Reinforcement Learning Based Mapless Robot Navigation

Linhai Xie

Kellogg College

University of Oxford

A thesis submitted for the degree of

Doctor of Philosophy

Trinity 2019

This thesis is dedicated to my parents
for their indispensable, altruistic support

Acknowledgements

In the epilogue of my PhD career, I feel thankful to many people. Those who deserve the most of my gratefulness are my supervisors Niki and Andrew. Their diligent supervision trains me and grows me, strict but full of tenderness. Listening to their "crazy" ideas is one of the biggest pleasure, though they frightened me a lot at the beginning. However, gradually I realised that they taught me through those ideas, as what the bishop has taught the arrested Valjean in *Les Miserables*, to see things in a higher plan. In their dual supervision, Niki watches more on my research direction, discussing all the possibilities with me patiently based on her long-accumulated academic experience. Andrew, on the other hand, can always catch the core concepts behind my poor expressions and help me to polish them on the paper. Thanks to them, otherwise many of the achievements in this thesis would have died before the birth.

Second, I'd like to thank Dr. Sen Wang and Dr. Yishu Miao. Sen guided my first step to robotics, like Venus to sailors in the dark sea, illuminating my way on this unknown territory. Those memorable days when he was sitting behind me in the office were the peak of my progress in robotic research. Yishu, my advisor and collaborator in machine learning, catalysed my study on deep reinforcement learning and bridging it with robotics. Now he becomes my boss for my internship in MO Intelligence which is a wonderful working place for me. Sen and Yishu are also my friends and mentors who I deeply appreciate their help on my life in Oxford.

Next, I want to thank to Xiaoxuan, Zhihua and other colleagues in Cyber-Physical Systems Group. Their enthusiasm to the work and exciting achievements encouraged me to dive deeper into my research. Besides, the endless anecdotes and fun gossips in the office also brought me laugh and happiness. Thanks to each of my friends in Oxford. I miss all the hotpot parties, drinks in the pub and our online reunions in DotA. My gratitude also goes to my girlfriend Yuanyuan for embracing me warmly

during the tough period and my parents for their indispensable and altruistic support.

Abstract

Navigation is the one of the most fundamental capabilities required for mobile robots, allowing them to traverse from a source to a destination. Conventional approaches rely heavily on the existence of a predefined map which is costly both in time and labour to acquire. In addition, maps are only accurate at the time of acquisition and due to environmental changes degrade over time. We argue that this strict requirement of having access to a high-quality map fundamentally limits the realisability of robotic systems in our dynamic world. In this thesis, we investigate how to develop **practical robotic navigation**, motivated by the paradigm of mapless navigation and inspired by recent developments in Deep Reinforcement Learning (DRL).

One of the major issues for DRL is the requirement of a diverse experimental setup with millions of repeated trials. This clearly is not feasible to acquire from a real robot through trial and error, so instead we learn from a simulated environment. This leads to the first fundamental problem which is that of bridging the reality gap from simulated to real environments, tackled in Chapter 3. We focus on the particular challenge of monocular visual obstacle avoidance as a low-level navigation primitive. We develop a DRL approach that is trained within a simulated world yet can generalise well to the real world.

Another issue which limits the adoption of DRL techniques for mobile robotics in the real world is the high variance of the trained policies. This leads to poor convergence and low overall reward, due to the complex and high dimensional search space. In Chapter 4 we leverage simple classical controllers to provide guidance to the task of local navigation with DRL, avoiding purely random initial exploration. We demonstrate that this novel accelerated approach greatly reduces sample variance and significantly increases achievable average reward.

The last challenge we consider is that of sparse visual guidance for mapless navigation. In Chapter 5, we present an innovative approach to navigate based on a few waypoint images, in contrast to traditional video based teach and repeat. We demonstrate that the policy learnt in simulation can be directly transferred to the real world and has ability to generalise well to unseen scenarios with minimal description of the environment.

We develop and test novel approaches towards the key issues of obstacle avoidance, local guidance and global navigation, towards our vision of enabling practical robotic navigation. We show how DRL can be used as a powerful model-free approach to tackle these issues.

Contents

1 Introduction **1**

1.1 Motivation 1

1.2 Research Challenges and Questions 3

1.3 Contribution 6

2 Background **9**

2.1 Reinforcement Learning 9

2.1.1 Markov Decision Process 9

2.1.2 Essential Elements 10

2.1.3 Comparison with Other Learning Paradigms 11

2.1.4 History 12

2.1.5 Methods 13

2.1.6 Deep Reinforcement Learning Algorithms 14

2.1.6.1 Deep Q Network 14

2.1.6.2 REINFORCE Algorithm 16

2.1.6.3 Deep Deterministic Policy Gradients 18

2.2 Map-based Robot Navigation 19

2.2.1 Sensing 20

2.2.2 Localisation and Mapping 21

2.2.2.1 Localisation 21

2.2.2.2 Map Representation 22

2.2.2.3 Simultaneous Localisation and Mapping 22

2.2.3 Path-Planning 23

2.3 Mapless Robot Navigation 24

2.3.1 Ranging Sensor Based Systems 25

2.3.1.1 Edge-Detection Based Methods 25

2.3.1.2 Potential Field Methods 25

2.3.1.3 Dynamic Window Based Methods 26

2.3.2 Vision Based Systems . . . 26
2.3.2.1 Optical Flow Based Approaches . . . 27
2.3.2.2 Appearance Matching-based Approaches . . . 28
2.3.2.3 Reactive Visual Approaches . . . 29
2.3.2.4 Feature Tracking-based Approaches . . . 30
2.4 Discussion . . . 30
2.4.1 From Simulations to the Real World . . . 30
2.4.2 Speeding up Training in Simulations . . . 31
2.4.3 Learning Global Mapless Navigation . . . 32

3 Visual Obstacle Avoidance: From Simulation to Real World 34
3.1 Introduction . . . 34
3.2 Related Work . . . 35
3.3 Problem Formulation . . . 37
3.4 Model and Training Details . . . 37
3.5 Experiments . . . 40
3.5.1 Training Efficiency with Different Models . . . 40
3.5.2 Real World Tests . . . 41
3.5.2.1 Action Prediction from Static Images . . . 41
3.5.2.2 Tests in Three Different Scenarios . . . 43
3.5.2.3 Tests in a Cluttered Environment . . . 44
3.5.2.4 Comparison with the Baseline Approach . . . 45
3.6 Conclusion . . . 46

4 Local Navigation: Speeding up Training with Guidance 47
4.1 Introduction . . . 47
4.2 Related Work . . . 48
4.3 Problem Formulation and Assumptions . . . 50
4.4 AsDDPG: Learning with Deterministic Guidance . . . 50
4.4.1 Heuristic Controller . . . 52
4.4.2 Network Architecture . . . 52
4.4.3 Critic-DQN . . . 53
4.4.4 Training . . . 55
4.5 Experiments for AsDDPG . . . 56
4.5.1 Speeding up Training Procedure with Various Hyper-parameters 57
4.5.2 Impact of Controller Parameters . . . 58
4.5.3 Training with Complex Environment and Sparse Reward . . . 62

4.5.4 Real World Tests . . . 64
4.6 SGuidance: Learning with Stochastic Guidance . . . 65
4.7 Model . . . 66
4.7.1 Heuristic Controllers . . . 67
4.7.2 Stochastic Switch . . . 67
4.7.2.1 Stick-breaking . . . 67
4.7.2.2 REINFORCE Algorithm with Baselines . . . 68
4.7.3 Algorithm . . . 69
4.8 Experiments for SGuidance . . . 69
4.8.1 Training Environments and Settings . . . 71
4.8.2 Navigation in Simulated Environments . . . 72
4.8.2.1 Reinforcement Learning with Stochastic Guidance . . . 72
4.8.2.2 Using Different Independent Controllers . . . 74
4.8.2.3 Using Different Switching Mechanism . . . 74
4.8.2.4 Further Comparison with Thompson Sampling . . . 75
4.8.2.5 Construction of Stochastic Switch Function . . . 78
4.8.2.6 Complex Environment and Simple Reward . . . 79
4.8.2.7 Inference without Guidance . . . 80
4.8.2.8 Learning with Different γ . . . 80
4.8.3 Navigation in a Real World Environment . . . 81
4.9 Conclusion . . . 83

5 Global Navigation: Mapless Navigation with Sparse Directional Guidance and Visual Reference 84
5.1 Introduction . . . 84
5.2 Related Work . . . 86
5.3 Task Description . . . 87
5.3.1 Task Decomposition . . . 87
5.3.2 Command Sub-Task . . . 87
5.3.3 Control Sub-Task . . . 88
5.4 Network Architecture . . . 88
5.4.1 Attention-Based Commander . . . 88
5.4.2 Controller . . . 90
5.5 Training . . . 90
5.5.1 Self-Supervised Commander Training Labels . . . 90
5.5.2 Training the Commander . . . 91

5.5.3 Reinforcement Learning for Control Policy 93
5.6 Experiments 94
5.6.1 Model Ablation Study in the Virtual World 94
5.6.2 Real-World Tests 97
5.7 Conclusion 99

6 Conclusion and Future Work **100**

Bibliography **104**

List of Figures

1.1 A comic (based on the illustration by YimeIsGreat) that images the near future where robots and human will have close cooperation. It also demonstrates some core concepts mentioned in this thesis: 1) Transferring the learnt policy from simulation to the real world. 2) Accelerating the learning process with extra guidance. 3) Learning to navigate with sparse directional instructions and visual references. 4

2.1 In an RL problem, the learning agent observes the state of the environment and carries out an action to affect the environment at each time step. According to different actions, the environment returns a numerical reward as the feedback to the agent, assessing whether it is the desired behaviour. 11

2.2 Three categories of model-free RL methods, i.e. policy-based , value-based and actor-critic approaches. 13

2.3 The composition of map-based navigation and mapless navigation systems. The local goals are the sequential waypoints along the planned path provided by the global planner. 19

2.4 A simple instance for GraphSLAM. The blue triangles represent poses of the robot at different time steps and the red circle denotes the different landmarks which construct the map of the environment. Furthermore the back lines are the trajectory of the robot while the dashed lines indicate the observations to landmarks by the robot 23

2.5 This figure illustrates the dynamic window approach. The white circle and rectangles are obstacles while the blue rectangle is the robot and the dashed lines are the sampled trajectories under different control actions. Based on the prediction, the robot will choose to go straightforward to avoid the obstacles on both sides. 26

2.6 Output of optical flow. The green arrows display the direction of moving pixels while the dots represent the static ones. 27

2.7 The ground segmentation in [154]. Images in the left column are the inputs where the area within black polygon is the proposed reference area for learning ground texture. The right column illustrates the segmentation results. 29

3.1 Network architecture of monocular image based obstacle avoidance through deep reinforcement learning. A fully convolutional neural network is firstly constructed to predict depth from a raw RGB image. It is then followed by a deep Q network which consists of a convolutional network and a dueling network to predict the Q-value of angular actions and linear actions in parallel. 37
3.2 Given a batch of training data, including current state x_t, action a, reward r, and resulting state x_{t+1}, the training procedure of D3QN is shown in the figure. $\oplus$, $\ominus$ and $\otimes$ are element-wise operations for addition, subtraction and multiplication. 38
3.3 Two simulation worlds in Gazebo used for training. 40
3.4 Smoothed learning curves of the three models with average rewards acquired by robot. 41
3.5 Experiments in different indoor environments, e.g. library, museum, attic and bedroom (from top to bottom). The underlying bars demonstrate the Q value for each linear and angular action predicted by the network, where the red ones indicate the actions greedily selected by the network. Notice that the first two are for linear speed actions whilst the rest are for steering actions. 42
3.6 Real world tests in three different scenarios. The curve below the image streams shows the steering actions selected by robots at each step. . . 43
3.7 Real world tests in a room with different number and placement of obstacles. Rectangles show boxes whilst stars and circles are chairs and trash cans respectively. 44
3.8 Real world test in dynamic environments. The obstacles in grey are fixed whilst coloured ones indicate the motion of movable obstacles at each time. The number on the obstacles indicates the changing sequence. 45
3.9 The left image is the feature detection result in the cluttered environment and the remaining two are in the corridor environment. 46

4.1 A deep neural network is trained with an actor-critic Reinforcement Learning approach to learn local planning for robot navigation. The critic-DQN network assesses both the performance of the external controller, e.g. a PID controller, and the policy network, selecting actions from a better one according to the situation. All the resulting learning samples are stored in the replay buffer. Therefore, the policy network can improve itself either by imitating the external controller or by examining its own policy. 51
4.2 Network architecture. Network layers are demonstrated by the rectangles. Orange arrows indicate the connectivity between network layers and some other components, e.g. input state and the output of the simple controller. The final action is selected based on the Q value predicted by the critic-DQN. 52
4.3 The three Stage simulation worlds used for training. The gray rectangles are obstacles while the blue one is the robot. The target position is randomly generated for each episode. 56
4.4 Smoothed learning curves with different network hyper-parameters. The figure illustrates the average performance of the network at each training step. Note that MoveBase is selected as the baseline approach whose average performance reaches a score at about 0.5. 57
4.5 The usage of the policy network within an episode through the entire training procedure in two different simulation worlds. 58
4.6 The policy learned by the robot with various proportional controllers at different stages of the training. Each image illustrates the trajectories of the robot as well as the switching results between the external controller and policy network across 200 episodes, respectively at the early, middle and late phases of the training procedure. 60
4.6 The policy learned by the robot with various proportional controllers at different stages of the training. Each image illustrates the trajectories of the robot as well as the switching results between the external controller and policy network across 200 episodes, respectively at the early, middle and late phases of the training procedure. 61
4.7 The final result (Reach the target, Crash and Time out) for each episode and the smoothed learning curves. 63

4.8 Policies learned with dense and sparse reward functions where the sparse reward policy takes 43 steps (8.6 sec.) to reach the goal while the dense reward policy takes 53 steps (10.6 sec.). 63
4.9 Gazebo 64
4.10 Real world 64
4.11 The trajectory of the robot in the real world experiment. The yellow rectangles are obstacles and blue circles indicate the sequential targets. 64
4.12 The architecture of the proposed framework. It consists of three sections namely: perception, control and stochastic switch. The perception section processes an observation and generates a corresponding input representation. The control section contains a standard DDPG controller and a number of independent controllers - in this case, a Proportional-integral-derivative (PID) controller and an Obstacle Avoidance (OA) controller. The stochastic switch determines which controllers' action should be selected for navigation. 66
4.13 **(a)** The 4 grey rectangles are obstacles and the blue square represents the robot. A sparse laser is mounted on the robot and its detecting area is illustrated as the green area. **(b)** It shows a more complex environment simulated by *ROS Stage*. **(c)** *ROS Gazebo* is also used for training a model that can be transferred to a real-world environment. Turtlebot 2 (a platform for ground robots) is employed as the mobile platform equipped with a depth camera. 71
4.14 The total reward achieved by our models and other baseline models as comparison. The curve represents the average value of 5 repetitive training procedures and the transparent area indicates the variance of the results. 73
4.15 The smoothed total reward obtained by incorporating different heuristic controllers with DDPG. **SGuidance (PID+OA)** utilises both PID and OA controllers while **SGuidance (OA)** and **SGuidance (PID)** only adopt one of them respectively. **SGuidance (movebase)** is guided by movebase and **DDPG (Vanilla)** is the baseline DDPG without heuristic guidance. 74

4.16 The accumulated reward obtained by learning with different switching mechanisms. **Stochastic** represents our default stochastic switch settings. **Argmax** and **Uniform** are the proposed argmax switch and uniformly distributed switch respectively. **ThompsonSampling** is the Thompson Sampling implemented with Gaussian prior. 75
4.17 Spatial density of using each controller (DDPG, PID or OA) in three different learning periods (top by **SGuidance** and bottom by **ThompsonSampling**). Each row represents the employment of a single controller and each column indicates the early (0-10k steps), middle (40k-50k steps) or late (80k-90k steps) training stages. Note that all grey areas are obstacles. 76
4.18 Illustration of the rate of reaching different final status in an episode when applying **SGuidance** and **ThompsonSampling** respectively. Each column represents a training period and different segments in each bar indicate the usage ratio of different controllers within this period. 77
4.19 Illustration of the impact of using different stochastic switch functions. The left y-axis shows the total reward of all the methods, and the right y-axis shows the total usage of the independent controllers. 78
4.20 Experimental results on the complex environment shown in Fig. 4.13b with sparse rewards. The robot only receives a reward when crashing or reaching the destination, besides a time penalty at each step. The first two rows represent the rate of getting different ending status (crash, time-out or success) of each navigation episode when deploying the guidance or not. The total reward of each episode is also displayed in the last row. 79
4.21 Use rate of each controller by **SGuidance** with different discount factor γ. Each figure draws the use rate of three controllers with a different γ. 81

4.22 **(a)** Comparison of the total reward achieved by **SGuidance** and **DDPG** models in ROS Gazebo simulator. Note that since Gazebo simulates physics properties, the control policy is more difficult to learn and the training takes longer time when compared to ROS stage. **(b)** Real world scenario. A turtlebot is used as the mobile platform and several boxes are placed in the room as obstacles. **(c)** The room layout and obstacles are the black areas. The blue curve represents the trajectory of the robot and the goals are plotted with red circles where the number indicates the sequence. 82

5.1 An example of the visual navigation task with sparse directional guidance. The robot automatically selects from the provided guidance based on its current observation. 85

5.2 The network architecture of SnapNav which consists of two modules. The commander firstly attends to a particular guidance instruction by finding the correct match with the current observation. It then publishes a high level command. The controller then predicts a low level robot action given the command, the estimated depth image and the previous predicted action. Note that commands are represented with the abbreviation "CMD", "EMB" denotes a linear embedding layer, "GRU" indicates the Gated Recurrent Unit [25] and $\bigoplus$ is the concatenation operation. 89

5.3 The videos are firstly segmented based on the estimated optic flow. Then the pseudo labels of direction varying commands (yellow and green sections) are assigned before the agent actually makes the turn. It is important to be labelled randomly instead of according to the ground-truth. The "`Stop`" command is only labelled at the final part of the video (the purple part) and the other frames are set to "`go forward`" as the default command (blue areas). 91

5.4 An example of randomised environments in *ROS Gazebo* and its map. An oracle commander is designed accordingly to give the robot correct commands before it reaches the turning and termination point. The red circle highlights a challenging situation where the intersection is not fully constrained with obstacles which can easily confuse the robot. 93

5.5 The Euclidean distance matrix of the encoded images in a sampled video. 96

5.6 The smoothed learning curves of DRQN, RDPG and DDPG respectively. Each algorithm is trained with 1 million steps in *ROS Gazebo* with the oracle commander. 97
5.7 An example real world test. The legends from top to bottom are: start position, guidance recording position, path in demonstrating phase, random obstacles in testing phase, robot observations and robot trajectory in testing. The attention location is illustrated by different colours in the robot trajectory which corresponds to the colour of actions in the guidance. The red and green circles emphasises the changing visual appearance and geometry of the environment between demonstration and testing. 98

Chapter 1

Introduction

1.1 Motivation

In this emerging era of artificial intelligence, the autonomous operation of mobile robots, drones and self-driving cars, is fundamentally predicated on the existence of a reliable and intelligent navigation system. The problem of **navigation** for mobile robots has been investigated for more than forty years [41]. Its aim is conceptually simple: to search for an optimal path from the current robot position to the destination point whilst avoiding obstacles. However, making the system work robustly across a wide variety of situations is in practice very difficult, as will be discussed below.

To safely reach the goal position, a conventional robot navigation system is explicitly divided into three components, a mapping and localisation module, a global path planner and a local planner. The local planner follows a sequence of waypoints proposed by the global planner and a good global plan mainly is contingent on an accurate map and good position estimation. The above navigation system is fundamentally based on a geometric description of environment, e.g. a map, and the dynamic model of robots.

Using a map brings multiple benefits. By projecting the high dimensional observation, such as a camera image, into a three dimensional pose in the map, the entire planning and control system becomes computationally tractable. Furthermore, the optimality of the global path can be easily ensured. However it also has some limitations. Firstly, the construction of an accurate map for the environment is time and labour consuming and always requires expert knowledge. Secondly, maintaining and updating the map can be even more expensive from a long-term perspective, especially in the face of dynamic changes. Thirdly, the control performance completely

depends on the mathematical model of the robot which however is usually simplified or linearised and eventually weakens the robustness of the navigation system.

Mapless navigation, as an alternative, is more widely considered as a remedy to relieve the navigation system from the prerequisite of a map as it usually models a direct mapping between sensory inputs and robot actions. Unfortunately, without a map, the global planning for the optimal path is extremely difficult. Therefore, mapless navigation is more frequently applied in tasks without explicit destinations, e.g. obstacle avoidance, or with a known destination in robot local coordinate frame, e.g. local navigation. In this context, mapless navigation is similar to the behaviour based navigation[87] which has no high level reasoning process based on environmental prior knowledge. However, they have a large difference in terms of global navigation where the robot has no access to the destination in its local coordinate system. To achieve global navigation, which the behaviour based navigation usually cannot tackle with, the mapless navigation system is usually provided with some kind of environmental information. This can be videos as in Visual Teach and Repeat (VT&R) [37] which are relatively simple to match with camera observations, but is dense and inefficient in terms of memory and communication. It is worth noting that both local and global navigation aim at reaching the destination through the shortest path and the only difference is whether the relative pose between the destination and the robot at each time step is given or not. Natural language [24], as one of the most efficient communication media, has also been explored as the guidance in mapless navigation but its sparsity and the cross-modality problem between language instructions and sensory observations largely hinders its application in real time control.

As an emerging paradigm, **Deep Reinforcement Learning (DRL)**, has been shown to be able to master complicated tasks such as Go [135] and digest observations from various domains, e.g. images [109], laser scans [146] and languages [133]. As a purely data-driven method, DRL outperforms traditional approaches in several ways. One is getting rid of handcrafted control rules that are usually sub-optimal since they largely depend on the limited experience and the cognition of the task from the designer. The ascendance of AlphaGo Zero [137] over human players has definitively proven that a purely data-driven method can surpass the limitations of human strategy. Another advantage is in its capability of learning from a high dimensional domain, either in terms of sensor observations or the physical model of robots, which is both information-theoretically and computationally hard for heuristic approaches.

In contrast with other machine learning mechanisms, e.g. supervised learning or imitation learning, reinforcement learning is more applicable to problems with a

complex reward structure and a strong requirement for sequential interactions with the environment [66]. Mapless navigation is a characteristic example where the agent may finally get a single positive reward only when it manages to successfully reach the destination after traversing a long distance, without crashing or running out of time. The final reward is the result of multiple actions on the way, including approaching the destination and avoiding obstacles. These challenges make DRL difficult to train as the search space is immense.

However, the practical application of DRL in real robotic tasks raises significant challenges discussed in the next section. Seeking solutions to these challenges is the ultimate goal of this thesis, with the vision of creating practical DRL approaches for mapless visual navigation that are efficient to train, generalisable, and capable of being deployed on real robots.

1.2 Research Challenges and Questions

Although DRL has shown an astonishing performance in virtual games, it remains challenging to achieve comparable accomplishments in noisy and dynamic real-world problems.

The most significant problem is its **inefficient sample usage**. In complex real-world environments, DRL is plagued by the challenge of high variance. This is caused by many reasons [66]. One is the exponentially growing data and computation needed for a high dimensional real robotics system which is termed as "Curse of Dimensionality" [11]. To fully explore the immense state and action space, numerous learning samples must be accumulated through the interactions with the real-world environment. Hence, most robot reinforcement learning problems cannot be trained from real-world samples, as robotic hardware is expensive and prone to damage. In addition, obtaining training episodes from real robots is slow and requires manual intervention. As a result, conducting the training procedure in a simulator and then adapting the learnt policy to real-world scenarios [146, 181] is a more feasible option. The training period can be decreased by speeding up the simulation and the virtual world can be easily reset without real damage after a collision. On the flip side, using a simulation also brings the problem of under-modelling and model uncertainty, prohibiting us from directly transferring the learnt policy from virtual to the real world, a problem often referred to as the "**reality gap**". One of the key challenges is therefore to bridge the gap between virtual and real worlds in terms of sensory observations such as visual appearance and physical phenomenons, e.g. robot kinetics.

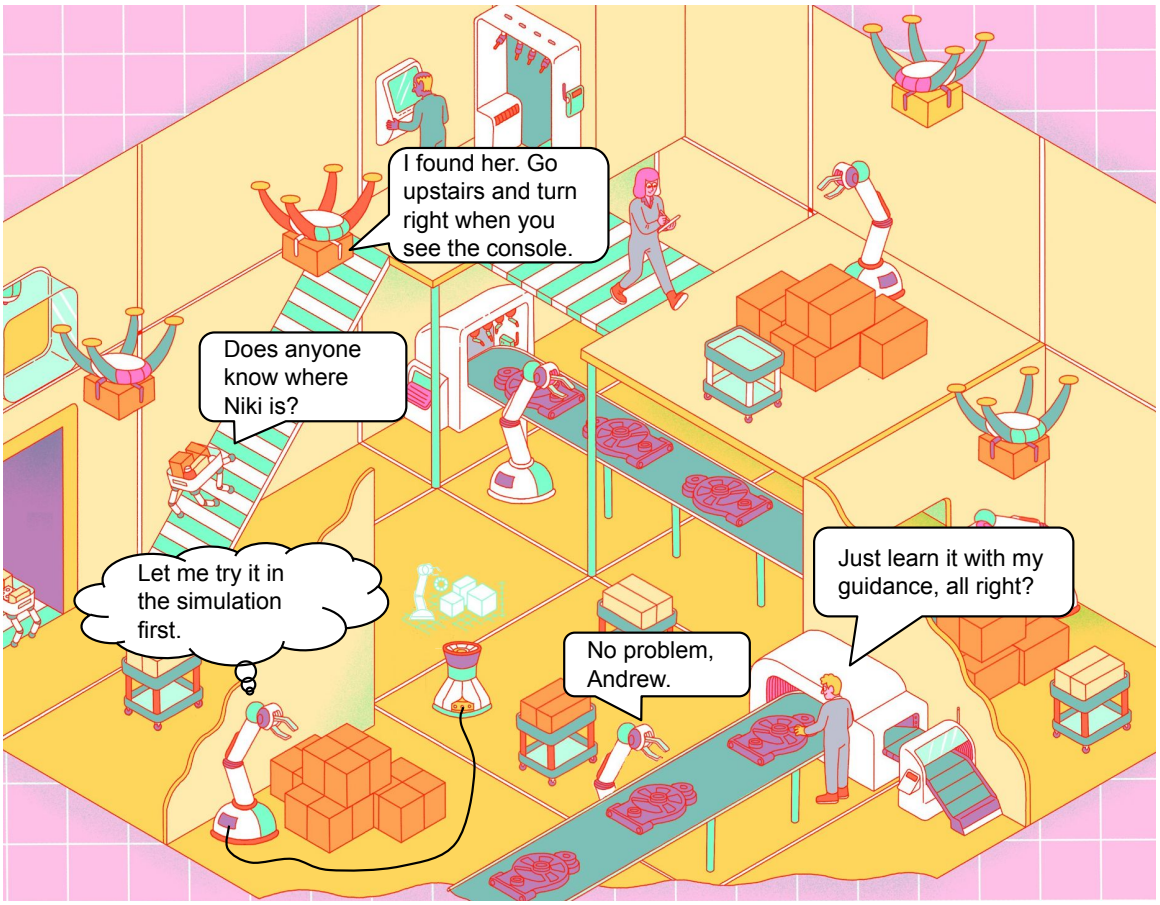


Figure 1.1: A comic (based on the illustration by YimeIsGreat) that images the near future where robots and human will have close cooperation. It also demonstrates some core concepts mentioned in this thesis: 1) Transferring the learnt policy from simulation to the real world. 2) Accelerating the learning process with extra guidance. 3) Learning to navigate with sparse directional instructions and visual references.

Another important challenge is the **generalisation ability** to transfer learnt experience from known to unknown scenarios. In our case, it refers to the ability to navigate in an environment with an unfamiliar geometry where it is impossible for the robot to reach the destination quickly by recalling the memorised route from the training environment. This is relatively easy for primitive tasks such as obstacle avoidance or local navigation where the optimal action can be learnt from local observations but is non-trivial for global navigation where additional guidance must be supplied.

Goal specification is another significant challenge. Since in reinforcement learning, the expected behaviour is implicitly indicated by the reward function, a suitable reward function is of importance but can be quite difficult to design in a real robotic problem. One contradiction, for instance, is between the sparse and dense (frequent)

reward. Giving a dense reward with the intention of influencing behaviour based on expert prior knowledge can largely speed up training but may finally encourage the agent to learn a sub-optimal policy for the task. On the other hand, a sparse reward function requires the learner to perform a great deal of random exploration as it only gains positive experiences after observing a wide variety of negative rewards i.e. rather than learning how to do the task directly (positive reward) it first has to learn how not to do the task (negative reward). This leads to a very slow convergence of the DRL policy.

With the goal of proposing a DRL mapless navigation system in real-world scenarios, this thesis aims to address the following questions:

1. **How can we transfer the learnt policy from virtual to real environments?**

   This is the key issue of utilising simulators for training DRL models, since it affects whether the learnt policy is finally valuable to the real robot problem. However, as noted above, training in a simulator is the only realistic way of rapidly training DRL networks. It can be considered from two perspectives, transferring the perception or action, which depends on the utilised simulator. Perception represents how the robot understands the surrounding environment from its observations and the action reflects the selected control signal given an observation. The simulated world is a highly simplified model of the real-world in terms of the appearance and diversity of objects as well as in the abstraction of physical phenomena. Hence the simulated robot perceives and behaves differently from the real one. How to reduce or even remove this gap is the main topic of *Chapter 3*.

2. **How can we accelerate training in simulation?**

   Although simulators greatly speed up the training process relative to the real-world, they still require a huge number of episodes to learn good policies. Enhancing data efficiency and reducing the training variance is our second approach for improving DRL algorithms.

   Even in a simulator which can run several times faster than the real-world, it may still take a few days to train a good network. In the machine learning community, a large amount of research has focused on introducing general bias [101, 127, 2, 45] that can generalise well to different problems. However, these algorithms do not consider the context of a specific task, which can be valuable

for training. The robot navigation task, in our case, does have existing solutions which could aid DRL training. Although these solutions may not be optimal, they still outperform a random exploration policy in most cases. How to fully exploit and benefit from these prior approaches and tightly combine them with DRL to accelerate training is the question discussed in *Chapter 4*.

3. **How can we carry out long term mapless navigation?**

   Mapless navigation is not difficult for short-term navigation tasks such as obstacle avoidance or local navigation with a known destination where the local observation of the robot is sufficient for making the action decision. However, the problem of long term navigation is different. It requires a global planning procedure for selecting the correct local action which usually relies on the description of the environment. Motivated by human behaviours when querying the route to the destination, we discuss how a mobile agent can learn to navigate with sparse directional guidance and visual reference in *Chapter 5*.

1.3 Contribution

In this section we firstly summarise the contributions of each chapter in this thesis as follows:

1. In Chapter 3 we propose a two-phase deep neural network to achieve monocular vision based obstacle avoidance. A D3QN network architecture is applied for learning a robotic task for the first time, with the added contribution of enhancing learning efficiency. As a significant and novel contribution, the estimated depth information is used for the first time to bridge the gap of monocular visual perception between simulations and the real world. Extensive experiments are conducted to show the high performance of our visual obstacle avoidance system in various real-world scenarios. This work is published in:

   Linhai Xie, Sen Wang, Andrew Markham, and Niki Trigoni. Towards monocular vision based obstacle avoidance through deep reinforcement learning. Robotics: Science and System 2017 workshop on New Frontiers for Deep Learning in Robotics.

2. In the fourth chapter we propose a new framework for effectively incorporating heuristic knowledge to overcome the high variance issue in learning a policy using DDPG. It is the first time that either expert or naive classic controllers can be 'softly' combined into the learning process. We demonstrate that they can

not only boost the learning process of an underlying off-policy DRL approach, i.e. DDPG in the early stage but also be completely discarded when they are outperformed by the learnt policy, leaving DDPG to learn from self-exploration. We present both deterministic (AsDPPG) and stochastic (SGuidance) techniques and compare their relative merits. Both algorithms largely accelerate and stabilise the training of DDPG, where SGuidance achieves near-oracle performance. Additionally, we demonstrate how the proposed switch mechanism dynamically learns which controller to choose based on the input and anticipated reward, outperforming switches which only learn from anticipated reward (e.g. Thomson sampling). We also show that this technique is particularly important for sparse rewards, where vanilla DDPG is unable to learn how to complete an entire episode. This work has been published as follows:

Linhai Xie, Sen Wang, Stefano Rosa, Andrew Markham, and Niki Trigoni. Learning with training wheels: speeding up training with a simple controller for deep reinforcement learning. In 2018 IEEE International Conference on Robotics and Automation (ICRA), pages 6276−6283. IEEE, 2018

Linhai Xie, Yishu Miao, Sen Wang, Phil Blunsom, Zhihua Wang, Changhao Cheng, Andrew Markham, and Niki Trigoni. Learning with stochastic guidance for robot navigation. Under review at IEEE Transactions on Neural Networks and Learning Systems.

3. In Chapter 5 we propose a novel visual navigation system which enables the robot to navigate in unknown environments with very sparse guidance. A two-level architecture of the network is designed and each module can be trained with different learning mechanisms and data from various domains and sources. It provides the navigation system with a higher degree of autonomy and can easily transfer the learnt policy from simulation to the real world. The high-level commander is trained with a novel self-supervised training approach with multiple learning signals. Finally the trained network is shown to navigate in real-world experiments robustly. This work appears in the following:

Linhai Xie, Andrew Markham, and Niki Trigoni. SnapNav: Learning mapless visual navigation with sparse directional guidance and visual reference. Accepted at 2020 IEEE International Conference at Robotics and Automation (ICRA).

