Deep Reinforcement Learning-Based Indoor Mapless Robot Navigation

September 2024

Yan Gao

in partial fulfilment of the requirements for the degree of

Doctor of Philosophy (Engineering)

School of Engineering

Cardiff University, United Kingdom

Abstract

Navigation is a critical capability for mobile robots, enabling movement from a source to a destination. Conventional methods depend on predefined maps, which are time-consuming and labour-intensive to construct. Mapless navigation removes this requirement by finding collision-free paths using partial environmental observations. With advances in computational power and machine learning, there is a growing shift toward deep reinforcement learning (DRL)-based mapless navigation, where robots learn actions directly from raw sensory inputs. This thesis focuses on developing and improving DRL-based mapless navigation systems.

Firstly, this thesis proposes a novel hierarchical reinforcement learning (HRL)-based mapless navigation framework. Specifically, it defines two different subgoal worthiness metrics: Predictive Neighbouring Space Scoring (PNSS) and Predictive Exploration Worthiness (PEW). PNSS relates to the explorable space for each subgoal, while PEW refers to the spatial distribution of obstacles, including the area of free space and the arrangement of obstacles around each subgoal. The PNSS and PEW models are developed to predict the PNSS and PEW values, enabling the robot to evaluate the worthiness of each subgoal. Then, these predicted PNSS or PEW values are incorporated into the high-level (HL) input representations, resulting in a more compact and informative representation. Additionally, a penalty element is introduced in the HL reward function, allowing the HL policy to consider the capabilities of the low-level policy when selecting subgoals. Moreover, this thesis proposes a novel subgoal space layout that enables the robot to explore locations further from its current position. Experiments in unknown environments demonstrate significant improvements over baselines.

Then, this thesis develops a novel reward function and neural network

(NN) structure for HRL-based mapless navigation, designed to address local minimum issues in complex environments. The reward function for training the HL policy consists of two components: extrinsic reward and intrinsic reward. The extrinsic reward encourages the robot to move towards the target location, while the intrinsic reward, calculated based on novelty, episode memory, and memory decaying, enables the agent to engage in spontaneous exploration. The proposed NN architecture incorporates a Long Short-Term Memory (LSTM) network to enhance the agent's memory and reasoning capabilities. Testing in unknown environments shows a significant improvement in success rates and effective resolution of local minima, especially where baseline methods fail completely.

Finally, this thesis introduces a DRL-based mapless navigation method that does not assume the availability of accurate robot pose information. It utilises RGB-D-based ORB-SLAM2 for robot localisation. The trained policy effectively directs the robot towards the target while improving pose estimation by considering the quality of observed features along selected paths. The quality of features depends on both their quantity and distribution. To facilitate policy training, a compact state representation based on the spatial distribution of map points is proposed, enhancing the robot's awareness of areas with reliable features. Additionally, a novel reward function is designed that incorporates relative pose error. It increases the policy's responsiveness to individual actions. Instead of establishing a predetermined threshold to assess whether the discrepancy between the SLAM-predicted pose and the true value exceeds an acceptable limit, a dynamic threshold is employed to assess localisation performance, improving the policy's adaptability to variations in SLAM performance across different environments. Experiments show the method outperforms related RL-based approaches in localisation-challenging

environments.

This thesis presents novel DRL-based mapless navigation approaches, making significant contributions to both theory and practical applications. Together, these contributions advance the field of autonomous navigation, offering more adaptable, efficient, and scalable solutions.

Acknowledgement

During my time as a PhD student, I have many people to thank. First and foremost, I would like to express my deepest gratitude to my supervisor, Dr Ze Ji. Without his guidance and support, I would not have been able to successfully complete my studies. I am also deeply grateful to my second supervisor, Dr Jing Wu, for offering invaluable advice during moments of uncertainty. I extend my thanks to all the members of our research group for the many enjoyable hours we have spent together.

Moreover, I would like to thank my parents and my brother for their unwavering support and encouragement, letting me know that home is always a source of strength.

Finally, I express my thanks to the China Scholarship Council for financially supporting my tuition and living expenses during my PhD studies.

Contents

List of Figures

List of Tables

List of Acronyms

UGVs	Unmanned ground vehicles
UAVs	Unmanned aerial vehicles
DL	Deep learning
RL	Reinforcement learning
DRL	Deep reinforcement learning
DNN	Deep neural network
HRL	Hierarchical reinforcement learning
GNSS	Global navigation satellite system
SLAM	Simultaneous localisation and mapping
RPE	Related pose error
HL	High-level
LL	Low-level
IPS	Indoor positioning systems
EKF	Extended kalman filter
ERL	Evolutionary reinforcement learning
LSTM	Long short-term memory
NN	Neural network
MLP	Multi-layer perceptron
DQN	Deep Q network
DDPG	Deep deterministic policy gradient
SAC	Soft actor critic
MDP	Markov decision process
POMDP	Partially observable markov decision process
PPO	Proximal policy optimisation

TD3	Twin delayed deep deterministic policy gradient
A2C	Advantage actor-critic
DDQN	Double DQN
CNN	Convolutional neural network

Publications

[1] Gao, Y., Wu, J., Yang, X. and Ji, Z., 2023. Efficient hierarchical reinforcement learning for mapless navigation with predictive neighbouring space scoring. IEEE Transactions on Automation Science and Engineering.

[2] Gao, Y., Ji, Z., Wu, J., Wei, C. and Grech, R., 2023, August. Hierarchical reinforcement learning-based mapless navigation with predictive exploration worthiness. In 2023 IEEE International Conference on Mechatronics and Automation (ICMA) (pp. 636-643). IEEE.

[3] Gao, Y., Lin, F., Cai, B., Wu, J., Wei, C., Grech, R. and Ji, Z., 2024. Mapless navigation via hierarchical reinforcement learning with memory-decaying novelty. Robotics and Autonomous Systems.

[4] (Under review) Gao, Y., Wu, J. and Ji, Z., 2024. Deep reinforcement learning for localisability-aware mapless navigation. IET Cyber-systems and Robotics.

Chapter 1

Introduction

1.1 Motivation

A mobile robot is an intelligent system capable of autonomous movement on roads and in various terrains, operating in real time. It integrates a range of advanced capabilities, such as environment perception, dynamic decision-making, and behavioural control. While these features enable significant advancements in both military and civilian applications. In the military, mobile robots can undertake dangerous tasks such as demining (Rachkov *et al.*, 2005) and reconnaissance (Matthies *et al.*, 2002). In civilian settings, mobile robot technology is used in various environments, such as warehouse automation with Unmanned Ground Vehicles (UGVs) (Laber *et al.*, 2020) or Unmanned Aerial Vehicles (UAVs) for power line inspection (Xing *et al.*, 2023). In the research of mobile robot technology, navigation is the core focus of mobile robot research and is essential for enabling autonomous movement.

The goal of navigation is conceptually simple: to find an optimal path from the robot's current location to the destination while avoiding obstacles. A conventional mobile robot navigation system is explicitly divided into three components: a mapping and localisation module, a global path planner, and a local planner. The robot first needs to create a map of the environment using various mapping methods (Millonig and Schechtner, 2007; Durrant-Whyte and Bailey, 2006), then localises both the target location and its current position within the map (Rai *et al.*, 2012; Durrant-Whyte and Bailey, 2006). The global planner generates a feasible path based on different path planning approaches (Gasparetto *et al.*, 2015), and the local planner follows a sequence of waypoints proposed by the global planner. Thus, the entire navigation process heavily relies on an accurate map, leading to the conventional navigation method being referred to as the map-based method.

However, map-based navigation has several limitations, primarily related

to the reliance on maps. Firstly, constructing a map is time-consuming and labour-intensive. Secondly, maintaining and updating the map can be costly in the long term, especially with dynamic changes in the environment. Thirdly, there is a high dependence on precise sensors for mapping work. These limitations have been highlighted in previous studies (Xie, 2019; Tai *et al.*, 2017).

Mapless navigation is widely regarded as an approach that eliminates the need for a map in the navigation system. Mapless navigation refers to the task of finding a collision-free path for a mobile robot with only partial observations of the environment. This capability is crucial for applications in unstructured environments, such as urban search and rescue, disaster relief, and domestic service robots, where constructing an accurate map is challenging.

Nowadays, extensive research (Patle *et al.*, 2019) has been conducted on path planning for autonomous navigation systems, which involves determining the optimal route for a robot or vehicle to travel from a starting point to a destination while avoiding obstacles and ensuring efficiency. This field has seen significant advancements, particularly in addressing challenges such as planning under perception uncertainty (Kurniawati, 2022). However, conventional path planning algorithms typically rely on handcrafted heuristic functions—predefined rules or strategies designed to estimate the cost or feasibility of a path. While these heuristic functions can provide effective guidance in specific, well-understood environments, they are often tailored to particular scenarios, making them less adaptable to unforeseen conditions. Lack of generalisation means that significant effort must be invested to reconfigure these algorithms whenever they are deployed in a new context. As a result, there is a growing interest in developing more flexible and adaptive

path planning methods that can learn and generalise across diverse environments, reducing the dependency on handcrafted heuristics and enhancing the robustness and efficiency of navigation systems (Mac *et al.*, 2016; Qureshi and Yip, 2018).

As an emerging algorithm, deep reinforcement learning (DRL) has achieved great success in numerous tasks including video games (Mnih *et al.*, 2015), simulation control agents (Lillicrap *et al.*, 2015) and robotics (Kober *et al.*, 2013). Reinforcement learning (RL) involves an agent interacting with the environment and learning an optimal policy through trial and error. As a purely data-driven method, deep reinforcement learning (DRL) outperforms conventional approaches in several ways. Firstly, it eliminates the need for handcrafted rules, which, while effective in certain scenarios, may have been derived based on domain-specific assumptions and can sometimes offer limited adaptability. Many studies have shown that DRL-based methods can surpass the limitations of human strategies (Li, 2017). However, it is important to note that handcrafted rules may also come with rigorous theoretical guarantees that are difficult to achieve in DRL. While DRL offers flexibility and adaptability, its reliance on empirical learning means it may lack the same level of formal theoretical assurance that handcrafted approaches can provide. Secondly, DRL excels in learning from high-dimensional domains, such as sensory data or robot models—domains that are both information-theoretically and computationally challenging for heuristic methods. Thirdly, in contrast to other machine learning mechanisms, RL is more suited to problems with complex reward structures and a strong requirement for sequential interactions with the environment. Therefore, DRL has demonstrated great performance in addressing mapless navigation tasks (Tai *et al.*, 2017; Zhelo *et al.*, 2018). It enables the generation of appropriate navigation behaviours

for mobile robots based solely on sensory information, without the need for a map. The working principle of DRL-based mapless navigation is straightforward: reward the robot for reaching the target location and penalise it for collisions (Dobrevski and Skočaj, 2021).

However, due to long decision horizons and sparse rewards that can only be earned upon reaching the target location, the agents of most DRL-based mapless navigation methods (Tai *et al.*, 2017; Dobrevski and Skočaj, 2021) often get stuck in local regions, encountering the local minimum problem in complex environments. Hierarchical reinforcement learning (HRL)-based mapless navigation methods have shown promising performance in addressing this problem (Ding *et al.*, 2018; Wöhlke *et al.*, 2021). This is because HRL allows a robot to decompose a long-horizon navigation task into several intermediate destinations (subgoals), which are significantly easier to reach than the distant long-term goal. Based on HRL, a navigation system can be divided into two levels: a high-level (HL) policy that selects subgoals as short-term targets, and a low-level (LL) policy that moves the robot to these subgoals at the locomotion level.

1.2 Research Challenges and Problems

Although DRL/HRL-based methods have demonstrated remarkable performance, several challenges remain. This thesis aims to address the following problems.

One major challenge is the limitations of HRL in addressing mapless navigation effectively. The existing HRL-based methods contain three limitations. Firstly, humans have the ability to assess the worthiness of different subgoals for exploration, enabling them to make informed decisions when

selecting subgoals (Wolbers and Wiener, 2014). However, the HL policies of most methods lack this capability. These policies select subgoals solely based on sensory observations. Widely used sensors, such as Lidar, optical cameras, and RGBD sensors, provide high-dimensional raw measurement data that are inefficient for training RL agents for complex navigation tasks. Secondly, in most methods, the subgoal layout is defined as a 3×3 grid map centred around the robot, allowing for 8 subgoals in the neighbouring cells (Wöhlke *et al.*, 2021). This can restrict the robot's exploration. Few studies have examined the effects of different subgoal space layouts. Thirdly, many methods do not adapt the HL planning to the capabilities of the LL policy (Eppe *et al.*, 2019; Yamamoto *et al.*, 2018). In robot navigation, it is unrealistic to assume that the robot can always reach a subgoal selected by the HL policy.

Another significant challenge relates to the performance of HRL in complex and cluttered environments, where agents may still encounter issues such as getting stuck in local minima. The reasons can be summarised as follows: Firstly, most HRL-based mapless navigation methods use a simplified reward structure when training their HL policies, such as rewarding proximity to the goal and penalising collisions (Zhou *et al.*, 2019; Bischoff *et al.*, 2013; Staroverov *et al.*, 2020). Secondly, many methods rely on random exploration strategies, like ϵ-greedy, during training (Zhou *et al.*, 2019; Bischoff *et al.*, 2013). Thirdly, most agents lack a spatial memory mechanism, which can lead to revisiting the same locations multiple times. Thus, the agent is easily trapped in a local region.

A further issue in most DRL-based mapless navigation methods is the assumption that robots have access to accurate ground-truth poses, typically provided by Global Navigation Satellite System (GNSS) signals (Tai

et al., 2017; Zhelo *et al.*, 2018; Jang *et al.*, 2021). However, this assumption is unrealistic in the real world, as GNSS signals are not available everywhere. In GNSS-denied environments, Simultaneous localisation and Mapping (SLAM)-based localisation methods may be required to assist in robot self-localisation. However, the performance of these techniques is highly dependent on the environment. For instance, visual SLAM heavily relies on tracked visual features. Its performance may degrade in environments where features are poorly observed, such as in areas lacking distinct features. Ignoring localisation performance can lead to robots failing to accurately localise themselves, resulting in potential issues due to decision-making based on unreliable state estimation. Therefore, the author believes that the perception module for localisation should be closely integrated with decision-making in the path planning process.

1.3 Aim and Objectives

The primary aim of this project is to develop a reinforcement learning-based navigation method that can allow a robot to robustly navigate in cluttered environments and be deployed in the real world without assuming localisation availability.

Based on the research challenges, this project aims to achieve the following objectives:

- Propose an HRL-based mapless navigation framework to enhance performance in long-range navigation tasks in cluttered environments.
- Develop a solution to enhance the effectiveness of the high-level (HL) policy within the hierarchical reinforcement learning (HRL) framework for selecting appropriate subgoals.

- Investigate effective approaches to incorporate a novelty mechanism to encourage robots to explore unknown environments more efficiently, hence addressing the local minimum issue.

- Propose a solution to eliminate the reliance on the assumption of localisation availability by integrating visual SLAM into the mapless navigation framework.

- Optimise traversal trajectories using RL to ensure the perception of high-quality visual features, hence preventing deteriorated localisation performance.

1.4 Contributions

In accordance with these research objectives, this section summarises the contributions claimed in this thesis.

By pursuing objectives to address the limitations of HRL-based mapless navigation methods, this thesis makes the following contributions:

- Introduce a novel HRL-based mapless navigation framework using RGB images and Lidar scans. The high-level policy predicts the worthiness value of each subgoal based on RGB images and Lidar observations, then selects an appropriate subgoal. The low-level policy controls the robot to reach the subgoal by generating speed commands based on Lidar observations.

- Propose two metrics for evaluating the worthiness of each subgoal. The first is Predictive Neighbouring Space Scoring (PNSS), which relates to the free neighbouring space available at each subgoal. The second is

Predictive Exploration Worthiness (PEW), which considers obstacle spatial distribution, such as the area of free space and the distribution of obstacles around each subgoal.

- Develop the PNSS and PEW model to predict the respective values for each subgoal. The predicted PNSS or PEW values are then utilised as part of the HL policy inputs, providing a compact and informative representation.

- Propose a new HL reward function that includes penalties for selecting subgoals that are not achievable by the LL policy, resulting in more reliable subgoal selection.

- Design a novel subgoal layout to improve overall performance.

- Demonstrate the effectiveness of the proposed HRL-based method, showing that it outperforms state-of-the-art methods in terms of task success rates and generalisability.

By pursuing objectives to address the local minimum issue of HRL-based methods, this thesis makes the following contributions:

- Propose a new HL reward function that integrates two principal components: extrinsic and intrinsic rewards. The extrinsic reward guides the robot towards the target location. The intrinsic reward, informed by novelty theory (Ruan *et al.*, 2022), episode memory, and memory decay, fosters effective exploration of the environment.

- Design a novel neural network structure that incorporates an LSTM layer to provide the agent with a memory mechanism. The network takes a sequence of the four most recent states as input and outputs the Q-values for selecting the corresponding subgoal.

- Demonstrate the effectiveness of the proposed reward function and neural network (NN) structure, particularly in addressing the local minima issue.

By pursuing objectives to address the issue of most RL-based methods relying on the assumption of localisation availability, this thesis makes the following contributions:

- Develop a novel DRL-based mapless navigation framework that uses RGB-D-based ORB-SLAM2 for localisation, eliminating the assumption of available ground-truth robot pose.

- Propose a compact state representation derived from the spatial distributions of map points. It enables the robot to be aware of the distributions of tracked features, thereby encouraging navigation along paths that offer reliable localisation.

- Introduce a novel reward function that incorporates related pose error (RPE). This enables the policy to be more responsive to the individual action, providing finer-grained feedback on the impact of each action on overall performance.

- A dynamic threshold is introduced to assess the performance of SLAM, enhancing the policy's adaptability to variations in SLAM performance across different environments.

- Demonstrate the substantial improvements in localisation accuracy and navigation task success rates achieved by the proposed method.

In Chapter 4, a novel mapless navigation framework based on HRL is introduced, enabling the robot to evaluate the worthiness value of each subgoal, which significantly improved the success rate of long-range navigation

tasks. However, this approach still faced the issue of local minima in unstructured environments that might contain terrains like long corridors and dead corners. To address this, Chapter 5 proposes a method for training the high-level policy, including a new reward function and a novel neural network structure, both of which help mitigate the local minimum problem. While both Chapters 4 and 5 effectively tackle mapless navigation and the local minimum challenge, they still rely on the assumption of localisation availability. In response, Chapter 6 introduces an RL-based method that eliminates this assumption, further advancing the capabilities of the navigation system.

1.5 Outline

The remainder of this thesis is organised into six chapters, which are briefly outlined as follows.

Chapter 2 reviews the relevant literature on conventional map-based and modern DRL-based mapless navigation. It summarises the limitations of existing methods and discusses related approaches for addressing these challenges.

Chapter 3 provides an overview of key concepts in RL and highlights the important DRL algorithms used in this thesis. Additionally, it discusses techniques for reducing localisation errors when using self-localisation methods to determine the robot's pose during navigation.

Chapter 4 addresses the limitations of HRL-based mapless navigation methods, seeking to develop a more efficient HRL framework for long-horizon navigation tasks.

Chapter 5 focuses on resolving the local minimum issues in HRL-based methods within complex environments.

Chapter 6 is devoted to developing a DRL-based method that does not rely on the assumption of available robot pose.

Chapter 7 concludes the thesis by summarising the contributions and limitations of the works, and proposing future research directions.

Chapter 2

Literature review

In this chapter, a comprehensive review of related works on robot navigation methods is presented. Robot navigation is a classic problem in robotics with applications across numerous industries. Rather than providing an exhaustive review of this vast field, this chapter focuses specifically on the navigation of wheeled robots in indoor environments. These methods can be broadly classified into two categories: map-based methods and mapless methods.

This chapter is organised as follows: First, Section 2.1 outlines the formulation of map-based navigation methods, covering sensing, localisation, mapping, and path planning. Following this, Section 2.2 reviews recent works in mapless navigation, particularly those utilising deep reinforcement learning and discusses their limitations. Such studies often assume that robots have access to ground-truth poses, which is unrealistic in real world environments. Therefore, at the end of this chapter, methods for reducing localisation errors when employing various techniques to determine the robot pose are explored. Finally, Section 2.3 concludes this chapter.

2.1 Map-Based Navigation

Conventional navigation methods are widely recognised as map-based approaches. This is because the navigation system fundamentally relies on a geometric description of the environment, e.g. a map. The traditional navigation task can be decomposed into three sub-tasks: perception, planning, and control. Perception involves sensing the environment, representing it, and localising the robot, essentially allowing the robot to determine its own position. Planning aims to find the shortest path to the destination. Finally, the control module guides the robot to execute the planned actions, ensuring

it moves toward the target location while avoiding obstacles.

2.1.1 Sensing

Robots typically estimate their state by extracting geometric relationships between themselves and objects in the environment, relying on sensors. Directly measuring distance is the simplest sensing method, and the raw data can be used for both obstacle avoidance and map construction. Widely used sensors in robot navigation systems include Lidar (Kwon and Lee, 1999; Scott *et al.*, 2000), ultrasonic sensors (Moravec and Elfes, 1985; Elfes, 1987), and depth cameras (Moradi *et al.*, 2006; Cunha *et al.*, 2011), as shown in Fig. 2.1.

(a) (b) (c)

Figure 2.1: (a) Lidar (b) Ultrasonic sensor (c) Depth camera

Lidar technology utilises light pulses or laser beams to measure the distance between the robot and obstacles. When the beams of light hit an object, they reflect back to the sensor, enabling the calculation of distance based on the elapsed time. Ultrasonic ranging involves emitting pulses of acoustic energy toward the obstacle's surface and measuring the time it takes for the echo to return. The most widely used depth camera is the Microsoft Kinect (Zhang, 2012), developed by Microsoft Corporation. This device fea-

tures an infrared laser projection system and a monochrome CMOS sensor, allowing it to capture 3D video data in various lighting conditions.

As an alternative to ranging sensors, the visual sensor has a wide application in robot navigation. It can capture external information about objects in the environment (DeSouza and Kak, 2002). While a single image cannot provide geometric information, this can be derived from two images using internal and external camera parameters. Moreover, with the ubiquity of cameras and the increasing semantic information contained in images, the influence of visual sensors in robot navigation research is growing exponentially.

2.1.2 Localisation

In this section, pure localisation methods that do not involve constructing a map of the environment are explored. Then, Simultaneous Localisation and Mapping (SLAM) approaches are reviewed.

In many map-based robot navigation systems, the environment is described in a pre-defined metric map that is always available to the robot. Thus, understanding the robot's state within the environment requires only a localisation system to provide the robot's pose in the map's coordinate system.

This section reviews prior work on Indoor Positioning Systems (IPS), focusing on robot navigation in indoor environments. Visual localisation is popular in robotics due to its low cost and high accuracy (DeSouza and Kak, 2002). However, its bottlenecks include limited computational power and a lack of robustness to environmental changes, such as variations in lighting. Another frequently used sensor in infrastructure-free IPS is the range finder, which outperforms others in terms of accuracy but has the disadvantages of

high energy consumption and cost (Lingemann *et al.*, 2005). Besides these two types of infrastructure-free IPS, other systems may utilise magnetic field sensors (Wang *et al.*, 2016a) or inertial sensors (Xiao *et al.*, 2014), which often involve a significant tradeoff between manufacturing cost and measurement accuracy.

Simultaneous Localisation and Mapping (SLAM) (Durrant-Whyte and Bailey, 2006) shall be the most widely used method in constructing a map of the environment and localising the robot. SLAM refers to the process by which a robot builds a map of the environment while simultaneously determining its position within that map. Notably, SLAM allows the robot to estimate its motion trajectory and create the map in real-time, without any prior knowledge. This capability makes SLAM a crucial technology in the field of robot navigation.

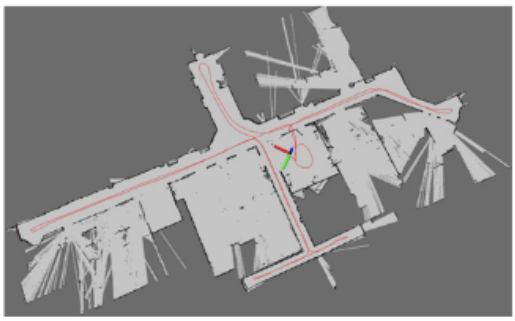
Generally, as the robot moves, its position changes and sensor observations are used to extract feature points from the environment. The robot then combines the positions of the currently observed feature points with its previous observations and movement data. This integration is often performed using an Extended Kalman Filter (EKF) to estimate the current position and update the environmental map. Since the robot's position information obtained through motion estimation has significant errors, it cannot rely solely on this method for accurate positioning. After using the robot's motion equation, it is necessary to use information from the surrounding environment to correct the robot's position. This process generally involves extracting environmental features and then re-observing these features' positions after the robot has moved.

There are various forms and solutions for SLAM, with probabilistic SLAM being widely accepted (Thrun *et al.*, 1998). At the theoretical level, proba-

bilistic SLAM can be addressed using several methods, such as the Kalman Filter (EKF-SLAM) (Bailey *et al.*, 2006), the Rao-Blackwellised Particle Filter (FastSLAM) (Montemerlo *et al.*, 2002), and Graph Theory (Graph-SLAM) (Thrun and Montemerlo, 2006). However, this is only at the theoretical level, and there are still many substantive problems when it is applied in a real environment. Firstly, it is computationally intensive and complex. It involves estimating joint states that encompass robot poses and the positions of stationary landmarks, where each landmark corresponds to a feature point on the map. As the robot's trajectory lengthens and the number of landmarks on the map increases, the size of the state vector for the robot grows linearly, and the covariance matrix grows quadratically. Secondly, dynamic environments significantly impact the practical implementation of SLAM. Assumptions in many SLAM algorithms, which typically assume a static environment, are often disrupted by pedestrians and movable objects. These disruptions can lead to conflicts during map updates through data associations. For instance, if the robot returns to a previously recorded position on the map, it may attempt to close the loop and reduce drift by matching current observations with the map. Significant changes in the environment at that location can cause the data association to fail, resulting in inconsistent maps.

2.1.3 Map Representation

The form of maps varies with the application of navigation and can be broadly categorised into metric maps and topological maps, as shown in Fig. 2.2.



(a)

(b)

Figure 2.2: (a) Metric map (b) Topological map (Beeson *et al.*, 2010)

Metric maps emphasise the precise representation of positional relationships between objects. They are typically classified as either sparse or dense. Sparse maps offer a degree of abstraction and do not need to represent all objects, while dense maps aim to capture detailed information about every object in the environment. People select representative features, known as landmarks, a sparse map is composed of these landmarks, with non-landmark areas being ignored (Meyer and Filliat, 2003). In contrast, dense maps aim to model everything in the environment. While sparse maps are often sufficient for localisation, dense maps are typically required for navigation (Meyer and Filliat, 2003; Filliat and Meyer, 2003). Dense maps are usually com-

posed of numerous small blocks, each representing a specific resolution. A two-dimensional metric map consists of numerous small grids, while a three-dimensional map is made up of many small volumes called voxels. Typically, each voxel can be in one of three states: occupied, free, or unknown, indicating whether or not there is an object within that voxel (Fairfield *et al.*, 2010).

Topological maps emphasise the relationships between map elements rather than the precise accuracy of metric maps. Essentially, a topological map is a graph consisting of nodes and edges that represent the connectivity between points (e.g., point A is connected to point B) rather than the specific process of travelling from A to B (Kostavelis and Gasteratos, 2015). This approach simplifies the map, making it a more compact representation. However, topological maps are less effective at representing complex structures. Key challenges remain in partitioning the map to create nodes and edges and in using topological maps for navigation and path planning.

2.1.4 Path Planning

Once the robot has localised itself and identified the target on the map, the next step is to find a path to the target. Path planning algorithms are utilised to generate collision-free paths with minimal cost. These algorithms can be categorised into offline and online methods based on whether the robot possesses a complete description of the environment in advance.

In offline algorithms, searching and creating the optimal path on a given map is straightforward because the complete environment map is available. Lozano-Perez et al. developed core methods in this field, introducing concepts like configuration space, visible graph, and cell decomposition in the early stages (Lozano-Pérez and Wesley, 1979; Lozano-Perez, 1990). These

methods are effective in simpler environments but can be computationally expensive in complex ones. As a result, optimisation algorithms such as genetic algorithms and particle swarm optimisation have been developed (Qin *et al.*, 2004; Piot *et al.*, 2016).

Online path planning algorithms are widely favoured due to their strong compatibility with SLAM, allowing robots to map environments as they traverse them. Early examples of online path planning algorithms include the artificial potential field method (Khatib, 1986) and collision cones (Chakravarthy and Ghose, 1998). Current online path planning algorithms have been improved, with more efficient approaches emerging, such as bionics (Mei *et al.*, 2006) and evolutionary algorithms (Vadakkepat *et al.*, 2000).

2.2 Mapless Navigation

As discussed above, the conventional robot navigation system heavily relies on the map of the environment. However, many research (Fan *et al.*, 2018; Tai *et al.*, 2017; Jin *et al.*, 2020) have pointed out some problems with using maps in robot navigation systems. Firstly, constructing an accurate map of the environment is time-consuming and labour-intensive. In addition, the control performance of conventional methods highly depends on the simplified mathematical models, which lead to unreliable navigation systems.

Mapless navigation is widely regarded as an approach that relieves the navigation system from the need of a map (Tai *et al.*, 2017; Zhelo *et al.*, 2018). Mapless navigation refers to the task of finding collision-free paths in situations where the mobile robot receives only local environmental information, without pre-constructed descriptions of the environment. The difference between map-based and mapless navigation is summarised in Table. 2.2.

Metric	Map-Based Navigation	Mapless Navigation
Application Scenarios	Autonomous vehicles (self-driving cars), GPS navigation (smartphones, cars, hiking), Indoor navigation (shopping malls, airports)	Robots in unknown or dynamic environments, Search and rescue operations
Required Prior Knowledge	Precise map data (2D or 3D), Detailed road, building, or terrain information	Minimal prior environmental knowledge, Real-time sensor data, algorithms for localisation
Real-time Adaptation	Limited to map updates; slower reaction to unexpected events, Requires external inputs for dynamic changes (e.g., traffic)	High adaptability to changing environments without pre-built maps, Continuously adapts using sensors and AI algorithms
Infrastructure Dependence	Heavy reliance on pre-existing infrastructure (maps)	Minimal infrastructure required (works in uncharted spaces)
Cost	Can be expensive due to high-quality maps and updates, Continuous costs for map updates	Can be less costly in terms of infrastructure, Requires high computational resources for real-time processing

Metric	Map-Based Navigation	Mapless Navigation
Reliability	High in well-mapped areas	Highly reliable in unpredictable environments, but can struggle with accuracy in some cases
Scalability	Scalable for large areas (city-wide, country-wide)	Typically suited for smaller or dynamic areas, but scalable with advancements
Computational Complexity	Relatively lower complexity once map is available	Higher complexity due to real-time processing of sensor data
Robustness to Environmental Changes	Less robust to sudden environmental changes (e.g., road closures, unexpected obstacles)	More robust, as it reacts to real-time environmental changes
Data Updating Frequency	Updates typically require manual or automated map revisions	Continuous and real-time updates as the system explores
Energy Efficiency	Can be more energy-efficient (especially in known environments)	Energy-consuming due to continuous sensor processing and decision-making

Deep reinforcement learning-based methods are currently receiving the most attention in the field of mapless navigation. With the powerful rep-

resentation learning capabilities, DRL-based methods enable the learning of control policies directly from raw sensory inputs, bypassing all intermediate steps of the conventional methods. Therefore, this thesis mainly focuses on DRL-based mapless navigation methods.

2.2.1 DRL-based Mapless Navigation

Tai et al. (Tai *et al.*, 2017) address the challenge of designing suitable movement behaviours for mobile robots when no prior map information is available. They propose a mapless navigation system based on DRL. The mobile robot in this system is equipped with a ranging sensor. This system utilises a transformation equation. The equation takes three inputs: a 10-dimensional local Lidar observation, the 2-dimensional relative position of the target, and the robot's 2-dimensional velocity from the previous time step. The output is a 2-dimensional velocity representing the robot's current action. In addition, they apply asynchronous DRL to train the model. Unlike traditional DRL methods, asynchronous DRL separates the training sample collection into a different thread. This approach enables multiple sample collection threads, enhancing efficiency and performance. The reward function is that a positive value will be assigned to the agent when reaching the target location and the agent will be penalised when colliding with an obstacle. In other cases, the reward is the change of distance from the robot to the target location between two consecutive timesteps.

A number of related works have adopted a similar method and reward function to address mapless navigation tasks. Dobrevski et al. (Dobrevski and Skočaj, 2021) present a novel local navigation method for guiding robots to global goals without an explicit environmental map. The proposed model, trained using a DRL framework based on the advantage Actor-Critic (A2C)

method, directly translates robot range-scan observations into action commands. Grando et al. (Grando *et al.*, 2020) introduce a DRL-based system designed for goal-oriented mapless navigation of Unmanned Aerial Vehicles (UAVs). Their approach leverages localisation data and sparse range data to train the intelligent agent efficiently.

In RL-based mapless navigation, the robot's sensors encompass not only range sensors but also visual sensors (Zhu *et al.*, 2017; Shao *et al.*, 2018). Zhu et al. (Zhu *et al.*, 2017) explore the hypothesis of using purely visual input for target-driven mapless navigation. They propose an actor-critic model whose policy is a function of the goal as well as the current observed image. They aim for their model to exhibit high adaptability and flexibility, eliminating the need for repeated training when the target image changes. Consequently, the model requires an understanding of the relative spatial relationship between the current and target positions, as well as an awareness of the overall scene layout. To capture this intuition, they develop a Deep Siamese Actor-Critic Network.

The method involves inferring the spatial relationship between the current position and the target by projecting the two input images into a shared embedded space, preserving their geometric relationship. The proposed network is a two-stream model designed for discriminative embedded learning. They use two Siamese layers with shared weights to map the current state and target into the same embedded space, then fuse these embeddings to form a joint representation. They also introduce a scene-specific layer to capture particular scene features, such as room layout and object arrangement, which processes the joint representation. The model then generates policies and value estimates similar to the A2C model. In their method, all scenarios share a common generic Siamese layer, while targets within a scenario use

the same scene-specific layer. This enhances the model's generalisation as the target image changes. Regarding the reward function, they provide a reward of 10.0 upon completing the task of reaching the goal. Additionally, to incentivise shorter trajectories, they introduce a small time penalty of -0.01 as an immediate reward.

In addition, many other DRL-based mapless navigation models using visual information as input demonstrate great performance. For example, Shao et al. (Shao *et al.*, 2018) propose an efficient DRL-based method to address visual navigation tasks. Specifically, they present the synchronous A2C with the generalised advantage estimator (GAE) algorithm. Gupta et al. (Gupta *et al.*, 2017) train a model using real-space scans. Mirowski et al. (Mirowski *et al.*, 2018) utilise a high-quality simulated environment to perform inter-city mapless navigation without deploying a real robot or using city maps.

2.2.2 Limitations

Recent research on DRL-based mapless navigation has demonstrated impressive performance in handling navigation tasks without relying on predefined maps. However, they only address those problems under limited situations. For example, the environments in which Tai Lei's method is trained and tested are simple, Gazebo-created environments with no complex obstacles (Tai *et al.*, 2017). This section discusses the current challenges facing most mapless navigation methods and explores how related work is addressing these issues.

Low Training Efficiency

Most methods (Tai *et al.*, 2017; Luong and Pham, 2021) highlight that various applications based on DRL must contend with the uncertainties of robot hardware and real-world environments. Consequently, numerous trials are necessary to enhance the stability of robot performance. Reducing the number of training iterations and improving training efficiency has thus become a significant challenge.

To address this issue, many theoretical (Wang *et al.*, 2016b; Schaul *et al.*, 2015b) and systematic (Mnih *et al.*, 2016) methods have been introduced. These methods improve training efficiency by optimising the learning mechanism itself, without relying on any additional information.

In addition, numerous works try to combine other learning techniques with DRL algorithms to address the issue of low training efficiency. Luong et al.(Luong and Pham, 2021) propose a DRL-based mapless navigation system with incremental learning. They employ an on-policy learning approach to effectively train the navigation policies. The traditional actor-critic algorithm struggles with sampling efficiency, and the Advantage Actor-Critic (A2C) method is not suitable for online learning models. To address these issues, Luong et al. propose an improved A2C model that incorporates the Kronecker-Factored Trust Region (ACKTR) (Wu *et al.*, 2017). This enhancement optimises the loss function by using K-factor and trust region adjustments within the A2C framework. In this model, the actor and critic networks share two fully consistent connection layers, reducing the number of training parameters and improving training efficiency.

Recently, many studies have also combined evolutionary strategies with DRL, called evolutionary reinforcement learning (ERL) (Such *et al.*, 2017; Bodnar *et al.*, 2020; Khadka and Tumer, 2018). Conventional RL often relies

on extensive sampling and trial-and-error to improve policy performance, particularly in high-dimensional spaces or with sparse rewards, leading to lower sample efficiency. However, ERL utilises evolutionary algorithms to evaluate multiple individuals in each generation, leveraging parallel computation to enhance sample efficiency. Genetic operations such as crossover and mutation can quickly generate new strategies, speeding up the exploration of the solution space.

Marchesini et al. (Marchesini and Farinelli, 2020b) propose a hybrid algorithm called Genetic Deep Reinforcement Learning (Genetic DRL), which is on the basis of ERL. This approach combines the strengths of traditional genetic algorithms and DRL. In Genetic DRL, the agent initially stores its experience from interacting with the environment in a replay buffer. Periodically, it generates a population of children, each with a distinct genome created by a mutation function. A separate thread and a copy of the training environment are instantiated for each child. The mutation function's weights are used to create these individuals. The children are then evaluated in their respective training environments using a copy of the mutation function's weights. The best-performing child replaces the current agent, thereby improving the overall training process. Genetic DRL leverages DRL for efficient sampling and uses the genetic algorithm to evaluate and discover better strategies. This combination enhances training stability, reduces training time, and improves overall training efficiency.

