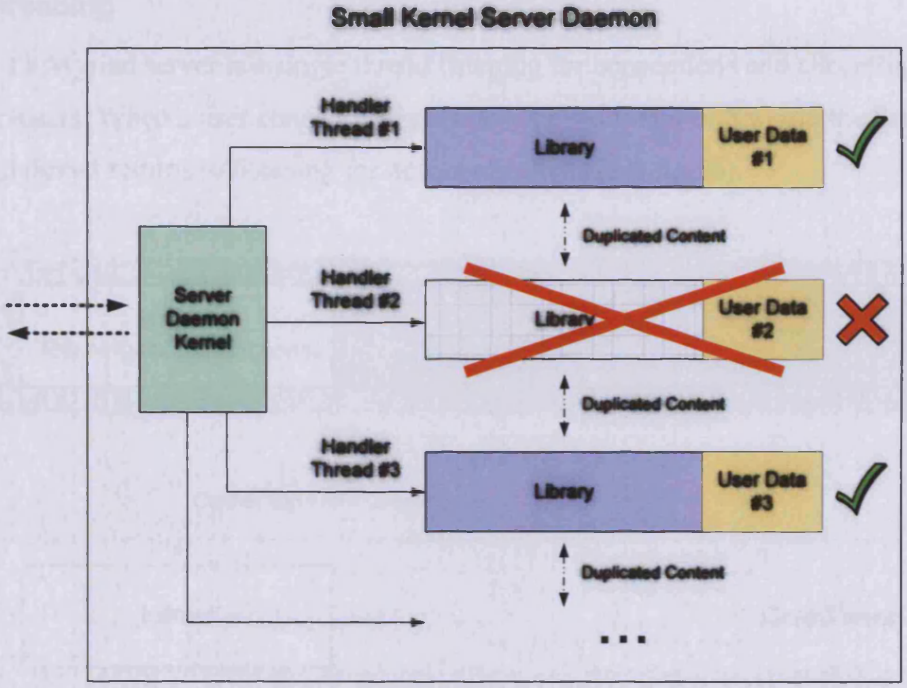


Figure 54. A small-kernel maintains separation of all client-handler code. If one client handler thread produces an error and fails, other clients are unaffected

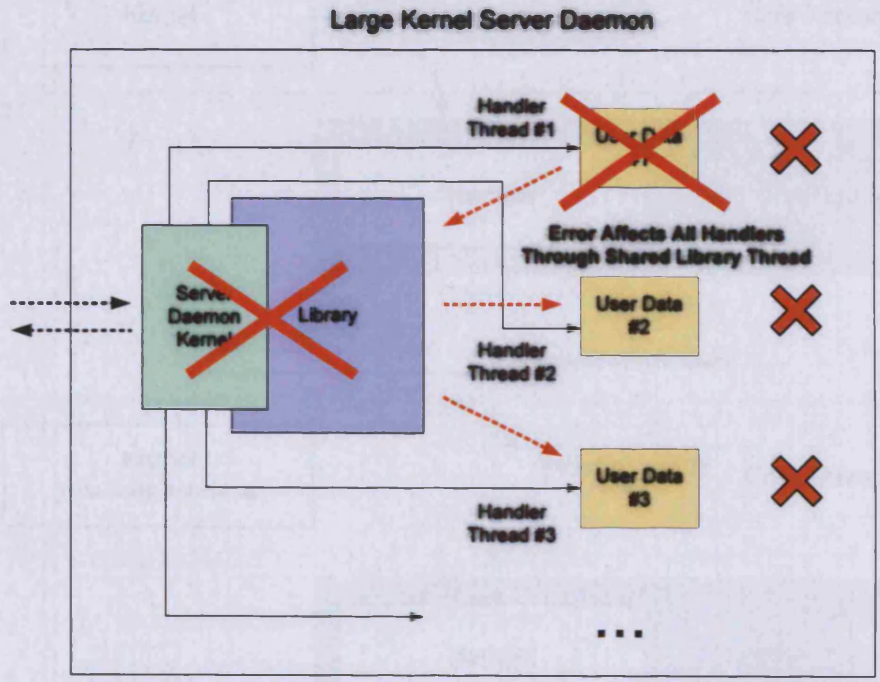


Figure 55. Failure in a client handler due to error in the large-kernel server daemon is more severe, as errors in the shared data/library space cause the error to harm all client handlers

7.1.1 Threading

The core of a Myriad server is a single thread listening for connections and allocating handlers to users. When a user connects, a secondary thread is spun off to run the handler and the original thread returns to listening for new connections (see fig.56).

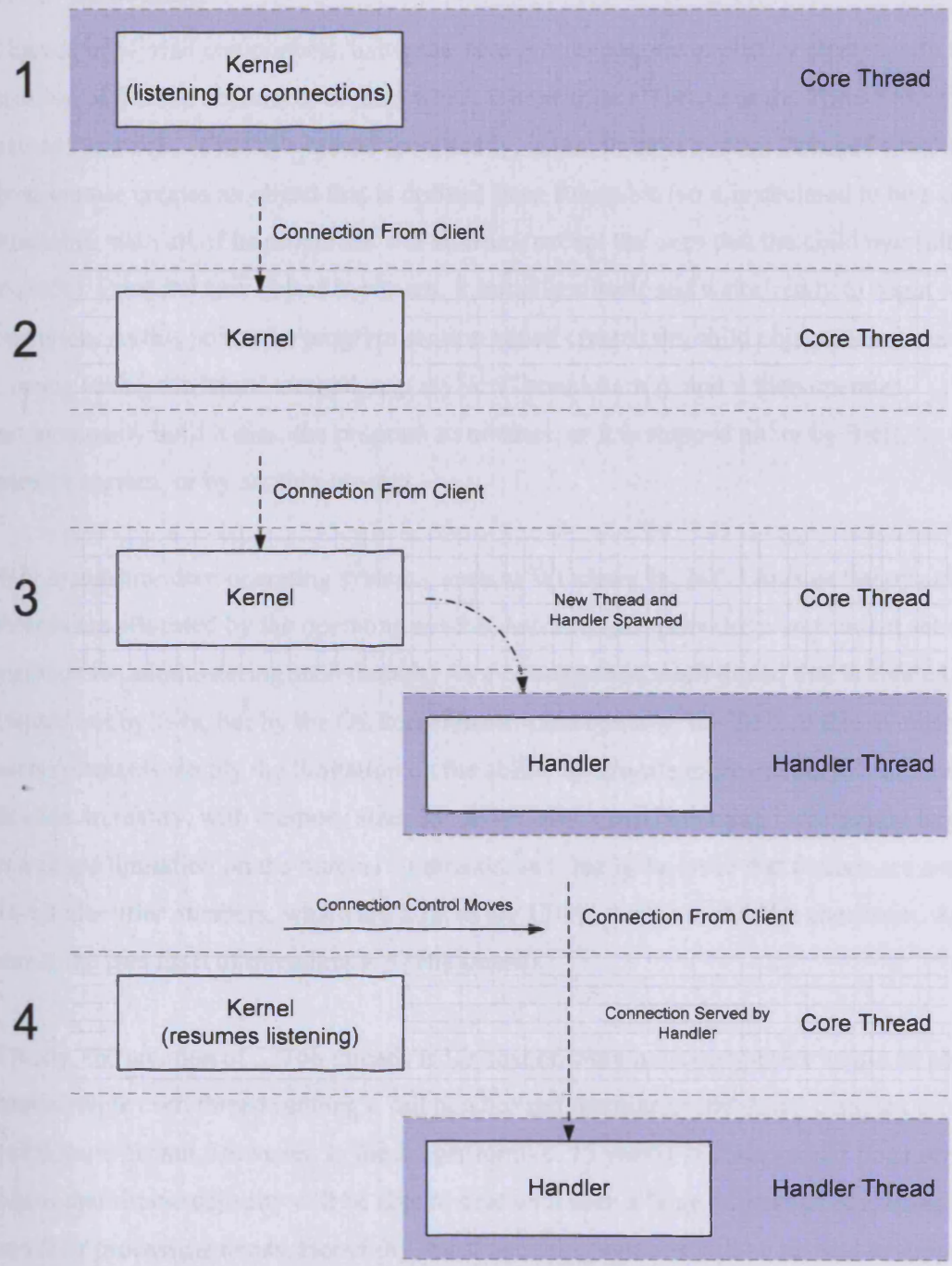


Figure 56. Diagram of how a multi-threaded server daemon accepts connections, allocates new threads, assigns connections to new threads and resumes listening for new client connections

When a user disconnects, the thread is marked as dead and when the next user connects, each thread is tested and dead threads destroyed before a new slot is allocated for the new user.

7.1.2 Allocation

Threads in Myriad components, using the Java prototypes, are explicitly created with the creation of Thread objects, or objects which inherit from a Thread or the Thread interface, named Runnable. What this means specifically, is that in most normal threaded situations, the programmer creates an object that is derived from Runnable (so it is declared to be a child of Runnable, with all of its properties and abilities, except the ones that the child overrides or extends). Once the new object is created, it initialises itself and waits, ready to begin operation. At this point, the program section which created the child object then starts it running (using the 'start' method, e.g. myNewThread.start()), and it then operates autonomously until it dies, the program terminates, or it is stopped either by itself, by the parent program, or by another process.

Within most modern operating systems, such as Windows 9x, NT, Linux or Solaris, all threads are allocated by the operating system, and Java just provides a convenient set of methods for administering such threads. As a consequence, each thread that is created, is created not by Java, but by the OS kernel itself. Conceptually, the limit of thread creation on such systems is simply the limitation on the ability to allocate more memory from more threads. In reality, with memory sizes for server computers becoming increasingly large, there is a single limitation on the number of threads, and that is the issue that threads are assigned 16-bit identifier numbers, which are effectively 15-bit, thanks to the 1bit checksum. As a result, the true limit of threading is 32768 threads.

Clearly, the creation of 32768 threads is far beyond what most computers would be able to handle, were each thread running a full handler and interpreter and doing complex processing at the same instant. However, in the longer term (c. 15 years), mid-range and large servers below mainframe capacity will be able to deal with such a large number of connected users and their processing needs. Hopefully, most operating systems will be revised to support 32-bit identifiers by that time.

Under the Myriad prototypes, we limit the number of concurrent threads to 25, this is an acknowledgement that while testing the prototypes on desktop-grade computing systems, Java is performance-limiting. Large numbers of threads in Java, while performing management tasks and control/image processing operations, are liable to completely consume the memory on the test systems, and as such, we choose to impose a small thread limit. This figure, however, is entirely arbitrary and may be altered with one line of code. It would, no doubt, be a wise decision to create a configuration file system for Myriad systems in production, allowing the administrator to alter the thread limit by changing a text file, for faster and more fine-grained control.

7.1.3 Maintenance

Each Myriad thread, once created, as an instance of the BasicWebServer2Starter class, is added to the array of starters, and is managed there in memory until such time as the user terminates the connection, or the thread dies. In these cases, once a new connection is discovered by the core thread, each thread in the array of starters is tested, all stopped threads (those with the 'stopped' flag set) are destroyed, as are all terminated threads. The new connection then has a BasicWebServer2Starter thread object created for it, and this is added to the array of starters in the first available position. In this way, we keep below the 25 thread limit, and also avoid giving threads identifiers which perpetually increase.

7.2 HTTP format

For the example URL:

http://localhost:3080/temp.jpg?commands=start%0D%0Aneg%0D%0Astop

A client web browser or HTTP client produces the following request text, which it sends to the web server, encoded in UTF-8 character format:

GET /temp.jpg?commands=start%0D%0Aneg%0D%0Astop HTTP/1.1
Connection: Keep-Alive
User-Agent: Mozilla/5.0 (compatible; Konqueror/3.1; Linux)
Referer: http://gemini.cs.cf.ac.uk/~dawnrider/interactive.php
Accept: */*
Accept-Encoding: x-gzip, x-deflate, gzip, deflate, identity

Accept-Charset: iso-8859-1, utf-8;q=0.5, *;q=0.5
Accept-Language: en
Host: localhost:3080

The first line is named the Object String (Request String in some languages), since it mentions the type of the request (GET or POST, marking how variables are sent to the server. GET attaches the variables to the end of the file being requested. POST sends them separately as an array similar to the UNIX environment variables) and the file being requested (/temp.jpg). The variables being sent are included as the part of the second term after the '?' character until the next space. The HTTP/1.1 denotes the HTTP specification that the client supports (values are 1.0 and 1.1; there is little distinction except that 1.1 connections have the option to be left open and returned to later by the client, rather than being completely asynchronous).

The User-Agent string is an identification of the web browser of client being used to request to the file. In the majority of cases this information is unused, but a number of web sites and servers have the option to customise the content returned to the user based on the web browser that they are using, taking advantage of the unique abilities of that client.

The Referer string shows which file was used to find the current file (through hyperlinking), typically used for tracking users and their paths through web sites and services.

The Accept line is complex and mostly unused, a feature of text-only web browsers, such as Lynx, which declares to the web server which types of file the client is willing or able to accept. Hence, a text-only web browser might have:

Accept: *.html, *.txt

showing that it only accepts HTML and plain text files, and not image types such as JPEG or GIF.

Accept-Encoding shows the types of data compression that the client accepts, for use with very large objects, such as data-filled web pages. To take advantage of this feature, web

servers must be configured specifically, a step only typically taken by servers with very large numbers of users.

Charset and Language strings show the preferred language of the web browser user, and the type of characters that the web browser supports for the encoding of non-ASCII characters. In this case, the web browser supports the Western set iso-8859-1 and UTF-8 the superset of all characters, allowing for support of Chinese sets, Arabic, Japanese, etc.

Finally, the host string shows the current server name and the port through which the client has connected, useful information for firewalls and port-selective services.

7.3 HTTP parsing

When an HTTP request is received, the key elements for a Myriad server, are those that define:

- The object being requested
- The query string including commands and other parameters

Objects and MIME types

The object being requested from the server, by the nature of a web server, is of prime importance, whether it be plain text, HTML or an image (Myriad servers presently transact GIF and JPG images). Due to the nature of the dynamic construction of data that can occur within a Myriad environment, however, it is highly possible that the requested resource may not exist at the time of the client request. As a result of this, while the file name is read, it is not additionally examined, MIME checked or data sent to the client until such time as the parameters sent using the query string are processed and commands executed.

Once execution of command sequence, or the examination of parameters is complete, however, data is actively sent, as the final act of the client handling thread, before it goes to sleep and awaits further connection or termination. At this point, there are two options, with respect to output to the browser; failure or data return. In the case of conventional data output, a file will be output through the socket connection to the client. First and foremost, the existence of the file in question must be established, prior to opening and sending. Using standard file system existence tests, the Myriad server is able to establish which response is

suitable. In the case of the file not existing at the point of sending, a '404 File Not Found' error is sent to the client, as required by the HTTP specification. Should the file be found, however, firstly, a '200 OK' success message is sent to the client, to make the client aware that the request has succeeded and data is due to be sent shortly. Once a '200' message has been sent, however, HTTP compliance requires the sending of a MIME-type string declaring the type of information which is to be sent, whether it be text, HTML, XML, a jpeg or gif image, or something else. This information is essential to web browsers and other HTTP clients, to allow them to prepare for the incoming data and realise how to view it correctly, failure to do this would result in confusion as the incoming data might be displayed incorrectly, for example, displaying the raw data of an image, rather than rendering it.

A good example of the MIME declaration failure issue is the Matlab Web Server previously mentioned. Due to the construction of the Matlab web server, output to the Matlab command history during execution (debugging messages, etc.) is also piped to the connected HTTP client and cannot be suppressed, where the aspiration is to limit output to only the output which includes HTML, data or image data with an appropriate Content Type declaration. If a content type declaration is not sent before output from the command history is sent, the client is forced to guess the MIME type and either defaults to text/plain (in the case of Internet Explorer), or attempts to discover the MIME type using the file extension of the file being requested. Unfortunately, the Matlab Web Server executable has the file extension .exe, indicating a Windows binary program, causing non-Internet Explorer web browsers to consider the output data, in fact, to be part of a binary file being downloaded, and thus, they prompt the user to save the data. Should the user choose to open this data file, once downloaded, however, they will find that their operating system (if Windows) will assume the extension indicates the MIME type and attempt to execute the text of HTML file, causing error messages. Using most modern alternative operating systems, such as Linux (+KDE/GNOME) and MacOSX, however, the start of the file will be examined to attempt to discover the data format, and at this point, a browser or a text editor will be invoked.

In order to resolve the MIME type of the object being sent, the file extension of the object is identified and compared to a list of known types, from which a Content-Type declaration is created and sent to the client. After this process, two carriage returns are also sent, to separate the content from the declaration, and data then follows.

Presently, Myriad prototype servers recognise a limited number of MIME types; 'text/html', 'image/jpeg', 'image/gif' and 'text/vnd.wap.wml'. WAP is the document type used for output to mobile phones and portable devices supporting a card and deck metaphor, with documents ending in the extension .wml. Myriad does not typically support text/plain as web browsers and clients assume HTML documents to be 'text/plain' with additional HTML tags embedded, and will revert to plain text viewing if formatting tags are not found. As a result, HTML documents and simple human-readable files such as comma-separated variable (CSV) data files are sent with the 'text/html' content type and left for the client to distinguish. Typical MIME differentiation code takes the form:

```
String extension = new String("jpg");
String extension2 = new String("wml");
String extension3 = new String("gif");
String fileToGetBackup = new String(fileToGet);
fileToGet = fileToGet.substring((fileToGet.length() - 3),
       fileToGet.length());
fileToGet = fileToGet.toLowerCase();
if(extension.equals(fileToGet))
{
      is_jpg = 1;
      outbound.writeBytes("Content-type: image/jpeg\r\n");
}
else if(extension2.equals(fileToGet))
{
   .  outbound.writeBytes("Content-type: text/vnd.wap.wml\r\n");
}
else if(extension3.equals(fileToGet))
{
      outbound.writeBytes("Content-type: image/gif\r\n");
}
else
{
      outbound.writeBytes("Content-type: text/html\r\n");
}
```

Query String and Parameter Extraction

Along with the object to be returned to the client, there is a second essential component to the use of Myriad servers through HTTP requests, and that is the Query String. While this section is not typically delineated from the return object for the client, web servers must separate the parameters from the object to begin processing and provide the revised dynamic object at the completion of the transaction.

The format of the query string itself, is a string of keys and values, associated by a equals sign:

X=5

Key-value pairs in the query string are then concatenated together with ampersands and prefixed as a whole by a question mark:

?x=5&y=3

As a result of this, the processing of a query string is a removal of the first character, followed by the separation of the string based upon ampersands, and the subsequent separation of the keys and values using the equals sign for each key-value pair (see fig.57).

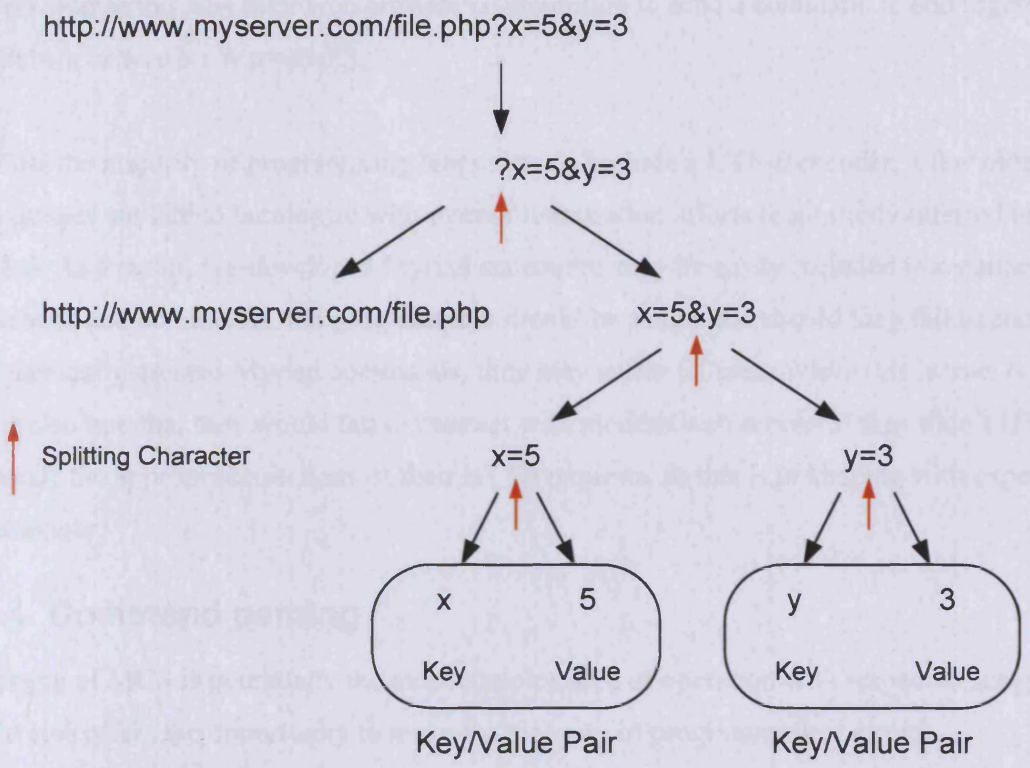


Figure 57. Graph of the string-splitting steps involved in separating the keys and values from HTTP URL variables

Because of the complexities of using this form with a range of global languages, there is a danger that there will be a collision between the separators and their use in the parameters themselves (suppose that the client wanted to transmit a parameter that actually involved an ampersand in the middle of a piece of data; the server would consider the key-value pair to have ended and try to evaluate a new pair, causing an error. As a result, and in order to include non-English characters beyond the ASCII character set, URLs and their request strings are encoded in UTF-8, a 8-byte representation of each character extending the potential characters far beyond the total of all languages presently in use. Query strings, are, therefore, encoded into UTF-8 prior to concatenation and transmission, at which point, colliding characters, such as *'&'* are converted into multiple hex pairs, prefixed by a percentage symbol (*%*). These cases are essential to smooth communication and decoding of the query string, but due to the nature of the character spaces, some characters are commonly converted to readable alternatives that may confuse the user, should they try and read them. For example, and encoded space may either be represented as a *'+'* or *'%20'* (*'+'* is the non-UTF version, although most web servers and clients still support this and prefer it), leading to

users wondering why their web browser is attempting to send a command to add together two numbers or words (*'test=4+5'*).

While the majority of programming languages do include a UTF-8 encoder, a few older languages are late to harmonise with internationalisation efforts (commonly referred to as i18n). As a result, pre-developed Myriad statements may be easily included into connection methods and commands, but programmers should be aware that should they fail to encode dynamically-created Myriad commands, they may suffer failures. While this is true, however, it is also true that they would fail to transact with modern web servers if they didn't UTF-8 encode the appropriate sections of their HTTP requests, so this is in keeping with expected behaviour.

7.4 Command parsing

Parsing of MCS is potentially the most complex area of operation with respect to scripting, and also offers the opportunity to make the majority of programmatical errors.

Most central to any interpretation system is a switching system, where a command is discovered and then a switch deduces what the command is, and executes a function based on that command, supplying additional information as parameters to the function. Nominally, MCS defines the first word of a command string to be the command, and subsequent words are parameters.

Command Sequence:

<command 1> <parameter 1.1> <parameter 1.2>
<command 2> <parameter 2.1> <parameter 2.2>

Exceptions are made from this structure only in the case of assignment, since assigning a value to a variable is a fundamental programming task, and one that has well-established syntax that is used in most other languages. As a result, while the most appropriate and in-keeping with the existing language format would be:

assign $x 5

Programmers are more comfortable with the use of direct assignment:

$x = 5

As a result, MCS makes a syntax exception for assignment, for the convenience of programmers, for the same reason that strongly-typed languages such as Java are migrating toward a more loosely-typed structure; the added burden of assignment using a syntax non-typical of contemporary languages may lead to instances of assignment using the contemporary format, which are flagged as errors or disregarded by the interpreter.

It should be noted, of course, that the BASIC standard did, indeed, define a command for assignment, *'LET'*, which was then followed by a modern assignment equation. Visual Basic and other modern BASIC derivatives have removed the *'LET'* and done direct assignment, however.

7.5 Pre-parsing & variable allocation

Variables in Myriad servers using Java are presently implemented as a *'vector'* object; a dynamic array that has a given size, but resizes as new elements are added. In this case, they are storing *'string'* objects (not standard null-terminated C arrays, but complex objects which present a string front-end with additional methods and attributes).

Before mentioning the operation of variable allocation, it is important to note that Java *'vector'*s are not very efficient, in that the programmer sets an upper limit and an incremental value; when the *'vector'* fills and another storage is requested, a duplicate array of a larger size (*current size + increment value*) is created in memory and the elements copied across, after which, the original array is destroyed. Not only does this cause large performance deficiency when the copying operation takes place, but the creation of the duplicate vector and the creation of each additional duplicate *'string'* object within the new object is expensive, due to Java's poor object creation profile.

It is also worth noting that this method of allocation logically performs poorly for extremely large arrays (*array > (½ available memory – OS used memory)*), because it actually involves having two arrays of equivalent size in memory at any time, potentially exceeding available memory. Finally, it forces the operating system to allocate and manage the release of large

sections of memory every time the *'vector'* is rebuilt, and not just directly, but through the Java Virtual Machine first (which has to perform OS functions); this alone produces a performance latency, but this can be exacerbated by operating system kernels with poor memory management. This situation could be avoided in C, C++ and most other conventional programming languages, through the use of pointers to produce linked lists and complex data structures manually, but unfortunately, Java is specifically designed to exclude pointers from the users' repertoire.

In building a command interpreter, there must be some form of variable allocation or definition that takes place, even in a non-declarative, dynamically-typed language. In essence, there are two choices for variable creation:

- Runtime Allocation (RA): Command sequences are run, and when a variable is discovered, a test is performed, to see if the variable exists, and if not, a creation is performed, and then the operation resumes. This is the most robust kind of allocation, since it allows the user to perform unusual tasks, such as dynamically creating variables using names derived from user (or other) input at runtime. This also only allocates memory for variables when they are actually used.
- Pre-Parsing (PP): When command sequences are originally loaded into memory, prior to execution, each line is examined, and all terms of the forms *$name* and *${name}* (our protected case for strong interpolation) are discovered and empty variables created in the variable vector, in preparation for use.

Both of these methods have merits, most essentially the flexibility of the RA method for user operation, should it ever be needed, against the essential simplicity of the PP method, where a simple string operation may perform the allocation prior to processing.

As is the case with many such situations, the distinctions between the two are more pronounced when we examine their problems. First and foremost, PP has intrinsic memory problems, as it pre-allocates all variable memory (see fig.58), potentially long before use, where RA does not (see fig.59). On small memory systems and embedded systems, this might be a significant issue, as it removed memory from the heap before it is specifically needed. As a result, another program might find itself starved of a section of memory in the meantime, provided of course, that it would return the memory, prior to the allocation to the Myriad component.

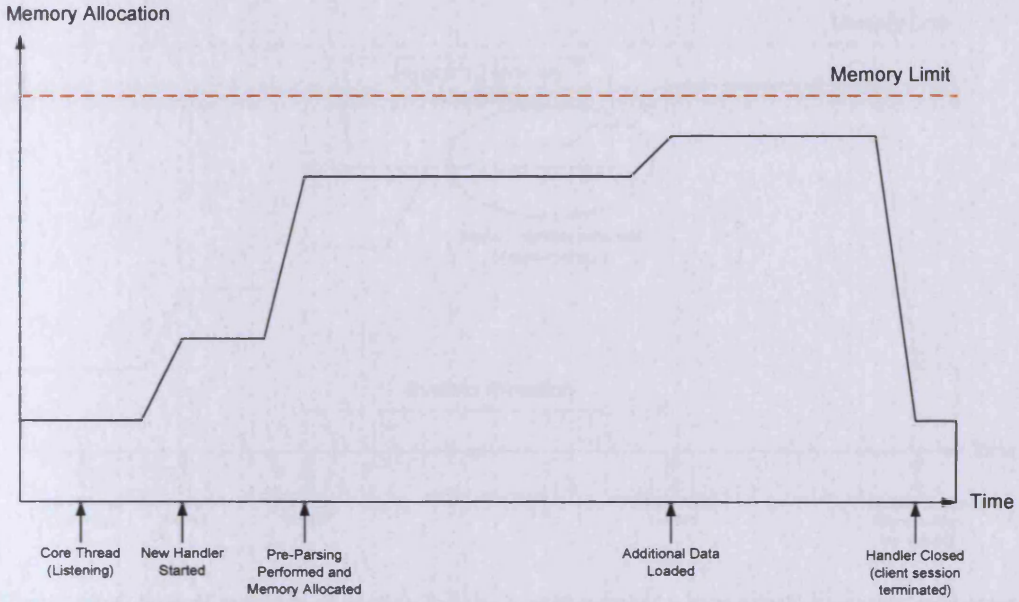


Figure 58. Graph of memory allocation against time for a pre-parsing mechanism for interpreter memory utilisation

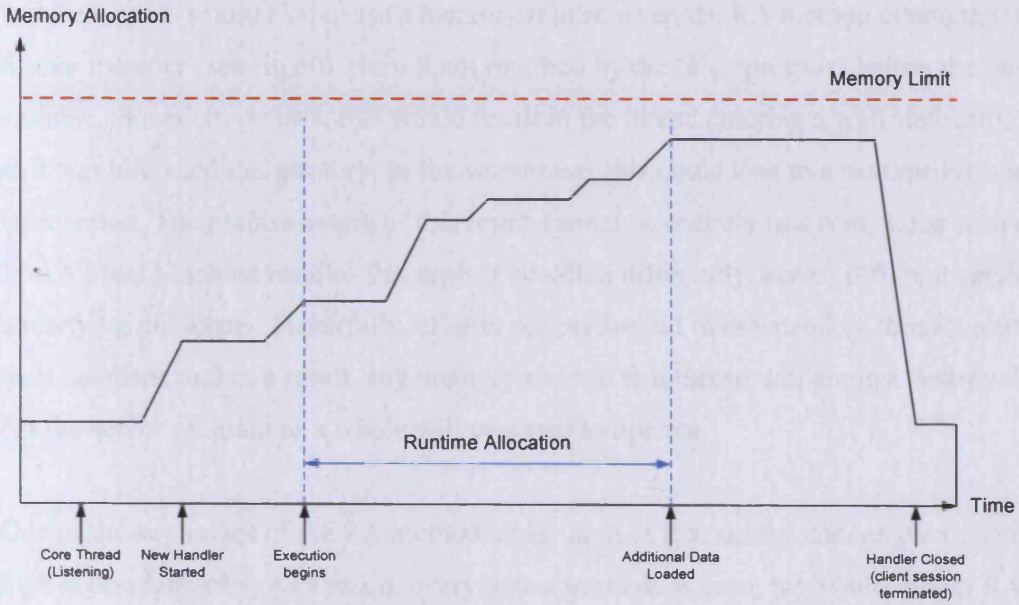


Figure 59. Graph of memory allocation against time for a runtime allocation system of interpreter memory utilisation

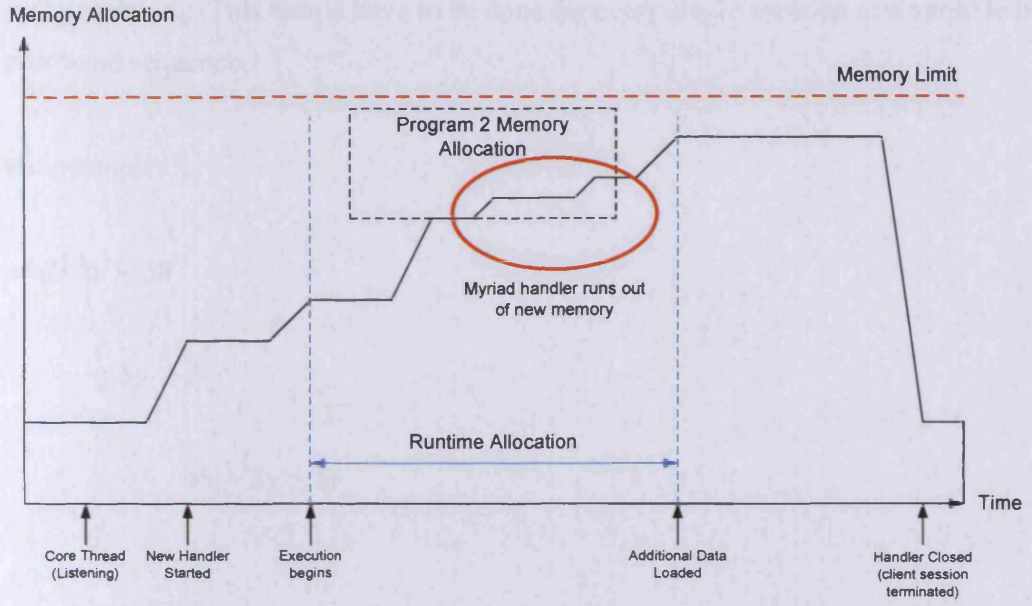


Figure 60. Graph of memory allocation failure in mid-process where another process consumes necessary memory before a Myriad RA-based interpreter can request it

If the server implemented an RA method, then this memory would be free until absolutely required, but it would also cause a memory failure when the RA method attempted to request further memory (see fig.60), were it not returned by the other program before the point of the variable allocation. At best, this would result in the thread entering a wait state until such time as it was allocated the memory. In the worst case, this could lead to a runtime error and termination. The precise nature of this result cannot be entirely resolved, since each different Java Virtual Machine handles this type of situation differently, across different versions and underlying platforms. Thankfully, all user scripts are run in independent threads provided to their handlers, and as a result, any memory error in that thread will at most destroy the thread, but the server program as a whole will continue to operate.

One of the key issues of the RA method under Java, is that strings are complex objects, with high access latencies. As a result, every time a variable is used, we would (under RA) test for existence of the variable, which would incur a performance penalty. While this might not be a significant problem in general operation, areas of scripting with high amounts of variable access, most especially tight loops, will generate a large number of existence checks in addition to their regular operations. The situation will worsen as the number of variable increases, as existence tests will have to do an exhaustive test the vector key elements, using

string matching. This would have to be done for every single mention of a variable in the command sequence.

For example:

```
while $x < 50
{
        if $y > 2
        {
                $x = $x + $y
        }
}
```

In this example, existence tests are performed five times, during this simple loop setting. This requires string-comparison searching of all variable names in the variables vector for each test, and repeated for each iteration of the loop until complete. Any other use of variables within the script will also incur this performance penalty, as the program cannot be certain as to whether a variable exists, except by testing.

Under Java, this design has an additional performance limitation, in that not only does it test for existence, but also performs string creation operations while the code is running, introducing a pause to running that is likely to be undesirable in time-sensitive tasks, such as machine operation. Not only this, but in order to produce RA methods in Java, additional 'getters' and 'setters' must be written to override regular *'vector'* element access, to do the existence test and create the non-existent variable as required. This process introduces yet another latency to all variable access and makes the modification of the vector code by less experienced programmers more problematic. Finally, due to the use of the *'vector'* as a data structure, if a new variable is created at runtime which forces the *'vector'* to resize itself, the resizing operation, as previous mentioned, will be very memory and processing-intense, damaging performance significantly and potentially introducing large pauses in operation. Moreover, because of the nature of the RA method, the program may not easily estimate the size of the variables vector before execution, preventing the setting of a reasonable starting size for the vector to minimise rebuilding. In the case of PP, for example, we can extend the method to include two parsing stages, an additional one to the normal, which simply counts

the number of unique variables (which admittedly will increase the time taken to complete the PP stage in their own right, but will be less than repeatedly rebuilding the vector for large numbers of variables). The benefits of such a method are quite ambiguous, but a good heuristic method for guessing the initial size of the vector and the increment could actively accelerate the pre-processing and minimise memory use.

Eventually, however, this discussion is focused on Java as a language and the shortcomings thereof. Contemporary programming languages, such as C++ and C will still appreciate a PP method, rather than the unpredictability of the RA, but the use of pointers, linked list and the ability to take advantage of more efficient memory management and performance functions will remove the majority of the assignment penalty, albeit not all of it.

7.6 Command Execution & Arithmetic Processing

For details of the Myriad prototype command loop and how arithmetic processing is performed by the prototype servers, please see Appendix C.8 and C.9, respectively.

7.7 I/O

Input and output are most essential mechanisms of MCS as a language and provide support for interaction with both the client and other systems.

Clearly, the first methods for I/O are the acceptance and response to HTTP requests that allow a Myriad server to be controlled and operated. As a result of this, we understand that standard input may accept textual data and output may be images, text, or other binary and ASCII-type forms.

Standard input:

- Text

Standard output:

- Text
- HTML
- Image data
- Other local binary/ASCII files

The limitation of output, however, is that demonstration servers prefer to transmit whole files from the server computing device, rather than responding directly to the stream. This is done for neatness in the prototype source code, and as it covers the majority of requirements, there is little need for a prototype to add duplicating functionality. In practical use, however, implemented servers should stream output directly, rather than saving files. Direct streaming would be preferable for speed, resource consumption and to prevent file system namespace collisions (e.g. multiple clients deciding to save an image named test.jpg into the same shared directory at the same time). This is also presently not implemented to permit future exploration of file-space sharing for collaboration and long-term archive and retrieval operations.

In summation, the present Myriad must save files and then stream these to the client, rather than presenting data directly.

7.7.1 File

The majority of file activity in prototype Myriad systems has been output, primarily for testing and logging, a usage pattern that is reflected in database I/O (as discussed later). For these tasks, MCS includes basic file interaction facilities (with the assumption that active use will lead to opportunities and requirements to extend this command set further). Current commands are:

- openfile <handle> <filename>
- closefile <handle>
- printfile <handle> <data>
- printlnfile <handle> <data>
- readfile <handle> <variable>
- readlnfile <handle> <variable>
- eoffile <handle> <variable> (1 = end, 0 = not end)

The use of file handles (which are user-defined strings) allows for several files to be opened and maintained at the same time, most useful for multi-file monitoring, logging and data integration from multiple sources.

e.g. *openfile alpha test.txt*
printfile alpha "$test is correct"

closefile alpha

In these cases, *'alpha'* is the file handle and identifies the file being operated upon. If files *'alpha'*, *'beta'* and *'gamma'* were active, printfile could be directed to the appropriate file for output, with specificity.

Printing to files may be through either of two mechanisms: 'printfile' and 'printlnfile'. The latter of these commands provides output with an additional carriage return character at the end of the data, providing a faster means of writing lines of information into a file. Data may be interpolated by surrounding it with quotation marks, or non-interpolated through the use of apostrophes as delimiters, or omitting delimiters altogether (in which case, the interpreter assumes apostrophes).

Reading from files operates in a similar manner to database querying in many ways, through the use of three commands: *'readfile'*, *'readlnfile'* and *'eoffile'*. The first of these is capable of reading the entire contents of a file into a variable for the user to decode or manipulate. The second reads a single line of the file into a variable, requiring that the user have some way of realising that the end of the file has been reached, and that no more data may be read. For this purpose, the 'eoffile' command tests for the end of the file, and reports the result to a variable (this mimics the operation of the *'dbfurtherresults'* command within database interaction); enabling the user to find out whether the file is exhausted before attempting to read a further line.

7.7.2 HTTP

The ability to form HTTP requests is as previously mentioned, core to MCS operation and the creation of complex Myriad networks. The design of most modern programming languages enables simple implementation of HTTP downloading functionality into MCS interpreters.

The MCS syntax for downloading a web document is:

geturl $url $variable

Where *$url* is the name of a variable containing the URL of the document to get, and *$variable* is the name of a variable to store the resultant data.

Because URL requests for Myriad components require the attachment of commands to the end of a URL through concatenation, *'geturl'* has only the capacity to address a URL in a variable, not by direct use of a URL in the command line.

Hence, the following does not work:

```
geturl http://www.myserver.com/test.html $return
```

But a simple assignment prior to the command has the same effect:

```
$myurl := 'http://www.myserver.com/test.html'
geturl $myurl $return
```

While this is mildly inconvenient, the creation of an interpolation mechanism and multiple parsing methods has not been undertaken at this point, since the 'geturl' command has only been required for complex commands and variables during testing.

It should be noted that the second parameter (that of the return variable) may be omitted in common tasks that require only that a request be made, rather than a result being essential (for example, asking a Myriad equipment controller to turn off lights, a task where returned data may not be essential, except for confirmation of the operation).

The use of *'geturl'* does, however, allow for the creation of complex Myriad control mechanisms, using concatenated strings:

```
$x := "http://www.myserver.com:3080/test.jpg"
$x .= "?commands=start\nnegate"
if $y > 2
        $x .= "\nthr"
$x .= "stop"
geturl $x $return
```

Either the newline character (*\n*) or the UTF-8 transformation (*%0D%0A*) may be used as delimiters, as a key feature of the 'geturl' implementation is that the variable is passed through the Java URL Encoder object prior to the request being sent, ensuring that output is UTF-8-compliant. Similar facilities may be found in other programming languages and toolkits, such as QT, which provides the functionality implicitly in the *QURL* class.

7.7.3 DB

While databases may be dealt with through gateways, the prototype Myriad servers include a set of database commands, due to the need for testing of database functionality prior to the completion of the gateway systems.

Java implements database support through complex interactions of complex objects (custom driver objects for each type or database and additional objects for query strings, connections and result rows) which are unfortunately highly non-portable to other languages for re-implementation. As a result, caution is urged in embracing database access in the manners introduced in the prototype source code; the use of database gateways is still more expedient for most simple tasks.

Database use is of two types:

- Reading
- Writing (insertion, modification, updates, deletion, construction)

Preceding both of these activities, connection must be made to the DBMS server daemon. To do this, the Myriad prototypes provide the 'dbconnect' command; a wrapper around the JDBC (Java Database Connectivity) mechanism, it accepts a specially formed database URL for connection. E.g.

dbconnect <server>/<database>?user=<user>&pass=<pass>

Where <server> is the machine name for the device running the database daemon (e.g. localhost, www.server.com, etc.), <database> is the name of the table group provided by that daemon, <user> and <pass> are the username and password respectively.

Correspondingly, there is a *'dbclose'* command, which terminates the current database connection. At present, only one connection may be active at a time, for each connected client; however, as it is frowned upon by system administrators to leave large numbers of database connections open for extended periods of time, this is unlikely to pose a difficulty for users.

Both reading and writing to a database are handled by the same mechanism; a command named *'dbquery'* (*'dbread'* exists as a useful alias for convenience and a more intuitive name for the poor of memory). In a similar manner to the database gateway, *'dbquery'* simply passes through SQL queries to the database. Since both reading and altering a database in SQL essentially involve the execution of a query string, this section is the same.

A typical use of *'dbquery'* would be:

```
dbquery "insert into test values(1,5);"
```

In keeping with other output methods from Myriad prototype servers, variables placed within the string are interpolated, allowing for the dynamic and responsive use of databases.

Subsequent to the use of *'dbquery'*, reading and writing operations diverge in their application. If a query was intended to obtain information, rows of data will be returned. The dbresult command will place the output of a query into a variable as a comma-delimited string. Each successive call to *'dbresult'* will produce a further row, until the result rows are exhausted. Calling the *'dbfurtherresults'* command will show whether there are more results to collect.

```
dbfurtherresults $z
if $z > 0
{
        dbresults $x
}

$x = "test1, test2, test3"
```

7.8 Image Processing

Beyond the basic functionality of Myriad servers, each server type includes a subset of instructions to provide a purpose other than as a generic data routing device (see fig.61).

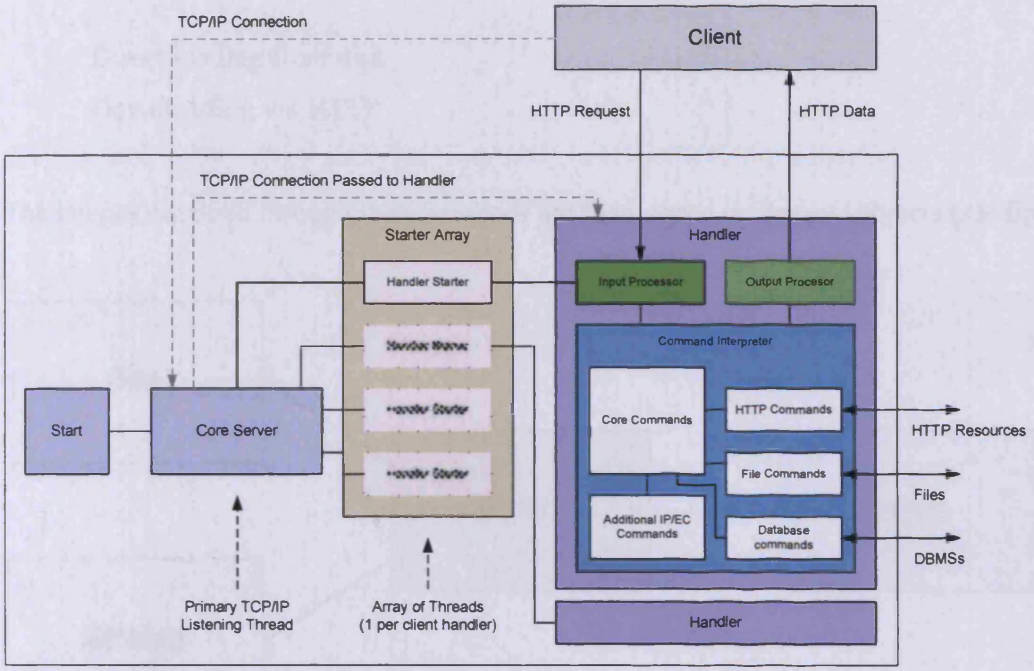


Figure 61. Outline of the structure of demonstration Myriad server daemons written in Java, using a multi-threaded construction

Image Processing functions in Myriad that form the server prototype named Myriad::IP are a combination of functionality from the Java Image class (loading, saving, addressing and compositing) with customised commands (channel RGBA/HIS isolation, advanced compositing) and algorithms developed in CIP and its predecessors.

7.8.1 Basics of Image Processing

For an introduction to image processing techniques and common operations, please see *Appendix B : Basics of Image Processing.*

7.8.2 Image I/O

Image input and output is the foundation of any image processing system. Of all the sections of IP development, therefore, this is the most essential in terms of diversity affecting system

use and potential. Without the capability to obtain an image or store a resultant picture, no Image Processing can be performed. Due to the nature of the task, I/O functionality is utilised from the Java native classes. Images in Java may be acquired in two ways (that are relevant to Myriad's purposes).

- Direct loading from disk
- Downloading via HTTP

The images obtained through these methods are then stored in *'Image'* objects (see fig.62).

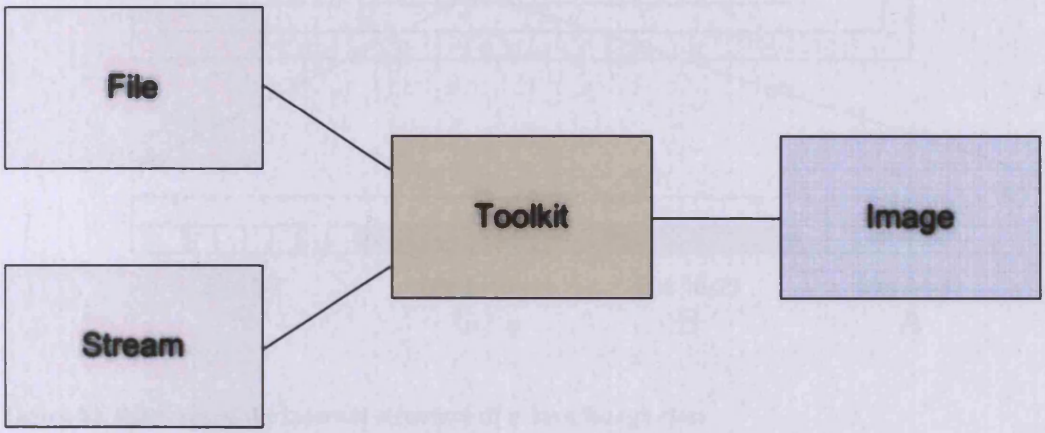


Figure 62. Simple outline of the high-level design of Java image loading through a toolkit class

Java's *'Image'* objects are wrappers around 1D arrays of 32bit packed-pixel elements; (that is, sets of 32bits composed of 4x8bit elements, 8 bits per channel (R,G,B and Alpha), values ranging from 0 to 255, unsigned). See fig.63.

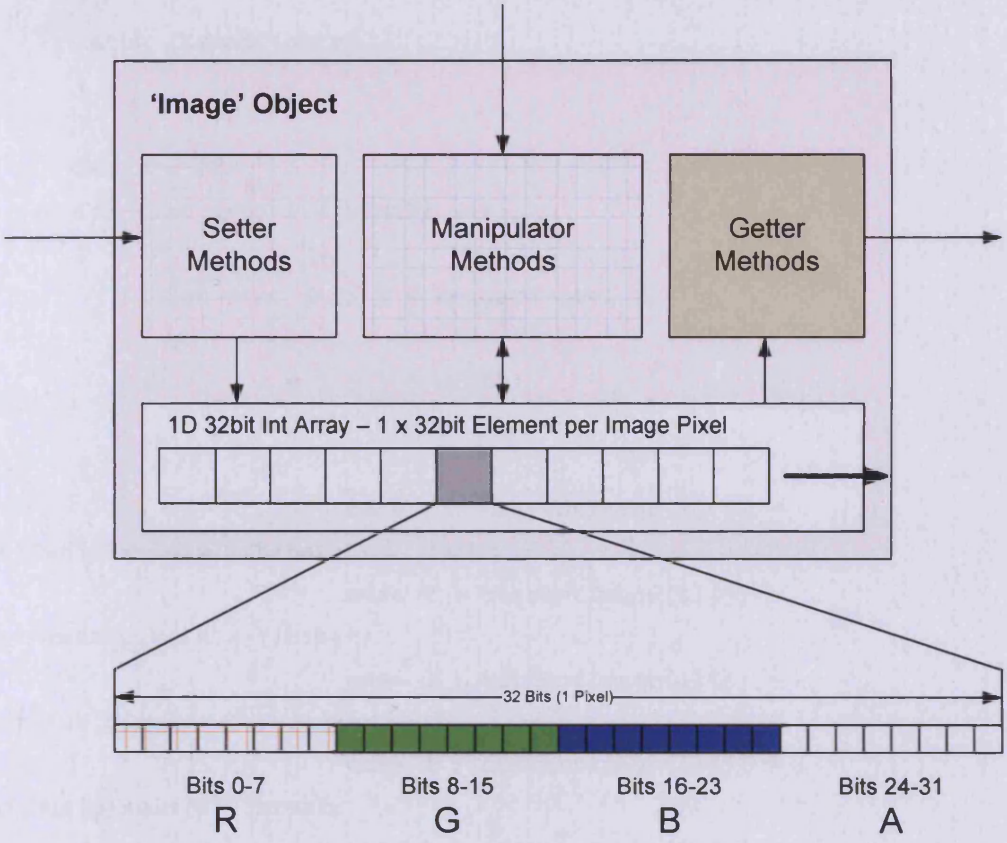


Figure 63. Diagram of the internal structure of a Java Image class

The *'Image'* class itself offers basic I/O operations and limited pixel extraction routines.

Unfortunately, the *'Image'* object does enable per-pixel reading, but only to 32bit Integer data types; the complete 32bits of the packed pixel. Generally, this type of structure is of limited use in Image Processing, and the colour channels must be individually broken out prior to processing. To do this, the *'Pixel Grabber'* class is used on the *'Image'* object to extract a 1D array of Int data types, which can then be broken up (note that Java's declaration that all data ought to be an object is exempted in the case of basic data types, for speed and technical issues, although these data types are mirrored by appropriate objects; in this case, *'Integer'* is the class which wraps an *Int* and adds additional methods).

```
PixelGrabber grabber = new PixelGrabber(tempImage, originX, originY, width,
height, pixels, 0, width);
      try
      {
      grabber.grabPixels();
```

```
	}
	catch (Exception e)
	{
	}
	int x = 0;
	for (int i=0; i < width; i++)
	{
		for (int j=0; j < height; j++)
		{
			x = i + j * width;
			switch (channelType)
			{
				case 1 : currentImage[i][j] =
getRed(pixels[x]);break;
				case 2 : currentImage[i][j] =
getGreen(pixels[x]);break;
				case 3 : currentImage[i][j] =
getBlue(pixels[x]);break;
				case 4 : currentImage[i][j] =
getHue(pixels[x]);break;
				case 5 : currentImage[i][j] =
getSaturation(pixels[x]);break;
				case 6 : currentImage[i][j] =
getIntensity(pixels[x]);break;
			} //close switch

		}//close inner for loop, using j
	}//close outer loop, using i to traverse the columns
	}
	catch (Exception e)
	{
	}
```

It should also be noted that Java does not contain an unsigned Byte data type, and as such, pixels must be grabbed to integers (consuming a larger amount of memory than is perhaps desirable).

Once this 1D array has been produced, the 32bit Int types may be split up using the classic 'shift & mask' techniques of binary processing.

```
public int getRed(int rgb)
    {
      int red = (rgb & 0xff0000) >> 16;
      return red;
    }

    public int getGreen(int rgb)
    {
      int green = (rgb & 0xff00) >> 8;
      return green;
    }

    public int getBlue(int rgb)
    {
      int blue = (rgb & 0xff);
      return blue;
    }
```

If this method seems inefficient, then that is a true representation of the task. The sum of objects required to perform this operation are:

1 x 'Image' object
1 x 'Pixel Grabber' object
1 x 1D 32bit array
3 x 1D 32bit channel arrays

Note that the channel arrays may be constructed using 16bit 'short integer' data types, but the transition between the 32bit and 16bit integers will take processing time, and will have to be reversed for any Image Processing operations designed for Int rather than short data.

It should be noted that most other languages and toolkits will give direct access to the 1D array in the image object upon loading, and a pointer may be then used to directly address each pixel position and extract the pixels one by one, shifting and masking them to obtain the 8bit channel values without the need for a PixelGrabber-style class, nor an intermediary

image-sized array to store the pixels in, prior to deconstruction. As a result, memory is saved, and the instantiation of two large objects is avoided. This is one of the points at which implementing image manipulation in Java becomes more difficult, slower and more resource-intensive than using QT, GTK, Win32 and other toolkits atop C or C++, and thus, less desirable.

7.8.3 Loading Images

As touched upon previously, the Java image classes are an interesting case of pure object-orientation implementation; some would argue, beyond reason. This is an interesting point of study, as it serves to illustrate the granularity of class design in Java, and the degree of which class functions are delineated and the intention that all functions be performed by a unique class, and that class have a specific and defined intent. It is rare to see classes enhanced with new functionality with revisions of Java, but rather, to see new frameworks be put into place, and new classes with distinctly different philosophies emerge to take the place of a conceptual group of older classes. There is some sense that this process may become more relaxed with the introduction of Java 1.5 and less rigid features, such as dynamic casting, but that remains to be seen.

Loading and saving in Java originates from the original design to produce a language able to develop embeddable applets for web browsers and shares many assumptions from that age. Firstly, saving of images in encoded forms, such as JPEG or GIF was not available to Java developers, except through the judicious addition of third party modules until the release of Java 1.4, as the creation of Java applets did not foresee that there would be any requirement to save images. Rather, images were intended to be used to embellish interfaces with elements such as images on UI buttons. It should be noted that the Java 1.4 image saving code was still difficult to get operating correctly (a maze of small single-operation classes based around streams and abstract encoder classes that had to be inherited from and algorithms introduced).

In order to load or save an image in Java, using the simplest form, an interface object must be used to transform the data from a stream to an Image object. In the case of saving, using a third party encoder set, we can name the interface class as an Encoder. For loading, however, the situation is clear, and yet more complicated. From the applet heritage, Java's developers assumed that an image would be used in a user interface, and that meant an Applet or later, the window of a stand-alone Application. Since a user interface on exotic hardware might not

share a traditional RGB colour model, but might be natively CMYK or some other form, it was decided that a user interface component should be able to create transformation objects that would accept a file or data stream, filter the data through the appropriate conversion algorithms and convert the image to the correct colour space, prior to injecting the data into an image object. This transformation object is named a 'Toolkit'. The essential problem with this design philosophy, is that it fails to take account of the situation where a person might wish to manipulate images in an application that does not provide a user interface, since the loading of images has become bound to the ability to obtain a Toolkit, which may only be provided by a user interface visual object. As a result, it is technically impossible to produce a 'headless' (that is, without a user interface) image-processing application in Java (see fig.64), and indeed, Myriad servers have been forced to acquiesce to this design goal by creating a user interface space which is never enabled and made visible. While testing proves that Linux systems with Java 1.4 JVMs and no X-windows server running will execute the application without error, it is quite possible that other systems and JVMs may fail to run the server due to the inability to support a UI object when there is no graphical server daemon running.

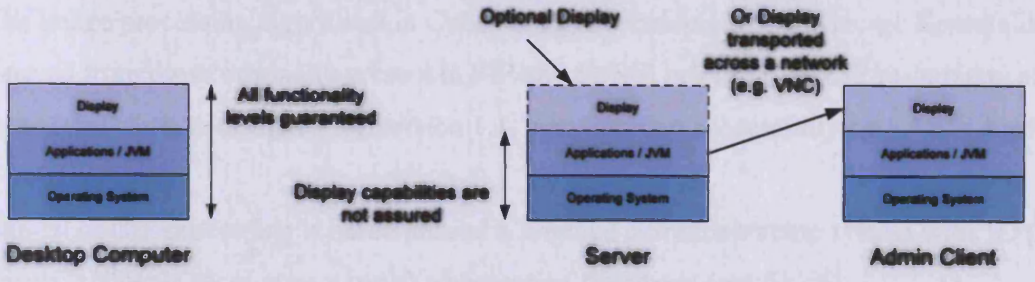


Figure 64. The use of Java applications in a 'headless' computing environment

Associating the image loading diagram shown above with the channel-extraction described previously, the total system required to examine a single channel of an image stored locally as seen in fig.65.

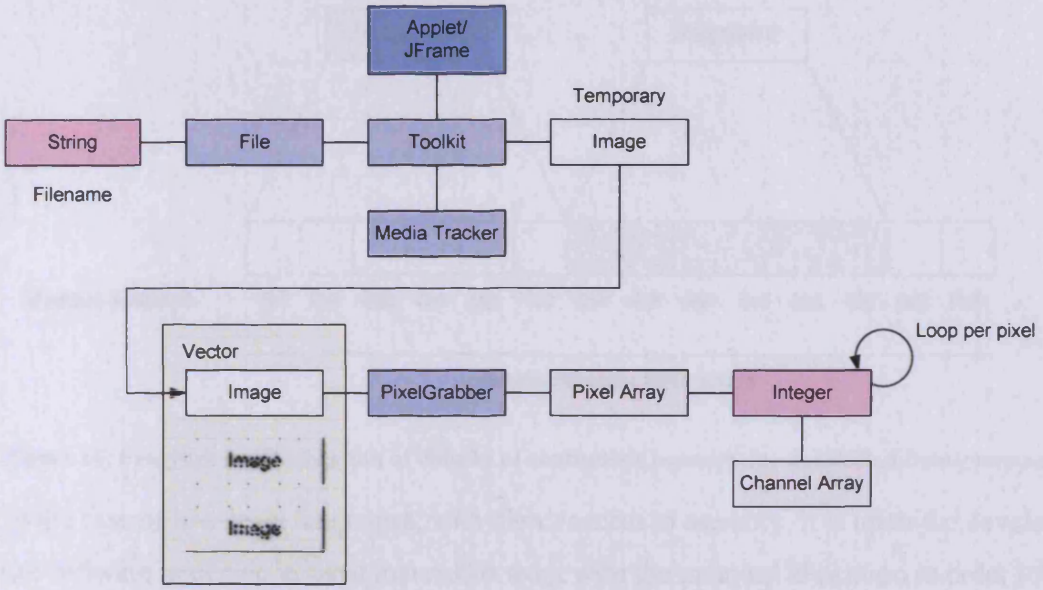


Figure 65. The Java process to load an image from a string URL and parse it into a simple 2D array of pixel intensities

7.8.4 Interfacing with CIP algorithms

The image processing algorithms in Cyber Image Processing by Mr. George Karantalis are derived from those originally present in PIP and SUSIE prior to that. CIP re-implements these operations in Java, designed for version 1.1, but operating successfully on 1.4 and above.

Design of CIP processing is based around a 2-image current/alternate system with 1D pixel arrays (elements = columns x rows) representing the pixels (see fig.66)

The choice of 1D arrays may seem somewhat odd to most developers expecting 2D representations to be a more meaningful data structure for a 2D object such as an image. Choosing 1D arrays appears to have been upon the basis of a quirk of the Java Virtual Machine specification. Due to the nature of memory allocation in modern operating systems, arrays are only able to be 1D.

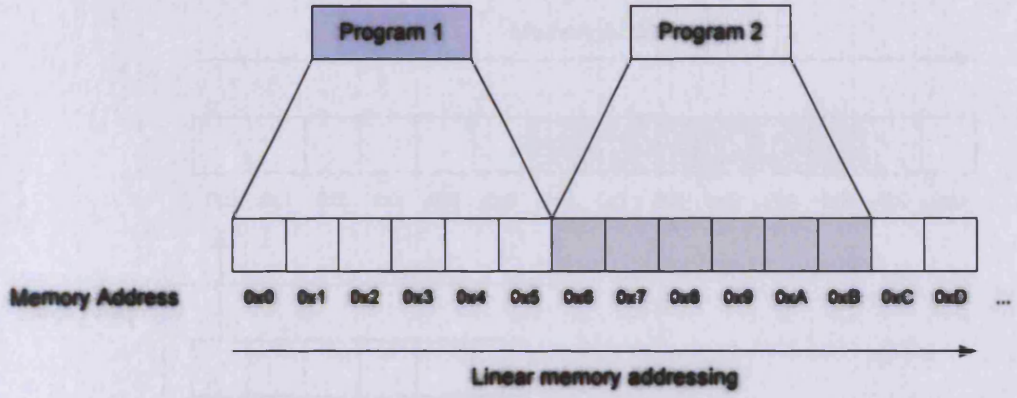


Figure 66. Program-level allocation of chunks of continuous memory for quantified data/commands

In the case of low-level languages, with direct access to memory, it is up to the developer of the software to decide in what manner to work with the memory allocation in order to further their own needs, to allocate by column (see fig.67), or by row (see fig.68).

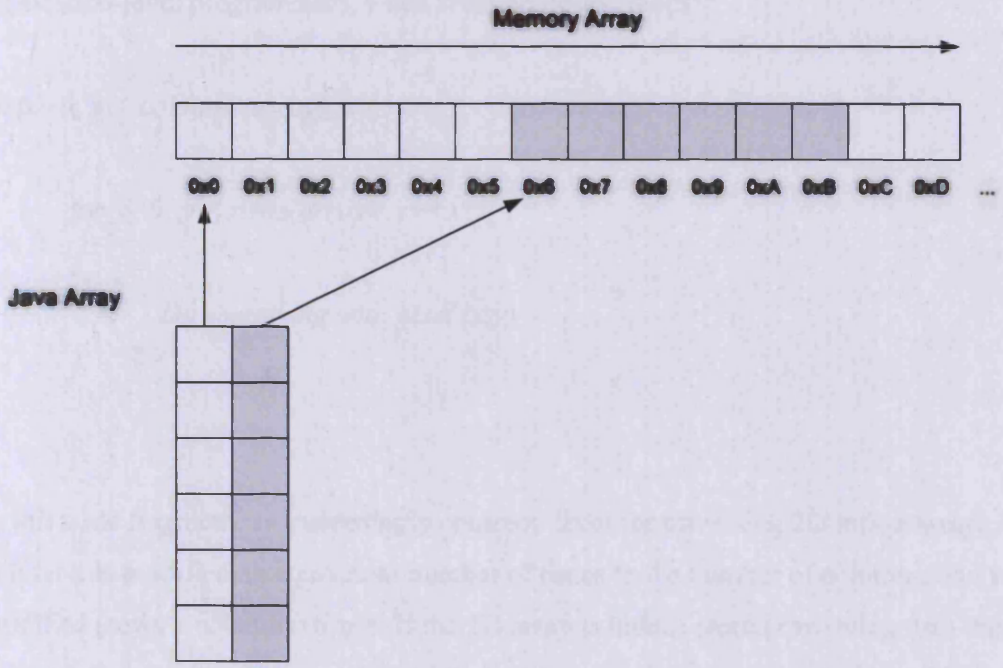


Figure 67. Diagram of column-wise allocation of memory to Java for storage of a 2D data array

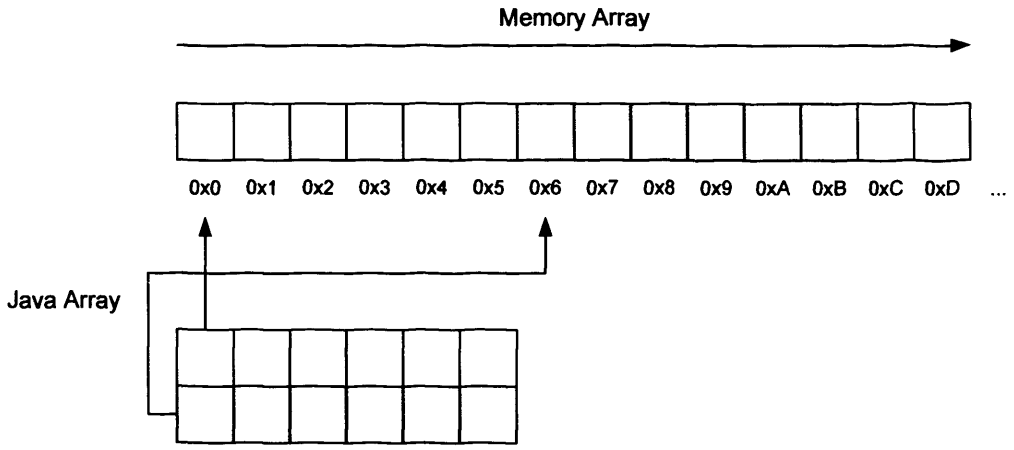


Figure 68. Diagram of row-wise allocation of memory for a Java array

The problem with Java's VM specification, is that it does not declare whether VMs ought to store 2D arrays row- or column-wise. In failing to do so, it presents a problem in that application-level programmers, when creating nested loops:

```
for(x=0; x < columns(array); x++)
{
        for(y=0; y < rows(array); y++)
        {
                Do something with pixel (x,y)
        }
}
```

In this code fragment, an exceedingly common form for traversing 2D image arrays, the pointer x is modified an equivalent number of times to the number of columns, and y is modified (rows x columns) times. If the 2D array is indeed stored row-wise, then the pointer is simply being moved along a linear block of memory, one element at a time. If the 2D array is stored column-wise, however, the process becomes far more complicated, as each increment of y needs a recalculation to move a full column length through memory as seen in fig.69.

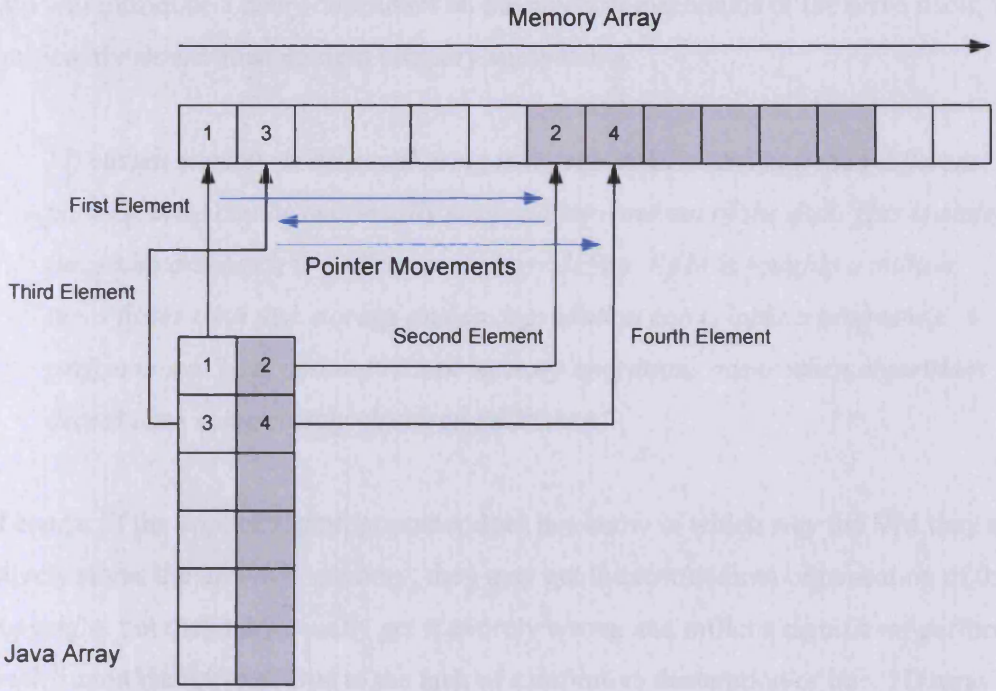


Figure 69. Diagram of iteration through a 2D array where the iterator is row-wise, but storage in memory is column-wise

This involves a calculation of the length of the columns and row positions, multiplied for each pointer movement between elements which may seem consecutive in the 2D Java array, which consumes multiple CPU cycles. This operation is then performed for every pixel in the image, which amplifies the difficulty.