Other contributions during my PhD career are also listed as follows:

1. Linhai Xie, Sen Wang, Andrew Markham, and Niki Trigoni. "GraphTinker: Outlier rejection and inlier injection for pose graph SLAM." In 2017 IEEE/RSJ

International Conference on Intelligent Robots and Systems (IROS), pp. 6777-6784. IEEE, 2017.

2. Linhai Xie, Yishu Miao, Sen Wang, Phil Blunsom, Zhihua Wang, Changhao Cheng, Andrew Markham, and Niki Trigoni. "Learning with Stochastic Guidance for Navigation." NeuraIPS 2018 workshop on Infer2Control.

3. Zhihua Wang, Stefano Rosa, Linhai Xie, Bo Yang, Sen Wang, Niki Trigoni, and Andrew Markham. "Defo-Net: Learning body deformation using generative adversarial networks." In 2018 IEEE International Conference on Robotics and Automation (ICRA), pp. 2440-2447. IEEE, 2018.

4. Lu Chris Xiaoxuan, Yang Li, Peijun Zhao, Changhao Chen, Linhai Xie, Hongkai Wen, Rui Tan, and Niki Trigoni. "Simultaneous localization and mapping with power network electromagnetic field." In Proceedings of the 24th Annual International Conference on Mobile Computing and Networking, pp. 607-622. ACM, 2018.

5. Changhao Chen, Yishu Miao, Chris Xiaoxuan Lu, Linhai Xie, Phil Blunsom, Andrew Markham, and Niki Trigoni. "MotionTransformer: Transferring Neural Inertial Tracking between Domains." In Proceedings of the AAAI Conference on Artificial Intelligence, vol. 33, pp. 8009-8016. 2019.

6. Hu, Qingyong, Bo Yang, Linhai Xie, Stefano Rosa, Yulan Guo, Zhihua Wang, Niki Trigoni, and Andrew Markham. "RandLA-Net: Efficient semantic segmentation of large-scale point clouds." arXiv preprint arXiv:1911.11236 (2019).

Chapter 2

Background

In this chapter, we provide a comprehensive overview of fundamentals and recent work on *Reinforcement Learning* and *Robot Navigation*. We firstly explore a few Deep Reinforcement Learning (DRL) solutions and discuss their key strengths and weaknesses. Then the formulation and limitations of map-based robot navigation are discussed, followed by the motivation and recent work on mapless navigation.

This chapter is organised as follows: Sec. 2.1 describes the formulation of a reinforcement learning problem and its solutions. Sec. 2.2 and Sec. 2.3 review the map-based robot navigation and the mapless robot navigation respectively. Finally the limitation of state-of-the-art DRL based navigation methods are discussed in Sec. 2.4.

2.1 Reinforcement Learning

Reinforcement Learning (RL) is a discipline of machine learning investigating how to learn from interactions, which is thought to be the most natural learning experience to humans. When an infant, who is born with no prior knowledge, sees and plays with the environment, they can swiftly accumulate information about the cause and effect in the physical world. It is achieved without any guidance and only from their interactions with the environment, namely, the actions and the sensory observations. This kind of learning mechanism never ends in our life and is fundamental to many intelligent agents.

2.1.1 Markov Decision Process

In the theoretical formulation, the RL problem is modelled as a *Markov Decision Process* (MDP) which maps from states to actions, so as to maximise a numerical

reward signal. Therefore we need to firstly overview the principle of an MDP [10].

The Markov property states that the future state is independent of the past given the present state. More specifically, in a sequence of states $x_0, x_1, x_2, ..., x_T$ from time 0 to T, we say a state $x_t \in X$ has the Markov property if and only if

$$P(x_{t+1}|x_t) = P(x_{t+1}|x_0, x_1, ..., x_t), \tag{2.1}$$

where X is the state space and P is the transition probability function which describes a distribution of what the next state will be given the current and historical states. Based on this property, the Markov process is a memory-less random process, namely a sequence of random state $x_0, x_1, ...$ satisfying the Markovian dynamics. When we sample this process according to the transition possibility function P, the obtained sequence is called an *episode*. Additionally, with a *reward* function r as a value judgement, we obtain the Markov reward process. Here the reward function r is used to assess the sampled episode by accumulating the instant reward r_t at each state transition as $\sum_{\tau=1}^{T} r_\tau$. Finally the Markov Decision Process is defined by introducing the *action* $a \in A$ in the transition function as $P(x_{t+1}|x_t, a_t)$. Therefore a MDP is also formulated as a tuple (X, A, P, r), consisting of a state space X, an action space A, a transition possibility function P and a reward function r. Additionally, the sequence $x_0, a_1 x_1, ...a_T, x_T$ sampled from time 0 to T is termed a trajectory s.

2.1.2 Essential Elements

The essential elements for RL problems are illustrated in Fig. 2.1. At time $t \in [0, T]$ the learner, which is termed an agent, takes an action $a_t \in \mathscr{A}$ according to the state x_t that represents the current state of the environment. After executing the action, the robot receives a reward r_t given by the environment according to the reward function and then transits to the next state x_{t+1}. Note that each action the agent takes at time t may affect all the subsequent observations, actions and rewards in the future, therefore the meaningful feedback of current action can be delayed by many steps. The target of this decision making process is to reach a maximum discounted accumulative future reward $R = \sum_{\tau=t}^{T} \gamma^{\tau-t} r_\tau$ which infers the optimal action sequence of reaching a goal state from the current state. Here γ is defined as a discount factor.

Besides the above definition, there are some other basic concepts in RL we would like to introduce here.

The policy π is the strategy for the agent to determine the action a_t based on the current observation x_t. It maps directly from the state to action:

$$a_t = \pi(x_t) \tag{2.2}$$

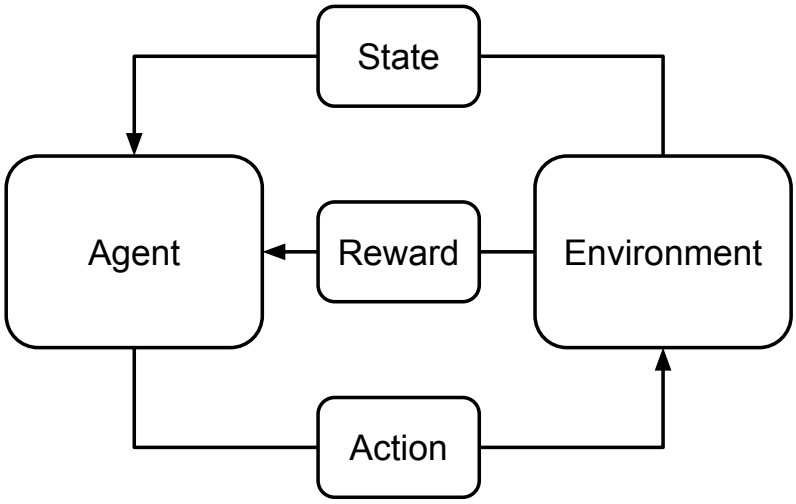


Figure 2.1: In an RL problem, the learning agent observes the state of the environment and carries out an action to affect the environment at each time step. According to different actions, the environment returns a numerical reward as the feedback to the agent, assessing whether it is the desired behaviour.

In this formulation we are assuming a deterministic policy but it can also be stochastic where the actions are sampled from a distribution. Furthermore, the actions can be either continuous and discrete. The continuous action usually results in a much larger search space for the learner and is more difficult to learn in most cases.

The value V is the expected sum of discounted reward in future $\mathbb{E}[\sum_{\tau=t}^{T} \gamma^{\tau-t} r_\tau | \pi]$ where the discount factor γ, also used in R, is used to downweigh the long-term future reward and avoid an infinite value in an endless sequence. Value V is defined given a specific policy π and state x as $V_\pi(x)$. The Q-value is similar to value V. But the difference is that the Q-value $Q_\pi(x, a)$ is further defined on the action as an extra parameter. Since the reward r_t is a measure of how beneficial the action a_t adopted under state x_t is, we can consider that the value $V_\pi(x)$ represents an assessment of the policy π given an state x. The Q-value $Q_\pi(x, a)$ further measures whether a specific action a is good under the state x.

2.1.3 Comparison with Other Learning Paradigms

RL is different from the broad classes of *supervised learning* or *unsupervised learning*. The former is the most widely used learning approach which aims to find a mapping function from input to output variables given labelled data. The labelled learning samples link every optimal output with an input variable. Training is carried out by minimising the error between the prediction from the agent and the label for each

input state. With sufficient training examples, the agent can generalise the learnt knowledge to unseen situations. This is an efficient learning approach but does not suit interactive problems where there is often not an oracle providing the optimal solution. Conversely in unsupervised learning, the learner aims to find the hidden structure of the unlabelled data which, however, does not serve the target of reaching a goal with a sequence of actions. Therefore, RL is the most suitable paradigm for learning from interactions. It also fits in many complicated robotic applications where the optimal trajectory is unavailable due to the problem of high dimensionality. Hence the solution must be explored from interacting with the environment.

RL has a featured challenge — the exploitation and exploration dilemma. On the one hand, the agent prefers to exploit the actions which it estimates that they will obtain a high reward. On the other hand, those highly rewarded actions can only be discovered by trying unexplored actions. Additionally, the design of the reward function is also of critical importance. With sparse and delayed rewards, e.g. only using the final result as the reward when playing chess, the agent may capture positive learning signal only after millions of trials. Nevertheless, learning with a dense but handcrafted reward function may prevent the agent from progressing to the optimal policy.

2.1.4 History

The history of RL can be divided into three threads and we briefly describe them according to [141]. The first one is derived from the optimal control [58] and a main approach is using value functions and dynamic programming [12]. Bellman has also introduced a discrete stochastic version of optimal control problem which is termed Markovian Decision Process (MDP) [10] and the policy iteration approach is later proposed by Ron Howard [56] to solve MDPs. These concepts and theories have paved the road for modern RL. Another thread is called trial-and-error which we are most familiar with. It is built based on the authentic "reinforcement" learning concept firstly phrased by Edward Thorndike as "Law of Effect" [149]. Briefly speaking, it is to increase the tendency of choosing actions with good feedback and decrease the preference of the one with bad feedback. By following this thread, the term "reinforcement learning" appeared for the first time and several important problems in RL are investigated, e.g. credit assignment, in [97]. The last thread is temporal-difference learning. The core idea is to learn the difference between the estimations of a value (future total reward as an example) at two successive time steps. The ideas from Klopf [65], Sutton [140] and other researchers have contributed a lot in

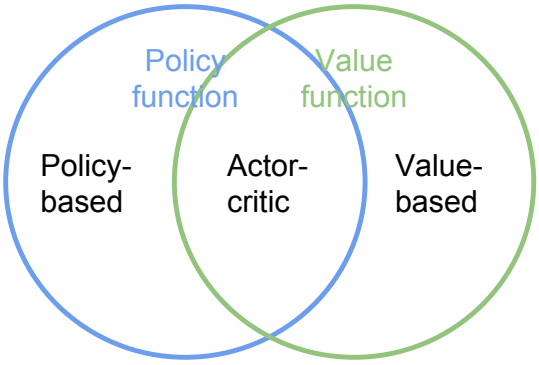


Figure 2.2: Three categories of model-free RL methods, i.e. policy-based , value-based and actor-critic approaches.

refining this thread. Finally, by combining the temporal-difference and trial-and-error threads, the actor-critic architecture was introduced in 1983 [8]. In addition, the famous Q-learning approach was proposed in [163], bringing temporal-difference and optimal control threads together.

In recent years, deep learning has stimulated a boom in research on Deep Reinforcement Learning (DRL). Both the value estimation and the policy searching problem in highly complex domains are relieved by the deep neural network that is theoretically able to approximate any function. In this thesis, three DRL algorithms are exploited and will be mentioned in different chapters with details, i.e. Deep Q-learning [103], deep deterministic policy gradient [79] and a "deep" version of REINFORCE algorithm [164].

2.1.5 Methods

In this thesis we focus on model-free approaches. That is because with recent development of deep neural networks, model-free RL has been proved to be effective in high-dimensional state spaces. However, learning the model of the real-world environment in a high-dimensional state space is very computationally expensive.

The model-free methods fall into three different categories, i.e. value-based, policy-based and actor-critic methods.

Policy-based approaches, e.g. REINFORCE[164], optimise the policy function π directly. A main approach is to estimate the gradients of the policy function w.r.t. the total rewards. By following those gradients, the policy will be improved to obtain greater rewards. Value-based methods learn from an opposite way as they only estimate the value or Q-values, such as Q-learning [163]. With the known sum

of future rewards the policy becomes easy, choosing the one that can maximise the future reward.

The primary advantage of policy-based approaches is that they have a better convergence property. This is because the policy function is continuously updated with gradients which keeps improving the policy and is is guaranteed to converge to at least a local minimum[114]. Another one is its natural compatibility in continuous or highly dimensional action spaces. The maximisation operation, which is usually used in value-based methods, can be prohibitively expensive when there are millions of actions to choose. The value function must accurately estimate the Q-value for each action to finally find the optimal one. The final advantage is that policy-based approaches can learn stochastic policies which can outperform a deterministic one in some cases, and this will be further discussed in Chapter 5. However its disadvantages compared to value-based methods are not trivial. Policy-based methods are easier to fall into local minima and their learning process suffers from the high variance[114, 67] which is the main motivation in Chapter 4.

The third category is termed as actor-critic which uses both the value function and policy function. Both functions are iteratively updated during the learning process. The critic function is updated towards a more accurate estimation of values or Q-values. Then the policy function is optimised through the directions suggested by the critic function and learns better actions in different situations. The gradients calculated from the critic function are more stable than the ones in policy-based methods since the latter are estimated from the total rewards of sampled trajectories while the former are based on the expectation estimated from the critic[67].

2.1.6 Deep Reinforcement Learning Algorithms

We are now in a position to discuss 3 different DRL algorithms which are implemented and broadly tested in this thesis.

2.1.6.1 Deep Q Network

DQN [104] is typically a value based DRL algorithm. Given the policy $a_t = \pi(x_t)$, the Q-value of a state-action pair (x_t, a_t) can be defined as follows

$$Q^{\pi}(x_t, a_t) = \mathbb{E}[R_t | x_t, a_t, \pi],$$

where $R_t = \sum_{\tau=t}^{T} \gamma^{\tau-t} r_\tau$ is the total future reward from time t, discounted by γ. The Q-value function can be computed using the Bellman equation

$$Q^{\pi}(x_t, a_t) = \mathbb{E}[r_t + \gamma \mathbb{E}[Q^{\pi}(x_{t+1}, a_{t+1})|x_{t+1}, a_{t+1}, \pi]|x_t, a_t, \pi].$$

With a deterministic greedy policy which chooses the optimal action each time as $Q^*(x_t, a_t) = \max_{a_t} Q(x_t, a_t)$, we can have the optimal Q-value function

$$Q^*(x_t, a_t) = \mathbb{E}[r_t + \gamma \max_{a_{t+1}} Q(x_{t+1}, a_{t+1})|x_t, a_t], \tag{2.3}$$

which indicates that the optimal Q-value we can obtain at time t is the current reward r_t plus the discounted optimal Q-value available at time $t+1$, rather than computing the Q-value function directly over a large state space.

Then the deep neural networks which is parameterised with θ^Q is used as a Q-value estimator $Q(x_t, a_t|\theta^Q)$. Given a specific learning sample (x_t, a_t, r_t, x_{t+1}), the loss function for optimising the network parameters is formulated as:

$$\mathcal{L}(\theta^Q) = [y - Q(x_t, a_t|\theta^Q)]^2 \tag{2.4}$$

where the the target $y = r_t + \gamma \max_{a_{t+1}} Q(x_{t+1}, a_{t+1}|\theta^Q)$. Note that DQN is only suitable in discretized action spaces due to the max operation contained in the greedy policy π over all possible actions.

Based on the above vanilla DQN, **double network** [156], **prioritised experience replay** [128] and **dueling network** [161] are 3 techniques that can largely improve its performance.

Double DQN In the above DQN formulation, the greedy policy utilises the same Q value to evaluate and select an action which, as indicated by Hasselt et al. [156], leads to an overoptimistic value estimation. To mitigate this problem the following target is applied in DDQN:

$$y' = r_t + \gamma \max_{a_{t+1}} Q(x_{t+1}, a_{t+1}|\theta^{Q'}) \tag{2.5}$$

where $\theta^{Q'}$ is the parameter for a target Q network which slowly tracks θ^Q. Therefore all the possible actions are assessed by the target Q network whilst the greedy selection is still made by the original Q network.

Prioritised experience replay As an off-policy algorithm, DQN stores the experienced learning samples in a replay buffer and uniformly samples a batch of data for training. [128] proposed to assign higher sampling probabilities to the learning samples which can make a larger learning progress and this significance to the learning progress is approximated by the error of Q value estimation. It demonstrates a faster learning speed as well as a better performance on multiple Atari games.

Deep Recurrent Q Learning DRQN [49] is proposed by Hausknecht et al. for solving the Partially Observable MDP, namely POMDP, where DQN degrades its performance due to the incomplete state observations. Its core difference to DQN is leveraging advances in Recurrent Neural Networks (RNN) to incorporate historical information for Q-value estimation.

Dueling network With the intuition that it is unnecessary for all actions to be estimated at each state s, Wang et al. [161] propose the dueling network architecture. In DQN only one stream of fully connected layers is constructed after the convolution layers to estimate the Q-value of each action-state pair. However, in the dueling network, two streams of fully connected layers are built to compute the value and advantage functions separately, which are finally combined together for computing Q-values. The definition of advantage $A(x_t, a_t|\theta^\alpha)$ is:

$$Q(x_t, a_t|\theta^\alpha, \theta^\beta) = V(x_t|\theta^\beta) + A(x_t, a_t|\theta^\alpha)$$

where θ^α and θ^β respectively represent the parameters of the advantage stream and value stream in the Q network. Then to solve an unidentifiable problem, the following formulation is finally utilised in dueling network:

$$Q(x_t, a_t|\theta^\alpha, \theta^\beta) = V(x_t|\theta^\beta) + [A(x_t, a_t|\theta^\alpha) - \frac{1}{|\mathscr{A}|}\sum_{a'} A(x_t, a_t^{'}|\theta^\alpha)]$$

where $\mathscr{A}$ represents the action space.

2.1.6.2 REINFORCE Algorithm

REINFORCE [164] is a policy based algorithm which is at a completely opposite position in the spectrum compared to DQN. In essence, REINFORCE learns a stochastic policy by reinforcing the probability of sampling actions $p = P^\pi(a_t|x_t, \theta^\pi)$ with high expected total rewards $\mathbb{E}(\sum_{t=0}^{T} r_t)$. Note that θ^π represents the parameter

of the policy network and here we do not consider the discount factor γ currently in the derivation firstly. The objective function is just defined as follows

$$J(\theta^\pi) = V_0 = \mathbb{E}[\sum_{t=0}^{T} r_t|\pi] = \sum_s \sum_{t=0}^{T} P(x_t, a_t|s) r_t$$

where s is a single trajectory of $x_0, a_1, x_1, a_2, ..., x_T, a_T$, which depends on the policy π, and $P(x_t, a_t|s)$ indicates the probability of the occurrence of x_t and a_t given s. Then by differentiating both sides we can obtain the following equation:

$$\begin{aligned}
\nabla_{\theta^\pi} J(\theta^\pi) &= \sum_s \sum_{t=0}^{T} \nabla_{\theta^\pi} P(x_t, a_t|s) r_t \\
&= \sum_s \sum_{t=0}^{T} P(x_t, a_t|s) \frac{\nabla_{\theta^\pi} P(x_t, a_t|s)}{P(x_t, a_t|s)} r_t \\
&= \sum_s \sum_{t=0}^{T} P(x_t, a_t|s) \nabla_{\theta^\pi} \log P(x_t, a_t|s) r_t \\
&= \mathbb{E}[\sum_{t=0}^{T} \nabla_{\theta^\pi} \log P(x_t, a_t|s) r_t|s].
\end{aligned}$$

During training, a Monte Carlo approach is used to approximate above gradients with N randomly sampled trajectories.

$$\nabla_{\theta^\pi} J(\theta^\pi) \sim \frac{1}{N} \sum_{n=1}^{N} \sum_{t=0}^{T} \nabla_{\theta^\pi} \log P(x_t, a_t|s) r_t.$$

Furthermore, by expanding $\nabla_{\theta^\pi} \log P(x_t, a_t|s)$ and discarding the items that do not depend on θ^π and have 0 gradients w.r.t. it we have:

$$\nabla_{\theta^\pi} \log P(x_t, a_t|s) = \sum_{t'=0}^{t} \nabla_{\theta^\pi} \log P^\pi(a_{t'}|x_{t'}, \theta^\pi).$$

Therefore,

$$\begin{aligned}
\sum_{t=0}^{T} \nabla_{\theta^\pi} \log P(x_t, a_t|s) r_t &= \sum_{t=0}^{T} r_t (\sum_{t'=0}^{t} \nabla_{\theta^\pi} \log P^\pi(a_{t'}|x_{t'}, \theta^\pi)) \\
&= \sum_{t=0}^{T} \nabla_{\theta^\pi} \log P^\pi(a_t|x_t, \theta^\pi) (\sum_{t'=t}^{T} r_{t'}).
\end{aligned}$$

After incorporating the discount factor γ,

$$\nabla_{\theta^\pi} J(\theta^\pi) \sim \frac{1}{N} \sum_{n=1}^{N} \sum_{t=0}^{T} \nabla_{\theta^\pi} \log P^\pi(a_t|x_t, \theta^\pi)(\sum_{t'=t}^{T} \gamma^{t'-t} r_{t'}) \tag{2.6}$$

$$= \frac{1}{N} \sum_{n=1}^{N} \sum_{t=0}^{T} \nabla_{\theta^\pi} \log P^\pi(a_t|x_t, \theta^\pi) R_t. \tag{2.7}$$

Finally the parameter of policy network θ^π is updated through:

$$\theta^\pi \leftarrow \theta^\pi + \frac{\partial}{\partial \theta^\pi} J(\theta^\pi)$$

Variance Reduction Since the gradient is approximated with a Monte-Carlo method, it can be very noisy especially in a long episode. Therefore, compared with directly applying Eq. 2.6, some baselines [142] b_t with zero expectations are proposed to reduce the variance of the gradients estimation as follows:

$$\frac{1}{N} \sum_{n=1}^{N} \sum_{t=0}^{T} \nabla_{\theta^\pi} \log P^\pi(a_t|x_t, \theta^\pi)(R_t - b_t)$$

where the baseline b_t can either be a moving average of the recent obtained rewards or learnt from a neural network by optimising the expectation of $R_t - b_t$ towards 0.

2.1.6.3 Deep Deterministic Policy Gradients

DDPG [79] falls into the category of actor-critic method which is a combination of value and policy based approaches. It consists of two networks, an actor network $\pi(x_t|\theta^\pi)$ and a critic network $Q(x_t, a_t|\theta^Q)$, deterministically estimating the action a_t given x_t and the Q-value of them respectively.

Similar to REINFORCE algorithm, the actor network is also updated by policy gradients. However, it applies following equation as the policy gradients as proved in [136]:

$$\nabla_{\theta^\pi} J(\theta^\pi) \sim \mathbb{E}[\nabla_{\theta^\pi} Q(x_t, a_t|\theta^Q)|x_t, a_t = \pi(x_t|\theta^\pi)] \tag{2.8}$$

$$= \mathbb{E}[\nabla_{a_t} Q(x_t, a_t|\theta^Q) \nabla_{\theta^\pi} \pi(x_t, \theta^\pi)|x_t] \tag{2.9}$$

which means the experienced total future reward R_t in Eq. 2.6 is the Q value estimated from the critic network and its derivatives w.r.t. θ^π follows a chain rule which can be decomposed into two parts. Accordingly, the learning process of the critic network is also similar but slightly different from DQN where the target y in Eq. 2.4 is changed to:

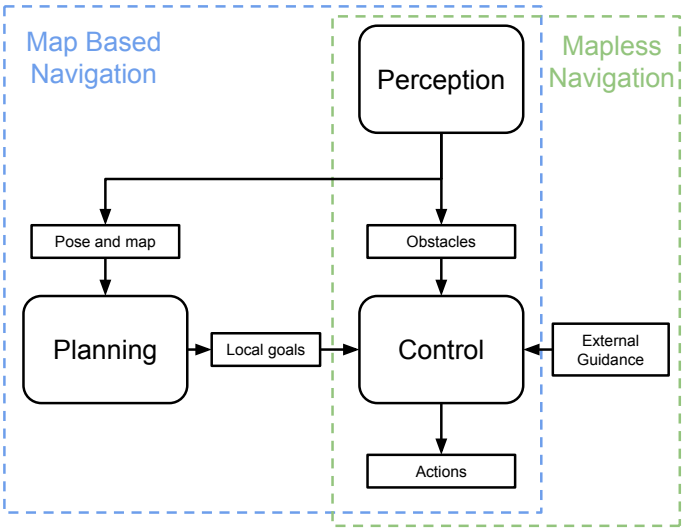


Figure 2.3: The composition of map-based navigation and mapless navigation systems. The local goals are the sequential waypoints along the planned path provided by the global planner.

$$y = r_t + \gamma Q(x_{t+1}, \pi(x_{t+1}|\theta^\pi)|\theta^Q),$$

as the critic network is responsible for evaluating the current policy of the actor network instead of searching for the optimal Q-value over possible actions.

Recurrent DPG RDPG, like DRQN, is the recurrent counterpart of DDPG which also contains RNN both in the critic network and the actor network and is updated with back-propagation through time (BPTT). The recurrent network effectively mitigates the problem of partial observability in some tasks and enables DPG to solve tasks that require preserving information over many time steps.

Note that here we only explain those DRL algorithms theoretically and their implementation details will be further introduced in each chapter.

2.2 Map-based Robot Navigation

Robot navigation is a classic robotic problem and has multi-spectral applications across a plethora of industries. Instead of carrying out an exhaustive review on this immensely large field, we only consider the problem of the navigation of wheeled robots in indoor environments.

Generally the goal of navigation is to reach a destination through a collision-free path with the least time cost. As shown in Fig. 2.3, it can be divided into

three sub-tasks, i.e. perception, planning and control. The perception task involves sensing, representing the environment and localising the robot, answering the question about *where I am*, while the path-planning is to search the optimal path towards the destination, solving the problem of *where to go*, and finally the control system navigates the robot according to the plan and simultaneously prevents collisions, telling the robot *how to go*. Note that because the control stage does not need a map, we will mention it in the next section (Sec. 2.3) as local navigation.

2.2.1 Sensing

Understanding the state of the robot in the environment usually focuses on extracting the geometrical relationship between the robot and the objects in the environment which relies on the on-board sensors of the robot. Therefore, measuring the distance between the robot and obstacle is the most straightforward sensing approach, as the raw sensing data can be directly utilised for obstacles avoidance and constructing the map of the environment. To this end, LiDAR [74, 130, 176], ultrasonic sensors [34, 107] and depth cameras [106, 26] are the common options.

LiDAR technology uses light pulses or laser beams to determine the distance between the sensor and the object. The beam travels to the object and is reflected back to the source and the elapsed time is then used to calculate the distance. Ultrasonic ranging is based on emitting a pulse of sound energy to a target surface and measuring the time taken for an echo to return. The depth camera, using the Microsoft Kinect [177] as an example device, consists of an infrared laser projector combined with a monochrome CMOS sensor, which captures video data in 3D under any ambient light conditions. One of its drawback is that the intense sunlight can completely blind the Kinect infrared sensor due to excessive infrared rays causing interference.

Vision [29], as an alternative to ranging sensors, is another sensor modality broadly exploited in robot navigation which captures the appearance information of the objects in the environment. Although geometric information cannot be obtained directly from a single camera image, it can be inferred from a pair of images given the intrinsic and extrinsic parameters of the camera. Additionally, by considering the ubiquity of a commercial camera and the semantic information contained in an RGB image, vision based navigation has an exponentially spreading influence in robot navigation research.

2.2.2 Localisation and Mapping

In this section, we review the pure localisation approaches that do not construct map of the environment, the different representations of a map and the Simultaneous Localisation and Mapping (SLAM) approach.

2.2.2.1 Localisation

In many map-based robot navigation systems, the environment has been previously described in a metric map and is always available to the robot. Therefore, to understand the state of the robot in the environment only requires a localisation system, providing the robot pose in the map coordinate system. This section reviews prior work on Indoor Positioning Systems (IPS) as the thesis only focuses on robot navigation in indoor environments. We classify IPS by looking at whether they need prior knowledge about the environment.

We firstly review those that assume prior knowledge of the environment. For systems utilising fingerprinting techniques, a survey of the Received Signal Strength (RSS) at the specified locations in the covered area is necessary, no matter what kind of sensor is employed. Then by comparing the similarity between the measurements and the fingerprints in the database, the localisation of the agent can be obtained. Wifi is a common choice as in [132]. However, another work [6] has utilised different sensor modalities including sound, light, colour and even the users motion. Fingerprinting is a completely data-driven method which means the data collected directly determines its performance. Once the environment is changed, the survey process must be conducted again and this process of obtaining environment information is usually expensive. There are also many systems applying lateration and angulation methods, which require the position of signal emitters like Access Points (AP) to be known. Thus by applying a physical model of signal propagation, the system can calculate the position of users with signals received from multiple or a single emitter [131, 68]. Some of these systems can achieve high accuracy but one of their limitations is that certain materials and the rearrangement of objects within the environment affect the propagation of radio waves and thus influence the measurement accuracy. Similarly, optical trackers such as the Vicon system [92] also fall into this category. It utilises multiple infra-red cameras to track reflective markers mounted on the moving objects and can reach a millimetre-level resolution.

On the contrary, for systems that do not require prior knowledge about the environment, they only utilise the information collected by the agents' sensors. One

representative systems is visual localisation [29, 126], which is popular in robotics due to the low cost of a camera and its high accuracy. Its bottleneck includes limited computational power and sometimes not robust to the change of environments, e.g. variation of light. Another frequently used sensor in infrastructure-free IPS is laser range finder [81, 82] which outperforms others in terms of accuracy. But it also contains the disadvantage of high energy consumption and monetary cost. Besides the above two kinds of infrastructure-free IPS, others may utilise magnetic field sensors [158] or inertial sensors [167] which often contain a severe tradeoff between manufacturing cost and measuring accuracy.

2.2.2.2 Map Representation

The environment can be described by a map in different forms. Commonly used representations for robot navigation are grid map, landmarks and point clouds.

The grid map representation was firstly proposed in 1987 [34] and has been broadly applied with ranging sensors based navigation systems. The world is modelled as a fine-grained grid over the continuous space of locations in the environment. Each grid contains a certainty value, indicating the possibility of this position being occupied by an object.

Landmark representations [76] are used when the observed features in the environment are sparse and can be identified, e.g. beacons. In this case, the map records the position and uncertainty of each feature, namely a landmark.

Point cloud [165] is a formulation suitable for 3D spaces which contains the positions of all the detected feature points in the environment. It is popular among navigation systems that use 3D LiDAR or camera sensors.

2.2.2.3 Simultaneous Localisation and Mapping

Simultaneous Localisation and Mapping (SLAM) [33] wherein the robot must incrementally build a consistent map of an unknown environment while simultaneously localising itself on this map. Note that both the trajectory of the robot and the map are estimated on-line without the need for any prior knowledge of location and the environment. This makes SLAM occupy an important place in the robotics community as it can provide the means to make a robot truly autonomous.

SLAM is formulated and solved in different forms where the probabilistic SLAM [150] is usually considered as the most broadly accepted formulation. And now, in the theoretical and conceptual level, it can be regarded as a solved problem with various solutions, e.g. extended Kalman filter (EKF-SLAM) [7], Rao-Blackwellized particle

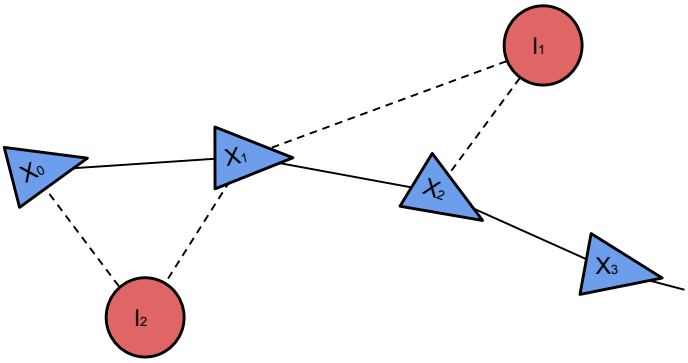


Figure 2.4: A simple instance for GraphSLAM. The blue triangles represent poses of the robot at different time steps and the red circle denotes the different landmarks which construct the map of the environment. Furthermore the back lines are the trajectory of the robot while the dashed lines indicate the observations to landmarks by the robot

filter (FastSLAM) [105] and graph theory (GraphSLAM) [151]. However, substantial problems remain open when considering a practical implementation in large scale problems.

Computational complexity is the first problem. The state-based formulation of the SLAM problem involves the estimation of a joint state composed of a robot pose and the locations of observed stationary landmarks where each landmark denotes a feature point in the map. Therefore, when the robot traverses a long path and an increasing number of landmarks are recorded in the map, the state vector grows linearly and its corresponding covariance matrix grows quadratically. The other problem is with dynamic environments. Real-world environments contain numerous dynamic objects such as human and other movable or changing objects like chairs and doors. These all disobey the static world assumption in most of the SLAM algorithms and can result in conflicts when updating the map through data association. For instance, if a robot comes back to a recorded place in the map after a long journey, it will try to close the loop by matching its current observation with the map to diminish the drift. A non-trivial change that has occurred in this place may lead to the failure of the association, introducing an inconsistent map.

2.2.3 Path-Planning

When the robot is able to self-localise and the goal is also singularised in the map, it is ready to find a path towards the destination. Path-planning algorithms are used to generate a collision free path to the destination with the least costs under a certain

criteria, e.g. time, energy or the length of path. It can be generalised into two different categories, i.e. off-line and on-line, based on whether the complete description of the environment is provided.

