Reinforcement Learning-based Autonomous Robot Navigation and Tracking

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

Feiqiang Lin

December 2023

School of Engineering
Ysgol Peirianneg

Abstract

Autonomous navigation requires determining a collision-free path for a mobile robot using only partial observations of the environment. This capability is highly needed for a wide range of applications, such as search and rescue operations, surveillance, environmental monitoring, and domestic service robots. In many scenarios, an accurate global map is not available beforehand, posing significant challenges for a robot planning its path. This type of navigation is often referred to as Mapless Navigation, and such work is not limited to only Unmanned Ground Vehicle (UGV) but also other vehicles, such as Unmanned Aerial Vehicles (UAV) and more. This research aims to develop Reinforcement Learning (RL)-based methods for autonomous navigation for mobile robots, as well as effective tracking strategies for a UAV to follow a moving target.

Mapless navigation usually assumes accurate localisation, which is unrealistic. In the real world, localisation methods, such as simultaneous localisation and mapping (SLAM), are needed. However, the localisation performance could deteriorate depending on the environment and observation quality. Therefore, To avoid deteriorated localisation, this work introduces an RL-based navigation algorithm to enable mobile robots to navigate in unknown environments, while incorporating localisation performance in training the policy. Specifically, a localisation-related penalty is introduced in the reward space, ensuring localisation safety is taken into consideration during navigation. Different metrics are formulated to identify if the localisation performance starts to deteriorate in order to penalise the robot. As such,

the navigation policy will not only optimise its paths in terms of travel distance and collision avoidance towards the goal but also avoid venturing into areas that pose challenges for localisation algorithms.

The localisation-safe algorithm is further extended to UAV navigation, which uses image-based observations. Instead of deploying an end-to-end control pipeline, this work establishes a hierarchical control framework that leverages both the capabilities of neural networks for perception and the stability and safety guarantees of conventional controllers. The high-level controller in this hierarchical framework is a neural network policy with semantic image inputs, trained using RL algorithms with localisation-related rewards. The efficacy of the trained policy is demonstrated in real-world experiments for localisation-safe navigation, and, notably, it exhibits effectiveness without the need for retraining, thanks to the hierarchical control scheme and semantic inputs.

Last, a tracking policy is introduced to enable a UAV to track a moving target. This study designs a reward space, enabling a vision-based UAV, which utilises depth images for perception, to follow a target within a safe and visible range. The objective is to maintain the mobile target at the centre of the drone camera's image without being occluded by other objects and to avoid collisions with obstacles. It is observed that training such a policy from scratch may lead to local minima. To address this, a state-based teacher policy is trained to perform the tracking task, with environmental perception relying on direct access to state information, including position coordinates of obstacles, instead of depth images. An RL algorithm is then constructed to train the vision-based policy, incorporating behavioural guidance from the state-based teacher policy. This approach yields promising tracking performance.

Acknowledgements

I would like to thank all those people and organisations who have supported and helped me during my PhD project.

I would like to give my heartfelt thanks to Dr. Ze Ji, who has fully supported my entire PhD project. I do not think I will make it here without his help and guidance.

I would like to thank my whole family. They have not pressured me but always support whatever I decide to do.

I also appreciate the friends and colleagues I have met here in Cardiff. We have had many happy times together.

Finally, thanks to the China Scholarship Council for providing me with stipends for four full years.

Contents

List of Figures

List of Tables

List of Algorithms

List of Acronyms

UGV Unmanned Ground Vehicle

UAV Unmanned Aerial Vehicles

DL Deep Learning

RL Reinforcement Learning

SLAM Simultaneous Localisation and Mapping

MC Monte-Carlo

TD Temporal-Difference

DNN Deep Neural Network

CNN Convolutional Neural Network

DQN Deep Q-Network

LSTM Long Short-Term Memory

MPC Model Predictive Controller

MDP Markov Decision Process

POMDP Partially Observable Markov Decision Process

PPO Proximal Policy Optimisation

TRPO Trust region policy optimisation

VO Visual Odometry

List of Publications

The work introduced in this thesis is based on the following publications:

- Lin, F., Ji, Z., Wei, C. and Niu, H. 2021. Reinforcement learning-based mapless navigation with fail-safe localisation. Presented at: 21st Towards Autonomous Robotic Systems Conference (TAROS 2021), Virtual and Lincoln, UK, 8-10 September 2021

- Lin, F., Ji, Z., Wei, C. and Grech, R. 2022. Localisation-safe reinforcement learning for mapless navigation. Presented at: IEEE International Conference on Robotics and Biomimetics (IEEE ROBIO 2022), Jinghong, China, 5-9 December 2022

- Lin, F., Wei, C., Grech, R. and Ji, Z. 2024. VO-safe reinforcement learning for drone navigation. Presented at: 2024 IEEE International Conference on Robotics and Automation (ICRA), 13-17 May 2024.

Chapter 1

Introduction

1.1 Background and motivation

Mobile robots come in various forms designed to operate across diverse domains such as land, air, underwater, and surface. Widely deployed in numerous industries and our daily lives, these robots serve a broad range of applications, including domestic service, scientific surveys, military defence, and aeronautic and maritime operations. Over decades of research and development, mobile robots have gained significant attention, leading to advancements in unmanned and autonomous capabilities. Those developments are highly desirable as mobile robots are well-suited to undertake tasks deemed tedious or risky for human beings.

The following subsections discuss the motivations and the challenges of Reinforcement Learning (RL)-based algorithms proposed by this thesis regarding robot navigation and moving target tracking.

1.1.1 Autonomous robot navigation

Autonomous navigation is undoubtedly an essential and fundamental ability for robots across a range of tasks, such as warehouse automation with Unmanned Ground Vehicles (UGVs) [1] or Unmanned Aerial Vehicles (UAVs) for power line

inspection [2]. Built upon this, robots can carry out various complex tasks. Methods for mobile robot navigation can be generally divided into two groups: non-learning-based and learning-based [3].

While non-learning-based navigation methods have demonstrated success on numerous occasions, they do come with certain limitations. One prominent constraint is the necessity for an explicit environment map. This requirement becomes challenging and may lead to unexpected failures in scenarios where no map is available, as seen in search and rescue operations (SAR) or when dealing with highly complex and dynamic environments. Additionally, an overreliance on manually designed path planning can restrict the generalization capability of mobile robots for deployment in diverse environments [4]. Furthermore, these methods often demand accurate analytical dynamic models of mobile robots, posing challenges in cases such as UAVs with complex dynamics.

To address the limitations outlined above, the recent advancements in Deep Learning (DL) and RL offer solutions through learning-based navigation methods, which will be explored in Section 2.1. While reinforcement learning navigation has demonstrated significant success in various applications, the issue of localisation during navigation is often overlooked. Many works assume that a robot has access to its accurate position. However, in real-world applications, a Simultaneous Localisation and Mapping (SLAM) system, such as Lidar-based or Visual Odometry (VO), is typically employed for robot self-localisation, and the accuracy of localisation can be influenced by the robot's chosen actions. Such a SLAM system relies on environmental features as spatial references for self-localisation and mapping. An action that directs the robot straight towards the target position may lead to travelling in a featureless area, resulting in a localisation failure. For example, a robot using Lidar-based localisation algorithms will lose its position when it follows decision commands to travel into a long symmetric space, such as a corridor or tunnel [5]. Additionally, violent behaviours caused by UAV's complex dynamics during agile

flight may also result in localisation failure as this may introduce inconsistencies between consecutive observations [6]. The navigation policy should avoid making such decisions that may lead to localisation failures.

Navigation policies trained with ground truth pose as most of the methods proposed in the literature may lead to localisation failures when deployed in real-world applications which utilise SLAM algorithms as discussed above, contributing to the sim-to-real problem. This is because such policies may overlook localisation performance along the navigation trajectory if corresponding rewards are not provided during training. Additionally, RL-based policies trained in simulation environments face another aspect of the sim-to-real problem, as observations such as images from simulators can significantly differ from real-world observations.

Problem 1: The above challenge serves as **one key motivation** of this project – to develop RL algorithms that train the navigation policies of mobile robots to be localisation-aware. The developed algorithms should empower the agent with the capability to navigate in an unknown environment, optimising not only its goal of reaching the target position with minimal time cost but also taking into account the distribution of environmental features for localisation fail-safe decision-making, which has not been taken care of in current research works. The agent relies solely on raw sensor information, such as laser scans or camera images, without any prior knowledge of the environment. For instance, robots carry out search and rescue tasks in structural disasters, such as fires or earthquakes, where environments are altered, on-site environment maps are unavailable, and the robots are only provided with the relative target position they need to reach. The robot will have to carry out localisation and navigation at the same time without getting lost in the environment.

1.1.2 Autonomous robot tracking

Besides navigation, cooperation is another essential capability for robots to achieve more complex tasks. Cooperation among multiple robots can achieve more than

what a single robot can accomplish. Such cooperation is evident in examples involving UAVs and UGVs. While air vehicles excel in tasks such as delivery, environmental exploration, and landscape photography, they face constraints, such as limited battery capacity due to their small sizes. Here, a ground vehicle can play a crucial role. UAVs can be recharged using power from ground vehicles, or ground vehicles can transport UAVs during parts of the journey, allowing the UAVs to conserve energy by not flying.

Additionally, ground vehicles can benefit from aerial vehicles, which can offer improved traffic and environmental information for ground vehicles as UAVs can provide a different view and often can look further, and in return, ground vehicles can enhance their route planning capabilities [7,8]. As demonstrated above, ground and aerial mobile robots can mutually benefit through cooperation. A fundamental aspect of achieving such cooperation is enabling the UAVs to autonomously track the ground vehicle. This capability is crucial for preparing landing operations or providing enhanced vision around the ground vehicle.

Current tracking strategies for UAVs are often based on trajectory planning, where the tracking task is decomposed into various sub-tasks executed sequentially. These components include sensing, mapping, planning, and trajectory optimisation [9]. However, this task decomposition has the potential to increase processing latency, involving time for computation and communication between different components. Moreover, it may result in a complex system that has the potential to accumulate errors throughout the pipeline [10].

RL algorithms have demonstrated significant capabilities in achieving complex UAV manoeuvres through end-to-end policies, resulting in a simple and straightforward system. In other words, such end-to-end control schemes allow a robot to map control signals directly from sensor inputs, such as depth images, with neural networks. With such a control structure, processing latency can be minimised. Also, it eliminates the need for environmental mapping, which is typically a challenging

problem [11]. However, there is a scarcity of research applying RL-based algorithms to tracking tasks in complex environments, which will be discussed in detail in the literature review chapter (Section 2.2).

Additionally, this thesis has found that training a tracking policy network from scratch using high-dimensional inputs, such as images, leads to local minima and low training efficiency. Consequently, this can result in catastrophic tracking outcomes, a topic that will be thoroughly explained in Chapter 6.

Problem 2: Hence, this constitutes **the second part of the project's motivation**: to train a UAV to autonomously track a moving target using a novel end-to-end RL training pipeline. The core challenge is to develop an innovative RL-based tracking policy that enables the UAV to keep the target vehicle centred in its camera's field of view while simultaneously avoiding obstacles in the environment. This is an essential function for scenarios, such as the above-mentioned search and rescue operations that are usually within complex and hazardous environments; UAVs need to track ground vehicles and also ensure collision avoidance.

Motivated by the above-mentioned challenges, this work will particularly focus on Unmanned Ground Vehicles (UGVs) and Unmanned Aerial Vehicles (UAVs). This does not only include research on single robot navigation but also the capability of target tracking that is highly demanded by tasks that involve multiple vehicles, such as using UGVs and UAVs cooperatively under the SAR scenario [12].

Note that, in this thesis, the term mobile robots will be interchangeably used with UGVs and UAVs in different contexts. Similarly, UAV, drone, and quadrotor all refer to aerial vehicles and are interchangeably used throughout the thesis.

1.2 Aim and objectives

This project aims to explore how a learning-based system can augment the capabilities of mobile robots. As outlined, several open and intriguing challenges persist.

Therefore, the project has a dual focus: 1) enhancing the navigation capabilities of a single mobile robot with a fail-safe localisation mechanism, and 2) achieving reliable autonomous tracking of a mobile vehicle by a UAV in environments with obstacles.

The specific objectives include:

- To develop new reinforcement learning reward spaces and training procedures to prevent mobile robot behaviours from causing localisation failures.
- To develop system structures (end-to-end or hierarchical) tailored to the dynamics and complexity of UGV and UAV and bridge the sim-to-real gap for policy trained in simulation to be applicable in the real world.
- To design a novel reinforcement learning reward space for UAV tracking, ensuring the target is centred in images without occlusion.
- To develop a novel RL framework for efficient training of vision-based UAV tracking policy.

1.3 Contributions

Based on the aim and objectives described above, this thesis has achieved the following accomplishments.

Localisation-safe mobile ground robot navigation:

- This thesis transforms the mapless navigation problem of ground vehicles from an inherent Partially Observable Markov Decision Process (POMDP) setting to a Markov Decision Process (MDP) setting. This is achieved by 1) terminating a training episode early, when localisation algorithms begin to diverge, and 2) reconstructing the state with historical information embedding obtained from the Long Short-Term Memory (LSTM) module.

- The thesis introduces a novel reward component to penalise mobile robots in the event of a localisation failure, a component that has not been discussed in previous works.

- A new training strategy is formulated to train the navigation policy that aims to not only reach the goal destination swiftly but also avoid moving into regions where localisation algorithms are susceptible to failure.

VO-safe vision-based UAV navigation:

- For vision-based navigation, to alleviate the sim-to-real problem introduced by unrealistic RGB images rendered by game-style training simulators, this thesis develops a high-level policy using semantic images instead of raw RGB images. In this way, the navigation policy can recognise localisation-unfriendly textures, such as water or trees, with limited training environments and episodes.

- To compensate for the undesired effect of the complex dynamics of UAVs on localisation performance, this thesis constructs a hierarchical navigation system for UAVs comprising an RL-based high-level policy and a conventional low-level policy. This structure leverages both the ability to comprehend high-level information of neural networks and the safety and stability provided by conventional controllers.

- The method proposed in this thesis enables a control policy trained in simulations to achieve vision odometry (VO)-safe navigation in real-world implementations without the need for retraining.

UAV target tracking:

- For the task of UAV target tracking, a novel reward space is introduced to penalise the UAV for cases when 1) the target is located at the edge of the observed image, 2) occluded by obstacles, and 3) outside of the defined distance ranges.

- Training a vision-based tracking policy from scratch using depth images tends to lead to local minima. To address this issue, a novel teacher-student training strategy is introduced, where the teacher is a state-based policy with accurate coordinates and radius information of obstacles, as opposed to depth images. Training with such state information enables policy training to converge to promising performance more efficiently. Importance sampling is employed to reformulate the RL training loss function, allowing training data from the state-based policy to guide the vision-based student policy. Consequently, the trained vision-based policy can achieve promising tracking performance and avoid local minima during training.

This research has significant implications for deploying RL-based navigation in real-world applications, as this work mitigates the assumption of ground-truth localisation of the robot. Prioritising localisation safety in navigation strategies for both ground and aerial vehicles significantly enhances the safety of autonomous mobile robots. The proposed RL-based tracking policies not only improve the efficiency and reliability of mobile robot operations but also simplify the overall system framework. These algorithms can be readily integrated into existing robot platforms and act as fundamental components for a variety of tasks, including surveillance, search and rescue, and environmental monitoring.

1.4 Outline

This section outlines the thesis's content, consisting of seven chapters.

Chapter 2 introduces relevant literature on learning-based navigation, control, and target tracking of UAVs, followed by an overview of key concepts in reinforcement learning and important RL algorithms in **Chapter 3**.

Chapter 4 introduces a localisation-safe navigation method for ground vehicle nav-

igation in both indoor and outdoor environments. To adapt such localisation-safe navigation concepts and training methods to vision-based UAVs, which have more complex dynamics and sensors, **Chapter 5** introduces a hierarchical control framework that combines learning-based and conventional control strategies to achieve VO-safe UAV navigation.

After developing localisation-safe navigation algorithms for individual ground and aerial robots, to establish the capability of multiple vehicle cooperation, **Chapter 6** presents an end-to-end learning framework to train a UAV to track a moving target.

Chapter 7 concludes the thesis. The limitations and challenges of the approaches developed in this thesis are discussed, and future research directions are proposed.

Chapter 2

Literature Review

This chapter will present a detailed literature review on the topics of this thesis.

Section 2.1 will provide an in-depth discussion of navigation methods developed in the literature. This section will start with a brief overview of conventional navigation methods and their limitations. Subsequently, it will delve into learning-based algorithms, encompassing both deep learning and reinforcement learning approaches. Four distinct topics will be presented regarding the implementation of RL-based algorithms, covering navigation that combines RL with classical control, uncertainty-aware RL-based navigation, RL-based exploration, and RL for multi-agent navigation.

In Section 2.2, special attention will be dedicated to learning-based UAV control and navigation algorithms. This focus arises from the complex dynamics of UAVs, necessitating additional considerations in control system design. The section will comprehensively introduce DL-based algorithms, RL-based methods, techniques to optimise RL policies for optimum performance, approaches that integrate RL with Model Predictive Controller (MPC), and methodologies for UAV modelling.

Building upon the foundational UAV control methods outlined in Section 2.2, Section 2.3 will delve into the specifics of autonomous target tracking by UAVs, aligning with the second focus of this thesis. The section will start by introducing

traditional non-learning-based approaches, encompassing both control-based and trajectory planning-based methodologies. Subsequently, it will discuss learning-based (mainly RL-based) methods that have been explored in the literature for addressing tracking challenges.

Limitations of methods found in the literature will be discussed in Section 2.4. These will formulate the research gaps that this thesis tries to bridge. And, the chapter will be concluded in Section 2.5.

2.1 Navigation by learning-based methods

The methods employed for mobile robot navigation can be broadly categorized into two main groups: non-learning-based and learning-based.

In the realm of non-learning-based approaches, critical components include sensing, avoidance, and path planning. For instance, the identification of obstacles can be accomplished through techniques such as optical flow [13] or Lidar-based [14], facilitating obstacle avoidance. Additionally, rapid exploration random tree algorithms (RRT) [15] are instrumental in designing effective path-planning strategies. Another noteworthy approach involves the use of SLAM [16]. Such a method entails running the SLAM algorithm to construct an accurate map of the environment. Subsequently, path planning methods like RRT or A^* [17] can be seamlessly integrated, enabling obstacle-free navigation based on the constructed map [3].

While non-learning-based methods have demonstrated success in various applications, a notable limitation is their reliance on explicit path planning. This requirement introduces challenges, particularly in scenarios where no map is available for an unfamiliar environment, as is often the case in search and rescue scenarios [18]. In such instances, or when dealing with exceedingly complex and dynamic environments, the necessity for explicit path planning may lead to unexpected failures. Additionally, an over-reliance on manually designed path planning [19] has the po-

tential to curtail the generalization capabilities of mobile robots, restricting their adaptability to diverse environments [3]. It is crucial to note that purely geometric intermediate representations may fall short of capturing the nuanced navigation affordances [4].

As stated above, traditional non-learning-based methods do exhibit certain limitations. Those limitations may be overcome by learning-based algorithms [20] [21]. The subsequent subsections present the discussion of learning-based navigation methods including DL-based algorithms and RL-based algorithms. As this thesis will focus on RL algorithms, it will discuss a bit more on RL learning methods.

2.1.1 DL-based navigation

Harnessing the powerful capabilities of neural networks [22], deep learning has emerged as a valuable tool in instructing mobile robots through expert demonstrations. A prevalent approach involves the establishment of an end-to-end pipeline, wherein raw sensor inputs are mapped directly to driving commands. Research works [23] [24] have introduced supervised learning methods trained with human demonstrations using Convolutional Neural Network (CNN) [25]. This enables robots to produce driving commands based solely on depth images or Lidar data, facilitating navigation in obstacle environments without the need for prior environment mapping (referred to as mapless navigation).

In a different vein, Kanezaki et al. [26] incorporate local maps as inputs to CNNs, generating the next move based on demonstrations conducted by a traditional planning algorithm A^*. The core innovation of this approach lies in the manipulation of obstacle maps through cropping, rotation, and rescaling, contingent on the agents' current and goal locations. This adaptive transformation enhances the robot's capability to handle a broader spectrum of environments, extending beyond those encountered during training. Furthermore, a map containing the robot's movement history is supplied, mitigating the risk of the robot getting stuck in local minima

and thereby reducing the probability of failures during navigation.

Supervised learning algorithms often entail training robots in virtual environments owing to the substantial data requirements. However, challenges arise when deploying these networks in real-world applications [27]. Tai et al. [28] address this issue by proposing an algorithm that, during the testing phase with real-world inputs, translates the input images back to the training world domain using cycleGAN. This translation ensures that the networks make more informed decisions within the familiar training domain. The consideration of uncertainty is a key aspect of this approach, enhancing the safety of driving commands. Similarly, Choi et al. [29] tackle uncertainty stemming from the lack of training data and measurement noise. Their approach involves the utilization of a mixture density network to address and account for uncertainties in the model, contributing to more robust decision-making in real-world scenarios.

While end-to-end proposals may appear straightforward, there is a high risk associated with entrusting all tasks to a single neural network. Navigation tasks can be effectively divided into distinct sub-tasks, with crash avoidance emerging as a critical component. Convolutional neural networks can be adeptly trained to predict crash probabilities based on sensor inputs. However, acquiring real-world crash data presents a formidable challenge. Lee et al. [30] address this issue by leveraging the video game GTA V to collect training data. The trained model, thus enriched with information from the virtual environment, can decide how collision probabilities are influenced by wheel angles, vehicle orientations, and distances. This innovative approach provides valuable insights into crash avoidance strategies in situations where real-world crash data is challenging to obtain.

In addition to accounting for the geometric aspects of navigation environments for effective obstacle avoidance, navigation decision-making should also consider factors like the physical affordance of the planned trajectory. For instance, traversing an off-road field covered in tall grass might seem feasible geometrically; however,

it poses challenges such as bumpiness and high energy consumption for vehicles on uneven terrain. The study by Kahn et al. [31] addresses this concern by incorporating affordance into trajectory planning. They employ supervised learning to train an LSTM network, predicting costs like collision, bumpiness, and future positions based on image observations. Subsequently, an MPC computes actions by minimizing costs derived from events predicted by the LSTM network.

However, the aforementioned approach overlooks prediction uncertainty. Authors in work [32] argue that uncertainties, particularly in terrain properties like traction, can significantly impact the prediction of dynamic rollouts and terrain traversability. To address this, they propose learning a Gaussian mixture distribution of observed terrain traction properties instead of a single value. Additionally, they train a density estimator to discern whether the observed terrain aligns with the training data distribution. If not, the subsequent planning phase avoids generating a trajectory for that terrain.

The majority of trained neural networks remain static without further online training during deployment. Liu et al. [33] introduce a lifelong learning framework to enable autonomous vehicles to continuously improve when encountering diverse and unfamiliar environments, and prevent the forgetting of previously learned skills. They employ the gradient episodic memory technique to counteract catastrophic forgetting when agents are trained with new samples from new environments. This framework maintains a small memory sample buffer from old environments to construct an update constraint. Consequently, the policy does not deteriorate when evaluated on these old samples after being updated with new environment-collected samples, facilitating continuous learning without forgetting.

2.1.2 RL-based navigation

While supervised deep learning policies excel at learning from demonstrations, their generalization ability is somewhat limited, and the process demands a substantial

amount of labelled data, leading to labour-intensive human efforts [34]. In contrast, reinforcement learning agents possess the capability to autonomously learn navigation strategies, exhibiting a greater potential for generalization to unseen situations during training [35].

Lei and Ming [36] employ a two-step approach to train robots using reinforcement learning for mapless navigation with depth image inputs. In the initial step, a perception network is trained to extract feature maps from raw depth images through supervised learning. Subsequently, another network, utilizing these feature maps as input, is trained using Q-learning to generate move commands. Notably, in this approach, perception and action decision networks are trained separately. Building on this methodology, they later integrate perception and decision networks into a single end-to-end network, employing Deep Q-Network (DQN) training [37]. The results demonstrate that this unified DQN-trained network outperforms both a supervised learning network and separately trained networks.

Several enhancements have been proposed in the realm of DQN reinforcement learning for navigation applications based on the above two works. Wang et al. [38] advocate for a modular architecture to train robots for navigating through complex environments. This approach partitions the navigation task into distinct local obstacle avoidance and global navigation modules, introducing an action scheduling mechanism to optimise exploration and exploitation strategies. They design a two-stream network to process different components of the inputs to produce better state features. Building on state-of-the-art DQN techniques, Ruan et al. [39] integrate advancements such as double networks and duel architectures into a unified framework, thereby augmenting the robot's navigation capabilities. Moreover, other improvements have been explored, including those related to sample efficiency [40], algorithm hyperparameter selection [41], and various other aspects aimed at refining the efficiency and effectiveness of DQN-based navigation systems [42–44].

While DQN is only available for discretized action spaces, Tai et al. [45] demon-

strate the application of reinforcement learning to navigation in continuous action spaces using synchronous Deep Deterministic Policy Gradient (DDPG) algorithms. Carlucho et al. [46] extend the scope of reinforcement learning to the three-dimensional navigation of autonomous underwater vehicles (AUVs). Navigating in 3D poses challenges for non-learning-based algorithms due to the high action dimension (6DOF) and complex coupled dynamics between vehicles and the underwater environment. Reinforcement learning proves adept at addressing these challenges. However, reinforcement learning in continuous action spaces may demand more data than [47]. Pfeiffer et al. [48] enhance sample efficiency by combining imitation learning and reinforcement learning. The policy network is initially pre-trained using imitation learning, and then the pre-trained networks undergo further training with the constraint policy optimisation (CPO) RL algorithm. This integrated approach contributes to improved efficiency in learning policies for continuous action spaces.

To ensure the applicability of policies in real-world applications, it is essential to utilize more realistic simulated environments. Significant efforts have been directed towards constructing simulators capable of rendering photorealistic images and simulating real-world physics. Examples of such simulators include SUNCG [49], Habitat [50], AI2-THOR [51] and Gibson [52]. These platforms aim to provide immersive and authentic virtual environments that closely replicate the visual and physical characteristics of the real world. Such advancements in simulation technology contribute to the robustness and adaptability of policies developed within these environments, enhancing their potential for effective deployment in real-world scenarios.

In addition to the above discussion on general reinforcement algorithms applications, researchers also pay attention to the implementation details of those algorithms to achieve better performance.

RL training with auxiliary task The complex details of network architectures and training processes in reinforcement learning become crucial, particularly

in scenarios involving visual inputs, such as images. Training networks with image inputs could be difficult and unstable. Given the high-dimensional nature of image inputs, thoughtful consideration must be given to employing specific network layers to reduce dimensionality and extract feature maps for an optimal robot state representation. An innovative technique known as reinforcement learning with auxiliary tasks is introduced to enhance the decision-making process by obtaining a more refined representation [53].

When reinforcement learning agents are trained with auxiliary tasks, they continue training with pseudo rewards derived from these auxiliary tasks in the absence of extrinsic rewards where the reward of the major task is sparse. This is possible because auxiliary tasks and the main task (navigation in this case) share a common representation extraction network [54]. Mirowski et al. [55] embrace this concept and train an agent capable of navigating a complex simulated environment. The auxiliary tasks employed in their study include depth prediction and loop closure. The paper contends that, while depth could be directly used as an input, providing depth as an auxiliary task loss yields denser training signals, as navigation and depth prediction share the same representation. The training method employed is Asynchronous Advantage Actor-Critic (A3C). In a related vein, Tongloy et al. [56] introduce the work of GPU A3C, enabling the algorithm [55] to run efficiently on GPU hardware, thereby enhancing computational efficiency and accelerating the training process.

These advancements collectively contribute to the refinement of network architectures and training strategies, particularly in the context of reinforcement learning with visual inputs.

Target-object-driven navigation Some applications extend to a realistic world application: target reach tasks. In this setting, the robot produces move commands based on the current camera image and the image of the target object, eventually navigating to the target object's position even in the absence of odometry or

Global Positioning System (GPS) information. Zhu et al. [57] propose a vision-based target-driven visual navigation system in an AI2-THOR environment, known for providing high-quality 3D real-life indoor scenes. The authors argue that the network, with image input, can learn a rough map of the environment, enabling the agent to both locate itself and identify the target object's position. The network in their study comprises two parts: the generic part, shared across all scenes (living room, bathroom, etc.), is a pre-trained ResNet. However, the scene-specific part requires retraining when applied to unseen scenes.

Building on the target-driven concept, Kulhanek et al. [58] integrate the idea of training with auxiliary tasks from work [54] and work [55] to empower robots with navigation abilities in more realistic environments. In this work, the reliance on pre-trained ResNet is thus abandoned in favour of a more adaptive approach.

Hsu et al. [59] introduce a classifier designed to categorize features in the embedding space. This enables the robot to automatically select the corresponding scene model while moving in a specific scene, thereby contributing to more dynamic and adaptive navigation strategies. These real-world simulation applications underscore the adaptability and potential of vision-based navigation systems in diverse and complex environments.

Chancan et al. [60] adopts a pipeline similar to work [57], training an agent to navigate through outdoor city environments with a current image observation and a target poison image as inputs. To efficiently represent the current agent position, a pre-trained feature extractor derived from visual place recognition (VPR) models is employed, resulting in a concise representation obtained from the current visual image observation.

Incorporating additional semantic information beyond the current observed image is another avenue for enhancing navigation capabilities. Druon et al. [61] leverages a pre-trained YOLO network to extract objects (e.g., microwave, sink, oven, fridge) from the image. A context grid map is then constructed based on the ex-

tracted objects and their corresponding bounding box positions. The target object is represented by a text word (e.g., "Bread") instead of an image, and a word2vec model encodes this text into a vector, forming part of the input. The input comprises three components: the RGB image, the context grid, and the target vector. In their evaluations, the proposed algorithm surpasses a baseline that lacks context grid inputs.

Analogous to target-driven navigation, visual servoing tasks involve moving the agent to a target view pose using images from the current and target viewpoints. Li et al. [62] propose employing DQN to address such tasks. In comparison to classical image-based visual servoing (IBVS), they argue that the reinforcement learning-based algorithm remains effective even when the overlap between the current view and the target view is minimal, a scenario where IBVS struggles to compute feature correspondence.

Socially compliant navigation In environments where humans are present, it becomes essential for robots to be socially aware [63–65]. This requirement is driven by the need for robots to navigate and interact with humans while avoiding collisions in dynamic environments. Tail et al. [66] employ a generative adversarial imitation learning strategy to train robots. This approach aims to teach robots a more human-like behaviour when interacting with human beings. By leveraging generative adversarial imitation learning, robots can learn to navigate and engage in social interactions with behaviour that aligns more closely with human norms and expectations. This approach contributes to safer and more socially attuned robotic behaviour in dynamic human-robot environments.

Additionally, they offer a simulator based on Gazebo wherein dynamic objects, such as pedestrians, move in a socially compliant manner. Environments with such dynamic elements prove valuable in generating exploration trajectories during the reinforcement learning training of robots, fostering awareness of social norms. Furthermore, these environments can furnish socially compliant expert behaviours for

robots trained through supervised learning. The dynamic objects within the environment are influenced by four forces, as defined in [66]:

$$\frac{dv_t}{dt} = F_{\text{desired}} + F_{\text{social}} + F_{\text{obs}} + F_{\text{flut}} \tag{2.1}$$

Here, F_{desired} defines the force propelling the object towards its navigation target position. It can be simplified as $\lambda(p_{\text{goal}} - p_t)$, where p_{goal} and p_t denote the navigation target and current poses, respectively, and λ serves as a weighting factor. F_{obs} represents the force repelling the object from static obstacles within the environment. The force F_{flut} emerges from environmental randomness and the inherent stochastic behaviour of the dynamic object. Additionally, F_{social} characterizes the force resulting from the influence exerted by other nearby dynamic pedestrians.

While the prior work by Tai et al. [66] exclusively predicts F_{social} and considers F_{desired} as inputs, Tsai et al. [67] took a different approach by training a Generative Adversarial Network (GAN) model. This model is designed to predict all three forces—F_{desired}, F_{flut}, and F_{social}—excluding F_{obs}. The predictions are based on sequences of historical trajectories that encompass both the robot and pedestrians within the given environments.

To solve such a navigation problem influenced by social-norm, considerable effort has been dedicated to human behaviour prediction [68–70], which is not the main interest of this paper and will thus not be discussed in detail.

POMDP-based Navigation In scenarios involving mapless navigation and reinforcement learning, robots may face challenges due to limited information about the whole scope of the environments and a lack of history information due to the reinforcement learning MDP setting, potentially leading to getting stuck in local minima like a long wall in front of the navigation goal [3]. A strategic solution to address this issue is to model the navigation problem as a POMDP. Memory-based recurrent neural networks [71] and deep recurrent Q-learning [72] present effective

approaches for tackling the complexities inherent in POMDP problems.

In the study by Meng et al. [73], an innovative approach was taken by combining the LSTM network with Twin Delayed Deep Deterministic Policy Gradients (TD3) to address POMDP tasks. The integration of the LSTM network plays a crucial role in encoding historical temporal information within a time series [74]. This enhancement contributes to the model's ability to capture and utilize past information effectively in the context of POMDP tasks.

Wang et al. [3] propose an innovative solution by introducing a fast recurrent deterministic policy gradient. Notably, this algorithm allows for online training without requiring the entire trajectory. The efficacy of this approach is demonstrated in practical applications, such as navigating UAVs through large-scale and complex environments. In such environments, where UAVs might easily become stuck in local regions, this algorithm proves valuable in ensuring efficient and dynamic navigation.

Navigation with instructions In scenarios where navigation instructions are available, such as guiding a stranger in an unfamiliar building, work by Devo et al. [75] incorporates textual instructions for a robot navigating through a maze. Instructions, like *"Go straight, at the next intersection turn left, at the next intersection go straight, at the next intersection turn right,"* are pre-processed using a pre-trained GloVe word encoding network [76] and a Bidirectional Gated Recurrent Unit (BI_GRU) network. The resulting embedding vector becomes part of the policy input, supplementing the current observed image input. The policy is trained using the GPU-based A3C reinforcement learning algorithm.

In a different approach, Xie et al. [77] presents instructions in the form of a sparse sequence of images paired with steering commands (e.g., turn left, turn right, stop). A commander block, constructed with dense layers, GRU layers, and CNN layers, is trained through attention-based supervised learning to provide steering commands. These commands, along with the currently observed image and sparse image sequence with instructions, are used as inputs for a reinforcement learning-

based controller, trained to follow the provided steering commands.

Brunner et al. [78] introduce an algorithm that addresses the balance between direction-following commands and localisation needs in robotic navigation. In their approach, after training, the robot is designed to explore its surroundings to improve localisation rather than solely adhering to directional commands. It's worth noting that, in this particular paper, the direction-following command is not learning-based, showcasing a strategy where the robot dynamically adjusts its behaviour during exploration for improved localisation.

Safe RL navigation Safe navigation in terms of collision avoidance is an essential factor to be considered during policy decisions. Previous work encoded safety requirements as a soft constraint by producing a negative value reward in addition to other task-related reward components. The overall objective function to be optimised will be a trade-off between safety and other task aims. It may violate the safety constraint if other reward components are larger than the safety violation punishment and are thus dominant in constructing the overall objective function. Hence, safety should be treated separately and be imposed as a hard constraint where safety is critical. This brings the constrained reinforcement learning:

$$\max_{\theta} J(\theta) = E_{\tau\sim\pi_\theta}[\sum_{k=0}^{\infty} \gamma^k r_t(s_t, a_t)] \tag{2.2a}$$

$$s.t.\ c_{\pi_\theta}(s_t, a_t) \in \mathcal{C}, \tag{2.2b}$$

where $c_{\pi_\theta}(s_t, a_t)$ is the overall safety cost with action a_t depending on the state s_t at time step t. The cost should not exceed an acceptable range as defined by $\mathcal{C}$.

Work [79] proposes to solve such a constrained optimisation problem by a Lagrangian method, which converts the constrained problem to an unconstrained min-max problem by introducing dual variables (Lagrangian multipliers). Within this work, the dual variables are learned along with the primal variables(policy parameters π_θ). Hence, the learning procedure is noisy and it may violate the safety

constraints before the convergence of the policy training [80]. Instead of learning the dual variables from the intermediate policy π_θ, work [80] computes the dual variables from scratch for each policy update during training such that the constraint is always satisfied. In addition to the safety constraint, this work also put a constraint on the update distance between two consecutive policy update steps similar to trust region optimisation to alleviate the difficulties brought by the requirement of the off-policy evaluation of the cost. To guarantee monotonically reducing violation of safety constraints, work [81] design the policy update dynamics following a trajectory resulting in the policy stably converging to the feasible set. This is done by converting the safety constraints to Lyapunov functions. They designed to learn an LSTM-based optimiser to achieve the desired update dynamics. The algorithm is tested by a navigation task with obstacle avoidance as the safety constraint. In comparison to works [79] and [80], this work shows a monotonical decrease in obstacle collision numbers as the training steps grow and is less noisy, while work [79] even diverges without the ability to satisfy safety constraints during the training.

Transfer and Adaptability of RL-trained Policies In certain scenarios, learned policies may need to be transferred to navigation tasks with settings different from their training environment, such as a sim-to-real gap [82], varying sensor configurations [83], or even different types of sensors [84]. Retraining policies from scratch in these situations can be time-consuming [85].

Most reinforcement learning policies are initially trained in simulated environments, introducing a sim-to-real gap when deployed in the real world [86]. This gap arises from the differences in the visual observation and physical dynamics between the training and deployment environments. Truong et al. [85] address this sim-to-real problem by proposing a Bi-directional domain adaptation method. They train a visual navigation policy using a decentralized distributed variant of the Proximal Policy Optimisation (PPO) reinforcement learning algorithm. To bridge the dynamics gap, a sim2real dynamics adaptor is trained, modelling the residual error

between simulation and reality dynamics. This dynamics adaptor refines the PPO-trained policy network to account for more realistic dynamics transfer. For real-world deployment, they train a real2sim observation adaptor using CycleGAN [87] to convert real-world images into simulation-like images. This bi-directional adaptation requires significantly less real-world data and achieves comparable navigation performance to direct fine-tuning in the real world.

In a similar vein, Luo et al. [88] adapt a reinforcement learning navigation policy with camera inputs to a robot with a camera at a different height. By separating the policy network into perception and inference parts, they fine-tune only the perception part and fix the parameters of the inference network. This approach avoids retraining the entire policy network and achieves adaptation by aligning the latent embeddings produced by the perception networks for both the old and new camera configurations.

Another study by Huang et al. [84] explores the adaptation of a policy trained with Lidar inputs to work with millimetre-wave (mmWave) inputs. They fine-tune both the perception and inference blocks of the policy network, introducing an extra contrastive loss related to the perception embedding difference between mmWave and Lidar inputs in addition to regular navigation rewards. For comparison, an observation adaptor is trained to transfer mmWave inputs to the Lidar domain as an alternative method without retraining the policy network similar to the real2sim part of work [85], which turns out to face challenges such as a 'phantom wall' and susceptibility to noise in mmWave inputs.

Chisari et al. [89] propose introducing uncertainty and regularization terms during training to account for unmodeled dynamics in racing tasks. Instead of capturing unmodeled dynamics with a dynamics adaptor, they utilize a multiplicative uncertainty model. Also, recessive and jerky control signals will result in introducing extra vehicle dynamics, which are difficult to model. To avoid stimulating such complex dynamics, which are unseen during training dynamics when applying the policy

to real racing cars, regularization penalties related to policy outputs are added to prevent the policy from generating excessive and jerky control signals.

2.1.3 Navigation by combining RL and classical control

While navigation policies trained through end-to-end reinforcement learning offer the advantage of handling multimodal inputs [90] and simplifying the overall vehicle control framework, concerns arise regarding reduced transparency and the potential compromise of safety in control signal outputs [91, 92]. To address these challenges, researchers advocate for a hierarchical system that integrates reinforcement learning policies with conventional controllers. In such hierarchical control pipelines, reinforcement learning policies are responsible for high-level decision-making, while execution is delegated to conventional controllers, providing theory-based convergence and safety guarantees.

In the context of autonomous highway driving, Mirchevska et al. [91] implemented a hierarchical system where a DQN-based high-level network selects a lane from a set of reachable lanes. The low-level conventional optimal control-based trajectory planning then plans a trajectory to the selected lane. Evaluation results demonstrate that the proposed algorithm outperforms purely conventional algorithms in terms of navigation time cost. Similarly, Qiao et al. [93] propose training a high-level behaviour planner to decide whether to change lanes or follow the front vehicle. In this case, the trajectory planner is also a reinforcement learning-trained policy, and the low-level execution control is implemented using a PID controller.

Another approach is presented by Brito et al. [94], where a high-level controller serves as the global planner to determine a sub-goal, and an MPC controller acts as the local planner to reach the sub-goal. This hybrid approach ensures that the low-level control signal adheres to the robot's kinodynamic and obstacle avoidance constraints. Comparative assessments against pure MPC navigation policies and pure reinforcement learning policies reveal that the hybrid MPC approach achieves

superior performance in terms of success rate and travel distance. In their subsequent work [95], they train the high-level RL policy to predict informative subgoal. The overall system is the same. Work [96] has followed the same hierarchical structure and applied it to build a quadrotor navigation controller.

In addition to explicit high-level navigation commands, high-level decision-making can involve implicit parameter decisions. Conventional non-learning-based navigation control systems often optimise or sample control signals using a fixed set of heuristically chosen parameters, which may need re-tuning for different environments. Neural networks, with their proficiency in environment perception, can handle the re-tuning task. In the work by Xu et al. [92], a high-level parameter decision policy network is trained using the Twin Delayed Deep Deterministic Policy Gradient reinforcement learning algorithm. This network selects parameters such as maximum linear and angular velocity, obstacle inflation radius, collision avoidance, path following, and goal-reaching cost weight factors for a DWA [97] planner. The algorithm adapts navigation behaviours in environments with varying configurations or numbers of obstacles, outperforming a DWA planner with fixed parameters in terms of reducing traversal time.

Similarly, Li et al. [98] propose a high-level policy to decide the reference velocity for the low-level safe controller, Constrained Iterative Linear Quadratic Regulators (CILQR). CILQR can solve optimal control for a non-linear dynamics system with non-convex constraints, but it tends to exhibit aggressive tracking of reference speed. The reference velocity can be selected adaptively by the RL policy according to different environmental conditions (e.g. crowdy or sparse). The proposed algorithm achieves a high completion rate and a low collision rate by selecting a suitable reference velocity.

In contrast to these works, Patel et al. [99] explore how conventional control formulations can aid and regulate reinforcement learning algorithms. They argue that a policy trained by reinforcement learning may over-reward collision avoidance

behaviour, leading to violations of robot dynamic feasibility. To address this, they propose using the conventional DWA controller to compute a set of dynamically feasible velocities and the cost of achieving corresponding velocities. These values are then used to construct the observation of the reinforcement learning policy, ensuring that the command from the reinforcement learning policy satisfies robot dynamics constraints and significantly reducing the dimensionality of the state space explored by the reinforcement learning policy.

2.1.4 Uncertainty-aware RL Navigation

Navigation policies developed through reinforcement learning algorithms encounter challenges when confronted with unfamiliar state inputs due to limited exploration during training [100]. In such instances, there arises a need for the navigation strategy to transition to a more conservative navigation policy to ensure safety when uncertainties in decision-making are apparent, for instance, slowing down speed [101].

To gauge the confidence of the learned policy in its action decisions based on the current observation, the study in work [102] introduces a method for estimating decision uncertainty in policies trained with the DQN algorithm. Instead of relying on a single Q-value prediction at the last layer of the Q network, the proposed approach involves adding multiple heads as the last layer. Each head comprises several neural network layers and produces a Q-value estimation based on its unique set of network parameters. The authors argue that, given the random nature of exploration and initialization, each head will generate a distinct Q-value estimation. As training progresses, these estimations converge to real values and become more consistent when the states are thoroughly explored. For unfamiliar states, where exploration may be limited, the Q-value estimations from different heads are expected to differ. By calculating the variance of Q-values across all heads, the method provides an estimation of the decision uncertainty. This uncertainty measure allows the navigation policy to determine its confidence level in action decisions based on the current

observation.

Motivated by the concept of uncertainty awareness, the study conducted by Rana et al. [103] involves the training of a reinforcement learning-based residual navigation controller atop a sub-optimal classical controller. In instances where the learned residual controller exhibits low uncertainty concerning the current observation, it generates incremental linear and angular velocity corrections, denoted as $[v_\delta, w_\delta]$, which are applied to the linear and angular velocity commands from the sub-optimal classical controller, denoted as $[v, w]$. Consequently, the control command becomes $\text{CMD} = [v + v_\delta, w + w_\delta]$. However, in cases where uncertainty is high regarding the correction command from the residual controller, the velocity correction suggestions $[v_\delta, w_\delta]$ are not accepted, and the control command remains solely derived from the classical controller: $\text{CMD} = [v, w]$. The uncertainty is quantified by the variance of the network output using the Monte Carlo dropout technique, as proposed in the work by Kahn et al. [104], without altering the network configuration.

In contrast to the MC-dropout technique, Liu et al. [105] argue that it imposes a substantial computational burden, rendering it impractical in resource-constrained environments. They propose an alternative approach by advocating for the use of evidential deep learning, a method capable of directly learning epistemic uncertainty without the need for sampling or network reconfiguration. This technique is subsequently applied in the training of an efficient and robust end-to-end navigation policy. Notably, the evidential technique is also endorsed by Wang et al. [106] for estimating policy uncertainty.

An additional source of uncertainty stems from observation uncertainty, attributed to factors such as sensor measurement noise or estimation uncertainty in the movements of dynamic objects [107]. In addressing this challenge, Fan et al. [108] introduce a novel approach by explicitly incorporating the uncertainty value as a component of the policy input. Post-training, the policy, now equipped with observation uncertainty inputs, demonstrates the ability to guide the robot away from

uncertain obstacles or dynamic objects following random patterns. Consequently, this leads to a reduction in collision rates when compared to the uncertainty-unaware baseline.

2.1.5 RL-based exploration

Beyond path planning, map-based reinforcement learning can also serve exploration purposes. Niroui et al. [109] task the robot with deciding which frontier point to explore based on the unexplored map during the exploration phase. The objective is to fully explore the environment as quickly as possible. Reinforcement learning is utilized to determine the next exploration point, and navigation to the chosen point is executed using the classic A^* algorithm. In their subsequent work [110], navigation is also conducted using reinforcement learning, highlighting the versatility of map-based reinforcement learning in handling various aspects of robotic navigation.

Diverging from the approach in work [109], where the networks primarily offer selection preferences for pre-processed frontier points, leaving the navigation to a conventional controller, Chen et al. [111] take a different path. Their method involves providing navigation steering commands based on a comprehensive set of inputs. These inputs encompass an estimated robot position, an RGB image, an RGBD image, a fine egocentric map, and a coarse egocentric map, forming an end-to-end pipeline. This enables the agent to simultaneously gather environmental information for mapping and obstacle avoidance. Notably, the robot poses are estimated using a motion model, which may accumulate estimation errors during exploration. The network is pre-trained using trajectories collected from human demonstrations.

While the end-to-end pipeline is a compelling concept, the authors of work [112] contend that such a method comes with the drawback of requiring a large number of training examples. In response, they propose a decomposition of the system into three distinct components: a neural SLAM, a global planner, and a local planner. The neural SLAM module is tasked with map construction and robot pose estima-

tion, trained through supervised learning. The global planner component undergoes reinforcement learning to predict a long-term goal to increase map coverage. This long-term goal is then employed by an analytic path planner to generate a short-term goal, serving as the navigation destination for the local planner. The local planner, in turn, is trained using imitation learning. This hierarchical algorithm surpasses the aforementioned end-to-end method [111] in terms of overall environment coverage.

Adopting a framework similar to [112], work [113] introduces an additional learnable module called the occupancy anticipation network, aiming to accelerate exploration. Unlike other methods that construct occupancy maps through projection, this module generates predictions based on RGBD images. Projection-based map construction is constrained to visible regions, but the occupancy anticipation network enables the robot to construct maps through predictions, allowing for reasonable imagination. For instance, the system can predict free space behind a table without directly navigating to that area for observation. This approach saves time by eliminating the need to physically travel to every location. The reward space for training the global planner is designed not only to enhance map coverage but also to improve the accuracy of the occupancy anticipation map.

The exploration policies trained in simulation environments may encounter challenges when deployed in untrained environments or the real world, where sizes and obstacles may differ. In addressing these sim-to-real issues, work [114] takes a different approach by constructing an exploration graph as inputs, where nodes represent poses from SLAM and frontier points. Utilizing Graph Convolutional Networks [115], which are scalable, the policy network is trained to produce commands to minimize the uncertainty of virtual landmarks. This novel approach allows for the successful transfer of a policy trained in one environment to be deployed in other environments of varying sizes, including real-world scenarios.

Navigation with Maps In contrast to mapless navigation, some algorithms incorporate environment maps as input, introducing a path-planning element to the

navigation process. Lv et al. [116] employ a technique where a map is converted into an image, with pixel values representing obstacles, robot position, or goal position. A dense network processes this image to generate move commands, and the network is trained using DQN. While this method addresses map-based navigation to some extent, concerns about its generalization to unseen maps persist [117].

Recognizing the limitations of single CNN-based networks in understanding planning aspects, Tamar et al. [117] propose a Value Iteration Network to approximate the classic value iteration planning algorithm. This differentiable network can be trained through reinforcement learning or imitation learning. The effectiveness of this approach is demonstrated in grid-world navigation tasks and a Mars rover navigation within the reinforcement learning framework. Pflueger et al. [118] enhance this network's training by incorporating the inverse reinforcement learning algorithm.

2.1.6 RL algorithms for multi-agent navigation

In navigation scenarios, deploying multiple robots in a single environment and training them collectively is a notable strategy. Long et al. [119] undertake the training of a single navigation policy using reinforcement learning with multiple robots in the same environment, sharing experiences to contribute to the training of the policy. All robots utilize this shared navigation policy during the training process, akin to asynchronous reinforcement learning methods. The results demonstrate that the robots develop cooperative behaviours during navigation, even in the absence of explicit cooperation rewards during training.

Expanding on this cooperative navigation strategy, Fan et al. [120] combine the navigation approach [119] with SLAM methods for pose estimation. This integration aims to ensure that robots can navigate to their goal positions through dense pedestrian crowds without getting lost.

Sun et al. [121] take a similar approach by combining the algorithm proposed in

[119] with the Reciprocal Velocity Obstacle (RVO) [122] algorithm. This combination enables four groups of agents to navigate to their designated positions without collisions. These efforts underscore the potential for collective training and navigation strategies to enhance cooperation and coordination among multiple robots in shared environments. Traditional methods like optimal reciprocal collision avoidance often incur high online computation costs. Learning-based methods offer a promising alternative by offloading this online computation to offline training procedures.

Chen et al. [123] present an approach where a value network is employed to train agents in collision avoidance with each other. The paper also introduces the idea of imposing a penalty when two agents take significantly different amounts of time to reach their goals. This penalty serves as an encouragement for cooperation among the agents, highlighting the potential for learning-based methods to enhance not only collision avoidance but also cooperative behaviours in multi-agent scenarios.

In contrast to previous methods where the target position of each agent is manually or randomly assigned, there is a potential to consider the target position assignment as a dynamic and learning-based process. Work [124] introduces an interlaced deep reinforcement learning method wherein each agent can autonomously and dynamically select a target position and navigate to the chosen location without colliding with other agents. The essence of this method lies in learning two key functions: a target selection Q function and a collision-avoidance Q function. The combination of these Q functions facilitates the guidance for agents in the navigation process, optimising the time taken for all agents to occupy their respective target positions. This approach represents a more adaptive and intelligent way for agents to handle target selection and navigation in a collaborative environment.

Indeed, multi-agent cooperation can significantly enhance the efficiency of information gathering, especially in the context of exploring an unknown environment. Viseras and Garcia [125] leverage deep reinforcement learning to achieve effective

information gathering with multiple robots. In their approach, robots are trained to explore a designated area and efficiently gather comprehensive information through cooperative actions. The design of the observation and reward functions within the reinforcement learning algorithm plays a crucial role in guiding the robots to collectively explore and gather information promptly. This application highlights the potential of reinforcement learning and multi-agent cooperation in optimising information collection tasks in complex and unfamiliar environments.