Lyon [LYON] explains this process with comparison to the C standard, which mandates 'row major' (row-wise) array storage, enabling C programmers to program loops on this basis:

> *"Unlike C, there is no way to know if Java stores arrays in row major order or column major order. This is a major problem for image processing programmers. For a two-dimensional array, this means that we cannot know which index to vary more quickly."*

Lyon also emphasises that large images and data structures organised in the incorrect manner pose an even greater problem for performance, as virtual memory (hard disk RAM simulation) will amplify the performance penalty of out-of-order data access, since every

jump will introduce a delay dependent on the physical mechanics of the drive itself, which is significantly slower than modern memory subsystems.

> *"If virtual memory is accessed using non-sequential addresses, then different parts of RAM can be continually swapped into and out of the disk. This is called thrashing and leads to performance degradation. RAM is roughly a million times faster than disk storage and so degradation can cripple a program's performance. Thus optimal virtual memory operations occur when algorithms access data using closely clustered addresses."*

Of course, if the application programmer does not know in which way the VM they are using actively stores the arrays in memory, they may get the row/column organisation of the for loops right, but they may equally get it entirely wrong and inflict a significant performance penalty upon themselves. Due to the lack of a definitive declaration of how 2D array storage is meant is meant to happen in VMs, in Sun's specification, different VMs use differing methods; some use column-wise allocation, while others are row-wise. Recent developments attempt to detect these issues while compiling and reduce the potential penalty, but it still exists. At the time CIP was developed, the penalty was significant, leading to a middle ground solution of converting all images to 1D arrays, and manually traversing the 1D array in order to solve the problem and ensure that the inner loop is always the more efficient type.

In order to communicate with the CIP image processing commands, previously mentioned methods of extracting 2D pixel arrays of intensities per channel of the image must be extended to adjust the final step from populating a 2D byte array to populating a 1D equivalent.

This process is actually far simpler than that of the 2D case, because the PixelGrabber, which is used to extract the pixel data from the Image object provides a 1D array by default, and thus the pixels need only be read sequentially from the output of the pixel grabber.

For the Java code for this operation, please see Appendix D.6

The HSI values for each pixel are slightly more complex to resolve, as they require the acquisition of all the RGB values and then the performance of calculations based upon the

values returned from those operations. These implement the following mathematical transformations:

Hue:

$$Hue = acos(((2 \times r) - g - b) / (2 \times \sqrt{(((r - g) \times (r - g)) + ((r - b) \times (g - b)))}))$$

Saturation:

$$Sat = (3 \times \min(r,g,b)) / (r + g + b)$$

Intensity:

$$Int = (r + g + b) / 3$$

For the Java code for these operations, please see Appendix D.7

It is likely that intensity will be the most commonly used channel in all use, due to the requirement for it in the production of greyscale and binary imagery. In the case of over-contrasted outdoor webcam images, however, the effect of poor iris control tends to induce close-to-greyscale representations in the images. For these purposes, and with the intention of performing rapid operations on streaming and short-periodic frame sequences, it may be meritorious for systems to assume that RGB channels are sufficiently similar to allow for the use of an R,G or B channel in substitute for the Intensity calculation, offering a slight speed improvement. In cases of large numbers of images of poor quality and limited time in which to process them, this method may be valuable, although it does potentially discard very large quantities of image data.

Once this process of 1D channel array creation is complete, the use of CIP image processing commands is almost at hand. All that remains is to transplant the image data arrays in the place of two placeholder arrays, named *'altimage'* and *'currimage'*; these are the nominated variables within which the traditional CIP Java operations expect to find the current and alternate image 1D pixel arrays. At this point, CIP operations may be initiated, executing as they normally would while incorporated within the full CIP application. Once complete, the

reversing process may be used to revert the 1D arrays into 2D and Java *Image* objects, usable by all other non-CIP image processing commands and Java methods (see fig.70).

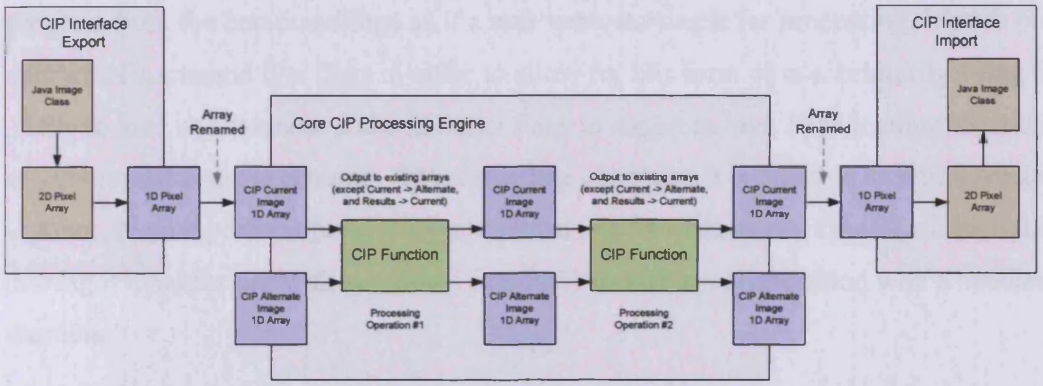


Figure 70. Operation of CIP functionality for image processing chains, segmented into I/O and core IPE functions

For the Java code for this transformation, please see Appendix D.8

7.8.5 Interfacing with Matlab

One of the most utilised mathematical development packages used in research environments is MatLab, an integrated development environment and interpreter for mathematical expressions and related extensions. The scripting language for Matlab can be actively extended through the use of 'toolkits', which are purposed libraries, such as the Image Processing toolkit, the GUI toolkit, and many more.

Conventional use of Matlab involves the creation of script files (named 'M files', with the filesystem extension '.m'), which are then executed, may call each other, and are able to have objects and data passed between each other. During development of M files, it is typical to produce an M file using the development environment text editor and then execute it with parameters through the Matlab internal command line window.

The use of Matlab functionality and M files in other applications, however, may not typically use the internal command line, and thus must examine alternative means of initiation of functions. For this, three options are available:

- System command line execution
- Inclusion of binary-compiled M files into native applications
- Execution through HTTP request via Matlab web server plug-in

For the first of these options, the use of Matlab and the starting of operations follows the patterns shown with the use of wget from PHP; the intention being to mimic the use of the program from the command line, as if a user were starting it for processing. Matlab provides a number of command line flags in order to allow for this form of use, primarily a flag for the M file to load and execute, and additional flags to suppress Java JVM loading, GUI display and command console echoing. With these flags in place, it is possible to start a Matlab instance, have it perform processing and return results without any kind of additional display, making it ideal for use with secondary applications and in collaboration with a headless server daemon.

In PHP, this type of functionality would be provided through a gateway script as described previously. The code for this specific solution is as follows:

```
<?php
$fr = fopen('instructions.m', 'a+');
fputs($fr, $instructions);
fclose($fr);
passthru('matlab –nodesktop –nosplash –minimize –r instructions.m');
?>
```

In addition to PHP, however, this type of interaction may be used by all programming languages with the capability of accessing the system command line, and as such, provides a generalised means of engaging the use of the Matlab executable environment with minimal customised code. The disadvantages of this system, however, are simply that each instance of Matlab created in this manner will consume the same amount of system resources as if it were started by a user for normal use (albeit without the graphical interface and JVM). This is especially true under X Windows environments, where Matlab is particularly heavy and slow to start up conventionally.

Options for the inclusion of Matlab code into C and C++ are numerous and are best assessed through the documentation provided with the program at the current time. However, examined versions enable the following:

- Creation of C and C++ source, which may be directly compiled

- Creation of native binaries which may be included through stack interaction and system messaging
- Provision of a small Matlab executable library with which to interpret M files, which may be linked to by custom-created code

The final option for the use of Matlab, is the Matlab web server plug-in. Sold somewhat erroneously as a web server program in its own right, it actually provides a command line interpreter similar to the one that might be use with the PHP gateway, except that it has GUI elements and other redundant functionality that is of little use to a headless server, removed. As a consequence of this action, this executable is far smaller and faster to execute, and also has the ability to start the first instance as a daemon, and subsequent requests operate as multiple threads of the daemon with shared resources, further reducing execution time and performance impact. This plug-in currently only operates under Windows NT-based operating systems, but it does provide hooks for both Internet Information Server and Apache.

To use the Matlab web server plug-in, an M file must be created, which accepts data as an argument and outputs results formatted with the assistance of HTML-constructing M files (although any direct command line output by the M file will be echoed to the client, allowing for non-browser access). The primary limitation of this system, is that the M file used as a gateway to this functionality must be declared within a file on the web server, only changeable by the system administrator. For example, Prof. Batchelor wrote an M file with the assistance of this author, named 'pesr.m'. This M file accepts a single argument; a string named 'argumentb' which is implicitly converted into a Matlab internal variable, and itself contains Matlab directives (from a single command to a complete script).

Using this mechanism, therefore, it is possible to send commands to Matlab using the following URL format:

http://vision.cs.cf.ac.uk/cgi-bin/matweb.exe?mlmfile=pesr&argumentb=<command_sequence>&Submit=submit

Where <command_sequence> is replaced by a UTF-8 encoded set of Matlab commands. In this URL, it is clear to see the M file in the sequence mlmfile=pesr and argumentb=, indicating the data that should be loaded into the implicitly declared variable argumentb.

In PHP, this interaction would be done with the following call to wget, as mentioned previously:

```
<?php
passthru('wget –q –O output.htm http://vision.cs.cf.ac.uk/cgi-
bin/matweb.exe?mlmfile=pesr&argumentb=<command_sequence>&Submit=submit');
?>
```

7.9 Equipment Control

In order to access equipment controllers, there are four typical hardware mediums:

- RS232 Serial Port
- Parallel Port
- Universal Serial Bus
- Ethernet (network) connectivity

Of these, serial port control is the most common for device control mechanisms, although USB devices are becoming increasingly common. For most equipment control purposes, however, the higher speeds of USB and multi-device connectivity are redundant. The two types of device control hardware utilised by Myriad prototype Equipment Control (henceforth, EC) servers are both RS232-based, and as such, will consume the majority of this section.

Due to the nature of device control and commonly custom-constructed hardware systems, the development of a generic server is not a realistic objective; that said, a Myriad server prototype is able to demonstrate the means of integrating such functionality within the framework, and may be used as the revised basis for customised server components for differing hardware.

Using Java for serial operations, the essential module for access to functionality is the COMM package (API). Presently available for Windows 9x series and Solaris 8/9, third party ports have been produced for alternative platforms such as Windows NT/2000/XP, Linux, FreeBSD and NetBSD. These all inherit the same classes and operational style from the COMM specification produced by Sun Microsystems, so they may be mostly directly replaced, albeit with a minimal change to the recognition of serial port identifiers. Possibly the most

confusing element to the use of the Java COMM package, is that they are used in the following procedure:

- Obtain a list of system serial ports
- Iterate over the list and extract port identifiers (e.g. COM1 (Windows), dsp1 (Linux).
- Compare port identifiers to programmer's string until the appropriate port is found
- Extract the port object, set parameters
- Obtain data input and output streams from the port
- Read/Write streams
- Close streams and port

While this may seem a perfectly good strategy for the use of serial ports, it does fail to suffice in one critical respect: it requires that the programmer provide a list of matching port identifiers and, greater than this, understand the target platform for the application, along with being able to produce a predictable list of identifiers. While this does not entirely prevent multi-platform use, it does reduce cross-platform use of Java with the COMM package to the less savvy programmer.

7.9.1 Serial connections

Windows

Serial port connections under Windows, both in the 9x and NT-based series is based around two channels, named COM1 and COM2, which are unique entities. There are also a sequence of higher COM channels, from COM3 upwards, which alternately map to the two base channels in hardware terms. So, COM3 is a redirected map onto COM1, and COM4 is mapped to COM2. At the hardware level, these are able to be differentiated, but since they share the same interrupt, high speed traffic over one of these aliased channels can affect other aliases on the same channel, and may be interrupted by interrupted on any alias to the same channel, as well. Standard on-board serial ports on x86 desktop computers will connect the first physical RS232 connector to COM1, and the second to COM2.

In order to control multiple serial port devices on a computer with a single hardware serial port, the Cardiff University Machine Vision group uses Serial-USB converters, which connect to a USB port on the computer, and on the opposing end, present a 9-pin D RS232 connector. A driver then acts as a bridge in software and intercepts traffic to the COM3 channel. As a

result, the computer has devices on COM1 and COM3 able to used in the normal manner, presented below.

Before attempting to test any serial port communication code demonstrated here, the reader is advised to carefully examine their Windows settings to ensure that all of the buffering systems present are disabled (they are numerous).

The following code is derived from code developed by Mr. Michael Daley of Cardiff University for inclusion in the JIFIC (Java Interface to the Flexible Inspection Cell) application discussed in the paper "Teleprototyping Environment for Machine Vision", by M. Daley and B. Batchelor in 2001 [BAT01]. Iteration is slightly changed, and port identifiers are compared against an external variable ('commport'), passed from application instantiation, allowing the server administrator to specify a serial port identifier on start-up of the daemon, rather than applying a constant string value.

```
public static CommPortIdentifier portId;
public static SerialPort serialPort;
public static Enumeration portList;
public static OutputStream outputStreamMMB;
public static InputStream inputStreamMMB;

.
.
.

commport = commString;

portList = CommPortIdentifier.getPortIdentifiers();

while (portList.hasMoreElements())
{
      portId = (CommPortIdentifier) portList.nextElement();
      if (portId.getPortType() == CommPortIdentifier.PORT_SERIAL)
      {
         if (portId.getName().equals(commport))
         {

         try {
```

```
        serialPort = (SerialPort)portId.open("SampleReadApp", 2000);
        } catch ( PortInUseException e ) {}

        try {
        inputStreamMMB = serialPort.getInputStream();
        } catch ( IOException e ) {}

        serialPort.notifyOnDataAvailable(true);
        try {
           serialPort.setSerialPortParams(9600,
                           SerialPort.DATABITS_8,
                           SerialPort.STOPBITS_1,
                           SerialPort.PARITY_NONE);
        } catch ( UnsupportedCommOperationException e ) {}

        try
        {
        outputStreamMMB = serialPort.getOutputStream();
        }
        catch   ( IOException e ) {}
        }
      }
} //End while
```

This process is relatively straightforward; the enumeration 'portList' is created using a list of all port identifiers from the COMM package. A while loop then continues as 'portList' has remaining entries, which are consumed when the first line within the loop takes the next element from the list. The type of this port is then checked to see whether it matches 'PORT_SERIAL', a constant from the COMM package. If this is the case, then the name of the port is checked against 'commport', and a matching port then is opened, has parameters set (baud rate, etc.) and data streams are obtained from it.

Due to the multi-threaded nature of the structure of Myriad servers, the data streams, port, port list and port identifier are declared to be 'static', which forces the entire server daemon to share the same instances of the objects, rather than maintaining unique copies. This means that writing to a serial port from three different client handlers will not result in the data being interleaved with each other as three sets of data streams try to write to the physical hardware at the same time, but rather, the single stream will marshal the data and write it sequentially.

Unix

The nature of Unix systems demands that serial port devices are mounted onto the file system as generalised data resources, addressed as if they were files. Serial ports are mounted in the /dev/ directory (denoting devices) and commonly with the name 'dsp'. The first serial port will be dsp1, the second dsp2, and so on. It is also typical for USB devices to be mounted in the same manner with the name 'usb', hence, 'usb1', 'usb2', etc. Control and translation of communication to devices takes place as a function of kernel treatment, but it is expected that bare USB connections are managed in precisely the same way as serial. As a result, using Java's COMM package under Linux should allow USB peripherals to be controlled in the same way as serial devices. The only difference between this and the Windows example, is that the list of port identifiers must be compared against the 'dsp<x>' and 'usb<x>' name types, rather than COM<x>.

7.9.2 Parallel Port / USB / Firewire

The present Java COMM specification also includes support for parallel port, Firewire and USB communication, in addition to serial access, but development is relatively slow, and this functionality has not filtered through to all of the third party editions, as of the time of writing. As USB industrial devices proliferate, this capability will become increasingly essential for medium-weight applications (although Java is still not mission-critical), with the advantages of:

- Higher speed
- Multiple devices per connection
- Bus-powered devices

Parallel-port communication is also possible under the current Java COMM specification, although the standard is in decline at this point. It is useful to note, however, that there are additional Parallel-USB converters, which operate in the same way as the serial converters, but provide a 25-pin D parallel port connection, rather than a serial one. This holds the promise of maintaining the use of parallel and serial devices for the long-term, even as computing devices progressively lose these capabilities in the push to become 'legacy-free'. Interestingly, these measures also enable devices such as Smart Phones and PDAs, with USB ports on-board to communicate with parallel and serial devices, despite their form factor preventing the inclusion of a full-sized port for those communications methods.

Firewire as a serial interface is fully supported by the current Java COMM specification, and again, this functionality is patchy, depending on the origin of the package in use. In general, however, Firewire support is only useful for high-bandwidth applications in this context, such as video streaming. The benefits of the peer-to-peer aspects of Firewire are unnecessary in the case of a desktop computer, as it would traditionally maintain control and thus derives no benefit from peer control of devices beyond that which already exists.

7.10 Prolog Gateway

The creation of a Prolog server component, to provide intelligent control in Myriad networks, follows the design of a gateway, rather than a freely-controlled and autonomous server. This difference occurs, because as a language interpreter, Prolog does not innately require all of the additional functionality introduced by MCS. In general terms, Prolog can provide functional equivalence to the basic logic operations and features present in MCS, along with implicit multi-threading through the use of Apache as a multi-threaded web server initiating the execution of each Prolog instance. Moreover, a Prolog component acts primarily as an intelligent translator of instructions to other sections of a Myriad network, or intelligently translates data being returned from those sections. The intention is to act in all instances as a controller, and very rarely as a pure data or processing resource in its own right. This is not to say that this cannot be done, but by and large, the core advantage of Prolog use within this context is for dynamically managing systems and resources with intelligent intentions, rather than engaging in post-consumption examination of data (at the very least, it is typical to use Prolog to intelligently select the data source and output from a resource, and control data capture in some manner)

The first component of a Prolog gateway system, is a PHP script, able to invoke the Prolog executable and return results. The structure of this type of script has been explored previously, and thus requires little explanation:

```
<?php
if($query)
{
$commandstring = "pl -f test.pl -g \"solve(" . "(" . $query . "))" . "\" -t halt";
$commandstring = str_replace("\'", "'", $commandstring);
passthru($commandstring);
```

```
}
?>
```

The central element of this script is the $query variable which is automatically instantiated by the PHP interpreter; the 'if' statement tests for the existence of this variable, and then proceeds to process it, if indeed it does exist. The query string is then embedded into a command line execution string as an element of the '-g' switch, indicating to the Prolog interpreter the goals to attempt to satisfy. This can include directives and predicates, enabling users to call current functions and add their own to the knowledgebase. The other switches are:

-f File to consult prior to attempting goals (includes pre-defined predicates)

-t Operation to perform on termination (completion/failure)

Note that \" is an escaped quotation mark, which must be escaped in the string assignment, otherwise it will cause premature termination of the RHS. The result of this assignment is:

pl –f test.pl –g "solve((<query>))" –t halt

The use of quotation marks in this string allow for the entire goal to be delimited and avoid the Prolog interpreter confusing strings with embedded spaces as the start of new command line argument; eg.:

pl –f test.pl –g solve((http_get url2)) –t halt

Without the quotation marks, the 'url2))' component would be interpreted as a new command line argument, and the goal would be only 'solve((http_get'.

The objective of such a script is to pass commands directly to the interpreter and return results without intervention, and this is the cause for the use of the 'passthru' command, where execution is performed and results are 'piped through' to the client, without reprocessing in PHP. This is not to say that through the use of 'system' and 'exec', PHP's other system command line execution methods, results cannot be intercepted by PHP and used as the basis for processing.

The second line within the 'if' command block is an example of the 'str_replace' command, which does non-regular expression replacement through a string of a target with a replacement:

str_replace(<search>, <replacement>, <string to search through>)

Because all data coming to the PHP script through encoded URLs will be escaped, although the general UTF decoding will be done, apostrophes will be escaped to \', which need to be returned to the single apostrophe before they are passed to the Prolog interpreter, which will not be expecting rogue slashes in the middle of command strings.

One of the limitations of SWI Prolog is also visible in this gateway code, in that the query itself is wrapped in a command named solve(), which is a custom predicate produced by the author and Professor Batchelor. It transpires that the calling of undefined predicates generates instant failure of the Prolog interpreter with an error message and no output; most specifically, it does not output processed data up to that point, nor does it announce the predicate that does not exist. Depending on the inclination of the programmer, there are two options for the processing of an error state:
Catch the error and terminate with an error message
Catch the error and continue without messaging

Unfortunately, SWI Prolog does not provide a catch-all predicate that can be executed once all other known predicates have failed to match. In order to correct this behaviour, a meta-interpreter recursive predicate has been written, which serves to pass through commands to the interpreter, and catches error states, preventing them from propagating up the execution tree to the core interpreter process, allowing for the possibility for controlled error reporting or continued operation.

```
solve(true).

solve((A,B)):- solve(A), solve(B).
solve(A) :- catch(A,_,(nl,write('Goal '), nl, write(A),nl , write('was not
recognised'), nl,halt)), solve(B).
solve(_) :- fail.
```

In the case of the current predicate, a bad predicate will fail the entire processing after the point at which is occurs, but will present an error message to the user, ensuring that an undefined predicate can be discovered and corrected.

The predicates required for generalised use of Prolog within a Myriad environment are up to the programmer in most cases, but there is a requirement within a networked system using HTTP as a communications protocol to provide a means to request HTTP resources (in the form of built-in predicates). This code is straightforward, and includes only the loading of a HTTP library that ships with SWI Prolog, opening a data connection and reading out atomic strings from that connection. The predicates for this are included within the test.pl file, which is consulted through the command line –f switch.

```
:-
use_module(library('http/http_open')).

http(A) :-
http_open(A, In, []),
read(In, X),
process(X,Z),
writeln(Z),
close(In).
```

The ':- use_module...' line loads the module (HTTP open commands and tools within the larger HTTP library) and is implicitly executed when the test.pl file is consulted. The section between 'http(A) :-' and the terminating full stop is a declared complex predicate (which can be considered as analogous to a function/subroutine in other languages). This runs the http_open command with the string A (a URL passed to http(A)), collects a data stream 'In', and leaves an empty list for further arguments '[]'. The next lines read from the 'In' data stream into the X variable, calls some undefined process(X, Z), which leaves output in the variable Z, and finally outputs Z to the standard output before closing the stream and terminating.

To download a file from a web server or Myriad component, therefore, all that is required, is to give the command:

http(<URL>)

E.g. *http('http://www.myserver.com/test.html').*

The PHP file, command line, Prolog interpreter, predicates file and the Apache web server all combine to form a complete gateway system as seen in fig.71.

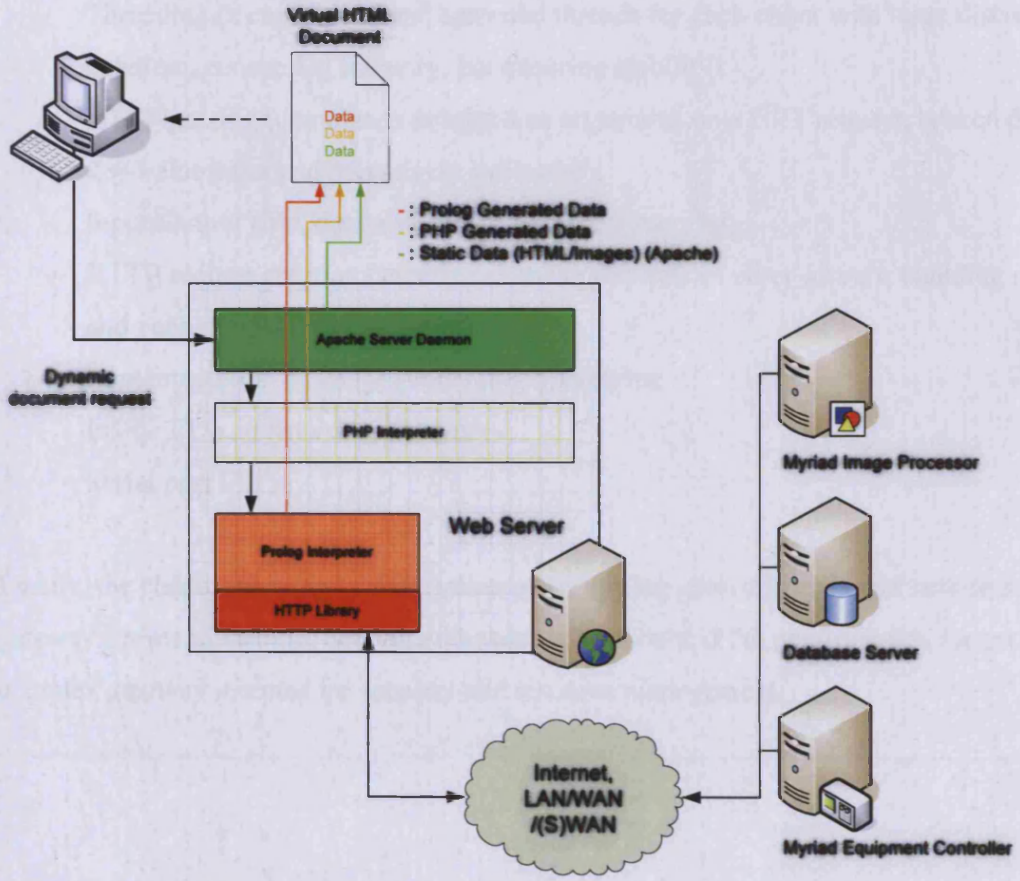


Figure 71. The structure of a Prolog gateway constructed with Apache, PHP and SWI Prolog in a Myriad network

As part of a larger whole, the Prolog gateway can be fitted into Myriad solution as an intelligent intermediary between other elements of the system and components such as image processors and equipment controllers.

Myriad servers are designed as multi-threaded web servers augmented with MCS, the scripting language, with a generic core of commands and a specialised set of commands

depending on the purpose of the server in question. An image processing engine, therefore, includes the core functions and image processing commands, where the equipment controller contains the core and hardware control commands.

This chapter discussed how the Myriad demonstration servers were written in Java, how they work in respect to a number of key issues:

- Threading (a core thread and spawned threads for each client with large discrete handlers, consuming memory, but ensuring stability).
- HTTP parsing (commands accepted as arguments on a GET request, broken down into key-value pairs and recursively evaluated).
- Input/Output (file, database and serial data streaming).
- HTTP request creation (servers can make requests of other servers, enabling meshes and peer-to-peer data exchange).
- Implementation of image processing operations
- Image I/O and Java data structures
- Serial port I/O

Finally, the chapter provided a description of the Prolog gateway script and how to construct gateway scripts in general, dealing with data redirection and the opportunities for creating complex gateway systems for security and resource management.

8 Interfacing with Myriad

Networked machine vision systems may expose their functionality to network-connected machines, but they sit idly as generic processing applications until they are controlled by another application, script or other tool. One of the most useful aspects of Myriad as an implementation of a networked vision system, is the ability of components to be controlled remotely through the use of HTTP requests. With the majority of modern programming languages able to create and execute HTTP requests (or able to utilise command-line programs such as wget to communicate on their behalf), machine vision capabilities are quickly and easily introduced to a wide variety of programs and fields of endeavour.

This chapter explores how the user may interact with Myriad components using HTTP requests, from a variety of different programming and scripting languages, with conceptual discussion and examples.

Operating Myriad components requires:

- The creation of a TCP/IP socket
- Emission of an HTTP request string through that socket

Using a purely text-based protocol, such as HTTP reduces the requirement of development on the user to simple string creation and socket creation and also allows for the use of less conventional control agents, such as a web browser or command line caching software, such as wget or curl.

8.1 Types of Controller

There are a number of controller types that may be used with Myriad systems, which fall into principle categories; browser, GUI & Application and Script.

8.1.1 Browser

Using a web browser as a means of operating and testing an HTTP-based system is highly useful for a number of reasons:

- It requires no additional programming; browsers are designed to emit HTTP requests, by definition
- All modern Internet-connected computers, palmtop devices and cellular telephones have some form of web browser as standard.
- Using Javascript and server scripting, complex interfaces may be speedily constructed

Web browsers and associated user agents, therefore, provide a very advantageous test harness for Myriad components and systems; also removing the need to write customised protocol code and test clients for advanced testing.

8.1.2 GUI

In the majority of small Myriad systems, the Graphical User Interface is the primary means of controlling the system, more so than large-scale applications and scripts. Design and development of these systems is often through visual layout tools and RAD construction IDEs, such as Visual Basic, Delphi, Kylix, H-Basic, Java tools (such as Eclipse), QT Designer and TCL/Tk.

The concept of a GUI should be distinguished from an application at this point, because of the lightweight nature of the GUI; it does not do a great deal of computational processing itself, but rather provides a tool for monitoring and basic control of far larger systems. While this may seem to be something of an artificial distinction, when working with a server-based framework, it does become increasingly relevant. Since the active development of multi-tasking operating systems, it has been a trend in program development to produce software systems which are less monolithic, and consist of a core program and a interface to act as a viewer and controller. In Myriad, we see much the same pattern, except that the core program may be anywhere in the connected world and the interface is able to administer more than one core program at one time, should it so choose. This is a neat construction, because it allows the user interface to be changed, while maintaining the integrity of the core program, even while it is constantly operating. This allows for multiple user interfaces, user interface development and replacement, and protects the core program from user interface stability problems.

Distinguishing the GUI from the core program requires the development of an interface system using RPC, direct method calling, message passing, or a client-server model. While

RPC and direct method calls are the most efficient of these systems, they do impose limitation, as to the freedom of interaction, as most RPC mechanisms are platform specific (unless they use heavy management and routing systems, such as CORBA) and the GUI must know the full range of methods prior to use. Also, RPC mechanisms do not offer the scripting flexibility of MCS, with respect to variable processing, local looping and local logic. This essentially leaves messaging and a client-server relationship as the two most viable means of interaction. Message passing is useful, but in itself, it does little; it must be attached to either an RPC concept, or a scripted client-server concept, which is a superset of messaging. As mentioned previously, offering scripting capabilities on a server system has many advantages, most key amongst those, network-independent automation.

Using a client-server model for interaction and HTTP requests to transmit data, Myriad provides a simple means for GUI programmers to develop Vision-assisted tools. All of the languages mentioned above include in their most recent iterations (after 1996/7) HTTP-request functionality, with the intention of downloading web pages and data for inclusion into their interfaces and providing the basis for networked applications. Indeed, past versions of Kylix failed to include network tools in the cheaper version and instead, chose to charge large sums of money for a Professional edition with the functionality returned, which may have been one of the most contributing factors in the market failure of the product (despite being the ideal cross-platform Delphi-type development environment).

In essence, for the conventional GUI programmer, operating a Myriad component is composed of two steps:

- Compose a simple URL string with the relevant commands
- Get a web page/image using the URL string

Using conventional HTTP request tools, this means that basic tasks can be performed through the user interface, without additional handlers, managers and threads having to be created, which massively reduces the load on the GUI programmer, and also enables abstraction from the hardware and software complexity built in the server, because, for example, the GUI requesting a light being turned on does not need to understand the protocols used to liaise with the logic board operating the light switching.

Most often, in GUI situation, this type of interaction will be one-shoot, without processing and tied to specific GUI elements; e.g. a click on a particular button simply sends a request for equipment to be started, where another asked for it to be stopped, without further processing on the GUI side.

In should be noted that a number of MSc and BSc students have produced projects including GUI tools for managing Myriad components and equipment (such as the Pan & Tilt camera) and have been able to build basic control systems from scratch within a few hours, commenting on the ease of integration of Myriad components.

8.1.3 Script

While the use of GUIs for control is useful, most especially in non-repetitive operator-focused, interactive systems, there are often occasions where more regular automation is required. Commonly this will be in the operation of equipment, but may also include regular processing of images or other data from a variety of sources. For these purposes, scripting languages are a most useful, lightweight manner of control that allows the programmer to create systems quickly and concisely. In many problems, where most of the tools already exist to provide a solution, there is a need for a programmatical glue that starts the process, collects and passes data between components and presents the final data or forwards it elsewhere for further processing. This type of solution will be seen most clearly in the example of the LinPacker 2D packing system, which will be elaborated on later.

Scripting languages, as such, are one of the key elements of large-scale and complex systems. In most cases, the languages themselves are simple, focused on string manipulation and performing most functions through the use of other programs, which they are able to manage thanks to access to the command line interface. A typical and most relevant example of such a feature, is the use of *'wget'* or *'curl'*, which are two command line programs, which simply download a file from a given URL.

Using wget, for example, a user might use it by typing:

>wget http://gemini.cs.cf.ac.uk/ -o test.htm

Which would download the index web page on the Gemini web server and save it to a file called test.htm in the working directory. Provided a scripting language able to create a string (which it could substitute for the http://gemini.cs.cf.ac.uk or test.htm), and with the function to execute command line instructions, any web page or object could be downloaded to the same directory as the script. Provided that the script was then able to read the file, the scripting language would then have full access to all the data on the Internet, despite the fact that the scripting language itself does not have Internet access capabilities.

As a result, scripting languages can be accurately defined as simple programming languages, with the ability to read (and beneficially, write) data files and execute programs and command sequences through the command line interface. These two attributes form a solid interface through which complex data may flow, and complex operations may be performed, without the additional requirement that the scripting language itself understand the operation of the executed programs.

While the range of scripting languages varies with different operating system platforms, good examples of scripting behaviour are: GNU Lisp, GNU Prolog, Ruby, Python, Perl, PHP, TCL, Bash (Tsch, Csh, etc.) and Awk. All of these are fully capable of addressing the command line, and therefore able to use *'curl'* and *'wget'* as described to interact with and control Myriad components and solutions.

In addition to the common scripting languages mentioned above, there is a schism in the operation of applications that has been created through the emphasis on RAD-developed user interfaces and the attempted deprecation of the command line as seen in contemporary Windows implementations (although this situation is being reversed with the recreation of a new, enhanced CLI named Monad in Longhorn (the successor to Windows XP)). Using Visual Basic, C++/QT, Java and others to construct GUI applications has formed a set of applications which are feature-rich, but sadly unable to be controlled through scripting. Using LabVIEW to construct Visual Basic/C++ applications inherently falls foul of this flaw, as the GUI applications controlling equipment and resources are unable to be easily scripted for automated tasks. There does exist a scripting language for Windows applications, called VB Script, or Visual Basic Scripting for Applications (VBA) which is able to attach to hooks for methods in applications, provided that the programmer adds the hooks in an appropriate manner. The limitations of VBA, however are manifold, since while it does retain a number

of the capabilities of Visual Basic, it itself is not controllable, preventing other (more capable) scripting languages from controlling the resources.

As mentioned above, using the QT toolkit with C++ can also lead to the creation of isolated GUI applications, which are not ideal in a similar way to their VB counterparts. For QT, there also exists a scripting facility, called QT Scripting For Applications (QSA), which acts in a similar manner to VBA, except that it is designed with the intention of providing automation capabilities to users of applications directly, allowing users to build macros and command sequences, where VBA is firmly targeted at GUI programmers themselves. The difference between QT and VBA, therefore, would seem to be, that QT lacks a good range of programmer automation tools and scripting capabilities. QT does, however, include a range of 'bindings' allowing for scripting languages such as Perl to control QT applications and even integrate directly with application operation, by being able to create GUI objects, do processing and liaise with program objects.

8.1.4 Application

As previously explained, the concept of the GUI is isolated from applications here, due to the degree of processing that applications do and the RAD-type development that is becoming increasingly prevalent when creating GUI programs presently. In short:

GUI – Dumb program which just passes commands through from the user to a larger system

Application – Optionally with a GUI, but a program which does large amounts of intermediate processing and may tie processing to other programs, data resources and knowledge bases.

The majority of modern application development (certainly on the more large-scale projects) is performed using C and C++ with associated toolkits, libraries and additional language plug-ins (for languages such as Python for scripting and Prolog for intelligent decision-making). While many of the proponents of Microsoft's .Net and Sun's Java J2EE may believe otherwise, there is very little penetration of virtual machine languages into this area of endeavour. There have been efforts to develop desktop-grade applications using these languages, but the majority of attempts result in GUI-like programs, rather than heavy-processing + GUI constructions that are the realm of modern user programs. While this is not

to say that such languages make it impossible to develop applications (for example, Myriad's prototypes are produced using Java), present commercial and mainstream Open Source software offerings for the conventional desktop are resoundingly C/C++ based. In the last decade, Sun Microsystems has dropped plans to convert the CDE and applications used on Solaris to Java (citing unspecified complexity and difficulties) and attempts to produce applications such as the Mozilla Project's ill-fated Grendel web browser/e-mail client written entirely in Java similarly failed. In point of fact, examining the development of applications under Linux using scripting languages such as Perl, Python and ECMA Script and language bindings to toolkits such as QT, it could be argued that these languages have seen greater success in producing desktop applications than Java.

Interfacing C and C++ applications with Myriad is only slightly more complex than the use of GUI programs or scripts, and this is primarily due to the wide array of toolkits and libraries that are presently in place, producing a lack of a definitive way in either language to access the Internet and do HTTP activities, by language specification. What is lacking in those areas, however, is made up for with the abilities of most toolkits themselves, such as QT, which specifically defines a QHTTP object, which is designed for producing HTTP requests and downloading content. It is, however, up to the individual programmer to research their own toolkit of preference, or add a network-aware library in order to provide access to Myriad systems. Almost all competent programmers of applications, however, should find no problem in using Myriad components, as it is generally a simple case of knowing an API, passing a request string to it and calling an execution of the request.

To take an example of an e-mail application; the application could be extended with a directive to divert messages to a Myriad image processing server on a corporate LAN, examining e-mails as they arrive, processing images embedded at the top of the e-mail, denoting the department from which the e-mail has been sent and containing a verifiable visual stamp to prevent automated spoofing of senders, to screen out junk e-mails and maliciously-falsified information.

While this example is not highly practical in that a unique text identifier might be used in place of an image, for more efficient sorting, it serves to illustrate the ease with which image processing functions may be added to conventional applications for specific tasks, while the problems themselves are most certainly not Machine Vision tasks. The integration of MV

tools into such mainstream tasks, where the programs and the tasks are not inherently MV-focused is one of the key benefits of Myriad systems, and offers a strong potential for Machine Vision to grow and explore new areas of deployment.

8.2 Language Examples

8.2.1 Java

The Java code required to interact with a Myriad server is based around the creation of an InputStream based upon a URL, which is used to form a URLConnection, and then passed through to an object which handles input streams.

```
String toDownload = new String(http://www.myserver.com/);
toDownload = URLEncoder.encode(toDownload);

URL URLtoDownload = new URL(toDownload);
URLConnection URLconnect = (URLConnection) URLtoDownload.openConnection();

InputStream URLinput = URLconnect.getInputStream();
// Data stream manipulation here

URLinput.close();
```

This source code takes a string containing the URL, encodes it into the URL character set (correcting special characters, etc.), forms a URL object, which is then used to create a URLConnection, an object encapsulating all the methods required to manage a connection to an HTTP server. One of the aspects of a URLConnection object, is that it contains a sub-object which is an InputStream. This can be extracted and used as a conventional Java InputStream object would.

The classes involved in this process may be included in any application with the addition of a single line to the start of the Java source file:

import java.net.;*

All of this is standardised code, and able to be reused by only changing the URL included in the string at the start, and utilising the InputStream at the end, if examining feedback from Myriad servers. If the intent is simply to initiate a process (e.g. Turn on lights), the

InputStream may be unused. For hard-coded operations, with no feedback and fixed, pre-encoded URLs, therefore, the code may be further refactored:

```
URL URLtoDownload = new URL("http://www.myserver.com/?commands=lightson");
URLConnection URLconnect = (URLConnection) URLtoDownload.openConnection();
```

A complete function would be:

```
public void lights_on()
{
URL URLtoDownload = new URL("http://www.myserver.com/?commands=lightson");
URLConnection URLconnect = (URLConnection) URLtoDownload.openConnection();
}
```

8.2.2 C++/QT

The C++ code, when using the QT toolkit is relatively similar to that of the Java in structure and use, although the object names and detailed syntax vary.

```
qInitNetworkProtocols();

QString url = QString::QString(http://www.myserver.com/?commands=lightson);
QUrl::encode(url);

QUrlOperator *op = new QUrlOperator();

op->copy( url ,QString("output.txt"), FALSE, FALSE);
```

In this code, we see the same distinct core segments of code that were present in the Java version:

1. Creation of a string to contain the URL
2. Encoding of the URL string to the correct character set
3. Creation of an object that will manage the connection
4. Execution of a method in the managing object to connect and request an HTTP file

Whereas in Java, the *URLConnection* class is used to manage a connection, in that it connects, and then waits for the user to request the data stream to obtain the data for manipulation, QT's *QUrlOperator* operates with a slightly different philosophy; it assumes that any use of it is with an intention to obtain data, and as a result, contains methods that not only make a connection, but also manipulate the data stream in one action. E.g. to obtain a data stream:

Java

1. Connect *URLConnection* to the HTTP server
2. Obtain a data stream from the *URLConnection*

QT

1. Obtain a data stream for a *QUrlOperator* (implicitly connection to server)

This QT methodology operates under the reasonable assumption that one does not wish to acquire a data stream to an unconnected object (since that can logically provide no data, and Java will generate an error if this were attempted), and thus, the majority of URL Operator actions will implicitly need a connection to be made; so in those cases, *QUrlOperator* makes the connection and saves the programmer an additional object creation and other lines of code, which could introduce further error.

The method of *QUrlOperator* used in this case, is *'copy'*, an operation which downloads an HTTP file and copies it to the local storage. *'copy'* accepts a string for the URL (not a dedicated URL object, which Java requires (saving another object instantiation and casting operations)), a string for the file name to copy the data to, and two flags denoting file saving details (overwrite existing files, etc.).

Finally, it should be noted that the first line of the code (*qInitNetworkProtocols();*) is required to announce to QT that network protocols such as HTTP and FTP are to be used; this line need only occur once in each source file, but prior to use of these features.

As with the Java, QT also requires that library declarations be included in the header of the source, of the form:

#include <qurloperator.h>
#include <qnetwork.h>
#include <qurl.h>

QT requires a declaration for each QT class that is used in the source, with the libraries named for the class name in lower case, completed with .h denoting a C-style header file. While this

does lead to slightly longer declarations than Java, it also means that developers know the exact names of the libraries they wish to use (rather than searching and guessing in Java, where URL is actually in the java.net library as java.net.url) and do not need to import large numbers of unused classes.

8.2.3 Perl

Using Myriad components in Perl requires the use of external applications, such as wget or curl, accessed via the system command line. While Perl does have a network library of its own, this method is clearer and far more concise.

```
#!/usr/bin/perl
system 'wget http://gemini.cs.cf.ac.uk:3080/?script=myscript.ips';
```

The first line of the script states to the operating system the interpreter which should be used to evaluate the script.

The second line runs a system command (located between the two backticks), which runs the wget program to attempt to download the file of the URL.

This is all that is required for Perl to be able to interact with a Myriad server. A more advanced form might build the URL first, adjust wget to output to a specified file and proceed to process the file afterwards:

```
#!/usr/bin/perl
$url = "http://gemini.cs.cf.ac.uk:3080/";
$url .= "?script=myscript.ips";
system `wget -q -O output.htm $url`;

open INFILE, "output.htm";
while(<INFILE>)
{
print $_;
}
close INFILE;
```

In this code segment, the string $url is initialised with the value of the server and port number, then concatenated with the query itself using the operator ".=". The modifications to wget will be explained in the examination of the PHP version of this code (sufficed to say, they force output to a file called output.htm).

Opening of files in Perl is described in the subsequent lines, with the downloaded file opened and attached to the file handle *'INFILE'*. The while loop, with the file handle named in angled brackets is a syntax which performs the loop per line of the file and removes the necessity of the programmer testing for the end of the file and reading in data either character or line-wise. Using this while structure, the line being read by the loop is implicitly injected into the special variable named *'$_'*, which may then be used in the same way as any other variable in the language. Thus, a loop of *'print $_'* merely outputs each line of the file to standard output. The programmer need not worry about whether the *End Of File* is reached, or whether the loop will be executed and *$_* will have a null value at the end of the loop, as the Perl while command takes care of these details.

8.2.4 PHP/HTML

PHP is most frequently used to provide the ability for HTML to be used with dynamic data, and thus be able to respond to the user, or information that is provided from another data source, such as a database. There are two methods that PHP developers may use in order to interact with a Myriad server using HTTP. They may either use wget or curl to download data directly, or they may inject a Myriad URL into a piece of HTML that is executed when the completed document is viewed in a web browser (this would be most common with data driven web sites where Myriad image data is to be merged into HTML documents for a web browser user).

PHP & curl/wget

In cases where PHP programmers wish to obtain data from a Myriad server and use that data for additional processing prior to output, e.g. to visually find the objects in a scene and then obtain additional background information on them from a database, presenting the complete

document to the client upon completion, it is essential that PHP itself obtain the HTTP data and process it. In order to do this, it is possible to use the PHP Network modules to download data, but far easier to simply use a dedicated program such as *wget* or *curl* to download the data to a file via the command line and read the file upon completion. To access system commands in PHP, the *passthru()* command is utilised, requiring only a string containing the text that would be used were a person performing the same action at the command line. E.g.

```
<?php
passthru('wget –q –O output.htm http://www.myserver.com/');
?>
```

Which is equivalent to:

- *wget –q –O output.htm http://www.myserver.com*

Where the form of wget is:

wget <flags> <url>

The flags in use are:
-q : suppresses output and error messages
-O <filename> : specifies the file to save the data to

Aside from wget, curl is a command line program that operates in an almost identical manner, with almost identical syntax. The curl equivalent of the above wget command is:

curl –f –o output.htm http://www.myserver.com

HTML Browser / Myriad Interaction

HTML, although a static language, may be made into a basic Myriad controller, since HTML is effectively a collection of the following elements:

- Text
- Text mark-up tags
- Links
- Imported HTTP objects

In this range of elements, Myriad data and actions can be performed through the use of HTTP requests produced by HTML image tags and script requests. For example, suppose that an image needs to be downloaded by a Myriad IP server and then processed, before being returned to the web browser client. This may be done using conventional HTML, without any additional dynamic processing; e.g.

```
<html>
<head></head>
<body>
<img src="http://www.myserver.com/output.jpg?commands=neg">
</body>
</html>
```

This is an incomplete Myriad script (so as not to confuse the demonstration), but it does illustrate that when a web browser attempts to read the HTML page, it will request the image from myserver.com and pass along the parameters that a Myriad server may use to process the image prior to returning it to the browser.

The HTML script tag may also be used to summon an arbitrary HTTP request to a server, initiated by the web browser.

```
<html>
<head></head>
<body>
<script src=http://www.myserver.com/?commands=lightson></script>
</body>
</html>
```

Which is similarly useful for text-based output from Myriad servers, or operations that require no output.

Both of these cases, of course, may be used in conjunction with PHP, so that PHP provides dynamic URLs in spaces in output HTML, and the web browser, when receiving the complete page, goes on to contact the Myriad servers and perform the requests.

8.2.5 VB

Operating Myriad servers using Visual Basic is a similar process to the Java and QT versions; a URL object is created first and passed to an object which proceeds to create the connection and collect the data.

```
Private Sub Command1_Click()
Set inet = CreateObject("khttp.inet")
Dim url
url = "http://www.myserver.com/?commands=start"
returnString = inet.OpenURL(url)
Text1.Text = returnString
Set inet = Nothing
End Sub
```

In this example, the inet object is created as an instance of *'khttp.inet'*, and a URL is also produced and assigned the string of the URL. The OpenURL method is then run with the URL and the result stored in returnString (an arbitrary variable), which is output by applying it to a text box (*Text1*). Finally, the http object pointer inet is deleted, leading Visual Basic to eliminate the object itself, as it has no connections to it anymore.

The khttp.inet class used in the example is a third party class produced for advanced HTTP access with Visual Basic 4, 5 and 6, attempting to overcome the deficiencies of the in-language libraries. The language version is named inet and provides http and ftp commands methods. Unfortunately, it has problems with HTTP 1.1 compliant servers, does not support HTTPs, has limited POST variable support, no encoding of binary POST data and tends to disconnect from servers prematurely, only downloading partial data. As a consequence, third party libraries are essential to the creation of Internet aware applications in Visual Basic.

Microsoft's Visual Basic .NET language revision is a version of the VB language rebuilt to operate atop the .NET Common Language Runtime (CLR), essentially a Virtual Machine in the Java mould. With the express expectation that VB.NET would be used to develop web services and network-aware applications, the network commands are more complete than those of the previous generation of the language, and although the author has not yet had the opportunity to test these capabilities, reports suggest that a similar process to the one described above is applied in order to download web data, and thus communicate with Myriad servers.

8.2.6 C#

With the creation of the .Net revisions of Visual Basic, Visual C++ and Visual J++, Microsoft introduced a new programming language, largely modelled on a combination of C++, Java and Visual Basic, named C# (pronounced 'C sharp'). Most Microsoft-focused programmers see it as the successor to Visual Basic 6, more so than VB.Net; C# has a cleaned up syntax, more intuitive syntax and a visual GUI designer identical to that of Visual Basic. The .Net architecture ensures that objects written in any .Net language may be used within any other and used as if they were native, so there is no compatibility issue with switching between the two languages.

Being GUI-centric, to create an application to manipulate a Myriad server, the programmer starts by opening Visual Studio, the IDE and selecting 'Visual C# projects'->'Windows Application'. The form is then generated, and a button can be selected in the toolbox and visually dragged out. Double-clicking on the button opens a code view for the button's 'Click' event, to which the following code can be added:

```
private void button1_Click(object sender, System.EventArgs e)
{
string urlString = "http://www.myserver.com/?commands=lightson";
HttpWebRequest httpRequest = (HttpWebRequest) WebRequest.Create(urlString);

// *** Retrieve request info headers
HttpWebResponse webResponse = (HttpWebResponse) httpRequest.GetResponse();
Encoding enc = Encoding.GetEncoding(1252);
StreamReader responseStream = new StreamReader(webResponse.GetResponseStream(),enc);
string outputHtml = responseStream.ReadToEnd();
}
```

The layout of this code is very similar to that which has been previously seen in the other programming languages:

- A string with the url is declared
- A request is created (*HttpWebRequest*, which encodes the string into UTF-8 automatically)
- The request is executed and the server response object is collected (*HttpWebResponse*)
- In contrast to other languages, C# requires that a character encoding method be explicitly declared (due to the limited internationalisation support in Windows)
- The server response object generates output stream with the returned data from the HTTP request. This is assigned to a *StreamReader* object, which provides advanced manipulation methods for the stream
- The data from the stream is assigned to a string object, ready for use

With this code, whenever the button on the GUI is clicked, a request will be sent to the (what we assume to be equipment control-) server to turn on whichever lights are connected.

In order for this code to operate, the libraries to be used have to be declared at the top of the source code, by including:

```
using System.Net;
using System.IO;
using System.Text;
```

For fast creation of basic control applications, however, if no feedback from the Myriad server is required, the code can be further reduced to:

```
private void button1_Click(object sender, System.EventArgs e)
{
HttpWebRequest httpRequest = (HttpWebRequest)
       WebRequest.Create("http://www.myserver.com/?commands=lightson");
HttpWebResponse webResponse = (HttpWebResponse) httpRequest.GetResponse();
}
```

Two lines of source code within the template created by the GUI designer enables control of a Myriad server! A GUI control application for operating (basic pan, tilt & zoom operations) the Sony pan & tilt camera described in chapter 3 could be produced (arranged, coded,

compiled) by a programmer within ten minutes of opening the Visual Studio development environment. All the programmer has to do is drag out buttons onto the GUI designer, copy and paste the code and change the Myriad URL to include the appropriate commands.

With minor changes to the included libraries, C# code can be recompiled to operate on Microsoft Pocket PC devices (provided they have the .Net runtime environment installed), making the creation of a camera control application trivial for a PDA equipped with a wireless LAN connection.

Whilst Myriad servers are useful programmable network-interfaced systems for machine vision tasks, they are general purpose tools and unable to act without first being initiated by either local or remote control. That control may come in the form of command sequences, single command or direction to execute a script; once that initiation occurs, however, MCS servers are able to act autonomously or in an interactive fashion. Hence, the user requires guidance as to how to best communicate with a Myriad server to begin operations. This chapter outlines modes of interaction and sample code for applications, scripts, lightweight GUIs and directly, through web browsers.

The most common contemporary programming and scripting languages were addressed:

- Java
- C++/Qt
- Perl
- PHP/HTML
- Visual Basic
- C#

---

9 Examples

Networked machine vision solutions operate in a very different manner to enclosed, stand-alone systems, and frequently apply to different types of problems than conventional systems. While this does enable the application of machine vision techniques to new avenues of investigation, it also requires illustration to gain an understanding of the methods of operation and data flows that are produced by a networked solution. In order to provide the reader with a fuller understanding of the functionality of networked vision systems, and Myriad as an implementation, therefore, this chapter describes a number of example problems and applications.

9.1 Linpacker

The following example is derived from the paper "2D Packing Using the Myriad Framework", L. T. Chatburn & B. G. Batchelor, presented to the SPIE Optics East conference in 2003. Written by the author and edited by Batchelor, it is present here with minor alterations for brevity.

9.1.1 Introduction to the Problem

In the context of this example, the term "packing" also refers to cutting (also called *depletion*), where we are required to "pack" holes as densely as possible into a given piece of laminate material, or a 3D space. Packing problems occur throughout industry, in such areas as manufacturing/using carpet, window glass, sheet metal, clothing, foot-ware, food products, etc. While packing arbitrary 2D and 3D shapes is extremely difficult and time-consuming, even the seemingly straight-forward task of packing rectangles can be very difficult: imagine having to pack several hundred rectangular objects of random sizes and aspect ratios. There are numerous classes of packing problem, reflecting the fact that there are many different sets of application constraints, arising in different industries. [BAT] For example, in certain cases, it is not permitted to rotate the shapes (patterned fabric), or to rotate them only by integer multiples of 90 degrees (leather). Furthermore, it is necessary, in some instances, to pack so that unpacking can take place automatically, using a robot with "fat fingers". In some cases,

such as processing leather, it is necessary to cut some shapes from certain parts of the material supplied, for example where the leather is supple and has a fine grain. Algorithms for packing rectangles can be modified slightly to allow arbitrary shapes to be put in place; we simply replace each shape by the minimum-area rectangle surrounding it. (Figure 72) The result is often inefficient but, for compact, nearly convex shapes, this approach can sometimes produce *adequate* results. Bearing in mind this comment and the fact that this chapter is concerned primarily with demonstrating the application of Myriad, we shall concentrate on packing rectangles in a 2D space. 3D packing is beyond the scope of this paper and will not be discussed further.

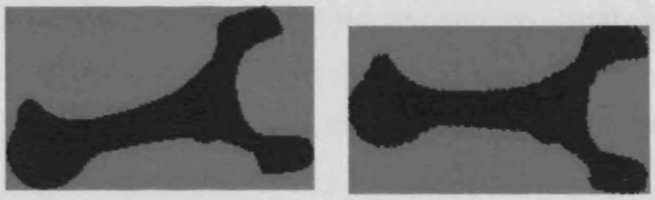


Figure 72. **Representing an arbitrary 2D shape by the minimum area rectangle, prior to packing. During packing, no other shape is allowed to intrude into the grey area. Left: rotation is prohibited. Right: rotation is permitted. The object has been rotated so that the axis of minimum second moment ("principal axis") is horizontal. Notice that this rectangle is smaller, along both axes, than that shown to the left. This results in more efficient packing.**

Packing is an exemplary problem in Artificial Intelligence; it is difficult to perform an exhaustive search for solutions, except in the simplest cases. It is perhaps less obvious that Machine Vision is also relevant to packing. Packing rectangles into a rectangular container can be performed without resorting to image processing. However, it is easier to understand the process, if we represent it visually. When packing irregular blob-like shapes into an irregular container, for example, when cutting leather from an animal hide, it is essential to represent them both, in the form of binary images, or some convenient coding for binary images, such as Chain Code. Packing 2-dimensional shapes therefore involves a range of image processing algorithms for shape analysis and avoiding "collisions", typically affine transformations (linear shift, rotation, scale change), parametric representation of shape and a variety of others (e.g. minimum-area rectangle, circumcircle, convex hull, etc.).

The important point to note is that packing relies heavily on image processing techniques and, for this reason, it was considered to be an ideal application for demonstrating Myriad. Packing

was chosen for this purpose because it illustrates the inherent simplicity of combining Machine Vision techniques and other computational resources through Myriad.

9.1.2 Packing in Industry

Packing of 2-dimensional objects occurs in numerous different guises in manufacturing industry. Here are just a few instances:

- Cutting leather to make shoes, gloves, clothing, bags, etc.
- Making sewing patterns that minimise the fabric used.
- Packing chocolates, cakes, fruit vegetables, etc. onto trays, ready for retail. (These are all somewhat variable in size and/or shape. [GRA])
- Retailing carpet that is cut to fit the customer's floor space, without creating undue waste.
- Packing and stacking flexible sacks of produce.
- Cutting laminate material in a flexible manufacturing cell.

In this context, multi-axis manipulators may be used but there are many situations in which programmable flat-bed cutters (milling machines, jig-saw, laser or water-jet) are used instead. For our purposes, these are all "robots" and require a computer to act as a controller. They require the definition of "templates", which may be derived from a computer model, or using vision, from a piece of laminate material that may even have been cut/sketched by hand. It is important to realise that the robot and its host computer may not be close to the template-generation system. In other words, remote control via a network, as is provided by Myriad, is essential. It also offers a number of additional benefits:

- Template designs and packing solutions can both be stored at corporate head-quarters, in databases providing company-wide access.
- Expensive, sophisticated equipment may be used to service multiple sites, thereby reducing both capital expenditure and labour costs, such as maintenance crews.
- Packing results may be tracked over time and all company sites, enabling trends in manufacturing to be recognised
- Objects designed at one site may be cut elsewhere and finally packed at a third location.

To summarise, the provision of a network-based packing system offers real prospects for improving manufacturing efficiency and management control. Myriad has been designed as a prototype for such a system.

9.1.3 Additional Technologies

KDE

Unix operating systems typically run a graphical X11 server and then a desktop environment above that. In the case of Solaris, the CDE, for example. The K Desktop Environment (KDE) is a similar case for Linux. Written using C++ and the QT toolkit, KDE is fully functionally comparable to other desktops, such as Windows or MacOS. Due to the monetarily free nature of Linux and the KDE, the extensive internationalisation and the ability of users to customise their own systems through access to the source code, the KDE is seeing strong acceptance in developing countries, corporations and governments.

The KDE project has developed extensions to QT, through the need for additional widgets and features, such as complete network transparency to applications and the RPC/messaging system, DCOP.

DCOP

CORBA has been used for communication and control between applications and systems proved very early in KDE development (version 1, now at version 3.5) to be slow and unwieldy. As a result, DCOP was created as a lightweight communications protocol between processes. DCOP operates as a server, where client applications announce the functions they provide and which subsequently controls and directs messages. Applications or scripting languages (such Perl, Ruby, Python, shell scripts) may then send commands to applications via the DCOP server, directly accessing the object methods that the author has chosen to expose, thus accessing a subset of their functionality. DCOP may only be used locally within a computing device, as the ICE library, upon which it is based, is unable to marshall commands across network interfaces. DBUS, the successor to DCOP has no such limitation and also has hooks in the Linux Kernel to support signal emission by hardware devices and control of hardware.

DCOP is used as our means to control the packing application, LinPacker.

LinPacker: An On-Line Packing Program

LinPacker is a program for packing rectangular objects. It was written by Thomas Nagy and began life as a personal project. It was later expanded into a small Open Source project. In order to fit it into the Myriad framework additional input/output routines were written by the author. LinPacker is written in C++, using the QT toolkit. The application has lost cross-platform capability, with the addition of DCOP facilities which force it to presently run on systems provided with the KDE desktop environment. However, it can be controlled from Myriad scripts and other programs via DCOP, which was essential for this project.

As a result of the DCOP integration into LinPacker, scripts and other applications may start and operate LinPacker in an automated fashion (see fig.73), adding input, managing processing and post-processing or directing the output, transparently to the user of the collected system. Using this mechanism, LinPacker, while initially designed to be a primarily GUI-based application, in point of fact, becomes a black box packing system that may be used by any other program, allowing those other programs to take advantage of 2D packing as a tool, without having to develop their own algorithms or interface with professional-grade packing libraries for simple problems. This Myriad example is also an apt example of the simplicity of using Inter Process Communication (IPC) such as this to incorporate complex functionality.

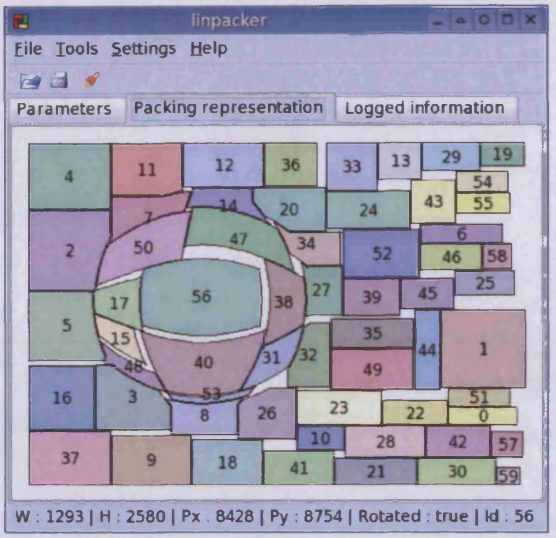


Figure 73. LinPacker packing display being inspected with fish-eye zooming accessibility feature

2-Dimensional packing is an interesting problem for a Myriad example, because it combines 2D forms, which can be discovered visually, and employs a degree of complex computation, with the potential to physically pack afterwards. Normally, this sort of demonstration would examine objects using a camera connected to a computer located next to the robot, eliciting some form of movement. In this example, however, we choose to examine a networked situation, where the objects to be packed are actually in the view of a camera which is remote to the site of the packing (see fig.74).

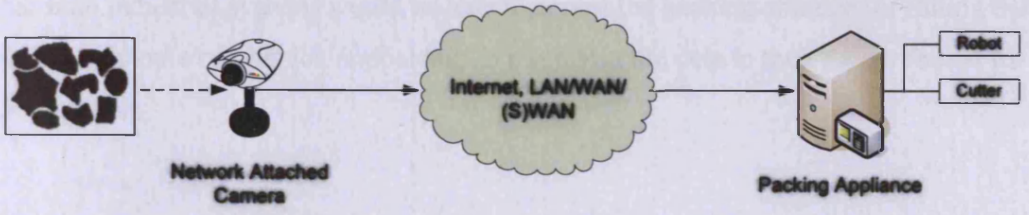


Figure 74. Network attached camera capturing an image of packing shapes, transmitted to a remote packing appliance able to cut or pack similar shapes using a robot

Moreover, whilst this would be a reasonable demonstration of networked image capture, it does little to assist the understanding of networked processing systems. As such, the system is adjusted further, such that the actual device performing the packing computation is remote to the computer that initiated the operation and requires the packing solution for employment with the robot/cutter (see fig.75).

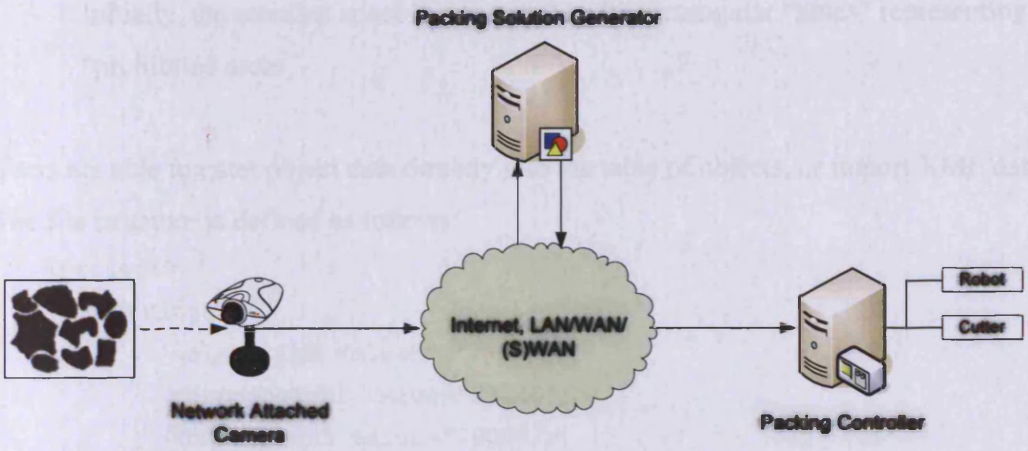


Figure 75. A networked packing system, where the packer initiates operations, but packing solutions are created remotely

While the author does not have access to a robot or cutting equipment that would be used in an industrial situation, such systems are often highly customised to their task in terms of the way in which they are operated and the controlling software provided to manage their use. In this case, therefore, the focus of the demonstration is not to provide an end-to-end packing system with a physical output, but to expect a packing solution presented in such a form that it may be printed using a conventional printing device, transitioning to the format of PostScript. Using PostScript as an output format, the vector-based design of the format allows confidence that final industrial systems would be able to accept the packing solution, or failing that, be able to employ a conversion application to transform the data to their native format for cutting or operation.

9.1.4 Inputs

LinPacker is designed around two essential components:

- A table containing the data (width, height, rotatability) describing the rectangular objects that are to be packed. (This is called the *Objects Table*.)
- A container into which we can place them. This might for example, be a simple rectangle. The packing area can be customised in size through a dialog box. Rectangular "prohibited areas", where objects may not be placed, can be specified by the user specifying both size and position. (We shall call the space within the initial container that is currently available for packing further objects the *remnant space*.)

Initially, the remnant space is a rectangle with rectangular "holes" representing the "prohibited areas".

Users are able to enter object data directly into the table of objects, or import XML data files. The file structure is defined as follows:

```
<project>
  <settings>
        <algorithm value="1"/>
        <bandeheight value="23040"/>
        <bandewidth value="7000"/>
        <nbgenerations value="250"/>
        <nbelements value="1000"/>
        <mutations value="5"/>
        <crossovers value="70"/>
        <useconstraints value="false"/>
  </settings>
  <objects>
        <object>
              <width value="4000"/>
              <height value="4000"/>
              <posx value="0"/>
              <posy value="0"/>
              <num value="16"/>
              <orientation value="1"/>
              <order value="16"/>
        </object>
  </objects>
  <constraints>
  </constraints>
</project>
```

Where data, such as settings, orientation of packing order, etc. are not available, these items may be omitted.

Packing Algorithms

LinPacker currently contains three algorithms for packing, with options for iteration, seeding and search-tree formation:

- Basic packing (fewer than ten objects)

- Genetic algorithm
- Complex packing with constraints.