In off-line path planning problems, since a full map of the environment is given, searching the optimal path is relatively easier compared to the on-line problem. Lozano-Perz and Wesley initiated this field with ideas such as configuration space, visibility graph and cell decomposition [85, 84] in early years. These methods can be effective in simple environments but are computationally expensive in complex scenarios. Thus some optimisation tools like genetic algorithms and particle swarm optimisation are exploited to generate more efficient solutions. [166, 117].

On-line path planning has wider application for robot navigation as it can tightly cooperate with SLAM algorithms where the robot simultaneously builds the map as it traverses the environment. Artificial potential fields is one of the main approaches [60], together with collision cones [23]. Similar to off-line problems, classic methods for on-line path planning are later improved with more computationally efficient optimisation tools [155, 96, 91].

A more comprehensive review of path planning approaches is available in [118].

2.3 Mapless Robot Navigation

In this section, we explore the application of mapless navigation in indoor environments. As shown in Fig. 2.3, unlike map-based navigation, mapless navigation is carried out with neither the pre-built description of the environment nor the on-line construction of a map. That means the robot perceives the world only with current or recent observations where the historical information (if used) is not associated together as a map. Note that it is still possible for the robot to form the current sensory observation as a local map. Therefore mapless navigation mainly solves the problem of obstacle avoidance, the back bone of robot navigation. Meanwhile, if there is a goal provided by a path planner in its local coordinate system, the system approaches this local destination as well. But if it is not goal orientated, it simply follows the traversable areas.

We review mapless navigation systems with two different sensor modalities in this section, i.e. ranging sensor based and vision based systems. With different sensing approaches, those navigation systems focus on their own specialised problems. Since obstacle detection is straightforward for ranging sensors, robotic researchers explored

them in the early ages and proposed many efficient control principles. While for the visual navigation systems obstacle detection is always regarded as the core difficulty.

2.3.1 Ranging Sensor Based Systems

In ranging sensor based systems, the obstacle detection is relatively easy as the distance between the robot and objects in the environment is available from the raw sensory data.

2.3.1.1 Edge-Detection Based Methods

The approaches based on edge detection [14, 71] were the earliest methods. An algorithm firstly detects two vertical edges of the obstacle with ultrasonic sensors. The line connecting those edges is defined as one of the boundaries of the obstacle. The robot is then steered around the obstacle and reaches the destination.

2.3.1.2 Potential Field Methods

This kind of approach is based on a concept of imaginary force [60]. It comprises the sum of a target-directed attractive force and repulsive forces from obstacles. In [69], Krogh proposed to take the velocity of the robot into consideration in the vicinity of the obstacles. Then, Krogh and Thorpe [70] integrated it with a global path planner and proposed the "generalised Potential Field". The above applications of potential field rely on the assumption that we have description of the environment. But in [4, 19], each ultrasonic sensory input generates a repulsive force. When the sum of them reaches a threshold, the robot stops and turns to the directions suggested by the attractive forces. In this way, the proposed potential field is able to be applied in mapless navigation problems.

The Virtual force field approach [15] combines fields with a local grid map. However, it needs to build an on-line certainty grid map to record the obstacles in the environment during navigation. Vector field histograms [16] were proposed based on virtual force field with the aim of being computationally efficient, robust, and insensitive to misreadings. Its core idea is to use the statistical representation of obstacles, through histogram grids which are suited to inaccurate sensor data, and to accommodate fusion of multiple sensor readings.

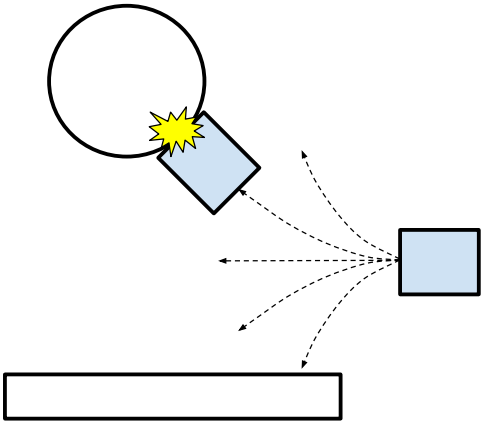


Figure 2.5: This figure illustrates the dynamic window approach. The white circle and rectangles are obstacles while the blue rectangle is the robot and the dashed lines are the sampled trajectories under different control actions. Based on the prediction, the robot will choose to go straightforward to avoid the obstacles on both sides.

2.3.1.3 Dynamic Window Based Methods

The dynamic window proposed by Fox et al. [36] is another pioneering work in obstacle avoidance. As shown in Fig. 2.5, it discretely samples in the robot's control space. For each sample velocity, it performs forward simulation from the robot's current state to predict what would happen if the sampled velocity were applied for some (short) period of time. Then each trajectory resulting from the forward simulation is evaluated, using a metric that incorporates characteristics such as proximity to obstacles, proximity to the goal, proximity to the global path, and speed. The illegal trajectories (those that collide with obstacles) are discarded. Finally it selects the highest-scoring trajectory and calculates the associated velocity to the robot. This method is derived directly from the dynamics of the robot, and is especially designed to deal with constrained velocities and accelerations.

It is improved in [18] which combines planning and real-time obstacle avoidance algorithms to generate motions for mobile robots that achieve the robot's task, while securely navigating in an unknown environment. Based on this, Damas and Santos-Victor [27] took the velocity of the obstacles into consideration and proposed an approach that is effective in dynamic environments.

2.3.2 Vision Based Systems

Since there is no explicit geometric information in the sensory observations, using vision is more difficult than applying ranging sensors for robot navigation. The core

Figure 2.6: Output of optical flow. The green arrows display the direction of moving pixels while the dots represent the static ones.

problem is how to detect the obstacles or traversable areas from the images. In this section we focus on reviewing the existing approaches with traditional computer vision techniques. The recent approaches via deep learning will be mentioned in later chapters.

2.3.2.1 Optical Flow Based Approaches

Optical flow was firstly introduced by Gibson [40] and became a fundamental concept in computer vision. As shown in Fig 2.6, it tracks the relative motion of pixels in a sequence of images based on two assumptions. The first is that pixel intensities of an object do not change between consecutive frames and the second is neighbouring pixels have similar motion. These two assumptions enable us to match pixels in two frames. Then the direction and magnitude of relative translation or rotation between paired pixels in two frames are calculated to represent the relative velocity of each feature point w.r.t. the camera. Note that the optical flow can also be estimated sparsely by tracking the detected feature points at each frame without considering above assumptions. Based on the intuition that the near obstacles move fast while the distant ones move relatively slow, it enables the robot to navigate in some simple environments like a corridor.

Santos-Victor and Sandini [125] proposed a method by mimicking the visual cortex of bees based on optical flow with two divergent stereo cameras which have no overlapping fields of views. The core idea is to compare the different velocity of the average optical flows calculated from the cameras on two sides which is then used to formulate the error signal for a Proportional integral Derivative (PID) controller [5]

to generate the actions.

In [21], the optical flow divergence is extracted from the sequential observations of a monocular wide-field camera. Based on this, the time of colliding with the objects in each direction are computed and the control signal is generated correspondingly.

Talukder et al. investigated the detection of dynamic objects based on the optical flow extracted from the stereo cameras [147]. The main idea is that the moving objects cause a discontinuity in optical flow and changes in magnitude w.r.t. the background pixels. This work is later improved by combining the optical flow with depth from stereo cameras [148].

In [63], the floor is segmented out by computing the plane normal from the image sequence. A dense optical flow is firstly obtained from images and is then used to calculate the normal for the sub-regions of the image.

2.3.2.2 Appearance Matching-based Approaches

Another kind of visual approach has a two-phase solution. Firstly it memorises the environment by recording the observed images or the extracted prominent features in the training phase. Then in the navigation phase, by matching the current observation with "memories" the robot can localise itself and generate correct actions to travel through the same environment. We term this kind of approach as appearance matching.

In [89], Matsumoto e.t al. stored the image sequence and the directional relations between two consecutive images along the route in the training phase. While in the testing phase, the robot detects its position, steering angle by comparing the current observation with recorded images sequence. It is extended by using an omnidirectional camera in [88] for a better matching accuracy and robustness.

In [179], the recorded image is encoded with a multi-dimensional histogram to describe the appearance features e.g. colours, edge density, gradient magnitude and textures. It largely reduces the memory consumption and enhances the retrieval efficiency.

Remazeilles et al. [119] proposed to model the environment as an image graph base in the off-line process. Then in each navigation task, relevant images are retrieved to form a desired route for the robot. Finally the 2d positions of the points matched between the images in sequence are used to generate control signals based on a potential field approach.

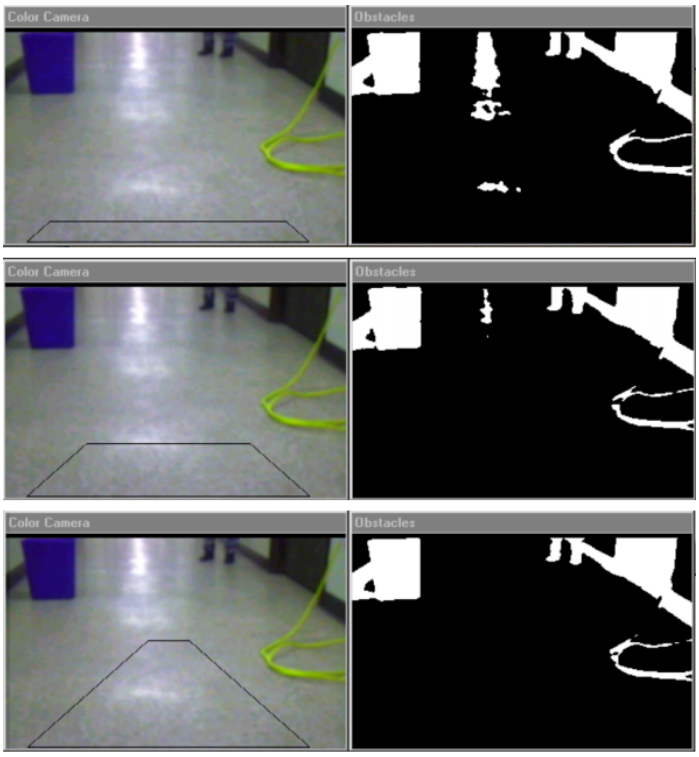


Figure 2.7: The ground segmentation in [154]. Images in the left column are the inputs where the area within black polygon is the proposed reference area for learning ground texture. The right column illustrates the segmentation results.

2.3.2.3 Reactive Visual Approaches

This class of methods are usually focused on detecting the movable areas based on some qualitative characteristics of the image and can avoid the computation of accurate numeric values such as distance and coordinates of points.

Shakey [111] is one of the pioneer systems in this field. It navigates in an artificial environment where the obstacles are differentiated from the textureless floor based on an edge detector. A similar method also appears in [54] where the robot is able to navigate in a real indoor environment.

The above approach suffers from issues with environments with textured floors. This is addressed by Ulrich and Nourbakhsh et al. in [154]. It is solved by learning the colour features of the ground plane within a reference area proposed in the image during a training phase as shown in Fig. 2.7. Thus the robot is able to classify pixels in the images between floor and obstacles during the navigation.

2.3.2.4 Feature Tracking-based Approaches

Tracking salient features in computer vision has contributed significantly to the field of visual navigation. The first step is to find feature points in a pair of images with feature detectors, e.g. edge [22, 47], corner [47, 120] or blob [9, 95]. Then features in two images are matched correspondingly. When there are enough matched features, the homography matrix is estimated and finally the relative rotation and translation between two images are calculated.

H-based Tracker is proposed in [113] which tracks the corners in the indoor environments. The tracker employs planar homographies and is initialised with 5 corner points on the ground. This work is extended in [78] to detect the obstacles by recognising the feature points above the ground planar. A similar approach is also mentioned in [83] where the obstacles are detected by searching for the feature points that cannot be registered on the ground plane with the assumption of locally planar ground. Additionally, in [180, 28], homography matrix is calculated and utilised to find the ground plane based on previously tracked image corners with Harris corner detector [47].

2.4 Discussion

Recent research has commenced the journey towards deploying DRL based navigation systems on physical robots and some state-of-the-art solutions have been proposed to solve the core challenges that mentioned in Sec. 1.2. However, they only address those problems under limited situations and a practical navigation system which is able to be generalised to different scenarios and deployed on various physical robots has still not been concretely established.

2.4.1 From Simulations to the Real World

Domain randomisation is one possible solution to this problem [152, 122]. The core idea behind this technique is to force the policy to learn with a randomised visual appearance of the environment. It is simple, but the eventual performance heavily depends on correctly selecting the distribution of random variables [108]. The dilemma here is that a highly variant randomisation complicates the task and prolongs training whilst a narrow distribution may not reach the target realistic domain.

Another branch bridges the "reality gap" by adapting visual observations between synthetic and real domains. Bousmalis [17] and Stein [139] both suggest to render

more realistic visual observations with generative models (virtual to real). These are more straightforward approaches than domain randomisation but they result in an additional computational burdens as they have to generate realistic images in parallel with optimising the DRL policy during training. This problem is tackled by preceding the adaptation operation in the real-world inference stage with an inverse adaptation direction (real to virtual) as presented in [175, 57]. However another problem which largely discounts the practicality of this category of methods is that the trained adaptation network is tailored to a specific virtual/real environment pair and lacks generalisation ability to environments with apparent visual change. This becomes even more apparent in indoor environments which have a huge diversity in visual appearance.

In Chapter 3 we are the first to propose a more practical and task-oriented type of solution. It minimises the discrepancy between synthetic and realistic data by projecting the visual observations to another modality, i.e. depth. It is easy to implement for both training and testing. The ground truth depth information is provided in most robotic simulators, e.g. *ROS Stage*[1], *ROS Gazebo*[2] and *Carla* [31]. Meanwhile, when testing in the real world, it can be estimated from visual observations with an off-the-shelf depth prediction network [75, 1] which is usually generalised to various indoor scenarios. Another reason for using depth information as the bridge is that the directly provided distances between the robot and objects in the environment facilitates easier learning of obstacle avoidance.

2.4.2 Speeding up Training in Simulations

Dealing with the inefficient sample usage of DRL has been a long-established topic in the machine learning community. To maintain the generalisation ability to various kinds of tasks in machine learning area, many theoretical [162, 127] and systematic [101] methods are introduced. Those methods improve the training efficiency by optimising the learning mechanism itself but do not consider any extra information. However, different from the generalised machine learning area, there are many conventional methods for robotic tasks such as mapless navigation that can be utilised as the prior knowledge for learning a policy.

Learning from demonstrations (or imitation learning) [32] leverages both the interactive learning pattern in RL for gaining the robustness and the efficient learning

[1]http://wiki.ros.org/stage_ros
[2]http://wiki.ros.org/gazebo_ros_pkgs

signal in supervised learning by forcing the network to mimic the behaviour of an expert. However, it is limited to expert controllers which themselves are usually difficult to establish in robotics.

Another method is to embed conventional controllers into the navigation system as in [170]. It only learns a planning strategy with reinforcement learning to assign different reference signals to a low level controller in different situations. Although this method effectively accelerates the learning process in the early stage, its final performance is bounded by the low level heuristic controller.

Therefore, in Chapter 4, we introduce a novel framework where all classes of heuristic controllers, no matter expert or simple, are only regarded as guidance to reduce the variance in the learning process of an off-policy DRL approach, e.g. DDPG. Under this framework the underlying DRL approach can learn rapidly from the guiding controllers in the early stage and automatically discard them when its policy has an upper hand and then continues to learn with self-exploration. The utilised heuristic controllers can be naive or expert as the performance of this "soft" guidance only affects the rate of acceleration brought to the learning process but does not restrict the learning of the underlying DRL approach.

2.4.3 Learning Global Mapless Navigation

Generalising the learnt navigation policy to unseen environments is still an unsolved problem in DRL, hence its application to global mapless navigation is also of great difficulty. Many pioneering works [99, 181, 98] do not consider this problem, as their shortest path policy is achieved by memorising all the environmental information with the deep neural network. Some others must be provided with an metric or graphic map to transfer the policy to a new environment [46, 20, 157].

For learning a transferable mapless navigation policy, solutions similar to conventional visual teach and repeat (VT&R) [37] were introduced in [143, 72, 53] recently. They only require a image sequence, that may or may not be paired with actions, as guidance which is much easier to acquire in contrast with a map. However, a constraint is that the movement between each adjacent piece of guidance must be incremental, otherwise the navigation system may easily get lost amongst the guidance. This results in the requirement for a high density of guidance information, impacting communication and memory, especially in a multi-agent setting. Additionally, system autonomy and robustness to altered or dynamic environments is also limited due to its strong reliability on the dense guidance. Interestingly, using a panoramic camera as mentioned in [53] can enhance the robustness to minor changes of the environment

as a wider field of view brings richer visual information and enables the robot to carry out the homing task[3].

A much more sparse and efficient guidance, i.e. natural language, has also been used to attempt to learn visual mapless navigation [159, 51] but only in simulation. The extreme sparsity, ambiguity of natural language and the difficulty of perceiving multi-modality observation (vision and language) for robots prevents natural language from being an accurate and reliable guidance for a real time robot control problem.

To establish a visual mapless navigation system whose navigation experience can be easily and efficiently generalised to unknown scenarios, deployed on physical robots and shared among multiple agents, we proposed a novel navigation system in Chapter 5. Its navigation policy can be transferred to new scenarios with the minimal description of the environment, i.e. a few snapshots of the environment paired with directional commands. Such a guidance can be easily extracted from one agent's experience and shared to others. The proposed system can also be straightforwardly deployed on real robots due to a two-level hierarchy where each layer learns a robust sub-policy with data from different domains.

Chapter 3

Visual Obstacle Avoidance: From Simulation to Real World

This chapter aims at addressing the challenging problem of bridging the reality gap from simulated to real environments and focuses on the particular task of monocular visual obstacle avoidance. Obstacle avoidance is a fundamental ability in robot navigation which is relatively simple for a DRL policy to learn. The main issue tackled here is whether the robot can learn to navigate around obstacles when only supplied with monocular visual observations from a simulator as training data. Our aim is to examine whether our proposed approach can transfer the perception of the learnt policy accurately from simulation to the real world, without being trained with real world data. Extensive experiments verify the effectiveness of the transferred obstacle avoidance policy and its robustness to various real world scenarios.

3.1 Introduction

When mobile robots operate in the real world, subject to ever varying conditions, one of the fundamental capabilities they need is to be able to avoid obstacles. Although it can be partially solved with a global planner to propose a collision free path according to the map, it is more important to accomplish this function based on the on-board sensors of the robot due to dynamic objects as well as changes in the environment.

As a long established problem in robotics, obstacle avoidance is typically tackled by approaches based on ranging sensors [35], e.g. laser scanner and sonar sensors. They can directly measure the relative position between the robot and obstacles which largely simplifies the problem. However, ranging sensors only capture limited information and some of them are expensive or are too heavy and/or power consuming for a particular platform e.g. a UAV. Monocular cameras on the other hand, provide

rich information about the robot's operating environment, are low-cost, lightweight and applicable for a wide range of platforms. Unfortunately, when perception of range is obtained by monocular vision, i.e., RGB imagery, the obstacle avoidance problem becomes surprisingly difficult. This is because the 3-D world is flattened into a 2-D image plane, eliminating direct correspondence between pixels and distances.

Deep Reinforcement Learning (DRL) has recently achieved, or exceeded, human-level performance in complex tasks based on high dimensional visual observations such as playing Atari games [104]. Its direct mapping from raw appearance inputs to action outputs relieves the system from manually designed feature extraction on images which can be unstable in many situations e.g. when changing lighting conditions or facing a textureless surface. In addition, DRL learns from the interaction with the environment and optimises its policy according to the experienced rewards, making it possible to eventually outperform heuristic approaches [138, 154, 13]. Although it possesses the above advantages, a large restriction of DRL based approaches is the demand for interactive learning samples, making training in the real world almost impossible, especially for mobile robots. It is time and labour expensive and can also incur safety problems. Therefore DRL currently only has remarkable performance in virtual environments where training is affordable. Transferring the learnt policy from simulation to the real world remains an open problem and a key limitation for applying DRL to real robot problems.

In this chapter, we propose a deep neural network based obstacle avoidance system, utilising an on-board monocular camera for environment perception. The training is purely carried out in simulation and can be eventually generalised to various real-world scenarios.

3.2 Related Work

A standard framework to solve the obstacle avoidance problem consists of two steps, firstly making use of visual information to infer traversable spaces and obstacles, and secondly applying conventional path planning strategies. Recovering visual geometry is a common approach to detecting obstacles, e.g. through optical flow [138, 90], detection of vanishing points [13] and even visual SLAM [109]. Segmenting traversable areas, such as floors, based on visual appearance [154] is also a popular method. Once the surroundings are understood, various conventional path planners can then be employed to drive robots along traversable routes [36]. Although the described methods are able to decouple planning from visual information and hence benefit

from conventional path planners, they usually require a large number of parameters which need to be manually tuned to plan feasible paths. It is also challenging for them to automatically adapt to the new operating areas.

Supervised deep learning based path planning which learns how to avoid collision is becoming increasingly popular. In particular, with the recent advances of deep learning, several end-to-end supervised learning approaches are proposed to directly predict control policies from raw images [61, 42, 144] without following the previous two-step framework. Therefore, they can avoid complex modeling and parameter tuning of conventional path planners. Convolutional Neural Networks (CNNs), for example, are trained to enable flying robots to navigate in complex forest environments [42]. However, due to their supervised nature, these approaches need manual labelling which is time-consuming and labour-intensive to obtain.

Self-supervised learning can be used to automatically generate labels for path planners with the aid of additional feedback. For instance, in [172] an algorithm is designed to label trajectory classes for a CNN based model by using 3D cloud points. [38] proposes to train a drone controller by predicting when it crashes. Although self-supervised learning is a feasible approach to benefiting from large datasets without human supervision, the learnt policy is essentially bounded by the label generating strategy.

Reinforcement learning explores policies though trials, and has been applied to vision based obstacle avoidance in [94]. However, the raw image is encoded as several levels of depth to predict a suitable control strategy. Deep reinforcement learning (DRL) has recently been shown to achieve superhuman performance on games by fully exploring raw images [104]. Since DLR usually utilises a much weaker learning signal compared with supervised learning, it requires a much larger training dataset. This makes it difficult to directly use DLR for robotic applications in reality. Therefore, simulations which have a failure-and-recovery mechanism are usually used for training rather than real world exploration [100]. The trained networks can then be transferred to real robots. Although this has been successful when using laser scanners [146] and depth images [145], it is significantly more difficult for vision based models [38]. Recently the authors in [123] propose to train a network as a collision predictor entirely in a 3D CAD model simulator and highly randomise the rendering settings, approximately regarding the real world as a subset of training data. Although their model can be extended into the real world, it requires substantial computational resources to generate the huge dataset and train on it and lacks a systematic method to tune the randomisation parameter.

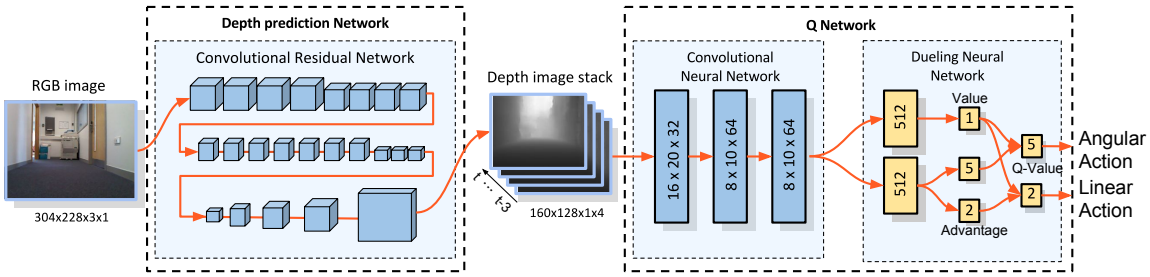


Figure 3.1: Network architecture of monocular image based obstacle avoidance through deep reinforcement learning. A fully convolutional neural network is firstly constructed to predict depth from a raw RGB image. It is then followed by a deep Q network which consists of a convolutional network and a dueling network to predict the Q-value of angular actions and linear actions in parallel.

3.3 Problem Formulation

We formulate the monocular vision based obstacle avoidance problem as a decision making process (MDP) which has been briefly introduced in Chapter 2. Here we model the rgb camera image x_t at time $t \in [0, T]$ as the agent observation and the desired control command $a_t = (a_t^v, a_t^\omega) \in \mathscr{A}$ as its action, where a_t^v and a_t^ω are the linear and anglar velocity respectively. Then, each time the agent chooses an action a_t according to the observation x_t it obtains a reward r_t. Again, the aim of the MDP is to maximise the accumulative future reward $R_t = \sum_{\tau=t}^{T} \gamma^{\tau-t} r_\tau$, where γ is the discount factor.

3.4 Model and Training Details

The system is modelled with a deep neural network as shown in Fig. 3.1. It consists of two modules, a depth prediction network and a Q-value estimation network.

Depth Estimation The depth estimation network is implemented by a fully convolutional residual network (FCRN) in [75], predicting depth information from a single RGB image. It firstly extracts depth information from the RGB image and converts it into feature vectors and then reconstructs the depth image accordingly.

The FCRN is initialised with a pre-trained model provided at `https://github.com/iro-cp/FCRN-DepthPrediction` which is optimised based on the NYU dataset [110]. Next, it is retrained with 10k images collected through a Microsoft Kinect[1] camera. The learning rate is set as 10^{-5}.

[1]https://en.wikipedia.org/wiki/Kinect

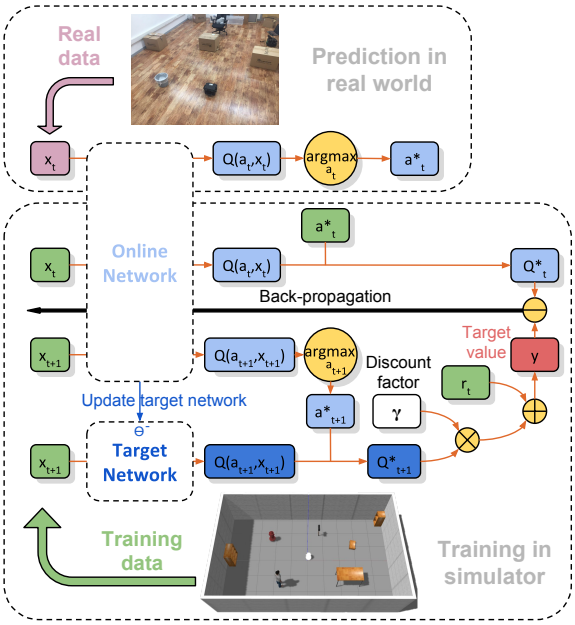


Figure 3.2: Given a batch of training data, including current state x_t, action a, reward r, and resulting state x_{t+1}, the training procedure of D3QN is shown in the figure. $\oplus$, $\ominus$ and $\otimes$ are element-wise operations for addition, subtraction and multiplication.

D3QN The Q network is implemented as the DQN combined with the double network and dueling architecture introduced in Chapter 2.1.6.1. As shown in Fig. 3.1, it consists of three fully convolutional layers and fully connected layers. The former are used to extract useful features for obstacle avoidance from depth images. It is worth noting that the input is four consecutive depth images in order to alleviate the problem of partial observability. All the dense layers are divided into two branches according to the dueling network architecture and each branch estimates the value and the advantage respectively. With this structure, the learning efficiency and final performance is improved by a large margin which is attributed to the decomposition of predicted Q values. The value represents the maximum of all possible Q values. Therefore the advantages, which are calculated by subtracting the value from Q values, are scaled around 0. It is more efficient for deep neural networks to learn through back propagation, but keep the relative relationship between all Q values unchanged. Hence, it boosts the learning of policy as the network only needs to learn accurate advantages to know which action to choose.

This network is optimised according to the double network mechanism which is illustrated in Fig. 3.2. The target network shares the exact same architecture as the online network. However, unlike the online network which updates weights by back-propagation at every training step, the weights of the target network are

fixed over a short period and then copied from the online network. Based on this two-network framework, [156] claim that the online network should be used to select actions whilst the target network should be used to solve the problem of overoptimistic value estimation [48]. More specifically, the resulting state x_{t+1} is employed by both the online and target networks to compute the optimal value Q'^* for time $t+1$. Then, with the discount factor γ and current reward r_t, the target value y at t is obtained. Finally, the error is calculated by subtracting the target value from the optimal value Q^* predicted by the online network, given current state x, and is then back-propagated to update the weights.

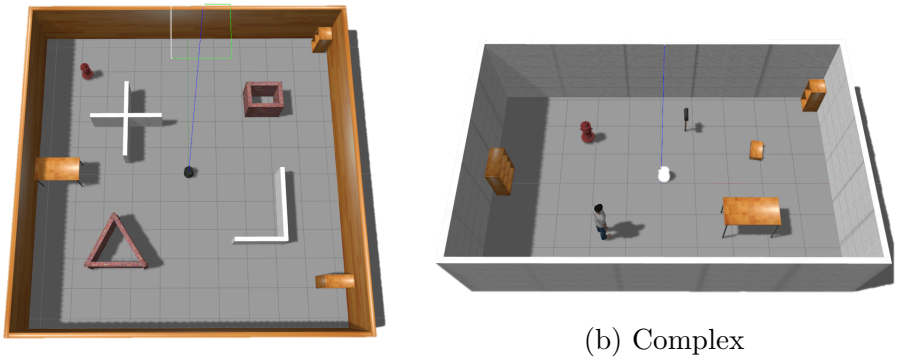
Reward Shaping The design of the reward function plays a crucial role in learning a valid policy. A more instructive reward can largely facilitate the training but may introduce human bias and cause the system to deviate from the optimal policy. Therefore we propose the following reward function:

$$r_t = \begin{cases} R_{crash}, & \text{if robot crashes} \\ v * \cos(\omega_t)\delta t, & \text{otherwise} \end{cases} \tag{3.1}$$

where v and ω are local linear and angular velocity respectively and δt is the time for each training loop which is set to 0.2 second. If a collision is detected, the episode terminates immediately with an additional punishment of $R_{crash} = -10$. Otherwise, the episode lasts until it reaches the maximum number of steps (500 steps in our experiments) and terminates with no punishment.

This reward function is different from the broadly utilised sparse reward function where the agent only receives the feedback until the termination of an episode. It provides detailed preference on robot's behaviour, encouraging the robot run as fast as possible and penalising it for rotating on one spot, which effectively reduces the amount of learning samples for the agent to converge and enhances the learning efficiency. Although this human preference in reward function may deviate the robot from the optimal policy which can be learned from a sparse reward function, it results in acceptable policy performance in the obstacle avoidance task through our experiments.

The D3QN model is trained in two simulation environments, simple and complex ones, both of which are built in the Gazebo simulator as shown in Fig. 3.3. The D3QN model is firstly trained in the simple environment before being further trained in the complex scenario. The trained model in the simulator is directly tested in several different real world scenarios. The linear velocity is set to be 0.4 or 0.2 m/s, whilst



(a) Simple

(b) Complex

Figure 3.3: Two simulation worlds in Gazebo used for training.

the angular velocity is $\frac{\pi}{6}$, $\frac{\pi}{12}$, 0, $-\frac{\pi}{12}$ or $-\frac{\pi}{6}$ rad/s, producing ten different behaviours. Throughout our experiments, a NVIDIA TitanX GPU is used for training whilst a laptop equipped with a NVIDIA GTX 860 GPU is used for real-time inference and testing in reality. The learning rate is set to 10^{-5} in an Adam optimiser [64].

3.5 Experiments

We carry out extensive simulated and real-world experiments to evaluate the proposed obstacle avoidance system. At each training episode the robot is spawned at the centre of the simulated environment, heading to a random direction. In our experiments, we set the control frequency as 5 Hz and the maximum of training steps as 10k which take approximately 5.5 hours without simulation acceleration. A Turtlebot 2[2] robot is used to test the control strategy in real-time which mounts a kinect sensor. However, only the rgb camera is utilised for real-world testing.

3.5.1 Training Efficiency with Different Models

To analyse the training efficiency and performance of the proposed D3QN model and the advantage of introducing dueling and double techniques for obstacle avoidance, two competing models, deep double Q network (DDQN) and DQN are compared. As shown in Fig. 3.4, the D3QN model outperforms the other two both in terms of training speed and reward performance. Unlike DQN whose average reward only reaches 10, networks with a double network structure learn policies with higher rewards. This may be because, the problem of overestimating the Q value hinders DQN to learn

[2]https://www.turtlebot.com/turtlebot2/

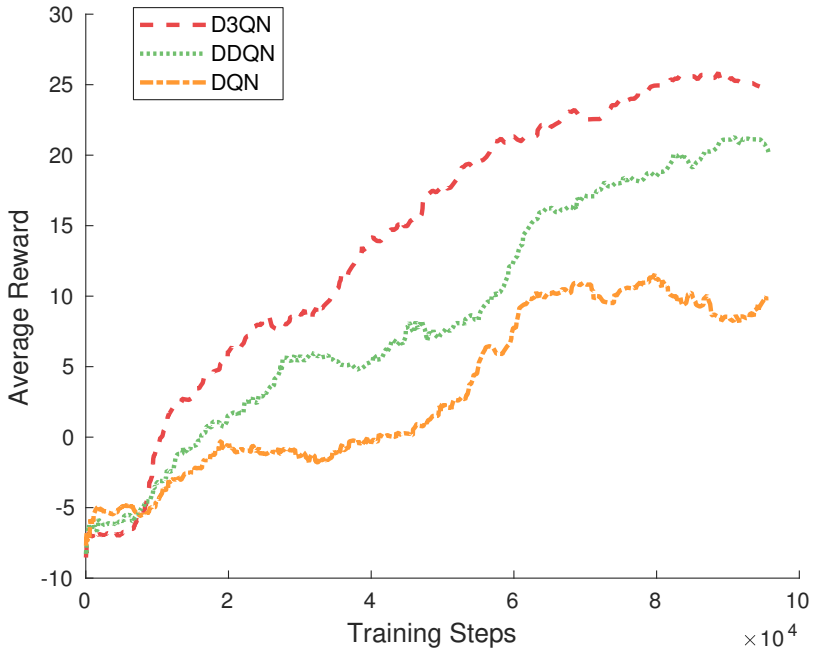


Figure 3.4: Smoothed learning curves of the three models with average rewards acquired by robot.

a robust policy from avoiding obstacles. With the dueling network learning the importance of each action in different scenarios, the D3QN model further has another advantage over DDQN. Not only does it demonstrate the superior performance on obstacle avoidance but also points to an interesting direction of applying D3QN to other robotic applications.

