

School of Engineering, Cardiff University

Process Design and Supervision:
A next generation simulation approach
to digitalised manufacturing

A thesis submitted in partial fulfilment for the degree of Doctor
of Philosophy.

Theocharis Alexopoulos
September 2022

ABSTRACT

Modern processes will increasingly have a digital counterpart which is an interactive representation of the physical system integrated into a digital environment. At the heart of this digital counterpart are simulators that use raw data and calculation models to automate supervision tasks and increase process autonomy. As such, simulators have become a critical part of process digitalisation. But, despite the exponential increase of digitalisation related research simulators have not evolved to fully utilise the latest practices for data value extraction.

This research work examines the current role of simulation within digitalised systems, identifies state-of-the-art simulator structural components and proposes a design architecture for next generation simulators. The proposed architecture provides a structured way to develop next generation simulation systems. At the same time, it embeds the latest data science related technologies into the simulator and enables the integration of the simulator with modern edge or cloud systems. To achieve that, the simulator is broken down into five elements and the function of each element is specified based on system performance, digital environment compatibility and development ease.

To demonstrate the effectiveness of the architecture, the author developed a vertical machining centre simulator that uses a mesh-based method to represent the process and the latest automated machine learning techniques to generate knowledge from the information extracted by the monitoring data. To verify the capabilities of the simulator a series of experiments were performed on a vertical machining system with a focus on spindle load measurement. The results show that the developed simulator estimates spindle load accurately despite input data noise and within the time restrictions occurring in real-time applications. All generated knowledge is stored and accessible for future simulator runs and finally, the system demonstrates its ability to extract value from all available data while reducing the raw data storage needs.

ACKNOWLEDGEMENTS

Although the completion of a PhD is credited to the author, in reality, it is the outcome of the author's work together with several people who play a key supportive role during the process.

First and foremost, I would like to thank my supervisor Paul Prickett for the hours/days he spent reviewing my work, helping me to avoid dead ends and being a friend during this long journey. Acknowledgement also goes to Rossi Setchi for her valued advice and support which enabled me to start the PhD and to find the required time to finish it. Long working hours would not be enough if my manager Andrew Hopkins hadn't provided me with the maximum possible flexibility. I would like to thank him together with my colleagues who treated my PhD as part of my work development.

The hours that I have worked to complete this work are time that I would normally spend with my family. I want to thank them for their understanding all these years and for the fact that they believed in me and that their patience was worth it.

Finally, I would like to mention the thousands of researchers and academics that built the world to which this thesis adds a small piece. Every thought and idea in this document is a continuation of what was already there and hopefully a trigger for future researchers.

ABSTRACT	I
ACKNOWLEDGEMENTS	II
1 INTRODUCTION	1
1.1 RESEARCH MOTIVATION.....	1
1.2 OBJECTIVE.....	3
1.3 THESIS STRUCTURE.....	3
2 LITERATURE REVIEW	5
2.1 DIGITAL MANUFACTURING	5
2.1.1 <i>Definition and history</i>	5
2.1.2 <i>Current trends – Digital twins</i>	9
2.1.3 <i>Digital twin simulation aspect</i>	12
2.1.4 <i>Digital twin data management aspect</i>	20
2.2 MANUFACTURING SIMULATION.....	21
2.2.1 <i>Common elements of simulators</i>	22
2.2.2 <i>Simulation model types</i>	24
2.3 MANUFACTURING DATA	28
2.4 LITERATURE REVIEW DISCUSSION.....	33
3 ARCHITECTURE OF THE SIMULATOR	35
3.1 SIMULATOR BOUNDARIES.....	35
3.2 PROPOSED SIMULATOR ARCHITECTURE.....	37
3.3 INPUT INTERFACE.....	41
3.4 SIMULATION ENGINE	48
3.4.1 <i>Simulation engine general characteristics</i>	48
3.4.2 <i>CNC process simulation engine</i>	54
3.4.2.1 Precision Selection	55
3.4.2.2 Mesh Generation	57
3.4.2.3 Virtual CNC Processing	62
3.4.2.4 Calculating derived data	65
3.5 LEARNING MODULE.....	68
3.5.1 <i>Learning module operation and characteristics</i>	69
3.5.2 <i>Learning module and simulation engine collaboration</i>	73
3.5.3 <i>1-sample-based learning</i>	75
3.5.4 <i>n-sample-based learning</i>	79
3.6 DATABASE.....	84

3.7	OUTPUT INTERFACE	90
3.8	ARCHITECTURE SUMMARY	96
4	APPLICATION IN VERTICAL MILLING	98
4.1	PROCESS SELECTION AND SYSTEM BOUNDARIES	99
4.2	INPUT INTERFACE AND DATA SOURCES	102
4.2.1	<i>CSV files</i>	102
4.2.2	<i>Part program and process setup</i>	104
4.2.3	<i>Source data pre-processing</i>	107
4.2.3.1	G-Code file pre-processing	107
4.2.3.2	CSV file pre-processing	111
4.2.3.3	Live data pre-processing	113
4.3	SIMULATOR DATABASE	115
4.4	MILLING SIMULATION ENGINE	122
4.4.1	<i>Simulation engine input data</i>	124
4.4.2	<i>Cutting tool mesh</i>	125
4.4.3	<i>Billet mesh</i>	128
4.4.4	<i>Machining simulation</i>	130
4.4.4.1	VMC process replication	130
4.4.4.2	Digital VMC derived data calculation	136
4.5	MILLING LEARNING MODULE	138
4.5.1	<i>ML model selection</i>	138
4.5.2	<i>Model training and lifetime management</i>	142
4.5.3	<i>Synchronisation for n-sample data models</i>	143
4.6	OUTPUT INTERFACE	146
4.6.1	<i>CSV report files</i>	146
4.6.2	<i>Simulator API for integration</i>	147
4.6.3	<i>Web API</i>	152
4.7	INTEGRATION INTO THE PHYSICAL WORLD	154
4.8	DEVELOPMENT TECHNOLOGIES	156
5	BENCHMARKING – RESULTS AND DISCUSSION	158
5.1	SIMULATION ACCURACY	159
5.2	KNOWLEDGE GENERATION	168
5.3	SIMULATOR SPEED	174
5.4	DATA REDUCTION	178
5.5	DATA UTILISATION	180
6	PROJECT DISCUSSION AND NEXT STEPS	182
7	CONCLUSION	187

7.1	RESEARCH CONTRIBUTION AND FUTURE WORK OVERVIEW.....	190
REFERENCES		193
APPENDIX A. VMC SIMULATOR (VERSION 0.65)		206
A.1	SOFTWARE INSTALLATION	206
A.2	VMC SIMULATOR DATABASE.....	207
A.3	LIBRARIES.....	209
A.4	SIMULATION FUNCTIONALITIES	212
A.5	LEARNING FUNCTIONALITY	216
APPENDIX B. JAVA CODE DOCUMENTATION SUMMARY		219
B.1	PERSISTENCE LAYER (MILLING-DATABASE).....	219
B.2	UTILITIES LAYER (MILLING-UTILS).....	220
B.3	GRAPHICAL USER INTERFACE – LOCAL (MILLING-VM)	225
B.4	GRAPHICAL USER INTERFACE – WEB APPLICATION (MILLING-TWIN).....	229

1 INTRODUCTION

During the last decades, the amount of data collected from manufacturing systems has increased exponentially. This has enabled a detailed digital representation and analysis of the production-related activities and has created a new virtual ecosystem in which each machine (and every other resource) operates with enhanced abilities. Simulation tools play a vital role in the creation of this ecosystem. However, their traditional way of operation is not enough to support current and future virtual ecosystems. This work proposes a new architecture and functionality of manufacturing simulation systems that can drive manufacturing digitalisation, take data utilisation to much higher levels and become the enabler for further developments in manufacturing operations and management.

1.1 RESEARCH MOTIVATION

This work is inspired by the concept of a virtual world where data is the raw material (Carriere-Swallow and Haksar 2019) and simulators can be the equivalent of machines that use this data to produce a higher value output (virtual product or service). Taking this approach makes it easier to point out bad data usage practices whose impact until recently was widely ignored (Data Utilization: Facts, Stats & IO Research.). The current practice of generating and storing Big Data without knowing which parts of it are useful or how it should be used, in the physical world would be the equivalent of using massive warehouses to store raw materials irrelevant to the manufacturing process or relevant to a process that adds low value and/or consumes the materials at low rates (in proportion to the available amount).

Industry 4.0, being or not being a revolution, has increased awareness around data and digitalisation and it has opened new routes towards improving manufacturing (Tamás and Illés 2016). This includes tackling the data-related issues identified above. However, taking new routes requires the development of new tools. Expanding on the data as raw material concept, to increase its utilisation rate, better, more efficient simulators (virtual machines) are needed. Although the core

meaning of simulation is still relevant, many current simulators are based on the logic of a previous era that makes them capable of only a narrow range of tasks. One could argue that similar to mass production machines, such simulators are fixed to specific input data that after a predefined calculation produce a predefined type of output with a focus on calculation speed and accuracy. However, modern manufacturing resources, in addition to updating their operators, must provide statistics to the managers, maintenance requirements to the relevant engineers, failure prognosis etc. It would be natural to have these tasks completed by the simulator, but the current simulation technology follows the aforementioned mass production logic. Digital twins filled the technology gap by employing a simulator for each need or requirement (Boschert and Rosen 2016)(Tao et al. 2019). This could be described as packing multiple machines to do one job which is not ideal in terms of efficiency. Moreover, data utilisation rates have improved at a very high computational cost and finally, the developed systems lack the flexibility required to embed them into the virtual ecosystem.

A redesign of the simulator architecture and the enhancement of its capabilities towards better data utilisation rate, simulator multitasking, and efficient usage of resources would boost the performance of digital twins. This will ultimately enable the virtual ecosystem to be more effective in the supervision and management of the physical one. The development of this new generation simulator must overcome the following challenges:

1. The simulator should be of a generic nature which allows for flexibility in implementation and should be modular when embedded into a digital twin to allow for both simulation of different machines performing a specific process and application of the same logic to a wide range of processes.
2. The simulator should operate in a mass personalisation way, meaning adapting the way it processes input data and producing different types of output depending on the requirements of the digital twin.
3. The simulation speed should be high enough to be in synch with the physical process.

4. Simulation accuracy and precision should satisfy process requirements (tolerances).
5. Ideally, all collected data should be utilised/processed at least once (high data utilisation) and only once (high processing efficiency).
6. Requirements for computing power and physical resources should be aligned with edge computing or ideally integrated computing capacity.
7. Knowledge generated by the simulator should be available and shareable.

A system that meets the above requirements would facilitate value extraction from collected data regardless of the source of it and enable the manufacturers to capitalise on the new technologies and practices that Industry 4.0 has generated. The next generation characterisation refers to the need for changing the current monolithic structure of simulators and the embedment of new data processing technologies as an essential part of their operation.

1.2 OBJECTIVE

This research aims to develop a new generation simulator architecture that can then be prototyped and tested. This architecture mitigates data management and utilisation issues that manufacturers currently face and contributes to a structured development of digital twins. This will then reduce the effort needed to digitalise productions and is an answer to the challenges presented in 1.1. More specifically the aim is to propose and report:

1. The elements of a next generation simulator architecture
2. The way these elements are integrated to develop the simulator.
3. The application of this architecture on a manufacturing process (prototype).
4. The advantages of using this architecture in a modern manufacturing process.

To support these aims the performance of the proposed architecture is demonstrated using a vertical machining centre as an example application.

1.3 THESIS STRUCTURE

The research work and results are presented using the following structure.

Chapter 1: Introduction to the current problems that industry faces and the field that this work will contribute to. Also, presentation of the thesis structure.

Chapter 2: A literature review of digitalised manufacturing technologies, modern manufacturing simulation tools and manufacturing Big Data processing and usage.

Chapter 3: Presentation of the next generation simulator architecture. Each element of the simulator is presented in detail along with its implementation guidance.

Chapter 4: Application of architecture in milling. The elements presented in Chapter 3 will be built around vertical milling process and verified in a 3-axis vertical machining centre (VMC) machine.

Chapter 5: Benchmarking of milling simulator. Reliability, accuracy, real-time proximity, data utilisation and other performance indicators are included in the assessment.

Chapter 6: Discussion of work results and suggestions for future work.

Chapter 7: Conclusions and sum up of work outcomes.

Appendices: Detailed technical information about the software and methods developed in the previous chapters.

2 LITERATURE REVIEW

To set the background of this research work this review begins with a brief description of the evolution and current state of digitalisation in manufacturing. Key technologies are identified, and further details are provided about relevant previous work. This includes a review of the current trends in digital twins which are the backbone of modern digitalisation, a more detailed study on simulation technologies and methods and how these are part of digital twins and finally a brief review of manufacturing data processing and management. Since simulation is a very broad field the review aims to indicate the current state of manufacturing simulation and the needs in this field. Where appropriate, descriptions and literature references for specific methods or technologies required to build the proposed simulator are provided where they are utilised in the relevant chapter.

2.1 DIGITAL MANUFACTURING

There are numerous definitions of the term 'manufacturing' or 'manufacture' including:

Manufacturing is the business of producing goods in large numbers (Cambridge English Dictionary. 2020).

Manufacturing is the business of making goods in large quantities in a factory (Macmillan Dictionary. 2020).

Manufacture is to make from raw materials by hand or by machinery (Merriam-Webster. 2020).

These definitions each imply that manufacturing is something that happens in the physical world and has a physical output. It therefore seems counterintuitive to use the term 'digital manufacturing' as digital is not normally physical. However, this term has evolved over many decades to reach its current meaning.

2.1.1 Definition and history

Computers have been part of manufacturing since the 1950s (Sanders 2012).

During the second half of the 20th century, new applications for computers were

discovered with concepts such as Computer Aided Design (CAD) (Sutherland 1963), Computer Aided Process Planning (CAPP)(Niebel 1965) and simulation programming (Gordon 1961) taking advantage of the increasing computing power and accessibility. Between 2000 and 2009, the term digital manufacturing became more popular and was used to describe a highly promising set of computer-based technologies. Reductions in product development times and costs, mass customization capability, increased product quality, and faster response to the market were identified as core benefits of using digital manufacturing (Chryssolouris et al. 2009). Already during that decade, the vision of a digital factory had been proposed. The idea, depicted in Figure 1, was that all of the computer-based tools supporting the planning and operation of a facility would be connected to a central database. This would allow for a better overview, coordination and control of the manufacturing process (Bracht and Masurat 2005)

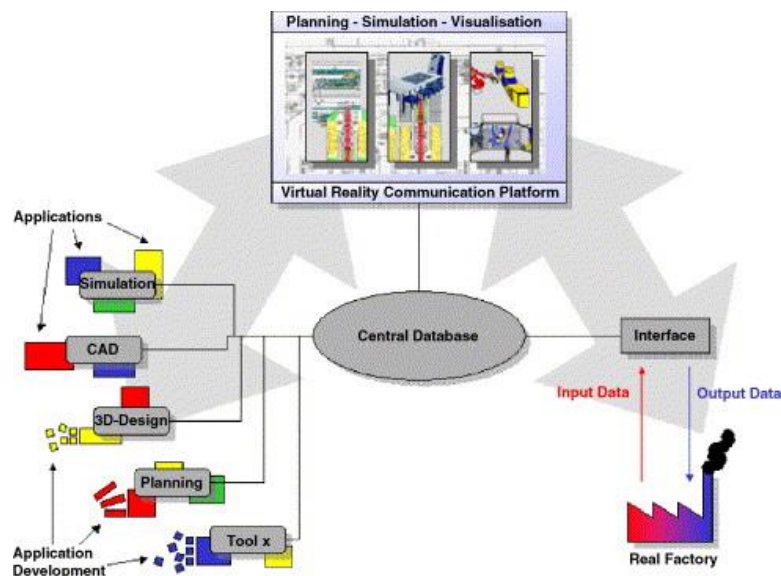


Figure 1 Early vision of the digital factory (Bracht and Masurat 2005)

At the beginning of the 2010 decade digital manufacturing development started to accelerate as the potential of web-based applications assisted by simulation tools able to perform complex tasks was being realised (Abdul Kadir et al. 2011). One of the most critical points in digital manufacturing history was in Hanover Messe 2011 when the Industrie 4.0 initiative began (Kagermann et al. 2011). It was not by luck that the initiative started in Germany. Manufacturing has been contributing more

than 20% to the German GDP (The World Bank 2021) and manufacturing activity at that time was steadily moving to lower-wage countries. High-wage countries had to become more efficient which required finding a new balance between economy of scale and economy of scope, planning for mass customisation with the usage of multidisciplinary skills and networking of the manufacturing resources (Brettel et al. 2014). The potential for growth in manufacturing and other sectors of the economy triggered further initiatives in Europe and around the World. Examples of these are Society 5.0, Smart Industry, Industrie du Futur, Industria Conectada 4.0, Manufacturing USA and Made in China 2025 (Proctor and Wilkins 2019). Digitalisation became part of national strategies with funding and new policies that accelerated developments in related technologies (Klitou et al. 2017).

A key element of digital manufacturing is the Internet of Things (IoT) and services which have been at the heart of Industry 4.0 since its initiation. The term IoT was introduced in 1999 (Rose et al. 2015) and was initially used for supply chain tracking (Ashton 2009). It gained popularity a decade later when innovations in cloud computing, datacentres, wireless communications and Machine to Machine (M2M) technologies formed the basis of IoT as we currently know it (Intel 2015). Networking everything became the enabler for technologies that have been disrupting business. This brought with it benefits such as: meeting individual customer requirements; business process flexibility; optimised decision taking; higher productivity and efficiency of resources; creation of opportunities in the service sector; response to an ageing workforce; better work-life balance (Henning et al. 2013).

In this new environment, the digital factory and the data that it was made of began to take shape. As centralized databases became more common in the early 2000s and because of the high volume of processed data they held, large organizations started placing portions of the data at local facilities (Harrington 2016). At the same time, ideas for the distribution of knowledge among the resources were also explored (Yan and Xue 2007). With the continuous exponential increase of data, this practice gained popularity and led to the adoption of distributed database architectures (Harrington 2016). The early vision of a digitalised factory with one

high-performance data server changed to the modern digitalised factory where the centralised database is hosted by cloud services and portions of this data are kept locally for faster processing. As a result, the distributed storage grew together with distributed processing which is referred as edge computing (Satyanarayanan 2017). Information and Communication Technology (ICT) developments over the same period made it possible to make process-related information available at any networked location and therefore enabled the connection of the physical production to the cloud (Adamson et al. 2017). This concept is shown in Figure 2.

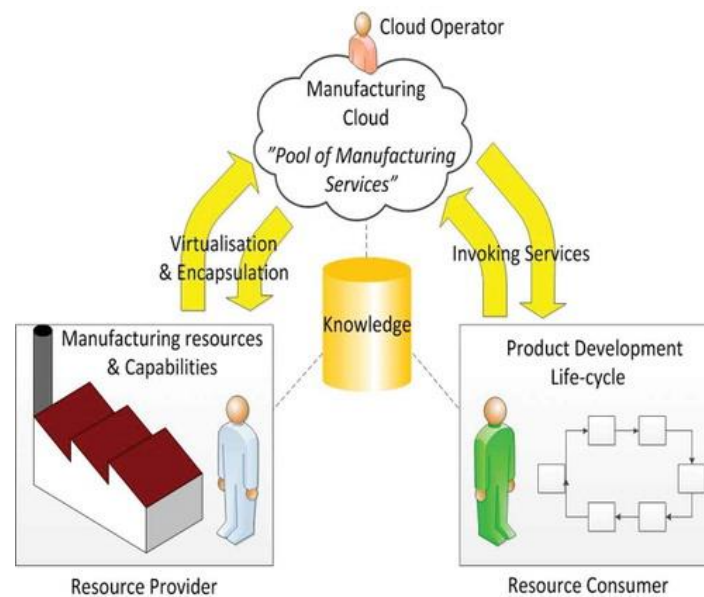


Figure 2 The Cloud Manufacturing concept (Adamson et al. 2017)

Connecting physical resources to the cloud led to the complete concept of cloud manufacturing, which was introduced in 2010 (Li and Mehnen 2013). Similar to the resource-sharing architecture of cloud computing (Mell et al. 2011) cloud manufacturing provides an application layer. This is where the user submits the task to execute or sends the design of the part to be produced. The cloud then searches in a pool of shared physical resources to find the most suitable ones to run the required processes (Li and Mehnen 2013). The user is not involved in tasks such as process design or line balancing as this is done by lower layers of the cloud that work with the cyber part of the physical machine (Wu et al. 2015). Since any pooled resource that could do the job is a candidate for being the suitable one, direct digital manufacturing evolved substituting the traditional manufacturing processes with additive layer manufacturing. This addition in cloud capabilities

further closes the gap between the customer needs and the manufacturing product and reduces the effort for process design for one-off products (Chen et al. 2015).

Due to technological advances in digital manufacturing, the mass customisation trend that began during the 1980s (Davis 1989) became a viable strategy with benefits for both product and service consumers (Fogliatto et al. 2012). The idea of product-centric control can be realised (Kärkkäinen et al. 2003) by allowing for per-product customisation controlled by highly flexible digital counterparts of the physical resources (Lyly-Yrjänäinen et al. 2016). Finally, the current digitalisation movement is marking a major turning point in history (Colombo et al. 2015) and will continue shaping the future as it is a critical part of today's competition (Sneider and Sternfels 2020).

2.1.2 Current trends – Digital twins

Industry 4.0 as defined by (Kagermann et al. 2011) generated new trends in manufacturing and accelerated developments in a range of technologies that have a direct impact on the value a company can generate (Baur and Wee 2015). Business models have evolved and higher levels of monitoring and control have improved resource management and therefore contributed to sustainable manufacturing (Carvalho et al. 2018). Figure 3 maps Industry 4.0 technologies with business value drivers.



Figure 3 The digital compass (Baur and Wee 2015)

Modern production systems have to cope with short product life cycles, mass customisation requirements and increasing competition in supply chain efficiency (Cohen et al. 2019). These needs have driven the evolution of digital manufacturing which in turn has produced three main areas of extensive research and development; product life cycle, smart factories, and value chain management (TWI 2020). This review will focus on the smart factory aspects of digital manufacturing as this is the area where the proposed simulator will be applied.

In a recent review of smart factory research, Strozzi et al. (2017) identified clusters of keywords by following the Systematic Literature Network Analysis approach. Each keyword represents a research subject and each cluster represents a group of closely related technologies. They found that smart manufacturing is directly connected to cloud technologies and production optimisation clusters. In other words, smart manufacturing is closely connected to decision-making (optimisation) and information sharing (cloud services). Figure 4 is an extract from a co-occurrence network. The node size corresponds to the number of occurrences of the keyword found in the literature examined by the authors. The links represent the most frequent literature references (link weights are not depicted).

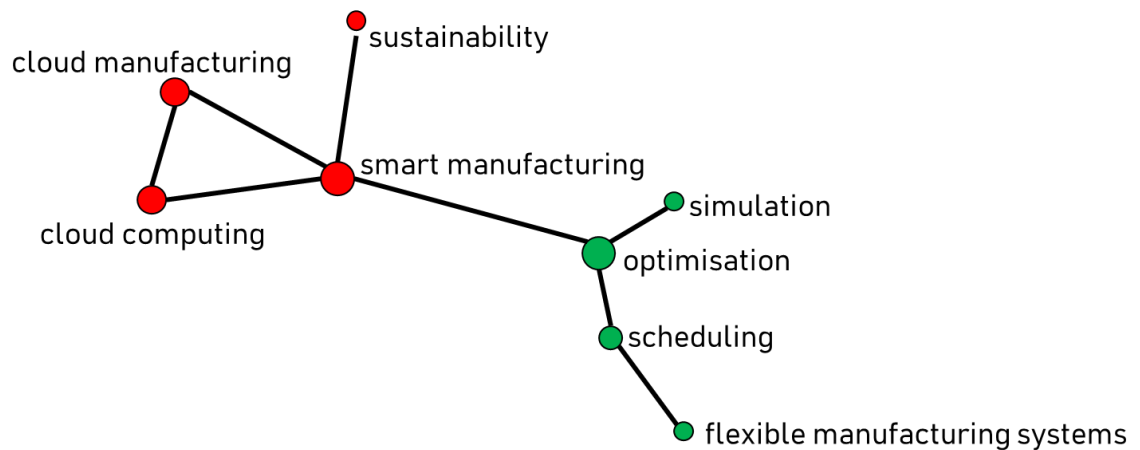


Figure 4 Network of smart manufacturing keyword in literature - adapted from (Strozzi et al. 2017)

As observed by Lu et al. (2020) digital twins connect any level of a physical entity (machine, person, factory etc.) with the cloud and allow for large gains in terms of operation efficiency and resource optimisation. Therefore, digital twins are systems that link the technologies of the smart manufacturing network in Figure 4.

From their first application as a concept in manufacturing in 2003 (Grieves 2015), through the NASA introduction almost a decade later (Glaessgen and Stargel 2012), and the exponential growth that followed (Tao et al. 2019) (Jones et al. 2020) and their strong presence within industrial internet of things projects (Costello and Omale 2019), digital twins have been closely related to smart processes and manufacturing digitalisation. In this context a digital twin can be an integration of the digital twins of subsystems (Kunath and Winkler 2018), it can be a simple virtual representation of a physical system that converts raw data to information, also called a digital shadow (Kritzinger et al. 2018), or a holistic solution that simulates a process, calculates optimum process parameters and controls the resources performing the process (Zheng et al. 2019). In a recent review, Jones et al. (2020) generated a list of digital twin characteristics to summarise current trends in digital twins. Table 1 shows this list of characteristics with a short description.

Table 1 Characteristics of digital twins. Adapted from (Jones et al. 2020)

Characteristic	Description
Physical Entity/Twin	The physical entity/twin that exists in the physical environment
Virtual Entity/Twin	The virtual entity/twin that exists in the virtual environment
Physical Environment	The environment within which the physical entity/twin exists
Virtual Environment	The environment within which the virtual entity/twin exists
State	The measured values for all parameters corresponding to the physical/virtual entity/twin and its environment
Metrology	The act of measuring the state of the physical/virtual entity/twin
Realisation	The act of changing the state of the physical/virtual entity/twin
Twinning	The act of synchronising the states of the physical and virtual entity/twin
Twinning Rate	The rate at which twinning occurs
Physical-to-Virtual Connection/ Twinning	The data connections/process of measuring the state of the physical entity/twin/environment and realising that state in the virtual entity/twin/environment
Virtual-to-Physical Connection/ Twinning	The data connections/process of measuring the state of the virtual entity/twin/environment and realising that state in the physical entity/twin/environment
Physical Processes	The processes within which the physical entity/twin is engaged, and/or the processes acting with or upon the physical entity/twin
Virtual Processes	The processes within which the virtual entity/twin is engaged, and/or the processes acting with or upon the virtual entity/twin

In the same paper, it was found that 36% of the reviewed publications were related to simulation, modelling, and optimisation (the most popular theme) and 33% were related to data management (the second most popular theme). These percentages are indicative of the fields that are most critical to digital twin related technologies and are the two main areas the current work contributes to.

2.1.3 Digital twin simulation aspect

Clearly associated with the services dimension, the simulations are enacted within and/or upon the virtual entity but must properly represent the physical entity if they are to be meaningful and robust. In this context, digital twins have been described as the modern form of simulation (Boschert and Rosen 2016). But this claim has been challenged since digital twins can control the process, interact with other systems, and their activity is bidirectionally connected with the activities in the physical world (Shao et al. 2019). A multidimensional approach is more suitable for describing the nature of such systems and the technologies involved. At the shop floor level, Tao and Zhang (2017) suggested a 4 dimension model which consists of: the physical shop floor, the virtual shop floor, the service system supporting both physical and virtual floors and the data generated and processed. The same authors in a later work added a fifth dimension, the connection between the physical and virtual worlds (Tao et al. 2019).

Simulation lies mostly in the service sub-system of the digital twin. Each digital twin depends on one or more simulators that must operate in a synchronous way to process the live data received from the physical system and supply other systems with real-time results (Brandstetter and Wehrstedt 2018). Since many services are involved in the operation of the digital twin and since a digital twin may be the

integration of multiple digital twins, the final system can reach very high levels of complexity (Kunath and Winkler 2018). Ciavotta et al. (2020) proposed a decomposition of manufacturing digital services into microservices to allow for lower-cost development and management of distributed systems. A microservice is an independently deployed, scaled and tested application with a single easily understood responsibility (Thönes 2015). Ciavotta et al. (2020) also pointed out the importance of low- and high-level communications among the microservices because of their role in updating the simulation models with real-time data and therefore maintaining the reliability of the digital twin.

The simulators of digital twin implementations that are reported in the literature are typically bespoke systems with lots of hardcoded elements. These are very much application specific and there are certainly thousands of implementations. This review section will therefore be restricted to some indicative recent examples from CNC-related digitalisation. In these examples, a brief description of the simulation model is provided along with necessary notes about the system it is embedded in and/or deployed data processing techniques.

Starting at the product design stage Zhou et al. (2021) show that the alignment between product design and manufacture can be improved using a digital twin enabled optimisation. They focus on the enhancement of product performance achieved by improved manufacturing. The developed tool supports the offline optimisation of cutting processes before actual machining. The data from machining trials is then used to train a reinforcement learning model. The trained model can then be used to provide feedback to improve the milling of future products, which in this example were centrifugal impellers.

Balderas et al. (2021) reported an approach intended to enable enhanced product design for manufacture of printed circuit boards (PCB) using digital twin based optimisations. This uses process setup data as inputs into a simulator that applies metaheuristics to calculate the optimum way to drill the PCB. The results consider the relationship between tool changes, tool path length and machining time in setting up the optimum path. Although relatively simple in nature there is a clear

benefit in better managing the drilling process. The intention is that future products can be better manufactured using the knowledge embedded within this digital twin.

At the start of the CNC-based manufacturing process, Zhu et al. (2021) demonstrated the potential for enhancing workpiece setup and orientation of thin-walled parts. They used deployed sensors, including an operator-mounted head camera, to generate machine tool coordinates and machined part positioning as inputs. The machining process is simulated both with geometrical calculations and using the ANSYS FE simulation package. This combination calculates thin wall part deformation, and the results can be used by the operator to optimise the initial toolpath and enhance the process for subsequent parts.

In a mostly theoretical study, the performance and health of a generic machine tool are examined based on its predicted and calculated stiffness (Zhao and Sun 2021). The approach deploys a method using accelerometer samples to determine machine tool stiffness, which is seen as an indication of its overall state. They use theoretical FEA models for predicting machine tool stiffness and Auto Regressive Moving Average (ARMA) model analysis of the measured vibrations to establish its current health and predict the future condition of the machine. The work does not investigate cutting-related phenomena and requires further development if it is to be practically applied.

Moving forward Akintseva et al. (2021) considered an approach to the representation of a machining process within a digital twin. They examined the enactment of a cylindrical grinding operation using data associated with the machine setup and parameters. Then they used the NC program together with process equations to calculate the expected theoretical behaviour of the machine throughout the process. The information is intended to support the reliable and predictable manufacture of future parts.

Aiming to demonstrate how machining processes could be improved Zhao et al. (2021) presented an approach aimed at the optimisation of CNC milling. This was developed using ontologies to model critical process parameters and configure simulations. The simulator fused cutting process data acquired from a spindle load sensor and acquired machine parameters to produce results based on a heuristics

algorithm that minimises the carbon footprint of the process. It was based upon a very limited milling operation and relates mainly to the optimisation of a process enacted on a virtual machine tool. The researchers correctly identify that the link to the actual machine tool needs a great deal more attention if the approach is to function in real life.

In a similar vein, Heo and Yoo (2021) applied the concept of a digital twin to optimise CNC machining in a repetitive mass production environment. This work utilised an application of the generic Manufacturing Digital Twin for Dedicated Equipment (MDT4DE) framework. The technique again required the capture of spindle loads, using an installed sensor, during the machining cycle. This was then synchronised with data collected from previous cycles, within a simulator that post-processes it with fixed and per-case mathematical methods. The result of the simulation provided updated settings for the cutting parameters for every step of an NC program. The focus of this work was on machining cycle times, and the authors noted the need for more research if the part features and cutting tool management are to be considered.

An indication of the process improvement and management potential of a digital twin application is provided in the context of addressing the issues of chatter within a CNC milling operation (Afazov and Scrimieri 2020). The stated aim of the paper is to detail a chatter model that can be integrated into a digital twin. The model is developed to use measured cutting forces applied to the cutting tool. A simulator then uses Fast Fourier Transform to calculate if chatter is likely. If chatter is confirmed, then the cutting parameters are altered to mitigate the problem. Although the paper suggests that this model could be integrated into a digital twin based approach this is not achieved in this work.

In the work of Liu et al. (2021) the application of a digital twin to enable real-time monitoring of a machining process by an operator has been considered. They compare physical machine data with ideal theoretical data representing the steps required to manufacture the product to visually present a machining operation. The developed digital twin does not use any simulation engine, so it is actually a state-of-the-art example of a digital shadow. The results of the comparison are displayed

in an enhanced form through augmented reality. The methodology provides real-time representations of a machining process but does not attempt to consider how it can be used to improve or control the process.

To complete the simulation in digital twin trends section of this review, the work of Friederich et al. (2022) should be mentioned. Their work is one of the first that attempts to identify the elements of the simulation within a digital twin. They recognise that this is needed in order to manage the increasingly complex system development. In addition, they talk about the knowledge extraction part which should reflect the developments in machine learning fields. The work however is an adaptation of modern data processing into a lab-based production scenario and lacks the depth and breadth required to embed engineering knowledge into the system and assist in the development of scalable production digital systems.

To understand the evolution of the data acquisition, sharing and simulation aspects of the digital twin, literature examples from earlier years of the technology are now presented. These are included as an indication of the progression towards the current state of digital twin technology and ambition. This review again starts at the product design stage and moves through the configuration and operation of the manufacturing system.

Schroeder et al. (2016) investigated digital twin based product optimisation. They proposed a system that could be used to capture and share the data from a physical device (in this case an industrial valve) during its use. This data was then fed into an Automation Modelling Language (AML) model to further develop the product and so demonstrate the potential of the digital twin to assist in product improvements. It should be noted that this valve was a relatively simple product with few parts.

The benefits of applying a digital twin approach to the development of an enhanced product design and monitoring process was also reported in the context of an element of a jet (Iglesias et al. 2017). The application of CAD and analysis to the development of a diverter tile was combined with experimental and installed performance data. The digital twin was then used to merge the information from three different numerical models to assess the installed system's state,

performance, and safety. This is an important stage in the development of the use of a digital twin in the ongoing development of improved products.

In the context of the deployment of a CNC-based manufacturing operation Lee and Wu (2015) proposed a methodology to create a virtual machine tool that can be adapted for multiple applications. The focus was on the acquisition of machining process information from the CNC controller data. They connect the virtual with the physical machine to create a flexible modular design. The data obtained from the connection with the controller is used to visually represent the process.

As an example of the complexity of the data-related challenges faced in the context of CNC milling, Stavropoulos et al. (2016) investigated the development and application of a tool wear model, based upon established theoretical representations of tool wear. Tool wear is investigated using experimental data and associated analytics together with offline experimental measurements to predict tool wear. The very limited single-tool basis of this model demonstrates the huge challenge faced in this arena and suggests that a more suitable approach may be needed if this is to be applied in real time.

In a similar context Soori et al. (2016) in one of a series of papers developed a virtual machine environment to calculate tool deflection and milling machining forces with available theoretical methods. The simulation results are used to optimise the programmed G-Codes to compensate for tool deflection to produce a better surface finish and better tool utilisation. This work is not about digital twins but suggests that the optimisation methodology could be applied to live cyber-physical systems. However, it should be noted that the “cost” of these improvements is a potentially unacceptable increase in cycle time, which suggests that real-life implementations requiring multi-factor optimisation will be much more challenging.

Lechevalier et al. (2015) developed a virtual milling machine that uses theoretical equations to simulate a milling process and calculate tool position and cutting forces. The machine is used to create monitoring data which is however not fused or adapted to the live data produced by the physical system. It does however

explore the important concept of data sharing and standardisation, in an early application of the MT Connect standard.

In a more practical application, Cai et al. (2017) recorded spindle power consumption and tool acceleration from a milling machine performing various cuts. The data from multiple cutting operations (using the same cutter) was then fused and fed to a linear model that estimates surface roughness. The model was constantly updated but in the limited context of the cutting operations being performed. The authors recognise the challenges arising from developing this approach for deployment upon different machine tools performing numerous different operations.

The final set of papers reviewed here relates to the wider potential for plant and factory management offered by the deployment of digital twins and the role of simulation in these early stages. Fysikopoulos et al. (2015) developed a tool consisting of multiple simulators using physical machine data and theoretical formulas to initialise a virtual copy of the machine. Then the virtual machine adapts (offline) to data produced by the physical system and the simulation model is used to tune the production setup in a more energy, cost, and waste-efficient way. This raises the interesting possibility of the use of data acquired to monitor tool condition (for example spindle load) in the wider context of the energy utilised in the process. However, the case study in this paper relates to the machining of a single hole, which is very far from a realistic machining problem.

With similar objectives, Choi et al. (2017) addressed the challenges associated with the connection of manufacturing functions to enable the control of production processes in real time. The basis of this approach was the definition of a unified environment of so called cyber-physical systems that were able to gather all data from shop floor sensors and share it using the cloud. The resulting smart manufacturing system allowed operators and managers to view the state of the production.

Schluse and Rossmann (2016) presented the concept of a so-called “experimentable” digital twin and considered how the information developed within a simulation model could enhance the subsequent operation of the plant. Based on

the modelling of advanced robotic systems a virtual test bed was proposed, enabling robots to be inserted into different applications, under the guidance of experts. Once deployed as a physical system the data is fed into the model and manipulated, and the result is communicated to the physical system. This is one of the early publications that identify the potential of integrating simulators as independent modules in the virtual world.

Weyer et al. (2016) proposed the integration of different simulation models using a database within the cyber-physical model of the shop floor. The aim was to make all information and tools available to all stages of production including the virtual commissioning of manufacturing lines. They explored the mitigation of production issues in the context of part of an automotive assembly plant.

Finally, from an integration point of view, Gabor et al. (2016) proposed that the simulator should not be an inseparable part of a cognitive system but an independent system that provides input to the system. Their work is a clear indication of how the shift from traditional simulation systems to digital twin technologies has introduced new needs that cannot be covered with traditional system designs.

From the above examples, it can be concluded that in earlier stages digital twins were primarily used to group and present sensor data produced in an identified process by the physical system. The simulation aspect was either a theoretical capability or a simple model triggering a reaction of the virtual system which could potentially be used to moderate the physical accordingly. On the other hand, the latest digital twin implementations have a more sophisticated simulation aspect. The highly bespoke nature of early implementations is still there but modularity in the digital twin subsystems is more frequent as external simulation packages are embedded.

This short review shows the need to standardise a simulator's architecture based on the modern needs of digital environments. As newer digital twin implementations tend to have a similar general architecture the next step is to specify the corresponding architecture of a simulator that is portable to multiple

systems and compatible with the design and the nature of the system that it is meant to be embedded to.

2.1.4 Digital twin data management aspect

The current understanding of digital twins is based on the new opportunities for data gathering from the physical world and the technologies allowing for the exchange of data between the physical and digital worlds (Schleich et al. 2017). In 2.1.3 the examples presented use one or more simulation modules to process big amounts of raw data and provide the desired output. However, apart from cases like Zhou et al. (2021) where the model is trained (and therefore retains training results) or Fysikopoulos et al. (2015) where the developed simulators have internal memory to store adaptation parameters, generally, there is no focus on what information is retained and/or reused. These issues are addressed in this section.

The concept of digital twin self-evolution has been applied since 2011 (Tuegel et al. 2011) and is presented as an important part of digital twins by Tao et al. (2018a). The latter however refers specifically to deep learning neural networks, and it only presents the challenges that the simulation models must overcome to enable a self-adaptation process.

Rathore et al. (2021) go a step further in their review and consider that simulation model evolution is a digital twin characteristic. The literature they base their report upon however is only related to neural networks. Their extended reporting of the work of Xu et al. (2019) as an example of self-evolution actually relates to a deep learning model which falls into the Tao et al. (2018a) perception of digital twin self-evolution.

Kong et al. (2021) reported a gap in digital twin internal data management and considered the potential for improvement in the overall digital twin performance. They proposed an ontology-based model to represent the data hierarchically. This represented a cycle starting with pre-processing, then transforming to the needed format and finally passing to top-level modules. Although the application is on a neural network-based model, it is one of the few works that report internal data management. Focusing on digital twin internal data management should not be confused with works such as Zhuang et al. (2021) that process physical system data

to update the virtual but do not report usage and retention for self-evolution or other internal processes.

2.2 MANUFACTURING SIMULATION

In the previous sections simulation was described as a module of a digital twin that processes inputs and passes outputs to system visualisation modules. However, simulation has a much wider meaning and application and has been an essential part of manufacturing for several decades. Regardless of model types and methods, production simulation on a computing system can be cost-effective and allow for quick assessment of production configurations. The evolution of simulation software will continue driving manufacturing efficiency (Mourtzis et al. 2014). The breadth of simulation technology by far exceeds the field of manufacturing, however, the context of this thesis restricts this review to within the manufacturing field unless there are specific reasons for considering a wider basis. In this context, Mourtzis (2019) continued the review undertaken earlier (Mourtzis et al. 2014) and analysed 12 categories of simulation tools (shown in Table 2).

Table 2 Product and production line simulation tool categories, adapted from Mourtzis (2019)

Computer Aided Design
Computer Aided Manufacturing
Computer Aided Process Planning
Digital Mock Up
Material Flow Simulation
Process Simulation
Layout Planning Simulation
Ergonomics Simulation
Manufacturing Execution Systems
Supervisory Control and Data Acquisition
Supply Chain Simulation
Design and Planning of Manufacturing Networks

Digitalisation in manufacturing has created a trend in the ways that systems are being simulated. William (2020) in a review identifies ten dominant simulation approaches since 2014. Figure 5 shows these approaches which are complimentary to each other and can be integrated into a simulation system of the Industry 4.0 era.

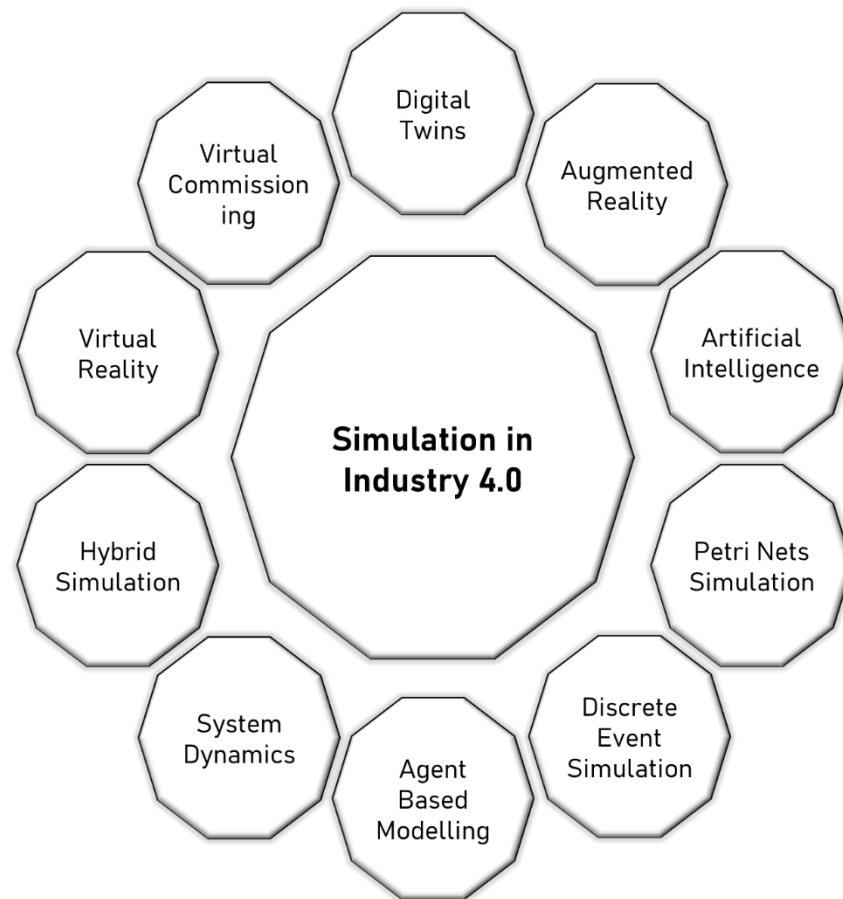


Figure 5 Simulation-based approaches in the context of Industry 4.0. Adapted from William (2020).

This research project focuses on developing a simulator architecture for manufacturing process simulation (see Table 2) that uses discrete event simulation and artificial intelligence in digital twins (see Figure 5). A review of common elements among simulators follows and describes the foundation of the developed architecture.

2.2.1 Common elements of simulators

Further to the top-level categorisations, simulation systems throughout their evolution have been sharing common elements, regardless of the application. Starting from the simulation model (which is how the simulation engine runs) there are two main types, discrete event and continuous (Law 2015). Roberts and Pegden (2017) refer to a third type, activity-based simulation, which has many similarities to event-based but is examining the activities taking place after every time step. Both of these papers recognise that discrete event simulation is the technique used in most cases. It should be mentioned that the categorisation assists in

understanding simulation methods but in practice, there are many hybrid models such as the ones described in Onggo et al. (2019). In discrete event simulation, the model depends on system state variables that are updated by the input data. This is the basic interface used by the model to receive data from other systems or directly from the user. The variables are then processed and output variables (results) are calculated for the corresponding timestep (Banks et al. 2005; Law 2015).

From a simulator architecture perspective, early works consider the graphical interface of a simulation system as an essential part of its architecture (Iwata et al. 1995; Rohrer 2000). A graphical interface speeds up model development, makes debugging of logical errors easier, communicates the assumptions and results in an easy-to-understand way and finally builds confidence that the model behaves as expected. Roberts and Pegden (2017) explain that before animation was used in simulation systems the results were in the form of textual reports. Finding model faults and establishing a basic level of reliability took much longer and was considerably more complicated.

With the advent of digital twins, data interfaces became a key element of a simulation system. The need for communication between multiple simulation models internally (Boschert and Rosen 2016) or external machine-to-machine communications (Chen and Lien 2014; Bao et al. 2019) put pressure on the developers to standardise the way data is communicated. OPC UA (OPC Foundation. 2021) and MTConnect (MTConnect. 2018) are two main examples of the effort to standardise communications at a machine-to-machine level.

Internal data management of modern simulation systems is discussed in section 2.1.4 and data outside the simulation system in section 2.3 of this thesis. However, it is worth mentioning that from a simulation evolution perspective, well before Industry 4.0, researchers and system developers realised the importance of structured and automated data management (Robertson and Perera 2002). The authors predict that simulators are meant to be integrated into the systems being simulated and therefore the way that data is provided should allow for automatic input, processing, and output from the simulation model. Almost a decade later a

database connected to the simulator was proposed as an architecture to mitigate the simulation integration and data exchange issues (Boulonne et al. 2010).

Nowadays, data modelling is essential in complex systems to deal with the wide range of inputs-outputs and the number of subsystems communicating. Kim et al. (2017; 2019) in a series of publications, demonstrate how the data model is different from the simulation model and in the latter work propose the steps to develop both models in parallel and integrate them into the simulation system.

Figure 6 summarises the elements that a typical simulation system is made of.

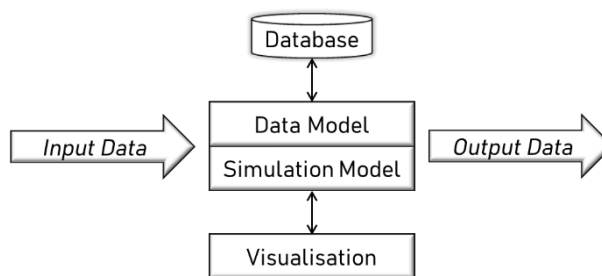


Figure 6 Common elements of simulation systems

2.2.2 Simulation model types

The simulation model is the core structure of a simulation system and typically the element that makes it unique. As a first step in developing a simulation system one needs to specify the boundaries and the scale of simulation (Banks et al. 2005; Law 2015). There are three levels of simulation that are covered by different simulation tools.

1. At the factory or supply chain level, tools like:
Simio (Simulation, Production Planning and Scheduling Software | Simio. 2022), Plant Simulation (Plant Simulation and Throughput Optimization | Siemens Digital Industries Software. 2021) and WITNESS (WITNESS Simulation Modeling Software | Lanner. 2021) can be used to create a factory level model that simulates material flows and assists in production management and in assessing the risks if material supply is disrupted or a critical resource fails.
2. At the machine level, tools like:
Process Simulate (Robotics and Automation Simulation | Siemens Digital

Industries Software. 2021),
RobotDK (Simulator for industrial robots and offline programming -
RoboDK. 2021) and
PowerMill (PowerMill | 5-Axis CAM Software | 5-Axis Machining | Autodesk.
2021)

can simulate the process with different parameters, program the relevant machine or sub-systems, provide various levels of detail about the final product characteristics and predict machine behaviour or risks during the process.

3. At the process level, it is typical that the simulation software companies have suites with a wide range of tools to cover every aspect of designing a system and simulating it at a high level of detail. Some good company examples are:

ANSYS (Engineering Simulation Software | Ansys Products. 2021), Dassault Systèmes (Simulation software - SIMULIA by Dassault Systèmes®. 2022) and Mathworks (MATLAB. 2022).

As research evolves there are always new bespoke tools being developed that are using dedicated libraries. Examples are:

- SystemC for C++ (SystemC Community. 2021),
- Desmo-J for Java (DESMO-J. 2017), and
- SimPy for Python (Overview — SimPy 4.0.2.dev1+g2973dbe documentation. 2021).

It is also frequent, especially for higher TRL level projects, to integrate custom programs with existing tools such as Java classes with AnyLogic libraries and interfaces (Advanced Modeling with Java | AnyLogic Help. 2021).

Looking into machining process simulation, Lorong et al. (2006) demonstrate the benefits of combining simulation models representing geometrical characteristics and simulation models representing physical properties. The former identify whether and how the cutting tool is removing material and the latter solves the physics-related equations that provide information about forces, heat or other physical phenomena. Zhang et al. (2011) extended this work by adding more

geometrical simulation model types as illustrated in Figure 7. Figure 7 is a hierarchical graph showing geometrical simulation categories and subcategories.

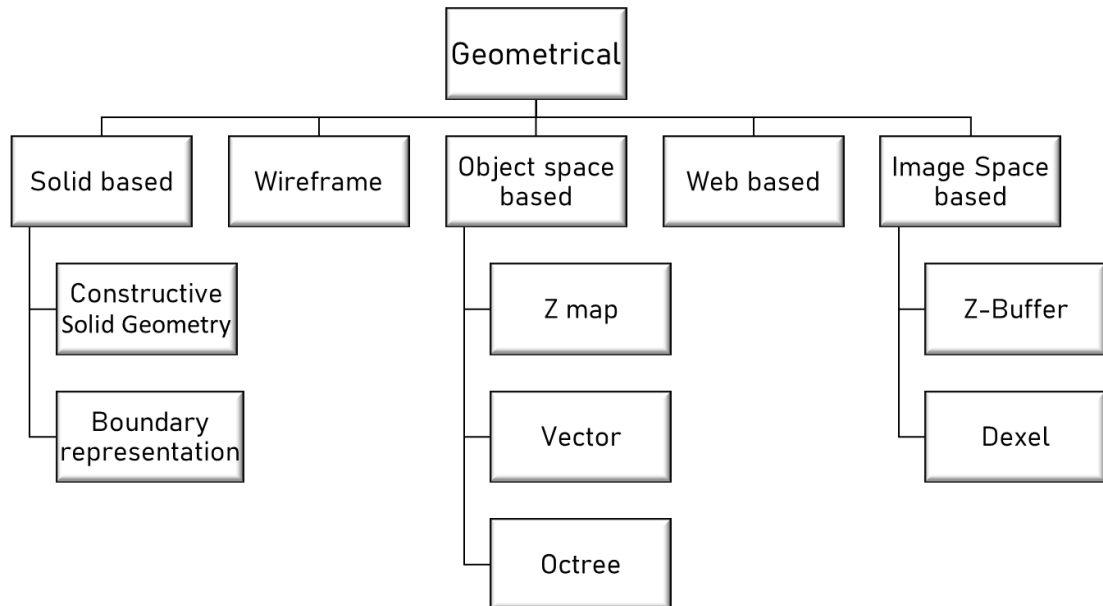


Figure 7 Geometrical simulation model categories and subcategories. Based on the work of Zhang et al. (2011)

During the last 30 years, all of the model types shown in Figure 7 have gone through periods of popularity or are used in new applications. This includes dexel models, which are one of the most consistently used until today. Dexels were introduced by Van Hook (1986) for computer 3D graphics as a low computing cost solution for creating virtual multicolour volumes. A dexel is a rectangular cuboid that is specified by its X and Y coordinate, the minimum Z coordinate (near Z), the maximum Z coordinate (far Z), a scalar representing its colour, and a pointer describing the Z axis distance until the next dexel. The versatility of this method, suitability for volume subtraction processes and ability to run efficiently on parallel processing systems made them popular in CNC machining simulations. Some application examples are:

- in milling (Joy and Feng 2017; Evgenii et al. 2018; Cao et al. 2020; Inui et al. 2020; Wang et al. 2020; Denkena et al. 2021; Röck 2021),
- in drilling (Meyer et al. 2021),

- in grinding (Siebrecht et al. 2014),
- in turning (Toubhans et al. 2021),
- in additive manufacturing (He et al. 2017; Böß et al. 2021) and
- in hybrid additive-subtractive processes (Sun et al. 2018).

In the same 30-year period, a key method in physics-related simulation has been the finite element method (FEM), especially in materials-related engineering (Belytschko et al. 2009). A search in Scopus database for finite element simulation (exact query: "TITLE-ABS-KEY(finite element simulation) AND (LIMIT-TO (SUBJAREA,"ENGI"))") returned 160k results while only the word simulation returns 2 million (exact query: "TITLE-ABS-KEY(simulation) AND (LIMIT-TO (SUBJAREA,"ENGI"))") meaning that 8% of all simulation models have some relation with finite elements, a significant percentage considering the wide range of simulation models and applications.

Finite elements analysis typically follows 4 steps (Chandrupatla and Tirupathi 2006; FEM. 2013; Bathe 2014).

1. Domain discretization or subdivision
2. Interpolation functions selection
3. System of equations formation
4. Solution

Finite element analysis can be used to develop very accurate simulations, but high resolution comes at a cost of high computing power requirements. In practice, finite element analysis is too slow for real-time applications since a typical solver needs to invert the stiffness matrix which in terms of computing has a quadratic or higher time complexity growth (Zhang and Gu 2020). As a result, compromises have to be made in order to develop a typical finite-element-based model that can run in real time (Marinkovic and Zehn 2019).

A representative example of a real-time application for the finite element analysis is the technique presented by Knezevic (2018) where a coarse model is used to catch up with sensor data. Since high accuracy is desirable in the majority of simulation applications, a few models have been proposed to bridge the gap

between offline and real time, avoiding the computation time problem. Hinchy et al. (2020) generated a finite-element-based model of bending process that runs every time the machine is about to process a batch of new parts. The model predicts the results, then the process begins and the operator can compare the simulation results with the actual part.

Other authors use finite element analysis to train a neural-network-based model initially offline (Zhang and Gu 2020; Seventekidis and Giagopoulos 2021). Then the neural-network-based model is used in parallel to the physical system to provide quick and accurate results for the physical system. The slow finite element analysis model may either stop running after initialisation or continue running in the background improving further the neural network. In the case of offline machining models, there have been numerous studies using the finite elements method in both 2- and 3-dimension problems. It is worth mentioning that the common modelling approach is to analyse orthogonal machining configurations since this simplifies the analysis and at the same time, it applies to a range of processes such as turning, milling, drilling etc. (Sadeghifar et al. 2018). Overall, simulation based on finite elements, despite the accuracy in results and the valuable information it can provide, it is not suited to the aims of this research project since it cannot support the real-time, flexible nature of a digital twin.

2.3 MANUFACTURING DATA

Manufacturing has historically been evolving in parallel with the increase in data variety, volume and complexity (Tao et al. 2018b). In a typical manufacturing data lifecycle the following steps are followed (Siddiqa et al. 2016; Tao et al. 2018b; Ren et al. 2019; Choudhary et al. 2022)

1. Raw data is collected from the data sources. This can be from a machine, a sensor, a human observation etc.
2. Raw data is transferred to a local or cloud-based storage system.
3. Raw data is pre-processed, by removing or repairing problematic samples and reducing its volume by removing duplicates or deleting non-usable samples (e.g., machine standby values).

4. Clean data is processed to extract its value. This could be extracting useful information, producing metadata to feed other systems, applying data analytics etc.
5. Analysis results are communicated (operationalisation).
 - a. Data processing results are converted to forms that can be visualised through human-machine interfaces (e.g., graphs or performance indicators).
 - b. Data processing results are fed to other higher-level systems through machine-to-machine interfaces (e.g., statistical data feeding process or production management systems)
6. Analysis results are maintained.
 - a. The models that were developed to process the raw data are verified with newly generated datasets
 - b. Steps 1-5 are retaken to improve models that don't perform well or to develop new ones.

Regarding the end of life of data, literature is mostly concerned about the destruction methods that follow legislations such as General Data Protection Regulation (General Data Protection Regulation. 2018) and there is no standard on when manufacturing data can be characterised as non-useful and be deleted.