Reinforcement learning can also be used for flocking tasks [126]. Wang et al. [127] employ DDPG reinforcement learning to train multiple UAVs to navigate to their respective destinations while maintaining a cohesive flock. Cooperative behaviours are encouraged by introducing positive rewards when a UAV maintains a distance of around 20 meters to its nearest two neighbours on the left and right sides. This strategy incentivizes the UAVs to coordinate their movements and remain in a collective formation during navigation.

Similarly, Zhou et al. [128] implement an algorithm for three Unmanned Surface Vehicles (USVs) to navigate in a triangular formation using reinforcement learning with reward shaping. The reinforcement learning framework, coupled with carefully designed reward signals, facilitates the cooperative navigation of the USVs, enabling them to maintain a specific formation during their movement. These applications showcase how reinforcement learning can be effectively utilized to train multiple autonomous agents to navigate collaboratively while adhering to specific formation requirements.

In contrast to the assumption of agents sharing the same policy, Lowe et al. [129] propose a modified actor-critic reinforcement learning training algorithm that allows different agents to act according to distinct policies. These policies can be cooperative, competitive, or a mix of both. The key modification in their approach involves training the critic by considering both the agent's own observation and the actions of other agents, while the actor's input is solely based on its own observation. This

modification introduces flexibility in the behaviour of different agents, enabling them to follow diverse strategies during training.

Furthermore, this work suggests training an ensemble of policies for each agent to enhance robustness. This ensemble approach introduces diversity in the policies of individual agents, promoting adaptability and resilience in different scenarios. This work highlights the potential of training heterogeneous agents with distinct policies, enabling a richer and more flexible range of behaviours in multi-agent environments.

2.2 Learning-based UAV control

2.2.1 Deep learning

To train a policy capable of controlling a quadrotor in urban environments, the work by Loquercio et al. [130] recommends employing supervised learning with demonstrations. This approach utilizes visual images as network inputs, generating a steering angle command and a collision probability. The utilization of high-level control commands facilitates learning from demonstrations obtained in safer contexts such as car driving or bicycling, eliminating the potential risks associated with collecting demonstration data while flying a quadrotor in dynamic urban environments. The collision probability serves the purpose of adjusting the forward speed of the UAV. This policy simplifies the traditional and intricate 'map-localise-plan' control pipelines. Remarkably, the policy trained in outdoor settings demonstrates effectiveness even in indoor environments, such as parking lots or indoor corridors.

Imitation learning is employed in the training of a quadrotor to adeptly follow a trajectory that traverses a sequence of gates [131]. Unlike traditional methods that represent gates using ground truth 3D positions, this approach relies on the observation of gates through onboard cameras. This bears a resemblance to the way humans utilize their eyes to execute tasks, emphasizing the practicality and real-world applicability of the learning process. The authors endorse a hierarchical

learning framework that integrates supervised learning with classical control. In this approach, the neural network produces a two-dimensional goal vector in the camera frame, along with a desired speed serving as the command signal for the low-level controller. The low-level controller, in turn, is capable of devising a minimum jerk trajectory based on the received command and subsequently tracking that trajectory. The demonstration data utilized in this context is derived from a global trajectory planning algorithm, which strategically plans a trajectory passing through all gates, providing essential labelled data.

Training a network from scratch with image inputs will be difficult. Work [132] adopts a two-component structure for the control policy designed for a visual UAV racing task, comprising perception and control. The training of the perception block involves fine-tuning YOLO(v5) to obtain invariant feature embeddings against background visual disturbances unrelated to the navigation task. The fine-tuning procedure minimizes the embedding differences between images taken at the same pose but augmented with different visual disturbances. Once fine-tuned, the perception block remains fixed. The control block is then trained by imitation learning.

Using the same hierarchical framework, Loquercio et al. [10] propose a prominent work that trains a high-level policy by imitation learning with demonstrations from a sampling-based planning algorithm [133]. The high-level policy produces 3 control points to construct a B-spline trajectory that will be followed by an MPC controller. The method has achieved high-speed flight in real quadrotors in various environments (e.g. forests, city streets etc.)

While visual inputs have the potential to offer a wealth of information, not all components are relevant or may even prove detrimental to the task at hand. Training neural networks with raw images can exacerbate training challenges. Unlike machine learning models, humans instinctively focus on crucial areas within an image rather than processing the entire visual field. In the work by Pfeiffer et al. [134], gaze data obtained from human demonstrations is employed to train an encoder-decoder

network, using imitation learning to predict attention on images. The output of the encoder serves as extracted features for the control network. Additionally, the control network undergoes imitation learning from an expert MPC to generate low-level thrust and body rate commands.

2.2.2 RL-based methods

The efficacy of policies trained through supervised learning is constrained by the quality of demonstrations provided by human experts or classical controllers, potentially resulting in suboptimal performance when optimal demonstrators are unavailable. In contrast, reinforcement learning algorithms empower quadrotors to explore autonomously during training, eliminating the need for expert guidance and facilitating the attainment of optimal performance. Several works have contributed to the development of reinforcement learning algorithms for quadrotor control, addressing various aspects such as exploration, convergence, tasks, and implementation details etc.

In the study by Hwangbo et al. [135], a novel approach is proposed for training a policy using reinforcement learning algorithms to stabilize quadrotors under challenging initial states, such as being manually thrown into the air at an upside-down configuration with a velocity of $5m/s$. These scenarios pose difficulties even for human expert pilots or conventional controllers like model predictive control. The presented work devises an actor-critic algorithm akin to the standard Trust region policy optimisation (TRPO), employing a distinct distribution distance measure method to achieve a faster convergence speed. Similarly, Molchanov et al. [136] aim to achieve a comparable task but utilize the standard PPO algorithm.

In the research conducted by Lin et al. [137], the objective is to train a controller for navigating a quadrotor through an inclined narrow gap using end-to-end reinforcement learning. The authors advocate that the end-to-end approach not only reduces computation time through parallel processing on GPUs but also mitigates

errors accumulated in each module of conventional pipelines. Their methodology involves initial pre-training of the network through imitation learning, utilizing demonstrations from a traditional planning-based method. Subsequently, the network undergoes refinement through reinforcement learning. The study demonstrates that reinforcement learning outperforms both imitation learning and traditional methods in achieving superior performance.

The study by Wang et al. [138] identified a limitation in standard policy gradient algorithms, attributing it to the bias and variance in Q-value estimation introduced during training. Specifically, their findings revealed that velocity controllers trained by DDPG were unable to bring a quadrotor to a desired velocity, resulting in non-zero steady-state tracking errors. In response to this challenge, the researchers drew inspiration from classical control theory and introduced an integrator to the system to eliminate steady-state errors. They augmented the state inputs of the network with an integral compensator, which accumulates state errors over time. The integration of this compensator allowed them to achieve nearly zero velocity tracking for the quadrotor. Notably, the learned controller exhibited robustness and generalization to quadrotors of different weights and sizes, surpassing the performance of a well-tuned PID controller in maintaining stable velocity control for previously unseen quadrotors

Policies trained through reinforcement learning often exhibit aggressiveness when robots are tasked with completing assignments as quickly as possible, leading to control commands that lack smoothness and introduce oscillations, as discussed by Mysore et al. [139]. This aggressive behaviour not only results in suboptimal performance but can also lead to increased energy consumption, unnecessary system wear, and even catastrophic crashes in the case of quadrotors with complex dynamics. In response to these challenges, the work by Mysore et al. [139] proposes the introduction of an action regularization loss during training to impose constraints on the policy output. The regularization comprises two components: a spatial term

aimed at minimizing action differences between similar states and a temporal term designed to reduce action differences between consecutive time steps.

Navigating through a predefined sequence of waypoints in a specific order with minimal time cost is a typical and challenging task for an autonomous quadrotor, serving as a valuable metric for evaluating controller capabilities. Work by Penicka et al. [140] proposes employing a topological path as a reference and training a policy through a reinforcement learning algorithm to guide the quadrotor along this topological path. Training a policy from scratch to control the quadrotor for achieving minimal flight time is a challenging task and may result in converging to a local minimum. To address this, the authors adopt a curriculum training approach. In the initial phase of curriculum training, the policy is trained to complete the task within a restricted maximum speed. This facilitates the network in learning to effectively control the quadrotor. In the subsequent phase, the constraint on maximum speed is removed, and the policy undergoes retraining to learn how to exploit the full capabilities of the quadrotor.

When the policy is provided with high-dimension inputs (i.e. images), it will be difficult to train the network from scratch to converge to good performance with reinforcement learning. Work [141] employs a teacher-student learning framework. Instead of explicitly presenting depth images for obstacle perception, the teacher policy, which is trained by a reinforcement learning algorithm, possesses full ground truth obstacle position information. This approach allows the teacher policy to maximize the vehicle's capabilities and achieve optimal racing results. Behaviour cloning is then applied to train the visual-based student policy, utilizing demonstrations from the state teacher policy.

Training a control policy for UAV navigation using visual image input poses another significant challenge in terms of difficulty in transferability to unseen environments. This difficulty arises due to the high dimensionality of images, coupled with substantial differences between simulated and real-world images. In efforts to

address these challenges, researchers in works such as [132, 142–144] advocate for a modular approach, dividing the policy network into a perception block followed by a control block. The primary focus lies in constructing the perception block using techniques like domain randomization or conservative learning. This emphasis aims to facilitate training convergence and the extraction of invariant features across diverse scenes, encompassing both virtual and real environments

2.2.3 Combination of RL and MPC

While reinforcement learning excels at optimising long-term objectives through offline exploration, directly applying its output control commands to a quadrotor with complex system dynamics can pose risks without a theoretically guaranteed safety measure. On the other hand, model predictive control-based controllers can plan online control sequences, ensuring adherence to dynamics and safety constraints. However, their planning horizon is inherently limited and short-term.

The work by Romero et al. [145] seeks to leverage the advantages of both methods for quadrotor control. This is achieved by training a neural network policy using a reinforcement learning algorithm, where the policy's outputs become parameters used to construct an objective function representing long-term costs. Subsequently, an MPC controller determines the control sequence by optimising this objective function while considering quadrotor dynamics and other system or environment constraints. The proposed algorithm demonstrates superior performance compared to both a control policy trained solely with PPO and a standalone MPC controller, as measured by navigation success rate and average navigation speed. Furthermore, evaluations indicate the hybrid control strategy's robustness in unforeseen scenarios and against unknown disturbances. This robustness may be attributed to the adaptive nature of the objective function, constructed by the reinforcement learning-trained policy, in contrast to the fixed set of objective function parameters employed by the pure MPC controller.

In contrast to constructing an objective with long-term cost prediction, the approach proposed by Song et al. [146] advocates employing a learning-based policy as a high-level reinforcement learning-based controller to determine hyperparameters for a low-level MPC controller, enhancing its adaptability. Conventionally, human experts heuristically craft the hyperparameters of MPC controllers, which may be suboptimal. The suggested structure not only reduces human intervention but also achieves optimal control performance through exploration in various environments. For evaluation, the researchers devised a task involving a quadrotor autonomously navigating a square gate attached to a pendulum. The high-level policy learns to determine the optimal time to traverse the gate, and the MPC controller plans the control sequence based on this selected timing. The success rate of the MPC controller, when utilizing the traversal time suggested by the high-level policy, surpasses that of controllers relying on randomly determined or heuristic-decided traversal times. Similarly, work by Romero et al. [147] employs a similar methodology for hyperparameter tuning of a Model Predictive Contouring controller designed for autonomous UAV racing tasks.

An alternative approach to enhance MPC involves leveraging deep learning techniques. MPC, while planning into future horizons, relies on a system dynamics model to predict future states. Deep learning can facilitate the learning of a highly precise system model using neural networks. However, executing partial differentiations on these networks, especially on embedded systems with limited computation resources, can be time-consuming. This prolonged computational time may not meet real-time requirements for systems demanding high-frequency control signals, such as those used in the control of high-speed quadrotors.

To address this challenge, the authors in work [148] propose approximating the neural network-modelled system dynamics up to the second order through a Taylor expansion around the current state. This approach stands in contrast to linearizing the neural network model at all timesteps within the planning horizon. Their find-

ings indicate that such an approximation introduces negligible errors in the context of agile quadrotor control

To alleviate the computational burden associated with MPC controllers, Wiedemann et al. [149] present an alternative approach distinct from the use of simplified linearized dynamics. Their proposal involves training a neural network-based solver. This solver takes the current state and future reference states within the planning horizon as input and generates a sequence of control signals. In comparison to the conventional MPC controller, this neural network-based solver achieves comparable performance with significantly reduced computation time, as evidenced by trajectory tracking errors in quadrotor control. Remarkably, it surpasses the performance of both a model-free reinforcement learning controller, PPO, and a model-based reinforcement learning controller, Probabilistic Ensemble with Trajectory Sampling (PETS)

2.2.4 Achieving the best performance of RL

It is arguable whether the reinforcement learning controller is better than the conventional non-learning-based controller. Song et al. [150] conducted a comprehensive performance comparison between classical optimal control and reinforcement learning control for an autonomous UAV racing task. By assessing these controllers in both simulation and real-world environments, the authors observed that the reinforcement learning-trained controller outperformed the classical optimal controller in terms of both success rate and completion lap time.

The authors posit that this superiority stems from the reinforcement learning controller's ability to directly optimise a task-level objective, which may be nonlinear and nonconvex. In contrast, optimal controllers are constrained to decoupling the task into planning a reference trajectory using an intermediate representation and designing a controller to follow this trajectory. This decoupling limits the quadrotor's overall capability. Additionally, the study demonstrated that the reinforcement

learning controller exhibits robustness to unmodeled dynamics, achieved through the adoption of domain randomization during training.

A crucial consideration in reinforcement learning is determining the control commands that the trained neural networks should output. Work [151] categorizes control commands into three groups based on their corresponding command hierarchy: Linear Velocity (LV), Collective Thrust and Bodyrates (CTBR), and Single-Rotor Thrust (SRT). The LV represents a high-level command requiring a full control stack to map the command into individual rotor thrusts. Importantly, this command does not rely on specific UAV dynamics, facilitating easy transferability to different UAVs. On the other hand, the CTBR command also requires a controller for individual thrust mapping. However, compared to the LV command, CTBR is a more low-level command that enables significantly more aggressive UAV maneuvers [151]. Lastly, the SRT provides direct control of individual thrust, allowing for the full utilization of the true actuation capability of the UAVs.

To compare the control performance of different action spaces, Three distinct policies, each with different action spaces as discussed above, were trained to execute a trajectory tracking task. Following extensive testing with 600 diverse trajectories covering the entire flight envelope of the quadrotor, the authors of the study conclude that the CTBR policy demonstrates robustness against model dynamics mismatch. Furthermore, it exhibits successful transferability between different domains without compromising agility. Conversely, the policy producing LV commands yields inferior performance, particularly for agile manoeuvres. Policies directly controlling SRT experience a notable reduction in performance when tested with models different from the training environments, especially for agile trajectories. SRT-based controllers exhibit significantly higher tracking errors in reference trajectories of slow-speed operations and undergo a higher number of crashes when tested on faster manoeuvres.

In work [152], the authors achieved a significant milestone by training a UAV controller capable of competing with three world-leading pilots in drone racing and

achieving victory in most races. This autonomous UAV policy is learned to make control decisions solely based on onboard computation resources and sensors, without relying on external systems such as motion capture.

The system architecture, like others, is divided into two essential components: the perception module and the control policy module. The perception module conducts state estimation using a visual-inertial-odometry algorithm based on IMU data and RGB images from the onboard camera. A neural network track gate detector is employed to detect gate corners from images, which are then utilized to estimate the gate's pose. On the other hand, the control policy is trained using the standard PPO algorithm.

Beyond the typical task-related reward, which encourages the quadrotor to navigate through track gates, an additional reward component is introduced in the reward space. This extra component incentivizes the quadrotor to focus on the target gate, thus enhancing perception. To bridge the gap between the simulation and real-world environments, the authors employ a Gaussian process to model state estimation errors and a k-nearest-neighbour regression to model dynamic errors. This approach utilizes data collected from the real world to enhance the realism of the simulation.

2.2.5 Learning complex dynamics

Achieving high-accuracy models for quadrotors, a necessity for conventional controllers like MPC, is challenging due to the complex aerodynamics involved in their operation, making analytical methods less effective. In response, data-trained neural networks have shown promising performance in modelling systems with complex dynamics [153]. Researchers have thus explored learning-based methods to model quadrotors. For instance, in Work [153], a two-layer neural network is employed to model a small quadrotor, specifically the Crazyflie. This network serves as the prediction model for an MPC controller, effectively controlling the Crazyflie.

The dynamics of a quadrotor are inherently complex, making analytical simulation challenging, and this complexity is compounded when the quadrotor is in flight with suspended payloads. Work [154] leverages the capacity of neural networks to learn the dynamics model of this intricate system. Similar to the approach taken in [153], the learned neural network serves as the prediction model for an MPC controller, enabling the generation of a control sequence

Accidental physical damage is a potential factor that can alter the dynamics of UAVs. In such instances, it becomes crucial for the controller's model to promptly recognize these changes and adapt to the new dynamics. Work [155] suggests employing neural networks for monitoring the system and facilitating adaptation to the altered dynamics when such changes occur

In addition to the complex dynamics of the quadrotor itself, external factors like wind disturbances can lead the quadrotor away from its nominal dynamics. Work [156] takes a different approach by employing neural networks to model the effect of wind disturbance as a residual error added to the quadrotor's nominal velocity. The performance of the neural network-augmented controller demonstrates a smaller tracking error compared to the controller using $L1$ adaptive control. Similarly, in work [157], an attempt is made to capture the complex aerodynamic disturbances introduced by airflow from propellers and the ground during near-ground operations, such as flying or landing, using neural networks. The neural network is employed to predict the additional forces and torques introduced by the ground effect

2.3 Autonomous tracking of moving objects with UAV

2.3.1 Non-learning-based methods

The related works for tackling moving objects for a UAV are mostly based on conventional non-learning and optimisation methods. For these works, they can be further categorized into two main streams: control-based algorithms and trajectory planning-based algorithms.

The control-based methods tackle the tracking task by directly computing optimal UAV control signals such as UAV x, y, z velocities and leave out the necessity of planning feasible trajectories. in [158], it uses a PD feedback controller to output control commands consisting of pitch, heading rate, and vertical velocity. The tracking error for the PD controller includes error components defined in the image space and the 3D world space. The control objective is threefold. The image space-based error components are designed to keep the target appearing in the image's horizontal centre and occupying enough pixels. The error component defined in the 3D world space is to require the quadrotor to stay at the same altitude as the target. Similar to [158], paper [159] computes heading, throttle, and pitch commands based on a cascade PID controller while the feedback error terms are all defined in image space to achieve the same control objective in [158]. Research [160] follows a similar pipeline as the above two works but with additional available control variables which are the pitch and angle of the camera gimbal.

These PID-based works are similar to the visual servoing technique and are generally limited to simple environments without obstacles and do not require avoiding obstacle collision or occlusion. Also, the target to be tracked is static or moves at a relatively low speed. While another control-based algorithm [161] does take environmental obstacles into consideration. It constructs a to-be-optimised objective function including collision avoidance cost and visibility cost in addition to the cost

of keeping the target in the image centre with a specified pixel size. The objective is optimised subject to the quadrotor dynamics. The controller follows the procedure of the horizon receding model predictive control method. This work however requires the real-time position information of the environment obstacles. The number of obstacles is also limited.

Trajectory planning-based algorithms, instead of optimising control commands, optimise trajectories that are collision-free and dynamically feasible. Research [162] builds a complete system including detection, localisation, and tracking modules. It assumes that the target is a sphere with a known radius. Thus the relative pose of the target can be determined from a single camera image with the camera projection model and nonlinear minimization problems. The trajectory of the target is presented by a polynomial which can be estimated by fitting with observations over some horizon steps. The estimated target trajectory will be used for target position prediction during the quadrotor tracker trajectory planning phase. A polynomial is used to represent the quadrotor's planned trajectory and is obtained by minimizing a carefully designed cost function. The quadrotor's to-be-optimised trajectory is also represented with a polynomial. Instead of simply constructing a cost function to minimize relative pose errors between the target and the quadrotor, this work proposes to minimize the velocity difference. It also sets up a constraint due to the field of view of the camera sensor such that the target will not leave the image. The camera field of view is approximated with an inscribed pyramid to formulate the constraint as a quadratic one. This will simplify the optimisation procedure. The quadratically constrained quadratic program (QCQP) is used to obtain the optimised tracking trajectory. With the trajectory estimation of the target and the trajectory planning of the quadrotor tracker, aggressive autonomous tracking is possible.

While the work introduced above does not have obstacles in the environment, work [163] includes the discussion of possible collision and occlusion caused by envi-

ronmental obstacles. A multi-objective cost function is constructed to optimise the tracker's trajectory which is represented by piece-wise polynomial functions in the B-spline form. The cost function is designed to reduce the relative distance between the target and the quadrotor and encourage smoothness while at the same time satisfying constraints including the dynamic constraint, the visibility constraint, the obstacle avoidance, and occlusion constraints. It assumes the obstacle to be in a sphere shape and requires the knowledge of position and size information of the obstacles. The target's moving ability is also limited.

In [164], it follows a similar methodology that optimises the trajectory by minimizing a multi-objective function. In order to solve this optimisation problem, a covariant gradient and steepest descent-based method is proposed. Unlike the works introduced above, It does not make assumptions about the shape of the obstacles. Real-world tests have shown the effectiveness and the robustness of the proposed algorithm. However, the environment is still quite simple and is not cluttered. The trajectory of the target is also easy to predict with straightforward moves. The planning also requires environment maps in advance.

The above works optimise directly a smooth trajectory for the quadrotor tracker and thus they only work in relatively simple environments. The difference between a path and a trajectory is that a path only carries position information while a trajectory has time information in addition to the position information.

In [165], it predicts a trajectory of the target over a horizon. After which, a reference trajectory for the tracker is produced by shifting the predicted target trajectory to a distance. Based on the reference trajectory, The search-based A^* is utilized to find a collision-free corridor near the reference trajectory. Finally, the tracking trajectory is determined with the quadratic programming optimisation algorithm similar to the works introduced above but subject to one more constraint which is that the trajectory has to lay within the obstacle-free corridor. This proposed algorithm is effective in clutter environments. It however does not consider

obstacle occlusion and the limited camera field of view constraints. The yaw trajectory is thus not optimised but is simply pointing to the target.

In [166], instead of directly planning a trajectory, it utilizes a graph-based search method to pre-plan a path represented by a set of viewpoints in sequence. A smooth trajectory is then fitted based on those pre-planned viewpoints. This two-step planning enables the algorithm to work in cluttered environments with arbitrary obstacle shapes. This however requires the construction of graphs and traversing through graphs during path planning, both of which consume large amounts of time. The weights of the edges between vortexes in the graph represent costs to fulfil obstacle avoidance and visibility requirements. Also, this work assumes that global environment maps are available and locations of the target are known over a future horizon. Thus it may not be suitable for more general applications with unfamiliar targets in unknown environments. A similar methodology is applied in their other work [167]. Instead of achieving obstacle or occlusion avoidance, they focus on optimising a trajectory that maximizes the colour difference between the target and the observed background for better target detection. In their following work [168], they create safe corridors based on the planned viewpoints which are then used to ensure collision avoidance during the smooth trajectory fitting process. Instead of knowing the target's future trajectory as a prior which is assumed to be available in their previous work [166], it also makes target path prediction based on history observation and the assumption that the target will avoid obstacles.

The coarse path planning phase of the works introduced above does not involve quadrotor dynamics cost. They simply apply acceleration or velocity constraints during the trajectory fitting phase. This may produce dynamically infeasible results. Also, they are planning globally which requires obstacle information of the whole environment as a prior. Research [9] uses kinodynamic search to include dynamic cost during the path planning phase. It plans locally with the current observed environments and replans the path if new obstacles are observed. This eliminates

the requirements of global environment maps and makes the algorithm feasible for unknown environments. For the generation of a smooth trajectory, the trajectory is presented by the use of the Bézier curve. The trajectory curve can then be obtained by optimising a set of control points.

As work [9] does not optimise the cost of target visibility regarding the camera's limited field of view and obstacle occlusion, they try to address this issue in their subsequent work [169]. In order to analytically represent the requirement of obstacle occlusion avoidance, they require a set of ball-shaped areas that cover the camera FOV to be free of obstacles. With the use of the Euclidean Signed Distance Field, this constraint can be differentiable and can thus be used for optimisation.

Those occlusion avoidance formulations introduced in the above works such as [164, 168, 169] often require constructing ESDF which contributes to the algorithm time cost. Work [170] redefined the visibility requirement from the view of the target which eliminates the need for (Euclidean Signed Distance Field) ESDF construction. The previous works pre-plan a path utilizing algorithms such as graph-searching or A^* to ensure collision avoidance and visibility which is then followed by a trajectory optimisation phase to refine the path by methods like path smoothing. This will result in inconsistency and may cause constraint violation. This work instead introduces those safety and visibility constraints in the trajectory optimisation phase.

Compared to control-based algorithms, trajectory planning-based algorithms can handle cluttered environments more naturally during problem formulations. By optimising trajectories, the algorithm can plan much deeper into the future such that they are capable of dealing with complex environments. However, this benefit comes with a cost which is that they require more computation resources and are more time-consuming. The overall system is also complex, where learning-based algorithms can reduce the complexity

2.3.2 Learning-based methods

To simplify the system, learning-based algorithms can help. There have been many works that use supervised learning or reinforcement learning for quadrotor control. A learning-based work [171] close to the task trains a vision-based drone with a position-based flocking control expert polity by imitation learning. The task is relatively simple just to follow a leader drone in an open space environment without obstacles. The behavior of the leader drone is not complex as well which also simplifies the task. Work in [172] proposes to use deep Q-learning to control a quadrotor following a ground vehicle involving discussion on obstacle avoidance, and limited field of view of sensors. The action space of the tracker and target are both discretized, however. The simulated quadrotor is not based on quadrotor dynamics but a simple 3D velocity model. It assumes the quadrotor has access to the position information of the environment information. Work in [173] focuses on noisy target relative position measurements. Authors in [174] extend the work into continuous action space based on the DDPG algorithm. The obstacle detection changes to a more realistic Lidar-based sensor. The quadrotor is still a simple 2D velocity model. Work [175] augments the algorithm by introducing Gaussian noise and Ornstein-Uhlenbeck (OU) noise to improve exploration. It also applies transfer learning such that the policy can be quickly adapted to new tracking tasks that are not presented during training. With similar environments and quadrotor settings, Research [176] introduced multiple quadrotors to track a single target. It applies a two-step training procedure whose action policy is first pre-trained by a PID control-based expert and then refined by the TD3 RL algorithm. The above works use simple non-real quadrotor physics models. Work [177] simulates the quadrotor based on real quadrotor dynamics and achieves real 3D obstacle avoidance and target tracking. The environment and test settings are still quite simple with a single obstacle whose position is known in advance. The target trajectories are simple direct lines or circles. The RL algorithm used is PPO algorithm.

As presented above, those works are based on very simple assumptions such as non-real quadrotor physics, no or simple environment obstacles, predictable target movements, etc. The environmental obstacle positions are known as inputs for decisions. Those assumptions may make the developed algorithms not capable of more real-world tracking tasks.

In real-world applications, cameras are usually equipped on drones for environment sensing because of their lightweight feature. This is adopted in work [178], this work uses a downlooking camera to track a ground vehicle. The constructed algorithm is end-to-end which takes in images as input and outputs speed commands to be carried out by the UAV controller. The reward space is designed to encourage the quadrotor to move to the target regarding distance and heading angle. There are no environmental obstacles involved. For environments with obstacles, the algorithms proposed in work [179] use camera images(RGB and Depth) together with other state information as inputs to achieve moving target tracking. The reward space is designed to avoid collisions, maintain a reasonable distance from the target, and keep the target pixels in the image centre. The simulator used is Airsim which provides realistic quadrotor physics and image rendering. This work however assumes the target moves on a fixed route. The action space is discretized into eight actions which commands the quadrotor to move along a direction at a fixed velocity and duration. Those settings undermined the agility of the quadrotors.

To achieve target tracking moving in random trajectories and occlusion caused by the obstacles, work [180] proposes a DQN-based tracking algorithm. The quadrotor is equipped with a down-looking camera and the target moves in indoor environments with items of furniture. The input is the observed RGB image. The target in such environments can be easily occluded by furniture. For example, the target may move under desks. In order to address such an issue, this work proposes a novel network structure. The observed image is first fed to a vision module consisting of several layers of convolution networks to produce low-dimension appearance

features. Such appearance features are not insufficient for decision-making when occlusion happens and the target disappears in the image. A motion module built by LSTM networks is used to encode a sequence of history and current appearance features from the vision module and history quadrotor actions to produce motion features. This will help estimate the target pose. The motion features together with the current appearance feature are fed to a decision module built by several fully connected layers to output action commands to be executed by the quadrotor. The reward space only has one component to reduce the distance between the target and the quadrotor.

Although this work has achieved good performance, there is no collision avoidance involved as the quadrotor flies above the furniture. Also, it can not actively avoid occlusion. This thesis also aims to solve these two problems and meet other requirements at the same time.

2.4 Research Gap

After reviewing the literature presented above, several limitations have been identified and are discussed below to justify the motivation behind this thesis.

While the works highlighted in the discussion have shown promising outcomes, it is noteworthy that, to the best of the author's knowledge, few learning-based algorithms have taken into account the impact of localisation performance on the overall navigation results. These approaches typically assume access to ground-truth robot poses throughout navigation, resulting in policies that resemble shortest-path strategies. This assumption is not valid in many real-world applications, where robot poses are not always available, such as under GPS-denied environments. Instead, real-world applications rely on localisation algorithms using external sensor observations, such as Lidar scans or camera images. This assumption made in most existing research works does not consider the quality of localisation along the navi-

gation trajectories. Essentially, in such works, robot perception and the navigation strategy are decoupled. This decoupling introduces the risk of severe navigation failures, especially when planned paths traverse featureless regions where localisation algorithms may fail.

Hence, this thesis removes the unrealistic assumption of availability of ground-truth robot poses and, instead, employs observation-based localisation algorithms, such as Orb-slam [11] or Hector slam [181], for pose estimation. This work makes efforts to develop localisation-safe navigation algorithms for both UGVs and UAVs to bridge this research gap.

For the UAV tracking task, conventional non-learning-based trajectory planning algorithms have demonstrated promising results. Nevertheless, these algorithms tend to construct a complex system, demanding substantial computational resources and time. On the other hand, learning-based end-to-end UAV control pipelines offer the potential to reduce system complexity and computation time significantly. However, most learning-based tracking policies, as discussed above, rely on rather simplistic assumptions that include non-realistic dynamics, the absence or simplicity of environmental obstacles, and predictable target movements. Notably, the assumption that environmental obstacle positions are known inputs for decision-making might limit the capability of these algorithms to handle more real-world tracking scenarios.

Therefore, to bridge the research gap identified in existing literature regarding learning-based tracking algorithms, this work also focuses on addressing the tracking task of a randomly moving target in complex environments full of randomly distributed obstacles for UAVs using visual inputs. This removes the assumption of simple environment settings commonly found in the literature. Additionally, this work will model physical UAV dynamics, rather than relying solely on simple velocity models as used in previous research. These settings aim to help develop tracking policies applicable to real-world applications and alleviate the simulation-to-reality

gap encountered by methods in the existing literature. Moreover, it has been observed that training an end-to-end tracking policy network with high-dimensional depth images as inputs may encounter local minima. To tackle this challenge, this thesis proposes a novel solution leveraging a teacher-student reinforcement learning framework to enhance the effectiveness of training the visual robot agent.

2.5 Summary

In this chapter, a detailed literature review is presented to explore various implementations of state-of-the-art reinforcement learning algorithms within the domains of mobile robot navigation, UAV control and autonomous tracking of targets by UAVs, offering a nuanced understanding of the advancements and methodologies employed in the existing body of research. Also, research gaps found in the existing literature are identified and discussed.

Chapter 3

Preliminary

This chapter introduces several preliminary concepts, including reinforcement learning and robot localisation algorithms, to enhance understanding of this thesis.

It begins with introducing key concepts formulated under the reinforcement learning framework in Section 3.1. This overview aims to enhance the comprehension of RL-based navigation and control methods developed in this thesis, which will be presented in subsequent chapters.

Following this section, Section 3.2 will introduce two prominent practical RL algorithms: DQN and PPO, which form the foundation of the algorithms developed in this thesis. It is followed by the description of the key design components for training RL-based policies that this thesis will focus on.

Robot localisation algorithms utilised in this thesis are introduced in Section 3.3. This introduction is necessary to comprehend the primary causes of localisation failures, which inspire the proposed work on strategies to avoid such localisation failures.

Section 3.4 concludes this chapter.

3.1 Reinforcement learning overview

Reinforcement learning trains agents by interacting with environments and optimising strategies (policies) aiming to collect maximum cumulative rewards from the environments.

During the learning process, the agent will be deployed to do explorations by taking actions based on observations in the environments to collect experience trajectories. Those collected trajectories will be stored in replay buffers and be used to optimise the action strategy. The agent will then exploit the improved strategy to collect trajectories to achieve higher reward returns and the procedure will iterate until the optimisation converges. Hence this is why the term *'reinforcement'* is used [182]. The strategy of the reinforcement learning agent can be represented by some mathematical models like parameterised distributions [183] or table matrices [182]. In order to achieve a better representation of the strategy with high-dimension state or actions, by utilising the approximation capability of the deep learning techniques, the strategy can be presented by deep neural networks [184], where the term *'deep reinforcement learning'* comes from.

To describe reinforcement learning in mathematical formulation, the next section will first introduce some key concepts used to define the problem that reinforcement learning tries to solve. After this, some important reinforcement learning algorithms will be discussed. This will help readers understand the foundations of algorithms developed in the literature and the methods proposed in the following chapters of this thesis.

3.1.1 Key concepts

The interaction process between agents and environments in reinforcement learning is described by the MDP mathematical framework. An MDP can be represented by a tuple $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, P, R, \gamma \rangle$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ denotes the action space,

the P describes the transition distribution model, R describes the reward function and γ is the future reward discount factor.

A vital assumption of the MDP is that the future system states depend only on the current state. The current state has all the information that is required to decide the future without the need for historical information:

$$\mathbb{P}[S_{t+1}|S_t] = \mathbb{P}[S_{t+1}|S_1, \ldots, S_t] \tag{3.1}$$

Where $\mathbb{P}$ is a symbol of probability.

Hence the transition model P, which is usually unknown, represents the transition probability from state s to state s' by taking action a while receiving reward feedback r:

$$P(s', r|s, a) = \mathbb{P}[S_{t+1} = s', R_{t+1} = r|S_t = s, A_t = a] \tag{3.2}$$

The definition of the state transition function can be written as :

$$P^a_{ss'} = P(s'|s, a) = \mathbb{P}[S_{t+1} = s'|S_t = s, A_t = a] = \sum_{r\in\mathcal{R}} P(s', r|s, a) \tag{3.3}$$

The reward function R predicts the reward r received after taking an action a based on state s:

$$R(s, a) = \mathbb{E}[R_{t+1}|S_t = s, A_t = a] = \sum_{r\in\mathcal{R}} r \sum_{s'\in\mathcal{S}} P(s', r|s, a) \tag{3.4}$$

The policy used to decide the action based on state observation is denoted as π, which is the behaviour function of the agent during the MDP process. The policy π maps state s to an action a, which can be either deterministic functions $\pi(s) = a$ or stochastic distributions $\pi(a|s) = \mathbb{P}_\pi[A = a|S = s]$.

The goal of reinforcement learning is to maximise cumulative rewards within an episodic trajectory, which usually penalises the future rewards with the discounting

factor $\gamma \in [0, 1]$:

$$G_t = R_{t+1} + \gamma R_{t+2} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{3.5}$$

Furthermore, the following definition has to be made in order to describe how reinforcement learning improves the behaviour strategy.

The **state-value function** $V_\pi(s)$ represents the expectation of cumulative future returns when the agent starts from state s based on a policy π:

$$V_\pi(s) = \mathbb{E}_\pi[G_t | S_t = s] \tag{3.6}$$

While the **action-value** (Q-value) function $Q_\pi(s, a)$ is the expected return after taking an action a based on current state s and a policy π:

$$Q_\pi(s, a) = \mathbb{E}_\pi[G_t | S_t = s, A_t = a] \tag{3.7}$$

The relation between the state value and the action value:

$$V_\pi(s) = \sum_{a \in \mathcal{A}} Q_\pi(s, a)\pi(a|s) \tag{3.8}$$

Additionally, an **advantage function** is used to calculate the difference between the action value function and the state value function, which implies how much better on average an action is than others :

$$A_\pi(s, a) = Q_\pi(s, a) - V_\pi(s) \tag{3.9}$$

Bellman equations are a set of equations which are used to calculate the value function at time step t by the sum of immediate reward and the value function of the next time step $t + 1$:

$$
\begin{aligned}
V(s) &= \mathbb{E}[G_t|S_t = s] \\
&= \mathbb{E}[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots |S_t = s] \\
&= \mathbb{E}[R_{t+1} + \gamma(R_{t+2} + \gamma R_{t+3} + \dots)|S_t = s] \\
&= \mathbb{E}[R_{t+1} + \gamma G_{t+1}|S_t = s] \\
&= \mathbb{E}[R_{t+1} + \gamma V(S_{t+1})|S_t = s]
\end{aligned}
\tag{3.10}
$$

Similarly, the equation for Q-value:

$$
\begin{aligned}
Q(s,a) &= \mathbb{E}[R_{t+1} + \gamma V(S_{t+1}) \mid S_t = s, A_t = a] \\
&= \mathbb{E}[R_{t+1} + \gamma \mathbb{E}_{a\sim\pi} Q(S_{t+1}, a) \mid S_t = s, A_t = a]
\end{aligned}
\tag{3.11}
$$

3.1.2 Value-based RL algorithms

Instead of optimising the policy directly, the value-based RL algorithms optimise the agent behaviour through value functions $V(s)$ or $Q(s,a)$ estimation. The estimation of these value functions is also the foundation of other reinforcement algorithms.

Monte-Carlo method

Monte-Carlo (MC) methods approximate the value functions $V(s) = \mathbb{E}[G_t|S_t = s]$ or $Q(s,a) = \mathbb{E}[G_t|S_t = s, A_t = a]$ by computing the observed mean return from experience data collected through interactions with environments. This kind of method requires full complete episodes $S_1, A_1, R_2, ..., S_T$, where T is the terminal time step. To take into account multiple visits of state s in one episode, The mean return for state s taking action a is

$$
Q(s,a) = \frac{\sum_{t=1}^{T} \mathbb{1}[S_t = s, A_t = a] G_t}{\sum_{t=1}^{T} \mathbb{1}[S_t = s, A_t = a]}
\tag{3.12}
$$

where $\mathbb{1}$ is a binary indicator and $G_t = \sum_{k=0}^{T-t-1} \gamma^k R_{t+k+1}$ can be computed from the data collected during each episode.

The optimal policy will be

$$\pi(s) = \arg\max_{a \in \mathcal{A}} Q(s, a) \tag{3.13}$$

It will be improved along with the approximation of the Q-value.

During trajectory collections, the agent usually uses a behaviour policy called $\epsilon-$ *greedy*. It selects an action based on current approximation $a = \arg\max_{a \in \mathcal{A}} Q(s, a)$ with a probability of ϵ, otherwise it will randomly select an action within the action space. Through enough cycles of iterations, the approximation of the Q-value will converge to the optimal value.

One obvious disadvantage of the Monte-Carlo method is that it is only suited for problems with short horizons, small state space and action space. Otherwise, it will take a long time to roll out a complete episode trajectory and update the value functions. The value estimation will also be of high variance as there are too many possible state-action pairs with large state spaces and action spaces, which require a large number of sample trajectories to cover.

Hence, temporal-difference (TD) methods are proposed such that the update of the value functions can be done with every single transition $(s_t, a_t, s_{t+1}, r_{t+1})$ the agent collects during exploration, which eliminates the requirement of a full episode trajectory and reduces the estimation variance [185].

On-policy Temporal-Difference (TD) method

The value estimation based on TD methods relies on the Bellman equation Eq. 3.10 and Eq. 3.11 such that the value function $V(S_t)$ can be estimated by $R_{t+1} + \gamma V(S_{t+1})$

(refer to TD target) with a single collected transition tuple $(s_t, a_t, s_{t+1}, r_{t+1})$:

$$\begin{aligned} V_{k+1}(S_t) &\leftarrow (1-\alpha)V_k(S_t) + \alpha G_t \\ V_{k+1}(S_t) &\leftarrow V_k(S_t) + \alpha(G_t - V_k(S_t)) \\ V_{k+1}(S_t) &\leftarrow V_k(S_t) + \alpha(R_{t+1} + \gamma V_k(S_{t+1}) - V_k(S_t)) \end{aligned} \tag{3.14}$$

Where α is the learning rate that controls the update extent of the value function based on the sample, k denotes the update step.

Similarly, for Q-value function estimation:

$$Q_{k+1}(S_t, A_t) \leftarrow Q_k(S_t, A_t) + \alpha(R_{t+1} + \gamma Q_k(S_{t+1}, A_{t+1}) - Q_k(S_t, A_t)) \tag{3.15}$$

Those values $V_k(t+1)$ and $Q_k(t+1)$ used to update the value functions are based on existing approximation (i.e. from update step k), which is different from the MC method which calculates the value functions from a full trajectory (i.e. Eq. 3.12). This technique is referred to as bootstrapping.

Sarsa RL algorithmm The Sarsa algorithm uses Eq. 3.15 to update the Q-value approximation at every single transition collected at each time step. It uses the same $\epsilon - greedy$ exploration policy to collect data as that of the MC method described above. Compared to MC methods, Sara can achieve faster convergence when handling problems with long horizons as well as problems with large action or state spaces. The overall procedure is described as follows:

1. Initialisation of values of Q function; initial state S_0; set t=0.
2. Select an action $a_0 = \arg\max_{a \in \mathcal{A}} Q(s_0, a)$ with probability of ϵ otherwise randomly selects $a_0 \in \mathcal{A}$.
3. For each time step, apply action a_t, and receives reward r_{t+1} and new state s_{t+1}.
4. Select the next action based on the new state with the same $\epsilon - greedy$ method

of step 2: $a_{t+1} = \arg\max_{a \in \mathcal{A}} Q(s_{t+1}, a)$ or randomly selects $a_{t+1} \in \mathcal{A}$. This is the action to be executed for the next time step.

5. Update the action value function Q based on $(s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1})$ with Eq. 3.15: $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(r_{t+1} + \gamma Q_(s_{t+1}, a_{t+1}) - Q_(s_t, a_t))$

6. Increase $t = t + 1$ and iterate from step 3.

Both the MC method and the Sarsa algorithm update the value functions with transitions collected from the policy to be updated. This means that the policy that generates the action decisions during the data collection phase is identical to the policy that is to be updated based on the collected data, which is known as on-policy training. When the exploration policy is different from the policy to be updated, it is called off-policy training. The on-policy algorithm will only use the collected data once and discard the data after updating the value functions on that data. Hence, the on-policy algorithms may suffer from sample efficiency as the collected data is not reusable.

Hence an off-policy algorithm named Q-learning is proposed [186].

Off-policy TD method: Q-learing

The Q-learning, in contrast to the Sarsa algorithm, updates the Q-value function with a modification of Eq. 3.15:

$$Q_{k+1}(s_t, a_t) \leftarrow Q_k(s_t, a_t) + \alpha(r_{t+1} + \gamma \max_{a \in \mathcal{A}} Q_k(s_{t+1}, a) - Q_k(s_t, a_t)) \tag{3.16}$$

The learning procedure works as follows:

1. Initialisation of values of Q function; initial state S_0; set t=0.

2. For each time step, select an action $a_t = \arg\max_{a \in \mathcal{A}} Q(s_t, a)$ with probability of ϵ otherwise randomly selects $a_t \in \mathcal{A}$.

3. After the agent applies the action a_t, it receives reward r_{t+1} and new state s_{t+1}.

4. Update the action value function Q based on $(s_t, a_t, r_{t+1}, s_{t+1})$ with Eq. 3.16: $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(r_{t+1} + \gamma \max_{a \in \mathcal{A}} Q_(s_{t+1}, a) - Q_(s_t, a_t))$

5. Increase $t = t + 1$ and iterate from step 2.

The key difference between Sarsa and Q-learning is that Q-learning does not utilise the current policy to decide a next-time step action a_{t+1}. It uses the best Q-value of the next state over the action space to update the Q value without considering which action (denoted as a^*) can achieve that maximal Q, and in the next time step, it may not necessarily follow that a^* [187] thus the algorithm is off-policy.

3.1.3 Policy gradient

In contrast to the value-based methods introduced above which try to estimate the value functions and find the optimal actions based on the estimated value functions, policy gradient algorithms aim to model and optimise a policy directly with interactions with the environments. The to-be-optimised policy is usually denoted as $\pi_\theta(a|s)$ parameterised by θ and is required to be differentiable. The objective function that parameter θ tries to optimise is defined by Eq. 3.17 which represents the cumulative reward for the episodic case from the start state shown as follows.

$$\begin{aligned} J(\theta) &= V^\pi(s_0) \\ &= \sum_{a \in \mathcal{A}} \pi_\theta(a|s) Q^\pi(s, a) \end{aligned} \tag{3.17}$$

For simplicity's sake, the parameter θ will be omitted from the policy symbol π_θ when the policy occurs in the subscript of other functions. Taking the above equation as an example, Q^π and V^π represent Q^{π_θ} and V^{π_θ} respectively.

To find the parameter that θ that maximises the objective function Eq. 3.17, the iterative optimisation algorithm gradient ascent can be used to update the parameter θ by taking gradient on the objective function:

$$\theta_{i+1} = \theta_i + \alpha \nabla J(\theta_i) \tag{3.18}$$

where i and $i+1$ denotes update step; α is the learning rate. The gradient indicates the direction following which the parameter θ change can improve the expected returns.

In contrast to value-based methods, the policy gradient algorithms which optimise the policy directly are more favoured for problems with continuous state or action space, because the number of state-action pairs will be infinite for such problems. Therefore, the estimation of the value functions (the state value function V and/or the action value function Q) for problems with continuous space will be computationally expensive or infeasible. To find an optimal from Q-value $a_t = \arg\max_{a \in \mathcal{A}} Q(s_t, a)$ is also not trivial, which is why the prominent algorithm DQN can only work with discrete action spaces.

Policy Gradient Theorem

According to the *policy gradient theorem* [188], the gradient of the objective function Eq. 3.17 has the following form:

$$\begin{aligned}
\nabla_\theta J(\theta) &= \nabla V^\pi(s_0) \\
&= \sum_s \sum_{k=0}^{\infty} \rho^\pi(s_0 \to s, k) \sum_{a \in \mathcal{A}} Q^\pi(s,a) \nabla_\theta \pi_\theta(a|s) \\
&\propto \sum_{s \in \mathcal{S}} d^\pi(s) \sum_{a \in \mathcal{A}} Q^\pi(s,a) \nabla_\theta \pi_\theta(a|s) \\
&= \mathbb{E}_\pi[\sum_{a \in \mathcal{A}} Q^\pi(s,a) \nabla_\theta \pi_\theta(a|s)]
\end{aligned} \tag{3.19}$$

where $\rho^{\pi}(s_0 \to s, k)$ represents the probability of transitioning from a state s_0 to the state s in k steps with policy π_θ; and $d^{\pi}(s)$ represents the stationary distribution of the agent state under the policy π_θ. Policies update using Eq. 3.19 is called *all-actions* method as it computes the gradient over the whole action space.

REINFORCE

The REINFORCE further replaces the sum over the action space by an expectation under the police π, and estimating the expectation by sampling:

$$\begin{aligned}\nabla_\theta J(\theta) &\propto \sum_{s\in\mathcal{S}} d^{\pi}(s) \sum_{a\in\mathcal{A}} \pi_\theta(a|s) Q^{\pi}(s,a) \frac{\nabla_\theta \pi_\theta(a|s)}{\pi_\theta(a|s)} \\ &= \mathbb{E}_\pi[Q^{\pi}(s,a) \nabla_\theta \ln \pi_\theta(a|s)] \qquad \text{; as } (\ln x)'=1/x \\ &= \mathbb{E}_\pi[G_t \nabla_\theta \ln \pi_\theta(a|s)]\end{aligned} \tag{3.20}$$

The REINFORCE method computes the return G_t with a full trajectory from time t which makes it a Monte-Carlo method. Hence, the following equation is used to update the policy parameter θ iteratively.

$$\theta \leftarrow \theta + \alpha \gamma^t G_t \nabla_\theta \ln \pi_\theta(A_t|S_t) \tag{3.21}$$

Here, γ^t is introduced because Eq. 3.19 is derived based on the assumption that $\gamma = 1$.

The full learning process is shown as follows:

1. Initialize the parameter θ of the policy.
2. Roll out a full trajectory with π_θ: $S_1, A_1, R_2, ..., S_T$.
3. For $t = 1, 2, ..., T$:
    - 1. Compute the return G_t
    - 2. Update θ according to Eq. 3.21

4. Repeat step 2 with updated π_θ

Actor-Critic

According to Eq. 3.20:

$$\nabla_\theta J(\theta) = \nabla V_\theta(s_0) \propto \mathbb{E}_\pi[Q^\pi(s,a)\nabla_\theta \ln \pi_\theta(a|s)] \tag{3.22}$$

It has been discussed that it is natural that one may use a nonlinear function to approximate the Q-value function. With prediction from the approximation function, the policy can be updated with every transition tuple without the need for waiting for a complete trajectory as that of the REINFORCE algorithm. This is why the actor-critic methods are introduced.

- **Critic** optimises the value approximator parameters w to improve the approximation of the value functions $Q_w(a|s)$ or $V_w(s)$ with loss defined by Bellman equations Eq. 3.11 and Eq. 3.10. It is called *critic* as it controls how much the objective function J (the expected return $V(s_0)$) can improve along the gradient direction as shown in Eq. 3.22.

- **Actor** optimises the policy parameters θ with $Q_w(a|s)$ according to Eq. 3.22. *Actor* means that it controls the agent's actions to collect rewards.

It has been discussed in [182] that this vanilla policy gradient method will be of high variance. It also suggests that baselines $b(s)$ that will not change with actions can be introduced to mitigate the high variance problem, which modifies the gradient Eq. 3.20: to the equation shown as follows:

$$\begin{aligned}\nabla_\theta J(\theta) &\propto \mathbb{E}_\pi[(Q^\pi(s,a) - V(s))\nabla_\theta \ln \pi_\theta(a|s)] \\ &= \mathbb{E}_\pi[A^\pi(s,a)\nabla_\theta \ln \pi_\theta(a|s)]\end{aligned} \tag{3.23}$$

where $A^\pi(s,a) = Q^\pi(s,a) - V^\pi(s)$ is the advantage function described in Section 3.1.1. Introducing the baseline $V(s)$ will not lead to a biased estimation of

the gradient as proved in [182].

3.2 Practical RL algorithms implementation

This section will first introduce two RL algorithms, DQN and PPO, upon which the algorithms proposed in this thesis are based. After this, the design focus of the RL training procedure this thesis follows will be discussed.

3.2.1 DQN

In theory, a table (known as Q-table) can be used to record the value of Q of all possible state-action pairs belonging to the state space and action space. This, however, quickly becomes impossible or computationally infeasible under the scenario where the state space is large or continuous. Therefore, researchers propose to utilise parameterised functions to serve as the approximation of the Q values, which is called function approximation. The Q value function parameterised with θ can be denoted as $Q(s, a; \theta)$.

With the improvement of deep learning techniques, the Deep Neural Network (DNN) shows great capability of approximating non-linear functions with high-dimensional inputs. Hence, research works have suggested the use of DNNs to approximate the Q value.

The most prominent work combining the DNNs technique with Q-learning is the DQN proposed in work [184], which achieves human-level performance on Atari games.

DQN proposes two key innovative techniques to address the issues of instability or even divergency introduced by the use of nonlinear function approximator [189]: experience replay and periodically updated target.

- **Experience Replay**: A replay buffer $D = e_1, ...e_t$ is designed to store all experience tuples collected during training, where $e_t = (s_t, a_t, r_t, s_{t+1})$ denotes

the experience data at time step t. Hence the replay buffer D stores the experience data over many different episodes. During the update of the Q value, it draws samples randomly from the experience pool D. The use of experience replay can improve data efficiency as one sample can be revisited multiple times due to the random sampling procedure. It can also remove correlations in the observation sequence, which makes the learning unstable [184].

- **Periodically Updated Target** The target values which are used to optimise the Q value are only updated periodically. For every C step, the target Q-value network clones the parameters from the Q network and is frozen between two consecutive cloning steps. This will help stabilise the training procedure and avoid short-term oscillations.