3.5.2 Real World Tests

Several experiments are conducted to directly test the trained models in the real world.

3.5.2.1 Action Prediction from Static Images

Firstly, we examine whether for arbitrary, complex scenarios, the network is able to predict a reasonable action that will avoid obstacles. As shown in Fig. 3.5, a number of RGB images taken by a hand-held camera in a variety of environments including library, museum, attic and bedroom are used to predict actions. The bars in the figure indicate the Q value of each action: the first two bars are for linear velocity 0.2m/s and 0.4m/s, whilst the others are for the five angular velocity settings: $\frac{\pi}{6}$rad/s, $\frac{\pi}{12}$rad/s, 0rad/s, $-\frac{\pi}{12}$rad/s and $-\frac{\pi}{6}$rad/s. Note that these scenarios are more complicated than the simulation ones used for training and none of them has been "seen" by the model

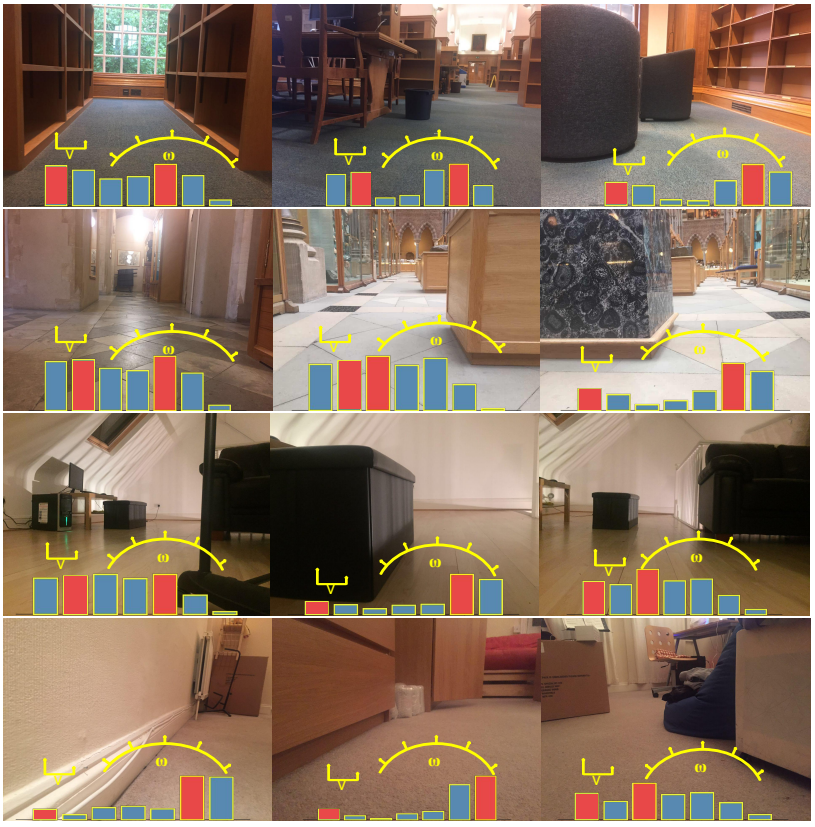
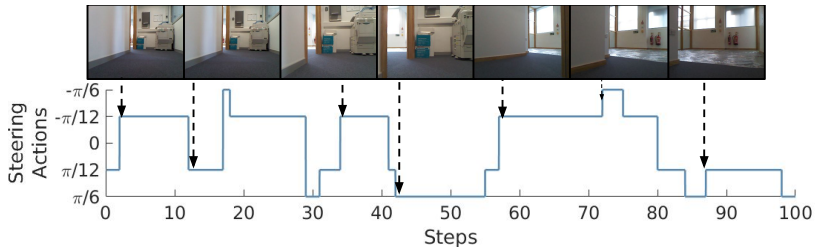
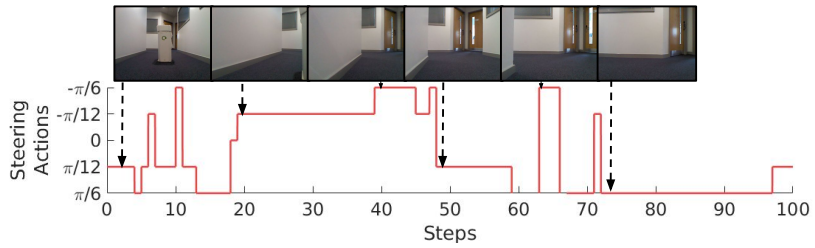


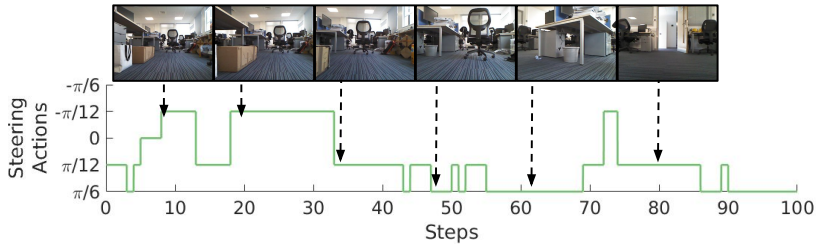
Figure 3.5: Experiments in different indoor environments, e.g. library, museum, attic and bedroom (from top to bottom). The underlying bars demonstrate the Q value for each linear and angular action predicted by the network, where the red ones indicate the actions greedily selected by the network. Notice that the first two are for linear speed actions whilst the rest are for steering actions.



(a) Passing a doorway.



(b) Driving through a corridor.



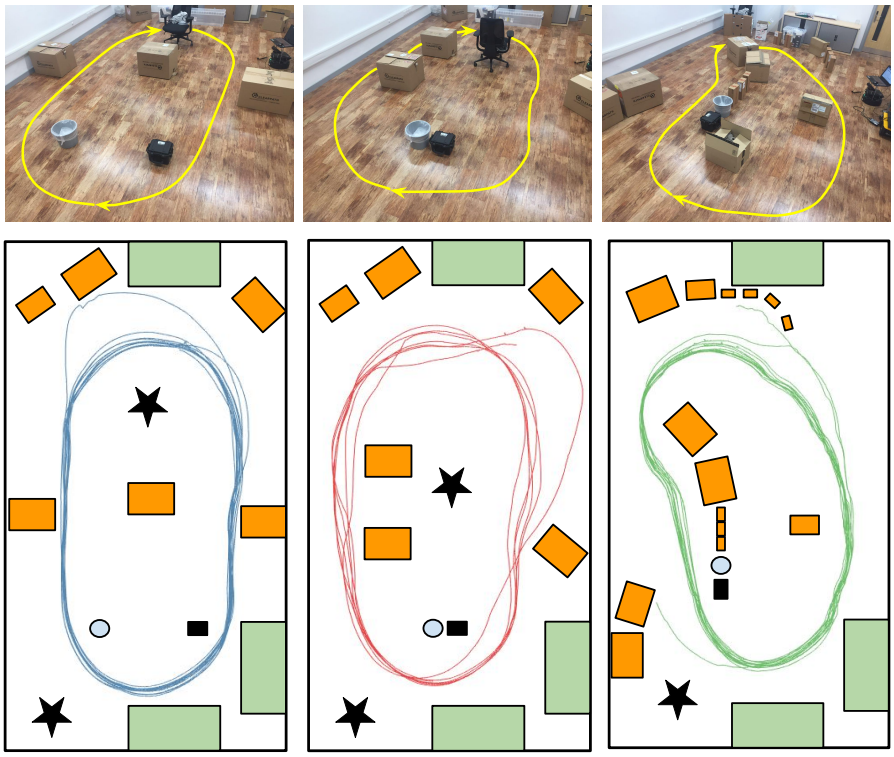
(c) In an office room.

Figure 3.6: Real world tests in three different scenarios. The curve below the image streams shows the steering actions selected by robots at each step.

before. Nevertheless the trained D3QN model is capable of producing reasonable actions to drive the robot according to the estimated Q values.

3.5.2.2 Tests in Three Different Scenarios

The trained D3QN model is tested for short-term (20s) navigation in three different scenarios including a doorway, a corridor and an office. The steering actions and some sample images of the three tests are given in Fig. 3.6. Specifically, Fig. 3.6a shows the procedure of the robot passing the doorway. Although the steering action of the robot is a little bit unstable when approaching an unseen obstacle (printer), it can still pass the doorway successfully. For the corridor case, an obstacle is placed in the middle of the corridor. As shown in Fig. 3.6b, the robot can navigate smoothly through the narrow space between the obstacle and the wall. Similarly, the robot can be controlled safely in an office room which is a more complicated environment with many previously unseen objects in the simulator. The experiments validate that the



(a) (b) (c)

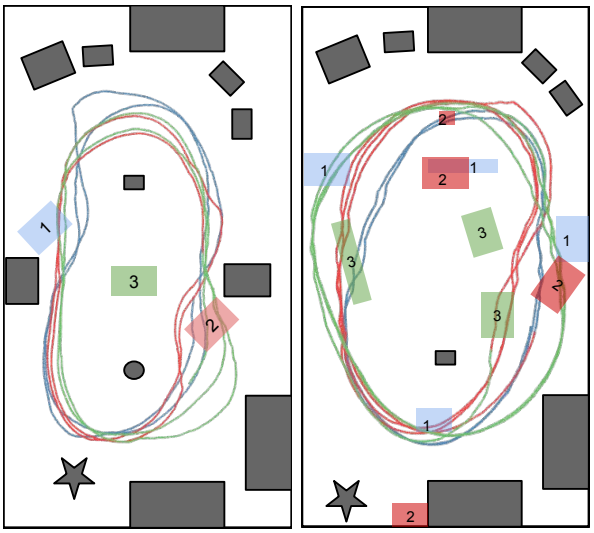
Figure 3.7: Real world tests in a room with different number and placement of obstacles. Rectangles show boxes whilst stars and circles are chairs and trash cans respectively.

trained D3QN model enables the robot to avoid obstacles by only using a monocular camera in different real environments, benefiting only from knowledge learnt in virtual environments.

3.5.2.3 Tests in a Cluttered Environment

Several long-term experiments are conducted in a cluttered room to further test the performance with dynamic layouts and objects. A Vicon system is used to record the ground truth poses of the robot.

Fig. 3.7 records the trajectories of the robot when it is operating around many obstacles. Green rectangles denote fixed furniture whilst the orange ones are movable boxes. Other obstacles include two chairs (stars), a trashcan (circle) and a small suitcase (back rectangle). From the results we can see that the robot usually chooses to go along a similar path. This is because after the Q value of each state and action pair is predicted by the network, the action is selected by a greedy policy, resulting in a fixed policy for all states. Since the reward function defined in the training phase



(a) Simple (b) Complex

Figure 3.8: Real world test in dynamic environments. The obstacles in grey are fixed whilst coloured ones indicate the motion of movable obstacles at each time. The number on the obstacles indicates the changing sequence.

prefers following a straight line to turning, the robot navigates along a loop with the smallest curvature to maintain a maximum linear speed.

Fig. 3.8 presents the results when the robot is tested on two dynamic environments. In this experiment we dynamically moved objects in the environment and verified that the robot was able to avoid them by using a monocular camera, which further verifies the effectiveness of the proposed method. The video of another test is available at `https://youtu.be/qNIVgG4RUDM`.

3.5.2.4 Comparison with the Baseline Approach

In this experiment the proposed approach (**Ours**) is compared with an optic-flow based baseline method [138] (**OF**). Note that the baseline method is implemented based on the git repository[3]. Tests are carried out in two different environments, a cluttered environment as shown in Fig. 3.7 and a corridor environment shown in Fig. 3.6b. In each environment 5 independent runs are carried out with an one-minute time limit for each run. Meanwhile the number of collisions (NC) and the distance before the first collision (Dsit) are selected as performance metrics.

Table 3.1 shows that our method significantly outperforms **OF** particularly in the corridor environment. As shown in Fig. 3.9, this is because the textureless objects

[3]https://github.com/zainmehdi/Optical-Flow-based-Obstacle-Avoidance

Table 3.1: Compare an optic-flow based baseline approach (**OF**) with **Ours** in two different environments respectively. Each test consists of 5 independent runs with a one-minute time limit. The used metrics are the number of collisions (NC) and the distance before the first collision or timeout (Dist).

	Cluttered		Corridor	
	NC	Dist(m)	NC	Dist(m)
OF	2.6	2.67	5.4	1.32
Ours	**0.6**	**6.28**	**0.2**	**10.35**

Figure 3.9: The left image is the feature detection result in the cluttered environment and the remaining two are in the corridor environment.

(e.g. walls) in the environment hinder the feature detection which is fundamental for the feature-based estimation of the optic-flow between two frames. This is a common weakness of the conventional visual geometry based approaches.

3.6 Conclusion

In this chapter, we proposed a novel and practical solution to successfully transfer the learnt DRL policy from simulation to the real world, for the task of monocular visual obstacle avoidance. Our novel insight used to tackle the reality gap is to project from the visual appearance domain of the scene to the synthesised depth domain. It is simple to implement and can facilitate the learning of avoiding visible obstacles.

Extensive experiments show that the network can be trained solely in the simulator and then directly deployed on real robots only by using monocular vision. In addition, for the first time, the D3QN architecture is applied for a real-time mobile robot control task which demonstrates boosted learning efficiency and robustness to noisy depth observations.

Chapter 4

Local Navigation: Speeding up Training with Guidance

Having explored the problem of bridging the reality gap on the task of obstacle avoidance, we are now in a position to investigate the second challenge. The aim is to speed up training in the simulation for learning a more complex task, i.e. local navigation. Local navigation requires the robot to 1) reach a specified destination in the local coordinate frame and 2) avoid obstacles on its way. Since either of the above goals can be partially solved by a simple conventional controller, in this chapter, a novel framework is proposed to leverage these simple controllers to guide and boost the training of an off-policy DRL approach, i.e. DDPG. The proposed framework is implemented with two algorithms respectively, AsDDPG for the deterministic guidance and SGuidance for a stochastic variation. The experimental results show that both algorithms effectively boost the training of DDPG especially when using a sparse reward function. We further note that SGuidance outperforms the deterministic version, reaching near-oracle performance.

4.1 Introduction

Local navigation, where a specific destination is defined in the local coordinate frame of the robot, is a more complicated task in robot navigation compared with obstacle avoidance as it requires a more nuanced control policy that can simultaneously approach the destination and bypass all the obstacles.

Deep Deterministic Policy Gradient (DDPG) [79] is an actor-critic algorithm that is suitable for such a continuous control task in principle, but in practice the cost of exploration in complex navigation environments can be prohibitive. Since an agent must stochastically explore a long sequence of states during each training episode,

high variance becomes the main bottleneck that hinders DDPG from learning effective DRL models. In order to mitigate this issue, conventional architectures generally require a huge number of learning samples, resulting in high computational and environmental costs.

In the machine learning community, researchers mostly focus on algorithmic techniques for accelerating the training of DRL, such as parallel training [101] and data efficiency [127]. However, these algorithms do not consider the context of a specific task, which can be valuable for training. The local navigation task, in our case, does have existing solutions which could benefit the training of DRL. Although these solutions may not be optimal, they still outperform a random exploration policy in most cases. How to fully exploit and benefit from prior approaches and tightly combine them with DRL to accelerate training is an important, yet open, topic.

In this chapter, we propose a new framework that allows an agent to switch between high variance controllers (e.g. DDPG), and low variance controllers (e.g. simple deterministic controllers), effectively allowing the DDPG component to be quickly bootstrapped instead of starting from completely random moves.

Furthermore we have proposed two different approaches of learning the switch policy. One is the Assisted DDPG (AsDDPG) which uses DQN to deterministically use the controller with the highest Q value. The other is the Stochastic Guidance (SGuidance) that learns a probabilistic distribution with REINFORCE algorithm to sample the action from different controller accordingly.

4.2 Related Work

Many works have applied DRL on robotic problems, e.g. navigation [146, 181, 124, 168, 153, 174, 173] and manipulation [45, 171]. Since most of the robotics problems involve continuous control, policy based approaches such as policy gradient [142] or actor-critic method [44], e.g. DDPG [80], are widely used. However, due to the high complexity of the environment, they usually require a large number of learning samples to cope with the high variance problem. Introducing inductive bias [52] is a common approach for alleviating the issue. [127] assigns higher weights to the data where the model has less confidence to improve the efficiency of sample usage. [55] leverages the concept of information gain when exploring new policies. There are other approaches that accelerate training without additional bias such as parallel training [101], where the variance is suppressed by simultaneously collecting multiple diverse learning episodes. In [162], an advanced network architecture is proposed

to factorise the estimation of Q-value into value and advantages. Unlike the above approaches where bias is tightly merged into the models, our framework incorporates extra knowledge as stochastic guidance without imposing any change to the underlying approach.

Yang et al. [173] proposed a hierarchical reinforcement learning based method. It learns basic skills with multiple independent actors and finally samples actions from the one with the highest Q-value estimation which is similar to the deterministic AsDDPG implementation. However our goal is only to assist the training of DDPG and eventually discard the other controllers.

Thompson Sampling [62] shares with SGuidance the idea of learning to switch among different controllers. The difference is that, instead of explicitly calculating the posterior for updating in Thompson Sampling, our framework directly employs neural networks to construct the latent distributions which are trained jointly with the DDPG component by backpropagation. The advantage is that the switch function can be easily built and conditioned on all of the sensor inputs so that it chooses different controllers according to different contexts/conditions. In addition, the target of our framework is more focused on training a better DDPG component, which is able to benefit from the low-variance gradient estimator due to the better samples generated by the stochastic switch.

In [77], Leonetti et al. introduced another approach to incorporate external information into the training of DRL approaches. It utilises a global planer to constrain the available actions for DRL based on a map of the environment, which however is not compatible with our mapless navigation task. Additionally, in our framework, DDPG can explore the full action space by itself alongside the guidance during the entire training process and can eventually work independently.

Xie et al. [170] proposed to learn a feedback planning policy with a DRL approach. The planner learns to assign different reference signal for a low level PD controller to control a bipedal robot in various situations. Although this method effectively accelerates the learning process in the early stage, its overall performance is bounded by the low level heuristic controller, no matter how good the learnt planning policy can be. Here, the heuristic controllers represent some sub-optimal controllers designed based on human knowledge.

The proposed framework is also related to imitation learning [32, 115]. In the imitating approaches, the demonstrator must be good enough to solve the problem. But our approach stays in a larger spectrum since it should be compatible with both simple heuristic and complex learnable controllers. This is because, how to use these

controllers is decided by the switching policy. It learns to use suitable controllers under different observations and these controllers might be completely useless in other cases. Furthermore, after obtaining a good DDPG component, all of the heuristics (or controllers) can be removed, and we are able to test in the environment just by the DDPG component without any of the other controllers.

4.3 Problem Formulation and Assumptions

We can consider the local navigation problem as a decision making process [10] where the robot is required to avoid obstacles and reach a target position. At time $t \in [0, T]$ the robot takes an action $a_t = [v_t, \omega_t] \in \mathscr{A}$ according to the observation x_t which controls the linear velocity v_t and angular velocity ω_t of the robot. After executing the action, the robot receives a reward r_t given by the environment according to the reward function and then transits to the next observation x_{t+1}. The goal of this decision making procedure is to reach a maximum discounted accumulative future reward $R_t = \sum_{\tau=t}^{T} \gamma^{\tau-t} r_\tau$, where γ is the discount factor. Note that since this chapter only focuses on accelerating training in simulation, we directly apply the geometric sensory data, e.g. laser scans or depth camera in different simulators, as the observation.

The reward function r_t at time t is defined as:

$$r_t = \begin{cases} R_{crash}, & \text{if robot crashes} \\ R_{reach}, & \text{if robot reaches the goal} \\ (d_{t-1} - d_t)\cos(\omega_t) - C, & \text{otherwise} \end{cases} \tag{4.1}$$

where R_{crash} is a large penalty for collision, R_{reach} is a positive reward for reaching the goal, d_{t-1} and d_t denote the distances between the robot and the goal at two consecutive time steps $t-1$ and t and C is a constant time penalty which encourages the robot to approach the goal quickly.

4.4 AsDDPG: Learning with Deterministic Guidance

The main insight behind our framework is to provide a simple controller to assist the network in policy exploration, accelerating and stabilising the training procedure. We term the deterministic implementation of our framework as *Assisted* DDPG, or AsDDPG for short. The intuition is simple: a naive control law will outperform a random strategy for simple tasks e.g. driving in a straight line.

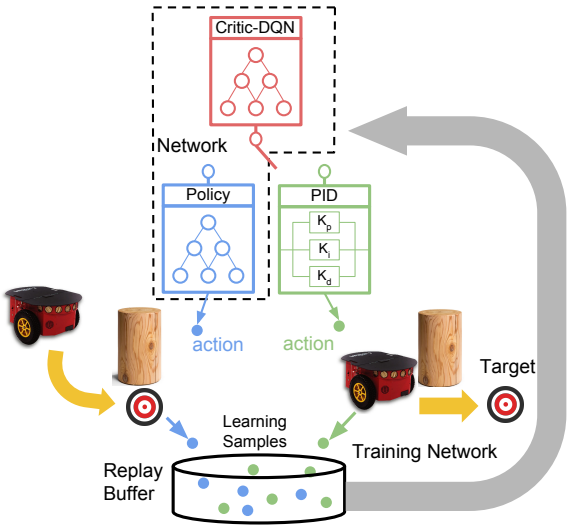


Figure 4.1: A deep neural network is trained with an actor-critic Reinforcement Learning approach to learn local planning for robot navigation. The critic-DQN network assesses both the performance of the external controller, e.g. a PID controller, and the policy network, selecting actions from a better one according to the situation. All the resulting learning samples are stored in the replay buffer. Therefore, the policy network can improve itself either by imitating the external controller or by examining its own policy.

However, instead of simply treating this controller as an independent exploration method like ϵ-greedy, we combine the critic network with a DQN to automatically judge which policy it should use to maximise the reward as shown in Fig 4.1. Essentially, the augmented critic network controls a switch which determines whether the robot follows the controller's suggested actions or the learned policy. This can avoid manually tuning parameters to decide when and how to use such an external controller, potentially deriving an optimal strategy.

Since AsDDPG is an off-policy learning method where the network learns from a replay buffer, regardless of the current policy, the actor network can benefit from learning samples generated by both its own recorded policy and the assisting controller. Initially, the simple controller will be chosen more frequently as the optimal policy. However, over time, the learned policy will outperform the simple controller in terms of total reward. Once training has converged, both the critic and the external controller can be discarded, and the robot simply navigates based on the learned policy.

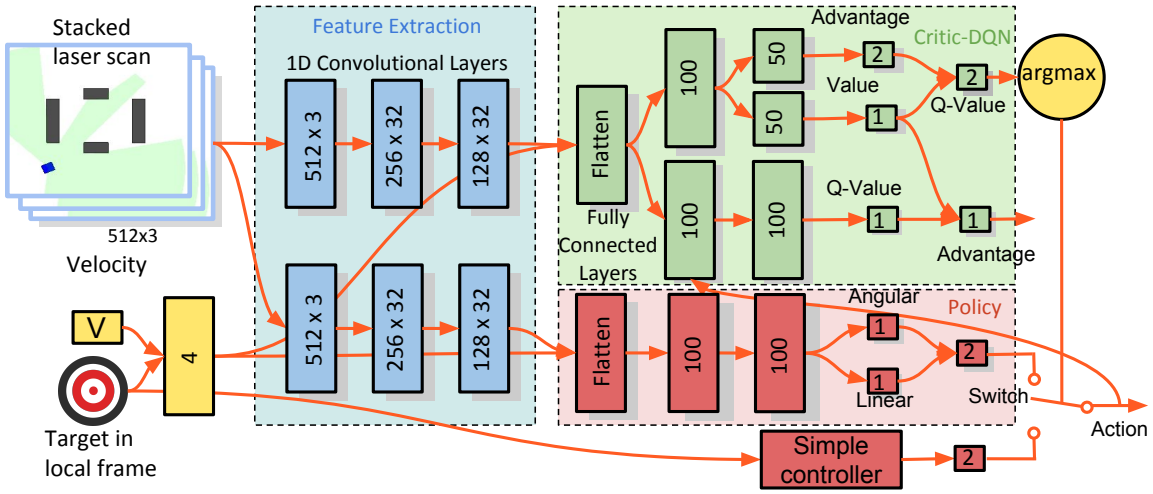


Figure 4.2: Network architecture. Network layers are demonstrated by the rectangles. Orange arrows indicate the connectivity between network layers and some other components, e.g. input state and the output of the simple controller. The final action is selected based on the Q value predicted by the critic-DQN.

4.4.1 Heuristic Controller

We firstly introduce the simple controller which is used as the external guidance. It is a P controller which is s special case of proportional-integral-derivative (PID) controller with proportional term [5] and derives action from the relative position of the destination $[x_{local}, y_{local}]$ in robot coordinate frame as:

$$a_t = K_p \cdot [x_{local}, y_{local}]^T, \tag{4.2}$$

where K_p is a two-dimensional coefficient vector for the proportional term. PID controller is one of the most widely used and successful control mechanisms. However, without considering geometric observation, it does not have an obstacle avoidance capability. The use of a simple P controller rather than a full PID controller is that the latter may introduce too much bias for the learning process and would possibly become harmful for reaching the optimal policy in the late training stage.

4.4.2 Network Architecture

Then, to bring the previously discussed intuition into practice, we implement the network architecture shown in Fig. 4.2. It includes three parts, namely feature extraction (blue), policy network and assistive controller (red), and the augmented critic network (green).

The first part of the system is the 1-D convolutional layers which are utilised to extract features from the stacked dense laser scans. The activation applied is ReLU. We find these convolutional layers to be typically important for the policy to reach both a good performance for obstacle avoidance and an acceptable generalisation ability in the real world. In [146], the author only uses a sparse laser scan (10 beams of a scan) which enables good generalisation in different scenarios. However, it is difficult for the robot to avoid small obstacles smoothly. Intuitively, decimating a high-fidelity observation loses information and is not ideal. Thus, we prefer to keep the dense laser scans as input and instead apply 1-D convolutional layers to learn efficient features for our task. Stacking inputs across multiple timestamps also provides more information on the environment.

The second part is the policy network with fully-connected layers, estimating the optimal linear and angular speeds for the robot based only on features extracted from the input state (e.g., laser scans, current speed and target position in local frame). Note that the activations for these two outputs are sigmoid and tanh, respectively. The external controller also generates a control signal (policy) based on the error signals between current and target positions.

Finally, the Critic-DQN constructed with fully-connected layers is the third part. It has two branches: one is the critic branch where the action is concatenated into the second layer; the other is DQN branch where we apply dueling [160] and double network architecture [156] to speed up the training and avoid overestimation. Note that there is no nonlinear activation for its output layers. We discuss the critic in more detail below.

4.4.3 Critic-DQN

The two branches of critic-DQN act respectively as 1) a criticiser, evaluating the action output from the policy network and generating policy gradients, and 2) a switch, deciding when to use the learned policies from the policy network or the external controller.

The critic branch is similar to the original DDPG. However, it estimates the advantage $A^\pi(x, a)$ from Q-value $Q^\pi(x, a)$ for each state-action pair. This is leveraged by a dueling network in DQN branch where the value $V^\pi(x)$ of each state will also be learned. With the definition of advantage, it can be simply calculated as $A^\pi(x, a) = Q^\pi(x, a) - V^\pi(x) \approx Q^\pi(x, s) - V^\pi(x)$. Note that the action considered by critic-DQN branch is the switching action s_t rather than a_t at each time step and is also different from the concept of a sampled trajectory s defined in Sec. 2.1.6.2. According to [142],

Algorithm 1 AsDDPG

procedure TRAINING
    Initialise $A(x, a|\theta^A)$, $Q(x, s|\theta^Q)$ and $\pi(x|\theta^\pi)$.
    Initialise target network $\theta^{A'}$, $\theta^{Q'}$ and $\theta^{\pi'}$
    Initialise replay buffer R and exploration noise ϵ
    for episode=1, M **do**
        Reset the environment
        Obtain the initial observation
        for step = 1, T **do**
            Infer switching $[Q_{policy}, Q_P] = Q(x_t, s_t|\theta^Q)$
            $s_t = argmax([Q_{policy}, Q_P])$
            if $s_t == 1$ **then**
                Sample policy action $a_t = \pi(x_t|\theta^\pi) + \epsilon$
            else
                Sample action a_t from external controller
            end if
            Execute a_t and obtain r_t, x_{t+1}
            Store transition $(x_t, a_t, r_t, x_{t+1}, s_t)$ in R
            Sample N transitions $(x_i, a_i, r_i, x_{i+1}, s_t)$ in R
            Optimise critic-DQN by minimising Eq. 4.3
            Update the policy according to Eq. 4.4
            Update the target networks
        end for
    end for
end procedure

compared with Q-value, estimating the advantage largely reduces the variance and is essential for fast learning especially for approaches based on policy gradient. The entire critic branch is denoted as $A(x, a|\theta^A)$ while the part before estimating the advantage is denoted as $Q^A(x, a|\theta^A)$.

The DQN branch estimates and compares the Q-values of either using the policy from the actor network or applying actions from the external controller based on the state inputs. It greedily switches to the one with a higher Q-value estimate. The DQN branch is denoted as $Q(x, s|\theta^Q)$.

4.4.4 Training

Learning samples can be defined as tuples $(x_t, a_t, r_t, x_{t+1}, s_t)$, where s_t is a binary variable, indicating the switching action. Among these variables, a_t will only affect the update for the critic branch while s_t is only for the DQN branch.

In the critic-DQN, both branches optimise their network parameters through bootstrapping. This means they learn from the temporal-difference (TD) error of the Q-value estimation with Eq. 2.3. More specifically, weights are optimised based on the following loss function L:

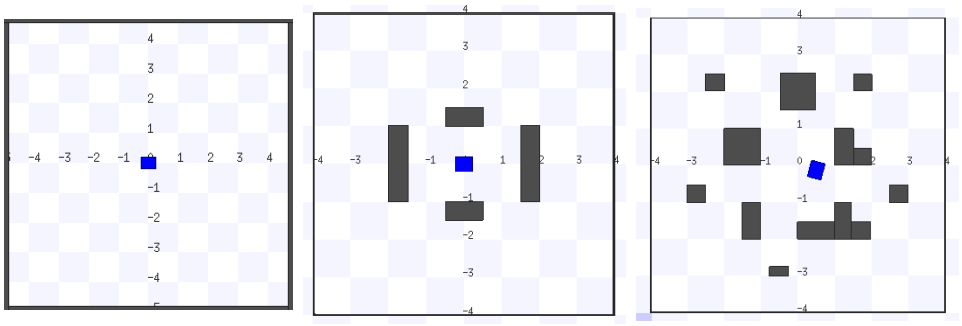
$$\begin{aligned} y_i^A &= r_i + \gamma Q^{A'}(x_{i+1}, \pi'(x_{i+1}|\theta^{\pi'})|\theta^{A'}) \\ y_i^Q &= r_i + \gamma Q'(x_{i+1}, \underset{s}{\operatorname{argmax}}\, Q(x_{t+1}, s_{t+1}|\theta^Q))|\theta^{Q'}) \\ L &= \frac{1}{N}(\sum_i (y^A - Q^A(x_i, a_i|\theta^A))^2 + (y^Q - Q(x_i, s_i|\theta^Q))^2). \end{aligned} \tag{4.3}$$

where $Q^{A'}$, Q' are target networks for the two branches and i is the indices of samples in the batch. Note that the critic branch is updated through its Q value estimation instead of the final output advantage which is used for generating the policy gradient.

For the actor network, its weights are adjusted with policy gradients which are defined in Eq. 2.8. This requires the critic branch to first compute the gradients of its advantage output $A^\pi(x, a)$ w.r.t. the action a. This is then transferred to the actor network to calculate the gradients w.r.t. the network parameters θ^π.

$$\begin{aligned} \nabla_{\theta^\pi} J(\theta^\pi) \approx \mathbb{E}[\nabla_{\theta^\pi} A(x, a|\theta^A)|_{x=x_t, a=\pi(x_t|\theta^\pi)}] = \\ \mathbb{E}[\nabla_a A(x, a|\theta^A)|_{x=x_t, a=\pi(x_t)} \nabla_{\theta^\pi} \pi(x|\theta^\pi)|_{x=x_t}]. \end{aligned} \tag{4.4}$$

Algorithm 1 outlines the entire training process of AsDDPG.



(a) Empty (b) Simple (c) Complex

Figure 4.3: The three Stage simulation worlds used for training. The gray rectangles are obstacles while the blue one is the robot. The target position is randomly generated for each episode.

4.5 Experiments for AsDDPG

Several experiments are conducted to evaluate the performance of the proposed AsDDPG algorithm against the original DDPG for the robot navigation problem. We train the networks in the Stage and Gazebo simulators and test the learned policy in a real world scenario.