Once the shape and container parameters are set, packing begins. The results are shown on a canvas beneath the object table. Randomised colour coding and numbering of objects is provided to help the user identify the packing results. When the process has been completed, the *Objects Table* is modified to include the position and orientation parameters that have been calculated for each object.

9.1.5 Output

LinPacker can generate output data in two ways:

- *Direct printing.* This takes place in the conventional manner. For example, a postscript driver can produce a paper template for cutting systems. Alternatively, alpha-numeric code can be created, to operate a robot.

- *Data export using XML files* This allows packing results to be made available "on-line" to CAD/CAM systems. While the data formats for the latter may vary, XML may be converted to other formats, using a variety of languages including Perl, Java, PHP or C++ with QT. This would allow the LinPacker-Myriad system to be integrated with existing manufacturing equipment and business management systems..

9.1.6 Operating Myriad and LinPacker Together

Controlling Myriad and LinPacker

There are two options available for system construction:

a) The Myriad IP server acts as the controller operating LinPacker
b) Another program operates as the overall controller and organises both the Myriad IP server and LinPacker.

Notice that LinPacker cannot control the system, although it would be possible to make a small modification to LinPacker to allow it to do so. However, this would create inconsistencies with the version of LinPacker presently through the Internet as the official edition. Additionally, as it provides greater clarity in terms of ease of explanation and understanding over the first method; hence Method (b) is preferred.

Communicating with LinPacker

LinPacker is limited in its control abilities compared to other Myriad components. However, by using DCOP, clients, it is possible to gain access to a number of internal functions. The functions specifically useful in this task are as follows:

- *Solvepacking(datafile.lnpk)* Loads the data file named by the client into the rectangles table and runs a packing operation on it.
- P*rintimage(datafile.lnpk, image.jpg)* Prints the packing solution currently rendered on the canvas.

LinPacker can save packing solutions in data files, although it is not yet possible to publish them via DCOP. (This facility will be added in the near future.) To access the DCOP functions, the control application must first start a DCOP client and the application to be controlled. It is then possible to send simple messages, or calls which return data, through one of the interfaces that the latter provides.

Scripting Myriad and LinPacker

The processes for control of the two elements of the Myriad-LinPacker system using a Perl script are straightforward; just a few lines of code needed for each. To operate the IP server, *wget* is used to download a web file from the server. The URL is then passed to *wget* with the commands embedded as an argument. Using the example above, the resulting command would add a single line to our control script:

system 'wget http://www.mymyriadserver.com/test.jpg?commands=start';

In order to perform the packing, the Myriad server requires a script, telling it to grab an image from a network camera, process it, extract each of the binary objects, then find the minimum-area rectangle for each one and finally save a data file:

```
downloadimagefile myimage http://gemini.cs.cf.ac.uk/eyeimage.jpg
sed                                 % Sobel edge detector.
thr 16                              % Threshold at level 16.
blb                                 % Fill any holes" in the "blobs".
store blobs                         % Save resulting binary image.
$x = 0                              % Set variable $x to zero.
```

```
cbl $y                                % Find no. of blobs. Store in $y.

openfile objectsfile datafile.lnpk  % Open a file
printlnfile objectsfile <project>   % A bit of output for the user
printlnfile objectsfile <objects>   % A bit more output for the user


while ( $x <= $y )                    % Loop, to process each blob
{
$x += 1                               % Increment the loop counter
use blobs                             % Reload stored all-blobs image
big $x                                % Find blob of rank size $x
crp                                   % Crop image to fit the blob
saveimage ${x}.jpg 100 jpeg
getwidth $width                       % Width of minimum-area rectangle
getheight $height                     % Height of min-area rectangle

printlnfile objectsfile <object>    % Even more output for the user
printlnfile objectsfile <width value="${width}"/>
printlnfile objectsfile <height value="${height}"/>
printlnfile objectsfile </object>

}

printlnfile objectsfile </objects>
printlnfile objectsfile </project>
closefile objectsfile
```

All of this script is gathered together into a rather lengthy URL to be used by *wget*:

http://gemini.cf.ac.uk:3080/?commands=start%0D%0Adownloadimagefile+myimage+http%3A%2F%2Fgemini.cf.ac.uk%2Feyeimage.jpg%0D%0Ased%0D%0Athr+16%0D%0Ablb%0D%0Astore+blobs%0D%0A%24x+%3D+0%0D%0Acbl+%24y%0D%0Aopenfile+objectsfile+datafile.lnpk%0D%0Aprintlnfile+objectsfile+%3Cproject%3E%0D%0Aprintlnfile+objectsfile+%3Cobjects%3E%0D%0Awhile+%28+%24x+%3C%3D+%24y+%29%0D%0A%7B%0D%0A%24x+%2B%3D+1%0D%0Ause+blobs%0D%0Abig+%24x%0D%0Acrp%0D%0Asaveimage+%24%7Bx%7D.jpg+100+jpeg%0D%0Agetwidth+%24width%0D%0Agetheight+%24height%0D%0Aprintlnfile+objectsfile+%3Cobject%3E%0D%0Aprintlnfile+objectsfile+%3Cwidth+value%3D%22%24%7Bwidth%7D%22%2F%3E%0D%0Aprintlnfile+objectsfile+%3Cheight+value%3D%22%24%7Bheight%7D%22%2F%3E%0D%0Aprintlnfile+objectsfile+%3C%2Fobject%3E%0D%0A%7D%0D%0Aprintlnfile+objectsfile+%3C%2Fobjects%3E%0D%0Aprintlnfile+objectsfile

+%3C%2Fproject%3E%0D%0Aclosefile+objectsfile%0D%0Astop

Optionally, we can save the script to a pre-defined text file, located on the server, and call it instead using the following URL:

```
http://gemini.cf.ac.uk:3080/?script=myscript.ips
```

Using LinPacker requires a few more lines of code but these are also quite straightforward:

```
use DCOP;                //use the dcop module
my $client = new DCOP;   //create a new dcop client object
$client->attach();       //connect to the dcop server

my $linpacker = $client->createObject("linpacker", "sinterface");
//start a new linpacker process and use the 'sinterface' interface

$linpacker->solvepacking(datafile);       //call the linpacker launch
(pack) method
```

In total, therefore, the actual control script to run both Myriad and LinPacker totals six lines of code (excluding comments and the Perl header).

The final script is:

```
#!/usr/bin/perl
use DCOP;
my $client = new DCOP;
$client->attach();
my $linpacker = $client->createObject("linpacker", "sinterface");
system 'wget http://gemini.cf.ac.uk:3080/?script=myscript.ips';
$linpacker->solvepacking(datafile.lnpk);
```

9.1.7 Combined system

The complete visual packing system was developed quickly, using a minimal amount of "programming glue". Myriad and LinPacker scripting as well as the communications facilities provided by DCOP make this possible. While a Perl script has been used, it would be a simple matter to add code directly to LinPacker to control the IP server before packing begins, which would be the typical construction of a system seeking to add Machine Vision methods to a

pre-existing solution or application. For example, should a manufacturer of a packing system choose to extend the system for their new and improved product with remote visual object acquisition, they would seek to add a button to the LinPacker user interface which would send a request for image capture and processing to a Myriad server. From this point, the data would then be output by the Myriad component in response to this request, and that data would then be loaded into the packing application using the existing loading mechanism. This model would be appropriate for many systems and may, in fact, require fewer or equal lines of code to the Perl script given above.

System architecture

The architecture of the final visual packing system is rudimentary; there are just three elements: the Myriad IPE, LinPacker and the Perl script that controls them. In this small hierarchy, the server also has a child: the network camera being used to capture the image (see fig.76).

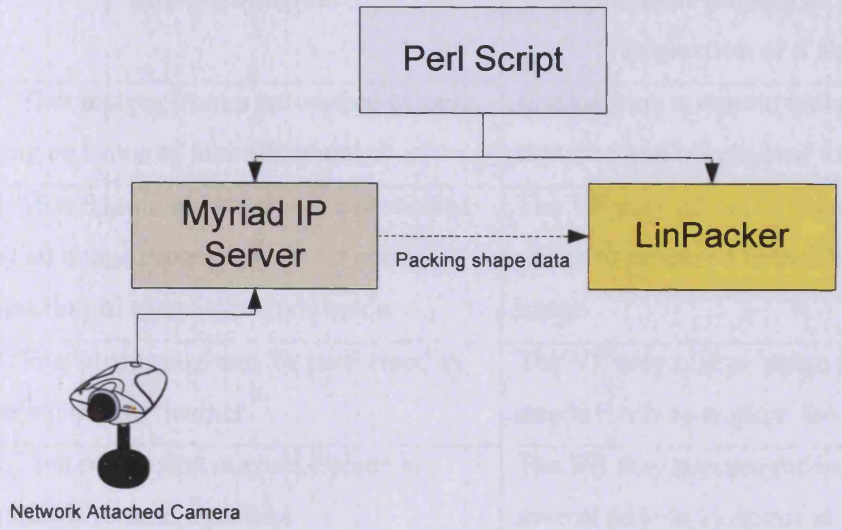


Figure 76. Block diagram of demonstration packing system

9.1.8 Operation

The visual packing procedure operates as follows:

1. Perl script send request to the IP server
2. The IP server collects image from camera
3. The IP server processes image, extracts objects and saves results

4. The IP server returns the request to the Perl script, indicating completion
5. Perl script sends DCOP message to LinPacker to start packing the saved data
6. LinPacker loads the data saved by the IP server
7. LinPacker performs the packing operation
8. LinPacker prints the packing solution

9.2 Train Set Inspection & Control

Due to the nature of the problem of remote installation and inspection of factory environments, and the limited access for demonstration involved, it is essential to be able to demonstrate the component aspects of the solution, and through that means, to be able to illustrate the validity of the combined system. To do this, the following sub-objectives must be successfully demonstrated:

Sub-Objective	Equivalent Objective for Remote Inspection of a Factory
1. That images from a networked camera may be captured and manipulated	Images from a remote factory may be captured and transmitted to the VE
2. That remote images may be processed by an image processing server and meaningful transformations made	The VE may utilise an image processing server to process a remotely-acquired image
3. That processing may be performed in an interactive manner	The VE may utilise image processing interactively to explore the remote scene
4. That processing may take place in multiple remote locations	The VE may process the images in several remote locations at the same time, examining the scene for different characteristics
5. Multiple cameras and locations may be monitored	The VE may monitor several cameras in the same area of the factory to obtain multiple viewpoints, or monitor different areas of a factory for comparison
6. That user interfaces may be utilised on	The VE may perform the exploration task

mobile computing devices	whilst using a mobile computing device
7. Remote equipment may be controlled through lightweight GUIs	The VE can control equipment in the remote factory to affect the scene and measure response

If these objectives can be met, the demonstration is sufficient to illustrate that the Vision Engineer is able to use one or more cameras placed in an industrial environment to explore the scene, assisted by image processing methods and able to affect the scene through controlled remote equipment in order to gain a greater understanding. Through the use of Myriad, the ability to use multiple cameras and multiple processing streams at the same time is demonstrated. In addition the ability to create GUIs quickly to enable control in a variety of manners to suit the problem (such as the VE requiring the use of mobile devices whilst on the move) is illustrated.

To best demonstrate the use of federated Myriad components and lightweight user interfaces for remote examination of an inspection environment, we shall utilise a train set built within a Flexible Inspection Cell (FIC) [BAT01] (see fig.77); reasons for this choice are manifold:

- A train set is very cheap compared to custom-built production line sections
- It is compact, fitting easily within the confines of the laboratory and FIC
- It may be controlled through simple PWM power application
- Control may be: speed, direction, path
- The train my be loaded and unloaded to simulate AGV systems
- Electrical hardware requirements are minimal

In using the train set and a FIC (see fig.78), a production line situation can be simulated with relative ease. System motion may be controlled (supposing that the Vision Engineer planning an MV system deployment is able to control the industrial equipment in the problem scenario) and lighting (enabling the VE to vary the light levels and directions in exploring the situation). With the support of a local operator to discuss the situation and assist with issues such as assessment of dust levels (white glove tests, etc.) along with multiple cameras or a pan-and-tilt camera, the Vision Engineer is able to both examine the scene and also interact with it to establish environmental details.

Figure 77. Train set built within a Flexible Inspection Cell (black-painted steel framework surrounding image)

The train set is built as two circular loops with connecting straight sections between the outermost edges. Transition between the loops is controlled by a set of points where each straight section meets a loop, totalling four sets of points.

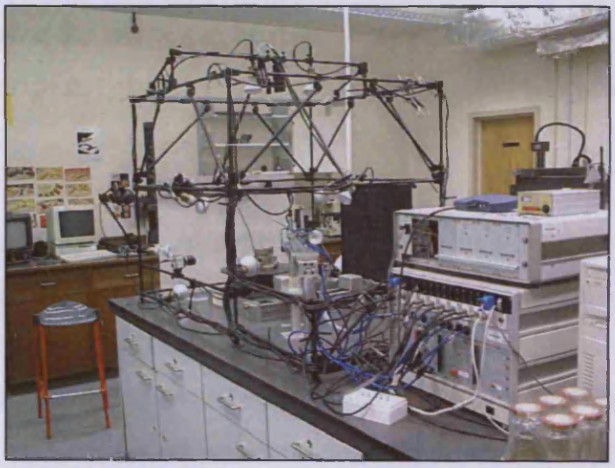


Figure 78. Photograph of the Flexible Inspection Cell prior to installation of the train set, clearly showing mounted cameras and light bulbs for variable viewpoint and lighting, control being provided by the MMB control box on the right hand side of the image

9.2.1 Networked Camera Images

The ability to capture images from networked cameras forms the basis of any attempt to provide a networked Machine Vision system. There must be some point at which an image is digitised, encoded and transmitted from a particular location to where the remote processing system is located.

Network Attached Camera – Direct HTTP Access

Axis NACs are able to have images downloaded from them directly, through requests for an undocumented temporary image file that the cameras generate during the acquisition process. Axis prefer to document a process for uploading of images to remote FTP servers for use from there, to avoid overloading the camera's server in the case of larger numbers of clients wishing to view the image.

There are advantages to direct downloading of images via HTTP from network attached cameras, rather than using FTP, however. Those are:

- Logical encapsulation of images and hosts
- Reduced latency between image capture and download by the processing engine or client
- Reduced bandwidth use vs. FTP upload + download for low-use systems

The first advantage is due to the understanding that an image comes from a camera which is being addressed, and thus the image can be relatively sure to have been obtained from that host. Images do not reside anywhere else, except on the device from which they were obtained.

In the second case, downloading images directly serves to limit the time between image creation and realisation on the client solely to the time taken to download the image from the host by the aforementioned client. In the case of FTPing the image to an intermediary host before downloading it by a client, assuming that the camera is uploading the image every x seconds, in the worst case, the period between image capture and receipt by the client is: (client download time + latency + x)

Finally, using direct downloading, unused images are not uploaded needlessly, since images are only produced and transmitted when there is a genuine request for them, which serves to benefit systems where images are rarely captured.

Conversely, however, there are tangible benefits to uploading images to an FTP server:

- Multi-camera systems have a single data warehouse, leading to multi-sensor aggregation
- Camera images can be logged and stored on a long-term basis

- Camera systems based on limited-performance System on a Chip (SoC) technologies are not overloaded by large numbers of connected clients
- Images stored with a heavyweight HTTP server (e.g. Apache) enable access control mechanisms, where camera hosting only performs simple serving

This demonstration shall focus on direct download of images, and leave FTP for discussion as an extension mechanism to improve the solution to address a variety of other scenarios.

Pan & Tilt Camera

Within an industrial environment, for the purposes of examining a scene, variation in images is highly beneficial; a greater understanding of the scene may be achieved through a panorama than a fixed image. As such, a pan and tilt-capable camera is a useful addition to the demonstration related here. Rather than obscure the structure of the solution with description of the Sony EVI-D31 camera and remote operation of that hardware, that element is addressed within the next demonstration, and the reader should refer to that with the understanding that such a camera may replace the fixed cameras described in this chapter.

9.2.2 Interactive Processing

Assessing the state of a remote site with a view to the installation of a Machine Vision system is essentially a highly dynamic problem that demands the ability to explore the situation, rather than perform established linear tests against imagery and data. The need for interactive processing options has been previously outlined, and needs no reiteration other than to emphasise that it is completely necessary in dealing with a problem where constantly-moving images from wearable computing devices are the primary source of data. Using a formal script-wise processing mechanism on these cases typically fails to satisfy requirements, as either the image is not composed correctly for the script and demands more rigid aspects than can be provided by a mobile camera device, or when the composition is corrected, the movement of the on-site assistant conspires to prevent capture of a consistent image. As a consequence, it must be assumed that all images captured by a mobile camera and computing solution tend to provide varying composition, requiring intelligent compensation. If the problem dictates the presence of a skilled Vision Engineer, as this one does, then it is prudent to utilise the skills of that VE, rather than develop some complex system of intelligent processing script selection and modification through the use of a functional or declarative programming language, such as Lisp or Prolog. These elements have their place in the

creation of networked MV systems, but with a Vision Engineer present, his or her abilities ought to be leveraged.

Interactive image processing using Myriad components involves the following steps:

1. Initial connection, sending the 'start' command, initiating a session
2. Repeated sending of appropriate commands, interspersed with the returning of data to the interface for observation
3. Closing of the session, by sending the 'stop' command (or 'end', the synonym)

The intention of such activity, is to enable single-command access to the Myriad server command set, with state being retained between command requests. This hooks directly into the Interactive Session (IS) facilities described in the coverage of the design and implementation of Myriad servers in previous chapters.

In terms of the server activity, once a connection is received on the appropriate port, the following occurs:

- A storage socket for the connection is created
- The array of client handlers is tested and inactive ones are destroyed
- A new thread (BasicWebServerStarter class) is spawned for the session
- A handler (BasicWebServer class) is created and attached to the thread
- With a 'start' command, the handler is marked as active

All further commands received from the IP address of the client are redirected towards the handler until it is marked as inactive. At this point, the next connection to the server will result in this handler and the associated socket and starter objects that have been created to assist it.

In order to demonstrate the use of interactive processing in the Image Processing Myriad prototype component, four types of interface will be illustrated:

- C# GUI application
- C++/QT application
- PHP + HTML interface

In all of these cases, the interfaces are developed to include three stages of execution:

- Start-up
- Main Function
- Shut-down

This is entirely consistent for normal GUI application creation, with a initiating stage (or constructor), a main interaction stage which continues until the user terminates the application, and a closing stage (or destructor). As a consequence, in most current object-oriented GUI-capable programming languages, Myriad commands may perfectly match the state structure of the application.

Taking the example of a template C# program, the three stages can be clearly seen:

Constructor:

```
public Form1()
{
      InitializeComponent();
      currentServerString = textBox3.Text;
      serverStart();
}
```

Main Thread:

```
static void Main()
{
      Application.Run(new Form1());
}
```

Destructor:

```
protected override void Dispose( bool disposing )
{
      serverStop();
      if( disposing )
      {
            if (components != null)
            {
                  components.Dispose();
            }
      }
```

```
        base.Dispose( disposing );
}
```

In terms of demonstration applications in these languages, the programs must contain:

- Two image placeholders to contain the current and alternate images
- A command entry field

Optionally, a command history pane is helpful, and in the application interfaces is to be demonstrated.

C# Application

Figure 79. Interactive image processing GUI produced in C# for use under Windows XP with a remote Myriad IPE.

For C# code to construct this interface, please see Appendix D.9

C++/QT Application

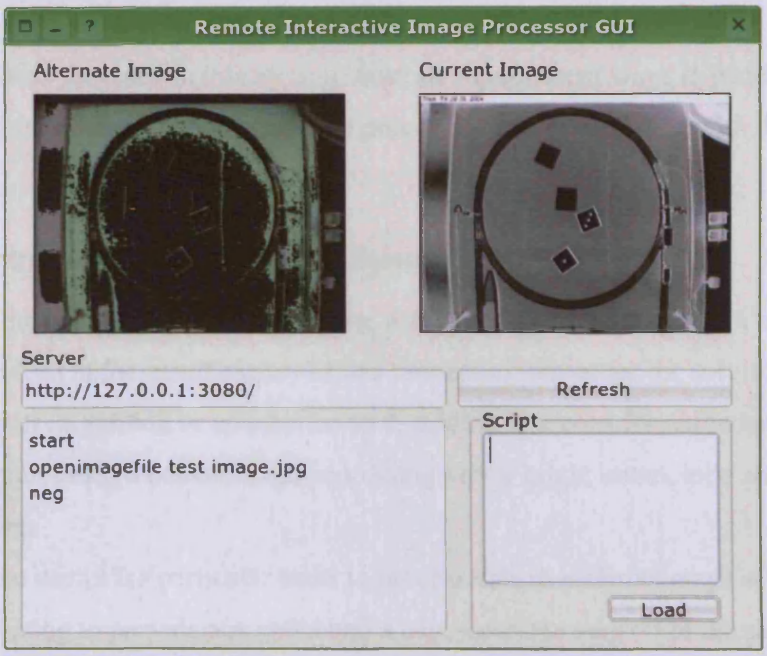


Figure 80. IIP GUI produced in C++ with the Qt toolkit for use under Linux, Window, MacOS/X and all Qt supported platforms (including embedded devices and PDAs) with a remote Myriad IPE

For the C++/QT code to construct this interface, please see Appendix D.10

PHP + HTML

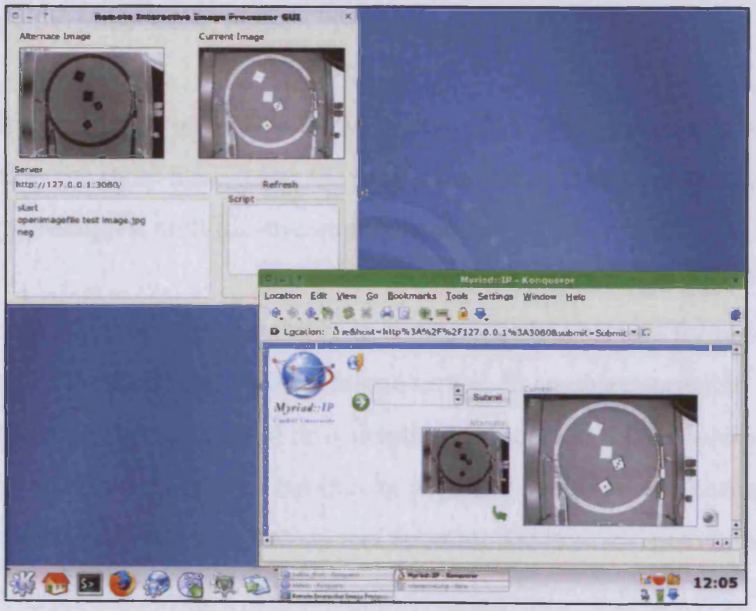


Figure 81. A PHP-generated HTML interface for a remote Myriad IPE alongside the Qt version, sharing a session on the server

For PHP code to construct this interface, please see Appendix D.11

The user interfaces provided in this section illustrate the different ways in which the commands and state retention for interactive processing can be achieved with the mentioned languages.

9.2.3 Managing Multiple Processing Systems

When performing Machine Vision tasks using networked systems, there are a number of factors that can lead to the insufficiency of one processing server for the solution required:

- The server might fail, or connection to it, leading to a need for redundancy
- For certain tasks, a conventional processing server might not include all the required operations
- It may be useful for particular tasks to process data in different ways at the same time (e.g. wanting to provide a negated and a edge-detected version of the same image on a regular basis)
- There may be situations where the performance of a single processing server is insufficient to perform the operations required within a limited time period, necessitating the use of a processing cluster, secondary individual processing servers, or a combination of the three.
- The Vision Engineer may wish to process several images from different cameras observing the same scene, at the same time

As a consequence, there may be an imperative to control multiple processing servers and command chains at the same time, using the same interface, either for redundancy, or expansion of processing, or multiple-stream management.

Using Myriad networked servers, the ability to control multiple servers for processing is a simple expansion of the facility to connect to one server. In an object-oriented programming language with methods, the function of downloading a processed image, with integrated processing commands may be broken out into an object or a method of a core application object and repeatedly accessed with parameters denoting the host and commands. To express this in C++/Java/C# style Pseudo-code:

```
Main()
{
```

```
// Process an image on the server myHost
ImageButton.image = getHTTPImage(myHost, myFile, myPort, myCommands);
// Process the same image in the same way on another server
ImageButton2.image = getHTTPImage (myHost2, myFile, myPort, myCommands);
// Process the image with different commands on server myHost
ImageButton3.image = getHTTPImage (myHost, myFile, myPort, myCommands2);
// Process a different image in the same way on myHost2
ImageButton4.image = getHTTPImage (myHost2, myFile2, myPort, myCommands);
}

Image getHTTPImage(string host, string file, string port, string commands)
{
string totalURL = new String("http://" . host . ":" . port . "/" . file . "?commands=" .
commands);
datastream imageStream = new Stream(HTTP, totalURL);
Image downloadedImage = new Image(imageStream);
return downloadedImage;
}
```

In the case of the remote industrial inspection problem, it is most useful that the Vision Engineer be able to process the same image in several different ways on a regular basis, in order to gain insight into the nature of the viewed scene. For example, with a wearable camera being used by the site-local assistant, every few seconds the VE wishes to take a snapshot and perform the following operations:

- A blob discovery and counting procedure
- Row and column integration functions to establish light levels
- Edge detection to document areas of high contrast
- Thresholding and blob functions to identify regions of a specific colour channel (e.g. searching for red objects that might lead to miss-identification of the red-coloured product to be inspected)

It is important to articulate that this type of multi-processing is typically not possible with rigidly-programmed integrated MV systems. Moreover, even if they offer facilities to offer simultaneously-transformed imagery, as the number of images and complexity of operational

chains increases, the limit of performance of a single computing device is readily reached, restricting use. For devices such as PDAs and Smart Phones, this limit is more rapidly attained, and as such, these devices potentially benefit measurably from the de-localisation of processing tasks across a network.

This type of multiple-system processing interface can be actively demonstrated using C# (desktop computing); see fig.82.

C#

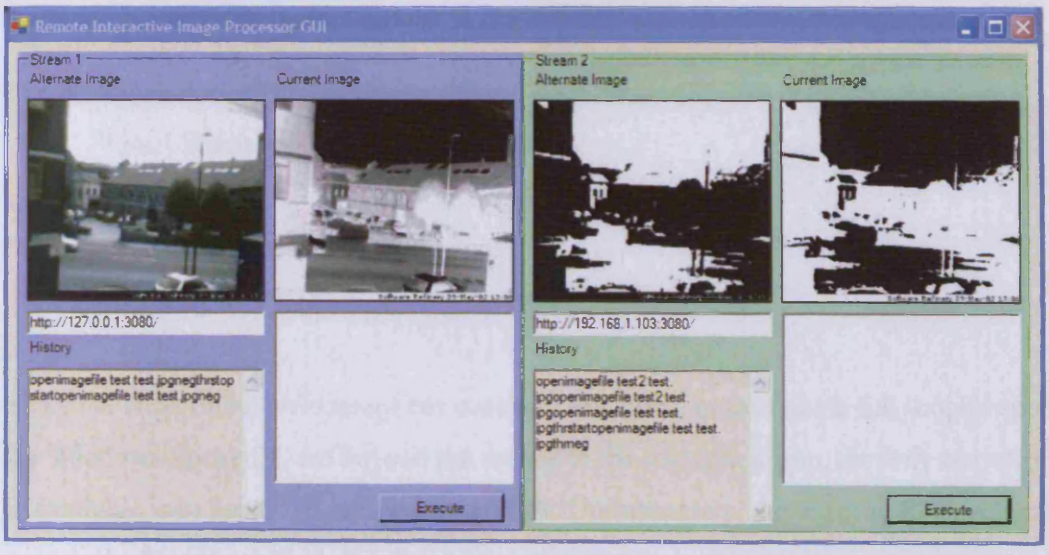


Figure 82. A C# graphical user interface managing two separate image processing tasks on the same source image, using two separate Myriad image processing servers

9.2.4 Mobile User Interfaces

Once a remote monitoring system has been established for the management of remote inspection of industrial environments, there is a further degree of freedom to obtained; whilst the Vision Engineer need no longer visit the industrial campus (henceforth, 'the Site'), they are still constrained to managing from in front of a wired computing device. Suppose that the time zone difference between the Site and the VE is 9 hours (mainland Europe to the west coast of America, for example), in this case it becomes increasingly problematic for the VE to be in their office at the time when the Site-local assistant is available to perform their function. Clearly, there is potential for the Vision Engineer to stay late at work, or arrive early, but when handling a large number of cases, this is highly impractical. It would be far more convenient to be able to take the monitoring computing device home or to social areas, such as coffee shops, leisure complexes and shopping centres. Using a PDA for this purpose,

the Vision Engineer would be able to be available to work, and yet not encumbered by location themselves.

The two options for this style of interface are:

- Full application
- HTML interface rendered by a PDA web browser

In order to create an application for a contemporary PDA, the following options are possible avenues of development:

For Windows PocketPC:

- Visual Studio + PPC base library set

For Linux (QTopia-based) handheld:

- QT Designer / KDevelop / Glade + QT/E Library

Since the costs of a development environment and libraries to create a full mobile application for Windows PocketPC are beyond the means of the research group, the only option presently at available is to use QT/E with a Linux PDA. Unfortunately, the existing PDA is PocketPC-only, and efforts to re-flash it with a Linux distribution have proved unsuccessful. As a consequence, it is not possible to program and deploy a full PDA native application in order to interact with Myriad networks.

The lack of a native application, however, is no constraint to the requirement to display a mobile interface for Myriad components. Since Myriad servers communicate with each other, and are controlled through the judicious use of HTTP requests, it is perfectly possible to develop web pages in conventional HTML, optionally with PHP elements to make them dynamic, and then to take advantage of the humble web browser as a tool for executing URLs and consequently controlling a distributed MV network. The further advantage of this approach, of course, is that the same HTML interface may be used in order to provide access to the system for Smart Phone devices and cellular telephones with in-built web browsers.

Most pertinently, the HTML interface is a useful tool for public-access systems and networks where the user group may vary, forcing full mobile binary applications to be installed on all

clients, in order to provide access. If a given mobile device may only require the facilities of a Myriad network for a few minutes, the installation of custom software is an onerous barrier to use that restricts proliferation of the technology. Utilising a lightweight HTML interface enables ease of use by promiscuous clients, without alteration to the client system, and critically, without delays engendered by downloading a client program. This aspect of lightweight interfaces, likewise confers the same benefits on larger computing devices acting as clients, such as desktop computers, laptops and tablet PCs, all of which commonly provide a web browser, but differing platforms for targeted binary programs. Therefore, the use of an HTML interface enables the establishment of a lowest-common-denominator access system to MV networks and ensures that a Vision Engineer or a skilled user is able to utilise the network wherever there is any network-connected computing device.

The sole difference between a desktop computer and a Windows PocketPC PDA in terms of HTML display, is the screen resolution, which is 320x240 on most present devices, with the usable screen space being 280x240 when viewing web pages using Pocket Internet Explorer 2003, which ships with all PocketPC 2003 devices, without exception. The majority of web designers will find this screen resolution highly constraining, and in terms of providing a meaningful interface to a networked Vision system, it is limiting in the number and size of controls that can be provided onscreen at any one time. The format for such systems, therefore, should be that of a scrolling HTML document, where the essential information and quick response tools should be made available 'above the fold' (a newspaper allusion, where headlines and most important content is contained in the top half of the front page, so as to be visible on top of a folded paper).

Thankfully, Pocket IE, Opera and other PDA web browsers also include a feature to dynamically rescale large images to display thumbnails on a low-resolution screen (nominally called 'Smart Rendering') (see fig.83).

Figure 83. Simplification of a web page by Pocket Internet Explorer, displaying automatic image resizing to adapt to the form factor of the PDA device

In terms of interaction, PDA devices normally include the following means of data entry (see fig.84):

- On-screen Keyboard (operated with the use of a stylus)
- Handwriting Recognition

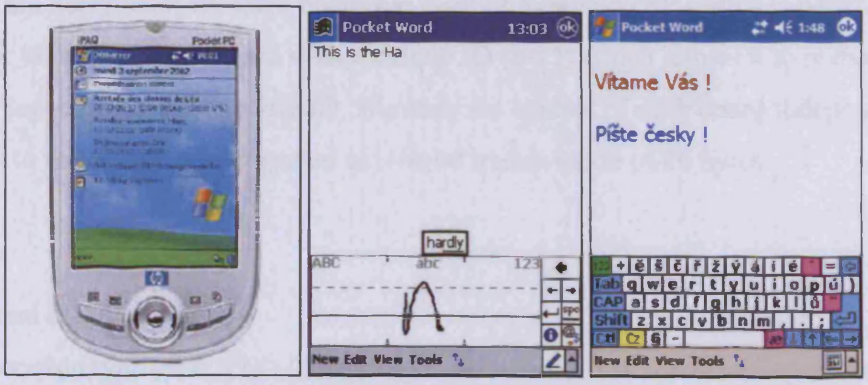


Figure 84. Hewlett Packard iPaq with handwriting recognition and on-screen keyboard user input methods

As a result, it is advisable for data fields and input-based components in HTML interfaces for mobile systems to be kept to a minimum, and replaced with static buttons and links where possible. There is a trade-off between data entry elements and number of static components, however, and the point at which a text entry field becomes the easier to use against a long list of every possible image processing command must be judged by the developer (ideally with the assistance of a controlled usability study group).

In the case of the remote visual inspection problem, the difference between the HTML interface for desktop systems and PDA clients is relatively minimal; with the careful reduction of the use of images, substituted with text links, the processed monitoring image may be continually displayed above the fold of the HTML page on the PDA web browser. This affords the user a higher degree of control and speed of response than if they were persistently required to scroll up and down the rendered page in order to access imagery and controls.

9.2.5 Control of Remote Equipment

Control of the train set as described at the start of this chapter, requires the use of J43 Design's M1 control boards. These boards provide the ability to power devices with pulses managed with Pulse Width Modulation, or use primarily with motors, but also suitable for pulsed devices such as small LED lighting sets. Stepper motors are also supported (driving methods are included with the Myriad prototype servers and have been used to operate an X-Y-θ table), but are not required for this demonstration.

To operate the M1 boards, commands must be transmitted to the RS485 bus, along which the M1 boards are daisy-chained. Each board has a set of mounted DIP switches which enables the operator to provide each board with a unique ID (0-15) which allows it to respond to commands tagged with that specific ID, allowing for control of each board independently. Commands to the boards are formatted as a serial transmission of 20 bytes:

Byte 1 : '*'
Byte 2 : board ID
Byte 3 : operation type letter ('D' – Diagnostic, 'S' – Setup, 'N' – Normal operation)
Byte 4 : command number
Byte 5 – 19 : command arguments
Byte 20 : checksum

Each board must be sent a setup sequence before any operation can take place. This sequence establishes pulse width, time between pulses and number of other factors that affect output performance. The setup sequence can be re-sent at any time throughout operation to alter

performance (slow/speed up motors, affect LED brightness), but it interrupts any current activity.

Each M1 board is able to support three sets of PWM devices, attached to terminals marked Focus, Iris and Zoom (reflecting the origins of the board as a controller for security camera lenses). Myriad Equipment Control servers (henceforth, EC servers) continue this naming convention within methods and command names (note that some commands are prefixed 'gary' with reference to Garry Jackson of J43 Design, who created the boards).

Command	Description
garysetup	Sends setup commands to the current M1 board
garyled	Flashes the LEDs on the current M1 board, for diagnostic purposes
garyzoomplus	Drives the zoom terminal in the positive direction
garyzoomminus	Drives the zoom terminal in the negative direction
garyfocusplus	Drives the focus terminal in the positive direction
garyfocusminus	Drives the focus terminal in the negative direction
garyirisplus	Drives the iris terminal in the positive direction
garyirisminus	Drives the iris terminal in the negative direction
garystepforward	Drives the stepper 'forward'
garystepbackward	Drives the stepper 'backward'
garyhomeforward	Drives the stepper 'forward' until homed
garyhomebackward	Drives the stepper 'backward' until homed
garystepperstop	Stops stepper motion
garyboard <int> Syn: useboard	Sets the current board ID

setduty <int>	Sets the PWM duty
setipc <int>	Sets the Inter-Pulse Clicks
setpc <int>	Sets the Pulse Clicks
setcp <int>	Sets the Click Period
Setsteps <int>	Sets stepper steps
setduration <int>	Sets duration of an operation
Setabssteps <int>	Sets Abs Steps

In order to operate the train, therefore, it is simply a task of creating an HTTP request for the Myriad EC server with commands to:

- Start a session
- Set the current M1 board ID
- Perform the setup for the current M1 board
- Send a command to drive the train back or forth
- End the session

To drive the train forward, therefore:

start
useboard 1
garysetup
setduration 0
garyfocusplus
stop

Note that a duration setting of 0 indicates that the operation should continue until stopped manually; to do this, set the duration to 1 click and do any normal motion operation, i.e. garyfocusplus once more.

Using garyfocusminus will drive the motors in the opposing direction, nominally in this case, to drive the train backwards.

Using a web browser and a PHP-generated HTML interface, a simple train controller appears as seen in fig.85 and fig.86 for palmtop devices.

Figure 85. Screenshot of PHP/HTML interface to control the train set. Motion controls at the top of the page, captured image of the track, top-down beneath, and options to switch cameras for alternate perspectives (along with rudimentary image processing commands)

Figure 86. Train control being performed via a web browser on a Compaq iPaq PDA. Note that the interface has been slightly simplified, and the browser has automatically resized the image to fit the smaller screen. Alternative camera names are placed as text beneath the image.

9.2.6 Conclusion

Combining the elements that have been presented so far, the problem of a remote Vision Engineer inspecting an industrial environment for the creation and deployment of a vision system is finally complete.

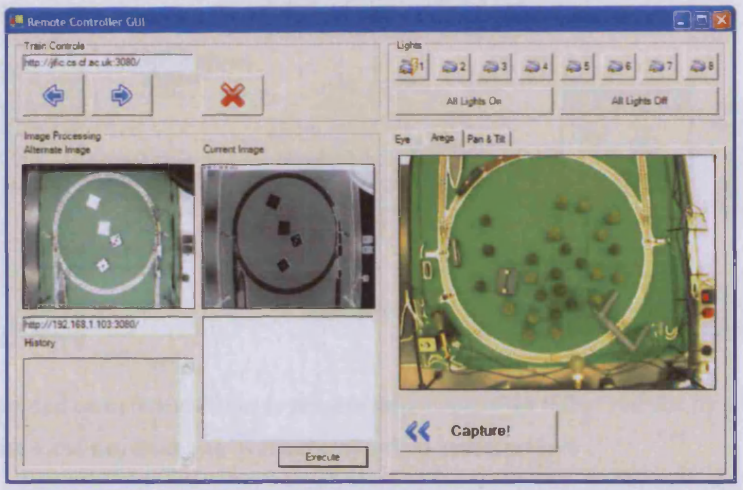


Figure 87. Combined interface for remote image processing, train control and lighting. Connected to 1x IPE, 2xEC and 3xNACs, merged into the same interface

Utilising the user interface provided in fig.87 along with an assistant at the factory (talking using a cellular telephone) as in fig.88, the VE is able to:

- Inspect the scene from multiple angles, using different cameras
- Perform image processing operations on images from each of those cameras (or several at the same time)
- Control equipment within the scene, influencing the flow of production
- Adjust the lighting through a separate server (or the same, if required)
- Discuss the situation with the assistant and perform tests for dust and other issues, using the assistant as a proxy, watching through the cameras
- Take advantage of resources at his/her office and laboratory which aren't available on site
- Roam using a PDA, whilst performing other tasks, or on the move

In short, the Vision Engineer is able to explore the industrial environment and interact with it as if there in person, but able to make more effective use of his or her time and resources, reducing the cost of development of vision systems to industry and enabling the vision

engineer to develop more solutions in the same period of time, by eliminating travel to the site.

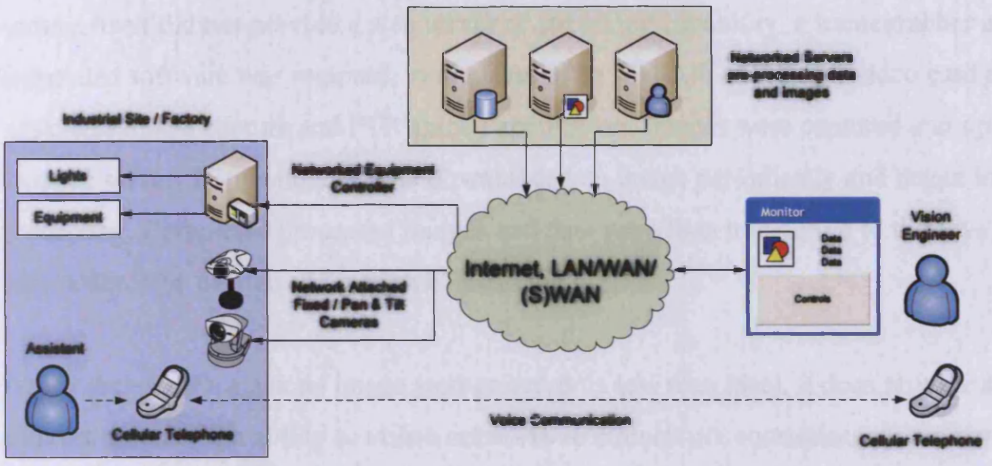


Figure 88. The intended complete solution to remote inspection of an industrial site by a Vision Engineer, with the aid of a site-local assistant and Myriad networked vision servers

9.3 Pan & Tilt Camera Control

During his time as a BSc student at Cardiff University, Simon Caton (previously referred to here with respect to his work in networking Matlab as a Myriad component) undertook work to enable Matlab control of a Myriad-based networked equipment system. His intention was to provide a GUI, enhanced by Matlab automated methods to utilise the Sony EVI-D31 pan and tilt camera. The specific goal was to enable calibration through multi-target interpolation and demonstrate use through locating and centring upon a given object within a scene (in this case, a large analogue clock).

Camera control was provided via RS-232 serial connection to the camera with a proprietary communications protocol. At this point, Mr Caton faced the challenge of implementing serial control methods via Matlab and creating a protocol handler for the task. The prototype Myriad Equipment Control server had recently been completed and was used for testing PWM device control in the train example; provisional methods had also been added to the server for control of the pan and tilt camera, which worked, but a user interface had not been developed beyond a simple HTML test harness. Mr Caton, therefore, decided to complete the Myriad EC command set and utilise it as a networked control engine, rather than develop a completely new codebase.

The design of his system was intended to establish a Java GUI atop Matlab, enable the GUI to communicate control to the server and Matlab to acquire images from the camera. Since the camera itself did not provide a web server or streaming capability, a framegrabber and associated software was required, in the form of an ATI All-in-Wonder video card and the VisionGS image capture and FTP upload application. Images were captured and uploaded to a separate server, from which Matlab downloaded an image periodically and began image processing. Completed processed images and data were then transferred to the Java GUI administered by Matlab and made visible to the user.

Whilst the use of a separate image storage server is less than ideal, it does provide sufficient demonstration of the ability to utilise networks to circumvent constraints placed upon systems by limited hardware. Since the development of the system (seen in fig.89), a number of camera systems have reached the market with on-board image acquisition and web servers and more recent versions of VisionGS also include the ability to host a small web server on the framegrabber machine in order to serve captured images directly, rather than rely on a third party computing device to act as a server.

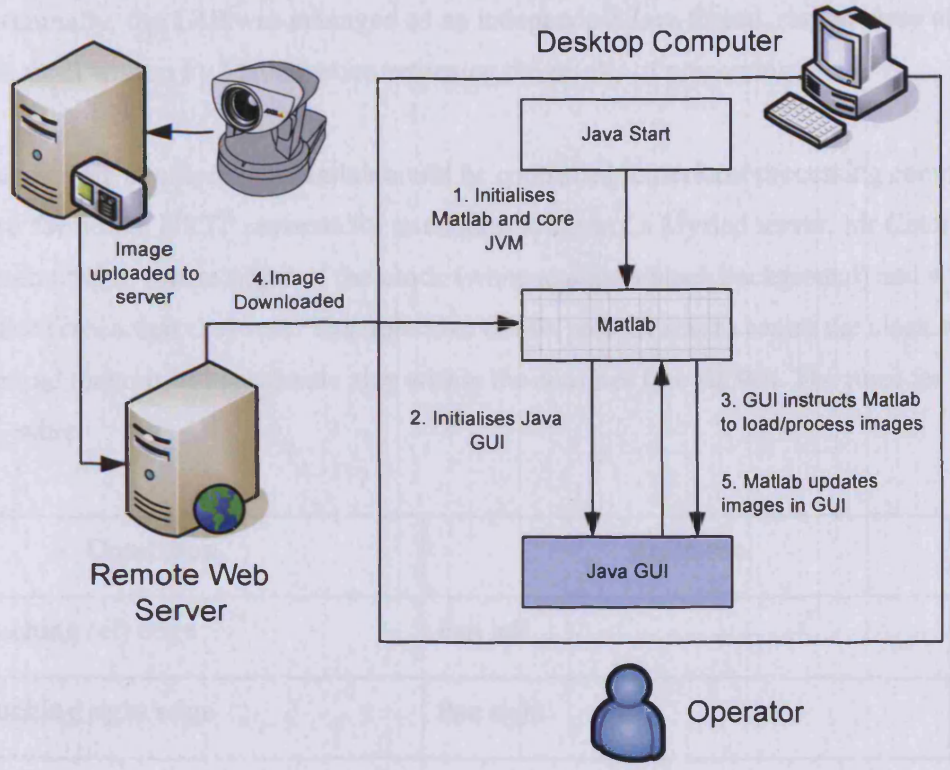


Figure 89. Matlab and Java used together to provide an image processing engine and GUI for use with networked image providers

Mr Caton reflects on this arrangement as follows:

> *"The provided interface between Java and Matlab enables any Java class to be executed in any Matlab file. This also means that an application could, if required, be written in a Matlab .m file. However, this interface is only one way; Matlab can control any Java program but Java cannot control Matlab. ... A new interface is needed because the camera is controlled by a Java program i.e. when the instruction is sent to the camera, it is sent from the Java application. Unfortunately, this feature is not facilitated by Matlab."*

Throughout the course of his project, Mr Caton was able to discover and utilise an undocumented bridge library supplied by Matlab to pass commands back to Matlab (after some command parsing and the creation of a meta-interpreter method within Matlab to execute commands, once received). Problems continued as Matlab calls from the GUI were non-blocking, and the GUI did not wait for Matlab to return results, creating synchronisation

issues. Eventually, the GUI was managed as an independent Java thread, sent to sleep after a command, until woken by Matlab when returning the results of processing.

Once images were acquired and Matlab could be controlled to perform processing commands and also to formulate HTTP requests for execution to control a Myriad server, Mr Caton's system attempted to locate edges of the clock (white against a black background) and which edges of the screen that they met. The objective of this process was to centre the clock within the image and make it of appropriate size within the confines (see fig.90). The rules for this, therefore, were:

Condition	Response
Clock touching left edge	Pan left
Clock touching right edge	Pan right
Clock touching top edge	Tilt up
Clock touching bottom edge	Tilt down
Clock touching all four edges	Zoom out
Clock touching no edges and clock blob occupies a small number of pixels	Zoom in
Clock touches no edges and occupies sufficient pixels	Operation complete. Terminate.

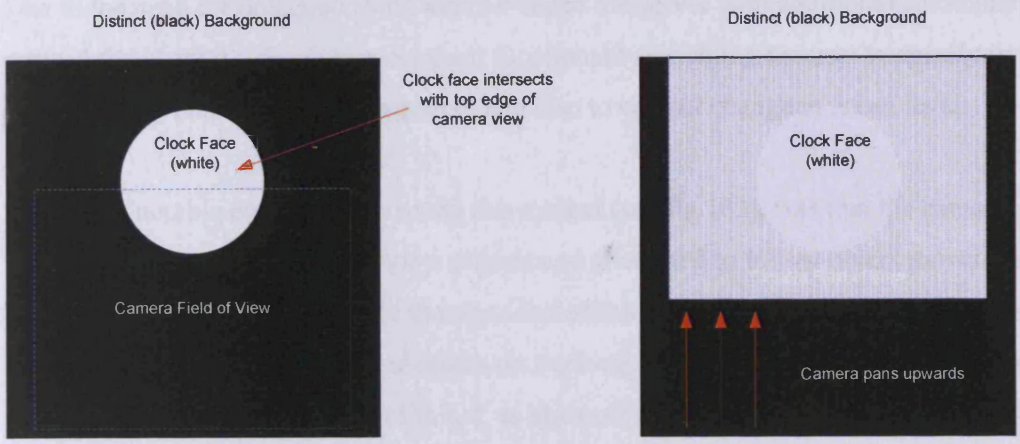


Figure 90. Diagram of the response of Mr Caton's software to clock face position in a captured image, with camera movement to centralise the blob

Mr Caton's software connected to the Myriad server on port 3081 and formed HTTP requests for motion in the usual manner. Commands available for camera control were:

Command	Description
Camstart	Initialise camera
Camhome	Returns camera to the Home position
camtiltup	Tilts the camera up, until stopped
Camtiltdown	Tilts the camera down, until stopped
Campanleft	Pans the camera left, until stopped
Campanright	Pans the camera right, until stopped
Camupleft	Pan/Tilt the camera up and left, until stopped
Camupright	Pan/Tilt the camera up and right, until stopped
Camdownleft	Pan/Tilt the camera down and left, until stopped
Camdownright	Pan/Tilt the camera down and right, until stopped
Camstop	Stop camera movement
Camzoomin	Zoom the camera in
Camzoomout	Zoom the camera out
Camzoomstop	Stop zoom movement

Due to the need for precision, Mr Caton extended the server with additional commands to support the camera's absolute movement functionality, enabling the user to specify in degrees the amount of pan and tilt they required, and also to control change in zoom level.

One of the notable elements of running this system (see fig.103), was that the camera also supported motion tracking of in-view objects and attempted to follow object movement. To this end, Mr Caton and the author incorporated additional commands into the Myriad server to initiate object target finding and start/stop tracking. Success with using this feature of the camera, however, was somewhat limited, as attempts to instruct it to track the motion of the train around the track within the FIC from an acute angle often failed, due to a high degree of target loss. Tracking provided by this model of camera is achieved through blob tracking, determined by pixel intensity values; if the train moved too quickly or altered direction sufficiently, the camera would frequently find a blob of a similar intensity and size in preference to the tracked blob.

Multi-Device Connectivity

Since the Sony camera required a dedicated RS-232 connection, it posed problems when the same computing device was desired to be used with J41 Design's M1 control boards to operate the train system. The computing device only included one serial port, mapped to COM1 under Windows. Realising this limitation, the author added a USB to Serial adaptor to the system, enabling serial devices to be connected via USB ports and resolved back to COM access through driver masquerading. Therefore, the system supported the M1 board chain for device control on COM1 and the pan and tilt camera on COM3.

With the system hardware able to address the two sets of devices, the next step was to enable the Myriad server program to address different COM ports and listen to different TCP/IP port numbers at runtime. Simple adjustment to the *start.java* class which acts as the system loader added command line argument parsing, and passed port numbers and COM details to the *CoreServer*, *BasicWebServer2Starter* and *BasicWebServer2* classes as they were instantiated. Therefore, it became possible to enter the following string at the command line:

> java start 3081 COM3

This started a new Myriad server instance, listening on port 3081 and able to control devices on COM3 (it should be noted that this also allowed for arbitrary comm. port string adjustment, implicitly enhancing the server for use with other operating system platforms (Linux, BSDs, MacOSX, etc.), for which Java abstracts interfaces, except for port names).

From this change, it became possible to run multiple servers on the same device at the same time, controlling different devices (or indeed the same devices) and listening on separate TCP/IP ports. Therefore, one is able to do the following:

> java start 3080 COM1
> java start 3081 COM3

This starts two independent servers, one controlling the train and associated devices through the M1 board on COM1, and another controlling the camera on COM2. Each server is fully multi-threaded, and the system is able to support:

- Clients to control the train and devices
- Clients to control the camera
- Clients to control both the train and camera

Train control is performed like so:

http://myserver.com:3080/?commands=trainstart

And camera control is performed like so:

http://myserver.com:3081/?commands=camstart

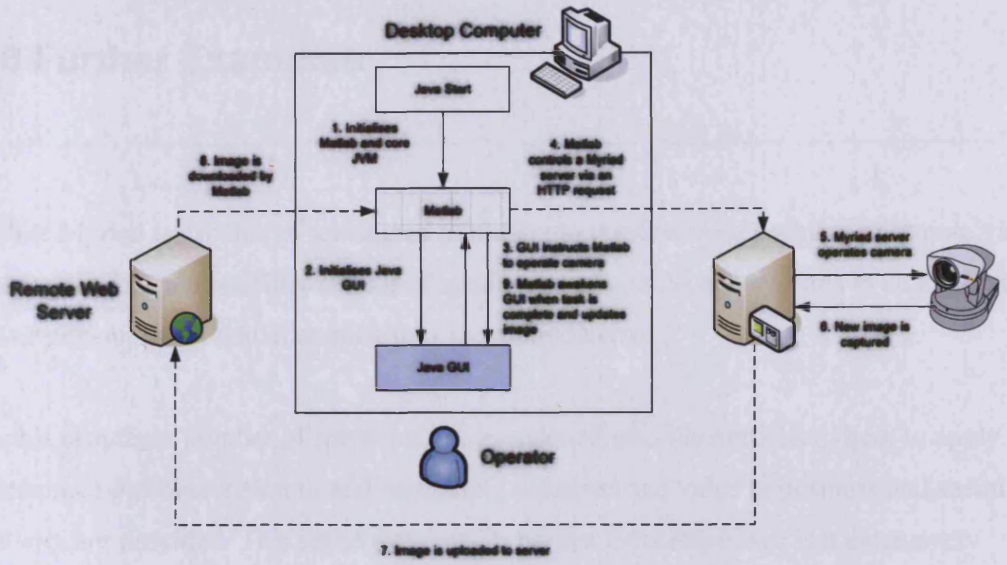


Figure 91. Complete Matlab & Java system with Myriad equipment controller and pan & tilt camera, illustrating a typical camera operation

---

In this chapter, three types of machine vision task were explored using Myriad as a prototyping tool:

- LinPacker (Restricted view inspection and analysis with no equipment)
- Train (Visualisation and feedback system with equipment control, similar to industrial quality control systems)
- Pan and Tilt camera control (Controlled image acquisition for inspection and analysis, dynamic with targeting)

These examples provide a demonstration of the viability and performance of networked machine vision systems implemented in the Myriad manner for conventional machine vision tasks, focused on inspection and equipment-controlled feedback mechanisms. While the responsiveness demonstrated does not lend itself to target systems and safety/mission-critical solutions, the ease of control and flexibility of operation, with interfacing in heterogeneous networked environments lead to rapid development of vision systems. These aspects of Myriad development are demonstrated in the ease of integration with LinPacker, Mr Caton's Matlab project and the creation of a wide variety of GUIs for use with the central problem of remote inspection of industrial environments.

---

10 Further Examples

While Myriad is capable of serving as a solution to the described problem of remote visual inspection for the feasibility testing of installation of new Vision systems in an industrial environment, the potential applications are more diverse.

In this chapter, a number of scenarios are envisioned, and illustrations of how to apply networked machine vision to add computing solutions and value to business and social activity are provided. This set of examples is neither exhaustive, nor is it extensively researched by the author. It does, however, aim to give the reader an understanding of the wider potential benefits of networked machine vision development and a realisation of the options available for the development of Machine Vision tools for a variety of tasks beyond those of classical factory inspection.

10.1 Car Park Management

An interesting challenge for Machine and Computer Vision, is the analysis of images of a car park to attempt to discover the number of spaces free for new cars, under the intended concept of providing a car park attendant in a multi-storey building with a simple number telling him or her how many spaces are available for a new customer.

This problem is difficult from a Computer Vision perspective, because of the problem of identifying free spaces, which while it might seem trivial (looking for a rectangle of clear tarmac), it becomes problematic as lighting changes, cars park over the parking outlines, reflecting car paints produce the illusion of a free space through environmental reflection, and a number of other issues. While this is the case, for the sake of this problem, we will assume that such problems are solved by the application of an unknown, but perfect algorithm, which can tell if a space pointed to by a camera is free. Once we assume this, we may deal with the Machine Vision aspects of the problem. Note that this problem might also be solved by placing clear markers in the centre of the spaces and counting the fully visible markers, but this still requires an algorithm to identify the markers with a number of similar problems of lighting and partial occlusion, so we may simply apply the former assumption.

The first issue with analysis of a car park, is to add cameras. If the car park is presently unmonitored, as most are, then cameras must be installed, and their images multiplexed in some way. To cover the full area of a multi-storey car park we can reasonably expect the number of cameras per storey to be 10 or above for a 100-car floor. For a 5-storey building, therefore, we can expect a minimum of 50 cameras to require installation.

Multiplexing 50 cameras and connecting them all via conventional RF cables (shielded, due to crosstalk between the 50 cables, once bundled) is a significant effort, and would require several miles of cable and large ducting to contain the cables. As a purely physical task, installation of such as system is demanding, and would require the attentions of a team of workmen for a large amount of time. Not only this, but the equipment required to multiplex the transmitted images is expensive and bulky for a 50 camera system. Moreover, once the cameras are multiplexed, a hardware selection and display system must be built to allow the operator to monitor each camera. Performing this task on a per-camera basis is not feasible for 50 cameras, and further, if the operator were to be presented with 10 monitors, and able to cycle the system between each of the five floors, the space required in the operator's office would be significant (see fig.92) and the process of finding a space, time-consuming for each new car wishing to enter.

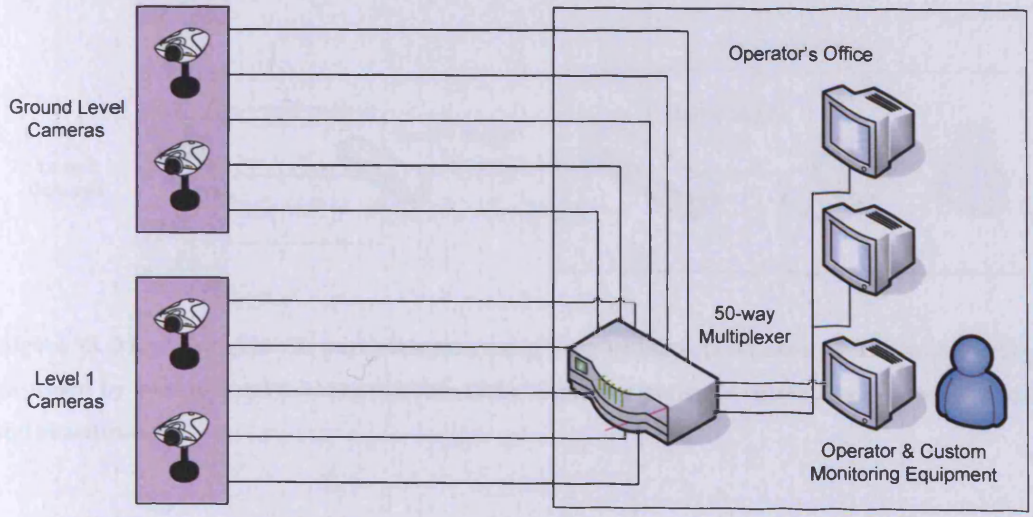


Figure 92. Monitoring a car park structure with a large number of analogue cameras, cabled to a multiplexing system in the operator's office with multiple monitors for display

Beginning to solve the problem using Myriad, therefore, we would start by utilising digital video cameras, since processing an image would demand capture of images to a digital format at some stage in the system. Using the Network Attached Cameras mentioned in the chapter on technologies (e.g. Axis 2100), a network could be produced for each floor, networking the cameras using conventional Ethernet cable to a switch, and then the switches could be connected to a master switch in the operator's office (see fig.93). The advantages of such a system are manifold:

- Ethernet cabling is cheaper per metre in comparison to properly shielded video cable
- Multiplexing of the images is done through the networking equipment, not through specialised hardware
- Multiplexing is also done per-floor at the switch, requiring only one Ethernet cable to be used to traverse the floors to the office, leading to 5 cables tracking between the floors and entering the office, rather than 50 and similarly, requiring less of an interface to connect and multiplex the cables in the office.

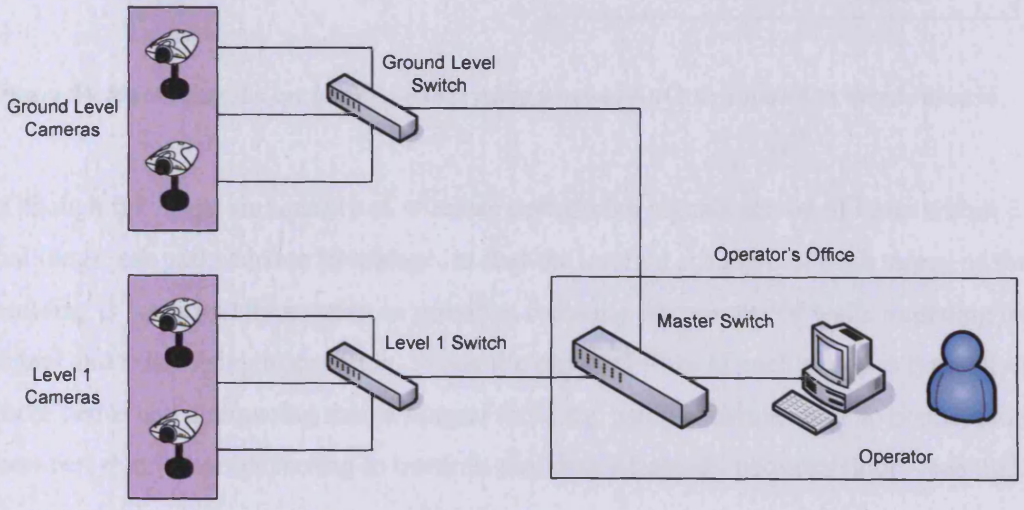


Figure 93. Monitoring the car park structure using Network Attached Cameras with multiplexing provided by conventional network switches and a single computer displaying camera images in a 'many-up' visualisation

The use of Network Attached Cameras in this manner, for security and remote observation of large sites is becoming increasingly common, with institutions such as the author's, Cardiff University, converting their systems. Where high-speed campus-wide networks are already in place, NACs allow for monitoring using low cost components with low incremental cost of

expansion (c. £200 per camera) and access through software adjustment, without the additional expense of cable and hardware multiplexing.

Returning to the car park, using Wireless NACs, the need for cable attaching the cameras to the switch on each floor can be removed, and the floor switches can be replaced by wireless access points (WAPs) for minimal cost (see fig.94).

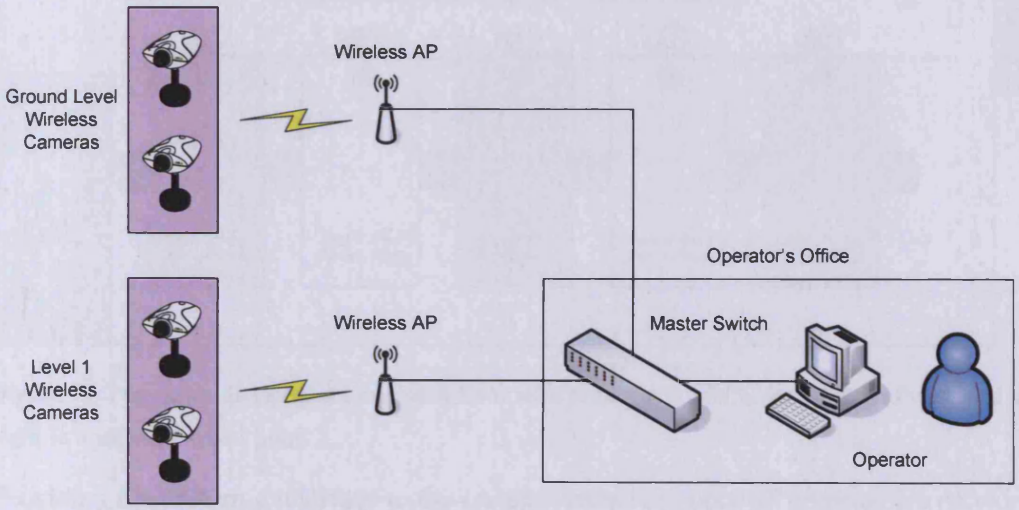


Figure 94. Monitoring the car park structure using wireless NACs as opposed to wired cameras

Although the range and quality of wireless networking signals can be of issue within buildings, car parks have a advantage, in that the internal structure of each storey of the building is intentionally as open as possible, reducing the number of walls impeding the signal and minimising attenuation. While the physical floor of each storey is typically far more dense and attenuating than a normal building, using Ethernet cable to connect the WAPs between storeys avoids having to transmit any data wirelessly between floors (see fig.95).

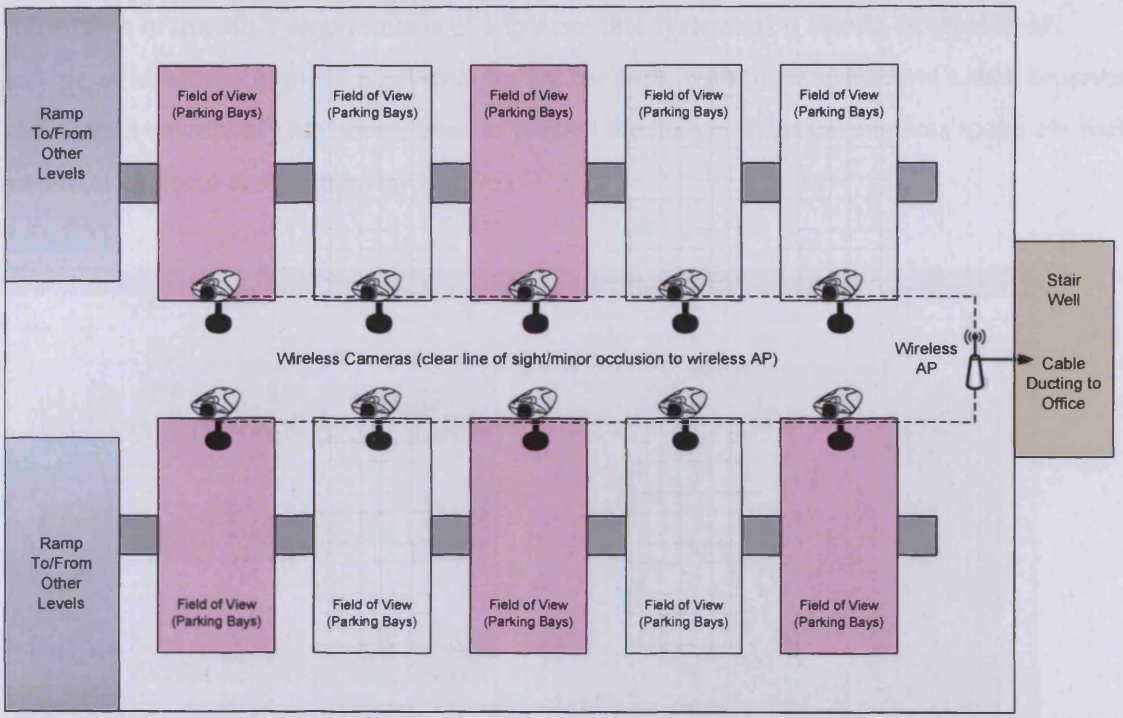


Figure 95. Top-down diagram of a car park floor with multiple WNACs, their fields of view and line of sight to a wireless access point

Providing a monitoring interface to the car park attendant, once all cameras are network and IP-based, is trivial, and requires COTS (Commercial, Off The Shelf) computing resources.

Most importantly, the cost of such a system is quite reasonable:

1 x Operator Computer	£1000
1 x Root switch	£100
5 x WAP	£250
50 x WNAC	£250 x 50 = £12,500
Total:	£13,850

With continual development of cheaper mass-produced wireless cameras, it is quite possible that the price per camera may fall to around £100 per unit, leading to a total system cost of less than £7,000 (excluding installation).

Regardless of the other requirements of a system, this system, as it stands, is capable of serving as a security monitoring system for the car park, with the assistance of a web browser and a small amount of web access code to present the images from the cameras to the car park attendant in a neat and usable way (fig.96).