On the other hand, Marchesini et al. (Marchesini and Farinelli, 2020a) find that mapless navigation models with discrete state spaces require significantly less training time compared to those with continuous state spaces. They argue that continuous state space-based algorithms (such as DDPG, PPO) can address issues that discrete state space-based algorithms cannot,

such as sampling in a continuous high-dimensional action space. However, DDPG and PPO are still challenged by longer training time compared to discrete action space-based algorithms.

Additionally, learning from demonstrations (or imitation learning) may be another choice to speed up the training (Duan *et al.*, 2017). It leverages both the interactive learning pattern in RL for robustness and the efficient learning signal in supervised learning by forcing the network to mimic expert behaviour. However, it is limited by the need for expert controllers, which are often difficult to establish.

Local Minimum Problem

The current work of DRL-based mapless navigation has shown promising performance in simple environments. However, a common feature of these methods (Tai *et al.*, 2017; Marchesini and Farinelli, 2020a; Mnih *et al.*, 2016) is their reliance on either random exploration strategies, such as ϵ-greedy, or state-independent exploration by maximising the entropy of the policy. Also, some methods utilise sparse rewards (Shao *et al.*, 2018), which are only granted upon reaching the target location. As a result, when testing these methods in complex environments, the agent tends to be stuck in local regions, i.e, the local minimum problem.

Nowadays, some methods aim to solve this issue through novel reward functions. Pathak et al. (Pathak *et al.*, 2017) introduce the concept of 'internal curiosity' to the reward function. They argue that an agent's ability to predict the consequences of its actions can be used to assess the novelty of states, or the intrinsic curiosity. Therefore, they develop an Internal Curiosity Module (ICM) that predicts the next state based on the current state and action. The error between the predicted state and the actual state serves as

the intrinsic reward. Pathak et al. demonstrate that if intrinsic rewards are effectively trained through supervised learning, the agent can be motivated to explore the environment more thoroughly. The intrinsic reward encourages the robot to explore the environment and encounter new states. it helps guide the robot away from local minima or premature convergence, enabling the agent to better utilise learned action strategies to complete the current task.

Inspired by Pathak et al.'s work, Zhelo et al. (Zhelo *et al.*, 2018) propose a reward function which incorporates extrinsic and intrinsic rewards. The extrinsic reward is similar to those of settings in other mapless navigation models. It is the difference in the distance from the target compared with the last timestep. This motivates the robot to get closer to the target position. Regarding the intrinsic reward, they propose to use the internal motivation of agents measured by curiosity to encourage the agents to explore new environments, so as to obtain new states, therefore, the exploration performance and the generalisation ability of the agents in an unknown environment are improved. The results show that exploratory behaviours due to curiosity help escape local minima of the policy function. Similarly, Khan et al. (Khan *et al.*, 2018) use internal agent signals as auxiliary representations to promote environmental exploration. They develop an agent that utilises micro-memorable and self-monitoring states, along with reward and action predictions.

In addition to 'curiosity,' researchers have incorporated 'novelty' into reinforcement learning (RL) methods. Novelty, a motivation for animals to spontaneously explore their environment, means that the animal can reward itself for encountering something new (Sutton, 1990; Ruan *et al.*, 2022; Fu *et al.*, 2017). In simple terms, in RL, a state that has been frequently vis-

ited by the agent is considered to have low novelty. Based on this, Ruan et al. (Ruan *et al.*, 2022) calculate the novelty of a state using classical count-based methods. When the agent encounters a state, it retrieves its memory to calculate how many times it has previously visited that state. This count is then used to adjust the reward: a higher count results in a smaller reward, and a lower count results in a larger reward. This method encourages the agent to explore more novel states.

Bellemare et al. (Bellemare *et al.*, 2016) propose a pseudo-count method that generalises count-based exploration to non-tabular cases, improving agent exploration efficiency in difficult games. Ostrovski et al. (Ostrovski *et al.*, 2017) enhance this method by replacing the model providing the pseudo-count with PixelCNN, demonstrating excellent exploration performance. Moreover, they find that the mixed Monte Carlo update significantly facilitates exploration.

Except for intrinsic motivation, several methods incorporate curriculum learning into RL to improve data efficiency in sparse reward RL (Klink *et al.*, 2020; Florensa *et al.*, 2017; Wöhlke *et al.*, 2020). Florensa et al. (Florensa *et al.*, 2017) propose Reverse Curriculum Generation algorithm for RL. In their method, the robot is trained in 'reverse'. Initially, the robot is trained to reach the goal from the start states that are close to the goal state. Building on this foundation, the robot is subsequently trained to solve the task from increasingly distant start states. They propose a method that automatically generates a curriculum of start states that adapts to the agent's performance, leading to efficient training on goal-oriented tasks.

Struggling in Long-Range Tasks

Although many studies have demonstrated great performance (Tai *et al.*, 2017; Marchesini and Farinelli, 2020a), these methods have not been tested with long-range tasks in challenging environments. Consequently, there is reason to suspect that these models might struggle to learn effective navigation strategies in more complex training scenarios. Long-range tasks would be more challenging, often involving scenarios with complex local maps that tend to lead to local minima, making it more difficult to generate suitable policies.

Hierarchical Reinforcement Learning (HRL) receives increasing attention as it is well suited for long-horizon and complex tasks (Levy *et al.*, 2017; Yang *et al.*, 2021; Nachum *et al.*, 2018). HRL methods typically design high-level (HL) policies that operate on a coarser time scale and control the execution of low-level (LL) policies. Recent HRL methods, such as Option Critic (Bacon *et al.*, 2017), Feudal Networks (FuN) (Vezhnevets *et al.*, 2017), and HiRO Networks (Nachum *et al.*, 2018), combine the concepts of subgoal generation and policy combination with neural networks and reinforcement learning. In FuN, Vezhnevets et al. employ a Manager module (HL policy) and a Worker module (LL policy). The Manager operates at a lower temporal resolution, setting abstract goals for the Worker to enact. The Worker generates primitive actions at each tick of the environment. This decoupled structure facilitates long-timescale credit assignment and encourages the emergence of sub-policies associated with different goals set by the Manager. These properties enable FuN to significantly outperform a strong baseline agent on tasks requiring long-term credit assignment.

In general, the HL policy can be trained to iteratively predict sequences of intermediate subgoals, which are then used as targets for the LL pol-

icy (Nachum *et al.*, 2018; Gupta *et al.*, 2020; Nair and Finn, 2019). Alternatively, some methods generate subgoal sequences using a divide-and-conquer approach (Jurgenson *et al.*, 2020; Parascandolo *et al.*, 2020).

HRL methods also can be categorised based on HL representations, such as symbolic representation (Yamamoto *et al.*, 2018), predicate-logic-based representation (Eppe *et al.*, 2019), and learned skills (Sharma *et al.*, 2019). Additionally, some methods do not set the representation form for the HL model in advance, and connect the experience in the buffer to a graph to form the representation through the accessibility estimation network (Eysenbach *et al.*, 2019).

Nowadays, many studies have proven that HRL-based navigation methods can perform well in complex navigation tasks (Ding *et al.*, 2018; Wöhlke *et al.*, 2021; Zhou *et al.*, 2019). This is because HRL enables a robot to decompose a long-horizon navigation task into several intermediate destinations (subgoals), which are much easier to reach than the long-term distant goal. A navigation system can be decomposed into two levels, where a high-level (HL) planning policy selects subgoals as short-term targets, and a low-level (LL) policy moves the robot to these short-term targets at the locomotion level. Such a hierarchical way is analogous to human navigation, therefore, hierarchical methods tend to be more interpretable than non-hierarchical methods (Epstein *et al.*, 2017).

Zhou et al. (Zhou *et al.*, 2019) utilises an HRL-based method to control the robot based on a topological map. The HL policy examines the nodes of the map and learns a policy over these nodes to select the subgoal for the LL policy to achieve. The LL policy takes in the subgoal and produces an action to reach the selected subgoal. In their work, the actions are defined to control the robot's moving direction and include 8 discrete components. The

reward function for the HL policy is set as that the agent receives a positive reward when it reaches the target and a penalty when a collision is detected. Additionally, to avoid unnecessary subgoal switches and encourage shorter trajectories to the goal, the agent also receives a small penalty for each state change at the high level. Regarding the Ll reward function, the agent receives a reward when the subgoal is reached and a penalty when collisions occur. Also, to promote faster subgoal reaching, a penalty is assigned to the agent for each moving step.

Wohlke et al. (Wöhlke *et al.*, 2021) propose a HRL-based navigation framework. The subgoal space is set as a 3×3 grid centred around the robot, which allows 8 subgoals in the neighbouring cells. The HL policy selects a sub-goal from the set of neighbours based on a rough map. This policy generally operates at a lower frequency than the LL state space transition. The LL policy is pursuing the subgoal. Similarly, Zhu et al. (Zhu *et al.*, 2021) employ the navigation tasks in two layers, where the higher layer decomposes the trajectory in discrete primitives corresponding to the decomposed grids, and the lower layer deals with movement control.

Furthermore, some methods try to replace the DRL-based LL model with conventional navigation methods. Xue et al. (Xue and Chen, 2023) combine DRL-based methods with classical navigation methods for UAV navigation. They argue that the trajectory generated by DRL-based end-to-end methods cannot be as smooth as traditional planning algorithms. Additionally, their control curves tend to be relatively rigid, increasing energy consumption. In their method, the classical motion planning algorithm and DRL algorithm are combined using a hierarchical structure. At the high level, they train a DRL-based model to generate a suitable subgoal. The reward function consists of two parts: a collision penalty and the difference in distance from

the target compared to the last step. Then, the conventional method, EGO-planner, controls the UAV to move to the subgoal.

Although HRL-based methods have demonstrated good performance, they have some limitations. However, some HRL methods do not consider the LL policy performance in HL planning (Eppe *et al.*, 2019; Yamamoto *et al.*, 2018). Additionally, some existing HRL-based navigation methods (Wöhlke *et al.*, 2021; Bischoff *et al.*, 2013) require prior knowledge, such as a grid map of the environment, which is often unrealistic. Moreover, the HL policies of many HRL methods select subgoals based on unrepresentative features or high-dimensional redundant data (Levy *et al.*, 2017; Nachum *et al.*, 2018). Moreover, the simplicity of the subgoal space layout (Wöhlke *et al.*, 2021) may impede performance.

In addition, some methods also use a hierarchical structure, but instead of selecting a subgoal, the HL model provides guidance for the LL model. Ding et al. (Ding *et al.*, 2018) propose a HRL-based navigation system for multi-agents, called Hierarchical Navigation Reinforcement Network (HNRN). In the HL policy, they train a Hidden Markov Model (HMM) to evaluate the agent's perception. Then, this HMM gives a hazard coefficient, which is used to control the LL decision-making process. In the LL policy, two sub-systems are introduced, a differential target-driven system and a collision avoidance DRL system. In the target-driven sub-task, they use the position and angle difference between the target position and the current position to drive the agent, which is optimal in the absence of obstacles. For the collision avoidance system, they train a DRL model for obstacle avoidance. The reward function for the collision avoidance model contains only two components: the coverage of the $2D$ laser and the penalty incurred after a collision. The reward function involves only a single task, significantly

reducing the training difficulty. The advantage of this hierarchical structure is that it decouples the target-driven and collision avoidance tasks, leading to a faster and more stable model. Experiments indicate that their method has higher learning efficiency and success rates.

Dey et al. (Dey *et al.*, 2023) develop a hybrid method that switches between complementary navigation strategies based on an HL planner trained with DRL using a dense reward function, specifically the geodesic distance to the goal. They distribute navigation decisions between two different methods: a DRL-based planner that takes RGB-D first-person input and a classical planner that takes a metric occupancy map as input. The HL planner exploits regularities between scene elements and planning performance, learning to make binary decisions between these two methods based solely on first-person inputs.

Xie et al. (Xie *et al.*, 2020) propose a visual navigation system based on DRL, called SnapNav. SnapNav only receives the currently observed image and outputs a specific action according to directional guidance. SnapNav also contains a hierarchical structure, a HL Commander and a LL Controller. The Commander provides command information for every image, but the command database only stores images of specific areas where the robot needs to change direction or stop. There are three primary instructions: turn right, turn left, and stop. The default action is to continue moving forward. At each timestep, the robot matches its current image with those in the commander's dataset to determine the appropriate instruction. The LL controllers are trained using DRL, where the input is the current RGBD observation, and the output is the corresponding action.

Lacking Memory Mechanisms

Another limitation of most methods (Tai *et al.*, 2017; Zhelo *et al.*, 2018; Jin *et al.*, 2020) is that the agents lack the memory mechanism. When humans perform navigation tasks, they avoid revisiting places that do not lead to the target location because these unavailable places are in their memory. However, the robots do not have memory mechanisms, so they may revisit those already visited areas, including the paths that have proven infeasible for the navigation task and will be trapped in a local region. In addition, the real environment often includes dynamic elements, necessitating that agents maintain continuous long-term memory. Most vision-based mapless navigation models (Zhu *et al.*, 2017) without memory mechanisms are not well-suited for long-range navigation in large indoor environments.

To enable agents with memory units, most methods combine the long short-term memory (LSTM) (Gers *et al.*, 2000) network with RL. LSTM is a type of recurrent neural network (RNN) architecture designed to address the limitations of conventional RNNs in learning long-term dependencies. LSTM networks utilise gating mechanisms to address the limitation and are effective for various time series (Yu *et al.*, 2019).

Hausknecht et al. (Hausknecht and Stone, 2015) propose to combine LSTM with DQN algorithm (Mnih *et al.*, 2015), called DRQN. In specific, they replace the first post-convolutional fully-connected layer with a recurrent LSTM to enable the agent to memorise past states. Similarly, Mnih et al. (Mnih *et al.*, 2016) propose an asynchronous RL method and utilise it to train a model combined with an LSTM network for addressing the navigation tasks in 3D mazes. Singla et al. (Singla *et al.*, 2019) propose a memory-based DRL method for avoiding collisions in indoor environments, with an LSTM network processing partial information acquired previously.

Hu et al. (Hu *et al.*, 2021) present a sim-to-real pipeline for DRL-based autonomous robot navigation in cluttered rough terrain, where an A3C-based policy is adopted (Mnih *et al.*, 2016). In the work, LSTM is employed to capture information from previous states. Mirowski et al. (Mirowski *et al.*, 2016) construct a stacked LSTM framework demonstrating enhanced data efficiency and task performance through the incorporation of supplementary navigation-related signals.

In addition to incorporating an LSTM network, some methods encode the environment with episode memory (Pritzel *et al.*, 2017; Ruan *et al.*, 2022). During an episode, Ruan et al. (Ruan *et al.*, 2022) save the observations at each tiemstep, creating an episode memory. Then they compare the current observation and those in memory to access the novelty of the state. However, the size of the episode memory for the agent grows linearly with the states explored by the agent. This can become problematic when the agent needs to accomplish a task in large environments.

2.2.3 Reducing Localisation Errors

Most mapless navigation studies usually assume that robots have access to their ground-truth poses (Tai *et al.*, 2017; Zhelo *et al.*, 2018; Marchesini and Farinelli, 2020a) and focus on path planning only. However, this assumption is unrealistic. In real environments, the robot needs to utilise localisation methods to obtain its pose. Localisation performance is highly dependent on the environment. For example, techniques of visual SLAM highly rely on the tracked visual features for global navigation satellite system (GNSS)-denied environments; while, for outdoor, GNSS-based localisation is prone to multipath reflections of radio signals caused by the environment. The ignorance of localisation performance would lead to robots failing to localise

themselves and possible catastrophic issues caused by decision-making based on unreliable state estimation. This thesis believes the perception module for localisation should be closely coupled with decision-making in the process of path planning. This section reviews various approaches aimed at reducing localisation errors during indoor navigation.

Non-Learning-Based Methods

SLAM is a widely used method for localising a robot within a map. However, pose estimates from the SLAM system can drift during navigation when they encounter environments with few features. To correct this drift, loop closures are necessary (Mur-Artal *et al.*, 2015; Deshpande *et al.*, 2023). Many methods utilise an Active Loop Closure (ALC) module to navigate the robot back to known areas, thereby achieving loop closures and reducing uncertainty (Deshpande *et al.*, 2023; Stachniss *et al.*, 2004; Lehner *et al.*, 2017). However, most methods use wide FoV sensors such as 360 degrees FoV Lidar or range finders, or use large compute modules (Lehner *et al.*, 2017; Stachniss *et al.*, 2004).

However, forming a loop is challenging for narrow-FoV visual SLAM. Deshpande et al. (Deshpande *et al.*, 2023) propose a lighthouse-based method for reducing the localisation error in narrow-FoV visual SLAM. The lighthouse is the place that makes the robot rotate in place. This operation emulates a panoramic view which is equivalent to having a 360 degrees view at the point. When the uncertainty is high, they make the robot travel to a lighthouse and perform an in-place rotation. This approach significantly increases the likelihood of loop closure, as the panoramic view captures a broader range of features dispersed over the local environment.

Learning-Based Methods

Many studies also utilise DRL to reduce the localisation error. Naveed et al. (Naveed *et al.*, 2022) train a model to predict whether an action leads to localisation failure in visual SLAM. Specifically, the simulator will provide an action which moves the robot to the target location without considering the issue of localisation. They utilise the DQN algorithm to evaluate the recommended action based on the current observed image. If the Q-value of the recommended action exceeds a certain threshold, the recommended action will be considered safe, and the robot will execute it. If the Q-value of the recommended action is below the threshold, the robot chooses actions randomly. The reward function penalises the robot if SLAM fails and assigns a reward based on the number of overlapping points. Naveed et al. believe that overlapping points between two consecutive images is a strong indicator of a successful localisation operation.

Prasad et al. present an approach closely related to Naveed et al.'s work (Prasad *et al.*, 2018), proposing a Q-learning-based method to avoid tracking failures in visual SLAM by utilising handcrafted features, such as the number of tracked features. In contrast, Naveed et al. employ DQN to directly extract information from raw images. While these DRL-based methods effectively prevent localisation failures, they focus primarily on detecting potential SLAM failures, without addressing the broader navigation-related tasks. Additionally, both approaches rely on pre-built path planning to guide the robot's movement (Naveed *et al.*, 2022; Prasad *et al.*, 2018).

However, several DRL-based mapless navigation approaches go beyond this by integrating SLAM or similar techniques to estimate the robot's pose at each timestep. These methods aim not only to move the robot toward its target but also to minimise localisation errors. For instance, Lin et al.

(Lin *et al.*, 2022) develop a DRL-based mapless navigation method that uses Hector SLAM (Kohlbrecher *et al.*, 2011) to track the robot's pose. They hypothesise that the estimated covariance in state estimation can serve as a measure of localisation performance, which can then be incorporated into the reward function. Their reward structure includes both conventional rewards (Tai *et al.*, 2017; Marchesini and Farinelli, 2020a) and a penalty when the estimated covariance exceeds a threshold, ultimately improving localisation accuracy and supporting successful task completion.

Similarly, Chen et al. (Chen *et al.*, 2023b) introduce a DRL-based, localisability enhanced navigation method specifically designed for dynamic human environments. They utilise Adaptive Monte Carlo Localisation (AMCL) for localisation and propose a novel reward function that encourages higher localisation accuracy. In their method, the reward is calculated as the difference between the estimated pose errors at two consecutive timesteps, penalising the agent if this error exceeds a predefined threshold. Like Lin et al.'s method, their reward function combines traditional navigation rewards with a focus on minimising localisation error. The resulting navigation policy not only moves the robot towards its destination but also chooses a path that enhances localisation quality. This approach leads to improvements in both the lost rate and arrival rate during testing.

2.3 Summary

This chapter provides a thorough overview of traditional map-based robot navigation methods, followed by an in-depth exploration of emerging mapless navigation techniques based on deep reinforcement learning. Additionally, this chapter highlights various strategies for minimising localisation errors,

a crucial aspect for enhancing localisation accuracy. By reviewing both conventional and cutting-edge approaches, this chapter offers a clearer understanding of the evolution of navigation technologies and the methodologies currently shaping the field of robotics.

Chapter 3

Preliminary

This chapter provides a comprehensive overview of the foundational concepts that underpin the research presented in this thesis. This chapter is organised as follows: It begins by introducing key concepts within the reinforcement learning (RL) framework (Section 3.1), followed by a discussion of typical RL methods (Section 3.2). Also, goal-conditioned reinforcement learning is discussed (Section 3.3). Section 3.4 then explores two prominent practical deep reinforcement learning (DRL) algorithms: Deep Q-Network (DQN) (Mnih *et al.*, 2015) and Deep Deterministic Policy Gradient (DDPG) (Lillicrap *et al.*, 2016), both of which are utilised in the research. These concepts are crucial for understanding the DRL-based navigation methods developed in this thesis, which will be detailed in the following chapters. Finally, Section 3.5 delves into visual Simultaneous Localisation and Mapping (SLAM) methods, with a particular focus on ORB-SLAM (Mur-Artal and Tardós, 2017), which is employed to provide robot pose estimation in the proposed work. Section 3.6 concludes this chapter.

3.1 Reinforcement Learning

Reinforcement Learning (RL) is a branch of machine learning that studies how to learn from interactions, considered the most natural learning experience for humans. RL was inspired by the observation that humans or some animals can learn certain behaviours associated with some form of reward signals. RL is akin to a baby that starts with no knowledge but learns the 'right' actions through exploration and guidance (rewards or punishments). The entity that learns behaviours in RL is called an agent.

In the learning process, the agent needs to explore its environment by taking some actions to discover the consequences of those actions. RL meth-

ods store the experiences of the agent's actions and the consequences in a buffer, using this information for learning. The agent is expected to exploit what it has learned about its actions to earn more rewards, thereby increasing its confidence in taking 'good' actions. This process gives rise to the term 'reinforcement' (Sutton and Barto, 2018). In the past, the method of parameterised distributions was always used to store the knowledge learned by the agents (Engel *et al.*, 2005). With the widespread use of DL, the method has been replaced by deep neural networks, which is where deep reinforcement learning comes from.

In the following, a mathematical theoretical formulation of RL is presented. This section first outlines the theoretical framework of an RL problem, followed by a discussion of key elements.

3.1.1 Markov Decision Process

In the theoretical formulation, the RL problem can be formulated as a MDP. A MDP is a sequential decision-making process where an agent chooses actions based on feedback from the environment (i.e., states and rewards) in a sequential fashion (Wiering and Van Otterlo, 2012). It means that each decision depends on the state and action outcomes from the previous step, forming a continuous sequence of decisions. MDP is based on an important assumption that each state satisfies the Markov property. A system satisfies the Markov property if the future states are independent of what happened before the current state. The Markov property states that the future state is independent of the past given the current state. In specific, there is a sequence of states, s_0, s_1, s_2,..., s_T in the period $[0-T]$. The state s_t $(t \in [0, T])$ has the Markov property if

$$P(s_{t+1}|s_t) = P(s_{t+1}|s_0, s_1, s_2, ..., s_t) \tag{3.1}$$

where P is the transition probability function that describes the distribution of the next state given the current and previous states. Based on this property, a MDP can be considered as a memory-less and random process.

Formally, an MDP can be represented by a tuple $\langle S, A, R, P, \gamma, \rho_0 \rangle$ (Wiering and Van Otterlo, 2012). S is the state space, A represents the action space, $R(s, a)$ denotes the reward function, $P(s'|s, a)$ is the system transition model, γ represents the discount factor and ρ_0 is the initial state distribution. According to the initial state s_0 and the transition function P, we can obtain a sequence of states, s_0, s_1, s_2,..., s_T. This sequence is called an episode. The reward function R is used to evaluate the episode by accumulating the instantaneous reward at each state transition as $\sum_{\tau=1}^{T} r_\tau$.

3.1.2 Key Elements and Concepts

The essential elements of an RL problem are illustrated in Fig. 3.1. In the period $[0 - T]$, at timestep t, the agent selects an action a_t based on the current state s_t. After executing the action a_t, the robot receives a reward r_{t+1} ($r_{t+1} = R(s_t, a_t)$, R denotes the reward function) from the environment and then gets the next state s_{t+1}. It is worth noting that the action taken at each timestep may affect all future states, actions, and rewards. Therefore, the reward r_{t+1} the agent receives cannot be the only judge of the 'goodness' of the action a_t.

The goal of reinforcement learning is to maximise discounted accumulative future rewards G_t:

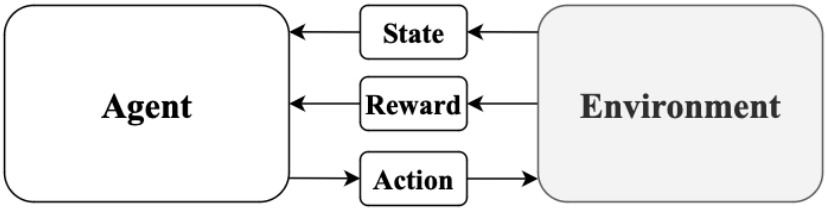


Figure 3.1: In RL, the agent observes the state of the environment and performs an action at each time step. Based on the action taken, the environment provides a numerical reward as feedback to the agent, evaluating whether the action aligns with the desired behaviour.

$$G_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+2} \dots \gamma^{T-t} r_T = \sum_{\tau=t+1}^{T} \gamma^{\tau-t-1} r_\tau \tag{3.2}$$

where γ represents the discount factor. The discount factor is used to reduce the value of future rewards and to avoid infinite rewards. Eq. 3.2 means that the agent needs to infer the optimal sequence of actions based on the current state.

In the following, some important concepts in RL are introduced. The policy π denotes the strategy the agent utilises to select action a_t based on the current state s_t. It can be represented as a stochastic distribution of actions given a state $\pi(a_t|s_t) : \mathcal{S} \times \mathcal{A} \to [0, 1]$, or a deterministic function that maps a state to an action $a_t = \pi(s_t) : \mathcal{S} \to \mathcal{A}$, where $a_t \in A$, A is the action space. The action space can be either continuous or discrete. A discrete action space means that the entire space contains a finite number of actions from which the agent can choose. In contrast, a continuous action space contains an infinite number of possible actions. In most cases, discrete action space-based models are easier to train. However, to enable more flexible manoeuvring motions in complex and diverse environments, continuous action space would be more desired.

The state-value function is the expected sum of discounted reward in future, as formulated below:

$$V_\pi(s_t) = \mathbb{E}_\pi[G_t|s_t] \tag{3.3}$$

$V_\pi(s_t)$ is defined given a specific policy π and state s_t.

The action-value function (also called the Q-value function) is similar to the state-value function. However, the difference is that the Q-value function is further defined on the action as an additional parameter:

$$Q_\pi(s_t, a_t) = \mathbb{E}_\pi[G_t|s_t, a_t] \tag{3.4}$$

The relation between the state value and the action value:

$$V_\pi(s_t) = \sum_{a \in \mathcal{A}} Q_\pi(s_t, a_t)\pi(a_t|s_t) \tag{3.5}$$

$V_\pi(s_t)$ can be considered as the evaluation of the policy π in state s_t. While $Q_\pi(s_t, a_t)$ further measures the 'goodness' of taking a particular action a_t in state s_t.

3.2 Reinforcement Learning Methods

3.2.1 Model-Free & Model-Based

RL methods can roughly be divided into two categories, model-free methods and model-based methods (Wiering and Van Otterlo, 2012). In reinforcement learning, the term 'model' refers to the agent's understanding or representation of the environment's dynamics. Specifically, a model predicts the next state and the reward that will result from taking a particular action in a given state. In model-based reinforcement learning, the agent builds or has

access to such a model, which it can use to simulate future states and rewards, enabling it to plan actions more effectively and optimise its policy before interacting with the environment. In contrast, model-free methods do not require an explicit model of the environment; instead, they learn directly from interactions with the environment, using trial and error to estimate the value of actions and states. Therefore, in model-free methods, the agent relies on experience to update its policy, whereas in model-based methods, the agent leverages a model to predict and plan ahead.

Regardless of whether the Markov property (Eq. 3.1) is satisfied, as long as such a model exists, it is possible to find a path that meets a specific requirement given an initial state. If the model is deterministic, the transition probability is definite. Then, we can know the probabilities corresponding to $s \rightarrow s'$ and the action a that would have the highest reward in state s. This way, $V_\pi(s)$ and $Q_\pi(s, a)$ can be estimated accurately. This is the model-based method.

However, in most scenarios, the model is unknown or not precisely quantified. Model-free methods aim to provide a more generalised approach. For example, Q-Learning (to be discussed below) performs strategy learning by continuously solving the $Q_\pi(s, a)$, without relying on a model based on statistical results for planning. It estimates the $Q_\pi(s, a)$ of each 'cell' in the table for modelling and solving. With the recent advancements in deep neural networks, model-free RL methods have demonstrated effectiveness in high-dimensional state spaces. However, developing a model of the real-world environment in such a high-dimensional state space is computationally intensive. Therefore, this thesis mainly focuses on the model-free methods. The model-free RL methods fall into three categories: value-based, policy-based and actor-critic methods. In the following, these three methods are

explored.

3.2.2 Value-based Methods

Value-based methods aim to estimate the value or Q-values accurately. With the known sum of future rewards, determining the policy becomes straightforward: choose the action that maximises the future reward. The classic value-based method shall be the SARSA and the Q-learning method (Watkins and Dayan, 1992).

SARSA

SARSA is the acronym for the transition tuple (s, a, r, s', a'). Review the Q-value function first:

$$Q_\pi(s_t, a_t) = \mathbb{E}_\pi[G_t | s_t, a_t] \tag{3.6}$$

then, based on Bellman equations, the following recursive property of the Q function holds:

$$\begin{aligned} Q_\pi(s_t, a_t) &= \mathbb{E}[G_t \mid s_t, a_t] \\ &= \mathbb{E}[r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots \mid s_t, a_t] \\ &= \mathbb{E}[r_{t+1} + \gamma(r_{t+2} + \gamma r_{t+3} + \dots) \mid s_t, a_t] \\ &= \mathbb{E}[r_{t+1} + \gamma G_{t+1} | s_t, a_t] \\ &= \mathbb{E}[r_{t+1} + \gamma Q_\pi(s_{t+1}, a_{t+1}) | s_t, a_t] \end{aligned} \tag{3.7}$$

The SARSA method mainly utilises the Temporal Difference (TD) method to update the Q-value.

$$Q(s, a) \leftarrow Q(s, a) + \alpha(r + \gamma Q(s', a') - Q(s, a)) \tag{3.8}$$

where $r + \gamma Q(s', a') - Q(s, a)$ is the TD error. The SARSA method updates the Q-value at every single collected transition after each timestep. The SARSA has many benefits: i) It does not require the environment to be an MDP or model-based. The valuation of the SARSA is derived from its own attempts; ii) The SARSA's valuation feedback is relatively fast and can be updated after each step taken.

On-policy & Off-policy

Before exploring the Q-learning method, the terms on-policy and off-policy should be discussed. On-policy and off-policy generally refer to whether the policy that generates the data is the same as the one that is to be updated using the data. From this point of view, the SARSA method is a typical on-policy algorithm. On-policy methods are considered simpler to understand and faster to converge. However, the drawback of on-policy methods is obvious, that is, it makes transitions very relevant, as a 'poor' choice at one step can have far-reaching consequences for many future steps.

Off-policy methods also have many advantages. Firstly, the process of learning the optimal policy can be distinct from the process of exploring the environment. Secondly, to be more suitable for real-world problems, operating off-policy enables the agent to learn from diverse data sources, including handcrafted controllers or human demonstrators.

Q-Learning

The Q-learning method is considered the off-policy version of the SARSA. The update rule of Q-learning can be obtained by changing one operation in SARSA:

$$Q(s,a) \leftarrow Q(s,a) + \alpha(r + \gamma \underset{a'}{max} Q(s',a') - Q(s,a)) \tag{3.9}$$

In Q-learning, it takes the maximum q-value when computing the estimate of the target. This method is straightforward: the Q-value of the current state is determined by which action achieves the highest value in the state it transitions into.

Q-learning has many strengths: i) Due to the 'max' operation, it is easier to determine the optimal path and the best policy in simple environments. ii) The implementation is straightforward: only the transition (s, a, r, s') needs to be logged and updated step by step. Additionally, its logic is easier to understand. However, it has some limitations: i) It is challenging to implement scenarios where the action space is continuous. Not only does the Q-learning algorithm suffer from this problem, but so does the SARSA. ii) The value of taking an action in one state depends entirely on the action that yields the highest value in its subsequent state. This contradicts the original definition and may lead to 'overestimation'.

3.2.3 Policy-based Methods

Policy-based approaches optimise the policy function π directly. A primary approach involves estimating the gradients of the policy function with respect to the total rewards. By following these gradients, the policy can be improved to achieve greater rewards. In summary, the policy gradient algorithms aim to model and optimise the policy directly.

The to-be-optimised policy is denoted as $\pi_\theta(a \mid s)$. It is parameterised with θ and must be differentiable. The objective function utilised in optimising is formulated as follows:

$$
\begin{aligned}
J(\theta) &= V^{\pi_\theta}(s_0) \\
&= \sum_{a \in \mathcal{A}} \pi_\theta(a|s) Q^{\pi_\theta}(s,a)
\end{aligned}
\tag{3.10}
$$

It represents the cumulative reward for the episodic case from the start state.

To find the parameter θ that maximises the objective function Eq. 3.10, the iterative optimisation algorithm gradient ascent can be used. This algorithm updates the parameter by taking the gradient of the objective function:

$$
\theta_{i+1} = \theta_i + \alpha \nabla J(\theta_i) \tag{3.11}
$$

where i and $i+1$ denotes the update step; α represents the learning rate. The gradient indicates the direction in which the parameter should change to improve the expected returns.

The episodic tasks have a finite number of interaction steps. According to the policy gradient theorem (Sutton *et al.*, 1999; Silver *et al.*, 2014; Lillicrap *et al.*, 2015), the gradient with respect to the policy parameters in the episodic case can be written as:

$$
\begin{aligned}
\nabla J(\boldsymbol{\theta}) &= \nabla v^{\pi_{\boldsymbol{\theta}}}(s_0) \\
&= \sum_s d^{\pi_{\boldsymbol{\theta}}}(s) \sum_a q^{\pi_{\boldsymbol{\theta}}}(s,a) \nabla \pi_{\boldsymbol{\theta}}(a|s) \qquad (3.12) \\
&= \sum_s \sum_{k=0}^{\infty} \gamma^k Pr(s_0 \to s, k, \pi) \sum_a q^{\pi_{\boldsymbol{\theta}}}(s,a) \nabla \pi_{\boldsymbol{\theta}}(a|s) \\
&= \mathbb{E}_\pi \left[\gamma^t \sum_a q^{\pi_{\boldsymbol{\theta}}}(s,a) \nabla \pi_{\boldsymbol{\theta}}(a|s) \right] \\
&= \mathbb{E}_\pi \left[\gamma^t \sum_a q^{\pi_{\boldsymbol{\theta}}}(s,a) \pi_{\boldsymbol{\theta}}(a|s) \frac{\nabla \pi_{\boldsymbol{\theta}}(a|s)}{\pi_{\boldsymbol{\theta}}(a|s)} \right] \qquad (3.13)
\end{aligned}
$$

where $\sum_a q^{\pi_{\boldsymbol{\theta}}}(s,a) \nabla \pi_{\boldsymbol{\theta}}(a|s)$ indicates the state value represented by the summed action values weighted by the gradients of the probabilities over all actions.

$\sum_s d^{\pi_{\boldsymbol{\theta}}}(s)$ is the sum of state values of visited states weighted by the expected number of timesteps required to reach the state s according to the policy $\pi_{\boldsymbol{\theta}}$. In simple terms, Eq. 3.12 increases the probability of selecting an action if that action results in a high expected discounted future return (the fewer steps required the better), and decreases it otherwise.

REINFORCE

The most classic policy-based approach is REINFORCE (Williams, 1992). The most straightforward way of calculating q^π to estimate Eq. 3.13 is to approximate it with the actual return collected from an interaction trajectory. This approach leads to the REINFORCE algorithm, which is essentially a Monte Carlo policy gradient method. Specifically, the gradient for the maximisation objective can be formulated as:

$$\begin{aligned}
\nabla J(\boldsymbol{\theta}) &= \mathbb{E}_\pi \left[\gamma^t \sum_a q^{\pi_{\boldsymbol{\theta}}}(S_t, a) \pi_{\boldsymbol{\theta}}(a|S_t) \frac{\nabla \pi_{\boldsymbol{\theta}}(a|S_t)}{\pi_{\boldsymbol{\theta}}(a|S_t)} \right] \\
&= \mathbb{E}_\pi \left[\gamma^t q^{\pi_{\boldsymbol{\theta}}}(S_t, A_t) \frac{\nabla \pi_{\boldsymbol{\theta}}(A_t|S_t)}{\pi_{\boldsymbol{\theta}}(A_t|S_t)} \right] \text{ (using sample} A_t) \\
&= \mathbb{E}_\pi \left[\gamma^t G_t \frac{\nabla \pi_{\boldsymbol{\theta}}(A_t|S_t)}{\pi_{\boldsymbol{\theta}}(A_t|S_t)} \right] \text{ (using Monte-Carlo estimate of } q^{\pi_{\boldsymbol{\theta}}}) \\
&= \mathbb{E}_\pi \left[\gamma^t G_t \nabla \log \pi_{\boldsymbol{\theta}}(A_t|S_t) \right] \text{ (using } \nabla \log x = \frac{\nabla x}{x})
\end{aligned} \tag{3.14}$$

The REINFORCE update rule is moving the weights of a policy towards the higher return direction with respect to its parameters at iteration i:

$$\theta_{i+1} = \theta_i + \alpha \gamma^t G_t \nabla \log \pi_{\theta_i}(A_t|S_t) \tag{3.15}$$

When the REINFORCE converges, it directly provides a distribution of the optimal policy. Sampling actions from this distribution is expected to

maximise the expected discounted return. It is worth noting that REINFORCE only updates after a whole episode is finished.