Despite being practically weightless (Edwin Cartlidge 2010) data requires storage hardware, managing software, and continuous maintenance in order to create value. The average yearly cost range of a data centre is between \$10 million and \$25 million (How Much Does Running A Private Data Center Cost Per Year. 2020). It is also estimated that the cost for on-premises storage per TB per month is roughly \$32 (Donovan 2018) This is for hardware, software, disaster recovery, continuous power supplies, networking, ongoing maintenance of applications and infrastructure, air conditioning, property and sales tax, and labour costs. Although data storage costs drop over time the rate of cost reduction is slowing down leading to minimal if any benefits for the end user (Rosenthal et al. 2012). On the environmental side, data centres have high energy consumption (Yao et al. 2014), which in turn induces a high CO₂ footprint (Avgerinou et al. 2017). It is indicative

that since the IT sector accounts for 2% of total CO₂ emissions, the EU has created a policy to improve sector energy efficiency with a particular focus on data centres (*EU Code of Conduct on Data Centre Energy Efficiency 2016*). Therefore, collecting and storing data that is not being used has both an economic cost and a significant environmental impact.

Due to the different data storage needs of each application, there is a wide variety of Database Management Systems (DBMS). However, two main database technologies dominate the data storage market. Figure 8 shows the list of the most popular DBMS. The ranking score is a measure of website mentions, search frequency, technical discussion frequency, current job offers, professional network profiles and social network relevance. Statista (2022) considers this score as representative of the popularity of each database system.

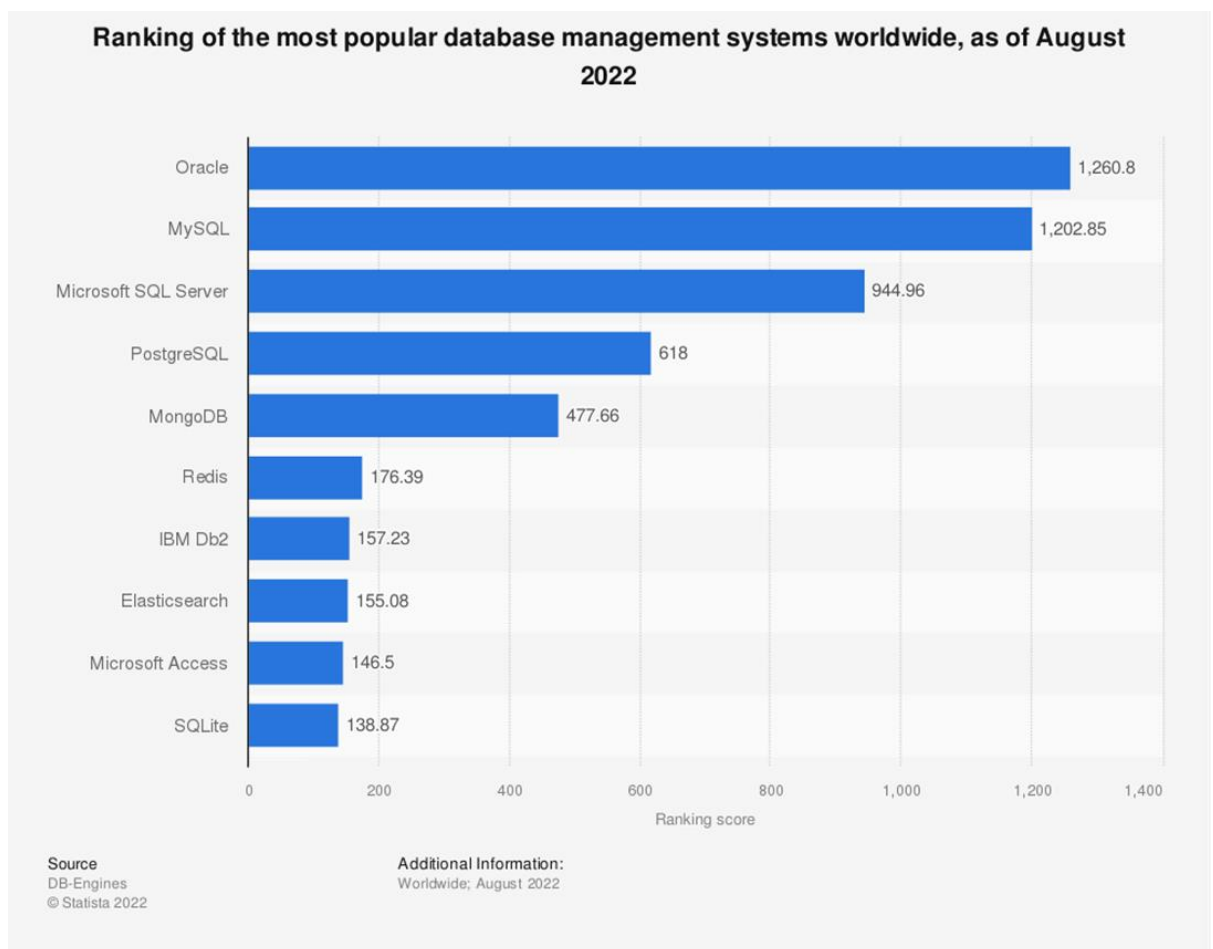


Figure 8 Ranking of DBMS systems (Statista 2022)

From Figure 8 it becomes clear that relational database management systems (RDBMS) dominate the market since only MongoDB and Redis are of a different type. It is also clear that there is no one system that fits all. Taking as an example SQLite which is used for the development of the simulator in section 4.3 it is a good solution for edge applications without database administration and for cases where the database can be copied and shared as a file (SQLite Home Page. 2022). At the same time, the syntax is very similar to MySQL making a migration to MySQL easier. MySQL however does not have the portability benefits of SQLite and it requires administration. In a more systematic approach, the work of Sahatqija et al. (2018) compares the various types of DBMS systems and provides some useful insight. RDBMS are very efficient in accessing structured information and data tables can be of gigabyte size without significant performance issues. The database schema requirement 'forces' the developers to follow specific rules and therefore extra safety is provided in terms of data integrity. A key benefit of RDBMS is data consistency when multiple users (or simulator elements) access the same field at the same time. Finally, generating reports from a RDBMS is straightforward since simple structured query language (SQL) queries can quickly aggregate, and report stored data. The second category is non-relational database management systems (NoSQL) which is an umbrella term for different types of database technologies. NoSQL systems do not require the definition of a schema and therefore they are more flexible in the way that the data is stored. Depending on the application they offer great scalability abilities, high speeds, and they can store unformatted complex data without the need for pre-processing.

In practice, there is no comparison between RDBMS and NoSQL. RDBMS is the first consideration for an application and if there are unique characteristics such as Big Data unstructured datasets or high-speed, multiuser access that require different technologies then a solution in the NoSQL list of databases would be sought.

Cloud storage offers reliability by keeping data fault free and available, is high-performance for large datasets and has a lower cost per terabyte because of resource sharing and balancing (Mansouri et al. 2017; Gajjam and Gunasekhar 2021). Cloud, however, is best for long-cycle, high-quality data analysis to support

decisions, and assist maintenance and other non-time-critical processes. Cyber-physical systems require real-time, low-latency processed data and autonomy which is where edge computing and storage excels (Chen et al. 2018). To avoid storing non-useful data in edge computing systems, Deng et al. (2018) propose a method to remove noise from monitoring system data and fix it immediately after acquisition. They also propose task planning to achieve an overall energy-efficient process that supplies the virtual part of a cyber-physical system with high-quality data. Noise removal techniques at early stages have also been proposed (Liu et al. 2020; Oleghe 2020). However, these are aimed at data collection quality. Signal noise reduction is a widely researched field related to signal processing; Despite the relevance with some of data reduction methods, it is outside the scope of this review.

In general, the removal of noise and data cleaning are data reduction methods. Data reduction reduces the complexity and size of data which subsequently reduces the computing power and storage capacity required to process it (Habib ur Rehman et al. 2016). Data reduction methods also result in the reduction of the velocity of data that enters the processing systems which is key in digital twins that have a direct connection to the data source. However, real-time velocity reduction on a data stream requires programming effort and computational resources to succeed (Habib ur Rehman et al. 2016). Pandey and Shukla (2020) use stratified sampling to reduce the amount of data per time unit that is processed from a data source. Barika et al. (2021) propose a dynamic scheduling technique that deals with variation in the velocity of data that a stream-workflow-based system produces and Alzyadat et al. (Alzyadat et al. 2019) use a fuzzy map approach to condense the data before processing.

The complexity and resource capacity required to identify useful data and extract value from it has led companies to store all collected data without capitalising on its value (Kusiak 2017). Data-related costs are highly dependent on the volume, however, there are studies that show that only 10% is used (Waterston 2021). In the review of (Kusiak 2017), it is identified that the main areas in which the manufacturing sector develops data solutions are monitoring, prediction, data

analytics and information and communication technologies. These are areas where data is fed to digital twins (Tao et al. 2019) which shows that in a modern manufacturing environment extracting value from data is closely related to developing digital twins.

2.4 LITERATURE REVIEW DISCUSSION

Digitalisation has been a long-term evolutionary process of manufacturing that integrates computer science, simulation and data science related technologies into production resources and facilities. Digital twins have become the latest vehicle for this integration but as more technologies are being introduced digital twins grow in terms of size and complexity. Since there are many new applications for digital twins to be explored, the research has been focused on the application field. This also explains the exponential increase of the digital twin and digitalisation-related publications. However, the structural limitations of the current common practice make every digital twin solution unique while many functional characteristics of the implemented systems are the same.

At the core of every digital twin are one or more simulators that process the supplied data. It is that exact place where the lack of a common structure and development methodology leads to redeveloping big parts of the system whenever a simulation model doesn't perform as expected, when new data streams become available or when new parameters need to be calculated. At the same time, this becomes a barrier to research since there is scope for improving the digital twins themselves, but this is not possible or not efficient to do at the moment since the results would refer only to a specific implementation.

To attract research in any field, there needs to be potential for benefits from the research outcomes. In the case of digital twins, a trigger would be to show

- how a simulator architecture facilitates the improvement of system performance and application flexibility,
- how this architecture enables the easy integration of new data processing techniques to increase the computational capabilities of the system and to improve the estimation of manufacturing process parameters.

In response to these research gaps and at the same time the research opportunities arising from the above, this work proposes a new simulator architecture. It then presents step by step how this generic architecture can be applied in CNC processes and uses vertical milling as a case study to demonstrate the benefits of its application.

3 ARCHITECTURE OF THE SIMULATOR

Modern process design should take into consideration the digital environment that coexists with the physical one to achieve business goals in the most efficient way. This is due to the level of autonomy that the business process can achieve if digital assets supervise the physical ones and leave only high-level or high-impact decisions to be taken by humans. From the material reviewed in the previous chapter, it is becoming clear that a key building element of the digital environment and its ability to control physical entities is the simulator replicating the physical assets of the business and/or the business asset interactions. The simulator is the tool that quantifies and predicts asset behaviours and therefore it is the foundation for introducing logic and autonomy into business systems. This leads to the core of this research work which is the next generation simulation system architecture that can support in a systematic way the design of modern processes and open new horizons in the way that processes are run. This chapter describes the elements of the simulator that were designed and implemented by the author for this research. It also explains how these elements interact and provides a guide with critical details for the implementation of the architecture. The description of the architecture is focused on manufacturing processes with more detailed examples related to CNC processes. Its applications are not restricted to manufacturing as the simulation principles remain the same in a wide range of digitalised systems. The engineering of a simulator with the potential for much wider application was the intention of the author in completing this research.

3.1 SIMULATOR BOUNDARIES

When talking about digitalisation technologies there is an overlap between the terms of digital twins and simulators. Before setting the boundaries of the simulator, it is important to distinguish the two terms. As explained in 2.1.3 a digital twin is a wrapper system (Definition of wrapper | PCMag. 2022) containing one or more simulators. As its name implies a 'digital twin' mirrors every element of a physical system which it can monitor and potentially control. The digital twin requires interfaces to communicate with its environment including the physical

system it represents, other production management systems and shop floor technical or managerial staff with various levels of expertise. On the other hand, a simulator is focused on a specific process (not on a specific physical system) and its purpose is to provide an accurate estimation of output parameters based on the given set of input parameters and the accumulated knowledge from previous usage and training. A simulator requires interfaces to exchange data with the system it is embedded within. This includes the Graphical User Interface (GUI) typically used by process experts who test scenarios, developers who test or integrate the simulator and users who require direct access to the simulator.

The above clarification isolates a simulator from the physical environment since its role is not to directly interact. A simulator receives data (for example from a user data input file or from a monitoring system data adapter), it reports simulation results, but it is not responsible for actively seeking input data nor for forcing other systems to change their behaviour. Therefore, the simulator lives purely in the digital world, it provides communication channels for input/output but requires a wrapper system to connect it with the rest of the digital or any part of the physical world. The GUI that was mentioned before is a wrapper module that connects the simulator with the user. Despite being essential in the simulator's development, upgrade, or integration into other systems it is not a core part.

The simulator as a whole is passive in nature. Its operation relies on the availability of input data that comes with a trigger or acts as a trigger itself. External systems should view the simulator as a tool to convert input data into output results and use it as such to bridge the gap between raw data and usable information. The external systems should not need to interfere with the simulator's internal operations but should only expect to get the results. From the simulator perspective, everything required to convert input data to results should be contained within the simulator. This also means that problems in internal data flow should be handled internally and not require the intervention of the external system. Under the same logic, the simulator should handle all processes that improve its performance including learning from actual data and storing internally required information by itself. Finally, the simulator architecture should contribute to minimising the resources

needed to calculate all process-related parameters. The final statement ensures that a digital twin will require the minimum amount of simulation modules to deliver its functionality.

In addition to the above specification of simulator boundaries, the scope of the architecture itself must be set. The Chapter 2 review is focused on manufacturing and therefore the proposed architecture primarily targets simulation systems used in manufacturing. This is by no means a restriction since the same architecture can be applied to IoT-based systems or other data-based simulations. Generic descriptions are used where possible to enable the reader to consider the architecture as the base for any relevant application.

On the other hand, the field that this work focuses on is the simulation of CNC processes where process monitoring systems exist. These systems would typically generate data at a rate between 0.1 and 100Hz which is accompanied by the process setup data. Much lower data generation rates are restricting the usage of machine learning and much higher rates would introduce hardware limitations that are not covered by simple speed and computational capacity considerations. It should be noted that the typical CNC controller has a computational capacity similar to a personal computer. Finally, the type of data expected by the monitoring system is data typically generated by sensors. There is no restriction set by the architecture on the data type and format but for the need of this PhD work sensor-generated data was considered as the base case.

3.2 PROPOSED SIMULATOR ARCHITECTURE

Similarities in structure and behaviours among simulators (regardless of the application) have been presented in section 2.2.1. Figure 6, is an attempt to summarise these similarities but there are two main issues with this simplified representation of the common architecture. The first is that the elements of Figure 6 do not describe the actual architecture of the simulator implementation. This can be demonstrated with two examples from the reviewed simulation systems that use machine learning to calculate results. Zhou et al. (2021) use a reinforcement learning model which is the core part of the simulation model. The simulation

engine is not separable from the machine learning model but there is a database retaining the artificial neural network parameters. In contrast, Zhao and Sun (2021) base the simulation model on theoretical FEA simulation models and use ARMA models to calculate results. The process simulation model is clearly based on theory and the machine learning functionality is an independent module complementing the simulation model. However, there is no requirement for a database to retain information between runs.

The second issue with Figure 6 is that it is not enough to assist in the simplification of a simulator implementation. Looking again at the examples of Zhou et al. (2021) and Zhao and Sun (2021) the development process is complicated because in the first case simulation and learning tasks are combined and in the second case knowledge is separated from the theoretical model and ultimately it is not retained. This lack of consistency in development would also have an impact on new simulator implementations and on the ability of another researcher to extend an existing work since there is no path to follow and every simulation system architecture is unique.

This work proposes an architecture in line with Figure 9. The intention is to: standardise and systematise the development of simulators, change the monolithic view of a simulator to a modular distributed computing entity, embed machine learning to simulation core operations, clarify the way this new generation simulator should operate, and finally ensure that the operation of the simulator is compatible with modern digitalisation needs. Figure 9 presents the idea of this work and serves as a reference point for the rest of this chapter.

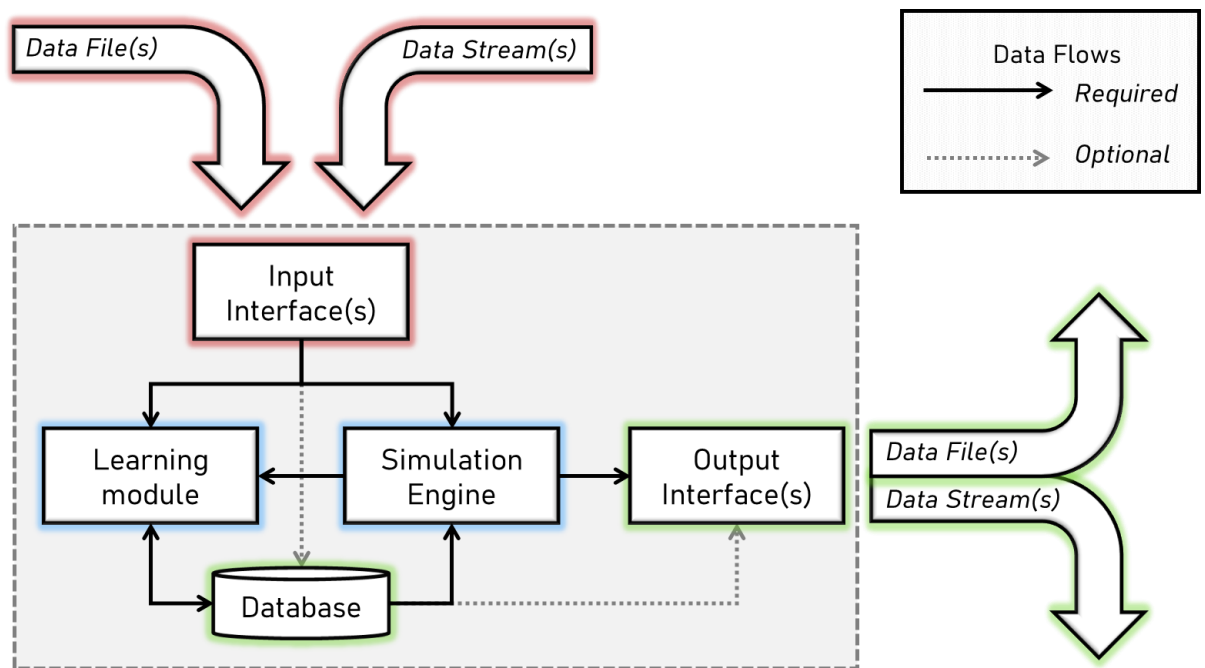


Figure 9 Next Generation Simulator Architecture

As an introduction, the proposed architecture is briefly described using a static and a dynamic approach before the details for each element are further discussed.

At the 'heart' of the architecture is the simulation engine. This is the implementation of the process simulation model together with the models that calculate derivative data. The simulation engine's objective is to replicate the physical process and accurately calculate the output parameters of interest. The element making the simulator smart is the learning module which uses supervised learning to update the simulation engine's parameters. Since modern systems have access to real-time process data the simulator can be adaptive so that it uses 'experience' to accurately estimate results. Then, the backbone of the architecture is the database where simulation engine parameters and all data relevant to the simulator and its operation are exchanged and/or stored. There is no intelligence without memory and the database is the link between past run analysis and current or future run estimation. Finally, the input and output interfaces are the means to receive data and communicate results. The input interface is responsible for providing a way to supply the simulator with raw data and for preparing this data based on the simulator's requirements. The output interface is responsible for publishing the results to other systems through a variety of communication technologies.

The dynamics of the architecture are better understood by following the data flows. The simulator consumes raw data and through flexibly defined steps it produces information and then knowledge. Data flows from the input interface that receives raw data, then cleans, fuses, and transforms it according to the simulation engine and learning module common standards and finally passes the pre-processed data to the simulation engine and the learning module so they can perform their respective tasks. The simulation engine uses the data prepared by the input interface to estimate the output parameters. To do this it first retrieves the latest learning model parameters from the database. It then uses the engine algorithms to calculate the results and passes the raw results to the output interface. In parallel to the simulation engine, the learning module receives the same pre-processed input data together with process monitoring data. By using supervised learning, it updates the learning model parameters stored in the database so the simulation engine can instantly run with the most updated information. Finally, the simulation results are sent to the output interface and published to external systems. In real-time applications, input and output interfaces are implemented in a server so the raw data is supplied through a connection with the data source and the results are immediately published to the wrapper system or other external systems (clients).

The above architecture is fairly generic but it cannot cover every type of simulation. As a minimum, it assumes three things:

- The field of application is computer simulation. Also, visual simulations whose aim is only to generate a virtual environment and navigate the user through it are not relevant.
- It is meaningful that implementations of the architecture evolve over time and that they use empirical data to surpass the performance of pure theoretical models.
- There is a way to measure and supply in a digitalised format the values that the simulator is meant to estimate.

This is the outline and the core limitations of the proposed next generation simulator architecture. In the following sections, each element of the architecture is described in detail and guidance for implementation is provided. To demonstrate the application and benefits of the architecture a significant part of this research work focuses on digital twins for CNC machining processes. However, the architecture has a much wider scope and it can potentially support all applications where digital twins supervise physical assets or any digitalisation project where the simulator is part of a dynamic digital ecosystem.

3.3 INPUT INTERFACE

Although the management of Big Data is not required to run a digital twin's simulator it must be the case that a modern architecture should be able to cope with raw data that is 'big'. Previous work (Kitchin and McArdle 2016) presents a wide range of Big Data characteristics which leads to the conclusion that there is not one approach in pre-processing raw data suitable for all cases. The input interfaces of the architecture must provide a flexible way to deal with any characteristic of the supplied data. This flexibility is achieved by combining algorithms following the logic of Figure 10 process model.

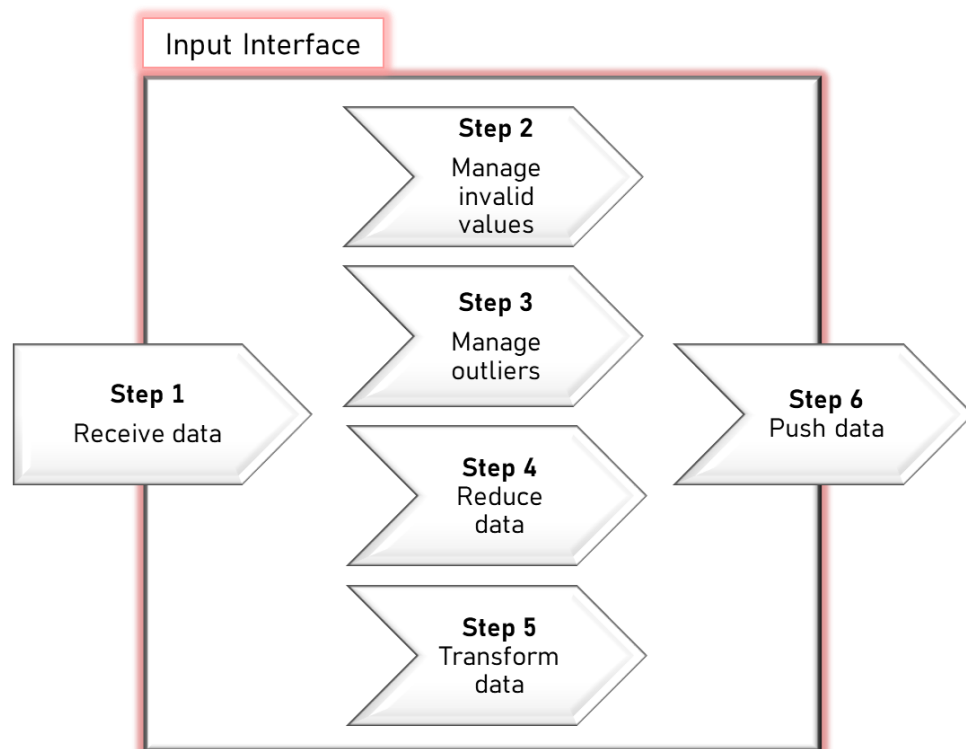


Figure 10 Input interface data processing steps

Except 1 and 6, all other steps of Figure 10 are neither mandatory for every implementation nor strictly ordered that way. Numerous possible combinations of input data characteristics could invalidate a strict interpretation. These steps are rather a guide written by the author to ensure that all data pre-processing is done by the input interface module. This means that the necessary actions can be taken at this stage and won't be confused with simulation engine tasks.

Having established the framework of tasks that the input interface is responsible for allows for a further explanation of the implementation and role of each step.

Step 1: Receive data

In a digitalised production the simulator will typically run as a service supporting the system it is embedded with. The input interface will be the part of the simulator waiting for input data to become available. This could be done with a webserver that waits for a client request containing data, a service that constantly checks a memory space, accessing a permanent storage media or any other way that data can be passed from one system to another. Passing data from the wrapper system or other external systems to the simulator may look like a trivial task in laboratory experiments tailored to a specific piece of equipment but in real industrial scenarios, it can be a complex task especially if bespoke data transfer software and hardware are used. The biggest challenge in complex scenarios is not data transmission itself but the risk of damaging the datasets or streams due to the multiple systems the data has to go through to reach its destination. Therefore, creating a simulator input interface that is 'design friendly' to the data source reduces the need for data processing by external systems and can potentially reduce the computing power needs of the digital ecosystem it belongs to.

To sum up, the aim of Step 1 is to 'bring inside the simulator' the data as produced by its source. This means all available data should be transferred without any alterations, omissions, or changes in its characteristics.

Step 2: Manage invalid values

To make data readable a first pass through it is required to ensure that each value conforms to a set of rules. As an example, the simulator may accept only integer values such as 8. Values like; eight, 8.0, '8', and +8, are easily understood by humans but may not be readable by a data parser. Further common issues are gaps (meaning no value), errors that may be indicated by 'N/A' or 'error' and zeros (which can be value 0 or interpretation of 'no value' from the data transmission software). In live transmissions this can mean, error stack traces, different size batch transmissions etc. It is up to the developer of the simulator to decide which values should be accepted. The suggested technique to ensure conformity is to create search patterns based on Regular Expressions (regex). The values that do not conform may be discarded (if the impact is minimal) or the data parser may be improved to recognise a wider range of formats or identify logical sequences that substitute the invalid values with valid ones. Since data input interfaces are developed at an early stage when the impact of data alterations cannot be measured it is suggested that any assumptions at this stage are recorded and reviewed during simulator verification. It should also be noted that where errors in data are produced because of an abnormal behaviour of the data source or errors in the transmission there is always the possibility that all neighbouring values are affected despite being valid.

To summarise, the management of invalid data overall ensures that only values in readable format are reaching the simulation engine and it is the first 'opportunity' to fix errors in the dataset. At the same time, this is the initial point of raw data processing meaning that decisions at this point do affect the simulator results.

Step 3: Manage outliers

There is a wide range of techniques to identify outliers. In smaller datasets, simple data plots can quickly show the region within which 'normal' data lies. In larger datasets box plots or statistical methods such as standard deviation distributions can be used to bring data in an easy-to-observe form and to systematically study anomalies. If a more sophisticated analysis is required due to data size or complexity, there are more advanced clustering techniques often using unsupervised learning. These however can only be used if processing power or

processing time are not restricted, otherwise, the simulator won't be able to cope with real-time processing requirements.

The challenging part of outlier management is the interpretation of the anomaly. An anomaly could be noise or a system's error, could be an incident where process parameters reached extreme values, could be one of a series of extreme values the frequency of which is important or other cases where the anomaly may or may not be a key part of the simulated process. The interpretation is another decision taken at an early stage that should be reviewed during the simulator's verification so the exclusion of real events is avoided.

Adding to the interpretation challenges, simulation modules that run with real-time data streams need suitable outlier management techniques since the data cannot be examined by a human operator with offline tools. In these cases, smoothing techniques are preferred which can also be combined with filters. In this way, any value that can be safely attributed to error(s) is filtered out and any anomaly will blend with 'normal' values to produce a more realistic simulator input. The suitable technique and the level of smoothing are to be determined by the developer. If the risks of altering the data are higher compared to false alarms caused by the simulation results, then outlier management could be omitted altogether. In this case, the simulation engine should have provisions for input data anomalies.

Summing up, outlier management aims at developing a systematic way to deal with data anomalies. There are many tools and techniques available to perform this task but altering values of the dataset will inevitably introduce a bias. Whether the bias assists or hinders simulation accuracy is to be decided by the simulator developers.

Step 4: Reduce data

The accuracy or completeness of the simulation results is not always directly related to the amount of supplied data. The same result could be achieved by reducing the size of input data. This could involve the removal of duplicates and/or the use of regression. It may also be achieved by reducing the number of data dimensions by merging related parameters or features and/or removing insignificant impact parameters. Outside the field of data science, data reduction

has been associated with the reduction in storage space requirements. However, data reduction methods have a wider range of applications and can supply the simulator with high-value data that is just enough to 'do the job' as opposed to lots of data that may or may not contain useful information. Especially in cases where Industrial Internet of Things (IIoT) technologies monitor the physical system, data comes from heterogeneous sources that operate autonomously (Wang 2017). In these cases, data synchronisation and merging are required to generate the input for the simulator.

As with previous steps, there is no set list of methods or techniques to follow. Both numerosity reduction and dimensionality reduction methods can be used and should always be considered since this can lead to lower system complexity and the reduction of computing capacity needs. For the same reason, it should be assessed by the developer whether applying data reduction improved the simulator as a system in terms of calculation speed, accuracy, system simplicity or any other performance indicator that is important for the implementation. Any additional process adds complexity and requires processing power. In addition, data reduction methods should be used with caution if data is compressed and then the simulation engine has to decompress it. These processes are internal to the simulator and there is no storage involved (except systems running on the cloud where processes are geographically spread and compression could accelerate transmission speed).

To summarise, the core idea of Step 4 is to improve data by concentrating its value. There is a wide range of methods typically borrowed by data science that can merge and reduce the size of supplied data, but this step should be omitted if there are no significant gains in the performance or the design clarity of the simulator.

Step 5: Transform data.

Before passing the input data to the simulation engine the data is typically transformed to comply with the requirements of the simulation engine. As mentioned earlier, the input interface is responsible for all pre-processing so the responsibilities of each element of the proposed architecture are clear and distinct. Therefore, the simulation engine should receive data as specified by the simulation model and should not do any further pre-processing.

Data transformation tasks are mainly reformatting, restructuring or value changing such as data normalisation or data mapping. These tasks may also be undertaken at an earlier stage since restructuring raw data before any check on the actual values may speed up all of the pre-processing processes. It may also be beneficial to combine data restructuring with other cleaning tasks although this could add complexity to the system without significant gains in performance. As with the previous step, there should always be a focus on the overall system performance so that any additional computing or development effort is justified.

Data transformation serves multiple purposes; the most important one is to make the data ready to be used as the input for the simulation engine. At the same time, better-structured data or suitably formatted data is also easier to comprehend by humans. Understanding data characteristics is critical in all data-related processing and it can shorten development times since dataset-related issues can be quickly spotted. In some cases, preliminary data transformation can be separated from the simulator to accommodate the needs of other applications. On the other hand, data transformation has less potential in adding value to data since it only changes the representation (unless direct access to data is by itself valuable). Large datasets or voluminous data streams may significantly increase the computing burden which could otherwise be lifted with minor modifications in the simulation model. Finally, any manipulation of data increases the risk of introducing errors therefore the developer needs to ensure that the benefits of transformation tasks outweigh the impact on the overall development process.

To summarise, data transformation is aimed at bringing data to the exact form required by the simulation engine or to one that assists pre-processing. It can be performed at various stages of data pre-processing, but its use should be justified otherwise it will unnecessarily increase the computational burden and complexity.

Step 6: Push data

After pre-processing is finished, the data is passed to the simulation engine (and if appropriate to the learning module). The simulation engine is triggered by the input interface and this is why the term 'push' is used. Step 6 is relatively straightforward since the data at this point always has the expected characteristics and the data

transmission is an internal system process therefore the risk of interferences is minimal. For the developers, a good balance between performance and design simplicity should be found. The input interface is a distinct module within the simulator and a clear distinction from other modules should be maintained to ensure design clarity. On the other hand, since the data is ready to be processed by the simulation engine any delay in data transfer has a direct impact on performance. It is therefore imperative that no data manipulation takes place at this point. As a general development guide, the transaction should take place using the Random Access Memory (RAM) of the computing system or a lower-level type of memory if possible. Any writes to persistent memory, possibly to a hard drive or to a cloud service, should be done asynchronously and in parallel to the main process. Finally, an efficient way to trigger the simulation engine should be used to avoid delays until the simulation engine loads. Typically, this would be achieved if the simulation engine runs as a service that waits for new data to become available.

Push data is the last step of raw data pre-processing and after this step the data should exit the input interface containing more concentrated value and with characteristics that enable processing by other modules.

The above steps aim at cases where sensors generate the real-time data supplied to the simulator. Although modern simulators run within digital twins where the real-time element is dominant, a simulator is not restricted to real-time operations. Digital twins are also used during process planning where the simulator runs with offline datasets, generated to replicate the case under investigation. A typical example is using G-Code language commands (part program) as input to a CNC machine simulator or providing just the design of the part to be manufactured without any details about the machine. Such cases don't require all steps of Figure 10 because the data supplied to the simulation engine is generated within the input interface. This data generation falls under one or more data transformation steps which for example transform a G0 command to a mathematical representation of a 3D line and then transform this line to a series of points that represent the toolpath. A more detailed description of an application is presented in section 4.2.3.

To conclude, raw data that is fed through the input interface is not necessarily a stream of values. Historically, the opposite is true since only in the last decade have simulators run in such a manner. Therefore, the input interface should be capable of coping with the real-time processing of a raw data stream but also offer offline processing capabilities to support planning or to simply provide backward compatibility for digital twins that are connected to legacy systems.

3.4 SIMULATION ENGINE

When thinking about a simulation model the first thing that comes to mind is the simulation engine. All other simulator modules, despite being part of the simulation model seem secondary as they exist to support the seamless engine operation. It is difficult to specify an ideal structure of a simulation engine since there are thousands of different simulation models that have been developed over the years with a very wide range of applications as described in section 2.2.2. It is also common in commercial software to provide a list of different models that the user can choose from when running a simulation. To provide the required flexibility, this section is split into two parts, one generic that can be used as a guide for the majority of simulation applications and a second one which focuses on the characteristics of a simulation engine for CNC processes. The second part can also be considered as an introduction to chapter 4 where the implementation demonstrating the suggested architecture is based on the principles presented here.

3.4.1 Simulation engine general characteristics

The simulation model is a combination of mathematical calculation formulas and logical workflows that emulate the behaviour of real-world systems. The simulation engine, shown in Figure 11, is the implementation of these formulas and workflows. It receives input data, feeds the mathematical formulas in a sequence defined by the model workflow and produces the results which are a description of one or more properties of the real-world system.

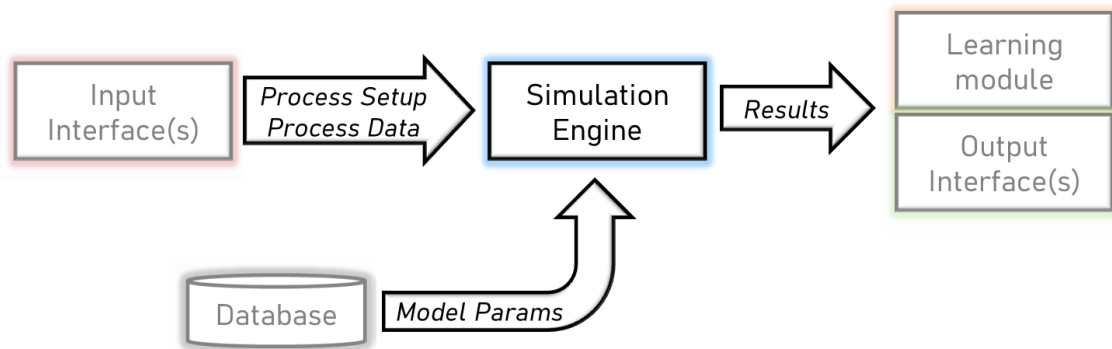


Figure 11 Simulation engine data flows

The mathematical formulas and the workflows are predefined, regardless of the number of cases that the engine can simulate. It therefore lacks the flexibility a real-world system has since the latter changes its behaviour depending on its usage, environment and numerous other parameters that affect its operation over time. To work around this limitation, the mathematical formulas primarily but also the parameters that the workflow depends on, are variables whose values are obtained by a database that is constantly updated.

Inside the simulation engine, there is a looping process taking place (illustrated in Figure 12): input - decision - calculation - output. The logic of this micro-process is that the simulation engine works as a multilevel data converter that receives input data which it converts step by step to produce the simulation results. This way the simulation model 'size' may increase but it becomes less complex and easier to test since each loop can be isolated and tested separately. This is an important part of the author's architecture since complex simulation models have long been identified as non-efficient. The discussion in Chwif et al. (2000) is a very good starting point for one to understand the history and reasons to avoid high levels of complexity. The article is still relevant because as the authors suggest, high computing capacity does not offset the impact of high complexity.

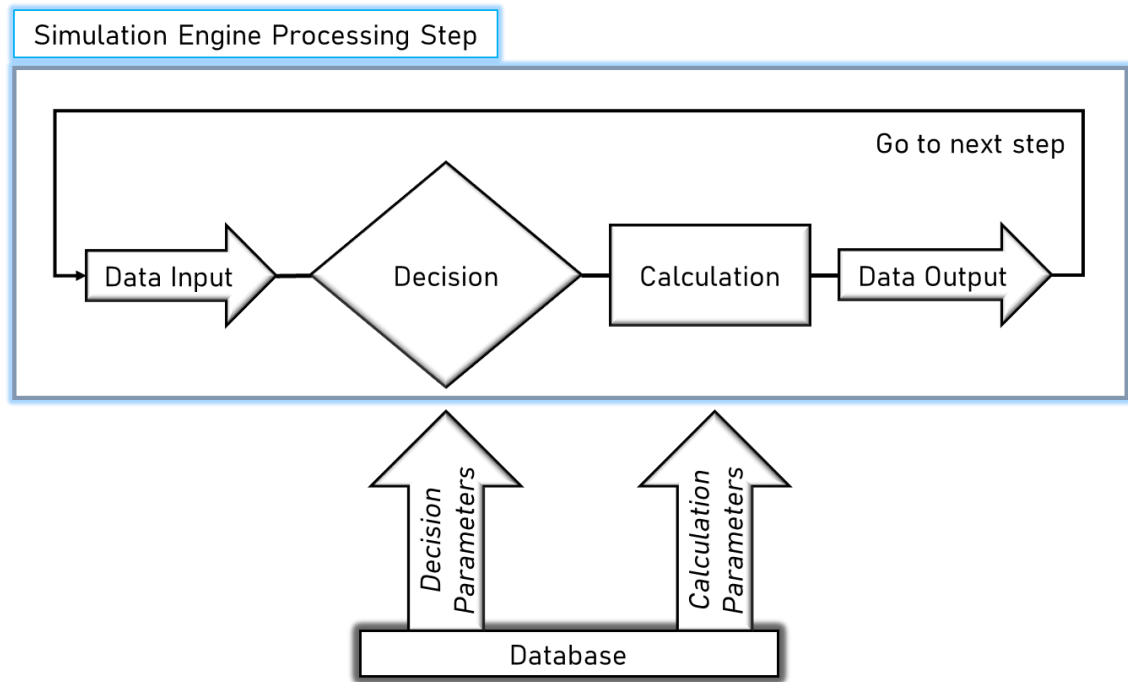


Figure 12 Simulation engine internal operation loop

Focusing back on the architecture of the simulation engine, Figure 12 elements are the highest level of detail that can be provided without taking into consideration the specifics of each application. The way each cycle runs is as follows:

1. The results from the previous loop or, if this is the first loop, the data from the input interface are available and ready to be used as the input data. It is recommended that no data transformation is required between loops (although this is not always possible).
2. The latest values for the decision algorithm parameters are updated. These values have been calculated by the learning module and are retrieved from the simulator database. It is very inefficient to read them from the database every time a new loop begins therefore it is recommended that value retrieval from the database is either done in parallel to simulation engine operation (multithreading) or it is done at longer intervals therefore without significant impact on the total simulation time.
3. Previous cycle results are loaded into the updated decision algorithm and the calculation formula for this loop is specified. From a programming point of view, the decision algorithms are typically one or more simple 'if' statements that select the appropriate calculation model. Simplicity is critical for every element

because a sophisticated decision algorithm (that requires multiple computer operations) has a high computational impact. The decision algorithm can also be used to break a complex calculation formula into many simple formulas (each one for a very specific case) but this should be done if development time and testing are not significantly affected.

4. The selected calculation formula parameters are updated (similar rules to step 2 apply).
5. Previous cycle results are loaded into the calculation formula and the new results are calculated. The new results are loaded into the memory, so they are ready to be used by the next cycle. During development, the formulas must be tested separately with purpose-built datasets so each formula can be verified. Should issues occur in the future, the testing datasets should be updated along with the formulas (e.g. in cases of division by zero or null value exceptions).

Real-time-data-based simulation is naturally discrete event-based since each data sample that arrives at the simulator is a new event changing the state of the simulation model. Not receiving an input sample (no time passing included) means that there is no input data to feed the calculation formulas that are responsible for updating the parameters of the model. For this reason, discrete event simulation modelling is the go-to method for simulators supporting digital twins, although lately there are efforts to use agent-based models to model real-time systems (Malleon et al. 2020). For the same reason, the presented architecture is built around and tested with discrete event models.

The above approach sets the foundation for the simulator development undertaken by the author, but it is too generic to explain how the new generation simulator engineered for this research is different from traditional approaches or even from simple decision support applications. Before delving deeper into the differences some more background knowledge should be provided. Current simulation model development falls under three categories: white box, grey box, and black box. White box models have a very transparent design typically based on theoretical formulas that describe the physical phenomena of the system and the results are always predictable. At the other end of the spectrum, black box models use historical data

to predict the output of the system based on the input without considering the internal structure and dynamics of the system. The formulas these models are made of are not related to physical phenomena but only to statistical or machine learning methods and therefore the output cannot be predicted in advance since it depends on the model's training dataset.

Big Data and machine learning have been very topical fields since 2010 and as a result, black box models have evolved in terms of reliability and efficiency. However, key drawbacks of such models are the low accuracy if input data is outside the range of the training dataset, the inability to deal with data heavily distorted by noise and ultimately the fact that black box models do not use the available knowledge around the physical phenomena of the system.

Grey box models are simply any combination of white and black box modelling. Grey box modelling is very demanding in terms of knowledge since it combines the system's physical phenomena theory, statistics and machine learning. For example, a current grey box development project involves 5 departments of Fraunhofer Institute (Grey Box Model – Integrating Application Knowledge in the Learning Process - Fraunhofer ITWM. 2021). On the positive side, this is the most flexible way to develop a simulation engine which will ensure that the model results will be valid even in cases not predicted during the simulator development phase. This means that effort will be saved in the long term when black box models require further training to deal with out-of-range cases. This research work has found that the approach of Yang et al. (2017) is a forward-looking way to model simulation engines. After presenting different combinations of white and black box models, they propose that model building should start with a computationally light, white model that is further enhanced by black models to deliver the overall model requirements. Their work is based on additive manufacturing, so the idea is not as refined for wider use, and it is open to interpretation regarding the exact characteristics of the developed model.

The adaptation of the idea of two models (white and black) lead to the architecture of the simulation engine engineered by the author in this research. The simulation engine is formed by two parts: the process digital replication part, and the process

data analysis part. The two parts are depicted in Figure 13 which also provides a more detailed view of the generic model of Figure 11.

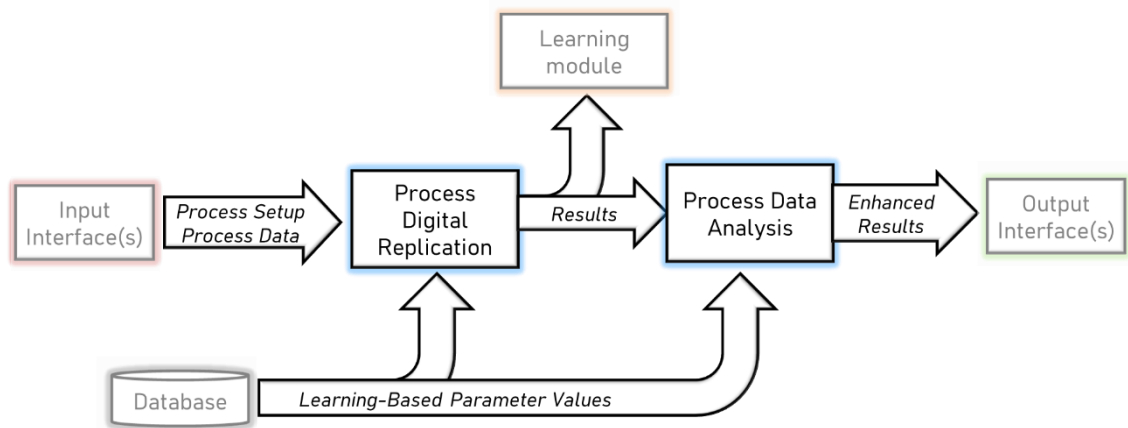


Figure 13 Detailed architecture of simulation engine

The process digital replication part is the digitalisation of the physical process that is being simulated. There, all knowledge related to 'how the process works' is applied, and the model estimates the characteristics of the simulated system while the process is running. For example, in a material removal process, the digital replication part starts with a digital part and a digital material removal tool. Then it brings the tool into contact with the digital part (as it happens in the physical process) and removes the affected part volume. Finally, it finishes with a representation of a finished part and (if applicable) a worn tool. These steps are done in parallel to the physical process, so the simulation model generates real-time results. The reason behind replicating the physical process is to create a space in the architecture where all of the theoretical knowledge about the process can be used. Developing a model very close to the physical process ensures that all available theory is relevant and applicable in the same way that it is applied in the physical world.

As soon as results from the digital replication become available the simulation engine uses any type of model to further analyse these results. This step typically produces the final simulation results which are based on the process digital replication but are not restricted by the limitations of theoretical models nor by the availability of training data. Furthermore, deployed black box models can be trained by the digital replica, so an initial, theory-based state of the model is

achieved even before the simulator receives data from the physical world. Both parts of the simulation engine will evolve with the enactment of the physical process since the theoretical calculations will change as new parameter values are received by the database. The way that empirical data is integrated into theoretical formulas is up to the developer although the straightforward way would be to modify parameters whose value is specified in empirical tables (for example in machining these can be material machinability, specific cutting force etc.). Finally, the simulation engine must be able to run without actual monitoring data being available. The architecture is developed to cover the real-time needs of modern simulators but at the same time, it should maintain its capability to run as an offline simulator that can be used for planning before running the process for the first time. In this case, the simulation engine will be fed with synthetic data or completely artificial process data, and it will run based on initial values set in the database. Learning from these datasets should be avoided unless the intention is to initialise the simulator (where high-quality synthetic data must be used).

3.4.2 CNC process simulation engine

To better demonstrate the architecture and to offer more ideas for CNC process implementations, a simulation model development methodology is presented. This method was developed initially with subtractive processes in mind, but it could be equally effective in additive processes. Usage in joining or forming processes would be limited and more importantly not aligned with the philosophy of the method. In addition, since this work did not apply the method on material displacement phenomena, the issues that the developer may face in implementations such as laser cutting where melt material moves away from the part or machining of thin walls where the billet elements move have not been investigated.

Differences between simulation models have an impact on what the elements of the simulation engine represent (as shown in Figure 12) as well as the number of iterations required to convert the input data into results. The proposed method engineered by the author aims at developing a mesh-based CNC simulation model. The core idea behind the model is that the machined material is represented by a mesh of cubes and if a cube's volume intersects with the volume of the cutting tool,

then the cube is flagged as 'machined'. At the same time, the part of the cutting tool whose volume intersects with the cube adds the event to its counter. All other calculations are based on this volume intersection check process. This is a simplified (if not simplistic) view of the model which will be analysed later in this chapter. Before getting into details about the simulation model, a breakdown of its main parts is presented (Figure 14).

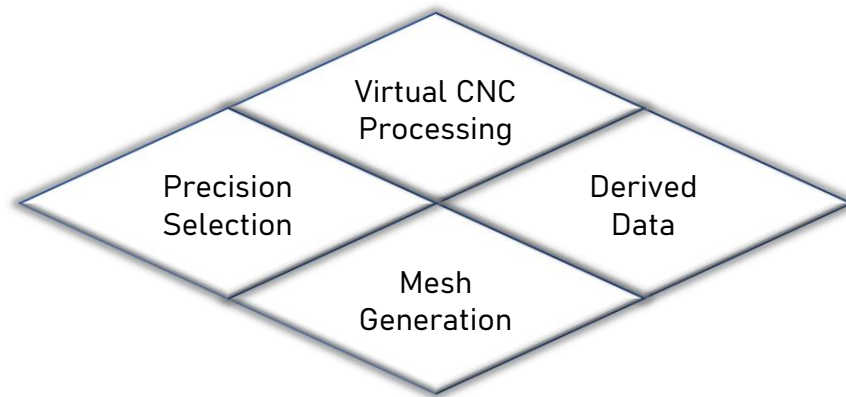


Figure 14 CNC model development breakdown

The order followed to develop each part of the model is not fixed since precision selection requires knowledge about the mesh generation mechanism and derived data is based on the outputs of the Virtual CNC processing. In the following sections, each part of the development process is described separately with details on the development purpose, application examples and the contribution of each part to the simulation engine operation. As with other elements of the simulator architecture, the final implementation characteristics depend on the objectives of the implemented system and the strategy that the developer decides to follow. The following, therefore, describes the work undertaken by the author in completing this research.

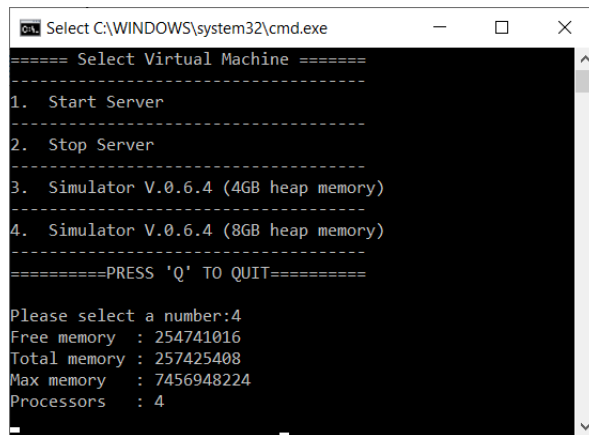
3.4.2.1 Precision Selection

Taking out model verification and model training that are both done during the development phase, the predictive capability (Oberkampff 2019) of an implemented simulation model depends on the application and on the calibration of the selected model. As soon as a model has been implemented and set to run autonomously the number of available calibration options is reduced and calibration is restricted to

decisions during the simulation engine run (as the decision is defined in Figure 12). Since the aim of the approach being considered herein is to support real-time applications performance speed is a critical part of the simulation model. The time that the simulator needs to process input data and calculate results is determined by the model's computational needs and the hosting system's computing power (as represented by the operations per second and size/speed of the random-access memory). Computing technologies that support real-time applications operate at the 'edge' meaning close to the data sources and therefore there is limited utilisation of cloud computing resources. Consequently, a 'straightforward' way to calibrate the model and achieve the optimum precision is to initialise the simulation based on two indicators.

1. The memory requirement of the mesh. If processing the mesh requires more Random Access Memory (RAM) than the available one then the results calculation speed will decrease dramatically or the system will run into an error and stop.
2. The time that is needed to do the calculations for one sample should be shorter than the mean time between sample arrivals. Data reduction in the simulator's input interface will assist in reducing the speed that samples arrive at the engine but then, if the engine cannot cope with the load, either the hardware has to be upgraded or the data reduction algorithm should be of a lossy/irreversible type meaning that part of the contained information will be lost.

Both indicators are related to the specific hardware that the simulator runs on and all specifications of hardware are (typically) known at the initialisation of the simulation engine. To ensure a smooth process the hardware resources should be reserved if this is not already done by the design/operating system of the physical host. The exact calculation of the mesh can be done during initialisation (e.g. mesh should require 50% of available memory) but thorough verification for multiple scenarios is needed during development. The simulator developed for this research work and presented in Chapter 4 calculates available memory and system capabilities during start-up (Figure 15) to prevent "out of memory" errors.



```
Select C:\WINDOWS\system32\cmd.exe
===== Select Virtual Machine =====
-----
1. Start Server
-----
2. Stop Server
-----
3. Simulator V.0.6.4 (4GB heap memory)
-----
4. Simulator V.0.6.4 (8GB heap memory)
-----
=====PRESS 'Q' TO QUIT=====

Please select a number:4
Free memory : 254741016
Total memory : 257425408
Max memory : 7456948224
Processors : 4
```

Figure 15 Simulator checking memory availability at start-up

3.4.2.2 Mesh Generation

Mesh-based simulation models typically begin the process by converting the examined physical entity to a structure of 3-dimensional shapes. The properties of each element are not necessarily the same. For example, if the part is made of stainless steel, then all elements have the same properties but if the part is a composite material (e.g. fibreglass and carbon) element properties change depending on the material that the represented element volume is made of. The most basic element property for every implementation is whether the element has been machined or not. This can be represented as a Boolean value in the majority of programming languages, and it is probably the fastest way to manage a CNC virtual process data. Should more element properties be examined, then each element becomes an entity with multiple parameters/characteristics. A final addition to the element properties is interactions with neighbouring elements which are defined through functions also attached to each element. Although in theory one could add 'everything' of interest to each element, in practice it is highly unlikely that the model will run faster than the real-time physical process. The computing burden will be significantly higher and comparable to running offline finite element method (FEM) models.

Mesh generation is a well-studied field and there are plenty of resources about element shape selection. These are however focused on the characteristics of the system they aim to represent and on the way that the elements are connected. This is not necessarily the case for a digital twin simulation model that aims at providing adequate details to safely manage the physical CNC machine in real time. This

research work found that for the representation of machined parts regular hexahedron (cube) shaped elements are very efficient. The reasons supporting the use of cubic elements are:

- A 3-dimensional matrix containing Boolean (true, false) values is enough to show which elements have been machined.
- The same matrix retains the physical position of each element simply by storing it in the respective position of the matrix (e.g. the element with indexes $x=1$, $y=1$, $z=1$ is the element at the corner closest to the origin of the global cartesian system the CNC machine uses).
- It allows for calculations based on element size as a unit instead of using actual cartesian units which turns most calculations with real numbers to calculations with integers.
- It simplifies the translation of physical part tolerances to mesh size requirements
- Engine verification and debugging are also simpler since cubic shapes make it easier to spot the state and position of each element without the need for visualisation tools (required to generate 3-dimensional scatter charts).

If the simulated process is subtractive then the cutting tools should also be simulated since their state has a high impact on process precision and reliability. Because of the wide variety of cutting tool shapes and technologies, any shape that keeps computation burden at a minimum may be used. A key difference compared to the machined part mesh is that only the cutting tool's surface is in contact with the part so it may be possible to use a 2-dimension mesh to represent only the contact surface. A good example is presented in Chapter 4 where the mesh for the cutting tool of a milling machine consists of rings (Figure 41). Another example would be the representation of a Wire Electro Discharge Machining (WEDM) cable that can be represented by a one-dimension mesh (a line with the length of the working part of the wire).

Regarding element size selection, the trade-off between precision and calculation speed has been discussed in the precision selection section 3.4.2.1 above. However, the simulator is only useful if it provides enough precision for the specific process

requirements and not only a rough prediction. The following list is an attempt to present all factors for element size specification assuming that there is no further manipulation of data or change in data representation when higher resolution mesh is needed which would be the case if variable mesh would be used or hybrid mesh using regression formulas instead of points would be created.

1. The first specification is the precision of the data samples arriving at the simulator. The element size should match the precision of the data samples so that the simulation engine calculations are on par with the quality of the provided data. To describe the idea through examples, if the coordinates of the cutting tool are provided with a $\pm 1\text{mm}$ precision, then the mesh should not be finer than 1mm since any result will have the precision of the input data. On the other hand, any mesh with an element size over 1mm will inevitably prevent the engine from extracting all value contained in the supplied data.
2. The tolerances for the part design are also a guide for mesh size requirements. Although the data and the simulation engine may be able to provide a very fine analysis, a digital twin that uses the simulator is focused on process requirements and is not a scientific tool pushing the boundaries of precision. It is therefore a waste of resources to provide a higher level of detail if this is not required by the part design.
3. Similar to the previous point, the tolerances of the machine/cutting tool processing the part are also an indicator of the highest level of mesh detail requirement. Using an example to explain the idea, if the precision of the machine is $\pm 0.1\text{mm}$ then even if the data produced by the machine has a precision of $\pm 0.001\text{mm}$ the data itself is flawed since it accurately captures the inaccuracies of the machine itself. Expanding further, such detailed calculations won't add value to the control precision of the digital twin supervising the process unless the aim is to monitor the deviation of the machine from the intended parameter values.
4. In most cases, the element size of the cutting tool should be matched to the element size of the mesh representing the machined part. This is better demonstrated through a graphical example. Figure 16 shows a comparison

between a cutting tool with a lower mesh resolution and one that has a matching resolution to the mesh of the part.

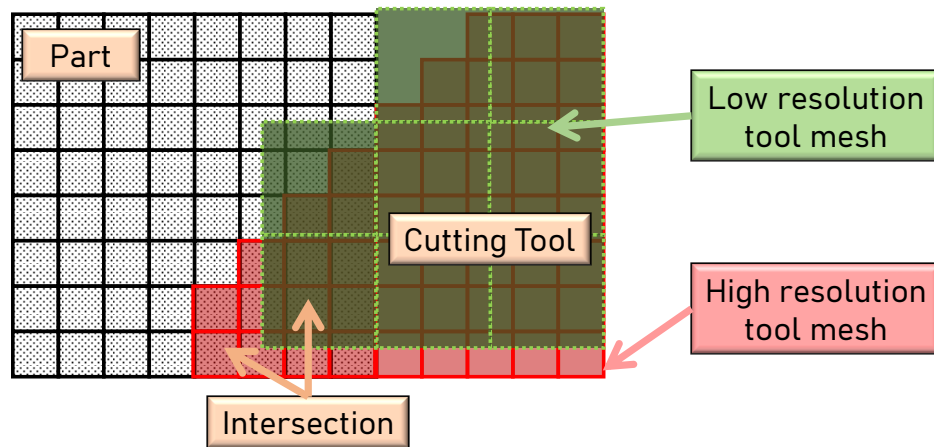


Figure 16 Matching vs non-matching cutting tool mesh.