For the training in the Stage simulator, we use three environments as shown in Fig.4.3. Both DDPG and AsDDPG are trained with the same reward function. Specifically, it contains a sparse part R_{reach} and R_{crash}, where the robot obtains a large positive reward for reaching the goal and a large negative reward for colliding with an obstacle, and a dense part as

$$r_t = \begin{cases} R_{crash}, & \text{if crashes} \\ R_{reach}, & \text{if reaches the goal} \\ \gamma_p((d_{t-1} - d_t)\triangle t - C), & \text{otherwise} \end{cases}$$

where d_{t-1} and d_t indicate the distance between robot and target at two consecutive time stamps, $\triangle t$ represents the time for each step, C is a constant used as time penalty. and γ_p is a discount factor.

Note that we introduced another reward discount factor γ_p. It is set to 1 for using policy network but to a smaller number (e.g. 0.5) for penalising the usage of the external controller when the dense reward is positive. This serves two purposes: 1) experimenting with the policy network more frequently and 2) learning faster on how to work independently of the external controller. Theoretically, the network can learn to gradually diminish the usage of the external controller since there is always a better policy instead of utilising the external controller.

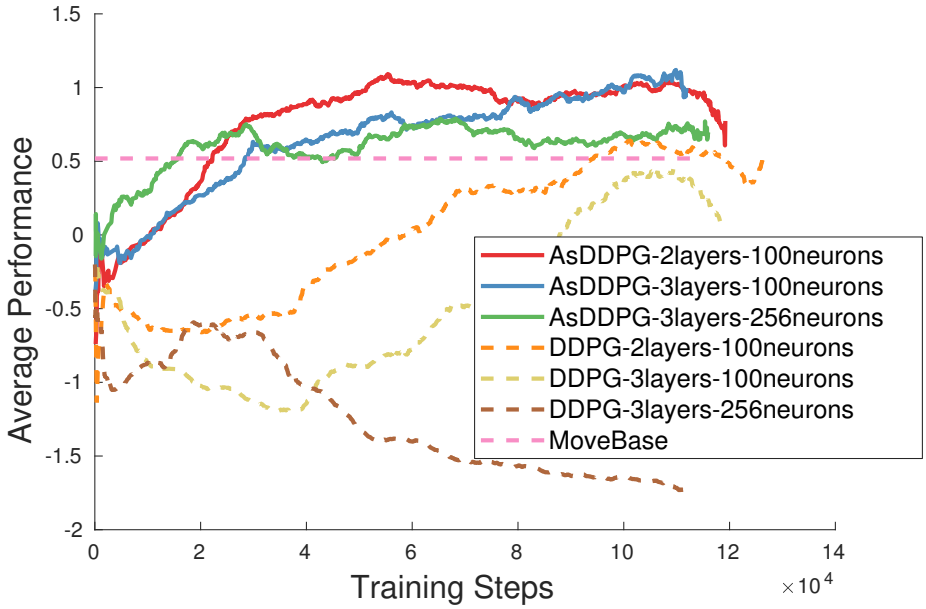


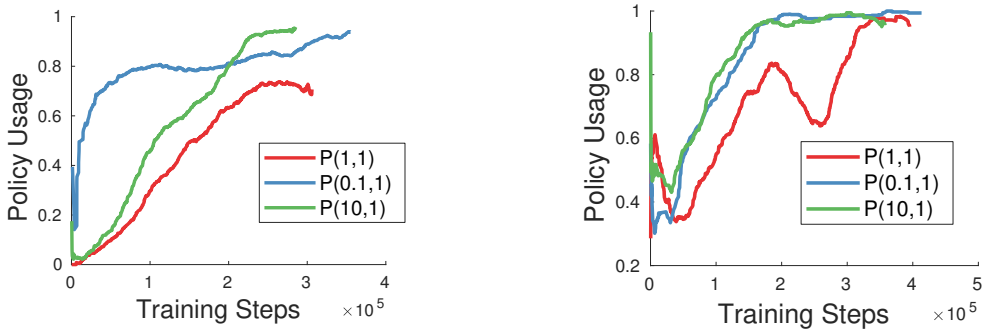
Figure 4.4: Smoothed learning curves with different network hyper-parameters. The figure illustrates the average performance of the network at each training step. Note that MoveBase is selected as the baseline approach whose average performance reaches a score at about 0.5.

The reward function above does not necessarily provide an objective performance metric since it is designed to alleviate the training difficulty. Hence, in this experiment we use a navigation task metric based on the time taken to reach the goal and whether the robot reaches the goal. More specifically, each step gives a time penalty of -0.01 and reaching the goal gives a positive reward of 2.

4.5.1 Speeding up Training Procedure with Various Hyper-parameters

A known problem of DDPG is the high sensitivity to network hyper-parameters. Manually tuning hyper-parameters to make DDPG converge is very time-consuming and is something that ideally could be avoided. Therefore, in this experiment, we examine networks with three distinct settings, including two or three fully connected layers with 100 neurons in each layer or three layers with 256 neurons.

The resulting learning curves over more than 100k training steps for the different hyper-parameters are shown in Fig. 4.4. They demonstrate that AsDDPG outperforms DDPG in terms of training efficiency, and is more stable than DDPG with different network hyper-parameters. Note that DDPG with three layers and 256 neu-



(a) Empty world

(b) Simple world

Figure 4.5: The usage of the policy network within an episode through the entire training procedure in two different simulation worlds.

rons per layer fails to learn a reasonable policy. Furthermore, we use MoveBase[1] as the baseline approach to show how the learnt policy compares with an existing deterministic approach. MoveBase package[2] is a widely used motion planner for mobile robot navigation and is implemented in the ROS package named Navigation Stage. It consists of a local planner [39, 36] and a global planner (implemented by A* algorithm). The global planner generates an optimal path from the origin to the destination on the global map of the environment, whilst the local planner dynamically avoids the newly detected obstacles while moving along the optimal path. However, in the experiment, the global planner is only provided with an empty map without recording any obstacle in the environment for a fair comparison. Meanwhile, the local planner is assigned with higher tolerance of deviating from the planned path according to the observed obstacles by increasing the parameter "pdist_scale".

4.5.2 Impact of Controller Parameters

In this experiment, several models are trained with different controller parameters in two Stage worlds (Fig 4.3a and Fig 4.3b) to investigate the sensitivity of AsDDPG to the controller parameters, i.e., how the controller parameters affect the training. We also examine how often the critic-DQN chooses the learned policy over the external controller, studying whether it can gradually become independent and learn a good policy.

The proportional controller $P(P_l, P_r)$ is configured with two parameters. It controls linear and rotational velocity as $v = P_l \cdot x_{local}$ and $\omega = P_r \cdot y_{local}$, where x_{local} and y_{local} are the coordinates of the target in the robot's local coordinate frame.

[1]http://wiki.ros.org/move_base
[2]http://wiki.ros.org/move_base

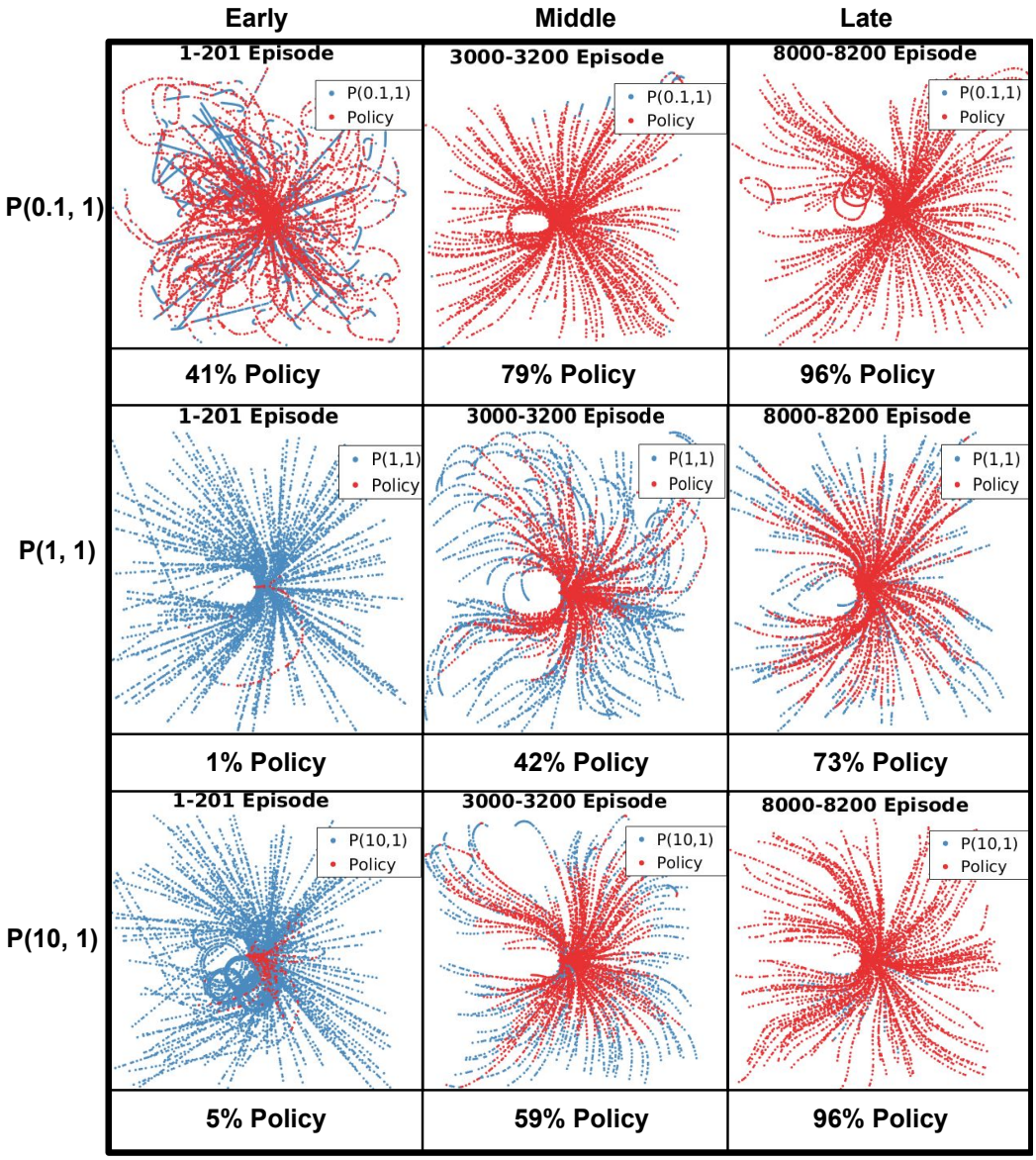
We investigate three settings of the controller by altering the linear gain parameter, namely $P(0.1, 1)$, $P(1, 1)$ and $P(10, 1)$. $P(10, 1)$ is the fastest controller, but suffers from overshooting of the target, requiring the robot to turn around. $P(0.1, 1)$ is the slowest controller, making very gradual changes to the robot's speed with consequent slow acceleration. $P(1, 1)$ is a controller that gives good performance without serious overshoot.

Fig. 4.5 shows how the ratio between the policy and the external controller chosen by the critic-DQN evolves over time during training. Firstly, it can be seen that the critic-DQN learns to sample less frequently from the controller over time, with more actions coming from the policy, although the parameters of the controller and the environment decides how fast this trend can be. One obvious phenomenon is that in both simulated worlds the network drops the slow controller $P(0.1, 1)$ rapidly since it takes a longer time to reach the goal in general and it is easy for the policy network to outperform it. However, the ratio between policy and controller for $P(1, 1)$ and $P(10, 1)$ differs in the two worlds. In Fig. 4.6a and 4.6b this behaviour is presented in more detail with robot trajectories at different training stages. These also show how often the critic network chooses the controller (blue trace) over the learned policy (red policy) as training progresses.

According to the results shown in Fig. 4.6a, regardless of the controller, the network first learns to apply its own policy for the first few steps, as shown by a concentration of red around the origin. This is because the total reward is heavily influenced by the initial heading. Considering the fastest controller $P(10, 1)$, it can be seen that initially the robot sometimes circles around the target, due to excessive speed. This problem is solved by the policy network by choosing a better heading at the beginning, demonstrating that the network can learn when to properly use the external controller.

Especially in the second row where $P(10, 1)$ is used, the robot sometimes hovers around the target when it is nearby. This problem is solved by the policy network by approaching the target with a more proper heading at the beginning.

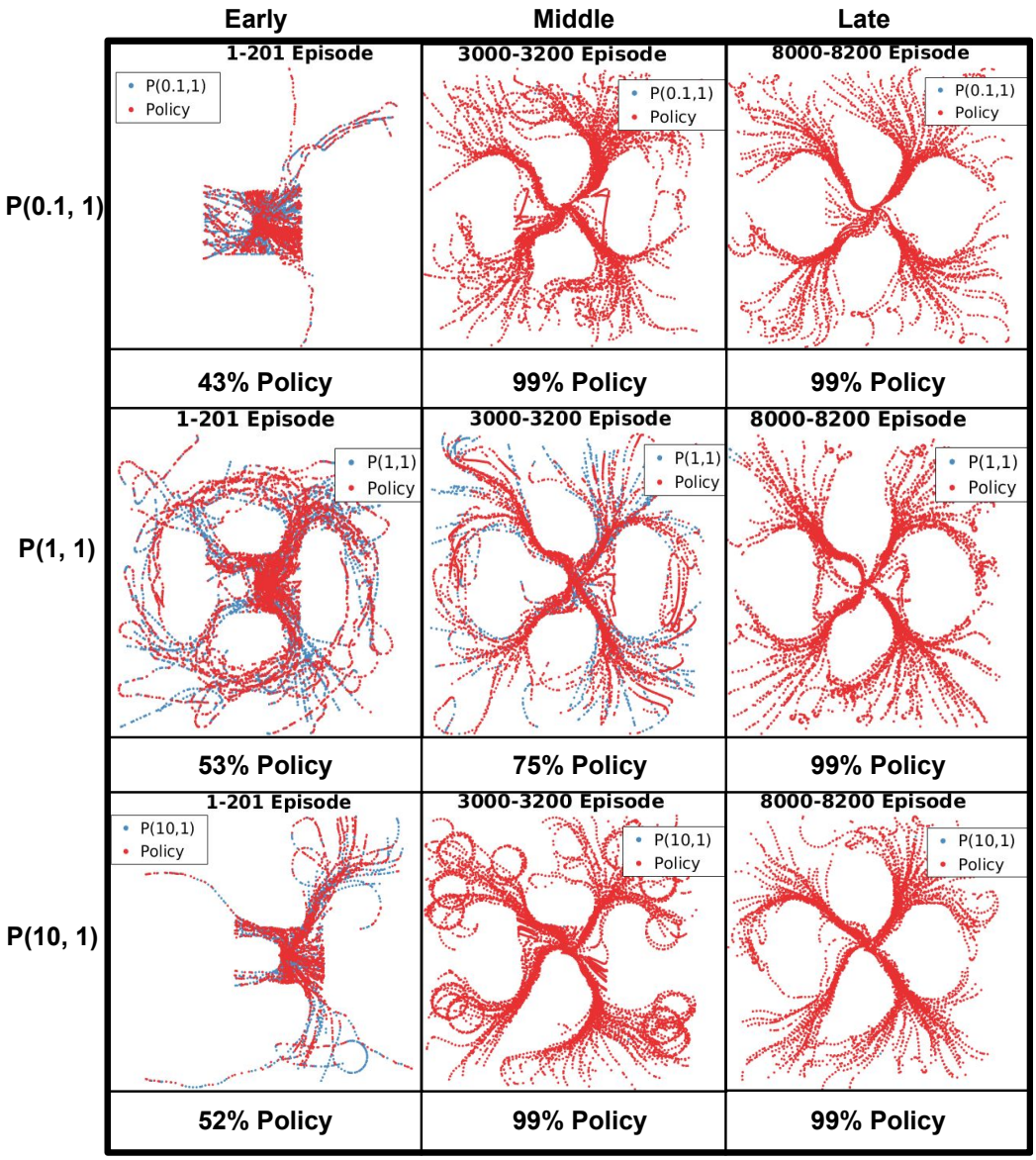
With $P(1, 1)$, it adopts a more accurate heading towards the target and navigates to it straightly. In the middle stage, the policy network learns a smoother but less optimal policy in terms of time. This could be a side effect of penalising the usage of the external controller through the reward function. However, this penalty remains essential for stabilising the switching strategy. Eventually, the path to the target learned by the policy network becomes more straight and optimal. Since $P(1, 1)$ is a good controller, the network does not discard it even after ≈ 8000 training episodes.



(a) Empty world

Figure 4.6: The policy learned by the robot with various proportional controllers at different stages of the training. Each image illustrates the trajectories of the robot as well as the switching results between the external controller and policy network across 200 episodes, respectively at the early, middle and late phases of the training procedure.

Fig. 4.6b shows the robot trajectories when training in the simple world environment. It can be seen that the network learns some distinct behaviours. The external controller is sampled less frequently than in the empty world even at the beginning. This is reasonable since a pure PID controller cannot deal with obstacle avoidance. Moreover, the learning speed is different with different controllers. For example, the



(b) Simple world

Figure 4.6: The policy learned by the robot with various proportional controllers at different stages of the training. Each image illustrates the trajectories of the robot as well as the switching results between the external controller and policy network across 200 episodes, respectively at the early, middle and late phases of the training procedure.

network learns to go around obstacles more easily with $P(1,1)$ than with others. With $P(10,1)$, the network learns much slower due to the confusing guidance from the controller. Although the network drops the controller early, it retains some undesirable behaviours like hovering around the target at the middle stage of training. However, it can be seen that eventually the critic becomes almost 100% independent

after $\approx$ 8000 episodes.

The experiments show that after a sufficient number of training steps, the learned policy can drop the external controller and is effective to navigate the robot around the obstacles. This verifies that the external controllers have little impact on the final performance of the AsDDPG if the networks are trained with sufficient episodes. However, different P values do affect the learning speed. An improper P value leads the agent to discard the P controller early and learn purely from random exploration whilst a better P controller can guide the learning procedure for a longer time and learn faster.

4.5.3 Training with Complex Environment and Sparse Reward

The next experiment aims to validate that AsDDPG can learn a good policy in a more complex environment as shown in Fig.4.3c using a sparse reward function. The dense reward function used in the previous experiments alleviates the difficulty of training by leading the robot to decrease its distance to the target for a higher instant reward. But, at the same time, it induces the network to learn a sub-optimal policy w.r.t. time because it is also driven by something else besides reaching the target fastest. In this experiment, we use a reward function where the dense part is simply a constant time penalty.

$$r_t^{sparse} = \begin{cases} R_{crash}, & \text{if robot crashes} \\ R_{reach}, & \text{if robot reaches the goal} \\ -C, & \text{otherwise} \end{cases} \tag{4.5}$$

This is a challenging reward function for random exploration, as the robot only receives a positive reward when it actually reaches the target.

The performance of DDPG and AsDDPG is illustrated in Fig. 4.7. It can be seen that DDPG seldom learns a proper policy to reach the goal and always collides with the obstacles. More specifically, in the early stages, DDPG runs out of time frequently, which is the worst case due to the accumulative time penalty. After approximately 80k training steps, it learns to crash to avoid the time penalty instead of reaching the goal. In contrast, although AsDDPG also runs out of time at the beginning, it transits to crashing as a better strategy within only 3k steps, and eventually learns to reach the goal for the maximal total reward. This further verifies that the proposed AsDDPG can effectively speed up the training and achieve a better performance with the assistance from the external controller, even in complicated environments with

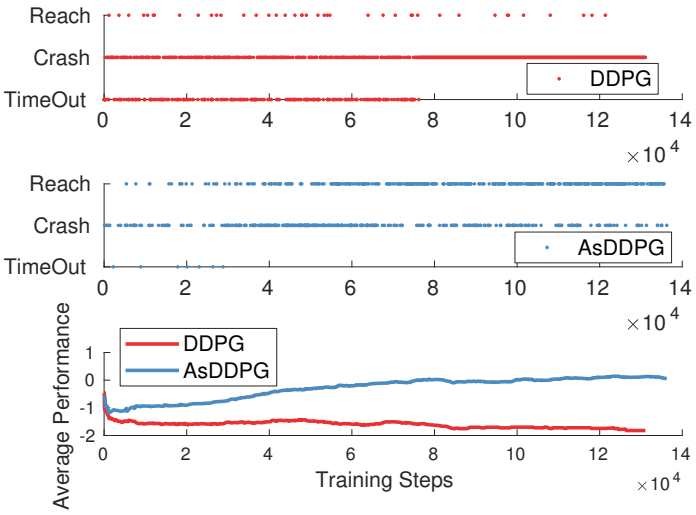


Figure 4.7: The final result (Reach the target, Crash and Time out) for each episode and the smoothed learning curves.

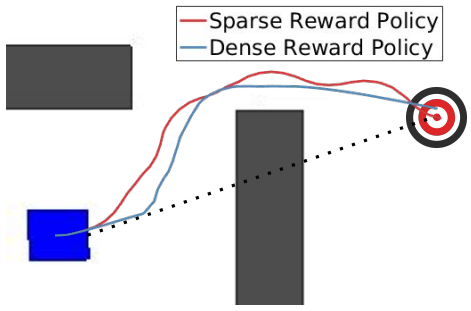


Figure 4.8: Policies learned with dense and sparse reward functions where the sparse reward policy takes 43 steps (8.6 sec.) to reach the goal while the dense reward policy takes 53 steps (10.6 sec.).

an extremely sparse reward function for which a random exploration guided network can hardly learn a good policy.

In addition, by using the sparse reward function, the network learns to drive the robot faster keeping a reasonable safe distance to the obstacles. This is shown in Fig. 4.8. With the dense reward, the robot tend to smoothly skirt around the obstacle by slowing down and executing a gentle rotation. However, with a sparse reward, the network learns to plan earlier to avoid obstacles, giving them a wider berth. As such, the robot can travel at a maximum speed, even if the path to the target is not the shortest.

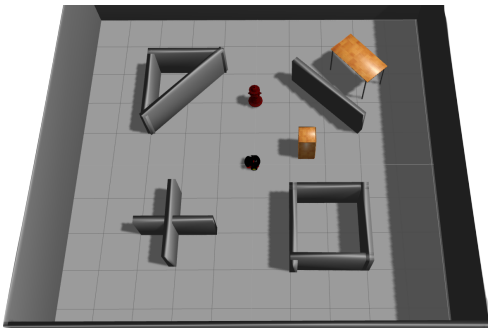


Figure 4.9: Gazebo

Figure 4.10: Real world

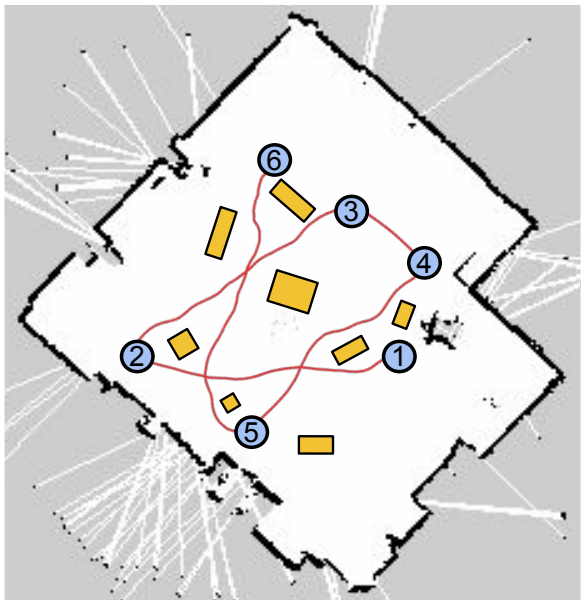


Figure 4.11: The trajectory of the robot in the real world experiment. The yellow rectangles are obstacles and blue circles indicate the sequential targets.

4.5.4 Real World Tests

In the real world experiment, a Pioneer robot which is equipped with a Hokuyo laser scanner is utilized. To localize the robot based on an existing map, we apply the AMCL ROS package. Since our network only acts as a local planner for reaching a near goal without any collision, it is combined with a global planner to achieve a complete navigation system. More specifically, after receiving the destination, the global planner generates a path to the target and each point of the path is transferred into the robot's local frame as a network input, together with the laser scans and the speed of the robot.

The simulated Gazebo environment shown in Fig. 4.9 is used to train the network. Then, it is directly tested on the real robot in the real world scenario with several

obstacles (Fig. 4.10). In this experiment, a map without any obstacles in the room is established and the robot is driven by the learned policy to reach several target points successively with obstacle avoidance. The robot trajectory and obstacles overlaid on the map are illustrated in Fig. 4.11. The trajectory of the robot is plotted as the red curve. The experiment shows that the robot can smoothly avoid all the obstacles and reach each target successfully.

4.6 SGuidance: Learning with Stochastic Guidance

After the investigation of learning a deterministic switching policy, we now turn to a stochastic counterpart. The main insight is that when learning the deterministic switching policy, a noise is manually tuned and introduced to explore a better switching policy which can hardly reach the optimal balance between the exploration and exploitation. By contrast, learning a stochastic switching policy can tackle the issue essentially and let DDPG eventually converge to a better policy.

More concretely, different from AsDDPG, the switching mechanism in SGuidance is constructed as a stochastic function updated by the REINFORCE learning signal [164] to maximise total reward. Meanwhile, the DDPG component is learned by employing the action selected by the stochastic switch, rather than directly using the output action generated by its policy network. Therefore, the switching mechanism helps DDPG avoid trivial explorations during the early training process, and learns to balance between exploration and heuristic guidance. More interestingly, once trained, the DDPG component can be used in isolation from the other controllers, in which case the switch is turned off and the navigation is carried out solely by the DDPG component. Similar to the idea of imitation learning [116, 115], the DDPG component is able to learn from the demonstrations given by the guidance, PID and OA (obstacle avoidance) components in our case, and instantly generalise to new situations (which PID and OA could not handle). Here, the guidance can be considered as a inductive bias for reducing the variance of gradient estimators, and the model is able to remove this bias after benefiting from it.

For quantitative evaluation, we firstly compare our model with the incorporation of the stochastic switch to the vanilla DDPG baseline and deterministic benchmarks for demonstrating the benefits brought by bootstrapping with additional primitive controllers. Then the influence of using different independent controllers is investigated, which shows that the framework has strong generalisation ability and it is

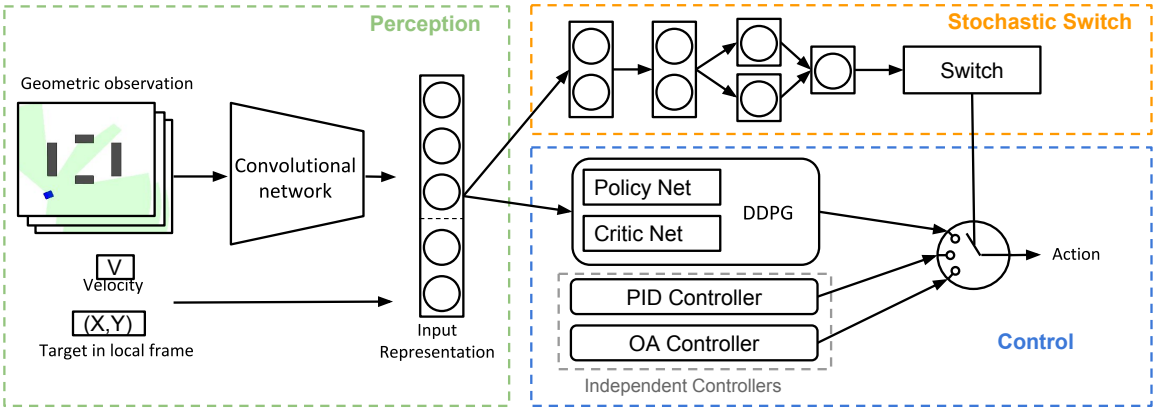


Figure 4.12: The architecture of the proposed framework. It consists of three sections namely: perception, control and stochastic switch. The perception section processes an observation and generates a corresponding input representation. The control section contains a standard DDPG controller and a number of independent controllers - in this case, a Proportional-integral-derivative (PID) controller and an Obstacle Avoidance (OA) controller. The stochastic switch determines which controllers' action should be selected for navigation.

able to accumulate the benefits from different simple controllers. In addition, we propose three variants of the switch mechanism including a uniformly random switch, an argmax switch (similar to AsDDPG) and a Thompson sampling switch for comparison. Finally, we show that the models can abandon the extra controllers when their usage rate declines below a threshold and are able to continue self-learning by only using the DDPG component. For qualitative evaluation, we test our model in a real world scenario: without further modification, the model trained in simulation is able to be directly transferred to carry out navigation tasks.

4.7 Model

The proposed model consists of three parts: perception, control and stochastic switch as shown in Fig. 4.12. At each time step, the perception part processes an observation and generates a corresponding input representation. Then different controllers can propose candidate actions based on the input representation. Finally, the stochastic switch determines which one of the actions to be carried out. Since the perception and the DDPG component are the same as the ones in AsDDPG we only introduce the heuristic controllers and the stochastic switch in SGuidance.

4.7.1 Heuristic Controllers

Different from AsDDPG, in this section, two independent controllers are utilised to aid DDPG to learn reasonable policies. One is the P controller introduced in Sec. 4.4.1. The additional one is a simple obstacle avoidance (OA) algorithm which can drive the robot without collision. It uses geometric observations to detect and avoid nearby obstacles by controlling the heading direction (rotational speed) of the robot:

$$|a^{\omega}| = \begin{cases} a^{\omega}_{max} \cdot \frac{|d_o - \beta|}{\beta}, & d_o < \beta \\ 0, & \text{otherwise.} \end{cases} \tag{4.6}$$

where d_o is the distance to the closest obstacle, a^{ω}_{max} represents the largest rotational speed and β indicates a pre-defined minimum safety distance. In the case where the distance between the robot and an object is less than the safety distance, i.e., $d_o < \beta$, the robot will rotate to avoid collision. These two controllers complement one another to provide candidate actions for stochastic switch. Note that the OA only produces a^{ω}, while the selected a^{v} is provided by the DDPG controller. Note that these are just two simple exemplar controllers, and it is possible to incorporate more either in number or sophistication.

4.7.2 Stochastic Switch

The PID controller, OA algorithm and DDPG are three independent sources that produce candidate actions for the switch network to (optimally) select. The switch network is a stochastic deep neural network which consists of a parameterisation network and a multinomial distribution. The switch effectively learns which controller to choose, based on the anticipated reward and conditioned on the input representation. We regard it as another MDP where the action s_t taken by the learning agent is to select among three controllers at each time step. This switching policy is trained end-to-end with the other deep learning components e.g. DDPG and the optional input representation network.

4.7.2.1 Stick-breaking

Conventionally, a softmax layer can be employed to provide the parameter θ for the multinomial distribution. Here, instead, we apply **stick-breaking construction** [134, 59, 93], which is an alternative to softmax.

The intuition is to introduce a bias that encourages more usage of the deep reinforcement learning algorithm, such as DDPG in our case. Since our framework is

designed to train a robust DDPG component that benefits from the stochastic guidance, we expect it to be used more often than others in this framework so that we are able to get rid of the simple independent controllers after a certain period of training. It basically transforms the modeling of multinomial probability parameters into the modeling of the logits of binomial probability parameters.

We firstly define $\alpha = f_s(x_t|\theta^s)$ as the unscaled logits from the fully connected layers of switching network $f_s(\cdot|\theta^s)$ given the input representation x_t and θ^s as the parameter of the switch network. Then the binomial logits η can be obtained with $\eta = \text{sigmoid}(\alpha)$. Notice that $\eta = [\eta_1, \eta_2]^T$ and $\alpha = [\alpha_1, \alpha_2]^T$ are all 2 dimensional vectors since we have 3 controllers in our case.

Therefore, the multinomial probability parameters $\xi = [\xi_1, \xi_2, \xi_3]$ can be generated with the stick-breaking function $f_{\text{SB}}(\eta)$ by two breaks:

$$\begin{aligned} \xi_1 &= \eta_1 \\ \xi_2 &= \eta_2(1-\eta_1) \\ \xi_3 &= (1-\eta_2)(1-\eta_1), \end{aligned} \tag{4.7}$$

where the f_{SB} can be generalised to more breaks $\xi_k = \eta_k \prod_{i=1}^{k-1}(1-\eta_i)$ $(1 < k < K)$ if there are K controllers.

Conditioned on the current observation x_t, we are able to construct the stochastic switch policy $s_t \sim \pi^s(s_t|x_t;\theta^s)$ as:

$$\xi = f_{\text{SB}}(\text{sigmoid}(f_s(x_t|\theta^s))) \tag{4.8}$$

$$s_t \sim \text{multinomial}(\xi). \tag{4.9}$$

At each time step t, the stochastic switch samples a decision s_t and ξ_1, ξ_2, ξ_3 corresponds to DDPG, PID and OA. Then, according to the decision s_t, the critic network of DDPG takes the final action $a_{s_t} \in \{a_{\text{DDPG}}, a_{\text{PID}}, a_{\text{OA}}\}$ as input and updates the networks accordingly. Meanwhile, the stochastic switch is updated by the REINFORCE learning signal so that the switch network is able to dynamically choose to learn through exploration (DDPG), or it can choose to use the output of a heuristic controller (PID or OA) as guidance by observing the environment.

4.7.2.2 REINFORCE Algorithm with Baselines

Since the gradients cannot be directly back-propagated through the discrete samples, we employ REINFORCE algorithm [164] to construct the gradient estimator for the switch network. Its theory has been fully explained in Sec. 2.1.6.2, and here we only

note that the gradients are calculated with the following equation, given the most recent episode:

$$\sum_{t=0}^{T} \nabla_{\theta^\pi} \log \pi(s_t|x_t, \theta^\pi)(R(x_t) - b^c - b(x_t))$$

where $R(x_t)$ is a deep neural network based estimator of the total future reward given x_t, b^c and $b(x_t)$ are the learnt input independent and dependent baselines that are both modelled with fully connected layers.