3.2.4 Actor-Critic Methods

The third category of model-free RL methods is the actor-critic method. 'actor' is a policy that is responsible for selecting an action based on the environment, and then the value function 'critic' criticises the actions taken by the 'actor' to provide the gradients for optimising the policy using the policy gradient theorem. Therefore, the actor-critic method is based on both the value function and the policy function.

Both the critic function and the policy function are iteratively updated during the learning process. The critic function is adjusted to provide a more accurate estimation of values or Q-values. Subsequently, the policy function is optimised based on the guidance provided by the critic function, enabling it to learn better actions in various scenarios. Gradients computed from the critic function tend to be more stable compared to those in policy-based methods. This is because critic gradients are derived from expected values estimated by the critic, whereas policy gradients are typically estimated from the total rewards obtained from sampled trajectories. The gradient is formulated as follows:

$$\nabla_\theta J(\theta) = \nabla V_\theta(s_0) \propto \mathbb{E}_{\pi_\theta}[Q^{\pi_\theta}(s,a)\nabla_\theta \log \pi_\theta(a|s)] \tag{3.16}$$

With predictions from the approximation function, the policy can be updated with every transition tuple, eliminating the need to wait for a complete trajectory as required by the REINFORCE algorithm. This characteristic allows for more frequent and incremental updates to the policy during learning.

3.3 Goal-conditioned Reinforcement Learning

Goal-conditioned Reinforcement Learning (GRL) differs from traditional RL. The GRL problem adds a goal space G to the MDPs of the standard RL paradigm. Standard RL is to achieve a single goal, while a GRL agent tries to maximise a multi-goal reward function $R(s_t, a_t, g)$, resulting in a goal-conditioned value function $Q^{\pi}(s, g, a)$ or policy $\pi(a|s, g)$ (Schaul *et al.*, 2015a). The GRL optimises the return based not only on a single-objective reward function that depends on states and/or actions but also on a goal vector. This multi-objective reward function aims to optimise the return with respect to achieving specific goals, in addition to traditional reward considerations. As a result, the input to the value functions and policy is augmented by an extra term: the goal. The learned policy is expected to exhibit different behaviours in the same state depending on the assigned goals, enabling a certain level of information or knowledge sharing.

3.4 Deep Reinforcement Learning Algorithms

With the rise of DL and its breathtaking achievements, people combine DL with RL, that is, DRL. Both the tasks of value estimation and policy search benefit from deep neural networks, which are theoretically capable of approximating any function. In this section, two DRL algorithms utilised in the research are introduced: Deep Q Network (DQN) and Deep Deterministic Policy Gradient (DDPG).

Deep Q Network (DQN)

DQN is a typical value-based method that aims to estimate the Q-values. DQN can be seen as an extension of the traditional Q-learning algorithm,

designed to address its limitations in handling high-dimensional state spaces. While Q-learning relies on a tabular representation of the Q-value function, which is impractical for large or continuous state spaces, DQN uses deep neural networks to approximate the Q-function. This allows DQN to handle more complex environments, such as those with pixel-based input, by leveraging the power of deep learning for function approximation. In essence, DQN is a modernised version of Q-learning, combining its foundational principles with deep learning techniques to solve more challenging reinforcement learning problems.

Review the definition of the Q-value function:

$$Q_\pi(s_t, a_t) = \mathbb{E}_\pi[G_t | s_t, a_t] \tag{3.17}$$

The Q-value function can be computed using the Bellman equation:

$$Q_\pi(s_t, a_t) = \mathbb{E}_\pi[r_{t+1} + \gamma \mathbb{E}_\pi Q(s_{t+1}, a_{t+1}) \mid s_t, a_t] \tag{3.18}$$

The goal of RL algorithms is to obtain an optimal policy that maximises the value function. The optimal Q-value function, $Q^*(s_t, a_t)$, can be formulated as below:

$$Q^*(s_t, a_t) = \mathbb{E}[r_{t+1} + \gamma \underset{a_{t+1}}{max} Q(s_{t+1}, a_{t+1}) \mid s_t, a_t] \tag{3.19}$$

It indicates that the optimal Q-value at time t is the current reward r_{t+1} plus the discounted optimal Q-value at time $t+1$. This approach avoids the need to compute the Q-value directly over a large state space.

DQN aims to train an NN parameterised with θ^Q to estimate $Q(s_t, a_t)$. In the training process, the estimator $Q(s_t, a_t \mid \theta^Q)$ updates θ^Q using $(s_t, a_t, r_{t+1}, s_{t+1})$. The loss function is:

$$\mathcal{L}(\theta^Q) = [(r_{t+1} + \gamma \underset{a_{t+1}}{max} Q(s_{t+1}, a_{t+1} \mid \theta^Q)) - Q(s_t, a_t \mid \theta^Q)]^2 \tag{3.20}$$

Therefore, DQN is only suitable for discrete action space as it is impossible to compute the Q-value of an infinite number of actions.

Nowadays, many methods based on DQN have been proposed to enhance the performance of the vanilla DQN. In the following, four such methods are introduced.

Double DQN (DDQN) (Van Hasselt *et al.*, 2016) In the vanilla DQN, the policy utilises the same Q-value function to evaluate and select an action. This leads to overestimation (Van Hasselt *et al.*, 2016). The valuation of a state should be the mathematical expectation (i.e., the weighted average) of the valuations of all the actions in that state. However, if the maximum value is taken every time, the difference between this maximum and the weighted average creates an error, which leads to overestimation. DDQN aims to solve this problem by proposing a target network parameterised with $\theta^{Q'}$. When updating the network, the loss function is:

$$\mathcal{L}(\theta^{Q'}) = [(r_{t+1} + \gamma \underset{a_{t+1}}{max} Q(s_{t+1}, a_{t+1} \mid \theta^{Q'})) - Q(s_t, a_t \mid \theta^Q)]^2 \tag{3.21}$$

The target network slowly tracks the original Q-network and evaluates all actions in the discrete action space, while the greedy action selection is still made by the original Q-network.

Dueling DQN (Wang *et al.*, 2016b) Wang et al. (Wang *et al.*, 2016b) believe that estimating all actions at every state may be unnecessary, therefore, they propose Dueling DQN. The dueling network takes the form of providing two separate valuation functions: one for the state value and one for the

action advantages independent of the state. This decouples the value of the action itself from the value of the state.

The above branch maps to an output called $V_\pi(s)$, which is the state valuation function. In simpler terms, this function estimates how much a state is worth on its own. The following branch maps to $A_\pi(s, a)$, which is known as the state-independent action advantage function. Dueling DQN assumes the following relationship between the two:

$$A_\pi(s, a) + V_\pi(s) = Q_\pi(s, a) \tag{3.22}$$

The dueling network directly implements this assumption. The final layer outputs $Q_\pi(s, a)$ by directly combining the two values. This network features two branches shared by the front network, while the back network determines its own parameters, represented by θ^α and θ^β separately.

$$A_\pi(s, a \mid \theta^\alpha) + V_\pi(s \mid \theta^\beta) = Q_\pi(s, a \mid \theta^\alpha, \theta^\beta) \tag{3.23}$$

In the final implementation, the following formulation is utilised:

$$Q\left(s_t, a_t \mid \theta^\alpha, \theta^\beta\right) = V\left(s_t \mid \theta^\beta\right) + \left[A\left(s_t, a_t \mid \theta^\alpha\right) - \frac{1}{|\mathscr{A}|} \sum_{a'} A\left(s_t, a'_t \mid \theta^\alpha\right)\right] \tag{3.24}$$

where $\mathscr{A}$ represents the action space and $a' \in \mathscr{A}$.

Prioritised Experience Replay Prioritised Experience Replay is a method that allows Q-Learning to learn more efficiently (Schaul *et al.*, 2015b). It addresses the problem of low convergence efficiency in the traditional Q-Learning algorithm. DQN stores the experienced transitions in a replay buffer and uniformly samples a batch of transitions for training. In (Schaul *et al.*, 2015b), the authors believe that for tasks that have been learned and

completed, there is no need to revisit them. Instead, it is better to focus on those areas with larger errors. Therefore, the transitions selected from the memory buffer are typically not equally weighted; transitions with larger errors are prioritised for updating.

Deep Recurrent Q-Learning (DRQN) (Hausknecht and Stone, 2015) DRQN is mainly solving the Partially Observable MDP problem (POMDP). In a POMDP, the agent cannot directly observe the state and can only observe the transition probability. In such a partially observable environment, the optimal policy needs to rely on information from the previous historical trajectory. Hausknecht et al. (Hausknecht and Stone, 2015) propose to replace the first post-convolutional fully-connected layer with a recurrent LSTM to enable the agent to memorise past states.

Deep Deterministic Policy Gradient (DDPG)

As discussed above, DQN is suitable for discrete action spaces. In the following sections, a continuous-space-based method, DDPG, is discussed. DDPG is a modern, model-free reinforcement learning algorithm that combines the principles of actor-critic methods with deep learning techniques to handle continuous action spaces. Traditional actor-critic methods consist of two components: an actor, which directly determines the policy, and a critic, which evaluates the action-value function (Q-value). While standard actor-critic methods are typically used for discrete action spaces, DDPG extends this framework by using deterministic policies and deep neural networks to approximate both the actor and critic. This makes DDPG particularly suited for environments where actions are continuous, as it learns a deterministic policy and uses function approximation to scale to complex, high-dimensional tasks. Thus, DDPG can be considered an advanced variant of actor-critic

methods, specifically designed for continuous action spaces with deep reinforcement learning. In an actor-critic mode, the actor is an independent model (parameterised with θ^{μ}), whose task is to select actions. The critic is also an independent model (parameterised with θ^{Q}), that aims to learn the value estimation. In DDPG, the actor network and critic network comprise the main network. There are two other target networks ($\theta^{\mu'}$ and $\theta^{Q'}$), corresponding to the actor (θ^{μ}) and critic network θ^{Q} respectively.

The specific training process is as follows: Firstly, at time t the actor network in the main network selects an action a_t based on the state s_t, $a_t = \pi(s_t \mid \theta^{\mu})$. However, in DDPG, noise (N_t) is typically added to the action during this selection process, $a_t = \pi(s_t \mid \theta^{\mu}) + N_t$. The purpose of adding noise is to broaden the range of values and increase the exploration in the action outputs. Secondly, after obtaining the transition $(s_t, a_t, r_{t+1}, s_{t+1})$ through the exploration, it is stored in the Replay Buffer. It is important to note that at this stage, the role of the actor network (θ^{μ})in the main network is complete and it will not perform any further actions. The next step involves performing the update computations. The update progress is mainly based on the following three formulations:

$$y_i = r_{i+1} + \gamma Q'(s_{i+1}, \pi(s_{i+1} \mid \theta^{\mu'}) \mid \theta^{Q'}) \tag{3.25}$$

$$\mathcal{L} = \frac{1}{N} \sum_{i} \left(y_i - Q\left(s_i, a_i \mid \theta^{Q}\right)\right)^2 \tag{3.26}$$

$$\nabla_{\theta^{\mu}} J \approx \frac{1}{N} \sum_{i} \nabla_a Q(s, a \mid \theta^{Q}) \mid_{s=s_i, a=\pi(s_i \mid \theta^{\mu})} \nabla_{\theta^{\mu}} \pi(s \mid \theta^{\mu}) \mid_{s_i} \tag{3.27}$$

y_i in Eq. 3.25 represents the target value. It denotes the process of iteratively updating the value estimator using a temporal-difference methodology.

It is worth noting that in Eq. 3.25, the action is selected by the actor network in the target network. Then, (s, a) is evaluated by the target critic network.

In Eq. 3.26, $\mathcal{L}$ is the loss function of the critic network in the main network, specifically the Mean Squared Error (MSE) Loss. y_i is obtained by Eq. 3.25. N indicates the number of samples selected from the Replay Buffer for this update step.

In Eq. 3.27, $\nabla_{\theta^\mu} J$ denotes the amount of updates when updating θ^μ in the main network, that is, $\theta^\mu = \theta^\mu - \nabla_{\theta^\mu} J$. As for the latter part, the outer refers to averaging, and the inner is associated with the chain rule, i.e.,

$$\frac{\partial J\left(\theta^\mu\right)}{\partial \theta^\mu} = E_s\left[\frac{\partial Q\left(s, a \mid \theta^Q\right)}{\partial a} \frac{\partial \mu\left(s \mid \theta^\mu\right)}{\partial \theta^\mu}\right] \tag{3.28}$$

When updating the target network, the following methods are utilised:

$$\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'} \tag{3.29}$$

$$\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'} \tag{3.30}$$

where τ is a number much less than 1, and in some experiments, it will be set to 0.001. That is, instead of directly assigning θ^Q to $\theta^{Q'}$, the value of $\theta^{Q'}$ is retained to a large extent and then slowly updated in the direction of θ^Q.

3.5 Simultaneous Localisation and Mapping

This section reviews the Simultaneous Localisation and Mapping (SLAM) algorithms primarily utilised in my works, focusing particularly on ORB-SLAM, which is one of the visual SLAM methods.

SLAM refers to the process in which a subject equipped with specific sensors constructs a model of the environment while in motion and estimates

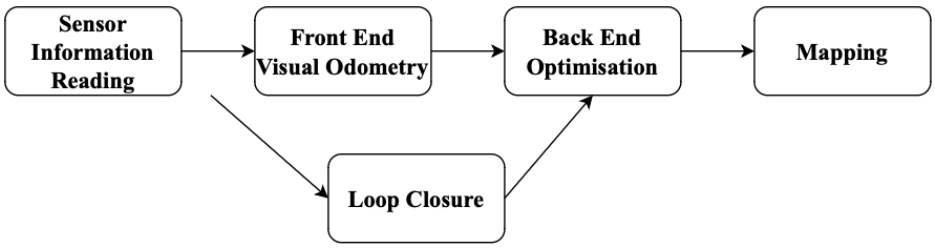


Figure 3.2: Overall visual SLAM framework.

its own motion without prior information about the environment (Durrant-Whyte and Bailey, 2006). When cameras are the primary sensors used, it is referred to as 'visual SLAM'.

3.5.1 Visual SLAM

The goal of visual SLAM is to achieve simultaneous localisation and mapping using images or visual data (Taketomi *et al.*, 2017). The framework of classic visual-SLAM methods is shown in Fig. 3.2.

1. Sensor information reading: In visual SLAM, this primarily entails reading and preprocessing camera images. For robots, this process may also include reading and synchronising data from encoders, inertial sensors, and other types of sensors depending on the application requirements.

2. Visual Odometry (VO): VO, also referred to as the Front End, focuses on estimating camera motion and local map structure from sequential images.

3. Back End Optimisation: The back end processes the camera positions measured by the visual odometry at different moments, along with information from loop closure detection, and optimises them to achieve

a globally consistent trajectory and map. It is referred to as the back end because it follows the VO.

4. Loop Closure: Loop Closure Detection determines whether the robot has returned to a previously visited position. When a loop is detected, this information is forwarded to the back end for further processing.

5. Mapping: It builds a map that meets the task requirements by using the estimated trajectory.

This is the basic framework of a visual SLAM system. Theoretically, the framework consists of five main modules: initialisation, tracking, mapping, relocalisation, and global map optimisation.

To start visual SLAM, it is necessary to define a specific coordinate system for camera pose estimation and 3D reconstruction in an unknown environment. During initialisation, the global coordinate system should first be defined, and a portion of the environment should be reconstructed as an initial map with respect to this global coordinate system.

Following the initialisation phase, tracking and mapping processes are undertaken to continuously estimate the camera's poses. During the tracking phase, the the reconstructed map is tracked in the image to determine the camera's pose relative to the map. To achieve this, 2D–3D correspondences between the image and the map are initially obtained through feature matching or feature tracking within the image. Subsequently, the camera pose is determined from these correspondences by solving the Perspective-n-Point (PnP) problem (Nistér and Stewénius, 2007). It is worth noting that most visual SLAM algorithms assume that the intrinsic camera parameters are available beforehand. Therefore, a camera pose typically corresponds to the extrinsic camera parameters, which include the translation and rotation of

the camera within the global coordinate system.

Relocalisation is executed when tracking fails due to rapid camera motion or other cases. Then, it is essential to recompute the camera pose. The last module is global map optimisation. It is understandable that the map accumulates estimation errors with the distance of the camera's movement. To suppress the errors, global map optimisation is typically performed. During this process, the map is refined by ensuring the consistency of the entire map information. When the map is revisited and a starting region is captured again after some time, reference information representing the cumulative error from the beginning to the present can be computed. This reference information is then used to establish a loop constraint, which serves to suppress errors during global optimisation. It should be noted that loop closure detection can be conducted using similar techniques employed in relocalisation.

Loop closure is always used to acquire such reference information. In loop closure, a closed loop is identified by matching the current image with previous images. If a loop is detected, the cumulative error that occurred during the camera's movement can be estimated.

Pose-graph optimisation is employed to mitigate accumulated errors by optimising camera poses (Grisetti *et al.*, 2010; Kümmerle *et al.*, 2011). In pose-graph optimisation, the relationships between camera poses are represented as a graph, and a consistent graph is constructed to mitigate errors. Bundle adjustment (BA) is utilised to minimise the reprojection error by optimising both the map itself and the poses of the cameras.

3.5.2 ORB-SLAM

To the best of the author's knowledge, ORB-SLAM (Mur-Artal *et al.*, 2015) is the most complete visual slam system. ORB-SLAM includes multi-threaded tracking, mapping, and closed-loop detection, with the map optimised using pose-graph optimisation and BA, making it an all-in-one package. Since ORB-SLAM is an open-source project, this comprehensive visual system can be easily utilised in my works. ORB-SLAM is not only a monocular-based SLAM system but has also been extended to support stereo visual SLAM and RGB-D visual SLAM. In this thesis, the RGB-D-based ORB-SLAM2 is primarily utilised.

One of the primary design principles of ORB-SLAM is the utilisation of the same features for mapping, tracking, and place recognition. In ORB-SLAM, they choose ORB feature (Rublee *et al.*, 2011), which are oriented multiscale FAST corners paired with a 256-bit descriptor. These features are extremely fast to compute and match while offering good invariance to viewpoint changes. This capability enhances the accuracy of BA.

ORB-SLAM2 operates with three primary parallel threads:

1. Tracking: The tracking thread localises the camera with every frame by identifying feature matches to the local map and minimising the reprojection error using motion-only BA.

2. Local Mapping: The local mapping thread manages and optimises the local map by performing local bundle adjustment (BA).

3. Loop Closing: The loop closing thread detects loops and corrects accumulated errors by performing pose-graph optimisation. After optimisation, it initiates another thread to execute full BA to compute the optimal structure and motion solution.

Another important concept in visual SLAM is the keyframe. Since the camera collects a large amount of image data, storing all of it is impractical. Continuous operation would lead to unacceptably high memory usage. Therefore, the main approach is to selectively save frames considered important (keyframe), under the assumption that the camera's trajectory can be accurately represented by these keyframes.

In this thesis, my work is more concerned with robot localisation based on ORB-SLAM2, and this will be further discussed in Chapter 7. In OBR-SLAM2, robust localisation is realised through reprojection optimisation. The reprojection error of map points to corresponding matched key points is represented as:

$$e_{i,j} = x_{i,j} - \pi_i(T_{iw}, X_{w,j}) \tag{3.31}$$

where $e_{i,j}$ is the reprojection error of a map point j in a keyframe i, $x_{i,j}$ represents the matched keypoint, $T_{iw} \in SE(3)$ is the pose of keyframe i, $X_{w,j} \in \mathbb{R}^3$ represents the 3D pose of map point j, and π_i denotes the projection function.

Then, to obtain the camera pose, a cost function needs to be minimised:

$$C = \sum_{i,j} (\rho_h(e_{i,j}^T \omega_{i,j}^{-1} e_{i,j})) \tag{3.32}$$

where ρ_h represents the Huber robust cost function, and $\omega_{i,j}$ is the covariance matrix linked to the scale at which the keypoint was identified.

3.6 Conclusion

This chapter introduces key concepts within the reinforcement learning framework and discusses typical RL methods. Building on this foundational understanding, two major DRL algorithms utilised in this thesis, DQN and DDPG,

are presented. The DQN algorithm is used to train the high-level policy of the hierarchical model in Chapters 4 and 5. The DDPG algorithm, another key applied method, primarily trains the low-level policy of the hierarchical model in Chapters 4 and 5, as well as the navigation method in Chapter 6. In addition, Chapter 6 introduces a navigation method that uses the ORB-SLAM algorithm to provide the robot pose. This chapter also provides an overview of the fundamental working principles of ORB-SLAM, explaining how it tracks the robot's movement and generates accurate localisation information, which is essential for the navigation process.

Chapter 4

Efficient Hierarchical Reinforcement Learning for Mapless Navigation with Predictive Neighbouring Space Scoring or Predictive Exploration Worthiness

4.1 Introduction

Solving reinforcement learning (RL)-based mapless navigation tasks is challenging due to their sparse reward and long decision horizon nature. Hierarchical reinforcement learning (HRL) has the ability to leverage knowledge at different abstract levels and is thus preferred in complex mapless navigation tasks.

However, designing an HRL framework for mapless navigation is non-trivial and this chapter seeks to improve existing frameworks in four directions as discussed below. First of all, in mapless navigation, the subgoal layouts of most HRL algorithms are defined based on a local grid map. For example, a 3×3 grid centred around the robot, which allows 8 subgoals in the neighbouring cells (Wöhlke *et al.*, 2021). Such a subgoal space limits the robot's exploration within a small area around its current location, while it would be more desirable to explore locations further away from the current location.

Secondly, many HRL methods assume that the low-level tasks are always achievable, i.e., the robot can always reach the subgoal selected by the HL policy (Eppe *et al.*, 2019; Yamamoto *et al.*, 2018). This assumption is unrealistic as it is common for the robot to encounter unreachable locations in complex environments.

Thirdly, mapless navigation can be based on different sensors including Lidar, cameras, or the combination of both (Tai *et al.*, 2017; Zhu *et al.*, 2017). Training RL agents from raw images or Lidar scans has been proven highly inefficient (Levy *et al.*, 2017; Nachum *et al.*, 2018). Although downsampled Lidar scans can provide sufficient information for short-term RL-based navigation (Tai *et al.*, 2017; Zhelo *et al.*, 2018; Marchesini and Farinelli, 2020a; Dobrevski and Skočaj, 2021), it does not provide sufficient information for

long-term navigation. On the other hand, image data contain much richer information, but has more redundancy and is more difficult to process. Existing works propose to represent images using an intermediate state space that covers essential information in a more compact way, but they generalise poorly in unseen environments or new tasks (Wöhlke *et al.*, 2021; Bischoff *et al.*, 2013).

Fourthly, when navigating, humans take into account both intrinsic and extrinsic factors to determine their subgoals, where intrinsic factors could be related to the objectives, preferences, and expectations, and extrinsic factors pertain to the environment's layout, landmarks, available resources and so on (Wolbers and Wiener, 2014; Ekstrom *et al.*, 2014). In simple terms, humans have the ability to distinguish the worthiness levels of different subgoals for further exploration, when selecting subgoals, and making appropriate decisions based on the observations or states (Wolbers and Wiener, 2014). For instance, humans can be aware that the space behind a couch is unoccupied, offering more path options. Similarly, when humans encounter a wall, they recognise that there is no viable path in that direction, and therefore, they will not select a subgoal in that direction. Conversely, when humans perceive a door, they understand that it leads to additional rooms, expanding the number of available path options. However, most HRL-based navigation methods lack such a similar capability. Numerous methods rely solely on current sensory input when selecting subgoals (Wöhlke *et al.*, 2021; Bischoff *et al.*, 2013; Levy *et al.*, 2017; Nachum *et al.*, 2018).

To address these limitations, this chapter proposes a novel HRL-based mapless navigation framework with Predictive Neighbouring Space Scoring (PNSS)/Predictive Exploration Worthiness (PEW). Similar to existing HRL frameworks, the proposed framework also has a high-level (HL) policy that se-

lects subgoals and a low-level (LL) policy that controls the wheels to reach the selected subgoals. However, this chapter seeks to make some improvements. Firstly, it proposes two subgoal worthiness matrices: Predictive Neighbouring Space Scoring (PNSS) and Predictive Exploration Worthiness (PEW), as shown in Fig. 4.1 and Fig. 4.4. The PNSS assesses the area of the explorable space for each of the candidate subgoal locations and the PEW is related to some attributes of the local area around a subgoal, such as the free space area, distribution of obstacles in the area, or the shape/orientation of the area. Then the PNSS/PEW model is developed to predict the PNSS/PEW values using raw images and Lidar data. The predicted PNSS or PEW scores are used to evaluate the worthiness level of each subgoal candidate for further navigation and the values will be included as part of the HL input, such that the agent will not solely rely on sensory inputs for decision making. The LL policy will generate velocity commands based on the current Lidar observations to control the robot to reach the subgoal selected by the HL policy. After reaching the subgoal, the HL policy will repeat the process to select the next subgoal, until the robot reaches the final target location.

In addition, a penalty term is included in the reward function for the HL policy, so that the subgoal selection process would avoid subgoals that are physically unrealistic to achieve. Finally, design a novel subgoal layout that encourages the robot to explore far places ahead (the blue grid in Fig. 4.1 and 4.2).

The rest of the chapter is organised as follows: Section 4.2 describes HRL with PNSS, while Section 4.3 introduces HRL with PEW. Section 4.4 covers the experiments, and Section 4.5 presents the results. Section 4.6 summarises this chapter.

4.2 HRL with PNSS

This chapter proposes a novel HRL framework, in which an HL policy selects subgoals in an abstract space and an LL policy is responsible for the locomotion control of the robot to reach the corresponding subgoals, as illustrated in Fig.4.1.

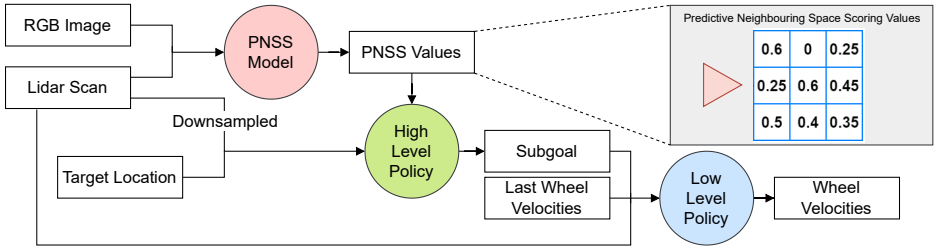


Figure 4.1: The overall framework with PNSS. The HL policy selects a subgoal based on the predicted PNSS values, the Lidar observation and the coordinates of the target coordinates. The PNSS values are predicted by the PNSS model for a set of explorable positions in front of the robot. The LL policy controls the robot to reach the subgoal. The process repeats until the robot reaches the target location.

For the selection of subgoals, we introduce a prediction mechanism to evaluate the worthiness level of a location for further exploration, named Predictive Neighbouring Space Scoring (PNSS). Briefly, the PNSS module is to predict the area of unoccupied space that can be observed at a corresponding location. A deep neural network is trained for the prediction of the PNSS values given the measurement of local surroundings. These PNSS values are then provided to the HL policy for the selection of the next subgoal. We design a reward function for training the HL policy including a penalty that occurs when the LL policy fails to reach a subgoal. The framework contains three key modules:

- PNSS module for estimating the PNSS values based on the current observation,
- HL policy that decides the next subgoal, and
- LL policy that is responsible for the locomotion control of the robot to reach the target subgoals selected by the HL policy.

4.2.1 PNSS Model

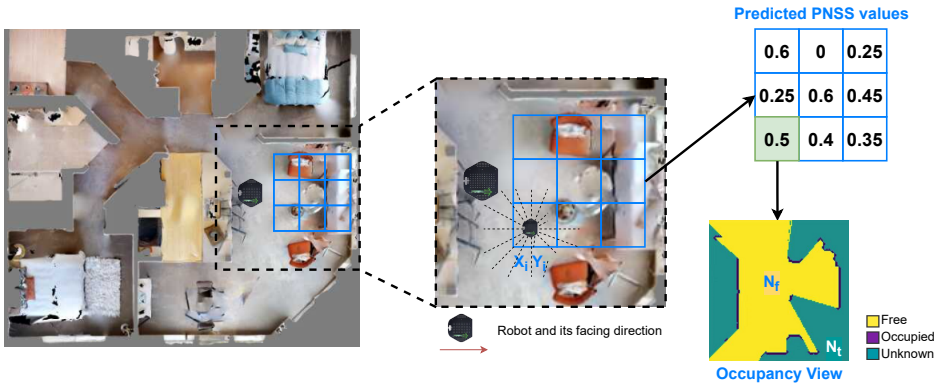


Figure 4.2: An example of the predicted PNSS values of 3×3 positions, which are located in the forward direction of its ego-centric view. Each cell in the layouts is 0.5 meters in width and length. The occupancy view is based on the Lidar observation at the corresponding location, which the simulator can convert into a 2D occupancy view.

As mentioned, we introduce a metric to score the explorable worthiness of a given location. As illustrated in Fig. 4.2, the explorability of a location is estimated from the current egocentric observation of the robot. The worthiness for exploration is related to the free neighbouring space available at each corresponding location. We introduce the PNSS metric, defined as

the proportion of the observable free space of a local region (see Fig. 4.2), formulated as:

$$s(x_i, y_i) = \frac{N_f}{N_t} \tag{4.1}$$

where x_i and y_i represent the coordinates of the i-th cell in the subgoal space with respect to the robot's coordinate frame, for which the score is calculated. N_t represents the area of the local region of interest (represented by 128×128 cells), and N_f represents the area of free space (i.e. number of non-occupied cells) measurable with Lidar observations at $[x_i, y_i]$, $s \in [0, 1]$.

The PNSS model is trained in a supervised manner (Nasteski, 2017). We use the iGibson simulation environment (Shen *et al.*, 2021) to obtain the ground-truth scores as training labels. For data collection, the robot is randomly placed at any arbitrary position of concern. As illustrated in Fig. 4.2, since the robot is equipped with a Lidar with the Field of View (FoV) of 360 degrees, the occupancy view can be obtained directly from the Lidar data. We then count the number of free cells N_f in the occupancy view and calculate the PNSS using Eq. (4.1).The higher the score is, the more free space the robot observes in the local region. A score of 0 indicates that the neighbouring areas are fully occupied.

In addition, for predicting PNSS, we also use a standard forward-looking camera that can only provide RGB images with a horizontal FoV of 58 degrees (Asus Xtion pro). We believe that the visual features from the RGB images with rich information will improve the prediction accuracy. This is validated in Experiments Section.

We adopt a network architecture inspired by the occupancy anticipation model (Ramakrishnan *et al.*, 2020). The PNSS network predicts the PNSS values for a few candidate locations based on the current observation of

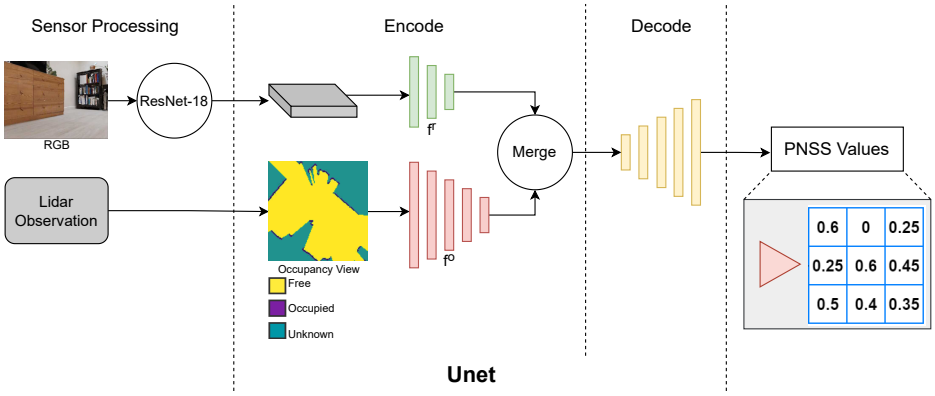


Figure 4.3: The PNSS model extracts features from the RGB image firstly. The Lidar observation is then projected into a 2D occupancy view. A UNet network is used for predicting the PNSS values for a 3×3 grid map.

RGB images and Lidar data. The main components are shown in Fig. 4.3 and summarised below:

- The sensor pre-processing module contains two parts: 1) feature extraction from RGB images using a pre-trained ResNet-18 model (Sarwinda *et al.*, 2021), which is selected for its ability to capture general visual features (e.g., edges, textures) learned from large-scale datasets like ImageNet (Ridnik *et al.*, 2021), providing robust input for further processing, and 2) the current Lidar observation that will be transformed into the occupancy view.

- Given the RGB features and the occupancy view, we encode them using UNet encoders (Ronneberger *et al.*, 2015) individually. The RGB features are encoded using a stack of three convolutional blocks denoted as f^r. The occupancy view is processed by a stack of five convolutional blocks into a feature vector, denoted by f^o.

- We then combine f^r with f^o through the Merge module to construct a

combined feature f^g. The Merge module contains layer-specific convolution blocks to merge all layers in $[f_i^r, f_i^o]$ (Ramakrishnan *et al.*, 2020). It can be formulated as $f^g = \text{Merge}(f^o, f^r)$.

- The combined feature, f^g, is decoded using a Unet decoder that outputs the PNSS values for the corresponding positions in the subgoal space, formulated as $S_{PNSS} = \sigma(\text{Decode}(f^g))$. Fig. 4.3 shows a 3×3 subgoal space, as an example.

4.2.2 High-Level Policy

The HL policy is used to generate the next short-term navigation subgoal for a robot to navigate. In this work, DQN is used to train the HL planning policy. The following subsections describe the main components of the HL policy, namely 1) the observation, 2) the action/subgoal space, and 3) the reward function.

Observation

The observation for the HL policy comprises three main components, denoted by $o_t^H = \{o_L || o_{PNSS} || g_{p\_t}^H\}$, where $||$ represents vector concatenation combining two vectors into one higher-dimensional vector. o_L is the Lidar reading at the robot's current location, $o_{PNSS} = \text{Flatten}(S_{PNSS})$ is the vector of PNSS values by flattening the S_{PNSS} matrix, whose size depends on the subgoal space. Starting from the row of subgoals farthest from the robot's current position, the subgoals are considered from left to right: first, second, third, and so on. Then, the first PNSS value corresponds to the first subgoal, the second PNSS value to the second subgoal, and so forth. $g_{p\_t}^H = (r_t, \theta_t)$ represents the polar coordinates of the target location with respect to the robot

frame. This work is based on the assumption that the global coordinates of the target and robot locations are available for a mapless point navigation task, usually known as the PointNav task (Abhishek Kadian* *et al.*, 2020).

Action Space (Subgoals)

The action space of the HL policy produces the subgoals for navigation that will serve as the goals for the LL policy, i.e., $A^H = G^L$, where, A^H is the high-level action space and G^L is the low-level goals.

Efficient robot learning for the high-level policy requires careful consideration of the action space in terms of the locations of the subgoals and the complexity of the subgoals (number of actions). Some previous works utilise a 3×3 grid centred at the robot's current pose (Wöhlke *et al.*, 2021) and most of them rely on 360 Lidar only. In this work, we move the 3×3 grid subgoal space to the front of the robot's current view. We consider the method more intuitive because the forward-looking camera in this work would provide rich information about the environment in front of the robot, hence predicting more accurate PNSS values. In addition, when predicting the PNSS value for each subgoal, the input consists of a 360-degree Lidar observation, which the simulator automatically converts into an occupancy view. This occupancy view can offer crucial insights into the distribution of obstacles behind the robot, aiding in the prediction of potential obstacles ahead. For example, in indoor environments, a high density of obstacles behind the robot suggests a lower likelihood of encountering a similar concentration of obstacles in the robot's forward path. On the other hand, these positions cover a larger range of explorable areas compared to those used by existing works (Wöhlke *et al.*, 2021). We compare the performance of different choices of such explorable positions in Experiments Section. Due to the spatial constraints with the

introduced subgoal space, when the above subgoals are invalid, (e.g., falling outside of the environment), we also introduce some additional rotation actions in the subgoal space (14 angles in this work). In such cases, the HL policy would encourage the robot to rotate with an angle. The action space can be formulated as below.

$$A^H = [grid_1, grid_2, ..., grid_9, angle_1, angle_2, ..., angle_{14}] \tag{4.2}$$

Therefore, there are in total 23 subgoals, i.e., HL actions, available for selection.

Reward Function

The HL policy will be rewarded or penalised in the different cases, as formulated below:

$$R^H = \begin{cases} r^H_{arrive} & \text{if } d_t \leq \delta^H \\ r^H_{collision} & \text{if collision} \\ r^H_{overtime} & \text{if } t_L \geq T \\ r^H_{approach} & \text{if approaching the subgoal} \\ r^H_{rotate} & \text{if rotate} \end{cases} \tag{4.3}$$

where r^H_{arrive} is a positive value when the robot reaches the target location, i.e., when its distance to the final target location, d_t, is within a radius δ^H; $r^H_{collision}$ and $r^H_{overtime}$ are the penalty values that occur when the LL policy fails to reach a selected position due to collision or timeout; $r^H_{approach} = d_{t-1} - d_t$ is the change of distance from the robot to the target location between two consecutive time steps. $r^H_{approach}$ is positive when the robot is getting closer to the target, and negative when moving away; $r^H_{rotate} = -c_r(|\frac{7\theta}{pi}|)$ is a term that penalises when the HL policy selects a subgoal to rotate. The

greater the rotation angle θ, the greater the penalty. c_r is a weighting factor that scales the penalty value. The term r^H_{rotate} is used to encourage smoother motions.