The loss defined to train the DNN parameters is the application of the Bellman equation Eq. 3.11 and Q-learning Eq. 3.16 shown as follows:

$$\mathcal{L}(\theta) = \mathbb{E}_{(s,a,r,s')\sim U(D)}\Big[\big(r + \gamma \max_{a'} Q(s', a'; \theta^-) - Q(s, a; \theta)\big)^2\Big] \tag{3.24}$$

where $(s, a, r, s') \sim U(D)$ denotes the uniform sampling procedure in the replay buffer D; θ is to-be-learned parameters of the Q networks; θ^- denotes the parameters of the target Q networks and is frozen during the update of θ.

3.2.2 PPO

Although policy gradient methods are well-defined, the authors of work [190] argue that it often results in large policy updates which lead to performance collapse. TRPO proposed in [191] based on [192] put a hard constraint on the policy update size at every policy update step. The constraint is constructed by measuring the difference between the old policy and the new policy with KL divergence.

As described in [191], it optimised an objective function defined by advan-

tage [193]:

$$\begin{aligned} J(\theta) &= \sum_{s\in\mathcal{S}} \rho^{\pi_{\theta_{\text{old}}}} \sum_{a\in\mathcal{A}} \big(\pi_\theta(a|s)\hat{A}_{\theta_{\text{old}}}(s,a)\big) \\ &= \sum_{s\in\mathcal{S}} \rho^{\pi_{\theta_{\text{old}}}} \sum_{a\in\mathcal{A}} \big(\beta(a|s)\frac{\pi_\theta(a|s)}{\beta(a|s)}\hat{A}_{\theta_{\text{old}}}(s,a)\big) \\ &= \mathbb{E}_{s\sim\rho^{\pi_{\theta_{\text{old}}}},a\sim\beta}\big[\frac{\pi_\theta(a|s)}{\beta(a|s)}\hat{A}_{\theta_{\text{old}}}(s,a)\big] \end{aligned} \tag{3.25}$$

where $\beta(a|s)$ is the behaviour policy used to collect samples. $\hat{A}(s,a)$ means the estimation of the advantage, which is usually approximated by nonlinear functions such as deep neural networks in deep reinforcement learning.

In TRPO, it utilises KL-divergence to measure the difference between two policies before and after an update and requires the difference to be smaller than a defined threshold:

$$\mathbb{E}_{s\sim\rho^{\pi_{\theta_{\text{old}}}}}[D_{\text{KL}}(\pi_{\theta_{\text{old}}}(.|s)\|\pi_\theta(.|s)] \leq \delta \tag{3.26}$$

During implementation, usually $\pi_{\theta_{\text{old}}}$ is used as behaviour policy, which leads to:

$$\begin{aligned} \theta = \arg\max_\theta J(\theta) = \arg\max_\theta \mathbb{E}_{s\sim\rho^{\pi_{\theta_{\text{old}}}},a\sim\pi_{\text{old}}}\big[\frac{\pi_\theta(a|s)}{\pi_{\text{old}}(a|s)}\hat{A}_{\theta_{\text{old}}}(s,a)\big] \\ s.t.\ \mathbb{E}_{s\sim\rho^{\pi_{\theta_{\text{old}}}}}[D_{\text{KL}}(\pi_{\theta_{\text{old}}}(.|s)\|\pi_\theta(.|s)] \leq \delta \end{aligned} \tag{3.27}$$

While the TRPO algorithm is complicated, PPO advocates for a similar idea of setting constraints on the policy update and uses a clipped surrogate objective function to simplify the pipeline.

The policy update of the PPO is via:

$$\begin{aligned} \theta &= \arg\max_\theta \mathop{\mathrm{E}}_{s,a\sim\pi_{\theta_{\text{old}}}}[L(s,a,\theta_{\text{old}},\theta)] \\ L(s,a,\theta_{\text{old}},\theta) &= \min\left(\frac{\pi_\theta(a|s)}{\pi_{\theta_{\text{old}}}(a|s)}A^{\pi_{\theta_{\text{old}}}}(s,a), g(\epsilon, A^{\pi_{\theta_{\text{old}}}}(s,a))\right) \end{aligned} \tag{3.28}$$

in which ϵ is a small hyperparameter and g is the clipping function:

$$g(\epsilon, A) = \begin{cases} (1+\epsilon)A & A \geq 0 \\ (1-\epsilon)A & A < 0. \end{cases} \tag{3.29}$$

The PPO method is relatively easy to implement and has achieved similar performance compared to the TRPO method. It is the algorithm this thesis frequently used in the following sections.

3.2.3 Training RL policies

The overall RL procedure is depicted in Fig. 3.1.

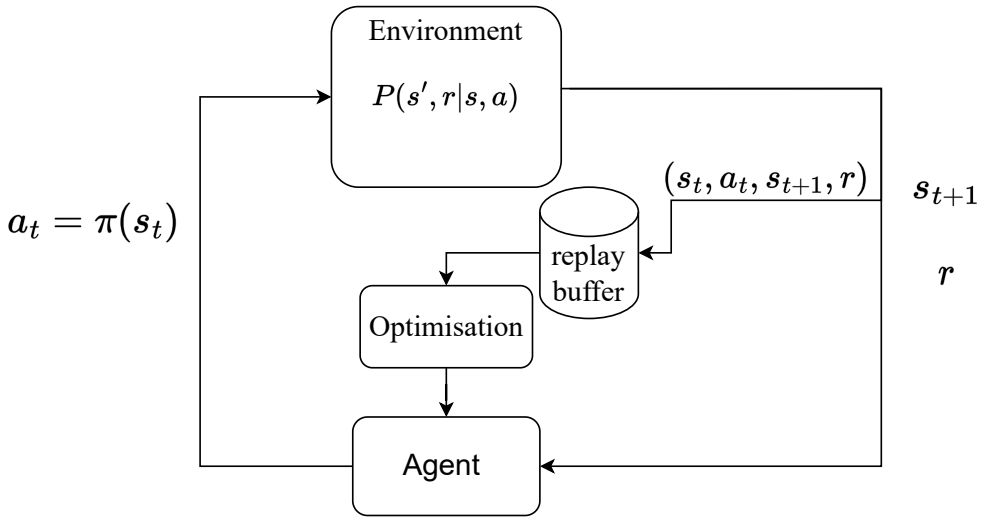


Figure 3.1: The RL interaction procedure: at time step t, the agent takes an action based state s_t and policy π. The environment will then produce the reward feedback r and transfer to a new state s_{t+1}. The agent receives the reward and the new state. The process will then iterate. The transition pair $(s_t, a_t, s_{t+1}, r_{t+1})$ will be also stored in the replay buffer. The policy will be updated according to the samples from the replay buffer to achieve a higher cumulative return G.

The following paragraphs will describe vital components of optimising RL-based policies.

Training Environments The training of RL policies requires a large number of experience data, which will consume an unacceptable amount of time if the training procedures are all performed in real-world environments. Also, during this trial-

and-error training procedure, robots may act unexpectedly and cause damage to the environment or themselves. Hence, most RL-based policies are first trained in simulation. The simulator engines have to be capable of modelling real-world physics and rendering various sensor outputs like Lidar scans or camera images.

The developers have to construct environments to model the robot tasks using simulators. In this thesis, Gazebo [194] and Flightmare [195] are used for training environment construction for ground and aerial vehicles, respectively. They can model the real-world physical dynamics of both ground and aerial vehicles. Also, they support communication and integration with the Robot Operating System (ROS) [196], which is widely applied for real-world robots. Hence, this thesis will also develop algorithms using ROS. This feature can help transfer the trained policy to real-world robots conveniently with minimum effort. This thesis will use them to design indoor and outdoor environments such as offices, lakes or forests for navigation and tracking tasks.

Action Space The action space can be continuous or discrete. For example, a robot moving with a continuous action space $a_t \in [0 \sim 1]m/s$ means that the robot velocity can be any value between 0 and 1. A discretised action space will be a set of values $a_t \in \{0, 0.5, 1.0\}m/s$, where the robot can only move at the velocity selecting one value from these 3 possible values.

It can be seen that using continuous action space can make the robot move smoother and make full use of the robot's agility. Training policies with continuous action space will request a larger amount of data as the training procedure has to sample actions over the entire continuous action range instead of sampling from a set of discrete action values, as is the case with a discrete action space. Also, agility may even be detrimental to the task. It will be shown in Chapter 5 that agile UAV movements will deteriorate localisation performance. However, a discrete action space will not be enough for a tracking task of UAVs because the target may move quickly and randomly and environments are full of randomly distributed obstacles.

The action space should be carefully designed. This thesis will design the action space to meet the specific requirements of the task.

Reward Space The design of the reward space is vital because it serves as the criterion to justify the decisions made by the policies. Based on this criterion, the policies can be optimised. The reward space should be designed to encourage actions that facilitate task accomplishment and penalize actions that hinder it. Additionally, reward signals should be provided at the right moments to prevent confusion for the robots. Typically, the reward space consists of several different components for various objectives, such as goal-reaching or obstacle avoidance. Therefore, the scales of different reward components are important, as they could lead to either too aggressive or too conservative behaviours.

The absence of certain reward signals will render the robot unable to acquire the expected abilities, which will be discussed in detail in Chapter 4 and Chapter 5 regarding localisation-safe behaviour. A novel reward space design for the tracking task will be illustrated in Chapter 6.

Training data collection Training policies with the DQN algorithm, which is off-policy, can use directly any experience data produced at any stage of the whole training procedure. Hence, any experience data can be stored in a replay buffer as long as the storage memory capacity allows.

While the DQN algorithm is only available for discrete action spaces, the policy gradient algorithm PPO is selected for tasks with continuous action spaces due to its stable performance. As an on-policy method, PPO requires that training data come exclusively from the current policy, eliminating the possibility of learning from data collected by other experienced agents directly. This thesis will propose a novel PPO-based algorithm such that data from a teacher policy can be reused to train a student policy, which will be discussed in detail in Chapter 6.

3.3 Localisation algorithms

The localisation algorithms vary for indoor and outdoor navigation based on the measurement sensor used. The subsequent subsections will provide distinct descriptions of these algorithms for each navigation scenario.

3.3.1 FastSLAM

In the context of outdoor navigation for a ground vehicle used in Chapter 4, it will utilise distance measurement sensors with known data associations, such as UWB, AprilTag and RGBD image features. This implies that the algorithm has the knowledge of information about which landmark a specific measured distance corresponds to.

The localisation algorithm employed is the FastSLAM algorithm [197], specifically a feature-based Rao-Blackwellized Particle Filter (RBPF) algorithm. In brief, within the Rao-Blackwellized Particle Filter framework, the joint probability of the map m and the robot poses x is factorized through Rao-Blackwellization, formulated as follows:

$$p(x_{1:t}, m|z_{1:t}, u_{1:t-1}) = p(m|x_{1:t}, z_{1:t})p(x_{1:t}|z_{1:t}, u_{1:t-1}) \tag{3.30}$$

where z and u represent the measurement and the control input respectively; and t is the time steps. Such factorisation allows the SLAM algorithm to estimate the trajectory $p(x_{1:t}|z_{1:t}, u_{1:t-1})$ of the robots first and subsequently process the mapping $p(m|x_{1:t}, z_{1:t})$ with the estimated poses.

The estimation of the trajectory $p(x_{1:t}|z_{1:t}, u_{1:t-1})$ using a particle filter method involves maintaining and updating a set of weighted particles $\mathcal{X} = \{< x^k, w^k > \}_{k=1,...,N}$ in accordance with the observations. Each particle in this set represents a potential estimation of the robot's trajectory. Additionally, each individual particle maintains a map estimation based on the estimated trajectory of that specific particle and the observations.

The update of those particles of the FastSlam algorithm is based on the sampling importance resampling (SIR) filter, which incrementally estimates the robot trajectory and the map based on the incoming received sensor measurements. The overall procedure consists of four steps: sampling, importance weighting, map estimation and resampling [198].

1. **Sampling**: the algorithm sample the generation t of particles $\mathcal{X}_t = \{< x_t^k, w_t^k >\}_{k=1,\dots,N}$ from a distribution based on the previous generation $t-1$: $\mathcal{X}_{t-1} = \{< x_{t-1}^k, w_{t-1}^k >\}_{k=1,\dots,N}$. As it is difficult to sample from the target distribution $p(x_{1:t}|z_{1:t}, u_{1:t-1})$, it is suggested to sample from a proposal distribution q. The proposal distribution q used in the FastSlam is the probabilistic motion model: $p(x_t|x_{t-1}, u_{t-1})$.

2. **Importance weighting**: As shown in the sampling step, the proposal distribution q differs from the target distribution$p(x_{1:t}|z_{1:t}, u_{1:t-1})$. A weight w^k has to be applied on the associated particle k. This weight is computed based on the importance sampling principle following a recursive formulation as shown in Eq. 3.31 (refer to paper [198] for detail).

$$\begin{aligned} w_t^k(x_t^k) &= \frac{target(x^k)}{proposal(x^k)} \\ &= \frac{p(x_{1:t}^k|z_{1:t}, u_{1:t-1})}{p(x_t^k|x_{t-1}^k, u_{t-1})} \\ &\propto p(z_t|m_{t-1}^k, x_t^k) \times w_{t-1}^k \\ &= |2\pi Q|^{-\frac{1}{2}} \exp\{-\frac{1}{2}(z_t - \hat{z}^k)^T Q^{-1}(z_t - \hat{z}^k)\} \times w_{t-1}^k \end{aligned} \tag{3.31}$$

where $\hat{z}^k$ represents the expected measurement calculated from the map and pose of particle k, and Q is the innovation covariance. When the actual measurement z^t aligns with the expected $\hat{z}^k$, it indicates a favourable estimation of the pose and the map. This alignment results in a higher weight, as illustrated in Eq. 3.31.

3. **Map estimation**: For each particle, the map estimation $p(m_t^k|x_{1:t}^k, z_{1:t})$ involves updating the localisation and uncertainty of all the landmarks observed in particle k, denoted as $\{< \mu_{i,t}^k, \Sigma_{i,t}^k >\}_{i=1,\ldots,n}$, based on the distance measurements. FastSLAM achieves this update for landmarks using the standard Extended Kalman Filter algorithm.

4. **Resampling** Particles are drawn with replacement proportional to the normalised importance weight $\hat{w}_t^k$.

$$\hat{w}_t^k = \frac{w_t^k}{\sum_{k=1}^N w_t^k} \tag{3.32}$$

During the resampling procedure, particles with high normalised importance weight will replace those with low weight. After resampling, all particle has the same importance weight, specifically $\frac{1}{N}$. This step is crucial since only a finite number of particles are used to approximate a continuous distribution. However, the resampling does not execute at every time step as doing so may remove potentially valuable samples. According to [198], an indicator called effective particle number N_{eff} is designed to determine when resampling should tasks place: signal when to resample:

$$N_{\text{eff}} = \frac{1}{\sum_{k=1}^N (\hat{w}_k)^2} \tag{3.33}$$

The resampling will be carried out when N_{eff} is less than $\frac{1}{2}N$.

3.3.2 Lidar-based Localisation

A 2D-Lidar sensor is used for Lidar-based indoor navigation in Chapter 4 and the Hector slam [181] is used for localisation, which estimates robot poses and updates an occupancy grid map using a scan matching technique.

The optimisation method aims to refine a robot pose $\xi = (p_x, p_y, \psi)$ in the world

frame, aligning it to best match the currently observed laser endpoints represented as $\boldsymbol{s_i} = (s_{i,x}, s_{i,y})$ in the sensor frame with the constructed grid map M from previous time steps. The cost function is formulated using the following equation:

$$\xi^* = \underset{\xi}{\operatorname{argmin}} \sum_{i=1}^{N} [1 - M(f(\xi, \boldsymbol{s_i}))]^2 \tag{3.34}$$

where $f(\xi, \boldsymbol{s_i})$ is to transfer the ith laser beam endpoint $\boldsymbol{s_i}$ represented in the sensor frame to the world frame according to a pose ξ, which is :

$$f(\xi, \boldsymbol{s_i}) = \begin{pmatrix} cos(\psi) & -sin(\psi) & p_x \\ sin(\psi) & cons(\psi) & p_y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} s_{i,x} \\ s_{i,y} \\ 1 \end{pmatrix} \tag{3.35}$$

The function $M(\boldsymbol{p})$ gives the map value at position $\boldsymbol{p}$. N is the total number of laser beams that have valid readings. Number 1 means the laser beam detected objects. From this, it can be learned that if the laser scan is in an open space where the laser beam can not detect any objects (valid reading beams N=0), the cost function will not be well-constructed.

The Hector slam algorithm uses the Gauss-Newton approach to solve such a problem, which linearized the cost function Eq. 3.34 using first-order Taylor expansion at around an initial guess ξ_0, with $\Delta\xi = \xi - \xi_0$:

$$\begin{aligned} \Delta\xi^* &= \underset{\Delta\xi}{\operatorname{argmin}} \sum_{i=1}^{N} [1 - M(f_i(\xi, \boldsymbol{s_i})) - \nabla M(f_i(\xi, \boldsymbol{s_i})) \frac{\partial f_i(\xi, \boldsymbol{s_i})}{\partial \xi} \Delta\xi]^2 \\ &= \underset{\xi}{\operatorname{argmin}} \sum_{i=1}^{N} [\Delta y - \nabla M(f_i(\xi, \boldsymbol{s_i})) \frac{\partial f_i(\xi, \boldsymbol{s_i})}{\partial \xi} \Delta\xi]^2 \\ &= \underset{\xi}{\operatorname{argmin}} \sum_{i=1}^{N} [\Delta y - J\Delta\xi]^2 \end{aligned} \tag{3.36}$$

where $\Delta y_i = 1 - M(f_i(\xi, \boldsymbol{s_i}))$ is the error residual. $J = \nabla M(f_i(\xi, \boldsymbol{s_i})) \frac{\partial f_i(\xi, \boldsymbol{s_i})}{\partial \xi}$ is the observation matrix.

Eq. 3.36 can be solved analytically:

$$\Delta\xi^* = \sum_{i=1}^{N}(H)^{-1}J^T\Delta y \tag{3.37}$$

where $H = J^TJ$ is the Hessian matrix. The optimisation is usually done for several iterative steps:

$$\xi_{k+1} = \xi_k + \Delta\xi_k^* \tag{3.38}$$

Based on the estimated poses, the algorithm will update the occupancy grid map using the laser scans.

The covariance of the pose estimation is approximated with the Hessian matrix H defined above:

$$\mathcal{C} = Var\{\xi\} = \sigma^2 \cdot H^{-1} \tag{3.39}$$

where the σ is the variance of the measurement noise.

3.4 Conclusion

This chapter introduces key concepts and algorithms within the reinforcement learning framework. With this foundational understanding of reinforcement learning, two major RL algorithms utilised in this thesis, DQN and PPO, are presented. Additionally, key design modules of RL policy training are discussed. Furthermore, detailed descriptions of localisation algorithms are provided to enhance comprehension of the subsequent chapters presented in this thesis.

Chapter 4

Localisation-Safe Reinforcement Learning for Mapless Navigation

As discussed in the literature review section, most RL-based works for mapless point goal navigation tasks assume the availability of the robot ground-truth poses, which is unrealistic for real-world applications. The navigation method proposed in this chapter will remove such an assumption and deploy observation-based practical localisation algorithms, such as Lidar-based localisation or Ultra-wideband (UWB) positioning, for robot self-pose estimation.

Although these localisation algorithms have demonstrated promising performance and robustness in various challenging environments, they still may encounter difficulties, such as loss of track of robot locations, in challenging environments where observations along robot trajectories are insufficient or ambiguous. Consequently, these localisation algorithms can introduce unexplored challenges for mapless navigation tasks that have not been addressed in previous research. This chapter will provide a thorough discussion of these novel problems.

This chapter introduces a localisation-safe mapless navigation strategy based on reinforcement learning for both open space and indoor navigation. As illustrated in Fig. 4.1, the robot may encounter difficulties of localisation in the middle of the

yard where no landmarks can be observed directly.

The following sections first introduce the navigation problem. The proposed method is described in Section 4.2 which includes the system overview and the novel RL agent training design, followed by experiments and results in section 4.3. The conclusions are presented in Section 4.4.

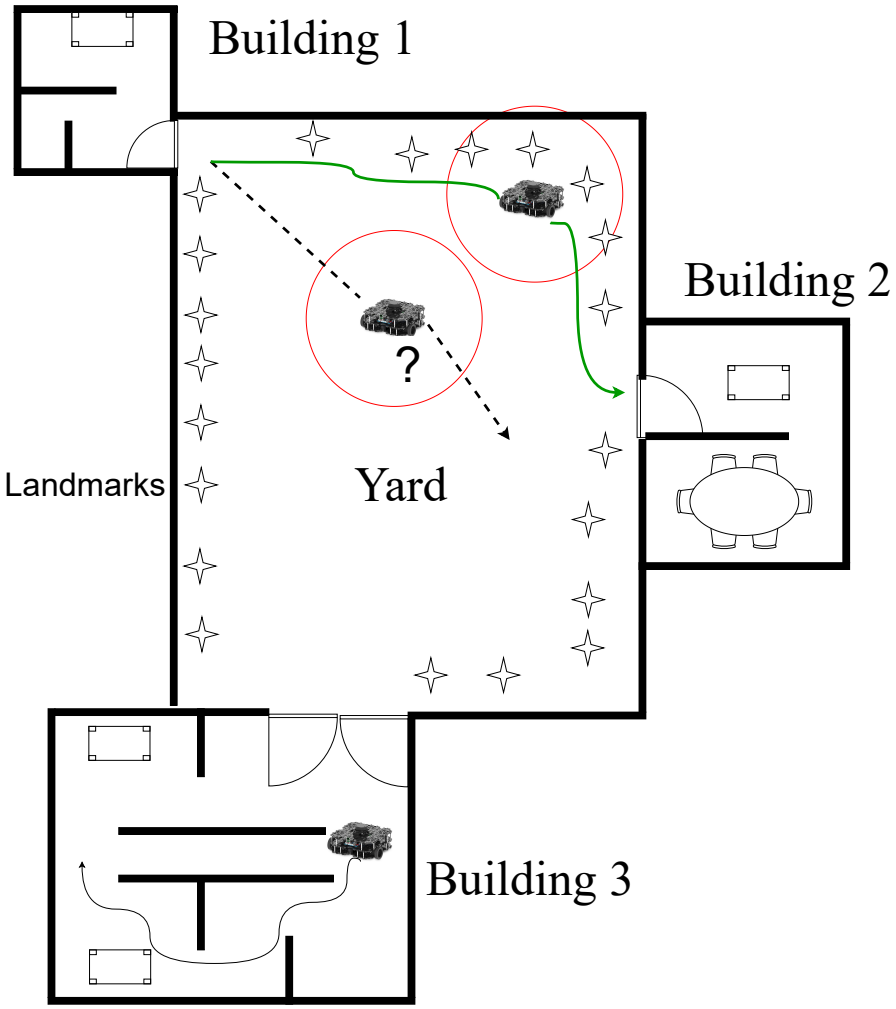


Figure 4.1: Navigation in buildings and yards. The robot may encounter difficulties of localisation in the middle of the yard where no landmarks can be observed directly.

4.1 Introduction

Mapless point goal navigation has emerged as a popular research focus, with the aim of instructing robots to navigate toward a target goal position in environments lack-

ing pre-existing maps, as seen in search and rescue scenarios. While conventional path-planning algorithms cater to a broad range of scenarios or environments, they exhibit well-known limitations [45]. Notably, these algorithms may necessitate handcrafted or constrained functions tailored to specific application conditions, resulting in a lack of generalization capability for diverse environments [127].

To enhance navigation generalisation ability, learning-based navigation, particularly RL, has garnered increasing attention. In contrast to other learning-based algorithms, such as supervised learning, RL demands minimal human resources or intervention during the training stage. Robots are trained by receiving continuous rewards or punishments through direct interaction with their environments. In the realm of mapless navigation, RL-based algorithms have demonstrated effectiveness, enabling mobile robots to achieve promising performance in collision-free navigation within unknown environments [45].

Mapless navigation tasks often employ two distinct goal position representation formats: images or coordinates of goal positions. In the case of image goals, agents may not require knowledge of their current poses but only need to recall environmental configurations to locate goal positions. Consequently, such agents may have limited generalization ability to unseen environments. The focus of this chapter is on goals represented by coordinates, requiring agents to estimate their own positions at each time step, which could apply to more general environments.

Among the RL algorithms discussed earlier for mapless navigation tasks in Chapter 2, the majority operate under the unrealistic assumption that robots have access to ground truth poses throughout their missions. However, this assumption is impractical for real-world applications. Typically, observation-based localisation algorithms, such as Particle-filter-based, Lidar-based or visual odometry, are essential for aiding robot self-localisation in practical robot navigation tasks.

Therefore, this work adopts a more realistic approach to tackle mapless navigation tasks by introducing observation-based localisation algorithms for robot self-

pose estimation.

The subsequent subsections will provide a detailed exposition of the problem.

4.1.1 Outdoor open-space navigation

The localisation algorithm used for open space navigation is based on distance measurement sensors. The localisation algorithm with such sensors is based on the fact that the observations of distances from three or more landmarks can decide the position of the sensor, as shown in Fig. 4.2. The landmarks can be anything that helps localisation, such as UWB anchors [199], Apriltags [200] or image features [11]. Due to the unknown position of the landmarks and the noise in sensor measurements, the estimation of the robot position is not this easy, as shown in Fig. 4.2. The SLAM technique is developed to solve such a mapping and localisation problem, which has been described in detail in Section 3.3.

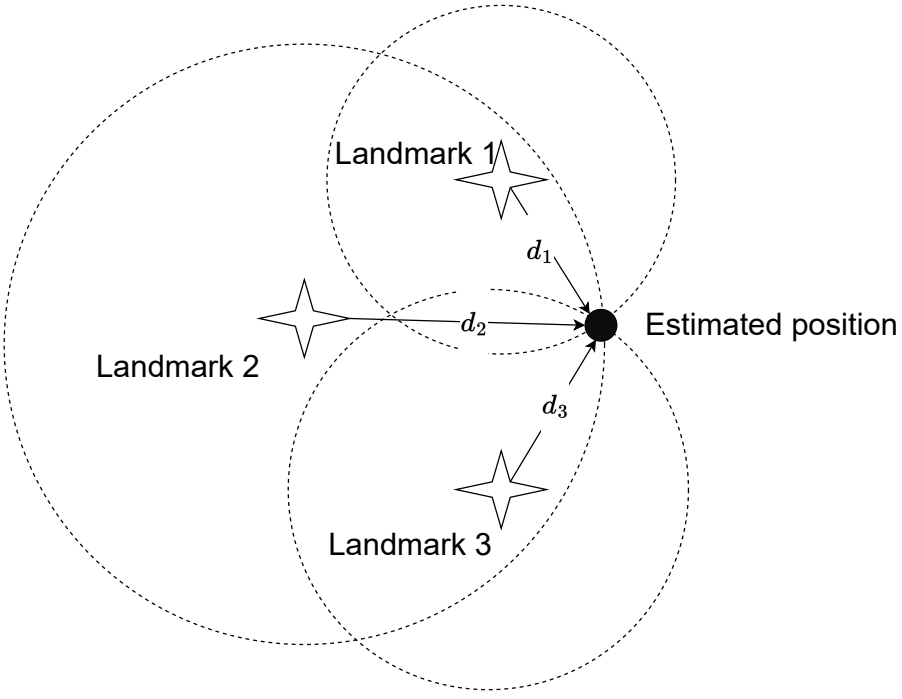


Figure 4.2: Position estimations from landmarks observation

It is evident that when the observed number of landmarks is insufficient, e.g. when most of the landmarks are out of the sensor observation range, the SLAM

algorithm encounters difficulty in determining the sensor's position. For instance, with two landmarks, there may be two possible positions based on distance observations from the landmarks. Notably, landmarks are typically situated on or near walls in open space scenarios. For example, Apriltags can be attached to walls to provide spatial references for localisation [201].

Previous research in RL has primarily focused on obstacle avoidance and minimizing trajectory distance, often neglecting the importance of pose estimation quality. This oversight stems from the assumption of constant access to accurate robot poses, effectively decoupling perception from path planning. However, navigation policies trained under such assumptions can lead to catastrophic failures. For instance, consider the navigation scenario depicted in Fig. 4.1, where the robot aims to navigate from Building 1 to Building 2. Navigation policies based on the assumption of readily available robot poses may choose a route directly pointed to the destination position (the black dashed trajectory in Fig. 4.1) as it represents the shortest path.

In reality, as the robot moves to the centre of the yard, the limited sensing range of the sensor (depicted by the red circle in Fig. 4.1) results in no observations of landmarks. Consequently, the robot loses its location and fails to determine the goal, thereby failing the navigation task. In contrast, the green trajectory, although slightly longer in distance, consistently observes enough landmarks along the path to aid in localisation, enabling successful arrival at the destination. This behaviour aligns with the approach proposed in this chapter.

4.1.2 Indoor navigation

The sensor employed for indoor navigation is a Lidar sensor. In contrast to the challenges of insufficient features (landmarks) observed in outdoor navigation, the Lidar-based odometry algorithm may encounter difficulties in the presence of ambiguous observations. This type of odometry becomes problematic and inadequately

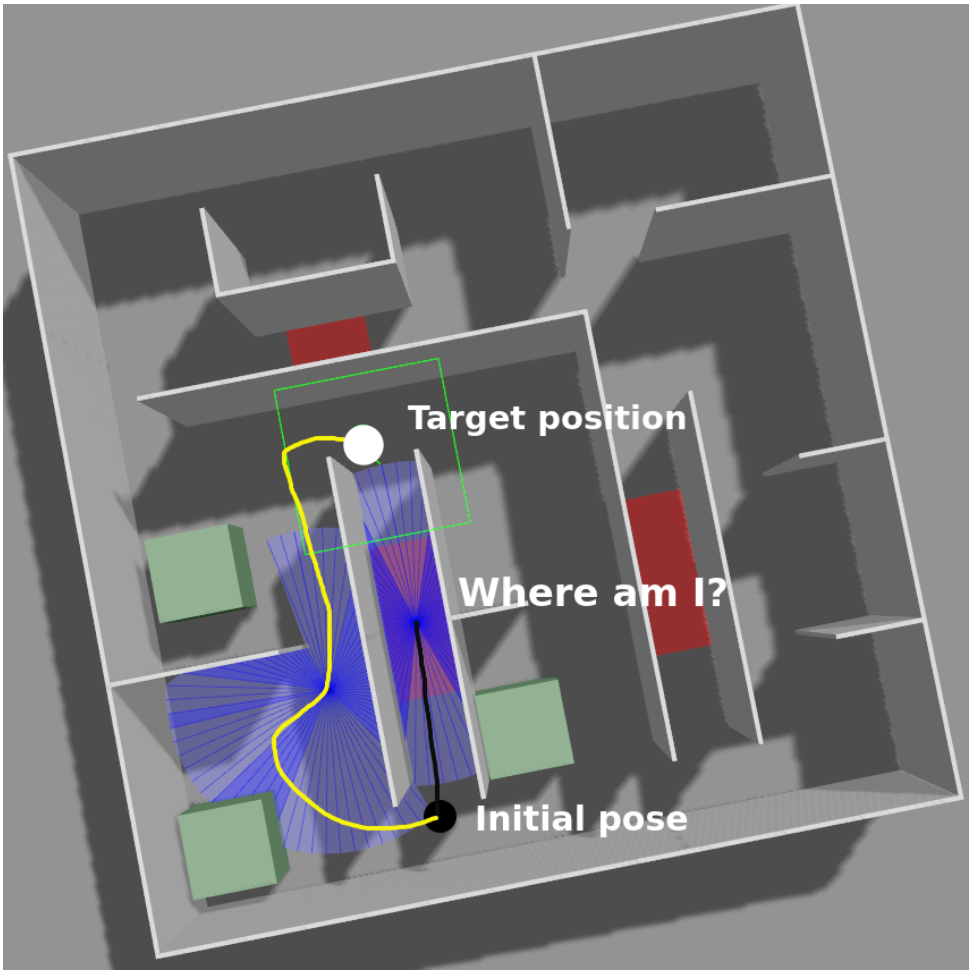


Figure 4.3: Navigation examples (red regions: regions where localisation fails; black trajectory: unsuccessful trajectory into the symmetric corridor; yellow trajectory: successful trajectory avoiding the symmetric corridor.)

constrained in long symmetric corridor environments, leading to inaccurate pose estimations [202]. Specifically, as illustrated by the red regions in Fig. 4.3, these corridor areas feature only two parallel and symmetric walls that are less distinct geometric features for the Lidar to accurately estimate motion changes.

The approach advocated in this chapter anticipates that robots trained with observation-based self-localisation for mapless navigation will possess the capability to recognize regions with deteriorated localisation. Consequently, these robots may choose alternative trajectories to avoid areas with challenges, such as symmetric corridor regions. As exemplified by the yellow trajectory in Fig. 4.3, this path, although slightly suboptimal in terms of length, proves more reliable in achieving

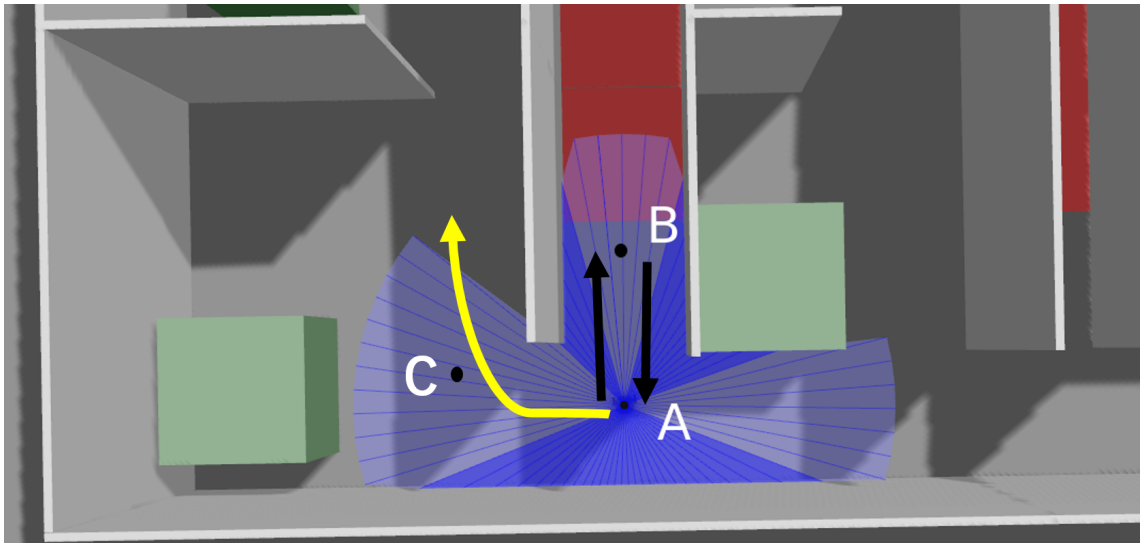


Figure 4.4: The robot gets stuck in local minimum (the goal region is at the other end of the corridor)

its goal owing to enhanced localisation performance.

However, one problem will arise when taking localisation performance into consideration. When robots are discouraged from entering localisation-unsafe regions, the robots are more likely to get stuck in local minimum, if this navigation problem is treated as a Markov Decision Process (MDP), as with many previous research works [203]. As shown in Fig. 4.4, the robot starts from point A and arrives at point B where it finds the corridor ahead is not traversable. It will return to the initial position A, where the same decision i.e. navigating to point B will be taken because history information is not used for decision making in the MDP setting. Consequently, the robot will be trapped in the local minimum (the black trajectories in Fig. 4.4). Hence, an MDP setting may sometimes fail the task.

To alleviate the problem above, the method proposed in this chapter considers that the use of history information will help robots get out of local minimum regions as illustrated by the yellow trajectory in Fig. 4.4. This is achieved through the utilization of a Long Short-Term Memory (LSTM) module in the policy network.

4.1.3 Summary

This chapter will develop seperate navigation strategies for indoor and outdoor scenarios tailored for a ground mobile robot, as depicted in Fig. 4.1 (illustrating navigation within and/or between buildings). The decision to separate navigation strategies arises from the fact that different sensors are used for indoor and outdoor navigation according to their specific capabilities. Consequently, the assumption in this chapter is that the robot employs a Lidar-based sensor for indoor navigation and sensors equipped with distance measurement capabilities, such as UWB sensors or RGBD cameras, for outdoor open-space navigation.

While there are separate strategies for indoor and outdoor navigation, the overarching training procedure remains similar, leveraging the RL-based algorithm introduced in this chapter. With this algorithm, robots are trained to navigate in a manner that mitigates the risk of localisation failures. This crucial ability is acquired through the implementation of a technique proposed in this work: a reward component employed to penalize behaviours that lead to localisation failures.

The formulation of the localisation-related reward design also brings about a local minimum challenge in indoor navigation. To tackle this issue, this chapter introduces a reconfigured state representation that incorporates both current observations and historical trajectory information. This transformation aims to shift the problem from a Partially Observable Markov Decision Process (POMDP) to an MDP model, effectively mitigating local minimum concerns.

The contributions of this chapter are summarised as follows:

- This chapter proposes an innovative RL-based mapless navigation algorithm that prioritizes localisation-safe navigation, effectively avoiding localisation failures during the navigation process.

- This approach incorporates a localisation performance-related reward, ensuring that robots are penalized when the quality of localisation begins to degrade,

which helps the learning of behaviours that maintain robust localisation.

- It introduces a method based on the determinant of pose estimation covariance as the localisation performance metric to decide the reward component for Lidar-based localisation.

- The LSTM network is introduced to feed robots with encoded history information, as part of state information of the RL algorithm, such that the original POMDP problem is converted to an MDP problem to address the local minimum problem.

4.2 Method

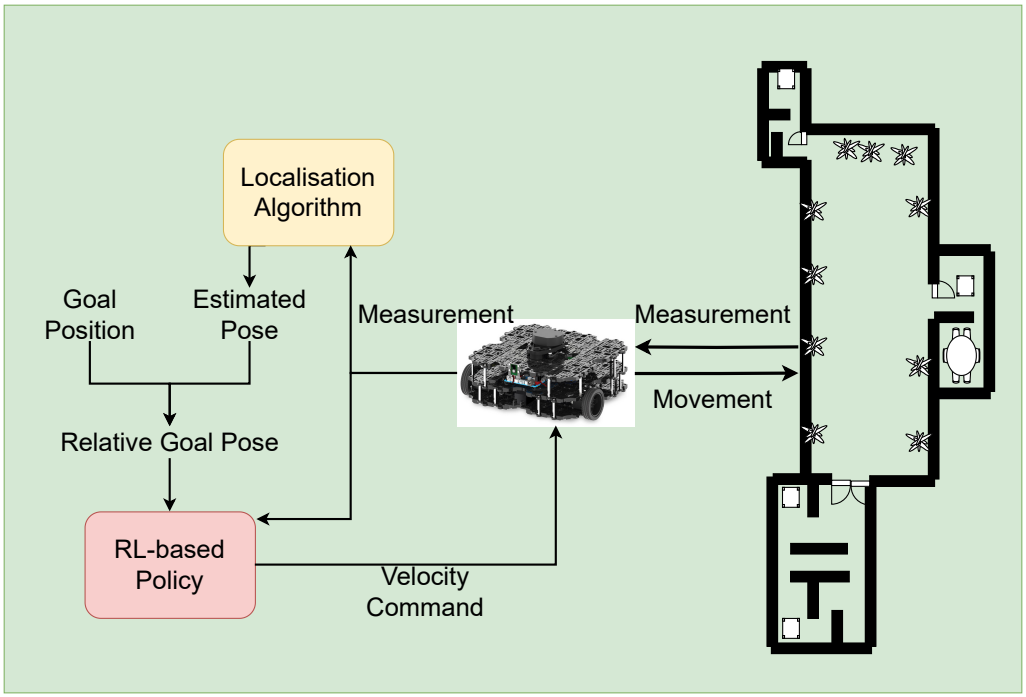


Figure 4.5: System overview. This work incorporates robot localisation algorithms into the training procedure, which has been ignored by most previous methods in the literature. A new RL-based policy is introduced that considers not only robot navigation and collision avoidance but also the localisation performance in the training process.

Fig. 4.5 illustrates the system overview of the method. Firstly, the robot receives sensor measurement data, e.g. Lidar range measurements in this case, from the

environment, and the measurements are input into the localisation algorithm for the computation of the estimated robot pose. Subsequently, the estimated robot pose and goal position are utilised to calculate the relative goal pose, denoted by the relative distance and relative heading in relation to the robot. The resulting relative goal pose, along with the measurement data, is then provided to the fail-safe localisation reinforcement learning agent, which determines the next action, which is the velocity command in the scope of this chapter. The robot acts according to the velocity command and moves in the environment. This iterative procedure continues until the robot successfully reaches the designated goal position.

The upcoming subsections will delve into detailed discussions of the localisation module and the RL-based agent for both indoor and outdoor navigation.

4.2.1 Configuration of RL-based navigation agent

The reinforcement algorithm for training both the indoor and outdoor navigation policies is based on the DQN, which has been discussed in Section 3.1.2.

This paragraph will recap several key concepts of reinforcement learning and the procedure of the DQN algorithm. Markov Decision Process is a mathematical framework to model problems that can be solved using RL. To describe a MDP, it needs a state space S, an action space A, a state transition model $p(s_{t+1}|s_t, a_t)$, and a reward function $R(s_t, a_t)$: $S \times A \to R$, which describes the feedback from the environment after agents taking actions a_t at state s_t. The objective is to obtain maximum overall discounted rewards $R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k}$ from environments. RL algorithms usually involve estimating a state-action pair value function (Q-value): $Q_\pi(s_t, a_t) = E[R_t|s_t, a_t]$ or state value function $V_\pi(s_t) = E[R_t|s_t]$, where π: $a \sim \pi(\cdot|s, \theta)$ described by parameters θ is the policy that agents try to learn.

The DQN represents the Q-value by deep neural networks $Q(s, a; \theta)$ and trains the neural network with a loss defined by the TD error Eq. 3.24. The policy for a DQN agent is to select an action that maximises the Q-value function: $\pi(s_t) =$

$\arg\max_{a \in A} Q(s_t, a; \theta)$.

Two noteworthy observations emerge from the above description. Firstly, Reinforcement Learning (RL) necessitates problems that satisfy the Markov property $p(s_{t+1}|s_t, a_t)$, wherein the next state s_{t+1} depends solely on the present state s_t, and the past does not influence the future. This property, however, is not satisfied for a POMDP problem. In POMDPs, agents lack access to state information s_t but only receive observations o_t from a state-conditioned distribution $p(o_t|s_t)$. The Markov property breaks down, as $p(o_{t+1}|o_t, a_t, o_{t-1}, a_{t-1}, ..., o_0) \neq p(o_{t+1}|o_t, a_t)$. In mapless navigation settings, where the robot can only observe its surrounding environment, the problem can be inherently cast as a POMDP. Additionally, in this work, the robot lacks ground truth poses and must estimate poses from history observations using localisation algorithms, such as Lidar or visual odometry, further aligning with the characteristics of a POMDP.

Secondly, the learned policy heavily relies on rewards and feedback from the environment. Given that this work aims to train agents to navigate based on localisation-safe policies, the introduction of a metric to determine a reward associated with localisation performance becomes necessary.

This chapter details the configuration of the proposed novel RL-based policy based on the above findings, which have not been discussed in previous works in the literature. The following section will discuss it in detail.

Configuration of the proposed RL-based navigation

As discussed above, mapless navigation that uses estimated localisation could be viewed as a POMDP problem from the following two perspectives.

Firstly, from the localisation perspective, the robot pose estimation at the current time step will require historical poses estimated as described in Eq. (3.30) introduced

in Section 3.3:

$$p(x_{1:t}, m|z_{1:t}, u_{1:t-1}) = p(m|x_{1:t}, z_{1:t})p(x_{1:t}|z_{1:t}, u_{1:t-1})$$

The above equation highlights that pose estimation, built on past inaccurately estimated poses resulting from insufficient or ambiguous observations, will also be inaccurate. Given that estimated robot poses and observations constitute part of the agent's state input data, continuous training on such problematic data can mislead the robot's learning process. To mitigate this, this work proposes ending training episodes when localisation begins to deviate from the ground truth. While one might consider using pose estimation errors as a localisation performance criterion, it takes time for position errors to accumulate to a noticeable scale. Experiences during this time can still detrimentally impact the training procedure.

For outdoor navigation, the suggested metric is the number of landmarks observed at the current timestep. In the case of indoor navigation, where a Lidar sensor is employed, this work recommends using the determinant of the estimation covariance calculated from the Hessian matrix as the metric. Further discussion on this metric will be presented later.

From the perspective of mapless settings, robots lack access to global maps, making them susceptible to getting stuck in local minimum regions where they may hesitate to enter certain localisation-unsafe areas, as illustrated in Fig. 4.4 in Section4.1. To address this challenge and transform the POMDP problem into an MDP for RL algorithms, this work proposes combining historical information $h_{t-1} = (o_{t-1}, o_{t-2}..., o_0)$ with the current observation o_t as an approximation of the current state s_t. Storing entire historical trajectories can be resource-consuming and inefficient for neural network inputs. In this work, it suggests utilizing the LSTM architecture, specifically designed for processing sequences of data. The LSTM encodes history information, represented by hidden values h_t at each time step t,

which are preserved as inputs for calculations in subsequent time steps, as illustrated in Fig. 4.6.

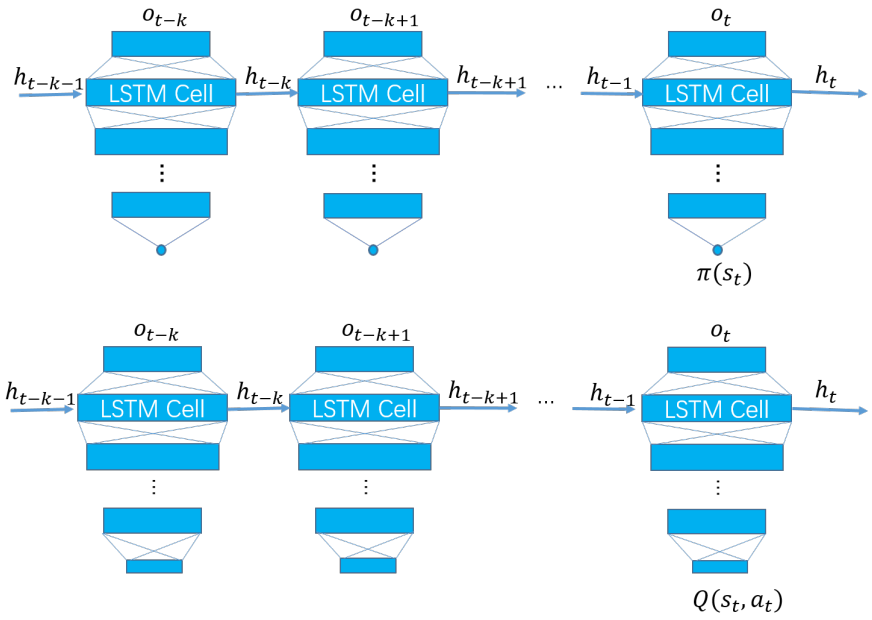


Figure 4.6: LSTM for encoding history information

4.2.2 Novel DQN agent implementation detail

The specific configuration details for DQN used in the localisation-safe navigation within this work are as follows. The state space of the DQN agent encompasses both the sensor observation measurement o_t and the relative goal position g_t. The latter includes the relative distance d_g and heading β with respect to the robot.

As the classic DQN is designed for handling discrete action spaces, the action space needs to be discretized. During each time step, the agent selects a linear velocity v_{linear} from a set of values $[v_{l_1}, v_{l_2}...v_{l_i}]$ and an angular velocity from a set of values $[w_1, w_2...w_j]$. i and j can be decided according to specific requirements.

Novel reward space

In this task, the agent is tasked with navigating to a designated goal position while simultaneously avoiding obstacles and minimizing its localisation uncertainty. Con-

sequently, the novel reward function proposed is defined as follows:

$$r = \begin{cases} r_{lost} & \text{if localisation fails} \\ r_{collision} & \text{if collision happens} \\ r_{goal} & \text{if } d_g < d_{gmin} \\ f \times (d_{t-1} - d_t) & \text{otherwise} \end{cases} \tag{4.1}$$

Here, r_{lost} is a negative value generated when the localisation algorithm reports failures, and $r_{collision}$ is also a negative value intended to penalize the agent for collisions with obstacles. On the positive side, r_{goal} is a reward value granted when the robot successfully reaches the goal position within a minimum acceptable distance, defined by d_{gmin}. The term $d_{t-1} - d_t$ is included to encourage the agent to make decisions that decrease the relative goal distance, and f represents the distance rate factor, which can be adjusted. Whenever one of the three situations (localisation failure, collision, or reaching the goal) occurs, the episode terminates, and a new episode begins.

Localisation failure indicators

This section discusses the details of the localisation failure signal, r_{lost}, proposed by this work. Previous research works seldom consider the penalty of r_{lost} to regulate agent behaviours.

However, this reward is crucial to prevent the robot from venturing into open spaces where no or very sparse features can be observed. As described in Section 3.3, it is evident that when the robot moves into open space, where the sensor lacks sufficient observed features, the second term in Equation 3.31 will not be computed. Consequently, the weight factors of the particles will not be updated. This results in the particle filter being unable to assess the quality of the particles, leading to a failure to perform re-sampling for correct estimation of the robot state using these

weight factors. As a consequence, the localisation algorithm will fail and rely solely on odometry, which is not accurate.

For the outdoor navigation agent, when the number of observed landmarks is below a certain threshold, the episode will terminate, and a negative r_{lost} will be assigned to penalize the robot.

For the indoor Lidar-based navigation agent, even with many laser beams providing valid readings, the loss function in Equation 3.34 may not be well-constrained when these readings are obtained from a symmetric corridor or a single straight wall. Consequently, the number of observed features proves to be an unreliable indicator of localisation failure.

As an alternative, this chapter suggests employing the determinant of the estimation covariance, denoted as $\det \mathcal{C}$. The covariance is computed using the Hessian matrix H, as outlined in Equation 3.39, as discussed in the localisation section (Section 3.3).

$$\det \mathcal{C} = \sigma^2 \det(H^{-1}) \tag{4.2}$$

The detailed procedure for training a DQN-based navigation agent is shown in Algorithm 1.

4.3 Evaluation

The evaluation of the outdoor navigation agent and the indoor navigation agent will be discussed separately in this section.

4.3.1 Outdoor navigation agent evaluation

Experiment setup

The outdoor navigation method proposed in this chapter undergoes testing within a 2-dimensional simulation environment utilizing a mobile robot featuring a 3-

Algorithm 1 DQN-based Navigation

1: Initialise action value function Q with parameters θ including the LSTM module discussed in Section 4.2.1,
2: Copy the initial Q parameters θ as the target value function $\bar{Q}$ initial parameter θ^-: $\theta^- = \theta_0$
3: Create an empty memory buffer D with a maximum capacity N
4: **for** $k = 0, 1, 2, ...$ **do**
5: Randomly set both the robot pose and the navigation goal position.
6: Construct the state from robot observation and the goal position: s_0
7: **for** $t = 0, ..., T$ **do**
8: Select an action $a_t = \arg\max_{a \in \mathcal{A}} Q(s_t, a; \theta)$ with probability of ϵ; otherwise randomly select $a_t \in \mathcal{A}$.
9: Send the action a_t command to the robot; wait to receive observation and calculate reward r_t using Eq. 4.1; construct the new state according to new observation: s_{t+1}.
10: Store transition $(s_t, a_t, r_t, s_{t+1)}$ in memory buffer D
11: Sample a minibatch n of transition data $(s_j, a_j, r_j, s_{j+1)}$ from D.
12: Compute the target value$y$:

$$y_j = \begin{cases} r_j & \text{if } s_{j+1} \text{ is the terminal state} \\ r_j + \gamma \max_{a'} \bar{Q}(s_{j+1}, a'; \theta^-) & \text{otherwise} \end{cases} \tag{4.3}$$

13: Construct the loss function:

$$\mathcal{L} = \frac{1}{n} \sum_{j=1}^{n} (y_j - Q(s_j, a_j; \theta))^2 \tag{4.4}$$

14: Update the Q network with performing gradient descent on $\mathcal{L}$ with respect to parameter θ
15: **if** s_{t+1} is terminal state **then**
16: break
17: **end if**
18: Set target $\bar{Q}$ parameter $\theta^- = \theta$ every K steps
19: If the episode termination condition is detected, as described in Section 4.2.2, break and go to step 4 ;
20: **end for**
21: **end for**

dimensional kinematic motion model. As depicted in Fig. 4.7, the black dots represent landmarks observable by the robot for localisation. Each landmark also serves as an obstacle, taking on a circular shape with a radius of 1 m, as illustrated by the blue regions in Fig. 4.7. The robot's observations encompass relative distances and angles to these landmarks within the robot's maximum observation range, set at 5.0

m with full 2π coverage. The objective for the robot is to navigate to a designated goal position, denoted by a blue star in Fig. 4.7. The red crosses signify estimated landmarks computed by the FastSlam localisation algorithm and observed during navigation.