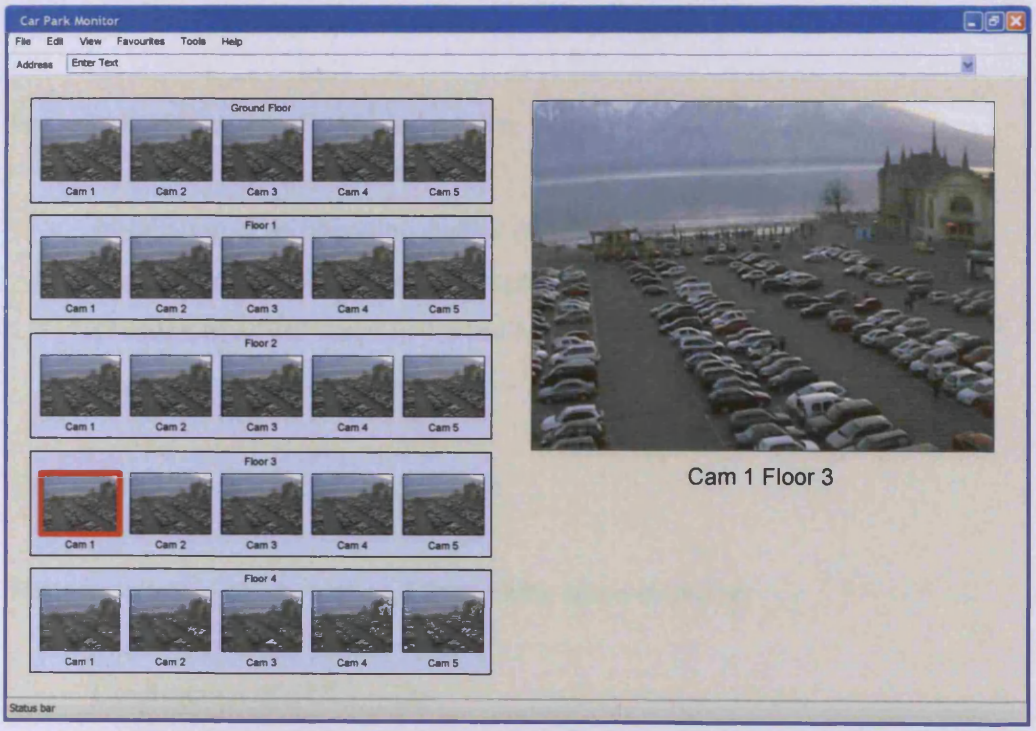


Figure 96. Car park monitoring GUI mock-up, showing 25 camera image thumbnails and selectable full-resolution display of camera images

To begin to assess the number of free parking spaces, the next evident step, is to add a mechanism for processing of the images to identify the free spaces. For the sake of this example, the algorithm is a black box, but the design of the system as a whole remains largely the same. With the inclusion of a small network-connected computing device, such as a micro-ITX form-factor computer with no moving parts, image processing facilities may be added. An image processing program could be added to the operator's computer and included in the user interface, but this would tie the processing to the one computer, which could be used for other tasks, such as e-mail and company tasks, maybe compromising the performance of the system when processing all 50 images to look for free spaces and simultaneously performing other functions. In addition, using pre-written Myriad image processing servers and using a simple script to control the process of capturing the images and

doing the processing, the development time for the entire system could be less than a day, rather than the several weeks required to write a user interface for the attendant and include image processing and image capture/download functionality from scratch. The costs of such development would outweigh the costs of a dedicated server and utilising existing tools.

Assuming a cost per developer of £25/hr.

Tasks for the inclusion of a Myriad system:

- Installation of an operating system (e.g. Linux) on the server ~ 1hr
- Installation of a Myriad server program ~ 1hr
- Creation of a PHP script to control Myriad processing ~ 4hrs
- Network installation of the server ~ 1hr

Total cost: £175 + £300 server equipment

Tasks for creation of a bespoke UI for parking space counting:

- Construction of a GUI ~ 1hr
- Developing code to download all 50 images ~ 2hrs
- Writing a range of IP tools to process the images ~ 30hrs
- Writing functions to connect the GUI, images and processing ~ 4hrs
- Testing and compilation ~ 10hrs
- Installation of the GUI ~ 1hr

Total cost: £1200

This, of course, is a conservative estimate based upon the assumption that the language being used to develop the GUI is perfectly capable of performing the tasks required of it, and the programmer does not have to perform complex steps to overcome obstacles. It is also assumed that if an Image Processing library is used, in the place of the creation of the custom functionality, the time taken for installation, testing and integration will be roughly similar to that quoted for development. Based on the past experience of the author, however, the development of a GUI tool such as the one described above requires a far higher degree of

skill than that of the production of a Myriad control script, and in the cases of such projects, the project duration may last for up to several months. Both of these factors would indicate a total cost for bespoke development of tens of thousands of pounds sterling.

One of the other most useful advantages of such as system to car park attendants, is the ability to provide them with an accurate assessment of the locations of the free spaces, e.g. "There are two free spaces on floor 4, one in zone A, one in zone D.", allowing them to direct their customers to the relevant spots, improving relations.

As an offshoot of using wireless connectivity for the camera systems, and using Myriad server technologies with lightweight interfaces, attendants may also use other devices for monitoring the car park (see fig.97). Using a palmtop computing device as a window into the system, with simple HTML and images, customised for the small size of the screen, a quick and effective tool may be created for a variety of purposes:

- Free space counting while roaming
- Security monitoring of the car park while in motion (tracking cars and people as the attendant intercepts, or for security personnel)
- Customer assistance (e.g. Ability to locate cars for customers who have forgotten where they parked, regardless of location in the car park, with access to all cameras)

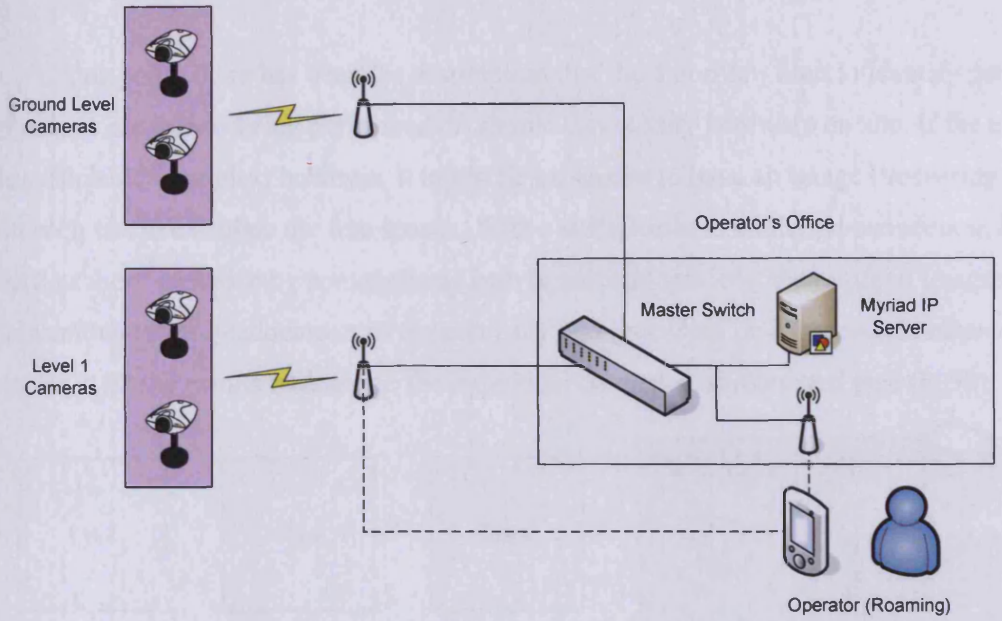


Figure 97. Car park operator roaming with a wireless PDA to monitor camera images

Extending the system – Signs

One of the most common features of a car park, is a sign denoting the availability of free spaces. Normally, this will be either a static sign (Free Spaces/Full), managed by the attendant. Using the system already outlined for allowing the attendant to monitor spaces, a Myriad Equipment Control server could also be added to the system, with a serial interface to a digital counter embedded into the spaces sign. Periodically, or when a new car enters the car park (using a normal light-beam trip switch), the EC system can run a test for the number of free spaces, and update the sign.

Expansion to Multiple Sites

As car park owning companies expand, they wish to manage a number of different sites as a natural process of business growth. Using the Myriad-based tools already described, to monitor a car park, and then duplicating these tools across a number of sites, there is potential to actively manage the business centrally and to aggregate data (see fig.98). The advantages of the networking aspects of the system described previously allow it to be extended across a wider area, with the Internet providing the backbone for a wide area network or secure connected system.

Up to this point, there has been the assumption that the algorithm used to identify parking spaces is capable of being performed on simple commodity hardware on site. If the algorithm is sufficiently complex, however, it might be expensive to have an Image Processing server on each site to calculate the free spaces. With a sufficiently fast Internet connection, however, such as those provided by conventional 1mb broadband services, the required images may be transmitted to the headquarters of the company and processed on a large-scale enterprise server, with the results returned to the individual car parks, as requested (see fig.99).

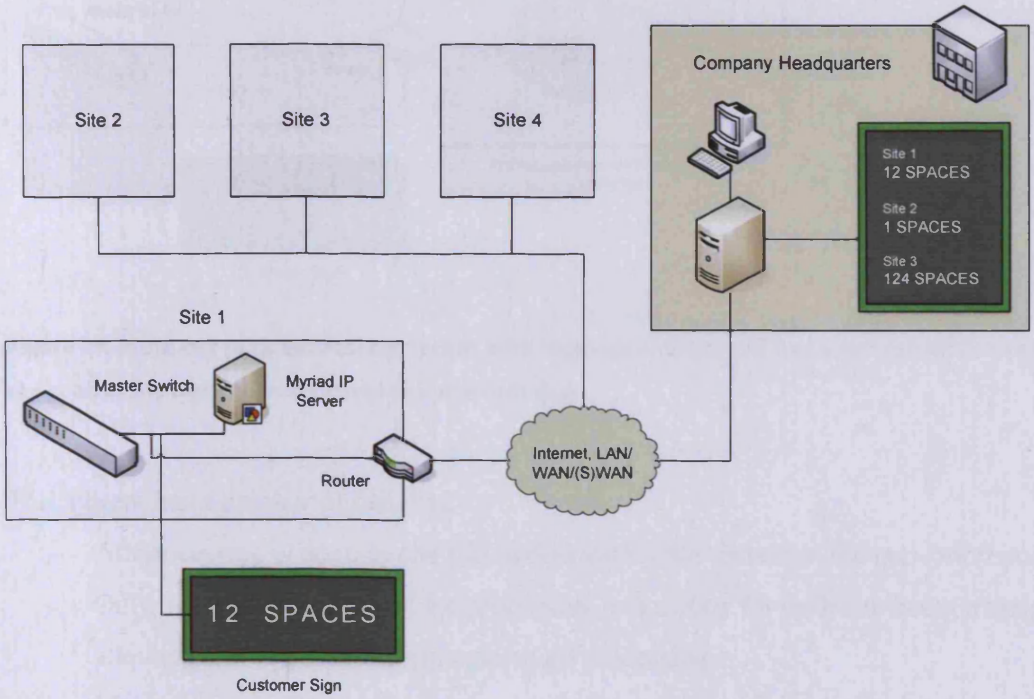


Figure 98. Multi-car park site data aggregation at a remote headquarters

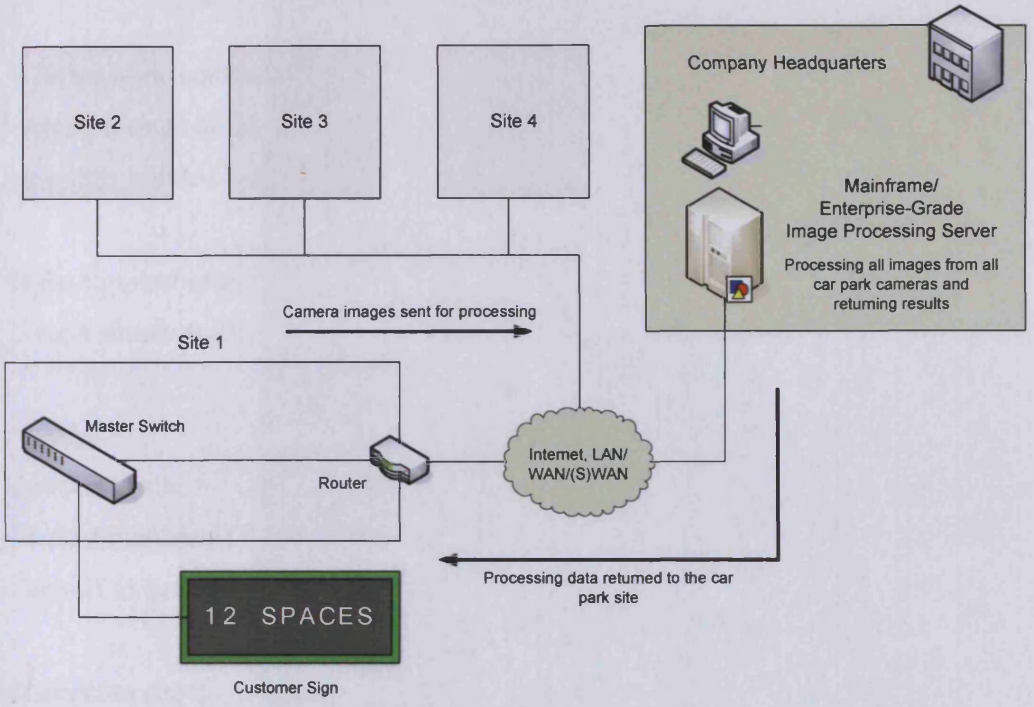


Figure 99. Multi-car park monitoring system with image processing and free space calculation performed at a central site and values returned to individual sites

This system has a number of benefits:

- All processing is done in one place, delineating the system in the network topology
- Only one algorithm is used for processing, not a copy for each site being managed, allowing fast and uniform changes to all processing
- Maintenance is done at the headquarters site, reducing downtime thanks to staff technicians, where a failure at a remote site might need an engineer to be called out.

Data Management

Car park management is a highly analogue field, and one which it is hard to develop metrics for; for the manager, there are questions of exactly how to measure the performance of each site, beyond simple counting of takings at the end of a period of time. Using the free space counting system devised previously, it is possible to devise a series of alternative measurements of performance of a car park site, and produce systems for logging of activity and subsequent analysis to allow for more management understanding of the business in real time.

With network-connected tools for discovering the number of free spaces, a central computing system, located at the company headquarters, and linked to the networks at each of the car park sites is able to poll each of the sites and discover the current number of free spaces.

In the simplest case, this data can be collected on a half-hourly basis, and stored in a database using a simple script, run automatically:

<header>
Connect to site 1
Request number of free spaces
Connect to database and add record of free spaces and time

Connect to site 2
.
.
.

Either through the creation of customised analysis tools, or through the use of an application such as a spreadsheet with graphical capabilities, this data can be graphed to show the efficiency of each car park site throughout the business day. Time series may be constructed, comparing the performance with that throughout previous weeks, months and years.

With this data, a management team can then perform a range of alterations to their business activity:

- The number of attendants can be increased at busier times of the day, and reduced at less busy times.
- Quieter car parks can be more actively promoted, or those which suffer periods of low custom can be improved through the addition of deals and special offers to improve attendance at off-peak times
- Understand when demand reaches a level at which a new car park in the same area might be viable
- Assess the impact of local competition and allocate resources to combat it

While these are fairly low granularity solutions, they can be used in a very focused manner, such as the offering of a special deal to a large neighbouring office complex of a car park to improve day-long staying custom.

Beyond the monitoring of free spaces, however, the development of other business metrics could be advantageous. Using time series on each of the images coming from the on-site cameras, an algorithm could be produced to anonymously track the length of stay of each car in the car park. This would provide valuable data as to the types of customer that use the car park; when they arrived during the day, how long they stay, and how much money they pay.

Changes in the Business Operation

Finally, the creation of network-accessible resources for monitoring of multiple car park sites from a central location also offers the opportunity to use that data in new business manners. One of the most interesting developments would be to offer pre-booking of parking spaces to customers through telephone or Internet services, or in association with other companies, such as shopping centres and leisure complexes. Take an example of a family, wishing to visit the cinema in a major city on a busy weekend. Normally, they would be able to telephone the cinema and book the tickets that they required, to guarantee their seats. Unfortunately, of course, they would not be able to be sure of their ability to find a parking space close to the cinema (or even at all, in some cases). As a result, they might postpone the visit, losing custom to both the cinema and the car park. If a networked Vision system were monitoring the car park, however, and the data from there were to be fed into the company's web site and web services, and vice versa, it would be possible for the customer to reserve a parking space and pre-pay for it in a single transaction with the cinema tickets, so that they could be assured of a successful excursion.

10.2 Non-Hospitalised Dermatological Tracking

One of the most current and complex developments taking place in public sector computing at the present time, is the effort to introduce digital, centralised patient records for the National Health Service. The objectives of this system are to store the details of a patients in a uniform database, able to be accessed by a health care professional whenever they deal with the

patient, regardless of whether they have previously met; avoiding the issues of attempting to locate patient notes from other institutions and doctors when the patient is unable or unwilling to provide a full history themselves.

As a tangential benefit of such a system, the details of a patient become a networked resource in much the same way as a business would use client records stored in a headquarters and distributed to the employees who need to deal with the client. In this process, security and access controls are clearly of importance.

We will examine the case of patients who have received primary care, and are released to return home, with regular follow-up sessions with the district nursing staff. In these cases, the District Nurse (henceforth, DN) will check up on the patient on a daily or weekly basis, perform tasks such as renewing dressings, and monitoring the patient for change in status that might require further intervention. Many of these cases involve recovery from injuries, disease and surgery with visible results, such as lesions and wounds.

Suppose that a patient has recently been released from hospital, recovering from minor burns. The DN visits the patient twice a week, to change the dressing on the wound and to examine it for changes in status, with a view to seeing improvement in the condition.

In the majority of cases, the progress of the wound will hopefully be towards recovery, with regular improvement, but in some cases, the wound may not heal, or may begin to become infected. The degree of severity will be based in contrast to previous examinations, unless the change is extreme and highly visible, which would lead to immediate hospitalisation. In cases of moderate change, however, the case for admission is less clear, and requires the consultation of a specialist. This process requires an appointment at the hospital and the transport of the patient both to and from the hospital, consuming a great deal of resources and discomforting the patient, should they be elderly or infirm.

This problem is somewhat similar to that of the installation of remote Vision systems as mentioned previously in this thesis, in that the person capable of doing detailed analysis of the situation is actually remote to the patient, and in their place, a trained operator (the DN) is working locally to the patient (see fig.100).

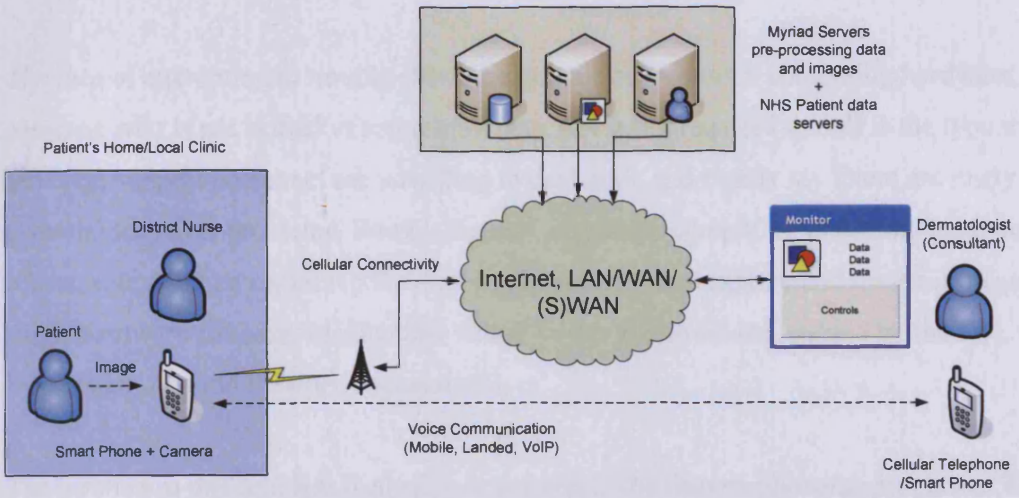


Figure 100. Diagram of non-hospitalised dermatological tracking system with interaction between a DN on site and remote consultant

Clearly, there are some differences in the two problems, however. First and foremost, the DN cannot be assured to be in a network-connected area. Secondly, the objective is not to obtain an impression of the environment and capture live images, but to collect a single high-definition image of a particular object; in this case, the burn site. If it were possible to capture an image and send it to the NHS's data centre for attachment to the patient record, a consultant might later be able to examine the record and make a judgement as to whether to admit the person or begin some form or outpatient treatment. The consultant would not have to visit the patient; nor would they have to be on call specifically when the image was captured, in most cases, as more extreme situations would be automatically referred to the hospital by the DN. This would leave the less urgent cases to be dealt with by a consultant as they were able to (although hopefully within a few hours of the DN visit).

The question, in this case, is how to obtain an image and transmit it to the data centre, from a patient's home, which in most cases, is unlikely to be connected to the Internet, and if it was, the issues of securing access to the data centre and support for the programs needed to read the data from a camera and transmit it are significant:

- Giving the DN copies of all the drivers on all platforms, for a flash card reader to download the image
- Supplying a program that should be installed to make a secure connection to the data centre, where the patient's computer may or may not be able to do so

The idea of attempting to trouble-shoot the process on unknown computing hardware, with someone who is not skilled in computing (as a DN is not required to be!) is the type that technical support personnel are unwilling to deal with, and rightly so. There are many potential technical problems. What is needed, is a stable computing platform, small and robust, able to make contact to the data centre securely and capture and transmit images as a single hardware solution. Ideally, this would be a portable device, carried by the DN, with long battery life and network connectivity.

The solution to this problem is already in existence; the camera phone/smart phone. These devices:

- Derive network connectivity through the cellular telephone network
- Are able to host secure transmission applications
- Include digital cameras with resolutions up to 1.3 mega-pixel (1280x1024) and rising, auto focus, auto iris control and optical zoom
- Are able to capture an image and utilise it without using a separate data medium, such as a flash memory card
- Have a far wider range and area of coverage than wireless networking systems, at present

The problems, however, are twofold:

- Transmission speed is limited at the present time, precluding the use of video and limiting the solution to static images
- A connection program would need to be developed to enable secure transmission of images.

For the first problem, the use of single static images, or small collections < 10 images should more than suffice for accurate portrayal of the state of the patient, in this case, the state of the burn wound itself. This is actually a positive advantage, since the limited bandwidth would provide a constraint against the system accumulating very large quantities of large images and video for each patient. Mirrored across the cases of 60 million patients throughout the courses of their lives, the scale of storage required would be substantial.

e.g. 10 x 20MB video x 60m = 12,000 TB

The latter problem might be solved through the use of smart phones with either in-built web browsers with > 128bit SSL connections to the data centre, or using Secure Shell file transfer, once again, with public/private key-based encryption. The advantage of smart phones being that they can run desktop-grade programs for these tasks, and provide TCP/IP-based connectivity to those programs, easing the creation of tools to support the DNs.

One of the problems that cannot be solved technologically, however, is that of good lighting control. The difficulty with capturing images of people in their own homes, is that the lighting is non-uniform, and when dealing with patients of differing races, who skin reflects the ambient light in a number of different ways, this situation worsens. The solution is only partial, unfortunately; actively training DNs to use the smart phones would be a requirement of such a project anyway, enabling them to take good photos, using the tools enabled on the phones and also giving training to taking neat, targeted images. Secondly, however, the DN would need a basic course in lighting control. Since the images from the camera would be evaluated by consultants, rather than automatically, the need for tightly controlled lighting is somewhat alleviated, but training should spend time teaching the DN how to manage the light in a location in terms of changing the lighting to gain a slightly better and more suitable environment where possible (turning off small lights, making sure that spot lights do not unduly light one side of the site over the other, etc.). If training could be given, well enough to ensure that illumination was at least uniform and enabled clarity of vision by the consultant, then that would suffice for the task. It might also be possible to add lighting information to the smart phone as a resource for DNs as they did their job, suggesting lighting changes and opportunities to improve image quality. A tool such as the Cardiff University Lighting Advisor might be ideal for this task. If there was any doubt as to the quality of the image, the patient could attend an appointment at the hospital, the current standard activity, providing a safety measure to ensure protection of the quality of care.

In the longer term, the images stored in the records database could be pre-processed, prior to their examination by a consultant, producing highlighted areas of interest and showing differences across a time series, to better inform decision making. It also might be highly useful to expand the image processing capabilities of the system and include a Neural Network, to attempt to classify the most urgent cases and notify a consultant that a patient's images need to be examined urgently (see fig.101).

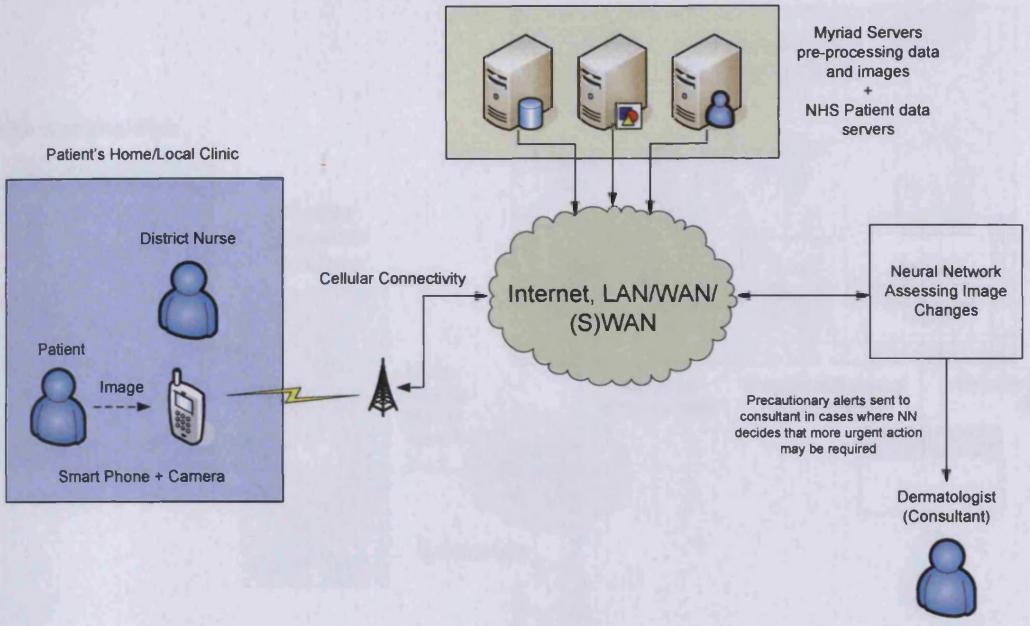


Figure 101. Diagram of tracking system with automated classification of images for prioritised assessment

10.2.1 Demonstration

In order to provide an initial exploration of this task, three elements must be demonstrated:

- Acquiring of an image by cellular telephone camera, transmission to a networked server
- Insertion of image and meta data into a database
- Image and meta data extraction from database and representation to a user

These elements are resolved in the following manner (see fig.102):

- Image is captured and sent via the mobile phone network to an SMTP outgoing e-mail server (provided by the cellular network provider) to a recipient e-mail server, supporting POP3 clients
- A Perl script periodically polls the recipient mail server, downloads e-mails (with images as in-line attachments), parses the data, extracts images and meta-data and inserts into the local database
- A PHP script running atop a web server, upon request by a connecting user queries the database and presents the stored images and meta-data ready for user inspection

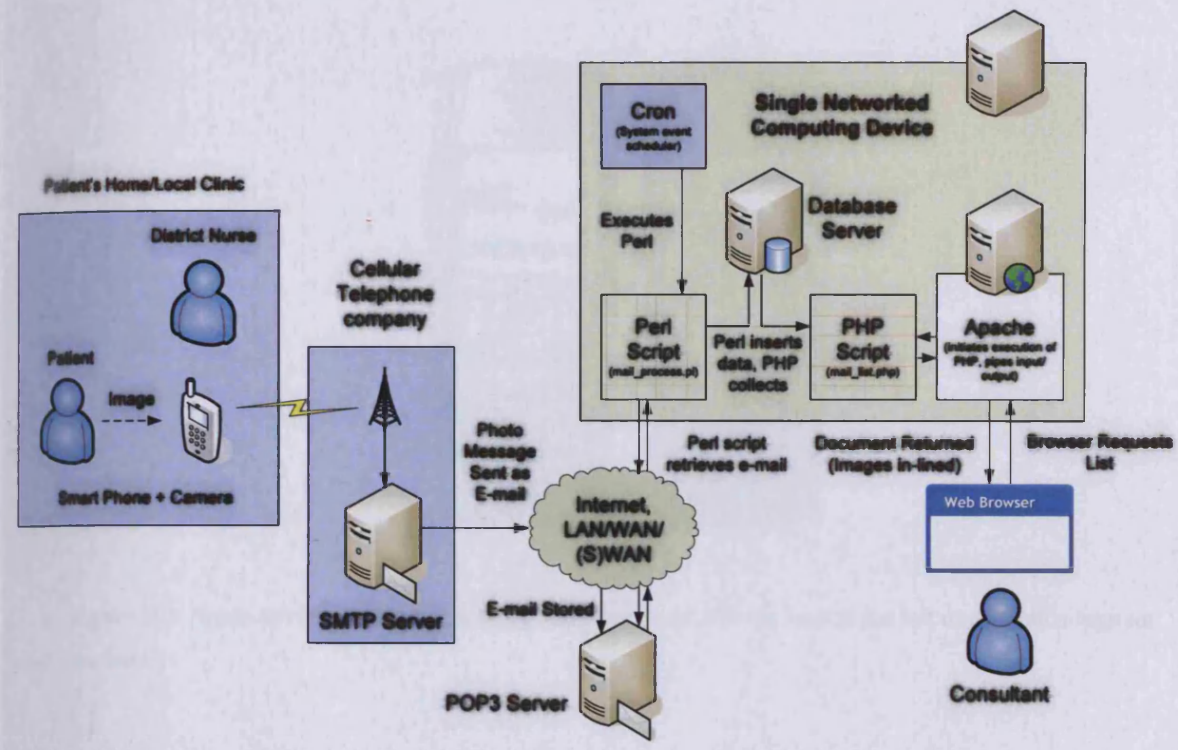


Figure 102. Diagram of demonstration system, acquiring and transmitting an image by camera phone via e-mail to a database, read from a client web browser

Image Capture & Transmission

This primary stage of the process requires that the person local to the scene be in possession of a cellular telephone with a camera, also capable of sending an image to an e-mail address. For the purpose of this demonstration, a Nokia 6610 (see fig.103) shall be used; critical features being:

- Colour passive matrix screen (128x128 x 4096 colours)
- Multimedia Messaging Service (MMS) (photo messaging)
- GPRS and HSCSD high-speed data connectivity
- J2ME Java support
- Tri-band (GSM 900/1800/1900 bands) for international coverage
- In-built digital still camera (352x288)

Figure 103. Nokia 6610i mobile phone, front and back (note camera lens to the left of the Nokia logo on the back)

It should be noted that this camera only outputs 0.1 million pixel images. More modern handsets are capable of providing up to 2 million pixels (1600x1200) with auto focus, auto iris, in-built flash bulbs and optical zoom functions. Hence, the reader is asked to kindly disregard the low quality of the images presented, as this is not a representation of modern technological potential, but rather, of equipment available to perform tests.

In this example, e-mail transmission of image data and accompanying meta data is used; however, this will not suffice for a target solution, as data transmission is not secure, and it would be inappropriate to send medical imagery and identifying patient data by such insecure means. E-mail is convenient for a demonstration as it enables a cellular telephone by default (without any additional software installation) to transmit both the image and meta data in a combined single document, containing:

- A time stamp
- Details of the sender (the 'From:' field)
- Details of the sender's computing device (agent string, header)
- A title (the 'Subject:' field)
- Freeform meta-data section (the body)
- In-line image integration (Base64 in-line attachment as a multi-part message)

Regardless of the means of transmission of data, these data elements are essential to the task of categorising information and associating it with patient records at the recipient end. In a production system, a Java GUI would manage the camera, acquire the image, compress it, gather meta-data from the operator, open a secure connection and transmit the file (most likely using SCP, file transfer over SSH). The data required would still be the same as in the e-mail utilised in the demonstration, and the format would be similar, or use XML to delineate elements, rather than rely on the implicit format parsing found in e-mail clients. J2ME SCP clients and JPEG encoders already exist, so it is merely a case of using manufacturer APIs to control the camera and providing a user interface to combine the features of these libraries into a complete application.

Using the Nokia 6610, the process of acquiring and transmitting the image and data is as follows:

- Press Up on the directional pad
- Direct the camera and press the left select button to take a picture
- Press the left select key to choose 'Options'
- Select 'Send'
- Type in additional meta-data as a text message (this will become the body of the e-mail message)
- Press the left select key to choose 'Options'
- Select 'Send to e-mail'
- Type in the e-mail address of the recipient account (in this demonstration, luke@linuxcomment.com)
- Press the left select key to choose 'Ok'

The telephone will then open a GPRS connection and send the data; this process may take several seconds, depending on network signal strength. Once an image has been obtained, however, transmission can be postponed, enabling the camera to still be used in areas with no cellular network coverage, and data to be sent once the handset is moved back into an area of coverage. So a DN may take a picture at a patient's home (which is in a black spot, or is out of range of the network), and send the image once the DN returns to a covered area.

At this point, the cellular telephone network provider converts the text message into an e-mail and emits it onto the Internet via their own SMTP servers, to be relayed to the mail server of

the recipient (linuxcomment.com in this case). This process may take several minutes (test transmissions took between 3 and 4 minutes).

E-mail Collection & Parsing

The process of collecting the e-mail messages from the recipient e-mail server relies on a Perl script, which connects to the server using POP3 (Post Office Protocol) and downloads each message in the mailbox in turn, extracting:

- The From: field
- The Subject: field
- The message body
- An in-line attachment

The demonstration script assumes that all of the above are present and that the e-mail only contains one attached image, encoded in Base64 (process of converting binary data to ASCII for the purposes of transmission and storage in ASCII-only data structures). These assumptions could be removed, but they ensure clarity for the reader and expediency for the author. The author has, as yet, not come across a cellular telephone able to contravene this, however, hence the assumptions are also fully sufficient for use with conventional handsets and cameras.

Since POP3 is a client-request-driven protocol, it requires an e-mail client to connect to the server and actively request a list of messages, rather than being able to notify the client whenever a message is received. For this the Perl script must be regularly executed in order to poll the server for new messages. The Unix 'Cron' task scheduling daemon is used to execute the Perl script every five minutes; alternatively, the Perl script could run a loop and enter a wait state for five minutes between execution (this would provide the same effect).

The Perl script:

- Connects to the mail server using the Net::POP3 library
- Collects a list of messages and message ids
- Downloads each message in turn and finds the From: and Subject: fields, storing them.
- Message parsing then finds the Content-description: fields denoting that subsequent data chunks are either body text or in-line attachments, and loops through the message

lines until the end of the block is found. Once this happens, the body or attachment is stored.

- When a message is finished, a connection to the database is established and data is inserted (into the table 'tracking')
- Completed message is deleted from server
- Further messages are parsed until the mailbox is empty
- Connection with the server is terminated and the Perl script terminates

For the full source of the Perl script, please see D.12.

The schema of the database table is:

id	mail_from	mail_body	mail_attach	mail_date
Int(11), Primary Key, auto_increment	*Text*	*Text*	*Text*	*Date*

In a target system, this table would have additional fields containing keys for the case number and patient identifier, which would connect the data to the patient and their case (the case id would suffice, but a patient id might improve the speed of some queries and add a secondary check to ensure that data is assigned to the correct patient).

Data Reporting

Examining the data once it is entered into the database is a generic database reporting task. For the demonstration, a PHP script is executed by the Apache web server residing on the same computer as the database server and the Perl script. The script itself queries the database and extracts the last five entries in the 'tracking' table (see fig.116); this data is then built into an HTML page, with an internal table with the data arranged.

```
<html><head></head><body> <center>
<?php
@ $db = mysql_pconnect("localhost", "root", "");
mysql_select_db("test");
```

```
$sql = "SELECT * FROM tracking order by id desc limit 5";
if(!$result = mysql_query($sql))
{
echo "Query failed";
}
else
{
if($myrow = mysql_fetch_array($result)) {
  do {
        $myrow[mail_body] = str_replace("\n", "<br>", $myrow[mail_body]);
        echo "<table align=center width=600 cellspacing=5><tr><td>
$myrow[mail_date]</td><td rowspan=2>

<img src=\" ${myrow[mail_attach]}\" border=1>

</td></tr><tr><td valign=top bgcolor=#EEEECC style=\"border:dashed 1px #000;
padding: 8px\"><i><b>
$myrow[mail_from]</b></i><br>$myrow[mail_body]</td></tr></table>";
  } while($myrow = mysql_fetch_array($result));
}
}
?>
</body></html>
```

Most of this script is standard PHP, embedded in the HTML page between the <?php and ?> tags. The first point of interest, is the database connection, provided through the mysql_pconnect method, which assigns the connection to a handle ($db) afterwards. The variable $sql is then created containing an SQL query, and the mysql_query method is called to execute it. At this point, a test occurs to see whether any results have been returned, and a do – while loop is run, fetching a row at a time from the result (*mysql_fetch_array($result)*) and storing it the array $myrow.

Inside the output loop, the $myrow[mail_body] (the body of the transmitted e-mail address) is cleaned up, converting new line characters (\n) into HTML barrel returns (
). Then output is done through an 'echo' command, producing HTML text with PHP variables embedded (e.g. *$myrow[mail_from]*), which are interpolated into their proper values. The loop then continues until no more rows are left from the database query result.

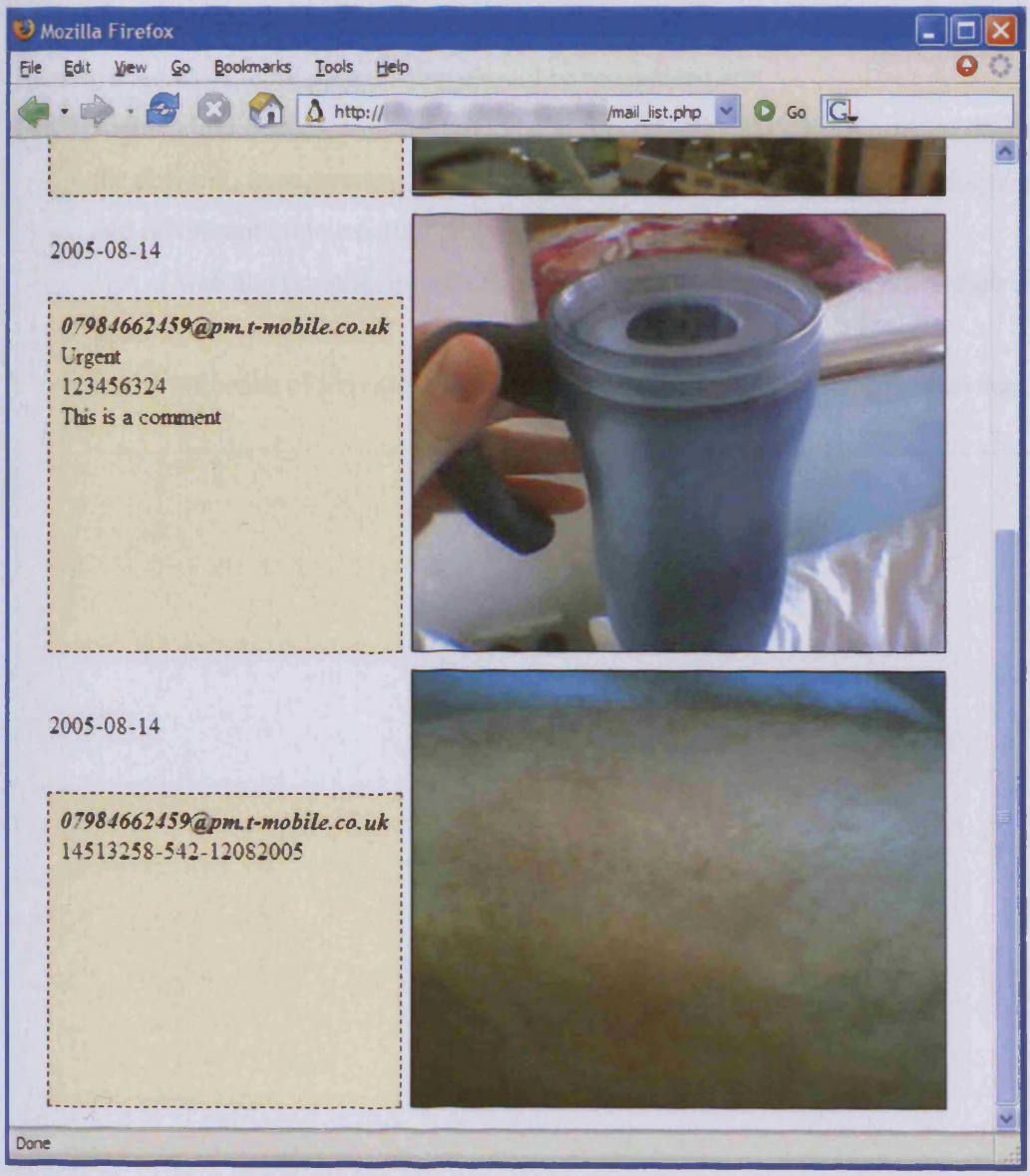


Figure 104. The mail_list.php script listing the e-mail data and images stored in the database

In this chapter, machine vision tasks were considered that fall outside the scope of conventional MV-driven targeting and quality control systems. These highly networked-focused tasks have involved networks, wireless connectivity, roaming, mobile devices, web servers and databases. Using machine vision techniques, visual input and automated physical output from these tasks may be merged with conventional networked systems and activities, expanding the range and efficiency of solutions.

The key features of these systems that should be recognised are:

- Lightweight inclusion of machine vision functionality (MV tools process data across the network, transparently, or whilst in a data store, requiring minimal programming and no disruption to existing programs).
- Use of web and portable interfaces, used as operators roam and perform their normal tasks
- Initial influence of Pervasive Computing into the workplace and universal data access

11 Conclusions & Future Work

11.1 Conclusion

This thesis has outlined Myriad, a distributed computing framework, with emphasis towards Machine Vision tasks. The concept of HTTP-based server components with in-built scripting language support has been outlined, with explanations how each of those features is relevant and beneficial to distributed computing in Machine Vision scenarios.

For the completion of this work, the following Myriad prototype components have been produced:

- Myriad Image Processing Engine server component
- Myriad Equipment Controller server component
- Prolog gateway system
- Database-backed access control gateway system
- GUI interfaces in a number of languages (C#, C++/QT, PHP->HTML, VB)
- C++/Qt early Myriad server core template

In addition, the computing elements for each of the examples have been produced (and are outlined in those sections and associated papers).

Throughout the creation of the Myriad prototypes, small testing scripts and programs have been produced to debug and test functionality. Rudimentary interfaces for handheld devices, such as PDAs and cellular telephones have been implemented and tested for the train control example, in order to illustrate utilisation of mobile technologies and wireless communication mechanisms.

Students at both BSc and MSc levels at Cardiff University have utilised Myriad components for a number of projects, including:

- Developing an AJ-100 equipment control implementation
- Tracking train movement automatically
- Targetting clocks in a scene and reading the time
- Calibrating pan & tilt camera systems

- Processing images using beyond-visual-range photography methods
- Controlling x,y,theta moving tables
- Developing visual acquisition and control for an existing classifier system

The combination of these endeavours has actively demonstrated the ability of Myriad components to support rapid development of vision-enabled applications in multiple programming languages through lightweight HTTP-based access to distributed computing resources.

11.2 Contributions & Analysis of Examples

This thesis contributes to both the fields of Machine Vision and Distributed Computing as reflections of each other.

In the field of Machine Vision, the thesis explores the implications and opportunities presented by adopting distributed computing techniques and networked computing tools for the expansion of Machine Vision tasks, away from stand-alone integrated vision systems typically deployed for factory-based inspection alongside production lines and towards more distributed solutions that less resemble current methods of work. In a distributed computing sense, questions of how to deal with consistent tasks through localised scripting, bridging conventional applications and network components and what new avenues of research are opened up by the idea of being able to apply vision methods to any networked application.

The example of the model train set, incorporating remote control of equipment and acquisition of visual data, for the interactive remote inspection of a factory environment for the creation and deployment of a target vision system illustrates the options for expanding conventional Machine Vision tasks in a non-disruptive, iterative manner. The example seeks to enhance the existing mode of operation by extension, rather than radical replacement. In terms of all of the examples contained within this work, the train set supports the current state of the Machine Vision industry and is of most immediate practical value to that avenue of endeavour. In distributed computing, the fusion of vision techniques, knowledge bases, mobile communication and and interactive camera systems is interesting for such applications as integrated monitoring, wearable computing devices and visually-enabled devices for tele-presence in a Pervasive Computing environment.

Looking at the integration of LinPacker and Myriad for the 2D packing problem, the benefits of distributed computing methods for Machine Vision are more actively explored, with remote data acquisition, utilisation with external applications through new generation messaging mechanisms and the options for large-scale, multi-site collaborative applications. The most compelling aspect of this example for contribution to Machine Vision, is perhaps the ability to exercise image processing and other Machine Vision methods in a conventional application through the judicious application of a network bridge script and Myriad distributed components. With the inclusion of such a bridge, any application is able to become a vision application and expand the focus of operation. It is, for example, quite reasonable to suggest that a word processing application with access to DCOP, DBUS or some other messaging and control mechanism would be able to include support for dictation through the medium of sign language. This would operate provided that one could capture a video stream from anywhere in the world, and feed it to a suitable Myriad component that was able process images and feed movement vectors to a classifier to identify gestures and take advantage of a separate knowledge base which could associate the gesture with a written term. All of these components could be non-local to each other, but the word processor could take advantage of the facility, without the need to completely programme the interface itself. In a Distributed Computing sense, it raises the question of how distributed computing techniques can affect traditional fixed applications through the simple extension of network access, in order to redefine applications with distributed tools. LinPacker was not created to be a networked application, nor was it designed to take advantage of visually-acquired data; yet, with the addition of an HTTP<->DCOP bridge, data to and from LinPacker is able to be manipulate on a network, and it is able to become a part of a much larger business process or multi-factory visual packing. Moreover, this enhancement is done with the minimum of additional scripting, posing a low threshold to entry for a range of other applications. In an even larger sense, the LinPacker example presents the option of connecting multiple different applications which similarly were never intended to be networked, to produce a meta-application, irrespective of locations of components around the world.

Monitoring a car park, using wireless cameras for space counting and collective multi-site administration is an interesting task in a Machine Vision sense, since it applies to an existing, practical problem that can benefit from a low cost vision system. Moveover, it provides features that are of direct use (space counting, security) that might be supported by analogue systems and an operator doing the task by hand, but for a relatively low cost. The example

demonstrates appropriate replacement of manual systems by computing systems, providing efficiency improvements for large car parks and features, such as automated free-spaces displays and aggregated company-wide monitoring. The first useful aspect of the example, is the ability of networked systems to multiplex large number of cameras for simultaneous acquisition of data, without requiring large numbers of monitoring and miles of expensive cabling. In extended research, questions of effective ways to include hundreds of cameras for monitoring and multi-sensor fusion for large-scale monitoring tasks are interesting, as they relate to broadcasting, security and swarm-based vision and environmental virtual resonstruction in Pervasive Computing environments. The second aspect of the car park example which is particularly useful, is the integration with remote business management and the ability to include data acquired from car park sites into web sites for user functionality and business planning. This type of easy integration with web-based systems and modern programming/scripting languages is a key benefit of Myriad's use of HTTP as a messaging and control mechanisms, and this type of neat bridging between web-based systems and distributed computing components is a key contribution to the field of distributed computing, as it illustrates the means and effectiveness of such a bridge.

Finally, the dermatological tracking example provides a good demonstration of how a distributed computing system can combine a traditional internal database-driven management/CRM system with new forms of visual data acquisition to improve workflows and reduce the locational difficulty of performing tasks. For the demonstration, simple web-based monitoring is provided, showing the ease at which Myriad networks can be managed using lightweight interfaces created using PHP/Perl to dynamically create HTML. Operational use of smart phones as capture devices, along with Java VM use and modes of image transportation were discussed, outlining many of the issues faced with the use of mobile computing devices for remote acquisition across inconsistent and unsecured networks. Considerations of lighting requirements and training needs for high quality image acquisition were discussed, and are important in uncontrolled environments. While not investigated in detail, the example does encourage future investigators to take the time to examine such issues, to ensure good image quality.

11.3 Lessons Learned

Throughout the following of this thesis and the creation of Myriad, a number of events and influencing factors have occurred to alter the structure of the final product.

11.3.1 XML

Initially, the means for communication between Myriad components was due to be XML, perhaps a simplified version of SOAP. The author interacted with a number of people for whom XML was an ideal mechanism for data storage and communication, but the actual use of XML in terms of interaction with parsers and the resource overhead of parser use is prohibitive. Prodominantly, programmers tend to envision their data as a linear sequence of variables, rather than in the hierarchical manner of XML, causing issues when faced with event-driven XML data file readers. Using XML parsers to import and export data to LinPacker (one of the author's contributions to the project), the immaturity of parsers was apparent, and the number of classes and interfaces that were required to be implemented to read less than a dozen variables was excessive. This is not to say that XML is not valuable for such tasks as configuration data storage, but that interface libraries above parsers (such as the .Net 'config' library) make XML reading highly effective. Unfortunately, most contemporary scripting and programming languages don't have such libraries, and this results in XML parsing being an onerous task for lightweight interfaces, small applications and applications with very simple XML requirements.

11.3.2 MCS Construction

The creation of MCS was far more complex than first envisioned, principally due to a chain of factors:

- Internal variable systems require string processing and interpolation mechanisms
- Internal variables require arithmetic operations, which themselves must consider interpolation and variable processing operations to resolve values
- A translation layer between incoming data and internal representation must be carefully crafted to ensure that commands are effectively delineated as they enter the system, but not to preclude commands embedded in variables as they enter the system (e.g. the pre-processor should not treat commands within strings as actual commands, until they are interpolated out). Moreover, this

translation layer should not affect incoming strings in ways that prevent retransmission as part of pass-through HTTP requests. This final issue is essential in the consideration of command delimiters, and is the cause of Myriad components using %0D%0A as a delimiter, rather than the classic unix \n carriage return character, since \n can be embedded and unaffected, with re-encoding prior to retransmission.

Interesting new movements in scripting languages towards systems such as Ruby on Rails and Python, whilst focusing on data-driven applications have diverged from traditional procedural scripting towards object orientation, data-centric processing and aspect-oriented programming paradigms. The full results of this movement on the means of programming simple procedural problems has yet to be realised, but there are also a new class of procedural languages both for programming and scripting that seek to fill the gap left by the migration of those languages, such as QSA, the Qt Script for Applications that comes with Qt 3.5 and Qt 4.0 installations. To a certain extent, MCS has been left behind, whilst being produced, by this change in language direction, but also engages neatly with these new procedural languages, and perhaps leads to a convenient means of implementing Myriad components using Qt and other leading toolkits, such as .Net 2.0.

11.3.3 Broader Networked and Distributed Computing tasks

In a greater sense, the project began with an intention to explore the opportunities for the development of traditional Machine Vision industrial inspection systems, with the inclusion of distributed computing methods, seeking to enable new options for operation of those systems and enhance the problems addressed to include the subset of problems that were hindered by locational limits such as hazardous environments. Throughout the execution of the project, however, initial thoughts that new classes of vision-inclusive distributed computing system might be achievable became more developed and more compelling. Primarily, the emergence of wireless networking and concepts of Pervasive Computing have become increasingly important in discussion of networked computing applications, from a point in 2001/2002 where wireless networking in the transparent Ethernet sense was really only just entering the public consciousness.

This development is most evident in the two classes of task that are addressed in the examples presented in this thesis:

- Traditional Machine Vision industrial inspection tasks, benefiting from distributed and networked enhancements for remote operation.
- Distributed and networked tasks that utilise visual data acquisition and processing as a means of interaction and are focused on the transacting of that data across a network, where the ability to perform operations on that data remotely is the primary benefit of the task at hand.

11.4 Future Work

There are many opportunities to continue the work outlined in this thesis, both in a practical sense, and also in a more theoretical manner. The next few years promise rapid growth of high-bandwidth networking:

- Wireless ethernet connectivity across wide areas and city-wide networks becoming a reality over the years 2006 and 2007
- Hybrid cellular/wireless ethernet devices and power-line connectivity being realised over the next five years, enabling widespread networking coverage with graceful degradation of performance as coverage declines
- High-bandwidth (>20mbps) small business and residential broadband internet access in the years 2006 and 2007

Combined with an increase in the diversity of network-capable computing devices:

- Palmtop computing devices with cameras, 3D graphics
- Smart Phones with built-in cameras, web browsing, Java support and hybrid wireless ethernet support
- Tablet PCs
- High-performance Set Top Boxes (STBs) for television with Personal Video Recorder (PVR) capabilities
- Network-enabled in-car satellite navigation and entertainment systems with heads-up displays

Both of these factors, along with the continued reduction in the cost of consumer computing devices lead to a conclusion of the impending emergence of Pervasive Computing as a significant technological change, affecting societal use of computer science in general.

Distributed computing initiatives, such as the GRID and the framework described in this thesis will increasingly become the glue between network-connected devices, desktop computers, servers and large-scale processing systems (e.g. clusters). Simplicity and transparency of network access and control become key factors as the number of devices and services available multiply, requiring frameworks to provide rapid distributed application development that supports portability and remote accessibility of data. Moreover, as the range of functions available through networked services increases, the need for flexibility and uniform interfaces to cope with the dynamic addition of services in applications will become a critical factor. In this type of environment, agent-based and plug-in-based systems are advantageous, allowing software modules to traverse networks to be present wherever users require them.

In this environment of Pervasive Computing, where a user's data and services are available from any device in any location, data entry and processing from those devices and locations becomes important, and visual acquisition through embedded cameras in PDAs, smart phones, web cams and network attached cameras are a key area of expansion for distributed applications. Tasks such as the Non-Hospitalised Dermatological Tracking example become increasingly practical as the quality of connectivity, camera performance and peripheral features (compensating flash bulbs) increase. In that example, vision techniques supported by recent hardware advances provided the basis for a new manner of performing a task (inspecting a wound) that improves efficiency of the medical assessment process and also encourages assessment of patients who might otherwise have difficulty in visiting a hospital, therefore improving treatment levels in the longer term.

In further developments of Myriad, there are a number of extensions that deserve exploration in order to expand the opportunities for use:

- A strongly-defined interface for additional functional modules would be beneficial, relying most likely on XML interface declaration to set up hooks in the command parser. This will enable the connection of external applications without recompilation and more dynamic integration of modules through downloading from networked repositories.
- Myriad would benefit from a range of administrative commands and modes of operation for monitoring, feedback and methods such as the dynamic reloading of

modules that change the purpose of the server component (e.g. unload the equipment control module, load the image processing module).

- Some form of long-term state saving for user sessions, beyond the simple non-destruction of sessions when the user ceases to respond.
- Support for user profiles, historys and security integrated with LDAP backends for state saving and component-level security. This is made difficult due to multi-server rights synchonisation issues.
- Research into best methods of user collaboration for tasks, including cross-component collaboration and implicit transportation of files and resources.
- C and C++ core server template versions to improve performance and support for complex data structures
- Optional use of Python, QSA (or another scripting language) as a replacement for MCS to reduce maintenance in terms of scripting
- DCOP, as used in the LinPacker example, has recently been rebuilt as DBUS, a system with the same functionality, except with enhancements for computer device control and an announcement protocol. Research into how this can be manipulated by Myriad components and bridged into Myriad networks for new tasks would yield interesting insights into web service, meta-application, computer and application interaction for the next decade.
- Expansion of Myriad components to take advantage of DNS Zero Configuration service announcements for indexing of components would be highly beneficial, once that standard becomes better-defined.

- Glossary

3G	3rd Generation mobile telephone technology
4G	4th Generation mobile telephone technology
ACL	Access Control List (user management system)
AD	Active Directory (user management system)
AP	Access Point (wireless networking)
API	Advanced Programming Interface
ASCII	American Standard Code for Information Interchange (character set)
ASP	Active Server Pages (scripting language)
B2B	Business 2 Business
BASIC	Beginners' All Purpose Symbolic Instruction Code
BSD	Berkeley Standard Distribution (Unix variant)
CAT-5	Category 5 UTP cable (networking)
CCD	Charged Couple Device
CDE	Common Desktop Environment (Sun Microsystems)
CF	Compact Flash flash memory device
CIP	Cardiff Image Processor
CORBA	Common Object Request Broker Architecture
CPU	Central Processing Unit
CRM	Customer Relationship Management
CS	Computer Science
DB	DataBase
DBMS	DataBase Management System
DCOP	Desktop COmmunications Protocol
DHCP	Dynamic Host Configuration Protocol
DTD	Document Type Definition
DTP	Desktop Publishing
EC	Equipment Controller
ECC	Error Correction Code
ECMA (script)	European Computer Manufacturer's Association (scripting language, formal specification based on Javascript)
EPROM	Erasable Programmable Read Only Memory
FIC	Flexible Inspection Cell

FTP	File Transfer Protocol
GPL	GNU General Public License
GSM	Global System for Mobile Communications
GTK	GIMP ToolKit
GUI	Graphical User Interface
HPC	High Performance Computing
HSCD	High Speed Cellular Data
HTML	Hyper-Text Markup Language
HTPC	Home Theatre Personal Computer
HTTP	Hyper-Text Transfer Protocol
IDE	Integrated Development Environment
IDS	Intrusion Detection System
IIS	Internet Information Server
IP	Image Processor, Image Processing
IP (address)	Internet Protocol address
IPE	Image Processing Engine
IS	Interactive Session (Myriad interaction mode)
J2ME	Java 2 Micro Edition
JFC	Java Foundation Classes
JMF	Java Media Framework
JVM	Java Virtual Machine
KDE	K Desktop Environment
KIO	KDE Input/Output system
LAN	Local Area Network
LCD	Liquid Crystal Display
LDAP	Lightweight Directory Access Protocol (user management system)
LISP	LIst Processing (programming language)
MAN	Metropolitan Area Network
MAV	Machine Assisted Vision
MCS	Myriad Component Script
MIME	Multipurpose Internet Mail Extensions
MMB	Mike's Magic Box (equipment control hardware)
MMC	Multi-Media Card flash memory device
MV	Machine Vision

NAC	Network Attached Camera
NDS	Netware Directory Service (user management system)
NIS	Network Information Service
NN	Neural Network
OO	Object Oriented / Object Orientation
ORB	Object Request Broker
OS	Operating System
OSS	Open Source Software
PCMCIA	Personal Computer Memory Card International Association
PDA	Personal Digital Assistant (palmtop computing device)
PHP	PHP: Hypertext Processor (scripting language)
PIP	Prolog Image Processing
PL/SQL	Procedural Language / Structured Querying Language (see SQL)
POTS	Plain Old Telephone System (analogue telephone network)
PP	Pre-Parsing (variable discovery method, Myriad)
QSA	QT Scripting for Applications (scripting language)
QT	Q Toolkit
QT/E	QT Embedded environment
RA	Runtime Allocation (variable discovery method, Myriad)
RAD	Rapid Application Development
RDP	Remote Desktop Protocol
RPC	Remote Procedure Call
RPI	Remote Procedure Invocation (see RPC)
RT	Real Time
RTSP	Real Time Streaming Protocol
SC	Single Command (Myriad interaction mode)
SD	Secure Digital flash memory device
SGML	Standard Generalized Markup Language
SMP	Symmetrical Multi-Processing
SOAP	Simple Object Access Protocol
SQL	Structured Querying Language
SS	Scripted Session (Myriad interaction mode)
SSH	Secure SHell
SSL	Secure Socket Layer (transmission encryption)

STB	(Television) Set Top Box
SWAN	Secure Wide Area Network (see WAN)
TCL	Tool Command Language
TCP/IP	Transmission Control Protocol / Internet Protocol
UCS	Universal Character Set (aka Unicode)
UDP	User Datagram Protocol
UPNP	Universal Plug N' Play (device control mechanism)
URL	Universal Resource Locator
USB	Universal Serial Bus
UTF (-8)	UCS Transformation Format (character set, 8 bit) (see UCS)
UTP	Unshielded Twisted Pair (cable type)
VB	Visual BASIC
VBA	Visual Basic scripting for Applications (scripting language)
VFS	Virtual File System (Gnome desktop environment)
VI	Virtual Instrument (LabVIEW components)
VM	Virtual Machine (see JVM)
VNC	Virtual Network Computing
VPN	Virtual Private Networking
WAN	Wide Area Network
WAP	Wireless Application Protocol
WiFi	Wireless Fidelity (802.11b wireless networking system)
WML	Wireless Markup Language (see WAP)
XHTML	eXtensible HTML (XML – HTML hybrid)
XML	eXtensible Markup Language
XSLT	eXtensible Stylsheet Language Transformations

References

[ALC] V. Alchanatis & A. Hetzroni & Y. Edan, "A Multispectral Imaging Sensor for Site Specific Application of Chemicals" ISHS Acta Horticulturae 2001

[ASH] C. Ashbacher, "Teach Yourself XML in 24 Hours", SAMS Publishing, 2000

[BALL] B. Ball & D. Pitts et al., "Red Hat Linux 7 Unleashed", SAMS Publishing, 2000

[BART] S.L.Bartlett et al. "Automatic solder joint inspection", IEEE Transactions on Pattern Analysis and Machine Intelligence, 1988

[BAT] B. Batchelor, "Intelligent Image Processing in Prolog", Springer Verlag London Ltd., 1991

[BAT2] B. Batchelor & F. Waltz, "Intelligent Machine Vision: Techniques, Implementations and Applications", Springer Verlag, 2001

[BAT3] B. Batchelor & P. Whelan, "Intelligent Vision Systems for Industry", Springer Verlag, 1997

[BAT4] B.G.Batchelor & B.K.Marlow, "Automated visual inspection of bottles, jars and tumblers", Sensor Review, 1982

[BAT94] B. Batchelor & M. Daley & E. Griffiths, "Hardware and Software for Prototyping Industrial Vision Systems", Proceedings of SPIE, October 1994

[BAT01] M. Daley & B. Batchelor, "Teleprototyping Environment for Machine Vision", Proceedings of SPIE, Feb 2001

[BERN] T. Berners-Lee & R. Fielding & H. Frystyk, RFC 1945 "Hypertext Transfer Protocol", May 1996

[BRAG] D. Braggins, "The Development of the Machine Vision Industry in Europe During the Decade of the 80s", Vision pp.13-19 - 13-28, 1990

[BRAT] I. Bratko, "Prolog: Programming for Artificial Intelligence" (2nd), Addison-Wesley, 1990

[CHAP] N. Chapman, "Perl: The Programmer's Companion", John Wiley & Sons, 1997

[DAL] M. Dalheimer, "Programming with Qt" (2nd), O'Reilly Verlag GmbH, 2002

[GER] Z. Geradts & J. Bijhold, "Pattern recognition and image processing in forensic science", Irish Machine Vision and Image Processing Conference, 2000 (also, http://geradts.com/html/Documents/IMVIP00/IMVIP.htm Sept. 2004)

[GON] R. Gonzalez & R. Woods, "Digital Image Processing", Addison-Wesley, 1992

[HAL] F. Halsall, "Data Communications, Computer Networks and Open Systems" (4th), Addison-Wesley, 1996

[HALE] J.A.G. Halé & P. Saraga, "The control of a printed circuit board drilling machine by visual feedback", Pattern Recognition: Ideas in Practice, 1978

[HAM] Hamlyn trust award, Centre for Criminal Justice Studies: Univ. Leeds, "UK Law Online", http://www.leeds.ac.uk/law/hamlyn/courtsys.htm Oct 2004

[HAN] U. Hansmann et al., "Pervasive Computing" (2nd), Springer-Verlag, 2003

[HANV] University of Hannover: Faculty of Horticulture, "Selected topics for MSc thesis work", Prof. Dr. Thomas Rath, http://www.gartenbau.uni-hannover.de/fb/msc/teaching-staff/rath.shtml Oct. 2004

[HERT] J. Hertz & A. Krogh & R. Palmer, "Introduction to the Theory of Neural Computation", Addison-Wesley, 1991

[HETZ] A. Hetzroni & G.E. Miles, "Machine Vision Monitoring of Plant Health." ASAE St. Joseph, 1992

[HUBB] Hubble Site, "Hubble Makes First Direct Measurements of Atmosphere on World Around another Star", http://hubblesite.org/newscenter/newsdesk/archive/releases/2001/38/text/ (viewed January 2005)

[JOHN] G. Johnson & R. Jennings, "LabVIEW Graphical Programming" (3rd), 2001 (1st, 1994)

[JYR] L. Jyrkinen, "Real-time image processing and visualisation brings medicine into the new age", Machine Vision News vol.4, 1999, http://www.automaatioseura.fi/jaostot/mvn/mvn4/medicine.html (viewed Sept. 2004)

[KELL] R.B.Kelley et al. "Three vision algorithms for acquiring workpieces from bins", Proceedings f the IEEE, 1983

[KON] Kondo, N., Y. Nishitsuji, P.P. Ling and K.C. Ting. "Visual feedback guided robotic cherry-tomato harvesting." Transactions of the ASAE, 1996

[LAN] USGS Landsat Project, http://landsat7.usgs.gov/, (viewed Sept. 2004)

[LEM] L. Lemay & R. Cadenhead, "Teach Yourself Java 1.2 in 21 Days" (3rd), SAMS Publishing, 1997

[LIB] J. Liberty, "Teach Yourself C++ in 21 Days" (3rd), SAMS Publishing, 1999

[LYON] D. Lyon, "Image Processing in Java", Prentice Hall, 1999

[MAR] J.A. Marchant, "A mechatronic approach to produce grading", Designing Intelligent Machines, 1990

[MCC] W. McCulloch & W. Pitts, "A Logical Calculus of Ideas Immanent in Nervous Activity", Bulletin of Mathematical Biophysics 5, 1943

[MED] D. Medinets, "Perl 5 by Example", QUE Corp., 1996

[MOOD] G. Moody, "Rebel Code: Linux and the Open Source Revolution", Perseus Publishing, 2001

[MYL] H. Myler & A. Weeks, "Computer Imaging Recipes in C", Prentice Hall, 1993

[MYL2] H. Myler, "Fundamentals of Machine Vision", SPIE, 1999

[NASA] NASA Headquarters Science Department , Universe discussion http://science.hq.nasa.gov/universe/index.html (viewed January 2005)

[NET] Netcraft Web Server Survey, http://www.netcraft.com

[NEVE] P. Neve, "Distributed Industrial Vision Systems", Machine Vision Systems Integration, 1991

[NOR] L. Norton-Wayne & M. Bradshaw. A.J. Jewell, "Machine vision inspection of web textile fabric", Proceedings of the British Machine Vision Conference, 1992

[NOV] J. Novotny et al., "An Online Credential Repository for the GRID: MyProxy", Proceedings of the IEEE International Symposium on High Performance Distributed Computing, 2001

[OTT] R. Otterbach et al., "Fast and robust recognition and localization of 2D objects", Proc. SPIE Vol. 2247, Sensors and Control for Automation, 1994

[PARK] J. R. Parks, "Industrial sensory devices", Pattern Recognition: Ideas in Practice, Plenum Publishing Corp. 1978

[PASS] P.J. Passmore et al., "Effects of Viewing and Orientation on Path Following in Medical Teleoperation." IEEE Virtual Reality 2001 Conference, Yokohama, Japan, pp109-216

[PEN] D. Penman & O. Olsson & C. Bowman, "Automatic inspection of reconstituted wood panels for surface defects", 'Machine Vision Applications, Architectures and Systems Integration', 1992

[PEN2] D. Penman & O. Olsson & D. Beach, "Automatic x-ray inspection of canned products for foreign material", 'Machine Vision Applications, Architectures and Systems Integration', 1992

[PER] G. Perry & S. Hettihewa, "Teach Yourself Visual Basic 6 in 24 Hours", SAMS Publishing, 1998

[PLE] R. Plew & R. Stephens, "Teach Yourself SQL in 24 Hours" (2nd), SAMS Publishing, 2000

[PRE] J. Preece et al., "Human-Computer Interaction", Addison-Wesley, 1994

[PRIG] I. Prigojin & A. Hetzroni, "Apple quality - prediction, control and assurance: system validation." ISHS Acta Horticulturae, 2001

[RAY] E. Raymond, "The Cathedral and the Bazaar", O'Reilly & Assoc., 1998

[RICH] T. Richardson et al., "Teleporting in an X Window System Environment", IEEE Personal Comm., No. 3, 1994

[RICH2] D. Richie, "The Development of the C Language", Association for Computing Machinery, Second History of Programming Languages conference, Cambridge, Mass., April, 1993

[ROB] K. Robinson & P. Whelan & J. Stack, "Segmentation of the Biliary Tree in MRCP Data", Proceedings of Opto 2002, SPIE, 2002

[ROD] L. Rodrigues, "Building Imaging Applications with Java Technology", Addison-Wesley, 2001

[RUM] J. Rumbaugh et al., "Object-Oriented Modeling and Design", Prentice Hall International, 1991

[SAL] P. Salus, "A Quarter Century of UNIX", Addison-Wesley, 1994

[SAN] J. Sanchez & M. Canton, "Space Image Processing", CRC Press, 1999

[SIN] Sinclair Optics, OSLO – Optical Design Software, http://www.sinopt.com (viewed November 2004)

[STER] L. Sterling & E. Shapiro, "The Art of Prolog", MIT Press, 1994

[SWE] D. Sweet et al., "KDE 2.0 Development", SAMS Publishing, 2000

[TENG] G. Teng & C. Li, "Development of Non-contact Measurement on Plant Growth in Greenhouses using Machine Vision", ASAE 2003

[THO] R. W. Thornton, "Driverless Tractors in the British Army of the 1980s", Proceedings of the 1st International Conference on Automated Guided Vehicles, 1981

[VAR] N. Varchaver, "The Patent King", Fortune magazine, 14th May 2001

[WALL] K. Wall & M. Watson & M. Whitis et al., "Linux Programming Unleashed", SAMS Publishing, 1999

[WALT] F. Waltz, H. Garnaoui, "Application of SKIPSM to binary morphology", Proc. SPIE Conf. on Machine Vision Applications, Architectures, and Systems Integration III Boston, 1994

[WAN] B. Wandell, "Foundations of Vision", Sinauer Associates Inc., 1995

[WAR] P. Ward, "Qt Programming for Linux and Windows 2000", Prentice Hall, 2000

[WELL] L. Welling & L. Thomson, "PHP and MySQL Web Development", SAMS Publishing, 2001

[WELL2] G. Wells & E. Baroth, "Telemetry Monitoring and Display Using LabVIEW" National Instruments User Symposium, 1993

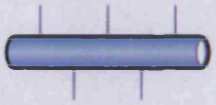
[WHI] P. Whitehead & E. Friedman-Hill & E. Vander Veer, "Java and XML", Wiley Publishing, 2002

[ZEM] Zemax Development Corporation, Zemax Optical Design Program, http://www.zemax.com (viewed November 2004)

Key Of Diagrams

General

User



Wired Network

Unwired Network

Wireless Access Point

Firewall

Networked Camera (Generic)

Monitor

Laptop Computer

Desktop Computer

Palmtop Computing device

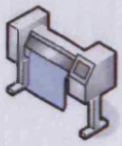
Gateway System

Mainframe

Telephone

Cellular/Mobile Telephone

Smart Phone



Plotter/Printer

Servers

Server

User Directory Server

Database Server

Mobile Data Server

Web Server

FTP Server

Real-Time Communications Server
(Instant Messaging, Internet Phone, etc.)