The part that is machined by the low-resolution cutting tool has different elements machined compared to the one with the matching resolution. Ultimately, the resolution of the virtual process matches the resolution of the tool and not the one of the machined part. Apart from the poor results, this means that the simulation engine uses computing resources for a high-resolution part mesh but produces results of a lower-resolution model. Moreover, a mismatch in element size may lead to the false perception that the simulation is of high resolution. The same issues would exist if the machined part would be of lower resolution. In this case, the calculated properties of the cutting tool would be of lower accuracy.

To be in line with what a mesh element is, the 2-dimension mesh depicted in Figure 16 is showing squares for demonstration purposes. The mathematical model of the mesh is formed by points and each point represents a square area. As a result, the properties of the area are the properties of the specific point representing the area. Figure 17 clarifies the difference and a practical application of this critical detail is presented in chapter 4.

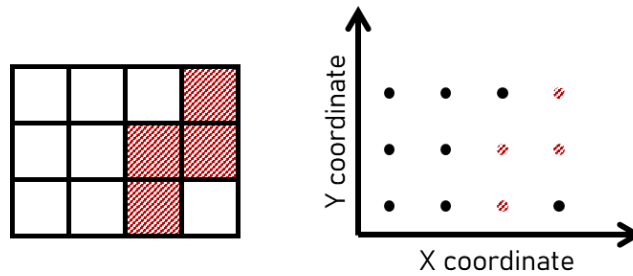


Figure 17 Left: Representation of a mesh with cubic shape elements. Right: Actual model of the mesh

Mesh generation is done at the beginning of the simulation, but it is a step that does not consume any input data (except meta-data that describes the part and cutting tool properties). If the part is pre-machined however, a lag may be introduced due to the time needed to initialise the elements of the mesh. It is therefore necessary to account for that extra time so the simulator and consecutively the digital twin can stay synchronised with the physical process. Another case to be considered is the concurrent processing of more than one part. This occurs when machining a batch of the same component using a multi-workpiece machining setup. If there is one tool doing the machining, then each machined part should have a separate mesh. Otherwise, a parallel running instance of the simulation engine should be considered.

After the mesh is generated, the physical parts of interest have a presence in the digital environment although they do so without any ability to interact with the environment. If the simulator is part of a digital shadow that simply replicates what is happening in the physical world, then the digitalisation of the asset may be the generation of the mesh and the direct projection of input data on its elements. An example would be an additive process where material deposition input data is received by the simulator as a set of cartesian coordinates and the only action required is to change the matrix value containing the element state from false to true to indicate that the element is added to the part. However, a full-scale CNC simulator should be able to support a digital twin which includes the calculation of interactions and future projections to support autonomous management of the asset.

3.4.2.3 Virtual CNC Processing

As briefly discussed in the beginning of section 3.4 the proposed simulation model engineered in this research for CNC processes aims at virtually replicating the process and then extracting from the virtual process the parameters of interest. Replication of the process means that virtual material is subtracted (or added) and every process parameter is recorded. To recreate the process virtually, first, the physical parts have to be digitalised and then the interactions between these parts should be mathematically modelled. The digitalisation of the materials is done by generating the mesh in the virtual space as described in the mesh generation section. In this section, the digitalisation of interactions is presented as well as the mechanism through which process parameters are monitored.

A distinction should be made between processes that do not have a tool coming into contact with the part (laser cutting, electro discharge machining etc) and processes with cutting tools (milling, turning etc.). The approach is different because in the first case, there is no physical interaction between the part and the cutting tool while in the second case, the physical interaction is a critical part of the material subtraction. In the first case, element temperature modelling may be used as the key part of the simulation while in the second case, element mechanical removal is important. It has to be noted that it is not within the scope of this research work to provide examples for every manufacturing process type. Many processes follow similar principles and are covered by the examples, but for everything else, the generic architecture should be used as a guide.

In no contact processes (subtractive or additive) only the mesh of the part exists in the digital space and therefore the results of the process are captured only from the properties of this mesh. This does not exclude the recording of statistics related to the physical machine itself or the calculation of the machine's position in the virtual space. The typical way that each mesh element is subtracted (or extruded/melted in additive processes) is by heating it to its melting point. Process quality and production rate can depend on numerous parameters but for this illustration, it is assumed that these are represented by the temperature that the element reaches. To follow the progress of the process along with the matrix

showing which elements have been machined (true/false) a second matrix is required to hold the element temperature property. If the element's temperature exceeds its melting point, then it is ready to be considered as machined and further checks can be done to verify that the element is indeed removed. It is apparent that representing this category of processes relies on heat transfer analysis. Heat transfer analysis increases the computation burden and there are problems such as plasma formation in laser cutting whose solution is still under investigation. The effectiveness of the suggested approach is that the theoretical equations can be simplified by using empirical data received from the machine's monitoring system. The temperature calculation algorithm may be completely replaced by an algorithm that models the connection between the supplied power and the result on the part. A methodology to create the required algorithms/models and thus mitigate the issues arising from theoretical-only models has been developed by the author and is presented in section 3.5.

Similar to element temperature-related calculations, other parameters may be calculated, as long as the ability of the engine to run faster than the physical process is maintained. The parameter which is always a requirement is whether the element has been machined (or printed). This is required so the digital twin knows both the state and the status of the process. For every iteration initiated by the arrival of a new input data sample, the affected elements are found by calculating the intersection of the part volume and the volume that is affected by an adequate amount of heat. In Figure 18 two examples are shown to demonstrate the model for material removal and material addition. Both the left and right figures are a snapshot of the process. A process snapshot is the state of the virtual system after all calculations related to a data sample have been done and model parameters are updated.

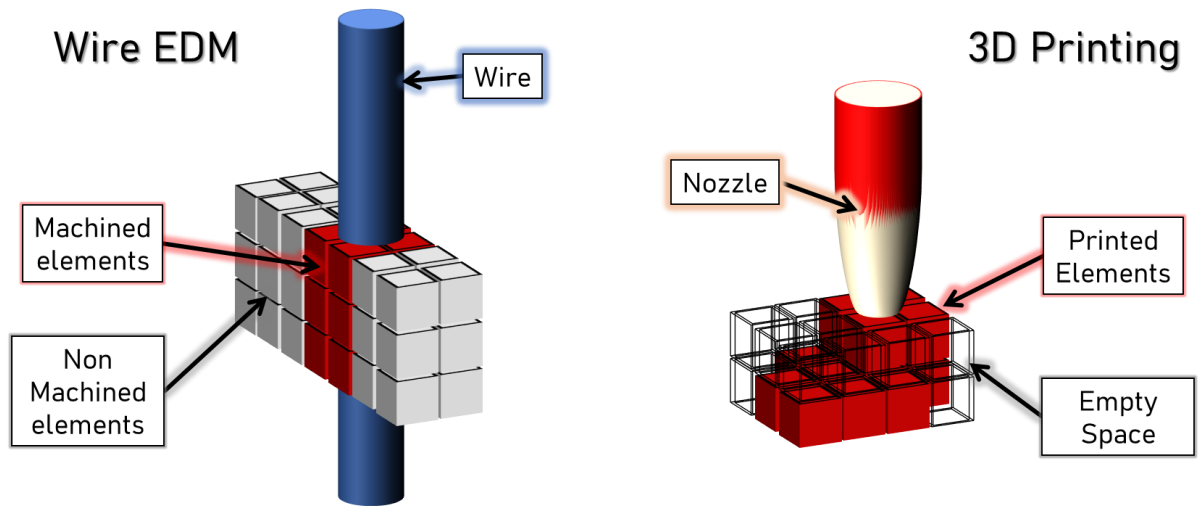


Figure 18 Left: WEDM machined volume calculation. Right: 3D printing extruded volume calculation

On the left, the red elements (cubes) are the ones being removed during the current snapshot of the Wire Electro Discharge Machining (WEDM) process. These cubes are intersected by the wire and therefore the mathematical matrix keeping the machining information will have the respective values updated from false to true. At the same time, it will be recorded that the removed volume is the volume of 12 elements (process waste) and that the material removal rate (process speed) is the volume of 12 elements divided by the time elapsed since the previous system snapshot (time between 2 data input samples). From a cutting tool perspective, it will be recorded that the wire's active part removed 12 elements in the elapsed time and that it is feasible to machine 12 elements without the wire breaking (for future comparison). It becomes clear that there are unlimited ways to use basic machining information if this is combined with relevant theory or additional monitoring data.

On the right part of Figure 18 an additive process is depicted. In this case, the red cubes represent the material that has been extruded and placed on the printer's table. The simulator then 'knows' at any point what is the expected shape of the printed part. If significant differences with the physical part are spotted (for example by comparing the digital part with a picture taken by the 3D printer) an error signal could be sent to the digital twin. Then the specific point of failure together with the snapshot's printing parameters could be added to a process failure report. In that case, it could also be recorded that printing elements with a

specific printing speed or alternatively, printing elements at a specific temperature is an indicator of over-extrusion which leads to rejected parts. The digital twin can then be used to suggest that steps are taken to reduce the temperature of the filament and slow down the process in any future attempt to print another (different) part with these settings.

The same idea would apply to other processes. Although not every process has been examined, the strength of the method presented by the author is that it performs the actual process in the digital world and all calculations are based on the process itself and not on some model that connects input data to output data or that describes the process based on relevant physical phenomena observations. Moreover, since the simulator receives updated model data, other simulation models may run in parallel and through the simulator's learning module provide their own estimation that can be inserted as an extra parameter into the calculations.

3.4.2.4 Calculating derived data

From the moment that raw data arrives at the simulator until it exits the simulator as process information there are different stages that it goes through. Initially, it is pre-processed within the input interface where data is converted from unusable (for the simulation engine) raw data to ready-to-use input data. Then by running the virtual CNC process the input data is converted to results that provide valuable information about the process. This information however is not necessarily the information required by the digital twin. derived data is the stage where the gap between the virtual process results and the digital twin requirements is bridged. This is done by using theoretical or empirical formulas that calculate the exact parameters that the simulator should estimate. It could be argued that the calculation of derived data is part of the CNC virtual processing stage but the key difference is that the virtual process is generating process information while the derived data stage is moving the simulation information further up the data, information, knowledge, wisdom (DIKW) pyramid.

There are numerous ways to use the CNC virtual process results but in general, there are two main categories:

- n-sample-based methods
- 1-sample-based methods

n-sample-based methods focus on data trends and the overall characteristics of a series of samples that are ordered and as the name implies they require more than one sample as input. These samples can be generated during part of a process run, a full run or from multiple runs. On the other hand, 1-sample-based methods analyse the characteristics of each sample independently and calculate a new result for each sample without considering trends or overall process trends or performance. n-sample-based methods are mostly statistical in nature since they rely on process data aggregation and comparison with other runs of the same process (for similar or different parts). n-sample-based methods provide an overview of the process and are very useful in highlighting differences between machines or production lines. This is key for production management and planning and is often a requirement of external systems that support operations at the facility level. Examples of n-sample-based data models are: process time and accuracy deviation; cutting tool wear effects; the comparison of historical usage between resources; the estimation of process reliability. n-sample-based methods are also very effective when the precision of the simulation engine is low but good enough to provide a process overview. For example, in a CNC milling process, the simulation engine may use a rough mesh to estimate the usage of a cutting tool and the real-time shape of the part. This may not be accurate enough for the useful estimation of the spindle load, but it is enough to provide a history of cutting tool usage and support cutting tool inventory management.

Figure 19 shows an example of data processing during the production of a part. 1-sample-based calculations are done every time a new sample is generated by the monitoring system while the n-sample-based are done at the end which is when enough samples are available to feed the relevant models.

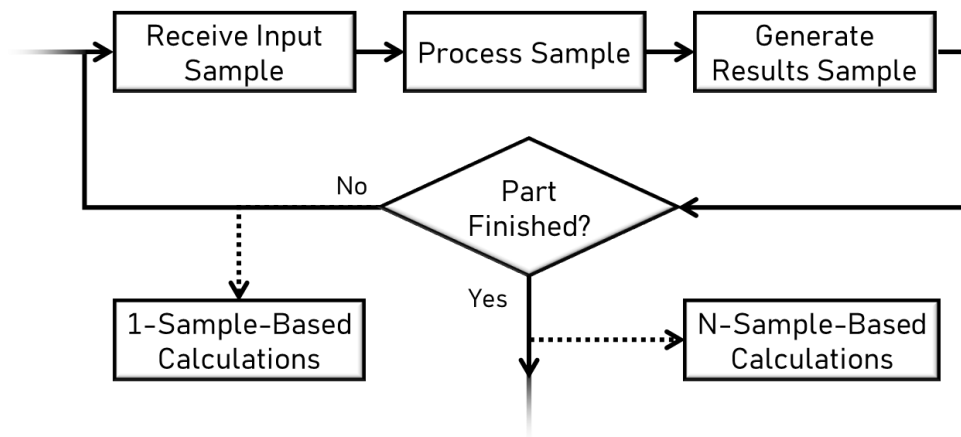


Figure 19 1-sample-based vs n-sample-based derived data

1-sample-based models are essential for results that are required interactively. The simulation engine runs in an event-based manner therefore the results come in the form of samples which are produced every time an input sample is converted by the virtual CNC process. The range of applications for these models is very wide. They can be simple data transformation formulas doing value conversion to appropriate units, calculations of removed material weights and volumes or estimation of the remaining quantity of consumables. 1-sample-based models may be theoretical formulas connecting results with other process parameters, such as the forces on the cutting tool or power consumption. They may utilise advanced statistical or machine learning models to provide average kerf estimations in a cutting process or surface finish quality or even be the interface between two other models typically one from process replication and one from derivative data calculation.

Regardless of the application, derived data parameters that are calculated by 1-sample-based models are more technical and therefore mostly directed towards machine operators and towards digital twin modules related to process control. This is due to the key difference between the time that the results become available in 1-sample-based and in n-sample-based approaches. The former produces real-time results and can tackle issues as soon as they occur while the latter can spot problems at the end of the process which may be 'too late' for autonomous systems. On the other hand, 1-sample-based models are sensitive to noise and

therefore can be less reliable, especially in cases of input data generated by sensors.

A way to mitigate this issue is a hybrid solution where samples are merged so that the aggregated sample is less affected by noise. In cases of high sampling rates, it is often appropriate to apply a smoothing technique, such as moving averages, filters, or regression algorithms, which will aggregate the samples over a short period of time. This may typically be the last second/minute so as to not affect significantly the performance of control-related tasks. These methods can also be applied by the input interface to reduce the number of samples that have to go through the virtual CNC process (lossy data reduction), improve the quality of each sample and finally increase the reliability of the results. Last but not least, from a programming point of view, derived data can be run as a separate service in a different computing system. This will free up resources in the system of CNC virtual processing and enable a more detailed replication of the physical process.

To sum up, virtual CNC processing has been created by the author to provide a way for the simulation engine to run a physical process in the digital world, but it is not producing enough information to support decision making. The derived data element fills the gap and extracts all information that is needed to make simulator results complete and appropriate for the application. At the same time, the approach gives a high level of flexibility and can therefore be used in a wide range of applications with different data/information requirements.

3.5 LEARNING MODULE

Machine operators go through training before running a process on a new machine. Training is required to learn relevant theory, learn the machine's controls and finally learn how to combine theory and controls to produce a part. Experienced machine operators may have run the process for different parts many times and over time they will have observed the differences between the theoretically estimated result and the actual result. The differences can be related to the way that the specific machine runs, the wear and maintenance of a specific machine, the workpiece material properties and the reliability of the process. Ultimately, an

experienced operator when compared to an inexperienced one has accumulated much more knowledge through an iterative process where the differences between process estimations and process results have been observed and can then be taken into consideration for the next runs. Nowadays, the wide range of available sensors can translate and/or capture human observations to data that is instantly available for processing. Experienced operators do check process monitoring values, such as the percentage spindle load, because a sensor measurement although being a very small part of the big picture is nevertheless a trustworthy quantifiable indication of the process state.

The anticipated function of a next generation simulator should follow the same evolutionary process as machine operators. It runs initially with the theoretical estimations and as it receives actual process data it gains experience and therefore adapts its estimations accordingly. In addition, like a human operator with knowledge gaps that are filled in the long term by observations and experience, the simulator starts its operation with an adequate theoretical background that is enhanced by monitoring data analysis. A key difference however is that a human operator may need weeks or months to accumulate the necessary experience while a simulator can gain the necessary experience with data generated within minutes of running the process. Overall, the examples of managing this introduction point to constantly running supervised learning models that aim to support a simulator teaching process. The architecture designed by the author relies on this process, and it is one of the key characteristics of this approach that distinguishes it from traditional simulator developments.

3.5.1 Learning module operation and characteristics

Figure 9 (the architecture diagram) shows that through the database, the learning module is responsible for providing the updated parameter values to the simulation engine. The architecture uses the simulation engine to embed process-related theory into the simulator and the learning module to embed empirical knowledge acquired by analysing process monitoring data. This is therefore the mechanism through which theoretical and empirical knowledge are merged. A consequence of using this mechanism is that the learning module becomes responsible for the

accuracy of the simulation engine results and as it will be explained in the following paragraphs, it can dominate the simulation process by completely changing the calculation model that the simulation engine should run.

There are more than one calculation formulas whose parameters are updated by the learning module so in practice the learning module is a group of submodules, each one responsible for updating a specific formula of the simulation model.

Figure 20 shows the typical data flow within one of these submodules.

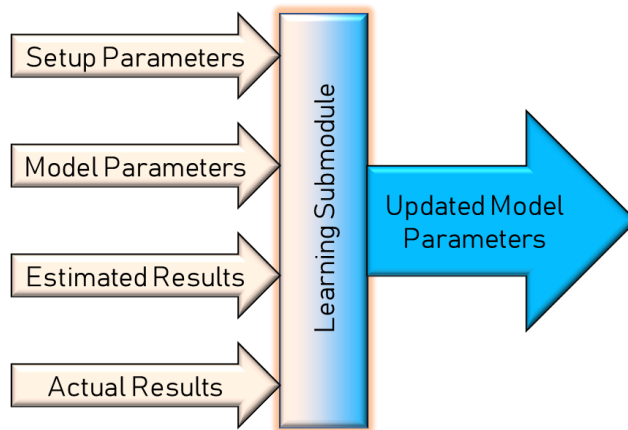


Figure 20 Learning submodule data flows

Learning submodules have one aim; to minimise the error between simulation engine's predictions and actual values. This is a typical machine learning (ML) problem and there is no restriction set by the architecture on the type of models or methods to use. The submodules only have to comply with design restrictions that ensure their seamless integration into the simulator.

Learning submodules use supervised learning because the goal is to predict values using historical datasets containing input and target data samples. In a straightforward development scenario, the submodule has to find the relationship between the process setup (input) and the output produced by this setup (target). It is outside the scope of this work to propose specific ML models or techniques but from an architecture point of view, the selected models should comply with the reliability, accuracy, and computing power requirement of the simulator. Nowadays, there is a long list of implemented ML models in a wide range of programming languages which reduces the need and effort of developing ML models from

scratch. This enables the developer to create a testing bed where different models are assessed through automated routines instead of selecting them manually.

A concept for submodule development is to add a data transformation layer to the submodule where the dataset is formatted according to the programming needs of available ML model libraries. In the majority of cases, the required format is arrays of values which is also the format of the datasets being kept in the computer's memory (meaning no transformation is needed at all). Then, through an automated iterative process, all available ML models are tested and the best one is selected for the specific submodule. The same process is repeated for every parameter that the learning module has to update and ideally, the process should be repeated for the submodules that do not perform well with new datasets. Selection and model assessment can be an independent software module, developed separately and used for all learning submodules. This distinction however facilitates development and does not suggest an alteration in the architecture. The ML model selection module is not an independent structural element but only an independent software part belonging to the learning module.

The input-target scenario described above is a simplification that assists in the explanation of the architecture but is not enough to cover simulator needs. Figure 20 depicts model parameters entering the learning submodule which can be briefly explained as the parameters of the submodule's model. To better understand the context within which the submodule and its ML model runs a big-picture analysis is needed. The idea adopted in this research of a next generation simulator is based on the concept that the simulator is a data consumer. Consuming data means that the data entering the simulator is not stored in a database as raw data after it is processed. Going back to the human operator example, the operators observe the process, translate observations to information and in combination with the process result they create a new experience which is the knowledge stored in their memory/brain and there is no need to re-watch the process or go through monitoring data to produce the next part. Every time they choose a machine setup, they recall the best parameters from memory (experience) and run the process with a better-tuned machine. The simulator does the same but instead of storing

the information in a brain, it stores it in the database in the form of process parameters. Then these parameters contain all knowledge derived from previous process runs so there is no need to store the raw data used to run the simulation and generate this knowledge. This is what 'model parameters' represents in Figure 20. In the idealised example shown in Figure 21 a mechanism of knowledge storage is presented.

ML Model	Parameter Update	Database						
$\begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{bmatrix} * C = \begin{bmatrix} 10 \\ 20 \\ 30 \\ 40 \\ 50 \end{bmatrix} \rightarrow C = 10$	No previous data	<table border="1"> <thead> <tr> <th>Parameter</th> <th>Value</th> <th>Observations</th> </tr> </thead> <tbody> <tr> <td>C</td> <td>10</td> <td>5</td> </tr> </tbody> </table>	Parameter	Value	Observations	C	10	5
Parameter	Value	Observations						
C	10	5						
$[2] * C = [32] \rightarrow C = 16$	$C = \frac{10 * 5 + 16 * 1}{6} = 11$	<table border="1"> <thead> <tr> <th>Parameter</th> <th>Value</th> <th>Observations</th> </tr> </thead> <tbody> <tr> <td>C</td> <td>11</td> <td>6</td> </tr> </tbody> </table>	Parameter	Value	Observations	C	11	6
Parameter	Value	Observations						
C	11	6						

Figure 21 Example of parameter update (equal weights)

The knowledge from the first dataset of the above example is that parameter C has the value 10 and this has been derived from the experience of 5 observations. Then another dataset arrives with a different value for C. The new observation is also integrated into the parameter value with equal weight to previous observations. Finally, in the database, value 11 is the knowledge of the simulator after processing 6 observations. Therefore, instead of keeping 6 observations that need processing every time the submodule runs, the simulator keeps the parameter value and the number of observations. The same would apply if the parameters stored in the database would be the weights of a neural network that has been trained with hundreds of thousands of observations. The neural network would contain all knowledge extracted and therefore the need for storage resources to keep the initial raw data would be minimised.

After the model parameters, the last input data type entering the submodule is the estimated results. It is the least obvious requirement since the ML model is trained by input and target values and it can maintain its capabilities by retrieving every time the last known parameters. The problem is that in many cases, the setup parameters (input) and the actual results (target) are not directly related. Modern

machines monitor dozens of key process parameters, but it is not guaranteed that these parameters are enough to estimate all output parameters requested by the digital twin. For example, based upon the continuous monitoring of the loading on the spindle motor the internal control of a milling machine may slow down the process if the cutting tool is worn. A learning module that does not have data related to the cutting tool protection mechanisms would identify this slowing down as a cause of increased cycle times and may propose to increase the feed rate since the spindle load has been managed so it is within acceptable values. Such an increase would eventually lead to a catastrophic failure of the cutting tool which is a result of the inability of the simulator-based supervision system to fully 'understand' the operation of the machine.

To prevent such issues, the learning submodule should test its accuracy against the full process and not only the narrow-scoped training datasets. This is done by using an objective function in the machine learning process that minimises the difference between simulator estimated results and actual results instead of focusing only on the actual results without 'knowing' the behaviour of simulator estimations. This is a relatively long process (compared to simple input-target relation), but it is necessary for an accurate and reliable simulator.

A second more obvious reason for supplying the learning submodules with estimated results is related to the separation of the simulation engine into two parts (Figure 13). The process' digital replication results are the input for process data analysis. The latter is also using parameters that are updated by the learning module therefore the replication results are the input for the learning submodule.

As a last note, because of the nature of the ML models, it is frequently the case that the whole model is supplied by the learning module and the process data analysis simply runs the supplied model (in contrast to having only one model parameter updated). This is accepted by the architecture but introduces the need for balancing between the learning module and the simulation engine.

3.5.2 Learning module and simulation engine collaboration

The last point in the previous paragraph leads to a plausible argument; if the learning module can build the full calculation model for parts of the simulation

engine, then why is there a separation between the two and why is learning not part of the simulation engine? The answer is found after examining the purpose of each simulator element. The simulation engine without the learning module would always receive the same parameters from the database and therefore would be a static system. A static system doesn't have memory and therefore it cannot adapt to changes. Even if the perfect simulator existed this would be accurate only for a static deterministic physical system. All physical systems evolve, and their performance typically deteriorates as they reach the end of their life so a static simulation system would only work for a very short time. The learning module on the other hand is only focusing on improving ML models. There is no direct engagement with the simulation engine operation and there are no results produced by the learning module itself. The only way that the learning module could become an active part of the simulation engine would be to merge them into one simulator element.

Merging the learning module with the simulation engine is the current state of the art for simulators used in Digital Twins. The issue that arises is that the simulation engine must run in real time and its calculation models need to have fixed parameter values for the time that they are used. The learning activities must be done either in advance of the process with (at best) synthetic data or after the simulation engine's operation completion. In both cases the latest datasets are processed, the ML models are updated and then the new ML models become available for the next machine run. It quickly becomes apparent that the approach although significantly better than traditional offline simulation is still inefficient, and the learning process unnecessarily blocks the simulation engine.

Overall, the architecture produced by the author in this research separates the two elements to allow them to work independently. The simulation engine needs to be fast while the learning module can run asynchronously and update the parameters at its own pace even by running on a different computer. In extreme cases, this could allow the complete substitution of the learning module by another one that has been developed and trained in a different environment. This type of flexibility becomes very relevant in cases where part production is moved to a new location

with different machines. The learning module captures all knowledge from the initial production site so an accurate production replication means that the calculation models of the simulator have similar values. If the values are not the same this means that the machines run the process differently and the difference can be spotted easily by checking each parameter value separately. The same scenario would apply when a company runs multiple machines and they would like to compare them. The latter is typically done by machine operators that know the specific characteristics of each machine, but it is a critical capability when a production runs autonomously.

Despite the effort of the architecture to provide independence to the learning module, this is not always possible since the simulation engine runs the ML models that the learning module 'builds'. If the model is a formula that has only a parameter value updated, then the simulation engine will always need the same time to calculate the formula results. If instead of the simple formula a deep neural network is used then updating the neural network structure will drastically change the time that the simulation engine needs to calculate the network outputs. These issues occur because the simulation engine focuses on calculation speed and the learning module on accuracy. A restriction is therefore required to ensure that the learning module will not impact overall simulator performance. This restriction can be a developer's convention that is implemented either by running verification tests on the learning submodules or by allowing only a fixed number/structure of parameters to be stored in the database. To sum up, the learning module can have an impact on a simulator's performance and the developers should be aware of it during the simulator's design phase.

3.5.3 1-sample-based learning

The learning module provides support for all calculation models of the simulation engine that need updating and therefore it supports both 1-sample-based calculation models and n-sample-based models. 1-sample-based learning is straightforward as a process because the data received is already pre-processed and the learning submodule only needs to test which ML model is the best fit. To clarify why the learning module contains only ML models the simulation engine

calculation formulas should be examined. If a simulation result is calculated using equation 3-1 then the simulation engine requires the sample data coming from the monitoring system of the physical system (input1...N) and the value of the learning parameter (P_{learn}) that is retrieved from the database.

$$R = f(input1, input2, \dots, inputN, P_{learn}) \quad 3-1$$

Where:

- R : Result
- f : Calculation formula
- $inputN$: The input data variables received from the monitoring system
- P_{Learn} : The parameter that is updated by the learning module

To simplify the example, it is assumed that the corresponding learning submodule receives a dataset for the first time (so there is no need to consider the previous value of P_{learn}). The submodule will need to provide a P_{learn} that minimises the cost function 3-2.

$$MSE = \frac{1}{s} \sum_{i=1}^s (R_i - f_i(P_{learn}))^2 \quad 3-2$$

Where:

- MSE : Mean Square Error
- s : Number of samples in the dataset
- R_i : The (actual) result of sample i as received from the monitoring system
- f_i : Simulation engine's calculation formula (inputs are constants)
- P_{Learn} : Learning submodule target parameter

It is noted that in 3-2 there is $f(P_{learn})$ instead of $f(input1...N, P_{learn})$ because each sample contains the inputs that feed the formula (known values). When input values of a sample are inserted into the formula then each sample creates 1 unique formula with only one variable P_{learn} . The usage of the term 'variable' for P_{learn} is not accurate for all cases. If P_{learn} was just a variable, then the learning submodule would simply find the value that minimises the MSE. That would be true if the result R has a linear relationship with the inputs (therefore P would be a constant

parameter in the formula). In most cases, however, P is also a function of the inputs.

$$P_{learn} = g(\text{input1}, \text{input2}, \dots, \text{inputN}) \quad 3-3$$

Where:

P_{learn} : Learning submodule target parameter
 g : Submodule's calculation formula
inputN The input data variables received from the monitoring system

And combining 3-2 with 3-3

$$MSE = \frac{1}{s} \sum_{i=1}^s (R_i - f_i(g(\text{input1}, \text{input2}, \dots, \text{inputN})))^2 \quad 3-4$$

So finally, the learning submodule has to minimise MSE by finding the best model for P_{learn} calculation. Minimisation of MSE should consider all samples that the simulator has received from the beginning of its life. Finding the best model for P_{learn} means that the submodule has extracted all knowledge contained in the information extracted by the physical system which generated the data. The above explanation does not intend to provide a technical solution, but it aims at clarifying the role of the submodules in 1-sample-based simulation formulas.

It becomes apparent that submodules are a key part of the architecture and at the same time, they are contributing factors to the complexity of the simulator. A key challenge for the submodule development is the selection of the relevant (directly or indirectly) input parameters to P_{learn} . Using all available inputs would have a significant impact on the computing resources the simulation engine needs to run the calculation model. As explained in 3.5.2 a neural network with fewer input parameters has fewer nodes per layer and therefore it requires fewer calculations to produce results. Especially in cases of deep learning models, an extra input parameter means thousands of additional calculations because of the extra nodes. This is also the case in linear regression although the extra number of calculations is typically insignificant.

Finding the relevant parameters for a dataset is a frequent problem in data science. The two most common approaches are based on Garson's algorithm or on sensitivity analysis. A brief comparison of key algorithms can be found (Olden and Jackson 2002). Garson's algorithm is based on the idea that if an input parameter is irrelevant then the absolute value of the weight of this parameter will be close to zero. This happens because neural network training nullifies the weights of parameters that have no impact. The same applies to linear regression algorithms which to some extent have similar characteristics to the formulas connecting the nodes of a neural network. Sensitivity analysis is applied by varying each input variable across its entire range while keeping all other input parameters constant. Then the individual contributions of each variable are assessed and the parameters that cause little to no variation are marked as non-contributing.

Both methods require a dataset to test the parameters and then to modify the way that the ML model is stored in the database. Another issue arising is how can the quality of the dataset be verified since it can be the result of a specific machine setup that is not affected by an otherwise important parameter. For example, a milling machine may generate a dataset while machining with a cutting tool that doesn't need coolant. The generated dataset will not be representative of the machine but only of the specific setup since it will not contain any information about parts that are cut using coolant. In addition to specific cases, this type of testing goes against the idea of a continuously evolving simulator because if a ML model is based on a specific dataset, then future changes will not be embedded in the simulation calculations. From an architecture point of view, these issues point out the importance of separating the simulation engine from the learning module. The solution is that in the future maintenance of the system, the developer may use any of the aforementioned methods on models that have been trained with all process data. Then, relevant modifications in the learning module and database can be made without affecting the simulation engine which will continue running without disruption by calculation model changes. This could be done through a learning module embedded process that checks each learning submodule for redundant input parameters.

A very common problem for the learning module is the noise in monitoring data. Many methods identify and remove noise but no method can ensure that no actual data is being lost during the denoising process. The golden rule for dealing with noisy data is to collect more data. The architecture is based on collecting and consuming all available data so in the long term it naturally filters the noise from the simulator itself. For further reading into the issue, paragraph 3.4.2.4 lists methods to tackle derived data noise sensitivity. These could also be applied to the learning module input data but overall, identifying the best method to deal with a noisy data source is not directly related to the simulator architecture and therefore it is outside the scope of this work.

With the 1-sample-based learning, a simulator which is a data consumer with a knowledge extraction and knowledge storage mechanism has been fully described. A macro-view of the architecture shows an additional benefit which is the distribution of development tasks. Although the simulator is complex, the development of each element can be done independently with specific targets. This simplifies the development process and enables collaboration between experts in different fields. In the end, just as the physical machine with separate subsystems transforms low-value raw materials into a high-value product so the simulator transforms the low-value raw process data into virtual process information and then into high-value knowledge stored in the database.

3.5.4 n-sample-based learning

From a learning point of view, n-sample-based learning does not have significant differences from the 1-sample-based one. The search for the best ML model is the same but with different factors affecting the type of model. This section is about learning submodules that receive samples reflecting a full process run or at least the result of a process stage (therefore excluding cases of simple data aggregation).

n-sample learning means that at least a few samples are available for ML model training something that is not the case with 1-sample-based learning. Most neural network-based models require at least hundreds of samples for training, so it is typical in n-sample-based learning (at least in the initial stages) that multiple

linear regression or general statistical methods are used. This is not necessarily a problem since n-sample-based learning is related to simulation engine results that reflect performance or show characteristics of a full run (typically to produce a full part). These results are frequently used to compare the manufacturing of two different parts and are very useful for production planning. Examples of parameters that the n-sample-based learning submodules focus on are the average part production time difference, finished quality (compared to other parts or machines), consumables and consumption per part or per operating hour, and average difference from theoretical values. The latter example will be further analysed since this type of analysis goes much deeper into the process characteristics.

n-sample-based learning parameters are in general not process-critical so the learning process may have offline characteristics. Database stored values for learning parameters are retrieved at the beginning of the process when the digital twin gets an estimation of the process performance/characteristics. They are also acquired at the end when the latest data is compared and the model parameters are updated. A delay of minutes or, depending on the process, hours does not affect the performance of the simulator. If the type of process run is new (for example new part or alternative machine) then theoretical values are used. The architecture promotes again the same pattern as in 1-sample-based learning which replicates the human operator learning process (starting with theory, evolving with experience).

The unique characteristic of n-sample-based learning is its capability to see the big picture of data analysis. Values of single samples are viewed within the context of the dataset they belong to and not as an isolated training sample of an ML model. For example, a parameter showing that the performance of a cutting tool is decreasing as the cutting tool wears. For demonstration purposes, it is assumed that there is no input showing the hours that the cutting tool has been used. The 1-sample-based learning will update the parameter with a value close to the average value describing the process. n-sample-based learning will process all samples at once and will therefore produce a ML model that describes a trend and not an

average value. It is a key ability of the simulator to identify these trends because digital twins are not only used to supervise and support process control but they are also used for prediction. Predictive maintenance, cutting tool management, and production scheduling are common activities that depend on an accurate prediction of the physical machine's behaviour.

Before considering the details of the n-sample-based learning mechanism a clarification is needed on whether data is consumed immediately (as implied in 1-sample-based learning) or if it is stored until it is ensured that there are enough samples for the n-sample-based learning submodules. The architecture promotes the immediate consumption of data, but it does not ban or block temporary or long-term storage. The drawbacks of storing long-term have been discussed in 2.3. For completeness, it should be added that until the simulator has reached maturity, meaning it is deployed and its performance targets are achieved, all data should be kept for further development, improvement and testing of the learning submodules. A compromise to storage is temporary storage. It offers more value extraction opportunities compared to immediate data consumption without the negative aspects of long-term storage. A simulator may keep all samples from a process run and feed with these samples an n-sample-based learning submodule. In addition, if specific data characteristics are repeatedly observed the simulator can store relevant information for the developers to assess whether new ML models are required. Simulator maintenance is ensuring seamless operation and at the same time, it is an opportunity to identify structural improvements based on the operation of the physical system since previous maintenance.

Learning is based on the comparison of estimated values with actual values. In 1-sample-based learning, the estimated value is calculated after the input data of a sample is available. The estimation, the actual value and the inputs are strictly mapped to each other. In n-sample learning the estimation of the process behaviour is done before the process starts. Based on the process setup, the simulator then calculates both inputs and results of the process in the form of samples. When the actual process runs, it produces samples that are not equal in number nor synchronised with the estimated ones. When the learning submodule

compares the differences between estimated process values and actual ones it must ensure that the machine did indeed run the process that it was expected to run. If for example the machine breaks down in the middle of the process or if manual intervention dramatically changes the way the physical system runs then the learning submodule must be able to realise that the data should not be used for learning since it is representing a different process. After verification of suitability, the submodule should synchronise the values to make them comparable. Finally, based on the differences between estimated and actual values the learning submodule will train its corresponding ML model.

Finding the level of similarity between estimated and actual values can be done with methods from the field of signal processing. The series of values of an input parameter or result shows the behaviour of a system or describes a phenomenon. This makes the value series a signal (Priemer 1990). Signal processing methodologies can trim, synchronise and normalise the signal so the learning submodule is able to use the supplied data for training purposes. The architecture of the simulator is not compatible with a specific method. Any method that can quantify similarity and synchronise the signals is acceptable for simulator development. Due to the wide range of available signal processing methods produced during the last 70 years, it is outside the scope of this work to review the literature of the field. However, for completeness, an example method will be discussed below.

Dynamic Time Warping (DTW) was in its basic form proposed by (Bellman and Kalaba (1958) and began gaining popularity after its application in speech recognition (Sakoe and Chiba 1978). A recent review lists a high number of DTW variations and extensions (Yadav and Alam 2018) and discusses its popularity due to its efficiency in measuring time series similarity (Senin 2008). DTW method is mentioned here because it quantifies the difference between 2 signals that consist of discrete samples and at the same time it connects the samples of the 2 signals so a sample-to-sample comparison can be done.

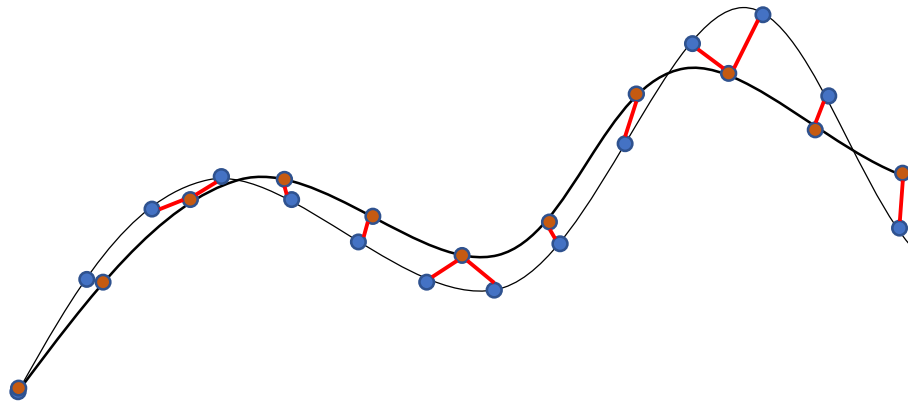


Figure 22 Mapping of samples of 2 signals with DTW

DTW is suitable for the needs of the architecture since with one pass it can prepare the n-sample data for the learning submodule. If the distance value (value showing how different the 2 signals are) is above a threshold defined by the developer, then the estimated values and the actual values are considered irrelevant and the submodule does not use the data for training. Otherwise, a sample-to-sample comparison can be done enabling all sorts of n-sample processing which can reveal discrepancy trends, consistent errors or even identify the manual changes to the process. An example of the latter is if the operator pauses the process for 10 seconds then one sample of the estimated process will be connected to a group of actual samples filling 10 seconds in time. An application of the method is demonstrated in section 4.5.3.

At the beginning of this research work, in addition to the learning module, a Data Synchronisation Module was designed by the author as an independent element of the architecture. This was because of the high potential for value extraction that n-sample-based learning has (if combined with signal processing methods). However, a strict definition of simulator elements makes data synchronisation part of the learning module. Therefore, the proposed design for the learning module is depicted in Figure 23.

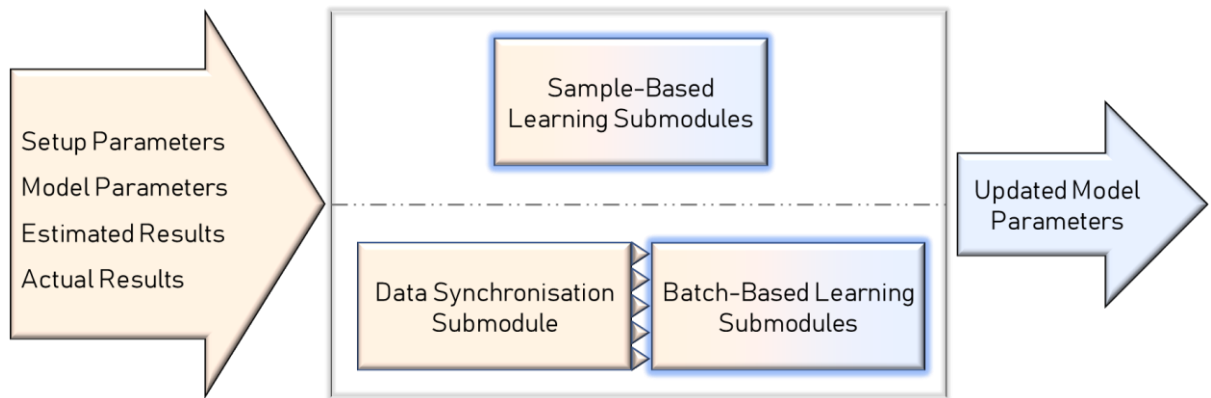


Figure 23 Learning Module Architecture

3.6 DATABASE

The simulation engine and the learning module are the two main actors of the simulator architecture, but they need support in order to communicate with external systems and with each other. The database is the main internal communication channel through which process information, simulation results and calculation models' parameters are made available to all architecture elements. The database is 'an organized collection of structured information, or data, typically stored electronically in a computer system'(What Is a Database | Oracle. 2021). This element is typically associated with the database management software system but any type of organised storage is acceptable for the functionality of the architecture. It may seem efficient to establish direct connections between internal modules due to the reduction of communication lag and the minimisation of the programming required. However, direct connections bring a number of issues that hinder scalability and violate principles that the architecture is based on. The parameters that are exchanged between the simulation engine and the learning module could be measured in thousands even in simple simulation scenarios where neural-network-based models are used. Regardless of the exact number, for every model and probably every parameter there should be a separate connection between the simulator elements. This means that each element should have receptors programmed specifically for the parameter to be exchanged. If a learning submodule model changes (which is a key part of the simulator evolution) then the receptors would need reprogramming for all elements that are supplying or

reading data from this submodule. Another consequence of direct communication is that the simulator elements cannot be developed and run independently. Since the simulation engine must run parallel to the physical system then the learning module would also have to catch up with real-time which would inevitably introduce mechanisms to separate the data exchange from the actual learning process. Last but not least, the simulator would operate less transparently because the information exchanged among the modules would not pass through a layer that can monitor or even filter internal operations and if required inform the developer about data exchange issues.

Introducing a database to the architecture adds a high level of flexibility to the simulator by making the development and operation of each element independent. The latest data/parameter values are always available either for the elements that request to read the specific value or for the elements that request to update the value. From a development point of view, a database can reduce the development effort because Database Management Systems (DBMS) provide ready solutions for storage and data-access speed and security. As mentioned earlier, the operation of the simulator is more transparent because all data exchanges can be viewed by monitoring the database. DBMS come with interfaces that are essential tools for verification and offer a way to check the 'internals' of the simulator without the need for modifications that expose data streams.

Regarding DBMS type selection, Relational Database Management Systems (RDBMS) are a good choice because the simulator's data is formatted and structured. Since the intention is to consume and not store all data the tables are relatively small thus allowing for fast data access. On the other hand, NoSQL would benefit cases where raw data is stored by multiple machines and the learning module operates offline or in the cloud to train its ML models. It should be noted that if the simulator is running locally and the raw data is dumped by the monitoring system in large files then there is rarely a need to change the format of the files or store them in a database. For example, the implementation in Milling presented in Chapter 4 keeps only the location of the raw data file in the relevant database tables and leaves the file intact.

Going back to architecture-specific storage requirements four simulator elements are interacting with the database (Figure 24). Although no element is restricted to only reading or writing, there are some typical data flows deriving from the role that each element has.

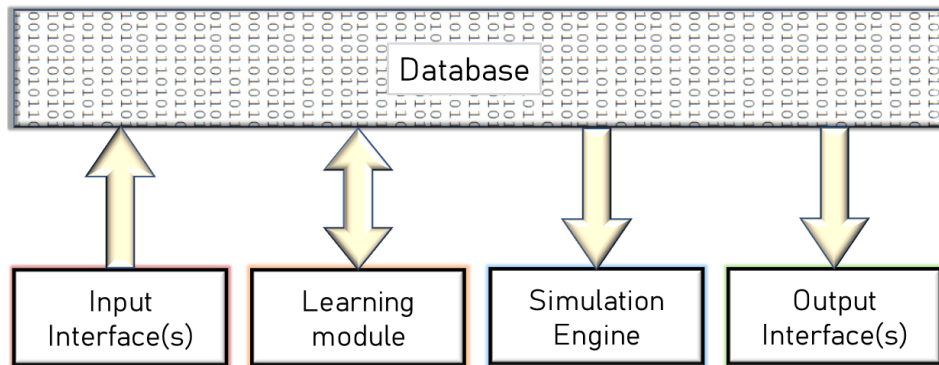


Figure 24 Typical data flows to and from the database

The input interface is the first element receiving and distributing data inside the simulator (after pre-processing). According to Figure 9, data flows are towards the simulation engine, the learning module and the database. Beginning from the latter, the typical information stored in it is about the process setup, logging information, and as far as it is required monitoring data. The database should keep metadata describing the process that the physical system is running. Metadata provides context to the raw data received and it assists in separating the runs for different parts, part types, tools or in the generic form different modes and processes of the physical system. This is later used to group learning model data and results. In a typical simulation scenario, the input interface will register a new process run type which is identified by specific (setup) parameters. Then the learning module will be updating a specific set of values that are related to a simulation with the specific setup. When the input interface registers a new setup then the learning module will begin to train a fresh set of parameters specific for this setup. There are of course cases of global parameters that are always being updated however there should be a mechanism deciding the range of setups that a parameter is valid for.

A clarification is needed at this point regarding data going from the input interface to the simulation engine and the learning module without being stored in the database. The database element is the memory of the simulator which provides

flexibility in communications among the modules. Data that is consumed immediately and especially data that does not change in format or structure over time does not have to be stored in the database. The input interface is a service that triggers the simulation engine and supplies it with sample data in a ready-to-consume form. Contrary to the exchange of data between the learning module and the simulation engine in this case, there is no need for receptors, nor does it make sense to write and immediately read a value that will disappear. In special cases where the sample flow must be temporarily stored a system of dual databases may be appropriate. The additional database may only store data in the computer's Random Access Memory (RAM) and therefore provide data flow transparency while being fast and with zero impact on the system's persistent storage capacity. Overall, the usage of a database should improve the operation of the simulator and in regard to the input interface, the developer should select which data is worth keeping and which data should be directed for immediate consumption.

The element that is fully dependent on the database is the learning module. That is due to the submodule operations that read parameter values and metadata from the database and after model training, they write the updated values back. It could be said that the learning module is constantly modifying the database with its results and the database is the agent that publishes the learning module's results. In RDBMS, developing the schema for ML models that may change in the future can be complex since relational databases work well with predefined structures. A solution is to serialise the model or to store it in JavaScript Object Notation (JSON, 2017). This way, a table as in Figure 25 could hold the learning module updated models.

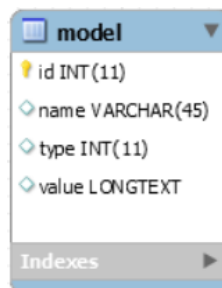


Figure 25 RDBMS table storing model data

In the above table, id is an identification number given so the model can be retrieved from the database, name is either a name or description for the model, type is the type of ML model so the learning module and simulation engine know how to read the value and value is the JSON object containing all model data. For example, if the model is a linear regression one where: $y = 5x + 32$ then the type will indicate linear regression and the value would be {"a": "1", "c": "32"}. If new data shows that a neural-network-based model produces more accurate results then the model type in the database would change and all values for neural network weights, number of layers, activation functions etc. would be stored in a (much longer) JSON object. JSON objects is one among many solutions which offer flexibility on par with a NoSQL database, but its performance is not the same as if the model formulas were split into separate table columns.

The unidirectional flow of data towards the simulation engine represents the update of calculation model parameters. The simulation engine reads the latest information from the database and as demonstrated in the previous example it can even build the calculation model on the fly. This is the same for decision-making or result calculation (Figure 12). Traditional simulators would not require a database to run because the calculation models are fixed. This can also be achieved with the proposed simulation engine if the default values for the calculation formulas are used instead of the values retrieved from the database. This of course would be done for compatibility reasons if older simulators are replicated or compared with a next generation one. Another exception to the typical data flow direction is if the results of the simulation engine are stored in the database before being processed by the learning module. This highly depends on whether the two elements run synchronously, or if the learning module runs offline. In the latter case, a buffer is needed to keep the engine results until they are processed. This is not the case however with results directed to the output interface. The output interface immediately forwards the results outside of the simulator and therefore there is no need for a data buffer. In any case, data buffers are used for temporarily stored data and they are not part of the architecture's database because they miss the

element of persistence which is required if for whatever reason the physical process is interrupted.

The last element that the database supplies with data is the output interface. This interface does not run any models but since it reports simulation results, all process information stored in the database is retrieved as part of the report. The database holds setup information, logging information and metadata which if combined provide the full picture of the physical system. Due to the wide range of potential implementations and reports required by a digital twin, there is no database specification for this element of the architecture. More information is provided in paragraph 3.7.

Up until now, the database is analysed as an internal interface which adds a high level of flexibility to the architecture. In addition, it has been mentioned that the database, due to its interfacing role, provides access to internal data transactions facilitating simulator verification. Another important role of the database is holding and transferring the captured knowledge. The learning module updates the ML models, but all data is kept in the database. A shared database among many machines can ensure a standardised management of the machines or it can expose the differences among these machines. The key difference compared to a human operator is that the database can be accessed over the internet from any system around the world. For example, if a product is designed in the UK but due to supply chain requirements the product is manufactured in several locations around the world, the machines running in the UK can have the same process supervision as the machines abroad. This will ensure that the same standards apply to the whole network directly by the machine. On the other hand, the network stores captured knowledge to a UK or cloud-based database. This means that the R&D department based in the UK can receive immediate feedback from the process and amend its

control without exposing process control details and algorithms to the local subcontractor.

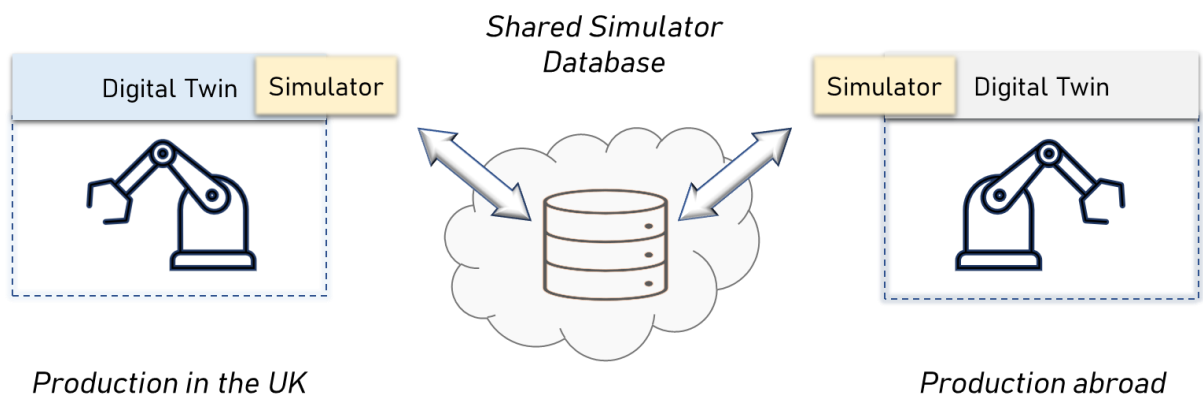


Figure 26 Knowledge sharing and capturing among physical resources

The portability characteristic of the database makes it a separate product sold by the manufacturer of the physical system. It is also a product that can be duplicated or that can be initialised and further developed for the intended application. The physical system has value because it can run the requested process, but the database can be sold separately as the know-how for running the process. This can be done through the digital twin that supervises the process and is capable of granting (or refusing) access to the physical asset user. If technologies from cloud manufacturing are considered many more cases of modern simulation technology exploitation can be developed. Reference to future applications however is done to demonstrate that the proposed architecture is future-proof and can open new horizons in data value extraction.

3.7 OUTPUT INTERFACE

Data that flows through the simulator is consumed by converting it step by step to information and finally to knowledge that is permanently stored in the database. At the same time, the simulator uses this knowledge in combination with the process setup and monitoring data to generate the results requested by the system it is embedded into. The element of the architecture that is responsible for publishing the results in the form of reports is the output interface. The output interface would typically run as a server that makes a live report available to the wrapper system.

However, the details highly depend on the specifications that the simulator must comply with.

Results coming out of the simulation engine are in a raw form making them non-usable by other systems that have not been specifically adapted. The most basic task of the output interface is to format these results according to what is expected by the wrapper system. Formatting could be anything from values' precision, and measurement units, to completely different representations such as reporting a process error when the result value is 'x'. In an ideal case, the simulation engine results would be publishable but since the majority of calculations are done in the simulation engine and the learning module it is more important to choose an efficient data format for the simulator's internal processes than using the publishing format.

In more advanced technology applications, the output interface acts both as a results post-processor and results publisher. Typical post-processing smooths, aggregates, or further enhances results. Beginning with smoothing, noise that passes the pre-processing stage will generate noise in the results. This becomes apparent in cases where outliers are not removed from input data to prevent losing valuable information. The results in these cases will be affected by the extreme values and if not handled properly (filtered or smoothed) they could propagate and lead to false instructions to the physical system's controller. Relevant techniques for smoothing have been discussed in paragraph 3.3.

Another characteristic of the simulation engine results is the speed that they are produced. High-speed input data will 'push' the simulation engine to produce results at a high rate. Not all parameters are real-time critical for the physical system supervision nor do all parameter values change at a high rate. In these cases, aggregating the values removes some of the computing burden of generating new reports and it does not add unnecessary computing burden to the systems that read the reports. Another aspect of aggregation is the combination of various sources of information into one report. Apart from combining data from all parameters the report may contain process setup information, accepted value

ranges or other present and historical information that is relevant to the operation of the physical system.

Combining various sources of information may introduce data synchronisation issues. Although in the majority of cases the simulation engine will provide the estimations and the database will provide non-time-dependent information, there are cases where the report compares historical with current estimated and/or actual values. Real-time data synchronisation is commonly based on matching timestamps of each data sample. If there is no reference point the problem becomes much more complex and there is a lag between the real-time and the synchronised data since a wide enough time window is needed to validate the synchronisation. From an architecture point of view, the output interface may perform such tasks, but it is recommended that these tasks are assigned to the simulation engine if the report supports a critical to the physical system operation.

From a user's perspective, the reports should be in a comprehensible format. If the user is another system, then the system should be able to import reported information. If the user is a human operator of the physical system, then the information should be more graphical. It is a key responsibility of the output interface to ensure that all results are communicated in the most efficient and complete way. This maximises the contribution of the simulator to the wrapping system and therefore to the end-user. Depending on the way that the simulator is deployed example reports may be XML files whose values are updated (such as in OPC or MTConnect server format), a JSON server providing JSON objects with the latest data, vectors parsed by a graphical user interface to generate graphs for human operators or even alert messages that are sent as an SMS or via email to remote systems or stakeholders. There are no restrictions regarding the means that the results are communicated however if the output interface runs on the same hardware as the rest of the simulator, computing load balancing should be taken into consideration.

Expanding further into the information communication channels there are three categories of ways to publish process reports. Firstly, traditional simulators typically create a results file on the hard drive of the computing system they run

on. As said earlier the file may have any format that is appropriate for the target user. Popular formats are EXtensible Markup Language (XML) and Comma Separated Values (CSV). In modern simulation systems, this method is still valid if the results are processed by other non-real-time systems. For example, in production planning it may be required that all systems are simulated and the results are used to find the best schedule. Alternatively, the results may be used to feed the elements of a production macro-model. This method is also suitable when the results are distributed to systems that are not connected to the simulator. This can be described as an export/import functionality where the simulator exports the results and any other software system that gains access to them can import the ones that are of interest and access them without any time or speed restrictions.

A second way to publish reports, which is suitable for scenarios closer to real time, is to use a broker (data buffer) that contains a certain number of reports (message queues) that wait for the receivers to get them. This approach has two main benefits. In real-time communications, if one system is slower or in general not synchronised with the report source then the broker will hold the reports until the other system is ready to read them. In other words, the lag between systems, temporary bottlenecks or minor disruptions can be overcome since communication is asynchronous. In addition to the flexibility of this method, the broker can act as a distributor to many unknown to the simulator systems without the need for the client systems to comply with the simulator-specific implementation (but only to the typically more standardised broker standards). The broker differs from the export/import functionality mentioned earlier since the intention is not to keep data permanently but to keep it for as long as it takes until it is used by the client.