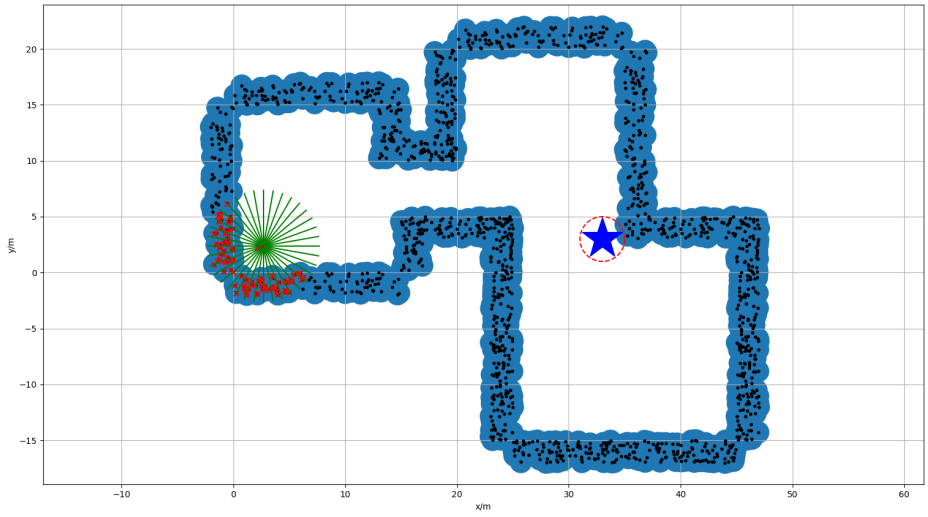


Figure 4.7: Environment: robot (green dot) and goal position (blue star); green lines indicate the 5-meter sensor detection range; all landmarks (black dots) have a 1-meter collision radius (blue region); red crosses are the estimated landmarks by the FastSlam.

During training, those landmarks and goal positions are generated randomly. the training procedure also uses randomly generated maps of different shapes. The robot linear velocity is set to be a constant value $v_l = 1.0$ m/s. The angular velocity is a selection from the following set of values $(-2.0, -1.0, 0.0, 1.0, 2.0)$ rad/s. Both linear and angular velocities are added with Gaussian noises during the robot execution to simulate odometry errors. The reward elements r_{lost}, $r_{collision}$ and r_{goal} are -300, -300 and 600 respectively and the distance rate factor f is 10.

For the DQN-based RL framework, measurement data need to be converted into a discrete structure. The full 2π coverage of the sensor is first divided into 36 groups according to relative angles (10 degrees per group). The observation o_t consists of two value lists: $[n_1 \cdots n_{36}]$ where each element represents the number of

observed landmarks in that angle group and $[lmin_1 \cdots lmin_{36}]$ where each element represents the value of the relative distance to the nearest landmark in that angle group. Using the restructured observation ($36 + 36$ dimensions) together with the relative goal position (2 dimensions), the agent state would be then represented by a 74-dimension vector. The input data is connected with 2 dense layers (512 nodes each) and the final layer uses a linear activation function, as shown in Fig. 4.8. Other DQN parameters are shown in Table 4.1.

parameter	value
learning rate	0.00025
discount factor	0.99
epsilon decay rate	0.998
replay buffer	1000000
target network update rate	per 10 episodes

Table 4.1: DQN settings

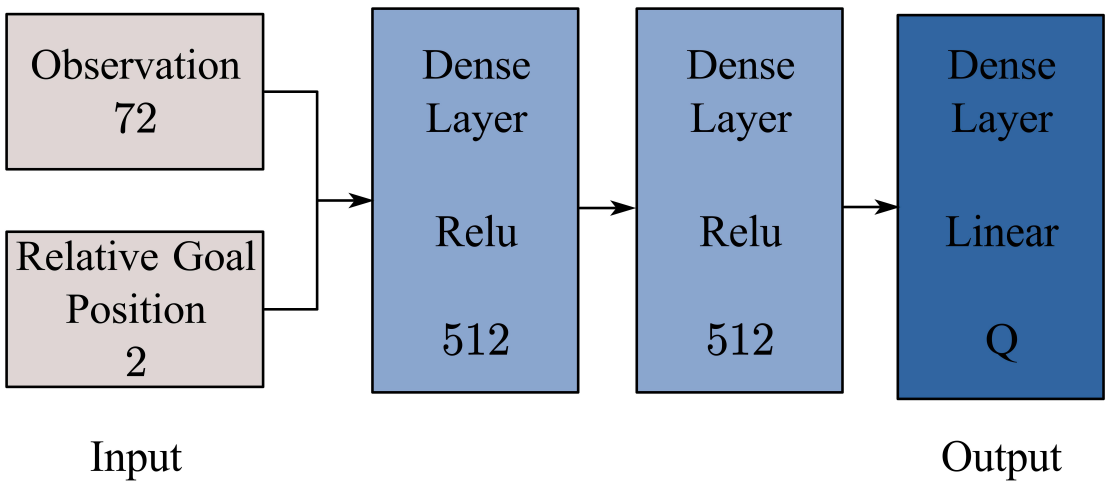


Figure 4.8: The outdoor navigation Q network structure

Training results

Baseline methods in the literature: As mentioned earlier, previous research often assumes access to ground truth robot poses. In Fig. 4.9, the trajectory illustrates the scenario when a robot is provided with true poses and trained without

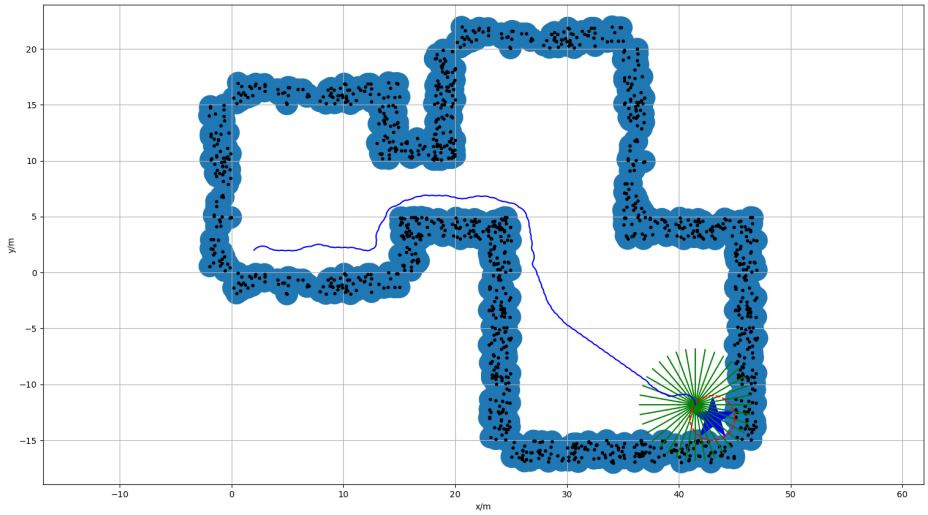
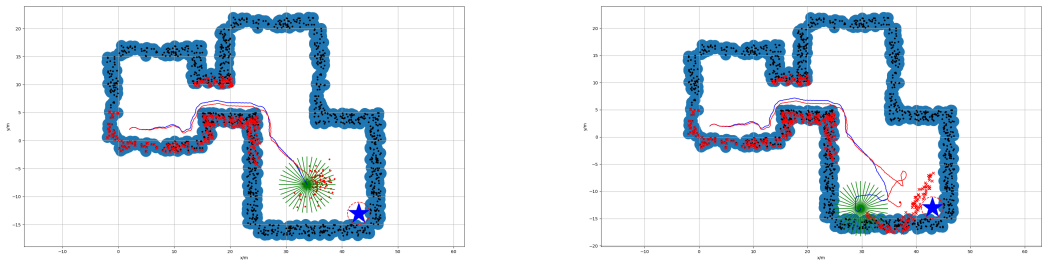


Figure 4.9: Robot navigation trajectories with ground truth poses provided



(a) (b)

Figure 4.10: Trajectories generated without localisation failure penalty (a) PF localisation diverges when no feature is observed, (b) PF localisation re-converges to wrong poses (blue star: goal; blue line: ground truth trajectory; red line: estimated trajectory)

the penalty for localisation failures. The robot navigates through an open space diagonally, reaching the goal position in a relatively short distance. However, in the real world, obtaining ground truth poses is not always feasible. When particle filter-based localisation is employed in the same task, the robot's trajectory diverges from the true path, as shown in Fig. 4.10, where the red and blue lines represent the estimated and ground truth trajectories, respectively.

During navigation, divergence occurs due to poor observations of environmental features, as depicted in Fig. 4.10a. In situations where navigation involves diverged

particles, observing new features triggers particle resampling, aiming to relocalise the robot and converge to a new pose based on the newly observed features. However, the estimated pose may lose its original track and converge to incorrect poses, as illustrated in Fig. 4.10b. This failure in particle filter localisation can lead to catastrophic consequences. In certain instances, the robot's goal position may become unreachable due to misaligned obstacles, as indicated by the red crosses at the bottom right corner in Fig. 4.10b. Consequently, the robot will never reach the goal position.

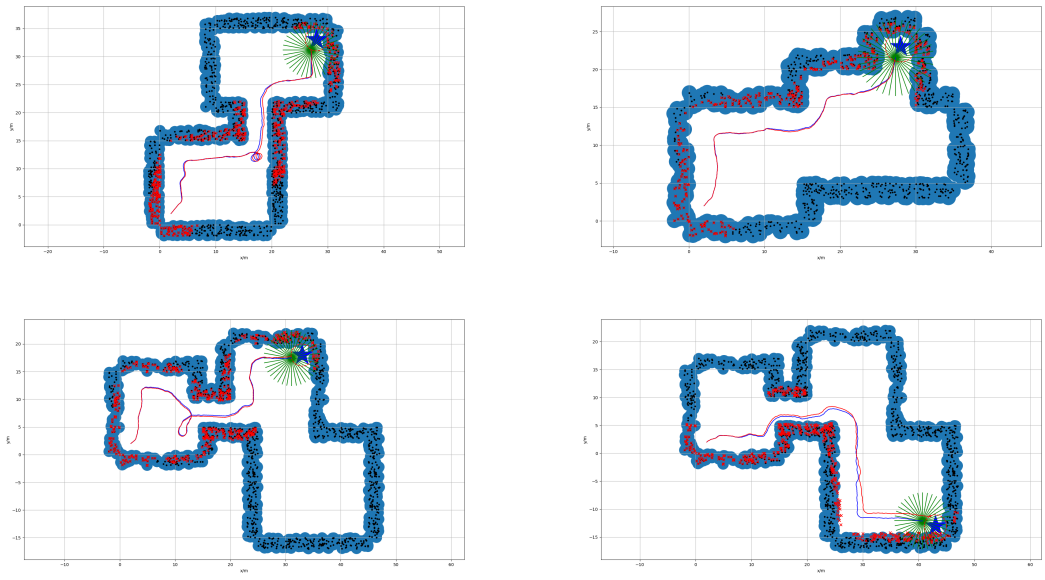


Figure 4.11: Trajectories generated by the proposed agent with localisation failure penalty: The trajectories stick close to landmarks instead of going directly into goal positions to keep observing enough landmarks. The localisation failure is avoided compared to navigation by other methods. (blue star: goal; blue line: ground truth trajectory; red line: estimated trajectory)

Performance of the proposed method: In the same set of experiments, the navigation policy trained with the additional localisation failure penalty r_{lost}, introduced in this work, was tested. Fig. 4.11 displays the trajectories estimated using the particle filter algorithm for localisation. As anticipated, the estimated trajectories closely align with the true trajectories. Notably, the new trajectories tend to stay close to landmarks, ensuring high-quality landmark observations for robust

localisation. Consequently, the robot successfully reaches the goal relying solely on PF-based localisation. The enhanced performance can be primarily attributed to the novel landmark-aware RL-based navigation policy, which encourages the robot to maintain a distance that ensures good feature observations for high localisation confidence.

The success rates, evaluated at different training episodes, are presented in Fig. 4.12. Eq. 4.5 defines the success rate, where N_{all} is the total testing number conducted at each episode and N_{success} records the number when the robot reaches goal regions successfully within the N_{all} tests. It is evident that the success rate increases with the number of training episodes and stabilises at around 80% after 3000 episodes. Given the simplicity of the network, the training process takes several hours on a workstation equipped with an Intel i7-8700 CPU and an Nvidia RTX-2080 GPU.

$$\text{success rate} = \frac{N_{\text{success}}}{N_{\text{all}}} \tag{4.5}$$

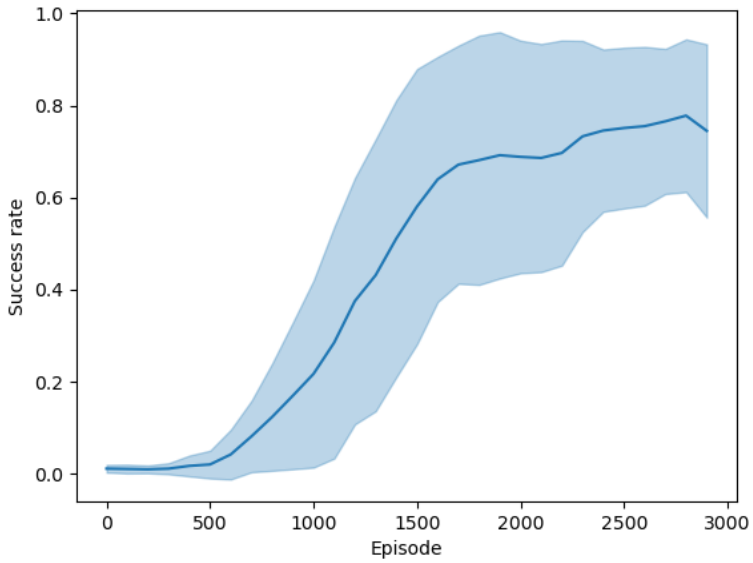


Figure 4.12: Success rate: It marks a success when the robot reaches the goal region for one navigation task. Otherwise, it is considered a failure when collision failure, time-out failure, or localisation failure occurs. 100 navigation tasks are tested for each episode.

4.3.2 Indoor navigation agent evaluation

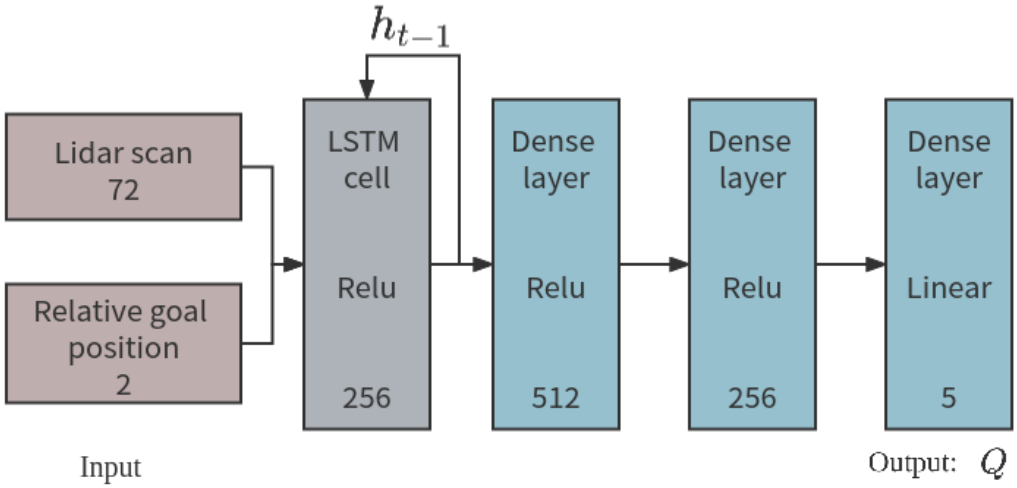


Figure 4.13: Agent network architecture

The indoor navigation algorithm is tested using a Turtlebot-3 mobile robot equipped with a 2D-Lidar sensor in Gazebo simulation environments as shown in Fig. 4.3.

The observation state o_t for the agent consists of laser scan data $(l_1 \cdots l_{72})$ and the relative pose to the goal, represented in the polar coordinate (relative distance and angle to the goal (d_g, β)). This would result in a 74-dimension state vector as the input for the agent network.

Given the discretized action space, the action set includes angular velocities: $\{-1.5, -0.75, 0.0, 0.75, 1.5\}$ rad/s. To simplify the problem, the robot's linear velocity is set as a constant value $v_l = 0.1$ m/s. The network configuration, as depicted in Fig. 4.13, comprises one LSTM cell and three Dense layers.

For pose estimation, as detailed in Section 3.3, the indoor navigation algorithm employs the well-known Lidar-based localisation algorithm, Hector SLAM. Notably, the maps constructed by Hector are not utilized by the agent, emphasizing the mapless navigation approach.

Inverse of Hessian Matrix as Reward Metric

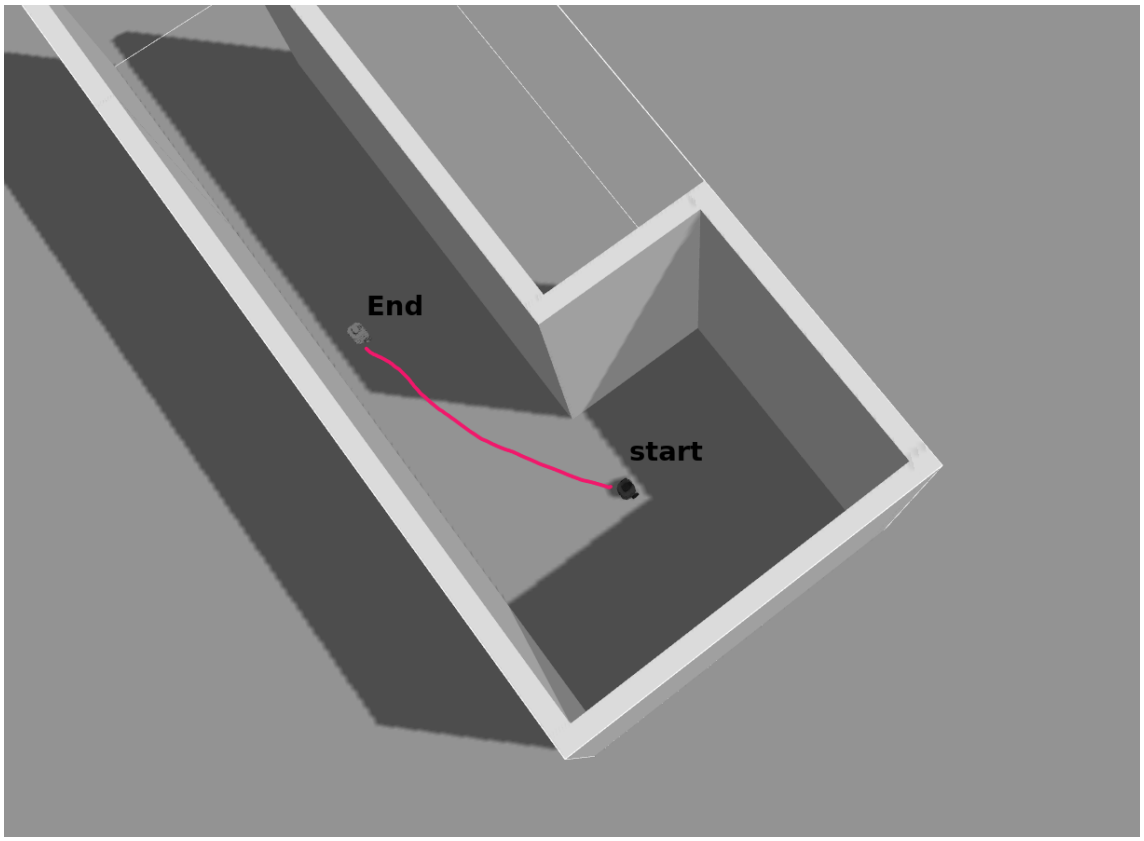


Figure 4.14: Robot travelling from room into corridor

This section will first demonstrate the effectiveness of the value of determinant $\det \mathcal{C}$, on how it may affect the robot's behaviours, e.g., if the robot could avoid areas with deteriorated localisation. For illustration purposes, Fig. 4.14 presents a scenario where the robot is positioned in a room with distinct geometric features and subsequently moves into a corridor following human commands. The corridor poses a challenge for Lidar-based localisation due to ambiguous symmetric features.

As discussed before, one obvious metric could be the error of the estimated pose. Therefore, the evaluation here compares the pose errors between the estimated robot poses and the ground truth poses and the determinants $\det \mathcal{C}$ along the trajectory. As shown in Fig. 4.15, it can be seen that, after the robot enters the corridor at the time step of about 380, the pose estimation error starts to rise gradually, while, in contrast, the determinant sharply moves upwards and remains high afterwards with

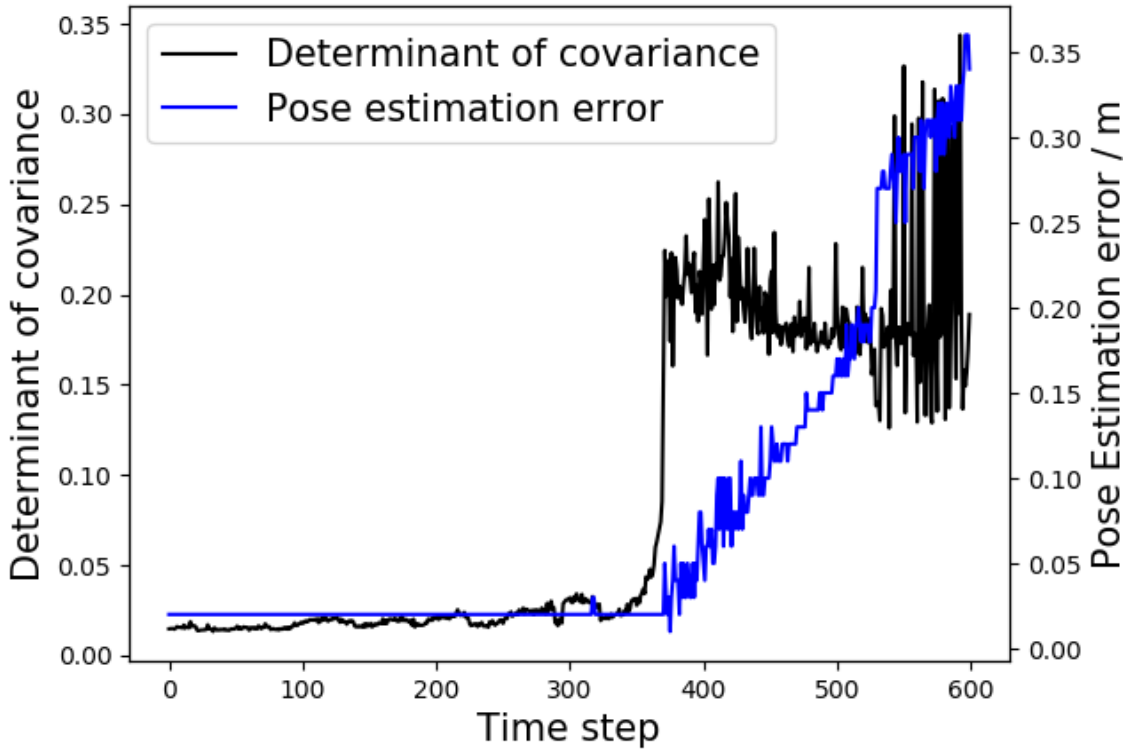


Figure 4.15: Determinant values of covariance and pose estimation errors

a certain level of fluctuations.

The pose estimation errors exhibit a gradual increase, taking a considerable number of time steps to reach a noticeable level. This characteristic is undesirable when considering it as a reward metric in the context of localisation safety and the MDP setting, as it fails to provide an immediate response to localisation failures. In contrast, as depicted in the figure, the proposed determinant of the covariance can promptly indicate when the robot begins to lose its track. Consequently, it enables the RL agent to be penalized immediately for such behaviours. This work has tested extensively on similar scenarios and has consistently observed similar changes of the determinant value.

DQN Training Results

The algorithm's training process for the DQN agent is outlined as follows: In each episode, the robot is respawned at a random position, with the goal position also being randomly determined. The episode terminates when any of the following conditions are met: 1) the robot collides with an obstacle or wall; 2) the determinant value of the covariance exceeds a predefined threshold, indicating deteriorated localisation (e.g., the robot entering a symmetric corridor area in this environment); 3) the robot reaches the goal region; 4) or the maximum time step is reached. The training of the policy network is summarised in Algorithm. 1.

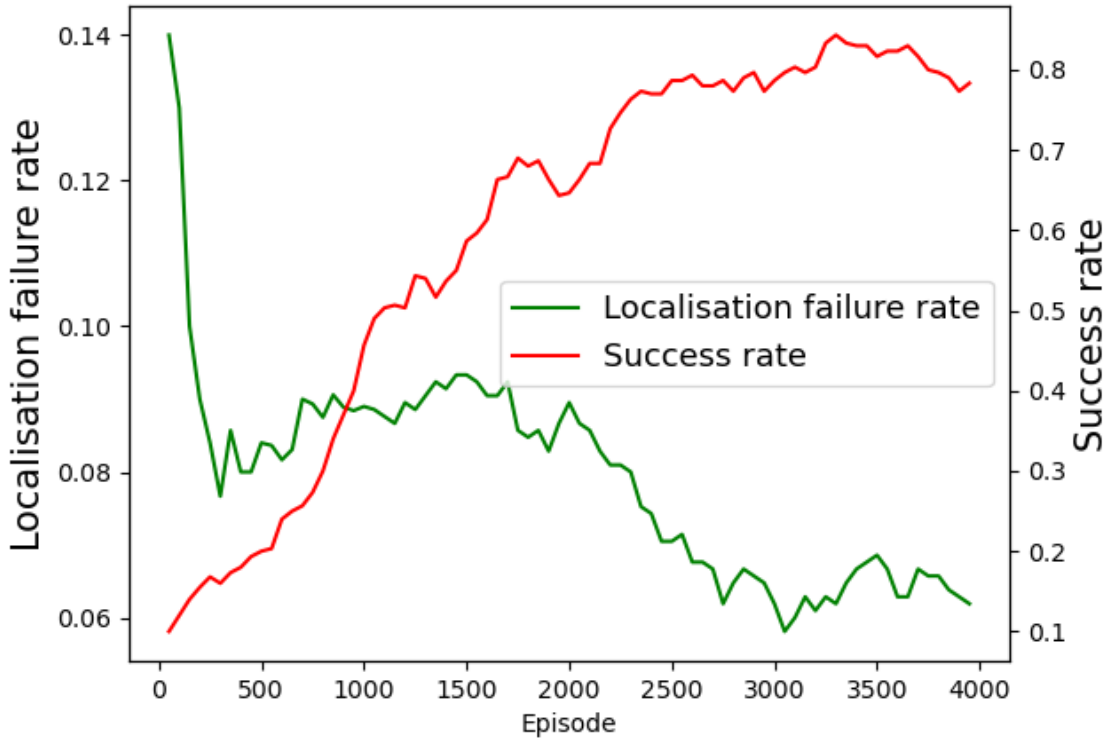


Figure 4.16: Success rate and localisation failure rate

The results of the agent's navigation policy are depicted in Fig. 4.16. The success rate defined in Eq. 4.5 reaches approximately 85% after the training of about 3300 episodes, indicating promising performance for this DQN-based learning implementation. Fig. 4.17 illustrates some trajectories of the agent trained with the proposed

method, where the red circles represent the goal regions. It is clear that the agent can reach the goal positions and avoid obstacles in the environment with different initial positions and goal positions.

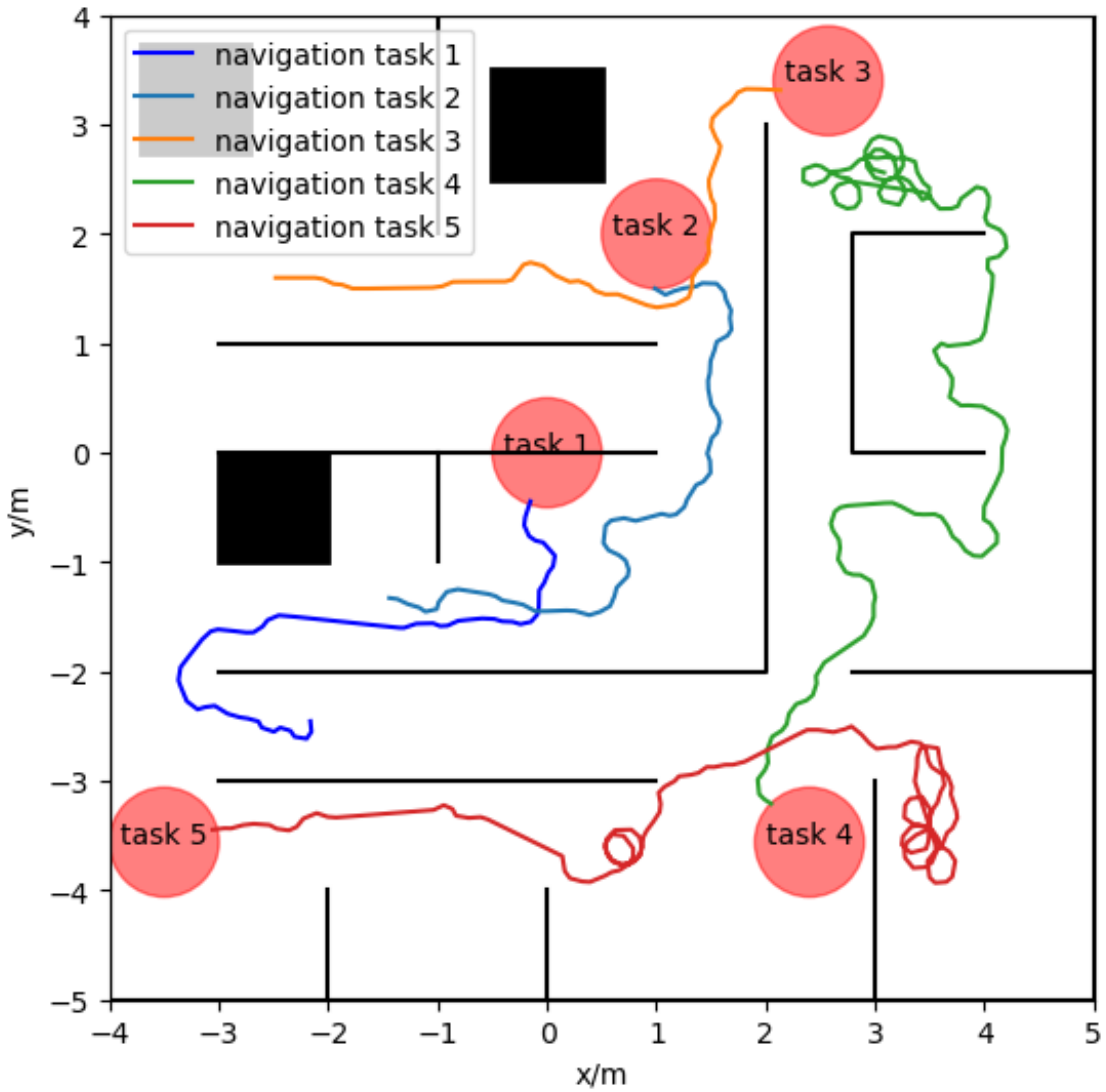


Figure 4.17: Example trajectories of the agent trained with the proposed method with randomly located start and end positions. (red circles: goal region)

Directly comparing with the original mapless navigation, which assumes the availability of ground-truth poses, would be unfair. To address this, the training procedure in this work periodically calculates failure rates specifically for cases that failed due to unsuccessful localisation problems—when $\det \mathcal{C}$ is above the threshold. As depicted in Fig. 4.16, the localisation-related failure rate drops from 14% to 6%

(green line), clearly contributing to the overall success rate of navigation. This validates the hypothesis that penalizing behaviours entering regions with deteriorated localisation is effective in improving navigation performance.

To further demonstrate the effectiveness of the proposed RL policy, specific scenarios have been created for illustration purposes. In these scenarios, the robot is initially spawned at the entrances of symmetric corridors, with the corresponding goals located at the other end of the corridor.

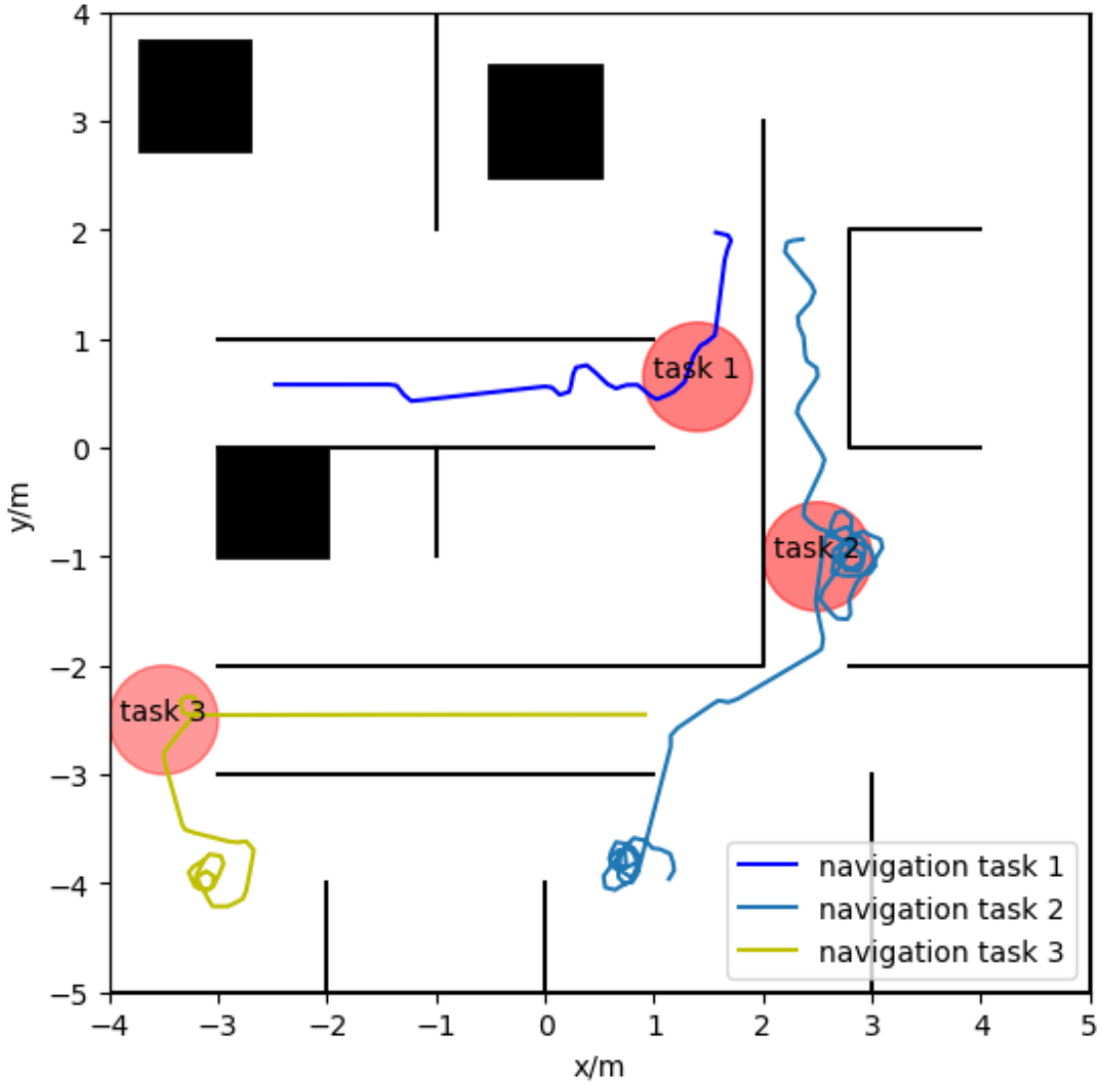


Figure 4.18: Trajectories of the baseline agent without localisation related reward r_{lost}: the robot travels directly towards goal regions without avoiding corridors. (red circles: goal region)

Navigation performance by other baseline methods: RL algorithms that do

not consider localisation performance would typically result in the robot choosing to traverse the corridor directly. As depicted in Fig. 4.18, the agent trained without the localisation-related reward r_{lost} exhibits behaviour similar to the shortest path strategy. However, traversing through corridors can lead to localisation failures and inaccurate pose estimations, as discussed previously (Fig. 4.15). Consequently, even if the agent passes through the goal regions, it may miss the targets because it lacks knowledge of its true state, ultimately ending up with an inaccurate position due to the unreliable pose estimation.

Navigation performance by the proposed method: With the proposed method in this work, as expected, the robot's capability to avoid entering localisation failure regions has been significantly enhanced. Fig. 4.19 displays trajectories generated using the new algorithm, illustrating that the robot successfully avoids corridor areas that are challenging for Lidar-based localisation.

In each task, the agent initially hesitates slightly around the corridor entrance to gather path information ahead. As the agent processes Lidar scan information, it becomes well-informed about the presence of a symmetric corridor (a challenging region for Lidar-based localisation) and opts to navigate along a more distant path. The acquired capability to avoid entering regions with deteriorated localisation is attributed to the introduction of the proposed punishment reward r_{lost} during training. The incorporation of an LSTM cell allows past observations to be encoded and transmitted as hidden state values to subsequent time steps, as illustrated in Fig. 4.6. In the context of localisation, this empowers the robot with the capability to recall its previous locations and the environmental features it has encountered. As a result, after the robot extensively explores its local surroundings, it accumulates sufficient information to make informed decisions. Ultimately, the robot can escape local minimum regions.

In addition, the navigation policy has been tested in unseen environments of different layouts, corridor entrance conditions and sizes (Fig. 4.20, Fig. 4.21, Fig. 4.22).

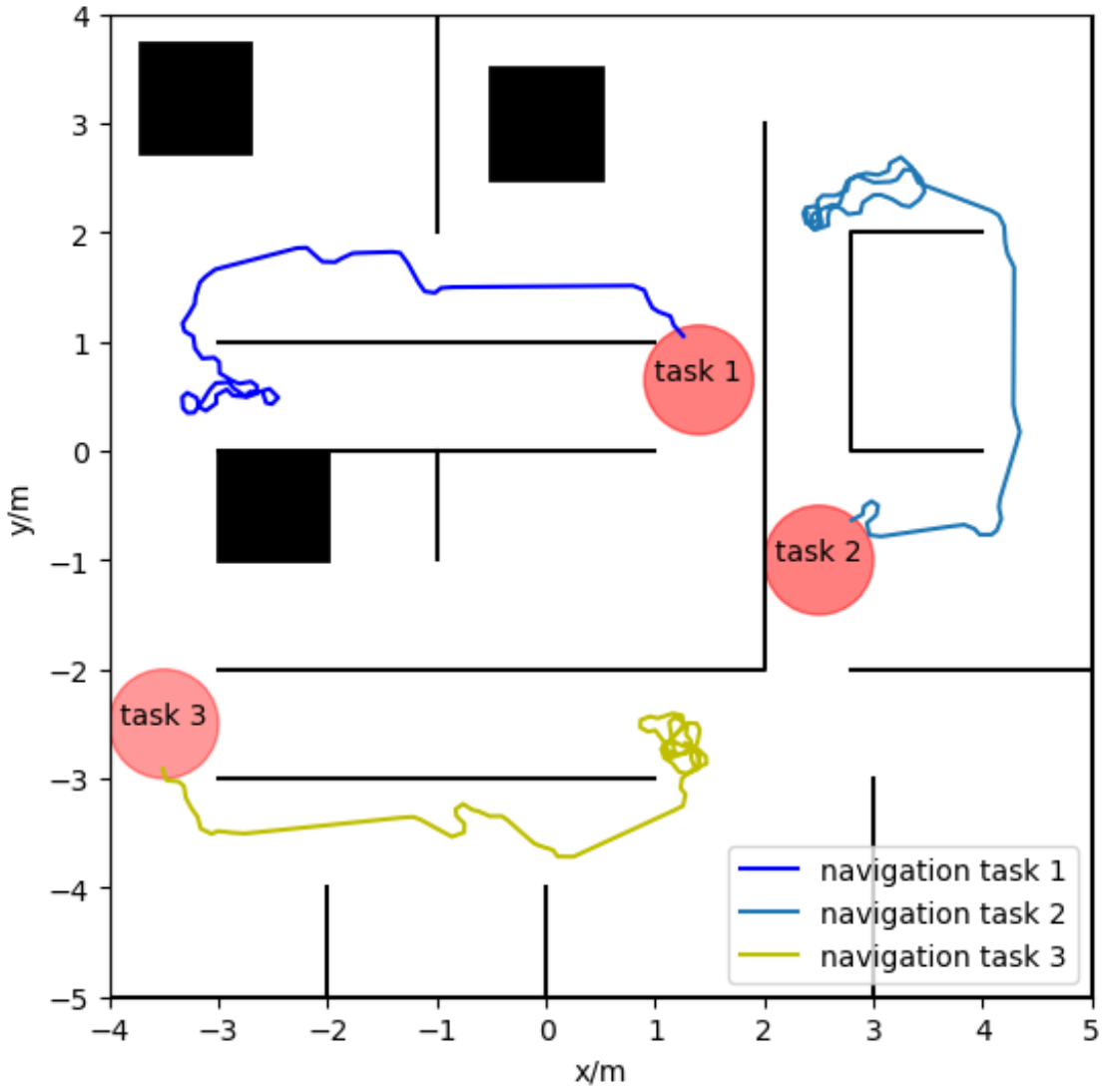


Figure 4.19: Trajectories of the agent trained with the method proposed in this chapter (red circles: goal region)

As shown in Fig. 4.20, the robot can still navigate to the goal position without travelling in the region dangerous for localisation (the red region). However, interestingly, the agent behaved more conservatively than needed in this case. At the beginning, the robot hesitated at the two entrances of the two corridors, including one corridor that is alike a symmetric corridor (the blue region in the figure), which is actually safe for localisation. The robot finally abandoned entering this passage and chose another one which presented more distinct features for localisation at the entrance of the corridor.

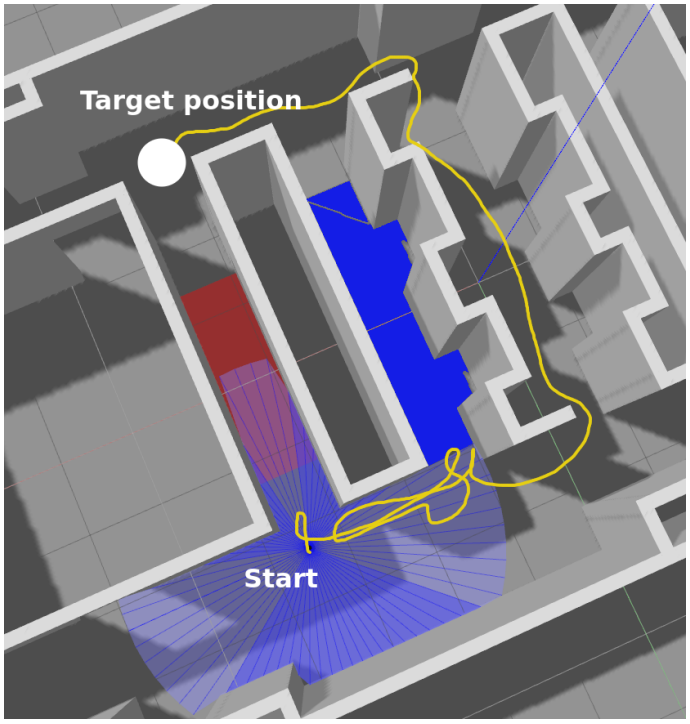


Figure 4.20: A navigation example in the unseen environment-1

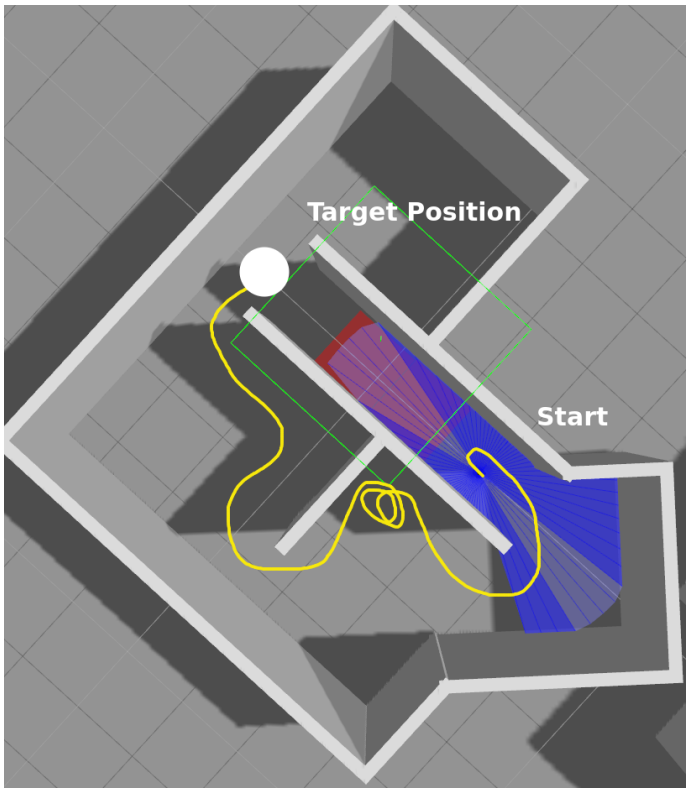


Figure 4.21: A navigation example in the unseen environment-2

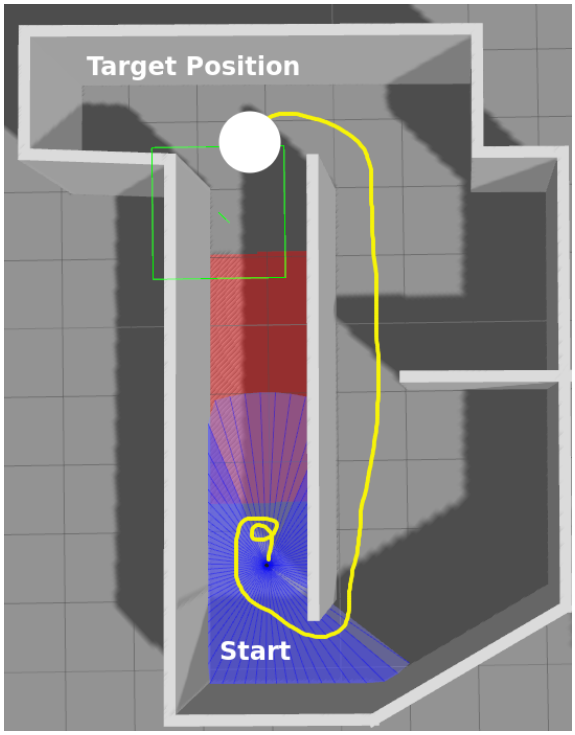


Figure 4.22: A navigation example in the unseen environment-3

4.4 Conclusion

This chapter introduces a novel RL-based mapless navigation method, eliminating the reliance on the availability of ground truth robot poses (as assumed in other works) by utilizing onboard localisation algorithms for pose estimation. To train robots to avoid navigation in regions where localisation algorithms might fail, a penalty term r_{lost} is designed to regulate robot behaviours. For outdoor navigation, localisation failure is detected when the number of observed landmarks falls below a threshold, while for indoor navigation, the determinant of the pose estimation covariance is proposed as the measure of localisation failures.

Given the use of pose estimation algorithms, the agent may encounter local minima more easily in indoor environments. To address this issue, the incorporation of history information for decision-making is proposed by integrating an LSTM cell into the robot policy network.

Experimental results demonstrate that the agent has successfully learned map-

less navigation while avoiding localisation-unsafe regions or getting trapped in local minimum regions. This achievement is attributed to the localisation-related penalty and the inclusion of historical information in the state representation.

However, it is important to note that the proposed method has been tested in relatively simple simulation environments without dense obstacles or dynamic objects. Future work could focus on evaluating the algorithm in more realistic and complex indoor and outdoor settings. Additionally, testing the algorithm with real-world robots and sensors poses an avenue for future research, necessitating the development of techniques to address the sim-to-real transfer problem.

Furthermore, it's important to highlight that this work has not integrated the indoor and outdoor navigation strategies into a unified system. Currently, they are tested independently in their respective environments. Future endeavours could focus on developing an autonomous switching module that determines whether to employ the indoor policy or the outdoor policy based on real-time observations. This can be formulated as an environment perception problem using neural network techniques. This integration would contribute to a more comprehensive and versatile robotic navigation system capable of seamlessly adapting to diverse environments and challenges.

Chapter 5

VO-Safe Reinforcement Learning for Drone Navigation

The previous chapter focuses on localisation-safe algorithms for ground vehicles. This chapter will extend the localisation-safe concept to a more complex navigation system: drones with visual odometry for localisation. Here in this chapter, the term drone is the same as UAV in previous chapters. Similarly, it is crucial for these drones to avoid areas with poor visual features, as such areas can result in degraded localisation or complete tracking loss. Several challenges have arisen from such a system with VO-safe navigation requirements.

To address the challenges, a hierarchical control scheme is proposed. In this scheme, an RL-trained policy serves as the high-level controller, generating waypoints for the next control step. A low-level controller guides the drone to reach these selected waypoints. Unlike other existing RL-based navigation approaches, the training of the high-level policy incorporates awareness of VO performance. This is achieved by introducing pose estimation-related penalties into the policy training.

To enhance a robot's ability to distinguish between perception-friendly areas and unfavourable zones, semantic images are used as input for decision-making, instead of raw images. This approach not only aids in effective decision-making but also

helps bridge the gap between simulation and real-world implementations.

The policy trained with the proposed method has been successfully tested in real-world scenarios. Thanks to the semantic inputs and the hierarchical structure, the learned policy can be directly applied to the real-world VO-safe navigation task without the need for retraining.

This chapter will start with an introduction in Section 5.1. This will be followed by the preliminary knowledge for this chapter in Section 5.2. Subsequently, the proposed method will be discussed in detail in Section 5.3. Following the method, Section 5.4 will present the evaluation results of the proposed methods. The chapter will conclude with a summary and insights in Section 5.5.

5.1 Introduction

Reinforcement learning algorithms have proven effective in guiding robots to goal points specified by relative coordinates without the need for prior environment maps, as exemplified by Tai et al. [45]. In tasks involving reaching specific goal points, the initial goal position is provided with relative coordinate numbers. The robot must then calculate its poses dynamically during navigation to reach the actual goal.

While many prior studies on reinforcement learning have concentrated on obstacle avoidance and path searching, fewer have considered the quality of pose estimation during navigation. This oversight often stems from the assumption that ground truth poses are somehow provided to the robots. In practical applications, obtaining accurate pose information is crucial. External sensor systems, such as the Vicon motion tracking system, can provide high-frequency pose information but are expensive and fixed in certain environments.

For lightweight mobile robots, onboard sensors like cameras and Lidar sensors offer a more practical solution. These sensors, combined with localisation algorithms, can provide odometry information for robots during navigation.