It is worth noting that, as discussed before, the reward function above takes the LL policy's capabilities into consideration for HL decision making. It is not realistic to assume that all LL tasks can be completed. We, therefore, introduce rewards, e.g. $r^H_{collision}$ and $r^H_{overtime}$, for the HL model such that it is penalised when the LL model fails to reach a subgoal. This will encourage the HL model to consider reachability in its subgoal selection. Alg. 1 details the algorithmic implementation of the HL policy. An 'epoch' refers to one complete cycle of training, where the model is trained on the current environment (which changes every time m is updated). After each epoch, the training environment is sampled again, ensuring different conditions or experiences for the model in subsequent epochs.

Algorithm 1: HL policy of HRL model with PNSS

Given:

- Pretrained LL policy π_{LL} ;
- HL policy π_{HL}, HL buffer D_{HL} ;

Initialise DQN of HL policy

for $m \leftarrow 0$ **to** M *epoch* **do**

  # Sample training environment Env 1, 2, 3...(once m is changed, sample another different environment in turn)

  for $j \leftarrow 0$ **to** J *training episode* **do**

    $t_{HL} = 0$,

    $done = 0$

    Sample a target location g_t

    while do

      # Obtain a subgoal g_s

      $g_s \sim$ epsilon-greedy$(\pi_{HL}(o_t^H))$

      while do

        $t_{LL} = 0$

        $a_{t_{LL}} \sim \pi_{LL}(o_t^L)$

        $t_{LL} = t_{LL} + 1$

        $o_t^L = o_{t+1}^L$

        if $t_{LL} \geq T_{LL}$, *or* g_s *reached, or collision* **then**

          break

      $t_{HL} = t_{HL} + 1$

      if $t_{LL} \geq T_{LL}$, *or* g_t *reached, or collision, or* $t_{HL} \geq T_{HL}$ **then**

        $done = 1$

      $D_{HL} \leftarrow (o_t^H, g_s, R^H, o_{t+1}^H, \text{done})$

      $\lambda_{t_{HL+1}} \leftarrow$ Adam $(\lambda_{t_{HL}}, D_{HL})$

      $o_t^H = o_{t+1}^H$

      if $done = 1$ **then**

        break

4.2.3 Low-Level Policy

The LL policy is used to train a robot to learn how to reach any given goal in a short range. It interacts directly with the environment selecting actions for the robot. DDPG is utilised to train the LL policy.

This subsection describes the details of the LL policy in three parts: the observation, the actions and the reward function.

Observation

The observation of the LL policy comprises three parts, denoted as $o_t^L = \{o_L || a_{t-1}^L || g_{p\_t}^L\}$. o_L denotes the Lidar data of its local surroundings. The action from the last timestep, a_{t-1}, is included in the observation because, due to the inertia, the robot's motion commands will have effects on its successor steps. Last, $g_{p\_t}^L$ is the target location represented in the polar coordinates of the robot frame.

Action

The LL policy directly controls the wheel velocities of the robot (a TurtleBot in the experiments), $a_t^L = \{v_{left}, v_{right}\}$. Each velocity action lasts for 0.1 seconds.

Reward Function

The reward function for training the LL policy is as follows:

$$R^L(o_t^L, a_t^L, g^L) = \begin{cases} r_{arrive}^L & \text{if } d_t \leq \delta^L \\ r_{collision}^L & \text{if collision} \\ r_{approach}^L & \text{otherwise} \end{cases} \tag{4.4}$$

where r^L_{arrive} is a positive value, when the robot reaches the target location, i.e., when its distance to the target d_t is within a radius δ^L; $r^L_{collision}$ is the penalty value that occurs when the robot collides with an obstacle; $r^L_{approach} = c_d(d_{t-1} - d_t)$ is the distance reward, where $(d_{t-1} - d_t)$ is the change of the distance from the robot to the target location at two consecutive time steps, and c_d is a weighting factor.

4.3 HRL with PEW

This section mainly introduces the HRL framework with PEW, as shown in Fig. 4.4. It contains three key modules, including the PEW model, HL policy and LL policy. Firstly, the PEW model predicts the PEW values for the positions of the subgoals. Then, the HL policy selects a subgoal based on current Lidar observations and the predicted PEW values. The LL policy is responsible for producing locomotion control of the agent to reach the subgoals selected by the HL policy. The difference between HRL with PNSS and HRL with PEW lies in the predictive model and the HL state input representation; all other components remain the same. The PNSS value denotes the proportion of free space within the local area around a subgoal. To further refine this measure, the PEW value is developed, which accounts for the distribution of obstacles, thereby complementing the PNSS value. We hypothesise that the quantity of obstacles surrounding each subgoal is a critical factor, while the spatial distribution of these obstacles also provides valuable contextual information.

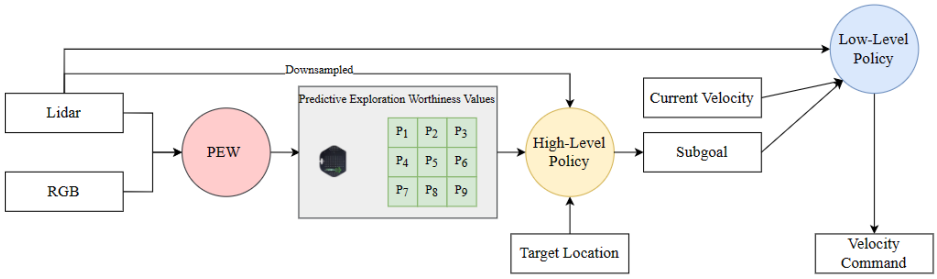


Figure 4.4: The overall framework with PEW. The HL policy selects a subgoal based on the PEW values of each subgoal $(P_1, P_2, ..., P_9)$, the Lidar observation and the relative goal position. The LL policy controls the robot to reach the subgoal. The process repeats until the robot reaches the target location.

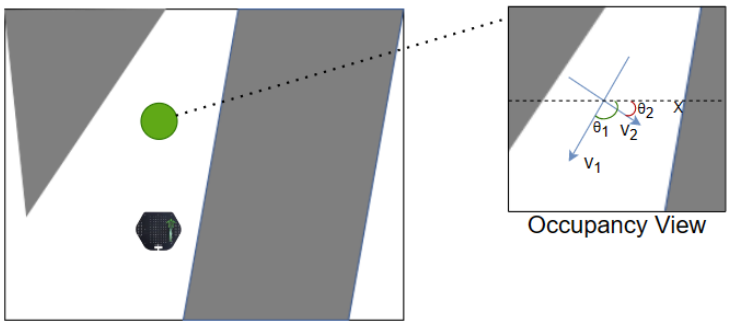


Figure 4.5: An example of what the PEW model predicts. In the left figure, the grey areas represent occupied regions, and the white areas represent free space. The green circle denotes one of the subgoals the robot can select. The right figure is the occupancy view measurable with Lidar at the position of the subgoal. The PEW model is used to predict the area of the occupancy view and key features of the area, in terms of the distribution of obstacles and shape/orientation of the free space. To describe the features of the free space, we use the eigenvectors and eigenvalues of the pixels in the free space, where V1 and V2 represent the two eigenvectors with V1 having a larger eigenvalue.

4.3.1 PEW Model

To estimate the exploration worthiness of each subgoal, we propose a new metric, Predictive Exploration Worthiness (PEW). We consider the worthiness is related to the neighbouring space at each corresponding position. Specifically, the PEW model is designed to predict the free space area, as well as other attributes, such as the distribution, orientations and shape of the space. Fig. 4.5 illustrates one example of an occupancy view. It is obvious that the area can ensure navigation safety, and a larger area may provide more path choices. However, in more complex situations, we need to consider other attributes.

(a) Subgoal 1 (b) Subgoal 2 (c) Subgoal 3

Figure 4.6: Occupancy views of some examples of complex subgoals. The white area is free space and the grey regions represent occupied or unknown space. All three figures contain the same area of the free space, except their geometric distribution and orientations.

For illustration purpose, Fig. 4.6 shows the occupancy views of three complex cases. In Fig. 4.6a and Fig. 4.6b, the areas of the free spaces around the two subgoals are similar, but the shape and distribution are different, and obviously, subgoal 2 is consider more preferable than subgoal 1 due to the complexity of the obstacles. On the other hand, we consider orientation would also be an important feature for navigation decision making. Fig. 4.6b

and Fig. 4.6c have similar shapes and areas. However, if the target location is located on the right-hand side, subgoal 2 would be more preferred as this is more likely to lead to the goal location. Therefore, we believe the area, distribution, orientation and shape are the key features for the PEW metric, which is formulated as,

$$P(x, y) = [S, E] \tag{4.5}$$

where x and y are the coordinates of the subgoal with respect to the robot coordinate frame, S represents the area of the free space and E denotes the distribution and shape.

$$S = N_f / N_t \tag{4.6}$$

N_t represents the total area of the local region of interest (represented by 128×128 cells), and N_f represents the area of the free space (i.e. number of non-occupied cells) measurable by Lidar at $[x, y]$, $S \in [0, 1]$.

Rather than predicting the occupancy map directly, we introduce a compact representation of the free space, based on the Principal Component Analysis (PCA) (Karamizadeh *et al.*, 2020). PCA is a widely used method for dimension reduction, where the principal components refer to the eigenvectors of the covariance matrix. The specific method is as follows:

- **Matrix of points.** The local region is represented by 128×128 cells. Therefore, the region can be seen as a 128×128 matrix. The coordinates of the free cells are extracted to form a $2 \times n$ matrix, $\begin{bmatrix} x_1, x_2, ..., x_n \\ y_1, y_2, ..., y_n \end{bmatrix}$, where n is the number of free cells. The first row represents the x-coordinates, and the second is the y-coordinates.

- **Subtract the mean for each point.** The mean of the x coordinates is computed, and then, for each x-point, the mean value is subtracted from the x coordinates. This procedure is repeated for the y-coordinates.

- **Covariance matrix calculation.** Calculate the 2×2 covariance matrix.

$$C = \begin{bmatrix} \sigma^2(x,x) & \sigma^2(x,y) \\ \sigma^2(x,y) & \sigma^2(y,y) \end{bmatrix} \tag{4.7}$$

- **Eigenvectors, eigenvalues of covariance matrix.** Calculate the two eigenvalues and two eigenvectors of the covariance matrix.

- **Rearrange the eigen-pairs.** Sort by decreasing eigenvalues d_1, d_2. The dominant direction can be determined by the eigenvector having the largest eigenvalue, as shown in Fig. 4.5 where v_1 shows a larger eigenvalue.

- **Calculate the orientations of eigenvectors.** Calculate the angles between the two eigenvectors and the x-axis, denoted by θ_1 and θ_2 respectively, as illustrated in Fig. 4.5.

Therefore, E in Eq. 4.5 can be defined as $[d_1, d_2, \theta_1, \theta_2]$. Therefore, $P(x, y)$ in Eq. 4.5 includes 5 values in total.

The PEW model is trained in a supervised manner. We use the iGibson simulation environment (Shen *et al.*, 2021) to obtain the ground-truth scores as the training labels. For data collection, the robot is randomly placed at any arbitrary position of concern. Since the robot is equipped with a Lidar with an FoV of 360 degrees, the occupancy view can be obtained directly from the

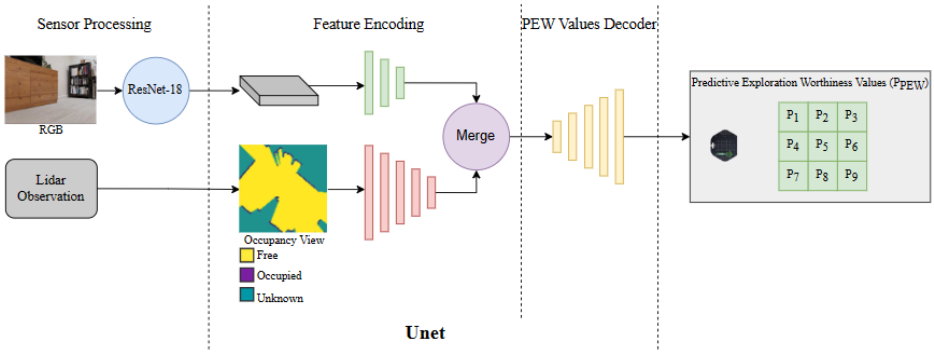


Figure 4.7: Network structure of the PEW model.

Lidar data. We then count the number of free cells N_f in the occupancy view and calculate the eigenvalues and orientations. If the neighbouring areas are fully occupied, N_f in Eq. 4.6, d_1, d_2, θ_1 and θ_2 are all set as 0.

We propose the network structure of the PEW model inspired by the occupancy anticipation model (Ramakrishnan *et al.*, 2020). The network structure is shown in Fig. 4.7. Firstly, the RGB image is processed by ResNet18 to extract features, while Lidar observations are converted into the occupancy view provided by iGibson (Shen *et al.*, 2021). Then, both are encoded separately using Unet (Ronneberger *et al.*, 2015). The RGB features are processed through a stack of three convolutional blocks, while the occupancy view is processed through a stack of five convolutional blocks. To create a combined feature, we merge these features using the Merge module, which comprises layer-specific convolution blocks to merge each layer of both encoded features. Finally, the combined feature is decoded using the Unet decoder that outputs the PEW values.

4.3.2 High-Level Policy

Regarding the HL inpurt representation, a state S_t^H at time t is defined as $S_t^H = [O_L||G_H||S_{PEW}]$, O_L is the current Lidar observation, and G_H is the relative goal position. $S_{PEW} = Flatten(P_{PEW})$ is a 1×45 vector by flattening the P_{PEW} matrix. All other components remain the same, including the HL reward function, LL policy and so forth.

4.4 Experiments

4.4.1 Simulation Environment

The iGibson simulator (Shen *et al.*, 2021) is used in this chapter. It is based on the Gibson dataset (Xia *et al.*, 2018) that includes a large number of complex and photo-realistic 3D domestic environments, such as houses, offices, restaurants and coffee shops. We use a Turtlebot provided by the simulator in the experiments. It is equipped with a camera that generates $3 \times 480 \times 640$ RGB images and a Lidar with 360 laser beams covering a FoV of 360 degrees.

In this chapter, 23 environments are selected from the Gibson dataset, where 10 are used for the PNSS model training, 10 are used for the HL policy and the LL policy training and 3 are used for testing (shown in Fig. 4.8). The training strategy for the PEW model is identical to that for the PNSS model.

One main cause of failures is the local minimum problem, i.e., robot being trapped in a local region (Zhelo *et al.*, 2018). To better demonstrate the improvement of this work in terms of solving the local minimum problem, the tests are performed at three difficulty levels. The difficulty levels are defined based on the distances from the robot to the destinations. Respectively,

the three difficulty levels correspond to tasks with distance ranges of $[2, 5]$, $[5, 8]$, and $[8, 10]$. Tasks in each of the above categories are initialised with randomly generated starts and destinations that range between the corresponding bounds above. For example, tasks in the first category have target distances of between $[2, 5]$ meters. Tasks with large ranges would be more challenging and would include scenarios with more complex local maps that tend to lead to local minima. Each test is performed with 500 episodes to compute the average success rate. The same start and goal positions are used for different algorithm configurations to ensure fair comparisons.

4.4.2 PNSS/PEW Model Training

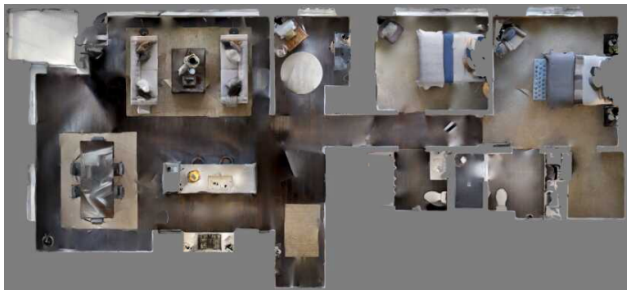
In the iGibson environment, we can directly acquire the ground-truth occupancy view of a given location. In this work, the occupancy view is represented as a 128×128 matrix, with each cell labelled as occupied, free, or unknown. Then from the occupancy view, the PNSS value and PEW value for the location can be calculated. We obtained 5000 sets of data from randomly selected poses in each environment, each consisting of the egocentric RGB image, the Lidar scan and the calculated ground-truth PNSS value and PEW value. In total 50000 sets of data are collected, where 40000 of them are used for training, 5000 are for validation and 5000 are for testing.

4.4.3 LL Policy Training

The LL policy is trained separately. For each episode, the robot is randomly placed in an environment. Since the LL policy is only concerned about short-range navigation, we limit the distance to the destination for each LL episode. The target is randomly sampled at least 0.5 meters away from the robot, but within a square that is centred at the robot, with each side of

(a) Env 1 (Allensville)

(b) Env 2 (Bolton)



(c) Env 3 (Chireno)

Figure 4.8: Example environments for testing.

4 meters. The parameters in Eq. 4.4 are set as below. The arrival reward, $r^L_{arrive} = 20$, is given, when the robot is no more than $\delta^L = 0.36$ meters away from the target position (the chassis radius of the Turtlebot is 0.36m). The collision penalty is set as $r^L_{collision} = -3$. The hyperparameter c_d in the distance reward, $r^L_{approach}$, is set to 10 empirically. The full length of an episode is 1500 timesteps. We train the LL policy for 20000 timesteps in one environment and then continue to the next environment, until a total of 1 million timesteps is reached. At each timestep, the agent explores the environment by taking random actions with a probability $\epsilon = 0.2$ and learnt actions with Gaussian noises with a probability $1 - \epsilon = 0.8$.

We use DDPG to train the LL policy. The actor network for DDPG has three MLP layers with the same size of 512. The critic network also has three MLP layers, the size of the first and last layers is 512, and the dimension of the second layer is 514, with two extra dimensions for the action. ReLU activation is used for each layer on both the actor and the critic networks except for the output layers. Hyperbolic tangent is used for the actor networks to activate the last layer, while the critic network has no activation on the output.

4.4.4 HL Policy Training

After the PNSS model, PEW model and the LL policy are trained, we then train the HL policy. For each episode, the robot is placed at a random location in an environment. We randomise the target positions within a sphere centred at the robot's position, with a distance between the corresponding range bounds. An episode ends in three cases: 1) when the LL policy cannot reach the subgoal within 1500 timesteps; 2) when the HL policy cannot reach the target position within 400 selections of subgoals; and 3) the robot collides

with an obstacle. An episode is considered successful, when the robot is no more than $\delta^H = 0.86$ meters away from the target position, and an arrival reward, $r^H_{arrive} = +20$, is given. The hyperparameter c_r in the rotation penalty term is empirically set to 0.05. The collision and overtime penalties are set as $r^H_{collision} = -3$ and $r^H_{overtime} = -3$.

We train the HL policy for 150 episodes in one environment and continue to the next environment, until a total number of 60000 episodes is reached. The HL policy uses a *epsilon-greedy* to explore the environment, with *epsilon* decaying linearly from 1 to 0.05 within the first $42,000$ episodes, and being kept 0.05 to the end of training. These parameters are empirically set after trial-and-errors.

We use Deep Q Network (Mnih *et al.*, 2015) to train the HL model. The network is represented by two MLP layers of sizes 512 and 256. ReLU activation is used only on the output of the first layer. The output of the network is the Q values of selecting the subgoals with a given observation.

4.4.5 Subgoal Layouts

Since there are infinite combinations of subgoal layouts (HL action space), it is impractical to evaluate all of them. We focus on three subgoal layouts, as shown in Fig. 4.9. Each cell in the layouts is 0.5 meter in width and length.

- **Layout 1** simply takes the 8 neighbouring cells as its next candidate subgoals. This is the same as used in (Wöhlke *et al.*, 2021).
- **Layout 2** includes another 3 cells in the forward direction of the robot. This would allow a robot to explore its next subgoal with a larger range.
- **Layout 3** is used in this work. We eliminate the cells behind the robot and introduce a 3×3 grid in the forward direction of the robot.

This will encourage the robot to explore further distance in the forward direction. We also include 14 rotation subgoals to avoid the robot being stuck in local minimum.

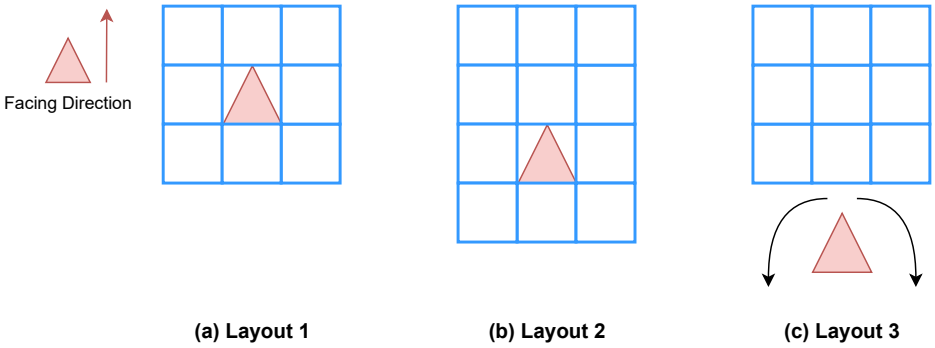


Figure 4.9: Three different layouts we mainly focus on (a) Layout 1 simply takes the 8 neighbouring grids as its subgoal space. (b) Layout 2 adds three more grids in front of the robot on the basis of Layout 1. (c) Layout 3 includes 9 subgoals in the forward direction and 14 rotation subgoals.

4.5 Results and Discussions

To study the performance of our proposed method, we carried out comprehensive experiments and analyses from various aspects, as follows:

- Overall performance of our proposed method in comparison with other RL-based mapless navigation approaches
- Performance of the PNSS value prediction
- Choice of RL algorithms for training
- Effectiveness of the PNSS module in comparison with using Lidar data and encoded RGB image features

- Comparison of different subgoal layout configurations
- Effectiveness of the proposed reward function

4.5.1 Performance Comparison with Other RL-based Approaches

To evaluate the performance of this work, we compare the proposed HRL-based method with three other RL-based algorithms, including non-hierarchical and hierarchical methods respectively.

Non-hierarchical RL-based Methods

There are a number of non-hierarchical RL-based mapless navigation methods. Most of them are inheritance of the work proposed in (Tai *et al.*, 2017). For comparative evaluation, we choose two approaches to compare with the proposed HRL-based method, including one DDPG-based approach for the continuous action space (Tai *et al.*, 2017) and one Double DQN-based algorithm (Van Hasselt *et al.*, 2016) in the discrete space (Marchesini and Farinelli, 2020a).

The input for both methods includes Lidar observations and the polar coordinates of the target. The work in (Tai *et al.*, 2017) also includes the velocity at the previous timestep. The output is the velocity commands, except that one is in the continuous action space and the other one is discrete. It is worth mentioning that the same network architecture as used in (Tai *et al.*, 2017) is deployed for the LL policy in this work, with the same reward function.

As mentioned, one reason for choosing the above methods is that it is known to be the most popular solution for RL-based mapless navigation. It

should be noted that there is no widely-deployed non-hierarchical navigation solution that uses both Lidar and raw images, which are also found inefficient based on the preliminary experiments. Also, the PNSS module or PEW module is not compatible with this non-hierarchical configuration. Therefore, in this work, we could not perform a direct comparison here, and only perform the comparison between our method using PNSS or PEW and the above-mentioned methods without PNSS or PEW.

The experiments were performed in three environments with different difficulty levels with all methods. The success rates of the testing tasks are shown in Table 4.1. It is obvious from the table that the proposed HRL-based method outperforms both the non-hierarchical approaches (Tai *et al.*, 2017; Marchesini and Farinelli, 2020a) in all three cases, except for tests with ranges of $2-5$m in Environment 3. The improvement is considered significant, especially for tasks with longer ranges. In the most difficult experiments (target range: $8-10$m) in three environments, the success rate of HRL with PNSS is nearly 40% higher than the discrete non-hierarchical method (Marchesini and Farinelli, 2020a), indicating that the proposed HRL-based method outperforms the non-hierarchical methods when faced with complex scenarios. We speculate that this is attributed to the ability of the proposed HRL-based method to tackle the local minimum problem which is more often encountered with longer ranges.

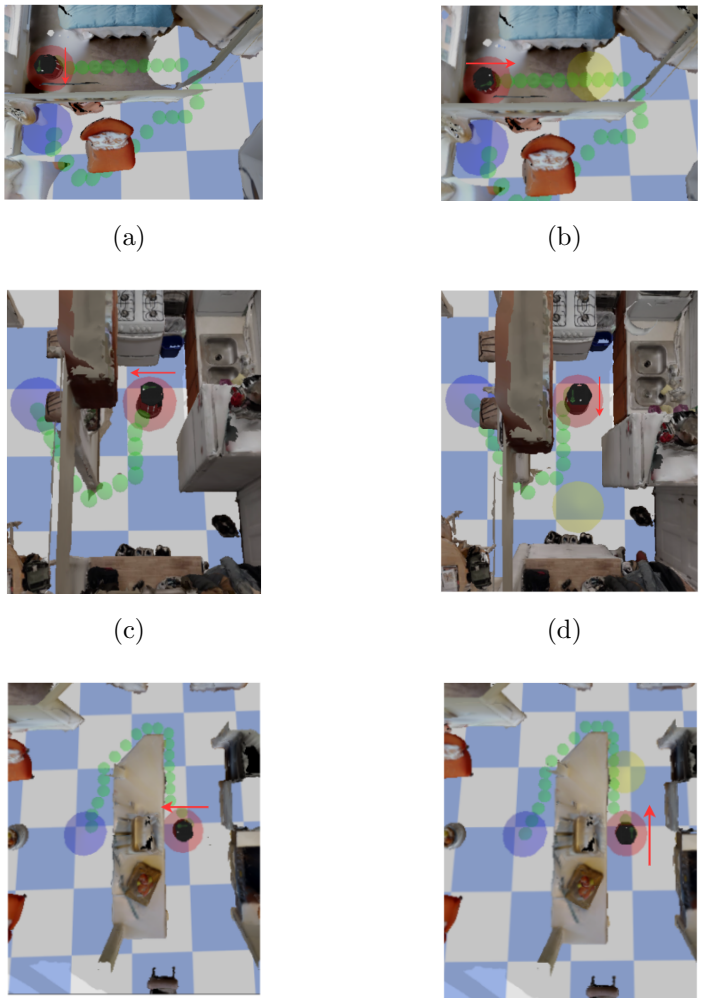
Fig. 4.10 illustrates a few cases of the local minimum problem, where a robot is trapped by a corner, a wall, or furniture. As can be seen in the figure, with the HL subgoals, the proposed HRL-based method would encourage the robot to explore further and could more effectively tackle such situations. This is attributed to the PNSS values for tackling the local minimum problem, which is more often encountered with longer ranges.

Table 4.1: Performance comparison with two non-hierarchical methods in the continuous and discrete space respectively (Tai *et al.*, 2017; Marchesini and Farinelli, 2020a)

Env	Target range	Continuous space	Discrete space	HRL with PNSS	HRL with PEW
1	2-5m	55.0%	38.4%	**59.4**%	57.4%
	5-8m	50.4%	23.4%	**58.8**%	54.6%
	8-10m	38.0%	14.2%	**52.6**%	41.8%
2	2-5m	68.0%	45.0%	**71.6**%	70.2%
	5-8m	57.6%	21.6%	**61.2**%	59.4%
	8-10m	42.0%	14.0%	**52.0**%	50.2%
3	2-5m	**74.8**%	65.0%	71.6%	73.6%
	5-8m	65.0%	29.8%	65.6%	**67.8**%
	8-10m	61.0%	26.2%	64.6%	**66.6**%

Fig. 4.11 illustrates the performance of the non-hierarchical methods and the proposed HRL model with PEW in a long-range navigation task. The discrete action space-based method fails to move towards the target and collides with the obstacle. While the continuous action space-based method is able to approach the target location without any collision, it is trapped by a long table, which is referred as the local minimum problem. In contrast, the proposed HRL-based approach effectively addresses this issue. We speculate that the PEW assists the robot in selecting a more appropriate subgoal, enabling the robot to explore additional locations and successfully escape the situation, thereby highlighting the proposed HRL-based method's superior performance.

In addition, table 4.1 shows that HRL with PNSS outperforms method HRL with PEW. This could be because the data predicted by the PEW model is more complex and may not be as accurate. Additionally, the PEW model predicts four times more data than the PNSS model, resulting in higher dimensionality of the HL state inputs, which affects the HRL model's performance under the same training conditions. Given that HRL with PNSS performs better, this section will focus on HRL with PNSS in the subsequent experiments.



(a) (b) (c) (d) (e) (f)

Figure 4.10: Examples of the robot being trapped by obstacles. The red and blue circles are the start positions and the target positions. The green lines are ground-truth paths. The red arrows are the robot's heading directions. (a), (c), (e) are three cases where the robot keeps heading towards the direction of the target and cannot get around the obstacle, using the non-hierarchical Lidar-based mapless navigation method (Tai *et al.*, 2017). (b), (d), (f) are the solutions provided by the proposed HRL model with PNSS in these situations. The yellow circles are the subgoals given by the HL policy that leads the robot to bypass the obstacles.

(a) Continuous space-based method

(b) Discrete space-based method

(c) Ours

Figure 4.11: Examples of long-range navigation tasks. Orange and blue circles represent the start position and the target position respectively. (a), (b): Green lines represent the robot's trajectory. (c): Green circles are the subgoals selected by the HRL model with PEW.

HRL Method

HiRO (Nachum *et al.*, 2018) is a state-of-the-art HRL method. Its HL policy uses conventional MLP neural networks, trained by TD3 (Fujimoto *et al.*, 2018), operating in the continuous state space as the LL policy. Since the subgoal space is continuous, it is not compatible to use the proposed PNSS model. Therefore, the inputs to the HL policy for this study include Lidar observations and the polar coordinates of the target location. In (Nachum *et al.*, 2018), they train their HL and LL policies jointly. The reward functions for both policies are distance-based. To ensure a fair comparison, we utilise the same reward function as defined in this work. The average reward per 1000 episodes for training the HL policy of HiRO is shown in Fig. 4.12.

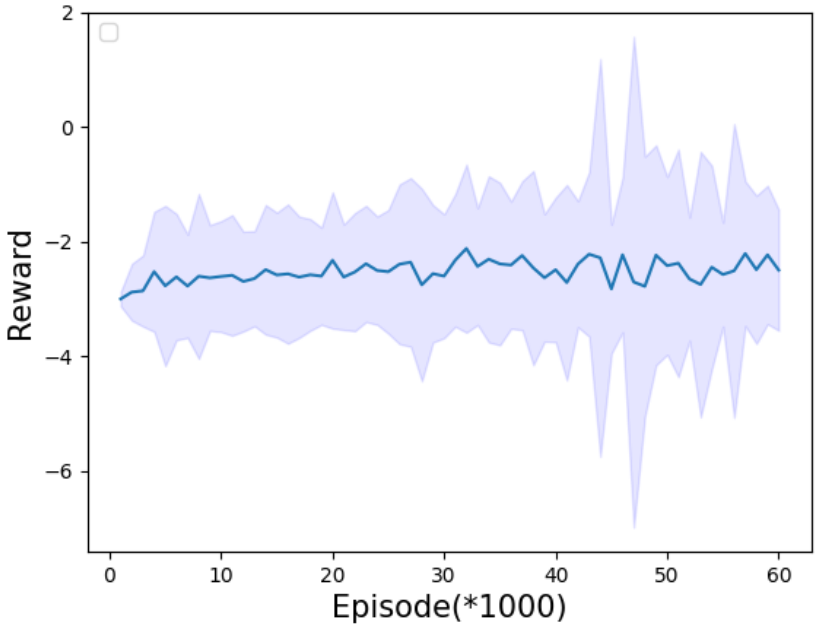


Figure 4.12: Average HL rewards achieved by the agent (HiRO), the shaded area represents the standard deviation.

As shown in the figure, the average rewards achieved by the agent do not show a sign of improvement by increasing the number of episodes for training. The rewards are all negative values too. Therefore, we consider that HiRO is unsuitable for such navigation tasks. It suggests that a continuous subgoal space may not be suitable for the HL policy, due to the large search space of subgoals for efficient policy training. Furthermore, simultaneous training of the HL and LL policies is more challenging for this problem due to the nonstationary problem. Efficient training of the HL policy requires a stable LL policy, which would need random explorations before it stabilises. The untrained LL policy would yield an unstable HL policy training and hence causes inefficiency or even failure in learning effective policies.

4.5.2 PNSS Value Prediction

This section evaluates the performance of the PNSS prediction module and test the hypothesis that RGB observation will help predict the PNSS value of a location. We show the results with 1) only Lidar observation and 2) both Lidar and RGB observations. The model is trained to predict the PNSS values using three subgoal layouts, as shown in Fig. 4.9. The performance of the prediction is measured using the L1 distance between the model's predicted PNSS values and the ground truth.

As shown in Table 4.2, it is clear that the accuracy of PNSS estimation is higher, when RGB observation is considered in addition to Lidar data, producing smaller errors based on the L1 distance metric. We have evaluated several NN structures and loss functions and found that it is difficult to further reduce the training errors. However, the effect of adding RGB observations is rather notable. This is expected as the hypothesis is that visual observations embed richer semantic information that would help in

Table 4.2: Performance of PNSS values prediction using different sensing modalities and subgoal layouts based on the L1 distance metric.

Subgoal Space	Lidar	Lidar + RGB
Layout 1	0.0996	**0.0921**
Layout 2	0.0889	**0.0791**
Layout 3	0.1016	**0.0813**

determining available free space for navigation. For the above reasons, we will only use RGB and Lidar data for PNSS estimation in the rest of the experiments.

To statistically evaluate the PNSS prediction module, we record the average prediction errors (L1 distance) for each of the 5000 groups in the test dataset. Fig. 4.13 shows the distributions of the prediction errors. The prediction errors for each subgoal in each layout are also plotted on a heat map, as shown in Fig. 4.14.

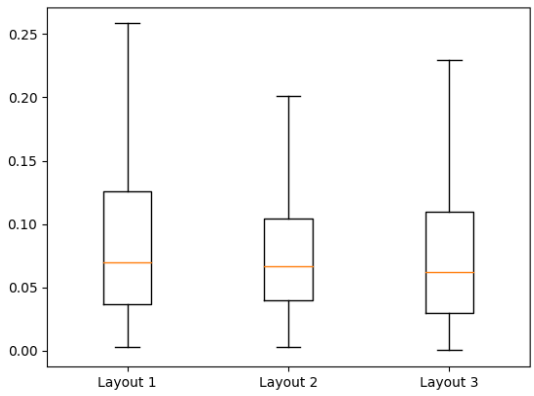
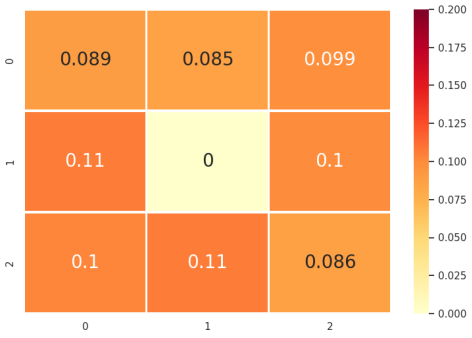
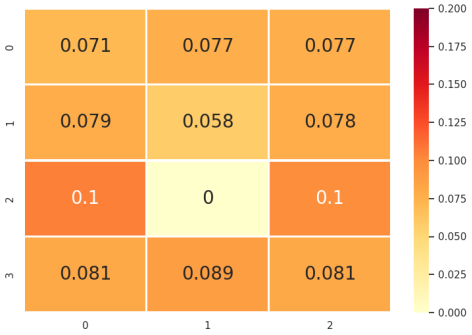


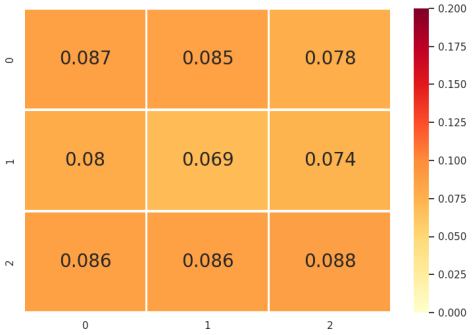
Figure 4.13: Box plot for average PNSS prediction errors with three layouts.



(a) Layout 1



(b) Layout 2



(c) Layout 3

Figure 4.14: Heat map for the prediction errors for each subgoal in each layout. In (a) and (b), 0 represents the location of the robot.

As can be seen in Fig. 4.13, Layout 1 has the largest error distribution that mainly ranges in 0.04-0.13. In contrast, Layout 2 has the smallest error distribution range. Considering that Layout 2 is similar to Layout 1 except the three extra subgoals in front of the robot, the reduction of prediction errors indicates that the model is more accurate in predicting the PNSS values of the subgoals in the front region of the robot, resulting in a lower mean error. Layout 3 contains more subgoals in front of the robot. The overall error distribution is similar to the other two layouts, except that it has the smallest median value, hence higher accuracy. On the other hand, the results further suggest that RGB images can contribute to the prediction of PNSS values, as the camera's FoV is only for the forward direction.