The last means of publishing reports is through a direct connection between the systems. This connection is typically the fastest way to communicate the reports, and it would probably be the method of choice if the simulator is part of a digital twin system without a connection to remote systems. Direct connections can mean anything between sending and receiving the data from the source as soon as it is available (with limited safety mechanisms) to sharing the same memory resource which makes the report transfer instant. A major drawback of this type of

connection is the lack of flexibility since errors can disconnect the systems and several reports are lost until the connection is re-established. It can be said that this is the price to pay in order to have performance as close to real-time as possible.

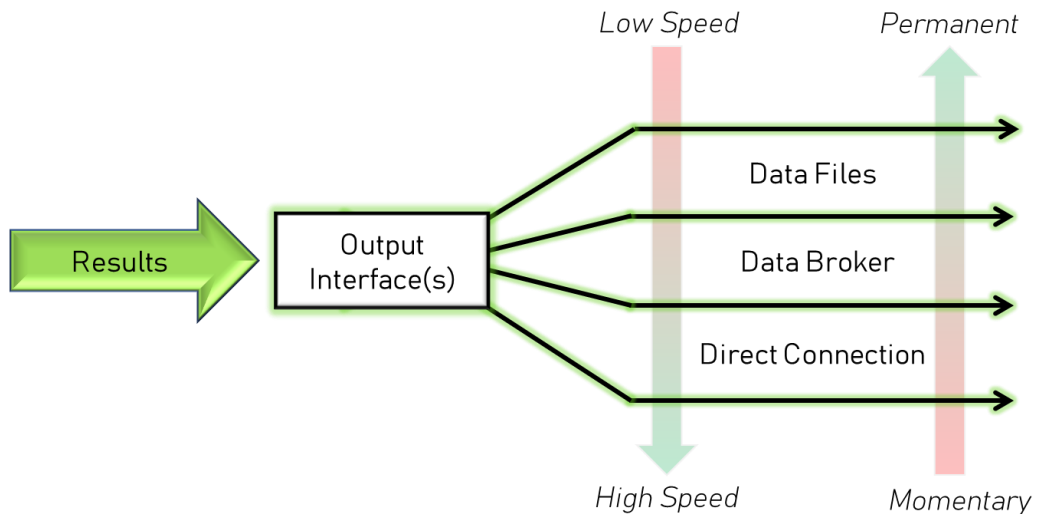


Figure 27 Summary of publishing methods

Simulator integration is the final step of simulator development. In practice, this is the connection of the Input and output interface to the wrapper system. For the digital environment a simulator is a black box that listens through the input interface and publishes answers through the output interface. Depending on the application different technologies could support these tasks although in modern digitalised environments usage of a server type is more suitable. For the input interface ideas have been presented in the relevant paragraph. For the output interface, some examples based on Figure 27 are:

A File Transfer Protocol (FTP) or other type of file server could be used to make the simulation results available. A server log or contents file, folder structure and/or files' metadata would be different ways to provide information so the user can locate specific physical system setups, dates of simulation runs, extreme cases or other characteristics of interest. This would be a good solution if human users access data without specialised software since the files can be downloaded and processed through embedded software of popular computer operating systems.

Database servers could offer a solution of permanent access that is faster than file servers. A database server, regardless of its type (RDBMS or NoSQL) has a structure that improves accessibility and can organise information in a way that client software can quickly search through the simulation reports and retrieve relevant information. The database server is not necessarily part of the output interface. The interface may simply write data to the server and then any other module or system can access the reports. That would be an example of a digital twin's database or a cloud database for any external application. The drawback of a database server is that the database clients need specialised software to access and process the reports.

A web server could be used for more direct connections to the simulator. The results are published as soon as they become available and the server clients can connect and retrieve the latest reports. Although this type of server could be using a database or a file server in the background to retrieve historical information, by itself it is suitable for temporary data buffering and flexible availability. If required, it can also support more direct connections to clients with technologies such as Server Sent Events (SSE) that push the reports to the clients immediately. This technology can also be used to host a GUI which makes it ideal when the simulator is a standalone solution and the wrapping system is just a GUI providing access to results.

Another solution which is gaining popularity in the last few years is an application server that hosts the simulator and provides access to users over the internet. This is similar to a web server with the difference that the access is with dedicated interfaces, and that the client processes the reports on the server and not locally (which frees client resources).

Finally, the broker solution to distribute data is probably the most popular in digitalised environments due to its high level of flexibility and the separation of the data generation layer from the message transfer layers. Technologies like Apache Kafka (Apache Kafka. 2022) have revolutionised the field and should be considered in cases where megabytes of data are generated and transferred per second.

The description of the above technologies is rather simplistic, but the intention is to provide context for the simulator integration and not to describe a technically complete solution. This work provides the architecture that makes the application of the above technologies efficient and finally, it is up to the developer to choose the most appropriate implementation tools.

3.8 ARCHITECTURE SUMMARY

A next generation simulator was designed and built by the author to support real-time, evolving and geographically spread digital environments. The presented architecture is the combination of key elements which deliver the functionality of a traditional simulator and at the same time, they create a digital entity with a human-like learning behaviour. In brief, the contribution of each of these elements is:

Simulation engine: Fills the simulation model formulas with input data and updated calculation model parameters to calculate the results. It does it in two stages; one that replicates the physical process using relevant theory and one that uses the results of the first stage to calculate any parameter that can be derived using ML models.

Learning module: Uses input data, monitoring data, simulation results and physical system setup information to adapt the simulation engine's calculation formulas. It comprises submodules each of which is responsible for the update of one parameter or ML model through supervised learning.

Input interface: Takes raw data and pre-processes it so the data is ready to be inserted in the simulation engine's calculation formulas.

Output interface: Publishes the simulation results in the format and speed needed by the simulator users (systems or humans).

Database: Keeps the updated parameters and ML models of the learning module that feed the simulation engine and form the knowledge that the simulator has accumulated during its life.

The proposed architecture can be widely applicable, and it is especially advantageous for digitalised systems that use digital twins to supervise a physical system/process. The simulators that adopt it fully utilise the available data while reducing the storage capacity needs. At the same time, they can use and share the accumulated knowledge more efficiently than human system operators.

In the next chapter, an implementation of the architecture is presented to provide further understanding of how the elements interact and collaborate to create a next generation simulation system. Due to the compromises a specific implementation requires, and the amount of work included in merging technologies from different fields of engineering it is not possible to demonstrate every aspect of the architecture. This limitation however is reflecting the genuine requirement for a team of experts to develop a simulator and not the potential of the architecture itself.

4 APPLICATION IN VERTICAL MILLING

The next generation simulator architecture engineered by the author in this research was presented in Chapter 3. This is laying the foundation for modern simulator development. In the same chapter, examples mainly from manufacturing were used to explain the nature of the contribution of each element of the architecture as well as how the elements work together to achieve the desired results. In the fourth chapter of this work, the architecture is applied to CNC milling. The application of the architecture and the details of its implementation, serve as proof of concept for different aspects of the architecture. Firstly, it shows how the elements can work together and can calculate accurately the simulation results while meeting the targets set in Chapter 1. Then, it provides an example of how the architecture promotes the continuous improvement of the simulator and its ability to embed new technologies. Finally, it shows how the architecture can contribute to more innovations by splitting development tasks. The intention is that this will allow for specialised developers and/or their software to create a state-of-the-art element without being affected by the rest of the implementation.

Chapter 4 presents the detailed CNC milling implementation. The discussion of results and the wider observations arising from this implementation are presented in Chapter 5. The source code of the simulator is split into four projects that can be found on GitHub under the following links:

Front end of simulator including GUI:

<https://github.com/harrycd/milling-vm>

Front end of simulator including web application:

<https://github.com/harrycd/milling-twin>

Backend of simulator including simulation engine and learning module:

<https://github.com/harrycd/milling-utils>

Backend of simulation including interface with the database:

<https://github.com/harrycd/milling-database>

4.1 PROCESS SELECTION AND SYSTEM BOUNDARIES

The proposed work has used a vertical machining centre (VMC) as a testbed. The monitoring and management of vertical milling is an extensively studied process which is an advantage when assessing the level of information produced, identified behaviour and usefulness of the results of the simulator. Results from other studies can be used in combination with newly generated simulator outputs to produce more accurate estimation of the range and impact of selected process parameters. An additional consideration was the integration of this research into an ongoing research program within which a VMC with an embedded monitoring system was made available. The machine is a Mazak Vertical Centre Smart 430A. The machining process data used in this thesis was acquired directly from a Mazatrol Matrix Nexus 2 CNC controller using an interface engineered within an associated research activity (Hill et al. 2019). The previous work (Hill 2020) had utilised spindle load variations as an indication of changes to tool condition during the repeated manufacture of a component. The effectiveness of the deployed system was confirmed by the measurement of the manufactured components using a coordinate measurement machine (CMM). However, this CMM based approach cannot be deemed to be realistic as an on-going methodology since such CMM measurements are not always possible and are always time consuming and therefore expensive. This work indicated that the information required for more advanced process management functions could potentially be acquired directly from the CNC controller. Making wider use of the information provided by the CNC controller was however not attempted in this previous research but was identified as being a major challenge.

Making sense of the CNC controller data means processing the data to extract the information it provides and presenting the results of the analysis to aid process management functions. It is made even more challenging when contemplating the manufacture of different components. It must be stated that the previous research on this VMC was confined to the manufacture of carefully selected test pieces that allowed the controlled application of the tool management techniques being developed. What is required to cope with the machining of a wider range of parts is the provision of some form of anticipated performance against which the actual

operation may be measured. This is seen as being met by the simulator being engineered by the author in this project.

To support the configuration and operation of the simulator an audit of the signals being acquired from the CNC controller was performed. It is important to again stress that these signals were identified in consultation with the VMC OEM (Mazak) as being of potential interest to on-going research, including this project, at the time of purchase. The OEM provided an interface to the information being utilised by the VMC controller to manage the milling process that was not normally currently available to the machine users. It can be envisaged however that future controllers will provide such access or equivalent information directly from the machine as part of the wider implementation of IIoT protocols.

The system currently in place on the VMC collected samples at an average rate of 30Hz (Hill et al. 2019). Each sample contains the following data:

- Time from the beginning of the process (sec)
- Spindle X axis coordinate (mm)
- Spindle Y axis coordinate (mm)
- Spindle Z axis coordinate (mm)
- Actual spindle speed (rpm)
- Load on X axis motor (% of max)
- Load on Y axis motor (% of max)
- Load on Z axis motor (% of max)
- Spindle load (% of max)
- Tool ID (position of the loaded tool in the carousel)
- Feed rate (mm/min)

To demonstrate the use of this information a basic milling process example can be developed. The simulation engine detailed in section 3.4.2 simulates the interaction between the cutting tool and the billet to produce the machined part as machining progresses. At this level the simulator is concerned with a tool and the billet upon which it is acting; Figure 28 shows the tool-billet system. Everything outside of this

part of the system, including the machine spindle, machine table and fixture interacts with this system through data interfaces discussed in the next section.

To initialise the simulator there are two types of information needed.

Simulation properties:

- Mesh size; calculated after analysing the available computer memory and assessing the computing power versus simulation speed requirements.
- Data sources; details for connection to the input data stream or path from which to read the input files.

Physical system properties:

- Cutting tool properties; tool dimensions and start position (spindle position coordinates).
- Billet properties; billet dimensions and mounted position.

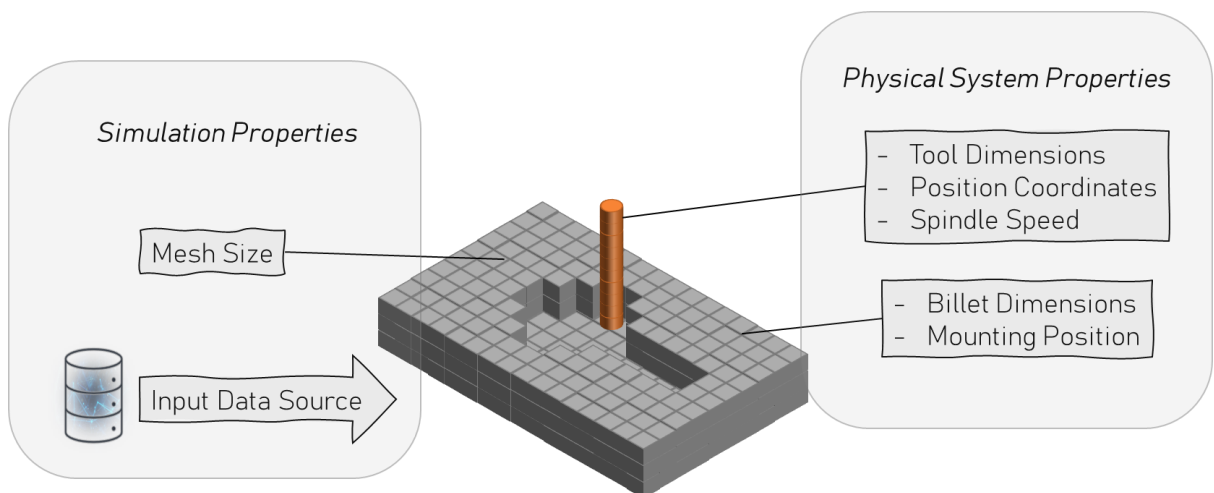


Figure 28 Simulation tool-billet system and initialisation parameters

As soon as the simulation begins, only information about the cutting tool (position, rotational speed etc.) and the billet is processed. External factors that may affect the process are translated to the impact they have on the tool and/or the billet. For example, if the milling machine table moves during the process, then this movement is translated to a billet movement and therefore the billet is repositioned. The relative tool-billet positions are adjusted as the tool moves up or down. In this way, the potentially complicated relationship between the cutter and

the workpiece can be represented within the simulator. Although this example is basic the same principle would apply to heat dissipation, vibrations or other parameters that require more complex calculations.

4.2 INPUT INTERFACE AND DATA SOURCES

Section 3.3 stated that the input interface should be adapted to the characteristics of the data source(s). This reduces the need for data manipulation outside the simulator which consequently reduces the risk of data losses or increased computational burden. For this application, the data source is the VMC which provides real-time monitoring data. For the purposes of this work, the monitoring data was transferred to a connected computer's hard drive in the form of CSV files (data dump). The G-Code files (part programs) were provided by the operators of the VMC in the same format that they are submitted to the VMC. G-Code is the most widely used CNC programming language and therefore for completeness the simulator has an embedded interpreter written by the author. Information about process setup (e.g. billet position, billet properties, cutting tools in carousel) was inserted manually. Manual insertion facilitated simulator verification and testing, but with minor modifications, the simulator could use a setup properties file instead. Each of these data sources is now considered in more detail.

4.2.1 CSV files

This is the main way to receive, and temporarily store process-related monitoring data or metadata based on a part program. These files as a minimum have the following columns:

Table 3 Minimum required CSV file columns (basic data sample)

Time	Time elapsed from a reference point in time (typically, process start)
X coordinate	The X coordinate of the spindle
Y coordinate	The Y coordinate of the spindle
Z coordinate	The Z coordinate of the spindle
Spindle speed	The rotational speed of the spindle

Tool ID	The carousel position of the cutting tool that is being used
---------	--

Each row of the CSV file is a data sample, and each sample is an instance of the process parameters. The simulator is a discrete event one therefore CSV files are ideal to record the simulated process along with the calculated parameters for each sample (discrete event). There is no specific order for the columns, but the name of each parameter is fixed. Table 4 is a non-exhaustive list of the simulator's parameters mapping

Table 4 CSV file title - parameter mapping

T	Time (sec)
X	X coordinate (mm)
Y	Y coordinate (mm)
Z	Z coordinate (mm)
SS	Spindle Speed (rpm)
XL	X axis load (%)
YL	Y axis load (%)
ZL	Z axis load (%)
SL	Spindle load (%)
T	Tool ID (-)
MRR	Material Removal Rate

Using CSV files adds a lot of flexibility in data storage as new parameters can be added to the same file without affecting the initial data. This is particularly useful when a new parameter is calculated from the sample captured by the monitoring system and added as a new column in the same file. An example CSV file is shown in Table 5. The MRR is calculated by the simulation engine using the coordinates of the tool and the value is added in the last column. (MRR requires initialisation of the billet size and position).

Table 5 Sample part from CSV file

t	X	Y	Z	SS	XL	YL	ZL	T	MRR
59.54659	1138	1361.5	1251	1027.985	7.105263	0	4.736842	28	0
59.54659	1138	1361.75	1251	1027.985	7.105263	0	4.736842	28	102
59.85125	1138	1362	1251	1027.988	4.95	0	4.65	28	238
59.85125	1138	1362.25	1251	1027.988	4.95	0	4.65	28	272
59.85125	1138	1362.5	1251	1027.988	4.95	0	4.65	28	340
59.85125	1138	1362.75	1251	1027.988	4.95	0	4.65	28	374
60.15595	1138	1363	1251	1027.997	4.684211	0	3.631579	28	0
60.15595	1138	1363.25	1251	1027.997	4.684211	0	3.631579	28	408
60.15595	1138	1363.5	1251	1027.997	4.684211	0	3.631579	28	408
60.15595	1138	1363.75	1251	1027.997	4.684211	0	3.631579	28	476
60.45279	1138	1364	1251	1028.043	5.684211	0	4.263158	28	442

The software interface that has been developed by the author to read and write data in CSV files is based on two Java arrays. A Java array is a group of values of the same type that can be individually referenced using their unique index. The first array is a one-dimensional String array (containing UTF-8 characters) that holds all CSV titles. The second is a two-dimensional 'double' array (containing double precision real numbers) that holds all file values. In general, the recommended way to parse CSV files is to use ready solutions such as Apache Commons (Apache Commons, 2021). This ensures compatibility with different CSV dialects and provides mechanisms to overcome errors in the dataset. However, for the purposes of the research project, a customised parser was developed to reduce the complexity of calculations and speed up the processing speed of the input interface.

4.2.2 Part program and process setup

In the physical world, the operator prepares the milling machine before the actual machining begins. Similarly, in the digital world, the simulator needs initialisation before it can simulate the machining process. More specifically, the simulation engine requires the billet position and properties, the list of available cutting tools, the size of the simulation mesh (precision) and finally the process data samples with each one representing the state of the cutting tool-billet system (Figure 28). Running the engine can be considered in respect of each element:

Billet: A library of billets is created by the user in advance. This forms a working source of billets that may be used as is the case in actual machining. New billets

can be added as necessary. During simulator setup, the user must select the billet to be machined so data related to billet shape, size and position is retrieved in order to place the billet on the virtual milling table with the correct dimensions and orientation. This procedure is very similar to the billet selection process used in actual machining applications.

Cutting tools: The user creates a library of cutting tools that are available for the physical machine in advance. The actual VMC has a carousel where the operator loads the cutting tools into an identified tool pocket and then the machine can pick and use the required cutting tool according to the instructions of the part program. Similarly, during the simulator's setup, the virtual carousel is loaded exactly as the physical one with the same cutting tool in the same position. Then the simulation engine loops through the tool IDs that are loaded into the carousel and retrieves data related to tool shape and size. For a typical milling cutter this data will include tool type, diameter and length.

Simulation parameters: Simulation precision, speed and memory size are configurable before the simulation engine runs. Simulation precision is determined by the mesh element size. Finer meshes for the billet and the cutting tools improve calculation precision but as explained in 3.4.2.2 this also depends on the precision of the data describing the milling process. Simulation speed can be increased by assigning more CPU cores to the simulator. In the current version, the simulator uses all available cores and therefore the speed can only change if the simulator runs on different hardware. The RAM required to run the program depends on the maximum available memory and mesh size. The Java virtual machine 'xmx' parameter sets the maximum available memory. Although the computer's operating system may use the hard drive when it runs out of RAM the simulator is programmed to stop and display an error and therefore prevent slow processing that is caused when temporary data is written on the hard drive. The mesh element size is the parameter that defines the amount of memory the simulator requires to process the mesh. A smaller element size will lead to a mesh with a higher number of elements. Each element requires a fixed amount of memory, so the number of elements is proportional to the required memory size. Finally, processing speed

depends on the element size and the process visualisation settings. If the simulator is run without process visualisation only element size affects processing speed, with obviously smaller elements resulting in slower processing. If process statistics are displayed, additional time is required to report the process and cutting tool-related data. Again, this means the more cutting tools the slower the process. If 3D part visualisation is also requested, then the process is significantly slower and slows down exponentially if the element size is reduced.

Simulation settings persistence: Most of the simulation settings (except for maximum memory size) are defined by the user through the GUI before the process starts. To store these settings the simulator setup is supported by the database. It should be noted that section 3.6 presented the role and functionality of the database from an architecture point of view. This does not exclude the usage of the same database to support the functional requirements of a specific implementation. In this case, critical parameters are stored in the database and are permanent for all simulator runs. Simulation settings include the mesh data for the cutting tools that are used by the simulator throughout the process which is described in section 4.4. All elements of a cutting tool are stored to keep a record of cutting tool usage. The number of elements depends on the element size and therefore the tool is restricted to be reused in simulations that run with the same element size. Essentially, the element size is a permanent characteristic, and it cannot be changed throughout the life of the simulator's database. On the other hand, due to database portability, the simulator may have multiple databases and its behaviour can change according to the setup and learning parameters contained in the database. More details about the data kept in the database of the simulator are presented in section 4.3.

Program data: The data source for toolpath and tool state, along with data type and characteristics are required in advance. The source can be a G-Code or CSV file for offline simulations, a CSV file for simulator learning or a data stream link for online process simulation and supervision. The input interface produced to support this research accepts standards-compliant G-Code; arcs, circles, and helical motion are fully supported, as well as all other primary G-Code commands. Macro functions,

variables, and most canned cycles are not supported (GitHub - GRBL. 2016). Process data samples in offline running come in the form of CSV files. The CSV file for process simulation must have the minimum number of required columns shown in Table 3 that conform to the mapping of Table 4. The CSV file for learning must have the minimum number of required columns plus a column for each parameter that the simulator is taught to estimate. Finally, in real-time simulation scenarios, the input data is a stream. The source of the stream is a simple link that the simulator can open a WebSocket to and get live data from the machine monitoring system in JSON format. The JSON property names should comply with the parameter mapping of Table 4.

4.2.3 Source data pre-processing

After initialisation with manual user settings and with information already in the database the simulator is ready to process the raw data (source data). As described in 3.3, raw data is pre-processed to a standard form that the simulation engine can use. Depending on the characteristics of the data source an appropriate order of steps from Figure 10 is selected which in the milling application is different for G-Code files, CSV files and live data streams.

More specifically, if the input data source is a G-Code file, the input interface first interprets it to a CSV file and then follows a series of steps until it stores its data in Java arrays. If a CSV file is provided, the process is similar to processing the CSV file derived from the G-Code. These are both offline simulator use cases and after all data is stored in arrays, the simulation engine iterates over the arrays to machine the billet mesh. If the cutting tool state samples are received from a live stream, then the simulation engine pre-processes and loads every sample to the next free position in the array, runs an iteration and waits until the next sample is available. Regardless of the way that samples are received the rest of the virtual machining process remains the same.

4.2.3.1 G-Code file pre-processing

In the physical world, CNC machines need an interpreter to turn G-Code file commands into internal controlling commands. Similarly, the Input Interface reads the commands and provides the simulation engine with prefilled Java arrays that it

requires to run. This uses a 5-step process designed and implemented by the author as part of this research. The method, which is detailed herein, uses temporary CSV files to store the interpreted data.

Step 1 is to receive the G-Code file itself. The G-Code files are provided by the VMC operators and stored on a hard drive that the simulator has access to. They are generally generated using the part CAD models and machine-specific post-processors. From the GUI that has been developed by the author to manage and run the simulator the data source is selected (Figure 29).

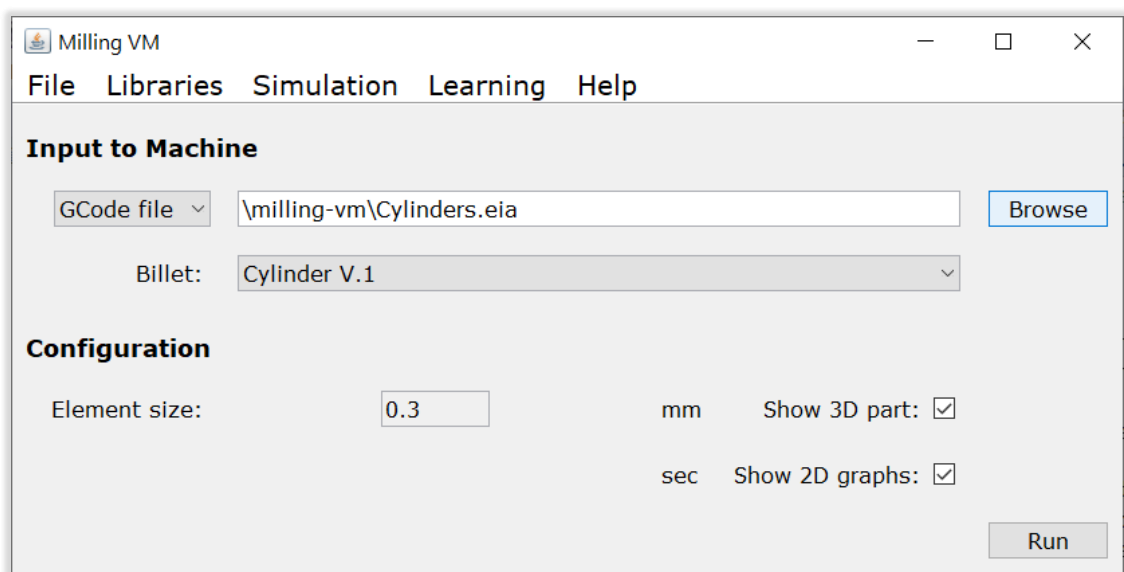


Figure 29 Simulator's GUI: Data source selection

Step 2 is to transform the G-Code file contents to a format that can be further processed which in this case is a CSV file format. To convert G-Code commands to CSV file rows, GRBL (GitHub - GRBL. 2016), an open-source external library, for controlling the motion of CNC machines has been modified and integrated into the input interface. The original library takes a G-Code command and produces a signal for the X, Y, Z axis motors of a CNC machine. The modified library analyses the G-Code command and calculates the coordinates that the cutting tool must pass from, the spindle speed and the feed rate. In the modified version, motor signals are not generated since the simulator does not control the VMC. Having the trajectory points and the feed rate, the time since the program started is calculated for each point. Finally, the results of G-Code interpretation are written in a temporary CSV

file which contains all essential simulation data as shown in Table 3. Technically, the data written in the CSV file is held in Java arrays in the memory of the computer. These arrays could be directly used by the simulation engine although the results would not be reliable due to the reasons explained in the next paragraphs. The temporary CSV file is created for verification and testing purposes and it would normally be omitted in a non- experimental version of the simulator.

The way that the interpretation is done, means each G-Code command generates at least 2 rows in the CSV file, one for the starting point one for the ending point and other points of the trajectory that the cutting tool must follow. If two consecutive commands have the same spindle speed and feed rate, then the ending point of the first command generates an identical entry in the CSV file with the starting point of the second command. To prevent the simulation engine from processing null events, Step 3 'cleans' the generated CSV file. There are two types of checks to do that. The duplicates check examines if two rows of the CSV file are identical and keeps the first line if duplicates are found. The data validation test checks if each row has the correct number of columns and if all values are numbers. If an error is found, then the problematic row is removed. The CSV file is then marked as cleaned.

Step 4 begins with a clean CSV file that may contain trajectory points which are far from each other compared to the cutting tool size. For example, if the cutting tool moves along the x axis and the G-Code command is G1 (linear movement), then the GRBL library will produce 2 points, the starting and the ending. With this type of input and since the simulation is discrete event, the simulation engine does not remove the material between the two points because it does not 'know' what has happened between the event of starting point and the event of ending point. Figure 30 left shows the result of a G1 command interpreted by the GRBL library.

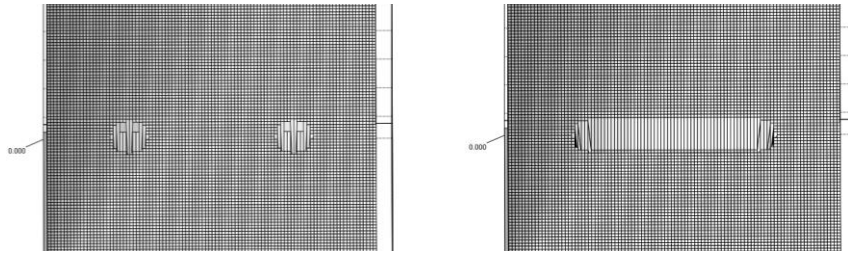


Figure 30 Left: Non smoothed trajectory. Right: Smooth trajectory (with interpolated points)

To smooth the movement of the virtual tool, trilinear interpolation is used. If any axial distance between 2 trajectory points is longer than the element size, then new trajectory points are introduced. To calculate the position of these points, first, the longest axial distance between the points is identified. Then, this distance is divided by the element size and finally, the result of the division is the number of new points introduced to smooth the trajectory. The new points have an axial distance smaller or equal to the element size. The element size value is used because the mesh of the billet consists of cubes with an edge size equal to the element size. With a different element shape the calculations would be different and typically more complex. It should be noted that if more points are introduced (for example to improve cutting tool movement precision) then the cutting tool movement will be too short to reach and machine the next cube.

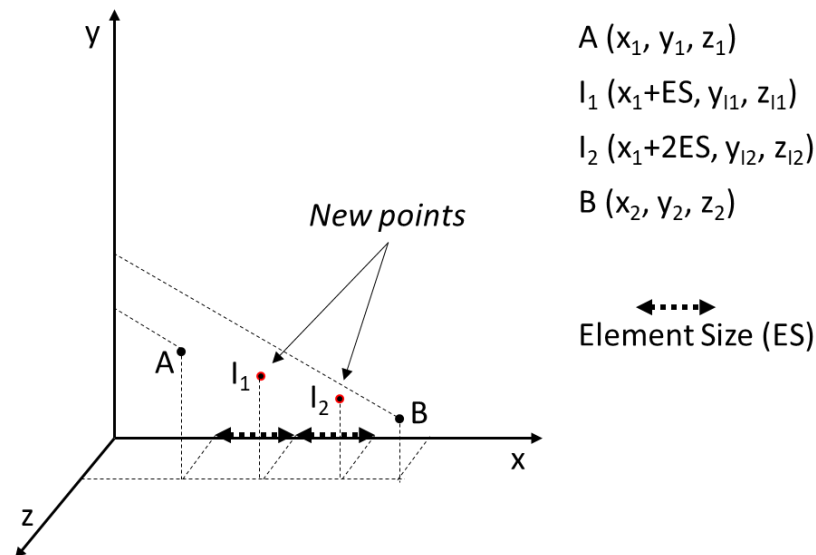


Figure 31 Cutting tool trajectory smoothing

Finally, in step 5, the smoothed CSV file is parsed, and pre-processed data is loaded into the Java arrays that the simulation engine gets input from. Every

column of the CSV file is copied to a separate array keeping the order of the values. For example, the values of a column with the title Z (in the CSV file) are copied to an array with the same name (Z). The row number of each value is used for indexing so the value of the column at the first row of the CSV file is placed at the first position of the array. Figure 32 is an example of how indexing is done.

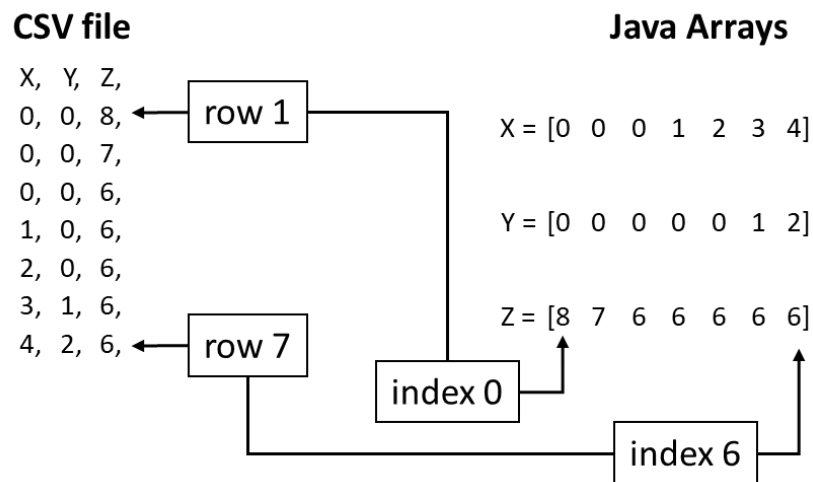


Figure 32 Transferring data from CSV files to Java arrays.

The steps used in the procedures deployed by the author to process a G-Code file are summarised in Figure 33. Under every step, the category of operation as indicated in Figure 10 is shown.

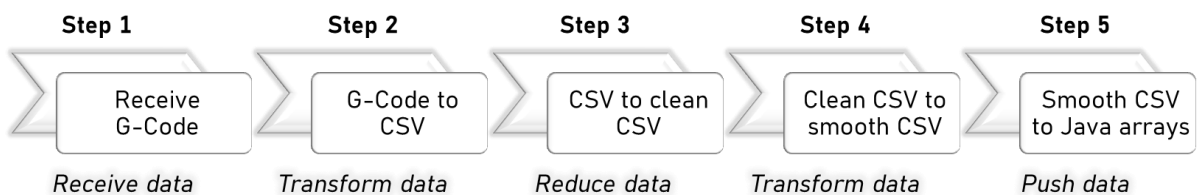


Figure 33 Processing of G-Code file

4.2.3.2 CSV file pre-processing

If the input is a CSV file, then pre-processing is similar to Figure 33 because the files are found on a hard drive like the G-Code files. Step 1 is the same, and from a user's perspective it is done through the GUI screen of Figure 29. For completeness, it should be noted that the CSV files are created by the system of (Hill et al. 2019) that continuously extracts data from the VMC. It will normally be the case that

these files will have been generated as a result of the controller enacting a programme that was initially input using a G-Code. As such they take a very similar overall form to those generated above but may have some content that has been produced by the process. The Java arrays that are passed to the simulation engine are created in step 2 while reading the data source file through a process shown in Figure 34. In this process the file contents that are initially recognised as text are split in titles and values. The titles of the CSV file are identified and stored in a Java array. Then all following lines are expected to contain numbers which leads to the first cleaning operation where all lines that contain non-numerical values are removed. The ones that do contain valid numerical values are stored in the Java array holding the values.

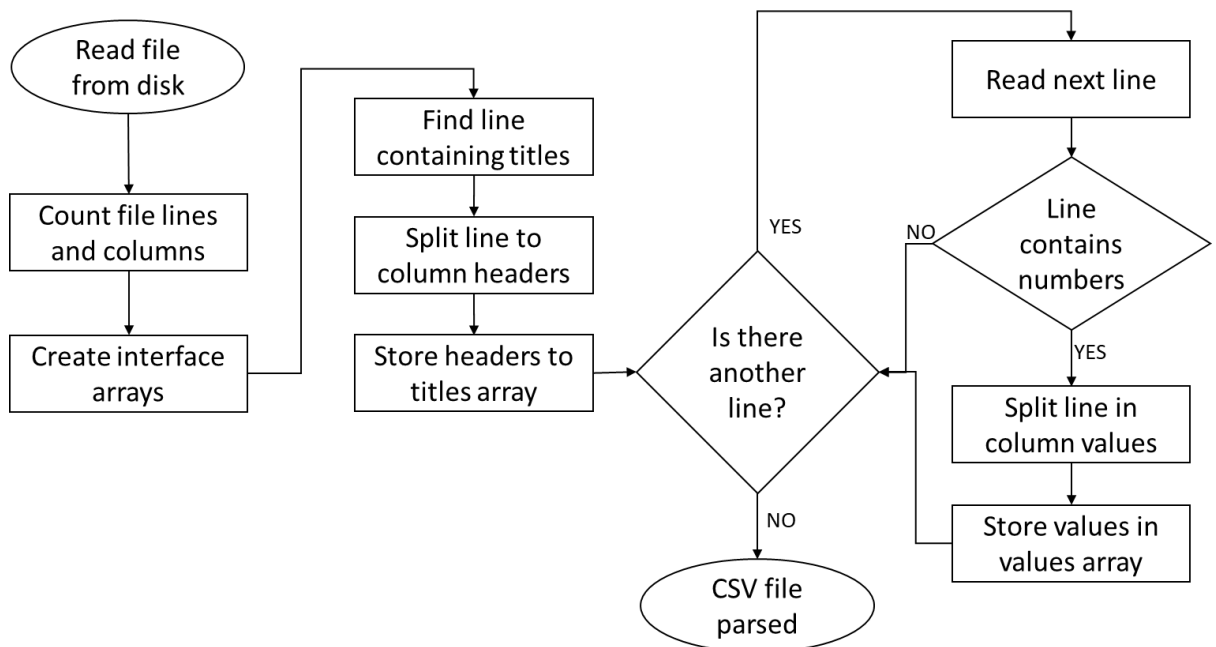


Figure 34 CSV file reading and first cleaning pass.

In the case of G-Code source data file, cleaning is speeding up the process by removing duplicate samples and by removing null values that would otherwise be converted to 0. If however cleaning is skipped the simulation engine will process the data without issues. When the source data is a CSV file then cleaning plays a critical role which is seen in all systems that process data generated by sensors. Monitoring data files are generated by an external system (Hill et al. 2019) so there may be errors in content or format. Some examples of errors are:

- 'N/A', 'empty', or 'null' values.
- A stack trace at the end of the file when the monitoring system runs into an error.
- Meta data related text appearing above the CSV file titles.

CSV cleaning removes the problematic lines and ensures that the remaining data can be processed safely by the simulation engine. During system testing, it was observed that almost all files had some type of error. The simulation engine sends a notification for each error that appears in the user interface, so the user can decide if the file contains too many errors to be considered a trustworthy data source.

Steps 3 and 4 are identical to steps 4 and 5 of the G-Code data source file scenario. The CSV data source file processing is summarised in Figure 35.

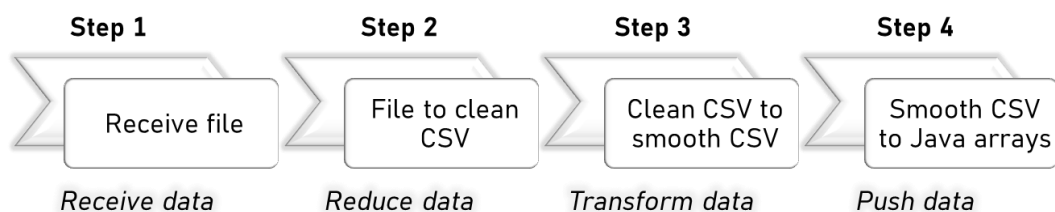


Figure 35 Processing of CSV data source file

Steps 2-4 are further analysed in 4.2.3.3 and depicted in Figure 36.

4.2.3.3 Live data pre-processing

If data arrives directly from the monitoring system, the processing is based on steps 2-4 of Figure 35 but with a different step 1. The current system monitoring the VMC is built to save data to CSV files. The system would require modifications that interfere with other research works and it was therefore decided to run the online case study with a virtual monitoring system. This is implemented by reading the contents of the CSV files (produced by the actual monitoring system) and supplying them to the input interface at the correct timing. The original samples have a timestamp so the virtual monitoring system supplies them at the exact same intervals. Each sample contains the data of one line of the CSV file. As soon as the sample is received the input interface proceeds to step 2. The remaining steps are technically similar to the offline scenario but are interpreted differently.

In offline processing, all data is processed before moving to the next step while in online processing monitoring data is not available in advance. When the simulator is running in online (supervisor) mode, the simulation engine runs in parallel with the actual machine and therefore, processes each sample as soon as it becomes available. From a programming perspective, the same Java classes are involved in each step, but the orchestration of actions is different. To demonstrate the online pre-processing and illustrate the differences between offline and online Figures Figure 36 and Figure 37 show in detail the steps taken to clean, smooth and push data to the simulation engine. If both figures are closely observed, it becomes clear that the steps are the same. The reason why the flow charts have been designed differently is to indicate that in the offline case each step runs once and finishes before the input interface proceeds to the next step. In the online case, the input interface moves to the next step for each sample or group of samples but as soon as the samples are pushed to the engine the cycle runs from the beginning. The actual process is identical. When running the simulator offline with a dataset of 100k samples, steps 2-4 are similar to running an online scenario for a group of 100k monitoring samples.

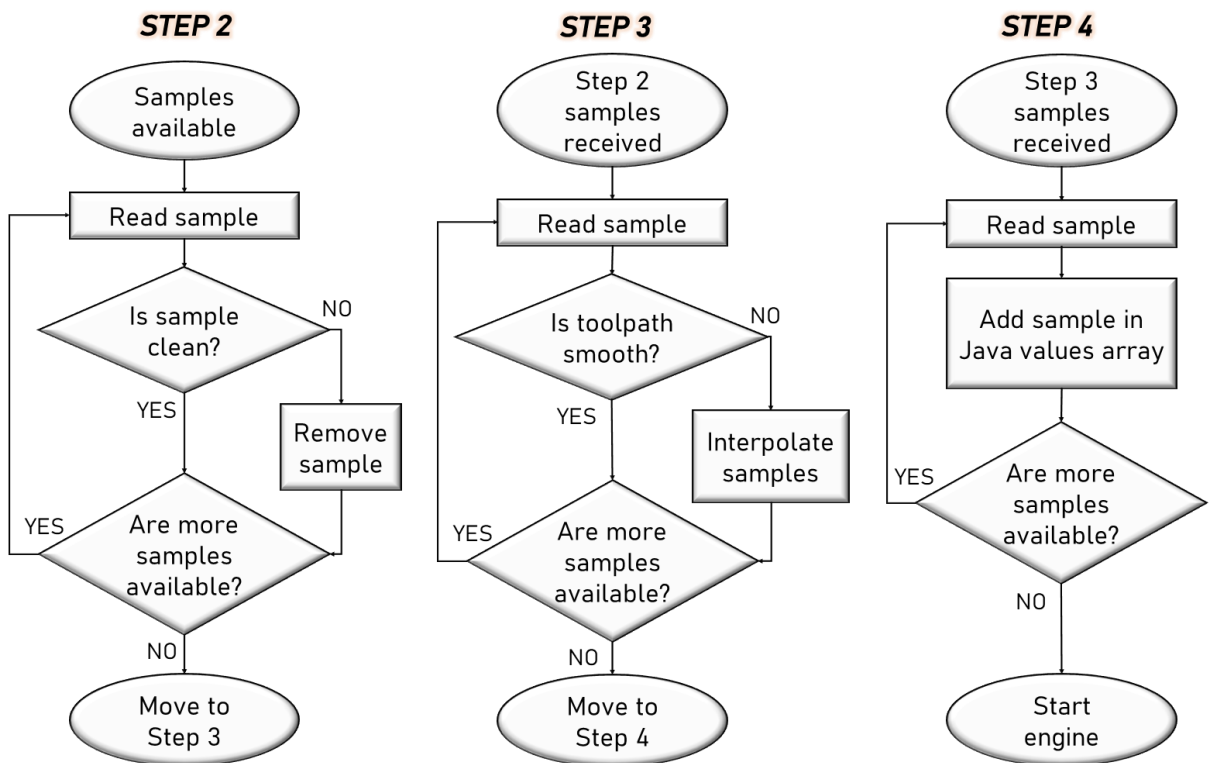


Figure 36 Offline CSV interpretation steps

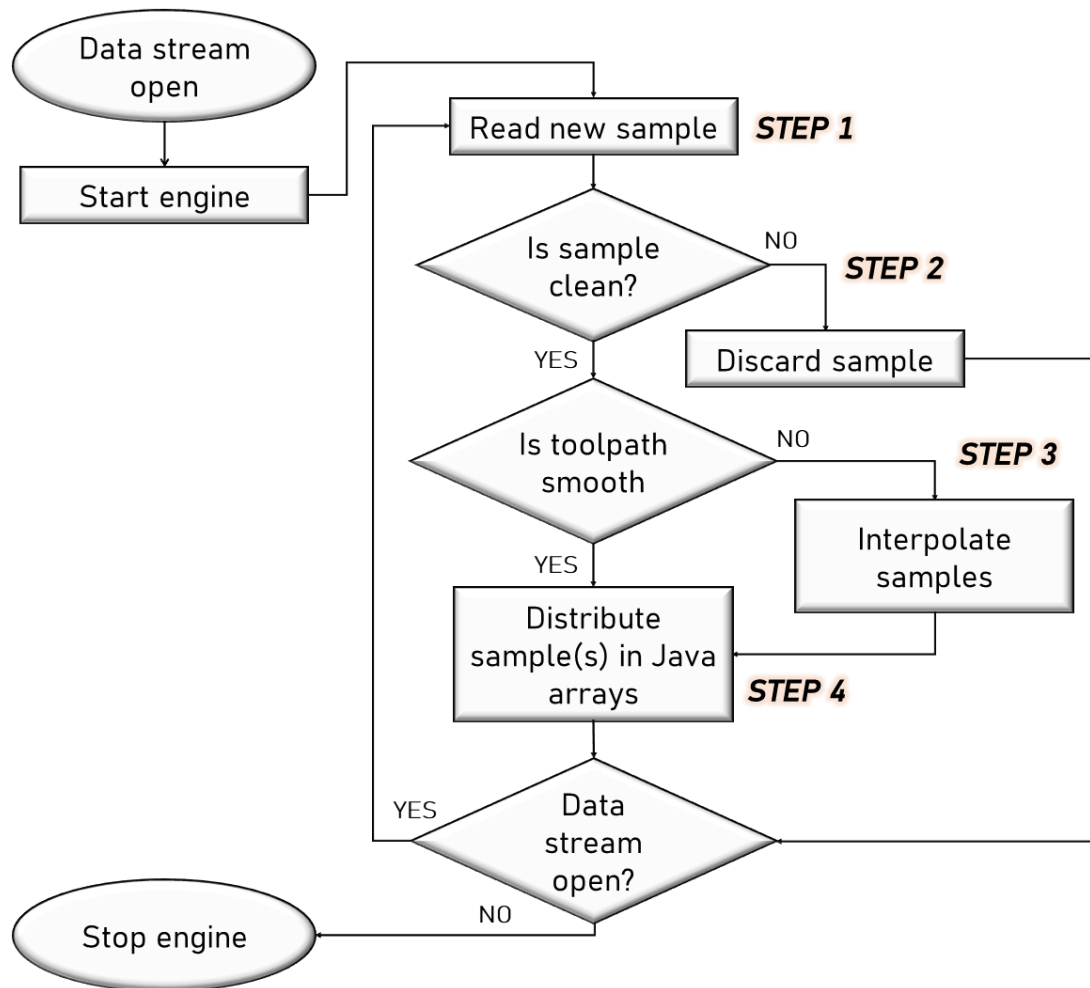


Figure 37 Online data interpretation steps

With the online pre-processing, the full capabilities of the input interface implementation have been described. The implementation however is flexible and with minor modifications in the data input (Step 1) the simulator would be able to accept data from other data sources.

4.3 SIMULATOR DATABASE

In section 3.6 the function of a simulator's database was described along with the technologies that can support such functionality. The VMC implementation of the database has all the described elements and in addition, it accommodates the functional needs of the simulator by keeping process-related settings, metadata and information that is required to produce enhanced reports. The database system developed by the author is a combination of SQLite which is a RDBMS and plain

files. A significant part of the simulator is dedicated to interfacing the simulator elements with the database which is also described in this section.

SQLite is a C-language library that implements a small, fast, self-contained, high-reliability, full-featured, SQL database engine. SQLite database files are commonly used as containers to transfer rich content between systems and as a long-term archival format for data (SQLite Home Page. 2022). This is the library used by the author to implement the simulator's database as described in Chapter 3.

Interactions with SQLite are enabled by the corresponding JDBC driver available at the SQLite developers' website. Structured Query Language (SQL) is used to access and manipulate the database.

The database is used internally and there are no third-party systems currently interacting with it. For debugging purposes, the database can be accessed with external tools such as SQLite Browser (DB Browser for SQLite. 2021) that allow for easy monitoring and manipulation of database fields. The simulator cannot operate without being connected to a database and therefore a database must be created or selected as soon as the simulator graphical user interface is started. SQLite is a relational database with a predefined schema and therefore the information is kept in tables. Figure 38 shows the full database schema which is the basis of the approach to representing the critical parameters associated with the milling of a piece of material engineered by the author. The database keeps simulator initialisation data, libraries for billets and cutting tools, the simulator's learning memory and a log for the GCode files representing parts that have already been processed. To keep the database small part of non-essential data is stored in flat files. This corresponds to the following database fields:

- learning_set > modelFilePath
- nc > ncPath
- nc > analysisPath
- nc > monitoringPath

Each field along with the Java classes developed to hold data for each entity is presented later in this section.

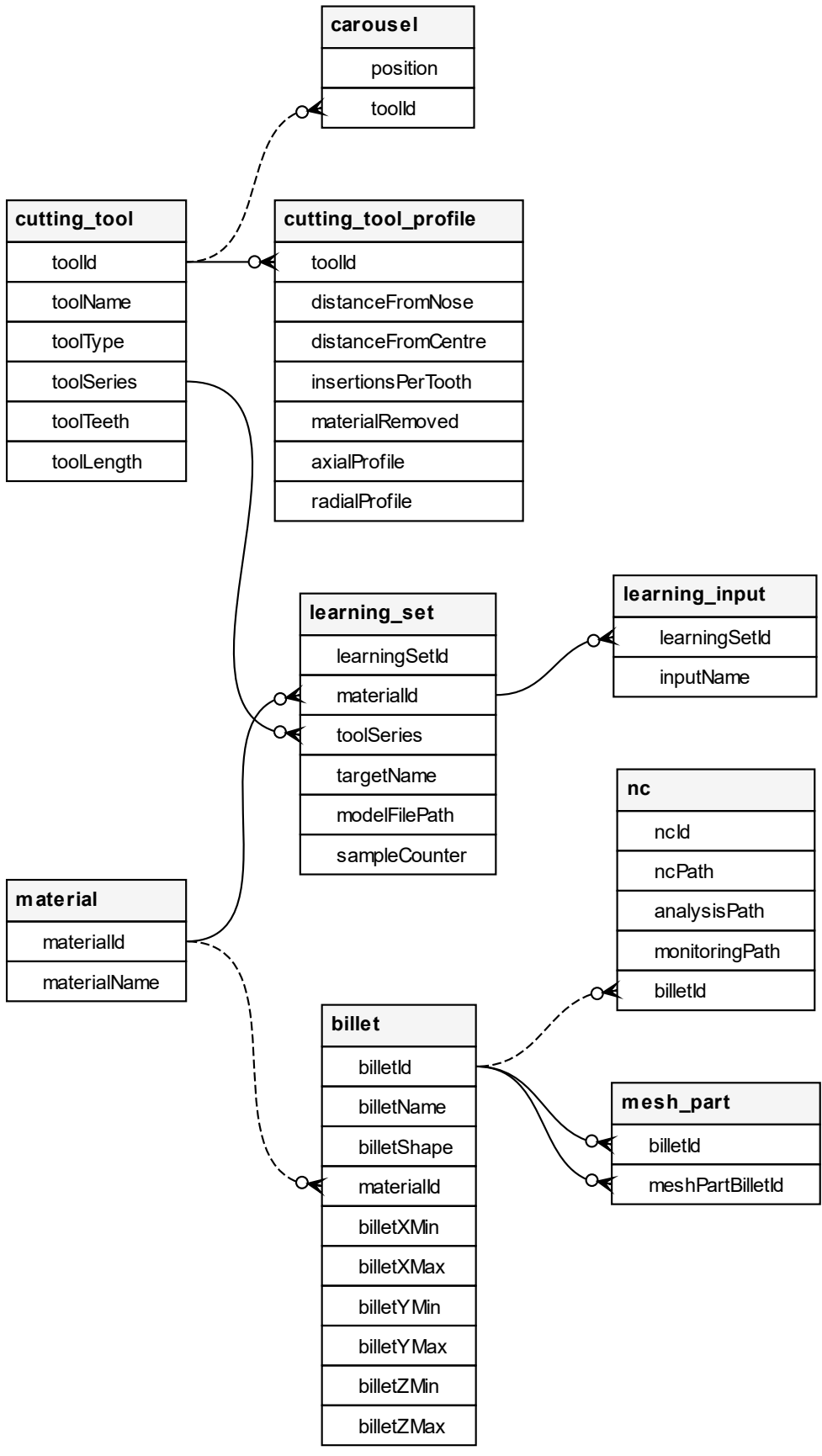


Figure 38 Simulator's database schema.

To interface the simulator with the database the persistence layer of the simulator has been implemented. This layer is based on Java classes whose attributes are very similar to the relevant table columns. Each class acts as a data transfer agent. For every 'write' type of transaction, an instance of the class is created, then loaded with data and then, using SQL, the data is passed to the database driver that is responsible for writing the data into the database. For 'read' type transactions, the driver is provided with the unique identifiers of the table rows to retrieve and the results are loaded into the corresponding class instances. Then, these instances are returned to whichever module requested the data. The flow diagrams for the read-write processes are shown in Figure 39.

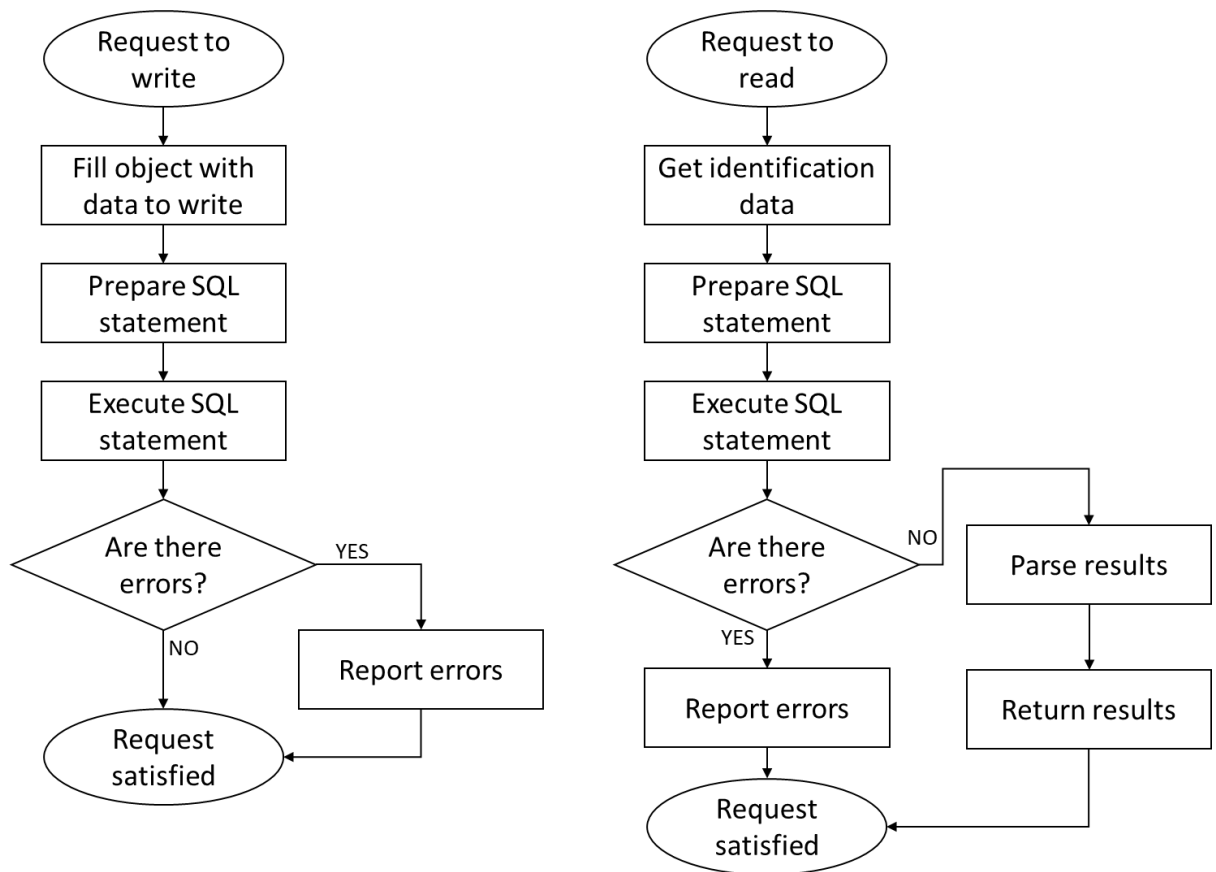


Figure 39 Interface operation. Left: Write to database. Right: Read from database

The Java classes that the author has programmed are presented through their Javadocs in Appendix B. However, some key classes are presented in tables Table 7 to Table 12. To facilitate the understanding of datatypes for each class Table 6 has a brief key of frequently used Java types.

Table 6 Java class attribute types

Type	Description	Range of values
int	Integer number	-2,147,483,648 ... 2,147,483,647
float	Floating point number	1.4×10^{-45} ... $3.40282347 \times 10^{38}$
long	Integer number	-9223372036854775808 ... 9223372036854775807
double	Floating point number	$4.9406564584124654 \times 10^{-324}$... $1.7976931348623157 \times 10^{308}$
boolean	Representing two truth values of computer logic	True, False
String	A series of characters (any character)	Maximum of 2,147,483,647 characters in each String
List	An ordered collection of (non-primitive) objects of any type.	Maximum of 2,147,483,647 objects in each List
Map	A non-ordered collection of key-value combinations (the value is retrieved after providing the corresponding key)	Maximum of 2,147,483,647 objects in each Map

In the following paragraphs, a description of each key class is provided, followed by a table that presents the class attributes. The first column of the table contains the type of the attribute, the second column is the name of the attribute and the third is the description of the data that this attribute holds. It should be stated that the nature of the attributes and data used in these classes represents the understanding of the milling process acquired by the author from personal experience in milling, from the reviewed literature and the experimental work conducted in this project.