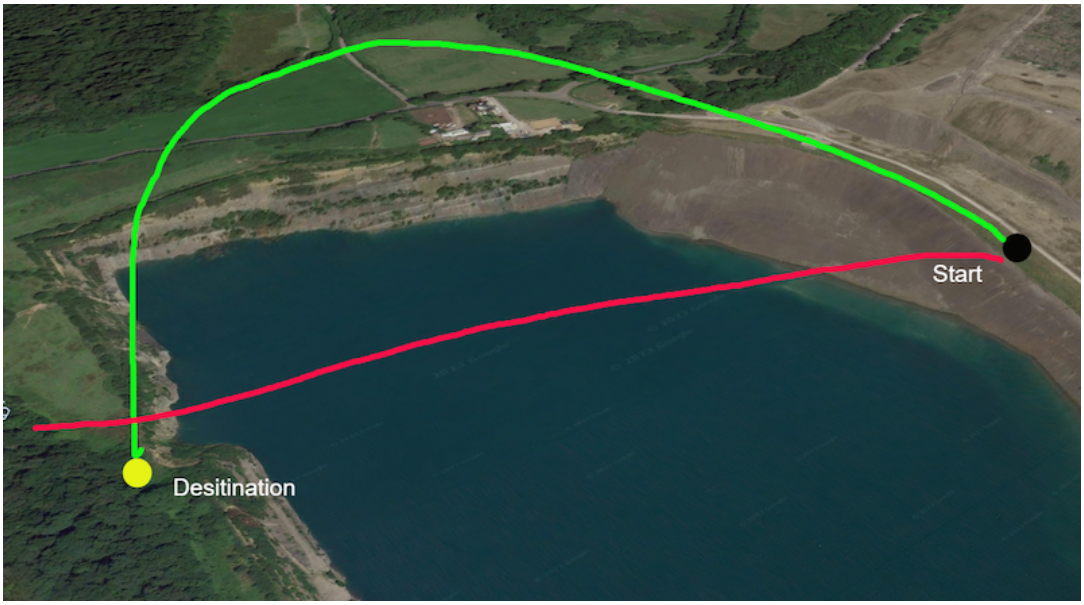


Figure 5.1: A drone travels from the start to the destination, crossing a lake. Features detected on the water surface are not consistently tracked, hence negatively impacting the VO performance. The red trajectory, despite being shorter, lacks reliable visual features for pose tracking. In contrast, the longer green trajectory enhances visual odometry performance by maximising reliable feature quality over terrain, houses, etc.

The performance of these algorithms, however, heavily relies on the observations received at different time steps. If observations along planned trajectories contain ambiguous or insufficient features, it can lead to the failure of localisation algorithms. Consequently, robots may receive inaccurate pose estimations, ultimately causing a loss of the goal position and, subsequently, the failure of the navigation task. Providing robots with inaccurate pose estimations can have catastrophic consequences, particularly in safety-critical tasks.

A real-world drone navigation scenario, as illustrated in Fig. 5.1, can help elucidate the aforementioned arguments. The drone is equipped with a down-looking camera and uses VO for pose estimation. The water area should be avoided due to insufficient and potentially harmful features from moving water, impacting pose estimation negatively. In this scenario, the green path, despite being longer in distance, is preferable to the shorter red trajectory (assuming no access to GPS information). The green trajectory enhances the drone navigation task by providing more valuable features for motion estimation. Conversely, the red path may result in the loss of

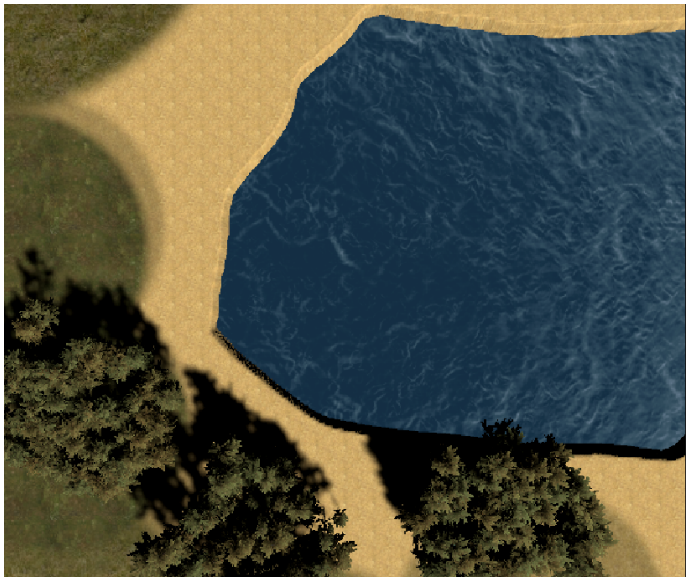
drone location when traversing over water and missing the destination at the end.

Most current research in RL-based path planning focuses solely on either efficiency, aiming for shorter paths, or safety, prioritizing collision-free routes. The above issue has been seldom noticed in previous works. Also, most RL-based robots are usually trained in simulation environments, where robot poses are always accessible. As a result, agents after training with ground truth poses tend to choose shorter paths (Fig. 5.1), ignoring the possible failed localisation in the real world, where ground truth poses are unavailable. Therefore, the assumption of the availability of the ground truth poses will introduce a huge gap between the simulation and real-world environments for deployment on the real robot.

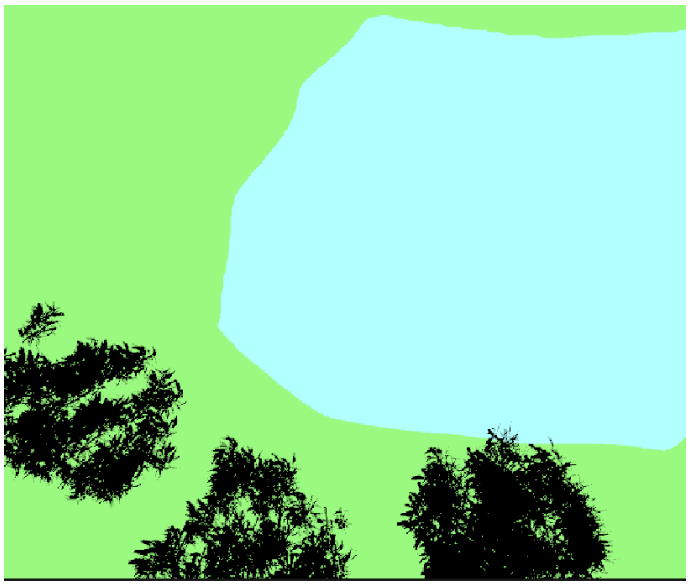
Another aspect of the sim-to-real problem arises from the fact that many reinforcement learning algorithms train policies in unreal simulated worlds, such as environments created by the Unity game engine [204] or PyBullet [205]. Visual observations obtained from such simulators differ significantly from real-world images. When these trained policy networks are presented with real-world image inputs, they may produce erroneous decisions.

To address the sim-to-real problem, this work suggests utilizing semantic inputs instead of raw image inputs. Ensuring that specific objects in simulators and the real world fall into the same semantic categories (e.g., house, tree, water) facilitates consistent input for both simulator and real-world applications. Additionally, using semantic inputs offers the advantage of distinguishing objects with similar appearances but varying effects on robot pose estimation, such as distinguishing between green terrains and moving green tree leaves. An RGB observation and its corresponding semantic observation are shown in Fig. 5.2.

For the control system, a hierarchical framework rather than an end-to-end approach is proposed. This involves a high-level controller generating short-term waypoints, coupled with a traditional low-level controller guiding the drone to these waypoints. This hybrid approach leverages the capabilities of neural networks while



(a) Simulated lake



(b) Semantic Mask

Figure 5.2: Simulated environments and semantic output from the simulator.

ensuring safety through the reliability of conventional controllers [91].

While end-to-end learning has proven effective for drone flight tasks, it presents problems in the context of navigation with VO-based pose tracking. This is particularly evident in agile motions where significant differences in observations between consecutive frames occur due to rapid and erratic movements, leading to challenges in VO feature matching.

Moreover, end-to-end policies are trained for specific robot configurations, mak-

ing them incompatible with other robots. In contrast, the high-level policy of a hierarchical framework merely decides the next waypoint without considering robot configurations. The low-level controller, tailored for individual robots, then guides the robot to the specified waypoint. This eliminates the need for retraining RL-based policies for different robots.

5.1.1 Summanry

To address the challenges, the algorithm of this chapter proposes adding an additional objective to be optimised, specifically focusing on visual odometry performance. This is introduced alongside the conventional objective of achieving the shortest path during trajectory planning, achieved by redesigning the reward space in the reinforcement learning training process.

To narrow the gap between simulation and real-world environments, the algorithm applies semantic masks to raw RGB images before inputting them into the policy. This preprocessing step enhances the model's ability to generalize across different environments.

Additionally, the algorithm advocates for the implementation of a hierarchical control system. This framework capitalizes on the advantages of safety-guaranteed conventional control while maintaining a vehicle-agnostic nature, thus eliminating the need for retraining when transitioning to other platforms.

The contributions of work introduced in this chapter are summarised below:

- It proposes an RL-based navigation framework to prevent odometry failures during flight.
- A novel reward space is introduced to encourage VO-safe behaviours.
- Semantic images are used to bridge the sim-to-real gap.
- The proposed approach combines learning-based and conventional control in a hierarchical scheme.

5.2 Preliminary

5.2.1 Drone dynamics

The drone is modelled as a 6-degree-of-freedom rigid body with a mass of m and a diagonal moment along $x-y-z$ axis of inertia matrix $\boldsymbol{J} = \text{diag}(J_x, J_y, J_z)$. Additionally, the four propellers' rotational speeds, denoted as Ω_i, are simulated as a first-order system with a time constant of k_{mot}. The input for this system is given by the commanded motor speeds $\boldsymbol{\Omega}_{\text{cmd}}$.

The drone can be represented with a 17-dimensional state, and its dynamics can be expressed as Eq. 5.1 [151, 206].

$$\dot{\boldsymbol{x}} = \begin{bmatrix} \dot{\boldsymbol{p}} \\ \dot{\boldsymbol{q}} \\ \dot{\boldsymbol{v}} \\ {}^{B}\dot{\boldsymbol{\omega}} \\ \dot{\boldsymbol{\Omega}} \end{bmatrix} = \begin{bmatrix} \boldsymbol{v} \\ \boldsymbol{q} \otimes \begin{bmatrix} 0 \\ {}^{\boldsymbol{B}}\boldsymbol{\omega}/2 \end{bmatrix} \\ \frac{1}{m}\left(\boldsymbol{q} \odot \left({}^{B}\boldsymbol{f}_{\text{prop}} + {}^{B}\boldsymbol{f}_{\text{drag}}\right)\right) + \boldsymbol{g} \\ \boldsymbol{J}^{-1}\left(\boldsymbol{\tau}_{\text{prop}} - {}^{B}\boldsymbol{\omega} \times \boldsymbol{J}\,{}^{B}\boldsymbol{\omega}\right) \\ \frac{1}{k_{\text{mot}}}\left(\boldsymbol{\Omega}_{\text{cmd}} - \boldsymbol{\Omega}\right) \end{bmatrix} \tag{5.1}$$

$\boldsymbol{q} = [q^o, \boldsymbol{q}^{\boldsymbol{v}}]^T$ is the attitude quaternion, which consists of a real part q^o and a vector $\boldsymbol{q}^{\boldsymbol{v}}$. Operator $\otimes$ stands for the quaternion multiplication, given two quaternions $\boldsymbol{q_1}$ and $\boldsymbol{q_2}$, the multiplication is:

$$\boldsymbol{q_1} \otimes \boldsymbol{q_2} = \begin{bmatrix} q_1^o q_2^o - \boldsymbol{q_1^v}^T \boldsymbol{q_2^v} \\ q_1^o \boldsymbol{q_2^v} + q_2^o \boldsymbol{q_1^v} + \boldsymbol{q_1^v} \times \boldsymbol{q_2^v} \end{bmatrix} \tag{5.2}$$

$\odot$ denotes quaternion rotation. For a quaternion $\boldsymbol{q}$ and a vector $\boldsymbol{e}$, the quaternion rotation is $\boldsymbol{q} \odot \boldsymbol{e} = \boldsymbol{q} \otimes [0, \boldsymbol{e}]^T \otimes \boldsymbol{q}^{-1}$.

Here, $\boldsymbol{g} = [0, 0, -9.81\,\text{m/s}^2]^\intercal$ represents Earth's gravity, $\boldsymbol{f}_{\text{prop}}$ and $\boldsymbol{\tau}_{\text{prop}}$ denote the collective force and torque generated by the propellers, and $\boldsymbol{f}_{\text{drag}}$ represents a linear drag term. The following shows the calculation of their values:

$$\boldsymbol{f}_{\text{prop}} = \sum_i \boldsymbol{f}_i \ , \quad \boldsymbol{\tau}_{\text{prop}} = \sum_i \boldsymbol{\tau}_i + \boldsymbol{r}_{\text{P},i} \times \boldsymbol{f}_i \ , \tag{5.3}$$

$$\boldsymbol{f}_{\text{drag}} = -\begin{bmatrix} k_{vx}^B v_x & k_{vy}^B v_y & k_{vz}^B v_z \end{bmatrix}^\top , \tag{5.4}$$

where $\boldsymbol{r}_{\text{P},i}$ denotes the position of propeller i expressed in the drone body frame, $\boldsymbol{f}_i$, $\boldsymbol{\tau}_i$ represents forces and torques produced by the i-th propeller respectively. The linear drage coefficients are denoted by (k_{vx}, k_{vy}, k_{vz}) [207]. The torques and forces generated by a propeller are commonly approximated by values proportional to the square of the rotational speeds of the propeller [151] as shown in Eq. 5.5. The thrust and drag coefficients, c_l and c_d, can be determined through experimentation on a test stand. The motor first-order time constant k_{mot} can also be acquired through measuring the rotational speed response of the propeller during these tests.

$$\boldsymbol{f}_i(\Omega) = \begin{bmatrix} 0 & 0 & c_{\text{l}} \cdot \Omega^2 \end{bmatrix}^\top \ , \quad \boldsymbol{\tau}_i(\Omega) = \begin{bmatrix} 0 & 0 & c_{\text{d}} \cdot \Omega^2 \end{bmatrix}^\top \tag{5.5}$$

A Semi-implicit Euler method with step size 2 ms is used for the drone dynamics integration.

5.2.2 Visual-odometry

Visual odometry is the process of estimating the motions of a monocular or stereo camera based on the received image sequence, which usually consists of several components including feature detection, feature matching, motion estimation, and local bundle adjustment [208].

Feature detection is to find salient keypoints like corners or blobs in one image observation that may be well-matched in other observed images, which should distinguish themselves from their immediate neighbourhood [209]. A good feature should have the property of distinctiveness and localisation accuracy (both in scale and po-

sition) [209]. There are various methods used as feature detectors (e.g. FAST [210], Harris [211], SIFT [212]). The feature matching is to find corresponding features in other images. The simplest way to match features between two images is to compare all feature descriptors in the first image to all other feature descriptors in the second image. A feature pair with similar descriptors across images are treated as the same feature observing from different camera poses. The 3-D locations of features in the real world will be estimated through triangulation with observations from different camera poses.

Motion Estimation Once there are features with known 3D poses after initialisation, the camera pose can be calculated by optimising reprojection error, which is called the perspective from n points (PnP) problem. It compute the camera orientation $\mathbf{R} \in SO(3)$ and position $\mathbf{t} \in \mathbb{R}^3$ by minimizing the reprojection error between matched 3D points $\mathbf{X}^i \in \mathbb{R}^3$ in world coordinates and keypoints $\mathbf{x}^i_{(\cdot)}$, either monocular $\mathbf{u}^i_{\mathrm{m}} \in \mathbb{R}^2$ or stereo $\mathbf{u}^i_{\mathrm{s}} \in \mathbb{R}^3$ in image frame, with $i \in \mathcal{X}$ the set of all matches [11]:

$$\{\mathbf{R}, \mathbf{t}\} = \underset{\mathbf{R},\mathbf{t}}{\operatorname{argmin}} \sum_{i \in \mathcal{X}} \rho \left(\left\| \mathbf{u}^i_{(\cdot)} - \pi_{(\cdot)} \left(\mathbf{R}\mathbf{X}^i + \mathbf{t} \right) \right\|^2_{\Sigma} \right) \tag{5.6}$$

where ρ is the robust Huber cost function and Σ the covariance matrix associated to the scale of the keypoint. The projection functions $\pi_{(\cdot)}$, monocular π_{m} and rectified stereo π_{s}, are defined as follows:

$$\pi_{\mathrm{m}} \left(\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \right) = \begin{bmatrix} f_x \frac{X}{Z} + c_x \\ f_y \frac{Y}{Z} + c_y \end{bmatrix}, \pi_{\mathrm{s}} \left(\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \right) = \begin{bmatrix} f_x \frac{X}{Z} + c_x \\ f_y \frac{Y}{Z} + c_y \\ f_x \frac{X-b}{Z} + c_x \end{bmatrix} \tag{5.7}$$

where (f_x, f_y) represents the focal length, (c_x, c_y) denotes the principal point and b is the baseline, all of which can be estimated from calibration.

Local Bundle Adjustment Different from the motion estimation step, it op-

timises poses and map points at the same time. This step optimises a set of camera poses from a sequence of frames $\mathcal{K}_L$ within a time window and all points $\mathcal{P}_L$ seen in those frames. The optimisation problem is the following:

$$\begin{aligned} \{\mathbf{X}^i, \mathbf{R}_j, \mathbf{t}_j | i \in \mathcal{P}_L, j \in \mathcal{K}_L\} &= \underset{\mathbf{X}^i, \mathbf{R}_j, \mathbf{t}_j}{\operatorname{argmin}} \sum_{j \in \mathcal{K}_L} \sum_{i \in \mathcal{P}_L} \rho\left(E_{ij}\right) \\ E_{ij} &= \left\| \mathbf{u}^i_{(\cdot)} - \pi_{(\cdot)}\left(\mathbf{R}_j \mathbf{X}^i + \mathbf{t}_j\right) \right\|^2_{\Sigma} \end{aligned} \tag{5.8}$$

5.3 Method

As described, the overall target of the problem is to navigate a drone to a designated goal within a specified time frame, while also avoiding regions unfavorable to visual odometry. The problem can be formulated as

$$\min_{\pi(x)} \mathcal{J}_o = \int_0^T \|\boldsymbol{p(t)} - \mathit{Goal}\|^2 \mathrm{d}t, \tag{5.9a}$$

$$s.t. \quad \boldsymbol{x_{t+1}} = f(\boldsymbol{x_t}, \pi(\boldsymbol{x_t})), \tag{5.9b}$$

$$x_0 \in \mathcal{X}, \tag{5.9c}$$

$$\boldsymbol{p(t)} \in \mathcal{P}, \ \forall t \in [0, T], \tag{5.9d}$$

where $\boldsymbol{x}$ represents the drone state and p specifies the drone position. Eq. 5.9b represents the drone's dynamic model which has been described in detail in the preliminary Section 5.2.1 and $\mathcal{X}$ is the possible initial state set. $\mathcal{P}$ denotes areas with high-quality visual features suitable for VO-based pose tracking. $\pi(\boldsymbol{x_t})$ is the policy to be optimised subject to dynamics (Eq. 5.9b) and VO constraints $\mathcal{P}$. The proposed method focuses on policy training within collision-free environments, emphasizing visual odometry performance. This assumption is reasonable, considering that collision detection is typically not required for open-space navigation.

To address such a problem, this chapter proposes a reinforcement learning-based algorithm with a hierarchical control flow. The overall framework and key modules

will be illustrated in the following subsection.

5.3.1 Design of the overall system

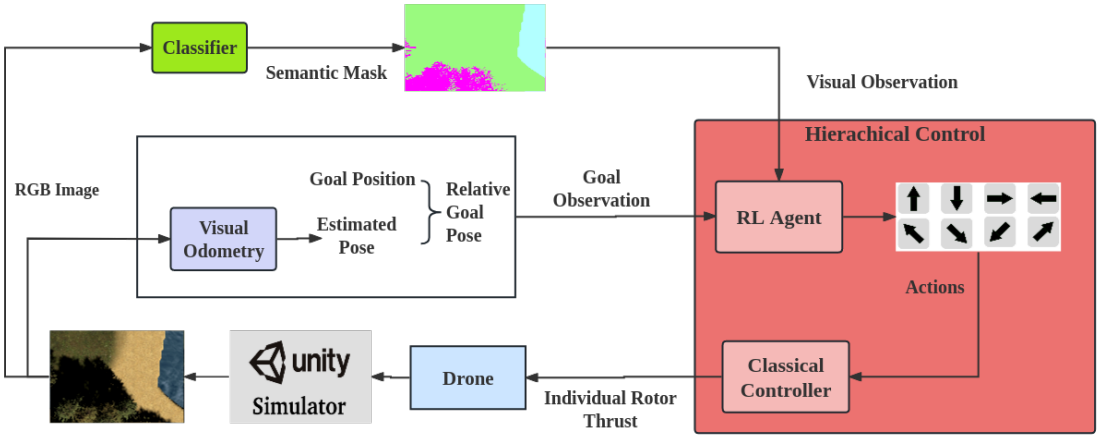


Figure 5.3: The proposed overall system framework. This thesis proposes a hierarchical control scheme tailored for the VO-safe navigation task for drones. It suggests incorporating a previously overlooked VO module for localisation during training and using semantic images to alleviate the sim-to-real gap issue.

A hierarchical RL-based control scheme is proposed in this work. The proposed framework and key modules are illustrated in Fig. 5.3. A drone equipped with an RGB camera is employed for VO-based navigation. Firstly, raw images undergo processing by the VO module to conduct pose estimation. With the ultimate goal position, this process yields relative goal positions, contributing to the RL agent's observations. The raw images are also input into a classifier to generate semantic images, constituting another component of the observations. Using these observations, the reinforcement learning agent decides on one action from eight possible moving directions. On the other hand, at the low-level control, a classical controller computes individual rotor thrusts to guide the drone along the selected direction for a predetermined fixed distance. This process repeats until the goal position is successfully reached.

The subsections will begin by providing a brief recapping of the fundamental concepts of reinforcement learning and the PPO algorithm. Following this, the

implementation details proposed in this chapter to address the VO-safe navigation problem will be outlined. Finally, a detailed description of the low-level controller will be provided.

5.3.2 Design of the VO-safe high-Level controller

This work adopts the PPO-clip method as it is easier for implementation and has proven to be more efficient than PPO-penalty in work [190]. The updating process of PPO-clip is through optimising the following objective function:

$$\theta_{k+1} = \arg\max_{\theta} \mathop{\mathrm{E}}_{s,a\sim\pi_{\theta_k}} [L(s, a, \theta_k, \theta)] \tag{5.10}$$

where L is given by

$$L(s, a, \theta_k, \theta) = \min\left(\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)} A^{\pi_{\theta_k}}(s, a), g(\epsilon, A^{\pi_{\theta_k}}(s, a))\right) \tag{5.11}$$

in which ϵ is a small hyperparameter and $g(\epsilon, A) = \begin{cases} (1+\epsilon)A & A \geq 0 \\ (1-\epsilon)A & A < 0. \end{cases}$

The intuition behind this is to increase the possibility of actions ($\pi_\theta(a|s)$) that are better than others, evaluated by the advantage $A^\pi(s, a)$ and lower the probabilities of worse actions. Clipping is employed to limit the update between the new and old policies, preventing divergence. For detailed information, please refer to Section 3.1 and Section 3.2.2. Subsequently, the implementation details of the proposed novel VO-safe navigation RL training will be presented.

Implementation details of VO-safe RL

The following section outlines the details of the proposed VO-safe RL-based navigation policy training. The state s is composed of the estimated relative goal position p_g and the observed images O_{img}: $s = [p_g, O_{img}]$. The action space A consists of 8 discrete actions, each representing movement along one of 8 possible directions for

a distance of 1 meter. The discretized action space can help accelerate and stabilise the training process.

VO-safe Reward space is designed to involve considering the drone's reliance on visual odometry for pose estimation. Hence, the agent's objective is extended beyond optimising path length to avoid actions that may result in VO failures. Unlike many prior research works, the proposed method introduces an additional penalty based on the VO status. The reward space is defined as follows:

$$R_t = R_s(p(t)) + R_G(p(t)) + R_{VO} \tag{5.12}$$

$R_s(p(t))$ is the progress reward designed to encourage the agent to reduce the distance from the robot to the goal:

$$R_s(p(t)) = \alpha \times (d_{t-1} - d_t) \tag{5.13}$$

where d_t is the distance from the robot to the goal at time step t and α is the distance weight. The goal reaching reward $R_G(p(t))$ is:

$$R_G(p(t)) = \begin{cases} 10 & d_t <= d_g \\ 0 & \text{otherwise} \end{cases} \tag{5.14}$$

A positive reward of 10 is provided when the distance to the goal is within a threshold d_g.

The last term R_{VO} is the VO-related reward for penalising actions leading to the deterioration of VO-based tracking, which is usually ignored in methods in the literature. Tracking is considered lost or failed when consecutive frames are not consistent with each other or do not observe enough features as instructed by

Eq. 5.6. R_{VO} is defined as:

$$R_{VO} = \begin{cases} -10 & \text{if visual odometry failed} \\ 0 & \text{otherwise} \end{cases} \tag{5.15}$$

The scales of the reward and penalty (10 or -10) are selected empirically though experiments.

Design of the policy and value function networks. The proposed networks are shown in Fig. 5.4. The training procedure is conducted in the Flightmare simulator [195], a Unity game engine-based simulator responsible for rendering raw and semantic images (480x360x3). Semantic images are converted and downsampled to grayscale (240x180x1). The grayscale image, along with the relative goal position, undergoes feature extraction shared by both the policy network and the value function network represented by a module consisting of 3 CNN layers and a linear fully connected (FC) layer.

Following feature extraction, the Critic calculates the value function through two additional FC layers, while the Actor determines the optimal action for the low-level control through 3 extra FC layers. The resulting control signals from the low-level controller are sent to the simulator, updating the drone state.

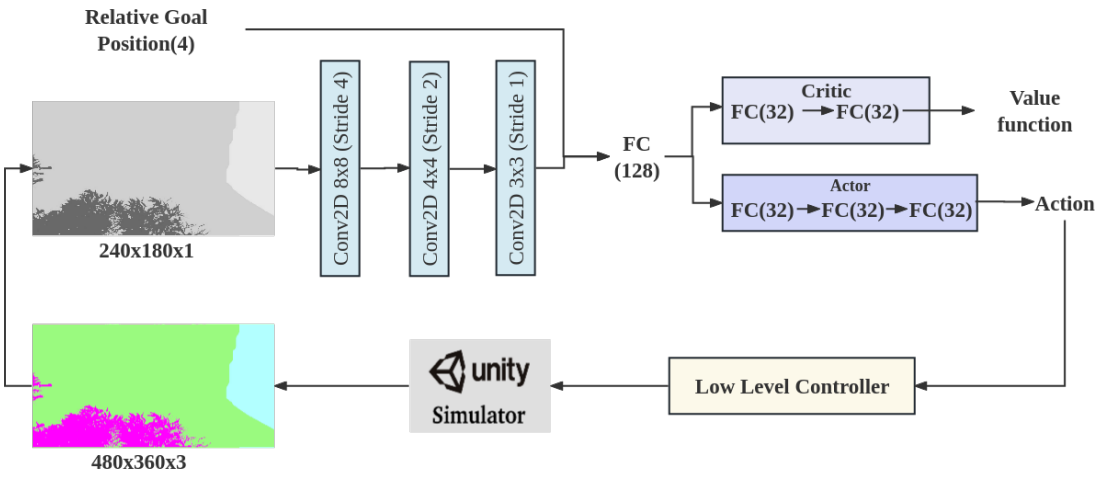


Figure 5.4: High-level policy network

The training procedure is shown in the Algorithm. 2

Algorithm 2 VO-safe high-level policy training

Input: initial policy parameters θ_0, initial value function parameters ϕ_0 with network structures described in Section 5.3.
for $k = 0, 1, 2, ...$ **do**
  Initialize rollout buffer D
  for $t = 1, ..., m$ **do**
    Sample an action from current policy $\pi(s, \theta_k)$.
    Execute action a_t by the proposed low-level controller in Section 5.3.3 and receive reward r defined in Eq. 5.12 and new state s_{t+1}.
    Store transition (s_t, a_t, r_t, s_{t+1}) in the rollout buffer D.
  end for
  Compute rewards-to-go $\hat{R(t)} = \sum_{h=t+1}^{T} \gamma^{h-t-1} R_h$ for trajectories in
  Compute advantage estimates $\hat{A}_t$ based on the value function V_{ϕ_k}
  Update the policy by maximizing importance sampling weighted objective:

$$\theta_{k+1} = \arg\max_{\theta} \frac{1}{|\mathcal{D}_k|T} \sum_{\forall \tau \in \mathcal{D}} \sum_{t=0}^{T} \min \left(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_k}(a_t|s_t)} A^{\pi_{\theta_k}}(s_t, a_t), g(\epsilon, A^{\pi_{\theta_k}}(s_t, a_t)) \right) \tag{5.16}$$

  via stochastic gradient ascent with Adam optimiser
  Update value function network by regression on mean-square error on samples:

$$\phi_{k+1} = \arg\min_{\phi} \frac{1}{|\mathcal{D}_k|T} \sum_{\forall \tau \in \mathcal{D}} \sum_{t=0}^{T} (V_\phi(s_t) - R(t))^2 \tag{5.17}$$

  via gradient descent algorithm with Adam optimiser.
end for

5.3.3 Designing of the Low-level controller

The low-level controller employs a classic PID controller to compute control signals u for motor controllers. This enables the drone to move toward the next waypoint determined by the high-level policy. The low-level controller follows a cascaded control structure, comprising a position controller followed by an attitude controller, as depicted in Fig. 5.5. The position PID controller receives position commands $p_d = [p_x, p_y, p_z]_d$, which outputs the desired attitude $\mathbf{\Theta}_d$ for the attitude controller

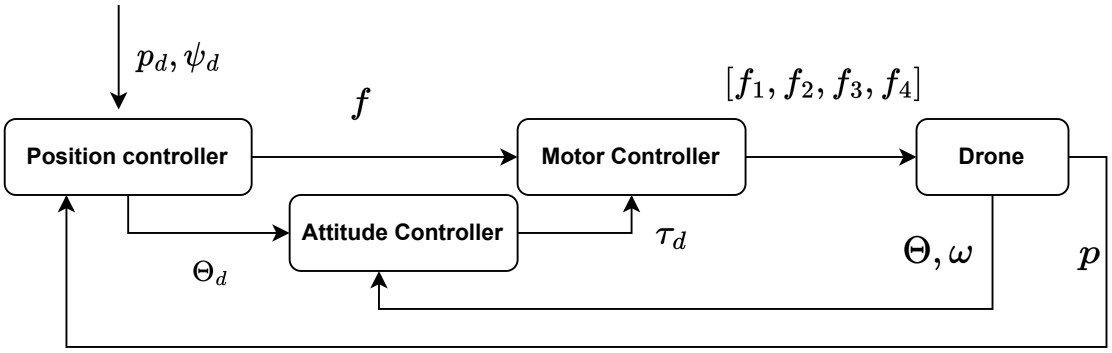


Figure 5.5: Low level controller

and also the collective thrust:

$$\begin{aligned} \mathbf{\Theta}_{hd} &= -g^{-1}A_{\psi}^{-1}(-K_{hD}(\dot{p_h}) - K_{hP}(p_h - p_{hd})), \\ f &= m(g + k_{zD}(\dot{p_z} - \dot{p_{zd}}) + k_{zP}(p_z - p_{zd})), \\ \mathbf{\Theta}_d &= [\mathbf{\Theta}_{hd}, \psi_d], \end{aligned} \tag{5.18}$$

where the control of horizontal position $p_h = [p_x, p_y]$ and altitude p_z are decoupled. The attitude control $\mathbf{\Theta} = [\psi, \theta, \phi]$ is also separated into two parts: $\mathbf{\Theta}_h = [\theta, \phi]$ and yaw ψ. The desired yaw ψ_d is provided by the task and $A_\psi = \begin{bmatrix} \sin\psi & \cos\psi \\ -\cos\psi & \sin\psi \end{bmatrix}$. m and g are the drone mass and gravity, respectively. K_{hD}, K_{hP}, K_{zD} and K_{zP} are the corresponding derivative and proportional weights.

The attitude PID controller calculates the desired moment τ_d such that the drone can fly in desired attitudes:

$$\begin{aligned} \omega_d &= -K_{\mathbf{\Theta}}(\mathbf{\Theta} - \mathbf{\Theta}_d), \\ e_\omega &= \omega - \omega_d, \\ \tau_d &= -K_{\omega P}e_\omega - K_{\omega I}\int e_\omega - K_{\omega D}\dot{e_\omega}, \end{aligned} \tag{5.19}$$

$K_{\mathbf{\Theta}}$ is designed to control the drone attitude to converge to desired attitude following an expected trajectory, and $K_{\omega P}$, $K_{\omega I}$, $K_{\omega D}$ denote the coefficients for the proportional, integral, and derivative terms, respectively. $u = [\tau_d, f]$ can then be

used to calculate individual motor control signals $[f_1, f_2, f_3, f_4]$.

5.4 Evaluation

5.4.1 Experiments setup

Hierarchical RL Agent (VO-safe). To simplify the nomination, the method proposed by this chapter is referred to as VO-safe. To train the proposed PPO agent, in each episode, the robot is spawned at a random position. The goal position is also randomly assigned. At each time step, the drone flies along the RL-decided direction for 1 meter. Each episode is terminated and restarted when any of the following cases occurs: 1) the drone reaches the goal within an acceptable threshold; 2) visual odometry reports failures; or 3)the predefined maximum time steps are reached.

End-to-end State-based RL Agent (GT-state-based). An end-to-end state-based agent from previous research [151] is also trained as the baseline for comparison. Ground truth state information $s = [p_g, \Theta, \omega]_{gt}$ is provided instead of images and VO inputs. In this case, the agent decides individual motor control signals $[f_1, f_2, f_3, f_4]$ directly. The reward space is similar to VO-safe, excluding the VO-related reward component R_{VO}. The overall training procedure is akin to VO-safe but eliminates the necessity for a low-level controller and follows the standard PPO training algorithm. For convenience, it is named **GT-state-based** in the following sections.

5.4.2 PPO training results

Fig. 5.6 displays the average rewards during the training of the policy. The state-based agent, which has access to ground-truth state information from the simulator, achieves the highest average reward after training, serving as a benchmark reference. This allows for the evaluation of the performance of the vision-based VO-safe agent proposed in this work. Notably, the performance of our visual agent is comparable

to that of the GT-State-based agent in terms of the total rewards received as shown in Fig. 5.6.

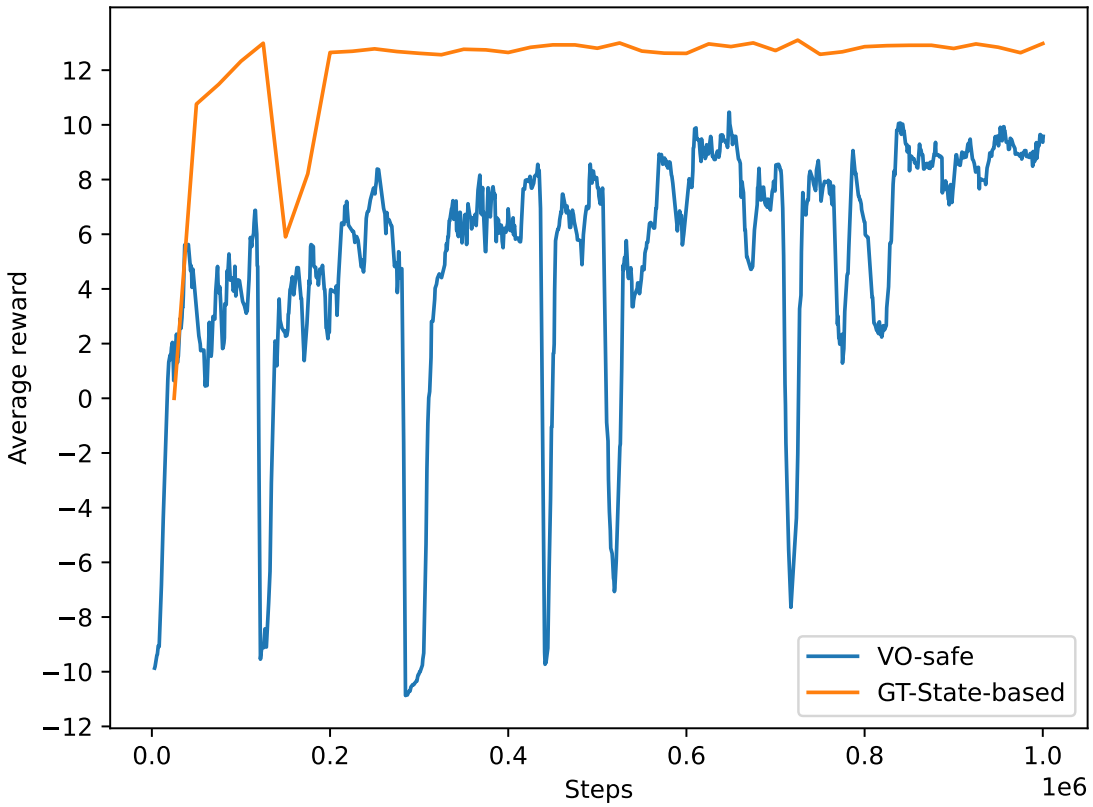


Figure 5.6: Average rewards

The test success rates are presented in Table 5.1. The proposed algorithm VO-safe achieved a success rate of 0.78 out of 1.0 during testing. In contrast, the GT-state-based agent, which is trained with ground truth poses and benefits from their availability during testing, can always reach the given goal.

VO-State-based. However, if the state-based agent is fed with VO-estimated poses (named **VO-State-based**), the success rate drops significantly to 0.56. The main failure reasons include the drone flying over VO-unfavoured regions, such as water, and occurrences of excessive flying velocities $\dot{p}$ or attitudes Θ, which will be discussed in detail in Section 5.4.3.

Fig. 5.7 illustrates trajectories of different agents, with red circles representing the goals. The state-based agent opts to fly directly towards the goal, lacking

Table 5.1: Success rates and VO performance

Metrics	VO-safe	GT-State-based	VO-State-based
Success Rate	0.78	1.0	0.56
RMSE (m)	0.38 ± 0.31	N/A	1.10 ± 0.5

consideration of visual information for pose estimation. This leads to failures when ground truth is unavailable, and visual odometry (VO) is utilized for pose estimation, exemplified by the VO-State-based agent in the figure. In contrast, the agent trained with the algorithm proposed by this paper chooses a path that avoids dangerous regions where there are no reliable features for VO, such as water or trees, known for ambiguous and less distinct features.

Fig. 5.8 displays trajectories in more photo-realistic environments not encountered during training. Remarkably, the agent proposed by this work successfully accomplishes tasks in these new environments without requiring retraining, thanks to its utilization of semantic images as input.

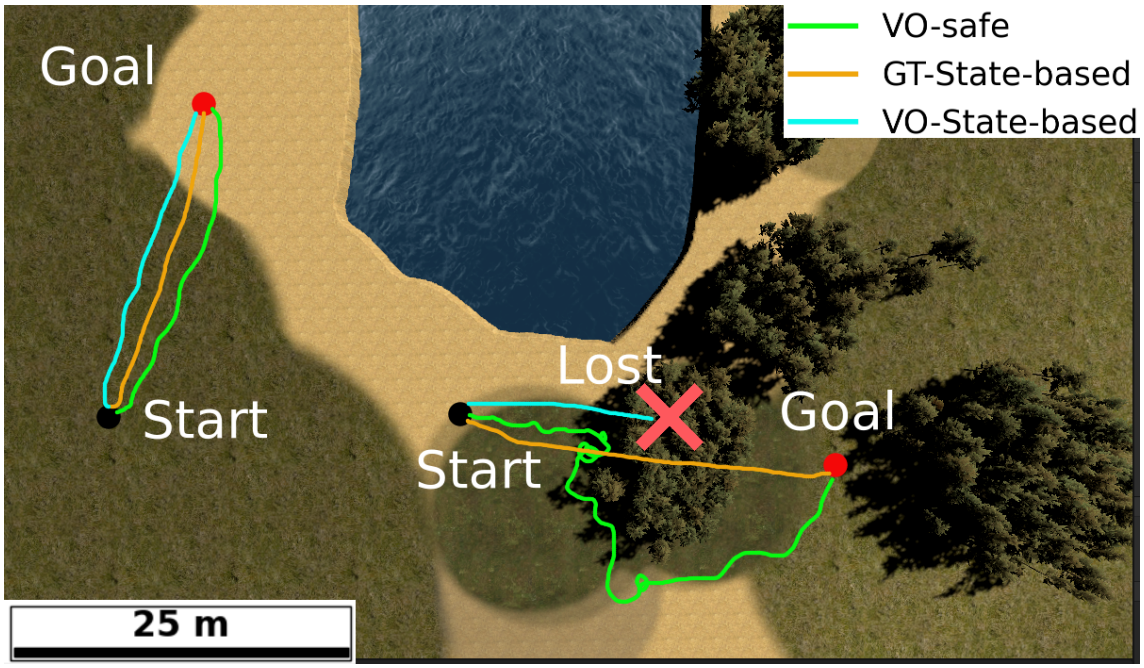


Figure 5.7: The VO-safe agents can navigate successfully while the VO-State-based fails due to VO being lost over trees.

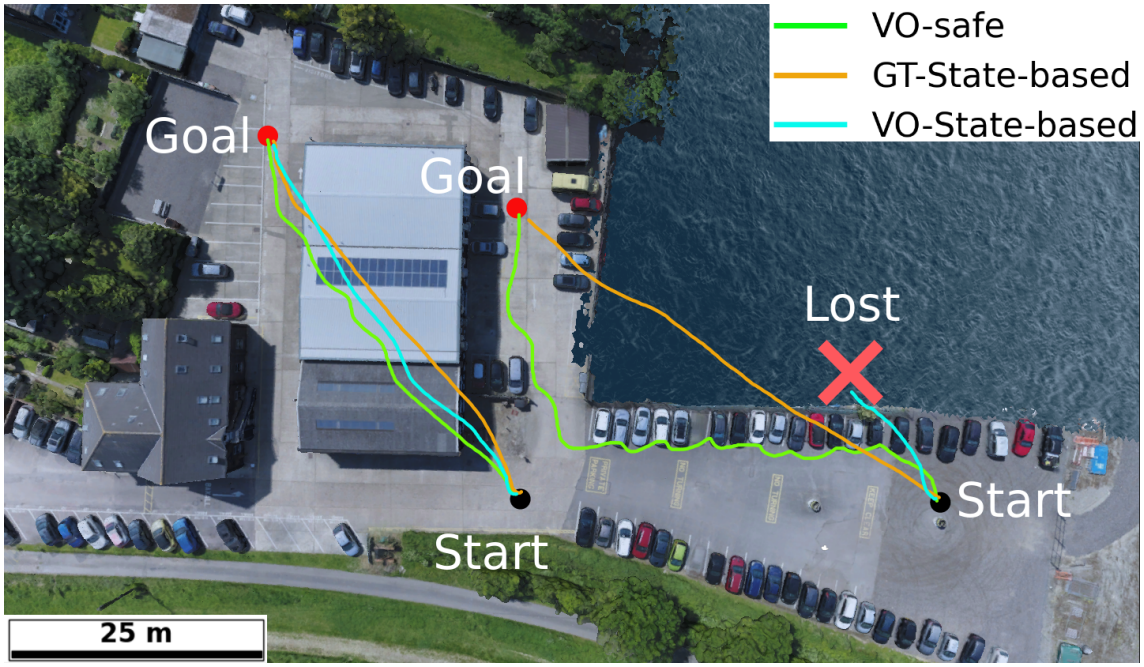


Figure 5.8: In a photo-realistic environment. VO-State-based fails above the water region, while VO-safe succeeds without retraining as with GT-State-based.

5.4.3 Hierarchical architecture

This section justifies the use of a hierarchical architecture by analyzing the impact of drone dynamics on visual odometry (VO) performance.

As illustrated in Table 5.1, the hierarchical method significantly improves the accuracy of poses estimated by VO, compared to the VO-State-based approach. This improvement is mainly attributed to the short-term goals generated by the high-level policy, preventing the drone from making abrupt and violent movements.

In contrast, the end-to-end VO-State-based approach tends to induce excessive motions to achieve high speed, leading to large discrepancies between consecutive images and, consequently, inaccurate localisation. This discrepancy is visually depicted in Fig. 5.9.

To assess the impact of excessive motions on visual odometry (VO) performance, the two agents, the VO-safe agent and the VO-state-based, are assigned an identical task where the drone is tasked with reaching four goals located at the corners of a square in a simulation environment (Fig. 5.10a and Fig. 5.10b).

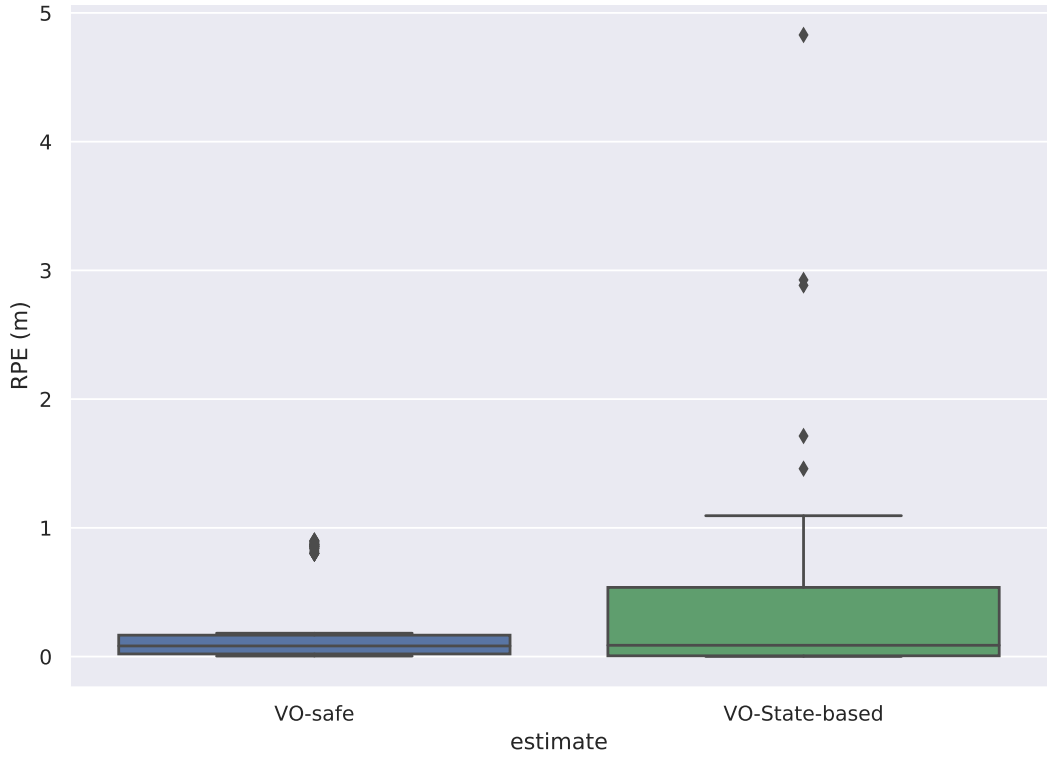
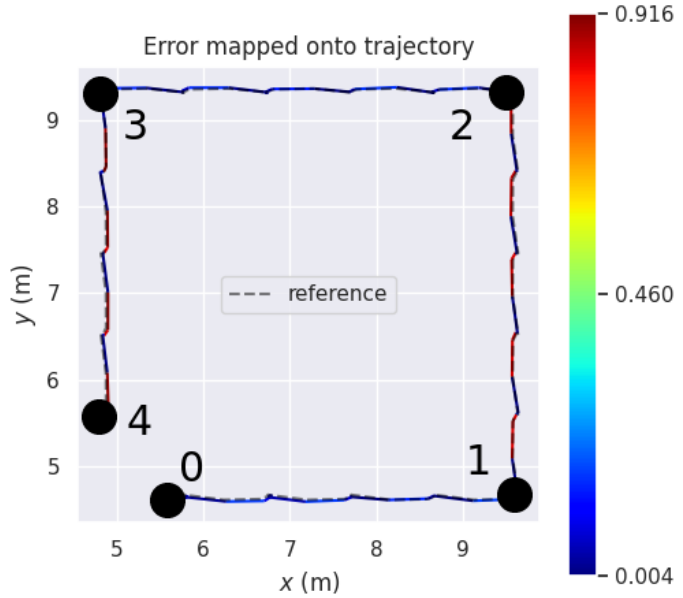


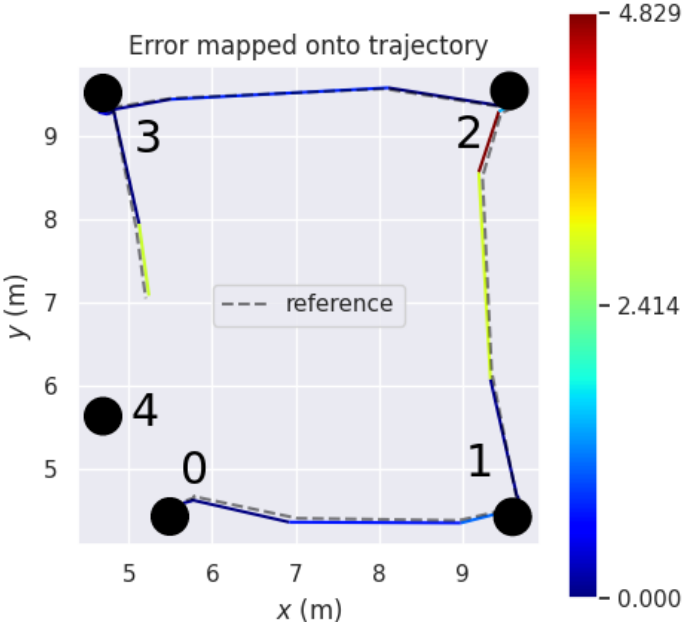
Figure 5.9: VO performance

The VO-safe method decomposes each task into a series of short segments, each 1 meter long. The VO estimation errors are then projected onto the drone's trajectory. In contrast, the end-to-end agent (VO-state-based) directly moves to the next target, with each waypoint 5 meters away. The drone's rapid and violent movements to reach each subsequent target lead to a drop in localisation accuracy, as depicted in Fig. 5.10b. Notably, VO loses track during the last segment, rendering the drone unable to navigate to target 4. Fig. 5.11 illustrates the attitude response to various distance commands.

Excessive attitude changes and high velocities have a negative impact on visual odometry (VO) performance and can lead to VO failures as shown in Fig. 5.10. Hence, previous research works which do not consider pose estimation quality during training could lead to failures during deployment where ground truth pose estimation is not available. This is part of the sim-to-real problem, which has not been studied thoroughly.



(a) VO performance (VO-safe agent)



(b) VO performance (VO-State-based)

Figure 5.10: VO Performances (VO-safe vs VO-State-based)

In this work, the hierarchical architecture is employed to enhance VO performance and reduce VO failure rates by decomposing long-distance paths into short-range goals (1 meter in this work), which introduce gentle drone response as shown in Fig. 5.11.

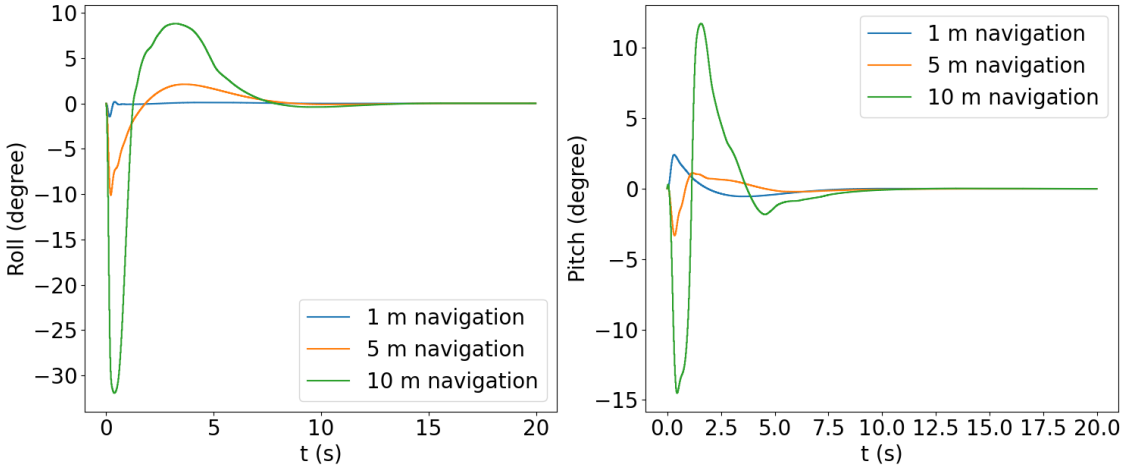


Figure 5.11: Drone attitude responses to navigation with different goal distances.

5.4.4 Real-world experiments

Real-world experiments are carried out to verify the proposed method. The drone configuration is shown in Fig. 5.12. A down-looking camera RealSense D435 is used for drone perception and visual odometry. An onboard computer (Intel NUC 11 pro) is used for image segmentation, high-level policy calculation and VO computation. No ground station computers were required.