The heat map (Fig. 4.14) shows the L1 distance between the predicted and true values for each subgoal in three layouts. The unit is proportion, i.e., the ratio of free space in the local region. As illustrated, the maximum L1 distance is 0.11, demonstrating the high accuracy of our prediction model. In addition, the prediction is less accurate in the PNSS of the subgoals on the rear and two sides than the forward direction of the robot. This demonstrates that the RGB images contribute to the improvement of the prediction accuracy, as mentioned above. Considering that Layout 3 has the lowest median PNSS value and overall more evenly distributed errors across all subgoals, we choose Layout 3 for this work.

4.5.3 RL Algorithms Used to Train the HL and LL Policies

To decide on the algorithms for training the HL and LL policies, we select several widely used RL algorithms respectively. We first use the TD3 (Fujimoto *et al.*, 2018), SAC (Haarnoja *et al.*, 2018), and DDPG (Lillicrap *et al.*,

2016) to train the LL policy separately. The training strategies and reward functions for the three methods are identical, as described in Sections 4.2.3 and 4.4.3. Since the LL policy is responsible for short-range navigation, we set the distance between the target location and the robot's initial location to $1 - 3$m for each episode. For testing, each test lasts for 100 episodes and the success rates of the three methods are as follows. DDPG produces the highest success rate of 77%, and the success rates trained by the TD3 and SAC are close, 72% and 71% respectively. This suggests that DDPG is more suitable among the three tested algorithms for short-term navigation tasks. Therefore, we select DDPG as the RL algorithm to train the LL policy.

For the HL policy, since the action space is discrete, we choose two widely used algorithms designed for discrete action space, namely DQN (Mnih *et al.*, 2015) and Double DQN (Van Hasselt *et al.*, 2016). Both methods have identical training strategies and reward functions, and utilise the same LL policy trained above.

In the process of training, we test both methods every 3000 episodes, with 50 episodes per test. The success rates are shown in Fig. 4.15.

The success rates of both methods start to rise after about 30000 training episodes, with the method using DQN maintaining a higher success rate than DDQN. The DQN-based policy stabilises at approximately 50% success rate and can reach up to 56%. In contrast, the DDQN-based method maintains a stable success rate of about 40% and remains below 50% overall.

Both methods are tested in the three environments (Fig. 4.8). The success rates are shown in Table 4.3. The agent trained by DQN achieves the best performance in 7 out of the 9 tasks. DDQN is only better in two configurations in Env 3, where, however, the gap between the two is considered insignificant at about 4%. However, in other configurations, the improve-

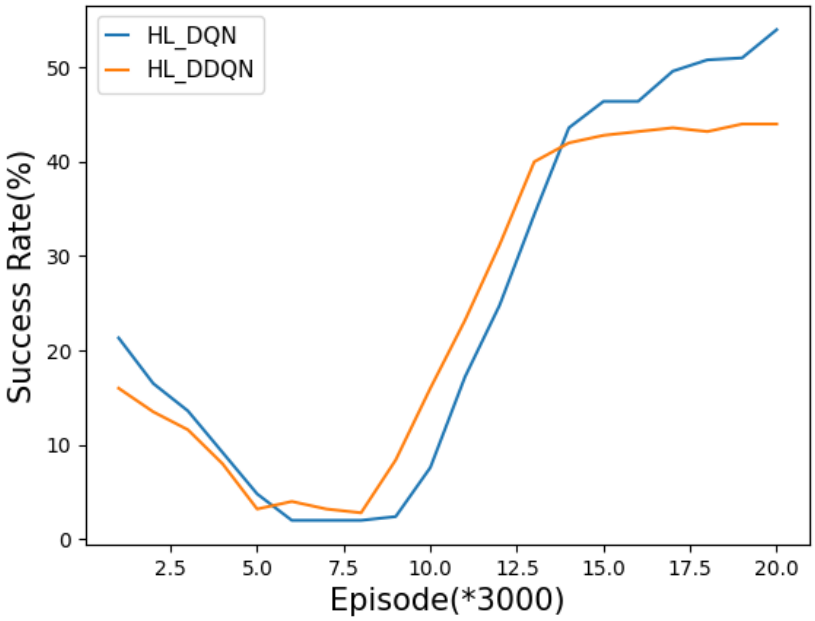


Figure 4.15: Success rates of the DQN and Double DQN algorithms for training the HL policy

ment by using DQN is considerably more obvious. Considering the overall higher success rates and simpler implementation, the DQN is deployed as a more suitable choice for the HL policy.

Table 4.3: Test success rates with different RL algorithms used to train the HL policy

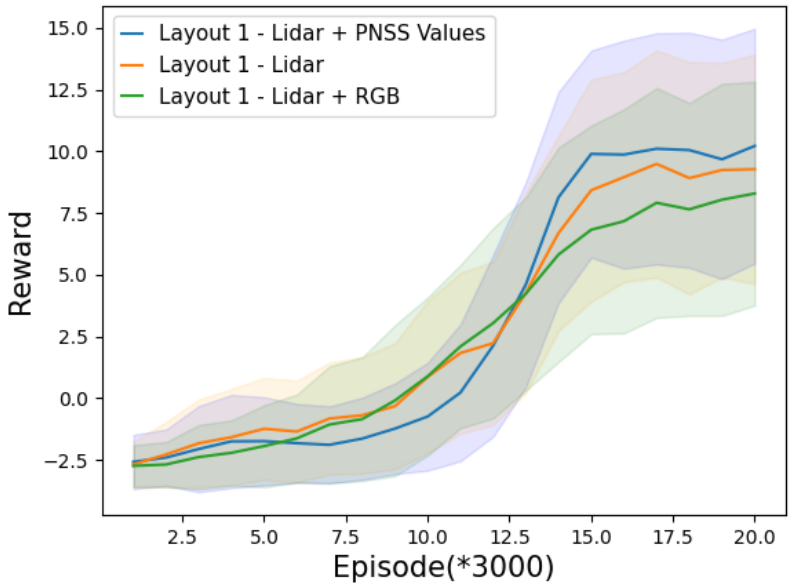
Env	Target range	DDQN	DQN
1	2-5m	59.2%	**59.4**%
	5-8m	54.0%	**58.8**%
	8-10m	43.6%	**52.6**%
2	2-5m	68.6%	**71.6**%
	5-8m	55.8%	**61.2**%
	8-10m	46.8%	**52.0**%
3	2-5m	**74.6**%	71.6%
	5-8m	65.0%	**65.6**%
	8-10m	**68.8**%	64.6%

4.5.4 Ablation Study: Observation Modality

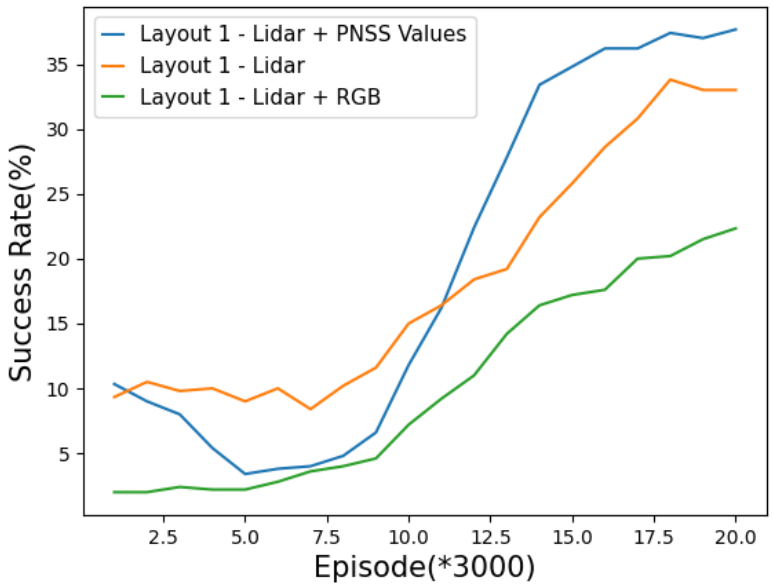
In this section, we evaluate the importance of the PNSS prediction in improving the HL policy and conduct ablation studies with different observation modalities, as follows:

- Only Lidar observations
- Lidar observations concatenated with RGB image features, which are extracted based on (Zhu *et al.*, 2017) using ResNet18.
- Our method, which is similar to the above, but uses predicted PNSS values instead of extracted ResNet18 features.

Fig. 4.16 shows the average reward per 3000 episodes and the test success rates of Layout 1 (Fig. 4.9a), which is one commonly used subgoal space (Wöhlke *et al.*, 2021). As mentioned above, RGB images are encoded using ResNet18 to obtain a compact representation, as used in (Zhu *et al.*, 2017). Fig. 4.16a shows that the agent with PNSS achieves the highest reward (blue line). The one with only Lidar and target information achieves the second (orange line) and the one with RGB image observation achieves the least (green line). Moreover, the shaded region of each curve is corresponding to the standard deviation of the rewards. It can be seen that both the upper and lower bounds of the rewards obtained by the agent with the PNSS values are higher than the other two methods, indicating that our proposed method has a better overall performance. Although the increase in reward does not demonstrate a statistical significance visually, this is considered primarily attributed to the inclusion of a large number of random episodes for training, encompassing varying levels of difficulty. During training, we also conduct testing for each method by running 100 episodes per test and calculating the success rates correspondingly. These tests are performed at regular intervals of every 3000 episodes of training. The success rates are shown in Fig. 4.16b for layout 1. It is clear that after 30000 training episodes, the success rates of all methods start to rise. The proposed HRL-based method (PNSS values and Lidar) clearly reaches a higher success rate than the other two methods, and stabilises at around 38%. The methods with only Lidar data and both RGB features and Lidar reach about 33% and 20% respectively.



(a)



(b)

Figure 4.16: Average rewards (a) and test success rates (b) achieved by the agent with different observation modality-layout 1

The success rates of tests in unseen environments (Environment 1, 2, and 3) are shown in Table 4.4 for Layout 1. The results demonstrate that the proposed HRL-based method using both the PNSS values and Lidar outperforms the other two sensing modalities in all cases. Combining RGB ResNet18 features and Lidar produces the least performance overall, while the Lidar-based method performs in between the other two. We observe that directly using the ResNet18 features or RGB data would not contribute to the performance improvement. This is due to the redundancy in the RGB features, where the HL policy struggles to extract information helpful for the task. For the above reason, we do not consider ResNet18-encoded features for the other two layouts here.

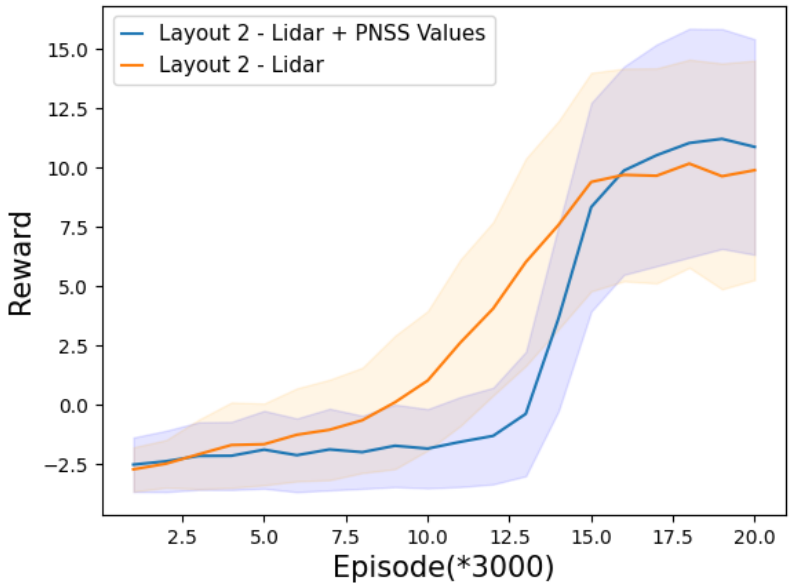
Table 4.4: Test success rates with different observation modalities-layout 1

Env	Target range	Lidar	Lidar + RGB	Lidar + PNSS
1	2-5m	50.8%	48.2%	**53.4%**
	5-8m	44.6%	33.4%	**54.0%**
	8-10m	31.8%	25.8%	**39.8%**
2	2-5m	60.2%	55.0%	**64.8%**
	5-8m	47.0%	34.2%	**49.6%**
	8-10m	30.6%	22.6%	**38.8%**
3	2-5m	68.0%	66.6%	**73.2%**
	5-8m	57.8%	55.0%	**63.2%**
	8-10m	55.8%	46.8%	**62.2%**

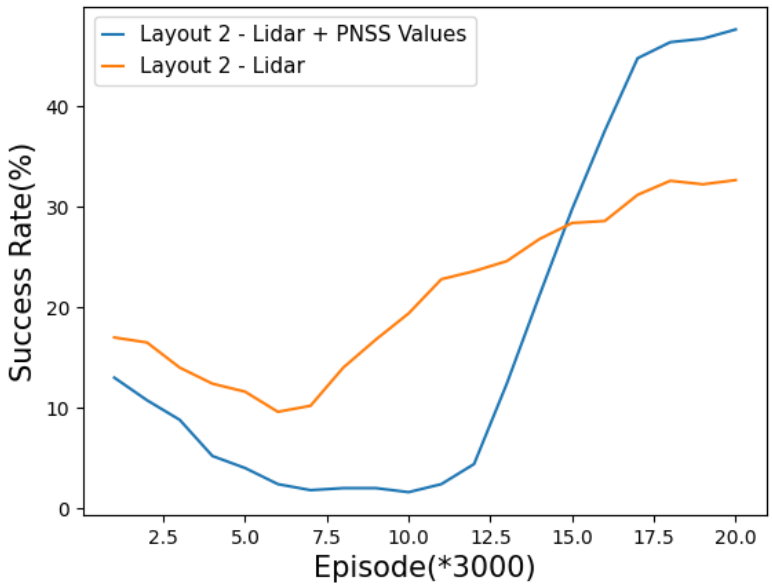
The same experiments are conducted with Layout 2 and Layout 3 too. The average reward per 3000 episodes and the test success rates during training are shown in Fig. 4.17 and Fig. 4.18 respectively. The corresponding

success rates of tests in Environments $1, 2$, and 3 are illustrated in Table 4.5 and Table 4.6.

These results suggest that extracting task-relevant information by pre-processing raw observations encoded as the PNSS values has played an important role in improving its performance. On the other hand, directly using the Lidar observations makes it more challenging to train the policy, because Lidar observation can only provide information about the robot's observed surroundings. It should be also noted that the tasks with larger ranges show more obvious performance improvement while using the PNSS data as observations.

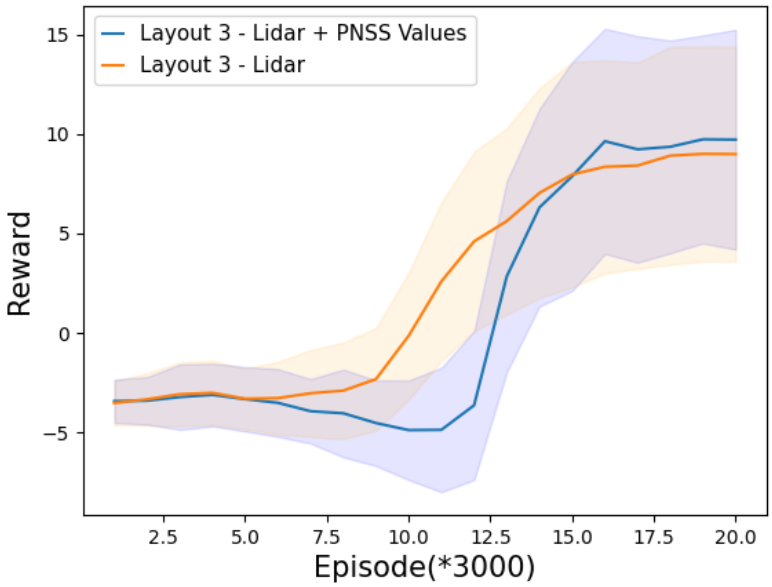


(a)

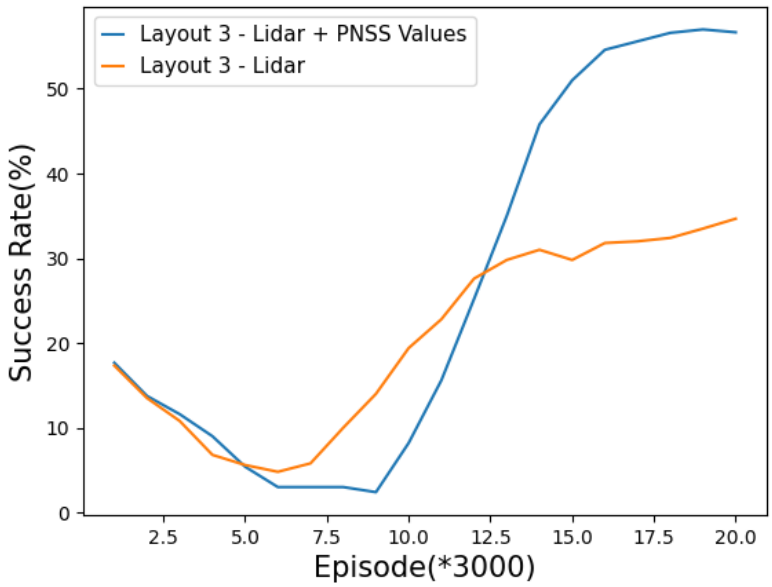


(b)

Figure 4.17: Average rewards (a) and test success rates (b) achieved by the agent with different observation modality-layout 2



(a)



(b)

Figure 4.18: Average rewards (a) and test success rates (b) achieved by the agent with different observation modality-layout 3

Table 4.5: Test success rates with different observation modalities-layout 2

Env	Target range	Lidar	Lidar + PNSS
1	2-5m	52.2%	**57.0**%
	5-8m	47.0%	**55.6**%
	8-10m	31.2%	**42.2**%
2	2-5m	60.0%	**67.8**%
	5-8m	48.2%	**56.4**%
	8-10m	32.2%	**41.8**%
3	2-5m	72.6%	**72.8**%
	5-8m	63.2%	**66.6**%
	8-10m	56.4%	**62.4**%

Despite the promising results with the proposed HRL-based approach, the overall success rates still present a gap from reliable deployment for real-world problems. One of the main reasons is the limited sensing capability and the complexity of the indoor environments in this work. The iGibson environment includes complex layouts and furniture of various shapes. Due to the sensing limitation, some furniture parts cannot be detected by Lidar, such as the legs of chairs which can easily lead to collisions. Also, some obstacles, like coffee tables, are above the horizontal scanning plane of the Lidar but can collide with the robot. The collisions will trigger signals for terminating corresponding episodes, hence limiting the average success rate. Overall, the proposed HRL-based method improves the success rate in most of the difficult tasks compared to other methods.

On the other hand, we also consider the practicality of deploying the policy for real-world control. The average computational times of the models

Table 4.6: Test success rates with different observation modalities-layout 3

Env	Target range	Lidar	Lidar + PNSS
1	2-5m	55.8%	**59.4**%
	5-8m	49.8%	**58.8**%
	8-10m	33.4%	**52.6**%
2	2-5m	59.6%	**71.6**%
	5-8m	39.6%	**61.2**%
	8-10m	25.4%	**52.0**%
3	2-5m	70.4%	**71.6**%
	5-8m	61.6%	**65.6**%
	8-10m	57.2%	**64.6**%

with different input modalities are measured too. The proposed method takes about 0.358s for calculating the PNSS values and the subsequent action. The method that uses Lidar and RGB image features takes about 0.186s for computation. The method with only Lidar needs the least computational time of 0.172s. This is reasonable as this work introduces another tailored step of predicting the PNSS values for the navigation task. Despite the longest computational time required, we consider our method highly feasible for real-world indoor robots that are usually operating at a low speed, as used in this work.

In addition, we also compare the training time required for the three methods. All methods are computationally expensive and would take a long time to run. The proposed method using both Lidar and PNSS values would take the longest time of about 8 days. The methods with 'Lidar' and 'Lidar + RGB features' as inputs require similar time durations of about 6 days.

We use a workstation with an Intel i9-10900X CPU (3.7GHz x 20) and an NVidia RTX-2080 TI GPU. This is partially attributed to the fixed number of episodes (60,000) for all three configurations. With the neighbour scoring mechanism of the PNSS method, a more valid sub-goal could usually be chosen by the HL policy for the LL policy to execute. The lower chance of collision with the PNSS-based method would lead to a longer time for each episode of the LL policy. The other methods, however, tend to have higher chances of earlier terminations due to collisions, hence shorter overall training time.

4.5.5 Ablation study: Subgoal Layouts

Table 4.7: Test success rates with different subgoal spaces

Env	Target range	Layout 1	Layout 2	Layout 3
1	2-5m	53.4%	57.0%	**59.4%**
	5-8m	54.0%	55.6%	**58.8%**
	8-10m	39.8%	42.2%	**52.6%**
2	2-5m	64.8%	67.8%	**71.6%**
	5-8m	49.6%	56.4%	**61.2%**
	8-10m	38.8%	41.8%	**52.0%**
3	2-5m	**73.2%**	72.8%	71.6%
	5-8m	63.2%	**66.6%**	65.6%
	8-10m	62.2%	62.4%	**64.6%**

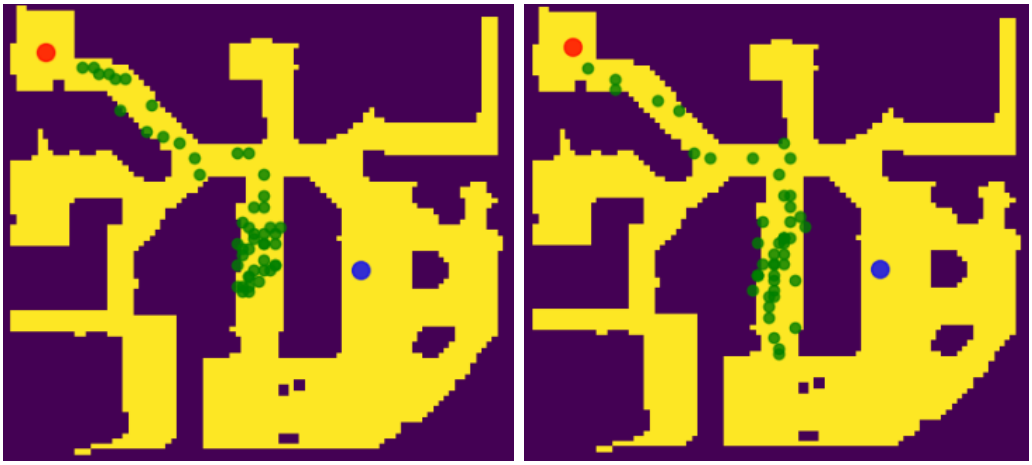
In order to investigate whether the choice of subgoal layout (see Fig. 4.9c) helps improve navigation performance, we conduct experiments with all three

different subgoal layouts (Fig. 4.9). The HL policy reward function for the case of 'Layout 3' subgoal space is described in subsection 4.2.2, while the reward functions for the other two cases are the same except that they have no rotation penalties.

We first report the test success rates of each layout in Table 4.7. One can see that the proposed subgoal space design helps the agent achieve the best performance in 7 out of 9 tasks. The proposed method has the highest success rate in all experiments executed in Env 1 and 2. In Env 3, which is slightly less complex than the other two (higher average success rates and more regular geometric features), the performances for the three layouts are considered similar, except that Layout 3 outperforms in the long range case. The chosen layout shows clear overall superior results and is considered more suitable for the majority of tasks. Especially in the difficult tasks ($8-10$m), the HRL model using 'Layout 3' form has an average success rate 56.4%, while the average rates of the models using 'Layout 2' and 'Layout 1' are 48.8% and 46.9% respectively. The result supports the hypothesis in that the proposed subgoal space helps the robot navigate better as it allows the robot to explore subgoals with a larger range.

Fig. 4.19 illustrates an example of a long-range navigation task with different subgoal layouts. As can be seen, the policies using subgoal Layout 1 and Layout 2 do not perform well. The robot is trapped in local regions in both cases. However, our method would encourage the robot to explore further and effectively tackle the local minimum problem.

By further investigating the results between the 'Layout 1' and 'Layout 2' cases, one can see that the robot performs better when it includes more subgoal options in front of it, i.e. Layout 2 in this case. This then suggests that predicting whether there is more free space further ahead is useful for



(a) Layout 1

(b) Layout 2

(c) Layout 3

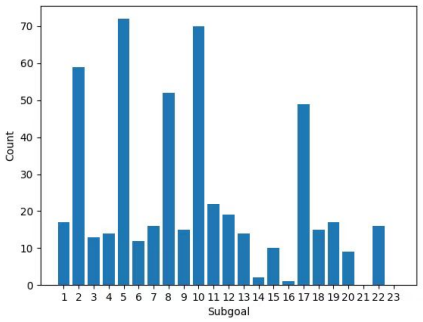
Figure 4.19: An example of the local minimum problem in a long-range navigation task. Red and blue circles represent the start position and the target position respectively. Green circles are the subgoals selected by the policies with different layouts.

navigation tasks.

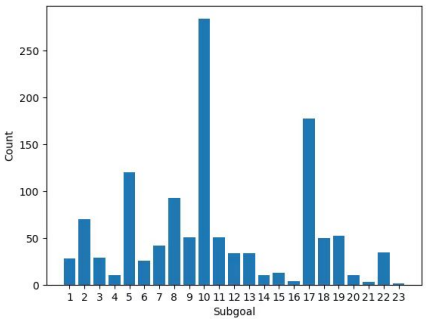
To understand how the subgoals are selected in the actual navigation tasks, we tested 100 episodes in Env 1 (Fig. 4.8a), and counted the occur-

rences that each subgoal was selected in the successful episodes. Fig. 4.20 shows the results grouped into the three difficulty settings. We observe that all the subgoals in Layout 3 were selected. When the task is relatively simple, that is, when the target position is close to the initial position, the subgoals selected by the HL policy are concentrated in the 9 grids directly in front of the robot, indicating that the robot is more confident in performing forward translation motions. However, with the increase in the difficulty of the tasks, the robot has to face a more complex scenario, so the proportion of rotation subgoals increases considerably, especially with small angle rotations to the left (subgoal 10) or right (subgoal 17), because, in the reward setting, the larger the rotation angle the HL policy selects, the greater the penalty will be. Only when necessary will the robot choose to rotate at a large angle in place.

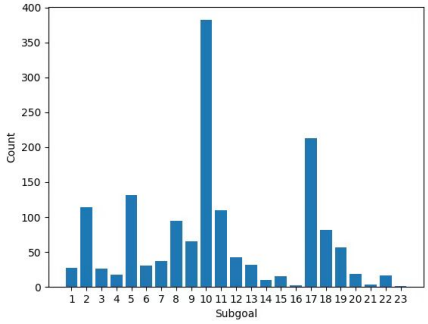
Although the subgoal space layout is set arbitrarily in this work, this provides another insight into HRL mapless navigation problems that subgoal layouts could be optimised or even learned in future work.



(a) 2-5m



(b) 5-8m



(c) 8-10m

Figure 4.20: We record the total number of occurrences different subgoals were selected in all successful episodes when the model is tested in Env 1 (Fig. 4.8a) on tasks of different difficulty settings. 1-9 refers to the 9 grids in front of the robot from near to far and from left to right. 10-16 indicates that the robot rotates to the right. The higher the number, the greater the rotation angle. Similarly, 17-23 represents HL policy selects the left-rotating subgoals.

4.5.6 Ablation Study: Reward Function

To validate the design of the proposed reward function, we perform the ablation experiment by removing the timeout penalty element from the HL reward function, which is proposed with considerations of the LL policy's capability. Specifically, we remove the item $r^H_{overtime}$ that penalises the agent if it could not reach the selected subgoal within a certain period of time, and keep the rest the same. After training, we test the two methods, with and without the timeout penalty. The success rates are shown in Table 4.8.

Table 4.8: Test success rates with different reward functions for training the HL policy

Env	Target range	Without timeout penalty	With timeout penalty
1	2-5m	**60.6**%	59.4%
	5-8m	56.4%	**58.8**%
	8-10m	46.8%	**52.6**%
2	2-5m	71.4%	**71.6**%
	5-8m	59.2%	**61.2**%
	8-10m	48.2%	**52.0**%
3	2-5m	**74.4**%	71.6%
	5-8m	65.4%	**65.6**%
	8-10m	**68.2**%	64.6%

The result shows that the inclusion of the timeout penalty can overall improve the success rates in most tasks. As hypothesised, unreachable or difficult-to-reach subgoals are not desired and the long time costed for the LL policy to attempt navigating to these subgoals should be integrated into

the reward function as penalty. Without the timeout penalty, the agent will continue choosing unreachable subgoals and subsequently impair the overall performance.

4.6 Conclusion

This chapter proposes a novel HRL-based mapless navigation method, where the high-level policy generates a subgoal for the low-level policy, while the low-level policy is responsible for manoeuvring the robot to the given subgoal at the locomotion control level. For the HL policy, this chapter introduces two novel scoring methods, namely Predictive Neighbouring Space Scoring (PNSS) and Predictive Exploration Worthiness (PEW). The PNSS and PEW allow the robot to predict the worthiness level for exploration. Also, they provide the agent with a compact state representation. The PNSS values indicate the explorable space around a given position and the PEW values are related to obstacle spatial distribution, such as the area of free space and the distribution of obstacles. The PNSS/PEW model is trained to predict the PNSS/PEW values for candidate positions around the robot based on the robot's current view. The PNSS or PEW values can be then deployed by the HL policy as observations in addition to Lidar.

Extensive experiments have been carried out to demonstrate the effectiveness of the proposed methods. This PNSS or PEW-based observation has shown significant improvements in success rate, due to its compactness and task-related nature. It should be noted that because of its more accurate predictions and more compact representation, HRL with PNSS performs better than HRL with PEW under the same training conditions. The results also demonstrate that our proposed method outperforms Lidar-based policy

or policy with both Lidar and ResNet18-based RGB image features. Different layouts for the subgoals are also studied, demonstrating the effectiveness of the proposed method. One notable improvement of the navigation performance is for longer range navigation, where Lidar-based methods or non-hierarchical methods would more likely be stuck in local minimum, as demonstrated by the experiment conducted in this chapter.

Chapter 5

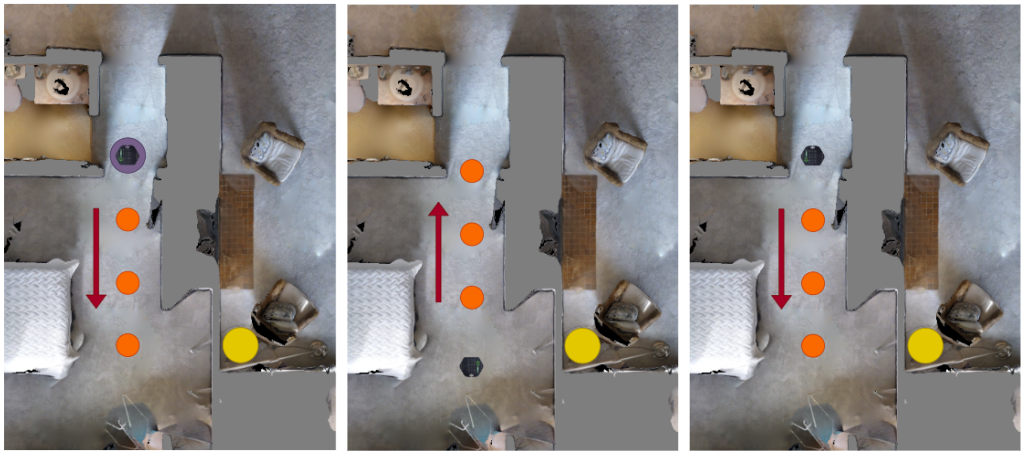
Mapless Navigation via Hierarchical Reinforcement Learning with Memory-Decaying Novelty

5.1 Introduction

Despite the effectiveness of HRL in addressing the local minimum problem as demonstrated in Chapter 4 and other works (Wöhlke *et al.*, 2021; Zhou *et al.*, 2019; Bischoff *et al.*, 2013), it remains limited in complex, cluttered environments. As one example, Fig. 5.1 illustrates one case, where the target is located behind a long wall. In this case, the HRL agent can easily be trapped in a local area. This is due to the following reasons. First, most (H)RL-based mapless navigation methods rely on a simplified reward setting, e.g. rewarded when getting closer to the goal and penalised for any collisions (Zhou *et al.*, 2019; Bischoff *et al.*, 2013; Staroverov *et al.*, 2020). The distance-based reward is usually simply calculated based on the Euclidean distance, which is unrealistic in cluttered environments. Second, exploration is usually based on a simple inefficient random strategy for exploring unknown areas, especially for complex cluttered environments (Zhou *et al.*, 2019; Bischoff *et al.*, 2013). Last, in the case of being trapped in local areas, it would be desirable for the agent to choose alternative paths. This chapter hypothesises that incorporating a memory mechanism could effectively address and resolve such issues.

To address these issues, this chapter proposes a new reward function for the HL policy, containing two main components, namely extrinsic reward and intrinsic reward. The extrinsic reward motivates the agent to move closer towards the target location, expressed as the change in the distance from the robot to the target at two consecutive HL steps.

The intrinsic reward in this work is inspired by the novelty theory defined in (Ruan *et al.*, 2022), which posits that animals reward themselves for identifying something novel. Analogously, an intrinsic reward can be designed to encourage robots to explore unknown environments by quantifying novelty



(a) (b) (c)

Figure 5.1: This is an example of a local minimum problem, where the purple and yellow circles denote the starting and target locations, respectively. The target is situated behind a long wall. (a) Initially, the HL policy selects subsequent subgoals (denoted by the orange circles) in a downward direction, leading the robot towards the target based on the simple Euclidean distance. (b) However, since there is no direct path to the goal due to the obstruction posed by the wall, the robot would need to find alternative routes to bypass the wall. (c) The robot will continue to be attracted by the goal while exploring areas further away from the target, potentially getting trapped in the local area. To address this, this chapter considers a memory mechanism or effective exploration motivation would enable the agent to avoid re-entering previously visited states, thereby mitigating the issue of local minima.

as an intrinsic mechanism that drives curiosity about the world (Bellemare *et al.*, 2016; Ostrovski *et al.*, 2017; Tang *et al.*, 2017). Episode memory is also an important attribute for designing an intrinsic reward (Pritzel *et al.*, 2017). Therefore, in this work, the intrinsic reward function has the following features. First, when the agent reaches a subgoal, previously visited areas or

states will be retrieved for calculating the reward. Specifically, this chapter uses a count-based method (Ostrovski *et al.*, 2017). Second, memory decaying is introduced in this work (Dodd and Gutierrez, 2005) as the basis for assigning rewards, where the size of the reward depends on the steps required to move from the current state to the corresponding state in the memory.

The rest of this chapter is organised as follows. Section 5.2 introduces the proposed method in detail. Section 5.3 introduces the experiment setup, followed by experimental results and discussions in section 5.4. Section 5.5 concludes this chapter.

5.2 Methods

This chapter proposes an HRL-based mapless navigation method, where the HL policy is responsible for selecting a subgoal in the subgoal space and the LL policy controls the locomotion of the robot to reach the corresponding subgoal. Fig. 5.2 illustrates the proposed framework. To train the HL policy, this chapter introduces a new reward function that includes two components: extrinsic and intrinsic rewards. Briefly, the extrinsic reward originates from the environment and motivates the agent to move towards the target location. The intrinsic reward depends on three key variables, namely novelty, episode memory and memory decay. In this section, the working principles of the HL and LL policies and their training processes will be discussed respectively.

5.2.1 High-Level Policy

The following subsections describe the main components of the HL policy, namely the observation, the action/subgoal space, the reward function, and the network structure.

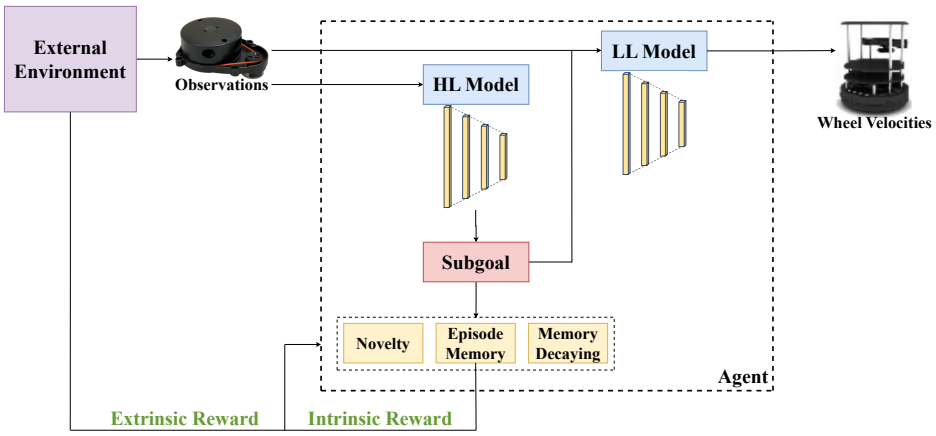


Figure 5.2: The overall framework. The HL policy selects a subgoal based on the HL input. The LL policy controls the robot to reach the subgoal, given the subgoal and other LL input information. The process repeats until the robot reaches the target location. Regarding HL model training, a new reward function is proposed. It has two components: extrinsic and intrinsic rewards. The environment provides the extrinsic reward, and the intrinsic reward is calculated based on novelty, episode memory, and memory decaying.

Observation

The observation of the HL policy comprised of 6 parts, denoted as $o_t^H = \{o_H||g_{p\_t}^H||a_{t-1}^H||r_{t-1}^H||x_t, y_t, \phi_t||N(x_t, y_t)\}$, where $||$ denotes vector concatenation. o_H is the current sensor reading, and $g_{p\_t}^H$ denotes the polar coordinates of the target location with respect to the robot frame. For the agent to learn policies through past experience, the action a_{t-1}^H executed at the last HL step $t-1$, i.e. the last selected subgoal, and the HL reward r_{t-1}^H at the previous step $t-1$, are also included. x_t, y_t, ϕ_t represent the coordinates of the current location and heading of the robot. The magnitude of the reward is contingent upon the frequency of the robot's visits to the current location, which

will be described in subsection IV-B-(3). To expedite the agent's acquisition of the correlation between the reward and visit frequency, we add $N(x_t, y_t)$, which is the number of visits to the current location by the robot, to the observation.