CuttingTool: The CuttingTool class holds identification information and a description of the characteristics of a cutting tool, shown in Table 7. This class is not used directly for recording the tool usage parameters, but it holds a reference to a list that contains all relevant data.

Table 7 CuttingTool class attributes

int	toolId	ID of this tool in the database
String	toolName	Description of cutting tool
String	toolType	Type of tool (endmill, ball nose, etc)
String	toolSeries	Series of tool (manufacturer code)
int	toolTeeth	Number of teeth
double	toolLength	The total length of the tool
List<CuttingToolProfile>	cuttingToolProfiles	Details for tool mesh

CuttingToolProfile: Before simulating the vertical milling process, a cutting tool mesh is generated (more details in paragraph 4.4.2). This class holds information related to the usage of an element of the cutting tool. Typically, all elements that form a cutting tool are grouped under a Java List and their reference is stored in the CuttingTool instance of the cutting tool that is being used. The attributes of CuttingToolProfile are listed in Table 8.

Table 8 CuttingToolProfile class attributes

int	toolId	ID of the tool that this element belongs to
double	distanceFromNose	Z-axis distance between the element and the tool's bottom face
double	distanceFromCentre	Distance between element and tool rotating axis.
int	insertionsPerTooth	Times the element tooth removed billet material
double	materialRemoved	Billet elements removed by this tool element
boolean	axialProfile	True if this element is on the side of the tool
boolean	radialProfile	True if this element is on the bottom face of the tool

Closely related to the cutting tool classes is the carousel table in the database. However, because of the simplicity of the table (each row holds 2 ids) there is no need for a separate custom class and information is passed as 2 integers.

Billet: This class holds information related to billet positioning and properties, shown in Table 9. In the current version of the simulator, the billet is constructed from scratch every time a simulation runs based on the data stored in the database. To simulate pre-machined billets, there is an option for complex billet shapes (a merge of simple-shaped billets).

Table 9 Billet class attributes

int	billetId	Unique ID of the billet
String	billetName	Description of the billet
int	billetShape	Type of shape indicating the way to generate billet mesh
int	materialId	ID of the material that the billet is made of
double	xBilletMin	Minimum X coordinate of the billet positioned on milling table
double	xBilletMax	Maximum X coordinate of the billet positioned on milling table
double	yBilletMin	Minimum Y coordinate of the billet positioned on milling table
double	yBilletMax	Maximum Y coordinate of the billet positioned on milling table
double	zBilletMin	Minimum Z coordinate of the billet positioned on milling table
double	zBilletMax	Maximum Z coordinate of the billet positioned on milling table
boolean[][][]	part	A 3D array holding state of each billet element (true=machined)

Complex shape billets do not require a separate class as each row of the database table contains the parent billet ID (the complex one) and the ID of one of the children. To simulate a parent billet, its children are generated and positioned accordingly on the milling table. (More details in paragraph 4.4.3)

Material: Billet material information is held in a separate class as shown in Table 10. It is assumed that dimensions together with material type sufficiently describe a billet and that two billets with the same dimensions, positioning and material will have the same behaviour.

Table 10 Material class attributes

int	materialId	Unique ID of the material
String	materialName	Description of the material

NC: The simulator keeps a log of the analysis it does on G-Code files. It therefore keeps the original location of the G-Code file, the location of the analysis CSV file (after an offline run of the process) and the location for monitoring data produced by the G-Code file, shown in Table 11. Although in theory the simulator will gather data in real time, for practical reasons (especially for validation) the simulator is able to keep a log of available process data that can either be processed offline or retrieved if a direct comparison between two runs is required. NC class attributes are:

Table 11 NC class attributes

int	ncId	A unique ID for this G-Code – analysis file connection
String	ncPath	Location of G-Code file
String	analysisPath	Location of analysis file
String	monitoringPath	Location of monitoring data file
int	billetId	ID of the billet used for the process

LearningSet: When the Learning Module trains a ML model with monitoring data, the model parameters along with model identification information are saved in the database. The LearningSet class holds this information and an instance of this class is created to retrieve the model from the database when a combination of billet material and cutting tool type has already a calculation model trained. The attributes of the LearningSet class are shown in Table 12.

Table 12 LearningSet attributes

int	learningSetId	ID of this set of learning parameters
int	materialId	ID of billet's material
String	toolSeries	Manufacturer's series code of cutting tool.
String	targetName	Target parameter of the learning model
String[]	Inputs	Input parameters for learning model
long	sampleCounter	Number of samples used to train the model
String	modelFilePath	Location of the file containing the learning model

4.4 MILLING SIMULATION ENGINE

Following the architecture presented in chapter 3, at the core of the simulator lies its simulation engine. The simulation engine is following 4 steps to calculate the expected results of running a program on the physical milling machine. These steps are:

1. Get simulation initialisation data.
2. Generate the mesh for cutting tools and billet.

3. Run the milling program virtually and calculate the simulation results.
4. Push the results to other simulator modules and/or store them.

The simulation engine is designed to be used both offline, to run GCode files, CSV files or assist the simulator's learning, and online to support the physical machine's supervision. The simulation engine requires initialisation data and cutting tool related data (position and state) to run. Initialisation data is provided by the user and stored in the database. This mirrors the manner in which a VMC will be operated, whereby the individual tool information is input before any manufacturing is attempted. This data is stored in a tool library within the VMC controller. This is then referenced by individual programs to provide tool offsets and related information required to enable accurate machining. During the simulation tool position and state are provided by either the GCode program or the monitoring system (CSV file). For offline running, all data is available before the engine starts. For online running, initialisation is provided in advance and tool related data is provided in the form of a live stream of samples recorded by the monitoring system.

Table 13 Simulation engine data requirements

Task	Initialisation	G-Code	Monitoring
G-Code execution	Yes	Yes	No
Learning	Yes	No	Yes
Process supervision	Yes	No	Yes

After the initialisation data is provided, the engine generates the mesh for the billet and the cutting tools that are involved in the process. All meshes are generated in advance, as detailed later in this section, before the engine begins to process cutting tool position data. The machine verifies that the billet mesh, the cutting tool meshes, the carousel setup and the cutting tool source of data are setup successfully (not null) before flagging process initialisation as successful.

The machining process is simulated by processing sample by sample the cutting tool state data and calculating for each step the simulation model output

parameters. The engine is calculating all directly related parameters every time a new sample is available but grouping, averaging or other basic statistical processes used mainly to inform the user are done by the output interface. The output of the simulation could be described as the output of a virtual monitoring system that produces raw data which needs to be post-processed to become suitable for the results reporting. After the simulation engine processes all available samples, the final results are pushed to the output interface for reporting. 1-sample-based generated data is saved in CSV files and all other relevant data (logs, statistics, etc.) are saved in the simulator's database (see paragraph 4.2 for the type of data being stored).

In the next sections, the simulation engine operation is described in detail to provide a better understanding of the functionality and the way that results are generated.

4.4.1 Simulation engine input data

Section 4.2 has presented the steps through which raw data is prepared for the simulation engine. The simulation engine has to be started manually by the user but after that point it runs continuously until all data is processed (offline) or until the input data stream is closed (online). In the introduction to this section all initialisation activities have been described and it would be redundant to repeat these details. However, it is important to repeat that it is not the responsibility of the simulation engine to do any data preparation. The engine expects that everything it needs to run is available either in the Java arrays that are loaded by the input interface or in the database. The simulation Engine will simply access these sources and if data is not found or is not in the expected form it will stop running and report an error with details on why the error occurred. Therefore, apart from the manual preparation of initialisation data and the operation of the input interface technically speaking there is no input data task assigned to the simulation engine. The actual operation of the engine starts directly with building the meshes of the cutting tool and the billet to be machined.

4.4.2 Cutting tool mesh

Simulation engine initialisation data includes the mesh element size value which as explained in 3.4.2 it is a critical parameter for the virtual representation of the cutting tool and the billet. Element size is defined during the creation of the database and remains constant throughout the life of the database. The cutting tool mesh is formed by rings whose width is equal to the element size. The dimensions of the cutting tool model (virtual) are the same as the actual cutting tool (physical). Since the cutting tool is revolving at high speed, the billet material that falls within the radius of the tool is removed. Therefore, although the tool has teeth the resulting cut takes the form of a cylinder.

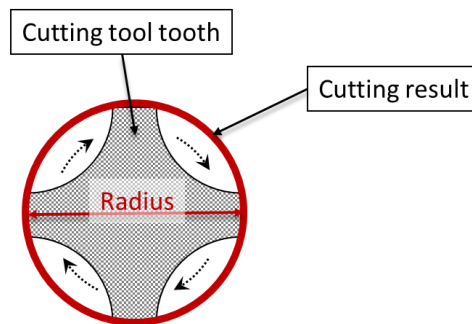


Figure 40 Resulting shape of removed material area

Based on the shape of the removed volume, the tool model is a cylinder with a radius equal to the cutting tool's radius and a height equal to the cutting tool's working height. These are the equivalent of the tool offset data stored for each cutter in the VMC tool library. The part of the tool that is used to mount the cutting tool in the spindle is not being considered. It is also assumed that only the surface of this cylinder can remove material therefore the resulting model resembles a hollow cylinder.

To generate the mesh, the cylinder is sliced into two types of rings. Concentric rings which form the mesh of the cutting tool's bottom face and stacked rings which form the sides of the tool.

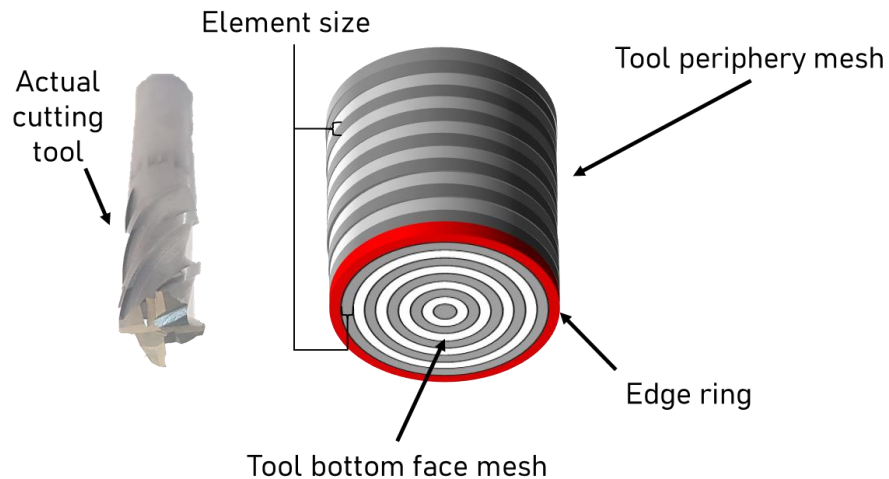


Figure 41 Endmill cutting tool mesh

The centre of all rings lies on the cutting tool's axis of rotation. Tool periphery (side) rings are stacked without gaps in order to form a continuous surface when viewed from the side (Figure 42). The thickness of each ring is equal to the element size. The external diameter of each ring may vary depending on the shape of the cutting tool. Bottom face rings have the same centre. The external minus internal diameter equals to the element size. Every ring (except the smallest and largest) has a smaller ring osculating internally and a larger ring osculating externally, creating that way a continuous surface when viewing the cutting tool from the bottom (Figure 42). The smallest ring has zero internal diameter which makes it a disk.

If a ring is visible from both side and bottom views (like the edge ring in Figure 41) it complies with restrictions for both bottom face and side rings and machines in both directions during simulation.

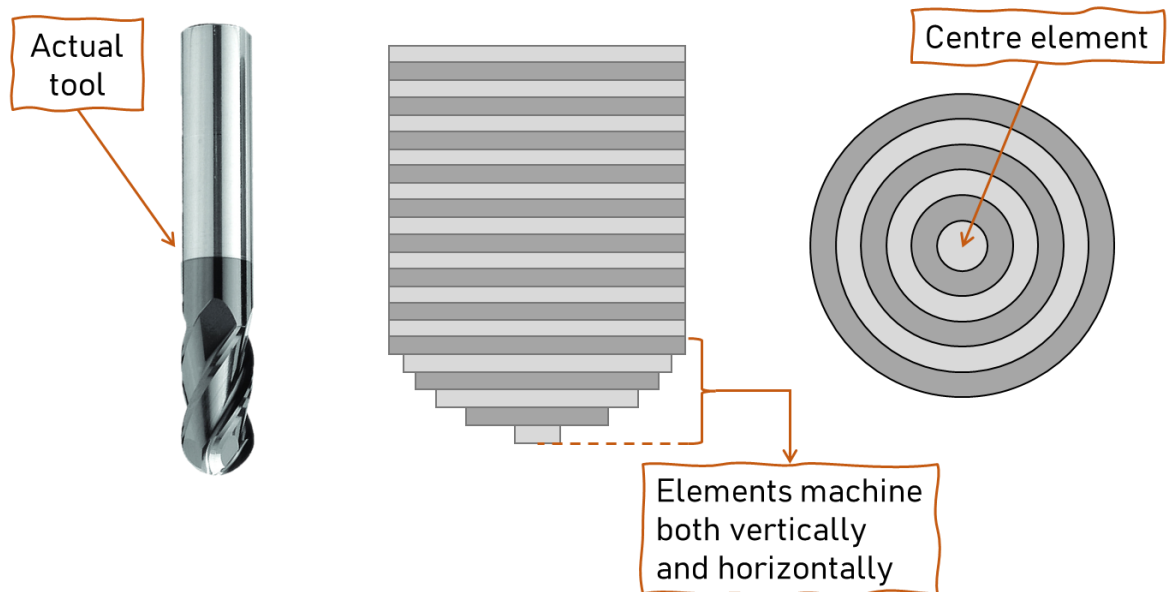


Figure 42 Ball nose cutting tool mesh Left: Side view of the actual tool. Centre: Side view of mesh. Right: Bottom view of the mesh

Each ring is identified by two parameters.

1. Distance from the tip of the cutting tool. This is the Z axis distance between the ring centre and the lowest Z coordinate that the tool can machine
2. Distance from the centre. This is the external radius of the ring.

No two rings are permitted to have the same values for both identification parameters.

Cutting tools have to be defined by the user before the simulation begins in order to enable the simulator to create the virtual carousel. The simulation engine that has been developed for this research project generates the cutting tool meshes based on the user inputs and populates the created virtual cutting tool in the classes shown in Table 7 and Table 8. Then the virtual cutting tools are stored in CuttingTool and CuttingToolProfile database tables that retain cutting tool information as well as data for each cutting tool mesh element. The simulator reuses the cutting tools and after every use, each tool element is updated accordingly. Common cutting tool geometries, such as end mills (Figure 41) and ball nose cutters (Figure 42) can be defined using this approach. It can also be applied to more complicated face and plane milling tipped cutters and other application-specific tools as required.

4.4.3 Billet mesh

Regardless of the shape of the physical billet, the simulation engine generates all billets as cuboids which are aligned with the machine coordinate system. A billet's cuboid is defined by the coordinates of its 2 opposite vertices. For consistency (and practicality during calculations) the first vertex is the one defined by the minimum values of X,Y,Z coordinates and the second vertex is the one where all coordinates have their maximum values. Therefore, the cuboid is defined by 2 3D space points $(x_{\min}, y_{\min}, z_{\min})$ and $(x_{\max}, y_{\max}, z_{\max})$. If the billet has a cuboid shape itself, then the mesh is generated by dividing the cuboid into smaller cubes whose edge length is equal to the element size (Figure 43).

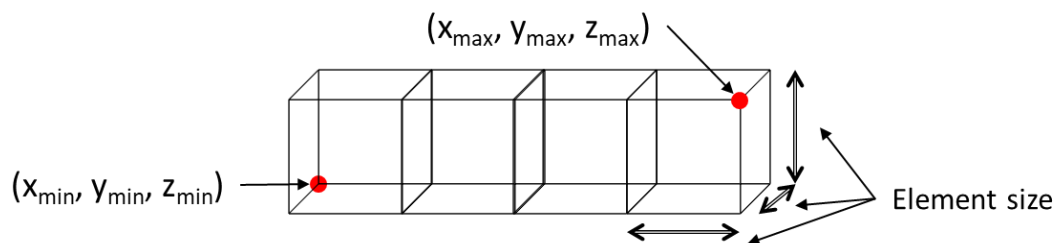


Figure 43 Basic billet mesh

If the billet size in one dimension divided by the element size has a remainder then this dimension is shortened until the remainder is zero. For example, if the dimensions of a billet are 55.3 x 45.7 x 10mm and the element size is 1mm the modelled billet dimensions will be 55mm x 45 x 10mm. In this example, the billet mesh will consist of 24750 cubic elements.

Since the billet may not be a cuboid or it may be pre-machined the simulation engine is capable of enacting the following 3 steps to ensure that an accurate billet representation is created.

Step 1: Definition of billet parts. The user must define the billet shape by combining billets with basic shapes through the provided GUI. The current version supports cuboids and cylinders. The user inputs multiple, simple-shaped billets first and then creates a complex billet that is the product of merging the simple ones. It is worth stating that in most real-life applications milling billets are usually simple cuboids in nature.

Billet Library :

BilletId	Description	Xmin	Xmax	Ymin	Ymax	Zmin	Zmax
5	Bottom Part	0.0	50.0	0.0	50.0	0.0	10.0
6	Top Part	10.0	40.0	10.0	40.0	10.0	20.0
7	Complex shape Billet	0.0	50.0	0.0	50.0	0.0	20.0

Figure 44 Specification of a complex-shaped billet (highlighted lines)

Step 2: Mesh generation for simple shapes. The simulation engine retrieves from the database the size and position of every simple-shaped billet and generates the mesh for this simple billet. If the billet is not of a cuboid shape, then a cuboid mesh is machined to produce the required shape. For example, if the required shape is a cylinder, then the simulation engine generates first a cuboid that the cylinder can fit in and then marks as machined all elements outside the cylinder boundaries.

Step 3: Composition of simple shapes. Finally, the container mesh is created. This is a cuboid mesh that can contain all simple-shaped meshes. Initially, all elements of the container are marked as machined. Then for each simple shape, the simulator copies the non-machined elements of the simple shape to the container. For example, if a simple shape has an element at position (x_s, y_s, z_s) which is not machined then the element of the container mesh at position (x_s, y_s, z_s) is labelled as non-machined. If two simple shapes overlap, then the result is the union of their non-machined elements.

Throughout the mesh generation and machining simulation process, every cube/element is identified and stored in computer memory by its vertex with the minimum axis coordinates $(x_{\text{element-min}}, y_{\text{element-min}}, z_{\text{element-min}})$.

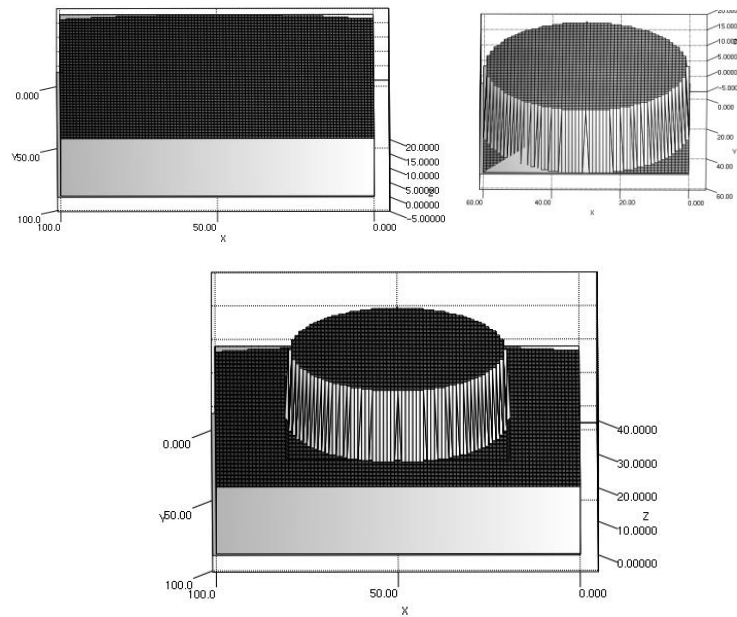


Figure 45 Composition of simple shapes (based on Figure 44 data). Top: Two simple shape billets. Bottom: Complex billet

4.4.4 Machining simulation

In section 3.4 process digital replication followed by data analysis was proposed as a means to embed into the simulator available theory as well as knowledge that can be extracted after training ML models with the monitoring data (Figure 13). The VMC simulator implementation demonstrates this approach by replicating the material removal process and then using the results to calculate other directly related parameters. Key parameters in this process are material removal rate (output of virtual replication) and spindle load (output of derivative data models) which have been used by the author to verify the effectiveness of the approach. However, the work has expanded to more parameters (mentioned in the following section and in Chapter 5) since it is equally important to stress the potential of this work and the multiple benefits of applying its outcomes.

4.4.4.1 VMC process replication

As soon as the java arrays are loaded with data (all data in the offline scenario, or at least one sample in the online scenario) the simulation engine starts. The simulator is event based therefore each sample represents a new event described by the new state of the cutting tool. For each sample, the simulation engine

calculates the effects that the new state of the cutting tool has on itself and the billet. Examples of effects are the removal of billet material, change of billet shape and cutting tool wear. The process that the simulation engine follows to digitally replicate the process and calculate the virtual monitoring parameters is as follows.

Firstly, the engine checks if the cutting tool ID indicated by the sample matches the ID of the cutting tool that is already loaded into the machine. If not, the mesh of the new tool is loaded. Secondly, the tool is moved to the position dictated by the coordinates of the sample being processed. To optimise the process, a series of checks are done to verify whether the cutting tool has touched the billet and which part of the cutting tool has done so.

Check 1: Is the cutting tool's bottom face below the top surface of the billet?

$$Z_{Tool} < Z_{BilletMax} \quad 4-1$$

Where:

Z_{Tool} : Z coordinate of the tool's bottom face

$Z_{BilletMax}$: Z coordinate of billets highest point (top)

Since this is a vertical milling process, if the cutting tool's bottom face is above the top surface of the billet there is no material removal taking place. The virtual monitoring system then records a default set of values which represents the no machining state (for example zero material removal rate). If the cutting tool is within the Z axis range of the billet, then the simulation engine picks a cutting tool element and checks if this element can machine part of the billet.

Check 2: Is the cutting tool element within the billet's Z axis range?

$$Z_{BilletMin} \leq Z_{ToolElement} \leq Z_{BilletMax} \quad 4-2$$

Where:

$Z_{BilletMin}$: Minimum Z coordinate of billet's elements (lowest point)

$Z_{ToolElement}$: Z coordinate of the examined cutting tool element

$Z_{BilletMax}$: Maximum Z coordinate of billet's elements (highest point)

If the element is out of the billet's vertical boundaries, then a default set of values for non-machining elements is recorded. If the element is within range the checks continue.

Check 3: Is the cutting tool element within the minimum and maximum billet X and Y axis coordinates?

$$X_{BilletMin} \leq X_{ToolElement} \leq X_{BilletMax} \quad 4-3$$

$$Y_{BilletMin} \leq Y_{ToolElement} \leq Y_{BilletMax}$$

Where:

$X_{BilletMin}$:	Minimum X coordinate of billet's elements
$X_{ToolElement}$:	X coordinate of the examined cutting tool element
$X_{BilletMax}$:	Maximum X coordinate of billet's elements
$Y_{BilletMin}$:	Minimum Y coordinate of billet's elements
$Y_{ToolElement}$:	Y coordinate of the examined cutting tool element
$Y_{BilletMax}$:	Maximum Y coordinate of billet's elements

Like the Z axis element check this one checks that the cutting tool element enters the billet X, Y axis boundaries. If not, default (no machining) values are recorded. If yes, the checks continue but from a billet's perspective. Yet, it is not verified whether the cutting tool element can machine the billet. Figure 46 demonstrates the reason why extra checks are required.

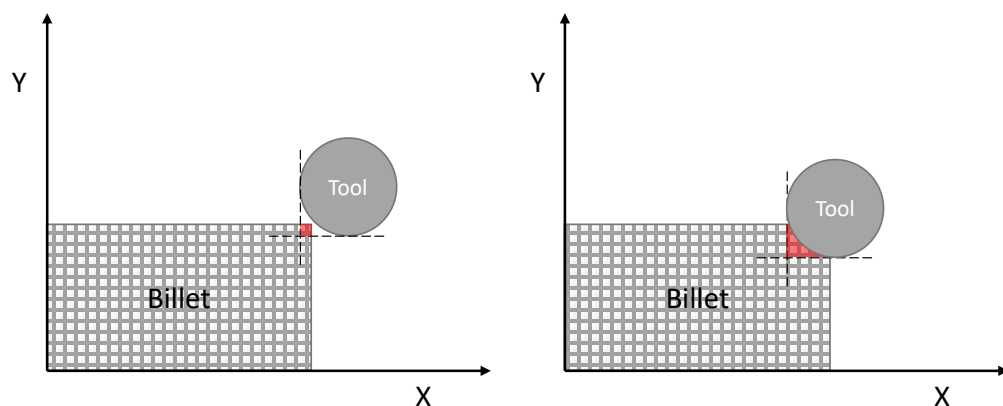


Figure 46 Process top view. Left: Non-machining cutting tool element. Right machining cutting tool element.

Check 4: Are there billet elements within the range of the cutting tool element?

The engine iterates over all billet elements that are indicated in red in Figure 46 and calculates the Euclidean distance between each element and the cutting tool element centre. If the distance is longer than the element's radius then the billet element is not machined. If the distance is shorter then this billet element is selected for the final check.

Check 5: Is the billet element machined?

If the billet element has already been machined, then the default (non-machining) values are recorded by the simulation engine for the cutting tool element. If the billet element has not been machined, then the simulation engine registers this billet element as machined and calculates the virtual monitoring parameters affected by the element machining.

Checks 4 and 5 are done at the same time but are presented as separate steps for a more transparent explanation. Checks 1-4 are optional from a theoretical perspective as they simply narrow down the elements that need to be checked. The theoretical equivalent would be to check all billet elements for every cutting tool element. In practice, this would increase the number of calculations by orders of magnitude and therefore the computer would be challenged to run the simulation engine at an acceptable precision, in parallel with the real process.

After all machined billet elements have been registered and recorded under the cutting tool element that machined them, the simulation engine continues to the next sample (if available). In offline mode, the simulation engine continues until all samples are processed. If there are no more samples available, it passes the output of machining to the derived data calculation models. In online mode, the engine output is passed to the derived data calculation models after every sample to allow for the immediate publishing of sample results by the output interface, presented to the user through the GUI. Finally, when the data stream is closed, the process summary is passed to the output interface to generate the final report and to store the required logs in the database.

The results of VMC process digital replication are a direct product of the material removal related calculations more specifically the calculated parameters are:

- Material removal rate
- Cutting tool usage (3-dimension profile)
- Part shape (raw data)
- CNC machining program status (percentage of program that is completed)

The Material Removal Rate (MRR) is calculated by counting the number of billet elements machined per time unit. This calculation is based on the following equation:

$$MRR = \frac{ELR}{dt \cdot ELSize^3} \quad 4-4$$

Where:

MRR: Material Removal Rate (mm³/sec)

ELR: Billet Elements Removed (-)

dt: Time between processed sample and previous sample (sec)

ELSize: Element Size (mm)

Cutting tool usage is very important for cutting tool life estimation and for cutting tool management. The basis of this estimation is much researched and presents a real challenge. It can be stated that knowing more about the actual work done by each tooth of the cutter can be of great benefit. This was therefore set as an initial target for the analysis generated by the simulator. As a result of the number of insertions per tooth a cutting tool that has been used at a spindle speed of 6000rpm does not have the same life left compared to a cutting tool that has worked for the same time or done the same job at 1000rpm. Therefore, cutting tool usage is further broken down to:

- Insertions per tooth
- Billet elements machined per cutting tool periphery element (Figure 41).
- Billet elements machined per cutting tool bottom face element.

The insertions per tooth calculation formula can be written as:

$$IpT = \frac{SS \cdot teeth}{60} \cdot dt$$

Where:

- IpT*: Insertions per tooth (insertions/sec)
- SS*: Spindle Speed (rpm)
- Teeth*: Number of cutting tool teeth
- dt*: Time difference between the recording of the processed sample and the previous sample (sec)

The number of insertions of each tooth of the cutting tool is added to previous records for the specific cutting tool and stored in the database of the simulator. For each cutting tool, a log of total insertions is kept and the information can be used by the operator to manage long process runs that may exceed the life of the tool. This prevents disruptions, potential damage to the VMC and part rejections. This approach matches the one normally adopted in industry where some aggregated measure of tool usage is applied to estimate the remaining useful life of a cutter.

Billet elements machined per cutting tool element (side or bottom face element) is a simple count of the number of billet elements that each cutting tool element has machined. The counted elements are then stored in the database and similarly to insertions per tooth can be used during process planning. Figure 47 shows the output provided to the operator regarding cutting tool usage. The X axis shows the distance from the bottom face (top) and the distance from the centre (bottom). The Y axis shows the number of elements machined.

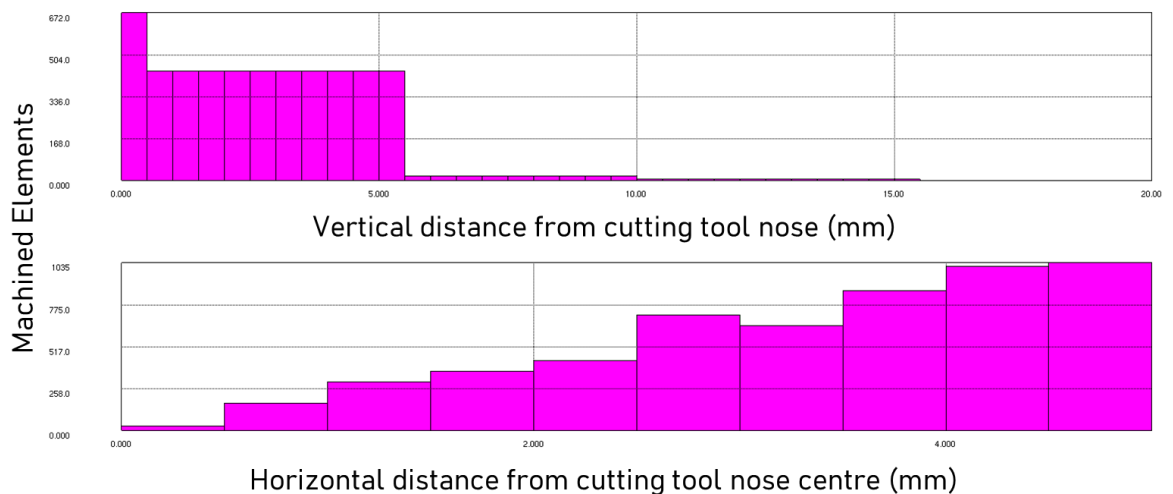


Figure 47 Cutting tool usage plots. Top: Tool side usage. Bottom: Tool bottom face usage

The above diagram is produced by simulating the machining of the cylinders that formed part of the previous work (Hill 2020). The diagram shows that the cutting tool tip element is being used by both dimensions and therefore is used significantly more than other tool elements. This is in accordance with the cutting tool tip wear that was observed during the process (Hill 2020).

4.4.4.2 Digital VMC derived data calculation

Running the virtual process by generating and machining a mesh of virtual elements allows for the studying of every aspect of the interactions between the cutting tool and the billet. In the previous section, the way that material removal related parameters are calculated was presented. The results of these calculations are then used as input for ML models that the learning module builds through a best model search and supervised learning process that is described in section 4.5.

The VMC monitoring system makes available measurements of the spindle load (SL). It can be assumed, based on established theory and on-site testing of this VMC, that SL is closely related to the condition of the tool, MRR, the spindle speed and the type of milling. The simulation engine does not have a hardcoded calculation formula as it does for the parameters of section 4.4.4.1, but instead, it uses the model that is stored in the database for the SL calculation. SL has been selected because the data needed to perform supervised training and to test the system is available through the previously deployed system (Hill et al. 2019). In general, for every parameter that can be monitored by the physical VMC, the simulator can develop a ML-based model to estimate it. The model selection and training process are described in section 4.5. It should be stated that the use of this data for this purpose in this research is novel and has not formed part of the previously conducted work.

The steps of calculating all derived data parameters are the same for all parameters. After the digital replication results have been calculated (all samples for offline simulations and at least one sample for online simulations) the simulation engine retrieves from the database the calculation models. Weka API (Witten et al. 2016) is used to rebuild the model (deserialization). The reconstructed model is a Java object with various functions one of which is to calculate the target

parameter from a set of input parameters. Parameter calculation speed depends on the calculation model but since the models are already trained calculation time is typically short enough to allow for real-time simulation. Figure 48 shows the process that the simulation engine follows to calculate the target parameter which for the current simulator's implementation is the SL.

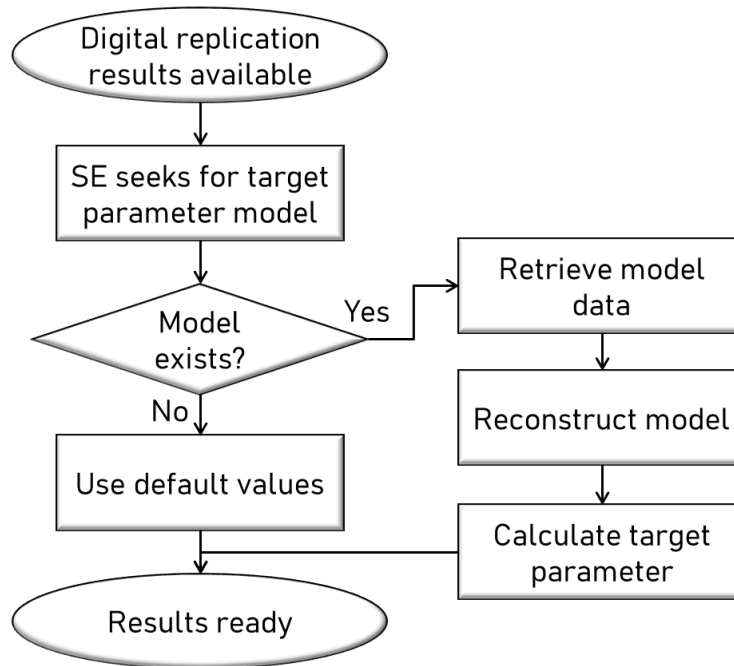


Figure 48 Derived data parameter calculation

In the case that no model exists the simulator will return zeros for parameter values along with an indication that there is no calculation model available. This is to prevent results from being pushed to the output interface. In the current implementation of the simulator the calculation models are retrieved only once if simulating offline and after every sample is processed if the simulator runs in real time.

After all calculations are finished the SE pushes the results to the output interface and updates NC table fields shown in Figure 38 in the database that keep information for the simulator's historical runs. The analysis CSV file that is created has an extra column added to show the estimated value of SL for each sample. As a last note, the process described in section 4.4 can be run with or without pre-trained models in the database. If no models are available, then this will be a traditional simulation process. This is typically the case when the simulator runs for

the first time a specific cutting tool – part material combination. The latter shows the evolving nature of the simulator that starts with a default, “traditional” behaviour and as new data becomes available it has the mechanisms to use the generated knowledge embedded in the ML models. In section 4.5 the details about how the ML models are selected, built, and stored in the database are provided so that the description of the evolving nature of the next generation simulator is complete.

4.5 MILLING LEARNING MODULE

In section 3.5 the description of the learning module covers a wide range of applications. Since it is not practically possible to demonstrate every case that the suggested approach can support compared to traditional simulators the VMC simulator focuses on three key functionalities:

1. Best ML model search
2. 1-sample-based model training and management
3. n-sample-based data synchronisation

These three functionalities enable the simulator to generate knowledge from the information extracted by the monitoring data in the most efficient way and to use this knowledge to evolve. Due to the extensive programming effort required to adapt the external libraries that support the Learning module, some processes are not fully automated as it would be required for a complete solution. Nevertheless, results that prove the concept can still be calculated since the learning module operates independently and asynchronously with the simulation engine. Section 3.5.2 explains how this is achieved.

4.5.1 ML model selection

Selecting a suitable model to calculate derivative parameters is a process with a very high impact on the performance of the simulator. The model type determines key simulation performance indicators including calculation accuracy, the range of application, model training speed and results calculation speed. As already mentioned, the parameter that has been selected to demonstrate the learning module’s operation is SL. SL is measured by the VMC as a percentage of a baseline

value and therefore there is data available for supervised training of the selected model type.

Building a ML model requires first specifying the input and target parameters. For the datasets that this work used to test the VMC simulator the selection was relatively straightforward since there was a strong indication that the MRR calculated by the simulator was related to the actual SL. Figure 49 shows part of the initial raw data graphs produced during the simulator's development phase.

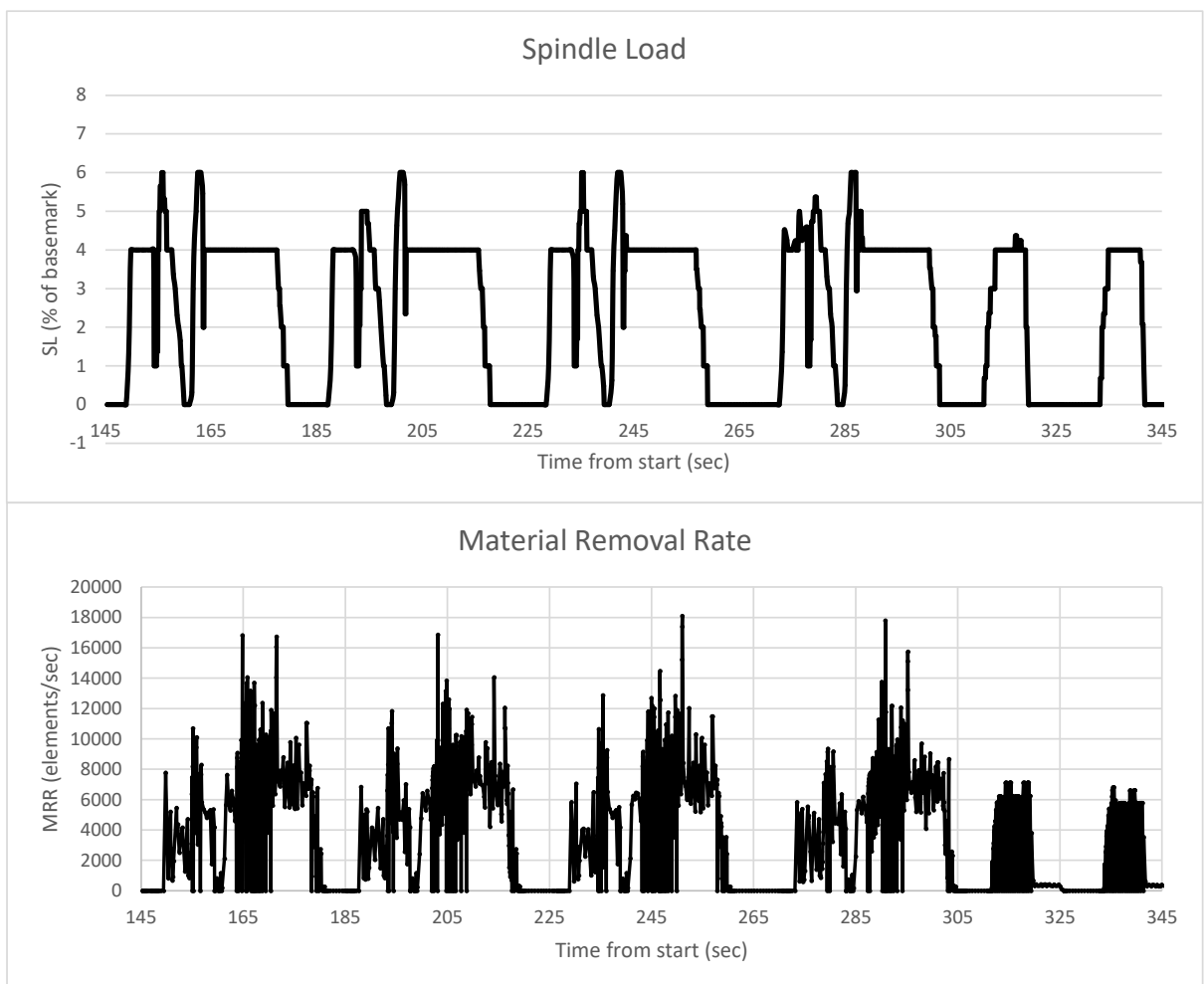


Figure 49 Top: Actual spindle load. Bottom: Simulator calculated Material Removal Rate

Figure 49 shows the first success of this research project since it verifies that the simulation engine correctly replicates the process. Full details are discussed in Chapter 5. Regarding model selection, the graphs led to further investigation which showed that after smoothing the data with a moving average of 40 samples MRR can be used as input to a linear regression model that gives a good prediction of SL. It should be noted that the actual SL values have relatively low resolution

compared to MRR since the monitoring system generates only integer percentage values.

As explained in 3.5 finding the best ML models manually requires a lot of effort and a high level of specialisation in machine learning. In addition, multiple models need to be built for each parameter of each cutting tool – part material combination. To overcome this issue WEKA (Witten et al. 2016) and Auto-WEKA (Kotthoff et al. 2019) libraries are embedded in the implementation of the learning module which together automate the best ML model identification process given appropriate setup parameters. Figure 50 shows the model creation process.

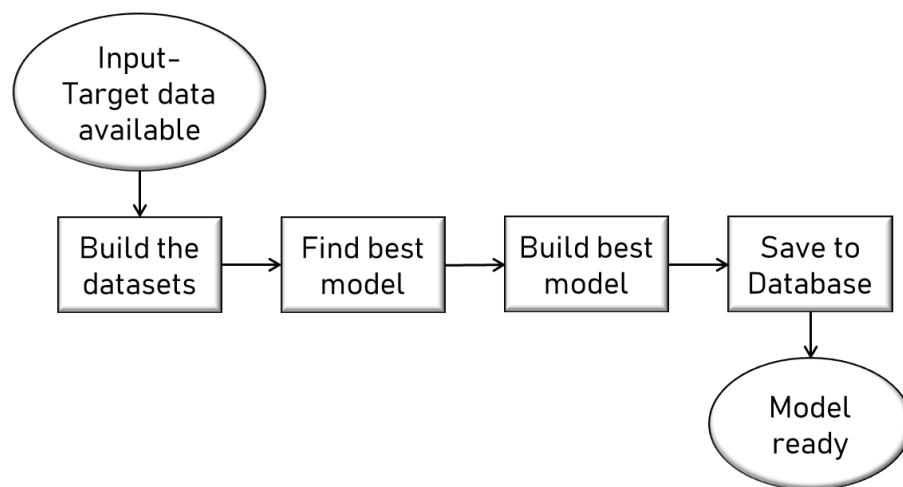


Figure 50 Automatic ML model generation

To create the model, a dataset to base the ML model on must be generated. To compile the MRR-SL dataset two sources of data are used. Data generated by the VMC monitoring system and published by the system of Hill et al. (2019) is passed to the input interface for the typical pre-processing. All target parameter values (in this case the SL) are passed directly to the learning module. The simulation engine is using the monitoring system data to run the VMC digital replication process and calculate the learning model input parameter values (in this case the MRR). Then, the results are fed to the learning module which builds a new dataset that contains only the input data as received by the simulation engine and the target data as provided by the input interface. It should be noted that if a target parameter is not related to the digital replication, then the user may select input data to come directly from the input interface. Technically, all input, output and temporary

system parameters are written in a CSV file. The user may select any combination of these parameters to create models.

After the dataset is compiled, the learning module builds the required Java objects and runs the Auto-WEKA model search functions. Searching for the best MRR-SL model took 13 hours on a computer with 16GB of RAM and a 2.0GHz dual core CPU (Intel® Core™ i5-4310U Processor. 2014). The performance numbers are provided to point out that this implementation of model searching is not currently suitable for real-time ML model searching with common embedded computing systems. However, modifications to the model search algorithm, the use of state-of-the-art computing systems locally or through cloud services and better dataset selection and management techniques could produce good enough models within minutes after the process started.

Auto-WEKA returns a model type and the optimum model parameter values. The learning module reads the results and uses WEKA functions to build the model as suggested by Auto-WEKA. The virgin model is then trained with the available data which initially is the dataset used to find the best model. This is a typical supervised learning process that is done by WEKA. Since WEKA can be used through its standalone graphical user interface the model can be tested there for verification or validation purposes. The trained model is then serialised and stored in the database by saving a file on the hard drive containing the model and updating the learning set table with the new model file path. From there, the simulation engine can retrieve the calculation model and use it as described in section 4.4.4.2.

At the end of the model creation process the learning module has added a submodule (see section 3.5) to the list of available submodules which are stored in the learning-set and learning-input database tables. The addition of more submodules increases the capacity and capabilities of the simulator because every submodule enables the simulation engine to calculate a new parameter. In other words, the simulator at the beginning of its life can run specific predefined parameters but as more submodules are created it can cover more needs of a digital twin's needs. This is a critical part of the implementation and together with

the portability of the database that allows for sharing of the submodules are key benefits of the suggested simulator architecture.

4.5.2 Model training and lifetime management

A major drawback of the submodule creation process is that it is based on a specific dataset and therefore the model will inherit the issues of the dataset. For example, if the dataset was generated from a problematic VMC process run then the model will only represent a problematic run. The simulator does not have an internal mechanism for identifying problematic runs but this can be mitigated by simply not creating submodules from rejected part data. In the long term, if problematic data is only a small part of the overall data, then the model will be less biased towards problematic runs. Retraining the models can naturally achieve that.

The current version of the simulator does not support retraining of an existing ML model since this requires additional programming effort to connect the data streams from the input interface to the learning module and also to merge WEKA and Auto-WEKA with the new libraries. To update an existing model, all historical data used to build the model is required so that 'old' data is merged with 'new' data and the model is trained with the combined dataset from scratch. For completeness, a logical way to do automatic retraining would be by integrating MOA (MOA. 2021) which is based on the work of (Bifet et al. 2010). MOA is used for data stream mining and it uses a machine learning model library with the capability to incrementally train a ML model. The incompatibilities between the three packages are being actively solved by the community supporting MOA but there is still work required which lies outside the scope of this research work.

The lack of a proper retraining mechanism makes the retention of monitoring data mandatory. This is clearly incompatible with the logic of the architecture which is to retain a minimum amount of data. The operation and results of the VMC simulator show that this weakness although significant in theory it does not affect the actual performance in all cases. The MRR-SL model required a few hundred of samples to calculate accurately the value of SL given the MRR for all runs of the VMC with the same cutting tool-part material combination. One VMC run produces tens of thousands of samples so either keeping the small part of data that is enough to

train the model or completely deleting data (since the model's accuracy suffices) are both reducing massively the amount of data that has to be retained. Chapter 5 provides a quantification of data storage minimisation. If, finally, the retraining issue must be mitigated then the integration of MOA (or similar) library is the proposed solution.

A key factor to consider when using the simulator is VMC hardware wear. The cutting tool is a typical example of hardware whose wear potentially changes its behaviour even during a single run. Neither the simulation engine nor the learning module account for wear if a wear-related input parameter has not been provided to the relevant submodule. In the cutting tool's case, the learning submodule responsible for SL calculation would require an additional parameter which is related to the tool's usage. If the cutting tool is machining the same type of material, then the user can add two input parameters to the model creation dataset. These two parameters can be:

- Cutting tool lifetime machined elements count
- MRR.

The learning module then creates a suitable ML model that remains accurate throughout the cutting tool's life. Since the models rely solely on the data that has been provided to build the model every behaviour can be modelled if it has been observed. However, there is no guarantee that a behaviour is predicted accurately if it is outside the range of the model-building dataset. Overall, the simulator can be a strong foundation for many new findings since the connection of a virtual process that can be monitored as required with any observation in the actual process which can be described with data is possible without the need for separate bespoke model implementations.

4.5.3 Synchronisation for n-sample data models

In section 3.5.4 n-sample-based learning was presented to complement the types of learning a simulator will generally perform. As explained, the model-building process remains the same and therefore the implementation of the VMC simulator demonstrates it with the MRR-SL model. The difference in n-sample-based learning is that if samples from multiple runs are used then the samples need to be

synchronised. As an example from the VMC, the cutting tool machines a specific point on the billet after 500 seconds from the beginning of the process and after the monitoring system has produced 1,500 samples. During a second run for an unknown reason the same feature was machined after 501 seconds from the beginning of the process and the monitoring system had already generated 1,535 samples. It is clear that both time from start and sample number are not suitable properties to use in order to synchronise the two datasets.

As a potential solution to the n-sample data synchronisation problem, section 3.5.4 proposes DTW. The VMC simulator implementation uses DTW as a means to compare different datasets, measure their similarity, and align the samples of the datasets. To embed DTW into the simulator, a Java library built after the work of (Salvador and Chan 2007) has been used as a starting point. Then, ideas that have been reported by Shokoohi-Yekta et al. (2015) were used to extend the DTW library and finally, the necessary adaptations were done so VMC data requirements regarding format and filters were satisfied.

Data synchronisation is an additional function of the learning module required for n-sample data learning. To create the model the learning module needs the parameters of reference to be specified. If the ML model calculates a parameter which depends on the feature of the part that is being machined then the parameters of reference should be the cutting tool coordinates. This is the typical setup for the majority of cases because the result is a direct comparison of the behaviour of the VMC between two runs. To demonstrate that, Figure 51 shows the same two datasets synchronised with different parameters of reference. The output is MRR which depends on the feature that the VMC machines. As indicated by the difference between the two traces, synchronisation based on tool coordinates produces excellent results (top graph) while synchronisation based on sample number is producing rather misleading outputs (bottom graph).

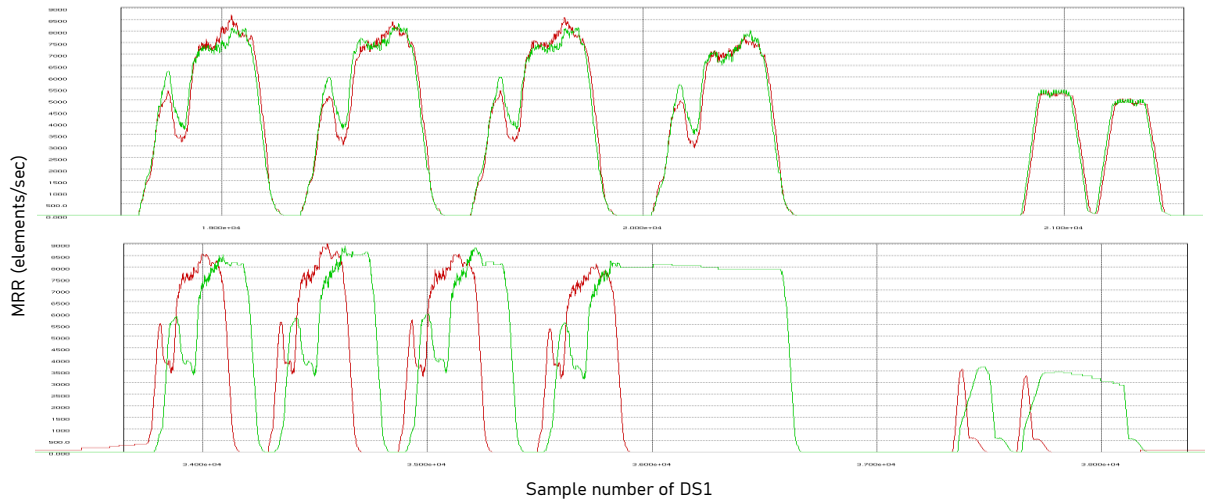


Figure 51 Examples of synchronised datasets with different reference parameters.
 Top: Cutting tool coordinates. Bottom: sample number

To build an n-sample-based data model, the simulator runs the whole process and calculates the target parameter. The learning module then synchronises the simulated process data with the actual VMC process data to make them comparable. Finally, the Learning module builds the model as described in section 4.5.1 using for target value the synchronised dataset from the VMC monitoring system. Figure 52 summarises the process.

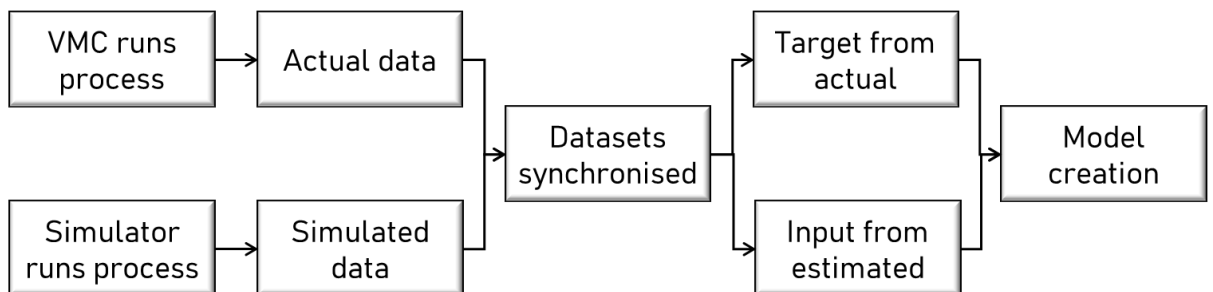


Figure 52 n-sample data model creation process

In the current version of the simulator, the full process is not automated but the steps of simulation running, data synchronisation and model creation have to be manually setup and run by the user.

With n-sample learning and synchronisation, the full capabilities of the VMC simulator have been described. The simulator demonstrates the key elements that make the architecture superior to other simulation systems and provides an example of how the evolution of the simulation mechanism is achieved. The actual

results and performance of the simulation engine – learning module combination are presented in Chapter 5. As a last note, there is a very high level of flexibility on how the learning module capabilities can be used and the demonstrated examples are a small percentage of what could be developed. In addition to the novelty of the implementation itself, the expectation is that it can become the foundation for more research contributions based on the quick modelling of the milling process characteristics and behaviour.

4.6 OUTPUT INTERFACE

There is not only one way to implement the output interface since the implementation depends on the needs of the simulator users. In its current form the VMC simulator has three ways to report the results:

- CSV files
- API for local apps (feeding a purpose-built GUI)
- API for web services

These reporting methods served different purposes during the simulator's development and are not necessarily needed or relevant to non-research applications. Retrospectively, the output interface could be further abstracted using more scalable technologies however this implementation covers the needs of the case study and since both the output interface and the user applications were developed by the author a more sophisticated connection would add complexity without any research value. In the following sections each method is explained.

4.6.1 CSV report files

From the beginning of the simulator's development the CSV files retaining available and calculated data were key in verifying all functionality and providing a quick way to access the simulation results. Applications such as MATLAB (MATLAB. 2022) or Microsoft Excel (Microsoft Excel. 2022) can import CSV files which allows for quick data plotting or data manipulation testing that was then implemented in Java programming language and embedded in the simulator. Similarly, the CSV files were used to test different ML models in WEKA software and develop the algorithm that handled all actions from data selection to final model creation. The format,

contents and naming conventions of the CSV files remain consistent therefore details provided in section 4.2.1 apply to the output interface.

All CSV files are registered in the database and stored as flat files in a predefined folder. External applications can access the files if they are granted access to the folder that contains them. This is the data file method described in section 3.7. Since Java is portable the simulator can run and generate the files regardless of the computer operating system that hosts it. The simulator does not perform any checks on whether the file is manipulated or completely deleted by another application however, unless the user specifically retrieves the file to do further analysis, the simulator itself does not re-use the output files therefore its operation is not affected. If run in debugging mode, the simulation execution can be paused and, in that case, the temporary CSV files can be examined if the simulation process needs to be checked.

A drawback of CSV files is that they contain 2-dimensional data. However, 2-dimensional representation is not suitable to report all results and therefore additional ways of results publishing are required.

4.6.2 Simulator API for integration

In the description of simulator boundaries (section 3.1) it is pointed out that the simulator is meant to be integrated into another system. To enable integration an Application Programming Interface (API) has been developed so that the application that integrates the simulator is able to use its functions. The integration is very similar to the way that the simulator itself integrates WEKA, DTW and all other libraries that are supporting the simulation process. The API provides full access to top-level functions that enable the wrapper (Definition of wrapper | PCMag. 2022) to run a simulation as well as to more technical functions that can manually control the behaviour of each element of the simulator. The implementation is split into separate projects that can be integrated into other applications using MAVEN (Apache Software Foundation 2022) a Java project management tool or simply imported as Java archive (jar) files (JAR File Overview. 2019).

In the current version, the API is used by the GUI that has been developed to create a standalone version of the simulator. Although a GUI is essential for development verification and for running the simulator as a standalone application, by design it is a wrapper that uses the simulator as an internal tool. The GUI is a separate project that imports the simulator and maps the graphical elements (buttons, text fields etc) to specific simulator functions. This in combination with Microsoft Excel, MATLAB and WEKA which were used to process the CSV output files is the full range of applications that were used to verify, validate, and run the simulator to produce the results that are presented in Chapter 5. Since the GUI is not part of the simulator it is presented separately in Appendix A.

The top-level functions that the API provides access to are for:

- Creating and managing billets
- Creating billet materials
- Creating and managing cutting tools
- Loading tools to the VMC carousel
- Running the simulation engine
- Running the learning module

As mentioned earlier the API provides access to more technical functions such as modifying the output CSV files or using the data pre-processor as a standalone function but these are complementary details and not the main functions that a wrapper would use. To explain how the API can be utilised, a setup, run and train scenario is described in steps. It is assumed that the application that integrates the simulator has first imported the relevant MAVEN dependencies or JAR files.

Step 1. Create a new part material.

The material of the part is used by the learning module to categorise its models. This is because machining two billets with the same geometry but of different materials produces different process results and therefore different monitoring data outputs. The new material can be added with the following method:

```
MaterialUtils.addMaterial(name);
```

Name is used to describe the new material so it is easier for the user to identify it and select it. After running this method the material is added to the database and therefore becomes available to be assigned to a billet.