Figure 5.12: Drone configuration

VO-failure due to tree movements

A real-world test was conducted to demonstrate that the movements of trees can cause visual odometry failures. The testbed comprises an OAK-DW camera and a 3DM-gx5-45 IMU unit placed in a stationary position. VINS-mono [213], a VO algorithm integrated with IMU inputs which has been proven to perform well for mobile robots, is utilized for pose estimation. High-accuracy GPS-RTK signal poses serve as a baseline for comparison. The observed RGB image is depicted in Fig. 5.13.

It is observed that the algorithm identified numerous features on trees, as shown in Fig. 5.14. This abundance of features should be sufficient for motion estimation if these features are static. However, during the test, the trees were moving due to environmental winds leading to random movement of observed features, causing the estimation algorithm to find consecutive frames inconsistent with each other. Consequently, the estimation relies solely on IMU integration for pose estimation, leading to drift in the estimation, as illustrated in Fig. 5.15. Therefore, it is reasonable for drones to avoid flying over areas with trees during navigation.

Figure 5.13: Trees observation

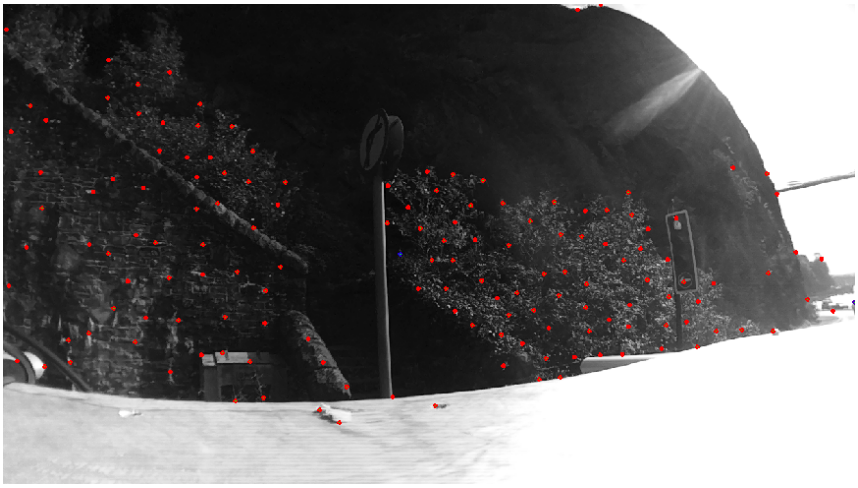


Figure 5.14: Features detected on trees

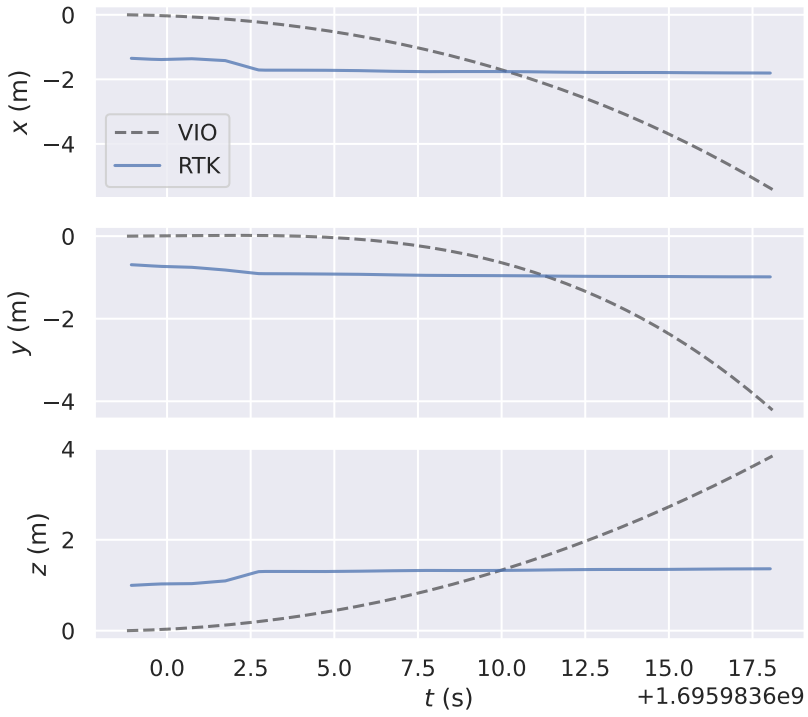


Figure 5.15: VO-failure due to trees movements: RTK is the nearly ground-truth value

VO-safe based navigation

The test scenario is depicted in Fig. 5.16. A region of the floor covered by white paper serves as the featureless area. The starting point (black) and the goal (red) are positioned on opposite sides of the white paper region. While the floor boasts rich visual textures suitable for visual odometry tracking, the white paper region

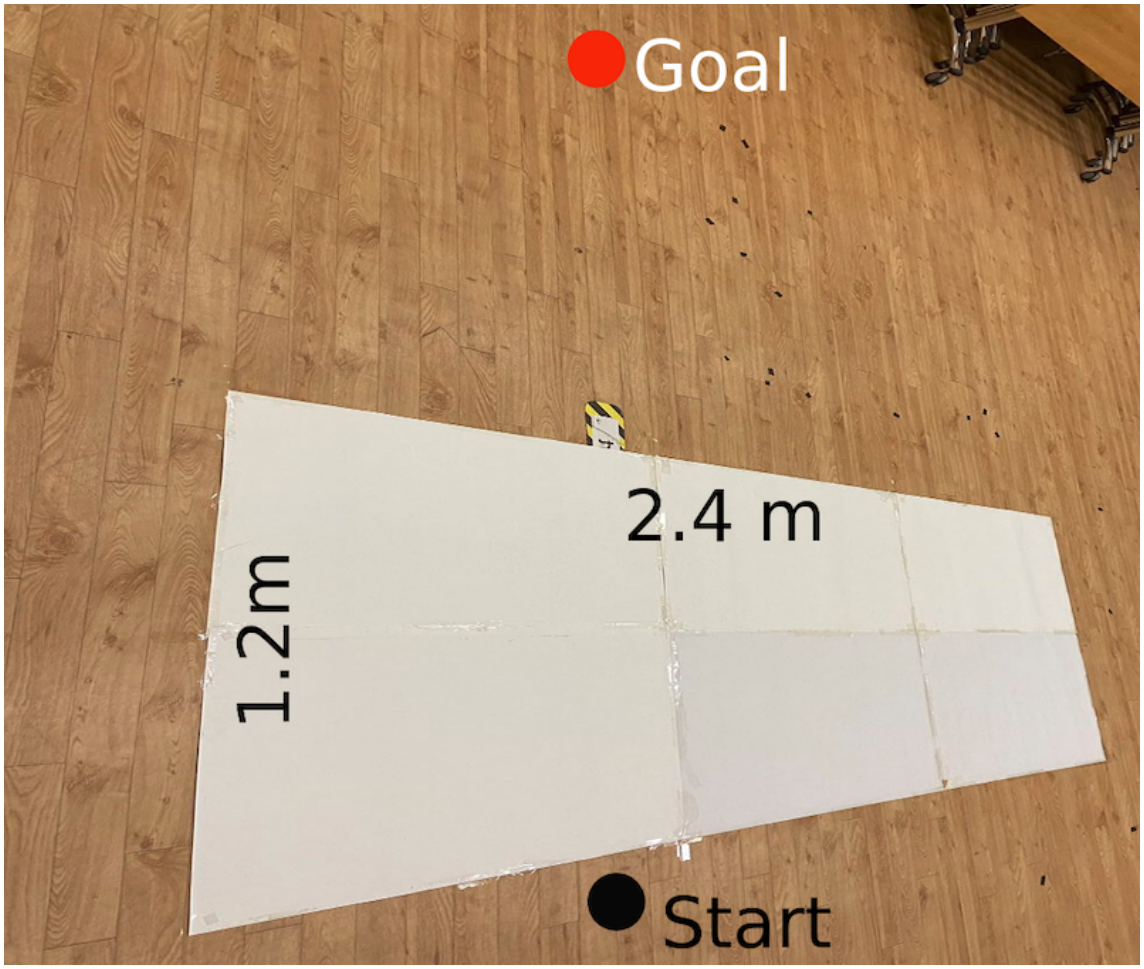
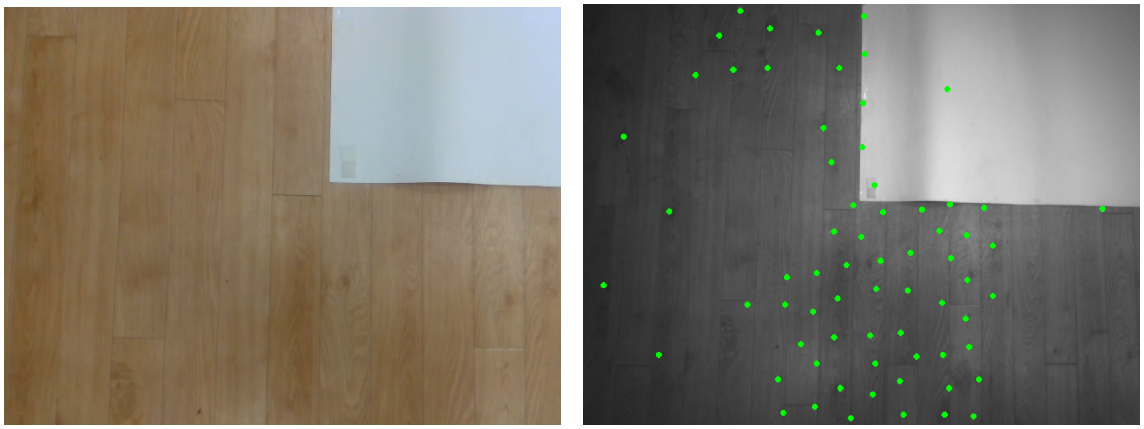


Figure 5.16: Test scenario

lacks textures as shown in Fig. 5.17. It is crucial for the drone to avoid entering the featureless area to prevent VO failures. Notably, the policy is not retrained for real-world experiments, thanks to the hierarchical scheme and the use of semantic inputs.

As illustrated in Fig. 5.18, the drone deviates from a straight-line path towards the goal, opting for a longer trajectory to bypass the paper. The plotted trajectory represents the visual odometry estimated result, which remains continuous without failures. Despite the initial crossing of the white region's edge, the drone, at an altitude of 0.7m, can still capture the textured floor within its field of view.

Last, The failed VO behaviour is displayed when the drone enters entirely the white paper region (see the area with red diagonal lines in Fig. 5.19, where no floor



(a) Floor observation. (b) Feature detected.

Figure 5.17: Features detected on the test scenario

textures can be observed). The drone was holding still initially. However, due to the lost track of VO, the drone started to rely on the IMU integration for pose estimation, which caused drifted pose estimation. This explains the drifting of the pose estimation.

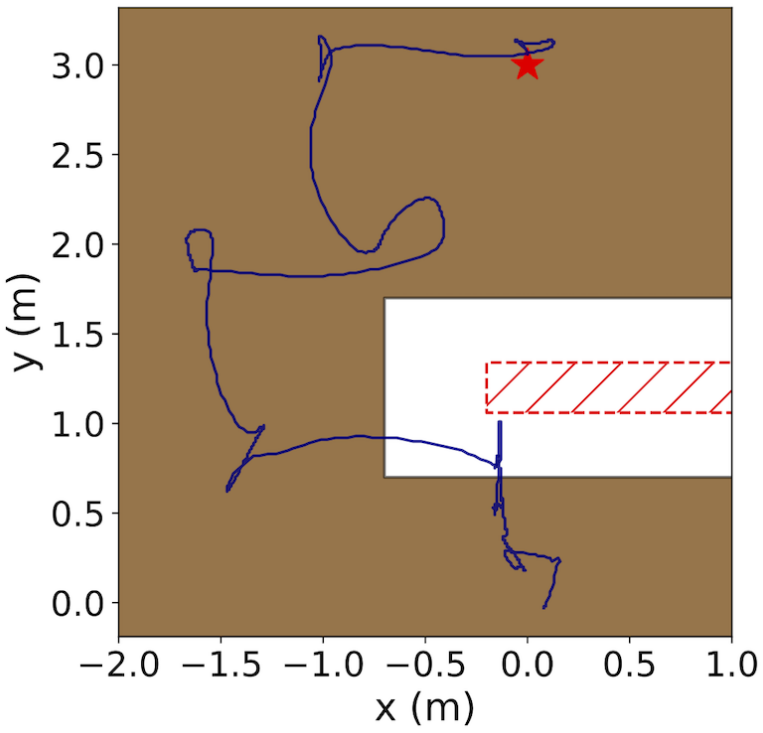


Figure 5.18: Real-world experiment result (red star denotes the goal).

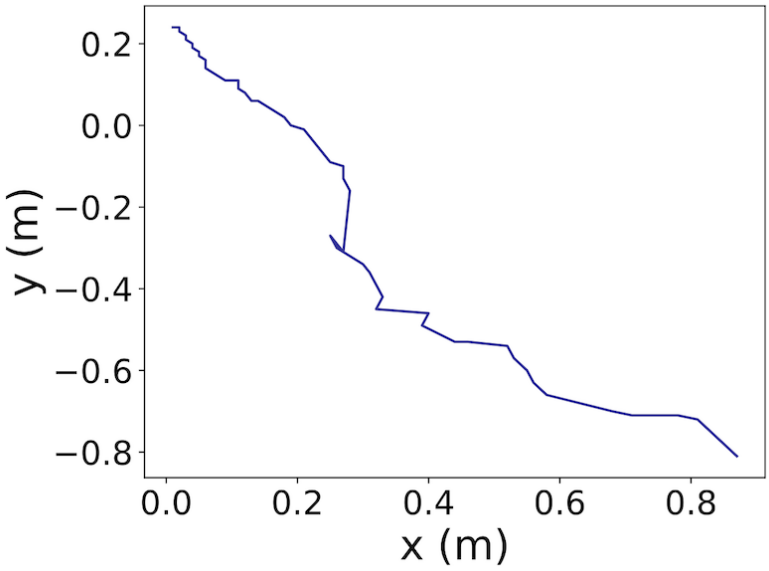


Figure 5.19: Odometry drifted while the drone is stationary with poor visual features.

5.5 Conclusion

This work introduces an RL-based method for drone navigation utilizing visual odometry for localisation, different from other approaches of relying on ground truth poses as presented in the literature.

The desired behaviour for VO-enabled drones should account for the quality of visual features, emphasizing the need to avoid feature-poor regions. To address this, a novel reward space is introduced. Additionally, for improved sim-to-real transfer, the use of semantic images, instead of raw RGB images, is proposed as inputs.

In terms of control, a hierarchical framework is suggested. The high-level policy is trained using the PPO algorithm, while the low-level controller employs classical methods, ensuring safety in theory for reliable motion control.

Evaluation results, both in simulation and real-world environments, demonstrate that the proposed method outperforms baselines, aligning with the hypothesized improvements.

The current action space is discretised and fixed to several 1-meter waypoints manually, which may reduce the manoeuvrability of the drone. Future work could

explore the use of a continuous action space for waypoint representation to achieve a smoother trajectory for the drone.

Chapter 6

Target Tracking for Quadrotors based on Reinforcement Learning

As discussed in Section 1.1.2, cooperation between UAVs and UGVs can enable the accomplishment of more complex tasks, with UAVs tracking UGVs being a fundamental capability for such cooperation.

This chapter introduces a target-tracking control approach for quadrotors, employing the PPO reinforcement learning algorithm. The algorithm enables a quadrotor equipped with an RGBD camera, referred to as 'The Tracker,' to pursue a moving object ('The Target') within a predetermined distance range. The objective is to control the quadrotor to keep the target centred within the camera's field of view while navigating through the environment, ensuring obstacle avoidance throughout the tracking process.

To accomplish this, a novel reward space is devised for the tracking task. Additionally, recognizing the inherent instability challenges of training control policy neural networks from scratch using depth images for obstacle perception, an innovative approach is introduced. This approach advocates for an alternative teaching strategy, where a state-based teacher encodes obstacle information using obstacle positions and their radius relative to the quadrotor, rather than relying solely on

depth images. This state-based teacher policy enables the depth image-based student to effectively acquire tracking and obstacle avoidance capabilities, mitigating the risk of potential local minima.

To facilitate this teacher-student learning process, a variant algorithm based on the PPO algorithm is proposed. This modification is tailored to optimise the learning performance between the teacher and student policies. The proposed control scheme is end-to-end, simplifying the overall system framework compared with non-learning-based conventional trajectory planning methods.

This chapter is organised as follows. Section 6.1 makes an introduction to the topic of this chapter. The proposed method is described in section 6.2, followed by experiments and results in section 6.3. The conclusions are presented in section 6.4.

6.1 Introduction

The application of autonomous aerial tracking finds extensive utility in various domains, including environmental surveillance, security, and aerial photography. Nevertheless, a notable challenge persists in enabling a quadrotor, defined as the tracker, to autonomously track a moving target in unfamiliar and cluttered environments [169]. Ensuring safety demands that the drone accurately perceives both the target and environmental obstacles using onboard sensors and promptly responds to unforeseen obstacles arising from the unknown environment and the limited field of view of onboard sensors. This task is particularly challenging for drones constrained by size, weight, and power (SWaP), which imposes limitations on computing and sensing resources.

The majority of existing research on autonomous aerial tracking focuses on non-learning optimisation-based trajectory planning methods. Typically, these methods decompose the tracking task into various sub-tasks to be executed sequentially, including components like sensing, mapping, planning, and trajectory optimisation [9].

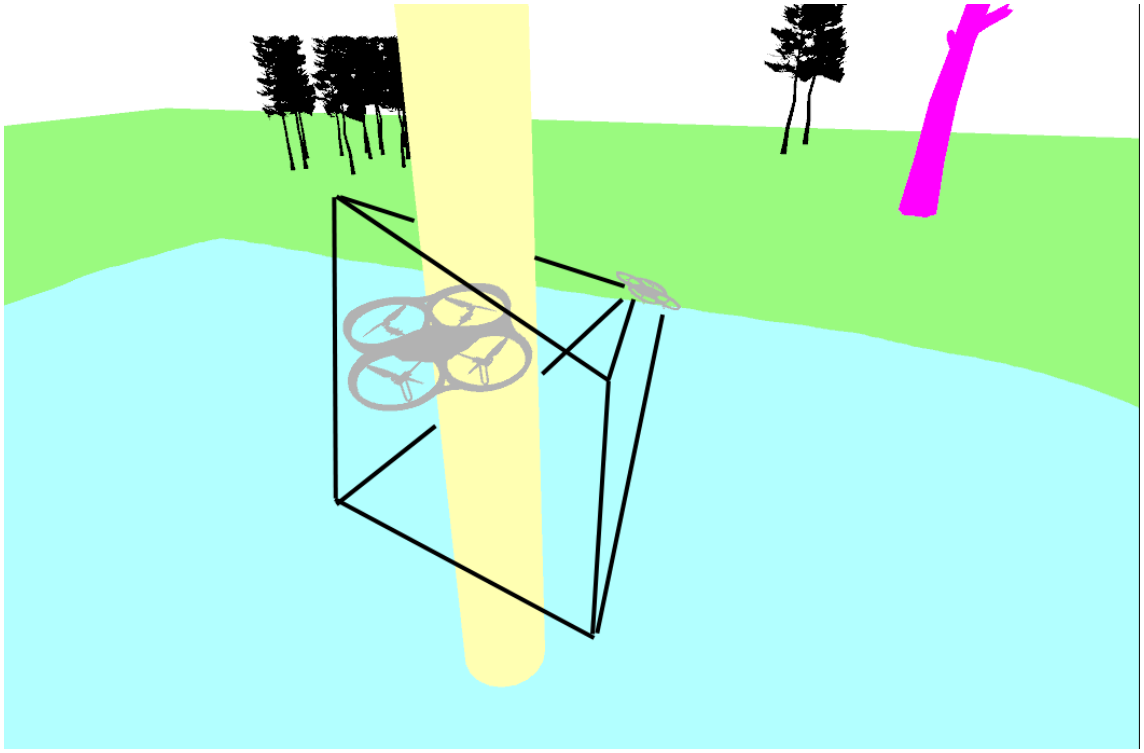


Figure 6.1: A Tracking task example: The tracker follows the target, keeps the target within the field of view, and avoids obstacles in the environment at the same time.

However, this decomposition of the task has the potential to elevate processing latency, involving time for computation and communication between different components. Moreover, it may lead to a complex system that has the potential to compound errors throughout the pipeline [10].

Therefore, the proposal is to leverage the capabilities of deep neural networks trained by reinforcement learning algorithms, which demonstrate proficiency in controlling robots to navigate through complex environments based on observations in an end-to-end way [141]. In an end-to-end control approach, these networks can directly map quadrotor control signals, such as individual rotor thrusts, from observations that include depth images and drone states (velocity, attitude, etc.). This eliminates the need for task decomposition, resulting in a simplified and straightforward system. Fig. 6.2 illustrates the comparison between the conventional non-learning-based system and the proposed learning-based end-to-end system. The structure of the conventional tracking planner is based on previous research work [9]

and [169].

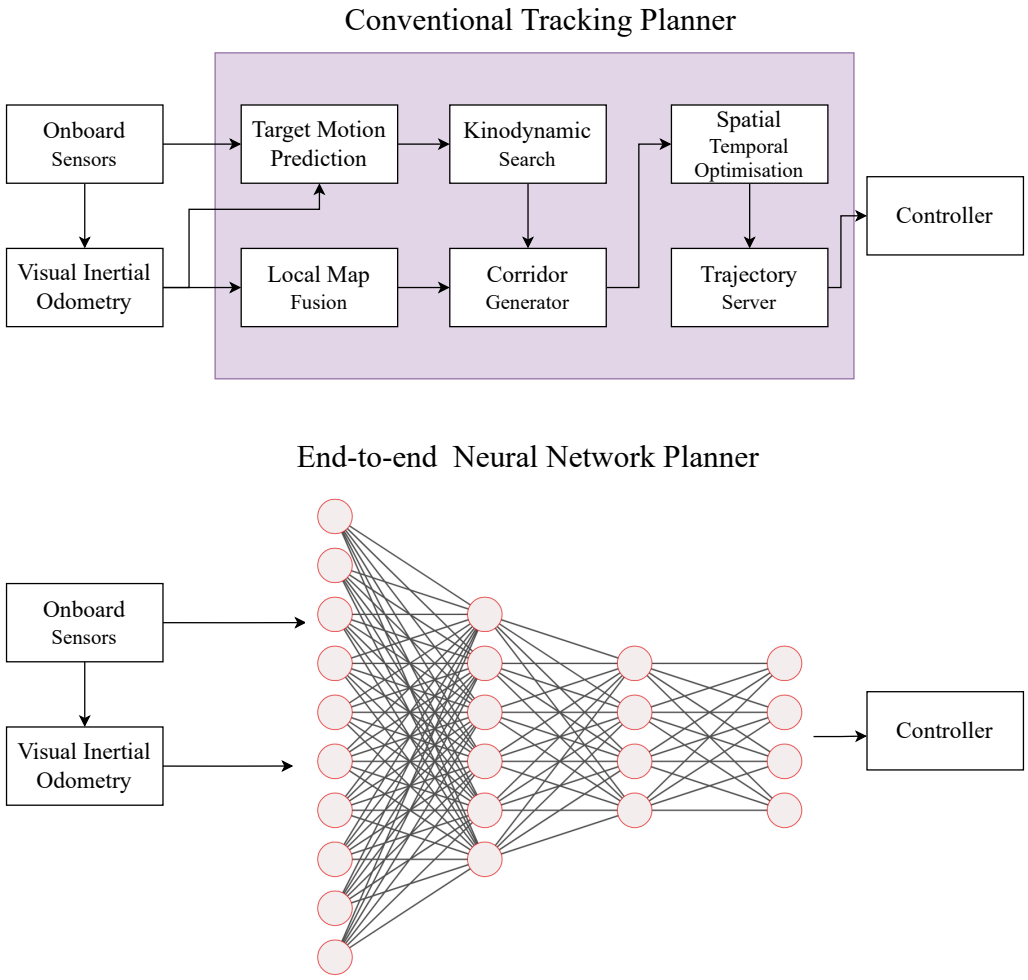


Figure 6.2: The comparison between non-learning-based and learning-based systems

While researchers have extensively employed reinforcement learning algorithms for drone control, the predominant focus has been on optimising drone maneuverability [152] and navigation [132]. Notably, there is a scarcity of research addressing the tracking problem utilizing reinforcement learning algorithms.

Hence, a reinforcement learning-based approach is suggested to address the autonomous tracking problem for the quadrotor. This chapter introduces three key contributions.

Firstly, in alignment with the tracking task requirements, a novel reward space is formulated. This reward space encompasses multiple components crafted to prompt the quadrotor to track the target within a designated safe distance range, navigate

around obstacles, and keep the target positioned close to the field of view (FOV) centre of the onboard camera without occlusions.

Secondly, handling environmental obstacles information presented through depth images poses challenges in training deep neural networks with image inputs from scratch, potentially leading to instability and convergence to local minima. While pre-training an auto-encoder network to encode depth images and obtain a low-dimensional state representation is an option, this approach is time-consuming, involving the collection of image samples and subsequent training of the auto-encoder.

Given that the network is trained in simulation environments where obstacle position information is available, an alternative approach is suggested in this chapter. Specifically, a state-based teacher tracking policy is trained, utilizing scalar obstacle information in terms of relative positions and the radius of surrounding obstacles instead of depth images. Training with such low-dimensional and more effectively presented information proves to be much quicker than training an auto-encoder. Once a satisfactory teacher policy is obtained, it is employed to guide the depth image-based student policy during exploration. The trained vision-based student policy becomes the solution for addressing the proposed tracking problem for quadrotors equipped with camera sensors in unknown environments.

Thirdly, modifications are introduced to the standard PPO training procedure to ensure the proper functioning of the teacher-student learning structure. Within the proposed training framework, a portion of the collected trajectories is produced by the teacher policy. Attempting to train the vision-based student policy with these trajectory examples using the standard PPO policy could lead to training collapse. This issue may arise from the substantial differences between the teacher policy and the student policy, while the PPO algorithm imposes constraints on policy differences between two consecutive updates.

To address this challenge and train the policy effectively, a variant PPO policy is devised, incorporating the importance sampling algorithm. This modification is

instrumental in mitigating the differences between the teacher and student policies during the training process.

6.1.1 Summary and contributions

The overall objective is to control a quadrotor in tracking a ground vehicle target while simultaneously satisfying the following requirements:

1) Visibility: Ensure that the target remains centrally positioned within the tracker camera images. The camera's central axis should be aligned directly with the target, guaranteeing maximum target visibility and accommodating abrupt tracking attitude changes. In scenarios where the quadrotor relies on visual features for estimating the relative pose between the tracker and the target, maintaining the target in the centre of the field of view is crucial.

2) Safe Distance: Maintain a predefined distance from the target to ensure both visibility and safety.

3) Occlusion Avoidance: Prevent obstacle occlusions by ensuring that no obstacles intersect with the line of sight from the tracker to the target. Occlusions have the potential to compromise the quality of relative pose estimation.

4) Collision Avoidance: Avoid collisions with obstacles to ensure the quadrotor's safety and prevent damage.

In summary, the contributions of this chapter can be outlined as follows:

- Introduction of a reinforcement learning-based algorithm, enabling quadrotors to effectively track moving targets in unfamiliar environments.

- Design of a reward space, specifically tailored to enhance the execution of the target tracking task.

- Implementation of a teacher-student learning scheme during the training of the control network.

- Reconstruction of the standard PPO algorithm using the importance sampling technique to ensure the convergence of the teacher-student framework toward a satisfactory result.

6.2 Method

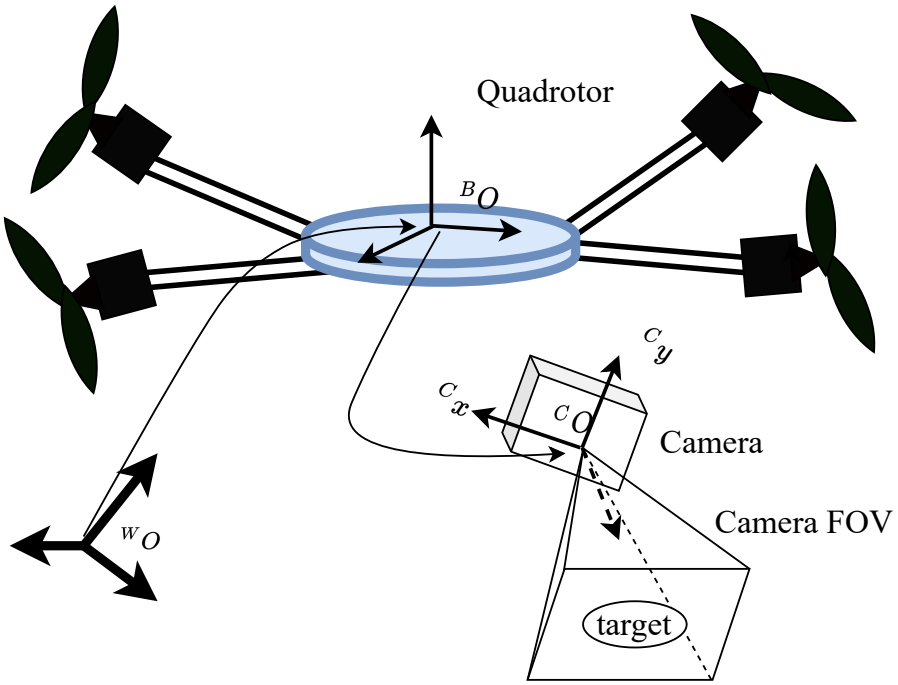


Figure 6.3: Example of a tracking task: The tracker's objective is to pursue the target, ensuring that the target remains within the field of view, while concurrently navigating through the environment to avoid obstacles.

The detailed requirements for the tracking task have been outlined in Section 6.1, covering crucial aspects such as visibility, safe distance, occlusion avoidance, and collision avoidance. For the sake of simplifying the problem, it is assumed that obstacles in the environment take the form of spheres.

The problem can be formulated as follows:

$$\min_{\pi(x)} \mathcal{J}_o = \int_0^T \|^C p_{target}(x, y, t)\|^2 \mathrm{d}t, \tag{6.1a}$$

$$s.t. \quad x_{t+1} = f(x_t, \pi(x_t)), \tag{6.1b}$$

$$x_0 \in \mathcal{X}, \tag{6.1c}$$

$$p(t) \in \mathcal{P}, \ \forall t \in [0, T], \tag{6.1d}$$

$$p(t) \in \mathcal{V}_t, \ \forall t \in [0, T], \tag{6.1e}$$

$$d_l \leq \|p(t) - p_{target}(t)\| \leq d_u, \ \forall t \in [0, T], \tag{6.1f}$$

where, x represents the quadrotor states, such as position p_t and attitude q. The notation ${}^C p_{\text{target}}(x, y, t)$ denotes the x, y position of the target in the quadrotor's camera frame at time step t. For clarity, unless specified, representations are under the world frame.

Visibility requirement can be reconfigured as Eq. 6.1a, which situates the target at the centre of the $x - y$ plane of the camera frame (refer to Fig. 6.3 for details). Eq. 6.1b represents the dynamics of the quadrotor, which is discussed in detail in Eq. 5.1. Here, $\mathcal{P}$ denotes the safe area that ensures obstacle collision-free movement for the quadrotor, aligning with the collision-free requirement. $\mathcal{V}_t$ describes the occlusion-free area at time step t. The lower and upper distance boundaries, d_l and d_u, respectively, are defined to satisfy the distance constraint, as indicated in Eq. 6.1f.

6.2.1 Teacher-student RL algorithm

This section will describe the development of a tracking controller through the adaptation of the PPO algorithm within the framework of reinforcement learning. The subsequent sections will first recap the foundational understanding of reinforcement learning and a detailed overview of the PPO algorithm. Following this, the imple-

mentation details tailored to address the specific tracking problem will be discussed.

Basic Concepts. General Reinforcement learning algorithms solve problems under the Markov Decision Process (MDP) framework. Physical processes are described by a state transition model $p(s_{t+1}|s_t, a_t)$ where states s fall in a state space S and actions are in action A. A reward function $R_t(s_t, a_t)$: $S \times A \to R$ is carefully designed to justify actions a_t taken at state s_t according to specific tasks. Strategies for solving tasks can be obtained by maximizing overall discounted rewards $R(\tau) = \sum_{k=0}^{\infty} \gamma^k r_t$ received during the whole process. State-action value functions $Q^{\pi(s_t,a_t)} = E_{\tau\sim\pi}[R(\tau)|s_t, a_t]$ and/or state value function $V^{\pi}(s_t) = E\tau \sim \pi[R(\tau)|s_t]$ are usually defined to reshape the objective function, where π: $a \sim \pi(\cdot|s, \theta)$ established by parameters θ represents the policy to be optimised. Also, advantage functions $A^{\pi}(s, a) = Q^{\pi}(s, a) - V^{\pi}(s)$ are used to describe how much better on average an action is than others.

PPO (Proximal Policy Optimisation). PPO is a policy gradient algorithm that directly trains the policy at each update step. It is recognized that the algorithm may diverge if the newly updated policy deviates excessively from the previous one. To mitigate this, PPO is designed to constrain the update, ensuring that two consecutive policies remain close to each other.

For the method discussed in this chapter, PPO-Clip is utilized, incorporating a designed clipping behaviour in objective functions as opposed to relying on KL-divergence. This choice is made to ensure that the updated policy remains in proximity to the previous one. The update of PPO-Clip policies follows a specific process:

$$\theta_{k+1} = \arg\max_{\theta} \mathop{\mathrm{E}}_{s,a\sim\pi_{\theta_k}} [L(s, a, \theta_k, \theta)] \tag{6.2}$$

were L is given by

$$L(s, a, \theta_k, \theta) = \min \left(\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)} A^{\pi_{\theta_k}}(s, a), g(\epsilon, A^{\pi_{\theta_k}}(s, a)) \right) \tag{6.3}$$

in which ϵ is a small hyperparameter and $g(\epsilon, A) = \begin{cases} (1+\epsilon)A & A \geq 0 \\ (1-\epsilon)A & A < 0. \end{cases}$

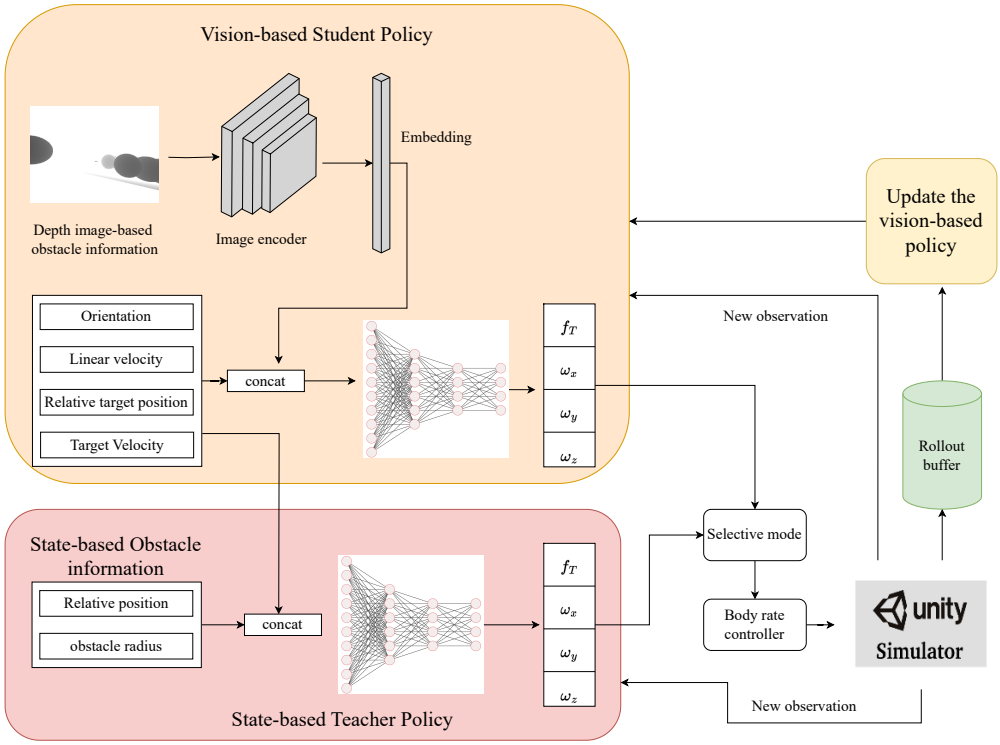


Figure 6.4: Taining the tracking policy follows a teacher-student learning pipeline: A state-based agent as the expert is pre-trained to guide exploration during the vision-based agent training procedure. At each time step, the action commands are either from the state-based expert or from the to-be-trained vision policy by a selection module. This teacher-student learning can help improve exploration efficiency and escape local minima

State-based Teacher PPO Policy The first phase of the method starts with the training of a state-based teacher, equipped with ground truth obstacle information, including relative positions and sizes of obstacles, as well as other quadrotor and target states information, as illustrated in Fig. 6.4. This teacher serves as an expert to guide the training of the vision-based student policy. This approach is motivated by the time-consuming nature of training neural networks with image inputs, which may also converge to local minima. The state-based teacher policy simplifies the training of neural networks, eliminating the need for image inputs. Additionally, it significantly reduces the overall training time by eliminating the necessity for rendering depth images and by utilizing relatively small neural networks

that do not involve CNN networks for image processing. This reduction in training time facilitates the efficient exploration of various hyperparameters, shared with vision-based policy training, such as the scale of reward values.

Hence, training a state-based teacher policy is reasonable. The training of the teacher policy follows the standard PPO policy training procedure, and the neural network parameter for the teacher policy is nominated as ξ.

Vision-based Student PPO Policy Update. The training procedure for the vision-based student policy is illustrated in Fig. 6.4. During the phase of collecting training samples, both the teacher and student policies receive current observations from the environment and make action decisions based on shared and individual observation components. The selective module chooses an action from the teacher policy with a probability of ε. Otherwise, it selects actions from the current student policy. The action consists of body rates $[w_x, w_y, w_z]$ and a collective thrust f_T command, which is then fed to a body rate controller to control the quadrotor in the environment. The environment generates a transition $[s_t, a_t, s_{t+1}, r_t]$, which is stored in the rollout buffer. Once a certain number of transitions are collected, the algorithm updates the student policy by randomly sampling transition data from the rollout buffer.

In the training process, all training samples are treated as if they originate from the teacher's policy and employ importance sampling to obtain a non-biased estimation of the cost function.

$$\begin{aligned} \underset{s,a\sim\pi_{\theta_k}}{\mathrm{E}}[L] &= \int_S \int_{\mathcal{A}} \rho^{\pi_{\theta_k}} \pi_{\theta_k} L da ds = \int_{\mathcal{S}} \int_{\mathcal{A}} \rho^{\pi_{\theta_k}} \pi_\xi \frac{\pi_{\theta_k}}{\pi_\xi} L da ds \\ &= \underset{s\sim\rho^{\pi_{\theta_k}},a\sim\pi_\xi}{\mathrm{E}} \left[\frac{\pi_{\theta_k}}{\pi_\xi} L \right] \end{aligned} \tag{6.4}$$

Similar to [214] and [215], the state density ratio is ignored as if the states come

from the behaviour policy π_{θ_ξ}. Hence Eq. 6.3 can be approximated by:

$$\begin{aligned}\theta_{k+1} &= \arg\max_{\theta} \mathop{\mathrm{E}}_{s,a\sim\pi_{\theta_k}}[L] = \arg\max_{\theta} \mathop{\mathrm{E}}_{s\sim\rho^{\pi_{\theta_k}},a\sim\pi_\xi}\left[\frac{\pi_{\theta_k}}{\pi_\xi}L\right] \\ &\approx \arg\max_{\theta} \mathop{\mathrm{E}}_{s,a\sim\pi_\xi}\left[\frac{\pi_{\theta_k}}{\pi_\xi}L\right]\end{aligned} \tag{6.5}$$

Algorithm 3 Teaching-based Proximal Policy Optimisation

Input: teacher policy π_ξ, initial student policy parameters θ_0, initial value function parameters ϕ_0
for $k = 0, 1, 2, ...$ **do**
  Initialize rollout buffer D
  for step $= 1, ..., m$ **do**
    With probability ε sample an action from $\pi(s, \xi)$.
    otherwise sample from $\pi_{(s,\theta_k)}$ as proposed in this chapter.
    Execute action a_t in the environment and receive reward r_t and new state s_{t+1}.
    Store transition (s_t, a_t, r_t, s_{t+1})
  end for
  Compute rewards-to-go $R(t)$
  Compute advantage estimates $\hat{A}_t$ based on the teacher value function V_ξ
  Update the policy by maximizing the proposed objective:

$$\theta_{k+1} = \arg\max_{\theta} \frac{1}{|\mathcal{D}|T}\sum_{\forall\tau\in\mathcal{D}}\sum_{t=0}^{T}\frac{\pi_{\theta_k}}{\pi_\xi}L \tag{6.6}$$

  via stochastic gradient ascent with Adam
  Fit value function by regression on mean-square error:

$$\phi_{k+1} = \arg\min_{\phi} \frac{1}{|\mathcal{D}_k|T}\sum_{\forall\tau\in\mathcal{D}}\sum_{t=0}^{T}(V_\phi(s_t) - R(t))^2 \tag{6.7}$$

  via gradient descent algorithm
end for

6.2.2 RL-based tracking algorithm implementations

The implementation details of the RL-based algorithm proposed in this paper will be discussed in the following sections.

The target is modelled as a 2D ground vehicle with the capability to move

freely along the x and y axes. The vehicle's motion is controlled by adjusting the acceleration along the respective axes. The altitude z of the target is fixed at $0m$ and the ground is situated at $-5m$. The dynamics of the target are described as follows:

$$\begin{bmatrix} \boldsymbol{x}(t) \\ \boldsymbol{y}(t) \\ \boldsymbol{v_x}(t) \\ \boldsymbol{v_y}(t) \end{bmatrix} = \begin{bmatrix} \boldsymbol{v_x}(t-1) * \Delta T + \frac{1}{2}\boldsymbol{a_x}\Delta T^2 \\ \boldsymbol{v_y}(t-1) * \Delta T + \frac{1}{2}\boldsymbol{a_y}\Delta T^2 \\ \boldsymbol{v_x}(t-1) + \boldsymbol{a_x}\Delta T \\ \boldsymbol{v_y}(t-1) + \boldsymbol{a_y}\Delta T \end{bmatrix} \tag{6.8}$$

The target is capable of navigating without colliding with obstacles in the environment. The intentional decoupling of movements along the x and y axes of the target enhances its agility, introducing complexity to the tracking task. This introduced challenge contributes to training a quadrotor with the capability to handle general tracking tasks.

State Space For the teacher policy, the state consists of three key information components: the environmental obstacles, the quadrotor state, and the target state. Thus, the state vector is $s = [p_{\text{obstacle}} \times n, r_{\text{obstacle}} \times n, v_{\text{tracker}}, q_{\text{tracker}}, p_{\text{target}}, v_{\text{target}}]$, comprising the relative positions p_{obstacle} and radii r_{obstacle} of the n closest obstacles, the states of the quadrotor, including linear velocity v_{tracker} and attitude q_{tracker}, and the states of the target, encompassing relative position p_{target} and linear velocity v_{target}.

In practical scenarios, obtaining low-dimensional state-based obstacle information $[p_{\text{obstacle}} \times n, r_{\text{obstacle}} \times n]$ directly may not be feasible. Instead, depth images O_{depth} are provided to capture environmental obstacle information eliminating the availability of p_{obstacle} and r_{obstacle}. Therefore, the state for the vision-based student policy is defined as $s = [O_{\text{depth}}, v_{\text{tracker}}, q_{\text{tracker}}, p_{\text{target}}, v_{\text{target}}]$.

Action Space Design Both the teacher policy and the student policy op-

erate within the same action space, utilizing body-rate signal denoted as $a = [f_T, w_x, w_y, w_z]$. This control scheme, as proposed in [152], enables agile flying. The specified action command is executed by a body-rate controller, which calculates individual propeller thrusts to control the quadrotor. It is worth noting that using low-level control signals as the action space, such as individual propeller thrusts, might pose safety concerns and encounter challenges in sim-to-real applications, as indicated in [151].

Reward Space Design The reward function is designed to encourage the quadrotor to track the 2D target while adhering to the constraints outlined in Eq. 6.1. The reward function is constructed with multiple components, each serving a distinct purpose aligned with the requirements of visibility, safe distance, occlusion avoidance, and collision avoidance. These components are described below:

$$R = R_{\text{visibility}} + R_{\text{safe\_distance}} + R_{\text{occlusion}} + R_{\text{collision}} \tag{6.9}$$

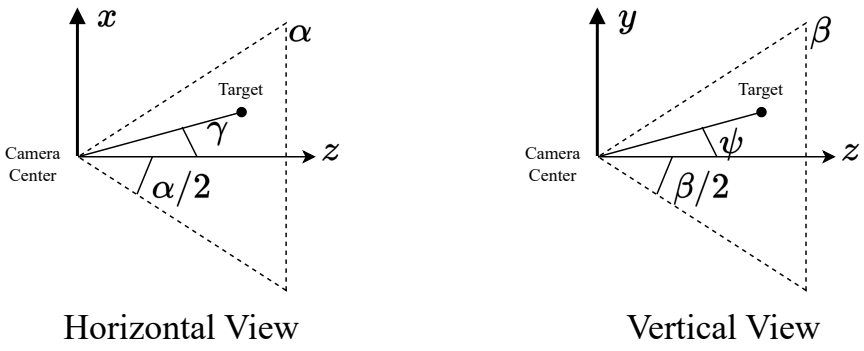


Figure 6.5: Relative angles between the target and the camera in horizontal and vertical views. z is the camera looking direction

The visibility reward component $R_{\text{visibility}}$ is to punish the angle of the target to the camera centre vector in the camera frame deviating away from the camera z axis. Fig. 6.5 describes the relative angles where α and β are the horizontal and vertical FOV constraints respectively. w is the weight factor.

$$R_{\text{visibility}} = w(R_{\text{horizontal}} + R_{\text{vertical}}) \tag{6.10}$$

The horizontal view is controlled by the quadrotor yaw which can be decoupled from the control of the target pose. Thus it can be constructed as a strict constraint :

$$R_{\text{horizontal}} = \boldsymbol{-0.2(e^{|\gamma|-\alpha/2} - e^{-\alpha/2})} \tag{6.11}$$

The vertical angle deviation cannot always be guaranteed to be zero, as it is controlled by the pitch of the quadrotor. Therefore, the agent pitch behaviour is penalized only when the relative vertical angle exceeds half of the vertical field of view, leading to the target disappearing from the image:

$$R_{\text{vertical}} = \begin{cases} \boldsymbol{-0.2e^{|\psi|-\beta/2}} & \boldsymbol{\psi \geq \beta/2} \\ 0 & otherwise \end{cases} \tag{6.12}$$

The second reward component $R_{\text{safe\_distance}}$ is to encourage a reasonable distance between the quadrotor and the target:

$$R_{\text{safe\_distance}} = \begin{cases} 0 & d_l \leq \|p_{tracker} - p_{target}\| \leq d_u \\ -10 & otherwise \end{cases} \tag{6.13}$$

$R_{\text{collision}}$ is activated when the distance between the quadrotor and an obstacle is smaller than the sum of the obstacle radius and the quadrotor radius. When a collision happens, the episode will terminate.

$$R_{\text{collision}} = \begin{cases} 0 & d^i_{obstacle} > r^i + r_{quadrotor} | i = 0, ..., N \\ -10 & otherwise \end{cases} \tag{6.14}$$

The reward term $R_{\text{occlusion}}$ is configured to prevent occlusion, a situation arising when obstacles intersect with the line of sight connecting the tracker and the target. This is ensured by necessitating that the distance between the obstacle and the line of sight is greater than the obstacle radius. Alternatively, the condition is met

if the closest point from the line of sight to the obstacle centre is located behind the camera. To enhance robustness in tracking occlusion avoidance, the obstacle occlusion radius is increased from r_1 to $r_{inflate}$, as illustrated in Fig. 6.6.

$$R_{\text{occlusion}} = \begin{cases} 0 & d^i > r_{inflate} \; or \; \overrightarrow{Op_t}\overrightarrow{Op_{occ,i}} < 0 | i = 0, ..., N \\ -1 & otherwise \end{cases} \tag{6.15}$$

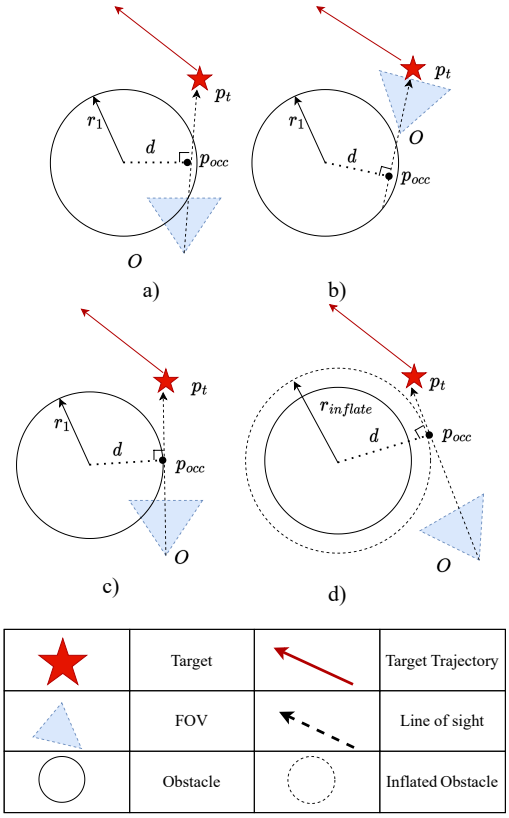
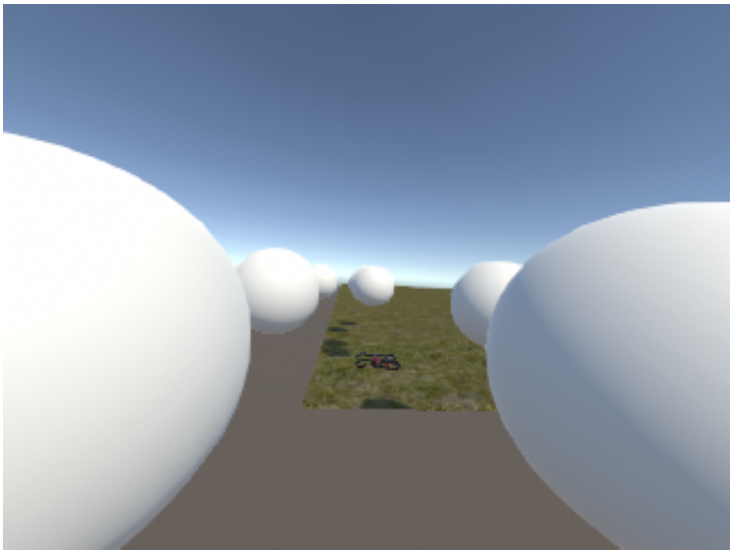


Figure 6.6: Occlusion description

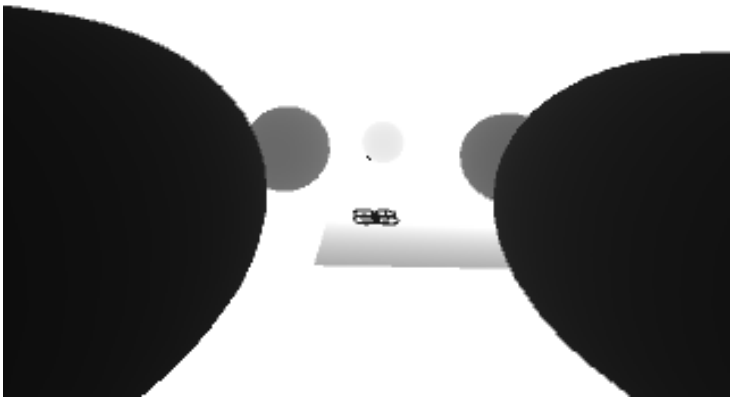
6.3 Evaluation

6.3.1 Experiments setup

The algorithm is evaluated in simulated environments using a quadrotor simulator named Flightmare [195]. Flightmare is built on the Unity game engine, which handles both physical dynamics simulation and image rendering. Examples of ob-



(a) RGB image



(b) Depth observation

Figure 6.7: Observation example

servations are depicted in Fig. 6.7.