Streaming Media Server (RTSP, etc.)

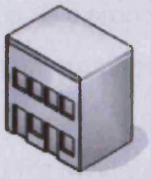
Authentication

User Authentication Details

User Authentication System

Locations

House (User, Staff home)



Building (Office, Company)

Town/City area

Undefined area, typically the Internet

Appendix A: Additional Technology Review

A.1 Neural Networks

Machine Vision, is a process of recognition and response using computers to attempt to solve problems through visual input. While computers are capable of solving some such problems, very often the human brain is far more advanced in obtaining the solution than the computer; hence the need for Machine Vision as a studied discipline. As a result, there are regular comparisons of Vision techniques to human activity, and this is justifiable, as one of the most frequent applications of MV as a whole, is to 'replace people' in workplace tasks e.g. Dissecting plants [BAT-88]. The ethics of doing so in certain situations can be dubious (loss of jobs being a critical result), but parallels are regularly drawn.

Hertz, Krogh & Palmer [HERT] define the brain is being superior for certain tasks, over and above computers for the following reasons:

- Robust and fault tolerant
- Flexible – can learn new environments, without reprogramming
- Able to manage data which is noisy, inconsistent, ambiguous and probable
- High parallelisation
- Small, compact and low power

The authors state that in the majority of tasks, the human brain exceeds the processing abilities and operational performance of a computer to a high degree, although they do concede that in discrete logic and arithmetic, computing devices typically dominate.

The functionality of an animal brain is interesting in the context of Machine Vision, in that human beings and computers do not see in the same way; humans use intelligence and knowledge to aid recognition [BAT] and while computers are able to process discrete logic, they often fail in more generalised situations, where a person is able to utilise Associative Memory (AM) as outlined by Hertz, Krogh & Palmer [HERT]. Associative Memory is the process of using memory of similar situations to those being experienced to compensate for lack of clarity in recognition.

The central case for reflection on the design of animal brains, is that there are many problems in where there exist no algorithmic solutions, or the solutions are dependent on relationships to existing data, which cannot be algorithmically expressed prior to operation, yet the application of AM is able to provide solutions.

To solve these problems, in 1943, McCulloch and Pitts [MCC] outlined a model for an interconnected network of nodes which solved problems in a collaborative manner. We presently refer to these models as Neural Networks (NNs). NNs are created to mimic animal brain structures and methods of learning. In essence, an NN exists as a set of operational nodes, which are heavily interconnected; the connections of nodes carry a weight number to indicate the value of a given path of logic, much like interconnected brain neurons. The core of a Neural Network system, is an iterative process, where a collection of sample data is given to the NN for processing, and a learning algorithm proceeds to teach the network and add weights to the system. As supervised learning takes place, solutions that the supervising person classifies as being correct add weight to connections that lead to the solution, and de-weight unhelpful connections, dynamically creating a solution algorithm. The performance of NNs is very much dependent on a number of factors:

- The quality and clarity of the training data
- How well the training data represents the board cross-section of all potential future input data
- The learning algorithm used to teach the system and add weights
- The accuracy of the human classifications of the data used to teach the system

It is also possible to use algorithms with success criteria to train a Neural Network, rather than have a human being provide the classifications for the data, which is discussed by Hertz et al. [HERT], but is not relevant here.

Neural Networks are of great benefit to Machine Vision systems, due the variety, ambiguity and noise, which occur in less than the most controlled (uniformity of objects, position, lighting, focus/zoom) situations. Hertz et al. describe the problem of being one of 'image reconstruction', where an imperfect image is modified to smooth out noise, non-uniform variation and per-pixel object discontinuity, with the intention to restore solid objects, lines, forms and spaces. They do however, caution that such problems, by definition, include "assumptions or a priori knowledge about the image", which constrain the operation of the

system with a n understanding of the scene being observed. Should these assumptions be false or absent, the resulting image cannot be considered to be any more insightful than the original imperfect source.

While Myriad does not include a Neural Network component in the initial demonstration systems, it should be noted that an NN may be built into a server component, or connected to an existing Myriad component (see fig.105). As a result, with additional development, Myriad systems can be developed to include NN benefits, most especially to compensate for generalised situations (beneficial in dynamic problems, roaming activities and prototyping situations). In the case of remote inspection problem, a Myriad component containing a Neural Network may be used to pre-filter images coming from the remote site, to clean up objects, highlight objects similar to those which are known from previous experience and to offer classification of the scene to the Vision Engineer.

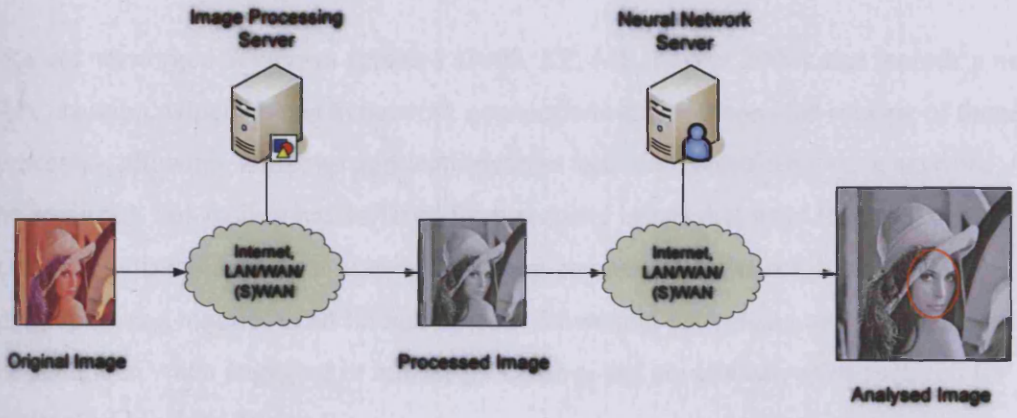


Figure 105 : Using chains of networked MV component servers to perform complex processing tasks

A.2 RPC

Remote Procedure Calling (RPC) is the term used for systems which are able to have requests sent to them over networks, or from local, but external programs, which initiate operations within the system.

A.2.1 Java RPC

With the refocusing of Java toward business to business (b2b) operations, Java's RPC framework has become well fleshed out; although it does still suffer from the JFC's issues of poor naming, counter-intuitive operation and excessive numbers of conflicting structures and frameworks. Java is also able to serialise objects, that is, it is able to take Java language

objects, create a representation of them and then reconstruct the object on the other side of the connection, albeit only for simple objects.

Java's RPC mechanism is able to interoperate with CORBA systems for auto-discovery of capabilities, a valuable ability.

A.2.2 Windows RPC

The Microsoft Win32 programming interface has extensive RPC facilities, along with additional message passing and control mechanisms which are exposed to all Windows applications, and this has been the case since the introduction of Win32 with Windows 95 and resolution in Windows 98. The actual form of the RPC mechanism, developers' facilities and means of using it vary greatly in 32bit and 64bit Windows environments, due the wide range of APIs, programming languages and their iterations (Visual C++ iterations (along with numerous core library variations), Visual Basic 4,5,6, .Net, C#, J++, etc.).

Recent networked Windows versions (2000, XP, ME, Server 2003) also include a network RPC daemon, which listens to network connections and manages the routing of those to local processes, allowing Windows applications to be actively controlled over a network. As might be predicted, this facility has suffered from security issues that have lead to it being commonly disabled on most network-attached computers; however, it does offer programmers a number of additional options for remote processing and management that can be beneficial when engaging in remote processing and control activities required for networked Machine Vision.

A.2.3 CORBA

Using a central server application, known as an ORB as a hub, CORBA (Common Object Request Broker Architecture) provides an RPC system with a number of added benefits. The ORB functions as a broker and an index of all the available services in a system or on a network. When new services are added, they announce themselves to the ORB and are indexed. When particular services are needed by a piece of software, it is able to consult the ORB, which directs it towards the appropriate service. The ORB also provides data translation services through IDL interfaces where two systems use different ways to format and serialise their data, allowing communications to be free of issues such as conflicting endian-ness.

Due to the complexity required of ORBs, they tend to be hard to develop and it is generally recommended that a 3rd party ORB be used. Due to the data moderation by the ORB and the time taken consulting the index and translating data formats, CORBA tends to be slow in use, and with high latencies. For these reasons, CORBA is best suited to low-data systems with large numbers of services that could not be easily searched in other ways.

Appendix B: Basics of Image Processing

Image Processing is the process of manipulating the data of a digital image, to convert it into a new form (more usable for the task), or to convert it and derive information as the result (e.g. the locations of objects within a scene).

Fundamental image processing techniques began with photographic manipulation tools, capable of processing film to alter the represented image. From this age originate a number of familiar IP operations, such as:

- Negation
- Thresholding
- Multi-image compositing
- Cropping
- Intensity-shifting
- Colour replacement
- Blurring and sharpening
- Contrast adjustment
- Rescaling
- Distortion and correction

In point of fact, one of the areas of modern image processing application is the use of IP tools to investigate the manipulation of photographic and video materials in forensic science.

Image Processing is a multi-stage process (see fig.106) involved with the pixel-level examination of an image. The stages are as follows:

- Pre-processing
- Core processing

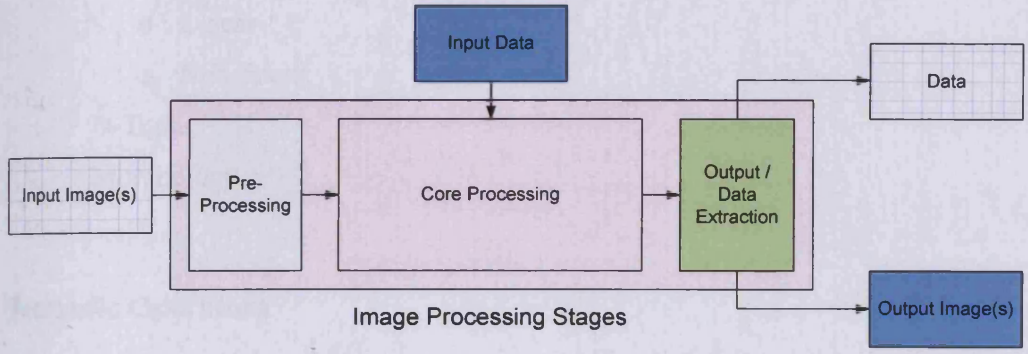


Figure 106. Diagram of the stages of image processing to perform a generalised single-image operation

Pre-Processing

In most IP tasks, images acquired are imperfect. They contain artefacts, distortions and results from the environment, optics and acquisition method. Commonly, these artefacts will include:

- CCD generated noise in reduced lighting
- Noise
- Dust
- Glare from lighting
- Optical distortion
- Encoding/compression artefacts
- Errant objects within the scene
- CCD and lens imperfections (dead pixels or localised point distortions)

In order to work with an image to obtain high-level information, these defects must be removed. To do this, the techniques named above are traditionally used, processing the image prior to the main operation; hence the definition of Pre-processing.

Core Processing

IP Operation Classes

Image Processing operations can be broken into distinct classes, based on the way that they transform the pixels of an image. The classes are:

- Monadic
- Dyadic
- Local Operators

  - Linear
  - Non-linear
- N-Tuple
- Morphology

Monadic Operators

In many case of image processing, there are needs for operations that are able to perform a transformation across all pixels of an image uniformly. These include:

- Negation
- Threshold
- Intensity shifting & multiplication

These are reliant on the contents of a single pixel within a single source image, which must be operated upon, and the result is mapped into a corresponding pixel in the transformed image (see fig.107).

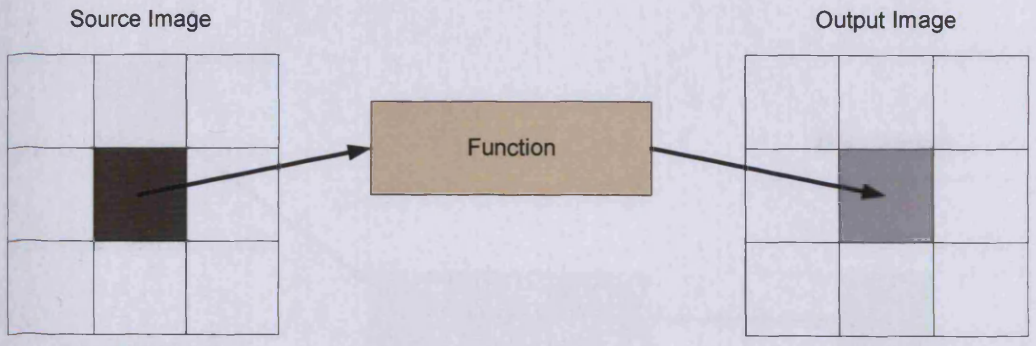


Figure 107. Diagram of the general principle of monadic image processing operations

Manipulating pixel data in this manner is the application of linear algebraic equations to an image, with the pixel value being a variable and the result of the equation forming the transformed image. These are the most simple of image processing tools, as they do not rely on additional images, nor the qualities of any other elements of an image, including the context surrounding the examined pixel. Monadic operators are the most frequent constituents of pre-processing sequences, where defects must be corrected, but no awareness of the contents of the image as a whole is required in order to perform what is a 'clean-up' task (see fig.108).

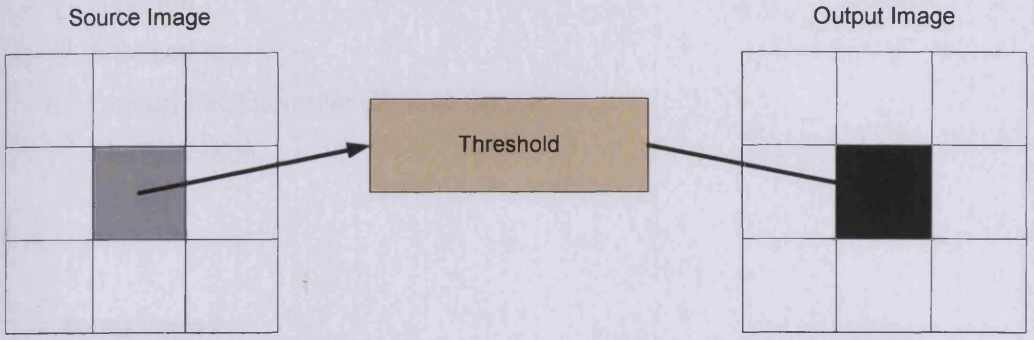


Figure 108. The outline of a simplified threshold operation on an image and the resultant output

Dyadic Operators

In contrast to monadic operators, dyadic ones involve the application of linear algebra to two images (see fig.109), the interactions of which form a single resultant image. Dyadic operators, therefore, perform compositing and contrasting tasks.

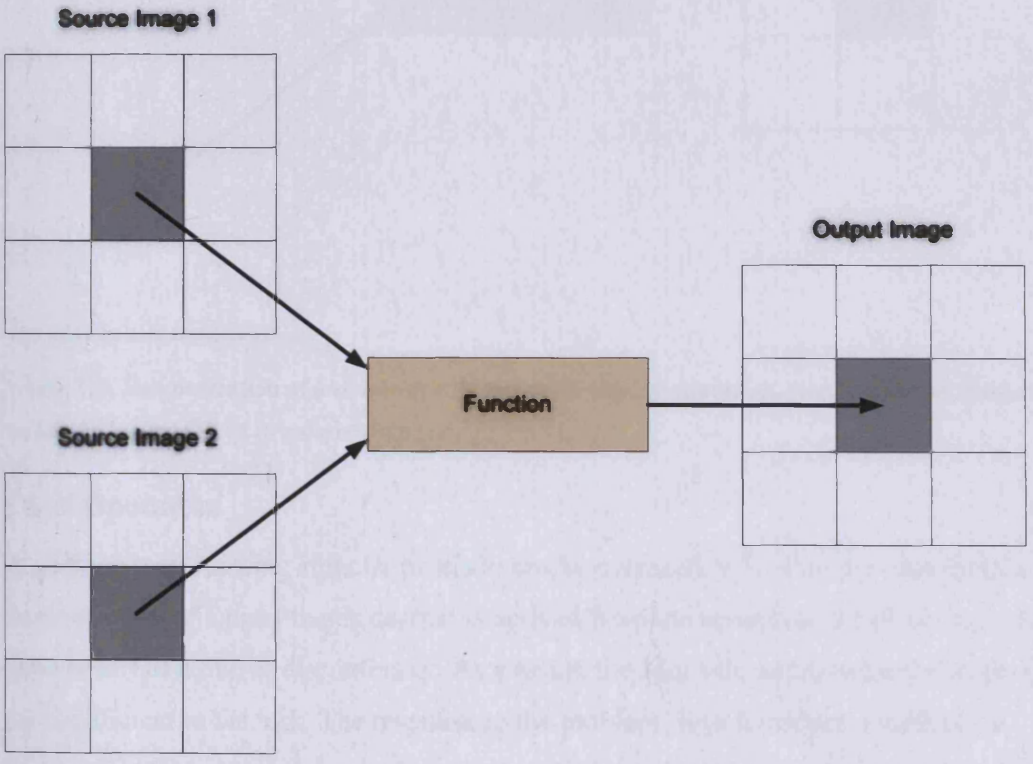


Figure 109. Diagram of the mean of operation of dyadic image processing functions, incorporating two images as input to a single function and constructing a single image using the output from that function

These include:

- Addition
- Subtraction
- Logical OR/Exclusive OR (see fig.110)
- Multiplication
- Division

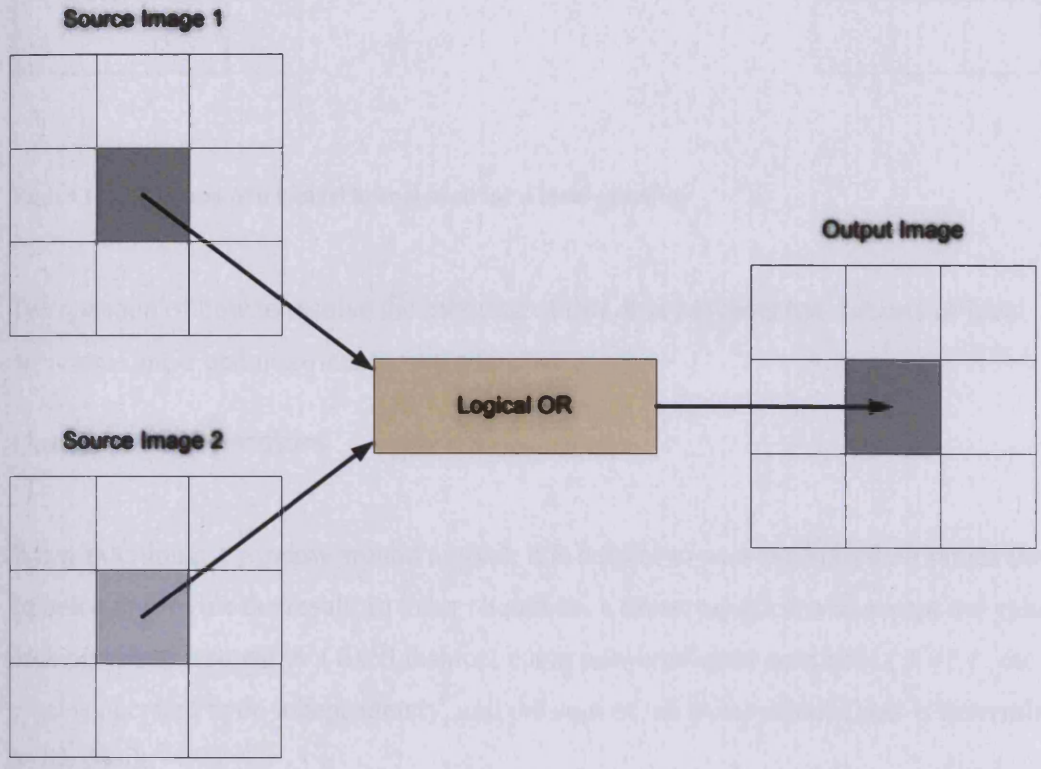


Figure 110. Demonstration of a simple dyadic image processing operation, constructing an image as the result of a logical OR of two source images

Local Operators

In addition to processing singular pixels to produce a result, it is often the case that the understanding of a pixel that is desired is derived from the context of the pixel, e.g. whether a point is part of a line or discontinuity. As a result, the Monadic and Dyadic forms of operator are insufficient to the task. The response to the problem, is to introduce a method of appreciating the surrounding pixels around a point, passing the complete collection through an equation and merging the data into an output pixel in the resultant image. Local operators, therefore, take a regularly-shaped matrix surrounding the source pixel, perform some processing and provide an output (see fig.111).

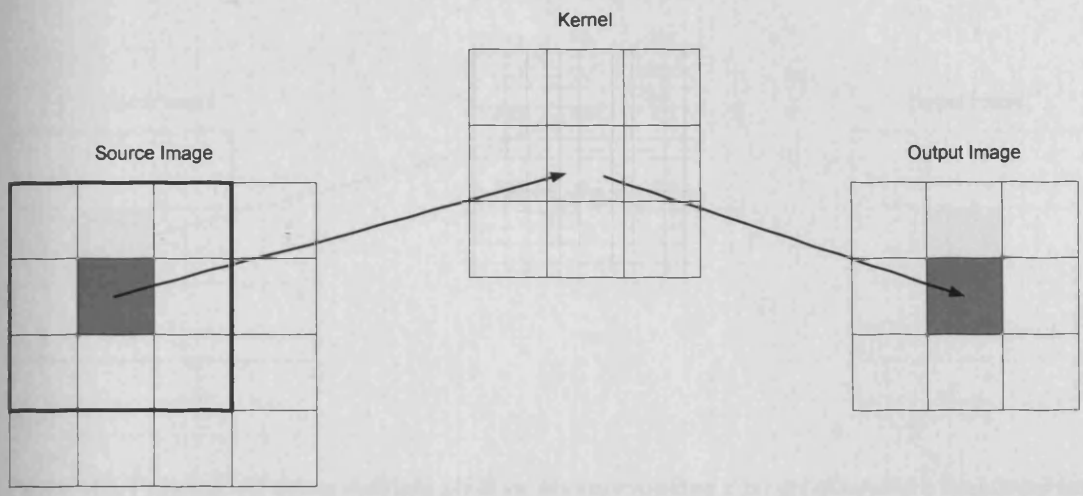


Figure 111. Diagram of a 9-pixel kernel used for a local operator

The question of how to resolve the merging of this data provides two subsets of local operators: linear and non-linear.

Linear Local Operators

When examining a window around a pixel, it is helpful to pass the bounding pixels through an equation to provide the result. In these situations, a linear equation will accept the values of each pixel and respond in a fixed fashion, using non-intelligent operators (+ - * / , etc.). Each pixel is operated upon independently, and the sum of all point calculations is determined to be the result.

In point of fact, since each pixel is processed independently, the process may be represented as a matrix corresponding to the window, with each pixel-equated element being filled by the function that is applied to the pixel (see fig.112).

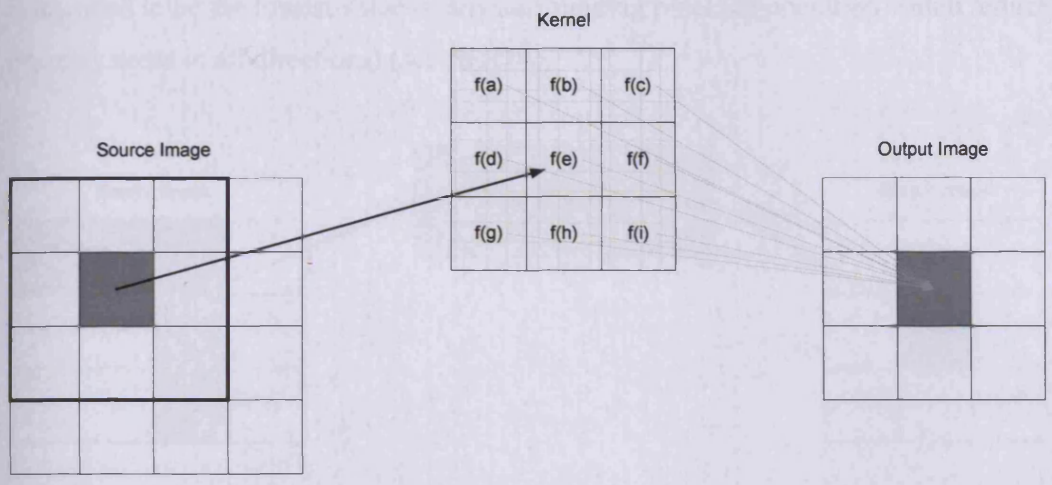


Figure 112. Principle of using multiple pixel values surrounding a target pixel and a functional kernel to develop an output image from an original source image

This matrix is commonly referred to as the 'kernel' and the passing of the kernel over the image, as 'convolution' (a 'kernel' is 'convolved' across the image), with the matrix containing multiplying factors for each pixel (see fig.113).

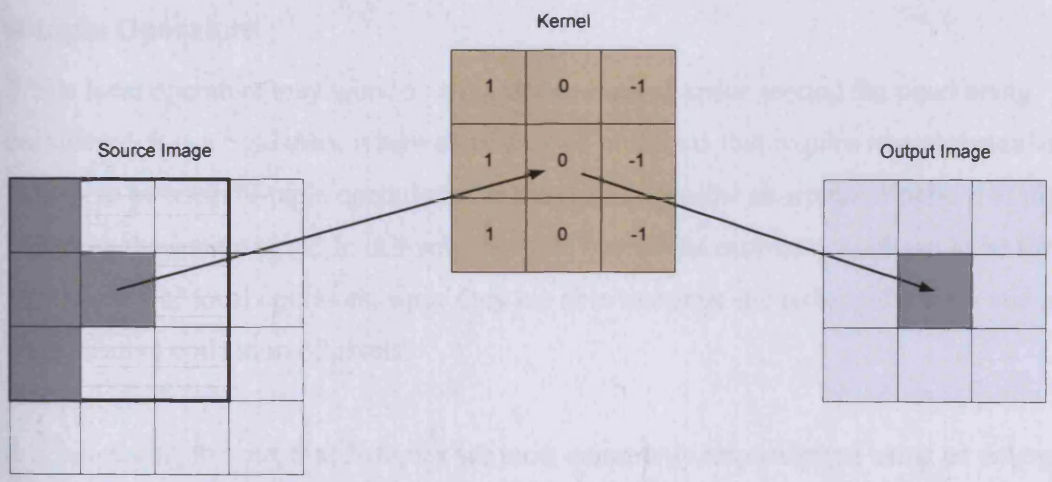


Figure 113. Demonstration of the output of a simple edge-detection kernel to provide an image processing operation

Non-linear Local Operators

In certain cases, is it important to be able to examine pixels intelligently, and not as independent equations which are summed. One such situation would be when the target pixel

is intended to be the lowest value of any surrounding pixel (an operation which reduces high-intensity areas in all directions) (see fig.114).

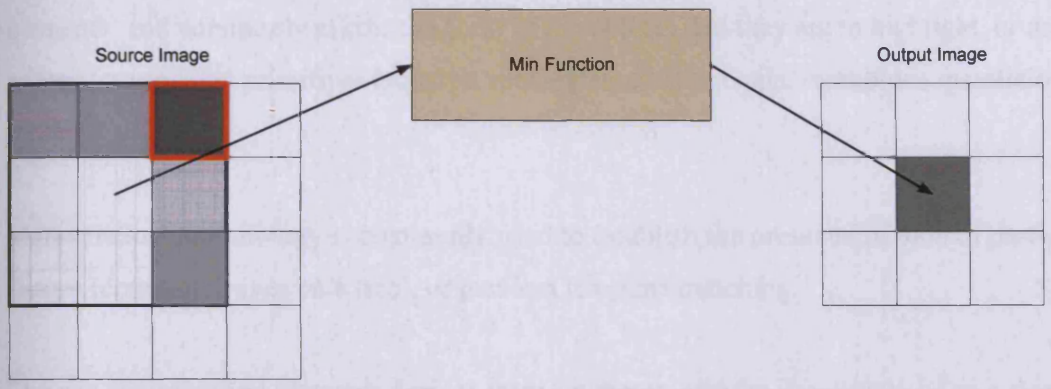


Figure 114. Using a non-linear local operator to find the minimal intensity value in a surrounding region and assign that to an output pixel

In these situations, a pixel value must actively be chosen, and the pixel used to determine the final value will vary, depending on the point in the source image, a process that cannot be expressed as a fixed matrix of multiplying factors.

N-tuple Operators

While local operators may work on a regular connected space around the pixel being considered, it is a rigid area, where there may be problems that require non-rectangular kernels to be used. N-tuple operations are those that consider an irregular pattern of pixels including the source pixel. In this way, N-tuple operations may be considered to be the most general case of local operators, since they are able to mimic the rectangular form and any other relative collection of pixels.

It is interesting to note, that N-tuples are most commonly implemented using an enlarged local operator rectangular kernel, with no weighting on unused pixels.

Mathematical Morphology

It is frequently the case in Machine Vision applications, that information must be gained about the structures of objects within a scene. Rather than processing to correct form, or manipulate the image, the intention is to understand the nature of the shapes presented in the

image, either in order to modify the forms, match shapes to a template, or produce a structural map of objects, such as skeletons. In these cases, a template is convolved over the image, and note made of where it matches visible features. These templates are known as 'structuring elements' and commonly mirror the form of the objects that they are to highlight, or are related to graphical primitives (squares, rectangles, circles, ovals, rhomboids, parallelograms, toroids).

Mathematical morphology is commonly used to establish the presence of blob of particular shapes (counting leaves on a tree), or perform template matching.

The use of structuring elements derives from set theory and the description of an object as a bounded set within an otherwise empty pixel space. When the set of the object is convolved with the set of a structuring element, two outcomes are possible:

- Erosion
- Dilation

In the case of the first, the desired output is a set of points where the structuring element may be positioned completely within the object such that the points in the structuring element and the object are both non-zero. Since placing a structuring element at the edge of an object will leave some of the pixels united with the element's pixels outside the object itself, this operation reduces the size of the object; hence the object seems to have edges eroded away; erosion.

Dilation is the opposite to erosion, in that it is concerned with the set of where the object and structuring element meet least (yet still meet), thus maximising the output area.

From these two operations, a number of further operations may be derived, such as Opening, which smooths the edges of an object, removes narrow connections between blobs and eliminates small blobs. This is achieved by performing an erosion operation followed by a dilation. The opposite of Opening is a Closing operation, which performs a dilation followed by an erosion. This also smooths edges of objects, but also serves to fill thin gaps and merge loosely-connected blobs.

Morphology continues to provide more complex operations, most of which are unsuitable for brief description; however, listed, they include:

- Boundary extraction
- Filling of regions
- Blob connecting (including island bridging)
- Convex hull
- Skeletonisation and finding of limb ends/joints

Appendix C: MCS

C.1 Myriad Command List

Algebra

<$var>++

<$var>--

Use: Increment/Decrement variable by +-1

<$var> := <String (interpolated): Assignment>

Use: Assigns the RHS string to the variable, "" to interpolate string, '' to not

<$var> .= <String (interpolated): Assignment>

Use: Concatenates the RHS string to the variable, "" to interpolate string, '' to not

<$var> = <Expression/Variable/Number: NumAssignment>

Use: Numerical assignment, Assigns the RHS to the variable, treated as a variable, number or complete expression

<$var> += <<Expression/Variable/Number: NumAssignment>

<$var> -= <<Expression/Variable/Number: NumAssignment>

<$var> *= <<Expression/Variable/Number: NumAssignment>

<$var> /= <<Expression/Variable/Number: NumAssignment>

<$var> %= <<Expression/Variable/Number: NumAssignment>

Use: Compound numerical alteration. (+= adds NumAssignment to $var, %= does the remainder)

General Operation

start

Synonyms:begin

Use: Begins a persistent scripting session

stop

Synonyms:end

Use: Ends a persistent scripting session

***<String: Label>

Use: Marks a point in the code to jump to

//

#

Use: Denotes a line of comment

goto <Variable: $var | String: Label | Integer: Line Number>

Synonyms: jump

Use: Jumps to a given point in the code

gosub <String: Subroutine Name>

Use: Jumps to a subroutine

File & Database I/O

openfile <String: FileHandle> <String (interpolated): Filename>

Use: Opens a file (normally text) for writing on local storage device

printfile <String: FileHandle> <String (interpolated): Output>

Use: Prints the output to the file with the FileHandle, "" to interpolate string, '' to not

printlnfile <String: FileHandle> <String (interpolated): Output>

Use: Prints the output to the file with the FileHandle, "" to interpolate string, '' to not, output ends with a '\n' newline character

closefile <String: FileHandle>

Use: Closes the file and filehandle

dbconnect <String: DB Connect String>

Use: Opens a connection to the database. The string is in single URL form, eg.

"jdbc:mysql://<host>/<database>?user=<username>&password=<password>"

dbclose

Use: Closes the current database connection

dbquery <String: Query>

Synonyms: dbread

Use: Executes a 'select' query on the database and readies the result set for reading with 'dbresult'. Query is a standard SQL query, use quotation marks around it to activate interpolation, apostrophes for no interpolation

dbresult

Use: Partially implemented, this reads the results of a query and outputs them to the server command line

dbupdate <String: Query>

Synonyms: dbwrite

Use: Executes a query altering the database. Query is a standard SQL query, use quotation marks around it to activate interpolation, apostrophes for no interpolation

Image I/O & Manipulation

openimagefile <String: ImageHandle> <String (interpolated): Filename>

Use: Opens an image file from local storage device, assigns to ImageHandle

downloadimagefile <String: ImageHandle> <String: URL>

Synonyms: openfile

Use: Opens an image file from a URL, assigns to ImagHandle

saveimagefile <String: ImageHandle> <String (interpolated): Filename> <Int(0-100): Quality> <String (jpeg|jpg|gif): Type>

Use: Saves an image to a file, in format Type at Quality

saveimage <String (interpolated): Filename> <Int(0-100): Quality> <String (jpeg|jpg|gif): Type>

Use: Saves the current image to a file, in format Type at Quality

use <String: ImageHandle>
Use: Selects a image handle and makes it the current image

store <String: ImageHandle>
Use: Reverse of 'use', stores the current image under an ImageHandle

makeimage <String: ImageHandle> <String(r|g|b|i): Channel> [<Int: Width> <Int: Height>]
Use: Builds an image with the ImageHandle from the current channel (implicitly the current CIP image, applying it as the channel, constrained by width & height

mergeimage <String: ImageHandle> <String(r|g|b|i): Channel> [<Int: Width> <Int: Height> [<Int: X Origin> <Int: Y Origin> [<Int: Current Width> <Int: Current Height>]]]
Use: Like makeimage, this command merges a channel with an existing image, constraining the input image to width & height, merging it with the top left corner at X,Y and constraining the image being merged into to Current Width & Height

stackchannel <String: ImageHandle> <String(r|g|b|i): Channel> [<Int: Width> <Int: Height>]
Use: Does makeimage with the current channel and then adds it to the top of the history stack

savestack
Use: Saves each image on the stack to the local storage device as stackimageX.gif where X is an integer denoting distance from the top of the stack

savestackthumbs
Use: The same as savestack, except that it saves thumbnails rather than complete images. Useful for debugging and producing user interfaces where the user needs to see the history stack

alternate
Synonyms: alt
Use: Replaces the current image with the previous image, currently on the top of the stack

Stack <Int: Stack Number>

Use: Like alt, this replaces the current image with an image an arbitrary (Stack Number) distance from the top of the stack

colourneg

Use: Negates the current image across all channels

red <String: ImageHandle>

green <String: ImageHandle>

blue <String: ImageHandle>

intensity <String: ImageHandle>

Use: Extracts the r, g, b or i channel from the ImageHandle image and makes a one-channel image as the current image, ready for processing

Image Processing Commands

Command	Description	Defaults
acn <Int: Level>	Adds level to the intensity of each pixel	
and	Binary AND the current & alternate images	
bed	Binary edge detector	
big [<Variable|Int: Blob Number>]	Finds the biggest (or biggest – Blob Number) blob in the image (use after blb)	Blob Number = 1
blb	Finds all blobs in a binary image	
bpt	Draws the bottom chord	
btl <Variable: Result>	*Part of blob detection*	
cal <Int: Level>	Copy only pixels with intensity > level, otherwise reduce to 0	
cbl	Counts binary blobs in an image (use after blb)	
cin	Integrate intensities across columns	
clc	Column run-length coding	
cnw	Count white neighbours of each pixel	
cox	Column maximum	
crp	Crops the image to the minimum area rectangle around white pixels	
csh	Copy intensities vertically from the bottom of the	

	image	
cwp <Variable: Result> [<Int: Level>]	Counts white (or intensity = Level) points in an image and stores the result in the user-specified variable	Level = 255
dbn	Direction of brightest neighbour	
dcl <Int: X> <Int: Level Y>	Draw cross lines	X = 0, Y = 0
dct	Draw colour triangle	
dil [<Int(4|8): Connectivity>]	Dilates white areas, based on Connectivity	Connectivity = 4
dld [<Int: Direction>]	Dilate white areas in Direction	Direction = 0
edd [<Int: Parameter>]	Non-linear edge detector Parameter = 0 : pixel = max of local pixels – pixel Parameter = 1 : pixel = pixel – min of local pixels Parameter = 2 : pixel = max of local pixels – min of local pixels	Parameter = 2
enc	Enhance contrast	
erd [<Int: Direction>]	Erode white areas in Direction	Direction = 0
ero [<Int(4|8): Connectivity>]	Erode white areas, based on Connectivity	Connectivity = 4
eul <Variable: Result>	Stores the Euler number of the image in the user-specified variable	
exp	Exponential intensity transformation	
exw	Expand white areas	
fcc		
getheight <Variable: Result>	Stores the image height in the user-specified variable	
getwidth <Variable: Result>	Stores the image width in the user-specified variable	
gft	Grass fire transform	
gra	Gradient edge detector	
grey	Converts image to greyscale	
gry <Int: Level>	Set all pixel intensities to Level	
hgr	Horizontal gradient	
hid		
hil <Int: Low> <Int: High> <Int: Value> <String(true|false): Threshold>	Highlight – Make all pixels between Low and High intensity Value. Set Threshold to true to threshold all other pixels	
hin	Halves intensities of all pixels	
hpf	High pass filter. Sharpens an image.	
inh	Horizontally invert the image	
inv	Vertically invert the image	

ior	Binary inclusive OR the current & alternate images	
jnt	Finds the joints of the skeleton	
kgr <Int: Num of Pixels>	Keeps blobs with pixels > Num of Pixels in image. Use after blb.	
ksm <Int: Num of Pixels>	Keeps blobs with pixels < Num of Pixels in image. Use after blb.	
lak	Finds lakes (holes) in a binary blob	
lav [<Int: Level>]		Level = 8
lgr	Largest gradient	
lme	Finds the limb ends of a skeleton	
log	Log transformation of all intensities	
lpc [<Int: Level>]		Level = 8
lpf	Low pass filter. Blurs an image.	
lpt	Draws the left chord	
lrt	Left to right transformation	
mar	Minimum area rectangle marked around blob	
max	Per pixel, find the maximum of the current & alternate images	
mcn [<Int: Constant>]	Multiplies all pixel intensities by Constant, and then normalises	
mdf [<Int: Level>]		Level = 5
min	Per pixel, find the minimum of the current & alternate images	
ndo		
neg	Negate the image	
per <Variable: Result>	Perimeter	
rid		
rin	Integrate intensities down rows of an image	
rlc	Row run-length coding	
rnd	Fills the image with pixels of random intensity	
rob	Roberts edge detector	
rox	Row maximum	
rpt	Draws the right chord	
rsh	Copy intensities horizontally from the right of the image	
sed	Sobel edge detector	
shp [<Int(4|8): Connectivity>]	Sharpen image	Connectivity = 8
skw	Shrink white areas	
sqr	Squares all pixel intensities in the image	

sqt	Square roots all pixel intensities in the image	
thr [<Int: Low Level> <Int: High Level>]	Thresholds an image	Low = Average Intensity, High = 255
tpt	Draws the top chord	
tra	Trace blob edges	
vgr	Vertical gradient	
xor	Binary XOR the current & alternate images	
yxt	Flip x and y axis	

C.2 Variables

Variables in Myriad systems are prefixed with a dollar ($) symbol and are stored as strings. These variables are dynamically cast out when used to strings, doubles floats or integers.

Variables are also recursively evaluated, so if a variable contains the name of another variable, the second variable is evaluated and that value returned.

Example:

$x = 5
$y = $x
print $y

Returns: 5

Key system variables, such as the file to return to the user and the current location of the command pointer are stored as user-accessible variables, prefixed with 'system_' in the variable name (eg. $system_returnfile). The file to return is useful in allowing the system to process the file that has been requested and to alter the file returned based on logical conditions.

For example:

If $lightlevel > 5

```
{
$system_returnfile = bright.jpg
}
else
{
$system_returnfile = dark.jpg
}
```

The command pointer is also useful in that it allows the program to interrogate itself and the user is able to carefully manage the flow of a script, with the ability to jump back and forth dynamically, with the pointer an integer, able to be stored in other variables, modified and retrieved. It is also possible to use the command pointer system variable to rewrite the script as it is interpreted, based on input, although this is has not been explored in great depth, but may be useful in certain situations.

C.3 Data forms/structures/types

Myriad stores all data in the form of strings, and implicitly casts between data types as required for a given operation. For example, the number 10.5 is stored as:

String: $x = "10.5"

Memory Data: [1 | 0 | . | 5 | #]

A printing operation will implicitly use the string value for output:

Print $x

Outputs 10.5 ("10.5")

The use of a variable in assignment attempts to apply the string value first and foremost.

$y = $x

$y = "10.5"

However, using the variable in an equation will result in evaluation as a floating point operation:

$z = $x + 1

$z = "11.5"

Using dynamic data typing requires the examination of the context of use and the casting of the variable between types as required, hence is not as fast as a strongly-typed language, where the interpreter does not have to attempt to resolve the type and is able to assume validity. In languages such as Java, this does produce an amount of performance degradation, but for the most part this is acceptably small in comparison to the time taken to create request handlers, process HTTP requests, run a command loop and maintain data structures for the interpreter.

Providing a 'weakly typed' language such as this does, however, provide a number of advantages:

- The user does not need to understand the concepts of data types and the distinctions of longs, ints, floats and doubles.
- The interpreter does not have to check data type compatibility in addition to context type checking.
- Users do not need to be provided with error reports and failures when data types are inappropriate to their request (e.g. Floating point numbers may not be added to integers.)
- The interpreted language need not process and understand casting directives from the user. In addition, syntax checking and exceptions need not be provided in complex operations, such as equation processing.

While strongly typed languages were preferred during the 1990s, there has been a movement toward weaker typing in newer languages, such as Java 1.5. This is due to an acceptance that while strong typing ensures data integrity, programmers tend to circumvent checking, rather than attempt to conform properly. In point of fact, strong typing often leads to users casting

data incorrectly as a fix to compiler errors, rather than by assured design, inducing errors as a result. Finally, strong typing forces the programmer of smaller applications and tools to spend disproportionate amounts of time on syntax fixing, rather than focussing on application structure and logical checking.

C.4 Arithmetic Evaluation

Processing arithmetic operations is also fully supported by MCS and is generally of the form

Variable = term operation term

A numerical example is:

$var = 5 + 5

Similarly, variables may be substituted for terms:

$var = $x + $y

Multi-operation equations are also available:

$var = $x + $y + $z

Terms may also contain operations, enclosed in brackets:

$var = ($x + $y) * 5

In which case, $x + $y is evaluated first, and then the * 5 operation is completed.

C.5 Conditions

In terms of conditional testing, present Myriad prototypes only implement a single if...else construction:

```
If <test>
{
}
else
{
```

```
}
```

In point of fact, the opening braces are not technically needed. Myriad disregards them and continues the scope of the condition until a closing brace. As a consequence, the form:

```
If <test>
        Command
<commands excluded from condition>
```

where the flow resumes after the single command is not supported, although it may be duplicated by:

```
if <test>
{
        Command
}

<commands excluded from condition>
```

In addition, else..if constructions are not supported, but may also be duplicated by nesting conditions:

```
If <term>
{
}
else
{
        if <test2>
        {
        }
}
```

Similarly, the 'not' test and Boolean combinations are also unsupported, so must by produced using judicious application of basic if...else structures:

To execute commands in when a test is untrue:

```
If <test>
{
}
else
{
        Commands
}
```

To AND two tests, use nested conditions:

```
If <test1>
{
        if <test2>
        {
                // Here, both test1 and test2 are true
                Commands
        }
}
```

To OR two conditions, use two sets of conditions sequentially:

```
If <test1>
{
}

if <test2>
{
}
```

Clearly this is an area where additional language expansions would make the usage easier for developers, but it will suffice for all fundamental conditional cases, where more evolved forms are 'syntactic sugar'.[2]

C.6 Looping

MCS supports three types of loop:

- While
- For finite limit, implicit iteration
- For custom iteration

The form of a while loop is relatively straightforward, as it incorporates the fundamental elements of an 'if' condition:

```
While <test>
{
        Commands
}
```

If the test succeeds, then once a closing brace is found, the command pointer is returned to the position for the condition and the process continues. If the test fails, then the pointer is placed at the closing brace, ready to be incremented by the interpreter and execute the next command.

The finite limit, implicit form of the for loop is one that is not often found in modern programming languages, but can be valuable, in that is performs a loop for a finite number of times, accepting only one argument, that of the number of times that the loop should be performed. This is either a number or a variable (starting with a $ symbol).

E.g.

For 5

[2] Syntactic Sugar is a term used by programmers and language developers to describe grammars which add no value in terms of additional functionality, but make current syntax faster, easier or more intuitive to use. Beneficial to developers, but not essential

```
{
        Commands
}
```

'Commands' is performed exactly five times, without the need to provide iterative start, stop and iteration arguments, helping to avoid edge conditions that may cause execution to be number +- 1. There is also a form which evaluates the value of a variable:

```
for $x
{
        Commands
}
```

This uses the normal Myriad term resolution mechanism, and will also evaluate expressions stored within that variable:

```
$x := '$y + 5'
$y = 3

for $x
{
        Commands
}
```

This will result in the loop being executed 8 times, as the resolving of $x leads to the resolution of the expression which it also contains.

The custom iteration form of the for loop is familiar to most developers as the common syntax used by Java, C, C++ and most modern programming and scripting languages (albeit without encircling brackets):

```
For $x = 0 ; $x < 5 ; $x++
{
        Commands
```

```
}
```

This comprises:

- A starting assertion ($x = 0)
- A running condition ($x < 5)
- An iteration expression ($x++)

C.7 Subroutines

Subroutines in Myriad are rudimentary, and mimic the use of subroutines in BASIC. At the end of the main section of code, after the final terminating brace, subroutines are placed, declared with their name only. Scoping is presently only universal, so there is no ability nor necessity to add parameters or create functions (subroutines with return values (although 'function' is an optional synonym for 'sub'). Subroutine declaration is therefore of the simple form:

```
Sub mySubroutineName
{
        Commands
}
```

To access a subroutine, 'gosub' is used (again, a piece of BASIC nostalgia):

```
Gosub mySubroutineName
```

Hence, to work with a subroutine, one may do the following:

```
$x = 5
$y = 3
gosub addXandY
print $x
}
```

```
sub addXandY
{
$x = $x + $y
}
```

At completion, $x will be printed as 8.

It should also be noted that Myriad supports alteration of program flow through labels and jumps to labels, with labels declared by a preceding '***', and jumps initiated with the 'jump' or 'goto' synonyms. Hence:

```
$x = 3
jump Test
$x = 5
***Test
print $x
```

Will print $x as 3, since the intermediary line was removed by the jump to the label.

C.8 Command Execution

Command sequences are traditionally parsed in a particular manner, for performance and neatness of code. For an interpreter, a command list is built, consisting of each possible command and a number, denoting the command in use.

1	For
2	If
3	Print
4	Let

A command sequence is then pre-parsed, with the command string broken into a sequence of 'tokens', the first of which is the command, which is translated into a command number, and the subsequent tokens are stored as elements of a parameter array.

Command sequence:

Print "This is a test"

Let $x = 5

Print OUT, "Made x equal 5"

Command Array:

Command Sequence Number	Command Type
1	3
2	4
3	3

Parameter Array:

Command Sequence Number	Param 1	Param 2	Param 3
1	This is a test		
2	$x = 5		
3	OUT	Made x equal 5	

While this form of parsing is highly useful in creating an efficient interpretation loop, it may be implemented in a number of ways, depending on data structures which are used, access methods and the string manipulation abilities of the language in which the interpreter is written. Since Myriad prototype servers are not created with efficiency foremost in mind, since they are primarily development environments for algorithms re-implemented in target systems, speed is not a primary consideration in their construction. Clarity of reading of the server source code, however, is a central aspect of Myriad initial design, since servers themselves are meant to be used as a basis for system understanding, teaching and a model for implementation of faster, more complete components in C, C++ and other more efficient languages.

As a result of the need for clarity of reading, prototype Myriad servers implement an interpretation method that most programmers will find simple, although often ugly. The existing servers provide command matching directly in the interpretation loop, without pre-

parsing, matching strings in situ. Unfortunately, this method is made all the more unpalatable with the incidental issue of C, C++ and Java (the child of C++) only supporting numerical values in their switching commands. As a result, Myriad server prototypes contain an interpretation loop that uses a series of conditional statements to identify and respond to commands. This takes the form of a chain of 'if ... else if ...' statements. Most experienced programmers will understand that this sort of loop is technically quite reasonable, since when this is compiled into binary form, it follows the assembly language construction that would create a sequence of tests such as this with a jump out to other sections of code as it was processed. A competent compiler and modern Java JVMs should compile this code into a similar form to the optimal assembly, regardless, and as such, this design is efficient and relatively intuitive.

The problem of this design, most fundamentally, is that strings are tested in the 'if' statements, rather than numbers, as a command list would convert to. As a result, each conditional test for a command will result in a string comparison. While a good JVM should also optimise string comparisons, this will not be as efficient as an integer comparison. Using an integer command number, assuming the number of commands is less than the standard integer size for the given computing device (for 32bit computers, that is 4 billion commands), any comparison will take one processing cycle (but a number of comparisons may be done per cycle for architectures with multiple pipelines and SIMD instructions). Unfortunately, using string comparisons, each letter in the string will take a comparison. For a five letter command, for example, five comparisons would have to be performed (see fig.115).

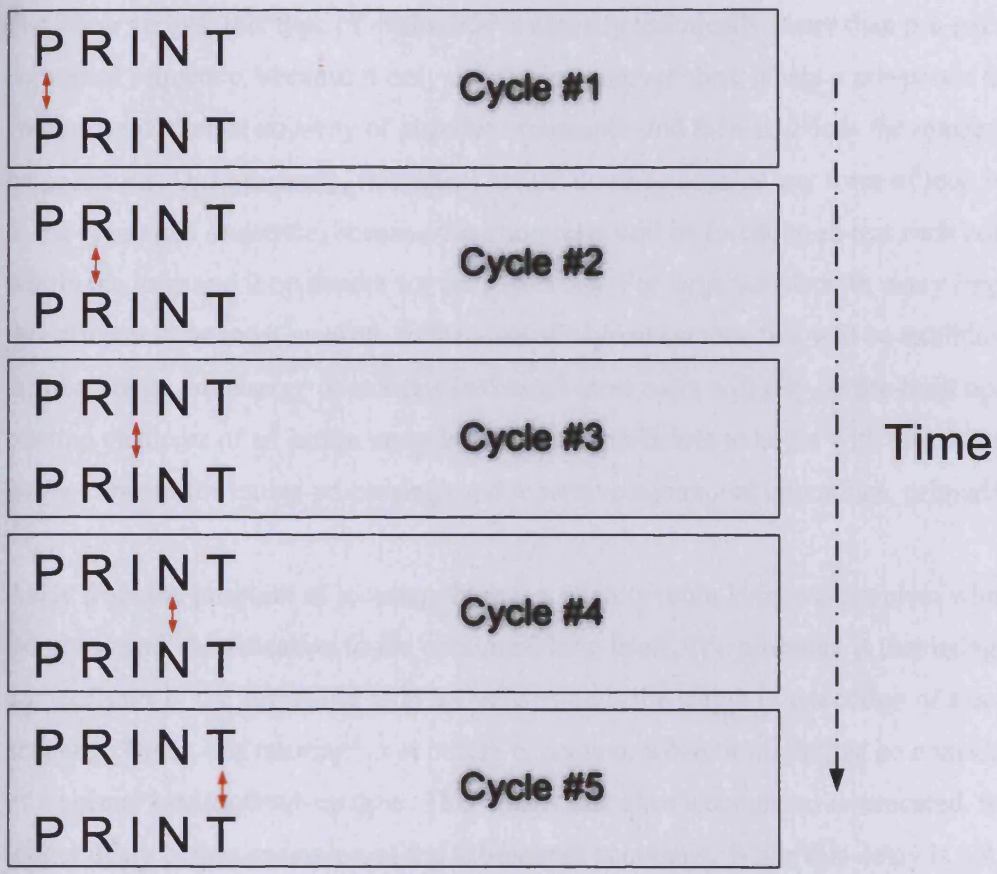


Figure 115. Diagram of successive character comparison for string comparison, linked to linear processing cycles

Clearly, this type of comparison is inefficient, although compilers will seek to optimise this process through a number of constraints:

- The length of strings will be compared before testing; if they are not equal, comparison will immediately fail
- Letters will only be compared until they fail to match. At that point, no remaining letters will be checked.

Still, this type of command matching is tangibly more expensive than comparing instruction numbers. Moreover, as this is done against each command that the system knows of, the larger the command repertoire, the longer the command loop will take to execute.

For linear scripts, this type of evaluation is actually technically faster than pre-parsing the command sequence, because it only does string comparisons, where a pre-parser does those comparisons against an array of possible commands and then also does the integer comparisons. Unfortunately, this effect breaks down as soon as any form of loop is included in the command sequence, because the interpreter will be forced to re-test each command within the loop and loop header for each iteration. For large scripts with many large loops, this effect will be most marked. In the case of Myriad servers, this will be exhibited with user-written image processing operations (although most users will rely on pre-built operations and reading elements of an image array in Java is so inefficient to begin with that this is a far larger concern for image processing) and repetitive equipment operations, primarily.

Aside from the problem of looping, there is a slightly more insidious problem with moving the command identification to the command loop itself. The problem, is that using string comparisons in the command loop actively extends the length of execution of a command sequence *once it is running*, not before execution, where it might just be considered a part of a normal loading/start-up time. This means that once a command is executed, there is a longer delay before execution of the subsequent command. While this delay is not great, for acutely time-sensitive applications, this is not the most ideal construction. In these cases, using a pre-parser would be essential.

For all of these limitations and failures of in-loop command checking, however, there are also benefits, in that Myriad servers are designed for teaching, giving a clear guide for re-implementation and alteration by students of varying degrees of programming skill. As a result, the clarity of this method of execution when viewing the source code is a key benefit that outweighs the performance issues. Moreover, since the servers are implemented using Java, the resulting code may be interrupted at any time due to Java garbage collection and other system activity that will alter timings between commands and potentially reduce performance substantially. On desktop systems and servers, the variety of running processes and other system performance profiles (especially memory management and CPU allocation routines) will provide far more impact than this change might. Sufficed to say, the most efficient command interpreter would be written with a pre-parser, running on an optimised Real Time Operating System, such as QNX, with a highly controlled process list and it would not be written in Java.

C.9 Arithmetic Processing

Due to the requirements for variable support, use of complex data structures during interpretation (e.g. image arryas, 2D, 1D) and customised operators, MCS requires an arithmetic processing system. The beginnings of such a system manifest at the start of the command interpretation loop where assignment tests take place. Because all variable names have been pre-parsed and therefore initialised, arithmetic functions are essential in two cases:

- Assignment
- Resolution of truth in conditions and looks

It is also possible to invoke the arithmetic processor to evaluate in-line equations in a form of enhanced interpolation e.g.

Print "This is a $test"

Print "This is a ($test + 1)"

Due to the complications and ambiguities of the above use (should *$test = 2* print *"This is a 3"* or *"This is a (2 + 1)"* ?), this is not done in the standard commands, rather, using a setup step such as:

$test2 = $test + 1

print "This is a $test2"

Assignment occurs at the beginning of the interpretation loop, because it is most convenient to test for the leading dollar ($) symbol at this point, denoting a variable. If this fails, then command evaluation and execution can proceed; if it succeeds, however, no tests have to be performed for other commands, accelerating execution times for the given line of commands.

Assignment begins with a test for the type of assignment in use:

- String assignment ($variable := ...)

- String concatenation ($variable .= ...)
- Numerical assignment ($variable = ...)
- Short numerical assignments (+=, *=, etc.)
- Iterator ($variable++)
- Deprecator ($variable--)

There is a slight modification to this arrangement, however; since iterators and deprecators are attached to the variable being assigned to, they are tested for and placed in a variable named tempOperator and the variable name is reduced by two trailing characters. When other assignments take place, the first word is retained, and the next word (or 'token') is applied to tempOperator, ensuring that once this process is complete, the variable name is always ready for use, and the operation to be performed is contained in tempOperator, uniformly.

Once this is complete, all assignment may be treated in the same manner, based on the value of tempOperator.

String Assignment

String assignment is dealt with first, due to simplicity and clarity. There are two forms:

- Assignment
- Concatenation (joining of strings)

Assignment and concatenation of strings is handled readily by Java and most modern programming languages, so the resulting code merely passes through to interpreter code:

```
If (tempOperator == ":=")
        TempString = stringToAssign
Else if (tempOperator == ".=")
        TempString .= stringToConcatenate
End
```

However, there is a modification to this, which is essential for any advanced string use; the addition of interpolation.

Strings wrapped in quotation marks are interpolated, while those in apostrophes are not interpolated, allowing for users to keep their variable names, if required.

$test = 1

$variable := "this is a $test"

Output: *This is a 1*

$variable := 'This is a $test'

Output: *This is a $test*

Because of these formations, string delimiters must be removed from both ends of the assignment string prior to assignment. A method named removeDelimiters (string, delimiter) exists for this reason and removes the delimiter from the string passed as the argument. It should also be noted that the apostrophe delimiter may be removed and spaces used instead, without ill effect.

Numerical Assignment

Evaluation of expressions is a potentially recursive process, based on the understanding that an expression is of the form:

<term> Operator <term>

Where Operator is a mathematical symbol and a term is:

- A variable to resolve
- A number (int, float, double)

- An advanced operator using the above (e.g. *cos(<term>)*)
- A mathematical numeric symbol (e.g. *PI*)
- An expression

The ability of a term to be an expression is what leads to the recursion and introduces complexity that will be seen in the solution.

This evaluation is performed using three methods:

Expression – Isolates the terms and operators and performs the operation, returning the result

Term – Requests resolution of terms by 'Factor' and returns 0 on failure to ensure integrity.

Factor – Resolves terms or initiates new expression processing

There is also a wrapper method, named 'ExtendedExpression' which wraps around Expression, accepting two expressions and a comparator, performing truth tests for conditions and loops.

Extended Expression

<expression> Comparator <expressions>

Where the comparator is =, >=, >, etc.

This resolves both expressions by calling Expression with each of them, and then tests the comparator with the two values, returning true or false. Using this wrapper allows for easy construction of loops and conditional statements.

Expression

The Expression process is formed from two parts, operator recognition and assignment of non-operators. A running total of the expression value is kept, as each word is read, resolved through Term and then operated against the running total.

Should the word fail to be and operator, then logically this is the start of the expression, and the result of *Term(word)* is performed and added to the result, which begins as 0.

Due to the nature of Term and Factor, which it uses to resolve terms, there are not problems of Expression cycling through an expression such as:

*($x + 5) * 3*

Because bracketed terms are caught by Factor and evaluated recursively in place, before the Expression root instance must use them, thus avoiding problems of operator precedence. Without nested expressions, however, evaluation will be done left to right, without priority or modification, resulting in results such as:

5 + 3 / 8 * 2 = 2

As 5 + 3 = 8, 8 / 8 = 1 and 1 * 2 = 2

Factor

Factor is the core of the arithmetic evaluation system, as it resolves the values of terms and in complex terms, resubmits them to Expression.

The sequence is:

Resolve variables (starting '$')
Resubmit to Expression if complex (starting '(')
Resolve PI and E to values
Resolve mathematical operators (cos, round, etc.)

Resolve numbers (int, float)

As each stage, if the term matches, the resolved value is returned and Factor completes, skipping the other tests.

Variable resolution is straightforward and involves extracting the value string from the variables vector and casting it into a double length floating point number. If this value is in fact an integer, it is cast out further down the tree, in Term and Expression, preserving precision for as long as possible.

If the term begins with a bracket, then the term is complex and contains a sub-expression. In this case, the end of the bracket must be found, collecting the expression, even though it may contain further nested expressions. To do this, consider the following term:

*($x + ((8 – 3) * 5))*

If the evaluation were to only send the string until the first closing bracket to Expression, the sent string would be:

$x + ((8 – 3

Clearly this is not acceptable, and to compensate, a cumulative counting mechanism is used. For each open bracket, +1 is added to a total; for each closing bracket, -1 is added. With the total set to +1 to account for the initial opening bracket at the start of the string, the string continues to be gathered until the total becomes 0 (see fig.116). This point gives the rightmost boundary for taking a sub string for submission to Expression.

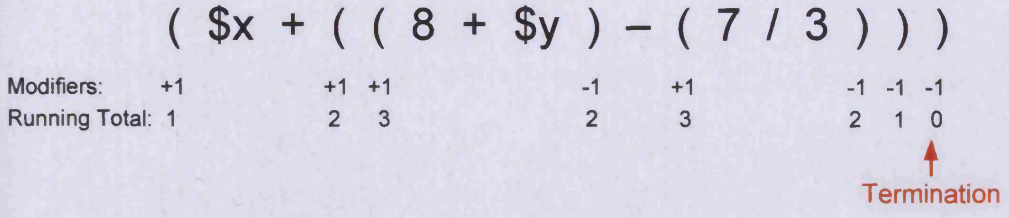


Figure 116. Diagram of bracket-counting procedure for segmenting nested arithmetic expressions and the running count of brackets

Using these values, a the resultant call to Expression would be:

Result = expression(String.substring(Start + 1, End – 1)

3. The cases of PI and E are simple substitution; if the term is equal to the strings 'E', 'PI', 'Pi' or 'pi', then straight values for E or Pi are returned. Note that 'e' is reserved for customisation of convolution filters, and is not matched.

Appendix D: Examples & Source Code

D.1 Redirecting to multiple servers with Perl

```
#!/usr/bin/perl

if($user == "luke" && $password == "test")
{
$portnum = "3080"; // Privileged server answers on port 3080
}
else
{
$portnum = "3081"; // Unprivileged server is on 3081
}

$url = "http://localhost:";
$url .= $portnum . "/";
$url .= $commands;

system("wget $url");
```

To use this, the Perl script is requested with an argument called *'commands'* and optional arguments *'user'* and *'password'*.

http://www.server.com/access.pl?commands=neg
http://www.server.com/access.pl?user=luke&pass=test&commands=neg

D.2 Database access gateway

The PHP required to perform database HTTP-bridging is relatively straightforward:

```
<?php
//Connect to the database (please supply your own database details here
@ $db = mysql_pconnect("$dbhost", "$dbuser", "$dbpasswd");
```

```
mysql_select_db("$dbname");

$argument = stripspecial($argument);
$sql = $argument;
$result = mysql_query($sql);

if($result)
{
        $row = mysql_fetch_row($result);
        do
        {
                $output = "";
                foreach ($row as $row_element)
                {
                $output .= $row_element . "||" ;
                }
                echo $output . "\n";
        }while ($result == mysql_fetch_row($result);
}
else
{
        echo "Query failed: $argument";
}
?>
```

Which will return the results of the SQL query '$argument' as a set of records with a field separator of two vertical bars and records separated by a carriage return ('\n'). Choice of a suitable separator may be altered by the developer so as not to conflict with appearances in the final data being used.

To use the Gateway, request it as a normal HTTP request:

http://myserver/gateway.php?argument=<sql>

Where the user should replace <sql> with their own SQL command string, myserver should be changed to the server IP or domain being used, and gateway.php should be the name of the PHP file.