Step 2. Create a new billet

The second step is to create the billet that the simulator will machine and calculate the virtual process outputs. A billet is in general a shape with a material property. Billet generation is done by calling:

```
BilletUtils.addBillet(  
    billetName, billetShape, materialId,  
    billetXMin, billetXMax,  
    billetYMin, billetYMax,  
    billetZMin, billetZMax  
);
```

Billet name describes the billet. Billet shape is indicated by an integer, and it is 1 for billets that are a combination of multiple shapes, 2 for rectangular-shaped billets and 3 for cylindrical-shaped billets. Material id is an integer that identifies the selected material for this billet. Billet x,y,z minimum and billet x,y,z maximum are the two edges of the smallest rectangle that can contain the billet (see Figure 43). After running this method a new billet is created in the database.

Step 3. Create a new cutting tool

To machine the billet a cutting tool must be created. This is done by running:

```
CuttingToolUtils.addCuttingTool(  
    toolName, toolType, toolSeries,  
    toolTeeth, toolLength  
);
```

Tool name describes the tool. Tool type can be end mill, ball nose mill or slot mill. Tool series is an identification number given by the manufacturer that is used by the learning module to identify cutting tools with identical initial characteristics. Tool teeth is the number of teeth that the tool has. The simulation engine uses the number to calculate the feed per tooth. Tool length is the distance between the tool's bottom face and the spindle chuck. This measure is specific to the tool as it is fixed in the VMC tool holder since the monitoring system records the coordinates of

the spindle and not the cutting tool coordinates. This method creates a cutting tool in the database which is then available to be loaded into the virtual VMC carousel and also to be used by the learning module to create a new calculation model.

Step 4. Load the cutting tool to the virtual VMC carousel

The created tool is then loaded into the carousel so the simulation engine can use it to machine the virtual billet (billet mesh).

```
CarouselUtils.addCarouselPocket(position, toolId);  
or  
CarouselUtils.updateCarouselPocket(position, toolId);
```

The physical carousel uses numbers to identify the available positions to place the cutting tool and the virtual carousel follows the same naming/identification convention. Tool id is the identification number given by the database for the tool that has been created by Step 3. If this is the first time that the simulator is run, then the pocket of the carousel should be added. Otherwise, the pocket can be updated with the cutting tool that is inserted into it.

Step 5. Run the simulation

After all simulation elements have been inserted into the database a simulation configuration should be created and then the simulator can run.

```
SimulatorConfig config = new SimulatorConfig();  
config.setBillet(billet);  
config.setInputFilePath(inputFilePath);  
config.setInputFileType(inputFileType);  
  
KPIs kpis = new KPIs();  
  
SimulatorUtils.simulateAnalysisFile(kpis, config);  
or  
SimulatorUtils.simulateGCodeFileUGS(kpis, config);
```

The simulation configuration object needs three parameters to be set. The billet that the simulation engine will machine, the input file that contains the part program and the input file type which can be either a 'G-Code' file or a 'CSV' file. Depending on the input file type the corresponding function should be run. This is

simulateAnalysisFile for CSV input files or simulateGCodeFileUGS for G-Code. An empty KPIs type object is also created and passed to the simulation engine. This object will hold the results of the simulation engine.

Step 6. Get results

The methods run in Step 5 load the results to the KPIs object and make it available to the application that calls them. A KPIs object holds data including:

- Mesh element size
- Minimum time difference between events
- The machined billet matrix containing true for each machined element and false for non-machined elements (see section 4.4.3).
- A list containing the cutting tools used during the machining process along with tool properties and usage information.
- The calculated parameters which include tool coordinates, material removal rate and all other parameters that were provided by the input file or that are calculated by the available learning module models.
- The text of the file that is generated by the virtual monitoring system. The text contains everything that is later written in the output CSV file

The KPIs object contains more information generated during the simulation process but the above list is what typically would be retrieved by the application that uses the simulator API.

With the above example, it becomes clearer that the simulator is developed to be embedded into another system. In a similar manner to the above example, all simulator functionalities could be used and all output data could be retrieved to be presented or reused by other systems. With this example, it can also be better understood why a CSV file is not adequate to report all of the simulation results. Each cutting tool in the list of cutting tools retained by the KPIs object contains a list of cutting tool elements. Each element has another list of independent parameters regarding element position, element usage and times that the teeth attached to the element have machined the billet. If this multilevel information is

entered in a CSV file then the output file would have tens of thousands of columns and lots of repeated data which makes it non-practical for full results reporting.

Overall, the API is a requirement for every simulator implementation and in this research work it plays a vital role in enabling user-friendly access. The developed GUI uses the API to access the simulator functionality and a user can quickly run the simulator and view results. The GUI and consequently the API have been used to demonstrate the offline simulator capabilities. The same API could be used to run it online, but modern manufacturing environments favour remote access and therefore it was decided by the author to demonstrate over-the-web access to the simulator through a web API.

4.6.3 Web API

When the VMC is running the operators of the machine need to access live information to ensure that there are no issues requiring their intervention. To spot a problem during the process, the simulator runs in parallel with the actual process and calculates the expected parameter values. Then these values are compared with the actual data and if the difference exceeds a threshold the operator is notified. To demonstrate this process a web application has been developed by the author. This accesses the simulator's real-time results and provides a graphical interface for the user to monitor the progress of part machining as well as the actual and estimated parameter values. This web application can be accessed with an internet browser and an internet connection that allows it to connect to the hosting web server.

Giving access to the simulator's functionality over the web requires a web API. A basic web API has been developed and it provides access to a limited set of functions that allow the web application to get live simulator results.

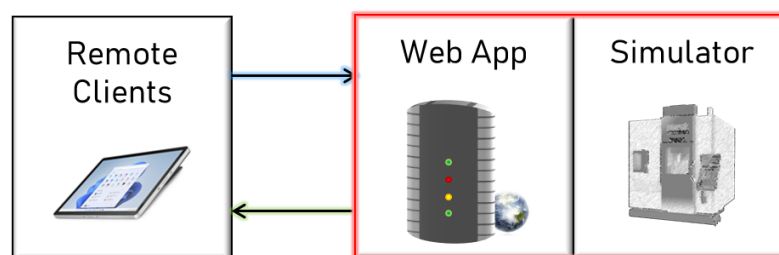


Figure 53 Remote access through the webapp

The logic behind the web application is the same as the GUI for local access. The web application is a wrapper system that integrates the simulator and provides its services to the users. It is hosted on an Apache Tomcat (Apache Software Foundation 2021) web server and it has been developed with Java programming language for the backend system and HTML and JavaScript for the frontend system (webpages).

To connect any application with the simulator over the internet (using Transmission Control Protocol/Internet Protocol) there are two steps to be followed. First, the input interface should be supplied with the machine setup (billet, cutting tools etc.) and the live monitoring data source. The setup information is then passed to the simulation engine that initialises the digital replication process. After successful initialisation, the input interface is waiting for monitoring data to arrive. Following initialisation, a second step is required so that the client application can receive simulation results. The client should establish a connection with the output interface by using Server Sent Events technology (SSE HTML Standard. 2022). With SSE the client connects to the server and waits for the server to send an 'event' which is in this case a JSON object containing the simulation results. As soon as the last sample has been simulated and sent to the client the SSE connection closes.

Due to the fact that it is not practically possible to keep the simulator constantly connected to the VMC monitoring system the above process has been altered in the version of the simulator used to produce the results of Chapter 5. The simulator uses data from a CSV file as monitoring data that arrives at the simulator at the exact time intervals that it would arrive from the actual monitoring system. This is achieved by reading the CSV file but submitting the sample to the input interface when the time from the virtual process start equals the time from start value of the sample. Regardless of the method that is used to get the monitoring data, the aim of this section is to explain how the simulator can be controlled remotely and produce results accessible by clients over the internet. As mentioned earlier, the web API is a basic implementation since full implementation like the API of section 4.6.2 would replicate the same functionalities and would not demonstrate additional capabilities of the simulator.

With the web API, the three ways that the VMC simulator can be integrated and/or utilised have been presented. This implementation is suitable for the purposes of this work and would need alterations in order to be deployed in an industrial environment. Section 3.7 is a more complete guide about technologies and methods that can be selected. In general, the purpose of both input and output interfaces is to connect the simulator with its environment, therefore, their characteristics are dictated by the characteristics and the technologies of the environment itself.

4.7 INTEGRATION INTO THE PHYSICAL WORLD

Although integration of a simulator is taking place in the digital world it is also important to provide an overview of how the described system is connected to the physical process. In Figure 54 the typical steps to run a CNC process on the actual VMC are depicted. First, the CAD design of the part is generated by the design engineer and then, through CAM software, it is translated into a G-Code file. This G-Code file is run on the machine that machines the part and at the same time generates data through its monitoring system. The simulator is simply connected to the data connections of the existing system and reads the inputs to the machine and also the outputs of the monitoring system. At the initial stage, there is no interference with the CNC process.

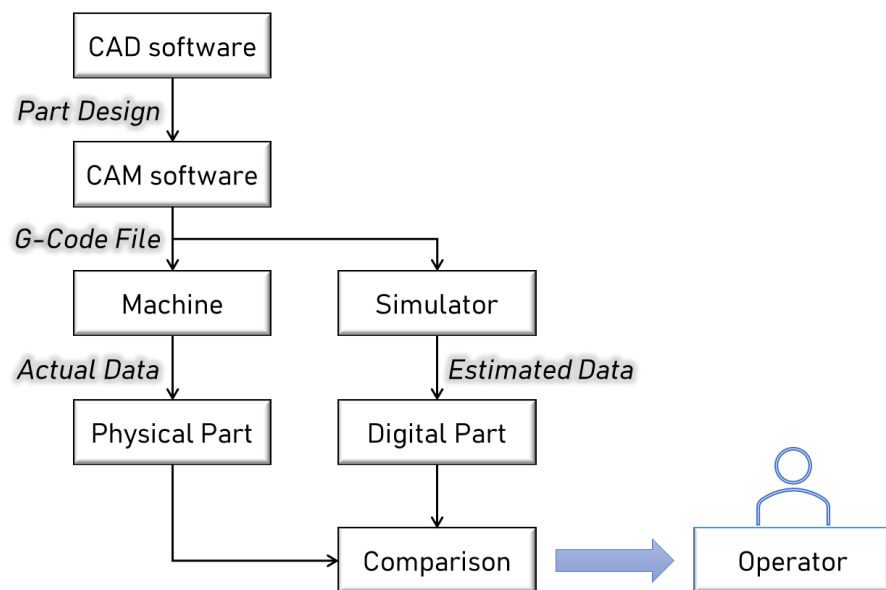


Figure 54 Simulator integration into a CNC process

The results of the simulator are provided to the operator who can spot the differences between the actual process and the expected process. Based on the observations of the operator, relevant actions are taken to bring the physical process parameters to the desired state. Provided that the simulator has been trained and gained the trust of the operator, the comparison between the actual and the simulated process can be done automatically, and the simulator can be connected to a digital twin that has access to the process control. Then, the digital twin acts within its control permissions to modify the process parameters and notify the operator about its actions or about the suggested actions that the operator should take. This way and step by step a traditional CNC process can be upgraded with a digital counterpart and then the digital counterpart that accumulates knowledge “next to” the operator can take over supervision and appropriate control tasks.

Although the above steps would offer great benefits to a company they have a cost both in terms of implementation of the digital counterpart and the time that is needed to develop a mature system that can be operated independently. It is therefore more appropriate that the above process is followed by OEMs since the creation of a digital twin specialised on a specific machine would be applied to hundreds or thousands of clients that use that specific machine. Reversely, clients generate data from hundreds or thousands of machines which gives the OEMs data for a wide range of cases and therefore enables them to bring the digital twin to maturity much faster than a manufacturer would do.

The above integration description is about CNC machines but it can be applied to every physical asset with digital counterparts. There are many examples of smart appliances where the manufacturer continues to have access to the device after product sale but due to obvious privacy and data ownership issues, it is often not possible to use the generated data for further development of the digital counterpart on a large scale. Ways to overcome such issues are outside the scope of this research work but as a final note, both OEMs and their clients have to work together in order to develop, apply and improve digitalisation technologies.

4.8 DEVELOPMENT TECHNOLOGIES

The software modules of the simulator have been developed using a combination of software development technologies. The project has been split into four sub-projects each one corresponding to the different application layers. The integrated development environment (IDE) used for all layers is Eclipse for Enterprise Java Developers (Eclipse IDE 2021-12 | The Eclipse Foundation. 2021) in combination with Git (Git. 2022) for source code change tracking and versioning and Apache Maven for managing imported packages. The biggest part of the code is written in Java programming language. In detail for each layer.

Persistence (data) layer: This layer is written in Java and Structured Query Language (SQL). It contains all classes responsible for accessing the simulator's database and for writing data files on the host computer's hard drive. The database used is SQLite (SQLite Home Page. 2022) and the driver used to access the database is the native Java database connectivity driver (sqlite-jdbc). The external libraries that this layer uses are:

SQLite JDBC	Driver to access the simulator's database
-------------	---

Utilities (controller) layer: All layers use utilities to interact with the persistence layer or to show information to the user. This layer is written exclusively in Java and uses the following list of external libraries.

Google JSON simple	Generates and parses JSON objects
Jzy3d	Generates 2D and 3D graph backend
FastDTW	Performs Dynamic Time Warping
UGS-core	Interprets G-Code
WEKA	ML library collection
Auto-WEKA	Automates WEKA ML model search

View (Graphical User Interface) layer: This package is the simulator's GUI that can be accessed locally. The GUI is developed using both Java's AWT and Swing

graphical interface components. There is one external library that is used for this layer:

JIDE common layer	Graphical interface components
-------------------	--------------------------------

Web application layer: The web application offering remote access to the simulator and connectivity with other software is built using Java, HTML and JavaScript. This layer implementation runs on an Apache Tomcat (Apache Software Foundation 2021) webserver and uses the following external libraries and scripts:

Apache Tomcat web server	Web server hosting the web application
Three.js	3D graphics for digital process view
Highcharts library	Interactive JavaScript charts
Micromodal	Generates the interface pop-up modals

The minimum requirements to run the simulator are mainly the requirements to run the Java Virtual Machine (JVM) plus the memory needed to build the virtual part and cutting tool models. As a starting point, it is recommended to use a system with at least 4GB RAM and a CPU with at least 1.6GHz speed per core. More memory enables the simulation of larger parts or of the same part with higher resolution. Higher CPU speed or more CPU cores reduce simulation time which is critical in real-time scenarios.

5 BENCHMARKING – RESULTS AND DISCUSSION

Architecture benchmarking can be done by applying the systems detailed to this point to build a simulator for a specific process and/or asset. Then the performance of the architecture can be quantified and at the same time, its applicability can be verified. The applicability of the architecture has been presented in Chapter 4 so this chapter will benchmark the associated VMC implementation. Although this single application does not represent the full capabilities of the architecture it provides enough evidence to support further developments and applications on other processes.

In this chapter, the performance of the simulator is measured against the values that are procured and recorded by the VMC monitoring system. It is ideal when a simulator can predict the actual behaviour and responses of the actual process during its operation. As such in the next section, the monitoring system values are considered as the target values that should be predicted. For performance parameters for which there is no monitoring data the benchmark value is stated along with the rationale behind choosing the specific value.

To demonstrate the efficacy of the approach applied the main indicators that are being assessed are:

- Simulation accuracy: monitoring data versus simulator estimations
- Knowledge generation: application of acquired knowledge to new parts
- Simulation speed: milling part program speed versus simulation speed.
- Data reduction: initial data size versus processed data size
- Data utilisation: processed data size versus total available data size.

In addition to the above, outcomes of this research work that are not part of the simulator's benchmarking but are significant findings are presented since these can be the starting points for future research. The graphs and figures presenting the results are generated through the GUI that has been developed by the author to control the simulator further proving the value of the engineered simulator. For all benchmarks and testing a laptop with 16GB of RAM running at 1867MHz and a 2.0GHz dual core CPU (Intel® Core™ i5-4310U Processor. 2014) was used.

5.1 SIMULATION ACCURACY

Accuracy is a key performance indicator for every simulation system, and it is a necessary part of every simulation project to test the accuracy in relation to the needs of the project. Due to the importance of this indicator, the first validation tests of the simulator were related to its accuracy in reproducing the process and calculating the output parameters. In Figure 55 the actual part is shown next to the representation of the part generated through the GUI using the local API and through the web application that reads information from the web API. This part has been used throughout Chapter 5 to demonstrate the results of the simulator.

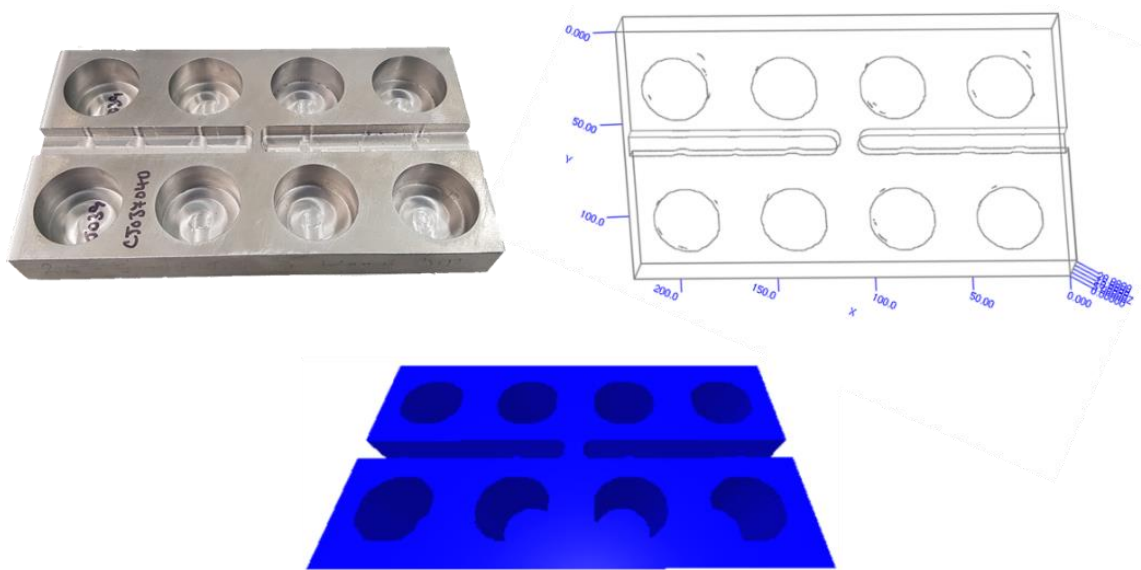


Figure 55 Actual part. Bottom: Web app representation. Right: GUI representation

Before comparing the two representations with the actual part the way that the graphics are generated should be explained. In the local application GUI, the default final part picture is a wireframe showing the edges of the part. Although this is not the most realistic representation of the part it has advantages in terms of practicality. Testing of the simulator was done on an average specification laptop (16GB memory, 2.0GHz dual core processor). Full visual representation of the machined part requires high computing power that slows down the simulation process and consumes a large amount of memory. Since high-resolution graphical representation is not related to simulation performance but only important for demonstration purposes, wireframe part representation made testing faster and

freed memory for the simulation engine to run. These observations were done after initially showing the parts as in Figure 56.

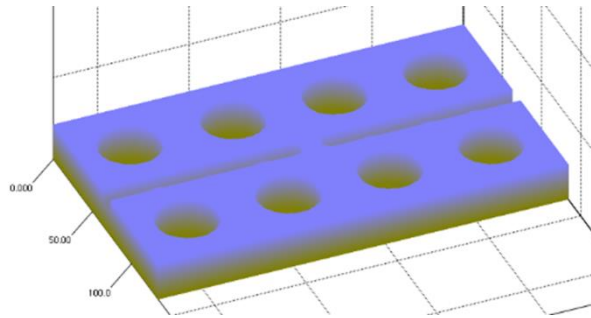


Figure 56 Local GUI alternative representation (showing all non-machined elements)

A second and more important reason for using a wireframe representation is that it provides a clear view of the features of the machined part. In Figure 56 some details of the slots are not visible and therefore viewing from specific angles and different zooms is required to verify that the simulator is replicating the machining of the part correctly and generating the graphics accurately. Regardless of the type of part representation the local GUI produces graphics at the maximum available resolution meaning that the results can be reliably assessed by viewing the part representation.

Going back to the results, the different methods of machine part graphic generation show that the simulator successfully machines the part and is providing the required data for various types of user interfaces. It should be noted that the GUI is not part of the simulator but it relies on the output interface to get all necessary data. Apart from the accurate overview of the part when the details are magnified the limitations of the simulator are becoming visible.

One special point of interest arising from the GUI representation of the part in Figure 55 can be explored in more detail using Figure 57. The two slots milled into the surface of the test piece were produced using eight separate machine operations, with four cuts forming each slot. These were enacted following the machining of each cylinder and were intended to provide a direct method to measure tool wear. The method of machining meant that the tool exit and tool entry points did not completely overlap resulting in some residual material being left in

the slots. Figure 57 shows that the simulator has correctly identified the 'anomalies' of the slot but it is not entirely accurate in their representation.

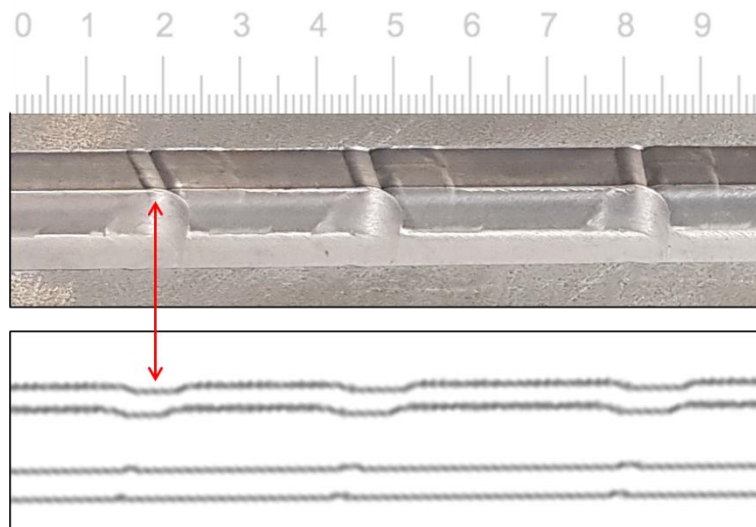


Figure 57 Details comparison between actual part (top) and simulated part (bottom).

During simulator validation, there were two possible causes for this type of inaccuracy identified. The most obvious is the mesh element size. If the detail or feature of the actual part is of similar size to the mesh element size, then pixelation of the detail is observed in the representation. This causes effects similar to the ones shown in Figure 57. If the detail is much smaller than the mesh element size, then the detail may not show at all in the simulated part. However, to generate the above graphs the simulator is using a 0.3mm mesh element size which should theoretically produce better results.

A deeper investigation showed that the root of the problem is the accuracy of cutting tool coordinates that are provided by the monitoring system. The example is run based on data generated by the monitoring system and the precision of the cutting tool position reported by the monitoring system is 1mm. This is not the precision of the machining operation but it is the value reported by the monitoring system. The simulator improves the accuracy through the smoothing of the monitoring data to ensure that issues like the one depicted in Figure 31 are mitigated. However, smoothing cannot improve the accuracy of input data and therefore the virtual cutting process of the above example has a maximum accuracy of $\pm 0.5\text{mm}$ despite the finer mesh. Such issues can be avoided if the G-Code file is supplied to the simulator (instead of monitoring data) but the specific

VMC uses custom G-Code commands and therefore it is not fully compatible with the library that the simulator supports (GitHub - GRBL. 2016).

A comparison between the web application and the actual part shows that the web application provides a good representation of the part which is being virtually machined in real time in the user's internet browser. The strength of the web application in terms of real-time access to the live simulation is at the same time its weakness in terms of precision. The quality of the graphical representation depends on the computing capacity of the client computer and therefore it can be much lower compared to the quality of the representation produced by the local GUI. In Figure 58 the mesh element size of the virtual part in the user's browser is 1mm while the API provided data from a mesh with a 0.3mm element size.

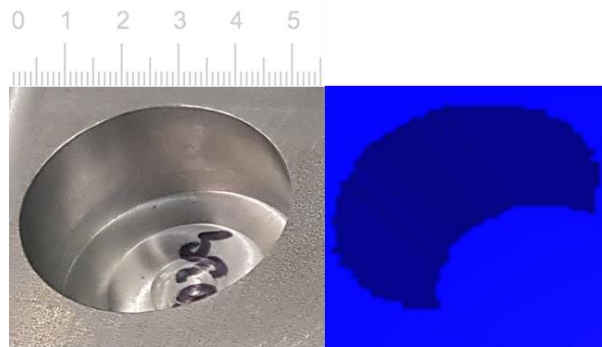


Figure 58 Actual part feature (left) versus the representation of the feature by the web application (right).

After ensuring that the simulator machines the parts as expected its application to identify the effects of tool wear on the cutting tool were investigated. The simulator calculates the number of billet elements that each cutting tool element removes and categorises tool usage in radial and axial directions. The former is given by the number of billet elements that the cutting tool's bottom face mesh removes, and the latter is the billet elements removed by the tool's periphery (shown in Figure 41). As there is no mechanism to monitor how the cutting tool usage is distributed on the actual cutting tool an indirect way of validating the results has been selected. Cutting tool usage is related to the tool wear but parameters such as spindle speed, feed rate or presence of coolant also affect the cutting tool's lifespan. For the purpose of this work, the examined monitoring data was selected from data generated by running the same part program for several sets of tests.

These each used a single cutting tool of the same type from a new condition to failure. Making use of a number of similar cutting tools under the exact same cutting conditions and with the same machining parameters had the effect of making tool usage the dominant factor for tool wear. The data has been generated during the work of Hill et al. (2019) who machined multiple billets to the shape shown in Figure 55.

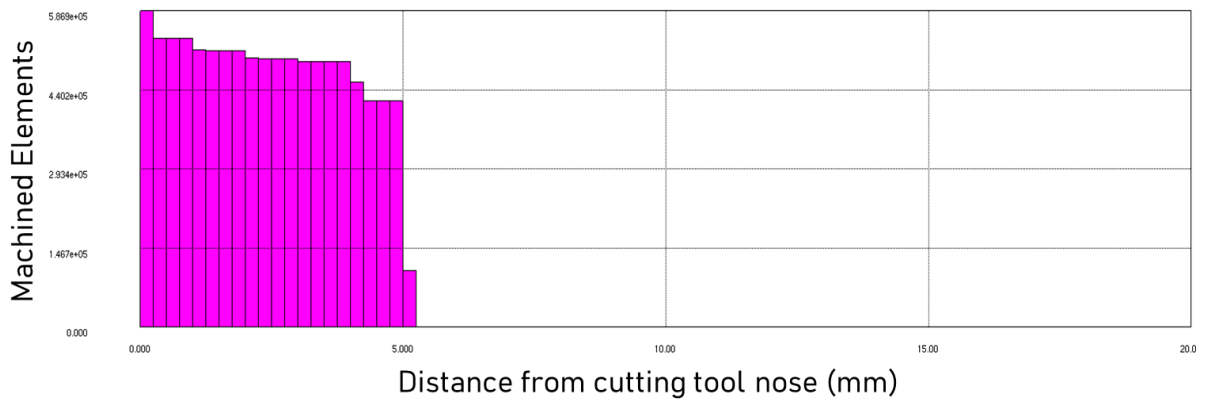


Figure 59 Axial cutting tool usage simulation results.

Figure 59 shows the number of billet elements machined by each element of the cutting tool periphery. To create the diagram a simulation was run with the following setup parameters:

- Mesh element size: 0.25mm
- Cutting tool: End mill of 10mm diameter and 20mm working length.

Each bar is a cutting tool element and the first bar from the left is the tip of the tool which is shown in Figure 41. The bar chart shows that only 5mm from the bottom of the tool has been used which is consistent with the machining process during which it machines the cylinders of Figure 55 by creating stacked cylinders of 5mm depth. An exception is the bar after 5mm which is suspected to be material left at the bottom of a cylinder and removed as soon as the tool moves ‘deeper’ and can therefore reach all elements at the cylinder corners.



Figure 60 Radial cutting tool usage, simulation results.

Radial cutting tool usage has been calculated by the same simulator run that generated the axial usage chart. Figure 60 bar chart shows the number of machined elements that have been machined by each element of the cutting tool's bottom face. Since the tool's bottom face is symmetrical, Figure 60 shows only its radius starting from the centre (rotational axis) on the left and finishing at the edge/tip of the tool on the right. Since the element size is 0.25mm and the tool diameter is 10mm the above figure contains 20 bars one for each radial element.

Figure 60 shows that the longer the distance from the tool's centre the more billet elements that the tool element machines. This is verified by the fact that the tool's bottom face is flat and therefore used only when the cutting tool is plunging into the billet. When the tool comes into contact with the billet the elements closer to its tip machine a significantly larger surface compared to the ones at the centre (Figure 61).

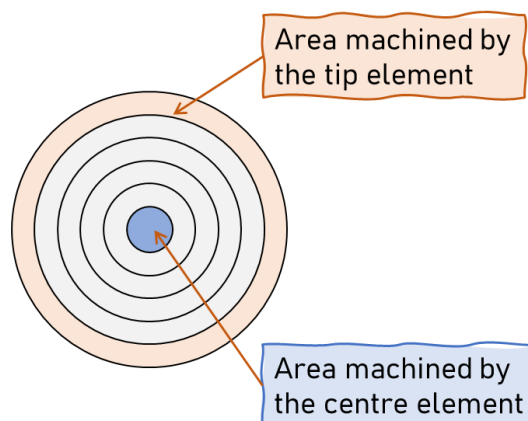


Figure 61 Machining area of cutting tool bottom face elements.

To validate the results the actual cutting tools used for the process were inspected. Two tools, one with minor wear and one with excessive wear, were

selected to identify both wear areas and wear progression. Figure 62 shows the cutting tools that were examined.



Figure 62 Cutting tools: Partially worn (top left and right) and catastrophically failed (bottom left).

The simulation results have been overlaid to the above pictures at scale to facilitate the comparison. Beginning from the axial wear it can be observed that indeed the area that the tool has been used the most and has worn accordingly is the first 5mm from the tool's tip. This validates the results that estimated usage in this area. In addition, the excessive wear of the 'corners' of the bottom left tool validates that the usage around the tip is higher than the wear further away from the bottom face. Equally accurate is the estimation that the radial usage is low at the centre of the tool's bottom face and much higher at the edge. Figure 62 radial profile picture shows excessive tool wear towards the edges which validates the simulator's results. However, close observation of the radial wear shows that the wear is not even as one would assume from the simulator results. The reason behind this is the modelling of the cutting tool. The simulator cutting tool elements have the shape of rings and they don't have teeth or flutes on them. The actual tool removes the billet material with the teeth and extracts the swarf through the flutes which consequently introduces points of concentrated forces. It is therefore a matter of modelling strategy and simulation targets whether these details are meant to be predicted or not. In case these details are important then as part of a future research activity the cutting tool's model would resemble the model of the billet meaning a representation with a 3-dimensional mesh of cubes.

The importance of this feature to the potential management of tool wear relates to the nature of the wear process and how it is currently identified. The tool condition of the partially worn tool shown in Figure 62 cannot be easily assessed with the cutter in the machine. Indeed, the second cutter has undergone a catastrophic failure partly because its condition could not be identified. It can be mentioned that this failure occurred despite the spindle load monitoring performed by the VMC controller and required the intervention of the operator to prevent further damage to both the workpiece and the machine. As more information is acquired and embedded within the simulator such events may be identified and thus their effect mitigated. This has the potential to be applied to cutting tools being utilised in real-life applications as opposed to the laboratory-based experiments depicted here.

The final and most quantified validation of the simulation accuracy is the comparison between the calculated results for the metal removal processes and the actual monitoring data for the same operations. Spindle load has been selected as the parameter to examine accuracy because it has an ideal combination of characteristics. It is not directly related to any of the individual parameters that can be extracted by the part program, but it is related to the material removal rate. This has been shown by the similarity of the curves shown in Figure 49. Although a theoretical approach could explain the reasons behind this relationship this work focuses on the fact that the simulator is not aware of the relationship but searches for models that reliably produce low error results. Spindle load is provided by the monitoring system in the form of a percentage of a reference value. It should be noted that the simulator calculates spindle load by using the formulas of a ML model which means that the units of the parameter do not matter provided that they are consistent. Figure 63 shows the initial validation of the simulator-generated model.

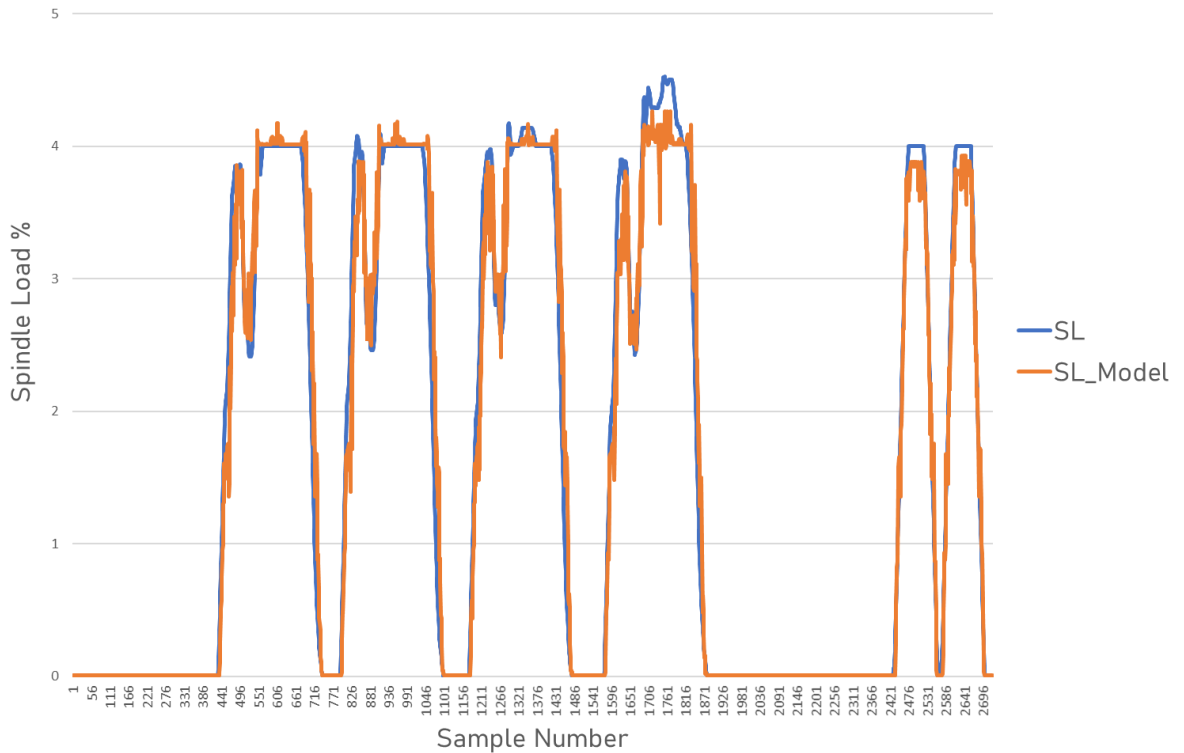


Figure 63 Calculation model performance validation against training set.

The data shown in Figure 63 is a small part of a VMC run depicting the machining of one cylinder followed by the machining of a groove. The data from the full run was supplied to Auto-WEKA in order to find the best ML model. The simulator's learning module then trained the selected model. The two curves depicted in Figure 63 are the actual spindle load (SL) data and the validation of the calculation model by the same dataset (SL Model). As it can be observed the prediction is very accurate. The mean square error between the prediction of spindle load and the actual values is 0.061 which can be converted to a mean absolute error of 0.247. It should be noted that the precision of the spindle load readings as they come from the VMC monitoring system is 1% which is 4 times higher than the mean absolute error. Monitoring precision is slightly improved by the simulator's pre-processing which smooths the data. For example, it can introduce a value of 4.5% between two samples with SL values of 4% and 5%. However, pre-processing cannot improve the accuracy of the data and therefore the simulator is capable of overcoming issues

that low-resolution readings introduce. The application of the same calculation model to different runs is presented in the following section.

5.2 KNOWLEDGE GENERATION

The first run of the simulator starts without any previous data to train the learning module models with; this is also the case every time the VMC runs with a new cutting tool and billet material combination. To deal with the lack of data the simulator performs its initial calculations with models based on linear regression which can produce results with only a few samples available. Then, as soon as enough data has been generated by the monitoring system, it switches to better trained and more accurate models the type of which is determined by running the Auto-WEKA model search algorithm.

For the case of spindle load in the cylinders machining scenario the simple linear regression model gave good results, considering the zero experience of the learning module. However, as can be observed in Figure 64 these results were not reliable. Figure 64 shows a part of the full run where the spindle load has been calculated with the simple linear regression model. For demonstration purposes, the calculation model has been applied to the full run although the simulator would have changed to a different model during the process and substituted all spindle load values with more accurate ones. It should also be noted that the figures in this section have been produced with Microsoft Excel from the simulator's output CSV files since the author has not implemented the zoom functionality in the simulator's GUI.

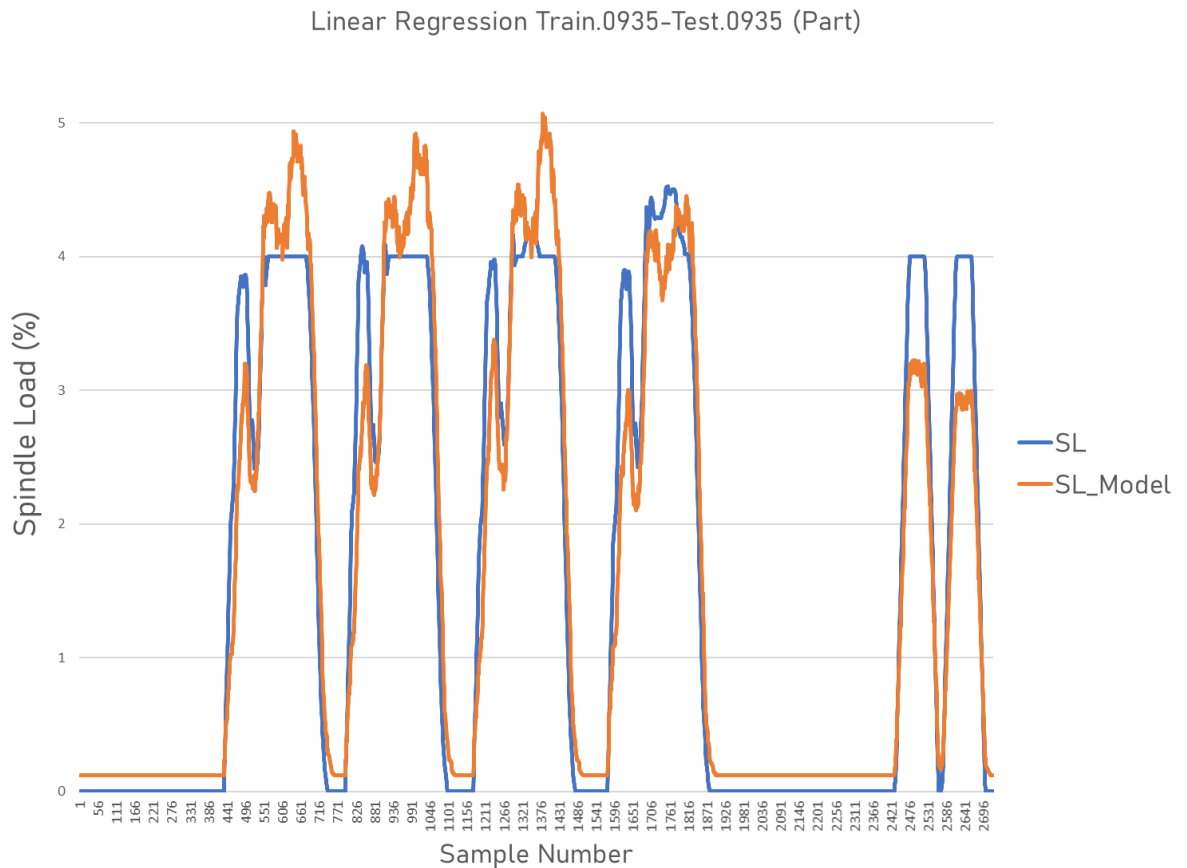


Figure 64 Spindle load calculation by the initial linear regression model

The simple linear regression model produces a mean square error of 0.139 and a mean absolute error of 0.372. The error is still low considering the precision of the monitoring system (± 0.5) but much higher compared to the mature model selected by Auto-WEKA and presented in the previous section (Figure 63). In addition, the error is not equally distributed. In Figure 64, it is clear that there are regions where spindle load is estimated accurately while there are other regions where the difference is in the order of 1%. This last observation is the reason why simple linear regression is not considered reliable.

In general, the accuracy of the simple linear regression model depends on the estimated parameter. Similarly, the number of samples needed to identify and train a more accurate model also depends on the parameter values, generated data noise and above all the strength of the relationship between the output parameter and the provided input parameters. In the examined case, the samples depicted in Figure 64 (about 2500 samples) were enough to identify a better model that could

make accurate estimations outside the dataset that it was trained with. The required number of samples was gathered in about 200 seconds from the moment that the cutting tool was loaded to the spindle of the VMC.

For the selection of a better ML model, AutoWEKA is supplied with the dataset containing the input and target parameter values. As previously the selected input was material removal rate and the target parameter was spindle load. AutoWEKA selected a bootstrap aggregation (bagging) algorithm (Breiman 1996) and the simulator applied it with the WEKA's default configuration that uses Reduced Error Pruning (Quinlan 1987) to build the decision/regression tree. The details of the model are selected automatically and therefore the suitability of the model is not examined separately but only through the actual results that the simulator produces. In general ML models perform well in specific types of datasets and bagging is suitable for the smoothed VMC dataset. This however does not assume that the simulator estimations are a priori correct since validation is done in the long term by an ongoing process of best model search and continuous model retraining every time that new data is generated by the monitoring system.

The performance of the simulator when the model has been trained with sufficient data has already been presented in Figure 63 along with the mean error of the estimated values. For process setups that the cutting tool is relatively new and the VMC runs with the recommended (by the machine OEM) settings the model's performance is similar to the one already presented. It was however essential to test the simulator with data that is generated with different cutting tool states or machine setups. Two different cases were therefore examined; process runs with variable cutting tool wear and process runs with VMC cutting tool protection turned off.

To test the simulator's performance versus cutting tool wear state, generated data from machining six billets with eight cylinders on each billet with the same cutting tool was used. Figure 65 shows how the actual spindle load value deviated from the estimated value as the cutting tool was getting worn.

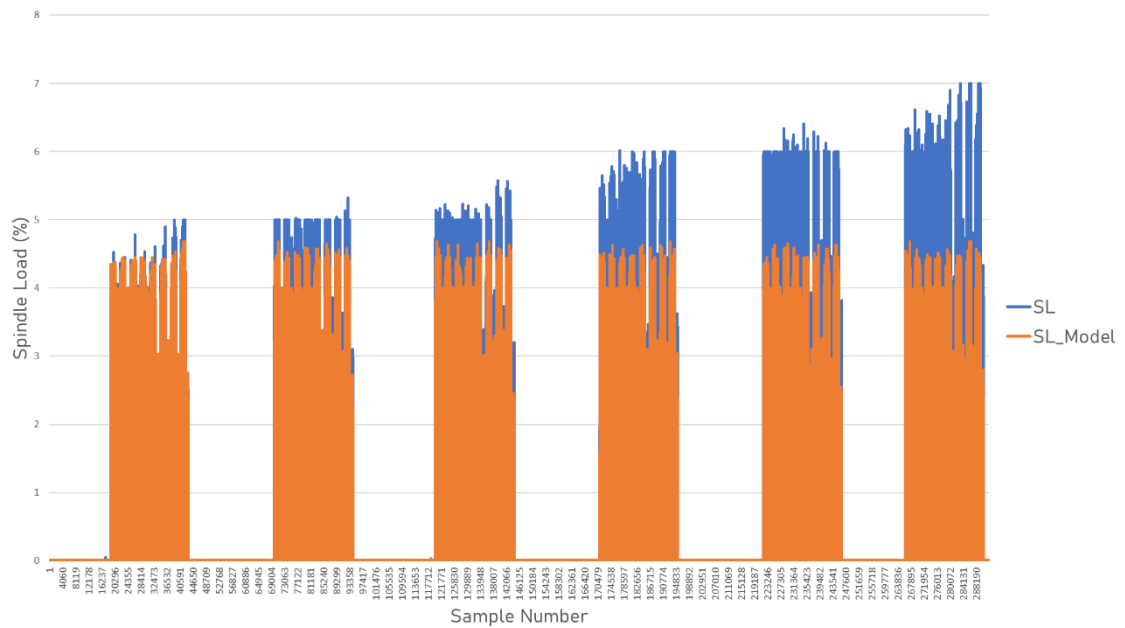


Figure 65 Estimated and actual spindle load for multiple runs with the same cutting tool

The results verify Hill (2020) findings about the relation between spindle load and tool wear but the simulator goes a step further. The work by Hill (2020) uses monitoring system-generated data of two identical runs and measures the spindle load difference when the cutting tool machines at the exact same coordinates. The simulator creates a calculation model from a small part of the generated data and then it can estimate the spindle load regardless of the cutting tool position or the part shape/feature that is being machined. This also enables the quantification of cutting tool wear since it can be measured by the difference between the estimated spindle load and the actual spindle load. Consequently, the simulator can use tool usage as input (this is already recorded) and the difference in spindle load as the target to create and train a new model that estimates tool wear in current and future runs.

The above example assumes that the VMC always runs with the same settings that create a predictable machine behaviour. However, for various reasons such as operator inputs, or the controller-generated modification of settings the VMC may change the way it machines the part. To test this case, datasets from running the VMC without the tool protection mechanism were obtained. The simulator estimation and the actual value of the spindle load are shown in Figure 66.

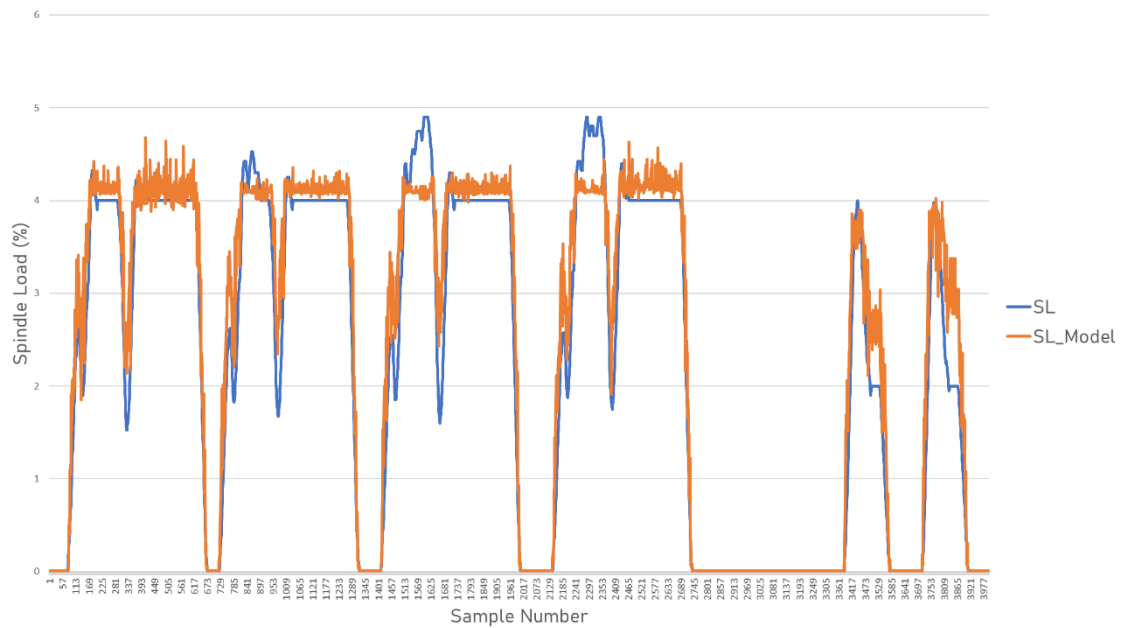


Figure 66 Estimated and actual spindle load for a run without cutting tool protection

Figure 66 shows the first part of a run where the cutting tool has the same wear as the cutting tool of the run that was used to train the spindle load calculation model. As expected, the spindle load is estimated correctly but after the second cylinder is machined a spike can be observed at a specific point of the actual spindle load curve. The difference rapidly increased and the tool broke while machining the 4th billet. For comparison, running the process with recommended settings allowed for machining six billets without the tool reaching the end of its life.

Up to this point, the simulator's knowledge generation has been examined only for one-sample-based learning. Although a similar process is followed for n-sample-based learning the additional requirement in that case is that the datasets need to be synchronised. The learning module uses dynamic time warping to synchronise the data of two runs and examine the differences and/or the behaviour of all other parameters. By default, the synchronisation is based on the coordinates of the cutting tool however the developed simulator can use any parameter to base the synchronisation on. To demonstrate the usefulness and effectiveness of the simulator's synchronisation capability an example is presented in Figure 67.

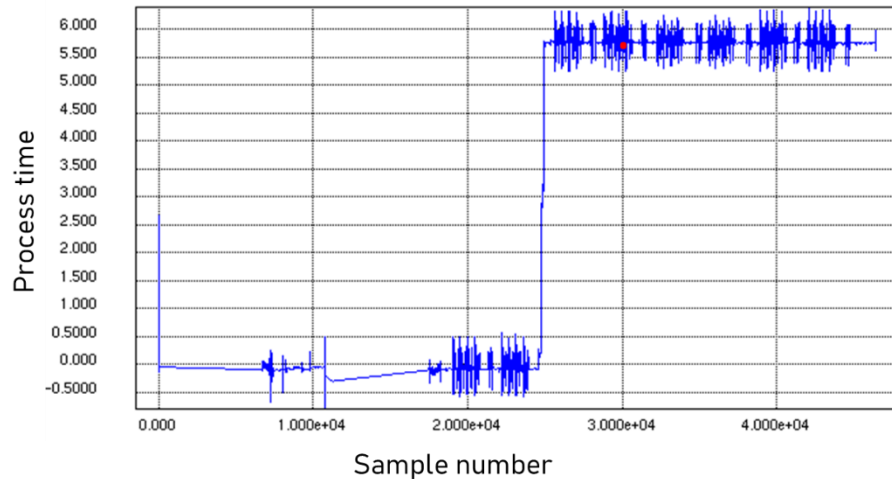


Figure 67 Process time differences between 2 runs

Figure 67 shows a comparison of process time between 2 runs with the exact same VMC setup. The two runs that generated the data are named A0935 and A1222. The red point on the curve of Figure 67 which has sample number: 30000 and process time: 5.7, means that the cutting tool of A0935 reached the coordinates indicated in sample 30000 5.7 seconds later compared to A1222. To investigate the reasons behind the difference between two otherwise identical runs, the user needs to check sample 30000 of A0935. It should be noted that the corresponding sample number of A1222 is different because the two datasets have a different number of samples.

The author implemented an additional tool in the simulator's GUI that shows the tool path and that can filter out sample ranges to enable the investigation of specific samples. Figure 68 shows how from the tool path sample 30000 is identified.

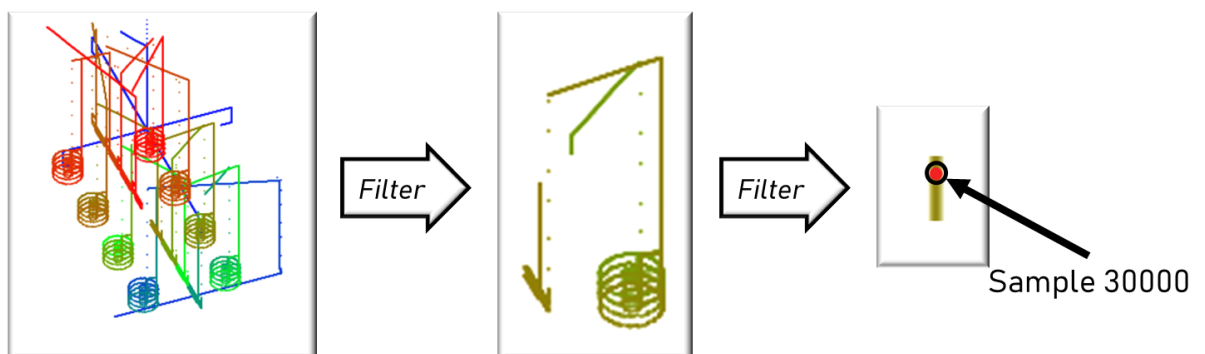


Figure 68 Process investigation: From process cutting tool path to a specific sample point

For completeness, the colours on the toolpath indicate the process time with blue points being the region that was machined first and red points indicating the regions machined last. The tool can be used to visualise any parameter whose value is shown with colour coding (blue for low value and red for high value) on the tool path.

The investigation shows that after machining the 4th cylinder in this billet the process was briefly paused without changes on the programmed cutting tool path. Further examination of all parameters would be done but the intention is to show the results of synchronisation and how this capability enables process data analysis. The example is comparing two process runs but an offline investigation could compare simulator estimations and actual run data to show the deviation of the actual process from the planned one.

5.3 SIMULATOR SPEED

It is critical for a simulator that supports a real-time system to be able to produce results 'in time'. For the purposes of this work it is assumed that 'in time' means less than the time needed for the actual process to run. The reason behind this assumption is that the simulator receives the latest monitoring values, processes them, and calculates results that are then used by the machine's control system to evaluate whether the process is running as expected. Therefore, these results need to be available before the next sample becomes available and cumulatively, the simulator should run the virtual process faster than the physical process.

Execution speed is directly related to the computing power of the system that the simulator runs on. As explained in chapter 3, in order to be practically usable in the future, the simulator should ideally be run on the computer embedded into the machine itself. The CNC machine embedded computers are typically systems with the power of a personal computer. Given a fixed maximum execution time and predefined computing power the only parameter that should be examined in relation to execution speed is the mesh element size.

To test the VMC simulator execution speed a number of tests were run on a personal computer with 16GB of RAM and a 2.0GHz dual core CPU. Figure 69 shows the execution time results.

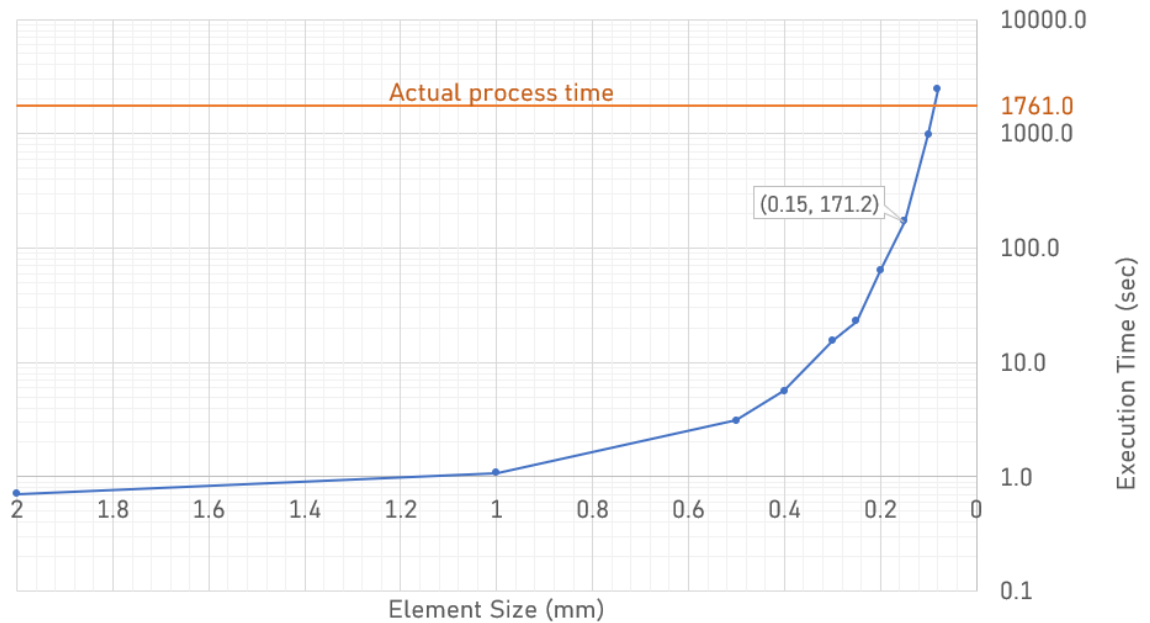


Figure 69 Execution time for various mesh element sizes

It can be observed in Figure 69 that execution time increases exponentially as the mesh element size decreases. The diagram also shows the actual process time (orange line) which leads to the conclusion that with the specific computing resources the smallest element size for real-time results is 0.1mm.

Due to the way that the virtual machining process is designed to run, execution time is not directly proportional to the number of samples produced by the monitoring system. To demonstrate this, the time required to process each sample produced by the monitoring system was recorded and the cumulative graph presented in Figure 70 was created. The graph does not show duplicate monitoring samples that the simulator's input interface removes but on the other hand it includes the additional samples generated by the simulator in order to smooth the data.