Action Space (Subgoals)

The HL action space, denoted by A^H, is constructed by a list of subgoals that are needed by the LL policy for short-term navigation. Effectively, the HL action space can be seen as the LL goals, i.e., $A^H = G^L$.

Some works use 8 adjacent regions centred on the robot as the action space (Wöhlke *et al.*, 2021). To provide the robot with more options in a complex and obstacle-laden environment, we add 2 additional regions in the front and rear regions of the robot, respectively, as shown in Fig. 5.3. In addition, since the HL input contains the past states, to initialise the agent when no historical experience is available, we use stand still as historical experience for the initial state. Therefore, the subgoal space consists of 13 subgoals, with 12 surrounding areas and one for keeping the robot standing still.

Reward Function

As mentioned, the proposed reward function contains two components: extrinsic and intrinsic rewards. The total HL reward for each step is the sum of the extrinsic and intrinsic rewards except when the robot reaches the target, or the agent remains standing still. The reward R_t^H is defined as

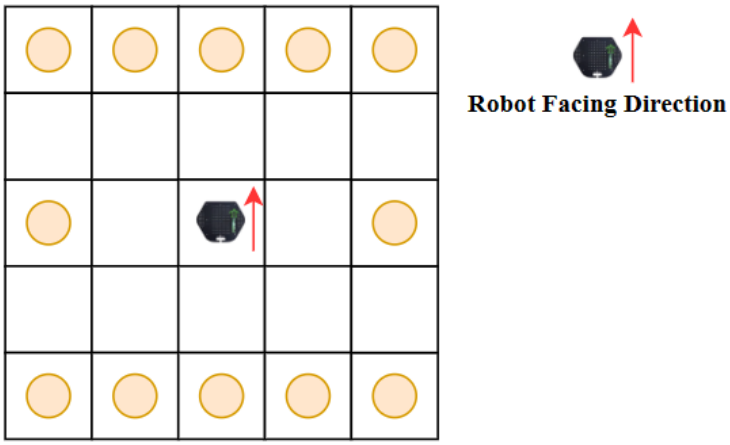


Figure 5.3: Subgoal space. This work sets the surrounding area, centred on the robot's current pose, as the subgoal space (yellow circle). Each grid is 0.35 meters in width and length.

$$R_t^H = \begin{cases} r_{arrive}^H & \text{if } d_t \leq \delta^H \\ R_{ex} + R_{in} & \text{other} \\ R_s & \text{stand still} \end{cases} \tag{5.1}$$

Three conditions are considered for defining R_t^H. R_s is a negative reward for penalising standing still. A positive value r_{arrive}^H is assigned to the reward, when the robot reaches the target location. This is based on a distance threshold condition when the distance to the target d_t is within a radius δ^H. R_{ex} and R_{in} are the extrinsic and intrinsic rewards, respectively, and their summation of them forms the reward in other situations. This approach has been demonstrated to be effective (Zhelo *et al.*, 2018; Dilokthanakul *et al.*, 2019).

The extrinsic reward is defined in a similar manner as employed in other non-hierarchical RL-based mapless navigation methods (Tai *et al.*, 2017; Zhelo *et al.*, 2018). Its mathematical expression is

$$R_{ex} = L_{ex_{t-1}} - L_{ex_t} \tag{5.2}$$

The extrinsic reward is defined as the change of distance from the robot to the target location between two consecutive HL steps. L_{ex_t} represents the distance from the target at HL step t. The extrinsic reward magnitude increases as the robot approaches the target with greater proximity. This is directly useful for the agent to be motivated by the extrinsic reward to learn to move towards the target location.

The intrinsic reward encourages the agent to explore unknown environments. Analogous to the behaviours of animals that rely on spatial memories to determine the novelty of a location (Kemp and Manahan-Vaughan, 2004), for navigation tasks, this chapter considers spatial memory an important mechanism for implementing robot intrinsic motivation for exploration. Specifically, in the work, we store the coordinates of all subgoals the HL policy has previously selected, formulated as below:

$$M = [x_0, y_0; x_1, y_1; ...; x_{t-1}, y_{t-1}] \tag{5.3}$$

The novelty of the current state is derived from the number of occurrences in which the agent has visited the corresponding location. We name this the Novelty Count (NC), denoted by $N(x, y)$.

A threshold δ^B is used as the condition to determine if the location has been visited. The number of visits at the current place $N(x_t, y_t)$ will be increased by 1 if the current location is within δ^B to the corresponding location.

A high NC indicates low novelty for the current state of the agent. The intrinsic reward function can be formulated as

$$R_{in} = \alpha N(x_t, y_t) \tag{5.4}$$

where $N(x_t, y_t)$ is the NC and α is a weighting factor that is negative.

The intrinsic reward encourages the robot to explore more areas efficiently by avoiding reaching locations that have been previously visited. However, this is not ideal in all situations. One example is when the robot arrives at a dead end, where alternative paths are limited or do not exist. In this case, the above intrinsic reward would prohibit the robot from leaving the local region through the only path that has been traversed. Although previously visited places are less favourable, we consider them still worth further exploration. In this work, this chapter introduces a memory decaying factor M_d to the intrinsic reward. M_d enables the robot to revisit areas it has previously explored after a certain period of time. Mathematically, this chapter defines M_d as a function of the time that the agent has spent travelling to the current location (x_t, y_t) from the corresponding location in the episode memory. M_d is formulated as follows:

$$M_d = e^{-(L_M - 1)/10} \tag{5.5}$$

where L_M represent the number of HL steps since the last time the robot visited the location. The memory decaying factor is designed to influence the magnitude of the intrinsic reward. This factor is expressed as a function in the range $[0, 1]$. The value of M_d depends on the distance between the current state and the robot's last visit to that state. Since the intrinsic reward is negative, the closer the distance, the larger the value of M_d, and vice versa. To model this, we use the exponential decay function, x^{th} power of e. Given the steep rate of decay, we divide the distance by a constant to moderate the decrease in M_d. After testing with various data, we chose to divide the number of steps by 10. M_d is zero if the location has not been previously visited. A high L_M indicates that the agent visited the corresponding place

long ago, leading to a small value of M_d. Conversely, if the agent recently visited a location, the value of L_M will be low. L_M is 1 if the agent visited the place in the previous HL step. Since α is negative, the agent will incur a significant penalty for revisiting recently visited locations. Integrating the memory decaying with the intrinsic reward function gives Eq. 5.4, as follows:

$$R_{in} = \alpha N(x_t, y_t) M_d \tag{5.6}$$

In order to avoid an infinitely small reward, a minimum value R_0 is set, and $R_{ex} + R_{in}$ thus becomes:

$$R_{ex+in} = \max(R_0, R_{ex} + R_{in}) \tag{5.7}$$

In addition, to enable the HL policy to take the LL policy's ability into account when selecting subgoals, we add some penalty elements to the reward function. The complete reward function R_t^H for the HL policy becomes:

$$R_t^H = \begin{cases} r_{arrive}^H & \text{if } d_t \leq \delta^H \\ r_{collision}^H & \text{if collision} \\ r_{overtime}^H & \text{if } t_L \geq T \\ R_{ex+in} & \text{other} \\ R_s & \text{stand still} \end{cases} \tag{5.8}$$

where $r_{collision}^H$ and $r_{overtime}^H$ are the penalty values that occur when the LL policy fails to reach a selected subgoal due to collision or timeout.

Network Structure

DQN is used to train the HL policy. As shown in Fig. 5.4, this work employs the LSTM network to equip the agent with memory and reasoning

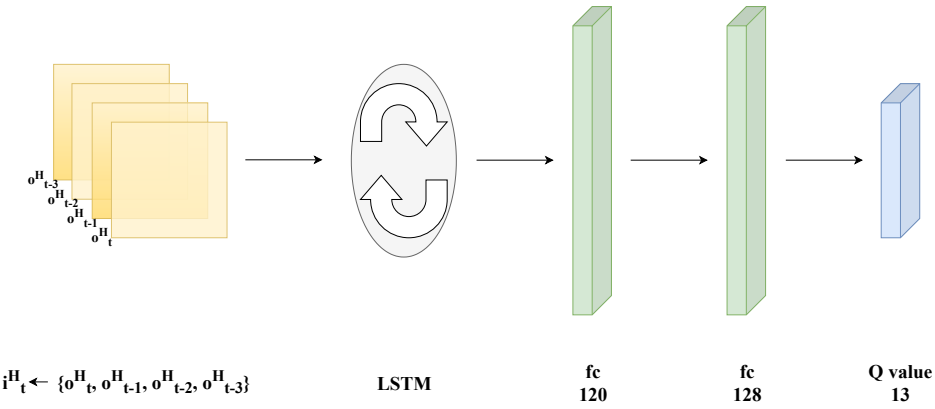


Figure 5.4: Network structure of HL policy. The input i_t^H includes the agent's current observation and also the observations from the last three HL steps. The output is the Q value of each subgoal.

capabilities. By leveraging memory units and gating mechanisms, the LSTM module enables the capture and utilisation of past information, thereby assisting the agent in making more informed policies. Specifically, we input a sequence of the four recent states $i_t^H = \{o_{t-3}^H, o_{t-2}^H, o_{t-1}^H, o_t^H\}$ into a single layer of 30 LSTM cells, facilitating the extraction and integration of relevant temporal patterns. These are followed by two fully connected layers with 120 and 128 units. The network output includes the Q values for selecting the corresponding subgoals with a given input. Alg. 2 details the algorithmic implementation of our HL policy.

5.2.2 Low-Level Policy

The LL policy is responsible for outputting the robot's control commands in order to navigate to any given short-term goal. This subsection will introduce the LL policy in terms of observation, action, reward, and network structure.

Algorithm 2: HL policy of our HRL model

Given:

- Pretrained LL policy π_{LL} ;
- HL policy π_{HL}, HL buffer D_{HL} ;

Initialise DQN of HL policy;

for $m = 0 \;\; M$ *epoch* **do**

  # Sample training environment Env 1, 2, 3...(once m is changed, sample another different environment in turn)

  for $j = 0 \;\; J$ *training steps* **do**

    $t_{HL} = 0$,

    $done = 0$

    Sample a target location g_t

    while do

      # Obtain a subgoal g_s

      $g_s \sim$ epsilon-greedy$(\pi_{HL}(i_t^H))$

      while do

        $t_{LL} = 0$

        $a_{t_{LL}} \sim \pi_{LL}(o_t^L)$

        $t_{LL} = t_{LL} + 1$

        $o_t^L = o_{t+1}^L$

        if $t_{LL} \geq T_{LL}$, *or* g_s *reached, or collision* **then**
          break

      $t_{HL} = t_{HL} + 1$

      if $t_{LL} \geq T_{LL}$, *or* g_t *reached, or collision, or* $t_{HL} \geq T_{HL}$ **then**
        $done = 1$

      $D_{HL} \leftarrow (i_t^H, g_s, R^H, i_{t+1}^H, \text{done})$

      $\lambda_{t_{HL+1}} \leftarrow$ Adam $(\lambda_{t_{HL}}, D_{HL})$

      $i_t^H = i_{t+1}^H$

      if $done = 1$ **then**
        break

Observation

The observation of the LL policy comprises five parts, formulated as $o_t^L = \{o_L||v^L||g_{p\_t}^L||a_{t-1}^L||r_{t-1}^L\}$, o_L represents the Lidar readings at the current pose, $g_{p\_t}^L$ represents the target location in the polar coordinates of the robot frame, a_{t-1}^L is the action produced by the LL policy at the last timestep and r_{t-1}^L is the reward by executing action a_{t-1}^L.

Action

This work uses a TurtleBot, which has a differential drive configuration. Therefore, the LL policy outputs the velocities of the two wheels, i.e., $a_t^L = \{v_{left}, v_{right}\}$.

Reward Function

The reward function for the LL policy is shown below:

$$R^L(o_t^L, a_t^L, g^L) = \begin{cases} r_{arrive}^L & \text{if } d_t \leq \delta^L \\ r_{collision}^L & \text{if collision} \\ r_{approach}^L & \text{otherwise} \end{cases} \tag{5.9}$$

where r_{arrive}^L is a large positive value that will be given when the robot reaches the target location, i.e., when its distance to the target d_t is within a radius δ^L; $r_{collision}^L$ penalises the policy when the robot collides with an obstacle; $r_{approach}^L = c_d(d_{t-1} - d_t)$ is the approaching reward, where $(d_{t-1} - d_t)$ is the distance difference to the target between the current and the last timesteps, and c_d is a weighting factor.

Network Structure

DDPG is used for the LL policy. The actor network for the DDPG-based agent has a single layer of 64 LSTM cells first, followed by three MLP layers, with the same size of 512. The critic network also contains a single layer comprising 64 LSTM cells and three MLP layers, where the sizes of the first and last MLP layers are both 512, and the second layer has two extra dimensions for the actions, making the size 514. Both the actor and critic networks use ReLU activation on each MLP layer, except for the output. The actor network uses hyperbolic tangent to activate the final layer, and the critic network has no activation on the output.

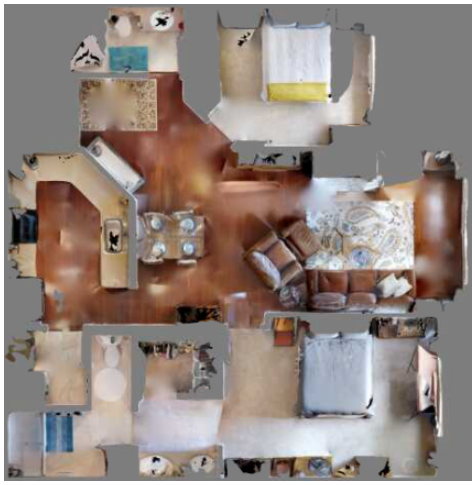
5.3 Experiments

In this work, the iGibson simulator (Shen *et al.*, 2021) is utilised. It is developed based on the Gibson dataset (Xia *et al.*, 2018) that contains 572 realistic 3D indoor environments. The robot we use for training is a TurtleBot equipped with a 360 laser beam Lidar covering a field of view (FoV) of 360 degrees.

This chapter uses a total of 10 different environments to train the HL policy and the LL policy separately. As mentioned, one main cause for failure in mapless navigation tasks is the local minimum problem. To better illustrate the effectiveness of our approach in addressing this issue, three complex indoor environments which have never been seen before are chosen for testing, as shown in Fig. 5.5. The dimensions of the environments are as follows: Env 1 is 9.5m $\times$ 8.5m, Env 2 is 12.0m $\times$ 8.0m, and Env 3 is 10.0m $\times$ 9.5m.

(a) Env 1 (Allensville)

(b) Env 2 (Artois)



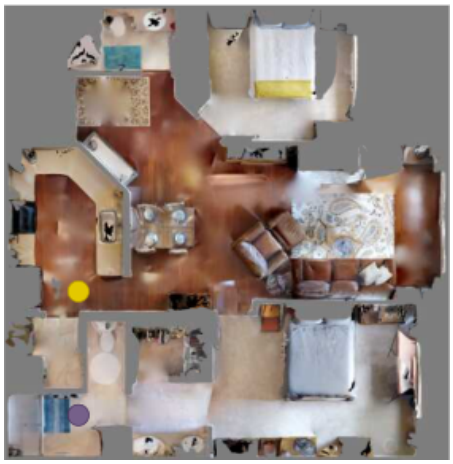
(c) Env 3 (Aulander)

Figure 5.5: Experiment environments for testing.

The tests in each environment are divided into three difficulty levels, which are defined according to the distance between the robot's initial location and the target location, ranging from 2-5m, 5-8m, and 8-10m. The greater the distance, the more difficult the task. Difficult tasks may contain complex scenarios and more likely result in local minima. We perform 500 episodes per test and record the average success rate, with the initial and target locations randomly generated by the environment. Every randomised initial state needs to satisfy the corresponding task difficulty. To ensure fair evaluation, the initial and target locations are the same across different methods when testing. In addition, to verify our method's superior performance in tackling the local minimum problem, we also choose 3 specific scenarios that are prone to this problem for demonstration, as shown in Fig. 5.6. In the three chosen scenarios, the length of the wall between the initial location and the target location is 5m, 6m and 3.5m, respectively. Each scenario is tested for 100 episodes, and the success rates are recorded.

(a) Scenario 1

(b) Scenario 2



(c) Scenario 3

Figure 5.6: Specific scenarios for testing. Purple and yellow circles represent the start location and the target location, respectively.

5.3.1 LL Policy Training

The LL and HL policies are trained separately. Considering that the reward function of the HL policy takes into account the performance of the LL policy, we train the LL policy first. In each episode, the initial location of the robot is randomised. Since the LL policy is responsible for short-range navigation, we randomly generate the target location within a square with a side of 2m centred on the robot.

The parameters in Eq. 5.9 are set as below. The arrival reward $r^L_{arrive} = 20$ is given when the robot is no more than $\delta^L = 0.36$ meters away from the target location (the chassis radius of Turtlebot is 0.36m). The collision penalty is $r^L_{collision} = -3$. The hyperparameter c_d in $r^L_{approach}$ is set to 10 empirically. The full length of an episode is 1500 timesteps. We train the LL policy for 20000 timesteps in one environment and then continue with other environments until a total of 8 million timesteps is reached. Our LL policy requires the computational time of 0.002s to compute the velocity action.

5.3.2 HL Policy Training

The HL policy training commences after the LL policy training is completed. In each episode, the robot is randomly placed at a location, and the target location is also randomly generated by the environment. However, the distance of each episode is set at 2-10m. The conditions for terminating each episode are as follows. The robot reaches the target location successfully, i.e. the distance between the robot and the target location is less than 0.86 m. In this case, the arrival reward $r^H_{arrive} = 20$ is given. When the robot collides with an obstacle, the collision penalty is set as $r^H_{collision} = -3$. If the target location cannot be reached after 200 subgoals selected by the HL

policy, this is considered too long and not worth further exploring. Also, if the LL policy cannot reach the subgoal selected by the HL policy within 800 timesteps, then the overtime penalty is $r^H_{overtime} = -3$. If the agent chooses to stand still, $R_s = -2.5$ will be given. In the intrinsic reward function, the weighting factor α is set as -0.5 and $R_0 = -2$. The boundary threshold δ^B is 0.3. These parameters are empirically set.

The HL policy is trained for 5000 steps in one environment and then switched to another environment until the total number of training steps reaches 1 million. Our proposed method takes about 0.0025s for selecting a subgoal.

5.4 Results

To investigate the performance of our method, we perform different experiments to address the following aspects.

- Choice of hyperparameter in intrinsic reward
- Overall performance of our work in comparison with other RL-based mapless navigation approaches
- Effectiveness of our proposed reward function in comparison with other reward functions used by most HRL methods
- Effectiveness of the intrinsic reward
- Effectiveness of our proposed neural network structure

5.4.1 Choice of Hyperparameter in Intrinsic Reward

In this section, we investigate the effect of the hyperparameter α in the intrinsic reward (Eq. 5.6) on training with our model. To ensure the reward (R_{ex+in}) is not infinitely small, a minimum value $R_0 = -2$ (Eq. 5.7) is set. Given the dynamic nature of the extrinsic reward R_{ex} and the memory decaying factor M_d, α primarily determines the value of N, the number of visits to the current location by the robot, against the reward size. That is, α predominantly governs the point at which the reward attains a lower bound. To investigate the effect of different α on our model, we set three values of α: -0.2, -0.5 and -2. A large N will be required for $\alpha = -0.2$ to attain the lower reward bound, but a small value for N is needed when α is -2.

In training, we test three models with different values of α every $25,000$ HL steps, with 50 episodes per test. The test success rates during training are shown in Fig. 5.7

It shows that when α is -0.5 and -0.2, the success rates tend to increase. However, the success rates are noticeably higher and remain stable at around 35%, when α is -0.5. On the other hand, when α is -2, the success rates steadily increase in the early stage, and start to decline after $500,000$ training steps, eventually dropping to about 15%. Considering the overall performance, $\alpha = -0.5$ is deployed as a suitable choice for our reward function.

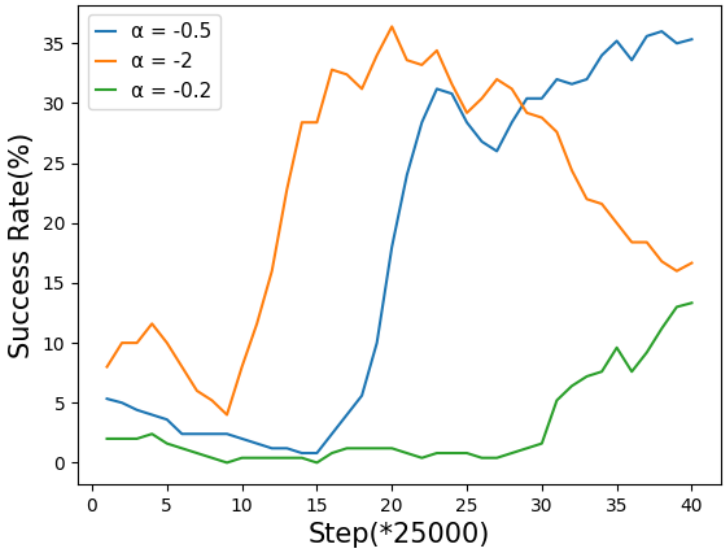


Figure 5.7: Test success rates with different values of α in the intrinsic reward during the training process.

5.4.2 Performance Comparison with Other RL-based Approaches

Comparison with Non-hierarchical Methods

We first compare our approach with two non-hierarchical RL-based methods proposed by Tai et al. (Tai *et al.*, 2017) and Marchesini et al. (Marchesini and Farinelli, 2020a) respectively. Tai et al.'s work is based on DDPG, applicable to continuous action space, while Marchesini et al.'s approach is trained with the Double DQN algorithm (Van Hasselt *et al.*, 2016), tailored for discrete space. Both methods utilise Lidar observations and the polar coordinates of the target as input. In addition, Tai et al.'s work incorporates the velocity from the previous timestep. The output of both approaches consists of the velocity commands. We train both models with the same reward functions

proposed in (Tai *et al.*, 2017) and (Marchesini and Farinelli, 2020a), respectively. In the process of training, we test each method every 25000 steps, with 50 episodes per test. The success rates are shown in Fig. 5.8. It is clear that the success rates of all methods have a rising trend. However, the success rates of our method are noticeably higher and remain stable at around 35%. The methods with continuous space and discrete space reach about 30% and 18%, respectively. Our method leverages HRL with a structure tailored for long-range navigation tasks. In addition, we incorporate novelty, episode memory, and memory decaying into the high-level reward function, which will encourage the robot to explore unknown environments and thus effectively solve the local minimum issues. In contrast, the continuous action space-based method uses a simpler training strategy with a straightforward reward function—primarily rewarding the robot for approaching the target, based on Euclidean distance and penalising collisions. This simplicity limits its success rate compared to our approach.

After training, tests are first performed in three unseen environments with different difficulty levels. The success rates of the testing tasks are shown in Table 5.1.

It is clear that our method outperforms both non-hierarchical RL-based methods in 7 out of 9 tasks. The gap becomes more notable as the task difficulty increases. In the tests with the target range of $8 - 10$m, the continuous space-based method (Tai *et al.*, 2017) and the discrete space-based method (Marchesini and Farinelli, 2020a) achieve average success rates of 23.5% and 7.5%, respectively, while our method demonstrates a much higher success rate of 30.8%. As mentioned, tasks at a higher difficulty level would be more prone to the local minimum problem. As expected, our method demonstrates its superior performance compared with the non-hierarchical

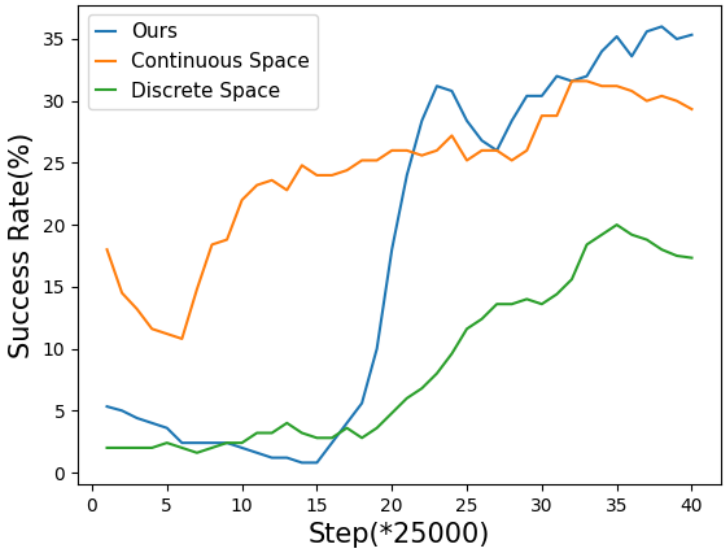


Figure 5.8: Test success rates of the continuous space-based method (Tai *et al.*, 2017), discrete space-based method (Marchesini and Farinelli, 2020a) and our method.

method.

Furthermore, as mentioned, we also test in three specific scenarios (Fig. 5.6). Both approaches fail to complete the tests in the chosen scenarios, while our method completes the tasks with success rates of 20.0%, 15.0%, and 30.0%, respectively.

Fig. 5.9 shows an example of scenario 1. The robot with the non-hierarchical method (Tai *et al.*, 2017) is trapped in a local region. In contrast, our method first moves downwards, due to the large extrinsic rewards obtained from the trajectory that shortens the distance to the target location. However, when the robot finds that the current path would not lead to the target location, the intrinsic reward encourages the robot to explore more areas instead of wandering in a local place.

Table 5.1: Performance comparison with two non-hierarchical methods in the continuous and discrete space respectively (Tai *et al.*, 2017; Marchesini and Farinelli, 2020a)

Env	Target range	Continuous space	Discrete space	Ours
1	2-5m	55.0%	38.4%	**59.4**%
	5-8m	**50.4**%	23.4%	46.6%
	8-10m	28.6%	10.2%	**38.2**%
2	2-5m	56.6%	45.0%	**59.0**%
	5-8m	37.8%	21.6%	**40.8**%
	8-10m	21.6%	7.0%	**27.6**%
3	2-5m	55.4%	45.0%	**57.8**%
	5-8m	**43.4**%	12.8%	40.4%
	8-10m	20.4%	5.2%	**26.6**%

The above results show that the non-hierarchical method struggles when faced with complex local minimum problems, whereas our method substantially improves the success rate and demonstrates effectiveness.

(a) Non-hierarchical method

(b) Our method

Figure 5.9: An example in the test of scenario 1. Purple and blue circles represent the start position and the target position, respectively. (a) The orange line represents the robot's trajectory generated by the non-hierarchical method (Tai *et al.*, 2017). (b) The orange circles are the subgoals selected by our HL policy. The numerical sequence represents the selection order.

Comparison with HRL-based Methods

HiRO (Nachum *et al.*, 2018) is one state-of-the-art HRL approach that leverages a conventional MLP HL policy, trained via an off-policy RL algorithm, TD3 (Fujimoto *et al.*, 2018). It operates in the continuous state space. The LL policy is trained to track target vectors that are generated by the HL policy. Notably, they train both policies jointly. In (Nachum *et al.*, 2018), the authors use a distance-based reward shaping technique in both policies, which, while not exactly the same as ours, shares similarities and ensures comparability with our method. We utilise the same HL (without intrinsic reward) and LL reward functions as proposed in our work. The average rewards per 10000 HL steps for training HiRO are shown in Fig. 5.10. The shaded region corresponds to the standard deviation of the rewards.

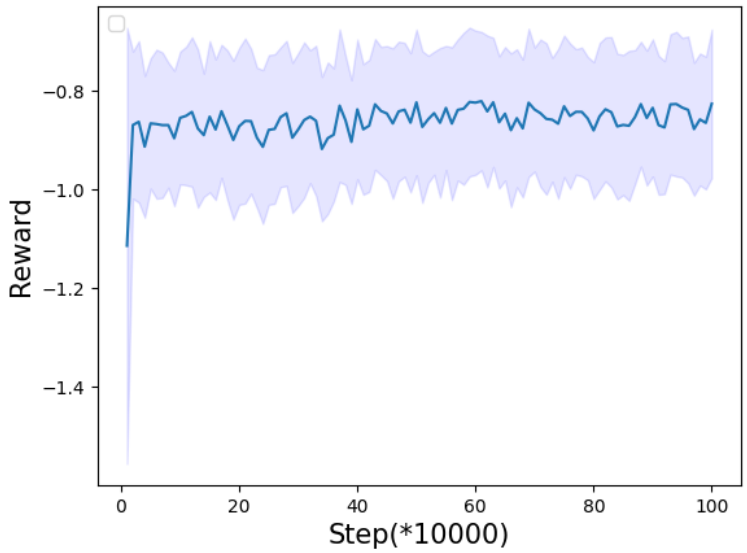


Figure 5.10: Average rewards achieved by the agent when training the HL policy of HiRO

The average rewards achieved by the agent do not show any indication

of improvement or stabilisation with increased training steps. Consequently, the results show that HiRO may struggle with complex navigation tasks. It also suggests that a continuous subgoal space may not be suited for the HL policy due to the considerable search space, thereby hindering the efficiency for policy training. Additionally, parallel training of both policies presents challenges in navigation tasks. Efficient training of the HL policy requires a stable LL policy, which may present considerable randomness in exploration during training, hence highly unstable. In the absence of a properly trained LL policy, the HL policy training may become unstable, resulting in failing to learn effective policies.

5.4.3 Effectiveness of The Proposed Reward Function

To validate the effectiveness of our proposed reward function, we train another HRL-based model. As the LL policy only controls the locomotion of the robot, we still use our LL policy for this model. For the HL policy training, we utilise the reward function commonly used by most HRL-based methods (Zhou *et al.*, 2019; Bischoff *et al.*, 2013; Staroverov *et al.*, 2020), i.e., the agent receives a positive reward for reaching the target location and is penalised for collisions. Regarding NN structure, instead of incorporating an LSTM network as with our HL agent, we replace it with two MLP layers to output the Q value for each subgoal. For evaluation purposes, we exclude the elements of novelty from the reward function and historical information for the NN. The HL inputs to the model include the current Lidar readings and the polar coordinates of the target. In the following experiments, we refer to this as the "basic HRL" method.

The results of the tests in three unseen environments and specific scenarios are shown in Table 5.2 and Table 5.3, respectively.

Table 5.2: Comparison between our method and the basic HRL-based model in the three unseen environments

Env	Target range	Basic HRL method	Ours
1	2-5m	46.8%	**58.4**%
	5-8m	32.6%	**46.6**%
	8-10m	20.4%	**38.2**%
2	2-5m	39.0%	**59.0**%
	5-8m	15.0%	**40.8**%
	8-10m	10.2%	**27.6**%
3	2-5m	51.2%	**57.8**%
	5-8m	29.8%	**40.4**%
	8-10m	13.0%	**26.6**%

Table 5.3: Comparison between our method and the basic HRL method in the three specific scenarios

Scenario	Basic HRL method	Ours
1	0.0%	**20.0**%
2	0.0%	**15.0**%
3	2.0%	**30.0**%

The results show that our method performs better in all the tasks, and this dominance is particularly evident in the specific scenarios, which can be prone to the local minimum problem. The basic HRL-based method achieves a success rate of nearly zero in the three specific tasks. Fig. 5.11 shows an

Figure 5.11: An example of the basic HRL-based method in the test of scenario 1. The orange points represent the subgoals selected by the HL policy.

example of when the basic HRL method is tested in scenario 1. The agent loiters in a local region; however, our method enables the agent to escape from the dilemma (Fig. 5.9).

It supports our hypothesis that our proposed reward function and the NN structure can encourage robots to explore unknown environments and enable them to have certain memory and reasoning abilities.

Furthermore, we conduct a comparative analysis of the training time. Both approaches entail substantial computational resources and necessitate considerable time for training. Specifically, our method demands more training time, approximately 216 hours. Conversely, the basic HRL method requires a shorter time of about 168 hours. We employ a workstation equipped with an Intel i9-10900X CPU (3.7GHz x 20) and an Nvidia RTX-2080 TI GPU to conduct our experiments. As described, 10 environments have been

selected for training in our work. In the training, we switch to a new environment after each training session. This requires frequent reloading and initialisation of new environments in the iGibson simulator. This process requires about 34.08 seconds, thus time-consuming. In addition, the HL policy needs to wait for the completion of the task of moving the robot to the subgoal under the control of the LL policy for each HL training step. As such, the total training time is considerably long. The longer training time required by our method is partially due to the fixed number of HL training steps (1 million) for both methods. Since our method encourages the agent to explore, it will select more subgoals in various locations. Therefore, the probability of the agent colliding or choosing an inappropriate subgoal increases. We terminate the current episode and start the next episode when the robot fails to reach the selected subgoal. This process involves re-initialisation, which introduces additional time overhead and prolongs the overall duration of training. In contrast, the basic HRL method repetitively selects 'safe' subgoals that may lead to local minima, as shown in Fig. 5.11. As a result, the episode does not end prematurely, requiring fewer episodes in total and, consequently, reducing the training time.

5.4.4 Ablation Study: Intrinsic Reward Metric

In this section, we study the effect of our proposed intrinsic reward in addition to the extrinsic reward, which is commonly deployed in previous work (Tai *et al.*, 2017; Zhelo *et al.*, 2018). Specifically, we train our HL policy without the intrinsic reward and the LL policy remains the same. Due to the lack of intrinsic rewards in the reward function, the HL reward will not be changed dynamically based on past states and the subgoal selection is only related to current observation. Therefore, the HL input includes its current Lidar

readings and the polar coordinates of the target with respect to the robot. Also, the LSTM network is removed to increase its learning efficiency. The success rates of the tests in the three unseen environments and the three specific scenarios are shown in Table 5.4 and Table 5.5, respectively.

Table 5.4: Comparison between our method and the method without the intrinsic reward in the three unseen environments

Env	Target range	Without intrinsic reward	Ours
1	2-5m	50.0%	**58.4**%
	5-8m	39.2%	**46.6**%
	8-10m	22.4%	**38.2**%
2	2-5m	40.0%	**59.0**%
	5-8m	20.4%	**40.8**%
	8-10m	13.6%	**27.6**%
3	2-5m	55.4%	**57.8**%
	5-8m	35.4%	**40.4**%
	8-10m	18.4%	**26.6**%

Table 5.5: Comparison between our method and the method without intrinsic reward in the three specific scenarios

Scenario	Without intrinsic reward	Ours
1	0.0%	**20.0**%
2	0.0%	**15.0**%
3	3.0%	**30.0**%

Figure 5.12: An example of the method without intrinsic reward in scenario 1. The orange points represent the subgoals selected by the HL policy.

The results show that our method has significantly higher success rates in all cases, suggesting that our proposed intrinsic reward plays an important role. Fig. 5.12 shows an example without the intrinsic reward tested in scenario 1. The robot moves downwards in order to obtain higher extrinsic rewards, and, due to the lack of exploration motivation and memory mechanisms, the robot will repeat the same path.

We can observe that the extrinsic reward is effective in most navigation tasks. However, the performance deteriorates in the presence of local minima, where the role of our intrinsic reward becomes more effective.

5.4.5 Ablation Study of The Proposed HL Network

To validate the design of the proposed HL network, we conduct two ablation experiments: 1) removing the LSTM network from our HL network, and 2) replacing the LSTM network with another network, named Frame Stack-

ing (Mnih *et al.*, 2015; Tian *et al.*, 2017), which enables the agent's memory capability.

Comparison Between Our Method and The NN Structure Without LSTM Network

We remove the LSTM network from our HL NN. The Q value of each subgoal is obtained through two MLP layers based on the HL input i_t^H, which only includes the current observation o_t^H. The rest, including the HL reward functions, training strategies, etc., are identical to our HL policy. In addition, the LL policy is also the same as ours. The average rewards per 10000 HL steps for our methods and the one without LSTM are shown in Fig. 5.13. The shaded region of each curve corresponds to the standard deviation of the rewards.

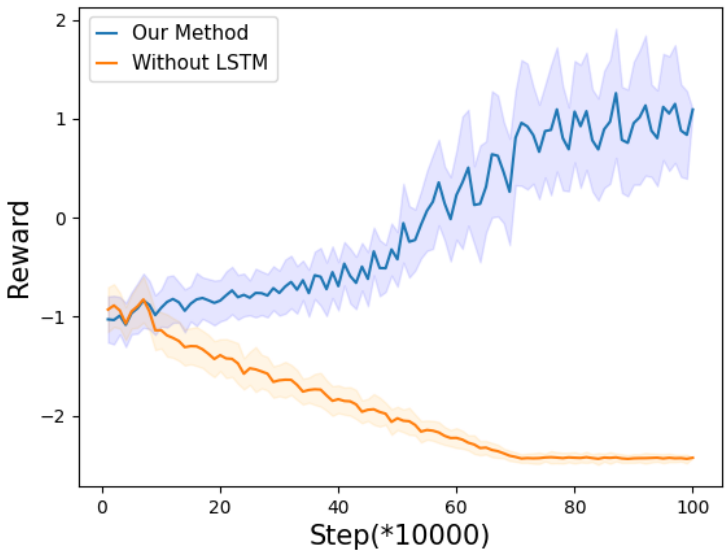


Figure 5.13: Average reward for training steps using our method and the NN structure without LSTM.