In point of fact, a minor extension to the Gateway would allow the user to specify their own field and record separators:

```
<?php
//Connect to the database (please supply your own database details here
@ $db = mysql_pconnect("$dbhost", "$dbuser", "$dbpasswd");
mysql_select_db("$dbname");

$fieldsep = stripspecial($fieldsep);
$recordsep = stripspecial($recordsep);

$argument = stripspecial($argument);
$sql = $argument;
$result = mysql_query($sql);

if($result)
{
        $row = mysql_fetch_row($result);
        do
        {
                $output = "";
                foreach ($row as $row_element)
                {
                $output .= $row_element . $fieldsep;
                }
                echo $output . $recordsep;
        }while ($result == mysql_fetch_row($result);
}
else
{
```

```
        echo "Query failed: $argument";
}

dbclose($db);
?>
```

To use this, add extra arguments to the request, with the same name as the $-prefixed variables (since PHP automatically discovers these arguments and initialises variables with the same names):

http://myserver/gateway.php?argument=<sql>&fieldsep=<sep1>&recordsep=<sep2>

Where <*sep1*> and <*sep2*> should be replaced with appropriate values.

D.3 Authentication gateway

```
<?php
//Connect to the database (please supply your own database details here
@ $db = mysql_pconnect("$dbhost", "$dbuser", "$dbpasswd");
mysql_select_db("$dbname");

$sql = "select * from users where username='$user' and password='$password'";
$result = mysql_query($sql);

if($result)
{
        if(mysql_num_rows($result) == 0)
        {
                // If this happens, authentication has failed.
                echo "Authentication failed. The username/password is incorrect.";
                exit();
        }
}
else
```

```
{
        echo "Could not check username and password. Action failed.";
        exit();
}
// If this point is reached, authentication didn't fail (in other words, if succeeded)


$fieldsep = stripspecial($fieldsep);
$recordsep = stripspecial($recordsep);

$argument = stripspecial($argument);
$sql = $argument;
$result = mysql_query($sql);

if($result)
{
        $row = mysql_fetch_row($result);
        do
        {
                $output = "";
                foreach ($row as $row_element)
                {
                $output .= $row_element . $fieldsep;
                }
                echo $output . $recordsep;
        }while ($result == mysql_fetch_row($result);
}
else
{
        echo "Query failed: $argument";
}
?>
```

D.4 Generic Myriad authentication gateway

Using *wget* to transmit a command sequence to the IP server is as follows:

```
<?php
//Connect to the database (please supply your own database details here
@ $db = mysql_pconnect("$dbhost", "$dbuser", "$dbpasswd");
mysql_select_db("$dbname");

$sql = "select * from users where username='$user' and password='$password'";
$result = mysql_query($sql);

if($result)
{
        if(mysql_num_rows($result) == 0)
        {
                // If this happens, authentication has failed.
                echo "Authentication failed. The username/password is incorrect.";
                exit();
        }
}
else
{
        echo "Could not check username and password. Action failed.";
        exit();
}
// If this point is reached, authentication didn't fail (in other words, if succeeded)

$argument = stripspecial($argument);

$url = http://www.myserver.com:8080/;
$url = $url . "?commands=" . $argument;

passthru($url);
```

```
?>
```

D.5. Database-backed command filtering gateway

```
<?php
//Connect to the database (please supply your own database details here
@ $db = mysql_pconnect("$dbhost", "$dbuser", "$dbpasswd");
mysql_select_db("$dbname");

$sql = "select * from users where username='$user' and password='$password'";
$result = mysql_query($sql);

if($result)
{
        if(mysql_num_rows($result) == 0)
        {
                // If this happens, authentication has failed.
                echo "Authentication failed. The username/password is incorrect.";
                exit();
        }
}
else
{
        echo "Could not check username and password. Action failed.";
        exit();
}
// If this point is reached, authentication didn't fail (in other words, if succeeded)

$argument = stripspecial($argument);

$user_rights = mysql_fetch_array($result);
$illegal_commands = split(",", $user_rights[blocked_commands);

foreach ($illegal_commands as $command)
```

```
{
$argument = eregi_replace("%0D%0A${command}[!%]+", "", $argument);
}

$url = http://www.myserver.com:8080/;
$url = $url . "?commands=" . $argument;

passthru($url);
?>
```

D.6 Java image pixel data array extraction

```
public int[] getImageChannel1D(int channelType, int originX, int originY,
int widthx, int heightx)
    {
      // This gets an image channel and makes it into a 1D array
      short width = (short) widthx;
      short height = (short) heightx;
      int pixels[] = new int [width * height];
      PixelGrabber grabber = new PixelGrabber(tempImage, originX, originY,
width, height, pixels, 0, width);
      try
      {
      grabber.grabPixels();
      }
      catch (InterruptedException e)
      {
      }
      int pixelLength = pixels.length;
      for(int pixelCounter = 0; pixelCounter < pixelLength; pixelCounter++)
      {
      switch (channelType)
      {
      case 1 : pixels[pixelCounter] = getRed(pixels[pixelCounter]);break;
      case 2 : pixels[pixelCounter] = getGreen(pixels[pixelCounter]);break;
      case 3 : pixels[pixelCounter] = getBlue(pixels[pixelCounter]);break;
      case 4 : pixels[pixelCounter] = getHue(pixels[pixelCounter]);break;
      case 5 : pixels[pixelCounter] =
getSaturation(pixels[pixelCounter]);break;
```

```
      case 6 : pixels[pixelCounter] =
getIntensity(pixels[pixelCounter]);break;
      } //close switch
      }
      return pixels;
    }
```

The methods getRed, getGreen, getBlue, getHue, getSaturation, getIntensity are all small custom-written functions intended to perform the appropriate shift and mask operations required in order to obtain the appropriate component of a packed pixel.

```
      public int getRed(int rgb)
    {
      int red = (rgb & 0xff0000) >> 16;
      return red;
    }

    public int getGreen(int rgb)
    {
      int green = (rgb & 0xff00) >> 8;
      return green;
    }

    public int getBlue(int rgb)
    {
      int blue = (rgb & 0xff);
      return blue;
    }
```

D.7 Java functions for HSI extraction

```
public int getHue(int rgb)
    {
      int r = getRed(rgb);
      int g = getGreen(rgb);
      int b = getBlue(rgb);

      int top = (2*r) - g - b;
```

```
      double bottom = 2 * Math.sqrt((double) ((r - g) * (r - g)) + (r -
b)*(g - b));

      int H = (int) Math.acos((double) top / (double) bottom);

      return H;
    }

    public int getIntensity(int rgb)
    {
      int r = getRed(rgb);
      int g = getGreen(rgb);
      int b = getBlue(rgb);

      int i = (int) (r + g + b) / 3;

      return i;
    }

    public int getSaturation(int rgb)
    {
      int r = getRed(rgb);
      int g = getGreen(rgb);
      int b = getBlue(rgb);
      int minimal = 0;

      if(r < g)
      {
            minimal = r;
      }
      else
      {
            minimal = g;
      }
      if (minimal > b)
      {
            minimal = b;
      }

      int s = (3 * minimal) / (r + g + b);
```

```
        return s;
    }
```

D.8 Converting a 1D pixel array to a Java Image

```
public void putImageChannel1D(String channelType, int widthx, int heightx, int[] pixies, Image
myImage, boolean otherOutput)
    {
        int[] pixels = new int[pixies.length];
        System.arraycopy(pixies, 0, pixels, 0, pixies.length);
        //This puts the contents of an image channel back into tempImage for working with
        //System.out.println("\nInto putimagechannel with width/height: " + widthx + " " +
heightx);
        short width = (short) widthx;
        short height = (short) heightx;

        Toolkit tk = Toolkit.getDefaultToolkit();
        ColorModel cm = ColorModel.getRGBdefault();
        int x = 0;
        for (int i=0; i < width; i++)
        {
                for (int j=0; j < height; j++)
                {
                if(channelType.equals("i") || channelType.equals("I"))
                {
                pixels[ i + j * width ] = 0xff000000 | (pixels[ i + j * width ] << 16) |
(pixels[ i + j * width ] << 8) | pixels[ i + j * width ];
                }
                else if(channelType.equals("a") || channelType.equals("A"))
                {
                pixels[ i + j * width ] = 0x00000000 | (pixels[ i + j * width ] << 24);
                }
                else if(channelType.equals("r") || channelType.equals("R"))
                {
                pixels[ i + j * width ] = 0xff000000 | (pixels[ i + j * width ] << 16);
                }
                else if(channelType.equals("g") || channelType.equals("G"))
                {
                pixels[ i + j * width ] = 0xff000000 | (pixels[ i + j * width ] << 8);
                }
                else if(channelType.equals("b") || channelType.equals("B"))
                {
                pixels[ i + j * width ] = 0xff000000 | pixels[ i + j * width ];
                }
                }//close inner for loop, using j
        }//close outer loop, using i to traverse the columns

        //Make a copy of the old current image before we do anything :) Stick it on the stack
```

```
        if(!otherOutput)
        {
                if(tempImage != null)
                {
                        historyStack.addElement(tempImage);
                }

        tempImage = tk.createImage( new MemoryImageSource(width, height, cm, pixels, 0,
width));
        }
        else
        {
        myImage = tk.createImage( new MemoryImageSource(width, height, cm, pixels, 0, width));
        }

    }//end putImageChannel method
```

The majority of this source code is relatively obvious, composed mostly of iteration through the pixels and the shift and mask operations required to rebuild the packed pixels. It should be noted that this does not provide a full packed pixel, but merely builds an image with the channel applied to the nominated channel (for example, an R channel would normally be reapplied to the R channel, but there is the option to apply it to the G or B or A channels, or apply it to all channels uniformly, converting it into an Intensity value

Once again, it is also interesting to note the two lines:

```
Toolkit tk = Toolkit.getDefaultToolkit();
ColorModel cm = ColorModel.getRGBdefault();
```

These obtain the image toolkit and the colour model from the main user interface element (e.g. the application frame or Applet window) in order to be able to rebuild an image. Without these elements, a Java application could not construct one of the generalised *'Image'* objects; another example of how tightly bound image operations in Java are to the user interface and rendering aspects of the language.

D.9 C# IIP graphical user interface

```
using System;
using System.Drawing;
```

```
using System.Net;
using System.IO;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;
using System.Web;
using System.Text;

namespace RemoteIIP
{
	/// <summary>
	/// Summary description for Form1.
	/// </summary>
	public class Form1 : System.Windows.Forms.Form
	{
		private System.Windows.Forms.PictureBox pictureBox1;
		private System.Windows.Forms.PictureBox pictureBox2;
		private System.Windows.Forms.Label label1;
		private System.Windows.Forms.Label label2;
		private System.Windows.Forms.TextBox textBox1;
		private System.Windows.Forms.Label label3;
		private System.Windows.Forms.TextBox textBox2;
		private System.Windows.Forms.Button button1;
		private System.Windows.Forms.TextBox textBox3;
		private string currentServerString;
		/// <summary>
		/// Required designer variable.
		/// </summary>
		private System.ComponentModel.Container components = null;

		public Form1()
		{
			//
			// Required for Windows Form Designer support
			//
			InitializeComponent();

			//
			// TODO: Add any constructor code after
InitializeComponent call
```

```
            //
            currentServerString = textBox3.Text;
            serverStart();
        }

        /// <summary>
        /// Clean up any resources being used.
        /// </summary>
        protected override void Dispose( bool disposing )
        {
            serverStop();
            if( disposing )
            {
                if (components != null)
                {
                    components.Dispose();
                }
            }
            base.Dispose( disposing );
        }

        /// <summary>
        /// The main entry point for the application.
        /// </summary>
        [STAThread]
        static void Main()
        {
            Application.Run(new Form1());
        }

        public Image GetImage(string sURL)
        {
            Stream str = null;
            HttpWebRequest wReq =
(HttpWebRequest)WebRequest.Create(sURL);
            HttpWebResponse wRes =
(HttpWebResponse)(wReq).GetResponse();
            str = wRes.GetResponseStream();

            return Image.FromStream(str);
        }
```

```
        public void updateImages()
        {
            string imageString= "";
            try
            {
                imageString = textBox3.Text + "?image=current";
                pictureBox1.Image = GetImage(imageString);
            }
            catch
            {
            }
            try
            {
                imageString = textBox3.Text + "?image=alt";
                pictureBox2.Image = GetImage(imageString);
            }
            catch
            {
            }
        }

        private void button1_Click(object sender, System.EventArgs e)
        {
            try
            {
                string urlString = textBox3.Text + "?commands=" +
textBox2.Text;
                HttpWebRequest httpRequest = (HttpWebRequest)
WebRequest.Create(urlString);
                HttpWebResponse webResponse = (HttpWebResponse)
httpRequest.GetResponse();
                Encoding enc = Encoding.GetEncoding(1252);
            }
            catch
            {
            }
            textBox1.AppendText(textBox2.Text);
            textBox2.Clear();
            updateImages();
        }
```

```
            private void textBox3_TextChanged(object sender,
System.EventArgs e)
            {
                  serverStop();
                  currentServerString = textBox3.Text;
                  serverStart();
            }
            private void serverStart()
            {
                  try
                  {
                        string urlString = textBox3.Text +
"?commands=start";
                        HttpWebRequest httpRequest = (HttpWebRequest)
WebRequest.Create(urlString);
                        HttpWebResponse webResponse = (HttpWebResponse)
httpRequest.GetResponse();
                        Encoding enc = Encoding.GetEncoding(1252);
                  }
                  catch
                  {
                  }
                  updateImages();
            }
            private void serverStop()
            {
                  try
                  {
                        string urlString = currentServerString +
"?commands=stop";
                        HttpWebRequest httpRequest = (HttpWebRequest)
WebRequest.Create(urlString);
                        HttpWebResponse webResponse = (HttpWebResponse)
httpRequest.GetResponse();
                        Encoding enc = Encoding.GetEncoding(1252);
                  }
                  catch
                  {
                  }
            }
```

```
    }
}
```

D.10 C++/Qt IIP graphical user interface

```
#include <qurloperator.h>
#include <qstring.h>
#include <qstringlist.h>
#include <qnetwork.h>
#include <qfile.h>
#include <qtextstream.h>
#include <iostream.h>
#include <qtimer.h>
#include <qurl.h>

void Form1::init()
{
        QTimer *internalTimer = new QTimer( this ); // create internal
timer
    connect( internalTimer, SIGNAL(timeout()), SLOT(updateImages()) );
    internalTimer->start( 15000 );
updateImages();
}


void Form1::updateImages()
{
qInitNetworkProtocols();

    QUrlOperator *op = new QUrlOperator();
    QString tempString = lineEdit1->displayText();
    tempString += "test.jpg?fullimage=current";
    op->copy( tempString ,QString("test.jpg"), FALSE, FALSE);
    connect( op, SIGNAL( finished(QNetworkOperation*)), this, SLOT (
updateImage2()));

    QUrlOperator *op2 = new QUrlOperator();
    tempString = lineEdit1->displayText();
    tempString += "test.jpg?fullimage=alt";
    op2->copy( tempString ,QString("test2.jpg"), FALSE, FALSE);
```

```
    connect( op2, SIGNAL( finished(QNetworkOperation*)), this, SLOT (
updateImage1()));

    QUrlOperator *op3 = new QUrlOperator();
    tempString = "test.txt";
    op3->copy( tempString ,QString("test.txt"), FALSE, FALSE);
    connect( op3, SIGNAL( finished(QNetworkOperation*)), this, SLOT (
updateText()));
}


void Form1::updateImage1()
{
QPixmap myPixmap;
    myPixmap.load("test2.jpg");
    pixmapLabel1->hide();
    pixmapLabel1->setPixmap(myPixmap);
    pixmapLabel1->show();
}


void Form1::updateImage2()
{
QPixmap myPixmap;
    myPixmap.load("test.jpg");
    pixmapLabel2->hide();
    pixmapLabel2->setPixmap(myPixmap);
    pixmapLabel2->show();
}


void Form1::updateText()
{
QStringList lines;
    QFile file( "test.txt" );
    if ( file.open( IO_ReadOnly ) ) {
        QTextStream stream( &file );
        QString line;
        while ( !stream.atEnd() ) {
            line = stream.readLine(); // line of text excluding '\n'
            lines += line;
```

```
        }
      textEdit1->setText(lines.join("\n"));
        file.close();
    }
}


void Form1::sendScript()
{
    QString tempString = lineEdit1->displayText();
    QString argsString = textEdit2->text();
    QUrl::encode(argsString);
    QUrlOperator *op4 = new QUrlOperator();
    tempString += "?commands=";
    tempString.append(argsString);
    op4->copy( tempString ,QString("testscript.txt"), FALSE, FALSE);
    textEdit1->append(textEdit2->text());
    textEdit2->setText("");
    updateImages();
}
```

D.11 PHP/HTML IIP graphical user interface

```
<?php
if(!empty($_GET)) extract($_GET);
include("pageheader.php");
if(!$host)
{
$hostsfile = fopen("iphosts.txt", "r");
$besthostload = 999;
while(!feof($hostsfile))
{
$hostline = fgets($hostsfile);
$hostarray = explode("|", $hostline);
$testhost = $hostarray[0];
$testhostDescription = $hostarray[1];
$testhostLocation = $hostarray[2];
$testhost = trim($testhost);
$testhostLocation = trim($testhostLocation);
if($testhost != "")
{
```

```
$load = fopen("$testhost/system_load.txt", "r");
$hostload = fgets($load);
if($hostload < $besthostload)
{
//echo "<br>host is $testhost and load is $hostload user sessions";
$host = $testhost;
$besthostload = $hostload;
$hostDescription = $testhostDescription;
$hostLocation = $testhostLocation;
}
fclose($load);
}
}
fclose($hostsfile);
}

if (!$started)
{
//This gets run if we are just starting up the UI
?>
 <center>
<h2><br><br><a href="<?php echo $PHP_SELF ?>?started=true&host=<?php echo
$host ?>" class=white><img src="core/myriad_logo.jpg"
border=0></a><br><br><br></h2>
<script src="<?php echo $host ?>/?commands=starthtml"></script>
<table bgcolor =#FFFFFF cellspacing=5 cellpadding=5><tr><td><img
src="core/connected.jpg"></td><td>You are connected to: <?php echo
$hostDescription ?><br>Located: <?php echo $hostLocation
?></td></tr></table>
</center>

<?php
}
else
{
// This is the body of the UI

if($url)
{
if(strstr($url, "http://"))
{
```

```
$commands = "downloadimagefile system_tempimage " . $url;
}
else
{
$commands = "openimagefile system_tempimage " . $url;
}
}
?>
 <div id="forumlayer" style="position:absolute; left:0px; top:83px;
width:250px; height:50px; z-index:103; visibility: hidden">
  <table width="100%" border="0" cellspacing="0" cellpadding="0"
height="50">
    <tr>
      <td bgcolor="#000000" height="1"></td>
      <td bgcolor="#000000" height="1"></td>
      <td bgcolor="#000000" height="1"></td>
      <td bgcolor="#000000" height="1"></td>
      <td bgcolor="#000000" height="1"></td>
    </tr>
    <tr>
      <td bgcolor="#000000" width="1"></td>
      <td class=bSB bgcolor="#FFFFFF" rowspan="4" valign="top">
      <form name=2 action="<? print($PHP_SELF) ?>" method="GET">
      URL: <input type=text name=url>

      <input type="hidden" name="started" value="true">
      <input type="hidden" name="host" value="<?php echo $host ?>">
      <input type="submit" name="submit" value="Submit">
      </form>
        </td>
      <td class=bSB width="30" height="50" bgcolor="#FFFFFF" valign="top">
        <table width="30" border="0" cellspacing="0" cellpadding="0"
height="18">
          <tr>
            <td height="5"></td>
            <td></td>
          </tr>
          <tr>
            <td width="30">
```

```
            <div align="center"><a href="javascript:;"><img
src="core/close.jpg" width="16" height="16" align="bottom" border="0"
onClick="MM_showHideLayers('forumlayer','','hide');"></a></div>
          </td>
          <td> </td>
        </tr>
      </table>
    </td>
    <td bgcolor="#000000" width="1"></td>
  </tr>
  <tr>
    <td bgcolor="#000000" height="1"></td>
    <td class=bSB height="30" width="30"><img src="core/cutcorner.gif"
width="30" height="30"></td>
    <td height="30" width="1"></td>
  </tr>
  <tr>
    <td bgcolor="#000000" height="1"></td>
    <td bgcolor="#000000" height="1"></td>
    <td bgcolor="#000000" height="1"></td>
    <td height="1"></td>
    <td height="1"></td>
  </tr>
</table>
</div>

<div id="myriadlogolayer" style="position:absolute; left:10px; top:10px;
width:120px; height:126px; z-index:1; visibility: visible"><img
src="core/myriad_logo2.gif"></div>
<table bgcolor=#FFFFFF width=950><tr valign="top"><td colspan=2>
<h3>                    
          <a href="javascript:;"
onClick="MM_showHideLayers('forumlayer','','show');"><img
src="core/download.jpg" alt="Download Image" align=absmiddle
border=0></h3></td></tr><tr><td align=right  valign=top>
<form name="commands" action="<? print($PHP_SELF) ?>" method="GET">
<p class="grey"><img src="core/forward.png" align=middle> <textarea
name="commands"></textarea>
<input type="hidden" name="started" value="true">
<input type="hidden" name="host" value="<?php echo $host ?>">
<input type="submit" name="submit" value="Submit"></p>
```

```
<?php
//$mode="form";
//include("fileupload.php");
if ($commands)
{
$commands = urlencode($commands);
echo "<script src=\"$host/?commands=$commands\"></script>";
//echo "Commands were : $commands";
}
?>
<p class="lightgrey">Alternate:<br>
<a href="<?php echo $host ?>/webcam.jpg?fullimage=alt" target="_blank"><img
src="<?php echo $host ?>/webcam.jpg?image=alt" border=1></a><br>
<br><a href="<? print($PHP_SELF) ?>?started=true&host=<?php echo $host
?>&commands=alt&submit=submit"><img src="core/alternate.jpg"
border=0></a></td><td  valign=top>

<p class="lightgrey">Current:<br>
<a href="<?php echo $host ?>/webcam.jpg?fullimage=current"
target="_blank"><img src="<?php echo $host ?>/webcam.jpg?image=current"
border=1></a>
<a href="<? print($PHP_SELF) ?>?started=true&host=<?php echo $host
?>&commands=grey&submit=submit"><img src="core/greyscale.jpg"
alt="Greyscale" border=0></a>
</td></tr><tr>
<td colspan=2><img src="core/imagestack.jpg">
<?php
echo "<script src=\"$host/webcam.jpg?stack=yes\"></script></td></tr><tr><td
colspan=2>";
echo "<script src=\"$host/webcam.jpg?status=yes\"></script>";
echo "<a href=\"$host/webcam.jpg?histogram=red\" target=\"_blank\"><img
src=\"$host/webcam.jpg?thumbhistogram=red\"></a>";
?>
</td>
</table>
<?php
}
include("pagetail.php");
?>
```

D.12 Perl script for mail collection

```
#!/usr/bin/perl
use DBI;
use Net::POP3;
my $pop3 = Net::POP3->new('mail.linuxcomment.com');
my $Num_Messages = $pop3->login('photos@linuxcomment.com', 'photos');
my $Messages = $pop3->list();
$from = "";
$subject = "";
foreach $msg_id (keys(%$Messages))
{
    $message = $pop3->get($msg_id);
my (@lines) = @$message;

foreach $line (@lines)
{
#print "Reading line";
    if($line =~ m/^From: (.*)/)
    {
       $from = $1;
      # Let's save what follows "From: "
      $from =~ s/"|<.*>//g;

    }
    elsif( $line =~ m/^Subject: (.*)/)
    {
       $subject = $1;
    }
}
print "Subject: $subject - From: $from";


      if ($from)
      {
            print "Got a phone message!";
            $contentType = 0;
            $startReading = 0;
            $attachmentComplete = 0;
            $attachment = "";
```

```
$bodyComplete = 0;
$body = "";
$tempString = 0;
foreach $line (@lines)
{
      if($line =~ /^Content-Disposition: attachment/)
      {
            $contentType = 1;
            $tempString = "";
      }
      if($line =~ /^Content-Disposition: inline/)
      {
            $contentType = 2;
            $tempString = "";
      }
      if($line =~ /------=_Part_/)
      {
            if($startReading == 1)
            {
                  $startReading = 0;
                  if($contentType == 1)
                  {
                        $attachment = $tempString;
                        $tempString = "";
                        $attachmentComplete = 1;
                  }
                  if($contentType == 2)
                  {
                        $body = $tempString;
                        $tempString = "";
                        $bodyComplete = 1;
                  }
            }
      }
      if($startReading == 1)
      {
            $tempString .= $line;
      }
      if($line eq "\n")
      {
            #print "Starting reading!\n";
```

```
                        $startReading = 1;
                }
        }
        print "From: $from";
        print "Body: $body";
        print "Attach: $attachment";
        if($body =~ /[a-zA-Z0-9]*/)
        {
                #Let's do the database access
                my $dbh = DBI->connect("dbi:mysql:test:localhost",
"root", "");
                my $query = $dbh->prepare("insert into tracking
(mail_from, mail_body, mail_attach, mail_date) values ('${from}','${body}',
'${attachment}',CURRENT_DATE())");
                $query->execute;
                $dbh->disconnect();
                $pop3->delete($msg_id);
        }
    }
    else
    {
        print "Not a phone message";
    }
}
$pop3->quit();
```

Appendix E: History of Machine Vision

This appendix examines the history of machine vision as a topic of endeavour; from the origins of the subject in pattern recognition, optics and robotics, to the development of the discipline itself, yet still tied to a confederation of those diverse elements.

Machine vision progression in the past quarter-century is explored worldwide, and the patterns of development, along with key issues, such as duplication of effort, lack of communication and degree of failure are examined. Summary description of developments in optics, electrical engineering and robotics are also provided.

Finally, having accounted for the history of the subject, we consider the present state of the vision industry, developments, trends and the state of the market for vision products.

E.1 Historical dilemmas

The modern history of Machine Vision is one that for good or ill is peppered with a wide range of researchers and developers operating sometimes in coordination with each other, but frequently in isolation, with great duplication of work across institutions (universities, commercial research departments), industries (wood cutting, steel cutting, textile production) and countries. With reference to Don Braggins' 1990 paper on the state of Machine Vision in Europe [BRAG], he clearly expresses this sentiment with respect to development during the 1980s:

> *"It is impossible to look over the decade of development of vision in Europe without feeling a sense of sadness at the enormous amount of wasted effort, both from business failures and from parallel development carried out in mutual ignorance.*
>
> *On the other hand, the multitude of business or technology failures does show the value of the free economy in creating the best deal for the user – in*

the long run – and strongly bears out the 'survival of the fittest' theory of evolution, as applicable in the economic world as in the animal world"

Many of the trends mentioned by Prof. Braggins are still evident in the Machine Industry at the present date of writing. While the advent of the Internet has overwhelmingly assisted with cross-border awareness of MV research, very few resources exist for education and production system sales as compared to other industries, such as computer software production. The fundamental cause for this dilemma is the nature of visual investigation; the majority of endeavour occurs with the inspection of specific products and environments with significant constraints. It is understood that Machine Vision does not mimic human sight nor reasoning, but through the application of constraints and foreknowledge of the situation in question, is able to perform deductive reasoning and actions as a result of simplified logical paths. As a result, work done is highly directed toward the problem at hand, and often does not include generic elements that most researchers may consider suitable for announcement or entry into the wider body of common knowledge.

For example: Knowing that a corporation requires a groupware information management system for e-mail, calendaring and task lists presents an implementer with a limited number of off-the-shelf products and solutions, which operate in a manner that most systems administrators will understand. Development on one of those limited products will rapidly become well known by the competing entities and by the client base, thus contributing concepts to the common understanding. However, Machine Vision systems for wood cutting are more specific in operation; questions to consider are:

- Size of the wood
- Colour of the wood (and variation)
- Knotting and grain shape, size, direction and variation
- Finish
- Cuts to be made
- Volume of off-cuts
- Lighting (ambient and controlled)
- Environmental dust
- Dust interactions with the camera and lighting
- Position of cutting implements and safety precautions

- And many more.

Due to these issues, there is reduced potential for the creation of a universal Wood Cutting Machine Vision System, and the degree of customisation for such a system would be high. As a result, the systems created for such problems are generally funded by existing wood cutting companies, for the resolution of unique problems in precise locations. Researchers then work on creating such a system, and while solving the problem, do not necessarily generate data, methods, tools or products valuable to others working in the field of wood cutting systems. Advancements remain localised, and are not merged into wider awareness. In all fairness, the sum of Machine Vision knowledge, due to the specificity of the problems is immense, and the number of unique products is likely far larger than deployed commercial off the shelf software offerings, but the difference is that while software plays host to bespoke solutions, the generic products lead market understanding and awareness.

Due to the lack of international announcement and monitoring of the Machine Vision sector, the degree to which information about trends can be derived after more than a few years becomes somewhat limited. It is best, therefore, to focus on the most seminal works associated with the subject, the local passage of developments within the United Kingdom and long-term patterns of development in other countries, followed by the present state of the market and research.

E.2 Early Machine Vision development

Examining the history of Machine Vision tends to show a range of organisations with a consistent ability to incorporate, make progress in the field, wind up and then pollinate the rest of the industry with expertise from the remaining staff. In general terms, the progression in development since the 70s has been to move from hardware-based solutions utilising pattern recognition techniques towards software processing mechanisms implemented on workstation/server systems and embedded CISC-like processors and chipsets.

70/80s	**Current**
Custom Signal Processing hardware	Software processing
Bespoke hardware	Commodity CISC-like chipsets
Assembly-level programming	High-level visual IDEs

Large industrial problems Diverse scale problems

In addition, of course, prices for solutions have declined, and performance has increased. In the case of Automated Visual Inspection / Machine Vision, the effects of these developments have been more productive than the majority of computing endeavours. In most computing problems, an increase in processing performance is useful and introduces reductions in running times for complex operations; however, prior to advancement, fields such as video film processing may compensate through the use of larger computing devices and tools such as render-farms and clusters. In the case of MV, however, whilst the progression of processing power has enabled the reduction from mainframe-level computing to workstations and mobile/embedded systems whilst adding commensurate improvement to the same device classes, reduction in adequate form factors enables utilisation of technologies where previously they simply would not physically fit. It makes for a strained engineer/computer scientist to attempt to install a mainframe beside a production line in a noisy/dusty/vibrating factory environment, but development has allowed a fan-less x86 embedded chipset to be placed in a small box exactly where needed, and remotely administered.

One of the alternatives to installing a mainframe beside a production line, of course, is to install a camera system and drag expensive well-shielded cable between the camera and the mainframe in a more desirable area of the site. Once again, this is an expensive solution which may degrade image quality and the accuracy of response to imagery; (wirelessly) networked digital cameras provide further improvements and additional features, such as inter-site broadcasting. This enhancement of facilities has also gone hand in hand with a reduction in cost (hundreds of pounds or a few thousand as opposed to hundreds of thousands, if not millions scarcely a decade and a half previously).

Commodity equipment, falling costs, ease of deployment and faster/cheaper development through visual development tools have all been factors contributing to increased use of MV tools and techniques in large industrial and particularly small-industrial environments. This movement in scenario has also broadened the tasks to which MV solutions have been applied:

70s/80s problem types:

- Targeted welding
- Circuit board drilling [HALE]

- Soldering connection inspection [BART]
- Visually-targeted tool/material selection [KELL]
- Inspection of containers (jars/bottles) [BAT4]

90s/current problem types:

- Checking for surface defects in wooden panels (by texture analysis) [PEN]
- High-speed printed web inspection (textiles, newspapers, etc.) [NOR]
- X-ray inspection of non-uniform objects (poultry cuts, canned foods, etc.) [PEN2]
- Intelligent farm produce grading [MAR]
- Satellite image processing [SAN]
- Visually-assisted mass-production CPU binning
- Surveillance image processing and storage
- Car number-plate detection for visual congestion charging
- Bar code tracking of stock

The trend in these examples, has been towards less uniform conditions and products, with more ambiguous tasks with higher tolerances. Also, the value of benefits demanded from such systems has also declined in proportion to the reduced cost barrier to entry. For example, all but the smallest local retailing businesses are able to utilise tools such as barcodes in an effective manner. Moreover, larger businesses such as international mail-order companies or supermarkets are only able to operate as a result of such technologies; being able to find stock in warehouses using AGVs and staff with handheld barcode readers providing rapid stock location, quantification and check-out facilities. Scaling of this kind, based on the application of MV techniques has increased efficiency in industry at all stages of production/operation of services and implicitly impacted many further areas of life than previous restricted Vision problem types.

In terms of the organisations involved, Machine Vision has a heritage with a similar migratory experience to the problems and types of hardware being utilised. The 1960s and 1970s, having been a hotbed for Vision theory development, had provided a range of algorithms and concepts especially in terms of object modelling, image analysis and pattern recognition (PR), ready for use in industrial situations. In the late 1970s and early 1980s, pattern recognition methods and those methods formally described as 'computer vision'

began to be considered for industrial use. In his seminal paper, *'Industrial Sensory Devices'* [PARK], J. R. Parks begins by lamenting the sparse use of PR in commercial applications:

> *"... The only significant commercial use is that of character recognition where, starting from the early systems for reading highly stylized character shapes printed (in magnetic ink) on bank checks, we now have devices capable of recognizing only lightly stylized handprinted characters, though at some cost"*

In short order, however, Parks outlines that industrial problems are ideal for the progress of commercial applications of PR, since they do allow for sufficient control of a scene to constrain a problem, similar to that of the controlled stylised characters on the machine-readable cheques. While natural vision tasks may be complex, this ability to simplify the problem through physical means enables problem solution.

> *"Regrettably, there are not many "natural" patterns which are susceptible to manipulation in this way. ... The industrial situation is attractive as it is susceptible to a degree of manipulation to make the problems encountered more tractable."*

Parks also mentions that initial tentative solutions to industrial vision tasks had begun to be developed in the late 1970s, but they faced two problems that still present themselves in Machine Vision solutions currently:

> *"It is conjectured that the reasons for this apparent failure are twofold and interdependent. One is the seduction of new entrants to the field by the beguiling familiarity and excellence of animal PR mechanisms. The other is the lack of glamour of apparently simple application, such as inspection, counting, grading and sorting of mechanical components, etc."*

While questions of lack of interest in industrial vision may be unsubstantiated, the decline in numbers of students of Engineering in higher education in the United Kingdom at present indicates a trend that reaches beyond the realms of Machine Vision and bears out Park's concern. As to the first problem, of vision novices assuming knowledge of vision because

human beings naturally acquire images and derive information, this problem is ongoing, so much so that it leads Batchelor & Whelan [BAT3] to comment:

> *"Everybody (including the customer) thinks that they are an expert on vision and will tell the vision engineer how to design the machine"*

While initial attempts to solve industrial vision problems met with high costs and high rates of failure due to these and other problems, they did show promise that needed to be made more economically viable. The financial aspects of such systems and the long-term potential of MV systems for batch-process manufacturing and machine inspection were discussed and refined in papers such as *'A Technical and Economic Assessment of Computer Vision for Industrial Inspection and Robotic Assembly'* by Kruger & Thompson, 1981.

From the early 1980s, industrial vision solutions began to proliferate, and their details were published in a number of papers (see the lists of tasks by decade previously in this chapter).

Due to constraints of commercial-of-the-shelf computing devices, however, it was down to dedicated hardware devices (particularly PLCs) to perform any image analysis of the type desired for MV tasks in the 1980s and early 1990s. The expensive nature of system production and the limited number of skilled researchers and developers in the field by the turn of the 1980s were but two factors that hampered creation of mainstream Vision systems. As recorded by Braggins [BRAG], large-scale industry, such as automotive construction and electronics production were the first to implement 'seeing computers' for critical tasks, thanks to their cultivation of dedicated research groups within their companies, who researched and provided solutions for their business needs. In America, the General Motors and Ford companies lead the way, whilst in Europe organisations such as Saab, Siemens and Philips were able to begin to apply vision elements to their tasks.

Alongside industry-driven implementation of Machine Vision tools for their own specific purposes, however, academic developments of MV tools and techniques continued apace. In image processing, development of many crucial methods occurred, including:

- 'Statistical and structural approaches to texture', [HARA], 1979
- 'Generalising the Hough transform to detect arbitrary shapes', [BAL], 1981
- 'Image analysis using mathematical morphology', [HARA2], 1987

- 'SKIPSM 'separated-kernel image processing using finite-state machines'', Waltz, Garnaoui [WALT]

One of the most pivotal points in MV progression, however, was the advent of dedicated suppliers of MV tools to third parties, which began in earnest at the start of the 1980s. In the current market, the clear historical example to choose is that of Cognex, which began in 1981 as Dr. Robert Shillman (a human visual perception lecturer at the Massachusetts Institute of Technology) decided to pursue a commercial career. Gathering the services of Marilyn Matz and Bill Silver (two of MIT's graduate students) to start the company (initially as a summer job; they later continued on to long-term careers with the company and fill the roles of Senior VPs of PC Vision Products and R&D divisions respectively), "Cognition Experts" abrev. 'Cognex' was formed. The initial product of the company was a tool named DataMan, one of the first commercial OCR (Optical Character Recognition) systems. Throughout the early 80s, companies such as Automatix in the US, CRS in the UK, SICK Ag., Robotronics and many more throughout Europe produced commercial Vision systems for use by industry and grew in diverse fields such as cameras, optics, robotic control and image analysis.

From these companies, the first widespread use of vision for business emerged, once again from the field of Pattern Recognition that had produced the controlled OCR systems of the 1970s that Neve discussed: the barcode. Inaugurated for retail use in 1970 by the US company Monarch Marking and closely followed by the Plessey Company in the UK in the same year, the need was rapidly seen for a uniform and international system of allocating codes. Developed by George Laurer in 1973, the Uniform Product Code (UPC) set enabled barcodes to become the first internationally-controlled form of product identification, to be read visually and making the most of several key advantages of vision:

- Rapid acquisition
- Non-contact reading
- Reading does not affect the product in any way, thus making it safe for foods and sensitive products such as magnetic media.

The ubiquitous nature of the humble barcode provided a strong market for vision tools and solutions to incorporate into the burgeoning desktop computing market of the 1980s and 1990s, integrating with point of sale systems (both terminal and discrete) and enabling automation of stock control systems. The further development into 2D dot matrix codes with

the ability to store far more information is still ongoing, but is being driven especially in large volume, small space markets such as CPU production and package courier shipping. This progress is encouraged and supplied by companies including Cognex, retaining the ties to the machine vision industry.

In parallel to this growth of industrial vision solutions and commercial use, however, there has also been a continuing interest in methods of developing solutions, since the majority of industrial vision systems cannot be developed in situ alongside a working production line in a factory; it is frequently distant from the vision engineer's workplace and can be hazardous in the case of many problems. As a result, laboratory-based development is most often desirable, and during the 1980s and 1990s, tools for vision system prototyping began to be created. Two significant articles detailing these types of tools are Batchelor's *"Interactive image processing as a prototyping tool for industrial inspection"*, Computers and Digital Techniques, 1979 and the 1980 paper, *"A research laboratory for automatic visual inspection"*, International Conference on Automated Inspection and Product Control by Batchelor, Marlow, Smith and Werson. The first of these papers is of particular interest, in that it outlines the intention and scope of an interactive image processing system:

> *"There is a variety of tasks and a huge variation among the components with which inspection is concerned. With this in mind, the author has attempted to devise a general-purpose laboratory for studying visual inspection problems. ... The primary objective of this article is to emphasise the usefulness of interactive image analysis as a prototyping tool for industrial inspection."*

The paper goes on to describe classes of image processing functions implemented as part of a software application named *'Susie'* which are commonplace in modern operational libraries, such as blob analysis, texture analysis and surface defect detection.

The second paper considers prototyping of vision systems in a broader sense, including room conditions, lighting, camera choice, optics and the inclusion of *'Susie'* within that larger understanding of the scope of machine vision.

As the 1980s an early 1990s progressed, discussion of vision system creation and control diversified, exploring such areas as optics (D. J. Purll, *"Optics for image Sensors"*, Automated

Visual Inspection, 1985), expert systems (A. Novini, *"Lighting and optics expert system for machine vision"*, Optics, Illumination and Image Sensing for Machine Vision, 1988) and user interface construction for target systems (F. M. Waltz, *"User interfaces for automatic visual inspection systems"*, Machine Vision Systems Integration, 1991).

Since this point in time, significant advances have taken place in the fields of hardware, hardware control, cameras, optics and intelligent systems. These developments are beyond the scope of this thesis and contribute little to an understanding of the early history and fundamental nature of machine vision as a discipline as outlined above. Rather than explore all of these aspects, the present state of vision industry should now be discussed, establishing the situation, within which current thesis work is undertaken.

E.3 Current Industrial State

The MV industry as a whole is presently bound quite tightly to manufacturing industry, and as a consequence, sees a direct correlation in spending with capital expenditure in this sector. This situation is examined by Leo O'Connor in his June 2003 market forecast for the MV industry [OCO]:

> *"Global economic woes hit the machine vision market hard during 2002. Continued economic uncertainty is tempering capital spending so far this year... The downturn in capital spending overall, but especially in the semiconductor and electronic industries, took its toll on the machine vision market, according to a new study by the Automated Imaging Association (AIA)."*

O'Connor continues to elaborate on the fortunes of the industry, but also quotes the following data from the AIA:
Machine Vision sales by industry (USA 2001):

Semiconductor	8000 units
Electronics	8000 units
Automotive	3800 units
Pharmaceuticals & Medical	3700 units
Containers	2000 units
Food	1100 units

Plastics	1000 units
Printing	800 units
Wood fabricators	400 units
Metal fabricators	400 units
Misc (paper, tobacco, textile, military)	10600 units
Total	39800 units

Performance for the year 2002 saw a decline in both revenues (15%) and units sold, although the average sale price rose. These figures demonstrate a clear pattern of an industry dominated by sales to heavy manufacturing industry, of the type that involves the creation of products with well-defined elements and neat examination. Most essentially, O'Connor presents an assessment for the size of the global MV market as a whole: $5.2b, 126,310 units.

In terms of different applicational requirements, the AIA report finds that the US output is matched to the following fields and proportions:

Inspection	55.7%
Locational Analysis	30%
Pattern Recognition	13.9%

Interestingly, semiconductors and electronics require locational analysis in almost all applications; thus it may be deduced that inspection and pattern recognition for examination and verification are the applications mostly required by other industries, less than locating and targeting systems. This profile is consistent with large number of quality control systems and feedback-controlled fabrication.

Finally, the product type of the units shipped is analysed for the USA 2001 market:

Vision Processors	13600 units
Frame Grabbers	8000 units
Smart Cameras	7300 units
Embedded Vision Processors	4300 units

Off-line Metrology	1200 units
3D-based MV systems	1000 units
X-ray-based MV systems	300 units
Web Scanners	200 units

One of the product areas that saw growth in spite of the industry decline was that of Smart Cameras, fuelled by the development of networked hardware Smart Cameras by companies such as Cognex, whose In-Sight line of intelligent cameras are able to support simple processing operations (blob finding, targeting, crack/discontinuity detection) and act as intelligent pre-processors for networked control systems, able to acquire this pre-processed data from multiple devices and perform educated responses. These developments are very much in keeping with Neve's *'Intelligent Camera'* products produced by Image Inspection Ltd., which performed in the same manner.

In this appendix, the reader has observed the history of machine vision as a subject, from the formation through the coming together of robotics, optics and pattern recognition to the development of machine vision from hardware components to combined software systems on commodity devices with hardware interfaces.

The progress in developmental tools, IDEs and prototyping environments has mirrored the growth of machine vision as an industry, with the necessity of improving the effectiveness in system creation for the limited number of trained vision engineers.

The progress of vision systems has seen a great deal of duplicated effort and lost productivity through business failure; it has, however, provided a solid foundation for industrial machine vision solutions that compose the majority of present machine vision market activity.

Appendix F: Technology Review

The extensive use of networking to form the basis for the solution of tasks, most especially for Vision and industrial development is still very much in its infancy. Since the development of the World Wide Web, and the entrance of the technology as a tool, organisations have grappled with understanding how large numbers of disparate techniques and standards might be brought together to form a realised benefit.

Networked vision systems and the Myriad implementation involve the interaction of a range of disciplines; hardware systems (commodity computing hardware, servers, desktops, PDAs, mobile devices, logic boards and embedded systems), software (scripting, Java, Linux) with networking technologies (wireless access, high speed ethernet connections, networked cameras, network security techniques), web technologies (web servers, HTTP, databases, XML, web services) and programming methods (Open Source development).

As such, the reader will be exposed to a wide range of technologies, techniques and terms of which they may have prior knowledge, or which may be beyond their previous scope of experience. This appendix serves as an introduction to implemented technologies, a description of wider considerations when applying those technologies to networked vision systems and provides the reader with requisite context for later technical discussion.

F.1 Java

Java was developed in 1991 by James Gosling at SUN Microsystems as an attempt to produce both a simple object-oriented language and one that could be used in programming applications across multiple platforms [LEM]. Originally known as Oak, a product of Project Green, it was intended to be a cleaned up and user-friendly version of the C++ programming language. Announced publicly in 1994, adopted by Netscape in principle into their Navigator web browser product in 1995, Java 1.0 was formally released in 1996.

In order to achieve the first goal (that of a simple OO language), Java was defined as a modified subset of C++, the object-oriented form of C, with features such as multiple

inheritance and pointers removed [LEM]. The Java language itself was then augmented by a set of standard object classes named the Java Foundation Classes (JFC) to perform standard operations, such as file I/O, string processing, creation of graphical widgets, etc. In comparison to other programming languages, the JFC performs a number of functions of the standard language, and other functions, such as windowing, which are generally offered as extensions to other languages. These are commonly referred to as Modules and Toolkits.

The secondary goal of cross-platform compatibility was achieved through the creation of the Java Virtual Machine (JVM). In contrast to other languages, where source code is either compiled into machine-specific programs (binaries) or run by an interpreter which sequentially runs the source file as it reads it, Java programs are compiled into a executable binary for a machine that does not actually exist. This machine is a model computer, designed to run Java applications. This machine is then emulated on real computers through a JVM program and serves to translate the Java code, now compiled into JVM machine code (named byte-code) into commands understood by the real system (see fig.117). For each system that Java will be run upon, a new JVM must be written, but this does allow Java code, once compiled to byte-code to be executed on new platforms without alteration. The slogan "Compile once, run anywhere" was coined for SUN marketing purposes, but illustrates the point in a succinct manner.

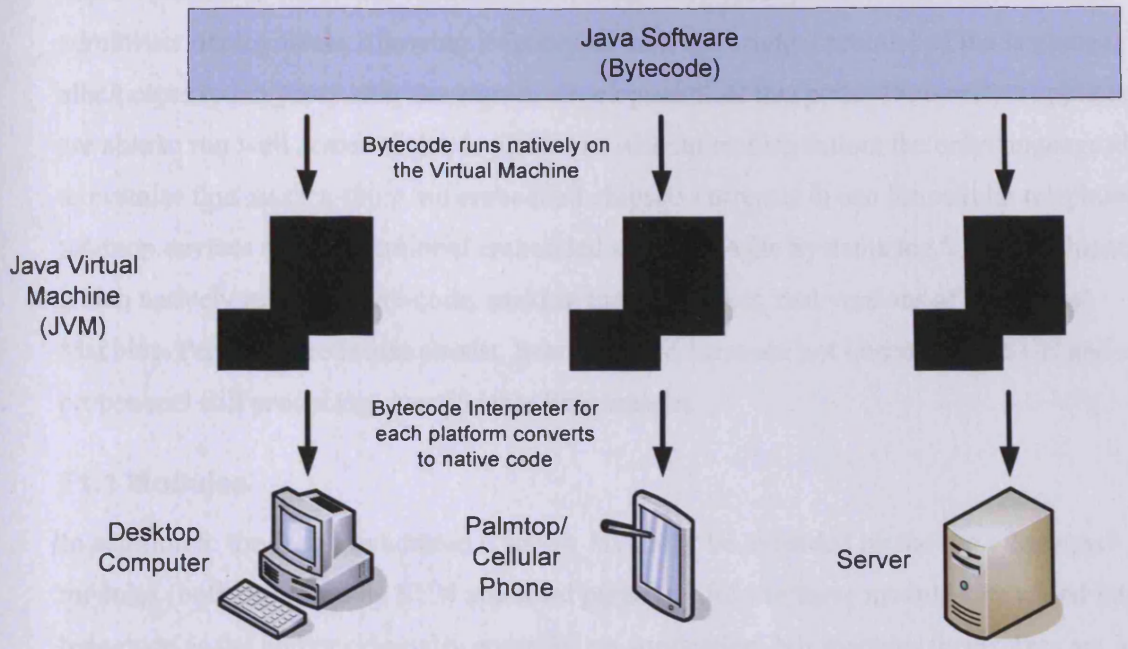


Figure 117 : Uniform Java 'byte-code' being executed uniformly across multiple computing form factors and devices through intermediary operation on a Virtual Machine

In reference to the JVM structure, it is worth mentioning that due to the overhead of running the JVM program and interpreting byte-code into machine-specific instructions, Java applications generally perform poorly compared to applications written in other languages (such as C++), which are compiled directly for the machine. There are a number of published benchmarks showing Java code running as fast as, if not faster than native code, but these typically show simple loops executing faster than native code, due to JVM optimisations. In these cases, the C++/C compiler is poor if it is unable to produce the same (or better) optimised code that the JVM produces. It is also worth noting that these benchmarks do not act in the manner of real applications, failing to use any objects or method requests, which provides an unrepresentative profile of an object-oriented language (and indeed, Java suffers large performance penalties when objects are created and used).

Due to differing JVMs on different platforms, performance of running Java applications may vary greatly, not to mention large memory and performance requirements of the JVM itself and characteristics that change depending on the version and host system.

Until the minor point releases of the version 1.3 Java, the language was highly unstable, with 1.1 code incompatible with 1.2 and then 1.3, and worse, across JVMs for the same versions.

Since version 1.3 stabilised, Java has become far more attractive as a solution and easier to administer deployments, allowing it finally to fulfil the original promise of the language, albeit close to ten years after the start of development. At this point, Java-written applications are able to run well across multiple platforms without recompilation; the only language able to promise this. In fact, there are embedded chipsets currently in use for cellular telephones, palmtop devices and conventional embedded systems (Ajile Systems Inc.'s AJ100 chipset), which natively run Java byte-code, making them, in effect, real versions of the Virtual Machine. Performance issues persist, however, and these are not improved by SUN and other proponents still producing questionable benchmarks.

F1.1 Modules

In addition to the Java Foundation Classes, Java may be extended by the use of external modules (both produced by SUN and third parties). Links to these modules are added into the byte-code as the author originally compiles an application, but modules themselves are not compiled into the application. When the program is actually run, compiled versions of the modules, present on the target system are included into the program. This allows modules to be rewritten (so long as the interface is maintained) without programs using those modules, having to be recompiled and redistributed.

COMM API

The Java Communications API is an extension to the standard Java Foundation Classes, which allows access to communications hardware on the host computer. When the API was first produced, that merely involved serial port functions, but it has been extended to Universal Serial Bus (USB) connections, Parallel port and Firewire connections. SUN currently supports downloadable versions of the COMM API for Solaris and Windows 95/98/ME. For other platforms, third party versions do exist, but caution should be used, since all do not share the same capabilities, syntax or internal methodology.

JMF API

According to Hansmann et al., *"...the Java Media Framework (JMF) is used to easily create players for real-time media, such as video, audio or text-based data."* [HAN]

The JMF is a wrapper around existing operating system video architectures, such as Video4Windows or Video4Linux. Effectively, it provides an API for controlling and

capturing video streams, converting them into Real Time Streaming Protocol (RTSP) networked video streams, and also reading RTSP streams at a client. This allows developers to capture, manipulate and transmit video from a camera using a computer with a framegrabber. Digital video sources, such as Firewire/USB video or webcam devices are also converted, by the operating system into Video4Win or Video4Lin streams, so are accessible also.

Unfortunately, due to the way that video is handled by differing operating systems, JMF applications are not suitable for cross-platform use (Sun presently only offers Solaris and Windows editions, both unstable test versions), and all systems requiring it must also install the JMF runtime. Moreover, the JMF framework is still highly developmental, leading to new revisions of the system breaking previous applications in significant ways, forcing code sometimes large-scale code changes. These factors lead to problematic deployment of JMF-enabled applications.

F.2 C++ /QT

C++ is an Object Oriented language that was developed as an offshoot of C, but now can be seen more as a superset of that language and the origin of the Java language also, according to Liberty [LIB]. While Java was designed to be a lightweight version of C++, it lost a number of pieces of functionality such as pointers, multiple inheritance, bindings to other languages and direct memory access (Lemay, [LEM]). C++ is typically used in coordination with C for operating system functions, servers, interfaces and user-level applications. Extensions are provided to the basic language using a range of libraries, both common to the language and from third party contributors and organisations. Often, these libraries come in collections aimed at providing specific classes of functionality, such as visual widgets, database connections, data structures, etc. These collections are referred to as Toolkits. Of the range of toolkits, perhaps the most visible of these are the GUI or Windowing Toolkits, which provide the abilities to render interfaces and interact with widgets and visual objects.

QT is a Toolkit developed by Trolltech AG. Supplementing the abilities of GUI toolkits, QT provides a range of functionality, from I/O to data structures, string manipulation, database connectivity, XML parsing and graphical processing. QT attempts to provide base classes for all tasks a programmer might wish to perform, leaving them with the task of connecting components and producing higher-level code, much like Java's JFC. Ward [WAR] describes a

key advantage to using QT in this manner: that the resultant program is reliant on QT for access to all low-level functionality; it need not know exactly how Windows, X or OSX renders a window to the screen, for example, because QT takes care of that interface. Since programs are therefore free of the majority of platform-specific code, they may be recompiled on any platform that has a version of QT available for it. Once compiled, it acts as a standard native application, without having to run virtual machines or install additional runtime libraries.

QT currently operates under a range of operating systems, including Linux, Embedded Linux, *BSD, Solaris, AIX, MacOS 9, MacOSX, Windows 9x/ME/NT/2000/XP, QNX and WindowsCE.

F.3 Image Processing Toolkits

Along with toolkits for GUI functions, there exist a range of libraries for Image Processing functions, which may be used with C++ applications. While there are too many to mention here, we will draw note to the following examples, which have been relevant in Myriad's construction.

F.3.1 QT basics

The QT toolkit itself offers a small range of image processing functions within the standard QImage and QPixmap classes, which are optimised by a compiler once the QT application itself is compiled, and as such, are useful substitutes for hand-written code. These functions include mirroring, flipping, scaling, negation, masking, depth conversion and re-colouring. While these functions are limited, they suffice for basic applications and the needs of standard GUI applications. For further commands, however, the programmer is required to write their own additional operations; while being given direct access to 32bit arrays of images in memory, with the option of using different methods for more complex access and manipulation.

F3.2 OpenCV

OpenCV is a comprehensive open-source Computer Vision library created by the Intel Corporation for use on x86 processors, running either Windows or a form of Unix (including Linux and the BSD family). After a number of years of development, Intel released the library

as an Open Source project maintained on SourceForge.com with the objective of developing a Vision-based community of programmers and integrating Vision techniques into projects and long-term developments of (CPU-intensive) programming technologies.

OpenCV is continually in a process of growth, benefiting significantly from the programming community, and also research organisations, such as academic institutions. Because of the easy deployment of the library and the freedom of use both for prototyping and commercial use, it provides a desirable platform for the development and integration of research work, with the assurance that improvements may then be immediately used by industry in a beneficial manner.

At the present time, OpenCV covers a wide range of Vision operations, including, but not limited to, 2D convolutions, morphology, object analysis and processing, 3D analysis and processing, 3D reconstruction, structural analysis, motion analysis and tracking and complex recognition tasks.

While Matlab is an ideal environment to develop prototype operations, explore Vision tasks, integrate with modelling and statistical systems and mathematically examine results with ease, OpenCV is a highly useful, easy to integrate library for the production of programmable target systems. For this reason, Matlab and OpenCV-based systems are highly complementary both in research and actual use.

F.3.3 Matlab

Matlab is a mathematical development and prototyping environment created by MathWorks Inc. centred around a procedural programming language with C/C++/Java bindings with an emphasis on the use of functions to support the development of non-linear scripts. Expanded through the inclusion of 'Toolkits' (pre-defined function libraries and additional non-Matlab binary interfaces to hardware and system APIs), Matlab offers a powerful development environment for experimental design of algorithms and systems. Due to the performance limitations of Matlab (a complex scripting language being interpreted atop a mathematical processing engine with all of the associated 'Toolkits'), Matlab-produced systems are suitable for non-critical and slow problems and most particularly for creating solutions (in the form of abstract algorithms) which may be re-implemented for a target system, which will then be deployed.

F.4 Embedded Programmable Chipsets

While there exist a wide array of custom logic boards for a variety of industrial purposes, such as framegrabbers and equipment controllers, these systems lack the programmable attributes of conventional all-purpose computing systems, such as desktop computers and servers. Using logic boards for industrial systems is desirable as opposed to full computer systems, due to the stability and predictability of repetitive operation, lack of moving parts and reduction in costs over systems which combine a number of components. These systems also benefit from a reduction in complexity that leads to fewer points of hardware failure. However, there are situations where tasks are varied, or must rely on detailed input and complex logic, which are beyond the abilities of a custom-produced logic board. In these cases, a general purpose processing system is required with the advantages of a logic board. To satisfy these requirements, a range of products termed Embedded Chipsets were produced; essentially, a general purpose processor mounted on a logic board, with larger amounts of memory, typically capable of running a full desktop-grade operating system and being programmed remotely or by serial connection.

Johnson and Jennings [JOHN] examine the case for the use of Linux on embedded (mostly x86 compatible) chipsets as opposed to proprietary solutions and resolve the following factors for choosing the embedded multi-purpose solution (paraphrasing for brevity):

- Low cost: Developers can take advantage of the software community and the work done in creating full desktop-grade operating systems. In addition, for short-run products, the expense of learning new systems and licensing software is alleviated.
- No cross-compiler is needed. Development systems operate using the same OS as the embedded device. Software developed and compiled on desktop systems is directly portable to the embedded system.
- Diversity of Hardware: The benefits of a desktop operating system allow embedded systems to operate with elements such as USB devices, old generation processors, unusual storage devices and output systems, which are designed for use with desktops. This also has benefits in interfacing with industrial equipment meant to be managed by a desktop computer.
- Adaptability: The general desktop kernel can be used on systems as small as mobile telephones and as large as mainframe servers, with complex hardware configurations,

executing all manner of applications, from GUI applications to servers and database management systems.

- Real-time Extensions: Using RT adaptions, embedded systems can be made interruptible as logic-based hardware systems, ensuring safety and immediate response, while retaining the benefits of large software applications.
- Multiple Pre-built Communications Options: Modern OSs are designed to interact across several communications mediums, from Ethernet-based networks, to wireless, mobile telephone data, Infra-red data (IrDa), implicitly providing embedded systems with pre-built networking abilities.

Primarily, these factors result in simple reductions in cost and time of development and licensing for control and management systems and integration with existing larger computing solutions

F.4.1 MMB

'Mike's Magic Box' (MMB) is a custom logic system built by Michael Daley of Cardiff University to provide a complex I/O interface to a conventional host computer.

The MMB is capable of:

- Multiplexing 8 channels of video
- Switching mains power to 10 devices
- Controlling compressed air supply
- Relaying data to 6 serial and 6 parallel connections

Prior to the use of Garry Jackson's array of logic boards, the MMB was the primary means of operating large ranges of equipment simultaneously. Most often, the MMB was used in the control of a Flexible Inspection Cell (FIC) [BAT94]. The MMB was also closely used with a piece of software named JIFIC as described in Batchelor and Waltz's *"Intelligent Machine Vision... "* [BAT2] to remotely operate the FIC and forms the basis of the *'Remote Operating Prototype Environment'* as described in [BAT01].

F.4.2 J43 Design Interface Tools

Cardiff University's PhD student, Mr. Garry Jackson developed a series of equipment control logic boards, based upon a commercial task for digital control of analogue zoom and focus for

high-resolution video cameras. Mr. Jackson's company, J43 Design, markets these boards under the name of 'M1'. Recently, these tools have seen the addition of the M2 board, with enhanced I/O capabilities.

Control (M1)

The control boards produced by J43 Design use a serial RS485 bus for communication, with multiple boards able to be daisy-chained together (see fig.118), each with a unique identifier, set through the use of a set of DIP switches.

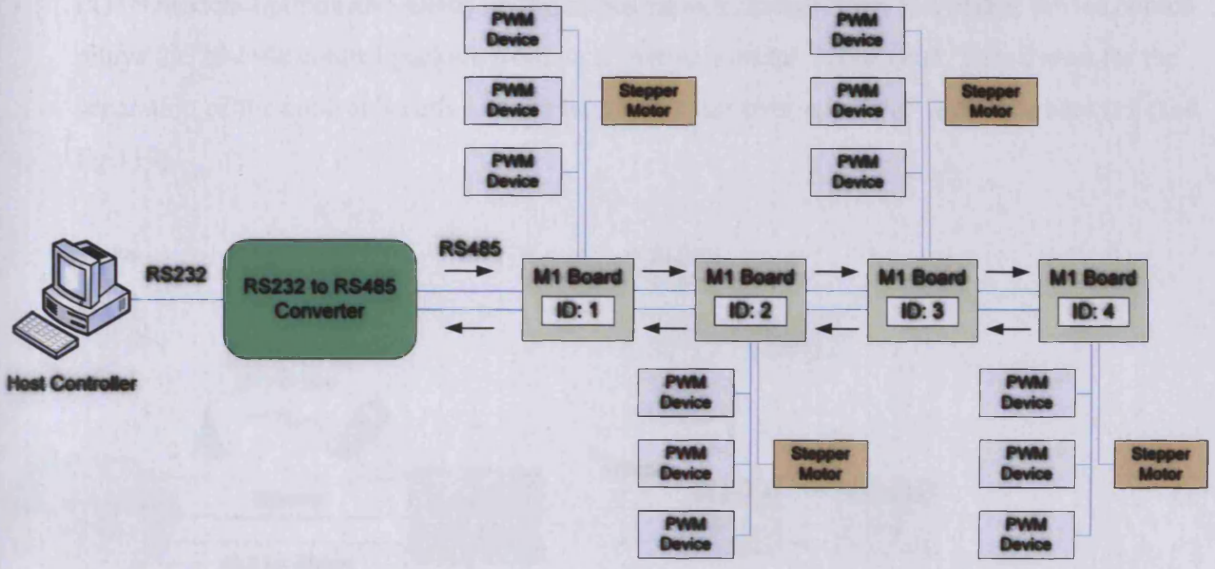


Figure 118 : Control of daisy-chained M1 boards and devices through initial RS232 output from a conventional computing device

Communication from a host computer to the board collection is performed through RS232 with an RS232 to RS485 bridge device. Interaction is two-way, with commands issued from the host as a set of 20-byte packets, eliciting a confirmation response or status information.

Equipment control is performed in one of two ways:

- Stepper motor control (1 per board)
- PWM H-bridge control (3 per board)

This allows for control of stepper motors, motors and switches, sufficient for control of small and medium devices and operating switching for larger systems.

Gateway

While the control boards are highly advantageous in general control, especially for prototyping environments, they are limited in terms of use due to the requirement of a local host on the RS232 connection. In industrial environments, the use of a full computer with associated moving parts and susceptibility to dust and environmental hazards, highly local to the control boards (RS232 length is limited in practical use to 5 metres) is not feasible. Responding to this limitation, J43 Design also produced a 'Gateway' board. With an RS485 connection on one side and a 10MB/s 10BaseT networking connection (cellular data and POTS modem options also exist) on the opposing side, the gateway is a bridge device, which relays the 20-byte control packets from an IP network to the RS485 bus. This allows for the separation of the control boards and the host computer over a local or wide area network (see fig.119).

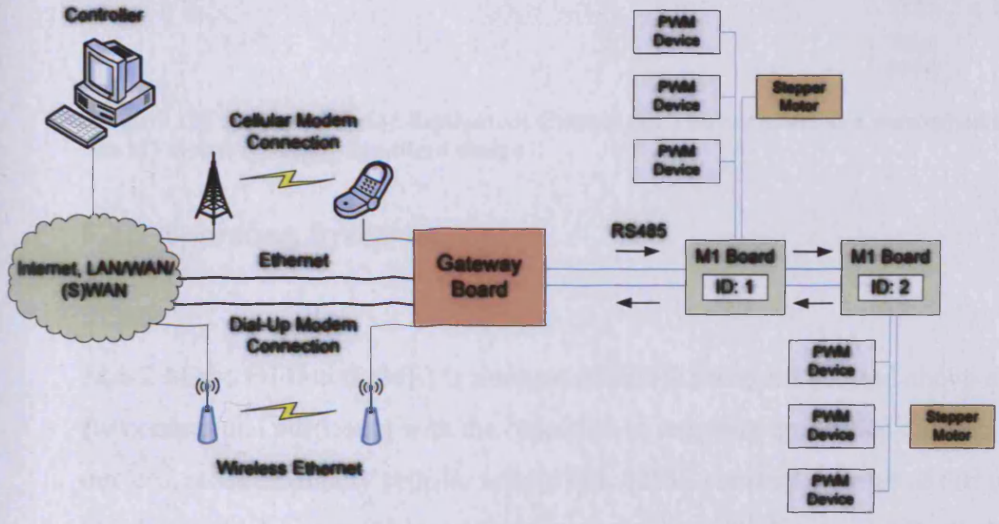


Figure 119 : Use of the 'Gateway' logic board to provide a network bridge for command sequences

The use of a gateway board does not negate the value of Myriad systems surrounding the M1 control boards; rather, it augments it in allowing the use of these boards in situations where they might not otherwise be reasonable or safe. While the gateway board does relay commands, it still relies on a host to have detailed protocol handlers and command construction, which is a level of granularity below that which is normal for user interfaces and hence, this difference is the ideal position for a Myriad server to inhabit (see fig.120).

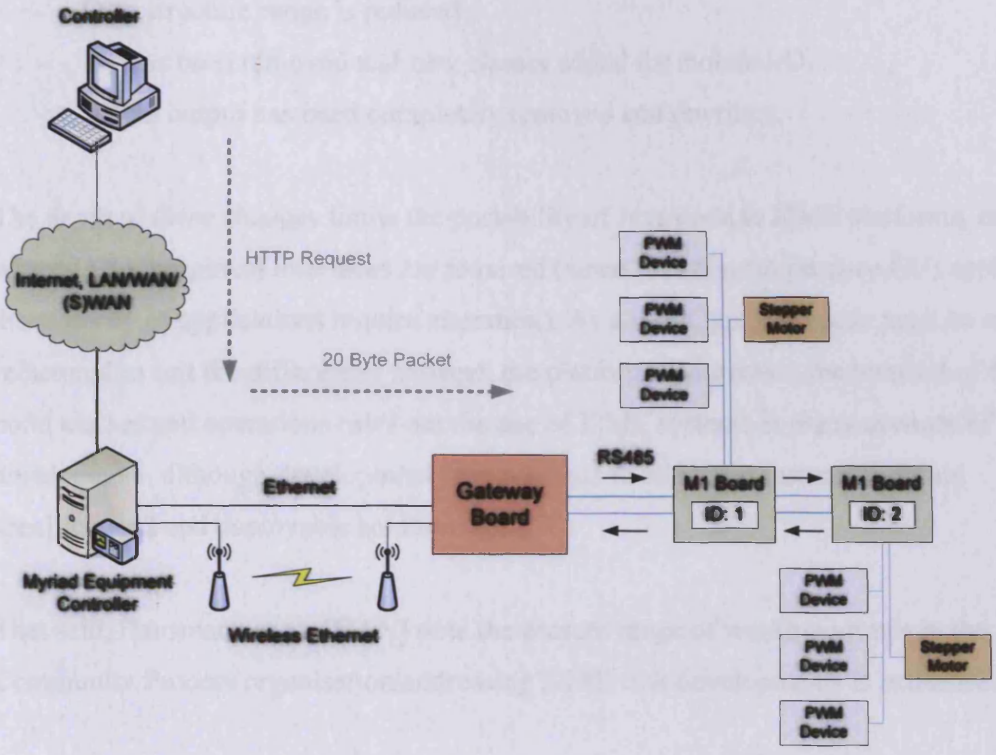


Figure 120 : Using a Myriad Equipment Control (EC) server to act as a networked interface to M1 systems, simplifying client design

F.4.3 Operating Systems

J2ME

Java 2 Micro Edition (J2ME) is a subset of Java 2 (versions 1.2 and above of Java, renamed for commercial purposes) with the objective of targeting embedded chipsets and mobile devices, most especially cellular telephones. J2ME contains a series of different device 'profiles' which customise display, input and output abilities to mobile devices and palmtop devices, where resources and form factors are more limited than a conventional desktop computer (Hansmann et al. [HAN]). J2ME typically runs using a JVM program that ships as an operating system component atop other systems, such as Windows CE or SymbianOS. There exist, however, a number of pure-Java embedded chipsets such as the AJ100 system produced by Ajile Systems Inc., which run Java natively on a 32 bit real time multithreaded processor (includes a native IEEE-754 (1985 standard) floating point arithmetic unit, two RS232 serial I/O ports, five 8-bit I/O connectors and PWM output for equipment control).