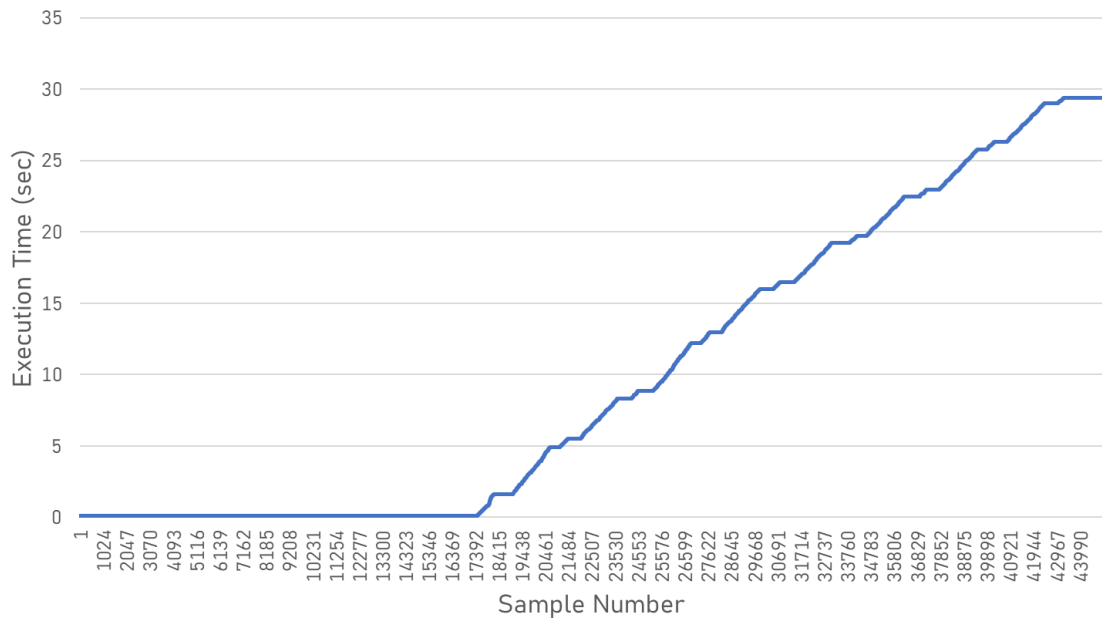


Figure 70 Execution time profiling

Figure 70 shows that it took close to zero time to calculate the results for the first 17390 samples and then the time needed per sample varies. The reason behind this extreme variation between processing times is that it takes close to zero time for the simulator to determine if the cutting tool is machining the billet. At the beginning of the process, the VMC performs various checks without machining and therefore the samples are quickly processed. When the VMC machines the billet then the simulator has to calculate the mesh elements that are machined and use the results to calculate all derived data that would otherwise get default values. As a result, and for the specific dataset, it would be safe to run the simulator with a mesh element size of 0.15mm but not 0.10mm. The mesh size setting is normally tuned after getting the first set of data and can remain constant for as long as the monitoring data capturing rate remains the same.

A workaround to speed up offline simulations or to decrease mesh size more than what the memory of the computer can support is to split the billet into multiple pieces that are simulated separately. The simulation then can be run either sequentially for each billet piece or in parallel at multiple computers. It took 171.2sec to run the simulation for a mesh size of 0.15mm and when the billet was split into 4 parts of the same mesh size it took between 47.7 and 49.3 sec to simulate each part. If the workaround aims to reduce the mesh size then the

equivalent mesh size if the billet is sliced in 4 pieces is 0.0945mm. At the end of the process the results of the simulation of each piece need to be combined which for the specific simulator means that the number of machined elements from each piece should be added to calculate the total material removed. It should be noted that this workaround works best for offline simulation. During online simulation, the latest sample that arrives at the simulator will typically fall into the volume of one piece. Therefore, if four nodes are running the simulation in parallel only one node will be machining the virtual part while the other three will be idle. This however is a limitation of the specific implementation and more advanced parallelisation techniques could be applied.

All simulation speed tests measured the performance of the simulation engine. The learning module models were already in the database and did not consume any computing resources or time to be created/trained. As explained in section 4.5 the ML model creation and training process is done in parallel to the simulation engine operation and it does not need to comply with the real-time restrictions. Contrary to the simulation engine calculation time requirements, the time to find an accurate calculation model cannot be estimated or predicted. Each dataset may have different levels of noise or contain data that is captured during extreme events which make the model-finding process more difficult. For completeness but without these numbers being indicative of the expected performance, AutoWEKA found the first model that was significantly better than the default simple linear regression model after 813 seconds. The selected ML model was a random tree forest algorithm that had a mean error of 0.127, roughly half the error of the linear regression model and double the error of the final bagging algorithm. The selected algorithm (bagging) was found after 47,268 seconds.

Model finding times verify the need to run the learning module asynchronously but also point to the direction of using cloud computing resources. For a specific cutting tool and billet material combination, the model will normally be sought once. Then, the same ML model will be retrained (if needed) and generally be reused by the simulator until the actual machine behaves differently than originally designed due to wear or due to damage. This enables the usage of external systems

that do not have to react in real time and more importantly, do not need to be reserved only for one machine. Model finding can be done by a service that always seeks for the best ML model regardless of the application.

In any case, the success of the simulator is that it needed 13 hours to find the best model (on the computer that the benchmarks were run) which is significantly less time compared to training the operator on the machining of a specific material with a specific type of tool. Moreover, since the database can be shared, the model-finding process has to be done once and then it will be distributed to all machines that are of the same type/model by the company or the machine builder themselves. The overall model management will be further discussed in chapter 0.

5.4 DATA REDUCTION

During the simulation process there are multiple data processing stages where the memory size required to store data changes. First, this happens during pre-processing where the raw data is cleaned and then enhanced until it becomes the data input for the simulation engine. To quantify the size changes, the datasets from cylinder machining were used to calculate the average data sizes presented in Table 14. For reference, the simulator first removes samples from the dataset that are not usable, are duplicates or are part of redundant information and the resulting dataset is the one referred to as 'cleaned'. Then new samples are added to the dataset to produce a smooth toolpath (see section 4.2.3.1). This is referred to as 'smoothed'.

Table 14 Dataset average size during pre-processing

Data state	Size (kb)	Change (%)
Raw	8,507	-
Cleaned	579	-93%
Smoothed	4,867	-43%

Cleaned datasets contain all usable information that can be extracted from raw data. This means that for the specific datasets the immediate gains in terms of storage space are 93%. This performance cannot be generalised because size

reduction purely depends on the dataset. If the raw data is of very high quality, then few (if any) samples would be rejected and therefore the change would be minimal. On the other hand, smoothed data depends on the resolution of the mesh that is built by the simulation engine. For example, if in the raw data the movement of 1mm along the X axis is represented by 100 samples of 0.01mm step each and the simulation engine runs with a mesh size of 0.1mm then the 100 samples of raw data will be converted to 10 samples of 0.1mm step in the smoothed dataset. Therefore, the size of 4,867 kb is for the specific mesh resolution (0.25mm in this case) and would change accordingly if the mesh size would be different. Similarly, the gains in space are relative to the mesh size and 43% gains are only representative of the specific dataset.

The second stage where the memory requirements change is when the simulator captures the knowledge after processing the dataset and stores it in one or more ML models. Due to the way that different learning models work there is a very wide range of model storage requirements. To calculate the model size, five datasets were used to train three different types of models. The datasets were progressively merged so model training was initially done with dataset 1 then with datasets 1 and 2 until the final training that was done with all five datasets merged into one. Figure 71 shows the storage space required for the training data and each of the models after training.

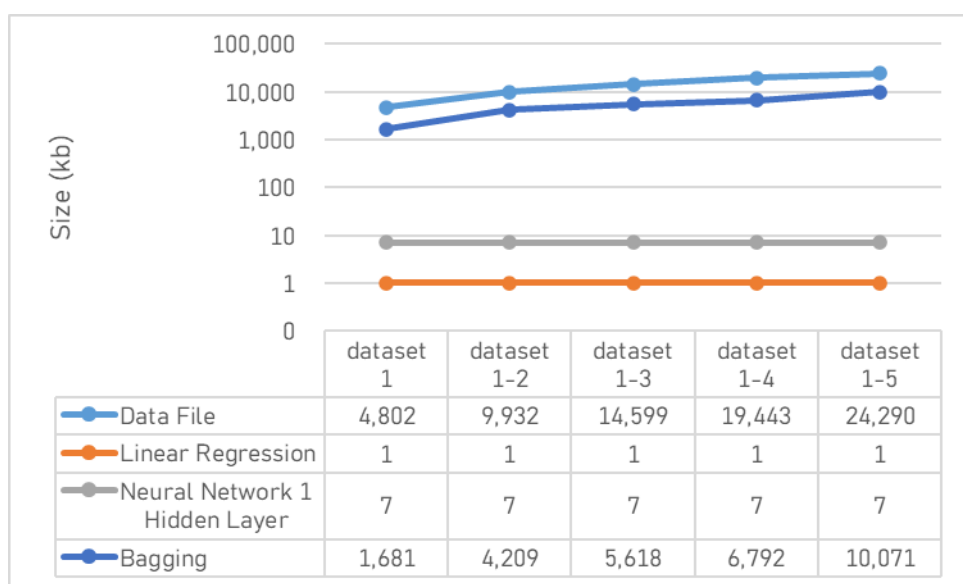


Figure 71 Memory size comparison of different ML models

It is clear from the chart that there are ML models with static requirements and others that grow dynamically. A brief explanation of Figure 71 is that both linear regression and neural networks have a fixed structure that adapts to the training dataset while bagging creates its structure based on the dataset. It should be noted that the spindle load prediction model requires only a fraction of the dataset to reach an acceptable level of accuracy. The minimum trained model size is 288kb and there is no need to retrain it with more data that will increase its size. The drawback is that the above size was achieved through manual training of the model and the simulator version that has been implemented for this work does not support a minimum required training identification algorithm.

The above experiments quantified the data reduction based on the assumption that only one model will be created by the datasets. In practice, multiple models will be required since the simulator will have more than one parameter connected to a model in the database. Depending on the model type the total memory required to store all models may be bigger than the size of the training dataset. In the cylinder example that would occur if the bagging algorithm would be used for the prediction of at least 5 parameters and if the training process would not be stopped by the user at the point where the models produce satisfactory results.

Overall, in the vast majority of cases, the simulator can be used as a way to convert the raw data into a form that is instantly usable through the simulation engine and that is occupying much less memory space.

5.5 DATA UTILISATION

An important aspect of this work is to ensure that all available data is utilised. Due to the way that the simulator works, all raw data is processed and therefore by being directly connected to the monitoring system ensures that no data is being wasted. Although this objective is achieved the objective itself is based on the assumption that data is fully utilised if all of it is processed and used by the simulator. In fact, there is no metric showing if all value contained in a dataset is extracted or how much value a dataset has.

When the value of information is assessed a parameter that is common in business management is the usable life of datum. In the case of the developed simulator, the initial dataset is relevant to a specific process run and even after pre-processing its usability is restricted to the analysis of that one run. The simulator converts the data to models that are applicable and valid for as long as the simulated machine or asset remains unmodified and reasonably maintained. Specifically for the material removal rate and spindle load combination which has been examined, the generated models are applicable every time that the same type of cutting tool machines the specific billet material. This expands the usable life of datum far beyond one run and potentially up to the life of the VMC.

The majority of metrics when data processing projects are considered aim at converting the value of data to profit. There are two hierarchical levels where the simulator contributes towards that direction. At a shopfloor level, knowing what the expected values of a process parameter should be can prevent part rejections (from incorrect machining) or damages from broken tools and/or excessive forces applied to the machine. Figure 65 and Figure 66 are indicative of the capability to identify long-term trends and spot issues as early as possible. It is important to state that this information relates to the actual work done by the cutting tool as output by the simulator. This is a better indicator of the condition of the tool than the often applied measures such as standard cutting times that are currently used. Such measures rely upon empirical conversions of the possible cutting and material combinations into a time that represents the time used for an operation that can be deducted from the total life of the cutter. At a company level, the created calculation models are stored in the database and over time the value of the database itself increases because it contains the experience of using a specific machine with different setups. Especially in the case of machine OEMs, the database becomes an independent product that can be sold as an optional addition to the physical machine. Conversely, the user of the machine can sell the database to any other user or to the machine builder at a price relevant to the maturity of the database on specific tasks. To sum up, the simulator architecture and its specific application in Chapter 4 not only extracts value from each dataset but it also creates a product on top of the day-to-day uses of the database knowledge itself.

6 PROJECT DISCUSSION AND NEXT STEPS

This work proposes a novel simulator architecture that can be embedded in digital twins or more generally in digital systems that supervise or manage physical processes or assets. The architecture is designed to fill current technology gaps and provide through a proof of concept an example of how to overcome challenges of modern digitalisation. It pushes the simulation to a new generation where the simulator development is more systematic and the simulation operations are distributed to different computing systems by breaking down the simulator into independent subsystems with clear responsibilities and contributions. The learning aspect of a simulator is treated as an essential part of a modern implementation which enables the adaptation of calculation results to each application scenario without changing the simulation model. The simulator as a whole although able to “live” in a distributed geographical environment and share its knowledge and subsystems with other simulators has clear boundaries and its core purpose to calculate required results from supplied input data remains the same. The development of a VMC simulator based on the proposed architecture is a live example of the potential benefits and a measure of performance levels that the simulator can achieve.

Revisiting the title of this report can be a starting point for defining the fields that this work contributes to. Process design is traditionally related to the proper sequencing of physical resources that at the end create a product or deliver a service. Nowadays, physical assets have digital counterparts therefore process design also includes the digital ecosystem that supports the process. This is where the current work blends with the existing practices since simulators are becoming an essential part of the design. One or more simulators run in parallel to the physical assets to ensure that the process runs as designed and prevent quality discrepancies that would lead to process failures. As a result, supervision is another field of contribution.

The backbone of digitalisation is data, and a simulator is the entity that calculates the expected values of key process parameters and can determine in a quantified

way how far from ideal is the process running. Supervision is therefore a natural field of application for a simulator and moving forward with fully digitalised production machines it is clear that the human operator who is now using raw data and their senses to supervise the process will simply use the results of the real-time simulation to check in depth multiple processes at a time. In combination with the capacity of digital twins to combine the power of simulation with decision algorithms and data visualisation technologies it is possible to see how future autonomous systems will gradually take over supervision tasks.

Like simulation, digitalisation is a very broad field. If the focus is put on current trends in manufacturing, the proposed architecture is the foundation for simulation systems that can be embedded in a machine. Without changing their digital structure, different parts of the system can run on different computers or on the cloud. For example, a simulation system can be distributed as follows:

- A milling machine has embedded in its controller only the simulation engine and the input/output interfaces. The setup calculates the expected parameter values in real time and assists the machine operator in process supervision. In future systems under the actions of the OEM it can be integrated with the machine control to provide a level of autonomy.
- A local high-specification computer is running the learning module. It is processing the data that is generated by all physical systems on the shop floor and selects and trains the best-performing ML calculation models. The same computer can be used for other machine learning tasks and support multiple simulators.
- The database where the ML models are stored is running on a cloud server and the milling machines at another manufacturing site have access to the same calculation models. This allows all machines, regardless of their geographical location to build upon and share the same knowledge and in addition to apply this knowledge even if they don't run the same part program.

Each module of the simulator can be developed independently which allows teams from different fields of expertise to collaborate without any interference since the

communication protocols are defined by the architecture of the simulator. Provided that the communication protocols are respected each team is free to develop any solution that is most appropriate to the specific application scenario.

Regarding CNC processes this work proposes a more specific operation for each of the simulator modules. The novelty in the CNC simulation is the introduction of a mesh-based method that the simulation engine is using combined with the automatic selection of the best-performing calculation models done by the learning module. This combination enables an iterative process where the virtual machining of a part feeds the automatic creation of calculation models. These models are then fed back to the simulation engine to improve its accuracy and enhance its results. Enhancement includes the calculation of previously unknown parameters for which the physical machine cannot generate data.

The proof-of-concept demonstrator for vertical milling introduced novelties that separate the simulator implementation from other implementations reported in the literature. The different approach in building the mesh for the cutting tool and the billet makes the simulation process efficient and very flexible in terms of adapting to available computational resources and to the required precision. Then, the 2 stages of process parameter calculation enable a precise digital replication of the process and at the same time it facilitates the usage of all existing knowledge either from milling process theory or machine learning techniques. Finally, the adaptation of DTW to synchronise data from different runs based on tool position is another novelty which can work as part of the learning module or even independently where 2 runs can be compared regardless of the feed rates.

All previous points are about novel characteristics of the simulator modules which are part of a system that, like other modular systems, receives input data through common data transmission protocols and makes the results available through common data distribution technologies. As a result, the system can use novel ways to operate but it is still easy to embed into parent systems such as digital twins without the need for specialised protocols or equipment. A difference with common practices is that the simulator does not define the GUI as part of the simulator. The development of a GUI is neither restricted nor discouraged and for the purposes of

this research work the author developed 2 separate GUIs (web-based and local application). The GUI however is intentionally separated from the architecture because the simulator is aimed towards machine-to-machine applications while GUIs are essential only for humans.

Overall, the proposed architecture and its applications in CNC machining promote a modern digital environment where all available data is processed, and the generated knowledge is shared with systems at different locations in real time. The architecture goes a step further by enabling the monetisation of the generated knowledge since the database is a separate entity that can be 'plugged in' to other simulators of the same machine type. Knowledge sharing boosts shopfloor autonomy and at the same time it quantifies the differences between 2 theoretically identical machines.

The applicability of this research work on a very wide range of cases increases its potential impact but at the same time it increases the requirements for further research that will apply the architecture in different fields. Since the author developed both the methods described in this work and the software that demonstrates them it is natural that there are implementation weaknesses and missed opportunities to embed additional novel techniques. More specifically, the author considers the following list as a starting point for multiple projects that would improve the next generation simulator and its applications.

- Modern mesh-based simulations vary the mesh element shape and size in regions of the part that are of higher interest and are generally handling computer memory much more efficiently. Due to the differences between CNC processes, the mesh characteristics should be explored and separately specified for each process and then the new simulation model to be tested against currently available systems.
- The field of automatic identification of the best ML algorithm is rapidly evolving and during the time that this research work was undertaken new cloud services were introduced that began as simple neural network training tools and are currently full ML automation suites. The capacity of the cloud would inevitably accelerate the knowledge generation process and above all it would provide

better automatic model creation algorithms since the AutoWEKA module is of limited model search capacity.

- It is clear that the simulator is envisaged to be part of a digital environment. To understand how future digital ecosystems should be there are three additional investigations needed. At the machine level, the simulator should be run with a variety of different cutting tool types, part materials and process setups. That would be ideally done in collaboration with a machine OEM that will train the simulator as part of their testing process. Then at the production line level, the simulator copies should be used for the same type of machines from different manufacturers and with different ages/histories. After initial training, the database would be compared to identify the difference in behaviour that is now done only by experienced operators. Finally, at the shop floor level, multiple simulators should be developed for different types of machines and embed them to a digital twin that will manage the generated knowledge and provide a service point for all personnel as well as a contact point for other shop floors which want to replicate the production of a product.

As a final point, every research work has gaps and weaknesses that enable other researchers to investigate further and find better solutions. A prerequisite for this is that research outputs are usable, and it is the belief of the author that both the developed VMC simulator and the proposed architecture can be very useful in future projects. The former to any project related to 3-axis milling process and the latter to potentially every development of a simulator even if it is outside the discussed fields of application.

7 CONCLUSION

Digitalisation has evolved to become an essential part of the supervision and management of assets. The digital systems that support digitalisation can become highly complex and therefore their development requires a modular approach where each element can be studied and developed separately. Simulation is a critical part of these systems and especially in the case of digital twins, it is the key subsystem that converts raw data to usable information and retainable knowledge. The architecture proposed in this work further breaks down the simulator development process into modules with specific roles and functionalities. Each module has been described in detail both from a theoretical perspective (chapter 3) and through a practical application in vertical milling (chapter 4).

At the beginning of this work, specific goals were set based on the issues that current systems face. These goals are presented in section 1.1 as a list of challenges which is reasonable to use as a measure of the success of this research project.

Answer to challenge 1 (*simulator modularity and applicability range*):

The architecture as presented in Chapter 3 can be the foundation for a wide range of manufacturing process simulation systems running both offline and in real time. Due to its flexibility in adapting to the needs of each application, there are no set limits to its applicability. Both the simulator subsystems and the simulator as a whole use common communication interfaces which makes them interact and not interfere with their environment. The system design is fully modular, and its parts are exchangeable with the same part from other simulation systems. A good example is the database that can be copied to another simulator of a similar system type.

Answer to challenge 2 (*simulator adaptability and personalisation*):

One of the proposed simulator's core strengths is its ability to learn from the system that it simulates and change its behaviour accordingly. At the same time if the physical system generates data for a process parameter, then the simulator can

create a supervised machine learning model that estimates the initially unknown parameter. Both attributes turn an initially generic simulator into a fully customized one which is unique in terms of what parameters it can calculate and what models it uses to do the calculations.

Answer to challenge 3 (*simulation speed*):

In section 5.3 the VMC simulator's calculation time is measured against the physical process time. The results presented in Figure 69 show that with low-specification computing hardware (relative to currently available computing systems) the simulator could easily reach a 0.15mm precision (and potentially values up to 0.1mm). It should be noted that this level of precision is close to standard tolerances (typically 0.005in – 0.127mm if not otherwise specified). Apart from the specific example which highly depends on the characteristics of the simulation setup and the computing hardware, due to the decentralised nature of the proposed architecture the computing burden can be spread to different systems. It is therefore possible to scale up the capacity of the simulation engine to calculate the results within given time restrictions.

Answer to challenge 4 (*simulation precision and accuracy*):

It depends on the physical process requirements whether a specific simulator setup can produce satisfactory results. In the previous answer, it is explained that the experimental setup used throughout Chapter 5 produces results within the typical tolerance required for such processes. The accuracy of the simulator is also demonstrated in section 5.1 and the results are very close to the actual validation values produced by the VMC monitoring system. Moreover, the simulator can split the simulation into multiple billet parts which can be used in extreme cases when higher resolutions or speeds are required. It is difficult to provide a definite answer for every possible simulation scenario but overall, the simulator can produce satisfactory results for relatively high precision and accuracy standards.

Answer to challenge 5 (*high data utilisation and processing efficiency*): As discussed in 5.5 all data that is generated by the monitoring system is processed. This is done once, and the extracted information is further processed to generate

the knowledge that is stored within the calculation models. Therefore, there is no data being wasted and at the same time there is no need to process the same dataset twice since the value is extracted in one go and stored in a more usable and accessible for the simulator form (the calculation models). It is worth noting that the simulator architecture promotes this behaviour by introducing the combination of a learning module which extracts the value with the database that stores the knowledge for future runs.

Answer to challenge 6 (*reasonable computing power needs*):

This is another case where the physical process requirements define the computational needs. If the example of Chapter 5 is used as a benchmark, then a low-specification laptop can provide the necessary computing power. CNC machine controllers are essentially common desktop computers that run either Windows or Linux operating systems. It is therefore feasible to integrate the simulator into the CNC controller without the need for specialised equipment. In addition, the proposed architecture allows for a distribution of the modules that comprise the simulator making it possible to expand the capabilities of a physical machine simply by connecting it to external computing systems.

Answer to challenge 7 (*knowledge sharing*):

The database of the simulator can be copied and moved to another system independently. Alternatively, a database can be a cloud service that allows all physical assets of the same type to store their knowledge (ML models) in it and make the generated knowledge accessible to every other system which shares the same database. In general, the proposed architecture facilitates module sharing because there is a standard way that each simulator module communicates. This means that there is no restriction on whether a module is dedicated to a specific physical asset or not.

From the above answers it can be concluded that the research project has been successful in delivering solutions to critical issues in the field of digitalisation.

7.1 RESEARCH CONTRIBUTION AND FUTURE WORK OVERVIEW.

It would be an unnecessary repetition to discuss each contribution of this project therefore a list of key findings and technologies that the author developed is shown below.

- + Review of current practices in developing simulators for digital twins.
- + Design of a new modular and flexible architecture to develop simulators.
- + Methodology to approach monitoring data pre-processing.
- + Mesh-based model for subtractive or additive processes.
- + Method to replicate machining in a digital environment.
- + New methodology to synchronise CNC process datasets using DTW.
- + Introduction of learning module structure into simulators.
- + Integration of automated machine learning techniques in simulators.
- + Methodology to retain and share extracted knowledge.
- + Approach to integrate a simulator into its digital environment.

Due to its capability to evolve, the simulator may not only facilitate the supervision of the physical system but also automate the process of identifying unique behaviours by systematically processing all available data. This is a big step towards manufacturing autonomy since the current practice is to develop simulation models after the patterns have been recognised and expand these models manually when new parameters are available. On top of this characteristic, the generated knowledge is sharable and can create a faster learning community of machines that can point operators and researchers to the critical parts of the generated or predicted data for past and future events.

Naturally, the proposed architecture and the suggested implementations in CNC processes were developed within the restrictions of a PhD project. The time and resources that can be reasonably dedicated to a PhD project are not enough to run a full feasibility study that clearly specifies all the fields where the proposed architecture is applicable and meaningful at the same time. In its preliminary stages this project was focusing on CNC processes. It became quickly clear that data processing and simulator evolution was relevant to a much broader field. To embrace the opportunity for a bigger impact the architecture was developed as a

more generic model which was then implemented in steps to assist in reproducibility of the work and ultimately to the understanding of its value. Consequently, due to the generalisation of the architecture it was not possible within the limits of a PhD project to provide evidence for the improvements the architecture brings in the full range of its aimed applications.

A second gap in the reported work is that the limitations in resources did not enable the author to test the simulator in cases where hundreds of parameters have to be simulated in real time. Further work would provide more data about the processing needs after scaling up the simulator in terms of number of simulated parameters and higher calculation speed, especially in distributed computing scenarios.

Finally, the simulator implementation that was used for the needs of this work is not a fully developed version of the simulator. Full integration of AutoWEKA was not possible to the extent that would allow all tasks to run automatically. This is due to the lack of support for integrating the library directly to another Java application. The workaround was to run AutoWEKA and select the best model manually based on AutoWEKA results. In addition, the real-time execution is based on a fictional data stream and there is no implementation to connect the simulator to a machine data interface such as MTCConnect. In practice, the latest version of the simulator is good enough for research purposes, but any production-ready implementation must mitigate this issue first.

In addition to the scope of this work which was to develop and demonstrate the simulator architecture, there were findings that should be noted. As stated in the first chapter of this report and detailed in section 2.3 manufacturing companies should rethink the amount and quality of data they collect and store. It is clear that there is a mentality of saving everything regardless of use but both collection and storage induce costs and vulnerabilities that are not always balanced by the benefits. Another finding is that there is a lack of guidance in the available literature about building new special shape mesh elements. When choosing the element shape for the milling cutting tool the author found a lot of published work

that compares known types of element shapes but little guidance about how to build a new type of element shape that would fit the needs of a simulation task.

It is worth noting again that all the achievements of this work and consequently its contribution have as a starting point the generic design of the architecture which distributes the development work and allows for a more in-depth investigation of each module. This enabled the author and can assist researchers to study and develop each module separately and finally create next generation simulator prototypes that contribute beyond the scope of the initial work. The development approach is effective whether researchers are studying a specific module, single developers are building a simulator for a new process or teams of developers create prototypes under time pressure. It is therefore the intention of the author to provide the foundation for two types of future work regardless of the manpower that will carry it out:

1. Development of simulators for different processes that will gradually build a more knowledgeable digital ecosystem.
2. Exploitation of the capability of the machines to process data to the point that machines and not scientists create the theory that describes the process state and behaviour.

REFERENCES

- Abdul Kadir, A., Xu, X. and Hämmerle, E. 2011. Virtual machine tools and virtual machining- A technological review. *Robotics and Computer-Integrated Manufacturing* 27(3), pp. 494–508. doi: 10.1016/j.rcim.2010.10.003.
- Adamson, G., Wang, L., Holm, M. and Moore, P. 2017. Cloud manufacturing – a critical review of recent development and future trends. *International Journal of Computer Integrated Manufacturing* 30(4–5), pp. 347–380. doi: 10.1080/0951192X.2015.1031704.
- Advanced Modeling with Java | AnyLogic Help. 2021. Available at: <https://anylogic.help/advanced/index.html> [Accessed: 29 May 2021].
- Afazov, S. and Scrimieri, D. 2020. Chatter model for enabling a digital twin in machining. *International Journal of Advanced Manufacturing Technology* 110(9–10), pp. 2439–2444. doi: 10.1007/s00170-020-06028-9.
- Akintseva, A. V., Pereverzev, P.P., Reshetnikov, B. V. and Irshin, A. V. 2021. Analytical basics of digital twin for CNC round grinding process. In: *Materials Today: Proceedings*. Elsevier Ltd, pp. 1740–1744. doi: 10.1016/j.matpr.2020.08.244.
- Alzyadat, W.J., Alhroob, A., Almukahel, I.H. and Atan, R. 2019. Fuzzy Map Approach for Accruing Velocity of Big Data. *International Journal of Advanced Computer Research* 8(5)
- Apache Commons. 2021. Available at: <https://commons.apache.org/> [Accessed: 4 February 2022].
- Apache Kafka. 2022. Available at: <https://kafka.apache.org/> [Accessed: 22 February 2022].
- Apache Software Foundation 2021. Apache Tomcat.
- Apache Software Foundation 2022. Apache Maven.
- Ashton, K. 2009. That 'internet of things' thing. *RFID journal* 22(7), pp. 97–114.
- Avgerinou, M., Bertoldi, P. and Castellazzi, L. 2017. Trends in Data Centre Energy Consumption under the European Code of Conduct for Data Centre Energy Efficiency. *Energies* 10(10), p. 1470. doi: 10.3390/en10101470.
- Balderas, D., Ortiz, A., Méndez, E., Ponce, P. and Molina, A. 2021. Empowering Digital Twin for Industry 4.0 using metaheuristic optimization algorithms: case study PCB drilling optimization. *International Journal of Advanced Manufacturing Technology* 113(5–6), pp. 1295–1306. doi: 10.1007/s00170-021-06649-8.
- Banks, J., CARSON II, J.S., Barry, L. and others 2005. Discrete-event system simulation fourth edition.
- Bao, J., Guo, D., Li, J. and Zhang, J. 2019. The modelling and operations for the digital twin in the context of manufacturing. *Enterprise Information Systems* 13(4), pp. 534–556. doi: 10.1080/17517575.2018.1526324.
- Barika, M., Garg, S., Zomaya, A.Y. and Ranjan, R. 2021. Online Scheduling Technique to Handle Data Velocity Changes in Stream Workflows. *IEEE Transactions on Parallel and Distributed Systems* 32(8), pp. 2115–2130. doi: 10.1109/TPDS.2021.3059480.
- Bathe, K.-J. 2014. *Finite element procedures*. 2nd Editio. Klaus-Jurgen Bathe.
- Baur, C. and Wee, D. 2015. *Manufacturing's next act | McKinsey*.

- Bellman, R. and Kalaba, R. 1958. On adaptive control processes. *IRE Transactions on Automatic Control* 4(2), pp. 1–9. doi: 10.1109/TAC.1959.1104847.
- Belytschko, T., Gracie, R. and Ventura, G. 2009. A review of extended/generalized finite element methods for material modeling. *Modelling and Simulation in Materials Science and Engineering* 17(4), p. 043001. doi: 10.1088/0965-0393/17/4/043001.
- Bifet, A., Holmes, G., Pfahringer, B., Kranen, P., Kremer, H., Jansen, T. and Seidl, T. 2010. MOA: Massive Online Analysis, a Framework for Stream Classification and Clustering. In: Diethe, T., Cristianini, N., and Shawe-Taylor, J. eds. *Proceedings of the First Workshop on Applications of Pattern Analysis*. Proceedings of Machine Learning Research. Cumberland Lodge, Windsor, UK: PMLR, pp. 44–50.
- Boschert, S. and Rosen, R. 2016. Digital Twin—The Simulation Aspect. In: *Mechatronic Futures*. Cham: Springer International Publishing, pp. 59–74. doi: 10.1007/978-3-319-32156-1_5.
- Böß, V., Denkena, B., Dittrich, M.-A., Malek, T. and Friebe, S. 2021. Dixel-Based Simulation of Directed Energy Deposition Additive Manufacturing. *Journal of Manufacturing and Materials Processing* 5(1), p. 9. doi: 10.3390/jmmp5010009.
- Boulonne, A., Johansson, B., Skoogh, A. and Aufenanger, M. 2010. Simulation data architecture for sustainable development. In: *Proceedings - Winter Simulation Conference.*, pp. 3435–3446. doi: 10.1109/WSC.2010.5679033.
- Bracht, U. and Masurat, T. 2005. The Digital Factory between vision and reality. *Computers in Industry* 56(4), pp. 325–333. doi: 10.1016/j.compind.2005.01.008.
- Brandstetter, V. and Wehrstedt, J.C. 2018. A Framework for Multidisciplinary Simulation of Cyber-Physical Production Systems. *IFAC-PapersOnLine* 51(11), pp. 809–814. doi: 10.1016/j.ifacol.2018.08.418.
- Breiman, L. 1996. Bagging predictors. *Machine Learning* 24(2), pp. 123–140.
- Brettel, M., Friederichsen, N., Keller, M., Rosenberg, N., Rosenberg, M. and Rosenberg, N. 2014. How virtualization, decentralization and network building change the manufacturing landscape: An Industry 4.0 Perspective. *International Journal of Science, Engineering and Technology* 8(1), pp. 37–44.
- Cai, Y., Starly, B., Cohen, P. and Lee, Y.S. 2017. Sensor Data and Information Fusion to Construct Digital-twins Virtual Machine Tools for Cyber-physical Manufacturing. *Procedia Manufacturing* 10, pp. 1031–1042. doi: 10.1016/j.promfg.2017.07.094.
- Cambridge English Dictionary. 2020. Available at: <https://dictionary.cambridge.org/dictionary/english/manufacturing> [Accessed: 14 April 2020].
- Cao, X., Zhao, G. and Xiao, W. 2020. Digital Twin-oriented real-time cutting simulation for intelligent computer numerical control machining. *Proceedings of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture* . doi: 10.1177/0954405420937869.
- Carriere-Swallow, Y. and Haksar, V. 2019. *The Economics and Implications of Data*. doi: 10.5089/9781513511436.087.
- Carvalho, N., Chaim, O., Cazarini, E. and Gerolamo, M. 2018. Manufacturing in the fourth industrial revolution: A positive prospect in Sustainable Manufacturing. In: *Procedia Manufacturing.*, pp. 671–678. doi: 10.1016/j.promfg.2018.02.170.

- Chandrupatla, T.R. and Tirupathi, R. 2006. Finite element method in Engineering.
- Chen, B., Wan, J., Celesti, A., Li, D., Abbas, H. and Zhang, Q. 2018. Edge Computing in IoT-Based Manufacturing. *IEEE Communications Magazine* 56(9), pp. 103–109. doi: 10.1109/MCOM.2018.1701231.
- Chen, D., Heyer, S., Ibbotson, S., Salonitis, K., Steingrímsson, J.G. and Thiede, S. 2015. Direct digital manufacturing: Definition, evolution, and sustainability implications. *Journal of Cleaner Production* 107, pp. 615–625. doi: 10.1016/j.jclepro.2015.05.009.
- Chen, K.C. and Lien, S.Y. 2014. Machine-to-machine communications: Technologies and challenges. *Ad Hoc Networks* 18, pp. 3–23. doi: 10.1016/j.adhoc.2013.03.007.
- Choi, S., Kang, G., Jun, C., Lee, J.Y. and Han, S. 2017. Cyber-physical systems: A case study of development for manufacturing industry. *International Journal of Computer Applications in Technology* 55(4), pp. 289–297. doi: 10.1504/IJCAT.2017.086018.
- Choudhary, F. (Gartner analyst), Vashisth, S. (Gartner analyst) and Erick, B. (Gartner analyst) 2022. *Use Gartner's MLOps Framework to Operationalize Machine Learning Projects*.
- Chryssolouris, G., Mavrikios, D., Papakostas, N., Mourtzis, D., Michalos, G. and Georgoulas, K. 2009. Digital manufacturing: History, perspectives, and outlook. *Proceedings of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture* 223(5), pp. 451–462. doi: 10.1243/09544054JEM1241.
- Chwif, L., Barretto, M.R.P. and Paul, R.J. 2000. On simulation model complexity. In: *2000 winter simulation conference proceedings (Cat. No. 00CH37165)*, pp. 449–455.
- Ciavotta, M., Maso, G.D., Rovere, D., Tsvetanov, R. and Menato, S. 2020. Towards the Digital Factory: A Microservices-Based Middleware for Real-to-Digital Synchronization. In: Bucchiarone, A., Dragoni, N., Dustdar, S., Lago, P., Mazzara, M., Rivera, V., and Sadovykh, A. eds. *Microservices: Science and Engineering*. Cham: Springer International Publishing, pp. 273–297. doi: 10.1007/978-3-030-31646-4_11.
- Cohen, Y., Faccio, M., Pilati, F. and Yao, X. 2019. Design and management of digital manufacturing and assembly systems in the Industry 4.0 era. *International Journal of Advanced Manufacturing Technology* 105(9), pp. 3565–3577. doi: 10.1007/s00170-019-04595-0.
- Colombo, A.W., Schleuter, D. and Kircher, M. 2015. An approach to qualify human resources supporting the migration of SMEs into an Industrie4.0-compliant company infrastructure. In: *IECON 2015 - 41st Annual Conference of the IEEE Industrial Electronics Society*, pp. 3761–3766.
- Costello, K. and Omale, G. 2019. *Gartner Survey Reveals Digital Twins Are Entering Mainstream Use*.
- Davis, S.M. 1989. From “future perfect”: Mass customizing. *Planning review*
- DB Browser for SQLite. 2021. Available at: <https://sqlitebrowser.org/> [Accessed: 22 February 2022].
- Definition of wrapper | PCMag. 2022. Available at: <https://www.pcmag.com/encyclopedia/term/wrapper> [Accessed: 6 March 2022].
- Deng, C., Guo, R., Liu, C., Zhong, R.Y. and Xu, X. 2018. Data cleansing for energy-saving: a case of Cyber-Physical Machine Tools health monitoring system. *International Journal of*

- Production Research* 56(1–2), pp. 1000–1015. doi: 10.1080/00207543.2017.1394596.
- Denkena, B., Pape, O., Krödel, A., Böß, V., Ellersiek, L. and Mücke, A. 2021. Process design for 5-axis ball end milling using a real-time capable dynamic material removal simulation. *Production Engineering* 15(1), pp. 89–95. doi: 10.1007/s11740-020-01003-5.
- DESMO-J. 2017. Available at: <http://desmoj.sourceforge.net/home.html> [Accessed: 29 May 2021].
- Donovan, J. 2018. Does it Pay to Move from On-Premises to Public Cloud Storage? Available at: <https://wasabi.com/blog/on-premises-vs-cloud-storage/> [Accessed: 30 June 2021].
- Eclipse IDE 2021-12 | The Eclipse Foundation. 2021. Available at: <https://www.eclipse.org/eclipseide/> [Accessed: 7 March 2022].
- Edwin Cartlidge 2010. Information converted to energy – Physics World. Available at: <https://physicsworld.com/a/information-converted-to-energy/> [Accessed: 28 June 2021].
- Engineering Simulation Software | Ansys Products. 2021. Available at: <https://www.ansys.com/en-gb/products#t=ProductsTab&sort=relevancy&layout=card&numberOfResults=100> [Accessed: 29 May 2021].
- EU Code of Conduct on Data Centre Energy Efficiency* 2016.
- Evgenii, K., Dmitry, K. and Alexander, P. 2018. Simulation of multi-axis machining using the BSP-dexel representation. In: *50th International Symposium on Robotics, ISR 2018*. VDE Verlag GmbH, pp. 123–126.
- FEM. 2013. Available at: <https://davis.wpi.edu/~matt/courses/fem/fem.htm> [Accessed: 5 June 2021].
- Fogliatto, F.S., Da Silveira, G.J.C. and Borenstein, D. 2012. The mass customization decade: An updated review of the literature. *International Journal of Production Economics* 138(1), pp. 14–25. doi: 10.1016/j.ijpe.2012.03.002.
- Friederich, J., Francis, D.P., Lazarova-Molnar, S. and Mohamed, N. 2022. A framework for data-driven digital twins of smart manufacturing systems. *Computers in Industry* 136, p. 103586. doi: 10.1016/J.COMPIND.2021.103586.
- Fysikopoulos, A., Alexopoulos, T., Pastras, G., Stavropoulos, P. and Chryssolouris, G. 2015. On the Design of a Sustainable Production Line: The MetaCAM Tool. (57588), p. V015T19A015.
- Gabor, T., Belzner, L., Kiermeier, M., Beck, M.T. and Neitz, A. 2016. A simulation-based architecture for smart cyber-physical systems. In: *Proceedings - 2016 IEEE International Conference on Autonomic Computing, ICAC 2016*. Institute of Electrical and Electronics Engineers Inc., pp. 374–379. doi: 10.1109/ICAC.2016.29.
- Gajjam, M.N.S. and Gunasekhar, D.T. 2021. Key challenges and research direction in cloud storage. *Materials Today: Proceedings*. doi: 10.1016/j.matpr.2021.01.609.
- General Data Protection Regulation. 2018. Available at: <https://gdpr-info.eu/> [Accessed: 14 June 2021].
- Git. 2022. Available at: <https://git-scm.com/> [Accessed: 7 March 2022].
- GitHub - GRBL. 2016. Available at: <https://github.com/grbl/grbl> [Accessed: 2 November

2020].

Glaessgen, E. and Stargel, D. 2012. The Digital Twin Paradigm for Future NASA and U.S. Air Force Vehicles. In: *53rd AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics and Materials Conference* & *20th AIAA/ASME/AHS Adaptive Structures Conference* & *14th AIAA*. Reston, Virginia: American Institute of Aeronautics and Astronautics. doi: 10.2514/6.2012-1818.

Gordon, G. 1961. A general purpose systems simulation program. In: *Proceedings of the December 12-14, 1961, eastern joint computer conference: computers-key to total systems control.*, pp. 87-104.

Grey Box Model – Integrating Application Knowledge in the Learning Process – Fraunhofer ITWM. 2021. Available at: <https://www.itwm.fraunhofer.de/en/departments/opt/machine-learning-hybrid-models/grey-box-model-integrating-application-knowledge-learning-process.html> [Accessed: 26 December 2021].

Grieves, M. 2015. Digital Twin : Manufacturing Excellence through Virtual Factory Replication This paper introduces the concept of a A Whitepaper by Dr . Michael Grieves. *White Paper* (March)

Habib ur Rehman, M., Sun Liew, C., Abbas, A., Prakash Jayaraman, P., Ying Wah, T. and Khan, S.U. 2016. Big Data Reduction Methods: A Survey. *Data Science and Engineering* 1. doi: 10.1007/s41019-016-0022-0.

Harrington, J.L. 2016. *Relational database design and implementation*. Morgan Kaufmann.

He, S., Zeng, X., Yan, C., Gong, H. and Lee, C.H. 2017. Tri-dexel model based geometric simulation of multi-axis additive manufacturing. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Springer Verlag, pp. 819-830. doi: 10.1007/978-3-319-65298-6_73.

Henning, K. (National A. of S. and E., Wolfgang, W. (German R.C. for A.I. and Johannes, H. (Deutsche P.A. 2013. Recommendations for implementing the strategic initiative INDUSTRIE 4.0. *Final report of the Industrie 4.0 WG* , p. 82.

Heo, E. and Yoo, N. 2021. Numerical Control Machine Optimization Technologies through Analysis of Machining History Data Using Digital Twin. *Applied Sciences* 11(7), p. 3259. doi: 10.3390/app11073259.

Hill, J. 2020. Optimisation of tool life through novel data acquisition and decision making techniques.

Hill, J.L., Prickett, P.W., Grosvenor, R.I. and Hankins, G. 2019. The practical exploitation of tacit machine tool intelligence. *International Journal of Advanced Manufacturing Technology* 104(5-8), pp. 1693-1707. doi: 10.1007/S00170-019-03963-0.

Hinchy, E.P., Carcagno, C., O'Dowd, N.P. and McCarthy, C.T. 2020. Using finite element analysis to develop a digital twin of a manufacturing bending operation. In: *Procedia CIRP*. Elsevier B.V., pp. 568-574. doi: 10.1016/j.procir.2020.03.031.

Van Hook, T. 1986. Real-time shaded NC milling display. *ACM SIGGRAPH Computer Graphics* 20(4), pp. 15-20.

How Much Does Running A Private Data Center Cost Per Year. 2020. Available at: <https://www.streamdatacenters.com/glossary/data-center-cost/> [Accessed: 28 June 2021].

Iglesias, D. et al. 2017. Digital twin applications for the JET divertor. *Fusion Engineering and*

Design 125, pp. 71–76. doi: 10.1016/j.fusengdes.2017.10.012.

Intel® Core™ i5–4310U Processor. 2014. Available at:
<https://www.intel.co.uk/content/www/uk/en/products/sku/80343/intel-core-i54310u-processor-3m-cache-up-to-3-00-ghz/specifications.html> [Accessed: 1 March 2022].

Intel 2015. The Internet of Things: Exploring the next technology frontier.

Inui, M., Huang, Y., Onozuka, H. and Umezu, N. 2020. Geometric simulation of power skiving of internal gear using solid model with triple-dexel representation. In: *Procedia Manufacturing*. Elsevier B.V., pp. 520–527. doi: 10.1016/j.promfg.2020.05.078.

Iwata, K., Onosato, M., Teramoto, K. and Osaki, S. 1995. A Modelling and Simulation Architecture for Virtual Manufacturing Systems. *CIRP Annals - Manufacturing Technology* 44(1), pp. 399–402. doi: 10.1016/S0007-8506(07)62350-6.

JAR File Overview. 2019. Available at:
<https://docs.oracle.com/javase/8/docs/technotes/guides/jar/jarGuide.html> [Accessed: 6 March 2022].

Jones, D., Snider, C., Nassehi, A., Yon, J. and Hicks, B. 2020. Characterising the Digital Twin: A systematic literature review. *CIRP Journal of Manufacturing Science and Technology* 29, pp. 36–52. doi: 10.1016/j.cirpj.2020.02.002.

Joy, J. and Feng, H.Y. 2017. Efficient milling part geometry computation via three-step update of frame-sliced voxel representation workpiece model. *International Journal of Advanced Manufacturing Technology* 92(5–8), pp. 2365–2378. doi: 10.1007/s00170-017-0168-6.

JSON. 2017. Available at: <https://www.json.org/json-en.html> [Accessed: 8 January 2022].

Kagermann, H., Lukas, W.-D. and Wahlster, W. 2011. Industrie 4.0: Mit dem Internet der Dinge auf dem Weg zur 4. industriellen Revolution. *VDI nachrichten* 13(11), p. 2.

Kärkkäinen, M., Ala-Risku, T. and Främling, K. 2003. The product centric approach: A solution to supply network information management problems? *Computers in Industry* 52(2), pp. 147–159. doi: 10.1016/S0166-3615(03)00086-1.

Kim, B.S., Kang, B.G., Choi, S.H. and Kim, T.G. 2017. Data modeling versus simulation modeling in the big data era: Case study of a greenhouse control system. *Simulation* 93(7), pp. 579–594. doi: 10.1177/0037549717692866.

Kim, B.S. and Kim, T.-G. 2019. Cooperation of simulation and data model for performance analysis of complex systems. *International Journal of Simulation Modelling* 18(4), pp. 608–619.

Kitchin, R. and McArdle, G. 2016. What makes Big Data, Big Data? Exploring the ontological characteristics of 26 datasets: <http://dx.doi.org/10.1177/2053951716631130> 3(1). doi: 10.1177/2053951716631130.

Klitou, D., Conrads, J. and Rasmussen, M. 2017. Key lessons from national industry 4.0 policy initiatives in Europe. *Digital Transformation Monitor, European Commission* (May), pp. 1–11.

Knezevic, D. 2018. *Enabling High-Fidelity Digital Twins of Critical Assets Via Reduced Order Modeling*.

Kong, T., Hu, T., Zhou, T. and Ye, Y. 2021. Data Construction Method for the Applications of Workshop Digital Twin System. In: *Journal of Manufacturing Systems*. Elsevier B.V., pp.

323–328. doi: 10.1016/j.jmsy.2020.02.003.

Kotthoff, L., Thornton, C., Hoos, H.H., Hutter, F. and Leyton-Brown, K. 2019. Auto-WEKA: Automatic model selection and hyperparameter optimization in WEKA. In: *Automated Machine Learning*. Springer, Cham, pp. 81–95.

Kritzinger, W., Karner, M., Traar, G., Henjes, J. and Sihn, W. 2018. Digital Twin in manufacturing: A categorical literature review and classification. *IFAC-PapersOnLine* 51(11), pp. 1016–1022. doi: 10.1016/j.ifacol.2018.08.474.

Kunath, M. and Winkler, H. 2018. Integrating the Digital Twin of the manufacturing system into a decision support system for improving the order management process. In: *Procedia CIRP*. Elsevier B.V., pp. 225–231. doi: 10.1016/j.procir.2018.03.192.

Kusiak, A. 2017. Smart manufacturing must embrace big data. *Nature* 544(7648), pp. 23–25. doi: 10.1038/544023a.

Law, A.M. 2015. *Simulation Modeling and Analysis (5th Edition)*.

Lechevalier, D., Shin, S.S.S., Woo, J. and Rachuri, S. 2015. A virtual milling machine model to generate machine- monitoring data for predictive analytics.

Lee, R.S. and Wu, C.M. 2015. Virtual machine tool simulation through network communication with real CNC controller. In: *2015 IFToMM World Congress Proceedings, IFToMM 2015*. doi: 10.6567/IFToMM.14TH.WC.OS13.081.

Li, W.D. and Mehnen, J. 2013. Cloud manufacturing. *Distributed Computing Technologies for Global and Sustainable Manufacturing, Springer Series in Advanced Manufacturing*. DOI= <http://dx.doi.org/10.1007/978-1-4471-4935-4>

Liu, S., Lu, S., Li, J., Sun, X., Lu, Y. and Bao, J. 2021. Machining process-oriented monitoring method based on digital twin via augmented reality. *International Journal of Advanced Manufacturing Technology* 113(11–12), pp. 3491–3508. doi: 10.1007/s00170-021-06838-5.

Liu, Y., Dillon, T., Yu, W., Rahayu, W. and Mostafa, F. 2020. Noise Removal in the Presence of Significant Anomalies for Industrial IoT Sensor Data in Manufacturing. *IEEE Internet of Things Journal* 7(8), pp. 7084–7096. doi: 10.1109/JIOT.2020.2981476.

Lorong, P., Yvonnet, J., Coffignal, G. and Cohen, S. 2006. Contribution of computational mechanics in numerical simulation of machining and blanking: State-of-the-art. *Archives of Computational Methods in Engineering* 13(1), pp. 45–90. doi: 10.1007/BF02905931.

Lu, Y., Liu, C., Wang, K.I.K., Huang, H. and Xu, X. 2020. Digital Twin-driven smart manufacturing: Connotation, reference model, applications and research issues. *Robotics and Computer-Integrated Manufacturing* 61, p. 101837. doi: 10.1016/j.rcim.2019.101837.

Lyly-Yrjänäinen, J., Holmström, J., Johansson, M.I. and Suomala, P. 2016. Effects of combining product-centric control and direct digital manufacturing: The case of preparing customized hose assembly kits. *Computers in Industry* 82, pp. 82–94. doi: 10.1016/j.compind.2016.05.009.

Macmillan Dictionary. 2020. Available at:

<https://www.macmillandictionary.com/dictionary/british/manufacturing> [Accessed: 14 April 2020].

Malleson, N., Minors, K., Kieu, L.M., Ward, J.A., West, A.A. and Heppenstall, A. 2020. Simulating Crowds in Real Time with Agent-Based Modelling and a Particle Filter. *2019:158:2* 23(3), pp. 1–20. doi: 10.18564/JASSS.4266.

- Mansouri, Y., Toosi, A.N. and Buyya, R. 2017. Data storage management in cloud environments: Taxonomy, survey, and future directions. *ACM Computing Surveys* 50(6), p. 51. doi: 10.1145/3136623.
- Marinkovic, D. and Zehn, M. 2019. Survey of finite element method-based real-time simulations. *Applied Sciences (Switzerland)* 9(14), p. 2775. doi: 10.3390/app9142775.
- MATLAB. 2022.
- Mell, P., Grance, T. and others 2011. The NIST definition of cloud computing.
- Merriam-Webster. 2020. Available at: <https://www.merriam-webster.com/dictionary/manufacturing> [Accessed: 14 April 2020].
- Meyer, I., Rentsch, R. and Karpuschewski, B. 2021. Universal approach for simulation and analysis of cutting edge engagement in orbital drilling processes. *CIRP Journal of Manufacturing Science and Technology* 32, pp. 346–355. doi: 10.1016/j.cirpj.2021.01.008.
- Microsoft Excel. 2022.
- MOA. 2021. Available at: <https://moa.cms.waikato.ac.nz/> [Accessed: 1 March 2022].
- Mourtzis, D. 2019. Simulation in the design and operation of manufacturing systems: state of the art and new trends. *International Journal of Production Research* 7543. doi: 10.1080/00207543.2019.1636321.
- Mourtzis, D., Doukas, M. and Bernidaki, D. 2014. Simulation in manufacturing: Review and challenges. *Procedia CIRP* 25(C), pp. 213–229. doi: 10.1016/j.procir.2014.10.032.
- MTConnect. 2018. Available at: <https://www.mtconnect.org/> [Accessed: 27 May 2021].
- Niebel, B.W. 1965. Mechanized process selection for planning new designs. *ASME paper* 737
- Oberkampff, W.L. 2019. Simulation Accuracy, Uncertainty, and Predictive Capability: A Physical Sciences Perspective. In: *Methods and Applications*. Springer, Cham, pp. 69–97. doi: 10.1007/978-3-319-70766-2_3.
- Olden, J.D. and Jackson, D.A. 2002. Illuminating the “black box”: a randomization approach for understanding variable contributions in artificial neural networks. *Ecological Modelling* 154(1–2), pp. 135–150. doi: 10.1016/S0304-3800(02)00064-9.
- Oleghe, O. 2020. A predictive noise correction methodology for manufacturing process datasets. *Journal of Big Data* 7(1), pp. 1–27. doi: 10.1186/s40537-020-00367-w.
- Onggo, B.S., Juan, A.A., Mustafee, N., Smart, A. and Molloy, O. 2019. Symbiotic simulation system: Hybrid systems model meets big data analytics. *Proceedings - Winter Simulation Conference 2018-Decem(2017)*, pp. 1358–1369. doi: 10.1109/WSC.2018.8632407.
- OPC Foundation. 2021. Available at: <https://opcfoundation.org/> [Accessed: 27 May 2021].
- Overview — SimPy 4.0.2.dev1+g2973dbe documentation. 2021. Available at: <https://simpy.readthedocs.io/en/latest/index.html> [Accessed: 29 May 2021].
- Pandey, K.K. and Shukla, D. 2020. Stratified Sampling-Based Data Reduction and Categorization Model for Big Data Mining. In: *Lecture Notes in Networks and Systems*. Springer, pp. 107–122. doi: 10.1007/978-981-15-3325-9_9.
- Plant Simulation and Throughput Optimization | Siemens Digital Industries Software. 2021. Available at:

- <https://www.plm.automation.siemens.com/global/en/products/manufacturing-planning/plant-simulation-throughput-optimization.html> [Accessed: 29 May 2021].
- PowerMill | 5-Axis CAM Software | 5-Axis Machining | Autodesk. 2021. Available at: <https://www.autodesk.co.uk/products/powermill/overview> [Accessed: 29 May 2021].
- Priemer, R. 1990. Signals and Signal Processing. In: *Introductory Signal Processing*. WORLD SCIENTIFIC, pp. 1–9. doi: 10.1142/9789814434409_0001.
- Proctor, M. and Wilkins, J. 2019. The Current State of Industry 4.0 Around the World.
- Quinlan, J.R. 1987. Simplifying decision trees. *International journal of man-machine studies* 27(3), pp. 221–234.
- Rathore, M.M., Shah, S.A., Shukla, D., Bentafat, E. and Bakiras, S. 2021. The Role of AI, Machine Learning, and Big Data in Digital Twinning: A Systematic Literature Review, Challenges, and Opportunities. *IEEE Access* 9, pp. 32030–32052. doi: 10.1109/ACCESS.2021.3060863.
- Ren, S., Zhang, Y., Liu, Y., Sakao, T., Huisingh, D. and Almeida, C.M.V.B. 2019. A comprehensive review of big data analytics throughout product lifecycle to support sustainable smart manufacturing: A framework, challenges and future research directions. *Journal of Cleaner Production* 210, pp. 1343–1365. doi: 10.1016/j.jclepro.2018.11.025.
- Roberts, S.D. and Pegden, C.D. 2017. The history of simulation modeling. In: *Proceedings - Winter Simulation Conference.*, pp. 308–323. doi: 10.1109/WSC.2017.8247795.
- Robertson, N. and Perera, T. 2002. Automated data collection for simulation? *Simulation Practice and Theory* 9(6–8), pp. 349–364. doi: 10.1016/S0928-4869(01)00055-6.
- Robotics and Automation Simulation | Siemens Digital Industries Software. 2021. Available at: <https://www.plm.automation.siemens.com/global/en/products/manufacturing-planning/robotics-automation-simulation.html> [Accessed: 29 May 2021].
- Röck, S. 2021. Predicting regenerative chatter in milling with hardware-in-the-loop simulation using a dextral-based cutting model. *Production Engineering* . doi: 10.1007/s11740-021-01040-8.
- Rohrer, M.W. 2000. Seeing is believing: The importance of visualization in manufacturing simulation. In: *Winter Simulation Conference Proceedings.*, pp. 1211–1216. doi: 10.1109/wsc.2000.899087.
- Rose, K., Eldridge, S. and Chapin, L. 2015. *The Internet of Things: An Overview*.
- Rosenthal, D.S.H.D.C., Rosenthal, D.S.H.D.C., Miller, E.L., Adams, I.F., Storer, M.W. and Zadok, E. 2012. The economics of long-term digital storage. *Memory of the World in the Digital Age, Vancouver, BC*
- Sadeghifar, M., Sedaghati, R., Jomaa, W. and Songmene, V. 2018. A comprehensive review of finite element modeling of orthogonal machining process: chip formation and surface integrity predictions. *International Journal of Advanced Manufacturing Technology* 96(9–12), pp. 3747–3791. doi: 10.1007/s00170-018-1759-6.
- Sahatqija, K., Ajdari, J., Zenuni, X., Raufi, B. and Ismaili, F. 2018. Comparison between relational and NOSQL databases. *2018 41st International Convention on Information and Communication Technology, Electronics and Microelectronics, MIPRO 2018 - Proceedings* , pp. 216–221. doi: 10.23919/MIPRO.2018.8400041.
- Sakoe, H. and Chiba, S. 1978. Dynamic Programming Algorithm Optimization for Spoken

- Word Recognition. *IEEE Transactions on Acoustics, Speech, and Signal Processing* 26(1), pp. 43–49. doi: 10.1109/TASSP.1978.1163055.
- Salvador, S. and Chan, P. 2007. Toward accurate dynamic time warping in linear time and space. *Intelligent Data Analysis* 11(5), pp. 561–580.
- Sanders, N. 2012. A Possible First Use of CAM/CAD. In: Tatnall, A. ed. *Reflections on the History of Computing: Preserving Memories and Sharing Stories*. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 43–56. doi: 10.1007/978-3-642-33899-1_3.
- Satyanarayanan, M. 2017. The emergence of edge computing. *Computer* 50(1), pp. 30–39. doi: 10.1109/MC.2017.9.
- Schleich, B., Anwer, N., Mathieu, L. and Wartzack, S. 2017. Shaping the digital twin for design and production engineering. *CIRP Annals - Manufacturing Technology* 66(1), pp. 141–144. doi: 10.1016/j.cirp.2017.04.040.
- Schluse, M. and Rossmann, J. 2016. From simulation to experimentable digital twins: Simulation-based development and operation of complex technical systems. In: *ISSE 2016 - 2016 International Symposium on Systems Engineering - Proceedings Papers*. Institute of Electrical and Electronics Engineers Inc. doi: 10.1109/SysEng.2016.7753162.
- Schroeder, G.N., Steinmetz, C., Pereira, C.E. and Espindola, D.B. 2016. Digital Twin Data Modeling with AutomationML and a Communication Methodology for Data Exchange. *IFAC-PapersOnLine* 49(30), pp. 12–17. doi: 10.1016/j.ifacol.2016.11.115.
- Senin, P. 2008. Dynamic time warping algorithm review. *Information and Computer Science Department University of Hawaii at Manoa Honolulu, USA* 855(1–23), p. 40.
- Seventekidis, P. and Giagopoulos, D. 2021. A combined finite element and hierarchical Deep learning approach for structural health monitoring: Test on a pin-joint composite truss structure. *Mechanical Systems and Signal Processing* 157, p. 107735. doi: 10.1016/j.ymssp.2021.107735.
- Shao, G., Jain, S., Laroque, C., Lee, L.H., Lendermann, P. and Rose, O. 2019. Digital Twin for Smart Manufacturing: The Simulation Aspect. In: *Proceedings - Winter Simulation Conference*. Institute of Electrical and Electronics Engineers Inc., pp. 2085–2098. doi: 10.1109/WSC40007.2019.9004659.
- Shokoohi-Yekta, M., Hu, B., Jin, H., Wang, J. and Keogh, E. 2015. Generalizing dynamic time warping to the multi-dimensional case requires an adaptive approach.
- Siddiqi, A., Hashem, I.A.T., Yaqoob, I., Marjani, M., Shamshirband, S., Gani, A. and Nasaruddin, F. 2016. A survey of big data management: Taxonomy and state-of-the-art. *Journal of Network and Computer Applications* 71, pp. 151–166. doi: 10.1016/j.jnca.2016.04.008.
- Siebrecht, T., Biermann, D., Ludwig, H., Rausch, S., Kersting, P., Blum, H. and Rademacher, A. 2014. Simulation of grinding processes using finite element analysis and geometric simulation of individual grains. *Production Engineering* 8(3), pp. 345–353. doi: 10.1007/s11740-013-0524-9.
- Simulation, Production Planning and Scheduling Software | Simio. 2022. Available at: <https://www.simio.com/> [Accessed: 13 August 2022].
- Simulation software - SIMULIA by Dassault Systèmes®. 2022. Available at: <https://www.3ds.com/products-services/simulia/products/> [Accessed: 29 July 2022].