State-based RL Agent (teacher) Initially, the algorithm firstly trains an end-to-end state-based agent using a standard PPO algorithm, which subsequently serves as the teacher policy to guide the vision-based agent. Given that the observation $s = [p_{\text{obstacle}} \times n, r_{\text{obstacle}} \times n, v_{\text{tracker}}, q_{\text{tracker}}, p_{\text{target}}, v_{\text{target}}]$ at each time step is a vector, both the policy network and the value network can be simply constructed using multiple fully connected (FC) layers.

Vision-based RL Agent As per the provided instructions, depth images are

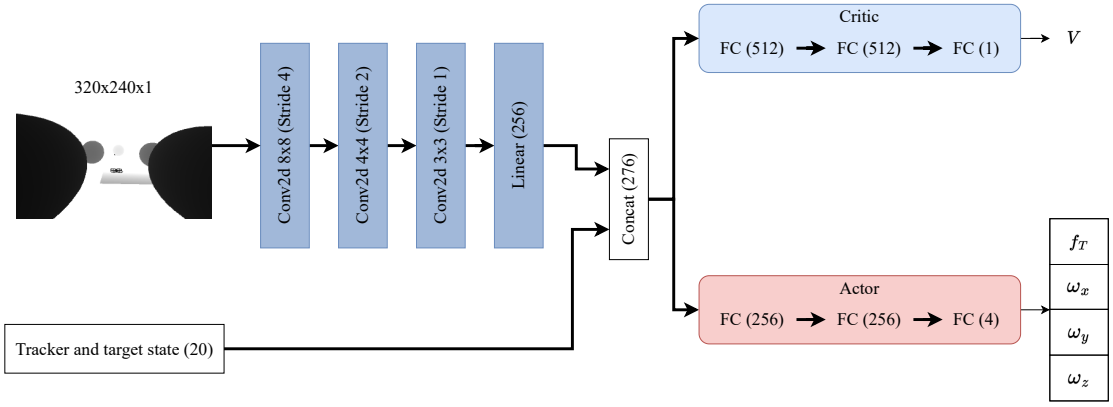


Figure 6.8: Vision-based value and action networks

employed to convey environmental obstacle information for the vision-based agent. The network architecture for the vision agent is illustrated in Fig. 6.8. The depth image, containing high-dimensional information, undergoes processing through the feature extractor module. This module is constructed with three Convolutional Neural Network (CNN) layers and a linear Fully Connected (FC) layer to produce a low-dimensional latent feature vector. Notably, the feature extractor is shared between both the value (critic) and the policy (actor) networks. Subsequently, the latent feature vector is concatenated with the additional information $[v_{\text{tracker}}, q_{\text{tracker}}, p_{\text{target}}, v_{\text{target}}]$, serving as inputs for the remaining FC layers in both the value network and the policy network.

An environment example can be seen from a top-down figure (Fig. 6.9). To facilitate the training of PPO agents, each episode starts spawning the target at a fixed position, and the target intends to navigate to a randomly assigned goal position. The quadrotor is initially spawned $2m$ away from the target at the episode's beginning. At each time step, the agent generates a body-rate control command based on received observations, aiming to control the quadrotor to follow the target and adhere to visibility, safety, occlusion, and collision constraints as previously outlined. The episode is terminated and restarted when any of the following situations occur: 1) the target reaches the goal; 2) the quadrotor collides with obstacles; 3) the quadrotor is too close or too far from the target; 4) the altitude of the quadrotor

is too high or too low; or 5) the maximum time steps of one episode are reached. Obstacles are randomly distributed in a 2D plane at the same altitude as the target.

During the training of the vision-based agent, It is observed that the optimisation process tends to converge to a local minimum when employing the standard PPO algorithm to train the vision-based agent. To address this challenge, The algorithm advocates for a teacher-student PPO training procedure, as discussed in Section 6.2. For ease of reference, the vision agent trained with the standard PPO algorithm is labelled as **Vision-standard**, while the agent trained using the proposed algorithm is denoted as **Vision-student**.

Table 6.1: State-based agent training progress

Training states	Stage 1	Stage 2	Stage 3
Training time steps	7.5×10^5	2.25×10^6	5×10^7
Average episode length	41.34	306.90	473.80
Average reward	-11.47	-14.89	-1.27

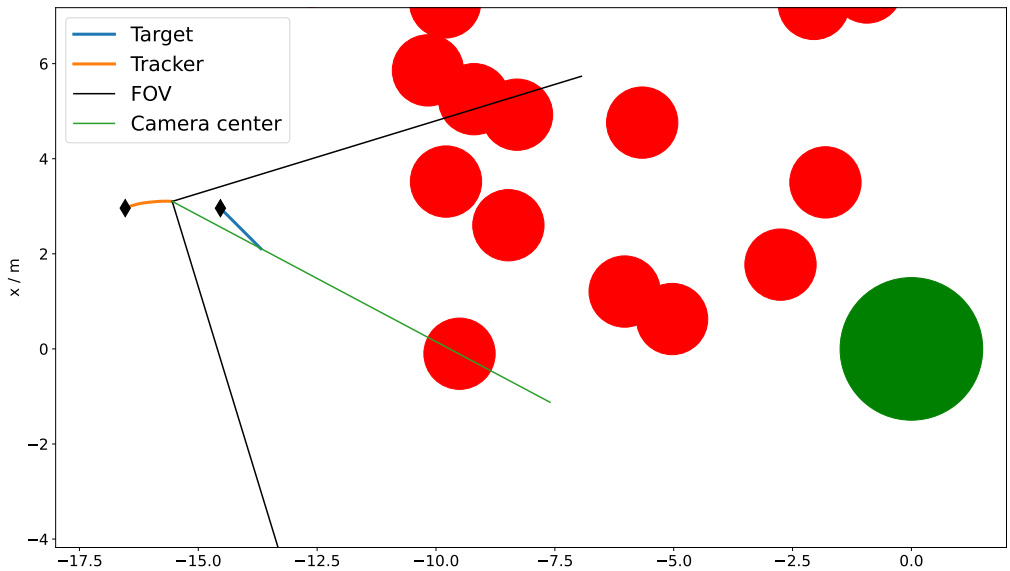


Figure 6.9: An environment example: red circles are the obstacles and the green circle is the navigation destination of the target. Black thin diamonds are the starting points for the target and the quadrotor.

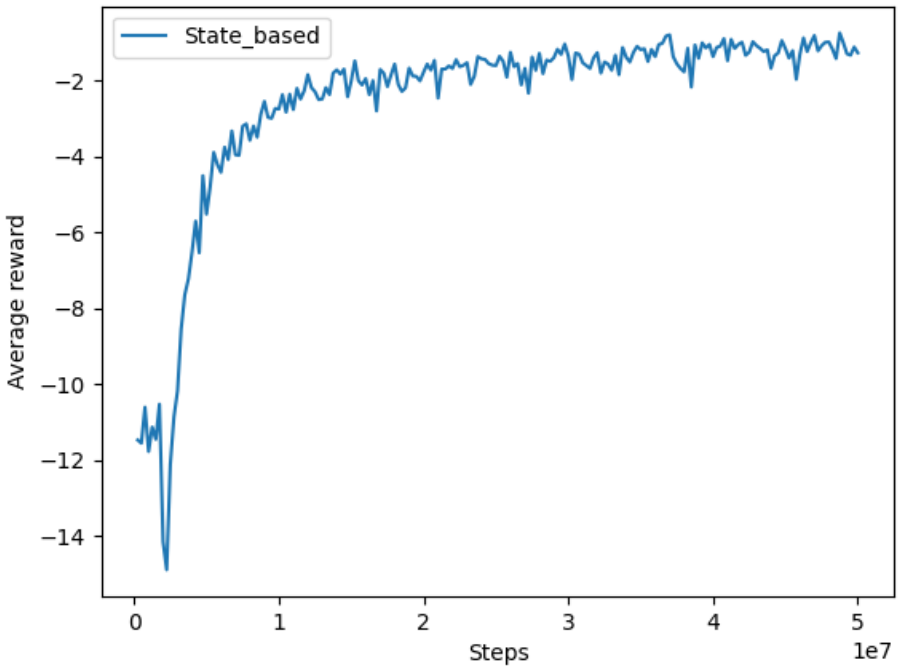


Figure 6.10: Average training reward of state-based agent

6.3.2 PPO training results

The results will be analysed according to three different agents: state-based, Vision-standard, and Vision-student.

State-based agent Fig. 6.10 illustrates the training results of the state-based agent's tracking policy, while Table 6.1 details the training progress across three distinct stages. In the initial stage (Stage 1 in Table 6.1), the average reward is around -11 since the agent does not know how to control the quadrotor at the very beginning. The episode will be terminated quickly as a result of violations of the safe distance or collision avoidance constraints. Under both situations, the agent will be punished with -10 rewards. The episode terminates shortly after an average of 60-time steps. The visibility and occlusion punishment will not accumulate to a noticeable scale in such short episodes. As the training time steps increase, the average reward decreases to -14 (Stage 2 in Table 6.1). The reason is that the agent starts learning to roughly control the quadrotor. The quadrotor stays alive

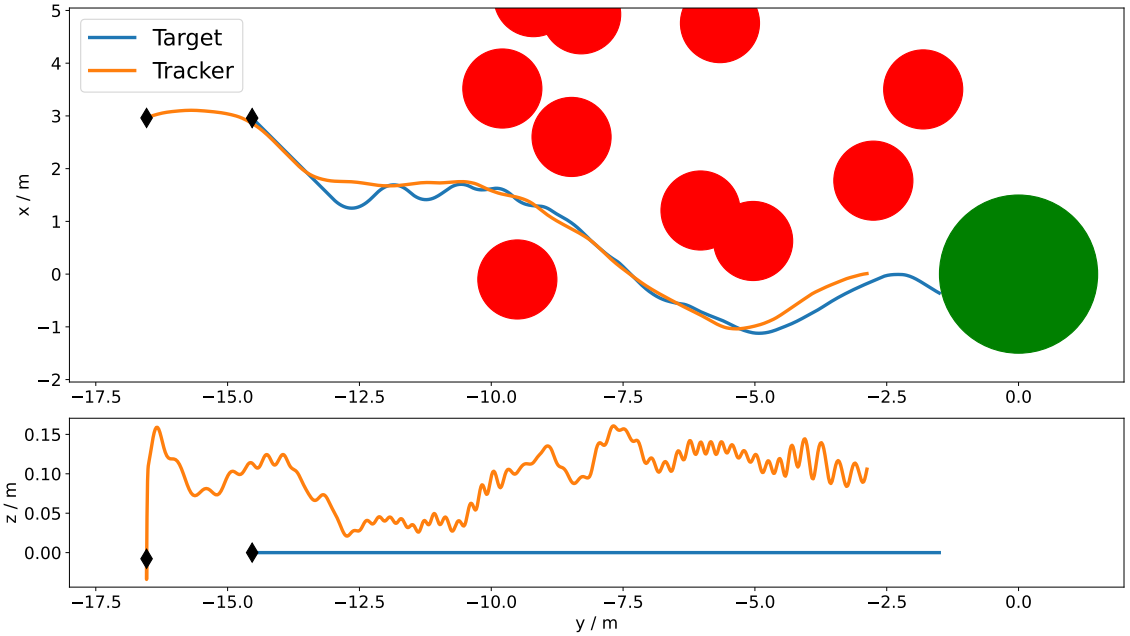


Figure 6.11: An tracking example of the State-based agent after training: the quadrotor can follow the target closely.

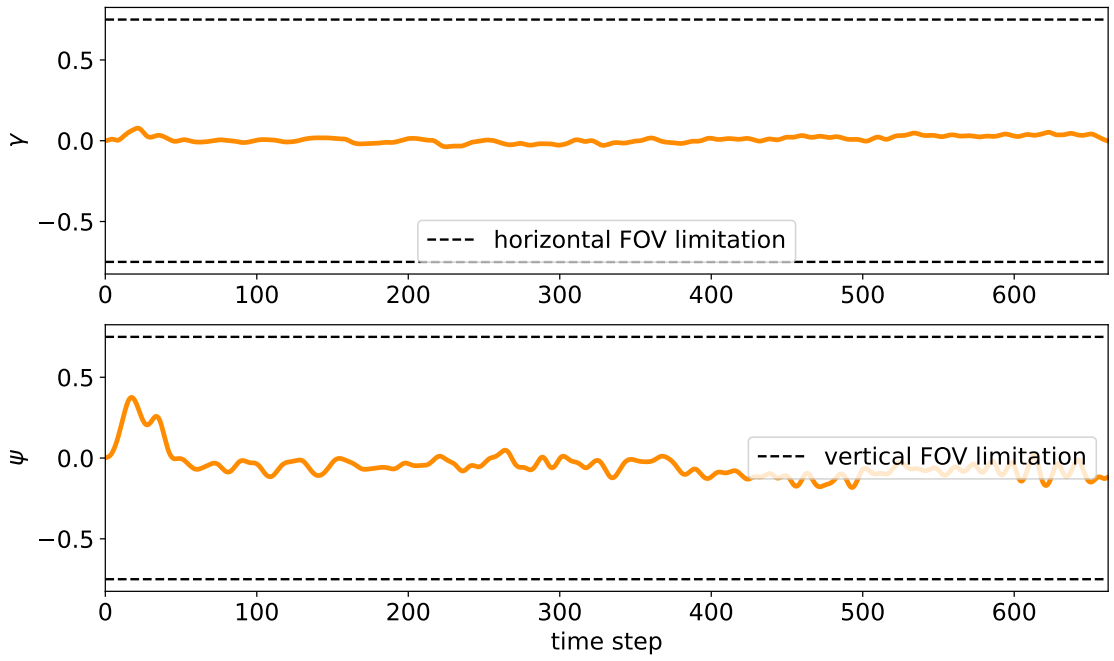


Figure 6.12: Relative angles between the target and the camera-looking direction: both angles are around zero, indicating the target is located in the image centre. (State-based)

a bit longer. In addition to the safe distance and collision avoidance punishment, the visibility punishment accumulates as the average duration of the episode reaches around 300 time steps. After a long period of training 5×10^7 time steps (Stage

3 in Table 6.1), the average reward converges to −1.5, this indicates a very good tracking performance where nearly all constraints are consistently met. This can be observed from Fig. 6.11 and Fig. 6.12.

Vision-standard agent The training results of Vision-standard agents are depicted in Fig. 6.13. As it shows, the average reward of the Vision-standard agent remains around −10 without change after a period of training. This suggests that the training converges to a local minimum which will be explained in detail in the subsequent analysis.

Table 6.2: Vision-standard agent training progress

Training stages	Stage 1	Stage 2	Stage 3
Training time steps	0.425×10^6	0.7×10^6	2×10^6
Average episode length	37.5	128.04	16.84
Average reward	−12.25	−18.53	−10.05

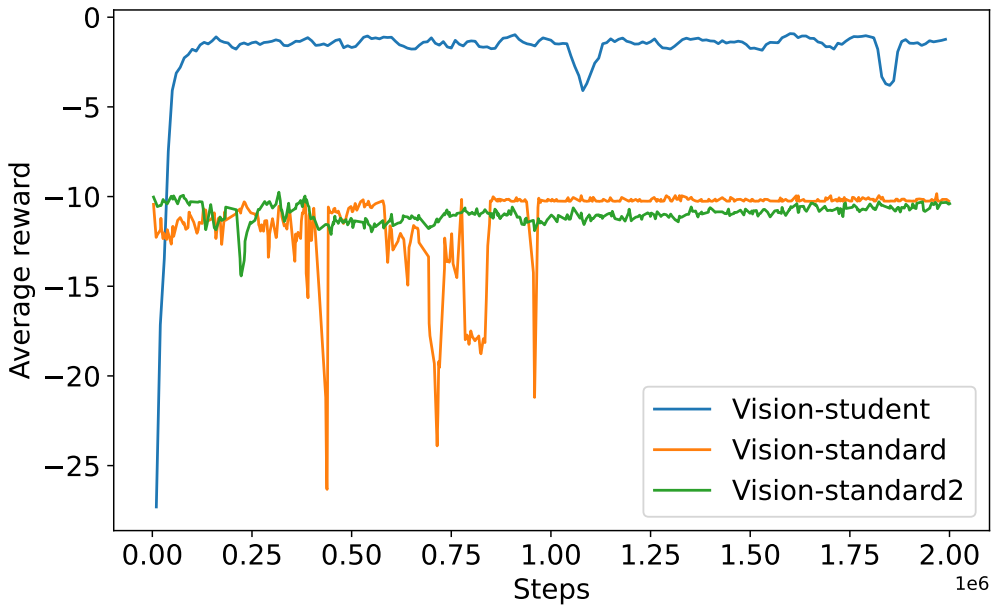


Figure 6.13: Average training reward of vision-based agents

At the early training stage (Stage 1 in Table 6.2), the Vision-standard agent is only capable of stabilizing the quadrotor in mid-air, attempting to follow the target at a slow speed. However, the target moves quickly toward its goal position,

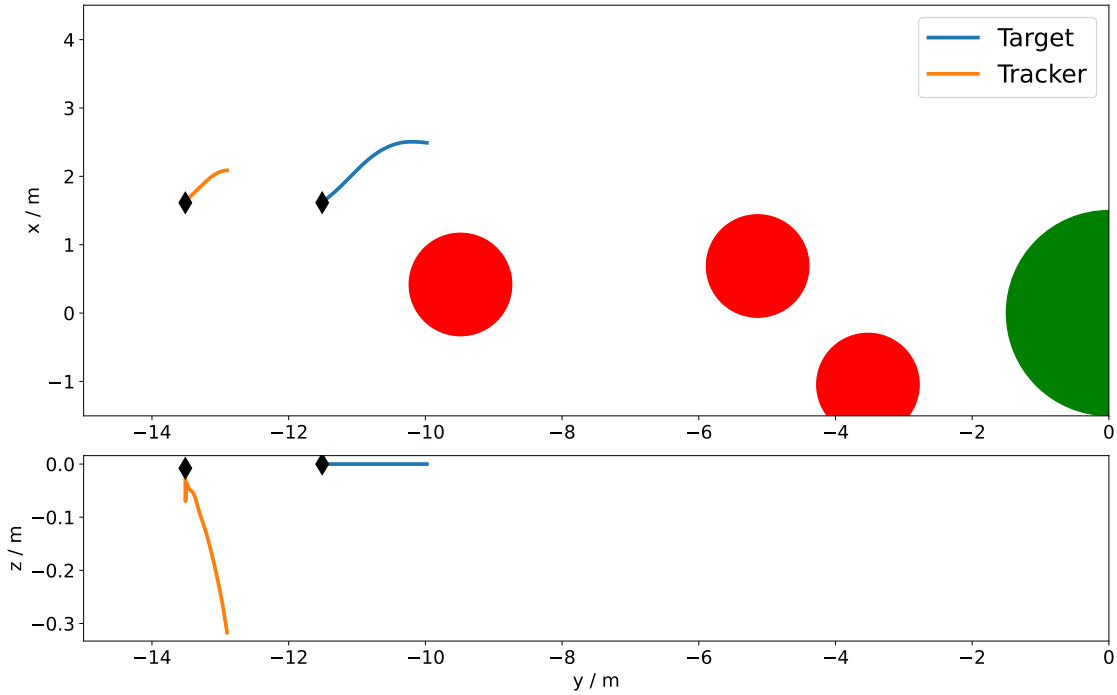


Figure 6.14: An tracking example of the Vision-standard agent at Stage 1: The trajectory is short as the target escapes the tracking range rapidly.

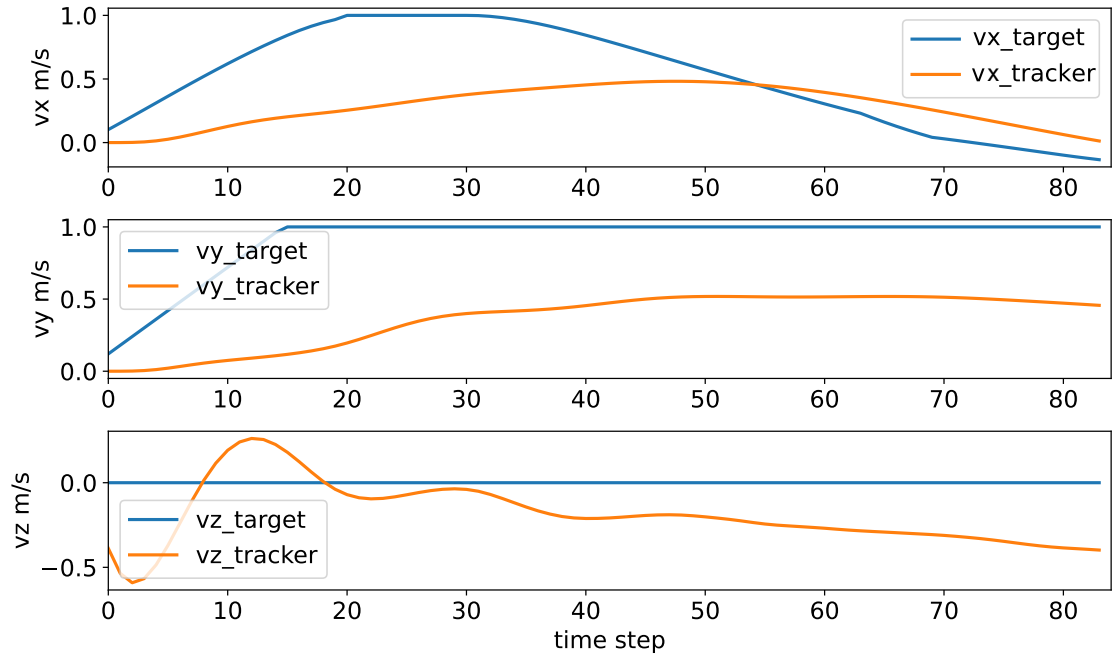


Figure 6.15: Velocity comparison of the quadrotor and the target at Stage 1: the quadrotor can not match the velocity of the target. (Vision-standard)

causing the distance between the quadrotor and the target to surpass the defined upper bound rapidly. Consequently, the episode length becomes very short. An

illustrative example of the tracking trajectory is presented in Fig. 6.14, while the velocity comparison between the quadrotor and the target is depicted in Fig. 6.15. The figure indicates that, during this early training stage, the agent moves at a relatively low speed compared to the target's velocity. This behaviour may stem from the agent's limited proficiency in controlling the quadrotor's movement, primarily focusing on stabilization at its initial position.

As the agent receives punishment when the target exceeds the tracking range, there is a motivation for the agent to enhance control over the quadrotor's movement, resulting in increased speed during the next learning stage ((Stage 2 in Table 6.2)). An illustrative trajectory is depicted in Fig. 6.16.

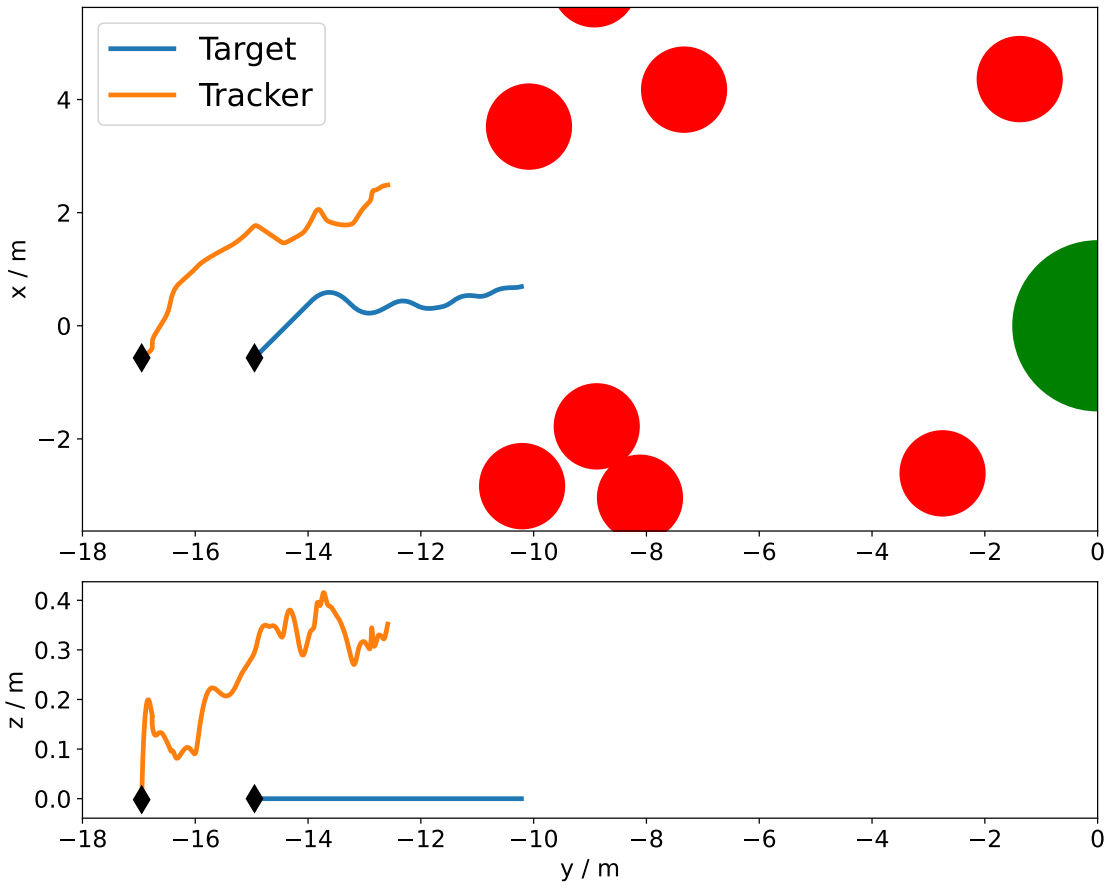


Figure 6.16: An tracking example of the Vision-standard agent at Stage 2: The quadrotor can follow the target to some extent.

During this stage, the quadrotor exhibits erratic behaviour, reflecting the agent's

lack of proficiency in controlling the high-speed movements of the quadrotor. The velocity comparison of agents in the trajectory depicted in Fig. 6.16 is illustrated in Fig. 6.17. The figure shows that the quadrotor velocity fluctuates around the target velocity, indicating the agent's efforts to learn and match the target's speed.

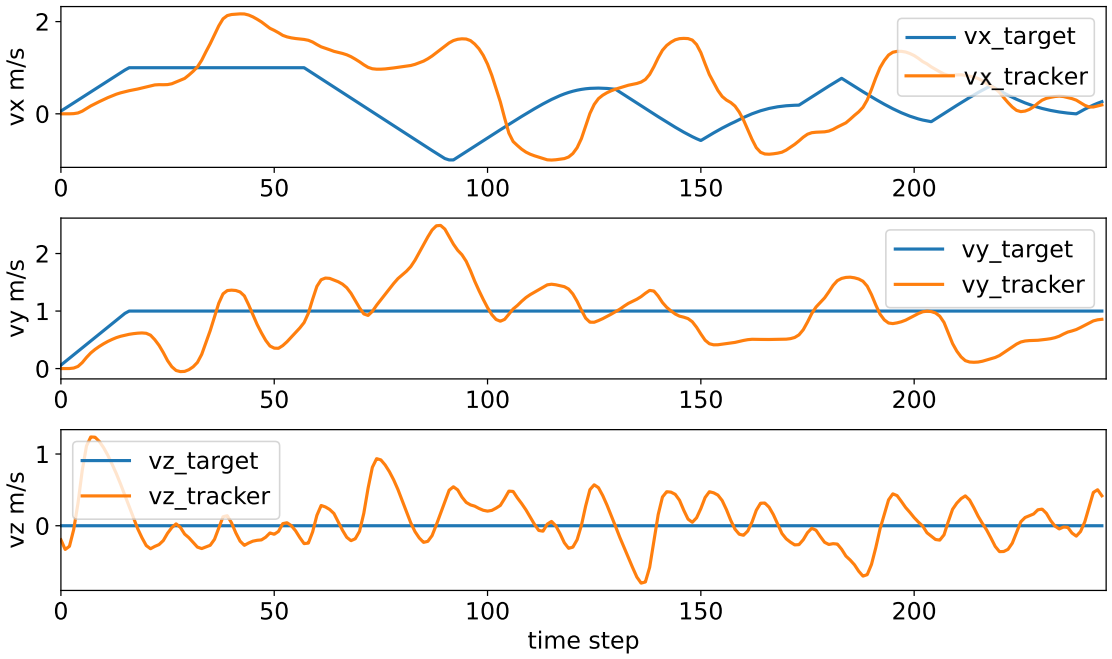


Figure 6.17: Velocity comparison of the quadrotor and the target at Stage 2. (Vision-standard)

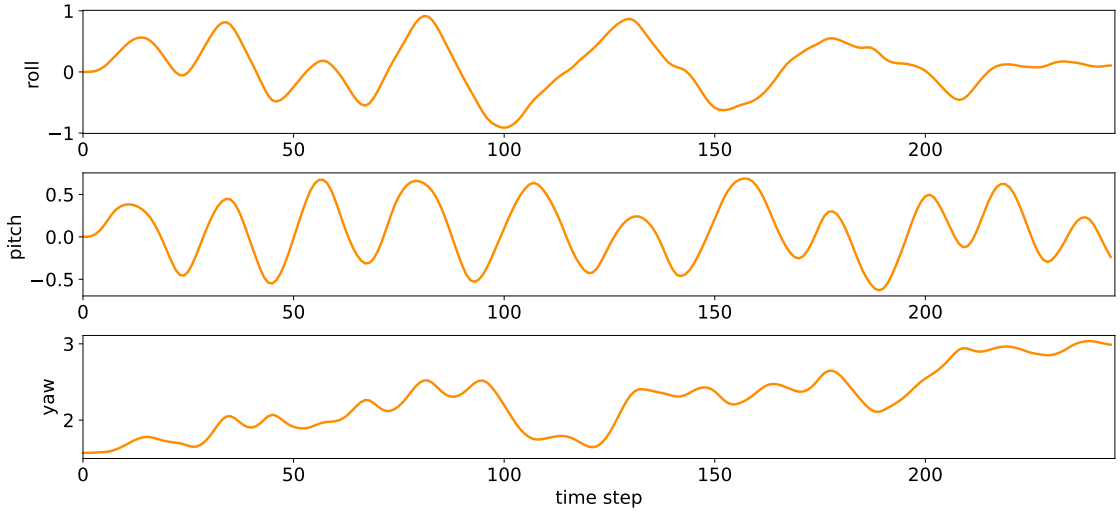


Figure 6.18: Abrupt change of the quadrotor attitude at Stage 2 (Vision-standard)

This fluctuation is also evident in the attitude change of the quadrotor, as de-

picted in Fig. 6.18. The violent changes in the quadrotor's roll and pitch angles contribute to an unstable-looking orientation of the quadrotor camera. Consequently, the target fails to remain in the camera centre as required. This deviation is noticeable in the plot of relative angles between the target and the quadrotor camera centre in Fig. 6.19. Here, γ and ψ represent the horizontal and vertical angles away from the camera-looking direction, respectively, as defined in Fig. 6.5. At certain time steps, the target may fall outside the camera's field of view (FOV). Such behaviour receives penalties through Eq. 6.11 and Eq. 6.12. The average episode reward during this stage jumped to around -18.53.

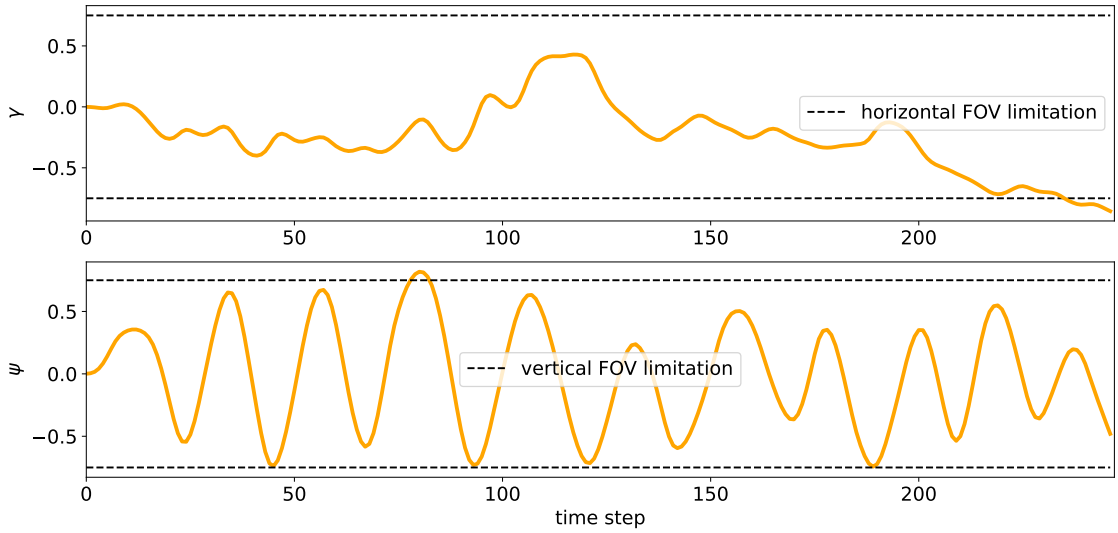


Figure 6.19: Relative angles between the target and the camera looking direction at Stage 2: The target may even be out of camera FOV (Vision-standard)

However, imposing such a high penalty leads to unexpected training outcomes. The agent strategically attempts to avoid these penalties by quickly destroying itself. This is accomplished by flying away from the target at an exceptionally high velocity (Stage 3 in Table 6.2). This behaviour is shown in Fig. 6.20 and Fig. 6.21. Through this strategy, the agent receives only a punishment of -10, significantly lower than the previously observed -18.53.

From an optimisation standpoint, this behaviour signifies a local minimum. Efforts to mitigate this issue by reducing the visibility punishment (using a smaller

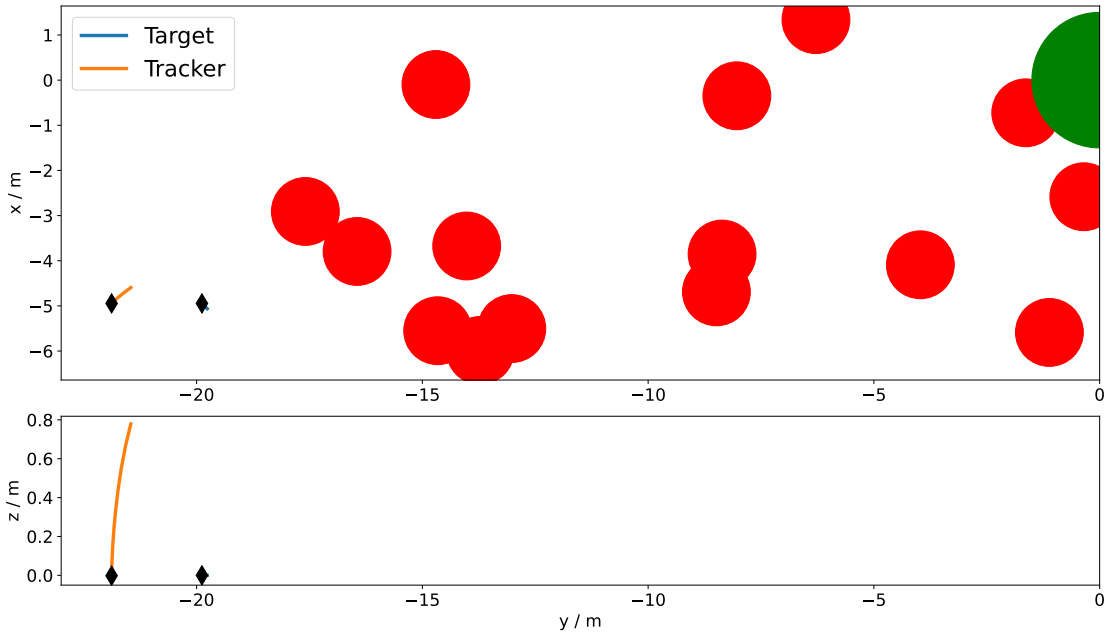


Figure 6.20: An tracking example of the Vision-standard agent at Stage 3: The agent flies away from the target as quickly as possible.

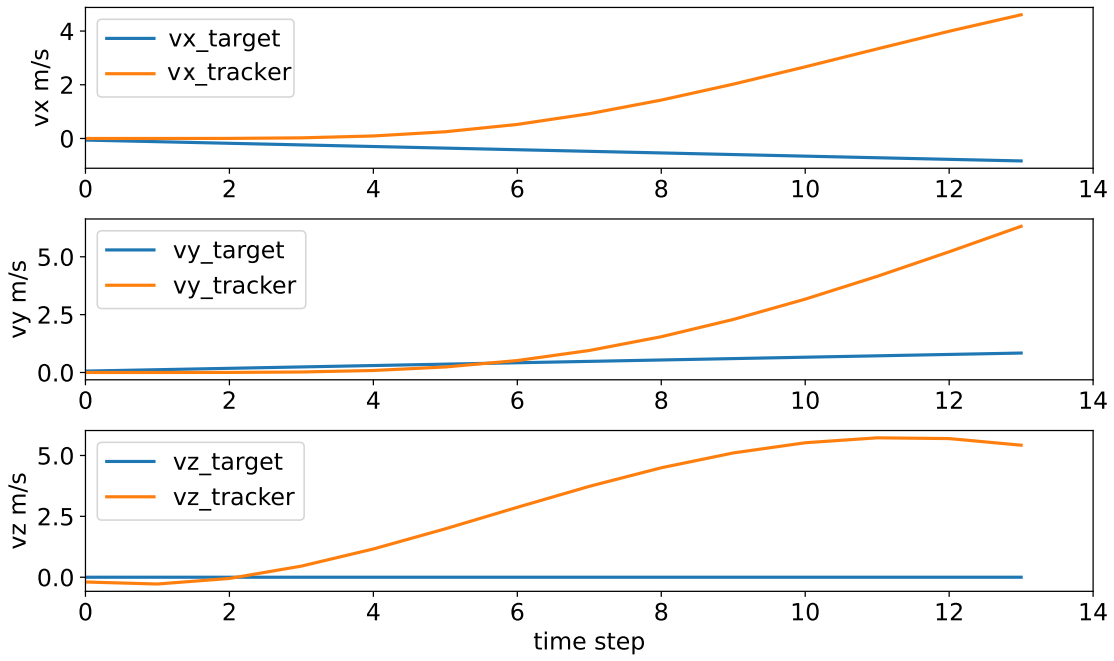


Figure 6.21: Velocity comparison of the quadrotor and the target (Vision-standard at Stage 3)

weight factor w as shown in Eq. 6.10, reduced from 1 to 0.1) were also made in training another vision-standard agent. However, this adjustment resulted in encountering the same local minimum (Vision-standard 2 in Fig. 6.13). To overcome

this local minimum, training examples collected from the state-based policy are utilized to enhance exploration in the action space for vision-based policies. The action demonstrations from the state-based teacher policy will guide the vision-based student policy out of the local minimum.

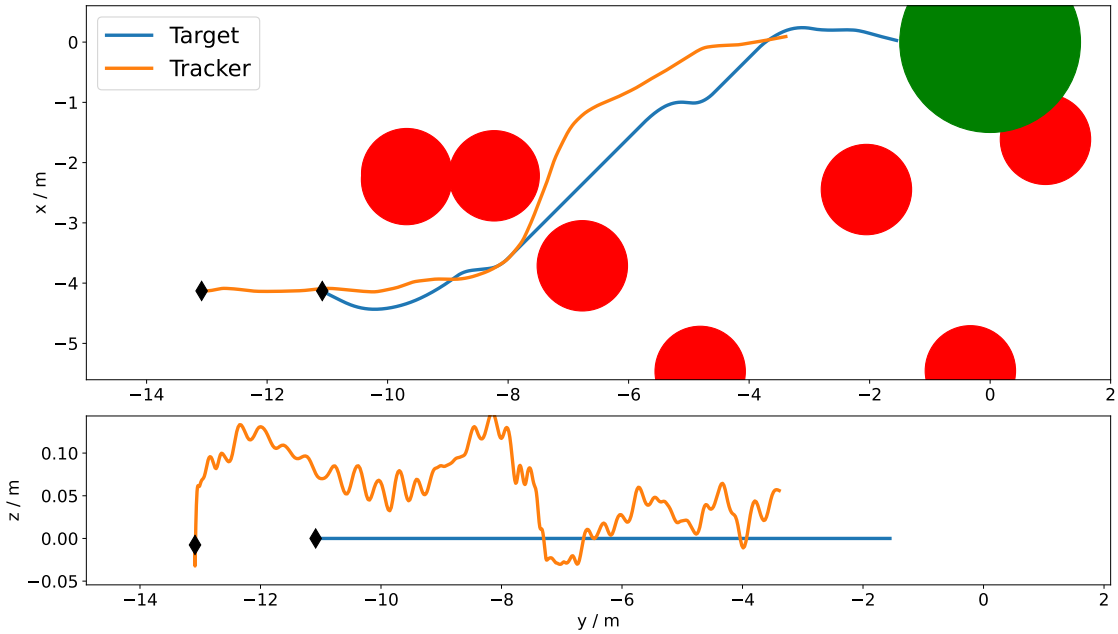


Figure 6.22: An tracking example of the Vision-student agent: The agent can follow the target and avoid obstacles.

Vision-student RL Agent To mitigate the local minima encountered during training of the vision-based agent, the proposed algorithm of this chapter leverages insights from the state-based agent, as discussed in Section 6.2 and illustrated in Fig. 6.4. Fig. 6.13 demonstrates that the average rewards of the vision-student agent converge to approximately -1.5, indicating robust tracking performance. Examining a tracking trajectory example in Fig. 6.22 and the associated velocity profile in Fig. 6.23, it is evident that the quadrotor effectively follows the target, matching its velocity while avoiding obstacles. The smooth and delicate changes in quadrotor attitude, as depicted in Fig. 6.24, further affirm the agent's proficient control capabilities.

The effectiveness of visibility performance can be assessed by examining relative horizontal and vertical angles between the target and the camera-looking direction

throughout the tracking procedure, as illustrated in Fig. 6.25. The plots reveal that the relative angles remain small and consistently within the camera FOV ranges, affirming the agent's capability to maintain target visibility.

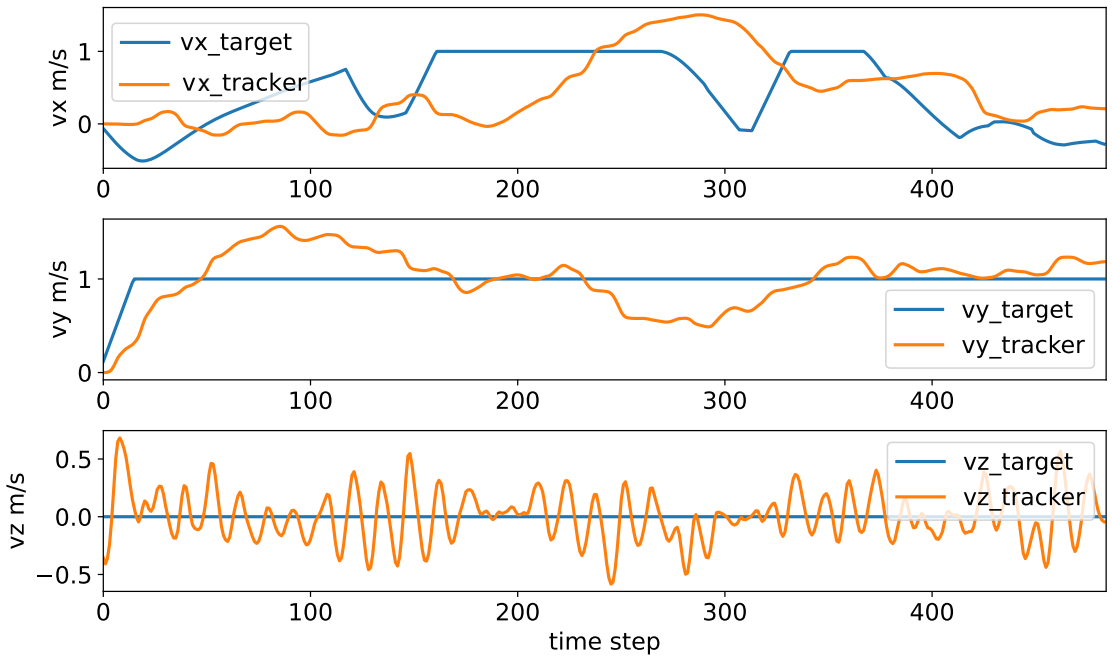


Figure 6.23: Velocity comparison of the quadrotor and the target (Vision-student)

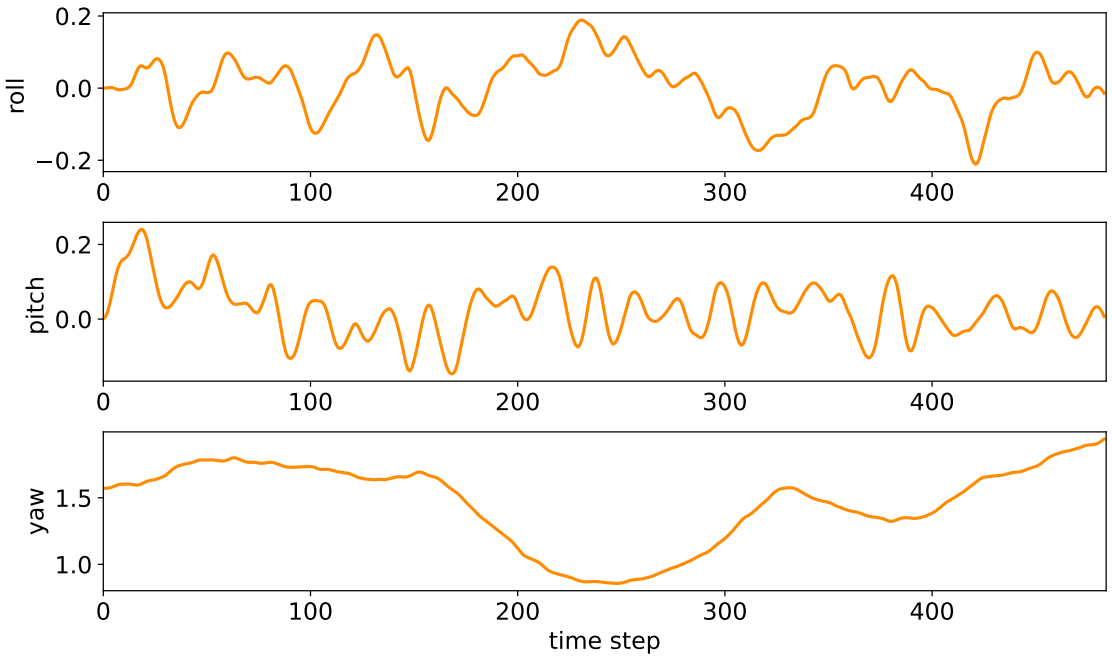


Figure 6.24: Delicate change of the quadrotor attitude (Vision-student)

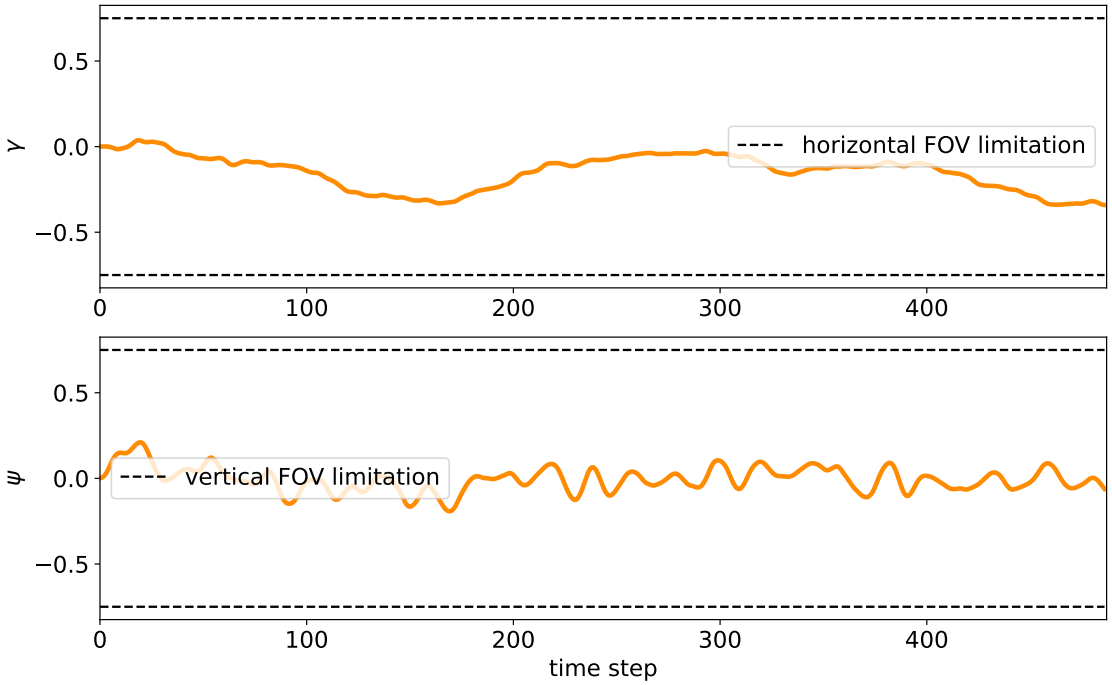


Figure 6.25: Relative angles between the target and the camera looking direction: both angles are small and always safisfy the FOV limitations (Vision-student)

6.3.3 Computation time

As discussed in Section 6.1 and illustrated in Fig. 6.2, traditional planning-based algorithms typically involve multiple sequential components. A comparison of computation times between the proposed end-to-end algorithm and three state-of-the-art planning-based algorithms, based on information from the work [170], is presented in Table 6.3. The proposed algorithm was tested on a desktop with an Intel Core i7-6700 CPU, identical to the setup in [170]. The table indicates that the network's forward step without GPU acceleration has a competitively low computation time compared to state-of-the-art path-planning-based algorithms. The conventional planning-based algorithms may also have communication costs between different components which the proposed method does not have. Additionally, it has been discussed in [135] that with a neural network policy for drone control coded with the Eigen library, the evaluation of the policy takes only 7 μs per time step, indicating the potential for computation time optimisation. Enabling GPU processing

further reduces the computation time, and GPUs are accessible when the quadrotor is controlled with a ground station computer. Additionally, many onboard computers, such as NVIDIA Jetson TX1 and TX2, Qualcomm Snapdragon, Odroid, etc., are designed to support GPU features, as highlighted in [216]. Therefore, the end-to-end algorithm shows significant potential for future drone applications.

Table 6.3: Computation time comparison

Method	Component(ms)					
	t_{path}	t_{corridor}	t_{ESDF}	t_{optimze}	t_{Network}	t_{total}
[217]	10.5	4.62	\	0.44	\	15.56
[169]	0.54	\	6.43	5.7	\	12.67
[170]	1.31	1.02	\	2.87	\	5.2
Vision-student (CPU)	\	\	\	\	7.4	7.4
Vision-student (GPU)	\	\	\	\	4.5	**4.5**

6.4 Conclusion

This chapter has established an end-to-end control framework for a drone engaged in a mobile target-tracking task, equipped with a forward-looking depth camera for perception (vision-based). Through the proposed training method, the vision-based drone can adeptly keep the mobile target within the centre of the image, ensuring maximum visibility and preventing occlusion from obstacles. Simultaneously, the drone can effectively avoid environmental obstacles.

The proposed method adopts a teacher-student training approach. Experimental findings show that training the policy network from scratch using depth image inputs leads to convergence challenges, resulting in a local minimum. To address this issue, a state-based teacher policy is trained as an expert to guide the vision-based policy. The input to the state-based teacher comprises low-dimensional data, including relative coordinates and radius of nearby obstacles, rather than high-dimensional image data. This low-dimensional input facilitates the state-based policy in achieving

satisfactory tracking performance post-training. During the training of the vision-based student policy, the state-based policy offers demonstrations. An adapted RL algorithm is introduced to leverage these demonstrations for effective training of the vision-based agent. This training pipeline ensures the successful accomplishment of the tracking task by the vision-based policy.

The proposed method, however, is exclusively tested in simulated environments, assuming obstacles are spherical in shape. Future work could broaden the algorithm's applicability to environments with diverse obstacle shapes. Additionally, the current output of the policy network consists of body-rate control signals, which might be too hazardous for direct drone control. A potential enhancement involves modifying the output to parameters that form a reference trajectory, such as control points for B-spline trajectories. This trajectory can then be forwarded to the controller for tracking, aligning with approaches found in [10].