The figure shows that, without the LSTM network, the agent cannot learn an effective navigation policy. This is mainly due to the dynamic nature of our reward function. Consequently, the agent might select the same subgoal for identical observations but may receive varying rewards. In the absence of pertinent memory and prior experiences, this could lead to inconsistency of state-action mapping, failing to optimise the agent's policy efficiently.

Comparison Between Our Method and The NN Structure Using Frame Stacking

We replace the LSTM network with another widely used method, Frame Stacking, which is an effective way to enable NNs to learn temporal context knowledge (Vanhoucke *et al.*, 2013; Tian *et al.*, 2017; Mnih *et al.*, 2015). Frame stacking is a kind of frame re-segmentation, which stacks temporal neighbouring states to form a state (Tian *et al.*, 2017). The inputs to the HL model include not only the current observation but also the last three HL steps' observations, denoted as $i_t^H = \{o_t^H || o_{t-1}^H || o_{t-2}^H || o_{t-3}^H\}$. The rest is identical in terms of the reward function, training strategy, LL policy, etc. The average rewards per 10,000 steps for our method and the method with frame stacking are shown in Fig. 5.14.

From the figure, we can observe that the average reward of the method with frame stacking would not continue growing within its training time (1 million steps), indicating that it could not learn an effective policy and may require more training steps and resources. In contrast, the average reward of our model is steadily increasing, proving that the use of the LSTM network greatly improves learning efficiency.

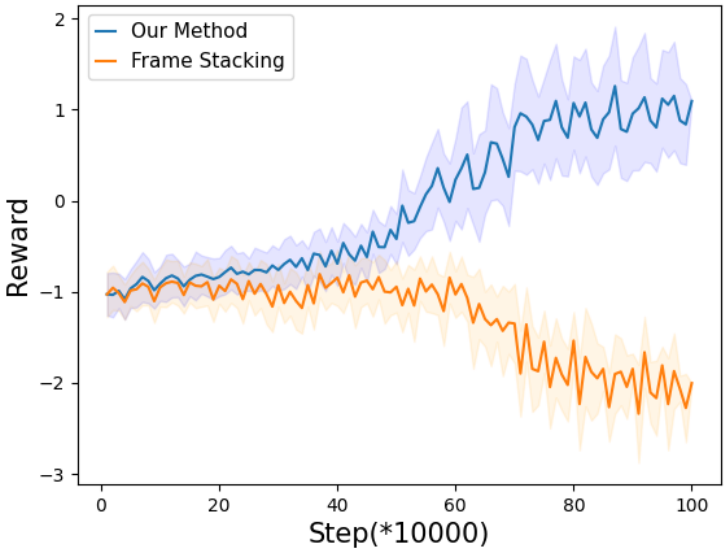


Figure 5.14: Average reward for training steps using our method and the NN structure utilising frame stacking.

5.4.6 Real World Experiments

Experiments were conducted to demonstrate the effectiveness of the proposed method in real-world environments. The policy, trained within simulation environments, was deployed directly onto a mobile robot (Turtlebot 3) without any modifications. As illustrated in Figure 5.15, the robot is equipped with two independent controllable wheels and a Lidar sensor (Laser Distance Sensor LDS-01), capable of 360-degree sensing, mirroring the simulation setup. The Robot Operating System (ROS) platform is used as the robot control and communication framework in our real-world experiments. Each wheel's velocity is computed and commanded by a laptop. It takes about 0.007s for ROS to export the velocity command to Turtlebot. In addition, as the robot is operated under direct speed-mode control, the robot maintains its cur-

rent velocity until a new command is received. The LL policy outputs zero motion only upon reaching a subgoal, ensuring smooth transitions between consecutive subgoals. The total computational and communication time is approximately 0.01 seconds, during which the robot maintains its velocity. Due to the short interval and low-speed operation, the computational time will not have an impact on the robot's motion. The experimental setup featured a target location positioned behind a long wall, approximately 6 meters in length, designed to test the proposed approach's ability to overcome local minima. In this scenario, the robot autonomously explores to first locate the door—the sole access point to the adjacent room—before proceeding to enter.

For comparative analysis, the non-hierarchical method was evaluated. Figure 5.15c shows that, using this method, the robot initially moves straight towards the goal but soon finds itself trapped in a local area close to the goal yet obstructed by a wall. In contrast, our method prompted the robot to explore more extensively, enabling successful navigation into another room. Figures 5.15a and 5.15b depict the robot exploring various subgoals along the corridor in both forward and backward directions, ultimately finding the door leading to the goal. This observation underscores our proposed framework's ability to encourage further exploration and effectively prevent revisiting already-explored areas.

(a) Our method - 1

(b) Our method - 2

(c) Non-hierarchical method

Figure 5.15: Experiments in a real environment with (a)-(b) our method and (c) the non-hierarchical method (Tai *et al.*, 2017).

5.5 Conclusion

This chapter proposes an HRL-based mapless navigation approach for complex tasks. A novel reward function and a neural network are introduced for training the HL policy. The proposed HL reward function consists of two components, namely extrinsic reward and intrinsic reward. The extrinsic reward is provided by the environment and encourages the robot to move towards the target location. The intrinsic reward is inspired by the novelty theory (Ruan *et al.*, 2022), which suggests that animals will reward themselves for identifying novelty. We determine the magnitude of the intrinsic reward by the novelty of the current state. To decide the novelty of a state, we introduce a count-based method according to episode memory. In addition, to enhance the robot's ability to navigate in complex environments, we incorporate the memory decaying mechanism into the intrinsic reward. Also, some penalty elements are included in the overall HL reward function to ensure the LL policy's performance is considered when selecting subgoals. In addition, an LSTM network is proposed to equip the agent with memory and reasoning capabilities.

Experimental results and analyses highlight the effectiveness of the proposed reward function and the NN structure in learning the navigation policy. Especially in complex tasks, the proposed approach performs better and significantly improves the success rate.

Chapter 6

Deep Reinforcement Learning for Localisability-Aware Mapless Navigation

6.1 Introduction

DRL-based mapless navigation methods usually construct a direct mapping between robot states and actions. In most works, the RL agent state comprises two primary components: the sensory observations and the position information of the target location, which is usually represented in terms of polar coordinates relative to the robot's frame (Tai *et al.*, 2017; Zhelo *et al.*, 2018). To obtain the accurate polar coordinates of the target, reliable robot localisation is necessary.

Such studies typically assume that robots have access to their ground-truth poses and focus solely on path planning (Tai *et al.*, 2017; Zhelo *et al.*, 2018), including the works presented in Chapters 4 and 5. However, the performance of localisation is highly dependent on the environment. Techniques of visual simultaneous localisation and mapping (SLAM) highly rely on the tracked visual features for global navigation satellite system (GNSS)-denied environments; while, for outdoor, GNSS-based localisation is prone to multi-path reflections of radio signals caused by the environment. The ignorance of localisation performance would lead to robots failing to localise themselves and possible catastrophic issues caused by decision-making based on unreliable state estimation. This chapter believes that the perception module for localisation should be closely coupled with decision making in the process of path planning, which should consider not only commonly used criteria such as path length, but also the richness of features in the surroundings that could greatly affect the localisation performance.

However, even state-of-the-art ORB-SLAM2 (Mur-Artal and Tardós, 2017) often suffers from localisation failures. Its performance may be degraded in situations where environmental features are poorly observed, such as when navigating in areas lacking distinct features. In addition, localisation failures

could occur even in feature-rich scenarios, i.e., when encountering rapid rotations or movement (Naveed *et al.*, 2022). Therefore, avoiding SLAM failures in navigation is necessary (Naveed *et al.*, 2022; Prasad *et al.*, 2018; Lin *et al.*, 2021, 2022).

However, ensuring the effectiveness of SLAM in navigation is nontrivial and needs several considerations. Firstly, some studies train RL models using the current observed image features or manually extracted features as input (Naveed *et al.*, 2022; Prasad *et al.*, 2018). They aim to assess the discrete actions and identify which action may fail the SLAM through RL. Image features inherently contain richer information; however, these representations often exhibit poor generalisation when encountering unseen environments. Secondly, some works have integrated the prevention of SLAM failures into the reward function when training an RL-based navigation model. Specifically, they incorporate Absolute Trajectory Error (ATE) as a factor. They impose a penalty on the agent based on the disparity between the robot pose returned by SLAM and the ground truth (Chen *et al.*, 2023b). However, the pose estimation error is the accumulated error from a sequence of actions executed in the past steps, rather than the result of the last executed action. Consequently, the agent may still incur a delayed penalty on the latest action caused by the accumulated error. On the other hand, discrete action space is more commonly employed for such problems (Chen *et al.*, 2023b; Lin *et al.*, 2021, 2022; Naveed *et al.*, 2022; Prasad *et al.*, 2018). To enable more flexible manoeuvring motions in complex and diverse environments, continuous action space would be more desired.

With these considerations, this chapter proposes a novel DRL-based mapless navigation approach with a continuous action space. Rather than assuming the availability of localisation, this method uses ORB-SLAM2 (Mur-Artal

and Tardós, 2017) with an RGB-D camera for localisation in indoor environments. This method aims to generate an effective policy for the control of the robot's movement towards the designated target, with the capabilities of obstacle avoidance and enhanced robot pose estimation by considering the quality of the observed features along the selected paths, hence consequently leading to optimal SLAM performance. Instead of directly using raw sensory observations as the state input, this work adopts a more impact representation. Many works have proved that enough overlap with already mapped points can prevent SLAM failure (Prasad *et al.*, 2018; Naveed *et al.*, 2022). On the other hand, it is more desirable to have more ORB features that can be detected by the robot for better localisation. In addition, a more uniform distribution of features in space can enhance the alignment of keyframes, thus improving the localisation accuracy (Chen *et al.*, 2023a). Firstly, map points are categorised into two classes: the currently tracked and those previously tracked and stored in the map. Then, the information about the distribution of these map points is extracted and incorporated into the state input. It may enable the agent to discern areas with reliable map points. Furthermore, this chapter introduces a novel reward function that incorporates Relative Pose Error (RPE) as a penalising factor. RPE can offer a more intuitive representation of the impact of SLAM resulting from each action. In addition, SLAM performance varies across different environments (Taheri and Xia, 2021). Therefore, instead of employing a pre-set threshold and penalising the robot when the RPE surpasses the threshold, this work adopts a dynamic threshold, wherein the threshold is influenced by the prevailing environmental conditions.

The rest of the chapter is organised as follows. Section 6.2 introduces the proposed method, especially focused on the sets of state representation,

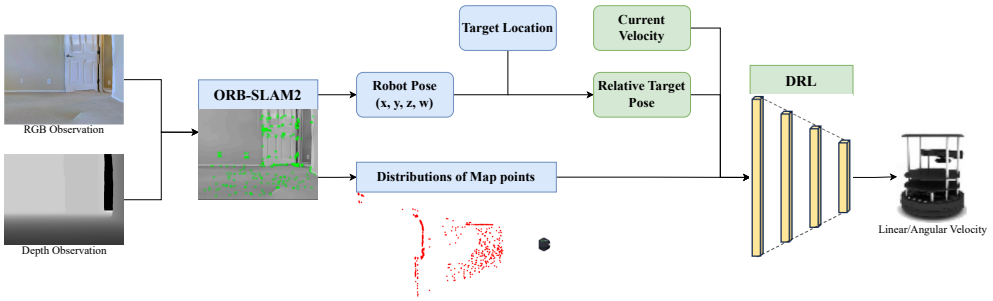


Figure 6.1: The overall framework. ORB-SLAM2 determines the robot pose through the current RGB-D observations and subsequently acquires the polar coordinates of the target location. Additionally, our proposed method captures information on the distributions of map points derived from SLAM. The input for the DRL policy comprises the target location, the distribution of map points, and the current velocity of the robot. The output consists of linear and angular velocity commands.

reward function, dynamic threshold and so forth. In section 6.3, the experiment design and the results are introduced. Finally, section 6.4 concludes this chapter.

6.2 Methods

This section introduces the proposed method, focusing on the design of state and reward representations.

6.2.1 State Representation

This method is based solely on an RGB-D camera. A compact and informative state representation is essential for effective and efficient training. Directly using raw RGB-D observations is impractical due to their high re-

dundancy and dimensionality. Our proposed representation O_t comprises four components denoted by $O_{dis}, O_{tgt}, O_{dep}, O_{vel}$.

O_{dis} contains essential information about the spatial distribution of tracked map points that are crucial for the error reprojection optimisation process, formulated in Eq.3.31. Previous studies show that tracking sufficient features or map points between consecutive frames is essential for reliable SLAM performance (Naveed *et al.*, 2022; Prasad *et al.*, 2018; Lin *et al.*, 2021). To ensure consistent map points being tracked, the robot would directly benefit from the awareness of the spatial distribution of the tracked map points. In this work, we partition the map points into two categories: 1) those that are currently tracked, denoted as T; and 2) those that are previously tracked and stored in the map, denoted as M.

The T distribution enables the robot to assess the current view's feature distribution for decision-making. As an example (Fig. 6.2), if T's distribution is predominantly concentrated on the right side of the robot, opting for a left turn might be deemed as a less favourable action, as the robot will lose track of these features. M provides contextual information of the map points of the surroundings. Such contextual information endows the robot with the predictive capability of future observations. With the same example above, if a significant quantity of M is located on the left side of the robot, but untracked currently. Executing a left rotation may not result in SLAM failure due to the pre-tracked features.

To effectively encapsulate the map point spatial distribution in a structured format, we partition the surrounding area into 24 regions in the polar coordinate frame centred on the robot, as illustrated in Fig. 6.2. Each map point is assigned to a corresponding region based on its location with respect to the robot's frame. Given that the horizontal angle of the RGB-D camera

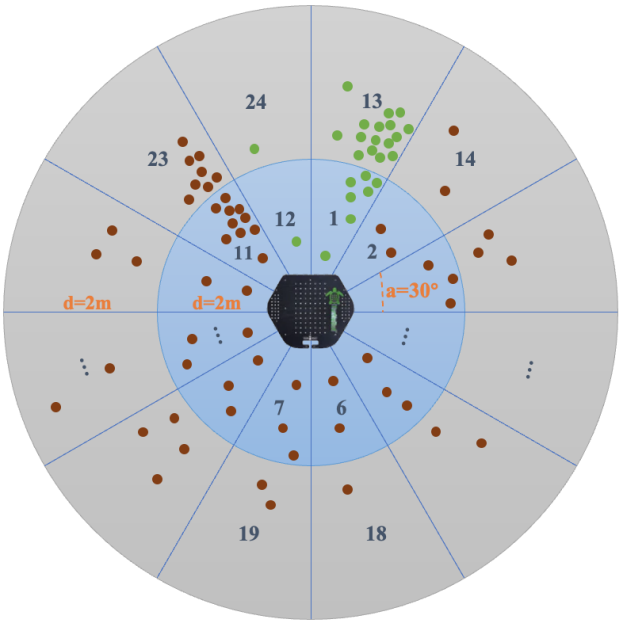


Figure 6.2: Illustration of spatial distribution representation of map points. The green points represent the currently tracked map points (distributed within regions 1, 12, 13, and 24). The brown points represent those previously tracked and may be distributed across any region.

(Xtion PRO LIVE) is 58 degrees, points in T are primarily distributed in regions 1, 12, 13, and 24. However, the points in M can be in all regions $(1-24)$. Subsequently, we extract the distribution information of T and M map points in each region, represented as:

$$\eta(p) = [N, \sigma_1, \sigma_2, \sigma_3] \tag{6.1}$$

where p represents the map points of interest. N is the number of p, and σ denotes the variances of the map point coordinates in three axes, respectively. The reason for including the variance is that they represent the distribution of p across three axes. A large variance means the data range is large on the corresponding axis; while a smaller value means the data are concentrated.

Therefore, O_{dis} can be formulated as:

$$O_{dis} = [\eta(T_1), \eta(T_{12}), \eta(T_{13}), \eta(T_{24}), \eta(M_1), \eta(M_2), \ldots, \eta(M_{24})] \tag{6.2}$$

where $T_{1,12,13,24}$ represents the T map points in region 1, 12, 13, and 24 respectively. Similarly, M_{1-24} is the M map points in region $1-24$.

O_{tgt}, O_{dep}, and O_{vel} are used to generate the policy that controls the robot's move to the target location and avoid collisions. O_{tgt} is the relative position of the target, represented in the polar coordinates with respect to the robot's frame ($O_{tgt} = [d, \theta]$). The pose is obtained from ORB-SLAM2 and is converted to the polar coordinates.

Regarding obstacle avoidance, many works show that the task can be effectively achieved using only range sensors (Tai *et al.*, 2017; Zhelo *et al.*, 2018; Marchesini and Farinelli, 2020a). The RGB-D camera, capable of extracting depth information, can be considered as a range sensor. Since the depth image D is represented as an $m \times n$ matrix, to be effective in training the policy, rather than using the raw depth data, we converted the matrix into a 1D vector $\mathbf{k}$, with the dimension of $1 \times n/p$ (p is a constant), simulating a 2D range sensor.

$$\mathbf{k} = \begin{bmatrix} \min(D[a:b, 0:(p-1)]) \\ \min(D[a:b, p:(2p-1)]) \\ \min(D[a:b, 2p:(3p-1)]) \\ \vdots \\ \min(D[a:b, (((n/p)-1)p):((n/p)p-1)]) \end{bmatrix}^T \tag{6.3}$$

where min returns the lowest number of an array, and $D[a:b, \cdots]$ returns the sub-column of each column, ranged from index a to b. Here, $a = m/2$ and $b = 3m/4$. The range from a = m/2 to b = 3m/4 in the depth map is

chosen based on the TurtleBot's height and the relevance of depth data for obstacle avoidance. The top half of the depth map (0 to m/2) is typically too high for the low-mounted robot to interact with, so it is disregarded. The lower portion (3m/4 to m) likely corresponds to the ground, which does not significantly impact obstacle avoidance. By focusing on the middle region, we ensure efficient processing while retaining key information for avoiding obstacles at the robot's height, optimising both performance and computational resources.

Last, O_{vel} denotes the robot velocity, including both the linear and angular velocities, which are essential in guiding the policy to decide its subsequent steps. Therefore, the state representation is formulated as

$$O_t = \{O_{dis}||O_{tgt}||O_{dep}||O_{vel}\} \tag{6.4}$$

where || denotes vector concatenation.

6.2.2 Reward Function and Network Structure

In this work, in addition to typical mapless navigation rewards, such as rewarding the agent on its arrival at the target or penalising actions of collision, we also included two other localisation-related rewards. The agent will be penalised when the estimated pose is diverging from the true trajectory, or when SLAM fails to track its trajectory.

$$R = \begin{cases} r_{arrive} & \text{if } d_s \leq \delta_s \text{ and } d_t \leq \delta_d \\ r_{collision} & \text{if collision} \\ r_{pose} & \text{if } e_t \geq \delta_p \\ r_{lost} & \text{if SLAM fails} \\ r_{approach} & \text{otherwise} \end{cases} \tag{6.5}$$

A positive reward, r_{arrive}, is assigned if the robot reaches the target. A two-threshold criterion is introduced. d_s and d_t represent the distances between the robot and the target, where d_s is estimated using SLAM, and d_t is the ground-truth position. δ_s and δ_d specify the corresponding distance thresholds. Allowing for a certain amount of error during ORB-SLAM2 runs, δ_d is slightly larger than δ_s. If the robot collides with an obstacle, a negative reward, $r_{collision}$, is assigned.

The reward r_{pose} is introduced to reward or penalise the agent based on the impact on the SLAM performance. RPE, which measures the drift of pose estimates from the ground truth over a fixed time span, is used to evaluate the performance of SLAM. Assuming that the ground-truth relative poses between $Q_{i-\Delta t}$ and Q_i is $Q_{i-\Delta t}^{-1} Q_i$ and the estimated relative poses between $P_{i-\Delta t}$ and P_i is $P_{i-\Delta t}^{-1} P_i$. The RPE is formulated below:

$$e_i = |trans((Q_{i-\Delta t}^{-1} Q_i)^{-1} (P_{i-\Delta t}^{-1} P_i))| \tag{6.6}$$

where $trans$ denotes the translation error, which is considered here.

RPE is employed as the representation of the error induced by the corresponding action. In this work, we consider the RPE error in one timestep, $\Delta t = 1$. When RPE surpasses a pre-defined threshold, it indicates that the last action has led to inaccurate pose estimation, and will receive a penalty. However, determining a reliable threshold is challenging, due to the substantial variation across different environments. Instead of setting a fixed threshold, we employ a dynamic threshold approach, by checking if the current RPE lies within an acceptable range. Specifically, we calculate the mean value and standard deviation of the RPEs of the last τ timesteps to provide an estimated range of RPE distribution of the environment. Then, the threshold, δ_p, is defined as:

$$\delta_p = \mu(E) + \alpha\ \sigma(E), \tag{6.7a}$$

$$E = (e_{t-\tau}, e_{t-(\tau-1)}, ..., e_{t-1}) \tag{6.7b}$$

where α is a constant, $\mu()$ represents the mean value and $\sigma()$ denotes the standard deviation.

If e_t exceeds the threshold δ_p, this is considered as surpassing the acceptable range. This approach endows the agent with the capability to adapt to diverse environments by providing the agent with an adaptive estimation of localisation performance in the current environment. When the RPE exceeds δ_p, a negative reward r_{pose} is applied. In addition, if SLAM encounters a complete failure, i.e., the number of currently tracked features is 0, r_{lost} is applied to the agent.

The reward $r_{approach}$ is defined as the distance difference from the target to the robot between the last timestep and the current timestep, $r_{approach} = c_r(d_{t-1} - d_t)$, where c_r is a hyperparameter. This encourages the robot to move towards the goal.

Regarding the network structure, as mentioned, a DDPG-based method is deployed in our work. The actor network in the DDPG framework consists of three Multi-layer Perceptron (MLP) layers, each with a uniform size of 512. The critic network also comprises three MLP layers. The size of the first and last layers is 512, and the dimension of the second layer is 514, incorporating two additional dimensions for the action. ReLU activation is applied to each layer in both the actor and the critic networks, except for the output layers. The actor network utilises hyperbolic tangent activation for the last layer, whereas the critic network has no activation applied to the output.

6.3 Experiments

6.3.1 Training in Simulation

The simulation environment, iGibson, is used in this work (Shen *et al.*, 2021), providing a range of photorealistic 3D indoor environments. The training procedure of our model is performed in a localisation-challenging environment, such as Env 1 in Fig. 6.3.

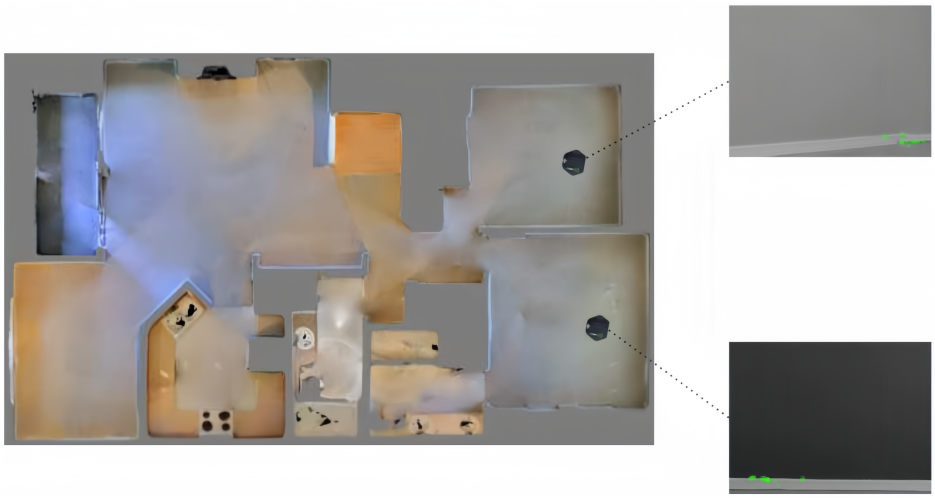


Figure 6.3: Env 1: Aloha and two examples of the robot visual features.

The rewards are set as follows: $r_{arrive} = 20$, $\delta_s = 0.5$, $\delta_d = 0.7$, $r_{collision} = -5$, $r_{lost} = -5$. For Eq. 6.7, α is 3, τ is 50. c_r is set as 100 in $r_{approach}$. These parameters are empirically determined through trial and error. The action space is continuous. The linear velocity is set to be positive, ranging from 0 to 0.06 m/s, whereas the angular velocity varies within the range of -0.1 to 0.1 rad/s.

The policy is trained for 5 million steps. It requires a large number of training steps primarily because of the complexity of the task. First, the state space in this problem is large, with numerous possible configurations

of the environment and the agent's state. This means that the model has to explore a vast number of possible states and learn the appropriate actions for each one. Additionally, the policy that needs to be learned is typically quite complex. Effective navigation involves not only making immediate decisions but also long-term planning, which requires the model to understand and optimise action sequences over extended periods. This complexity in both the state space and the required policies slows down training and necessitates a large number of interactions with the environment to converge on an optimal policy. In each episode, the start location is randomised and the target is also randomly sampled in the environment, with the condition that the target must be 1 to 5 meters away from the start location. The episode terminates when any of the following conditions are met: 1) SLAM fails, i.e., insufficient valid map points can be effectively tracked; 2) The error in either the estimated x-coordinates or y-coordinates from the ground truth exceeds 0.15, or the error in the estimated orientation is greater than 0.5; 3) The robot collides or surpasses the time limit (3000 tiemsteps); and 4) The robot successfully reaches the target location. Successful arrival at the goal is determined by the following two conditions: 1) the distance from the estimated position to the target is less than 0.5m, and 2) the distance from the ground-truth position to the goal is less than 0.7m. The work was performed on a workstation with an Intel i9-10900X CPU (3.7GHz x 20) and an NVidia RTX-2080 TI GPU, which took about 145 hours.

6.3.2 Evaluation

Baseline

To validate the effectiveness of our method, we select several approaches as the baselines.

- *Naive policy (NP-GT)*: We train a simple mapless navigation model, which uses the ground truth as the robot pose, obtained from the simulation environment. Since ground truth poses are used, this policy does not need to consider the accuracy of the poses. Therefore, the input representation contains O_{tgt}, O_{dep}, and O_{vel}. We use the same reward function proposed in (Tai *et al.*, 2017).

- *Naive policy (NP-SLAM)*: In contrast to the NP-GT, the robot poses are supplied by ORB-SLAM2. The rest remains the same as the method above.

- *Policy with image input (P-Image)*: In (Naveed *et al.*, 2022), raw image observation is employed as the RL input and subsequently identifies which action may lead to SLAM failures. Inspired by this approach, we utilise the same neural network structure to extract image features as inputs for our method. For a fair comparison, the reward function is the same as ours.

- *Policy trained by ATE-based reward function (P-ATE)*: In (Chen *et al.*, 2023b), a reward metric is proposed to provide feedback on actions that affect localisation accuracy. The difference between the ATE errors at the previous and the current timesteps is utilised for the reward. Additionally, the agent incurs a substantial penalty if the current error surpasses a predefined threshold. For comparison, we train a model

using its reward function, but the state representation is the same as ours for a fair comparison.

Results

In the process of training, we test each method every 100000 training steps, with 100 episodes per test. The success rates are shown in Fig. 6.4. *NP-GT* achieves the highest success rate, eventually stabilised at around 65%. The success rate of *NP-SLAM* is significantly lower and shows a declining trend after 4 million training steps, reaching about 30%. In contrast, our approach demonstrates a notably higher success rate, approximately 20% higher than that of *NP-SLAM*. The success rates of *P-ATE* and *P-Image* do not increase with the number of training steps, and ended up at about 5% and 3% respectively.

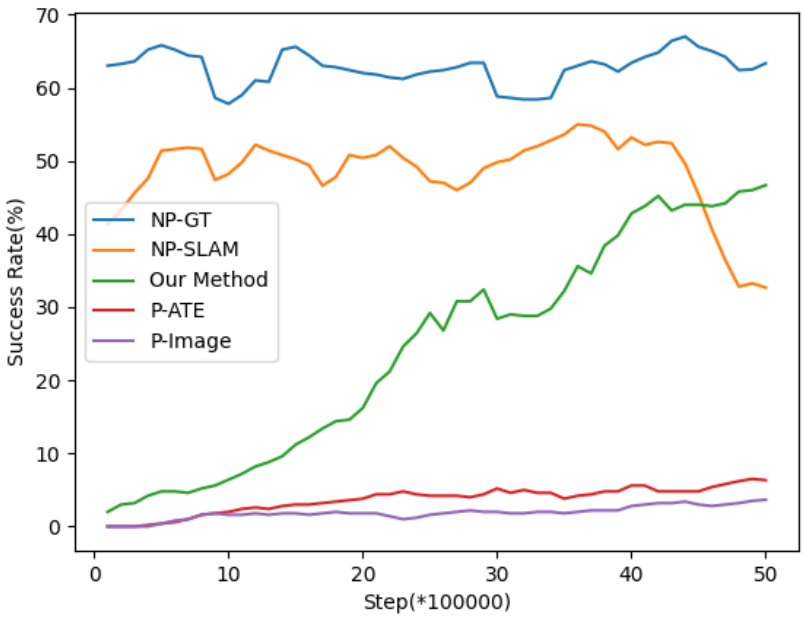


Figure 6.4: Success rates achieved by different methods.

After training, we conducted performance evaluation on our method and the baselines in Env 1. For each method, 100 tests were performed. The initial position of the robot and the target are generated randomly for each test. The same start and goal positions are used for different algorithm configurations to ensure fair comparisons. The success rates of the testing tasks are shown in Table 6.1.

Table 6.1: Test success rates in Env 1.

Method	Success Rate
NP-GT	61%
NP-SLAM	33%
Our method	**49**%
P-Image	2%
P-ATE	5%

It is obvious that the highest success rate of 61% is achieved when the robot pose is precise. In contrast, when relying on SLAM for pose estimation, the success rate exhibits a considerable decrease, falling to 33%. This suggests that the robot's movements, generated by *NP-SLAM*, fail to ensure accurate SLAM localisation, consequently, resulting in navigation failure. Our proposed method has the highest success rate among the related baselines, surpassing the *NP-SLAM* by 16%. Our policy demonstrates the capability to enhance the robot pose estimation.

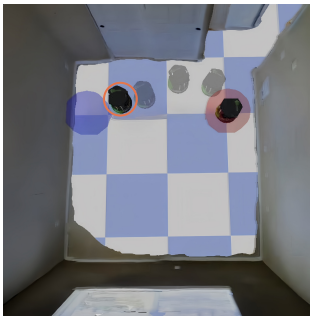
Fig. 6.5 shows two instances where our method successfully completes the task, while the *NP-SLAM* leads to SLAM failure. In the first example (Fig. 6.5a and Fig. 6.5b), the left and right sides of the environment are

walls with poor features, while the upper and lower parts contain various environmental features. The *NP-SLAM* directs the robot to opt for the shortest path. However, when confronted with an entire wall that lacks sufficient features, SLAM failure occurs. In contrast, with our method, the robot traverses a path that may not be the shortest, but allows the robot to observe more environmental features, thereby ensuring the efficacy of SLAM.

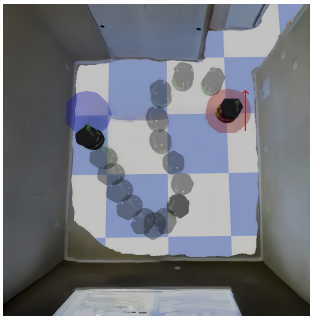
In the second example (Fig. 6.5c and Fig. 6.5d), the initial heading direction is opposite to the target. The policy initially directs the robot to execute a large-angle rotation in place, before it proceeds to move towards the target. However, the substantial changes in environmental perception during the rotation result in a significant error in SLAM localisation. Consequently, the task fails due to the considerable localisation error incurred during navigation. In contrast, our approach avoids the large rotation and guides the robot along a path closer to the door, leveraging a greater number of visual features to enhance the accuracy of its localisation.

Furthermore, utilising the ATE-based reward function proposed in (Chen *et al.*, 2023b) yields a mere 5% success rate, suggesting its ineffectiveness in training the policy. Penalising the agent based on the discrepancy between the current estimated pose and the ground truth is inappropriate. This is because the error may not arise solely from the last executed action but could be accumulated from a sequence of past actions executed. Moreover, applying a uniform criterion to assess SLAM performance limits its generalisability, as the performance of SLAM varies significantly across diverse environments. Judging whether the localisation fails or not using identical criteria is imprecise. The experimental results further validate the efficacy of our proposed reward function.

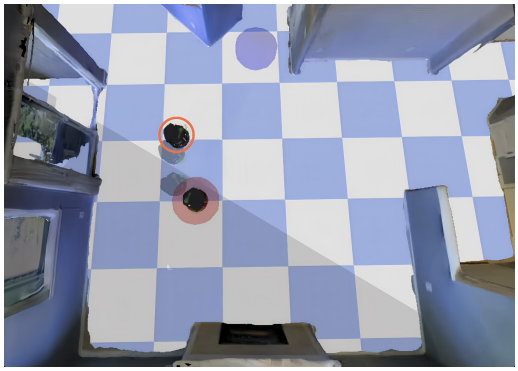
Additionally, in accordance with our expectations, our proposed method



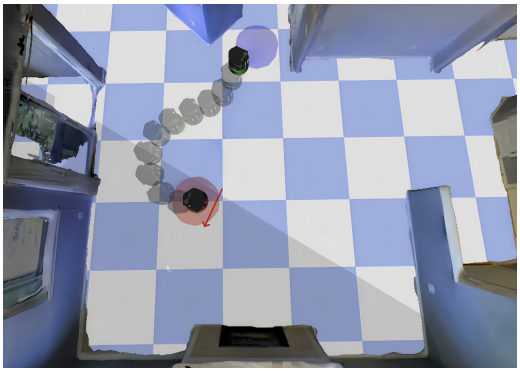
(a) Case 1 - NP-SLAM



(b) Case 1 - Our Method



(c) Case 2 - NP-SLAM



(d) Case 2 - Our Method

Figure 6.5: Two examples of SLAM failures caused by the *NP-SLAM*. The red and blue circles denote the starting and target locations, respectively. The arrow represents the initial direction of the robot. In (a) and (c), the orange circle represents the locations of the SLAM failures.

outperforms the method using raw images as inputs in terms of success rate. This is mainly due to the high dimensionality of raw RGB images, where the policy struggles to extract useful information for the task. In addition, this may require more training steps and resources for training. The result suggests that our proposed input representation is more effective in training the policy.

To verify the capacity of generalisation of each method, we test each in an unknown environment, (Env 2 as shown in Fig. 6.6). Each method is tested for 100 episodes, and the corresponding success rates are presented in Table 6.2.

Figure 6.6: Env 2: Arona and two examples of the robot visual features.

Since Env 2 has a simpler layout than Env 1, each method exhibits a higher success rate. Similarly, the success rate is the highest when the robot pose is true and decreases notably when the pose is derived from SLAM. Notably, our method maintains the highest success rate, surpassing the *NP-SLAM* by 18%, the method with ATE-based reward function by 50%, and the utilisation of raw images by 52%.

Table 6.2: Test success rates in Env 2.

Method	Success Rate
NP-GT	65%
NP-SLAM	42%
Our method	**60**%
P-Image	8%
P-ATE	10%

The experiments in virtual environments reveal the remarkable performance of our proposed method in terms of success rate and generalisation capability.

The proposed approach eliminates the assumption of available localisation and utilises ORB-SLAM to provide robot poses. SLAM performance may degrade in environments with sparse or poorly observed features. This method integrates the perception module for localisation with decision-making in path planning. The trained policy directs the robot toward the target while simultaneously improving SLAM localisation accuracy. To test this, two environments with few features, which are challenging for SLAM, were selected. The results show that the method performs effectively, with the policy guiding the robot toward feature-rich areas, enhancing localisation as the robot observes more environmental details. ORB-SLAM is a widely used visual SLAM method, and many works have proved that ORB-SLAM performs excellently in feature-dense environments (Mur-Artal and Tardós, 2017). We believe the proposed method will perform even better in more feature-rich environments.

6.3.3 Study on RPE Dynamic Threshold

In this section, we study the effect of our proposed dynamic threshold δ_p. Specifically, we train another model with a pre-set threshold. When the RPE exceeds the threshold, r_{pose} in Eq. 6.5, is used. The rest of the reward function remains the same. To set a suitable threshold, we run the robot for $10,000$ timesteps randomly and record the RPE generated at each timestep. Then, we rank them in descending order. Subsequently, we select the smallest RPE value (0.0026) that falls within the top 10% of the ranked list as the threshold. After training, we also conduct tests in Env 1 and Env 2. The success rates are shown in Table 6.3.

Table 6.3: Test success rates in Env 1/ Env 2.

Method	Success Rate (Env 1/ Env 2)
Pre-set threshold	4% / 4%
Our method	**49**% / **60**%

The results show that our method has significantly higher success rates in both environments. It suggests that pre-setting the threshold is not suitable for training an effective policy. In contrast, our proposed dynamic threshold significantly enhances the performance of the model.

6.4 Conclusion

This chapter introduces a novel DRL-based mapless navigation without relying on the assumption of the availability of localisation. This method uses RGB-D-based ORB-SLAM2 for localisation in indoor environments. The

trained policy not only directs the robot towards the target location but also enhances robot pose estimation by considering the quality of the observed features along the selected paths. This chapter proposes a compact state representation based on the spatial distributions of map points, thereby enhancing the robot's awareness of areas with reliable map points. Additionally, RPE is incorporated into the reward function instead of ATE. This allows the policy's responsiveness to each single action. Furthermore, a dynamic threshold is introduced to enhance the policy's adaptability to variations in SLAM performance across different environments. The experiments in localisation-challenging environments demonstrate that the proposed method outperforms existing methods in terms of success rate.