The introduction of stochastic switch can be considered as an inductive bias for learning to navigate with better action samples. Updated by REINFORCE, the stochastic switch is able to sense the environment, avoid trivial explorations and select better actions for learning DDPG policies. In addition, as the independent controllers are incorporated via the stochastic switch, the negative influence of the introduced biases from the heuristics is limited. The independent controllers can be explicitly disabled once DDPG is sufficiently trained, although this naturally happens to a large degree anyway.

4.7.3 Algorithm

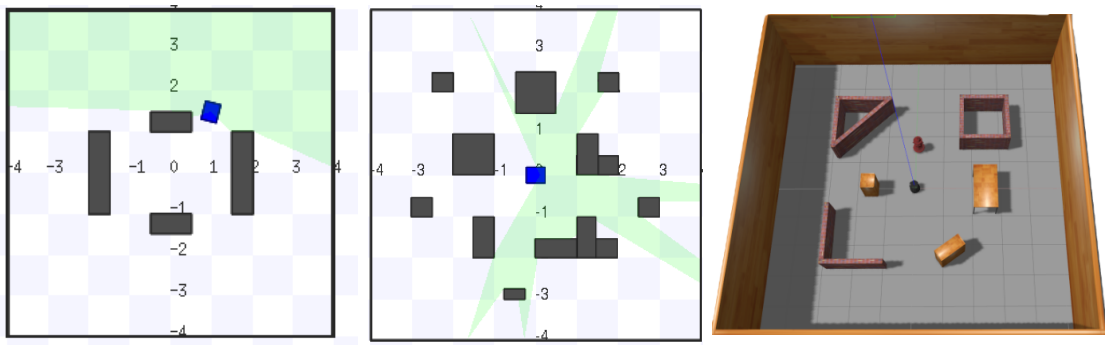
The brief algorithm of training DDPG with our stochastic guidance (SGuidance) is demonstrated in Algorithm 2. Since DDPG is an off-policy approach, a replay buffer R is applied to store all the transitions (x_t, a_t, r_t, x_{t+1}). A batch of transition is sampled at every training step to update DDPG. In contrast, the learning of switching policy with REINFORCE is an on-policy process, thus we only save the current trajectory of switching (x_t, s_t, r_t), where $t = 1, 2, ...T$ to calculate the gradients for updating the switching network, including the control variate, at the end of each episode.

4.8 Experiments for SGuidance

Both simulated and real world experiments have been carried out to validate our proposed architecture. In this section, we experiment our models in both simulated and real world environments. We quantitatively evaluate the training efficiency and performance in the simulator, and compare to the baseline models. Then, we qualitatively test the model in the real world robot navigation task.

Algorithm 2 SGuidance

procedure TRAINING
    Initialise switching network $f_s(x_t|\theta^s)$ and buffer R_s.
    Initialise DDPG network $Q(x, a|\theta^Q)$ and $\pi(x|\theta^\pi)$.
    Initialise the target network of DDPG.
    Initialise replay buffer R and exploration noise ϵ.
    for episode=1, M **do**
        Reset the environment.
        Initialise replay buffer R_s.
        Obtain the initial observation x_t.
        for step = 1, T **do**
            Sample switch $s_t = f_s(x_t|\theta^s) \in \{0, 1, 2\}$.
            Sample $a_t^0 = \pi(x_t|\theta^\pi) + \epsilon$.
            Get a_t^1 from PID controller.
            Get a_t^2 from OA controller.
            Execute $a_t = a_t^{s_t}$ and obtain r_t, x_{t+1}.
            Store transition (x_t, a_t, r_t, x_{t+1}) in R.
            Sample a batch of transitions from R.
            Update the actor and critic network of DDPG.
            Update the target network of DDPG.
            Store switching trajectory (x_t, s_t, r_t) in R_s.
        end for
        Update switching network with trajectory in R_s.
    end for
end procedure



(a) *ROS Stage simple* (b) *ROS Stage complex* (c) *ROS Gazebo*

Figure 4.13: **(a)** The 4 grey rectangles are obstacles and the blue square represents the robot. A sparse laser is mounted on the robot and its detecting area is illustrated as the green area. **(b)** It shows a more complex environment simulated by *ROS Stage*. **(c)** *ROS Gazebo* is also used for training a model that can be transferred to a real-world environment. Turtlebot 2 (a platform for ground robots) is employed as the mobile platform equipped with a depth camera.

4.8.1 Training Environments and Settings

The proposed framework is trained in two different simulators. The first one is a light-weight simulator, *ROS Stage*[3] (Fig. 4.13a and Fig. 4.13b), in which a large amount of repetitive experiments are conducted for showing the learning curve, demonstrating the improvements brought by stochastic guidance, and comparing to other baseline models. In this simulator, we mount the mobile robot with a laser scanner to provide the geometric information of surroundings. Hence, the convolutional neural network (in Fig. 4.12) is not used in this case, and the laser scans are directly concatenated with the other observations as input representation. By accelerating the simulation time, we obtain the **quantitative evaluation** through a lot of repetitive experiments in *ROS Stage*. Note that the simple environment as shown in Fig. 4.13a is our default training environment and experiments in a more complex environment are carried out in Sec. 4.8.2.6.

The other simulator, *ROS Gazebo*[4] (Fig.4.13c), contains a physical engine and can accurately simulate the dynamics of the mobile robot. Thus the model trained in *ROS Gazebo* is directly applied to the real world scenario to **qualitatively evaluate** the navigation performance, but it has a larger computational overhead compared to *ROS Stage*. Here, depth images are utilised to observe surroundings, therefore a 3-layer

[3]http://wiki.ros.org/stage
[4]http:/wiki.ros.org/gazebo˙ros˙pkgs

convolutional network (the filters are [4,4,3,8], [4,4,8,16] and [4,4,16,32] respectively) is constructed to provide input representations based on depth images.

In each training episode the robot starts at the origin point (centre point) with a random heading direction and the destination is randomised within the area beyond obstacles. When the robot collides with an obstacle or reaches the destination, the current episode terminates. The action control frequency is 5Hz (0.2s delay between every two consecutive time steps) and the switching frequency is 1Hz. For all the experiments carried out in *ROS Stage*, the training process lasts for 100k steps and is repeated for 5 times. The averaged learning curves as well as the variance[5] are illustrated for demonstrating the performance.

Regarding the hyper-parameters, the hidden layers of the critic and actor networks contain 100 ReLU units in each layer, while the output layer of actor network applies tanh and sigmoid respectively for rotational and linear velocity. When updating DDPG parameters, 32 learning samples are randomly sampled from a rank based prioritised experience replay [127] as a training batch, and the learning rate for the actor network, the critic network and the stochastic switch are 10^{-4}, 10^{-3} and 10^{-3} respectively, and the rest follows [80].

4.8.2 Navigation in Simulated Environments

Here, we discuss the experiments carried out in *ROS Stage* for quantitative evaluation. The default settings for applying stochastic switch with DDPG is to learn with guidance from both PID and OA controllers using the REINFORCE algorithm and control variates (CV).

4.8.2.1 Reinforcement Learning with Stochastic Guidance

Fig. 4.14 compares the proposed and competing models and demonstrates the benefits brought by learning with stochastic switch. **SGuidance** is our model with stochastic switch that dynamically choose the action from the candidates proposed by the controllers of DDPG, PID and OA. As shown in the figure, **SGuidance** achieves significantly better performance than the **DDPG** baseline. Meanwhile, **DDPG** suffers from the high variance issue according to the wide transparent area around the learning curve, while **SGuidance** is much more stable. This is due to the high complexity of the environment that leads to the highly variant learning samples provided by DDPG, which might lead to trivial explorations. In addition, the stochastic gradient

[5]Note that the variance mentioned here is the variance of the smoothed learning curves.

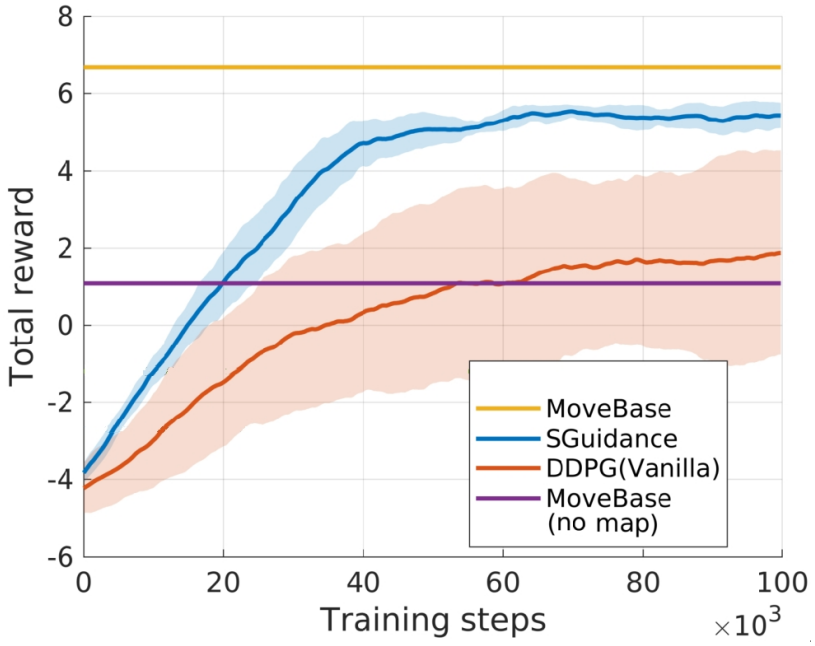


Figure 4.14: The total reward achieved by our models and other baseline models as comparison. The curve represents the average value of 5 repetitive training procedures and the transparent area indicates the variance of the results.

estimator of DDPG applies biased approximation which makes it difficult to guarantee the convergence and stability. By contrast, **SGuidance** is able to benefit from the heuristic simple controllers at the early stage of the training procedure instead of starting from completely random moves.

In this experiment, we also plot the rewards of **MoveBase** (no map) and the complete **MoveBase** for comparison. Note that the MoveBase package has been introduced in Sec. 4.5.1 which is able to generate the optimal actions to the destination with a global map of the environment. Therefore, we regard the complete **MoveBase** as the an oracle controller in this experiment which sets a upper bound of the performance whilst the **MoveBase** (no map) as a fairly compared baseline. Note that the specific settings of **MoveBase** (no map) is described in Sec. 4.5.1. As shown in Fig. 4.14, **DDPG** is able to obtain comparable performance to **MoveBase**. **SGuidance**, however, significantly surpasses the deterministic **MoveBase** model. Even without the access to the global map, **SGuidance** has shown its strong ability to navigate in the environment by just using the geometric information. Basically, the simple deterministic controllers cannot be applied independently for carrying out navigation task (the accumulative rewards are both under 0). However, when incorporated with DDPG via the stochastic switch, they contribute significantly towards alleviating the high variance issue encountered with vanilla DDPG.

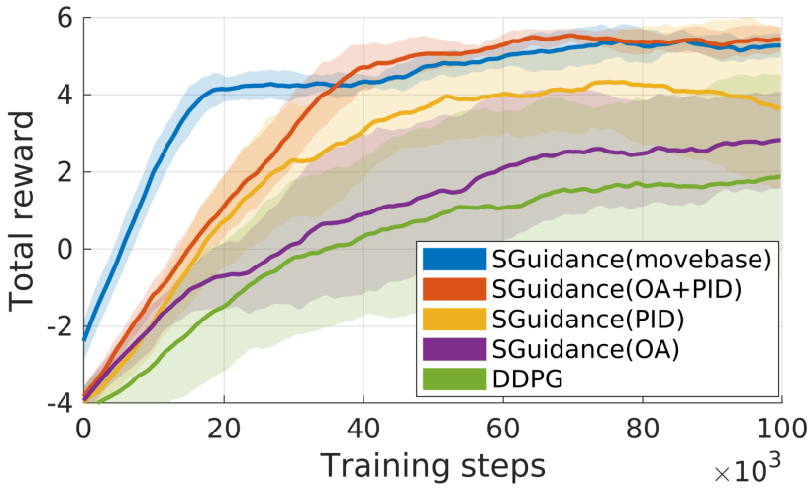


Figure 4.15: The smoothed total reward obtained by incorporating different heuristic controllers with DDPG. **SGuidance (PID+OA)** utilises both PID and OA controllers while **SGuidance (OA)** and **SGuidance (PID)** only adopt one of them respectively. **SGuidance (movebase)** is guided by movebase and **DDPG (Vanilla)** is the baseline DDPG without heuristic guidance.

4.8.2.2 Using Different Independent Controllers

This experiment shows the impact of different independent controllers on training performance.

As illustrated in Fig.4.15, **SGuidance (PID + OA)** achieves the best performance when compared to the DDPG with only PID or OA and the DDPG without any independent controllers. It demonstrates that the contribution of the stochastic switch is greatly enhanced by adding more controllers, which yields more stable learning curves and better navigation performance. Interestingly, the PID controller brings more benefits than the OA controller in this context, and their benefits could be accumulated with the help of the stochastic switch. Additionally, when MoveBase is utilised as the independent controller alongside DDPG, there is an obvious improvement on learning speed, especially in the early stage of training. These results show that the proposed framework enables us to incorporate a large range of different controllers, from naive to more sophisticated ones. This is a massive advantage over imitation learning which requires a good demonstration to learn from.

4.8.2.3 Using Different Switching Mechanism

Fig. 4.16 compares the stochastic switch to other switching variants, including argmax switch, a uniformly random switch and a switch based on Thompson Sampling [121].

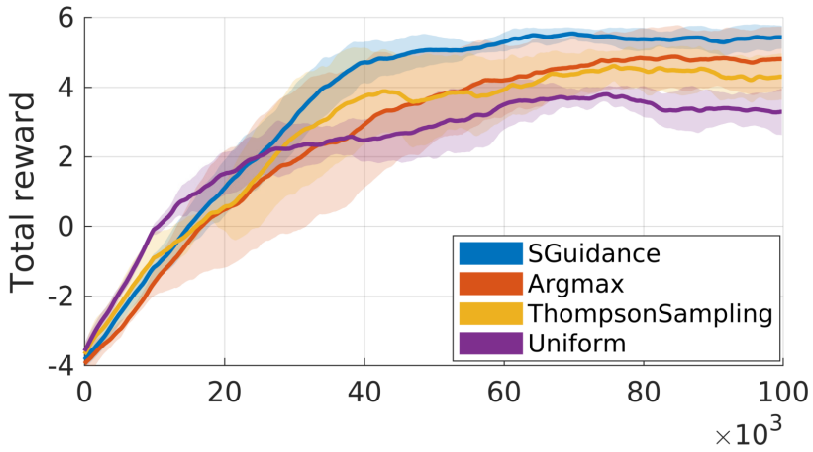
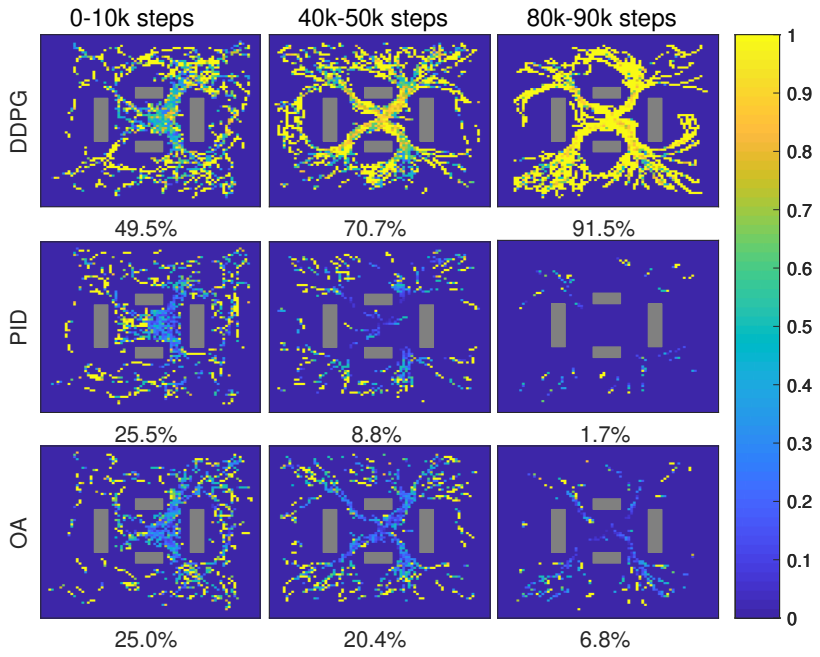


Figure 4.16: The accumulated reward obtained by learning with different switching mechanisms. **Stochastic** represents our default stochastic switch settings. **Argmax** and **Uniform** are the proposed argmax switch and uniformly distributed switch respectively. **ThompsonSampling** is the Thompson Sampling implemented with Gaussian prior.

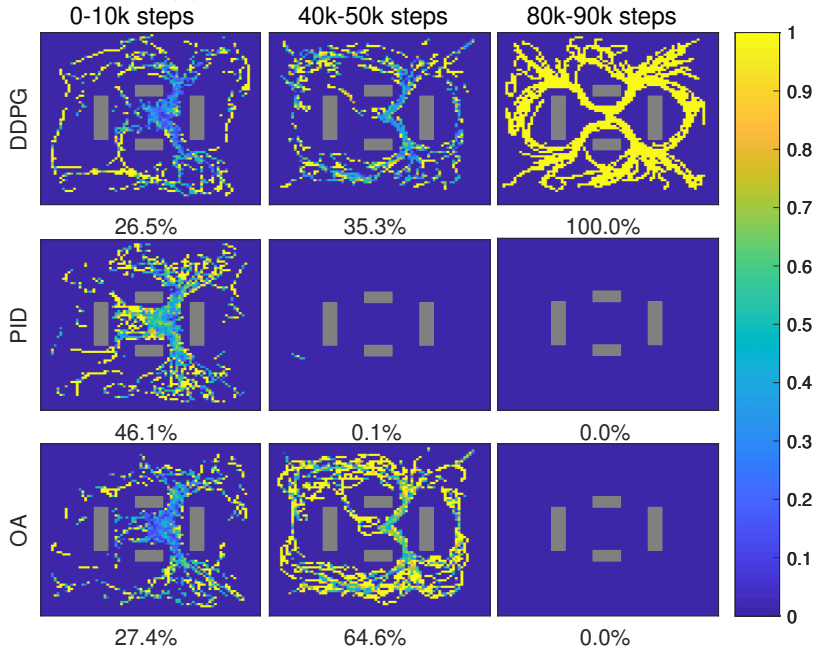
The uniform switch assigns fixed uniform probability to DDPG, PID and OA controllers, while the argmax switch applies biased argmax output instead of stochastically drawing samples from the stochastic switch network. Thompson sampling [121] is implemented with a Gaussian prior distribution and estimates the posterior of the total reward when deploying each controller. As illustrated in Fig. 4.16, **SGuidance** has the best performance while **Uniform** is the worst. Note however, that **Uniform** actually learns the fastest in the first 20k steps. **ThompsonSampling** has a comparable performance to **Argmax**. The former learns slightly faster mainly due to the stochasticity and keeps seeking a potentially better switching policy. Later **Argmax** outperforms **ThompsonSampling** and **Uniform** as it selects controllers based on current observations while both **ThompsonSampling** and **Uniform** do not. Compared to **SGuidance**, **Argmax** has larger variance on the total reward. This is because **Argmax** is a biased sampler and the introduced bias in turn damages its final performance since there is less exploration.

4.8.2.4 Further Comparison with Thompson Sampling

To understand the benefits obtained by learning a situation-aware switching policy, we compare the usage of different controllers between **SGuidance** and **ThompsonSampling** during training. Fig. 4.17a and Fig. 4.17b demonstrate the spatial density of controller usage, where each row represents the employment of a single controller and each column indicates the early (10k-20k steps), middle (50k-60k steps) or late



(a) The switching policy of SGuidance



(b) The switching policy of Thompson Sampling

Figure 4.17: Spatial density of using each controller (DDPG, PID or OA) in three different learning periods (top by **SGuidance** and bottom by **ThompsonSampling**). Each row represents the employment of a single controller and each column indicates the early (0-10k steps), middle (40k-50k steps) or late (80k-90k steps) training stages. Note that all grey areas are obstacles.

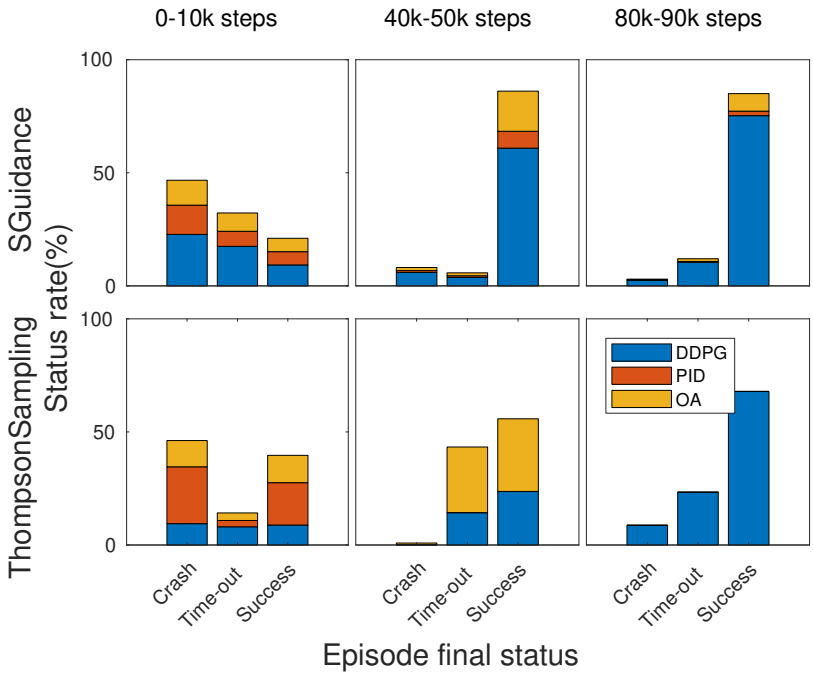


Figure 4.18: Illustration of the rate of reaching different final status in an episode when applying **SGuidance** and **ThompsonSampling** respectively. Each column represents a training period and different segments in each bar indicate the usage ratio of different controllers within this period.

(90k-100k steps) training period. Note that the training is based on the *ROS Stage simple* environment in Fig. 4.13a and in each episode the robot is initialised in the centre and the destination is randomised.

Figs. 4.17a and 4.17b show the main differences between the switching policies of **SGuidance** and **ThompsonSampling**. **SGuidance** explores the strengths and weaknesses of each controller in different situations while **ThompsonSampling** does not. Notice in the second row of Fig. 4.17a that **SGuidance** gradually abandons candidate actions from PID controller when it is still surrounded by the obstacles and only uses it when approaching the target. The OA controller (third row) is occasionally adopted to avoid obstacles. However, in 4.17b, it is shown that the usage of each controller does not depend on the position in the map. In fact, **ThompsonSampling** completely abandons PID and OA controller eventually. Since it can only estimate the overall performance of the controller in an episode, it will easily converge to DDPG when it is confident enough about the superiority of DDPG after a long training period without considering the context.

Fig. 4.18 plots the rate of reaching different final status and the average usage of each controller in an episode when applying **SGuidance** and **ThompsonSam-**

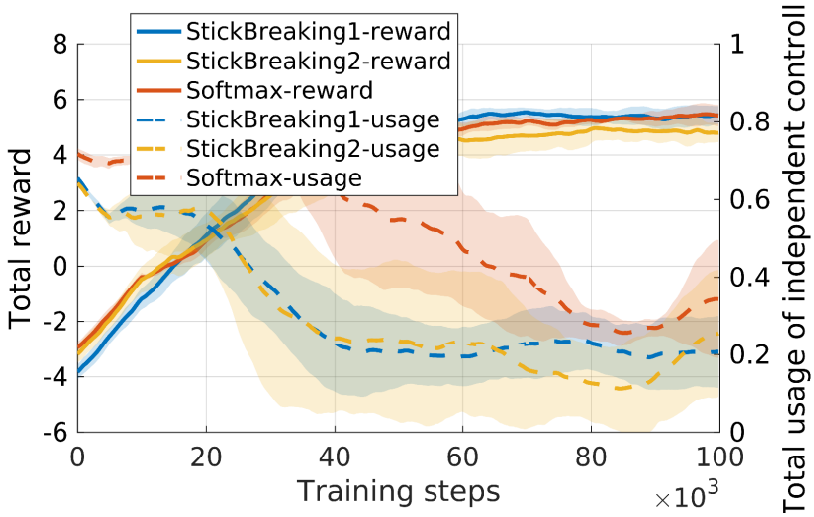


Figure 4.19: Illustration of the impact of using different stochastic switch functions. The left y-axis shows the total reward of all the methods, and the right y-axis shows the total usage of the independent controllers.

pling respectively. In the first row, we can see that the proportion of each controller differs in different classes of episodes when applying **SGuidance**. It avoids utilising the incorrect controller in most places, thus it hardly ever crashes with PID or OA controllers. Note, however, that with **ThompsonSampling**, the sample frequency of each controller is equally distributed in different final statuses as it only controls the overall sampling rate of each controller, regardless of the specific situation.

4.8.2.5 Construction of Stochastic Switch Function

Since we have introduced a bias, which can be considered as a preference, of selecting different controllers with stick-breaking, this part will investigate the effect of setting different orders for PID and OA controller in stick-breaking. In Fig. 4.19, **StickBreaking1** (DDPG, PID, OA) represents the order we introduced in the Sec. 4.7.2.1 and the **StickBreaking2** (DDPG, OA, PID) represents an alternative order of the independent controllers. More specifically, according to Eq. 4.8, **StickBreaking1** and **StickBreaking2** both set ξ_1 with DDPG controller and give different order with PID and OA controller where ξ_2 is assigned with the PID controller in **StickBreaking1** but with OA controller in **StickBreaking2**. As shown in the figure, **Softmax** is able to obtain almost same total rewards compared to **StickBreaking1**. However, according to the total usage of independent controllers, the DDPG component is being less used in **Softmax** than in **StickBreaking1** and **StickBreaking2**. Hence, the softmax function is a safe choice to construct the stochastic switch function which

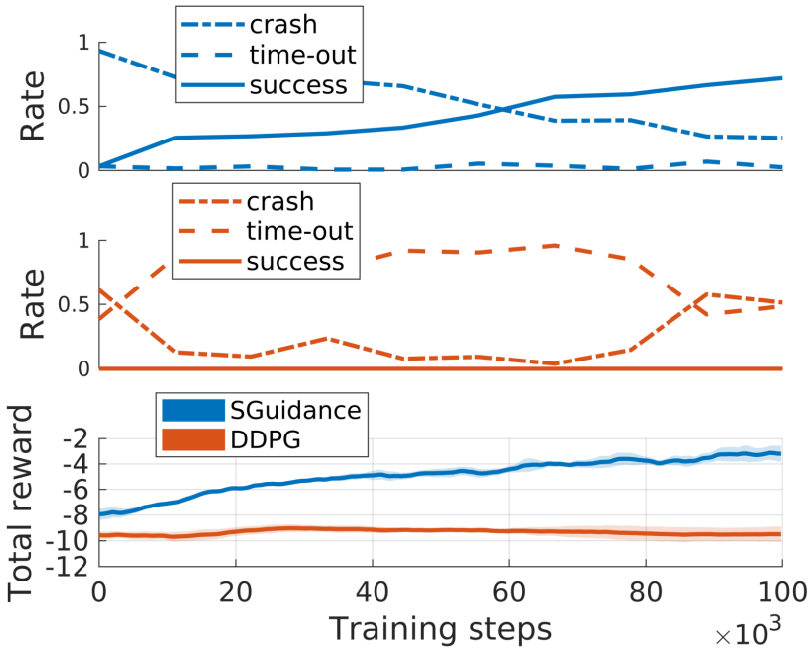


Figure 4.20: Experimental results on the complex environment shown in Fig. 4.13b with sparse rewards. The robot only receives a reward when crashing or reaching the destination, besides a time penalty at each step. The first two rows represent the rate of getting different ending status (crash, time-out or success) of each navigation episode when deploying the guidance or not. The total reward of each episode is also displayed in the last row.

assigns equal importance on all controllers and the DDPG policy. However, allocating more preference on the DDPG policy with stick-breaking can discourage the dependence on simple controllers and learn more from self-exploration. This would be more helpful if we have clear prior-knowledge about the performance of simple controllers when they are too naive to handle the task. On the other hand, although the two stick breaking functions have similar total usage of independent controllers, **StickBreaking2** performs slightly worse than **StickBreaking1**, which shows that the order of independent controllers has a small effect on the performance.

4.8.2.6 Complex Environment and Simple Reward

In this experiment, we test **SGuidance** in a more cluttered environment as displayed in Fig. 4.13b where the robot has to bypass more obstacles to reach randomly assigned destinations. In addition, the reward function is again simplified as shown in Eq. 4.5 where the robot only receives a reward when crashing or reaching the destination, besides a time penalty at each step.

The first two rows of Fig. 4.20 provide the trend of obtaining different ending status of each episode through the training period with/without the stochastic guidance respectively. Since the stochastic guidance incorporates the PID controller which can easily collide with obstacles, it hardly ever ends the episode with time-out status. With the growing performance of the DDPG controller, **SGuidance** reaches the destination more frequently. However, the naive DDPG without any guidance never explores to reach the destination safely. Therefore it only learns to avoid obstacles. Interestingly, it later begins to crash after a short period of travel, which leads to a similar total reward as ending up the episode with time-out status. The last sub-figure exhibits the corresponding learning curves with and without stochastic guidance.

4.8.2.7 Inference without Guidance

This experiment explores whether the trained DDPG policy is able to independently carry out navigation with all the heuristic controllers turned off after it has been trained with stochastic guidance.

We firstly train the DDPG with **SGuidance** and compare the inference performance of keeping or turning-off the stochastic guidance. The experiment is carried out in the *ROS stage simple* environment with reward defined in Eqn. 4.1.

As shown in Table 4.1, the overall performance only has a slight decrease after switching-off the guidance in testing.

Interestingly, when we carry out the same test in the *ROS stage complex* environment, turning-off the guidance even improves the success rate (**SR**) and reduces the crash rate (**CR**). This is because **SGuidance**, which is similar to **Thompson-Sampling** at this point, keeps exploring potentially better options. However, this exploration also occasionally leads to erroneous actions especially in the late training period or the testing phase and is counter productive. This applies more to the complex environment as a single wrong action may lead to a crash.

4.8.2.8 Learning with Different γ

In this experiment we investigate how different values of the reward discount factor γ affects the switching policy learnt by **SGuidance**. As shown in Fig. 4.21, with a smaller γ, **SGuidance** samples the DDPG less frequently since it becomes a more greedy controller and has a worse long-term performance. As a consequence, when the policy in DDPG cannot reach the destination, **SGuidance** prefers the OA controller which, at least, can avoid collisions, i.e. large instant negative rewards.

Guidance	Poli-U	PID-U	OA-U	SR	TR	CR	AR
Test in Simple World with Dense Reward							
Keep	88.1%	3.5%	8.4%	92%	7%	2%	5.01
Turn-off	100%	0%	0%	90%	4%	6%	4.91
Test in Complex World with Sparse Reward							
Keep	81.7%	2.5%	15.8%	81%	14%	5%	-2.29
Turn-off	100%	0%	0%	83%	16%	1%	-2.34

Table 4.1: Comparing the performance between keeping/turning-off the guidance after training in terms of the usage of three controllers (**Poli-U**, **PID-U**, **OA-U**), the rate of the final state of each episode (**SR**: success rate, **TR**: time-out rate, **CR**: carsh rate) and also the average total reward of each episode (**AR**).

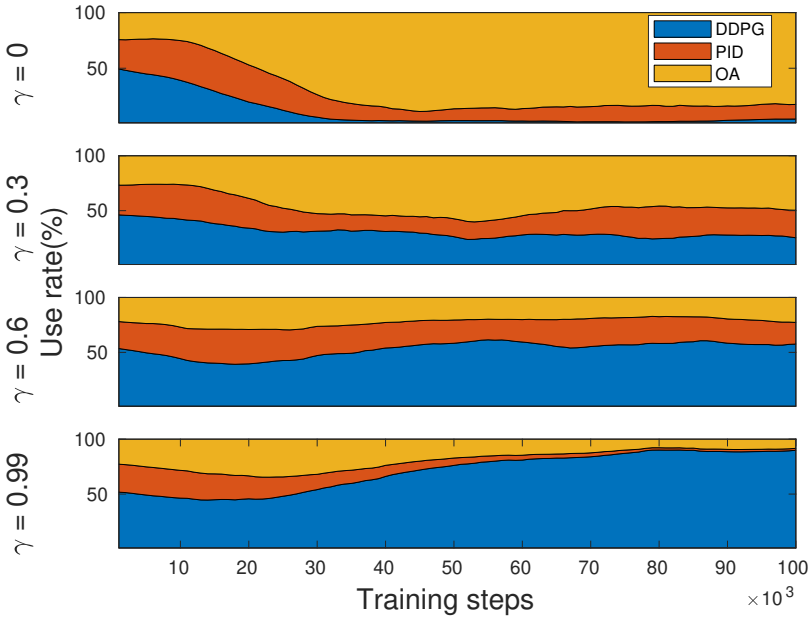
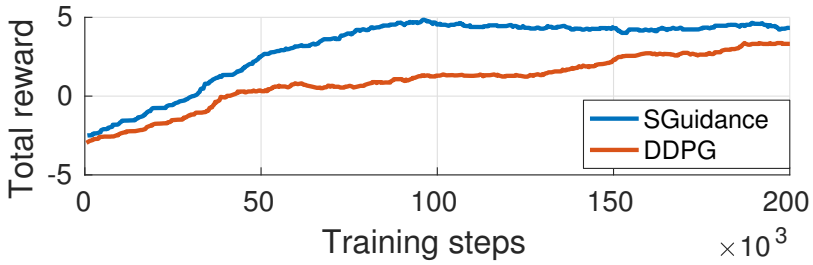


Figure 4.21: Use rate of each controller by **SGuidance** with different discount factor γ. Each figure draws the use rate of three controllers with a different γ.