J2ME is different from conventional Java in a number of ways:

- Floating point operations are removed

- Data structure range is reduced
- I/O has been removed and new classes added for mobile I/O
- Visual output has been completely removed and rewritten

The depth of these changes limits the portability of Java code to J2ME platforms, especially where I/O or graphical interfaces are required (since Java aims to produce GUI applications, the majority of applications require alteration). As a result, program code must be extensively refactored to suit the differences between the platforms. Moreover, the removal of floating point classes and operations rules out the use of J2ME systems in many avenues of development, although development is in progress to add these features to future specifications and deployable solutions.

That said, Hansmann et al. [HAN] note the present range of working groups in the Java Community Process organisation addressing J2ME task developments is extensive:

- Wireless mobile phones and communicators
- Point of sale terminals
- Handheld computers
- Automotive systems
- Set-top-boxes (STBs) and interactive television
- Generic network-connected GUI devices

Linux

There is little to say about embedded Linux, since it essentially entails a version of the standard Linux kernel, with unwanted features, such as large disk support, SMP operation and other high-end features removed. The embedded kernel operates in an identical manner to desktop or server kernels and may be programmed in the same way (Johnson & Jennings [JOHN]). Sympathetically programmed applications (that is to say, ones which do not rely on high-end features which would be absent) will often run on embedded systems as they would on desktops, without recompilation, provided the kernel minor version is the same. This allows for easy development on desktop and server systems, prior to deployment, where the kernel provides unified APIs for I/O and general functions. Windowing and display operations, may potentially require translation, but toolkits such as QT/Embedded provide uniform APIs for those as well.

Embedded Linux systems often operate atop a Real-Time (RT) modified kernel[3]; a kernel modified to operate on the given embedded system, with access to timing systems and being fully interruptible, a feature which speed-critical hardware systems find essential. While applications themselves are not innately RT, they can be used in a threaded environment, where interruptions raise events, which can be managed by other threads and communications between threads are then able to take place, giving data to normal programs or transferring data into the memory used by those programs, for later use. Examples of LabVIEW and Real Time Linux being used together are provided in Johnson & Jennings' book [JOHN] by P. Daly of the US National Optical Astronomy Observatories and Aachen University's (Germany) T. Leibner.

WinCE

Windows CE (WinCE) is the version of Microsoft Windows written for palmtop devices (and to a lesser extent, embedded chipsets), later renamed Pocket PC, Pocket PC 2002 and most currently Windows CE .Net.

Windows CE is the core of the system, based around a completely fresh code base, which actually shares no code with Windows, neither the 9x/ME series nor the NT/2000/XP series. At the present time, however, WinCE does compose the majority of the palmtop operating system market, due to a wider range of features than the aging PalmOS.

The principle reason for the relative market success of Windows CE has been the ability to develop applications for the platform using Microsoft's Visual Studio visual Rapid Application Development environment package, alongside desktop counterparts. At the current time, WinCE is also the key driving force behind the production of new devices with increased resources and processing power, since it has in excess of twice the memory footprint and requirements of PalmOS, or embedded Linux. While this does increase the cost of the platform as a whole, as Hansmann et al. [HAN] mention, the ethos of WindowsCE development is that the PDA is a miniature desktop PC, rather than a targeted device. As a result, many more features, such as IrDA support, USB connectivity and external monitor support for presentation projection are included into WinCE devices.

[3] Such as that offered by TimeSys' TimeStorm product line (http://www.timesys.com/), RTLinux by FSMLabs (http://www.fsmlabs.com/, likely the most popular real time Linux product) and LynuxWorks' BlueCat software, which goes hand in hand with their LynxOS hard-RT operating system product (http://www.lynuxworks.com/)

At the present time, Microsoft is extending Windows CE with additional components to produce 'Windows XP Embedded', forming the basis for gaming, Set Top Box (STB) and Home Theatre PC (HTPC) platforms. Windows CE has also been applied to other areas of computing development, such as intelligent controllers for in-car automobile non-critical systems for BMW, albeit with mixed results.

It should be noted that Windows CE is not a system currently capable of being made Real Time-aware, in either the soft or hard sense. As a consequence, it may not be used for tasks which demand assured rapid response and immutable feedback to events, such as industrial equipment control or the critical systems of those same BMW automobiles referred to above.

SymbianOS

SymbianOS (aka Epoc) is a custom-developed operating system for mobile devices and small form factors, originating in the field of palmtop devices, along with Windows CE and PalmOs, and has found the majority of the current deployments among cellular telephones, especially Smart Phones. As with PalmOS, SymbianOS is limited by the lack of desktop operating system for direct portability of application code from the desktop computing space, but it does provide specific support for mobile form factors, avoiding the restrictions of re-implementing desktop programs and mechanisms merely on a smaller scale. For this reason, SymbianOS was the first mobile OS to integrate camera devices and streaming video functionality. SymbianOS, therefore, may be considered as a lightweight, mobile-centric, modern multi-threaded OS.

QNX

QNX is a POSIX-compliant UNIX-like micro-kernel operating system for embedded devices, which also runs on Intel x86, PowerPC, MIPS, ARM, xScale and StrongARM architectures, designed with a view to producing a lightweight, high-performance and responsive environment for mission-critical systems and based around hard real-time principles. Originally conceived as the product of University of Waterloo, Canada, students Gordon Bell and Dan Dodge, Qunix was formally released in 1982 by their development company Quantum Software Systems, later to be renamed QNX. Continual developments, redesigns and refactoring, with the assistance of a community of embedded systems developers, QNX is widely considered to be the most reliable embedded operating system presently in use. The

most current version of QNX is named QNX Neutrino, and features a further redesign to emulate many architecture features of Linux, enabling simplified porting of applications and hardware drivers. For high-reliability computing devices, QNX is a preferred choice, especially in safety-critical roles where low failure rates and real time operational interruption are essential.

F.5 Palmtops & Mobile devices

Traditional computing equipment form factors vary between laptops, small form factor computers, desktop computers, servers and mainframes. With the constant reduction in size and power requirements of processors, memory and logic chips, however, a market for digital organisers emerged, serving as a computational equivalent of a paper organiser, with the ability to synchronise data with a desktop computer. As the quality and performance of embedded chipsets improved, however, it led to the birth of the cellular telephone industry, and alongside that, the palmtop market, for small systems capable of a full range of computing tasks. Current palmtop devices match and even exceed the performance of commodity computers from five years ago, making them a valuable target for software systems, especially, in the case of Myriad and Machine Vision in general, roaming graphical controllers and image capture equipment, thanks to advances in CCD development and in-built digital camera technologies.

The use of mobile devices and connectivity via wireless communications to standardised networks such as the Internet creates a new form of computer interaction, presently termed '*Pervasive Computing*'. Hansmann et al. [HAN] define this in the following manner:

> *"A new class of devices make information access and processing easily available for everyone from everywhere at any time. Users get enabled to exchange and retrieve information they need quickly, efficiently and effortlessly, regardless of their physical location."*

In terms of a Machine Vision worker, this means access to equipment control, image processing tools, imaging resources and knowledge bases (such as a Lighting Advisor) from any location. Equipment on a production line can be monitored by an Engineer from a moving train, or images captured by a palmtop device in a public place may be remotely analysed and stored at an office by a worker, for later review.

F.5.1 PDAs

Personal Digital Assistants (PDAs), or Palmtops as they are often referred to, are hand-held mobile computing devices that developed from digital organisers. Most commonly used for Personal Information Management (PIM) applications, such as address books, to-do lists and calendaring, with the advent of the PDA as a distinct platform, more varied applications have appeared, from word processing to voice note taking, presentation and video playback, gaming and web browsing. At the time of writing, palmtops generally include LCD colour screens, capable of a resolution of 320x240, sound reproduction, input via stylus and on-screen keyboard (although small extendible keyboards are available for some models) and communication via cellular telephone connections (either directly or through Infra-Red) and wireless ethernet access. Hansmann et al. [HAN] note that the palmtop computer is now one of the leading platforms for internet-connected, mobile pervasive computing initiatives.

More recently, PDAs have increasingly provided more memory, faster processors, external storage through flash memory devices and greater extensibility (such as the Compaq iPaq, which can be stored in protective hardened sleeves, which also include extra batteries and slots for laptop PCMCIA expansion cards, or connect directly to external monitors and LCD projectors).

F.5.2 Cellular Telephones

Cellular telephones are also a platform for future computing growth, especially for roaming interfaces with equipment and monitoring. Due to the ease of connecting them through the telephone network to the Internet and wired networks, and their rapidly developing processing capabilities, they present a very real target for development of lightweight user interfaces. With the development of colour displays and more powerful processors, with the ability to download and execute non-telephonic programs, such as organiser functions, many companies see a near future convergence of the cellular telephone and the PDA, or, indeed, the PDA being subsumed into mobile telephones. Such devices are commonly known as 'Smart Phones'.

F.5.3 'Smart Phones'

As the feature set of mobile telephones increases, their usage patterns begin to mimic those of palmtop devices. However, since the persistent trend in cellular telephone development has been reduction in size and increase in battery life, certain advanced features, such as colour

screens have taken several years to enter the market. It is still the case that the large screens and advanced power-consuming features of PDAs are ill-suited to a mobile telephone form factor. The Smartphone concept, however, provides a form-factor in-between the PDA and the telephone, akin to a larger mobile, accompanied by a colour screen and a stylus for input such as is seen with a palmtop device. Hansmann et al. [HAN] report that the use of such devices is limited, but steadily growing, as the mobile telephone market grows to be accustomed to the wider range of features. The cost in terms of size and battery life, however, still limits acceptance as many telephone users specifically require that their devices only perform audio communication functions. With the advent of high speed data services across cellular networks, such as GPRS and 3G technologies, however, the Smartphone develops into more than a simple PIM (Personal Information Management) device and into the role of a network client.

The software used to construct Smartphones is part of a subset of embedded programmable device operating systems and is limited to Linux, Windows CE and Symbian OS. These systems allow device manufacturers to provide ranges of solutions with unified software bases, easing the development of applications, reducing costs of custom operating system development, and enabling multiple communications protocols over Internet-based connections. These reasons, it should be noted, are much the same as those mentioned by Johnson & Jennings [JOHN] in their review of embedded Linux systems for automated tasks, marking a clear convergence between desktop, mobile and embedded systems.

F.6 Cameras

In addition to traditional analogue cameras and video cameras, which require a framegrabber or a scanner to convert the image into a digital form, ready for Vision operations, the development of high quality and high resolution CCDs has created a market for purely digital cameras and video cameras.

F.6.1 Digital Cameras

The most high resolution CCDs at the present time may be found in still-image digital cameras, the likes of which we see in consumer and professional compact and SLR cameras. With true optical resolutions of up to 8 million pixels and super-sampled resolutions of almost twice that figure (such as the Canon Powershot Pro 1), the clarity of digital images now exceeds that of traditional film, with large flash memory cards (up to 4GB with Compact

Flash 1 ¼" hard disks produced by IBM and Hitachi under the 'Microdrive' brand; 8GB and 12GB drives are expected in the near future) able to store hundreds of those images. While these resolutions are not needed for the majority of Vision applications and indeed, sequences of images from these cameras are of lower resolution due to memory and bandwidth concerns, the ability to watch a video stream and then obtain a very high quality snapshot is compelling.

F.6.2 Card Cameras

In addition to consumer digital cameras, there also exist a range of PCMCIA cards with cameras built in. These cards simply fit into a free PCMCIA slot and are able to take snapshots from a mounted camera, such as the Winnov Videum Traveler PCMCIA camera. While the quality of the images produced is frequently limited (typically 640x480 snapshot and 320x240 live (18-25fps, depending on image complexity) and camera rotation and focussing is also constrained, they provide a useful way to obtain images from a PDA in a mobile situation. All current palmtop 'Clié'devices produced by Sony Corporation, in fact, include a camera as a standard feature, again with a 640x480 resolution. Compact Flash I/O and Secure Digital I/O cards are also available with cameras attached, to provide image capture facilities to almost all modern mobile computing devices.

With wireless Internet access (802.11x, Bluetooth, cellular phone + IrDA) added, a PDA with a mounted camera can use Myriad image processing tools to troubleshoot situations with explorative image processing techniques, producing a powerful diagnostic Vision tool.

F.6.3 Camera phones

In the face of stiff competition, the cellular phone handset manufacturers attempt to produce phones with even more elaborate features, in an effort to outdo each other. One of the results of this process, are telephones with cameras built in, or able to be connected to a particular port. While the resolution is poor (often 640x480 interpolated, including visual artifacts), they offer similar benefits to the use of camera cards in PDAs. With many handsets also sporting in-built Java VMs (including all new Nokia and Motorola models) and the majority of Smartphones also supporting a Java VM, along with the connectivity that defines a cellular telephone device, these may be used as fast problem-solving devices. The added benefit of using telephones in this way, is the high quality of cellular coverage, especially across densely populated countries, such as the UK, far beyond that of wireless ethernet access, enhancing

roaming potential and reducing the requirements of existing wireless internet access for solving mobile Vision problems.

It is worthy of note that during development of this project, the quality of cellular telephone cameras is slowly improving, with the latest cameras able to achieve 1280x1024 resolution, which is more than ample for Vision tasks, where issues such as the limitations of iris and zoom controls in uncontrolled lighting conditions will restrict the acquisition of satisfactorily high quality images. Optional additions, such as in-built flash facilities and automatic focus/iris control are progressively becoming baseline features at the time of writing. With larger images and improved features, coincidental to PIM tasks, video messaging and multi-media, the need for increased storage capabilities in cellular telephones is becoming evident. With this in mind, newer models in 2005 are boasting Secure Digital flash card expansion slots and incorporated Compact Flash Microdrives, offering several gigabytes of storage capacity.

F.6.4 Web cams

A class of digital camera with a low resolution, capable of streaming video down USB or Firewire connections to a host computer at low cost, web cams have become a highly effective means of establishing low-grade video conferencing systems. Supporting optical resolutions similar to those of mobile telephone cameras, and with the added issues of typically poor focus control and auto-illumination compensation, web cams generally are not suited for anything but the most rough shape capture. That said, they are also exceptionally cheap, and for video conferencing Vision Engineers and operators or for detecting macroscopic changes (e.g. person entering a room), they provide exceptionally good value.

The next generation of webcam closely shadows the development of the mobile telephone camera, with higher resolutions enabled through the use of faster serial connectivity from technologies such as Firewire and USB2, whose bandwidth improves on previous generations sufficiently to enable better-than-TV quality live video.

F.6.5 Pan and Tilt Camera

The Sony EVI-D31 pan and tilt camera utilises a 752x585 'Hyper HAD' colour CCD, producing output via either RCA or S-Video connections, along with monaural audio through an independent RCA connector.

Features:

- 12x Optical zoom
- Horizontal view angle 4.3-48.8 degrees
- Focal range: 1cm – 80cm
- Auto focus, iris, exposure
- Pan/Tilt horizontal: +- 100 degrees
- Pan/Title vertical: +- 25 degrees
- Controllable variable shutter 1/50 – 1/10000

Using a RS-232 serial connection, the camera may be moved using 2-byte hexadecimal pairs, which afford absolute and relative movement in pan, tilt and zoom. For normal operation, this requires a host computer, a frame grabber and a capture application. The development Myriad implementation uses VisionGS[4] as the primary capture application, with images stored and subsequently uploaded to FTP servers and web sites.

Current developments in this form of camera are tending towards implementation using high-performance CCDs and Firewire (IEEE 1394) connectivity to transmit digital imagery directly and skipping the framegrabber and capture software. Due to the limited range of Firewire cable integrity and the necessity of computational capture, rather than direct video recording (although Firewire->video recorders are beginning to emerge in the market), these products are not as favoured as CCIR models for surveillance purposes at the time of writing.

F.6.6 Network Attached Cameras

In contrast to the previous cameras, which operate in conjunction with a host system (desktop, PDA or telephone), there are a number of commodity camera systems available, which contain embedded chipsets and connect directly to a Local Area Network. These systems typically involve an embedded Linux chipset, and operate a range of servers for HTTP and FTP, such as Apache and ProFTPd, the standard Linux services.

Network Attached Cameras (henceforth, NACs) have a number of benefits over the use of a desktop-grade host system with a framegrabber or streaming from a digital camera:

[4] VisionGS is available in both personal (free) and business editions from E-training PR, Germany and is able to stream video and snapshots, FTP data to remote servers, act as a server in its own right, and provide basic pre-processing and watermarking features for output images. http://www.visiongs.de

- No moving parts
- System dedicated solely to image capture and transmission.
- Cameras are lightweight and easily relocated
- Using ethernet connections, they are able to operate immediately within an Intranet
- Stability is high, due to predictable operation and unchanging firmware
- Cost is lower than a desktop system and camera

Taking all of these elements, NACs allow the development and integration of multi-camera systems for Vision tasks, with low cost of deployment and low cost of expansion. It is on this basis that NACs are used with Myriad in our technical examples, since the costs of outfitting our train demonstration, for example, with three computers, cameras and software would have been prohibitive.

Also of interest, is the emergence of the Wireless Network Attached Camera (WNAC), such as the Linksys WVC11B, which is a standard NAC, except with the addition wireless connectivity. Due to the limited bandwidth of 802.11b WiFi networking (max 11mbps, more typically 2mbps or 5mbps, TV video is generally assumed to require 40mbps uncompressed), the camera produces 320x240 live video, and also uses MPEG-4 video compression to attempt to maximise image size. The quality of the imagery is limited, but with future iterations of the camera, with higher-bandwidth 802.11g/a/n connectivity, the potential exists for TV quality or higher images from networked cameras whose only wiring is that to a power supply. Using a suitable battery pack for power, the option exists to enable such cameras to perform as mobile digital broadcasting TV cameras for networks, useful for roaming diagnosis of equipment and in the case of our problem, high-quality video acquisition for environmental discovery and processing.

F.7 Wireless/Cellular connectivity

Due to the networked nature of Myriad systems, the reader will encounter a number of terms and types of network connection over the course of learning about Myriad. As such, the current forms of connectivity are worthy of note.

F.7.1 Wireless

Wireless connectivity for computing equipment follows two forms:

- Network connectivity

- Device to device connectivity (peer to peer)

These means of connection are important to a networked project such as Myriad, since they allow for physical freedom of components, such as cameras and host computers around a site, without the limitations of wiring. In short, while Myriad offers wide-area freedom of movement, wireless connectivity offers local freedom of movement in a very complementary manner.

Wireless network connectivity is a rapidly developing field of endeavour at this point of writing, and indeed, it was barely begun in mainstream use even at the advent of this project. The ability to connect to systems from any non-wired system and from roaming clients is a powerful and potentially dramatic change in computing use, the ramifications of which will change the direction of computing in much the same way that Internet access changed research, and networking prior to that. Pervasive computing as a concept relies on a base of unwired network connectivity and roaming abilities between sites, which wireless networking provides.

802.11b

802.11b is the ISO designation for wireless ethernet communication over the 2.4GHz radio band. Also referred to as Wi-Fi (the commercial name given to the standard by the Wi-Fi Alliance trade body), 802.11b transmits at up to 11mbps (1.375MB per second) at a range of up to ½ a kilometre, however, data rates and ranges are substantially reduced with radio interference and impedances such as walls. Due to these problems, Wi-Fi networks are generally short-range and enclosed in buildings and built-up areas, known as 'hot-spots'. 802.11b devices are connected to wired networks through Access Points (APs) and act as wired computers would (this is known as Infrastructure mode). Wireless connections are also used to connect short distances between devices without additional networks or components (this is known as Ad-hoc mode) and to connect two networks where wiring is difficult or impossible (this is known as wireless bridging).

It should be noted that by default, 802.11b has no intrinsic data security, and information is broadcast in plaintext, able to be captured by anyone using a suitable radio receiver (such as another 802.11x network controller) and appropriate software (such as AirSnort, the wireless packet logger and analyser, produced as an extension to Snort, the network analyser used by

administrators commonly as an Intrusion Detection System on large networks). Initial 802.11b implementations had an optional encryption method named WEP (Wireless Encryption Protocol), which was unfortunately largely unused and whose encryption keys (either 64bit (40bit effective) or 128bit (104bit effective)) could be obtained using brute force deduction when comparing several captured packets. This process admittedly takes a substantial amount of processing power and time, but it does make it possible to decrypt and spoof WEP packets.

To resolve the weakness of wireless network communication, WPA (Wi-Fi Protected Access) was introduced as an encryption protocol. WPA operates by requiring the implementation of wired Ethernet (802.1x) authentication methods using exchanged keys and dynamic key systems. WPA wraps 802.1x with a customised protocol named Temporal Key Integrity Protocol (TKIP), which provides methods for mixing multiple encryption keys in complex sequence per packet, checking the integrity of packets and dynamic creation and application of new keys.

802.11g

As further areas of the radio spectrum were opened up by governmental deregulation, a new standard for wireless connections was developed, to take advantage of the increased range of frequencies and with less protocol overhead, allowing more data to be transmitted at higher speeds. 802.11g operates in effectively the same manner as 802.11b, except that the transmission rate is 55mbps (6.875MB per second). 'G' devices can automatically sense 'b' networks and reduce speed to act as 'b' devices, and 'g' networks are able to host both 'b' and 'g' devices. The range of the g variant is still limited, due to the power level of the signal, but the data rate is higher at the same distance from the source, compared to 'b'. 802.11g wireless connectivity also exists within the same security limitations of the 802.11b specification, and the implementation of WEP and WPA over this mechanism are identical.

802.11a, 802.11i, 802.11e

Shortly after the creation of 802.11b, another new standard was proposed, 802.11a. The 'a' system boasts higher bandwidth than 'b', but also uses a different region of the spectrum, sometimes interfering with 'b' systems. Due to the incompatibility between 'a' and 'b', 'a' was not accepted by the market, and 'g' systems began to be implemented atop existing 'b'

infrastructure instead. There also exist a number of other 802.11x projects currently in development that intend to produce wireless specifications targeted at specific aspects of networking application, such as 802.11i, which focuses on high degrees of security and 802.11e, which is designed to maximise quality of service and reliability.

802.11n (WiMAX)

With the limited range and bandwidth of previous 802.11 standards and prompted by further deregulation of the wireless radio spectrum, efforts are underway to define a standard for long-range high-bandwidth data wireless connectivity. This standard is presently named 802.11n, and semi-officially referred to as WiMAX. Data transmission rates vary based on distance, and distances of transmission also vary, but the expectations are that the standard will include 150mbps across distances of 3-5miles. The use of such large-scale hotspots has led to the development of the idea of Metropolitan Area Networks (MANs) with wireless access (W-MANs), providing cities with high speed universal data and providing a solution for last-mile networking to outlying areas where cabling is prohibitively expensive.

Resolution of the 802.11n standard is expected to take place in early 2006, with significant market presence and take-up by 2008-10. The promise of pervasive computing at that point, with the addition of data sources, roaming devices and high-quality camera devices, along with Network Attached Cameras provides a fertile ground for the creation of new Machine Vision mobile tasks.

Bluetooth

In contrast to the 802.11x systems, Bluetooth is a peer-to-peer wireless networking technology. Key differences are:

- Bluetooth is lower power, making it useful for battery-operated devices
- Bluetooth's bandwidth is significantly lower (500kbps, or 62.5KBps)
- Range is shorter than that of 802.11x

For these reasons, Bluetooth is most often used in peripherals, such as keyboards, mice, microphones and wireless headsets for cellular telephones. It also finds use for short-distance synchronisation of cellular telephones with computers and palmtop devices, for purposes of maintaining address books, calendars and to-do lists.

F.7.2 Cellular Data Connectivity

In addition to 802.11x and Bluetooth wireless connectivity, there is the option to transmit data over the cellular telephone network, using a variety of data standards, beginning with conventional modem technology and progressing to high-bandwidth technologies, such as 3G data transport.

GSM Data

Using a cellular telephone is much like using a conventional telephone, in that it transmits analogue audio. Using a modem, data can be encoded into tones at one end of a connection and decoded at the other, identically to using a modem across a normal landed telephone line. Modern cellular telephones include modems as standard, using Infra-Red (IrDA) connections or RS232 serial cables to connect to a computing device and pass through data (see fig.121). The bandwidth limit for GSM data calls is limited and varies based on the network and signal quality, but it typically is restricted to 28,800bps and standard bandwidth is 9,600bps.

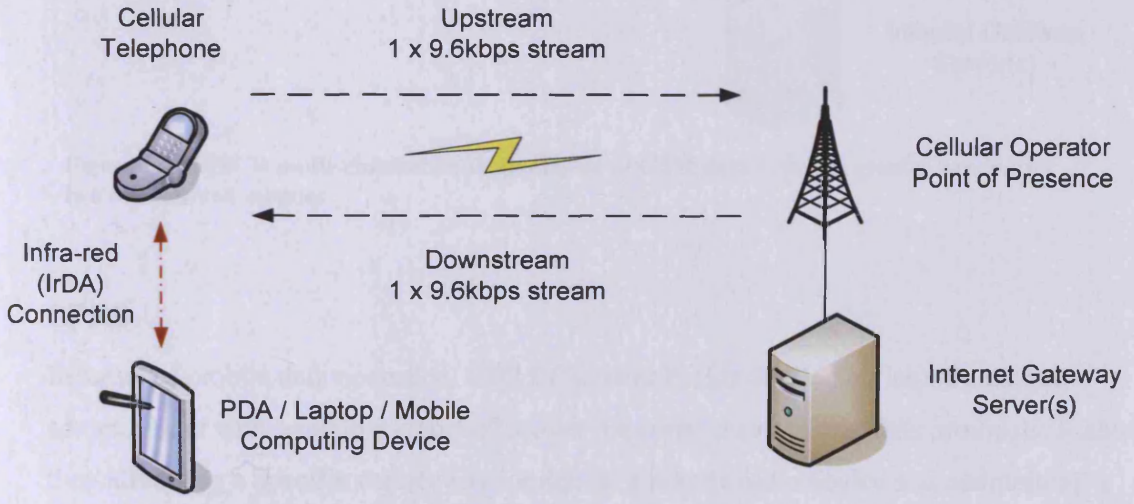


Figure 121 : Conventional single-channel cellular telephone data through dial-up serial connectivity (GSM Data). Modem capabilities provided by cellular telephone to mobile devices (PDAs, etc.) through IrDA connection.

HSCD

High Speed Cellular Data is an evolutionary development of GSM data in many ways. Prior to the advent of cheap broadband Internet access for the general public, ISDN would combine two connections to an Internet Service Provider to form one larger connection. This was

known as a dual-channel (or multiplexed) connection. The same was done with dial-up modems, but there was little interest, as home broadband emerged shortly thereafter.

HSCD simply takes a GSM data connection and multiplexes two or three connections into one; typically two download streams and one upload (see fig.122). Using this method, cellular data connections are able to achieve up to 44kbps data rate, but only if the telephone network supports HSCD.

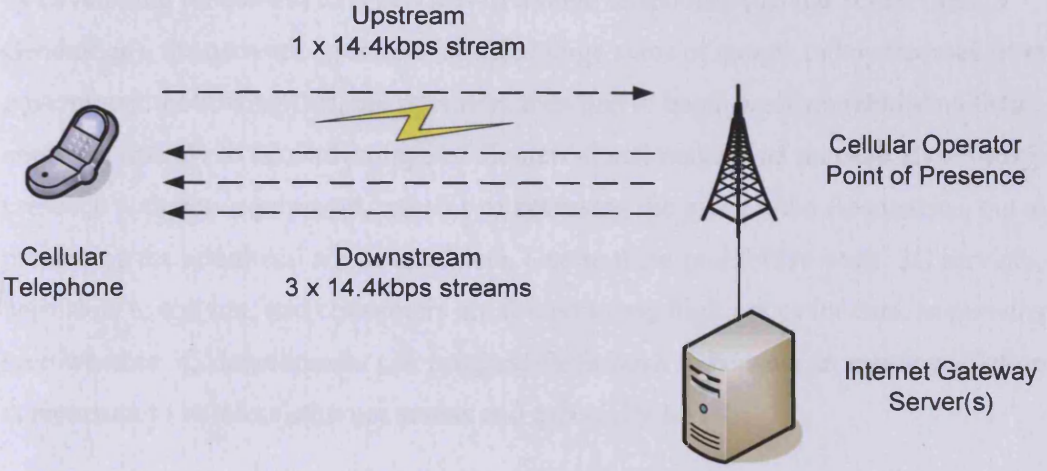


Figure 122 : HSCD multi-channel implementation of GSM data provides greater bandwidth in a multiplexed manner

GPRS

In terms of mobile data operation, GPRS (General Packet Radio Service) is a unique advancement with a number of benefits over the conventional GSM data protocols. Rather than allocating a specific connection for data to a mobile radio device and maintaining a connected line, GPRS operates using switching of multiple users over single connections. This process is achieved using packet switching and interleaving techniques, where devices are only network-present when they are in the process of sending or receiving data, at other times, they do not require bandwidth, and this allows for packets from multiple devices to merged into a data queue. Using switching in this manner, dial-up data connections are not necessary, and data acquisition is more immediate, without a logon preamble and negotiation system. Theoretical bandwidth limitations of GPRS are 171.2kbps, although the quality of radio signal and local cell load can affect performance adversely. With the development of 3G telephone networks, wireless hotspots, WMANs and hybrid Wireless/Cellular telephones,

against high per-megabyte charging by cellular operators, GPRS has unfortunately failed to gain significant market share, although packet-switching techniques are likely to continue in other technologies.

3G

With governmental deregulation of further areas of the radio spectrum that benefited wireless data communications through Wi-Fi and Bluetooth, swathes of the spectrum were sold off to cellular telephone companies. Expecting to be able to take advantage of this increase in range by developing broadband data services to mobile telephones (named 3G services, 3rd Generation), the network operators invested large sums of money to buy licenses from the government. Following this, the operators then had to begin work on rebuilding their networks entirely to take advantage of the new signal ranges and reinstall all of their points of presence with new equipment, capable of accessing the given radio frequencies, but also processing the additional amounts of data. Due to these prohibitive costs, 3G services have been slow to roll out, and consumers are forced to pay high prices for data. In remains to be seen whether 3G development can progress far enough to become an enticing solution in comparison to wireless ethernet access and especially MANs.

F.8 XML

According to Whitehead, Friedman-Hill and Veer [WHI], eXtensble Mark-up Language (XML) is a mark-up scheme, where users are able to design their own mark-up language. XML is descended from SGML (*ISO 8879, Information processing -- Text and office systems -- Standard Generalized Markup Language (SGML) – Ratified in 1986*) and can be considered to be a generic framework for all mark-up languages. HTML is, therefore, one possible outcome of XML (although HTML has a number of language quirks which don't conform to XML standards, leading to a hybrid version, XHTML, which is fully compliant).

As described in Ashbacher's XML teaching guide [ASH], XML documents consist of data within a set of nested formatting tags, of the form:
<tag>data</tag>

Tags must always be closed in this manner (except when they close themselves by placing a '/' at the end of the tag, e.g. <tag/>). Tags also are allowed to have attributes, which are applied in the tag as a name-value pair:

<box colour = "red">data</box>

XML documents are very broad in construction, so much so, in fact, that they may seem to hold no advantage over a custom data format; if both the data and method of marking it up are left to the programmer, how is this different from simply making up a proprietary format? The answer is, that while XML is generic in this way, the structure is still standardised, and with the assistance of conversion tools, may be made into any more rigid structures at a later date, improving cross-format cooperation. While XML document formation may be custom to the task, however, they also must include a second document, known as a Schema, which provides an outline of the document structure (available nodes, attributes, how nodes may be nested). Using a schema with an XML document provides a means to decode the structure and extract relevant sections, even in an unrelated system. Most of all, XML documents are always human-readable, allowing for good debugging and long-term safe storage of data, even though the way the document is rendered may change.

Data stored as XML is also able to be parsed with the assistance of a third document, known as a style sheet, written in a language named XSLT (eXtensible Stylesheet Language Transformations), which defines how a document format might be converted into another format, such as HTML. In this way, two systems, which both input and output XML may talk to each other provided a single translation document is created, which describes how to convert one format to the other (see fig.123).

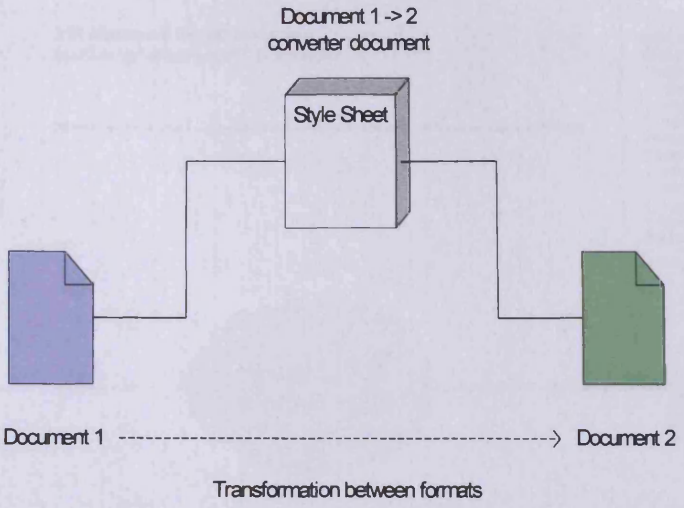


Figure 123 : Use of an XSLT style sheet to translate data formats between systems

This is the ideal of XML development in the long term; that all systems will use XML as a transmission format, and style sheets will then be created to enable all systems to understand each other.

XML, however, has a few problems. Firstly, documents must be parsed in a hierarchical manner, requiring heavily recursive parsing systems, or a good generic parser and an object-oriented language capable of easy response to node discovery events. Secondly, each currently-available parser operates differently, and often does not use the full capabilities of XML, leaving it up to the programmer to finalise the conversion (most especially in the case of style sheets, where many parsers currently ignore them). Examples of the different parsers currently available and their usage are described by Whitehead, Friedman-Hill and Veer [WHI]. Finally, using XML adds data overheads to files (see fig.124), which, with small amounts of atomised data, can lead to a great deal of wasted space with tag data outweighing the valuable data by far. This therefore leads XML systems to prefer or even require high-bandwidth communications with each other.

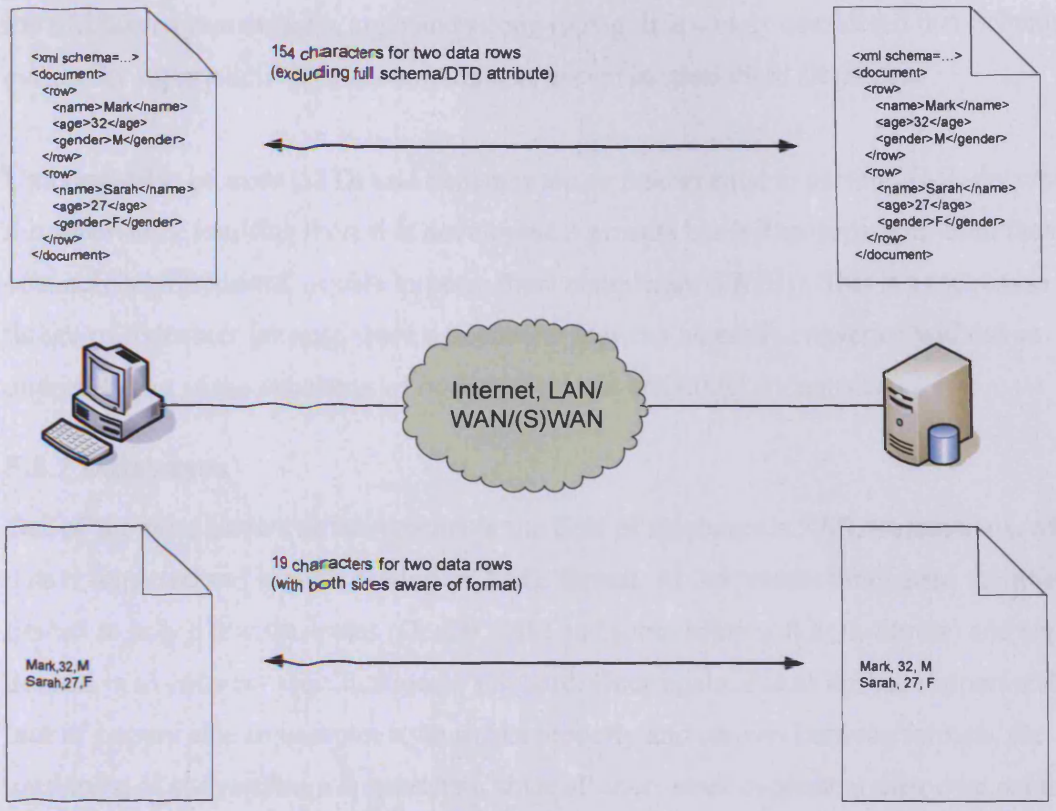


Figure 124 : Comparison of meta-data overhead for simple data tasks from XML and pure data formats

F.8.1 DTDs &Schemas

With the creation of XML documents, there needs to be some form of template descriptor to allow the reader, whether it be a program or a human being, to understand the structure of the XML document. Initially, this requirement was filled by a separate document, known as a Document Type Definition (DTD), (discussed by Whitehead et al. [WHI]), which described the tags a document had, how they could be nested and what data types and attributes were allowed per tag. While this was useful, the construction of DTDs was complex, with a less human-readable syntax than the XML for which one of the key selling points was the human-readability. Not only was the creation difficult, but DTDs omitted a number of essential features, such as support for namespaces, modularity and value constraints, which caused problems in data validation.

As a response to the limitations of DTDs, XML Schemas were proposed. Essentially similar to DTDs as a separate document that would illustrate the structure of the XML document, a Schema was a descriptor written in XML itself (with a specified Schema and DTD which is used globally by all Schema developers, which describes the format of a Schema itself) with

the addition of namespaces, logic and strong typing. It is widely considered that Schemas will eventually supersede DTDs in common use, except in specialised situations.

Unfortunately, because DTDs and Schemas are so fundamental to parsing XML documents, the uncertainty resulting from this development process has left most parsers, even those commercially produced, unable to parse them completely ([WHI]). This is a root cause of the failure of stylesheet parsing, since a document may not be easily converted without an understanding of the structures of both the original and target documents.

F.8.2 Databases

One of the most current developments in the field of databases is XML transactions, where data is imported and exported using an XML format. At the present time, these facilities are limited to only a few databases (Oracle, DB2 and soon, Microsoft SQL Server) and are not defined in an industry specification or standard. Once again, due to limited support and the lack of parsers able to interpret style sheets properly and convert between formats, the usefulness of such systems is restricted, since all users must implement their own parsing operations per format.

F.9 Web

The World Wide Web forms the backbone of Myriad systems, with all communications taking place over HTTP connections, and integration with existing Internet or Intranet systems.

F.9.1 HTTP

Hyper Text Transfer Protocol is the protocol underlying modern web communication. It consists of a send or receive header and subsequent data. The header describes the server and client systems, the type of file being transmitted, the request string, the protocol version and date/time information.

The sequence of HTTP usage is as follows:

1. Connection from client to server is established.
2. Request is made client to server.
3. Server responds with return header and data.

4. Client accepts data and disconnects.

As can be seen from this illustration, HTTP is an asynchronous, client-initiated protocol, based solely around the supply of documents as requested. It is not possible for a server to initiate communication to a client. This situation is most ably illustrated in Sir Tim Berners-Lee's RFC 1945, in characterising this as a 'Request-Response' paradigm. In the case of Myriad, where HTTP is used as the key communications mechanism, this therefore requires a client to repeatedly request updates, or having both client and server be HTTP-listening servers, in which case, data can be regularly transmitted the client in the form of requesting documents from the server and encoding data in the request (see fig.125).

Modern HTTP is technically categorised into implementations of two standards, 1.0 and 1.1. The 1.0 standard was widely introduced in 1990, but was only formally addressed at length with the production of the Request For Comments (RFC) document (1945) produced by Tim Berners-Lee, R. Fielding and H Frystyk in 1996 – "Hypertext Transfer Protocol – HTTP/1.0", with expression that HTTP needed revision into a more formal standard. This lead to the creation of HTTP/1.1, (RFC 2616) by Fielding et al., addressing issues of keeping connections alive, caching policies, internationalisation support, authentication and security, MIME encoding of data and compression methods.

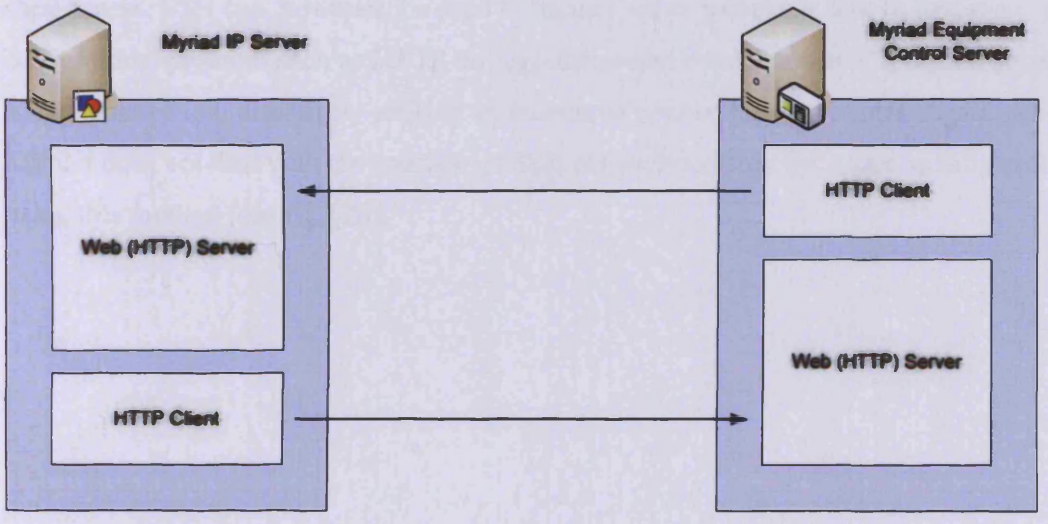


Figure 125 : Using a mutual serving pair to provide two-way communication mechanisms with HTTP

F.9.2 HTTPS

While HTTP is a useful protocol and has benefits in the fact that data is sent un-encoded, allowing plain text files to be human readable, it does not have facilities for encrypting data to

make transmissions secure. HTTPS is an extension to HTTP, using a either the SSL (Secure Socket Layer) or TLS (Transport Layer Security) encryption mechanisms. In these cases, public and private keys are exchanged in the header of requests; these are acquired through the purchase of security certificates, with trust authorities, to enable encryption of data from client to server and vice versa. This design allows data to be encoded at either end of an insecure chain of Internet hosts, who are charged with relaying the data (and who might engage in caching and monitoring, a traditional man-in-the-middle attack). Myriad does not seek to implement full SSL or TLS capabilities within the server, due to complexity and the significant risks posed by producing a naïve implementation with exploits or security flaws. Instead, using commodity web server, a gateway application using standard libraries such as Apache's mod_SSL module may be created which can serve as a secure means of access to Myriad servers.

SSL encryption is typically 40bit or 128bit, depending on the implementation.

F.9.3 SSH

Secure SHell implementations may be thought of as the Telnet equivalent of SSL, effectively encrypting traffic between a client and a host using public and private encryption keys. SSH allows a user to connect to a server securely, normally for administrative functions, such as shell access. SSH can, however, be used to 'tunnel' other protocols; that is, to run a conventional protocol such as HTTP through the secure connection as if it were a normal socket connection, effectively making an unsecured connection also secure. Again, whilst Myriad does not deal with the creation of SSH connections directly, it can be fully secured using this method (see fig.126).

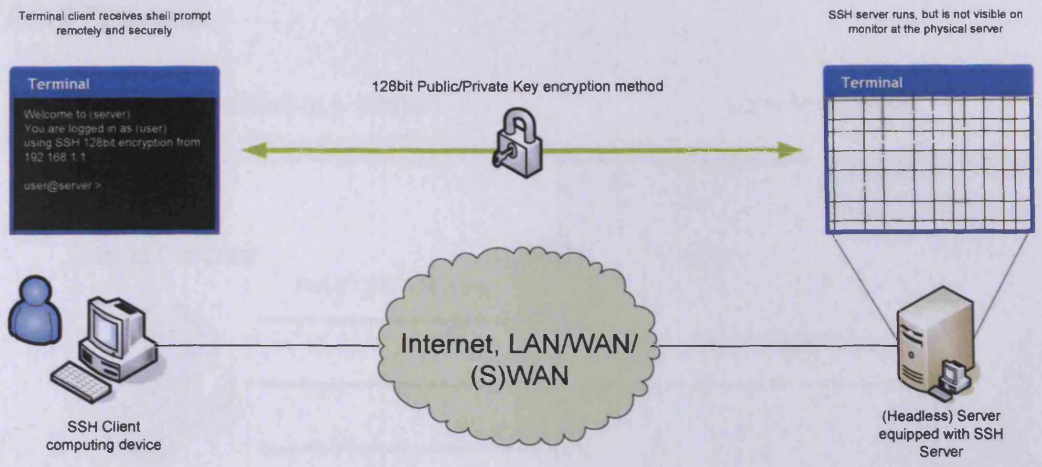


Figure 126 : SSH using encrypted traffic to 'remote' a shell session from a server to a connected client with certificate-based encryption techniques

F.9.4 VPNs & S/WANs

Virtual Private Networks are very similar to SSH connections, in fact, almost identical. A VPN connection is the establishment of a connection between a client and a server (known as an 'endpoint'), which is fully encrypted for all protocols, but runs over an unencrypted network, such as the Internet. There is no application or user-level knowledge of tunnelling required; the computer simply transmits data, which is encrypted and tunnelled by the networking stack at the time of transmission. VPNs are most commonly used as a way for users to remotely connection to a Local Area Network, normally at the workplace, in a secure manner. Using VPNs in this way, the remote computer acts as if it were connected directly to the LAN, with a LAN IP address, and able to communicate with all local systems as normal.

VPN techniques are also used for a second class of systems, called S/WANs, Secure Wide Area Networks, where there are a number of LANs needing to communicate with each other, but the only medium is an unsecured network such as the Internet. In these cases, secure tunnelled connections are established between all of the LANs, and they are united into one large, long-distance network, with secure communications across all sites.

F.9.5 Firewalls

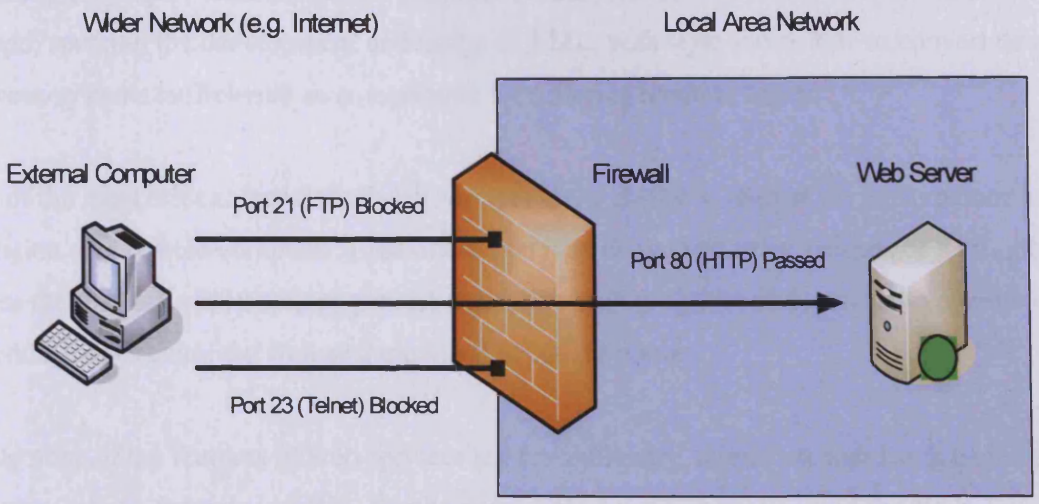


Figure 127 : Simple port-blocking method of firewalling for a server (either utilising a software firewall or hardware device)

Firewalls are a key feature of networks connected to other networks, such as the Internet. Essentially, all network traffic must pass between two ports (which form a connection), one at the client and one at the server. Ports are numbered and with established protocols (such as FTP (21) and HTTP (80)), are fixed. In order to ensure that certain services are only available within a network and not to the outside world, firewalls simply block certain port numbers from being able to connect, thus preventing access (see fig.127).

Firewalls are a useful tool to secure the boundaries of a network, and as will be shown later, can be used in combination with gateway devices, programs and scripts to regulate access to resources in simple or complex manners.

F.10 Web Services

With ongoing development of the Internet as a tool within business and institutional organisations, the purposes of new projects have moved from informational intentions toward sales and customer service, and beyond that towards a medium for communication and relationship between organisations. These systems are commonly referred to as B2B (Business to Business) systems. Using the Internet as a communication medium, data, product supply management and invoicing systems are able to transact with each other.

Due to the fact that most business software systems operate at the present time using differing, proprietary data formats, a common format with conversionary abilities was needed; spurring the development and usage of XML, with style sheets able to convert data between systems sufficiently to compensate for differing methodologies.

One of the most critical features of web services is the ability to request the performance of an operation on a remote computer. Essentially this may be thought of as a means of RPC, albeit across the Internet and in a more generic form, allowing programs and systems to operate as general services using the web as a medium; hence the name.

While none of the features of web services are revolutionary, merely standardising generic communications formats and RPC mechanisms, they provide a background for the communication between general systems across organisations without excessive amounts of custom glue programming.

F.10.1 SOAP

The Simple Object Access Protocol (SOAP) is notionally the messaging protocol of the Web Service concept. Much like WAP, SOAP is an XML document schema, outlining the way XML documents should be formatted and objects serialised within the SOAP document. SOAP documents (often referred to as SOAP 'packets') are transmitted between hosts, containing either data or RPC commands. Basic data structures may be serialised, from integers, floats and doubles to strings and arrays. Complex objects are beyond the scope of SOAP's design, but due to Web Service systems being so disparate, highly complex objects are unlikely to be understood by the recipient, so simple data types suffice for most purposes.

F.10.2 Advantages

Web services are most advantageous in the short term on the basis of their ability to provide services to organisations which facilitate business activity, either by providing data and time specific responses, or by providing systems with simple interaction, yet encapsulating complex data and processes which may not be desirable to expose.

In the longer term, however, web services provide a mechanism for developing long-distance systems and operating programs across the medium of the Internet in a generic manner, blurring the locational and organisational boundaries of current computing projects. Using

virtual file system technologies (KIO Slaves, Gnome VFS), network transparency can emerge, unifying system and remote data and programs, transparently to the user and other programs.

F.10.3 Limitations

Despite their technical advantages, web services have a number of key drawbacks:

- XML is heavy in meta-data/data ratios for highly atomised data
- There is no facility for conveying error conditions, nor for detecting or compensating for network failures
- Each service acts in a unique manner, with no common subset of commands
- Web services are not, by default, scriptable. This forces the controller to organise every action, or the service to provide commands to suit every eventuality. Highly complex, responsive operations are difficult and suffer from network latencies and failures.

F.11 Scripting Languages

Being able to write small, interpreted scripts of commands to control applications has been a consistent feature of computing facilities throughout their history. More recently, with the advent of the Internet, the ability to organise a variety of applications, process input and collate results in a manner which is efficient and simple to program has refocused programmers on scripting capabilities rather than monolithic, compiled applications.

F.11.1 Perl

Originally designed to be a scripting language for automating operating system tasks and reporting data (eg. the results of logging systems), Perl's features and commands emphasise string processing functionality. Growing in parallel with the Internet, Perl found a natural area of expertise in managing text-based news systems and was one of the first languages to be used in conjunction with early web server technologies to create dynamic web pages; for the world wide web, at a fundamental level, is a textual system. While languages more centred on database access, such as PHP and ASP have taken over the role of dynamically creating web pages for light tasks, Perl is still used for the most complex of web tasks, including those which use other operating system programs and functions, and involve complex string processing, for which Perl is best suited.

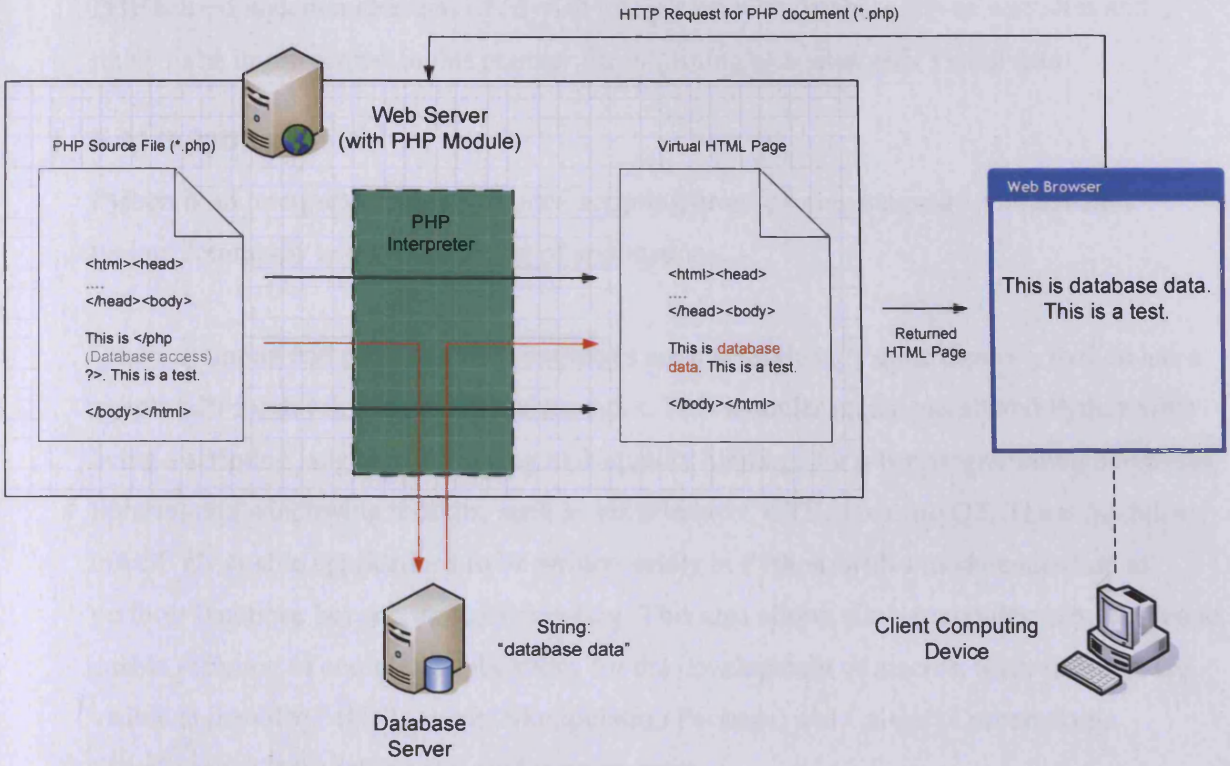


Figure 128 : Web serving dynamic data using PHP and database access, evaluating hybrid HTML-PHP script files

F.11.2 PHP

After the initial generation of dynamically created web sites, organisations and individuals found themselves producing more complex sites, which relied on working with databases, such as E-commerce systems. For this, a second generation of database-centric scripting languages developed, led by PHP and ASP.

PHP began life as PHP/FI (Personal Home Page/ Form Interpreter) a set of Perl scripts by Rasmus Lerdorf, produced in 1995. Later rewritten in C, it developed a number of new features, including database interaction. Rewritten once again by Andi Gutmans and Zeev Suraski in 1997, it was renamed PHP (with the recursive acronym: PHP: Hypertext Pre-processor), it was released as PHP 3.0.

Although it boasts more limited string manipulation than Perl, PHP code is embedded directly into web pages (see fig.128), alleviating the need for complex template systems and providing the means for implementing database queries in pre-formatted positions in web pages.

PHP is used in demonstrations of Myriad integration with database driven web sites and is likely to be implemented in this manner for informing web sites with Vision data.

F.11.3 Python

Python is an interpreted object-oriented scripting/programming language with dynamic typing, commonly used for scripting of applications.

Where scripting and programming languages are conventionally quite discrete, Python has a simple API for developing extension modules. This modular nature has altered Python from being a scripting language, to having high-quality bindings for other programming languages, libraries and windowing toolkits, such as wxWindows, GTK, Java and QT. These modules effectively enable applications to be written solely in Python, with a module attached to perform functions beyond those of scripting. This also allows simple modules to be written to enable scripting of complex applications for the development of macros. Such facilities are visible in the GIMP (GNU Image Manipulation Package) and Caligari Corporation's 'TrueSpace' 3D modelling and rendering program.

While demonstrations are not presently written for Myriad using Python, it is worthy of note that it can easily be used as a scripting bridge with other applications and programming languages, enhancing them with scripting and Myriad connectivity, even if they are not natively capable of such interaction.

Further, should MCS not be suitable for a formal Myriad specification for wider use and long-term development, Python is a strong candidate for replacement as the central means of scripting Myriad components.

F.11.4 ASP

Active Server Pages may be seen, first and foremost as the counterpart of PHP and Perl for Microsoft's IIS platform. Taking middle ground between the two, syntactically and functionally similar to PHP in many ways, ASP may also be extended using binary components (which often operate with Microsoft's Visual Basic language as well). ASP does, however, suffer from security issues due to the operation of binaries on systems where they have greater permissions and access than is desirable. This leads ASP to becoming potentially a source for full super-user level scripting of a system, if not attended to by a cautious

administrator and careful programmers. ASP does, however, show the full range of operations required to construct a gateway interface application for Myriad purposes, or to integrate Myriad components into larger web-based systems.

ASP has recently become far more attractive for web development, thanks to the revisions which have taken place through Microsoft's .NET programming initiative. ASP .NET source code is no longer purely interpreted, but compiled Just-In-Time to bytecode which may be executed on the .NET Common Language Runtime (CLR), which operates in the same manner as the Java Virtual Machine. While this may lead to slightly slower execution as opposed to older ASP implementation, it does allow ASP to utilise Windows .NET components as native ASP libraries, enabling the use of Windows core database libraries for example, rather than the re-implemented ASP libraries, providing for better integration and operating system feature inheritance. In addition, .NET allows all source code produced by .NET languages to be compiled into the same bytecode (named Common Interface Language (CIL) or Microsoft Intermediary Language (MSIL)), enabling compiled components to be used together, regardless of the language from which they originated; hence, an ASP script might benefit from directly using an image processing library produced in Visual C++, as if it were ASP, without the programmer having to be concerned with managing language bindings and cross-compilation.

F.11.5 VBScript

Visual Basic (VB) Script is a scripting language designed for control of Windows applications and components using a VB-like lightweight scripting language. Typically used to script office applications and business processes, it was extended by IIS to allow it to organise collections of web components to produce dynamic output. Due to security considerations in addition to the limited functionality, VBScript is rarely used, with Windows developers preferring to use either VB or Visual C++ directly, or to implement web applications using ASP, which is far more analogous to PHP and more intuitive than the application of a full desktop object-oriented scripting language for web programming, especially in the area of textual manipulation.

F.12 Intelligent Control / Reasoning

F.12.1 Prolog

Prolog is a programing language based on predicate logic, performing functions as a result of reasoning against a set of language statements, assertions and set processing directives. Along with languages such as Lisp and Haskell, Prolog is considered part of the class of 'functional' programming languages. As a result, Prolog can be thought of as a language for acting upon databases of facts, assertions and assumptions based around those facts to deduce conclusions and courses of action, rather than discrete logic processing of normal object oriented or procedural languages. As a result of this, Prolog statements can be used to act across sets of data, using these to inform operations, resolving situations where conventional discrete logic would fail due to ambiguity. In addition, predicate logic can be used to emulate discrete logic, through forcing a truth situation in a given instance and acting upon that truth in a standard manner.

Prolog, therefore, is a useful tool in systems where ambiguity routinely exists. In many cases of Machine Vision systems, combinations of lighting, equipment and environmental control allow for a high degree of constraint on the task, reducing processing to fixed objects (which the system can be sure are always in the same position when the image is captured or physical action is taken) and fixed conditions. Beyond these problems, however, there exists a class of problems where such assumptions cannot be relied upon, and conditions and objects may vary. Analysing objects and particularly people in outdoor and active environments (such as public buildings) are good examples of this type of task. One of the most extreme examples, however, and one that can typify this set of problems, is that of mobile analysis and prototyping of the kind found when taking the previous example of configuring Vision systems remotely. In this case, the object being inspected varies, as does the lighting, the position of the camera (because it is being moved by a local operator), the rotation of the object and the general quality of the image, which may be impaired by bad environmental conditions, such as dust. In these situations, it is imperative that the system being used to analyse the situation and define a course of action, be able to act upon the complete set of potential inputs (such as all rotations of the object being inspected), make assumptions and logical progressions to reasonable conclusions. It is for situations like these that Machine Vision systems can most appreciate the addition of Prolog elements and more dynamic

situations can apply Vision techniques in a manageable manner to what is a highly complex and ambiguous problem.

WinProlog/MacProlog

MacProlog and WinProlog are commercial implementations of Edinburgh standard Prolog interpreters produced by LPI Associates. MacProlog is the Apple Macintosh edition of the software, and has been used extensively by Prof. B. Batchelor for the creation of PIP, lending it the initial 'P' in the acronym (Prolog Image Processor). WinProlog is the equivalent environment for Windows.

Win/MacProlog applications may be either interpreted or compiled for use with conventional applications, forming libraries or through the use of APIs, becoming integrated as program sub-sections. Further to this, LPI also produce a Prolog web server plug-in for Windows, ProWeb, which is a command interpreter that operates in much the same way that Perl and PHP interpretation operate, as an extension to the IIS web server. The limitations of this product, however, are two-fold. Firstly, it only operates under Windows, which is a disadvantage for embedded systems and non-Windows platforms in the increasingly heterogeneous environment that networks are becoming. Secondly, the plug-in is designed specifically to work with Microsoft Internet Information Server, which due to security issues and limitations on dynamic content in comparison to the Apache web server, has resulted in many network administrators refusing to use IIS. This situation is the case within Cardiff University itself, for example, where the network administrators refuse to allow IIS servers to operate on their network, without exception. While the Prolog plug-in does notionally support use with Apache under Windows, the Apache plug-in architecture has changed with version 2, which leaves Prolog incompatible, and attempts by multiple parties to operate the Prolog interpreter with Apache 1.3 have met with failure.

SWI Prolog

SWI Prolog is an Open-Source project contributed to by academics, educational institutions and individuals; developing a Prolog interpreter based around the Edinburgh standard which operates as a text-based interactive processor and also is capable of running in an automated fashion using command line execution and addition execution switches. This mode of operation makes SWI Prolog highly useful for use with Myriad systems, since a scripting language is able to create a file of Prolog commands, execute SWI Prolog, consulting this

command file and manage the returned results; as a result, a gateway web script may be written around the SWI Prolog program to effectively make it web-capable and available as a Myriad component.

SWI Prolog is also capable of being extended with a number of different modules for the creation of user interfaces, bindings to other languages, compilation into binary programs and direct facilities to access web protocols and control devices.

The most compelling reasons for the use of SWI Prolog are, however, threefold. Firstly, the interpreter is free, fully available for use without licensing charge, enabling it to be added to a developing or existing system without substantial increase in overall system cost. Secondly, the source for the project is freely available, as an Open Source project under the GPL license, making customised extensions such as control of non-standard equipment or interaction with unique resources easy to develop, once again, without addition source code viewing fee. Finally, the interpreter is included with the majority of current Linux distributions (such as SuSE, Mandrake and Debian), with binary packages available for download and installation for most other systems, and the source code offered for self-compilation should a specific system binary not be available. SWI Prolog is also able to be compiled without alteration for all major Unix varieties, including Solaris, AIX and MacOSX. This ease of use is a tangible benefit for deployment of a Prolog system within a Myriad solution, and as such, is the recommended means of augmenting such systems with Prolog. As a result, this implementation will be used as the preferred element in subsequent Prolog examples.

CKI Prolog

CKI Prolog is the personal product of Sieuwert Van Otterloo of the University of Utrecht, which is an Edinburgh interpreter written in Java for inclusion in Java programs. Operation of CKI Prolog, is through direct inclusion of the Prolog interpreter classes within an application, and as such, there is a tight dependency to redistribute the interpreter files with the final program. The use of CKI Prolog was entertained for a time, with Myriad components, most specifically to produce the MCS interpreter on the prototype servers, developing upon the work done with the use of CKI Prolog with CIP to add basic scripting to that GUI application. After consideration, however, it was decided that due to the speed of execution of Prolog interpretation being so slow, the resource overhead that the implementation of a Prolog interpreter in Java imposes and the difficulty that would be created should a more

performance and mission-critical Myriad server be attempted to be produced in another programming language such as C++, Prolog should form a distinct Myriad component. Including CKI Prolog in the core Myriad server prototypes would force a programmer producing a non-Java Myriad component to create their own Prolog interpreter, include a differing interpreter library and resolve the differences between implementations, or create a direct MCS language interpreter.

Unfortunately, due to the Java requirement, a CKI Prolog Myriad component would require significant computing resources, unable to initially process files of scripts unless wrapper objects were created and be slow to start up and operate in comparison to a fully-featured compiled interpreter such as the SWI Prolog program. As a result, CKI Prolog is not used for the sample Myriad Prolog server. Should anyone wish to use CKI Prolog, however, they would need to write command line input, output and file management objects that would manage data for CKI Prolog. With these wrappers in place, CKI Prolog could be used with a gateway script and a web server in much the same way as SWI Prolog is. Alternatively, the use of CKI Prolog could be further extended to include a basic HTTP server as an extension in much the same way as a Myriad server operates, which is distinctly more complex to produce than to use a general web server, a gateway script and a command line Prolog interpreter.

Other Prolog Implementations

There are a number of Prolog implementations that are specific as libraries to unique languages, such as Visual Basic libraries, which have not been covered here. Sufficed to say that the majority of modern Prolog interpreters are based around the Edinburgh standard, which has been adopted as the ISO standard for Prolog. There are a number of differing dialects, however, especially when it comes to the inclusion of additional elements, such as database connectivity and networking modules. In these cases, standards do not exist, and the users of such libraries must refer to their interpreter as to the capabilities available, and how those may be used. Most interpreters, however, will share both interactive and scripted operating modes, the interface for which will either be the command line or an interface object within the programming language to which it is to be bound.

One relatively ubiquitous implementation is the GNU Prolog interpreter which is produced as a project from the GNU Foundation. As most programmers will be aware, GNU is a recursive

acronym for GNU is Not Unix, which emerged in the 1970s to provide free use and free source versions of proprietary low-level user-land Unix utilities, such as cat, cp, ls and many other primary commands. In time, these implementations, through their free distribution and adoption licences became used in almost all Unix distributions, and form a core part of Linux, BSD, Solaris, AIX, HP-UX and other modern operating environments. Having gained certain renown, the GNU moved on to providing Open Source implementations of international standards, such as the Prolog interpreter and the GCC compiler, based on the ISO C and C++ specifications. GNU Prolog operates in the same manner as SWI Prolog (both interactively and scripted, using similar flags and syntax), albeit with a presently less diverse repertoire of extension modules. As a consequence, GNU Prolog's HTTP capabilities were not well expressed in the documentation, and SWI Prolog, also a free implementation was chosen for Myriad use. Other than that module, however, the two may be used interchangeably. GNU Prolog is also available for all modern UNIX-compliant systems and the Windows architecture.