Chapter 7

Conclusion and future work

This chapter concludes the work presented in this thesis. The main conclusions are summarised in Section 7.1, while future research directions are discussed in Section 7.2.

7.1 Conclusion

In the emerging era of artificial intelligence, the autonomous operation of mobile robots, such as UAVs and UGVs, fundamentally relies on a reliable and intelligent navigation system. As recognised by many researchers, conventional robot navigation methods have significant limitations, primarily due to the reliance on maps. Mapless navigation offers a solution by eliminating the need for a predefined map. In particular, deep reinforcement learning (DRL)-based methods are gaining increasing attention in mapless navigation because they enable direct mapping between sensory inputs and robot actions.

Despite its promise, DRL-based mapless navigation faces many unresolved challenges. This thesis focuses on improving the performance of DRL-based mapless navigation. Based on the proposed research objectives, the following sections summarise each research project. Section 7.2 discusses the limitations of this thesis and outlines important future research directions.

Improved HRL-based Framework for Mapless Navigation

Solving reinforcement learning (RL)-based mapless navigation tasks is challenging due to their sparse reward and long decision horizon nature. HRL has the ability to leverage knowledge at different abstract levels and is thus preferred in complex mapless navigation tasks. However, the high-level (HL) policies of most methods lack the ability to distinguish the worthiness of each

subgoal. It is computationally expensive and inefficient to learn navigation policy end-to-end from raw high-dimensional sensor data, such as Lidar or RGB cameras.

In conclusion, Chapter 4 presents a novel HRL-based mapless navigation framework. Specifically, it introduces two different subgoal scoring methods: Predictive Neighbouring Space Scoring (PNSS) and Predictive Exploration Worthiness (PEW). The PNSS model estimates the explorable space for each subgoal, while the PEW model predicts the spatial distribution of obstacles, including the area of free space and the arrangement of obstacles around each subgoal. These PNSS or PEW values are then used as part of the HL inputs, providing a compact and informative state representation for subgoal selection. This work also proposes a new subgoal space layout and studies the effects of different subgoal space layouts. Moreover, a penalty term is introduced in the reward function for the HL policy, so that the subgoal selection process takes the performance of the low-level (LL) policy into consideration.

Comprehensive evaluations demonstrate that using the proposed PNSS or PEW module consistently improves performances over the use of Lidar only or Lidar and encoded RGB features. The proposed method shows significant improvements in success rates when tested in unseen environments. Additionally, the results demonstrate that the proposed subgoal space layout performs better in long-range tasks.

Resolving Local Minima in HRL-based Methods

Hierarchical Reinforcement Learning (HRL) has shown superior performance for mapless navigation tasks. However, it remains limited in unstructured environments that might contain terrains like long corridors and dead corners,

which can lead to local minima. This is because most HRL-based mapless navigation methods employ a simplified reward setting and exploration strategy.

In conclusion, Chapter 5 proposes a novel reward function for training the high-level (HL) policy, which contains two components: extrinsic reward and intrinsic reward. The extrinsic reward encourages the robot to move towards the target location, while the intrinsic reward is computed based on novelty, episode memory and memory decaying, making the agent capable of accomplishing spontaneous exploration. This work also designs a novel neural network structure that incorporates an LSTM network to augment the agent with memory and reasoning capabilities.

The proposed method is tested in unknown environments and scenarios prone to the local minimum problem to evaluate its navigation performance and ability to resolve local minima. The results show that the proposed method significantly increases the success rate when compared to advanced RL-based methods, achieving a maximum improvement of nearly 28%. The proposed method demonstrates effective improvement in addressing the local minimum issue, especially in cases where the baselines fail completely. Additionally, numerous ablation studies consistently confirm the effectiveness of the proposed reward function and neural network structure.

Development of a DRL-based Method for Localisability-Aware Mapless Navigation

Most current works of mapless navigation assume accurate ground-truth robot pose is available. However, this is not realistic, especially for indoor environments, where SLAM is needed for pose estimation, which highly relies on the richness of environment features.

In conclusion, Chapter 6 proposes a novel DRL-based mapless navigation method without relying on the assumption of the availability of robot pose. The proposed method utilises RGB-D-based ORB-SLAM2 for robot localisation. The trained policy effectively guides the robot's movement toward the target while enhancing robot pose estimation by considering the quality of the observed features along the selected paths. To facilitate policy training, a compact state representation based on the spatial distributions of map points is proposed, enhancing the robot's awareness of areas with reliable map points. Furthermore, this work suggests incorporating the relative pose error into the reward function. In this way, the policy will be more responsive to each single action. In addition, rather than utilising a pre-set threshold to assess the localisation performance, a dynamic threshold is adopted to improve the policy's adaptability to variations in SLAM performance across different environments.

The experiments in localisation-challenging environments have demonstrated the remarkable performance of the proposed method. It outperforms the related RL-based methods in terms of success rate.

7.2 Future Work

Deploy The Works on Real Robots

The works presented in Chapters 4 and 6 were tested in virtual environments. Future work will focus on applying models trained in these virtual environments to real robots. Real-world environments pose challenges that are difficult to replicate in virtual settings, such as the impact of lighting on the camera, the friction between the robot tyres and the ground and so forth. It requires the development of techniques to address the sim-to-real transfer

problem.

Avoid Dynamic Obstacles

In this thesis, the focus is navigating in static environments. Dynamic objects significantly complicate obstacle avoidance, as model-free approaches struggle to predict their future positions. However, the proposed networks in this thesis could be integrated with an object tracking module to model the relative velocities between the robot and observable moving objects. Combined with an attention mechanism, this information could help the DRL agent learn a robust obstacle avoidance policy in dynamic environments.

Apply in Complex Outdoor Environments

All scenarios addressed in this thesis involve indoor environments. However, many applications of mapless navigation occur in settings where map construction is challenging, such as post-disaster relief situations. Future work aims to train and apply the models in complex outdoor environments.

Multi-robot Navigation

In this thesis, the proposed mapless navigation systems are many for a single mobile robot. However, multi-robot cooperation is essential in scenarios such as search and rescue, where systems can collaboratively search for survivors and deliver rescue supplies in disaster areas. Therefore, future work aims to enhance the proposed system for multi-robot applications. Several key questions emerge in this context: 1) How can each robot retrieve experiences? 2) How can the limited knowledge of each agent be effectively propagated? 3) How can information from various agents be integrated to plan the optimal path? These questions offer promising avenues for further research.

Apply in Extremely Large Environments

The experimental environments in this thesis are all relatively small. Specifically, the work presented in Chapter 5 relies on episode memory, which may encounter limitations when applied to very large environments, as it is impractical to store all visited locations. Therefore, for navigation in large environments, it is essential to store a series of typical positions. Future work will focus on developing a model to predict whether a position is typical, allowing the robot to store and use these positions to maintain a general environmental awareness.

Bibliography

Abhishek Kadian*, Joanne Truong*, Gokaslan, A., Clegg, A., Wijmans, E., Lee, S., Savva, M., Chernova, S. and Batra, D. 2020. Sim2Real Predictivity: Does Evaluation in Simulation Predict Real-World Performance? vol. 5, pp. 6670–6677.

Bacon, P.-L., Harb, J. and Precup, D. 2017. The option-critic architecture. In: *Proceedings of the AAAI conference on artificial intelligence*. vol. 31.

Bailey, T., Nieto, J., Guivant, J., Stevens, M. and Nebot, E. 2006. Consistency of the ekf-slam algorithm. In: *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, pp. 3562–3568.

Beeson, P., Modayil, J. and Kuipers, B. 2010. Factoring the mapping problem: Mobile robot map-building in the hybrid spatial semantic hierarchy. *The International Journal of Robotics Research* 29(4), pp. 428–459.

Bellemare, M., Srinivasan, S., Ostrovski, G., Schaul, T., Saxton, D. and Munos, R. 2016. Unifying count-based exploration and intrinsic motivation. *Advances in neural information processing systems* 29.

Bischoff, B., Nguyen-Tuong, D., Lee, I., Streichert, F., Knoll, A. *et al.* 2013. Hierarchical reinforcement learning for robot navigation. In: *Proceedings*

of The European Symposium on Artificial Neural Networks, Computational Intelligence And Machine Learning (ESANN 2013).

Bodnar, C., Day, B. and Lió, P. 2020. Proximal distilled evolutionary reinforcement learning. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. vol. 34, pp. 3283–3290.

Chakravarthy, A. and Ghose, D. 1998. Obstacle avoidance in a dynamic environment: A collision cone approach. *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans* 28(5), pp. 562–574.

Chen, W., Li, W., Yang, A. and Hu, Y. 2023a. Active visual slam based on hierarchical reinforcement learning. In: *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, pp. 7155–7162.

Chen, Y., Qiu, Q., Liu, X., Chen, G., Yao, S., Peng, J., Ji, J. and Zhang, Y. 2023b. Deep reinforcement learning for localizability-enhanced navigation in dynamic human environments. *arXiv preprint arXiv:2303.12354* .

Cunha, J., Pedrosa, E., Cruz, C., Neves, A. J. and Lau, N. 2011. Using a depth camera for indoor robot localization and navigation. *DETI/IEETA-University of Aveiro, Portugal* 27(Jun), p. 6.

Deshpande, M., Kim, R., Kumar, D., Park, J. J. and Zamiska, J. 2023. Lighthouses and global graph stabilization: Active slam for low-compute, narrow-fov robots. In: *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 4070–4076.

DeSouza, G. N. and Kak, A. C. 2002. Vision for mobile robot navigation: A survey. *IEEE transactions on pattern analysis and machine intelligence* 24(2), pp. 237–267.

Dey, S., Sadek, A., Monaci, G., Chidlovskii, B. and Wolf, C. 2023. Learning whom to trust in navigation: dynamically switching between classical and neural planning. In: *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, pp. 5235–5242.

Dilokthanakul, N., Kaplanis, C., Pawlowski, N. and Shanahan, M. 2019. Feature control as intrinsic motivation for hierarchical reinforcement learning. *IEEE transactions on neural networks and learning systems* 30(11), pp. 3409–3418.

Ding, W., Li, S., Qian, H. and Chen, Y. 2018. Hierarchical reinforcement learning framework towards multi-agent navigation. In: *2018 IEEE international conference on robotics and biomimetics (ROBIO)*. IEEE, pp. 237–242.

Dobrevski, M. and Skočaj, D. 2021. Deep reinforcement learning for map-less goal-driven robot navigation. *International Journal of Advanced Robotic Systems* 18(1), p. 1729881421992621.

Dodd, W. and Gutierrez, R. 2005. The role of episodic memory and emotion in a cognitive robot. In: *ROMAN 2005. IEEE International Workshop on Robot and Human Interactive Communication, 2005.* IEEE, pp. 692–697.

Duan, Y., Andrychowicz, M., Stadie, B., Jonathan Ho, O., Schneider, J., Sutskever, I., Abbeel, P. and Zaremba, W. 2017. One-shot imitation learning. *Advances in neural information processing systems* 30.

Durrant-Whyte, H. and Bailey, T. 2006. Simultaneous localization and mapping: part i. *IEEE robotics & automation magazine* 13(2), pp. 99–110.

Ekstrom, A. D., Arnold, A. E. and Iaria, G. 2014. A critical review of the

allocentric spatial representation and its neural underpinnings: toward a network-based perspective. *Frontiers in human neuroscience* 8, p. 803.

Elfes, A. 1987. Sonar-based real-world mapping and navigation. *IEEE Journal on Robotics and Automation* 3(3), pp. 249–265.

Engel, Y., Mannor, S. and Meir, R. 2005. Reinforcement learning with gaussian processes. In: *Proceedings of the 22nd international conference on Machine learning*. pp. 201–208.

Eppe, M., Nguyen, P. D. and Wermter, S. 2019. From semantics to execution: Integrating action planning with reinforcement learning for robotic causal problem-solving. *Frontiers in Robotics and AI* 6, p. 123.

Epstein, R. A., Patai, E. Z., Julian, J. B. and Spiers, H. J. 2017. The cognitive map in humans: spatial navigation and beyond. *Nature neuroscience* 20(11), pp. 1504–1513.

Eysenbach, B., Salakhutdinov, R. R. and Levine, S. 2019. Search on the replay buffer: Bridging planning and reinforcement learning. *Advances in neural information processing systems* 32.

Fairfield, N., Wettergreen, D. and Kantor, G. 2010. Segmented slam in three-dimensional environments. *Journal of Field Robotics* 27(1), pp. 85–103.

Fan, T., Cheng, X., Pan, J., Manocha, D. and Yang, R. 2018. Crowdmove: Autonomous mapless navigation in crowded scenarios. *arXiv preprint arXiv:1807.07870* .

Filliat, D. and Meyer, J.-A. 2003. Map-based navigation in mobile robots:: I. a review of localization strategies. *Cognitive systems research* 4(4), pp. 243–282.

Florensa, C., Held, D., Wulfmeier, M., Zhang, M. and Abbeel, P. 2017. Reverse curriculum generation for reinforcement learning. In: *Conference on robot learning*. PMLR, pp. 482–495.

Fu, J., Co-Reyes, J. and Levine, S. 2017. Ex2: Exploration with exemplar models for deep reinforcement learning. *Advances in neural information processing systems* 30.

Fujimoto, S., Hoof, H. and Meger, D. 2018. Addressing function approximation error in actor-critic methods. In: *International conference on machine learning*. PMLR, pp. 1587–1596.

Gasparetto, A., Boscariol, P., Lanzutti, A. and Vidoni, R. 2015. Path planning and trajectory planning algorithms: A general overview. *Motion and operation planning of robotic systems: Background and practical approaches* pp. 3–27.

Gers, F. A., Schmidhuber, J. and Cummins, F. 2000. Learning to forget: Continual prediction with lstm. *Neural computation* 12(10), pp. 2451–2471.

Grando, R. B., de Jesus, J. C. and Drews-Jr, P. L. 2020. Deep reinforcement learning for mapless navigation of unmanned aerial vehicles. In: *2020 Latin American Robotics Symposium (LARS), 2020 Brazilian Symposium on Robotics (SBR) and 2020 Workshop on Robotics in Education (WRE)*. IEEE, pp. 1–6.

Grisetti, G., Kümmerle, R., Stachniss, C. and Burgard, W. 2010. A tutorial on graph-based slam. *IEEE Intelligent Transportation Systems Magazine* 2(4), pp. 31–43.

Gupta, A., Kumar, V., Lynch, C., Levine, S. and Hausman, K. 2020. Relay policy learning: Solving long-horizon tasks via imitation and reinforcement learning. In: *Conference on Robot Learning*. PMLR, pp. 1025–1037.

Gupta, S., Davidson, J., Levine, S., Sukthankar, R. and Malik, J. 2017. Cognitive mapping and planning for visual navigation. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. pp. 2616–2625.

Haarnoja, T., Zhou, A., Abbeel, P. and Levine, S. 2018. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In: *International conference on machine learning*. PMLR, pp. 1861–1870.

Hausknecht, M. and Stone, P. 2015. Deep recurrent q-learning for partially observable mdps. In: *2015 aaai fall symposium series*.

Hu, H., Zhang, K., Tan, A. H., Ruan, M., Agia, C. and Nejat, G. 2021. A sim-to-real pipeline for deep reinforcement learning for autonomous robot navigation in cluttered rough terrain. *IEEE Robotics and Automation Letters* 6(4), pp. 6569–6576.

Jang, Y., Baek, J. and Han, S. 2021. Hindsight intermediate targets for mapless navigation with deep reinforcement learning. *IEEE Transactions on Industrial Electronics* 69(11), pp. 11816–11825.

Jin, J., Nguyen, N. M., Sakib, N., Graves, D., Yao, H. and Jagersand, M. 2020. 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)*. IEEE, pp. 6979–6985.

Jurgenson, T., Avner, O., Groshev, E. and Tamar, A. 2020. Sub-goal trees a framework for goal-based reinforcement learning. In: *International conference on machine learning*. PMLR, pp. 5020–5030.

Karamizadeh, S., Abdullah, S. M., Manaf, A. A., Zamani, M. and Hooman, A. 2020. An overview of principal component analysis. *Journal of signal and information processing* 4.

Kemp, A. and Manahan-Vaughan, D. 2004. Hippocampal long-term depression and long-term potentiation encode different aspects of novelty acquisition. *Proceedings of the National Academy of Sciences* 101(21), pp. 8192–8197.

Khadka, S. and Tumer, K. 2018. Evolution-guided policy gradient in reinforcement learning. *Advances in Neural Information Processing Systems* 31.

Khan, A., Kumar, V. and Ribeiro, A. 2018. Learning sample-efficient target reaching for mobile robots. In: *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, pp. 3080–3087.

Khatib, O. 1986. Real-time obstacle avoidance for manipulators and mobile robots. *The international journal of robotics research* 5(1), pp. 90–98.

Klink, P., Abdulsamad, H., Belousov, B. and Peters, J. 2020. Self-paced contextual reinforcement learning. In: *Conference on Robot Learning*. PMLR, pp. 513–529.

Kober, J., Bagnell, J. A. and Peters, J. 2013. Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research* 32(11), pp. 1238–1274.

Kohlbrecher, S., Von Stryk, O., Meyer, J. and Klingauf, U. 2011. A flexible and scalable slam system with full 3d motion estimation. In: *2011 IEEE international symposium on safety, security, and rescue robotics*. IEEE, pp. 155–160.

Kostavelis, I. and Gasteratos, A. 2015. Semantic mapping for mobile robotics tasks: A survey. *Robotics and Autonomous Systems* 66, pp. 86–103.

Kümmerle, R., Grisetti, G., Strasdat, H., Konolige, K. and Burgard, W. 2011. g 2 o: A general framework for graph optimization. In: *2011 IEEE international conference on robotics and automation*. IEEE, pp. 3607–3613.

Kurniawati, H. 2022. Partially observable markov decision processes and robotics. *Annual Review of Control, Robotics, and Autonomous Systems* 5(1), pp. 253–277.

Kwon, Y. D. and Lee, J. S. 1999. A stochastic map building method for mobile robot using 2-d laser range finder. *Autonomous Robots* 7, pp. 187–200.

Laber, J., Thamma, R. and Kirby, E. D. 2020. The impact of warehouse automation in amazon's success. *Int. J. Innov. Sci. Eng. Technol* 7, pp. 63–70.

Lehner, H., Schuster, M. J., Bodenmüller, T. and Kriegel, S. 2017. Exploration with active loop closing: A trade-off between exploration efficiency and map quality. In: *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, pp. 6191–6198.

Levy, A., Konidaris, G., Platt, R. and Saenko, K. 2017. Learning multi-level hierarchies with hindsight. *arXiv preprint arXiv:1712.00948* .

Li, Y. 2017. Deep reinforcement learning: An overview. *arXiv preprint arXiv:1701.07274* .

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D. and Wierstra, D. 2015. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971* .

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D. and Wierstra, D. 2016. Continuous control with deep reinforcement learning. In: Bengio, Y. and LeCun, Y., eds., *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*. Available at: http://arxiv.org/abs/1509.02971.

Lin, F., Ji, Z., Wei, C. and Grech, R. 2022. Localisation-safe reinforcement learning for mapless navigation. In: *2022 IEEE International Conference on Robotics and Biomimetics (ROBIO)*. IEEE, pp. 1327–1334.

Lin, F., Ji, Z., Wei, C. and Niu, H. 2021. Reinforcement learning-based mapless navigation with fail-safe localisation. In: *Annual Conference Towards Autonomous Robotic Systems*. Springer, pp. 100–111.

Lingemann, K., Nüchter, A., Hertzberg, J. and Surmann, H. 2005. High-speed laser localization for mobile robots. *Robotics and autonomous systems* 51(4), pp. 275–296.

Lozano-Perez, T. 1990. *Spatial planning: A configuration space approach*. Springer.

Lozano-Pérez, T. and Wesley, M. A. 1979. An algorithm for planning collision-free paths among polyhedral obstacles. *Communications of the ACM* 22(10), pp. 560–570.

Luong, M. and Pham, C. 2021. Incremental learning for autonomous navigation of mobile robots based on deep reinforcement learning. *Journal of Intelligent & Robotic Systems* 101(1), p. 1.

Mac, T. T., Copot, C., Tran, D. T. and De Keyser, R. 2016. Heuristic approaches in robot path planning: A survey. *Robotics and Autonomous Systems* 86, pp. 13–28.

Marchesini, E. and Farinelli, A. 2020a. Discrete deep reinforcement learning for mapless navigation. In: *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 10688–10694.

Marchesini, E. and Farinelli, A. 2020b. Genetic deep reinforcement learning for mapless navigation. In: *Proceedings of the 19th International Conference on Autonomous Agents and MultiAgent Systems*. pp. 1919–1921.

Matthies, L., Xiong, Y., Hogg, R., Zhu, D., Rankin, A., Kennedy, B., Hebert, M., Maclachlan, R., Won, C., Frost, T. *et al.* 2002. A portable, autonomous, urban reconnaissance robot. *Robotics and Autonomous Systems* 40(2-3), pp. 163–172.

Mei, H., Tian, Y. and Zu, L. 2006. A hybrid ant colony optimization algorithm for path planning of robot in dynamic environment. *International Journal of Information Technology* 12(3), pp. 78–88.

Meyer, J.-A. and Filliat, D. 2003. Map-based navigation in mobile robots:: Ii. a review of map-learning and path-planning strategies. *Cognitive Systems Research* 4(4), pp. 283–317.

Millonig, A. and Schechtner, K. 2007. Developing landmark-based pedestrian-navigation systems. *IEEE Transactions on intelligent transportation systems* 8(1), pp. 43–49.

Mirowski, P., Grimes, M., Malinowski, M., Hermann, K. M., Anderson, K., Teplyashin, D., Simonyan, K., Zisserman, A., Hadsell, R. *et al.* 2018. Learning to navigate in cities without a map. *Advances in neural information processing systems* 31.

Mirowski, P., Pascanu, R., Viola, F., Soyer, H., Ballard, A. J., Banino, A., Denil, M., Goroshin, R., Sifre, L., Kavukcuoglu, K. *et al.* 2016. Learning to navigate in complex environments. *arXiv preprint arXiv:1611.03673* .

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D. and Kavukcuoglu, K. 2016. Asynchronous methods for deep reinforcement learning. In: *International conference on machine learning*. PMLR, pp. 1928–1937.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G. *et al.* 2015. Human-level control through deep reinforcement learning. *nature* 518(7540), pp. 529–533.

Montemerlo, M., Thrun, S., Koller, D., Wegbreit, B. *et al.* 2002. Fastslam: A factored solution to the simultaneous localization and mapping problem. *Aaai/iaai* 593598, pp. 593–598.

Moradi, H., Choi, J., Kim, E. and Lee, S. 2006. A real-time wall detection method for indoor environments. In: *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, pp. 4551–4557.

Moravec, H. and Elfes, A. 1985. High resolution maps from wide angle sonar. In: *Proceedings. 1985 IEEE international conference on robotics and automation*. IEEE, vol. 2, pp. 116–121.

Mur-Artal, R., Montiel, J. M. M. and Tardos, J. D. 2015. Orb-slam: a versatile and accurate monocular slam system. *IEEE transactions on robotics* 31(5), pp. 1147–1163.

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

Nachum, O., Gu, S. S., Lee, H. and Levine, S. 2018. Data-efficient hierarchical reinforcement learning. *Advances in neural information processing systems* 31.

Nair, S. and Finn, C. 2019. Hierarchical foresight: Self-supervised learning of long-horizon tasks via visual subgoal generation. *arXiv preprint arXiv:1909.05829* .

Nasteski, V. 2017. An overview of the supervised machine learning methods. *Horizons. b* 4(51-62), p. 56.

Naveed, K., Anjum, M. L., Hussain, W. and Lee, D. 2022. Deep introspective slam: Deep reinforcement learning based approach to avoid tracking failure in visual slam. *Autonomous Robots* 46(6), pp. 705–724.

Nistér, D. and Stewénius, H. 2007. A minimal solution to the generalised 3-point pose problem. *Journal of Mathematical Imaging and Vision* 27(1), pp. 67–79.

Ostrovski, G., Bellemare, M. G., Oord, A. and Munos, R. 2017. Count-based exploration with neural density models. In: *International conference on machine learning*. PMLR, pp. 2721–2730.

Parascandolo, G., Buesing, L., Merel, J., Hasenclever, L., Aslanides, J., Hamrick, J. B., Heess, N., Neitz, A. and Weber, T. 2020. Divide-and-conquer monte carlo tree search for goal-directed planning. *arXiv preprint arXiv:2004.11410* .

Pathak, D., Agrawal, P., Efros, A. A. and Darrell, T. 2017. Curiosity-driven exploration by self-supervised prediction. In: *International conference on machine learning*. PMLR, pp. 2778–2787.

Patle, B., Pandey, A., Parhi, D., Jagadeesh, A. *et al.* 2019. A review: On path planning strategies for navigation of mobile robot. *Defence Technology* 15(4), pp. 582–606.

Piot, B., Geist, M. and Pietquin, O. 2016. Bridging the gap between imitation learning and inverse reinforcement learning. *IEEE transactions on neural networks and learning systems* 28(8), pp. 1814–1826.

Prasad, V., Yadav, K., Saurabh, R. S., Daga, S., Pareekutty, N., Krishna, K. M., Ravindran, B. and Bhowmick, B. 2018. Learning to prevent monocular slam failure using reinforcement learning. In: *Proceedings of the 11th Indian conference on computer vision, graphics and image processing*. pp. 1–9.

Pritzel, A., Uria, B., Srinivasan, S., Badia, A. P., Vinyals, O., Hassabis, D., Wierstra, D. and Blundell, C. 2017. Neural episodic control. In: *International conference on machine learning*. PMLR, pp. 2827–2836.

Qin, Y.-Q., Sun, D.-B., Li, N. and Cen, Y.-G. 2004. Path planning for mobile robot using the particle swarm optimization with mutation operator. In: *Proceedings of 2004 international conference on machine learning and cybernetics (IEEE Cat. No. 04EX826)*. IEEE, vol. 4, pp. 2473–2478.

Qureshi, A. H. and Yip, M. C. 2018. Deeply informed neural sampling for robot motion planning. In: *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, pp. 6582–6588.

Rachkov, M. Y., Marques, L. and de Almeida, A. T. 2005. Multisensor demining robot. *Autonomous robots* 18, pp. 275–291.

Rai, A., Chintalapudi, K. K., Padmanabhan, V. N. and Sen, R. 2012. Zee: Zero-effort crowdsourcing for indoor localization. In: *Proceedings of the 18th annual international conference on Mobile computing and networking*. pp. 293–304.

Ramakrishnan, S. K., Al-Halah, Z. and Grauman, K. 2020. 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*. Springer, pp. 400–418.

Ridnik, T., Ben-Baruch, E., Noy, A. and Zelnik-Manor, L. 2021. Imagenet-21k pretraining for the masses. *arXiv preprint arXiv:2104.10972* .

Ronneberger, O., Fischer, P. and Brox, T. 2015. U-net: Convolutional networks for biomedical image segmentation. In: *Medical image computing and computer-assisted intervention–MICCAI 2015: 18th international conference, Munich, Germany, October 5-9, 2015, proceedings, part III 18*. Springer, pp. 234–241.

Ruan, X., Li, P., Zhu, X. and Liu, P. 2022. A target-driven visual navigation method based on intrinsic motivation exploration and space topological cognition. *Scientific Reports* 12(1), p. 3462.

Rublee, E., Rabaud, V., Konolige, K. and Bradski, G. 2011. Orb: An efficient

alternative to sift or surf. In: *2011 International conference on computer vision*. Ieee, pp. 2564–2571.

Sarwinda, D., Paradisa, R. H., Bustamam, A. and Anggia, P. 2021. Deep learning in image classification using residual network (resnet) variants for detection of colorectal cancer. *Procedia Computer Science* 179, pp. 423–431.

Schaul, T., Horgan, D., Gregor, K. and Silver, D. 2015a. Universal value function approximators. In: *International conference on machine learning*. PMLR, pp. 1312–1320.

Schaul, T., Quan, J., Antonoglou, I. and Silver, D. 2015b. Prioritized experience replay. *arXiv preprint arXiv:1511.05952* .

Scott, A., Parker, L. E. and Touzet, C. 2000. Quantitative and qualitative comparison of three laser-range mapping algorithms using two types of laser scanner data. In: *Smc 2000 conference proceedings. 2000 ieee international conference on systems, man and cybernetics.'cybernetics evolving to systems, humans, organizations, and their complex interactions'(cat. no. 0*. IEEE, vol. 2, pp. 1422–1427.

Shao, K., Zhao, D., Zhu, Y. and Zhang, Q. 2018. Visual navigation with actor-critic deep reinforcement learning. In: *2018 International Joint Conference on Neural Networks (IJCNN)*. IEEE, pp. 1–6.

Sharma, A., Gu, S., Levine, S., Kumar, V. and Hausman, K. 2019. Dynamics-aware unsupervised discovery of skills. *arXiv preprint arXiv:1907.01657* .

Shen, B., Xia, F., Li, C., Martín-Martín, R., Fan, L., Wang, G., Pérez-D'Arpino, C., Buch, S., Srivastava, S., Tchapmi, L. *et al.* 2021. igibson

1.0: A simulation environment for interactive tasks in large realistic scenes. In: *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, pp. 7520–7527.

Silver, D., Lever, G., Heess, N., Degris, T., Wierstra, D. and Riedmiller, M. 2014. Deterministic policy gradient algorithms. In: *International conference on machine learning*. PMLR, pp. 387–395.

Singla, A., Padakandla, S. and Bhatnagar, S. 2019. Memory-based deep reinforcement learning for obstacle avoidance in uav with limited environment knowledge. *IEEE transactions on intelligent transportation systems* 22(1), pp. 107–118.

Stachniss, C., Hahnel, D. and Burgard, W. 2004. Exploration with active loop-closing for fastslam. In: *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)(IEEE Cat. No. 04CH37566)*. IEEE, vol. 2, pp. 1505–1510.

Staroverov, A., Yudin, D. A., Belkin, I., Adeshkin, V., Solomentsev, Y. K. and Panov, A. I. 2020. Real-time object navigation with deep neural networks and hierarchical reinforcement learning. *IEEE Access* 8, pp. 195608–195621.

Such, F. P., Madhavan, V., Conti, E., Lehman, J., Stanley, K. O. and Clune, J. 2017. Deep neuroevolution: Genetic algorithms are a competitive alternative for training deep neural networks for reinforcement learning. *arXiv preprint arXiv:1712.06567* .

Sutton, R. S. 1990. Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In: *Machine learning proceedings 1990*, Elsevier, pp. 216–224.

Sutton, R. S. and Barto, A. G. 2018. *Reinforcement learning: An introduction*. MIT press.

Sutton, R. S., McAllester, D., Singh, S. and Mansour, Y. 1999. Policy gradient methods for reinforcement learning with function approximation. *Advances in neural information processing systems* 12.

Taheri, H. and Xia, Z. C. 2021. Slam; definition and evolution. *Engineering Applications of Artificial Intelligence* 97, p. 104032.

Tai, L., Paolo, G. and Liu, M. 2017. 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)*. IEEE, pp. 31–36.

Taketomi, T., Uchiyama, H. and Ikeda, S. 2017. Visual slam algorithms: A survey from 2010 to 2016. *IPSJ transactions on computer vision and applications* 9(1), p. 16.

Tang, H., Houthooft, R., Foote, D., Stooke, A., Xi Chen, O., Duan, Y., Schulman, J., DeTurck, F. and Abbeel, P. 2017. # exploration: A study of count-based exploration for deep reinforcement learning. *Advances in neural information processing systems* 30.

Thrun, S., Burgard, W. and Fox, D. 1998. A probabilistic approach to concurrent mapping and localization for mobile robots. *Autonomous Robots* 5, pp. 253–271.

Thrun, S. and Montemerlo, M. 2006. The graph slam algorithm with applications to large-scale mapping of urban structures. *The International Journal of Robotics Research* 25(5-6), pp. 403–429.

Tian, X., Zhang, J., Ma, Z., He, Y. and Wei, J. 2017. Frame stacking and retaining for recurrent neural network acoustic model. *arXiv preprint arXiv:1705.05992* .

Vadakkepat, P., Tan, K. C. and Ming-Liang, W. 2000. Evolutionary artificial potential fields and their application in real time robot path planning. In: *Proceedings of the 2000 congress on evolutionary computation. CEC00 (Cat. No. 00TH8512)*. IEEE, vol. 1, pp. 256–263.

Van Hasselt, H., Guez, A. and Silver, D. 2016. Deep reinforcement learning with double q-learning. In: *Proceedings of the AAAI conference on artificial intelligence*. vol. 30.

Vanhoucke, V., Devin, M. and Heigold, G. 2013. Multiframe deep neural networks for acoustic modeling. In: *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*. IEEE, pp. 7582–7585.

Vezhnevets, A. S., Osindero, S., Schaul, T., Heess, N., Jaderberg, M., Silver, D. and Kavukcuoglu, K. 2017. Feudal networks for hierarchical reinforcement learning. In: *International conference on machine learning*. PMLR, pp. 3540–3549.

Wang, S., Wen, H., Clark, R. and Trigoni, N. 2016a. Keyframe based large-scale indoor localisation using geomagnetic field and motion pattern. In: *2016 IEEE/RSJ international conference on intelligent robots and systems (IROS)*. IEEE, pp. 1910–1917.

Wang, Z., Schaul, T., Hessel, M., Hasselt, H., Lanctot, M. and Freitas, N. 2016b. Dueling network architectures for deep reinforcement learning. In: *International conference on machine learning*. PMLR, pp. 1995–2003.

Watkins, C. J. and Dayan, P. 1992. Q-learning. *Machine learning* 8, pp. 279–292.

Wiering, M. A. and Van Otterlo, M. 2012. Reinforcement learning. *Adaptation, learning, and optimization* 12(3), p. 729.

Williams, R. J. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning* 8, pp. 229–256.

Wöhlke, J., Schmitt, F. and van Hoof, H. 2020. A performance-based start state curriculum framework for reinforcement learning. In: *Proceedings of the 19th International Conference on Autonomous Agents and MultiAgent Systems*. pp. 1503–1511.

Wöhlke, J., Schmitt, F. and van Hoof, H. 2021. Hierarchies of planning and reinforcement learning for robot navigation. In: *2021 IEEE international conference on robotics and automation (ICRA)*. IEEE, pp. 10682–10688.

Wolbers, T. and Wiener, J. M. 2014. Challenges for identifying the neural mechanisms that support spatial navigation: the impact of spatial scale. *Frontiers in human neuroscience* 8, p. 571.

Wu, Y., Mansimov, E., Grosse, R. B., Liao, S. and Ba, J. 2017. Scalable trust-region method for deep reinforcement learning using kronecker-factored approximation. *Advances in neural information processing systems* 30.

Xia, F., Zamir, A. R., He, Z., Sax, A., Malik, J. and Savarese, S. 2018. Gibson env: Real-world perception for embodied agents. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. pp. 9068–9079.

Xiao, Z., Wen, H., Markham, A. and Trigoni, N. 2014. Robust pedestrian dead reckoning (r-pdr) for arbitrary mobile device placement. In: *2014 International Conference on Indoor Positioning and Indoor Navigation (IPIN)*. IEEE, pp. 187–196.

Xie, L. 2019. *Reinforcement learning based mapless robot navigation*. Ph.D. thesis, University of Oxford.

Xie, L., Markham, A. and Trigoni, N. 2020. Snapnav: Learning mapless visual navigation with sparse directional guidance and visual reference. In: *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 1682–1688.

Xing, J., Cioffi, G., Hidalgo-Carrió, J. and Scaramuzza, D. 2023. Autonomous power line inspection with drones via perception-aware mpc. In: *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, pp. 1086–1093.

Xue, Y. and Chen, W. 2023. Combining motion planner and deep reinforcement learning for uav navigation in unknown environment. *IEEE Robotics and Automation Letters* 9(1), pp. 635–642.

Yamamoto, K., Onishi, T. and Tsuruoka, Y. 2018. Hierarchical reinforcement learning with abductive planning. *arXiv preprint arXiv:1806.10792* .

Yang, X., Ji, Z., Wu, J., Lai, Y.-K., Wei, C., Liu, G. and Setchi, R. 2021. Hierarchical reinforcement learning with universal policies for multistep robotic manipulation. *IEEE Transactions on Neural Networks and Learning Systems* 33(9), pp. 4727–4741.

Yu, Y., Si, X., Hu, C. and Zhang, J. 2019. A review of recurrent neural

networks: Lstm cells and network architectures. *Neural computation* 31(7), pp. 1235–1270.

Zhang, Z. 2012. Microsoft kinect sensor and its effect. *IEEE multimedia* 19(2), pp. 4–10.

Zhelo, O., Zhang, J., Tai, L., Liu, M. and Burgard, W. 2018. Curiosity-driven exploration for mapless navigation with deep reinforcement learning. *arXiv preprint arXiv:1804.00456* .

Zhou, X., Bai, T., Gao, Y. and Han, Y. 2019. Vision-based robot navigation through combining unsupervised learning and hierarchical reinforcement learning. *Sensors* 19(7), p. 1576.

Zhu, Y., Mottaghi, R., Kolve, E., Lim, J. J., Gupta, A., Fei-Fei, L. and Farhadi, A. 2017. Target-driven visual navigation in indoor scenes using deep reinforcement learning. In: *2017 IEEE international conference on robotics and automation (ICRA)*. IEEE, pp. 3357–3364.

Zhu, Y., Wang, Z., Chen, C. and Dong, D. 2021. Rule-based reinforcement learning for efficient robot navigation with space reduction. *IEEE/ASME Transactions on Mechatronics* 27(2), pp. 846–857.