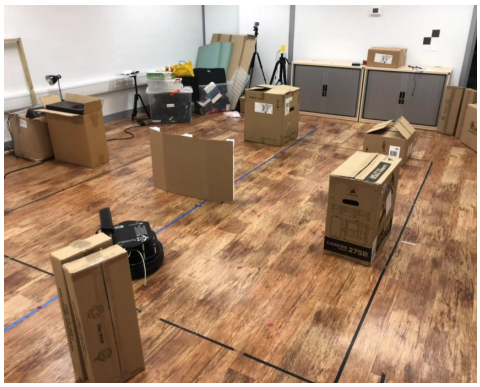
4.8.3 Navigation in a Real World Environment

In this experiment, we qualitatively analyse the performance of our model applied in real world environments. The model is trained in a simulated world built by *ROS Gazebo* (Fig. 4.13c) with the dense reward function as Eq. 4.1. It is then directly transferred into the real world scenario without any fine tuning in order to verify the effectiveness and strong generalisation of the model. The learning curves, which are illustrated in Fig. 4.22a, show that **SGuidance** can outperform naive **DDPG**.

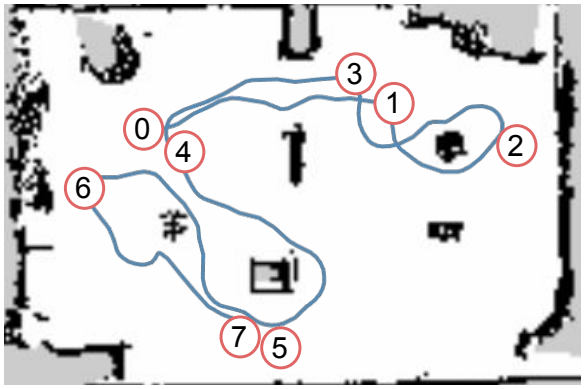
A Turtlebot 2 robot mounted with a Kinetic depth camera is used as the mobile platform. Hence, to deploy the trained model on the real robot, we apply the



(a) Learning curves in Gazebo Simulator



(b) Real world environment



(c) Destinations and navigation trajectory

Figure 4.22: **(a)** Comparison of the total reward achieved by **SGuidance** and **DDPG** models in ROS Gazebo simulator. Note that since Gazebo simulates physics properties, the control policy is more difficult to learn and the training takes longer time when compared to ROS stage. **(b)** Real world scenario. A turtlebot is used as the mobile platform and several boxes are placed in the room as obstacles. **(c)** The room layout and obstacles are the black areas. The blue curve represents the trajectory of the robot and the goals are plotted with red circles where the number indicates the sequence.

depth image as the model input. Additionally, since the dimension of depth image is dramatically increased, a 3-layer convolutional neural network is employed (as in Fig. 4.12) to provide geometric representations. Other inputs, i.e. velocity and goal location, are concatenated with the geometric representation into a dense input representation.

Since the ground truth of the robot locations is not available in the real world environment, we apply the off-the-shelf AMCL ROS package[6] for providing the estimation of the robot location, and calculating the destination position in the local coordinate frame. In order to improve the localisation accuracy, we record the map of the environment with Gmapping ROS package[7]. It is worth mentioning that this

[6]http://wiki.ros.org/amcl
[7]http://wiki.ros.org/gmapping

global map is not used by the navigation component of the model during training or testing. The obstacles are laid out in the room as illustrated in Fig. 4.22b. The aim of this experiment is to employ the learned policy and control the robot to reach several destinations successively without any collision. Fig. 4.22c shows the trajectory of the robot (blue curve). Notice that the robot can smoothly avoid all the obstacles and reach each target successfully by only learning in simulation with the proposed stochastic guidance model.

4.9 Conclusion

In this chapter, a new framework for effectively incorporating heuristic controllers is proposed to overcome the issue of high variance in DDPG, as applied to the task of local navigation. Our key novel insight is that, alongside learning the policy of DDPG, we also learn a context-dependent switching strategy to decide whether to sample actions from the heuristic controllers as the training data for the DDPG component. Therefore, unlike existing approaches, this "soft" guidance 1) can be either simple or expert controllers; 2) is only utilised when they can benefit the training of DDPG; and 3) is automatically discarded when DDPG outperforms them.

Extensive experiments demonstrate that both our deterministic and stochastic variants of this framework significantly bootstrap the learning process and can surpass state-of-the-art baseline models especially when using a very sparse reward function. In deployment, the trained DDPG policy can be used without requiring input from the guidance controllers. Thus, this chapter demonstrates how to efficiently learn practical robotic navigation.

Chapter 5

Global Navigation: Mapless Navigation with Sparse Directional Guidance and Visual Reference

After investigating the solutions of deploying the learnt policy on physical robots and accelerating the training procedure in the simulated environments, we now focus on the ultimate challenge of learning global planning with mapless navigation. Global planning is very challenging in this case due to the lack of a spatial map of some form, and thus requires additional environmental information as the guidance. The type of guidance to use, and how to effectively combine it with real-time robot control are the key problems to address. In this chapter, we introduce a practical mapless robot navigation system, SnapNav. It is a learning based navigation system which only requires a few sequential directional commands and visual reference images to navigate in unknown environments. We introduce a two-level 'command-and-control' hierarchy for this task. For the high-level commander we propose a novel self-supervised training approach to enhance its performance with limited real-world data. We also evaluate the utility of using feed-forward or recurrent policy networks for the low-level controller. The experimental results prove that SnapNav can be directly deployed on physical robots and robustly navigate in unseen real-world environments.

5.1 Introduction

"Where is the reception?"
"Go straight and turn left when you see the exit."

The above conversation is a very common and efficient example of human interaction when querying the route to an unknown destination. These kinds of instructions

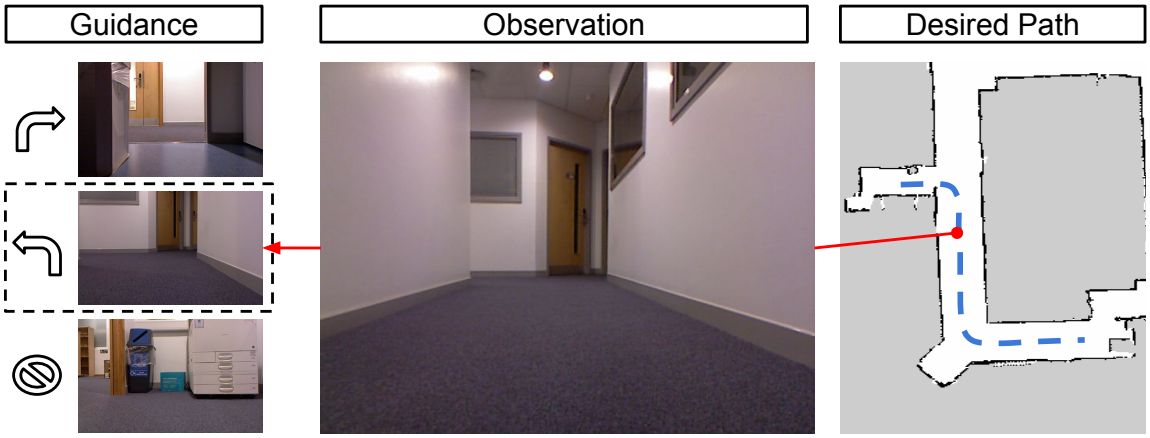


Figure 5.1: An example of the visual navigation task with sparse directional guidance. The robot automatically selects from the provided guidance based on its current observation.

that consist of a sequential action paired with a visual reference are widely used not only in navigation but also in various kinds of activities such as reading the user guide for a new product. These types of navigational instructions have two prominent characteristics. The first is the direct conjunction between the action and the visual observation spaces which provides specific guidance about what to do and where to do it. The second notable feature is the inherent sparsity as complex instructions can be distilled into a few key actions at visually important waypoints or cues, relying on the innate capacity of humans to navigate between these points. Thus, this form of guidance yields accurate instructions that can be efficiently communicated.

This observation triggers an interesting question: can robots mimic these human behaviours to navigate in a completely unknown environment when supplied with very sparse guidance? Such a robot system will be highly efficient at communication and possess a strong ability to generalize towards unfamiliar scenarios, without requiring a pre-built map of the world. In addition, we could imagine that such a system is likely to be more robust to dynamic changes, subject to the waypoints remaining visually distinctive.

In this chapter, we present a deep neural network based system, SnapNav, as a practical solution to mapless visual navigation in unknown environments. It has two characteristic features which are outlined as follows. Firstly, it can navigate in an unknown environment with only a few pieces of guidance. As shown in Fig. 5.1 the guidance consists of a snapshot image and the desired action (turn left, turn right, stop) before each turning or termination point along the path. Secondly, the

navigation system is designed with a two-level hierarchy to fully benefit from training data from different domains, i.e. appearance or depth observations, simulation or reality, and to be able to straightforwardly deploy the learnt policy on real robots.

5.2 Related Work

Learning-based mapless visual navigation has demonstrated remarkable performance in complex environments. Many works have previously proposed to learn a shortest path strategy by encoding the environmental information into the parameters of deep neural network [99, 181, 98]. Although these techniques exhibit robust navigation capabilities, these agents are actually overfitting the training environment and cannot apply the learnt experience to previously unseen scenarios. Another branch of mapless navigation approaches is local navigation [146, 169, 178] where the prerequisite of a known relative goal position largely restricts its real world applications to ones where accurate location exists.

Recently, researchers have introduced learning agents that can follow a demonstrated path [72, 53, 143, 112] which is similar to the traditional Visual teach and repeat (VT&R) paradigm [37] in robotics. This class of solutions require simple descriptions of the environment when navigating in an unknown environment, e.g. raw camera image sequences, or additionally, the labelled actions, instead of a pre-defined map. Among them only [53, 112] can be deployed on real robots. However the former is trained with a large amount of manually labelled real-world data from an omni-camera while the latter requires an explicit localisation of the demonstrated image in the sequence, largely discounting its practicality. Furthermore, reliance on a long video stream hinders efficient communication between agents.

Natural language instructions, as a form of extremely sparse guidance, is also introduced in visual navigation [159, 51]. Although it imitates human behaviour and can produce an agent with a greater degree of autonomy, the difficulty of visual language grounding, i.e. associating the perception from two completely different modalities, together with the ambiguity of the natural language itself, limits the performance of language guided navigation. The agent mentioned in [51] takes a list of thumbnail images of the street view coupled with natural language instructions as guidance and is similar to our SnapNav but only learns high level navigation strategies in a simulated toolkit named StreetNav. It does not learn low-level tasks required for a real robot such as obstacle avoidance. In some ways, it is also overfitting to the fact that streets are well defined and marked to aid human driving through visual cues

such as intersections, traffic lights and stop signs. Their approach thus uses a single policy network, trained and executed solely in simulation, and as we demonstrate in this chapter, this is not sufficient for deployment on a real robot.

5.3 Task Description

The task we are investigating in this can be classed as visual path following with sparse guidance. In particular, we consider the challenges of real-time perception and control of an autonomous robot, and take the robot out of pure simulation into reality.

5.3.1 Task Decomposition

Although it is conceptually straightforward to solve the entire task with a single policy network as in [72, 51], it is non-trivial to deploy those systems on a real robot. On the one hand, due to the absence of a simulator that can simultaneously render realistic camera data and precisely capture the dynamics of robots, command and control strategies that are deployable in reality cannot be learnt with a single simulator. On the other hand, manually labelling real world data is labour expensive [53] and limitations on training samples rarely lead to a robust control policy due to the lack of exploration. However, by decomposing the problem into a high level **commander** and a low level autonomous **controller**, we can learn sub-policies with separate kinds and sources of data to optimise each sub-task. More concretely, since the visual matching of current observations and snapshots in guidance heavily relies on the appearance of the observed objects, the command strategy demands data with high visual fidelity but can largely ignore the robot dynamics. Conversely, the control policy only focuses on the geometry of the surroundings for obstacle avoidance, robot dynamics and occasional commands/high-level actions. Therefore this policy can be more easily trained in a robot simulator where the robot dynamics as well as depth observations are finely modelled, building on work presented in the previous chapters. The robot can also be exposed to a wide range of arbitrary world setups, allowing for more robust local navigation. Each sub-task will be formulated more specifically in the following parts.

5.3.2 Command Sub-Task

The commander C is supplied with n pairs of guidance $\{\mathbf{G}_i = (\mathbf{S}_i, m_i) | i = [1, 2, ..., n]\}$ where $\mathbf{S}_i$ and m_i represent snapshots and guidance commands respectively. Each

snapshot $\mathbf{S}_i$ in the guidance records a first-person-view RGB image of the area where the robot should either alter direction or stop at a particular point. Since the robot is only given the order to vary direction or whilst terminating, there are only three types of commands in guidance, $m_i \in \{$"`Turn right`", "`Turn left`", "`Stop`"$\}$, with the implicit action being to carry on in a straight path.

Then with all pieces of guidance and the current image observation $\mathbf{O}_t$ from an on-board camera, the commander C predicts a high level command $c_t \in \{$"`Turn right`", "`Go forward`", "`Turn left`", "`Stop`"$\}$ at each time step t as $c_t = C(\mathbf{O}_t, \mathbf{G}, \mathbf{h}_c)$ where $\mathbf{h}_c$ is the hidden state of the GRU cell in commander.

5.3.3 Control Sub-Task

The control sub-task is formalised as a Markov Desision Process (MDP). At time $t \in [1, T]$ the robot takes an action $\mathbf{a}_t \in \mathscr{A}$ according to the observation $\mathbf{X}_t$. After executing the action, the robot receives a reward r_t given by the environment according to the reward function and then transits to the next observation $\mathbf{X}_{t+1}$. The goal of this MDP is to reach a maximum discounted accumulative future reward $R_t = \sum_{\tau=t}^{T} \gamma^{\tau-t} r_\tau$, where γ is the discount factor.

More specifically, the action is the control signal of the robot, $\mathbf{a}_t = (a_t^v, a_t^\omega) \in \mathscr{A}$, where a_t^v and a_t^ω respectively denotes the expected linear and rotational velocity at time t. The observation $\mathbf{X}_t$ is a first-person-view depth image which can be directly accessed in a simulator, e.g. *ROS Gazebo*[1], or estimated from an RGB image with an off-the-shelf estimator [43] in the real world. The reward function r_t at time t is defined as:

$$r_t = \begin{cases} R_{crash}, & \text{if robot crashes} \\ R_{reach}, & \text{if robot reaches the goal} \\ d_{t-1} - d_t, & \text{otherwise} \end{cases} \tag{5.1}$$

where R_{crash} is a penalty for collision, R_{reach} is a positive reward for reaching the goal, d_{t-1} and d_t denote the distances between the robot and the goal (the next turning point or the final destination) at two consecutive time steps $t-1$ and t.

5.4 Network Architecture

5.4.1 Attention-Based Commander

To accomplish the command sub-task, the commander is designed with convolutional neural network (CNN) layers and linear embedding (EMB) layers to process raw

[1] http://wiki.ros.org/gazebo_ros_pkgs

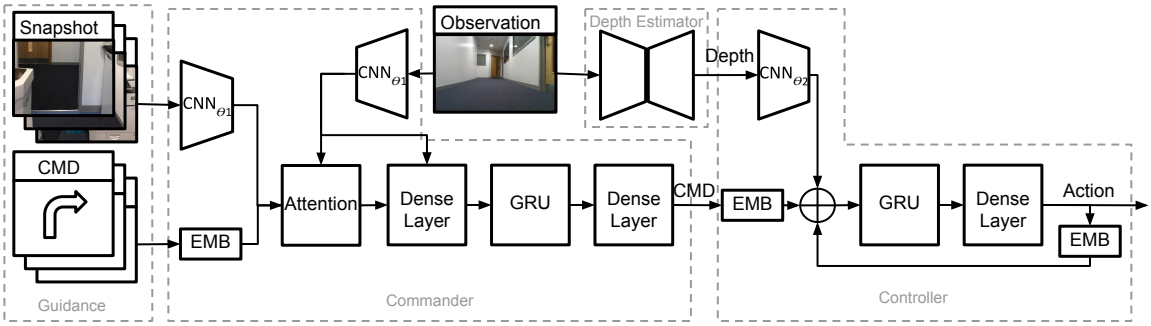


Figure 5.2: The network architecture of SnapNav which consists of two modules. The commander firstly attends to a particular guidance instruction by finding the correct match with the current observation. It then publishes a high level command. The controller then predicts a low level robot action given the command, the estimated depth image and the previous predicted action. Note that commands are represented with the abbreviation "CMD", "EMB" denotes a linear embedding layer, "GRU" indicates the Gated Recurrent Unit [25] and $\bigoplus$ is the concatenation operation.

inputs, followed by a hard attention component and a recurrent policy network.

As illustrated in Fig. 5.2, image inputs to commander are firstly encoded by 5 convolutional layers, whereas commands are input to a linear embedding layer. More specifically, the snapshots $\mathbf{S}_i$ and commands m_i $(i = 1, 2, ...n)$ in the guidance $\mathbf{G}$ and the current observation $\mathbf{O}_t$ are transformed to vectors respectively as $\mathbf{v}_i^S = \text{CNN}_{\theta_1}(\mathbf{S}_i)$, $\mathbf{v}_i^m = \text{EMB}(m_i)$, $\mathbf{v}_t^O = \text{CNN}_{\theta_1}(\mathbf{O}_t)$. Since the snapshots and the current observation are the same type of data, the CNNs for encoding them have the same parameters θ_1.

Given vectorised inputs $\mathbf{v}_i^S$, $\mathbf{v}_i^m$ and $\mathbf{v}_t^O$, the goal is to choose the guidance which contains the most relevant snapshot w.r.t. the current observation. Intuitively, due to the sparsity of the guidance, only the highly related snapshot should be matched for the command strategy at each time step. Thus, hard attention is preferable to soft attention which simply sums over all guidance instructions with different weights.

The hard attention is usually modelled with a discrete stochastic layer which is non-differentiable and thus has to be optimised with gradient estimation methods other than conventional backpropagation [102]. Fortunately, as shown in [51, 86], an alternative is to adopt a generalisation of the max-pooling operator to choose the optimal guidance instruction. This bypasses the non-differentiable problem:

$$(\mathbf{v}_{i*}^S, \mathbf{v}_{i*}^m) = \underset{(\mathbf{v}_i^S, \mathbf{v}_i^m)}{\text{argmax}}[\text{softmax}(-||\mathbf{v}_i^S - \mathbf{v}_t^O||_2)]. \tag{5.2}$$

It results in a sub-differentiable model and can be combined with other gradient-based models. Our experiments later prove that the attention performance can be

improved by a large margin either by combining other gradient estimators such as the REINFORCE algorithm [164] with back-propagation or training with auxiliary tasks i.e. metric learning. Then the attended snapshot vector together with the embedded command is processed by a dense layer and concatenated with the encoded observation. Finally, a recurrent network is used to predict the attended command when the selected snapshot and the current observation are similar enough. If there is low similarity, the default "`Go forward`" command is issued. Therefore the predicted command from the commander can be represented as $c_t = C(\mathbf{O}_t, \mathbf{G}, \mathbf{h}_c)$ where $\mathbf{h}_c$ is the hidden state of the recurrent network.

5.4.2 Controller

Given the command predicted from the commander $c_t \in \{$"`Go straight`", "`Turn right`", "`Turn left`", "`Stop`"$\}$, the last action $\mathbf{a}_{t-1}$ of the controller and the depth image provided by the simulator or a depth prediction network $\mathbf{X}_t = D(\mathbf{O}_t)$, the controller outputs the best action to navigate through the environment, based on its trained policy. Similar to the commander, the raw depth image is encoded with a CNN as $\mathbf{v}_t^X = \text{CNN}_{\theta_2}(\mathbf{X}_t)$, whilst the command is embedded as $\mathbf{v}_t^c = \text{EMB}(c_t)$. They are then used to predict the action by the controller $\mathbf{a}_t = \pi(\mathbf{v}_t^X, \mathbf{v}_t^c, \mathbf{h}_\pi, \mathbf{a}_{t-1})$ where $\mathbf{h}_\pi$ is the recurrent hidden state in the controller and θ_2 represents the parameters of the depth encoding CNN.

The recurrent network is of importance in the controller as it is required to decide precisely when to carry out the command for turning left or right. This is because the high level commands are aligned to maximal visual matches, which may not be precisely aligned to the desired turning point. This decoupling yields higher levels of autonomy, as the low level controller decides when it is best to turn.

5.5 Training

In this section we introduce our training mechanisms for the controller and commander respectively as the commanding strategy and the control policy are learnt separately with different methods and data.

5.5.1 Self-Supervised Commander Training Labels

Rather than relying on manually annotated video, in this section we present a novel, self-supervised technique to create pseudo-labels. Given a raw video, the optical flow

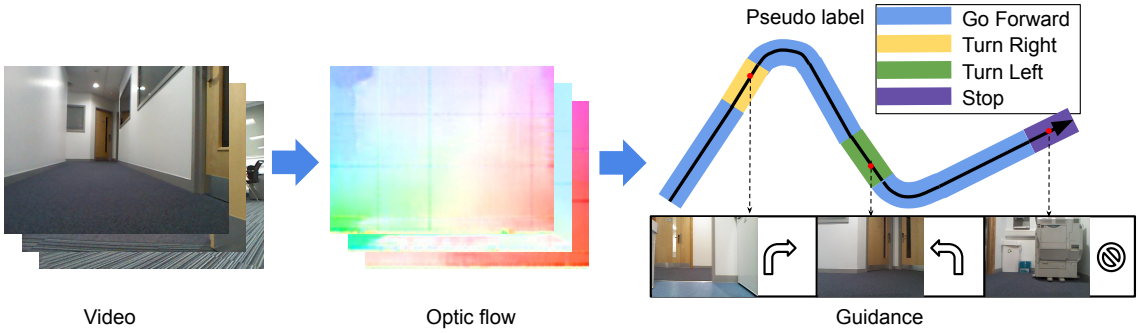


Figure 5.3: The videos are firstly segmented based on the estimated optic flow. Then the pseudo labels of direction varying commands (yellow and green sections) are assigned before the agent actually makes the turn. It is important to be labelled randomly instead of according to the ground-truth. The "`Stop`" command is only labelled at the final part of the video (the purple part) and the other frames are set to "`go forward`" as the default command (blue areas).

between subsequent frames is firstly estimated with FlowNet [30] and then segmented where optical flow is high i.e. at likely turning points. This assumes that images collected along a straight, continuous trajectory are highly similar, whereas images around a corner show high disparity. Next, a snapshot is randomly sampled near the end of each segment and, together with k ($k = 20$ in this chapter) nearest frames, is labelled with a random command drawn from {"`Turn right`", "`Turn left`", "`Stop`"}. It is worth noting that the command of turning left or right before each turning point is not labelled according to the real actions taken by the agent during data collection but assigned randomly. This is the key to preventing the commander network from overfitting to small real-world datasets. Finally, the remaining frames are all labelled with the command "`Go Forward`" as the default prediction of the commander.

Note that only a small dataset with 5k real sequential images are collected for fine-tuning the commander network. Before that, we initialise the network with 100k sequential images collected from *ROS Gazebo* with a standard navigation package *ROS Navigation*[2] automatically.

5.5.2 Training the Commander

Given the labelled data, the commander can be optimised with several different learning signals, which can be used separately or jointly. We discuss the relative merits of each approach below.

[2]http://wiki.ros.org/navigation

Command Loss The basic approach is to minimise the cross-entropy loss between the probability distribution of predicted commands $p \in \{0,1\}^M$ and the pseudo command labels $y \in \{0,1\}^M$ as:

$$Loss_{cmd} = -\frac{1}{T}\sum_{t=0}^{T}\sum_{m=0}^{M} y_t^m \log(p_t^m) \tag{5.3}$$

where T and M represent the length of the sequence and the number of categories of the command given a sequence of data. This relies on the sub-differentiable property of argmax$(\cdot)$ as mentioned in [51] and can be optimised using standard back-propagation algorithms.

Learning Attention Policy with REINFORCE Our novel insight in this chapter is to use the REINFORCE [164] algorithm to estimate gradients for learning a better attention policy which, due to the sub-differentiable argmax function, does not train well. The framework of REINFORCE algorithm usually models the policy learning process as an MDP where the agent is the attention layer in commander (it is independent from the MDP for control policy learning). Note that the attention layer does not contain any trainable parameter, therefore the attention policy $\pi(u_t|\mathbf{G}, \mathbf{O}_t; \theta_1)$ completely relies on the CNN encoder that is parameterised by θ_1. Given the guidance $\mathbf{G}$ and the sequential observations $\mathbf{O}_t$, the attention policy induces a distribution over possible interaction trajectories $s = \mathbf{O}_1, u_1, ..., \mathbf{O}_T, u_T$ where u_t and $\mathbf{O}_t$ are the attended location of snapshots and the observation at time step t. The target is to maximise the reward accumulated along an interaction sequence s as $J(\theta_1) = \mathbb{E}_{p(s;\theta_1)}[\sum_{t=1}^{T} \gamma^{T-t} r_t]$. Note that the probability of the occurrence of trajectory s depends on the attention policy and the reward r_t at each time is proportional to the command prediction accuracy. The gradients are approximated using Monte-Carlo with N learning samples:

$$\begin{aligned} \nabla_{\theta_1} J &= \sum_{t=1}^{T} \mathbb{E}_{p(s_{1:T};\theta_1)}[\nabla_{\theta_1} \log \pi(u_t|\mathbf{G}, \mathbf{O}_t; \theta_1) R_t] \\ &\approx \frac{1}{N}\sum_{n=1}^{N}\sum_{t=1}^{T} \nabla_{\theta_1} \log \pi(u_t^n|\mathbf{G}^n, \mathbf{O}_t^n; \theta_1) R_t^n. \end{aligned} \tag{5.4}$$

Metric Learning As a further alternative to the REINFORCE learning signal, metric learning [129] can also be applied to explicitly improve the image encoding, by forcing the output embeddings to lie within a metric space. This can be considered as an auxiliary task alongside the command prediction.

Figure 5.4: An example of randomised environments in *ROS Gazebo* and its map. An oracle commander is designed accordingly to give the robot correct commands before it reaches the turning and termination point. The red circle highlights a challenging situation where the intersection is not fully constrained with obstacles which can easily confuse the robot.

Similar to [129], the triplet loss is adopted for metric learning. As each video is segmented whilst generating the pseudo command labels, an anchor image can be sampled from one of the segments. Then neighbouring images can be defined as positive (similar) images whilst all the images in other segments are labelled as negative (dissimilar) ones. Hence, after being encoded by the CNN, a triplet, which contains an anchor image vector $\mathbf{v}^a$, k^+ positive image vectors $\mathbf{v}^+$ and k^- negative image vectors $\mathbf{v}^-$, can be randomly generated from each raw image sequence and its loss function is formulated as follows:

$$Loss_{metric} = [\frac{1}{k^+}\sum_{i=1}^{k^+} l_2(\mathbf{v}_i^+ - \mathbf{v}^a) + \sigma - \frac{1}{k^-}\sum_{j=1}^{k^-} l_2(\mathbf{v}_j^- - \mathbf{v}^a)]_+, \tag{5.5}$$

where $[\cdot]_+$ is the hinge function, $l_2(\cdot)$ denotes the Euclidean distance, σ is a margin which is set to 1 and k^+ and k^- are set beneath 20 to ensure a high similarity in positive images and a balanced proportion between positive and negative samples. This loss can be jointly minimised with the command loss through gradient back-propagation, therefore, it is more convenient and simple compared with utilising the REINFORCE algorithm.

5.5.3 Reinforcement Learning for Control Policy

The control policy is purely learnt with DRL in virtual environments. To enhance the generalisation ability of the controller and decrease the possibility for the agent to

memorise the environment, the geometry of the training environment is randomised every 40 episodes and in each episode the desired path for the robot to follow is also randomly generated as shown in Fig. 5.4. Then, for a robust transfer from a simulated robot to the real one, we use a finely modelled Turtlebot2 robot[3] in both the virtual and real world which is equipped with a Microsoft Kinect[4] to capture both depth and RGB images. Note that SnapNav can use either the groundtruth depth provided by the kinect or the estimated one, compatible to the system only with a monocular camera.

We train the control policy with several different algorithms, e.g. DDPG [79], RDPG [50] and DRQN [49]. All these DRL algorithms are firstly trained with the oracle commander which always gives the agent correct commands according to the robot position in the desired path for 1M training steps and later fine-tuned with the noisy predicted commands from the learnt command policy for 0.5M training steps. We show a constantly improved overall performance with different commanders in Table. 5.1. The training is carried out on a single GTX970 GPU and each run takes about 20 hours where the control frequency is 5 Hz and the simulator is 4x times faster than the real world.

5.6 Experiments

We carry out a model ablation study and real world tests to evaluate the optimality of the proposed model for SnapNav, and its generalisation ability to real-world scenarios.

5.6.1 Model Ablation Study in the Virtual World

Each model in the ablation study is tested in random environments similar to Fig. 5.4 with 1000 independent runs and the success rate (SR) of reaching the final destination is used as the metric.

Attention policy learning We compare several different models and learning signals for training the commander. The most basic model does not utilise the attention mechanism and simply sums over all the encoded guidance for command prediction. We term this **AllSum**. Then, since soft attention is reported to have similar performance as **AllSum** in [51], we consider the hard attention model as described in Sec. 5.4.1 which is purely optimised with the command loss learning signal and is

[3]https://www.turtlebot.com/turtlebot2/
[4]https://en.wikipedia.org/wiki/Kinect

Table 5.1: Success Rate (SR%) with different commanders as well as using the DRQN purely initialised with the Oracle commander or further fine-tuned with each commander.

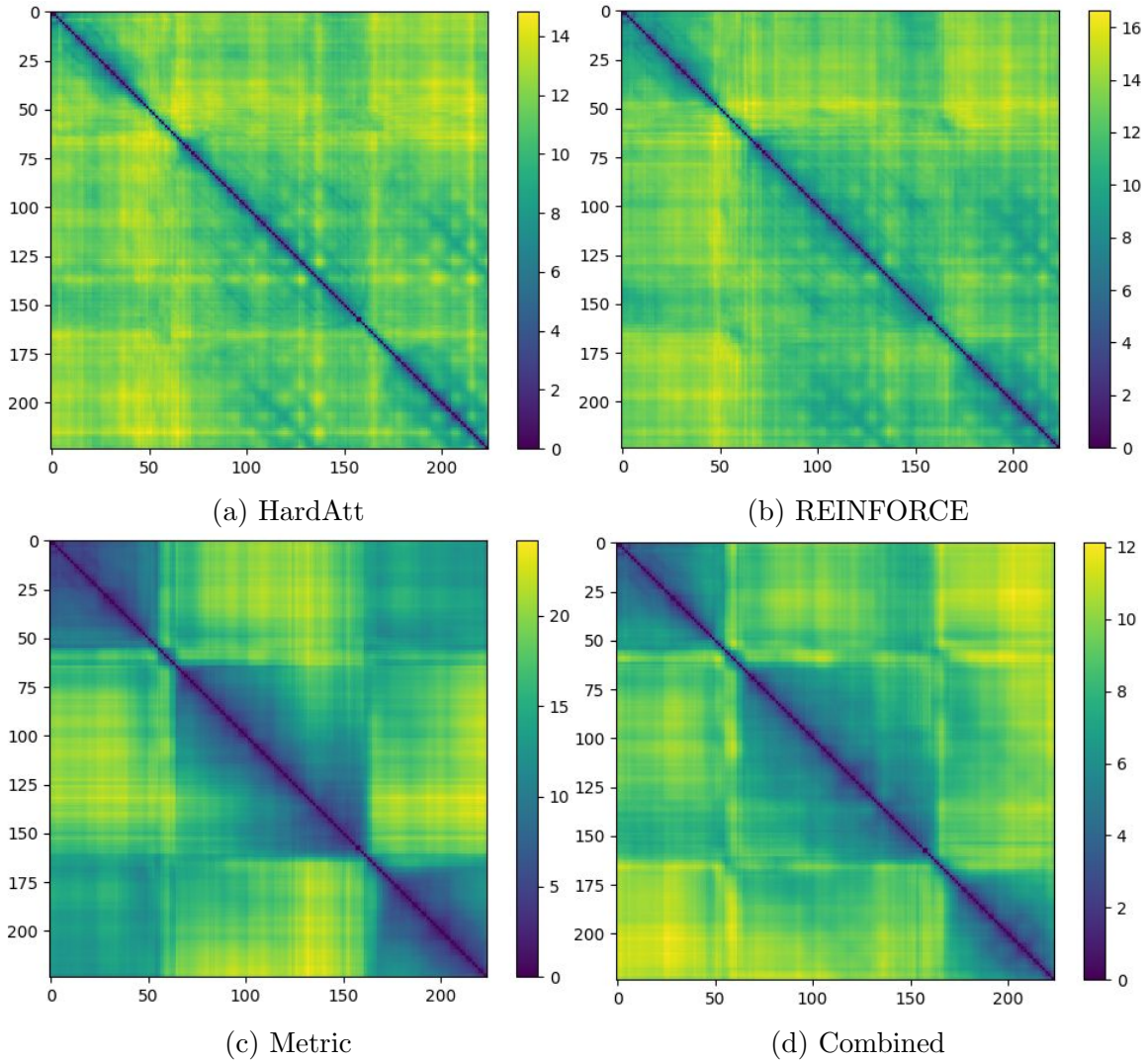
	DRQN	Fine-tuned DRQN
AllSum	50.2%	54.4%
HardAtt	60%	61.6%
REINFORCE	**72.7**%	74.2%
Metric	70.2%	73.6%
Combined	71.2%	**74.9**%
Oracle	85%	-%

termed as **HardAtt**. Next, we add the REINFORCE (**REINFORCE**) or metric learning (**Metric**) learning signal. We also combine three learning signals together (**Combined**).