Chapter 7

Conclusion and future work

This chapter concludes the study undertaken in this work. The main conclusions of this thesis will be highlighted in Section 7.1, while recommendations for future research directions are discussed in Section 7.2.

7.1 Conclusions

In conclusion, this thesis contributes to the development of navigation algorithms addressing the impact of practical pose estimation algorithms during mobile robot navigation. It also introduces an end-to-end tracking algorithm for autonomous UAV control, allowing it to follow a mobile target while simultaneously avoiding environmental obstacles. The successful tracking is achieved by designing an effective reward space and a novel teacher-student training scheme. Combining the capabilities of localisation-safe navigation and tracking, ground vehicles and aerial vehicles hold the potential to cooperatively execute complex tasks in unknown environments, such as search and rescue missions, in the future.

Ground Robot Navigation:

- This thesis enhances the mapless navigation capability for ground vehicles by reconfiguring the problem from a POMDP to an MDP setting. A novel approach is developed, which involves terminating training episodes early when localisation algorithms diverge, and reconstructing the state using historical information embeddings from an LSTM module.

- A unique reward component is introduced to penalise mobile robots in the event of localisation failure, a critical aspect previously overlooked in the existing literature.

- Furthermore, a novel training strategy is formulated, which trains navigation algorithms not only to reach destinations with minimal time cost but also to avoid regions where localisation algorithms are prone to failure.

Vision-based UAV Navigation:

- Addressing the challenge of vision-based navigation, a hierarchical navigation system is suggested for UAVs, combining an RL-based high-level policy with a conventional low-level policy. This hybrid structure leverages the strengths of neural networks in comprehending high-level information while ensuring safety and stability through conventional controllers.

- Moreover, this thesis proposes the use of semantic images as inputs for high-level policies rather than raw RGB images. This suggestion mitigates the sim-to-real problem encountered when unrealistic RGB images generated by simulators are used for training.

- Importantly, the proposed structure and input enable seamless transfer of control policies trained in simulations to achieve VO-safe navigation in real-world applications, eliminating the need for retraining.

UAV Tracking of a Moving Target:

- For the task of UAV tracking moving targets, a novel reward space is designed to guide the UAV's behaviour, particularly penalising instances where the target is occluded or positioned at the edge of the observation image.

- To tackle challenges in training vision-based tracking policies, a novel teacher-student training strategy for the tracking policy is developed. This strategy leverages experience data from a state-based teacher policy, effectively guiding the vision-based student policy away from a local minimum during training.

7.2 Future work

The proposed navigation strategy has been tested in relatively simple simulation environments without dense obstacles or dynamic objects. Future work could concentrate on evaluating the algorithm in more realistic and complex indoor and outdoor settings. Additionally, testing the algorithm with real-world robots and sensors presents an avenue for future research, requiring the development of techniques to address the sim-to-real transfer problem. Also, the proposed navigation and tracking algorithms are tested separately. In future work, these algorithms can be evaluated jointly as a whole system to accomplish more complex tasks.

The current tracking algorithm assumes the availability of relative poses between the UAV and UGV. However, this assumption may oversimplify real-world implementations. In crowded environments, UAVs may lose sight of the ground vehicle for several time steps. To address this, the tracking could be enhanced by having the UAV learn to predict the vehicle's moving intention by observing the vehicle's operating environment and historical behaviours in future work.

Another focus could involve exploring one step beyond tracking, specifically designing landing algorithms for UAVs on moving UGVs. The complex aerodynamics during UAV landing, which poses challenges for achieving steady landings, has not

been extensively discussed. Potential approaches include model-based learning algorithms that utilise neural networks to model complex dynamics and use the neural model for landing planning. Alternatively, model-free methods could be explored, training a landing policy following an end-to-end pipeline

Cooperative landing strategies between the UAV and UGV can also be explored. For example, UAVs, with their superior traffic views, could be assigned more authority to decide the landing location. Alternatively, when the UAV is close to running out of power and has limited manoeuvrability, the UGV could take a more active role in catching the UAVs. This intelligent workload distribution could potentially be learned through interaction during training.

Bibliography

[1] Joshua Laber, Ravindra Thamma, and E Daniel Kirby. The impact of warehouse automation in amazon's success. *Int. J. Innov. Sci. Eng. Technol*, 7:63–70, 2020.

[2] Jiaxu Xing, Giovanni Cioffi, Javier Hidalgo-Carrió, and Davide Scaramuzza. Autonomous power line inspection with drones via perception-aware mpc. In *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1086–1093. IEEE, 2023.

[3] Chao Wang, Jian Wang, Yuan Shen, and Xudong Zhang. Autonomous navigation of uavs in large-scale complex environments: A deep reinforcement learning approach. *IEEE Transactions on Vehicular Technology*, 68(3):2124–2136, 2019.

[4] Somil Bansal, Varun Tolani, Saurabh Gupta, Jitendra Malik, and Claire Tomlin. Combining optimal control and learning for visual navigation in novel environments. In *Conference on Robot Learning*, pages 420–429. PMLR, 2020.

[5] Julian Nubert, Etienne Walther, Shehryar Khattak, and Marco Hutter. Learning-based localizability estimation for robust lidar localization. In *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 17–24. IEEE, 2022.

[6] Antoni Rosinol Vidal, Henri Rebecq, Timo Horstschaefer, and Davide Scaramuzza. Ultimate slam? combining events, images, and imu for robust visual

slam in hdr and high-speed scenarios. *IEEE Robotics and Automation Letters*, 3(2):994–1001, 2018.

[7] Travis Manderson, Stefan Wapnick, David Meger, and Gregory Dudek. Learning to drive off road on smooth terrain in unstructured environments using an on-board camera and sparse aerial images. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1263–1269. IEEE, 2020.

[8] Minghan Wei and Volkan Isler. Air to ground collaboration for energy-efficient path planning for ground robots. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1949–1954. IEEE, 2019.

[9] Zhichao Han, Ruibin Zhang, Neng Pan, Chao Xu, and Fei Gao. Fast-tracker: A robust aerial system for tracking agile target in cluttered environments. In *2021 IEEE international conference on robotics and automation (ICRA)*, pages 328–334. IEEE, 2021.

[10] Antonio Loquercio, Elia Kaufmann, René Ranftl, Matthias Müller, Vladlen Koltun, and Davide Scaramuzza. Learning high-speed flight in the wild. *Science Robotics*, 6(59):eabg5810, 2021.

[11] Raul Mur-Artal and Juan D Tardós. Orb-slam2: An open-source slam system for monocular, stereo, and rgb-d cameras. *IEEE transactions on robotics*, 33(5):1255–1262, 2017.

[12] Zendai Kashino, Goldie Nejat, and Beno Benhabib. Aerial wilderness search and rescue with ground support. *Journal of Intelligent & Robotic Systems*, 99(1):147–163, 2020.

[13] Kahlouche Souhila and Achour Karim. Optical flow based robot obstacle avoidance. *International Journal of Advanced Robotic Systems*, 4(1):2, 2007.

[14] Liang Chen, Jian Yang, and Hui Kong. Lidar-histogram for fast road and obstacle detection. In *2017 IEEE international conference on robotics and automation (ICRA)*, pages 1343–1348. IEEE, 2017.

[15] Steven M LaValle, James J Kuffner, BR Donald, et al. Rapidly-exploring random trees: Progress and prospects. *Algorithmic and computational robotics: new directions*, 5:293–308, 2001.

[16] Hugh Durrant-Whyte and Tim Bailey. Simultaneous localization and mapping: part i. *IEEE robotics & automation magazine*, 13(2):99–110, 2006.

[17] Peter E Hart, Nils J Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

[18] Stefan Kohlbrecher, Johannes Meyer, Thorsten Graber, Karen Petersen, Uwe Klingauf, and Oskar Von Stryk. Hector open source modules for autonomous mapping and navigation with rescue robots. In *RoboCup 2013: Robot World Cup XVII 17*, pages 624–631. Springer, 2014.

[19] Markus Wulfmeier, Dushyant Rao, Dominic Zeng Wang, Peter Ondruska, and Ingmar Posner. Large-scale cost function learning for path planning using deep inverse reinforcement learning. *The International Journal of Robotics Research*, 36(10):1073–1087, 2017.

[20] Viktor Rausch, Andreas Hansen, Eugen Solowjow, Chang Liu, Edwin Kreuzer, and J Karl Hedrick. Learning a deep neural net policy for end-to-end control of autonomous vehicles. In *2017 American Control Conference (ACC)*, pages 4914–4919. IEEE, 2017.

[21] Xuesu Xiao, Bo Liu, Garrett Warnell, Jonathan Fink, and Peter Stone. Appld: Adaptive planner parameter learning from demonstration. *IEEE Robotics and Automation Letters*, 5(3):4541–4547, 2020.

[22] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[23] Lei Tai, Shaohua Li, and Ming Liu. A deep-network solution towards model-less obstacle avoidance. In *2016 IEEE/RSJ international conference on intelligent robots and systems (IROS)*, pages 2759–2764. IEEE, 2016.

[24] Mark Pfeiffer, Michael Schaeuble, Juan Nieto, Roland Siegwart, and Cesar Cadena. From perception to decision: A data-driven approach to end-to-end motion planning for autonomous ground robots. In *2017 ieee international conference on robotics and automation (icra)*, pages 1527–1533. IEEE, 2017.

[25] Keiron O'Shea and Ryan Nash. An introduction to convolutional neural networks. *arXiv preprint arXiv:1511.08458*, 2015.

[26] Asako Kanezaki, Jirou Nitta, and Yoko Sasaki. Goselo: Goal-directed obstacle and self-location map for robot navigation using reactive neural networks. *IEEE Robotics and Automation Letters*, 3(2):696–703, 2017.

[27] Alex Bewley, Jessica Rigley, Yuxuan Liu, Jeffrey Hawke, Richard Shen, Vinh-Dieu Lam, and Alex Kendall. Learning to drive from simulation without real world labels. In *2019 International conference on robotics and automation (ICRA)*, pages 4818–4824. IEEE, 2019.

[28] Lei Tai, Peng Yun, Yuying Chen, Congcong Liu, Haoyang Ye, and Ming Liu. Visual-based autonomous driving deployment from a stochastic and uncertainty-aware perspective. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2622–2628. IEEE, 2019.

[29] Sungjoon Choi, Kyungjae Lee, Sungbin Lim, and Songhwai Oh. Uncertainty-aware learning from demonstration using mixture density networks with sampling-free variance modeling. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6915–6922. IEEE, 2018.

[30] Kangwook Lee, Hoon Kim, and Changho Suh. Crash to not crash: Playing video games to predict vehicle collisions. 2017.

[31] Gregory Kahn, Pieter Abbeel, and Sergey Levine. Badgr: An autonomous self-supervised learning-based navigation system. *IEEE Robotics and Automation Letters*, 6(2):1312–1319, 2021.

[32] Xiaoyi Cai, Michael Everett, Lakshay Sharma, Philip R Osteen, and Jonathan P How. Probabilistic traversability model for risk-aware motion planning in off-road environments. *arXiv preprint arXiv:2210.00153*, 2022.

[33] Bo Liu, Xuesu Xiao, and Peter Stone. A lifelong learning approach to mobile robot navigation. *IEEE Robotics and Automation Letters*, 6(2):1090–1096, 2021.

[34] Aravind Rajeswaran, Vikash Kumar, Abhishek Gupta, Giulia Vezzani, John Schulman, Emanuel Todorov, and Sergey Levine. Learning complex dexterous manipulation with deep reinforcement learning and demonstrations. *arXiv preprint arXiv:1709.10087*, 2017.

[35] Kai Zhu and Tao Zhang. Deep reinforcement learning based mobile robot navigation: A review. *Tsinghua Science and Technology*, 26(5):674–691, 2021.

[36] Lei Tai and Ming Liu. A robot exploration strategy based on q-learning network. In *2016 ieee international conference on real-time computing and robotics (rcar)*, pages 57–62. IEEE, 2016.

[37] Lei Tai and Ming Liu. Towards cognitive exploration through deep reinforcement learning for mobile robots. *arXiv preprint arXiv:1610.01733*, 2016.

[38] Yuanda Wang, Haibo He, and Changyin Sun. Learning to navigate through complex dynamic environment with modular deep reinforcement learning. *IEEE Transactions on Games*, 10(4):400–412, 2018.

[39] Xiaogang Ruan, Dingqi Ren, Xiaoqing Zhu, and Jing Huang. Mobile robot navigation based on deep reinforcement learning. In *2019 Chinese control and decision conference (CCDC)*, pages 6174–6178. IEEE, 2019.

[40] Barzin Moridian, Brian R Page, and Nina Mahmoudian. Sample efficient reinforcement learning for navigation in complex environments. In *2019 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*, pages 15–21. IEEE, 2019.

[41] Hao-Tien Lewis Chiang, Aleksandra Faust, Marek Fiser, and Anthony Francis. Learning navigation behaviors end-to-end with autorl. *IEEE Robotics and Automation Letters*, 4(2):2007–2014, 2019.

[42] Hongji Huang, Yuchun Yang, Hong Wang, Zhiguo Ding, Hikmet Sari, and Fumiyuki Adachi. Deep reinforcement learning for uav navigation through massive mimo technique. *IEEE Transactions on Vehicular Technology*, 69(1):1117–1121, 2019.

[43] Wenyu Zhang, Jingyao Gai, Zhigang Zhang, Lie Tang, Qingxi Liao, and Youchun Ding. Double-dqn based path smoothing and tracking control method for robotic vehicle navigation. *Computers and Electronics in Agriculture*, 166:104985, 2019.

[44] Sivapong Nilwong and Genci Capi. Outdoor robot navigation system using game-based dqn and augmented reality. In *2020 17th International Conference on Ubiquitous Robots (UR)*, pages 74–80. IEEE, 2020.

[45] Lei Tai, Giuseppe Paolo, and Ming Liu. Virtual-to-real deep reinforcement learning: Continuous control of mobile robots for mapless navigation. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 31–36. IEEE, 2017.

[46] Ignacio Carlucho, Mariano De Paula, Sen Wang, Bruno V Menna, Yvan R Petillot, and Gerardo G Acosta. Auv position tracking control using end-to-end deep reinforcement learning. In *OCEANS 2018 MTS/IEEE Charleston*, pages 1–8. IEEE, 2018.

[47] Jiajun Duan, Di Shi, Ruisheng Diao, Haifeng Li, Zhiwei Wang, Bei Zhang, Desong Bian, and Zhehan Yi. Deep-reinforcement-learning-based autonomous voltage control for power grid operations. *IEEE Transactions on Power Systems*, 35(1):814–817, 2019.

[48] Mark Pfeiffer, Samarth Shukla, Matteo Turchetta, Cesar Cadena, Andreas Krause, Roland Siegwart, and Juan Nieto. Reinforced imitation: Sample efficient deep reinforcement learning for mapless navigation by leveraging prior demonstrations. *IEEE Robotics and Automation Letters*, 3(4):4423–4430, 2018.

[49] Shuran Song, Fisher Yu, Andy Zeng, Angel X Chang, Manolis Savva, and Thomas Funkhouser. Semantic scene completion from a single depth image. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1746–1754, 2017.

[50] Manolis Savva, Abhishek Kadian, Oleksandr Maksymets, Yili Zhao, Erik Wijmans, Bhavana Jain, Julian Straub, Jia Liu, Vladlen Koltun, Jitendra Malik, et al. Habitat: A platform for embodied ai research. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 9339–9347, 2019.

[51] Eric Kolve, Roozbeh Mottaghi, Winson Han, Eli VanderBilt, Luca Weihs, Alvaro Herrasti, Matt Deitke, Kiana Ehsani, Daniel Gordon, Yuke Zhu, et al. Ai2-thor: An interactive 3d environment for visual ai. *arXiv preprint arXiv:1712.05474*, 2017.

[52] Fei Xia, Amir R. Zamir, Zhi-Yang He, Alexander Sax, Jitendra Malik, and Silvio Savarese. Gibson env: real-world perception for embodied agents. In *Computer Vision and Pattern Recognition (CVPR), 2018 IEEE Conference on*. IEEE, 2018.

[53] Jonáš Kulhánek, Erik Derner, and Robert Babuška. Visual navigation in real-world indoor environments using end-to-end deep reinforcement learning. *IEEE Robotics and Automation Letters*, 6(3):4345–4352, 2021.

[54] Max Jaderberg, Volodymyr Mnih, Wojciech Marian Czarnecki, Tom Schaul, Joel Z Leibo, David Silver, and Koray Kavukcuoglu. Reinforcement learning with unsupervised auxiliary tasks. *arXiv preprint arXiv:1611.05397*, 2016.

[55] Piotr Mirowski, Razvan Pascanu, Fabio Viola, Hubert Soyer, Andrew J Ballard, Andrea Banino, Misha Denil, Ross Goroshin, Laurent Sifre, Koray Kavukcuoglu, et al. Learning to navigate in complex environments. *arXiv preprint arXiv:1611.03673*, 2016.

[56] T Tongloy, S Chuwongin, K Jaksukam, C Chousangsuntorn, and S Boonsang. Asynchronous deep reinforcement learning for the mobile robot navigation with supervised auxiliary tasks. In *2017 2nd International Conference on Robotics and Automation Engineering (ICRAE)*, pages 68–72. IEEE, 2017.

[57] Yuke Zhu, Roozbeh Mottaghi, Eric Kolve, Joseph J Lim, Abhinav Gupta, Li Fei-Fei, and Ali Farhadi. Target-driven visual navigation in indoor scenes using deep reinforcement learning. In *2017 IEEE international conference on robotics and automation (ICRA)*, pages 3357–3364. IEEE, 2017.

[58] Jonáš Kulhánek, Erik Derner, Tim De Bruin, and Robert Babuška. Vision-based navigation using deep reinforcement learning. In *2019 european conference on mobile robots (ECMR)*, pages 1–8. IEEE, 2019.

[59] Shih-Hsi Hsu, Shao-Hung Chan, Ping-Tsang Wu, Kun Xiao, and Li-Chen Fu. Distributed deep reinforcement learning based indoor visual navigation. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2532–2537. IEEE, 2018.

[60] Marvin Chancán and Michael Milford. Citylearn: Diverse real-world environments for sample-efficient navigation policy learning. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1697–1704. IEEE, 2020.

[61] Raphael Druon, Yusuke Yoshiyasu, Asako Kanezaki, and Alassane Watt. Visual object search by learning spatial context. *IEEE Robotics and Automation Letters*, 5(2):1279–1286, 2020.

[62] Yimeng Li and Jana Košecka. Learning view and target invariant visual servoing for navigation. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 658–664. IEEE, 2020.

[63] Truong Xuan Tung and Trung Dung Ngo. Socially aware robot navigation using deep reinforcement learning. In *2018 IEEE Canadian Conference on Electrical & Computer Engineering (CCECE)*, pages 1–5. IEEE, 2018.

[64] Aniket Bera, Tanmay Randhavane, Rohan Prinja, and Dinesh Manocha. Sociosense: Robot navigation amongst pedestrians with social and psychological constraints. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 7018–7025. IEEE, 2017.

[65] Jun Jin, Nhat M Nguyen, Nazmus Sakib, Daniel Graves, Hengshuai Yao, and Martin Jagersand. Mapless navigation among dynamics with social-safety-awareness: a reinforcement learning approach from 2d laser scans. In *2020 IEEE international conference on robotics and automation (ICRA)*, pages 6979–6985. IEEE, 2020.

[66] Lei Tai, Jingwei Zhang, Ming Liu, and Wolfram Burgard. Socially compliant navigation through raw depth inputs with generative adversarial imitation learning. In *2018 IEEE international conference on robotics and automation (ICRA)*, pages 1111–1117. IEEE, 2018.

[67] Chieh-En Tsai and Jean Oh. A generative approach for socially compliant navigation. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2160–2166. IEEE, 2020.

[68] Sujeong Kim, Stephen J Guy, Wenxi Liu, David Wilkie, Rynson WH Lau, Ming C Lin, and Dinesh Manocha. Brvo: Predicting pedestrian trajectories using velocity-space reasoning. *The International Journal of Robotics Research*, 34(2):201–217, 2015.

[69] Aniket Bera, Nico Galoppo, Dillon Sharlet, Adam Lake, and Dinesh Manocha. Adapt: real-time adaptive pedestrian tracking for crowded scenes. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1801–1808. IEEE, 2014.

[70] Hao Xue, Du Q Huynh, and Mark Reynolds. Ss-lstm: A hierarchical lstm model for pedestrian trajectory prediction. In *2018 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 1186–1194. IEEE, 2018.

[71] Nicolas Heess, Jonathan J Hunt, Timothy P Lillicrap, and David Silver. Memory-based control with recurrent neural networks. *arXiv preprint arXiv:1512.04455*, 2015.

[72] Matthew Hausknecht and Peter Stone. Deep recurrent q-learning for partially observable mdps. In *2015 aaai fall symposium series*, 2015.

[73] Lingheng Meng, Rob Gorbet, and Dana Kulić. Memory-based deep reinforcement learning for pomdps. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5619–5626. IEEE, 2021.

[74] Yong Yu, Xiaosheng Si, Changhua Hu, and Jianxun Zhang. A review of recurrent neural networks: Lstm cells and network architectures. *Neural computation*, 31(7):1235–1270, 2019.

[75] Alessandro Devo, Gabriele Costante, and Paolo Valigi. Deep reinforcement learning for instruction following visual navigation in 3d maze-like environments. *IEEE Robotics and Automation Letters*, 5(2):1175–1182, 2020.

[76] Jeffrey Pennington, Richard Socher, and Christopher D Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014.

[77] Linhai Xie, Andrew Markham, and Niki Trigoni. Snapnav: learning mapless visual navigation with sparse directional guidance and visual reference. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1682–1688. IEEE, 2020.

[78] Gino Brunner, Oliver Richter, Yuyi Wang, and Roger Wattenhofer. Teaching a machine to read maps with deep reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

[79] Yinlam Chow, Mohammad Ghavamzadeh, Lucas Janson, and Marco Pavone. Risk-constrained reinforcement learning with percentile risk criteria. *The Journal of Machine Learning Research*, 18(1):6070–6120, 2017.

[80] Joshua Achiam, David Held, Aviv Tamar, and Pieter Abbeel. Constrained policy optimization. In *International conference on machine learning*, pages 22–31. PMLR, 2017.

[81] Chuangchuang Sun, Dong-Ki Kim, and Jonathan P How. Fisar: Forward invariant safe reinforcement learning with a deep neural network-based opti-

mizer. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 10617–10624. IEEE, 2021.

[82] Klaas Kelchtermans and Tinne Tuytelaars. Rara: Zero-shot sim2real visual navigation with following foreground cues. In *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1704–1710. IEEE, 2022.

[83] Wei Zhang, Ning Liu, and Yunfeng Zhang. Learn to navigate maplessly with varied lidar configurations: A support point-based approach. *IEEE Robotics and Automation Letters*, 6(2):1918–1925, 2021.

[84] Jui-Te Huang, Chen-Lung Lu, Po-Kai Chang, Ching-I Huang, Chao-Chun Hsu, Po-Jui Huang, Hsueh-Cheng Wang, et al. Cross-modal contrastive learning of representations for navigation using lightweight, low-cost millimeter wave radar for adverse environmental conditions. *IEEE Robotics and Automation Letters*, 6(2):3333–3340, 2021.

[85] Joanne Truong, Sonia Chernova, and Dhruv Batra. Bi-directional domain adaptation for sim2real transfer of embodied navigation agents. *IEEE Robotics and Automation Letters*, 6(2):2634–2641, 2021.

[86] Abhishek Kadian, Joanne Truong, Aaron Gokaslan, Alexander Clegg, Erik Wijmans, Stefan Lee, Manolis Savva, Sonia Chernova, and Dhruv Batra. Sim2real predictivity: Does evaluation in simulation predict real-world performance? *IEEE Robotics and Automation Letters*, 5(4):6670–6677, 2020.

[87] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *Proceedings of the IEEE international conference on computer vision*, pages 2223–2232, 2017.

[88] Qian Luo, Maks Sorokin, and Sehoon Ha. A few shot adaptation of visual navigation skills to new observations using meta-learning. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 13231–13237. IEEE, 2021.

[89] Eugenio Chisari, Alexander Liniger, Alisa Rupenyan, Luc Van Gool, and John Lygeros. Learning from simulation, racing in reality. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 8046–8052. IEEE, 2021.

[90] Aleksandra Faust, Kenneth Oslund, Oscar Ramirez, Anthony Francis, Lydia Tapia, Marek Fiser, and James Davidson. Prm-rl: Long-range robotic navigation tasks by combining reinforcement learning and sampling-based planning. In *2018 IEEE international conference on robotics and automation (ICRA)*, pages 5113–5120. IEEE, 2018.

[91] Branka Mirchevska, Maria Hügle, Gabriel Kalweit, Moritz Werling, and Joschka Boedecker. Amortized q-learning with model-based action proposals for autonomous driving on highways. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1028–1035. IEEE, 2021.

[92] Zifan Xu, Gauraang Dhamankar, Anirudh Nair, Xuesu Xiao, Garrett Warnell, Bo Liu, Zizhao Wang, and Peter Stone. Applr: Adaptive planner parameter learning from reinforcement. In *2021 IEEE international conference on robotics and automation (ICRA)*, pages 6086–6092. IEEE, 2021.

[93] Zhiqian Qiao, Jeff Schneider, and John M Dolan. Behavior planning at urban intersections through hierarchical reinforcement learning. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2667–2673. IEEE, 2021.

[94] Bruno Brito, Michael Everett, Jonathan P How, and Javier Alonso-Mora. Where to go next: Learning a subgoal recommendation policy for navigation in dynamic environments. *IEEE Robotics and Automation Letters*, 6(3):4616–4623, 2021.

[95] Max Lodel, Bruno Brito, Alvaro Serra-Gómez, Laura Ferranti, Robert Babuška, and Javier Alonso-Mora. Where to look next: Learning viewpoint recommendations for informative trajectory planning. In *2022 International Conference on Robotics and Automation (ICRA)*, pages 4466–4472. IEEE, 2022.

[96] Colin Greatwood and Arthur G Richards. Reinforcement learning and model predictive control for robust embedded quadrotor guidance and control. *Autonomous Robots*, 43:1681–1693, 2019.

[97] Dieter Fox, Wolfram Burgard, and Sebastian Thrun. The dynamic window approach to collision avoidance. *IEEE Robotics & Automation Magazine*, 4(1):23–33, 1997.

[98] Jinning Li, Liting Sun, Jianyu Chen, Masayoshi Tomizuka, and Wei Zhan. A safe hierarchical planning framework for complex driving scenarios based on reinforcement learning. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2660–2666. IEEE, 2021.

[99] Utsav Patel, Nithish K Sanjeev Kumar, Adarsh Jagan Sathyamoorthy, and Dinesh Manocha. Dwa-rl: Dynamically feasible deep reinforcement learning policy for robot navigation among mobile obstacles. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6057–6063. IEEE, 2021.

[100] Owen Lockwood and Mei Si. A review of uncertainty for deep reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, volume 18, pages 155–162, 2022.

[101] Myounghoe Kim, Joohwan Seo, Mingoo Lee, and Jongeun Choi. Vision-based uncertainty-aware lane keeping strategy using deep reinforcement learning. *Journal of Dynamic Systems, Measurement, and Control*, 143(8):084503, 2021.

[102] Felipe Leno Da Silva, Pablo Hernandez-Leal, Bilal Kartal, and Matthew E Taylor. Uncertainty-aware action advising for deep reinforcement learning agents. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pages 5792–5799, 2020.

[103] Krishan Rana, Ben Talbot, Vibhavari Dasagi, Michael Milford, and Niko Sünderhauf. Residual reactive navigation: Combining classical and learned navigation strategies for deployment in unknown environments. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 11493–11499. IEEE, 2020.

[104] Gregory Kahn, Adam Villaflor, Vitchyr Pong, Pieter Abbeel, and Sergey Levine. Uncertainty-aware reinforcement learning for collision avoidance. *arXiv preprint arXiv:1702.01182*, 2017.

[105] Zhijian Liu, Alexander Amini, Sibo Zhu, Sertac Karaman, Song Han, and Daniela L Rus. Efficient and robust lidar-based end-to-end navigation. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 13247–13254. IEEE, 2021.

[106] Zizhao Wang, Xuesu Xiao, Bo Liu, Garrett Warnell, and Peter Stone. Appli: Adaptive planner parameter learning from interventions. In *2021 IEEE international conference on robotics and automation (ICRA)*, pages 6079–6085. IEEE, 2021.

[107] Xiaolin Tang, Kai Yang, Hong Wang, Jiahang Wu, Yechen Qin, Wenhao Yu, and Dongpu Cao. Prediction-uncertainty-aware decision-making for autonomous vehicles. *IEEE Transactions on Intelligent Vehicles*, 7(4):849–862, 2022.

[108] Tingxiang Fan, Pinxin Long, Wenxi Liu, Jia Pan, Ruigang Yang, and Dinesh Manocha. Learning resilient behaviors for navigation under uncertainty. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5299–5305. IEEE, 2020.

[109] Farzad Niroui, Kaicheng Zhang, Zendai Kashino, and Goldie Nejat. Deep reinforcement learning robot for search and rescue applications: Exploration in unknown cluttered environments. *IEEE Robotics and Automation Letters*, 4(2):610–617, 2019.

[110] Xiang Cao, Changyin Sun, and Mingzhong Yan. Target search control of auv in underwater environment with deep reinforcement learning. *IEEE Access*, 7:96549–96559, 2019.

[111] Tao Chen, Saurabh Gupta, and Abhinav Gupta. Learning exploration policies for navigation. *arXiv preprint arXiv:1903.01959*, 2019.

[112] Devendra Singh Chaplot, Dhiraj Gandhi, Saurabh Gupta, Abhinav Gupta, and Ruslan Salakhutdinov. Learning to explore using active neural slam. *arXiv preprint arXiv:2004.05155*, 2020.

[113] Santhosh K Ramakrishnan, Ziad Al-Halah, and Kristen Grauman. Occupancy anticipation for efficient exploration and navigation. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part V 16*, pages 400–418. Springer, 2020.

[114] Fanfei Chen, Paul Szenher, Yewei Huang, Jinkun Wang, Tixiao Shan, Shi Bai, and Brendan Englot. Zero-shot reinforcement learning on graphs for

autonomous exploration under uncertainty. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5193–5199. IEEE, 2021.

[115] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

[116] Liangheng Lv, Sunjie Zhang, Derui Ding, and Yongxiong Wang. Path planning via an improved dqn-based learning policy. *IEEE Access*, 7:67319–67330, 2019.

[117] Aviv Tamar, Yi Wu, Garrett Thomas, Sergey Levine, and Pieter Abbeel. Value iteration networks. *Advances in neural information processing systems*, 29, 2016.

[118] Max Pflueger, Ali Agha, and Gaurav S Sukhatme. Rover-irl: Inverse reinforcement learning with soft value iteration networks for planetary rover path planning. *IEEE Robotics and Automation Letters*, 4(2):1387–1394, 2019.

[119] Pinxin Long, Tingxiang Fan, Xinyi Liao, Wenxi Liu, Hao Zhang, and Jia Pan. Towards optimally decentralized multi-robot collision avoidance via deep reinforcement learning. In *2018 IEEE international conference on robotics and automation (ICRA)*, pages 6252–6259. IEEE, 2018.

[120] Tingxiang Fan, Xinjing Cheng, Jia Pan, Pinxin Long, Wenxi Liu, Ruigang Yang, and Dinesh Manocha. Getting robots unfrozen and unlost in dense pedestrian crowds. *IEEE Robotics and Automation Letters*, 4(2):1178–1185, 2019.

[121] Libo Sun, Jinfeng Zhai, and Wenhu Qin. Crowd navigation in an unknown and dynamic environment based on deep reinforcement learning. *IEEE Access*, 7:109544–109554, 2019.

[122] Jur Den Van Berg, Ming Lin, and Dinesh Manocha. Reciprocal velocity obstacles for real-time multi-agent navigation. In *Proceedings - IEEE International*

Conference on Robotics and Automation, pages 1928–1935, Pasadena, CA, USA, 2008.

[123] Yu Fan Chen, Miao Liu, Michael Everett, and Jonathan P How. Decentralized non-communicating multiagent collision avoidance with deep reinforcement learning. In *2017 IEEE international conference on robotics and automation (ICRA)*, pages 285–292. IEEE, 2017.

[124] Yue Jin, Yaodong Zhang, Jian Yuan, and Xudong Zhang. Efficient multi-agent cooperative navigation in unknown environments with interlaced deep reinforcement learning. In *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 2897–2901. IEEE, 2019.

[125] Alberto Viseras and Ricardo Garcia. Deepig: Multi-robot information gathering with deep reinforcement learning. *IEEE Robotics and Automation Letters*, 4(3):3059–3066, 2019.

[126] Chao Yan, Chang Wang, Xiaojia Xiang, Zhen Lan, and Yuna Jiang. Deep reinforcement learning of collision-free flocking policies for multiple fixed-wing uavs using local situation maps. *IEEE Transactions on Industrial Informatics*, 18(2):1260–1270, 2021.

[127] Chao Wang, Jian Wang, and Xudong Zhang. A deep reinforcement learning approach to flocking and navigation of uavs in large-scale complex environments. In *2018 IEEE global conference on signal and information processing (GlobalSIP)*, pages 1228–1232. IEEE, 2018.

[128] Xinyuan Zhou, Peng Wu, Haifeng Zhang, Weihong Guo, and Yuanchang Liu. Learn to navigate: cooperative path planning for unmanned surface vehicles using deep reinforcement learning. *Ieee Access*, 7:165262–165278, 2019.

[129] Ryan Lowe, Yi I Wu, Aviv Tamar, Jean Harb, OpenAI Pieter Abbeel, and Igor Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments. *Advances in neural information processing systems*, 30, 2017.

[130] Antonio Loquercio, Ana I Maqueda, Carlos R Del-Blanco, and Davide Scaramuzza. Dronet: Learning to fly by driving. *IEEE Robotics and Automation Letters*, 3(2):1088–1095, 2018.

[131] Elia Kaufmann, Antonio Loquercio, Rene Ranftl, Alexey Dosovitskiy, Vladlen Koltun, and Davide Scaramuzza. Deep drone racing: Learning agile flight in dynamic environments. In *Conference on Robot Learning*, pages 133–145. PMLR, 2018.

[132] Jiawei Fu, Yunlong Song, Yan Wu, Fisher Yu, and Davide Scaramuzza. Learning deep sensorimotor policies for vision-based autonomous drone racing. *arXiv preprint arXiv:2210.14985*, 2022.

[133] W Keith Hastings. Monte carlo sampling methods using markov chains and their applications. 1970.

[134] Christian Pfeiffer, Simon Wengeler, Antonio Loquercio, and Davide Scaramuzza. Visual attention prediction improves performance of autonomous drone racing agents. *Plos one*, 17(3):e0264471, 2022.

[135] Jemin Hwangbo, Inkyu Sa, Roland Siegwart, and Marco Hutter. Control of a quadrotor with reinforcement learning. *IEEE Robotics and Automation Letters*, 2(4):2096–2103, 2017.

[136] Artem Molchanov, Tao Chen, Wolfgang Hönig, James A Preiss, Nora Ayanian, and Gaurav S Sukhatme. Sim-to-(multi)-real: Transfer of low-level robust control policies to multiple quadrotors. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 59–66. IEEE, 2019.

[137] Jiarong Lin, Luqi Wang, Fei Gao, Shaojie Shen, and Fu Zhang. Flying through a narrow gap using neural network: an end-to-end planning and control approach. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3526–3533. IEEE, 2019.

[138] Yuanda Wang, Jia Sun, Haibo He, and Changyin Sun. Deterministic policy gradient with integral compensator for robust quadrotor control. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 50(10):3713–3725, 2019.

[139] Siddharth Mysore, Bassel Mabsout, Renato Mancuso, and Kate Saenko. Regularizing action policies for smooth control with reinforcement learning. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1810–1816. IEEE, 2021.

[140] Robert Penicka, Yunlong Song, Elia Kaufmann, and Davide Scaramuzza. Learning minimum-time flight in cluttered environments. *IEEE Robotics and Automation Letters*, 7(3):7209–7216, 2022.

[141] Yunlong Song, Kexin Shi, Robert Penicka, and Davide Scaramuzza. Learning perception-aware agile flight in cluttered environments. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1989–1995. IEEE, 2023.

[142] Antonio Loquercio, Elia Kaufmann, René Ranftl, Alexey Dosovitskiy, Vladlen Koltun, and Davide Scaramuzza. Deep drone racing: From simulation to reality with domain randomization. *IEEE Transactions on Robotics*, 36(1):1–14, 2019.

[143] Jiaxu Xing, Leonard Bauersfeld, Yunlong Song, Chunwei Xing, and Davide Scaramuzza. Contrastive learning for enhancing robust scene transfer in vision-based agile flight. *arXiv preprint arXiv:2309.09865*, 2023.

[144] Tianqi Wang and Dong Eui Chang. Robust navigation for racing drones based on imitation learning and modularization. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 13724–13730. IEEE, 2021.

[145] Angel Romero, Yunlong Song, and Davide Scaramuzza. Actor-critic model predictive control. *arXiv preprint arXiv:2306.09852*, 2023.

[146] Yunlong Song and Davide Scaramuzza. Policy search for model predictive control with application to agile drone flight. *IEEE Transactions on Robotics*, 38(4):2114–2130, 2022.

[147] Angel Romero, Shreedhar Govil, Gonca Yilmaz, Yunlong Song, and Davide Scaramuzza. Weighted maximum likelihood for controller tuning. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1334–1341. IEEE, 2023.

[148] Tim Salzmann, Elia Kaufmann, Jon Arrizabalaga, Marco Pavone, Davide Scaramuzza, and Markus Ryll. Real-time neural mpc: Deep learning model predictive control for quadrotors and agile robotic platforms. *IEEE Robotics and Automation Letters*, 8(4):2397–2404, 2023.

[149] Nina Wiedemann, Valentin Wüest, Antonio Loquercio, Matthias Müller, Dario Floreano, and Davide Scaramuzza. Training efficient controllers via analytic policy gradient. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1349–1356. IEEE, 2023.

[150] Yunlong Song, Angel Romero, Matthias Müller, Vladlen Koltun, and Davide Scaramuzza. Reaching the limit in autonomous racing: Optimal control versus reinforcement learning. *Science Robotics*, 8(82):eadg1462, 2023.

[151] Elia Kaufmann, Leonard Bauersfeld, and Davide Scaramuzza. A benchmark comparison of learned control policies for agile quadrotor flight. In *2022 Inter-*

national Conference on Robotics and Automation (ICRA), pages 10504–10510. IEEE, 2022.

[152] Elia Kaufmann, Leonard Bauersfeld, Antonio Loquercio, Matthias Müller, Vladlen Koltun, and Davide Scaramuzza. Champion-level drone racing using deep reinforcement learning. *Nature*, 620(7976):982–987, 2023.

[153] Nathan O Lambert, Daniel S Drew, Joseph Yaconelli, Sergey Levine, Roberto Calandra, and Kristofer SJ Pister. Low-level control of a quadrotor with deep model-based reinforcement learning. *IEEE Robotics and Automation Letters*, 4(4):4224–4230, 2019.

[154] Suneel Belkhale, Rachel Li, Gregory Kahn, Rowan McAllister, Roberto Calandra, and Sergey Levine. Model-based meta-reinforcement learning for flight with suspended payloads. *IEEE Robotics and Automation Letters*, 6(2):1471–1478, 2021.

[155] Mariah L Schrum and Matthew C Gombolay. When your robot breaks: Active learning during plant failure. *IEEE Robotics and Automation Letters*, 5(2):438–445, 2019.

[156] Alexander Spitzer and Nathan Michael. Feedback linearization for quadrotors with a learned acceleration error model. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6042–6048. IEEE, 2021.

[157] Guanya Shi, Xichen Shi, Michael O'Connell, Rose Yu, Kamyar Azizzadenesheli, Animashree Anandkumar, Yisong Yue, and Soon-Jo Chung. Neural lander: Stable drone landing control using learned dynamics. In *2019 international conference on robotics and automation (icra)*, pages 9784–9790. IEEE, 2019.

[158] JeongWoon Kim and David Hyunchul Shim. A vision-based target tracking control system of a quadrotor by using a tablet computer. In *2013 international*

conference on unmanned aircraft systems (icuas), pages 1165–1172. IEEE, 2013.

[159] Alex G Kendall, Nishaad N Salvapantula, and Karl A Stol. On-board object tracking control of a quadcopter with monocular vision. In *2014 international conference on unmanned aircraft systems (ICUAS)*, pages 404–411. IEEE, 2014.

[160] Hui Cheng, Lishan Lin, Zhuoqi Zheng, Yuwei Guan, and Zhongchang Liu. An autonomous vision-based target tracking system for rotorcraft unmanned aerial vehicles. In *2017 IEEE/RSJ international conference on intelligent robots and systems (IROS)*, pages 1732–1738. IEEE, 2017.

[161] Tobias Nägeli, Javier Alonso-Mora, Alexander Domahidi, Daniela Rus, and Otmar Hilliges. Real-time motion planning for aerial videography with dynamic obstacle avoidance and viewpoint optimization. *IEEE Robotics and Automation Letters*, 2(3):1696–1703, 2017.

[162] Justin Thomas, Jake Welde, Giuseppe Loianno, Kostas Daniilidis, and Vijay Kumar. Autonomous flight for detection, localization, and tracking of moving targets with a small quadrotor. *IEEE Robotics and Automation Letters*, 2(3):1762–1769, 2017.

[163] Bryan Penin, Paolo Robuffo Giordano, and François Chaumette. Vision-based reactive planning for aggressive target tracking while avoiding collisions and occlusions. *IEEE Robotics and Automation Letters*, 3(4):3725–3732, 2018.

[164] Rogerio Bonatti, Yanfu Zhang, Sanjiban Choudhury, Wenshan Wang, and Sebastian Scherer. Autonomous drone cinematographer: Using artistic principles to create smooth, safe, occlusion-free trajectories for aerial filming. In *Proceedings of the 2018 international symposium on experimental robotics*, pages 119–129. Springer, 2020.

[165] Jing Chen, Tianbo Liu, and Shaojie Shen. Tracking a moving target in cluttered environments using a quadrotor. In *IEEE International Conference on Intelligent Robots and Systems*, volume 2016-November, pages 446–453, 2016.

[166] Boseong Felipe Jeon and H Jin Kim. Online trajectory generation of a mav for chasing a moving target in 3d dense environments. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1115–1121. IEEE, 2019.

[167] Boseong Felipe Jeon, Dongsuk Shim, and H Jin Kim. Detection-aware trajectory generation for a drone cinematographer. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1450–1457. IEEE, 2020.

[168] Boseong Jeon, Yunwoo Lee, and H Jin Kim. Integrated motion planner for real-time aerial videography with a drone in a dense environment. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1243–1249. IEEE, 2020.

[169] Qianhao Wang, Yuman Gao, Jialin Ji, Chao Xu, and Fei Gao. Visibility-aware trajectory optimization with application to aerial tracking. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5249–5256. IEEE, 2021.

[170] Jialin Ji, Neng Pan, Chao Xu, and Fei Gao. Elastic tracker: A spatio-temporal trajectory planner for flexible aerial tracking. In *2022 International Conference on Robotics and Automation (ICRA)*, pages 47–53. IEEE, 2022.

[171] Fabian Schilling, Julien Lecoeur, Fabrizio Schiano, and Dario Floreano. Learning vision-based flight in drone swarms by imitation. *IEEE Robotics and Automation Letters*, 4(4):4523–4530, 2019.

[172] Sarthak Bhagat and PB Sujit. Uav target tracking in urban environments using deep reinforcement learning. In *2020 International conference on unmanned aircraft systems (ICUAS)*, pages 694–701. IEEE, 2020.

[173] Jiseon Moon, Savvas Papaioannou, Christos Laoudias, Panayiotis Kolios, and Sunwoo Kim. Deep reinforcement learning multi-uav trajectory control for target tracking. *IEEE Internet of Things Journal*, 8(20):15441–15455, 2021.

[174] Bohao Li and Yunjie Wu. Path planning for uav ground target tracking via deep reinforcement learning. *IEEE access*, 8:29064–29074, 2020.

[175] Bo Li, Zhi-peng Yang, Da-qing Chen, Shi-yang Liang, and Hao Ma. Maneuvering target tracking of uav based on mn-ddpg and transfer learning. *Defence Technology*, 17(2):457–466, 2021.

[176] Jiahua Wang, Ping Zhang, and Yang Wang. Autonomous target tracking of multi-uav: A two-stage deep reinforcement learning approach with expert experience. *Applied Soft Computing*, 145:110604, 2023.

[177] Hu Duoxiu, Dong Wenhan, Xie Wujie, and He Lei. Proximal policy optimization for multi-rotor uav autonomous guidance, tracking and obstacle avoidance. *International Journal of Aeronautical and Space Sciences*, 23(2):339–353, 2022.

[178] Jiang Zhao, Han Liu, Jiaming Sun, Kun Wu, Zhihao Cai, Yan Ma, and Yingxun Wang. Deep reinforcement learning-based end-to-end control for uav dynamic target tracking. *Biomimetics*, 7(4):197, 2022.

[179] Hy Nguyen, Srikanth Thudumu, Hung Du, Kon Mouzakis, and Rajesh Vasa. Uav dynamic object tracking with lightweight deep vision reinforcement learning. *Algorithms*, 16(5):227, 2023.

[180] Gui Fu, Hongyu Chu, Liwen Liu, Linyi Fang, and Xinyu Zhu. Deep reinforcement learning for the visual servoing control of uavs with fov constraint. *Drones*, 7(6):375, 2023.

[181] Stefan Kohlbrecher, Oskar Von Stryk, Johannes Meyer, and Uwe Klingauf. A flexible and scalable slam system with full 3d motion estimation. In *2011 IEEE international symposium on safety, security, and rescue robotics*, pages 155–160. IEEE, 2011.

[182] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[183] Yaakov Engel, Shie Mannor, and Ron Meir. Reinforcement learning with gaussian processes. In *Proceedings of the 22nd international conference on Machine learning*, pages 201–208, 2005.

[184] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

[185] Gerald Tesauro et al. Temporal difference learning and td-gammon. *Communications of the ACM*, 38(3):58–68, 1995.

[186] Xiaofei Wang, Kimin Lee, Kourosh Hakhamaneshi, Pieter Abbeel, and Michael Laskin. Skill preferences: Learning to extract and execute robotic skills from human feedback. In *Conference on Robot Learning*, pages 1259–1268. PMLR, 2022.

[187] Lilian Weng. A (long) peek into reinforcement learning. *lilianweng.github.io*, 2018.

[188] Richard S Sutton, David McAllester, Satinder Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. *Advances in neural information processing systems*, 12, 1999.

[189] JN Tsitsiklis and B Van Roy. An analysis of temporal-difference learning with function approximationtechnical. *Rep. LIDS-P-2322). Lab. Inf. Decis. Syst. Massachusetts Inst. Technol. Tech. Rep*, 1996.

[190] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[191] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897. PMLR, 2015.

[192] Sham M Kakade. A natural policy gradient. *Advances in neural information processing systems*, 14, 2001.

[193] Sham Kakade and John Langford. Approximately optimal approximate reinforcement learning. In *Proceedings of the Nineteenth International Conference on Machine Learning*, pages 267–274, 2002.

[194] Nathan Koenig and Andrew Howard. Design and use paradigms for gazebo, an open-source multi-robot simulator. In *2004 IEEE/RSJ international conference on intelligent robots and systems (IROS)(IEEE Cat. No. 04CH37566)*, volume 3, pages 2149–2154. Ieee, 2004.

[195] Yunlong Song, Selim Naji, Elia Kaufmann, Antonio Loquercio, and Davide Scaramuzza. Flightmare: A flexible quadrotor simulator. In *Conference on Robot Learning*, 2020.

[196] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, Andrew Y Ng, et al. Ros: an open-source robot operating system. In *ICRA workshop on open source software*, volume 3, page 5. Kobe, Japan, 2009.

[197] Michael Montemerlo, Sebastian Thrun, Daphne Koller, Ben Wegbreit, et al. Fastslam: A factored solution to the simultaneous localization and mapping problem. *Aaai/iaai*, 593598, 2002.

[198] Giorgio Grisetti, Cyrill Stachniss, and Wolfram Burgard. Improved techniques for grid mapping with rao-blackwellized particle filters. *IEEE transactions on Robotics*, 23(1):34–46, 2007.

[199] Marcelo J Segura, Fernando A Auat Cheein, Juan M Toibero, Vicente Mut, and Ricardo Carelli. Ultra wide-band localization and slam: A comparative study for mobile robot navigation. *Sensors*, 11(2):2035–2055, 2011.

[200] Edwin Olson. Apriltag: A robust and flexible visual fiducial system. In *2011 IEEE international conference on robotics and automation*, pages 3400–3407. IEEE, 2011.

[201] Navid Kayhani, Adam Heins, Wenda Zhao, Mohammad Nahangi, Brenda McCabe, and Angela P Schoelligb. Improved tag-based indoor localization of uavs using extended kalman filter. In *Proceedings of the ISARC. International Symposium on Automation and Robotics in Construction, Banff, AB, Canada*, pages 21–24. sn, 2019.

[202] Shibo Zhao, Hengrui Zhang, Peng Wang, Lucas Nogueira, and Sebastian Scherer. Super odometry: Imu-centric lidar-visual-inertial estimator for challenging environments. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 8729–8736. IEEE, 2021.

[203] Oleksii Zhelo, Jingwei Zhang, Lei Tai, Ming Liu, and Wolfram Burgard. Curiosity-driven exploration for mapless navigation with deep reinforcement learning. *arXiv preprint arXiv:1804.00456*, 2018.

[204] Arthur Juliani, Vincent-Pierre Berges, Ervin Teng, Andrew Cohen, Jonathan Harper, Chris Elion, Chris Goy, Yuan Gao, Hunter Henry, Marwan Mattar, and Danny Lange. Unity: A general platform for intelligent agents. *arXiv preprint arXiv:1809.02627*, 2020.

[205] Erwin Coumans and Yunfei Bai. Pybullet, a python module for physics simulation for games, robotics and machine learning. `http://pybullet.org`, 2016–2021.

[206] Brett T Lopez and Jean-Jacques E Slotine. Sliding on manifolds: Geometric attitude control with quaternions. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 11140–11146. IEEE, 2021.

[207] Fadri Furrer, Michael Burri, Markus Achtelik, and Roland Siegwart. Rotors—a modular gazebo mav simulator framework. *Robot Operating System (ROS) The Complete Reference (Volume 1)*, pages 595–625, 2016.

[208] Davide Scaramuzza and Friedrich Fraundorfer. Visual odometry [tutorial]. *IEEE robotics & automation magazine*, 18(4):80–92, 2011.

[209] Friedrich Fraundorfer and Davide Scaramuzza. Visual odometry: Part ii: Matching, robustness, optimization, and applications. *IEEE Robotics & Automation Magazine*, 19(2):78–90, 2012.

[210] Edward Rosten and Tom Drummond. Machine learning for high-speed corner detection. In *Computer Vision–ECCV 2006: 9th European Conference on Computer Vision, Graz, Austria, May 7-13, 2006. Proceedings, Part I 9*, pages 430–443. Springer, 2006.

[211] Christopher G Harris and JM Pike. 3d positional integration from image sequences. *Image and Vision Computing*, 6(2):87–90, 1988.

[212] David G Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60:91–110, 2004.

[213] Tong Qin, Peiliang Li, and Shaojie Shen. Vins-mono: A robust and versatile monocular visual-inertial state estimator. *IEEE Transactions on Robotics*, 34(4):1004–1020, 2018.

[214] Thomas Degris, Martha White, and Richard S Sutton. Off-policy actor-critic. *arXiv preprint arXiv:1205.4839*, 2012.

[215] Josiah P Hanna, Scott Niekum, and Peter Stone. Importance sampling in reinforcement learning with an estimated behavior policy. *Machine Learning*, 110(6):1267–1317, 2021.

[216] Daniele Palossi, Antonio Loquercio, Francesco Conti, Eric Flamand, Davide Scaramuzza, and Luca Benini. A 64-mw dnn-based visual navigation engine for autonomous nano-drones. *IEEE Internet of Things Journal*, 6(5):8357–8371, 2019.

[217] Zhichao Han, Ruibin Zhang, Neng Pan, Chao Xu, and Fei Gao. Fast-tracker: A robust aerial system for tracking agile target in cluttered environments. *arXiv preprint arXiv:2011.03968*, 2020.