F.13 GRID

Often referred to as 'Internet 2' (although that moniker is generally taken to refer to the ill-fated high speed network development that preceded and forms the basis of GRID computing), the GRID is a high-speed, multi-connected network between large institutions (mostly academic at the present time), of sets of high-performance computing nodes. The key attribute of the GRID, is the ability of systems to distribute their workload around different GRID nodes to solve complex tasks. In many ways, the GRID may be thought of as a distributed High Performance Computing (HPC) cluster.

SUN Microsystems currently provides a GRID computing module for use with Java programs, named JXTA.

In order to avoid addressing the GRID as simply another high-speed network, careful consideration of how to optimise tasks for the node-based topology must be undertaken, and software produced that operates in a substantially different manner than conventional Internet applications. As such, GRID development is beyond the remit of this thesis, and will not be discussed further, although the author does envision many machine vision opportunities for GRID use, such as natural-language image content searching and indexing.

Framework for Distributed and Networked Vision Systems

L. T. Chatburn, G. R. Jackson, M. W. Daley & B. G. Batchelor
School of Computer Science,
Cardiff University, Cardiff, Wales, CF24 3XF , U.K.

Email: *bruce.batchelor@cs.cf.ac.uk*

ABSTRACT

Hitherto, machine vision systems have been designed as integrated units, in which one or more cameras, a dedicated image processing engine and control/image display console are connected together and hence are all located in close physical proximity to one another. Myriad is a web-based framework that allows these components to be separated from one another and enables complex hierarchies and meshes of inter-communicating cameras and image processing elements to be built. Any component of a given type can be interchanged with any component of the same type, wherever they are located. Myriad networks can employ single- or multiple-stream processing. Several users may be active at a given moment and an unlimited number of users can passively observe what another one is doing, provided that the appropriate permissions have been granted. Myriad networks can encompass intelligent programmed controllers, and/or human controllers, working through a standard web browser. A sub-system can record its observations in a standard database, which can then be interrogated in the normal manner.

Keywords: *Machine Vision, Network, Internet, Distributed computing, Remote control.*

1. MACHINE VISION IN MANUFACTURING INDUSTRY

There are several broad categories of functions performed by Machine Vision systems in manufacturing industry:

(a) Examining and/or measuring a product, whether this be a simple component, or a complex assembly. The function to be performed might involve inspecting, measuring, counting grading, monitoring or controlling a machine.

(b) Management control. Functionally, the requirements here are broadly similar to those needed for *(a)*, although high speed operation is probably not needed

(c) Prototyping. This involves experimental and quantitative analysis of new and previously unseen applications. Prototyping may be conducted "off line" (in the company's R&D laboratory, a nearby university, a consultant's office, or the premises of a vision systems supplier), or *in situ* (i.e. by inspecting a small proportion of the factory produce)

(d) Education and training, most notably for people involved in the design and use of systems covered under *(a)* or *(b)*.

Of these, *(b)* and *(d)* are less important commercially and place lower demands on the system designer. In what follows, we shall refer to a *target system* (i.e. a high-speed machine that is capable of performing 100% inspection on the factory floor) and a *prototyping system* (a general purpose analytical tool, used to study new and previously unseen applications and thereby assist in the design of target systems). For the purposes of this article, the requirements for systems that are to be used for both management control and training/education can be satisfied by a prototyping system. (Since many management control functions can be performed by a machine of modest speed, we are able to implement this group of tasks using the *Myriad* framework, described below.)

1.1 Target Systems

The most common use of Machine Vision in industry is to assist in quality control. This may require visual monitoring of the product, or the manufacturing process. In order to design a machine to perform either function, a range of image processing techniques must be explored, after first taking great care to design the best possible lighting-viewing configuration. (The lighting design process also involves the use of a prototyping system.) The lighting-viewing conditions must be optimised, in order to highlight the relevant features, by creating good image contrast. A target vision system may either provide feedback to other equipment, such as that used in machining, or to a human operator.

Many production-line inspection tasks simply require that defective products be identified and removed from the line automatically. Other applications require quantitative measurement of position, dimensions, angles, area, volume, etc. In some cases, this information provides feed-back to a robotic device which moves the object to a standard position, or applies further machining operations, such as cutting, welding, painting, turning, etc. The essential feature of a target system is that is must operate without human intervention. It must be fast, accurate, reliable, robust and fail-safe. It must also be easy to test and calibrate.

1.2 Prototyping Systems

Target systems are relatively static, being aimed at solving problems with a known and fixed level of unpredictability. They do not require human intervention, except during calibration, testing and reprogramming to accommodate software updates. On the other hand, prototyping systems are highly interactive. Each processing step may be ordered directly by the human operator, although it is important that he/she be able to define and employ "high level" macros, implementing procedures that have previously been found to be useful. Compared to target systems, prototyping vision systems may be quite slow, provided that the effectiveness of the interaction does not suffer. They must have a comprehensive command repertoire that is capable of being extended indefinitely, either by adding macros ("high level" operations), or by writing code to implement "low level" operators. Interactive image processing has been studied by one of us (B.G.B.) since the mid-1970s and has a well-proven record as a tool for assisting in the design of vision systems. [BAT01] Many interactive prototyping systems rely on command-line control, although some use pull-down menus [PHO] and others a visual programming interface. [NEA] In the light of their experience, the authors favour a command-line system that is operated under the control of a Prolog program. [BGB] The advantage of this is that writing macros is trivial. Synonyms and default parameters can also be defined in a straightforward way. It is possible to reconfigure a Prolog-based system for a novice user, who speaks only one (foreign) language in a matter of minutes! Prolog has a range of features making it suitable for use in Artificial Intelligence applications, including: recursion, general non-declared data types, data matching, instantiation and back-tracking.

2. LIMITATIONS OF THE CONVENTIONAL APPROACH

The design of the image-acquisition sub-system (i.e. lighting and optics) is a specialised topic and will not be discussed in detail here. Let it suffice to say that a range of lighting methods, sufficient for many experimental investigations involving "fist sized" parts (the most common size of industrial artifact), can be built around a relatively small collection of illumination sources, including fibre optic devices and arrays of LEDs. An interactive image processing system usually consists of a video camera, interfaced to a desktop computer, via a proprietary frame-grabber card. (Nowadays, the last mentioned may be an integral part of the computer.) These units are interconnected to form an integrated unit that must be physically located close to the scene being viewed. This arrangement has certain disadvantages:

(i) The incremental cost of adding a new camera is high. (Usually, a new computer is also needed to process the video data.)

(ii) The system is bulky and heavy.

(iii) The camera-computer cable limits the separation between them. On occasions, this may cause the computer to be placed in dangerous situations, for example where there is a lot of dust in the atmosphere. By placing a computer very close to a production facility. An integrated system can also create obstacles that impede access and/or movement. The alternative is to install special-purpose cabling, which is prone to picking up electrical noise and is both expensive and fragile.

(iv) The user is required to sit close to the scene being viewed. This may expose him/her to noise, dirt and the unpleasant conditions existing in a factory.

(v) The user must travel to whatever location is required for the camera. This may mean that a highly skilled vision engineer is working inefficiently, visiting a factory for several days, when only short periods of productive activity are possible

(vi) There is only one user, who cannot easily share his/her experience with other people in a natural way. A vision engineer cannot, for example, "talk through" a problem interactively with a group of people, unless they are close to the system and its user.

A number of commercial systems now provide remote operability to vision systems. (For example LabView [LAB]) These systems, however, are somewhat limited in being able to provide local site remote operation, or limited wide area control. On the other hand, *Myriad*, the system designed and built by the authors and described below, was designed

from the outset as a framework for building networks of Machine Vision sub-systems that may span large distances. Myriad permits computer control of lighting, camera, image processing engine, database for storage of results, overall system control, display of results and simple electro-mechanical actuators to be physically separated from each other. Myriad provides vision system components like these as service modules that are made available via a Local Area Network (LAN), a (secure) Wide Area Network (WAN), or fully exposed, to the Internet. These modules form nodes on a mesh that can have many units of each kind, co-operating together in a fully integrated way, or in cliques. Myriad is dynamic; new service modules can be added / removed to the network while it is running.

3. EMERGING TECHNOLOGIES

There are several new technological developments that have had a significant impact in the development of the Myriad software.

3.1 Java

Java was first announced in 1995 but it is really only in the past two years that it has approached its design objective of being a stable cross-platform language. Indeed, work is continuing to fine-tune it. Java is an object-oriented language, derived from C++, with many of the complex features removed. It is built around the *Java Foundation Classes (JFC)*, which cover a range of programming requirements, including user interfacing, drawing, string manipulation and network access. [JAV] Java source code is compiled to create *byte-code*, which is the machine language for a *Java Virtual Machine (JVM)*. The JVM is broadly similar in concept to a cross-platform emulator. JVMs with exactly the same input specification are available for a wide range of standard computing platforms, including those running Windows, OS X, Linux and Unix operating systems. The byte-code defines operations for the JVM, not for its host computer. Thus, compiled Java programs will run, without change, on any of the platforms just mentioned. However, achieving cross-platform compatability comes at a price: the JVM introduces a significant overhead in terms of memory requirements and processing speed. Hence, Java does not run with comparable speed to compiled code (e.g. C, C++).

3.2 QT Toolkit

QT is a toolkit for the C++ language, produced by Trolltech AS [QT]. It provides the equivalent functions for C++ as the *Java Foundation Classes (JFC)* do for that language. QT is often claimed as being more versatile and easier to use than the JFC. [DAL] Like Java, QT attempts to provide a cross-platform tool-kit. However, it is less successful than Java in this respect, since it is not possible to compile a QT application just once and then use it anywhere. However, QT does provide significantly better performance than Java and has lower memory requirements. QT also supports a wider range of platforms than Java.

3.3 Web Services

The term *Web services* refers to computing resources that are not available locally but are accessed, via the Internet, through the use of *XML (eXtensible Mark-up Language)*. [XML] Consider how a simple calculator might be implemented as a Web Service, using a form in a web page. The user defines a mathematical expression that is to be evaluated. This is then sent to another computer, somewhere else in the world that is available to perform that task. Once it has been performed, an XML document containing the answer, is returned to the initiating computer. So far, few developers have exploited the potential of Web Services. In general terms, Myriad conforms to the model of a Web Service, although there are some differences.

3.4 Wireless technologies and Portable Computers

Wireless networking uses h.f. radio to transmit data at high speed between digital devices. It is possible to connect remote computers to LANs, secure WANs, or the Internet. Interest in wireless access technology is growing rapidly, as it enables portable devices to transmit and receive data in areas where cabling is undesirable for technical reasons, too expensive or just impossible. Currently, there are two families of wireless technologies available: *Bluetooth* and WiFi (*802.11*). [WIF]

Small, portable computational devices, such as palm-top computers, personal digital assistants *(PDAs)* and tablet PCs, are often provided with wireless communication facilities, based on cell-phone or wireless LAN technology. Although, these devices all currently have limited processing power, Machine Vision can make good use of them, for portable

image gathering, image display and remote control of a Myriad network. Although the processing abilities of such devices currently limit their potential for image processing, they can obtain virtually unlimited computing power via the Myriad network.

3.5 Embedded Systems

Embedded systems have seen substantial development in recent years and can now run full desktop grade operating systems and applications. They are physically small and have few moving parts. Specialised chip-sets exist for "heavy duty" processing tasks such as pattern recognition, digital image processing, time-series analysis, Fourier transforms, etc. Specialised chip-sets also exist for interfacing to USB, "Fire-wire" (IEEE 1394), Ethernet and other networks, as well as for device control. It is possible now to purchase versatile and low-cost, device-control hardware that is operated directly via the Internet. One family of devices of this kind can control LED and mains lighting, fans, heaters, d.c. motors, solenoids, pneumatic and hydraulic valves, an (X,Y,Theta)-table, a multi-axis robot, and even a model train! (Section 4) The control PCB (the *"Gateway"* board in Figure 3(c)) is given its own unique IP address and can control several slave boards, controlling both input and output devices.

3.6 Network Cameras

Some video cameras allow direct Internet connection, thereby alleviating the need for a host computer. For example, Axis Communications AB. sell a camera system, which is web friendly. [AXI] It is assigned an IP address and operates as a web server, despatching images, either continuously, or one at a time, via its Ethernet port. Thus, digital images can be obtained on demand and displayed using a standard web browser, or processed using software such as Myriad.

4. MYRIAD: A NEW CONCEPT FOR MACHINE VISION

4.1 Myriad Networks and Components

Figure 1 shows the structure of a generic Myriad system. Myriad is a framework, for building and operating networked vision system, and does not refer to any one specific configuration. More elements of each type can be added to the network, virtually at will. These component sub-systems may be local to each other, or anywhere in the world, as long as they are able to communicate via the Internet, or some intranet. Myriad components may be any of the following:

Controllers (user interfaces, control applications, scripts)
Fully controllable servers (scriptable or dumb)
Data sources (such as networked cameras).

Each element in a Myriad system, if not purely a controller such as a user interface, responds to HTTP requests, which can contain embedded commands to a server. (Web browsers normally issue HTTP requests.) A web browser can provide a rudimentary user interfaces for a Myriad system. Each of the other elements in a Myriad system operates conceptually as a web server and is able to despatch images and/or other data on demand. They are also able to perform certain operations that are not immediately visible, such as filtering an internal image. Later, that element may receive a request to despatch that processed image to the command module, or to a third device.

Myriad servers incorporate a basic scripting language (*Myriad Component Script, MCS)*, which defines commands that are embedded within the incoming HTTP request. In this way, the receiving module can perform conditional operations, looping, data I/O and file I/O. Variables and common mathematical operations are also supported. Servers may also form HTTP requests of their own, enabling them to control other Myriad components.

Currently available Myriad components are:

Web cameras ("webcams")
Databases
Image Processing
Networked cameras
Equipment controllers
Prolog-based programmable intelligent controllers

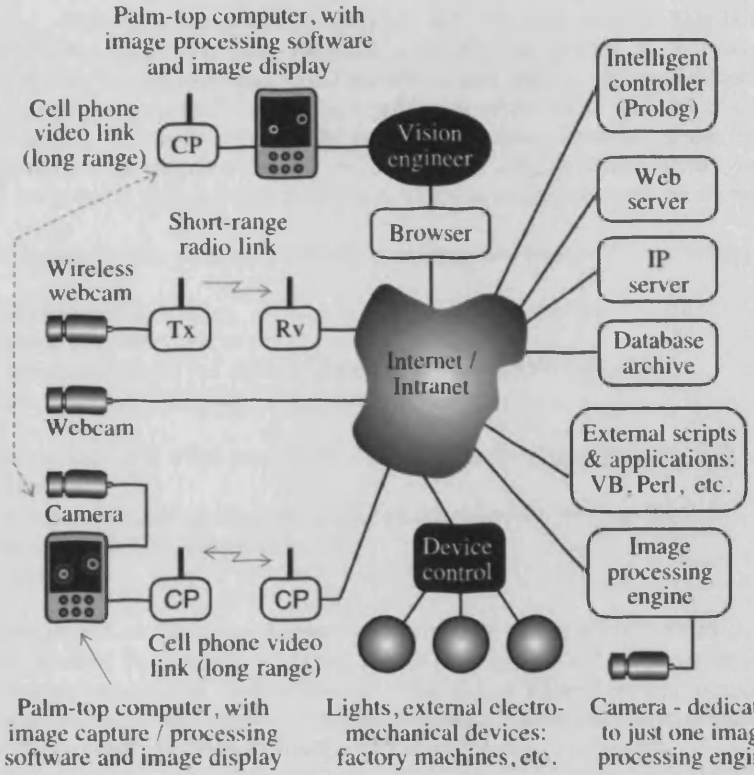


Figure 1 Distributed vision systems. Notice that cameras may be connected by cable, short-range radio, or cell-phone. Each of the modules shown here may be replicated many times, to create a complex mesh of inter-operating sub-systems.

4.2 Communications in a Myriad Network

Operating Myriad server components is similar to working with conventional web servers and scripting languages, which those provide. Communication takes place through HTTP requests, structured as illustrated in the following example:

http://server:3080/temp.jpg?commands=start%0D%0Aneg%0D%0Astop % A command sequence can replace *"neg"*

These may be produced by any application or scripting language, provided it has the capability to:

Open a TCP/IP socket

Send a string through the socket connection

This behaviour includes almost all currently used programming and scripting languages, including C/C++, Java, Visual Basic, Prolog, Perl, Python, PHP, Ruby, TCL and many more.

It is worth noting that any HTTP request may be formed by typing the appropriate URL, with embedded commands, into a web browser address bar. This is useful for testing Myriad sub-systems and allows "light-weight" command interfaces to be written in HTML. The latter can be written and used by anyone, working in any location, provided, of course, that they know the appropriate URLs and commands for the available Myriad elements. Since Myriad communication and control conforms to standard web protocols, image processing and device-control tasks may be easily harmonised and integrated into other web tasks/operations, such as business management.

4.3 Scripting

MCS allows for macro and autonomous operations, which are critical to networked systems, for two reasons, explained below.

Networked systems may suffer from variable, unpredictable and very long latency. This inhibits a component from exercising effective real-time control over other elements in the Myriad network. As a result, common time-sensitive operations, such as informing the network about local conditions and activity, automated looping, etc. are impractical, unless they are performed autonomously. For example, consider the relatively simple task of capturing an image from a networked camera, processing it, calculating the light level in the scene and adjusting the lighting to compensate for fluctuation in ambient illumination. A non-scriptable server (B), with a high network latency, might perform this task as follows. The top-level controller is designated as machine A. (Typical execution times are shown in square brackets, thus [...].)

(a) A sends the initial command to B, ordering it to capture an image and analyse it by measuring the peak intensity. [2000 ms]
(b) B performs the operation specified in (a). [300 ms]
(c) B returns the maximum intensity value to A. [2000 ms]
(d) A decides on the appropriate action and sends a command to A to turn the lights on. [2050 ms]
(e) B turns the lights on. [150 ms]

The total time for these operations is 6500 ms. On the other hand, a scripted server (B), with a similar network latency would be considerably faster:

(f) Client sends initial request for capture, analysis and scripted instructions for response. [2,000ms]
(g) Server performs operation, decides to turn lights on. [400ms]

The total time is now 2400ms.

It often happens that two Internet devices lose contact with each other, as a result of network overload, or malfunction of one of the intermediate routers. As a result, a Myriad controller may lose connection with one or more of its server modules, which would then become idle. Meanwhile, the controller is waiting for a response that might never arrive. This last point emphasises the great importance of programming both control and server modules so that they are able to recover autonomously from a deadly embrace by being aware of real-time.

Scripting enables a controller to issue "high level" commands, that allow a server module to operate semi-autonomously or, if necessary, with full autonomy. This frees the networked vision system from network limitations, without which it would be unable to function effectively. MCS provides facilities for

(a) Defining and operating on variables, including scalars, vectors, arrays, and strings. The evaluation of arithmetic expressions is also possible.
(b) Controlling input/output, including image capture from remote cameras, data and image transfer to/from other Myriad components, I/O to remote files and databases
(c) Arithmetic expressions
(d) Constructing functions to be executed by other modules under the control of the "server". (Control module *A* programs "server" *B*, which in turn programs another module *C*.)
(e) HTTP request creation, execution and response analysis
(f) Flow control (IF-THEN and other conditions, looping, recursion)
(g) Image processing
(h) Controlling remote devices. Application specific terms, such as *"light-stripe generator"*, *"back-light"*, or *"over-head camera"*, may be used only within the controller, where they are translated into generic commands issued to the server. However, MCS does not limit Myriad to this mode of operation, as the server can be programmed remotely.

Here is an example of a simple script:

```
start                                                    % Start the interactive session
downloadimagefile testimage http://gemini.cs.cf.ac.uk/webcam_image.jpg   % Load image from the web
neg                                                      % Image processing command: negate
saveimage temp.jpg 100 jpeg                              % Save the image
stop                                                     % Stop the interactive session
```

4.3.1 System architecture

Myriad can form a range of different system structures, which may be have an hierarchical or mesh form. (Figure 2) In view of the fact that Myriad components are standard network services, they can be built into existing systems with little effort. Indeed, for most languages, connecting and transmitting data or commands are trivial exercises, involving the creation of just a few lines of code. Nevertheless, they allow existing applications or systems to take advantage of a range of vision components and data sources.

4.3.2 Hierarchical systems

With the ability of Myriad components to be controlled, or data gathered from them upon command, a hierarchical network can be formed by such components and is overseen by a controller application, user interface, or script. (Figure 2(a)) For smaller problems, this might be a shallow structure, with the controller processing the data and performing most of the logical functions.

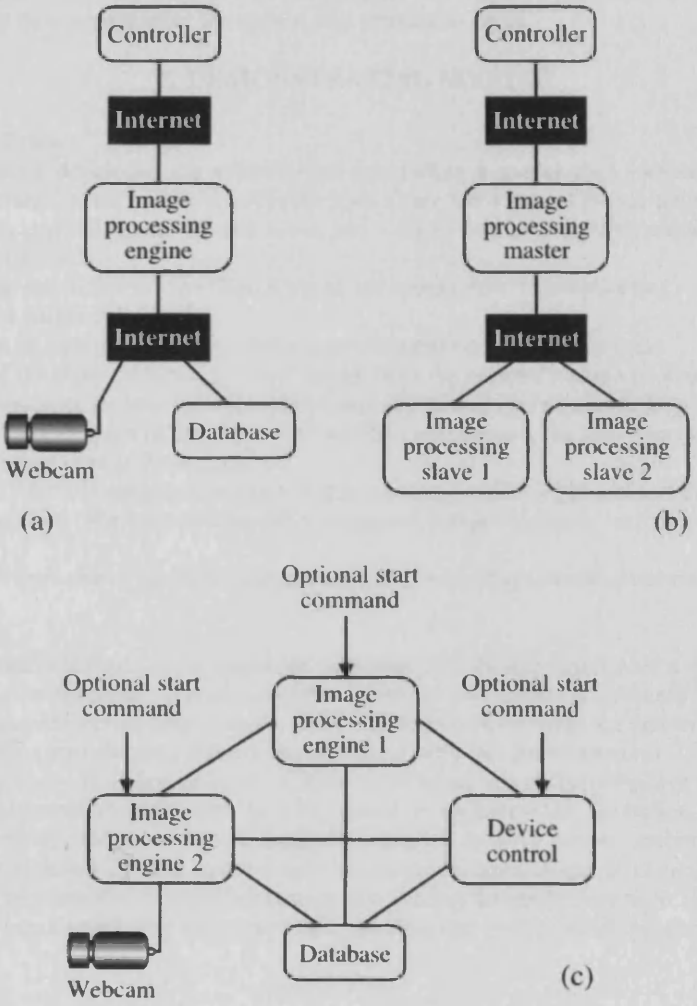


Figure 2. Some of the simpler Myriad networks. (a) Hierarchy, resembling a conventional integrated machine vision system. The database might typically be used to hold an archive of inspection results and can be interrogated at any time, using a standard web browser (not shown). (b) A master-slave arrangement. The controller is unaware of the division of labour that the image processing master decides is necessary. (c) A simple mesh. Two image processing systems are able to enter data into a database and to consult it, to find out what the other is doing. The device control is also able to interrogate the database, and enter data into it.

For larger hierarchical systems the components of the systems can be connected together in a vertical "daisy-chained" manner; components are able to pass HTTP requests to other components. This also allows components to act as controllers of sub-networks of components.

Scripting can be used to conceal complexity lower in the hierarchy and also facilitates the formation of clusters of processing elements, or load-balancing. To do this, the top-level controller communicates with a master element, which allocates tasks to sub-ordinate components. (Figure 2(b))

4.3.3 Meshes

See Figure 2(c). Myriad components are able to form HTTP requests, after an initial command signal. (Self-starting embedded devices do not even need this.) Components may then run autonomously and interact with each other, without further interaction by a controller. This form of non-hierarchical, peer-to-peer relationship is typical of basic monitoring operations; there is no host device organising the system on a continuous basis.

5. DEMONSTRATING MYRIAD

5.1 Controlling a Model Train

A Myriad network has been developed for viewing and controlling a model train located in the authors' research laboratory. [MYR]. The track layout is shown in Figure 3(a). There are 4 sets of switch points, two overhead cameras, one side-view camera with controlled pan, tilt and zoom, and a set of four LED lighting modules. Using a standard web browser (Figure 3(b)), the user can

(a) Start/stop the train and define its direction of travel and speed (slow, medium or fast).
(b) Switch each of the points individually.
(c) Set the brightness of each of the LED modules separately and on a continuous scale.
(d) Select any one of the three cameras. A "live" image from the selected camera is displayed continuously, until a new camera is chosen, or the user asks for some image processing operation(s) to be performed.
(e) Control the pan, tilt and zoom of the side-view camera. (This camera has auto-focus and automatic gain control, so these do not require user action.)
(f) Select any of the Myriad-enabled computers that is currently available, to perform the image processing that is subsequently specified. We have successfully processed images remotely on computers in U.K., and U.S.A. using Myriad.
(g) Digitise an image from one of the three cameras and process it, using a simple command, or a macro.

5.2 Interface Hardware

The organisation of the interface hardware is explained in Figure 3(c). Notice that it uses a web-friendly controller, the *Gateway* board. ([J43] It does not require a host computer and has its own unique IP address.) It has an Ethernet interface and can operate up to sixteen *M1* device control cards. [J43] The latter each provides the following facilities:

(a) One stepper motor with micro-stepping drive controller (not used in this demonstration)
(b) Three 2-pin ports driven by an H-bridge. Each of these is switched at a sufficiently high frequency (about 22 KHz) to allow the power delivered to a load such as a DC motor, to be controlled, by varying the mark:space ratio. The polarity is also reversible. (These ports were originally designed to drive motors operating the aperture, focus and zoom of a video camera lens.) The DC motor within the model locomotive can be operated directly by one of these ports. The LEDs can be controlled in a similar manner. By altering the mark:space ratio, the speed of the train, or the LED brightness can be adjusted. The twin solenoids operating the switch-points require two additional diodes, as shown in Figure 3(c).

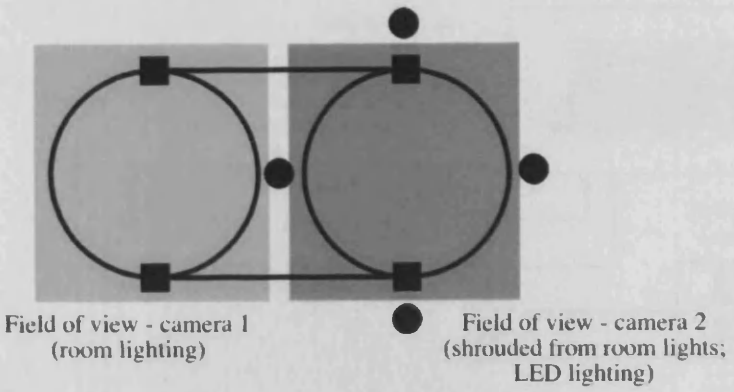


Figure 3(a)

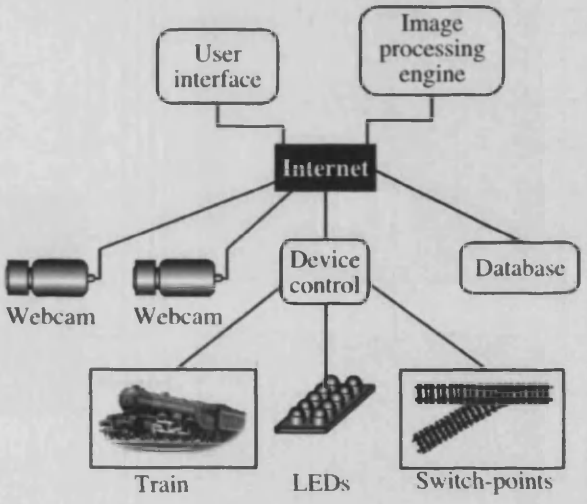


Figure 3(b)

5.3 Other Demonstrations

While our real interest lies in designing vision systems for industrial applications, a model railway provides cheap, controllable hardware that can easily be assembled to perform a variety of interesting functions. A model railway is particularly useful for training purposes and for demonstrations to non-technical personnel, since most people have had some child-hood experience with toy trains. Many of the functions needed to control a train are, in a very broad sense, conceptually similar to those encountered in manufacturing. For example, both require intelligence applied locally and globally. In addition, there are clearly identifiable parallels in sequencing, as well as recognising and avoiding unsafe operations. Furthermore, some interesting computational procedures, including numeric sorting and analysing reverse Polish expressions, involve *last-in/first-out* stores (stacks) and are often explained by analogy with a railway marshalling yard. The point is that controlling a model railway is not a trivial exercise and there is considerable scope for extending the demonstration, with minimal additional expenditure, to parallel some significant industrial applications.

One particularly interesting task that we plan to demonstrate in the near future involves coordinating the analysis of the two plan-view and side-view (pan-and-tilt) cameras, so that high-resolution views of (a small part of) the side of the train

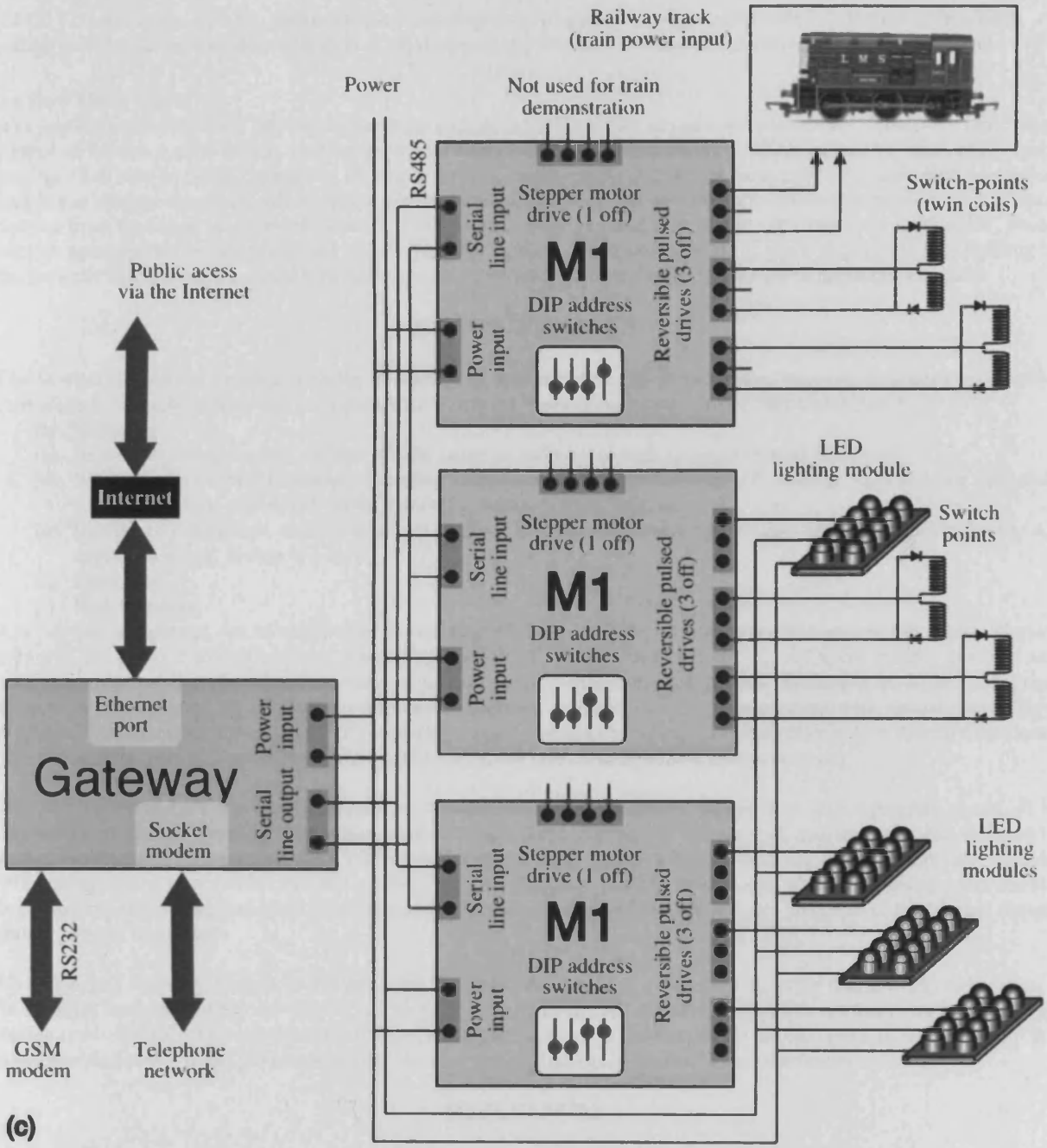


Figure 3. Controlling a model train. (a) Track layout. (b) Control hierarchy. (c) Hardware for operating the train, switch-points and LED lighting. An alternative arrangement uses a standard desk-top computer in lieu of the "Gateway" board.

can be obtained, wherever it happens to be. Myriad is ideally suited to tasks of this nature, since it is able to execute several independent image processing streams and then combine their results.

The hardware configuration shown in Figure 3(c) is also able to control devices operated by stepper motors. For example, a 3-motor robot, such as an (X,Y,Theta)-table, can be operated directly using the present set-up. In addition, to

the (X,Y,Theta)-table, a 2-axis, pneumatic pick-and-place arm, (together forming a robot with $3^1/_2$ degrees of freedom) will be used for yet another demonstration of Myriad: packing "random" 2-dimensional shapes, under visual control.

5.4 How Many Users?

It is perfectly feasible for a Myriad network to operate several robotic devices simultaneously. However, multi-user control of the same *slow* device, such as the model train, is clearly impractical, as conflicts cannot be resolved by time-sharing. (It is easy to derail the train by driving it through switch-points that are set incorrectly.) An unlimited number of people can observe the model railway track simultaneously, using Myriad; several users can simultaneously view images derived from the same, or different cameras. There is no reason why one user should not control the locomotive, while another operates the switch-points and a third the LED lighting. Multi-user control of rapid response (LED) lighting is theoretically feasible but the visual feed-back currently provided by the webcams is too slow to make this effective.

6. FINAL REMARKS

The Myriad framework enables complex ensembles of web cameras, image processing engines, databases and device controllers to be built. Myriad currently allows the following types of components to be inter-connected:

(a) Webcams.
(b) Image processing engines. (These may be based on software or high-speed, dedicated hardware.)
(c) Web-friendly control hardware, operating such devices as stepper motors, DC motors, LED lighting modules, solenoids, relays, pneumatic valves, hydraulic valves, heaters, fans, etc.
(d) Intelligent controllers, written in a conventional programming language, or one designed specifically for AI applications (e.g. Prolog or Lisp)
(e) Databases
(f) Web browsers.

Any Myriad component can be substituted for another of the same type, situated anywhere else in the world. Myriad networks can perform multiple-stream processing and can have several active users. An unlimited number of users can passively view activity elsewhere, provided of course, that the originator has granted permission to do so. A Myriad network can be operated by autonomous intelligent programs, acting *in lieu* of, or cooperating with, human controllers. A Myriad network also allows an image processing engine to enter observational measurements in a standard database. This data can then be analysed in the normal way by a human user, or a high-level control program.

The Myriad framework has been designed to achieve maximum flexibility, rather than high operating speed. It is impossible to specify "typical" execution times because these depend on the network traffic. Since the Internet is publicly accessible, transmission delays between the components within a Myriad network are unpredictable, making real-time operation impossible. For this reason, "local intelligence" is often needed and, wherever possible, data should be routed via high-speed dedicated communication channels. Provided these are Internet compatible, the Myriad design philosophy can be applied.

Myriad is still evolving. Indeed, it will never be "finished" because it is a general computing frame-work, rather than a finite entity designed for a fixed objective. The reader is invited to visit the Myriad web site and have fun operating the model train. [MYR] The application of Myriad to the packing of rectangles of mixed sizes is described in the accompanying paper. [CHA] We hope to provide other demonstrations of Myriad in the near future.

REFERENCES

AXI Axis Network Cameras, Axis Communications Axis Communications AB, Lund, Sweden, URL: *http://www.axis.com/products/video/camera/how.htm*

BAT B. G. Batchelor & F. M. Waltz, *"Intelligent Machine Vision: Techniques, Implementation & Applications"*, Springer-Verlag, London, Spring 2001, ISBN 3-540-76224-8.

BLU Bluetooth, Bluetooth Sig, inc. URL: https://www.bluetooth.org/

CHA L. T. Chatburn & B. G. Batchelor, *"2-dimensional Packing Using the Myriad Vision Framework"*, this conference"

DAL M. K. Dalheimer, *"Qt vs. Java: A Comparison of Qt and Java for Large-Scale, Industrial-Strength GUI, Development"*, Klarälvdalens Datakonsult AB, Sweden. URL: *http://linux.sarang.net/ftp/mirror/development/widget/qt/pdf/qt-vs-java-whitepaper.pdf*

J43 Gateway and M1 boards, J43 Designs Ltd., Neath. Wales, U.K. Email: garry@j43-design.co.uk

JAV Java Foundation Classes, Sun Corporation, Santa Clara, California, USA, URL: java.sun.com/docs

LAB LabView, graphical development environment for signal acquisition, measurement & analysis National Instruments Corporation, Austin, Texas, USA, URL: http://www.ni.com/aap/

MYR Myriad demonstration software, L. T. Chatburn & B. G. Batchelor, Cardiff University, Wales, UK, URL: *gemini.cs.cf.ac.uk/~dawnrider/train.html*

NEA Neatvision, Image Analysis & Software Development Environment, Dublin City University, Dublin, Ireland. URL: *http://www.neatvision.com/*

PHO Photoshop, Image editing software, Adobe Systems UK Ltd., Uxbridge Middlesex, UK. URL: *http://www.adobe.com/products/photoshop/main.html*

QT QT Toolkit, Oslo, Norway, URL: http://www.trolltech.com

WIF Wi-Fi Alliance, Austin, Texas, URL: http://www.weca.net/OpenSection/index.asp

XML XML (eXtensible Mark-up Language, W3C World Wide Web Consortium, European Research Consortium for Informatics and Mathematics, Sophia Antipolis, France, URL: http://www.w3.org/XML/

2D PACKING USING THE MYRIAD FRAMEWORK

L. T. Chatburn & B. G. Batchelor
School of Computer Science,
Cardiff University, Cardiff, Wales, CF24 3XF , U.K.

Email: *bruce.batchelor@cs.cf.ac.uk*

ABSTRACT

Myriad is a framework for building networked and distributed vision systems and is described in a companion paper in this conference. Myriad allows the components of a multi-camera, multi-user vision system (web-cameras, image processing engines, intelligent device controllers, databases and the user interface terminals) to be interconnected and operated together, even if they are physically separated by many hundreds, or thousands, of kilometres. This is achieved by operating them as Internet services. The principal objective in this article is to illustrate the simplicity of harmonising visual control with an existing system using Myriad. However, packing of 2-dimensional blob-like objects is of considerable commercial importance in some industries and involves robotic handling and/or cutting. The shapes to be packed may be cut from sheet metal, glass, cloth, leather, wood, card, paper, composite board, or flat food materials. In addition, many 3D packing applications can realistically be tackled only by regarding them as multi-layer 2D applications. Using Myriad to perform 2D packing, a set of blob-like input objects ("shapes") can be digitised using a standard camera (e.g. a "webcam"). The resulting digital images are then analysed, using a separate processing engine, perhaps located on a different continent. The packing is planned by another processing system, perhaps on a third continent. Finally, the assembly is performed using a robot, usually but not necessarily, located close to the camera

Keywords: *Machine Vision, Packing, Network, Internet, Distributed computing.*

1. INTRODUCTION

This article should be read in conjunction with the accompanying paper, which describes the Myriad framework employed in this application. [CHA03]

1.1 Myriad

Myriad is a frame-work for building complex ensembles of a variety of web-based components, including:

(a) Video cameras (including "webcams").
(b) Image processing engines. (These may be based on software, or high-speed, dedicated electronic hardware.)
(c) Hardware controllers, operating such devices as stepper motors, DC motors, LED lighting modules, solenoids, relays, pneumatic valves, hydraulic valves, heaters, fans, etc.
(d) Intelligent system controllers, written in a conventional programming language, or one designed specifically for AI applications (e.g. Prolog or Lisp)
(e) Databases
(f) Web browsers, operated by skilled or novice users.

Any Myriad component can be substituted for another of the same generic type, situated anywhere else in the world. Myriad incorporates a scripting language. *(Myriad Component Script, MCS)* Communication within the network takes place using conventional HTTP requests.

Myriad was designed so that networks can be connected together easily for rapid, prototyping of a wide variety of industrial vision applications. Myriad networks can perform multiple-stream processing and can have several active users. An unlimited number of users can passively view activity elsewhere in the Myriad network, which can be operated by autonomous intelligent programs, acting *in lieu* of, or in cooperation with, human controllers. A Myriad network also

allows an image processing engine to enter observational measurements in a standard database. This data can then be analysed in the normal way, either by a human user, or a high-level control program.

1.2 Packing

In the context of this article, the term "packing" also refers to cutting (also called *depletion*), where we are effectively required to pack holes as densely as possible. Packing problems occur throughout industry, in such areas as manufacturing/using carpet, window glass, sheet metal, clothing, foot-ware, food products, etc. It is obvious that packing arbitrary 2D and 3D shapes can be extremely difficult and time-consuming. However, even the seemingly straightforward task of packing rectangular objects can be very difficult; if there hundreds of them with random sizes and aspect ratios. There are numerous classes of packing problem in industry, reflecting the fact that there are many different sets of application constraints. [BAT97] For example, in certain cases, it is not permitted to rotate the shapes arbitrarily (e.g. patterned fabric), or to rotate them only by integer multiples of 90 degrees (leather). Furthermore, in some instances, it is necessary to pack the shapes in such a way that unpacking can be performed automatically, using a robot with "fat fingers". In some cases, such as handling leather, it is necessary to cut some shapes from certain parts of the material supplied, for example where the material is supple, defect free and has a fine grain. Algorithms for packing rectangles can be modified slightly to allow arbitrary shapes to be put in place; we simply replace each shape by the minimum-area rectangle surrounding it. (Figure 1) The result is often inefficient but, for certain mixtures of shapes, this approach can sometimes produce *adequate* results. (Small shapes, can fit into the gaps between bigger shapes and into holes within shapes.) Bearing this comment in mind and the fact that this article is concerned primarily with demonstrating the application of Myriad, we shall concentrate on packing rectangles in a 2D space. Moreover, we shall deploy one specific set of packing programs, implemented in a publicly accessible program called *LinPacker* [NAG]. 3D packing is beyond the scope of this paper and will not be discussed further.

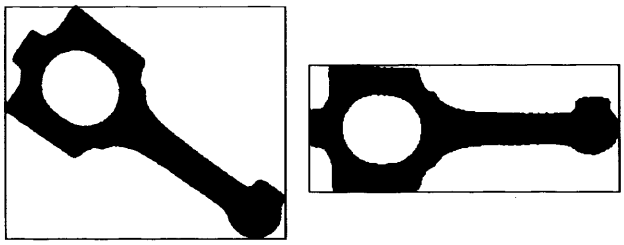


Figure 1. Representing an arbitrary 2D shape by the minimum-area rectangle. Left: rotation is prohibited. Right: rotation is permitted. The object has been rotated so that the axis of minimum second moment ("principal axis") is horizontal. Orienting all of the shapes like this often results in a more efficient packing.

Packing is an exemplary problem of Artificial Intelligence, since it is impossible to perform an exhaustive search for solutions, except in the simplest cases. It is perhaps less obvious that Machine Vision is also relevant to packing. Packing rectangles into a rectangular container can be performed without resorting to image processing. However, it is easier to understand the process, if we represent it visually, as in Figure 2. When packing irregular blob-like shapes into an irregular container, as we do, for example, when cutting leather from an animal hide, it is essential to represent them both, in the form of binary images. (Some convenient coding for binary images, such as Run-length or Chain Code may be used for convenience. [BAT97] Packing 2-dimensional blob-like objects involves a range of image processing algorithms: affine transformations (linear shift, rotation, scale change) and parametric representation of shape and a variety of others (e.g. minimum-area rectangle, circum-circle, convex hull, etc.). The important point to note is that packing complex blob-like objects necessarily relies on image processing, since there is no simple parametric way to represent either the shapes or the container.

1.3 Packing in a Distributed System

Packing is not our primary area of research and, as a result, we have not improved any of the heuristics that already exist for this challenging task. Specifically, we have not attempted to improve on existing 2D packing procedures, nor consider packing in 3- or high-dimensional spaces. However, the ability to plan a packing strategy based on visual data that has been acquired remotely is of potential significance for a variety of industries. We do not survey these in detail here. Instead, let it suffice to note that cutting leather and fabric to make shoes, gloves, clothing, bags, etc. is an

archetypal application of this kind; a fashion designer defines the shapes (i.e. garment/shoe components) to be cut out but is unlikely to work in the factory where they are actually cut out and sewn together. Other applications are likely to be found in the food industry, where natural objects of variable shape and/or size are to be assembled. Many other applications that supposedly require 3D packing are more easily solved by representing them instead as multi-layer 2D packing problems. Packing fruit is one example.

Multi-axis manipulators may be used to perform the physical movement of parts and materials needed for packing but many situations exist in which programmable flat-bed cutters (milling machines, jig-saws, laser and water-jet cutters) are used instead. For our purposes, these are all *robots* and benefit from visual control. Many applications require the use of real or computer-based templates, These may have been cut/sketched by hand. Sometimes, the robot and its controller may be far removed from the point where templates are created. (e.g. a CAD system, or design studio fitted with a webcam). An important motivation for separating robotic handling/cutting from the other parts of the system is cleanliness and safety. In some situations, the remote control of a packing robot, via a networked system is important for management reasons:

Template designs and packing solutions can both be stored at corporate head-quarters, in databases that allow company-wide access.

Expensive, sophisticated equipment may be used to service multiple sites, thereby reducing both capital expenditure and labour costs.

Packing results may be tracked over time and all company sites, enabling trends in manufacturing to be recognised

Objects designed at one site may be cut elsewhere and finally packed at a third location.

Network-based packing systems offer the potential for improving manufacturing efficiency and management control.

2. TECHNOLOGIES

This topic was discussed in the companion paper. [CHA03] However, there are certain additional issues that are of particular interest to the way we use Myriad for packing.

2.1 Open Source and Linux

The term *Open Source* refers to the global movement towards the free sharing of software. It relies upon the easy communication provided by the Internet and programmer integrity. The principal open-source operating system is *Linux*, which is version of Unix and currently enjoys considerable popularity, with over a million program developers. Linux and associated programs are most often licensed under the *GNU General Public Licence*. [GPL] This allows free copying and distribution of software, provided that the source code of any modification is made publicly available throughout the world. Linux, the *Apache* web server and *MySQL* database system are examples of Open Source software and were used in the development of Myriad.

2.2 KDE (K Desktop Environment)

Unix operating systems typically run a graphical X11 server and then a desktop environment above that. (*CDE* operates above Solaris and *KDE* over Linux.) KDE was written using C++ and the QT toolkit [QT] and is functionally comparable to desktops such as MacOS and Windows. However, KDE and Linux are free and, through the use of QT, provides complete network transparency to programs and the RPC/messaging system, DCOP.

2.5 DCOP

CORBA [COR] has been used for communication and control between programs but was found to be slow and unwieldy. In response to this, DCOP [SWE00, DCO] was created as a lightweight communications protocol between computing processes. It operates as a server; client programs announce the functions they provide and DCOP subsequently controls and directs messages between them. Thus programs written in languages, such as Perl, Ruby, Python, etc. can then send commands to others, via the DCOP server. DCOP may be used both locally and across a network. We have used DCOP to control the LinPacker packing program.

2.6 Perl

Perl is a scripting language that is particularly well suited for text processing and symbol manipulation.[PER] It has been often been used to create dynamic web sites, as it is able to alter the structure of HTML pages in response to user input from a command-line interface. A Perl program can, for example, be used to reconfigure a system based on Unix. Perl was used to control the Myriad- LinPacker system described below.

3. LINPACKER: AN "OPEN" PACKING PROGRAM

LinPacker [NAG] is a program for packing rectangular objects and was written by Thomas Nagy. It began life as a personal project and was later expanded into a small *Open Source* project [GPL]. LinPacker is written in C++, using the QT toolkit. In order to fit it into the Myriad framework, additional input/output routines were written by the present authors. The addition of the DCOP, communications protocol forces it to run on systems provided by the KDE desk-top environment. This has resulted in the loss of cross-platform capability. However, it can now be controlled from Myriad scripts, which was essential for this project.

3.1 Inputs

LinPacker is designed around two essential components:

A table containing the data (rectangle width, height and rotational freedom) describing the objects that are to be packed. (This is called the *Objects Table.*)

A container into which these shapes are to be packed. This might for example, be a simple rectangle. In addition, rectangular "prohibited areas", where objects may not be placed, can be specified by the user. (We shall call the space within the initial container that is currently available for packing further objects the *remnant space.*) Initially, the remnant space is a rectangle with rectangular "holes" representing the "prohibited areas". These might represent defects, holes or other obstacles to packing, within the initial remnant space. The packing area can be customised through a dialog box.

Users are able to enter object data directly into the table of objects, or import *XML (eXtensible Mark-up Language [XML])* data files. The file structure is defined as follows:

```
<project>
  <settings>
        <algorithm value="1"/>            %Packing algorithm to use
        <bandeheight value="23040"/>      %Length of packing solution
        <bandewidth value="7000"/>        %Width of packing solution
        <nbgenerations value="250"/>      %Algorithm setting variable
        <nbelements value="1000"/>        %Algorithm setting variable
        <mutations value="5"/>            %Algorithm setting variable
        <crossovers value="70"/>          %Algorithm setting variable
        <useconstraints value="false"/>   %Use constraint areas
  </settings>
  <objects>
        <object>
              <width value="4000"/>       %Rectangle width
              <height value="4000"/>      %Rectangle height
              <posx value="0"/>           %X Position in solution
              <posy value="0"/>           %Y Position in solution
              <num value="16"/>           %Rectangular ID number
              <orientation value="1"/>    %Rotational freedom
              <order value="16"/>         %Order of packing number
        </object>
  </objects>
  <constraints>
  </constraints>
</project>
```

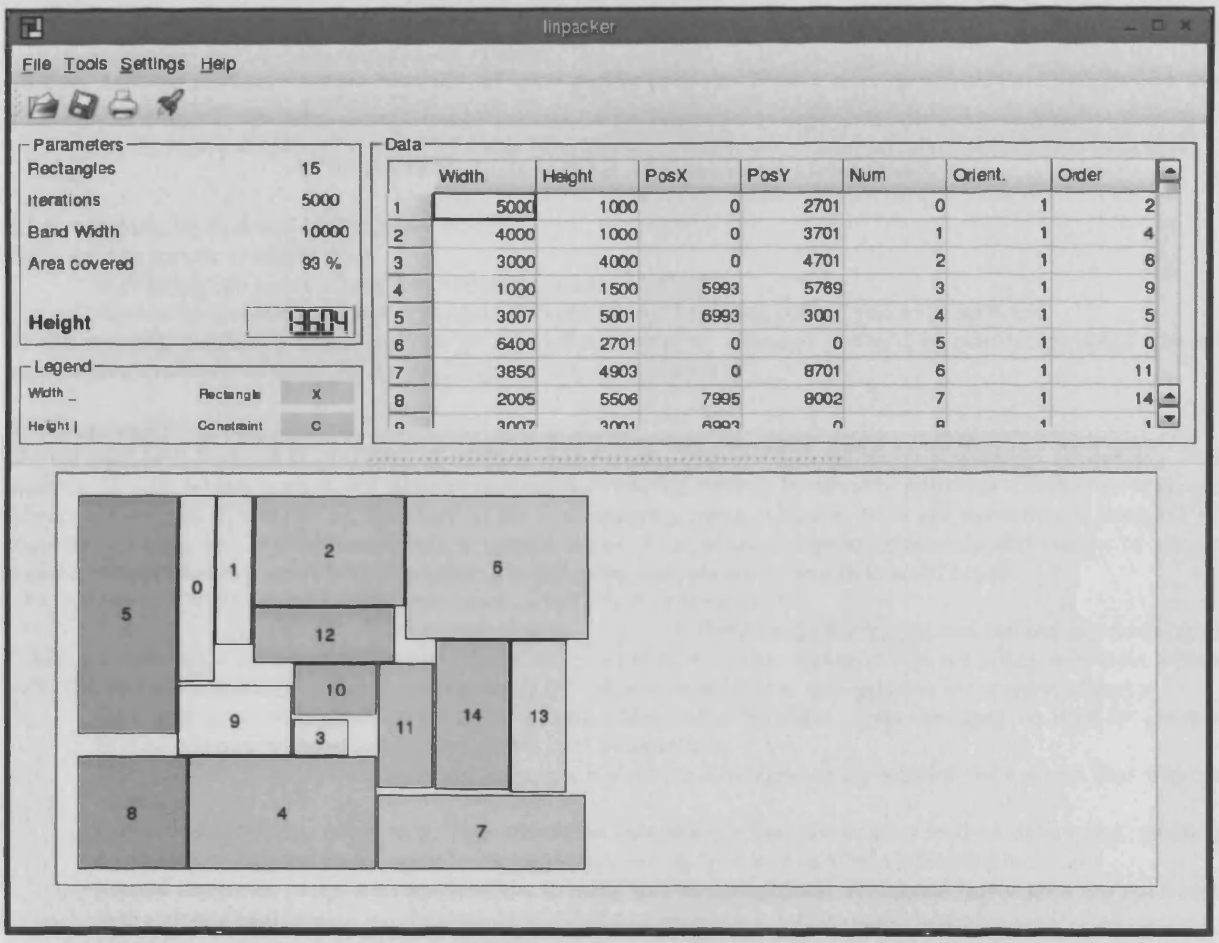


Figure 2. Typical display generated bu LinPacker. The Objects Table is located in the top right and the packing area at the bottom of the window. The top left section displays status information, including a measure of the packing efficiency. The program provides random colouring of the fitted rectangles. This is intended to assist visibility.

3.2 Packing Algorithms

LinPacker currently contains three algorithms for packing, with options for iteration, seeding and search-tree formation:

Genetic algorithm (any reasonable number of rectangles)
Genetic algorithm with constraints (recommended for fewer than 50 rectangles)
Simulated annealing (recommended for more than 50 rectangles)

Once the shape and container parameters have been set, packing begins. The results are shown on a canvas beneath the object table. "Random" colour coding and numbering of objects is provided to help the user appreciate the packing results. (Figure 2) When the process has been completed, the *Objects Table* is modified, by the addition of the position and orientation parameters that have been calculated for each rectangle.

3.3 Output

LinPacker can generate output data in two ways:

1. *Printing*. This takes place in the conventional manner.
2. *Data export using XML files* This allows packing results to be made available "on-line" to CAD/CAM systems. While the data formats for the latter may vary, XML is a human-readable language, which can be converted

easily to other formats. This allows the Myriad-LinPacker system to be integrated with existing manufacturing equipment and business management systems.

To date, we have not interfaced the Myriad-LinPacker system to control a robot or programmable cutter, although this would be a straightforward task.

4. OPERATING LINPACKER AND MYRIAD TOGETHER

4.1 Controlling Myriad and LinPacker

There are two options available to us:

(a) The Myriad server acts as the controller operating LinPacker

(b) Another program operates as the overall controller and organises both Myriad and LinPacker.

At the moment, LinPacker cannot operate as the system controller, although it would be possible to make a small modification to allow it to do so. As it provides greater clarity, we chose (b).

4.2 Communicating with Myriad

Myriad may be thought of as providing specialised web servers, with facilities for image acquisition, processing and analysis, as well as data storage and (simple, non real-time) device control. It currently possesses a scripting language: *Myriad Component Script (MCS)*, described in the accompanying paper. [CHA03] MCS has the ability to form HTTP requests, allowing one Myriad component to control others. This process relies on commands that consist of simple parameters appended to a normal HTTP request. The following example shows how this can be done:

```
http://www.mymyriadserver.com/test.jpg?commands=start
```

% Fetch image file *test.jpg* and perform operation *start*

Additional commands are separated by *carriage return* characters, which are represented by the string *%0D%0A* within the URL, which is composed of unicode characters. [UTF] Myriad benefits from this approach in a number of ways:

Data may be stored remotely, by creating image files and/or text files. This data may be used by another Myriad component immediately, or retrieved later by its creator.

Many modern programming/scripting languages are able to issue requests for remotely stored web files without difficulty.

Requests are textual, not binary. This simplifies data storage and recovery, as well as debugging; creating commands is straightforward, even in languages that have only rudimentary string-handling facilities.

Myriad integrates easily with much of the existing web-based software. Of special importance are databases with scripting facilities.

Myriad components benefit from security, caching, load balancing and access direction methods that are available for web servers.

Content messages (dynamic results or status messages) may be substituted or alternated directly with existing static web data.

Myriad network components may be controlled, or tested, simply by directing a web-browser to a Myriad URL.

4.3 Communicating with LinPacker

LinPacker is limited in its control abilities compared to other Myriad components. However, by using DCOP, client programs can gain access to a number of its internal functions. Of special note are the following functions specifically:

Solvepacking(datafile.lnpk) Loads the data file named by the client into the rectangles table and runs a selected packing routine on it.

P*rintimage(datafile.lnpk, image.jpg)* Prints the packing solution currently rendered on the canvas.

LinPacker can save packing solutions in data files, although it is not yet possible to publish them via DCOP. (This facility will be added in the near future.) To access the DCOP functions, the control program must first start a DCOP client and the program to be controlled. It is then possible to send simple messages, or calls, which return data.

4.4 Scripting Myriad and LinPacker

The processes for controlling the Myriad-LinPacker system are straightforward using a Perl script; just a few lines of code are needed for each. To operate Myriad, the Unix system program 'wget' is used to download a web file from the Myriad server. A URL is then requested with the commands embedded as an argument. Using the example given in Section 4.2, we would add a single line to the control script:

```
system 'wget http://www.mymyriadserver.com/test.jpg?commands=start';
```

In order to perform the packing, Myriad requires a script, instructing it to capture an image from a network camera, process it, isolate each of the "blobs" that the image contains, calculate the parameters describing their minimum-area rectangles and finally save a data file. Here is such a sequence expressed using the image processing commands described in [BAT97]

```
downloadimagefile myimage http://mycomputer.cardiff.ac.uk/eyeimage.jpg
sed                                 % Sobel edge detector.
thr 16                              % Threshold at level 16.
blb                                 % Fill any holes in the "blobs".
store blobs                         % Store resulting binary image.
$x = 0                              % Set variable $x to zero.
cbl $y                              % Find no. of blobs. Store in $y.

openfile objectsfile datafile.lnpk  % Open a file
printlnfile objectsfile <project>   % Start LinPacker data output
printlnfile objectsfile <objects>   % Continue data output


while ( $x <= $y )                  % Loop, to process each blob
{
$x += 1                             % Increment the loop counter
use blobs                           % Reload the stored blobs image
big $x                              % Find blob of rank size $x
crp                                 % Crop image to fit the blob
saveimage ${x}.jpg 100 jpeg
getwidth $width                     % Width of minimum-area rectangle
getheight $height                   % Height of min-area rectangle

printlnfile objectsfile <object>    % Data output for the rectangle
printlnfile objectsfile <width value="${width}"/>
printlnfile objectsfile <height value="${height}"/>
printlnfile objectsfile </object>

}

printlnfile objectsfile </objects>
printlnfile objectsfile </project>
closefile objectsfile
```

This script is converted into the following, rather lengthy URL to be used by *wget*:

http://mycomputer.cardiff.ac.uk:3080/?commands=start%0D%0Adownloadimagefile+myimage+http%3A%2F%2Fmyc
omputer.cardiff.ac.uk%2Feyeimage.jpg%0D%0Ased%0D%0Athr+16%0D%0Ablb%0D%0Astore+blobs%0D%0A%24
x+%3D+0%0D%0Acbl+%24y%0D%0Aopenfile+objectsfile+datafile.lnpk%0D%0Aprintlnfile+objectsfile+%3Cproject
%3E%0D%0Aprintlnfile+objectsfile+%3Cobjects%3E%0D%0Awhile+%28+%24x+%3C%3D+%24y+%29%0D%0A
%7B%0D%0A%24x+%2B%3D+1%0D%0Ause+blobs%0D%0Abig+%24x%0D%0Acrp%0D%0Asaveimage+%24%7
Bx%7D.jpg+100+jpeg%0D%0Agetwidth+%24width%0D%0Agetheight+%24height%0D%0Aprintlnfile+objectsfile+%
3Cobject%3E%0D%0Aprintlnfile+objectsfile+%3Cwidth+value%3D%22%24%7Bwidth%7D%22%2F%3E%0D%0Ap
rintlnfile+objectsfile+%3Cheight+value%3D%22%24%7Bheight%7D%22%2F%3E%0D%0Aprintlnfile+objectsfile+%
3C%2Fobject%3E%0D%0A%7D%0D%0Aprintlnfile+objectsfile+%3C%2Fobjects%3E%0D%0Aprintlnfile+objectsfile
+%3C%2Fproject%3E%0D%0Aclosefile+objectsfile%0D%0Astop

Optionally, we can save the script to a pre-defined text file, located on the server, and call it instead using the following URL:

```
http://mycomputer.cardiff.ac.uk:3080/?script=myscript.ips
```

Using LinPacker requires a few more lines of code but these are also quite straightforward:

```
use DCOP;                 % Use the dcop module
my $client = new DCOP;    % Create a new dcop client object
$client->attach();        % Connect to the dcop server
my $LinPacker = $client->createObject("LinPacker", "sinterface");
% Start a new LinPacker process and use the 'sinterface' interface
$LinPacker->solvepacking(datafile);  % Call the LinPacker pack method
```

In total, therefore, our actual control script to run both Myriad and LinPacker totals seven lines of code. The final script is:

```
#!/usr/bin/perl
use DCOP;
my $client = new DCOP;
$client->attach();
my $LinPacker = $client->createObject("LinPacker", "sinterface");
system 'wget http://mycomputer.cardiff.ac.uk:3080/?script=myscript.ips';
$LinPacker->solvepacking(datafile.lnpk);
```

5. THE COMBINED SYSTEM

5.1 System Introduction

The complete visual packing system was developed quickly, using a minimal amount of "programming glue". Myriad and LinPacker scripting as well as the communications facilities provided by DCOP make this possible. While we have used Perl script, it would be a simple matter to add code directly to LinPacker to control Myriad before packing begins.

5.2 System Architecture

The architecture of the final visual packing system is rudimentary. (Figure 3) There are just three elements: Myriad, LinPacker and the Perl script that controls them. In this small hierarchy. Myriad also has a child: the network camera being used to capture the image.

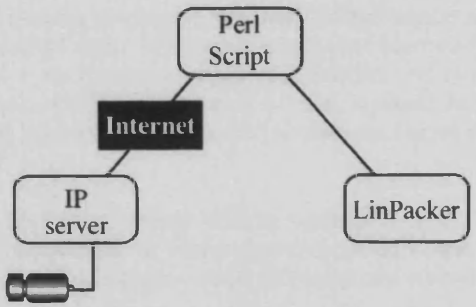


Figure 3. Myriad-LiPacker system.Both sub-systems are controlled by a Perl script, while the camera is controlled by the IP server.

5.2 Operation (Figure 4)
The visual packing procedure operates as follows:

(a) Perl script sends a request to the IP server
(b) Myriad collects image from camera
(c) Myriad processes image, extracts objects and saves results
(d) Myriad returns the request to the Perl script, indicating completion
(e) Perl script sends DCOP message to LinPacker to start packing the saved data
(f) LinPacker loads the data saved by Myriad
(g) LinPacker performs the packing operation
(h) LinPacker prints the packing solution.

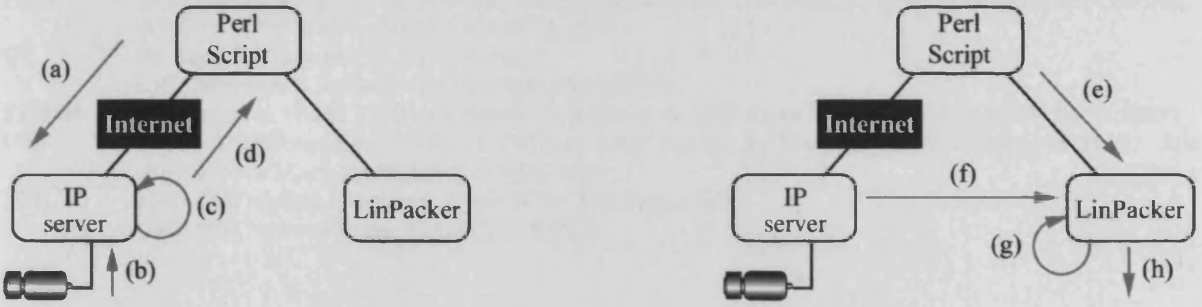


Figure 4.Operation of the Myriad-LinPacker system. Data transfers, indicated by labels (a)–(h), are defined in the text. Left: controlling the IP server, capturing an image, fitting the minimum-area rectangle around each of the blobs and saving the data. Right: Controlling LinPacker to gather the packing data, process it and output the resulting packing solution to the user.

5. FINAL REMARKS

The solution described here is an example of the type of application for which Myriad was designed. Although this is not claimed to be a robust working system that could be used within a factory, it does illustrate the ease with which a prototype networked Machine Vision system may be constructed, using the Myriad framework. Packing 2D shapes has

proved to be an ideal application for demonstrating this capability, since it requires the integration of acquisition and analysis of images and the involvement of complex AI heuristic methods. In the companion paper, Myriad is shown to be capable of operating simple mechanical devices (a model train) and several cameras.

Integrating Myriad with a general packing program (LinPacker) did not require any code change. Moreover, this requires a mere seven lines of unremarkable Perl script. (This script could have been made even shorter, if LinPacker were edited directly.) The ability to integrate a multi-camera, multi-user network of vision modules with other systems whilst writing very little "glue" code is one of the key features of Myriad. It should be noted too that no detailed knowledge of the intricacies of Machine Vision algorithms is needed. (The user merely needs to know what each image processing operator does, not how it does it.)

In the near future, the Myriad-LinPacker system will be updated to pack ellipses and irregular shapes. Further demonstrations of Myriad are planned around the visual control of a model train, telling the time from an analogue clock that is viewed by a webcam and operating a robotic camera with automatic control of pan, tilt and zoom.

REFERENCES

BAT97 B. G. Batchelor & P.F.Whelan, *"Intelligent Vision Systems for Industry"*, Springer Verlag, London & Berlin, 1997, ISBN 3-540-19969-1. Now on line at URL: http://www.eeng.dcu.ie/~whelanp/ivsi/

CHA03 L. T. Chatburn, G. R. Jackson, M.W. Daley & B. G. Batchelor "Framework for Distributed and Networked Vision Systems" this conference.

COR CORBA, application messaging and remote procedure call system. URL: http://www.corba.org (accessed 18/10/2003)

DCO DCOP: Desktop Communications Protocol, URL: accessed http://developer.kde.org/documentation/library/kdeqt/dcop.html (accessed 18/10/2003)

GPL GNU General Public License, free software distribution/copying license. URL: http://www.gnu.org/licenses (18/10/2003)

NAG LinPacker software, Thomas Nagy, l'Ecole des Mines de Nantes, France. URL: http://LinPacker.tuxfamily.org (18/10/2003)

PER URL: http://www.perl.com (accessed 18/10/2003). See also L. Wall, et al, 'Programming Perl' 3rd Edition, O'Reilly & Associates, ISBN 0596000278, 2000.

QT QT Toolkit, Trolltech AS, Oslo, Norway, URL: http://www.trolltech.com (accessed 18/10/2003)

SWE00 D. Sweet et al. "KDE 2.0 Development" D. Sweet et al. 2001 Sams Publishing, ASIN 0672318911, 2000.

UTF U(CS) Transformation Format-8 (UTF-8) (also known as *Unicode*), Ken Thompson 1992, URL: http://www.w3c.org/International (18/10/2003)

XML eXtensible Markup Language, World Wide Web Consortium, URL: http://www.w3c.org/XML/ (18/10/2003)