Simulator for industrial robots and offline programming - RoboDK. 2021. Available at: <https://robodk.com/> [Accessed: 29 May 2021].

Sneader, K. and Sternfels, B. 2020. From surviving to thriving. *McKinsey Insights*

Soori, M., Arezoo, B. and Habibi, M. 2016. Tool deflection error of three-axis computer numerical control milling machines, monitoring and minimizing by a virtual machining system. *Journal of Manufacturing Science and Engineering* 138(8), p. 081005. doi: 10.1115/1.4032393.

SQLite Home Page. 2022. Available at: <https://www.sqlite.org/index.html> [Accessed: 22 October 2020].

SSE HTML Standard. 2022. Available at: <https://html.spec.whatwg.org/multipage/server-sent-events.html#server-sent-events> [Accessed: 7 March 2022].

Statista 2022. Most popular database management systems. Available at: <https://www.statista.com/statistics/809750/worldwide-popularity-ranking-database-management-systems/> [Accessed: 15 August 2022].

Stavropoulos, P., Papacharalampopoulos, A., Vasiliadis, E. and Chryssolouris, G. 2016. Tool wear predictability estimation in milling based on multi-sensorial data. *The International Journal of Advanced Manufacturing Technology* 82(1–4), pp. 509–521. doi: 10.1007/s00170-015-7317-6.

Strozzi, F., Colicchia, C., Creazza, A. and Noè, C. 2017. Literature review on the 'smart factory' concept using bibliometric tools. *International Journal of Production Research* 55(22), pp. 1–20. doi: 10.1080/00207543.2017.1326643.

Sun, Y.J., Yan, C., Wu, S.W., Gong, H. and Lee, C.H. 2018. Geometric simulation of 5-axis hybrid additive-subtractive manufacturing based on Tri-dexel model. *International Journal of Advanced Manufacturing Technology* 99(9–12), pp. 2597–2610. doi: 10.1007/s00170-018-2577-6.

Sutherland, I.E. 1963. Sketchpad a man-machine graphical communication system. In: *AFIPS Conference Proceedings - 1963 Spring Joint Computer Conference, AFIPS 1963*. Association for Computing Machinery, Inc, pp. 329–346. doi: 10.1145/1461551.1461591.

SystemC Community. 2021. Available at: <https://accelera.org/community/systemc> [Accessed: 29 May 2021].

Tamás, P. and Illés, B. 2016. Process improvement trends for manufacturing systems in industry 4.0. *Academic Journal of Manufacturing Engineering* 14(4), pp. 119–125.

Tao, F., Cheng, J., Qi, Q., Zhang, M., Zhang, H. and Sui, F. 2018a. Digital twin-driven product design, manufacturing and service with big data. *International Journal of Advanced Manufacturing Technology* 94(9–12), pp. 3563–3576. doi: 10.1007/s00170-017-0233-1.

Tao, F., Qi, Q., Liu, A. and Kusiak, A. 2018b. Data-driven smart manufacturing. *Journal of Manufacturing Systems* 48, pp. 157–169. doi: 10.1016/j.jmsy.2018.01.006.

Tao, F., Zhang, H., Liu, A. and Nee, A.Y.C. 2019. Digital Twin in Industry: State-of-the-Art. *IEEE Transactions on Industrial Informatics* 15(4), pp. 2405–2415. doi: 10.1109/TII.2018.2873186.

Tao, F. and Zhang, M. 2017. Digital Twin Shop-Floor: A New Shop-Floor Paradigm Towards Smart Manufacturing. *IEEE Access* 5, pp. 20418–20427. doi: 10.1109/ACCESS.2017.2756069.

The World Bank 2021. Manufacturing, value added (% of GDP) - Germany | Data. Available

at: <https://data.worldbank.org/indicator/NV.IND.MANF.ZS?locations=DE> [Accessed: 13 August 2022].

Thönes, J. 2015. Microservices. *IEEE Software* 32(1). doi: 10.1109/MS.2015.11.

Toubhans, B., Lorong, P., Viprey, F., Fromentin, G. and Karaoui, H. 2021. A versatile approach, considering tool wear, to simulate undercut error when turning thin-walled workpieces. *The International Journal of Advanced Manufacturing Technology* . doi: 10.1007/s00170-021-07243-8.

Tuegel, E.J., Ingraffea, A.R., Eason, T.G. and Spottswood, S.M. 2011. Reengineering aircraft structural life prediction using a digital twin. *International Journal of Aerospace Engineering* . doi: 10.1155/2011/154798.

TWI 2020. What is Digital Manufacturing? – TWI. Available at: <https://www.twi-global.com/technical-knowledge/faqs/what-is-digital-manufacturing> [Accessed: 27 May 2020].

Wang, J., Quan, L. and Tang, K. 2020. A prediction method based on the voxel model and the finite cell method for cutting force-induced deformation in the five-axis milling process. *Computer Methods in Applied Mechanics and Engineering* 367. doi: 10.1016/j.cma.2020.113110.

Wang, L. 2017. Heterogeneous Data and Big Data Analytics. *Automatic Control and Information Sciences* 3(1), pp. 8–15. doi: 10.12691/acis-3-1-3.

Waterston, S. 2021. Data Utilization: Facts, Stats & IO Research. Available at: <https://iotechnologies.com/blog/data-utilization-research> [Accessed: 31 March 2020].

Weyer, S., Meyer, T., Ohmer, M., Gorecky, D. and Zühlke, D. 2016. Future Modeling and Simulation of CPS-based Factories: an Example from the Automotive Industry. *IFAC-PapersOnLine* 49(31), pp. 97–102. doi: 10.1016/j.ifacol.2016.12.168.

What Is a Database | Oracle. 2021. Available at: <https://www.oracle.com/database/what-is-database/> [Accessed: 4 January 2022].

William 2020. Rebuttal for “ Simulation in industry 4 . 0 : A. 149(January), pp. 1–12.

WITNESS Simulation Modeling Software | Lanner. 2021. Available at: <https://www.lanner.com/en-gb/technology/witness-simulation-software.html> [Accessed: 29 May 2021].

Witten, I.H., Frank, E., Hall, M.A. and Pal, C.J. 2016. *Data Mining, Fourth Edition: Practical Machine Learning Tools and Techniques*. 4th ed. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Wu, D., Rosen, D.W., Wang, L. and Schaefer, D. 2015. Cloud-based design and manufacturing: A new paradigm in digital manufacturing and design innovation. *CAD Computer Aided Design* 59, pp. 1–14. doi: 10.1016/j.cad.2014.07.006.

Xu, Y., Sun, Y., Liu, X. and Zheng, Y. 2019. A Digital-Twin-Assisted Fault Diagnosis Using Deep Transfer Learning. *IEEE Access* 7, pp. 19990–19999. doi: 10.1109/ACCESS.2018.2890566.

Yadav, M. and Alam, M.A. 2018. Dynamic Time Warping (DTW) Algorithm In Speech: A Review. *International Journal of Research in Electronics and Computer Engineering* 6(1), pp. 524–528.

Yan, H.S. and Xue, C.G. 2007. Decision-making in self-reconfiguration of a knowledgeable

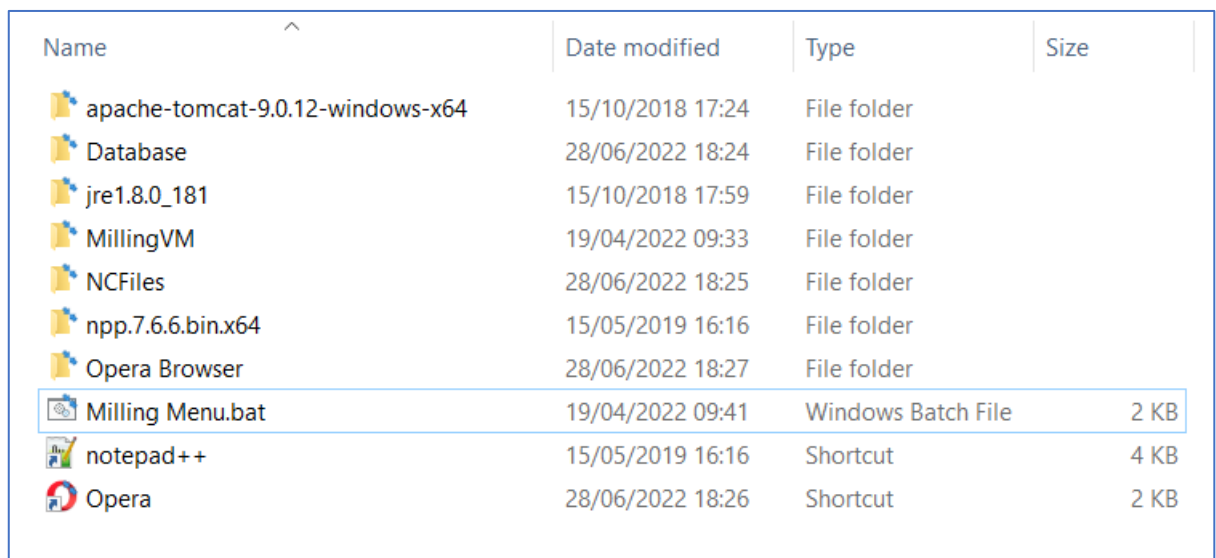
- manufacturing system. *International Journal of Production Research* 45(12), pp. 2735–2758. doi: 10.1080/00207540600711895.
- Yang, Z., Eddy, D., Krishnamurty, S., Grosse, I., Denno, P., Lu, Y. and Witherell, P. 2017. Investigating Grey-Box Modeling for Predictive Analytics in Smart Manufacturing. American Society of Mechanical Engineers Digital Collection. doi: 10.1115/detc2017-67794.
- Yao, Y., Huang, L., Sharma, A.B., Golubchik, L. and Neely, M.J. 2014. Power cost reduction in distributed data centers: A two-time-scale approach for delay tolerant workloads. *IEEE Transactions on Parallel and Distributed Systems* 25(1), pp. 200–211. doi: 10.1109/TPDS.2012.341.
- Zhang, Y., Xu, X. and Liu, Y. 2011. Numerical control machining simulation: A comprehensive survey. *International Journal of Computer Integrated Manufacturing* 24(7), pp. 593–609. doi: 10.1080/0951192X.2011.566283.
- Zhang, Z. and Gu, G.X. 2020. Finite-Element-Based Deep-Learning Model for Deformation Behavior of Digital Materials. *Advanced Theory and Simulations* 3(7), p. 2000031. doi: 10.1002/adts.202000031.
- Zhao, L., Fang, Y., Lou, P., Yan, J. and Xiao, A. 2021. Cutting Parameter Optimization for Reducing Carbon Emissions Using Digital Twin. *International Journal of Precision Engineering and Manufacturing* 22, pp. 933–949. doi: 10.1007/s12541-021-00486-1.
- Zhao, P. and Sun, B. 2021. Adaptive Modification of Digital Twin Model of CNC Machine Tools Coordinately Driven by Mechanism Model and Data Model. *Journal of Physics: Conference Series* 1875(1), p. 012003. doi: 10.1088/1742-6596/1875/1/012003.
- Zheng, Y., Yang, S. and Cheng, H. 2019. An application framework of digital twin and its case study. *Journal of Ambient Intelligence and Humanized Computing* 10(3), pp. 1141–1153. doi: 10.1007/s12652-018-0911-3.
- Zhou, Y., Xing, T., Song, Y., Li, Y., Zhu, X., Li, G. and Ding, S. 2021. Digital-twin-driven geometric optimization of centrifugal impeller with free-form blades for five-axis flank milling. *Journal of Manufacturing Systems* 58, pp. 22–35. doi: 10.1016/j.jmsy.2020.06.019.
- Zhu, Z., Xi, X., Xu, X. and Cai, Y. 2021. Digital Twin-driven machining process for thin-walled part manufacturing. *Journal of Manufacturing Systems* 59, pp. 453–466. doi: 10.1016/j.jmsy.2021.03.015.
- Zhuang, C., Gong, J. and Liu, J. 2021. Digital twin-based assembly data management and process traceability for complex products. *Journal of Manufacturing Systems* 58, pp. 118–131. doi: 10.1016/j.jmsy.2020.05.011.

Appendix A. VMC SIMULATOR (VERSION 0.65)

The simulator has been developed as a self-contained package that can be extracted and run either in a Windows or Linux environment. The following example is based on the Windows operating system, but the process is very similar for a Linux system since Java supports portability and it is an excellent programming language for portable applications. In the following sections, all steps of installing the simulator and running an example are described. The intention is to provide a teach-by-example document for potential users of the VMC simulator.

A.1 SOFTWARE INSTALLATION

1. Download or copy MillingVM.zip from the location that it is provided.
2. Extract MillingVM.zip to a folder. The folders shown in Figure 72 are created into the folder where the .zip file has been extracted. These folders contain (in the order that they appear in Figure 72): Tomcat webserver, SQLite files, Java Runtime Environment, simulator's Java archive files (.jar), example numerical control and monitoring data files, notepad application to read data files, Opera browser to access the web application, batch file to start the simulator and shortcuts.

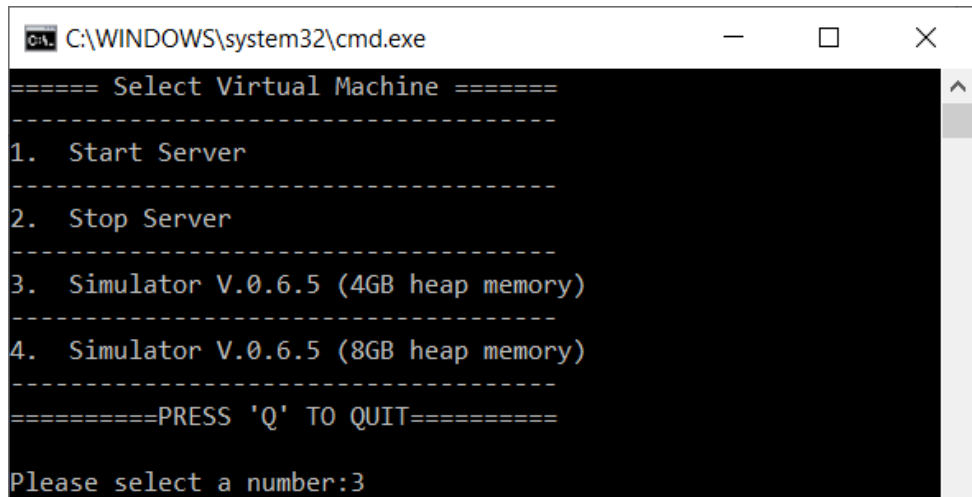


Name	Date modified	Type	Size
apache-tomcat-9.0.12-windows-x64	15/10/2018 17:24	File folder	
Database	28/06/2022 18:24	File folder	
jre1.8.0_181	15/10/2018 17:59	File folder	
MillingVM	19/04/2022 09:33	File folder	
NCFiles	28/06/2022 18:25	File folder	
npp.7.6.6.bin.x64	15/05/2019 16:16	File folder	
Opera Browser	28/06/2022 18:27	File folder	
Milling Menu.bat	19/04/2022 09:41	Windows Batch File	2 KB
notepad++	15/05/2019 16:16	Shortcut	4 KB
Opera	28/06/2022 18:26	Shortcut	2 KB

Figure 72 VMC simulator folders

3. Run MillingVM Menu.bat

Option 1 starts the web server, option 2 stops the web server, option 3 reserves 4GB in memory to run the simulator's local GUI and option 4 is similar to option 3 but with double the reserved memory. Q exits the simulator system start menu.



```
C:\WINDOWS\system32\cmd.exe
===== Select Virtual Machine =====
-----
1. Start Server
-----
2. Stop Server
-----
3. Simulator V.0.6.5 (4GB heap memory)
-----
4. Simulator V.0.6.5 (8GB heap memory)
-----
=====PRESS 'Q' TO QUIT=====
Please select a number:3
```

Figure 73 Simulator system start menu

4. Type 3 (or 4 depending on the available memory) and press enter to run the simulator's local GUI application. After selecting an option, information about the computer is displayed.

A.2 VMC SIMULATOR DATABASE

When the GUI is started the user must create a database or select an existing one which will hold all simulator input and output data. Using different databases is mainly required when different element sizes are examined otherwise multiple setups can be stored in one database. To create a database:

5. Click on "New" button.
6. Type in the available text field the name of the new database and modify the element size as needed. The time step field overrides the default timestep of the simulator but this is not supported for the majority of the simulator's functions.
7. Click "Save" and the new database has been created.

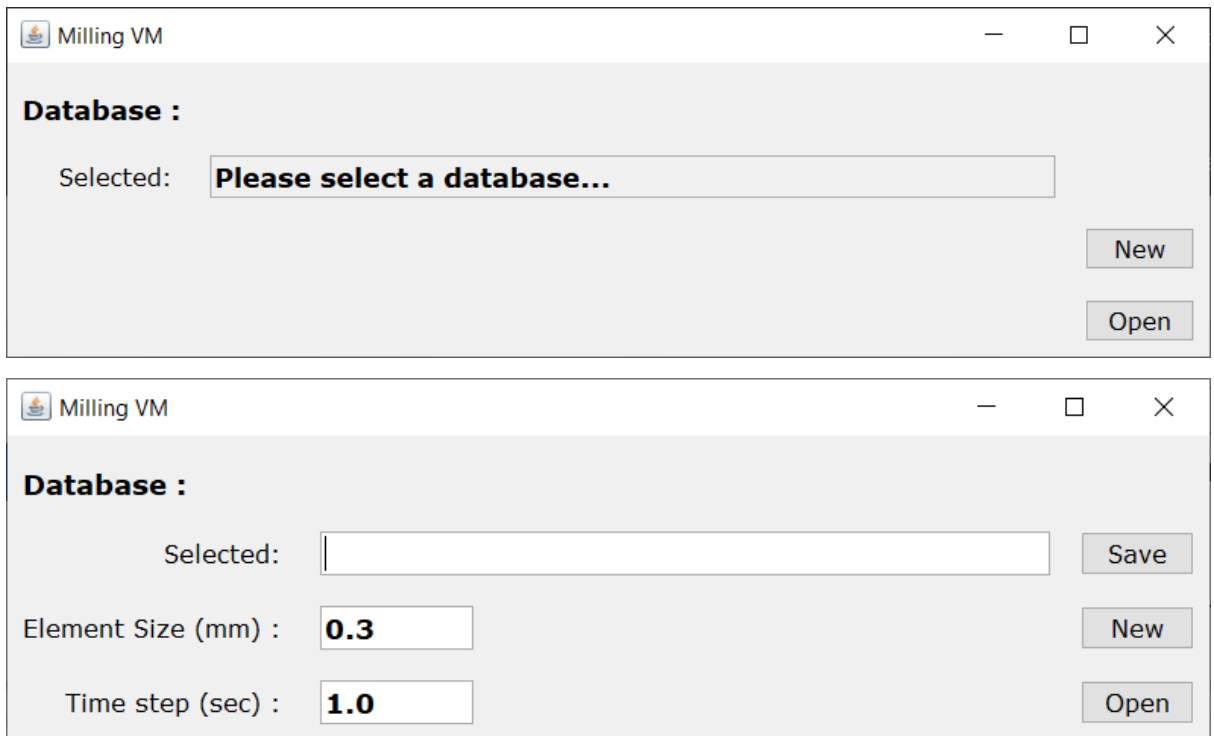
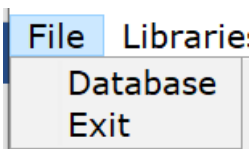


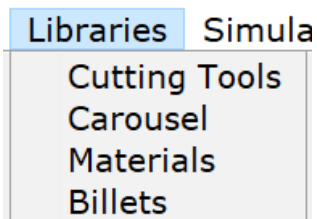
Figure 74 Top: Database selection screen. Bottom: Creation of new database

After the database has been created (or selected) the full view of the simulator's local GUI opens. From there, the user can enter parts and cutting tools, configure the simulator setup and analyse both the physical and the virtual process. In brief, the menu options have the following functionalities.



File > Database shows the database selection tool

File > Exit exits the application

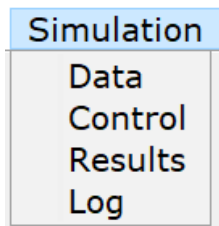


Libraries > Cutting Tools opens the cutting tools creation and management tool

Libraries > Carousel shows the carousel setup panel

Libraries > Materials opens the materials creation and management tool

Libraries > Billets opens the billets creation and management tool

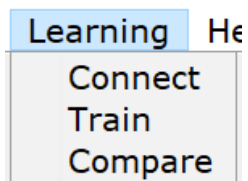


Simulation > Data opens the monitoring data explorer.

Simulation > Control opens the simulation setup tool.

Simulation > Results shows the results after a simulation is run.

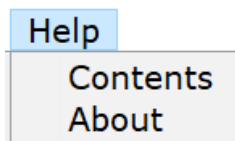
Simulation > Log shows the log of events related to the simulator.



Learning > Connect opens the panel to connect NC files to monitoring data sources

Learning > Train opens the learning module training tool

Learning > Compare opens the dataset comparison tool



Help > Contents opens simulator's help

Help > About shows details

A.3 LIBRARIES

Before running a simulation, the cutting tools and billet need to be specified.

8. Click on "Libraries > Cutting Tools" and in the panel that opens click on "New" button.

9. Fill out the new tool form. Tool Name is a unique name describing the cutting tool. Tool type is the type of cutting tool. Tool Series is the manufacturer's code for this tool. Number of teeth is the number of teeth/flutes. Total length is the length of the cutting tool. Cutting length is the distance between the bottom of the tool and the spindle (spindle is the coordinates of the spindle as measured by the monitoring system and not necessarily the centre of the physical spindle).

New Tool Parameters :

Tool Name :

Tool Type :

Tool Series :

Number of teeth :

Total length :

Cutting length :

Diameter :

Figure 75 New cutting tool form

10. Click save to save the new cutting tool in the database. The new tool should now appear in the list of available cutting tools.

After all cutting tools have been defined, the tools should be loaded into the VMC carousel. The user needs to load only the tools that are being used by the part program.

11. Click on Libraries > Carousel and click the "New" button repeatedly to create new pockets in the virtual carousel. The virtual carousel should have the exact number of pockets as the physical one.

Carousel

ToolId	Position	Description
0	1	
1	2	toolName : toolType : tool... ▾
0	3	toolName : toolType : toolSeries
0	4	toolName : EndmilltoolType : End
0	5	
0	6	
0	7	

New

Delete

Figure 76 Carousel management panel

- Mount each virtual cutting tool to the corresponding pocket by clicking in the relevant cell in the Description column and selecting the cutting tool. The pocket number should correspond to the physical pocket number since the monitoring system will use the same numbering to specify which cutting tool is being used.
- Click on “Libraries > Materials” and click “New”

New Material Parameters :

Material Name :

Torque Factor :

Save

Figure 77 New material form

- Specify a material name and click “Save”. Ignore the Torque Factor field since this is a placeholder for future functionality to facilitate material categorisation. The new material should now appear in the list of available billet materials
- Click on “Libraries > Billets” and click “New” button.
- In the description field, add a unique name for the billet. Then select the billet shape from the dropdown list and depending on the shape selection specify the relevant dimensional parameters and the billet material. If a complex shape is selected, then the billet is constructed by merging the volumes of billets that already exist in the library. Complex billets can also be merged to create a new billet.

Billet (mm)

Description:

Shape:

X min: X max:

Y min: Y max:

Z min: Z max:

Material:

Figure 78 Specification of rectangular billet

17. Click “Save” and the new billet should appear in the list of available billets.

Billet Library :

Billetid	Description	Xmin	Xmax	Ymin	Ymax	Zmin	Zmax
1	Demo Bottom Part	0.0	50.0	0.0	50.0	0.0	20.0
2	Demo Top Part	0.0	15.0	0.0	15.0	20.0	30.0
3	Demo Part	0.0	50.0	0.0	50.0	0.0	30.0

Figure 79 List of available billets including a complex billet (Demo Part)

A.4 SIMULATION FUNCTIONALITIES

After the libraries have been filled the simulator has the minimum required information to run and analyse a part program or a monitoring data file. It is recommended that before running a simulation the monitoring data file is checked through the simulator's data explorer. This step helps to understand the actual process and be in a better position to justify whether the simulator results are correct or if there have been mistakes when specifying the library elements (e.g. wrong dimensions, or wrong billet positioning). Therefore, to simulate the part program the next steps are:

18. Select Simulation > Data and click browse to select a monitoring data file.

The data file should be in CSV format and only the columns with numerical data can be processed. If the datafile is read successfully its column names will appear as parameter names to select from.

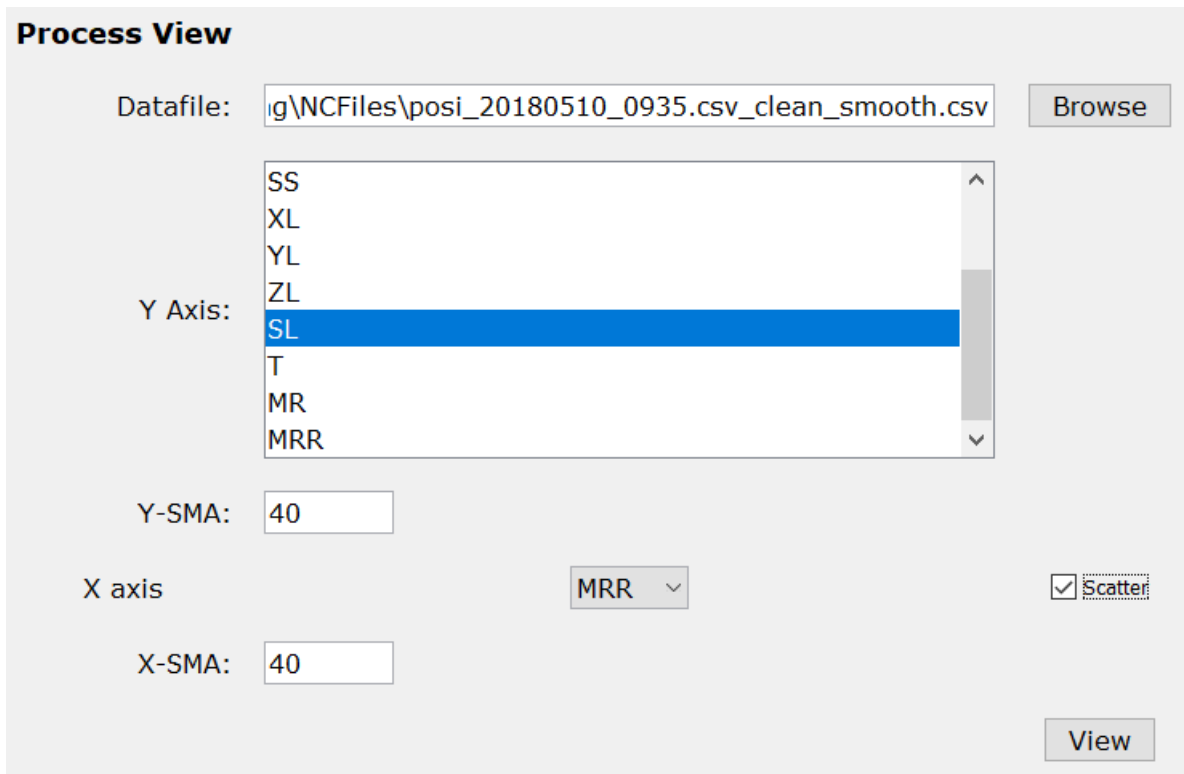


Figure 80 Parameter and options selection for data visualisation

19. Select the parameter of interest from the Y Axis list. Multiple parameters can be selected by holding down Ctrl key. Then select versus which parameter would you like to plot. If none is selected in the X axis dropdown selector then the Y Axis parameter is plotted versus sample number. The SMA options are to apply a simple moving average on the parameter values to smooth them. The value represents the moving average window. There is also an option to select a scatter chart instead of the line chart which is the default functionality. A scatter chart is recommended when the relationship between two parameters should be classified based on the stage of the process.
20. After the data is processed, the GUI moves automatically to the Simulator > Results section to display the results. The result has 3 main parts. The top is a double sliding bar which allows the user to view only a range of samples and therefore focus on details of the dataset. The second section is a 3D depiction of the toolpath with colour coding showing the value of the Y Axis parameter. The lowest value is depicted in blue colour, the highest with red, values between minimum and maximum are displayed by mixing blue and red accordingly and zero values are depicted in grey colour.

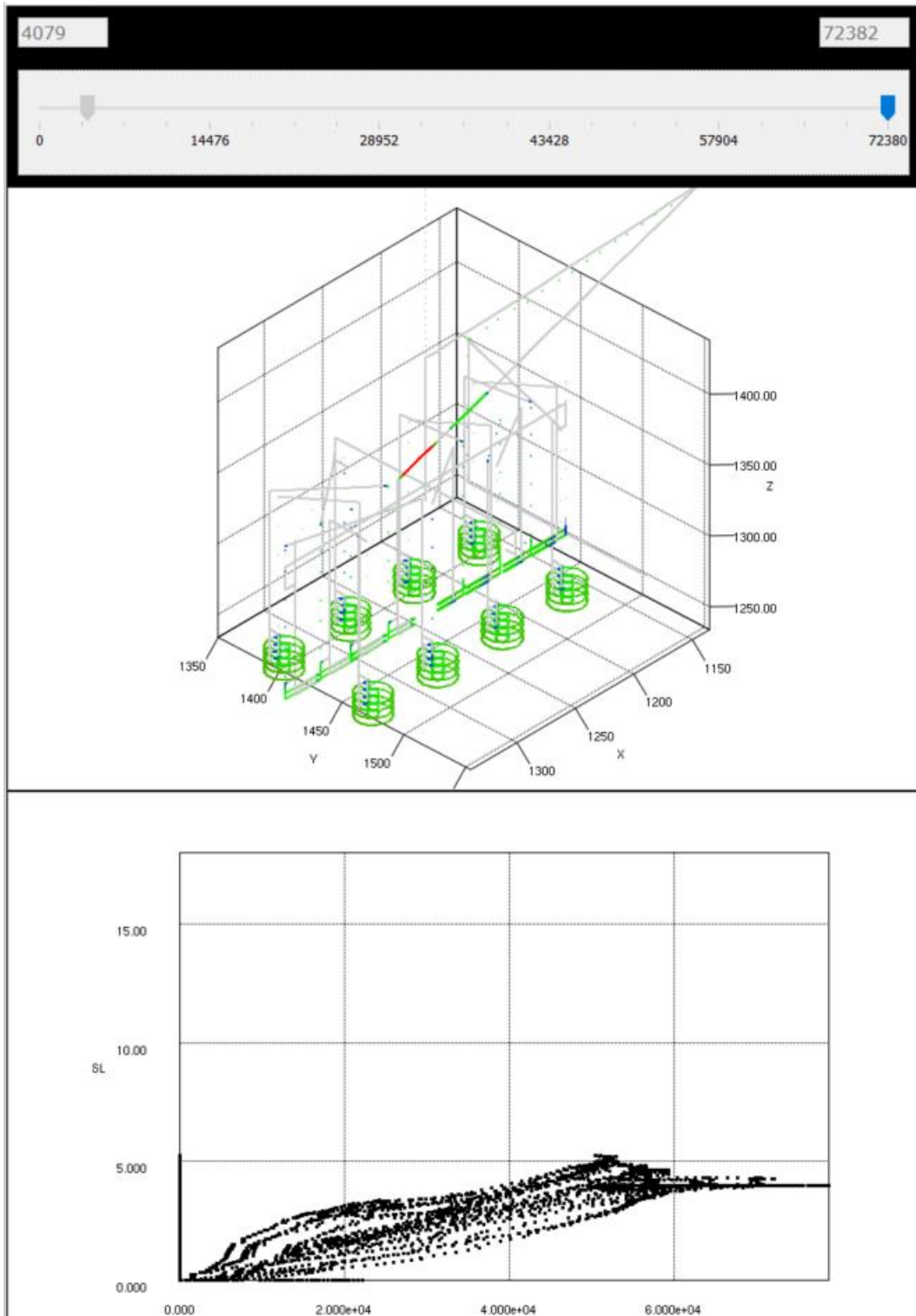


Figure 81 Data analysis results panel

21. To simulate either a part program or a monitoring data file click on “Simulation > Control”. In the simulator setup panel select the G-Code part program or a monitoring .csv file to use as input for the simulator. The type of file is recognised automatically. Then from the Billet dropdown menu

select the billet to be machined. Finally, select the way that the results should be displayed. 3D part shows the final part after virtual machining and 2D graphs are plots of each calculated parameter. Click “Run” for the simulation to begin.

The screenshot shows a software interface with two main sections: 'Input to Machine' and 'Configuration'. In the 'Input to Machine' section, there is a 'CSV file' dropdown menu, a text input field containing the path 'C:\Users\Alexo\Desktop\Milling\NCFiles\Cylinders 0.25.db.csv', and a 'Browse' button. Below this is a 'Billet:' dropdown menu with 'Cylinder V.2' selected. The 'Configuration' section includes an 'Element size:' input field with '0.25', a unit selector set to 'mm', and two checkboxes: 'Show 3D part:' (checked) and 'Show 2D graphs:' (checked). A 'Run' button is located at the bottom right of the configuration area.

Figure 82 Simulation setup

22. While the part program is being simulated, the simulator GUI switches to the log section so the user can see progress both by the progress bar on the top and the information displayed in the text area.

The screenshot shows the 'Machine Log' section of the simulator. At the top right, there is a 'Clear' button and a green progress bar. The log text area contains the following information: 'No Selection', 'Cleaning CSV Data File...done', 'Interpolating toolpath points...done', 'Parsing CSV Data File...done', 'Preprocessed CSV Data file contains 44987 samples.', 'Calculating Material Removal Parameters...Simulation error: Carousel pocket 30.0 is empty or c', 'Total elements: 897 x 537 x 85', 'Machined elements (mr):10834120', 'Elements machined: 10834120 Non machined: 30109445', 'done', 'Adding MRR to analysis file', and 'File lines: 44988 Column lines: 44988'. A vertical scrollbar is visible on the right side of the log text area.

Figure 83 Log section while running simulation

23. After the simulation finishes successfully the output files are generated in the same folder as the input file and the relevant 2D and 3D graphs are displayed (depending on the selections in Figure 82)

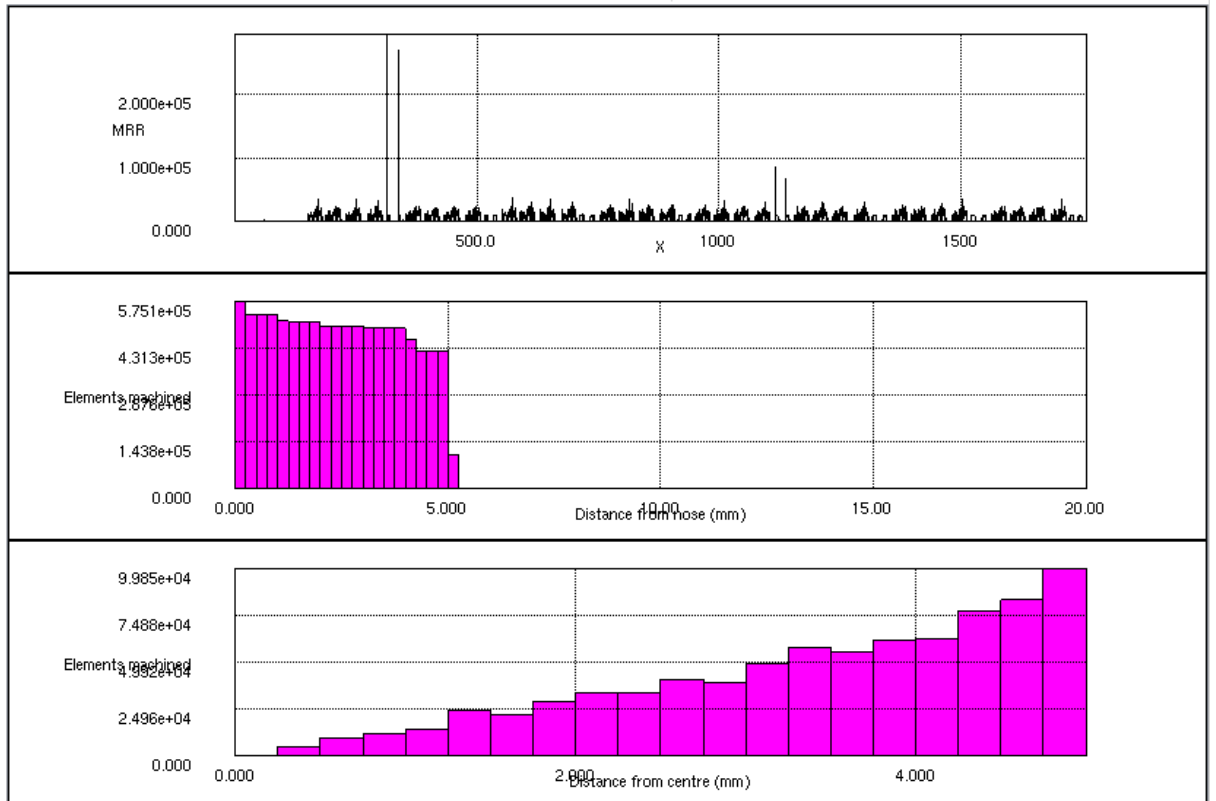


Figure 84 2D results

24. The output .csv files contain the values of the calculated parameters therefore the user can open these files with the data explorer and analyse the results.

A.5 LEARNING FUNCTIONALITY

The learning module can be run manually to create new models or update existing ones. In the current version and due to issues with AutoWEKA integration, the simulator is using the ensemble model presented in section 5.2. To train a new model:

25. Click on “Learning > Train” to open the model training panel. Select the material of the billet and the output .csv data file. If the file is read successfully, the inputs and targets text areas will be filled with the output file parameters.
26. Select one or more inputs and outputs by holding down the Ctrl key and clicking on the parameter name. Then click on “Train” button. The current

version does not support models with multiple output parameters so if two output parameters are selected then two models will be created (or updated if they already exist). The models are stored in the database and their existence can be verified by running the simulator again and checking the output file for new parameters. If the parameter name is X then the model calculated one is added to the file as X_theor.

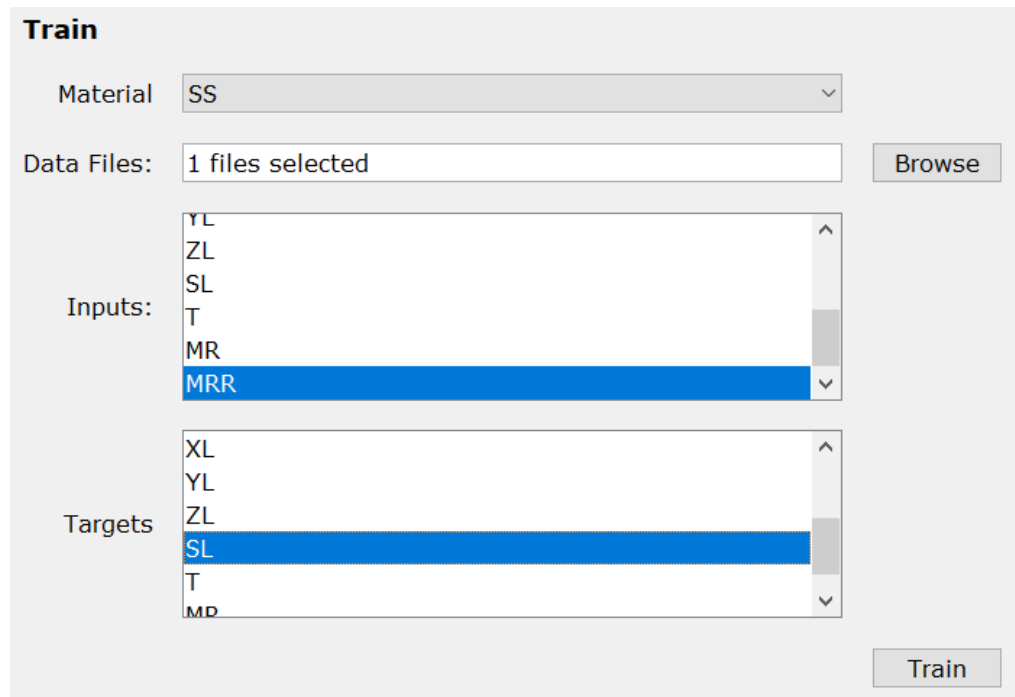


Figure 85 Model training panel

27. To compare the values of a parameter from two different output .csv files click on “Learning > Compare”. The simulator GUI will switch to the compare panel.
28. Click on browse buttons to select the two output files to compare. If both files are loaded successfully the Sync Params field will be filled with the common parameters. Select the parameter(s) that the simulator will use to synchronise the two datasets. Then select the parameter to compare from the Target Param dropdown menu. There is an option to show only the difference between the two datasets otherwise the values of each dataset will be displayed side by side. Finally, select the simple moving average window to smooth the datasets by filling in the Smoothen field.

Files to compare

File 1: CSV file

File 2: CSV file

Sync Params:

- FR
- SL
- XL
- YL
- ZL
- X
- Y
- Z

Target Param:

Only difference:

Smoothen:

Figure 86 Dataset comparison setup panel

The GUI will switch to the results section to display the generated plots. Due to the significant programming effort required to implement the n-sample-learning functionality, the current software version does not support n-sample-learning learning. However, the intention is to add a train button in the compare panel to create n-sample-learning models based on the differences identified between the two datasets.

Appendix B. JAVA CODE DOCUMENTATION SUMMARY

The Java, HTML and JavaScript code written for the purposes of this work will be submitted together with the thesis. The code itself is 14,688 lines therefore it is not practical to add it in the appendices. However, to facilitate understanding the Javadoc summary is presented below which covers the backend of the simulator.

B.1 PERSISTENCE LAYER (MILLING-DATABASE)

The software packages of this layer are responsible for Create Retrieve Update Delete (CRUD) transactions with the database. This layer is the interface between the database and the rest of the application so any changes in the database affect only this layer and not the application as a whole.

Package `uk.ac.cf.milling.db`

```
package uk.ac.cf.milling.db
```

Contains classes to initialise, connect and manage the database.
Class names typically correspond to the names of the database tables.

Author:

Theocharis Alexopoulos

Class Summary

Class	Description
BilletDB	Contains CRUD methods to manage billet information kept in the database.
CarouselDB	Contains CRUD methods to manage carousel data in the database
CuttingToolDB	Contains CRUD methods to manage cutting tools in the database
CuttingToolProfileDB	Contains CRUD methods to manage cutting tool profiles files in the database
DB	Basic class for connecting and initialising the database.
LearningSetDB	Contains CRUD methods to manage machine learning models in the database
MaterialDB	Contains CRUD methods to manage materials in the database
NcDB	Contains CRUD methods to manage NC files in the database
SettingsDB	Utility class to retrieve or update the settings of the application Add or remove a setting is not allowed as everything is created during database creation

Package uk.ac.cf.milling.objects

package uk.ac.cf.milling.objects

Contains classes for custom made objects to load data from database on and use throughout the application

Author:

Theocharis Alexopoulos

Class Summary

Class	Description
Billet	This is the class to create the initial billet that will be machined in order to create the part
CuttingTool	Holds all information related to a cutting tool and contains methods related to cutting tool management and manipulation
CuttingToolProfile	Holds parameter values for the surface profile of the cutting tool.
KPIs	Holds values for virtual process KPIs and temporary stores simulation run information that is required to do the simulation and report results.
LearningSet	Holds the parameter values for the trained model.
Material	Holds the parameter values for billet materials.
Nc	Holds the parameter values for NC files and the generated analysis files or reports.
SettingsSingleton	Holds the application global settings and is also used to store objects available to multiple parallel threads.
SimulatorConfig	Contains simulator configuration parameters and corresponding getter and setter methods.

B.2 UTILITIES LAYER (MILLING-UTILS)

The software packages in this layer are the main part of the application's backend. They facilitate the communication between the frontend and the database and carry out the vast majority of computational tasks related to VMC simulation.

Package uk.ac.cf.milling.utils.data

package uk.ac.cf.milling.utils.data

Contains classes related to data transformation and manipulation

Author:

Theocharis Alexopoulos

Class Summary

Class	Description
ConvertUtils	Data type conversion - convenience methods.
DataManipulationUtils	Core class for data processing required by input module.
GCodeAnalyser	Custom G-Code translation methods.
GCodeAnalyserUGS	Adaptor methods to connect UGS library to the simulator's input module
IoUtils	Convenience methods for reading and writing csv files and reports to disk

Package uk.ac.cf.milling.utils.webapp

package uk.ac.cf.milling.utils.webapp

Utilities to support web application functionality

Author:

Theocharis Alexopoulos

Class Summary

Class	Description
MonitoringUtils	Methods for real time response to web application requests.

Package uk.ac.cf.milling.utils.db

```
package uk.ac.cf.milling.utils.db
```

Classes for handling high level database related utilities.
These are the main way to communicate with the persistence layer.

Author:

Theocharis Alexopoulos

Class Summary

Class	Description
BilletUtils	Methods accessing persistence layer for Billet creation and management.
CarouselUtils	Methods accessing persistence layer for Carousel pocket creation and management.
CuttingToolProfileUtils	Methods accessing persistence layer for CuttingToolProfiles creation and management.
CuttingToolUtils	Methods accessing persistence layer for CuttingTool creation and management.
DatabaseUtils	High level access to database creation.
DBUtils	High level access to database initialisation.
LearningSetUtils	Methods accessing persistence layer for ML model creation and management.
MaterialUtils	Methods accessing persistence layer for Material creation and management.
NcUtils	Methods accessing persistence layer for Nc object creation and management.
SettingUtils	Methods to read and set simulator settings.

Package uk.ac.cf.milling.utils.learning

```
package uk.ac.cf.milling.utils.learning
```

Contains machine learning related classes for model building and data synchronisation.

Author:

Theocharis Alexopoulos

Class Summary

Class	Description
DataSynchronisation	Core methods to synchronise data with DTW
Models	Contains utility methods to create, train and manage ML models.

Package uk.ac.cf.milling.utils.plotting

package uk.ac.cf.milling.utils.plotting

Contains current and legacy classes for plotting data and results.

Author:

Theocharis Alexopoulos

Class Summary

Class	Description
ChartXYZZoom	Adds 3D zoom functionality to 3D graphs. Integration to 2D graphs (for X,Y zoom) not tested.
ChartXZoomController	Adds X axis zoom functionality (useful in 2D graphs).
ColourScatter3D	Methods for advanced multicolour scatter plotting.
Plotter2D	A frame to show a list of charts
Plotter3D	Top level management methods for plotting 3D charts.
Plotter3D_V1	3D plotter version 1. Produces a scatter chart showing all part elements in black colour.
Plotter3D_V2	3D plotter version 2. Produces a plot showing the surfaces of the part.
Plotter3D_V3	3D plotter version 3. Produces a point cloud showing the surfaces of the part.
Plotter3D_V4	3D plotter version 4. Produces a wire diagram showing the edges of the part.
Plotter3DBackup	Contains legacy plotting methods used up to version 0.4 of the simulator.
PlotterUtils	Methods for registering and de-registering charts in the results panel.
PlotUtilsold	Legacy methods to reproduce PhD initial graphical representations.

Package uk.ac.cf.milling.utils.simulation

```
package uk.ac.cf.milling.utils.simulation
```

Simulator engine and simulation run management classes.

Author:

Theocharis Alexopoulos

Class Summary

Class	Description
HardwareCheckUtils	Under development! Methods to check hardware specs before simulator initialisation.
KPIUtils	Methods to maintain process performance and statistics and to provide required data for the output interface reports.
MRRCalculator	The implementation and support methods for the simulation engine.
ProfileBuilderUtils	Methods to generate the cutting tool profile.
SimulatorUtils	Top level methods to start a simulator run.

Package uk.ac.cf.milling.utils.test

```
package uk.ac.cf.milling.utils.test
```

Application debugging and testing utilities

Author:

Theocharis Alexopoulos

Class Summary

Class	Description
AutoWekaTest	Independently running tests for AutoWEKA. Multiple integration issues with latest version of WEKA.
Benchmarker	Methods to test simulator's execution time.
DataCompare	Methods to debug data synchronisation issues. The relation between each point connection is printed to a file.
DuplicatePositionMerger	Verification tests for dealing with duplicate monitoring data.
WekaTest001	Thesis results verification tests.
WekaTest002	Thesis results verification tests.
WekaTest003_Size	Thesis results verification tests.

B.3 GRAPHICAL USER INTERFACE – LOCAL (MILLING-VM)

The packages presented below create the GUI of the local application. They are the most complete interface for the user to supply the simulator with input data, run it and view the results in the form of graphs, logs, reports or simple exported files. The development is based on Java Swing and AWT elements.

Package uk.ac.cf.milling.app

`package uk.ac.cf.milling.app`

Contains classes required to start the simulator interface.

Author:

Theocharis Alexopoulos

Class Summary

Class	Description
MillingMachineGUI	This is the starting point for running the simulator. Initialises the local graphical interface.

Package uk.ac.cf.milling.graphs

`package uk.ac.cf.milling.graphs`

Contains builders for graphs panels.

Author:

Theocharis Alexopoulos

Class Summary

Class	Description
ProcessGraphsPanel	Constructs the panels for 2D and 3D graphs.

Package uk.ac.cf.milling.gui

`package uk.ac.cf.milling.gui`

Top level management of GUI components and architecture.

Author:

Theocharis Alexopoulos

Class Summary

Class	Description
ButtonTabComponent	The tab component mainly for navigating around results.
CustomOutputStream	This class extends from <code>OutputStream</code> to redirect output to a <code>JTextArea</code>
DefaultPanelElements	Convenient methods that facilitate reuse of GUI elements.
GUIBuilder	Initialises the GUI, builds its skeleton and triggers specialised builders to create the panels that the user sees.
MenuBar	Builder for the menu bar at the top of the GUI.

Package uk.ac.cf.milling.gui.file

`package uk.ac.cf.milling.gui.file`

Classes to build the database selection panel.

Author:

Theocharis Alexopoulos

Class Summary

Class	Description
DatabasePanel	Builds the database selection or creation panel.

Package uk.ac.cf.milling.gui.help

`package uk.ac.cf.milling.gui.help`

Contains classes to build the help section of the local application.

Author:

Theocharis Alexopoulos

Class Summary

Class	Description
<code>HelpPanel</code>	Builds the help section panel.

Package uk.ac.cf.milling.gui.learning

`package uk.ac.cf.milling.gui.learning`

Contains classes to manage and present ML related functionality.

Author:

Theocharis Alexopoulos

Class Summary

Class	Description
<code>ComparePanel</code>	Builds the data synchronisation and comparison panel.
<code>DataConnectionPanel</code>	Builds the NC and analysis data connection panel.
<code>TrainPanel</code>	Builds the train model panel.

Package uk.ac.cf.milling.gui.library

`package uk.ac.cf.milling.gui.library`

Panel builders for libraries submenu.

Author:

Theocharis Alexopoulos

Class Summary

Class	Description
BilletLibraryPanel	Builds billets list panel.
CarouselPanel	Builds carousel setup map panel.
MaterialLibraryPanel	Builds materials list panel.
NewBilletPanel	Builds new billet form panel.
NewMaterialPanel	Builds new material form panel.
NewToolPanel	Builds new cutting tool form panel.
ToolLibraryPanel	Builds cutting tool list panel

Package uk.ac.cf.milling.gui.simulation

`package uk.ac.cf.milling.gui.simulation`

Builds panels for simulation submenu.

Author:

Theocharis Alexopoulos

Class Summary

Class	Description
ControlPanel	Builds the panel that contains simulator setup and run functions.
MachineLogPanel	Builds the panel that displays and holds the logs of simulator.
ProcessPanel	Builds the process and data analysis panel.
ResultsPanel	Builds the tab based panel that shows graphs and simulation reports.

Package uk.ac.cf.milling.tests

```
package uk.ac.cf.milling.tests
```

Contains independent implementations of customised GUI elements.

Author:

Theocharis Alexopoulos, Oracle and/or its affiliates

Class Summary

Class	Description
CustomComboBoxDemo	Sample code for customised combo-boxes.
TableRenderDemo	Sample code to integrate comboboxes into a table.

Package uk.ac.cf.milling.utils.runnables

```
package uk.ac.cf.milling.utils.runnables
```

Contains runnables that create threads for simulation processes that take too long and would otherwise freeze the local application.

Author:

Theocharis Alexopoulos

Class Summary

Class	Description
MLCompareRunnable	Starts the 2 dataset synchronisation and comparison process and fills the results panel tab with the generated graphs.
MLTrainRunnable	Start the ML model creation and training process.
SimulateProcessRunnable	Starts the simulation engine with data supplied through the simulator control panel.
ViewProcessRunnable	Starts the process and data analysis that fills the corresponding panel with graphs and data.

B.4 GRAPHICAL USER INTERFACE – WEB APPLICATION (MILLING-TWIN)

Finally, the web-based user interface with the simulator is a digital twin prototype which is mainly written in HTML and JavaScript. The Javadoc produced contains only the required classes to handle the requests and responses of the web

application. The details for the front-end functionality can be viewed in the supplied code files.

Package `uk.ac.cf.twin.milling.servlets`

```
package uk.ac.cf.twin.milling.servlets
```

Contains classes required to manage digital twin connections with the simulator.

Author:

Theocharis Alexopoulos

Class Summary

Class	Description
<code>ServletMain</code>	Handles all requests and responses of the digital twin.
<code>UserLoggedFilter</code>	Filters the requests to ensure that user is logged in.