From the results shown in Table 5.1, we can clearly see that commanders that employ the attention mechanism significantly outperform **AllSum** which proves that attending to a specific snapshot is essential for the command prediction task. Compared with prior art, our introduction of either the REINFORCE algorithm or metric learning yield further, substantial gains.

To better understand the reasons for this difference, Fig. 5.5 illustrates the Euclidean distance matrix of a sampled video after being encoded by the commander. It is shown that compared with **HardAtt** and **REINFORCE**, **Metric** and **Combined** learn an encoder that clearly distinguishes images from different segments more clearly. Thus, this should make it easier to attend to the correct snapshot during navigation. However, according to Table 5.1, the REINFORCE learning signal is more significant at improving the overall performance of the commander compared with metric learning. This may be because that REINFORCE algorithm refines the attention policy directly according to the accuracy of the command prediction, whilst metric learning is a manually introduced bias in the encoding space which is not entirely related to the task of navigation.

Control policy learning Selecting a suitable DRL approach to train the controller is key to the entire system performance. From the learning curves shown in Fig. 5.6, we firstly prove the necessity of introducing the recurrent network into the controller model by comparing **DDPG** and **RDPG** where the latter is the recurrent version of the former. Because the command is assigned to the controller before the robot



(a) HardAtt (b) REINFORCE

(c) Metric (d) Combined

Figure 5.5: The Euclidean distance matrix of the encoded images in a sampled video.

reaches the turning point, the recurrent network becomes important for memorising recent commands.

Then we investigate the improvements brought by restricting the action and search space by exploiting a value-based approach **DRQN**. The action space is discretised into three options, i.e. going straight forward with the maximum linear velocity and turning left or right with the maximum angular velocity and a slower linear velocity. Correspondingly, the Q-network only is required to estimate the Q-values of these options given the current observation and the hidden state of the recurrent network. This substantially narrows down the search space and boosts the learning efficiency.

Table 5.2 looks deeper into the performance of the learnt policies. **DDPG** terminates most episodes with collisions as it attempts to maintain the expected total

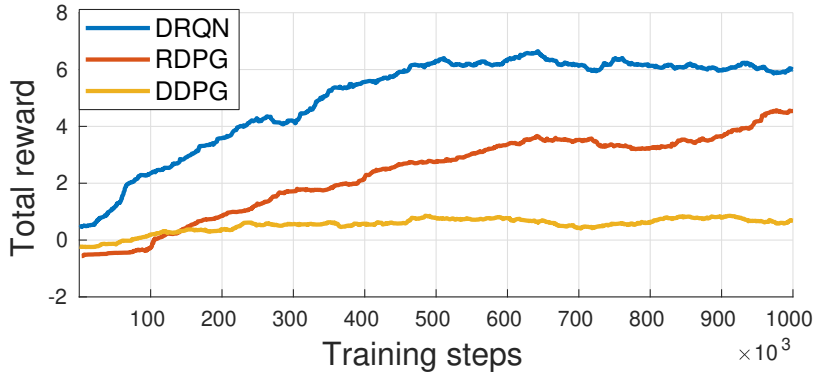


Figure 5.6: The smoothed learning curves of DRQN, RDPG and DDPG respectively. Each algorithm is trained with 1 million steps in *ROS Gazebo* with the oracle commander.

Table 5.2: Distribution of rewards over 1000 testing episodes for different controllers. Note that **Success** means the successful arrival of the final destination while **Time-Out** and **Crash** denote the situations where the robot cannot reach the destination within 300 steps or collides with an obstacle respectively.

	Success	TimeOut	Crash
DDPG	3.4%	31.8%	64.8%
RDPG	60.9%	36.6%	**2.5**%
DRQN	**85**%	**10.3**%	4.7%

reward by minimising the probability of moving in an opposite directions to the destination, as this incurs an immediate negative reward. In contrast, its recurrent version (**RDPG**) achieves a much higher success rate due to the possibility of memorising the historic commands and heading to the correct directions accordingly at intersections. However the large exploration space for continuous action hinders the network from learning a stable and robust turning behaviour in highly randomised junctions, especially in the intersections with uncommon geometries as highlighted in Fig. 5.4. Finally, **DRQN** which explores in a discretised action space shows significant gains.

5.6.2 Real-World Tests

In addition to the extensive experiments in virtual environments, we also evaluate the learnt policy in real-world scenarios, to demonstrate its utility in real robotic scenarios. To make the commander generalise better to a larger amount of guidance while testing in the real world, we constrain the attention policy with a manually tuned threshold for switching to the next snapshot in guidance when it is similar

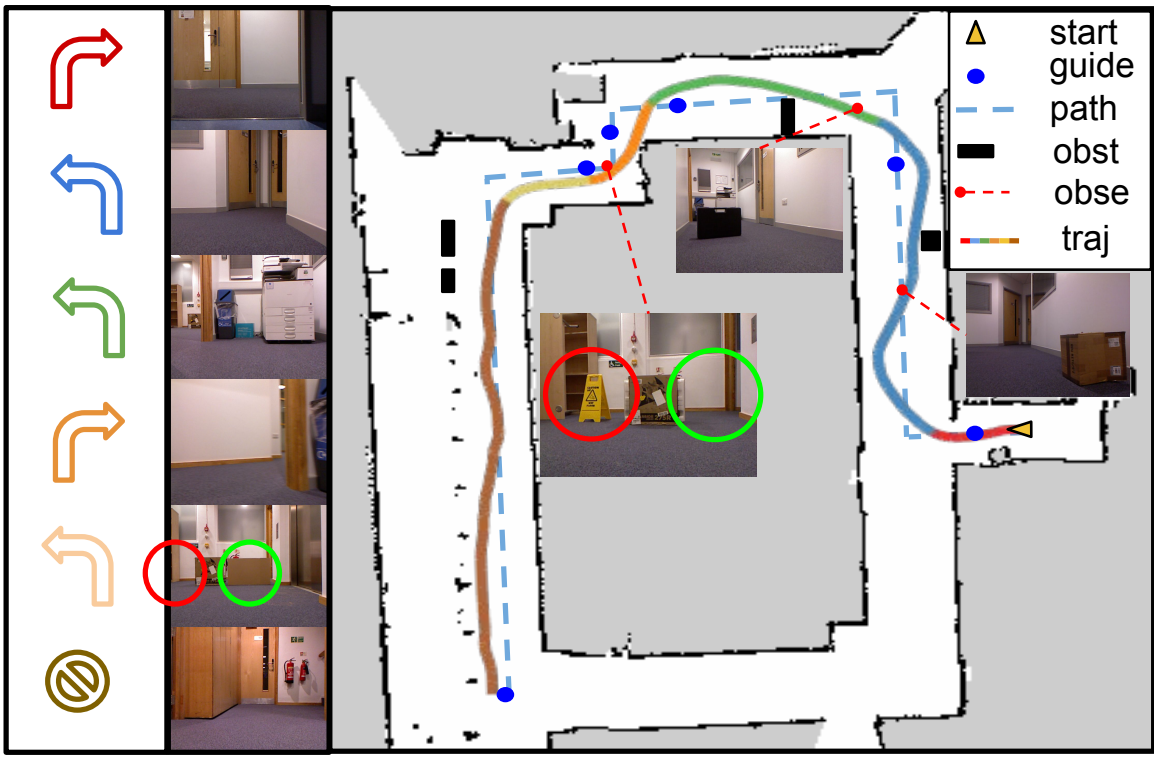


Figure 5.7: An example real world test. The legends from top to bottom are: start position, guidance recording position, path in demonstrating phase, random obstacles in testing phase, robot observations and robot trajectory in testing. The attention location is illustrated by different colours in the robot trajectory which corresponds to the colour of actions in the guidance. The red and green circles emphasises the changing visual appearance and geometry of the environment between demonstration and testing.

enough to the current observation. Fig. 5.7 demonstrates an example real-world test which qualitatively examines the obstacle avoidance ability and the robustness against the changing environments of SnapNav.

Furthermore, two baseline models are proposed for quantitative comparison as shown in Table 5.3. **SimOnly** implements a similar model as [51] with additional depth observation and is purely trained in the simulator through DRL. The other one is the supervised variant of the deep **VT&R** model in [73] based on the limited real-world data. The **SimOnly** baseline is only trained in simulation which shows a degraded performance compared to **SnapNav** and proves that directly transferring the command and control policy learnt in a single network is non-trivial. **VT&R** is a densely guided counterpart to **SnapNav** that is directly trained with the supervision of real world data. In our experiments, it possess the worst performance where it never learns to avoid any new obstacles. It verifies our hypothesis that the limited

Table 5.3: Evaluations in real world environments. The metrics used are: success rate (**SR**), the average travelled distance before collisions or reaching the destination (**Dist**) and the number of images used as guidance (**ImgNum**). Each model is tested with 5 independent runs.

	SR	Dist(m)	ImgNum
SnapNav	**80%**	**24.29**	**6**
SimOnly	40%	20.19	**6**
VT&R	20%	18.32	326

training data can not yield a robust navigation policy and also proves the high degree of autonomy brought by our two-level hierarchy.

5.7 Conclusion

In this chapter we have proposed a practical solution for global mapless navigation. The fundamental concept is similar to the recent StreetNav [51] which provides a few snapshots paired with directional commands at key waypoints as the navigational guidance. However, StreetNav does not transfer well to new scenarios, due to the tight coupling between the training data and the learnt policy, and has only been tested in simulation. Our proposed technique makes this system practical by using a two-level hierarchy, decoupling perception from actuation. Through extensive experiments, we show that SnapNav yields significant improvements over StreetNav, in particular:

- The attention policy is significantly improved by our novel self-supervised learning mechanism which only needs a very small amount of real-world data to train successfully.
- The navigation system is successfully adapted to a real-time robot control task and can be straightforwardly deployed on physical robots due to its two-level hierarchy.
- The improved loss metric is key to robust attention on the correct visual waypoint, showing good immunity to scene changes.

We believe this is the first work to demonstrate practical, global mapless navigation using sparse visual guidance in indoor environments and is a step towards increased robotic autonomy.

Chapter 6

Conclusion and Future Work

This thesis has investigated multiple tasks for mapless navigation for mobile robots. Practical solutions are proposed to address some challenging problems for successfully applying DRL approaches in real-world robotic problems.

Due to the infeasibility of training in the real world, the first question is how to transfer the learnt policy from the simulated world to the real world, the success of which heavily depends on the realism of the simulator. Due to the absence of a virtual environment which simultaneously captures the kinetics of the robot and renders camera observations with high visual fidelity, directly deploying the trained policy to physical robots is challenging. In Chapter 3 we explore this challenge by solving the task of **monocular visual obstacle avoidance**. We utilise a simulator with a high performance physics engine but simplified camera observations, which highlights that transferring the perception is our key problem. The **reality gap**, in our case, is mitigated by projecting the robot observations from the visual appearance domain into the complementary depth domain which has several advantages. The first one is the simplicity of implementation. It only requires the use of an off-the-shelf depth estimator, fine-tuned with a small amount of paired data. Conversely, domain randomisation needs a carefully selected noise distribution and domain adaptation must learn the adaptor network from scratch for different simulated environments. Another advantage is that the geometric information provided in the depth domain largely facilitates the learning of obstacle avoidance, speeding up the training process. Thus, in this chapter, we demonstrate how to bridge the reality gap through domain simplification.

Chapter 4 explores the question of how to **learn efficiently** in a **local navigation** task. We propose a novel framework to incorporate navigation knowledge from other sub-optimal but simple controllers into the training of an off-policy DRL approach as guidance. This is achieved by learning a switching policy alongside the navigation

task to sample actions from a better controller depending on the specific situation. Hence the underlying DRL approach can initially learn from the prior controller guidance and subsequently improve its own policy through self-exploration once it has surpassed the performance of the prior controllers. We separately examine a deterministic switching policy and a stochastic counterpart where the latter is proved to be superior as it is naturally more robust to local minima. The main advantage of the proposed framework compared to imitation learning is that it not only mimics the behaviour of the controllers but also learns to outperform them. In some ways, our method can be regarded as learning through inspiration, rather than imitation, allowing the student to surpass the teacher. We note that both simple and expert prior controllers can contribute to increasing the efficiency of the training process.

The solution for the key overarching question, **global mapless visual navigation**, is described in Chapter 5. The proposed system can navigate with only sparse visual reference and directional guidance which can easily **generalise the policy to unseen scenarios** with a minimal description of the environment. Such a human-like behaviour endows the robot with a higher autonomous ability compared to VT&R. Our novel formulation of the system in a two-layer hierarchy allows for task decomposition, which is a key step towards making the system practical and realisable. This also makes training more efficient compared with trying to learn the entire task with a single policy network. The hierarchy facilitates exploitation of training data from different domains and sources, i.e. appearance or depth observations, simulated or real perception and action, allowing the learnt policy to be straightforwardly deployed on real robots.

In the future, the work in this thesis can be further explored and expanded on in the following aspects:

1. Visual obstacle avoidance in dynamic environments: At Chapter 3, we only investigated navigation in relatively static environments with no fast moving objects. Dynamic objects greatly increase the difficulty of monocular visual obstacle avoidance because it is hard for a model-free approach to predict future position of these moving objects. However our proposed depth based perception network could be integrated with an object tracking module to model the the relative velocity between the robot and all the observable moving objects. Through combination with an attention mechanism, this information may effectively help the DRL agent to learn a robust obstacle avoidance policy in highly dynamic environments.

2. Hierarchical reinforcement learning: The proposed framework in Chapter 4 can simultaneously learn the navigation policy using the underlying DRL approach and a switching policy through REINFORCE learning signal to select among different navigation controllers with a shared reward function and replay memory. However, from another perspective, we can regard the switching network as a high level commander and all the controllers as low level actuators and assign to them separate replay memories and decomposed reward functions. This transformation enables the framework to deal with more sophisticated tasks where each actuator is only responsible for learning a single and easy function according to the decomposed reward and based on the samples recorded in their own replay memories. Then, the commander learns to dispatch a specific actuator under different situations according to the overall reward function.

3. Combining behaviour based robotics with DRL: Since the tasks and algorithms in Chapter 3 and 4 are relevant to behaviour based robotics, it will be interesting to combine this area with DRL approaches. It will develop more intelligent behaviour based algorithms. For instance, for algorithms based on multiple controllers that have different simple behaviours respectively, a DRL agent can learn the optimal combination strategy of all behaviours according to sensory inputs in different situations. Furthermore, by utilising recurrent neural networks as the memory, it is also possible to expand behaviour based agent to carry out long term navigation tasks more easily.

4. Multi-robot navigation: In Chapter 5, the proposed mapless navigation system naturally yields an efficient means of communication to share navigation experiences, rather than transmitting detailed point-cloud maps or occupancy grids. Hence it is promising to establish a multi-robot navigation system within this framework of sparse guidance. Within this area, a number of questions come immediately to mind, namely: 1) How to retrieve the experience for each robot given a published query? 2) How to efficiently propagate the limited knowledge of each agent through the entire network? 3) How to integrate all the information acquired from different agents to plan the optimal path to the destination? These can all be interesting questions for further discussions.

In summary, over the next decade, mobile robots are likely to play increasingly important and ubiquitous roles in daily life e.g. in the arenas of housekeeping, manufacturing, transportation and security. In such a situation, it is impractical and

inefficient to manually define and assign each robot with domain-specific knowledge and experience. Robots will need to interact within a common framework by sharing knowledge, allowing multiple agents to explore potential solutions without any human input or labelling, moving towards true cognition as opposed to simple pattern recognition. This thesis, which proposes DRL based practical robot mapless navigation solutions in real world scenarios, is a very first step towards that end.

Bibliography

[1] Ibraheem Alhashim and Peter Wonka. High quality monocular depth estimation via transfer learning. *arXiv preprint arXiv:1812.11941*, 2018.

[2] Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, OpenAI Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay. In *Advances in Neural Information Processing Systems*, pages 5048–5058, 2017.

[3] Antonis A Argyros, Kostas E Bekris, Stelios C Orphanoudakis, and Lydia E Kavraki. Robot homing by exploiting panoramic vision. *Autonomous Robots*, 19(1):7–25, 2005.

[4] Ronald C Arkin. Motor schemabased mobile robot navigation. *The International journal of robotics research*, 8(4):92–112, 1989.

[5] Karl Johan Åström and Tore Hägglund. *PID controllers: theory, design, and tuning*, volume 2. Instrument society of America Research Triangle Park, NC, 1995.

[6] Martin Azizyan, Ionut Constandache, and Romit Roy Choudhury. Surroundsense: mobile phone localization via ambience fingerprinting. In *Proceedings of the 15th annual international conference on Mobile computing and networking*, pages 261–272. ACM, 2009.

[7] Tim Bailey, Juan Nieto, Jose Guivant, Michael Stevens, and Eduardo Nebot. Consistency of the ekf-slam algorithm. In *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3562–3568. IEEE, 2006.

[8] Andrew G Barto, Richard S Sutton, and Charles W Anderson. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE transactions on systems, man, and cybernetics*, (5):834–846, 1983.

[9] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. Surf: Speeded up robust features. In *European conference on computer vision*, pages 404–417. Springer, 2006.

[10] Richard Bellman. A markovian decision process. *Journal of Mathematics and Mechanics*, 6(5):679–684, 1957.

[11] Richard Bellman. *Dynamic programming*. Courier Corporation, 2013.

[12] Richard Bellman and Robert E Kalaba. *Dynamic programming and modern control theory*, volume 81. Citeseer, 1965.

[13] Cooper Bills, Joyce Chen, and Ashutosh Saxena. Autonomous mav flight in indoor environments using single image perspective cues. In *ICRA*, pages 5776–5783. IEEE, 2011.

[14] Johann Borenstein and Yoram Koren. Obstacle avoidance with ultrasonic sensors. *IEEE Journal on Robotics and Automation*, 4(2):213–218, 1988.

[15] Johann Borenstein and Yoram Koren. Real-time obstacle avoidance for fast mobile robots. *IEEE Transactions on systems, Man, and Cybernetics*, 19(5):1179–1187, 1989.

[16] Johann Borenstein and Yoram Koren. The vector field histogram-fast obstacle avoidance for mobile robots. *IEEE transactions on robotics and automation*, 7(3):278–288, 1991.

[17] Konstantinos Bousmalis, Alex Irpan, Paul Wohlhart, Yunfei Bai, Matthew Kelcey, Mrinal Kalakrishnan, Laura Downs, Julian Ibarz, Peter Pastor, Kurt Konolige, et al. Using simulation and domain adaptation to improve efficiency of deep robotic grasping. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4243–4250. IEEE, 2018.

[18] Oliver Brock and Oussama Khatib. High-speed navigation using the global dynamic window approach. In *Proceedings 1999 IEEE International Conference on Robotics and Automation (Cat. No. 99CH36288C)*, volume 1, pages 341–346. IEEE, 1999.

[19] Rodney Brooks. A robust layered control system for a mobile robot. *IEEE journal on robotics and automation*, 2(1):14–23, 1986.

[20] Jake Bruce, Niko Sünderhauf, Piotr Mirowski, Raia Hadsell, and Michael Milford. Learning deployable navigation policies at kilometer scale from a single traversal. *arXiv preprint arXiv:1807.05211*, 2018.

[21] Ted Camus, David Coombs, Martin Herman, and Tsai-Hong Hong. Real-time single-workstation obstacle avoidance using only wide-field flow divergence. In *Proceedings of 13th International Conference on Pattern Recognition*, volume 3, pages 323–330. IEEE, 1996.

[22] John Canny. A computational approach to edge detection. In *Readings in computer vision*, pages 184–203. Elsevier, 1987.

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

[24] David L Chen and Raymond J Mooney. Learning to interpret natural language navigation instructions from observations. In *Twenty-Fifth AAAI Conference on Artificial Intelligence*, 2011.

[25] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

[26] Joao Cunha, Eurico Pedrosa, Cristovao Cruz, António JR Neves, and Nuno Lau. Using a depth camera for indoor robot localization and navigation. *DETI/IEETA-University of Aveiro, Portugal*, 2011.

[27] Bruno Damas and José Santos-Victor. Avoiding moving obstacles: the forbidden velocity map. In *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4393–4398. IEEE, 2009.

[28] Nguyen Xuan Dao, Bum-Jae You, and Sang-Rok Oh. Visual navigation for indoor mobile robots using a single camera. In *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1992–1997. IEEE, 2005.

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

[30] Alexey Dosovitskiy, Philipp Fischer, Eddy Ilg, Philip Hausser, Caner Hazirbas, Vladimir Golkov, Patrick Van Der Smagt, Daniel Cremers, and Thomas Brox. Flownet: Learning optical flow with convolutional networks. In *Proceedings of the IEEE international conference on computer vision*, pages 2758–2766, 2015.

[31] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. Carla: An open urban driving simulator. *arXiv preprint arXiv:1711.03938*, 2017.

[32] Yan Duan, Marcin Andrychowicz, Bradly Stadie, Jonathan Ho, Jonas Schneider, Ilya Sutskever, Pieter Abbeel, and Wojciech Zaremba. One-shot imitation learning. *arXiv preprint arXiv:1703.07326*, 2017.

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

[34] Alberto Elfes. Sonar-based real-world mapping and navigation. *IEEE Journal on Robotics and Automation*, 3(3):249–265, 1987.

[35] CDR HR Everett. Survey of collision avoidance and ranging sensors for mobile robots. *RAS*, 5(1):5–67, 1989.

[36] Dieter Fox, Wolfram Burgard, and Sebastian Thrun. The dynamic window approach to collision avoidance. *IEEE Robotics & Automation Magazine*, 4(1):23–33, 1997.

[37] Paul Furgale and Timothy D Barfoot. Visual teach and repeat for long-range rover autonomy. *Journal of Field Robotics*, 27(5):534–560, 2010.

[38] Dhiraj Gandhi, Lerrel Pinto, and Abhinav Gupta. Learning to fly by crashing. *arXiv:1704.05588*, 2017.

[39] Brian P Gerkey and Kurt Konolige. Planning and control in unstructured terrain. In *ICRA Workshop on Path Planning on Costmaps*, 2008.

[40] James J Gibson. The perception of the visual world. 1950.

[41] Georges Giralt, Ralph Sobek, and Raja Chatila. A multi-level planning and navigation system for a mobile robot: a first approach to hilare. In *Proceedings of the 6th international joint conference on Artificial intelligence-Volume 1*, pages 335–337. Morgan Kaufmann Publishers Inc., 1979.

[42] Alessandro Giusti, Jérôme Guzzi, Dan C Cireşan, Fang-Lin He, Juan P Rodríguez, Flavio Fontana, Matthias Faessler, Christian Forster, Jürgen Schmidhuber, Gianni Di Caro, et al. A machine learning approach to visual perception of forest trails for mobile robots. *RA Letters*, 1(2):661–667, 2016.

[43] Clément Godard, Oisin Mac Aodha, and Gabriel J Brostow. Unsupervised monocular depth estimation with left-right consistency. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 270–279, 2017.

[44] I. Grondman, L. Busoniu, G. A. D. Lopes, and R. Babuska. A survey of actor-critic reinforcement learning: Standard and natural policy gradients. *IEEE Trans. Syst., Man, Cybern. C*, 42(6):1291–1307, Nov 2012.

[45] Shixiang Gu, Ethan Holly, Timothy Lillicrap, and Sergey Levine. Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates. In *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, pages 3389–3396. IEEE, 2017.

[46] Saurabh Gupta, James Davidson, Sergey Levine, Rahul Sukthankar, and Jitendra Malik. Cognitive mapping and planning for visual navigation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2616–2625, 2017.

[47] Christopher G Harris, Mike Stephens, et al. A combined corner and edge detector. In *Alvey vision conference*, volume 15, pages 10–5244. Citeseer, 1988.

[48] Hado V Hasselt. Double q-learning. In *NPIS*, pages 2613–2621, 2010.

[49] Matthew Hausknecht and Peter Stone. Deep recurrent q-learning for partially observable mdps. In *2015 AAAI Fall Symposium Series*, 2015.

[50] Nicolas Heess, Jonathan J Hunt, Timothy P Lillicrap, and David Silver. Memory-based control with recurrent neural networks. *arXiv preprint arXiv:1512.04455*, 2015.

[51] Karl Moritz Hermann, Mateusz Malinowski, Piotr Mirowski, Andras Banki-Horvath, Keith Anderson, and Raia Hadsell. Learning to follow directions in street view. *arXiv preprint arXiv:1903.00401*, 2019.

[52] Matteo Hessel, Hado van Hasselt, Joseph Modayil, and David Silver. On inductive biases in deep reinforcement learning. *CoRR*, abs/1907.02908, 2019.

[53] Noriaki Hirose, Fei Xia, Roberto Martín-Martín, Amir Sadeghian, and Silvio Savarese. Deep visual mpc-policy learning for navigation. *arXiv preprint arXiv:1903.02749*, 2019.

[54] Ian Horswill. Visual collision avoidance by segmentation. In *Intelligent Robots and Systems*, pages 87–99. Elsevier, 1995.

[55] Rein Houthooft, Xi Chen, Yan Duan, John Schulman, Filip De Turck, and Pieter Abbeel. Vime: Variational information maximizing exploration. In *Advances in Neural Information Processing Systems*, pages 1109–1117, 2016.

[56] Ronald A Howard. Dynamic programming and markov processes. 1960.

[57] Tadanobu Inoue, Subhajit Choudhury, Giovanni De Magistris, and Sakyasingha Dasgupta. Transfer learning from synthetic to real images using variational autoencoders for precise position detection. In *2018 25th IEEE International Conference on Image Processing (ICIP)*, pages 2725–2729. IEEE, 2018.

[58] Rudolf Emil Kalman et al. Contributions to the theory of optimal control. *Bol. soc. mat. mexicana*, 5(2):102–119, 1960.

[59] Mohammad Khan, Shakir Mohamed, Benjamin Marlin, and Kevin Murphy. A stick-breaking likelihood for categorical data analysis with latent gaussian models. In *Artificial Intelligence and Statistics*, pages 610–618, 2012.

[60] Oussama Khatib. Real-time obstacle avoidance for manipulators and mobile robots. In *Autonomous robot vehicles*, pages 396–404. Springer, 1986.

[61] Dong Ki Kim and Tsuhan Chen. Deep neural network for real-time autonomous indoor navigation. *arXiv:1511.04668*, 2015.

[62] M. J. Kim. Thompson sampling for stochastic control: The finite parameter case. *IEEE Trans. Automat. Contr.*, 62(12):6415–6422, Dec 2017.

[63] Young-geun Kim and Hakil Kim. Layered ground floor detection for vision-based mobile robot navigation. In *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA'04. 2004*, volume 1, pages 13–18. IEEE, 2004.

[64] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv:1412.6980*, 2014.

[65] A Harry Klopf. Brain function and adaptive systems: a heterostatic theory. Technical report, AIR FORCE CAMBRIDGE RESEARCH LABS HANSCOM AFB MA, 1972.

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

[67] Vijay R Konda and John N Tsitsiklis. Actor-critic algorithms. In *Advances in neural information processing systems*, pages 1008–1014, 2000.

[68] Manikanta Kotaru, Kiran Joshi, Dinesh Bharadia, and Sachin Katti. Spotfi: Decimeter level localization using wifi. In *ACM SIGCOMM Computer Communication Review*, volume 45, pages 269–282. ACM, 2015.

[69] Bruce Krogh. A generalized potential field approach to obstacle avoidance control. In *Proc. SME Conf. on Robotics Research: The Next Five Years and Beyond, Bethlehem, PA, 1984*, pages 11–22, 1984.

[70] Bruce Krogh and Charles Thorpe. Integrated path planning and dynamic steering control for autonomous vehicles. In *Proceedings. 1986 IEEE International Conference on Robotics and Automation*, volume 3, pages 1664–1669. IEEE, 1986.

[71] Roman Kuc and Billur Barshan. Navigating vehicles through an unstructured environment with sonar. In *Proceedings, 1989 International Conference on Robotics and Automation*, pages 1422–1426. IEEE, 1989.

[72] Ashish Kumar, Saurabh Gupta, David Fouhey, Sergey Levine, and Jitendra Malik. Visual memory for robust path following. In *Advances in Neural Information Processing Systems*, pages 765–774, 2018.

[73] Ashish Kumar, Saurabh Gupta, and Jitendra Malik. Learning navigation subroutines by watching videos. *arXiv preprint arXiv:1905.12612*, 2019.

[74] Young D Kwon and Jin S Lee. A stochastic map building method for mobile robot using 2-d laser range finder. *Autonomous Robots*, 7(2):187–200, 1999.

[75] Iro Laina, Christian Rupprecht, Vasileios Belagiannis, Federico Tombari, and Nassir Navab. Deeper depth prediction with fully convolutional residual networks. In *3DV*, pages 239–248. IEEE, 2016.

[76] John J Leonard, Hugh F Durrant-Whyte, and Ingemar J Cox. Dynamic map building for an autonomous mobile robot. *The International Journal of Robotics Research*, 11(4):286–298, 1992.

[77] Matteo Leonetti, Luca Iocchi, and Peter Stone. A synthesis of automated planning and reinforcement learning for efficient, robust decision-making. *Artificial Intelligence*, 241:103–130, 2016.

[78] Bojian Liang and Nick Pears. Visual navigation using planar homographies. In *Proceedings 2002 IEEE International Conference on Robotics and Automation (Cat. No. 02CH37292)*, volume 1, pages 205–210. IEEE, 2002.

[79] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[80] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. 2016.

[81] Kai Lingemann, Andreas Nüchter, Joachim Hertzberg, and Hartmut Surmann. High-speed laser localization for mobile robots. *Robotics and autonomous systems*, 51(4):275–296, 2005.

[82] Kai Lingemann, Hartmut Surmann, Andreas Nuchter, and Joachim Hertzberg. Indoor and outdoor localization for fast mobile robots. In *Intelligent Robots and Systems, 2004.(IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on*, volume 3, pages 2185–2190. IEEE, 2004.

[83] Manolis IA Lourakis and Stelios C Orphanoudakis. Visual detection of obstacles assuming a locally planar ground. In *Asian conference on computer vision*, pages 527–534. Springer, 1998.

[84] Tomas Lozano-Perez. Spatial planning: A configuration space approach. In *Autonomous robot vehicles*, pages 259–271. Springer, 1990.

[85] Tomás Lozano-Pérez and Michael A Wesley. An algorithm for planning collision-free paths among polyhedral obstacles. *Communications of the ACM*, 22(10):560–570, 1979.

[86] Mateusz Malinowski, Carl Doersch, Adam Santoro, and Peter Battaglia. Learning visual question answering by bootstrapping hard attention. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 3–20, 2018.

[87] Maja J Mataric. Behaviour-based control: Examples from navigation, learning, and group behaviour. *Journal of Experimental & Theoretical Artificial Intelligence*, 9(2-3):323–336, 1997.

[88] Yoshio Matsumoto, Kazunori Ikeda, Masayuki Inaba, and Hirochika Inoue. Visual navigation using omnidirectional view sequence. In *Proceedings 1999 IEEE/RSJ International Conference on Intelligent Robots and Systems. Human and Environment Friendly Robots with High Intelligence and Emotional Quotients (Cat. No. 99CH36289)*, volume 1, pages 317–322. IEEE, 1999.

[89] Yoshio Matsumoto, Masayuki Inaba, and Hirochika Inoue. Visual navigation using view-sequenced route representation. In *Proceedings of IEEE International conference on Robotics and Automation*, volume 1, pages 83–88. IEEE, 1996.

[90] Chris McCarthy and Nick Bames. Performance of optical flow techniques for indoor navigation with a mobile robot. In *ICRA*, volume 5, pages 5093–5098. IEEE, 2004.

[91] Hao Mei, Yantao Tian, and Linan Zu. A hybrid ant colony optimization algorithm for path planning of robot in dynamic environment. *International Journal of Information Technology*, 12(3):78–88, 2006.

[92] Pierre Merriaux, Yohan Dupuis, Rémi Boutteau, Pascal Vasseur, and Xavier Savatier. A study of vicon system positioning performance. *Sensors*, 17(7):1591, 2017.

[93] Yishu Miao, Edward Grefenstette, and Phil Blunsom. Discovering discrete latent topics with neural variational inference. In *International Conference on Machine Learning*, pages 2410–2419, 2017.

[94] Jeff Michels, Ashutosh Saxena, and Andrew Y Ng. High speed obstacle avoidance using monocular vision and reinforcement learning. In *ICML*, pages 593–600. ACM, 2005.

[95] Krystian Mikolajczyk and Cordelia Schmid. Scale & affine invariant interest point detectors. *International journal of computer vision*, 60(1):63–86, 2004.

[96] Hua-Qing Min, Jin-Hui Zhu, and Xi-Jing Zheng. Obstacle avoidance with multi-objective optimization by pso in dynamic environment. In *2005 International Conference on Machine Learning and Cybernetics*, volume 5, pages 2950–2956. IEEE, 2005.

[97] Marvin Minsky. Steps toward artificial intelligence. *Proceedings of the IRE*, 49(1):8–30, 1961.

[98] Piotr Mirowski, Matt Grimes, Mateusz Malinowski, Karl Moritz Hermann, Keith Anderson, Denis Teplyashin, Karen Simonyan, Andrew Zisserman, Raia Hadsell, et al. Learning to navigate in cities without a map. In *Advances in Neural Information Processing Systems*, pages 2419–2430, 2018.

[99] Piotr Mirowski, Razvan Pascanu, Fabio Viola, Hubert Soyer, Andrew J Ballard, Andrea Banino, Misha Denil, Ross Goroshin, Laurent Sifre, Koray Kavukcuoglu, et al. Learning to navigate in complex environments. *arXiv preprint arXiv:1611.03673*, 2016.

[100] Piotr Mirowski, Razvan Pascanu, Fabio Viola, Hubert Soyer, and et.al. Learning to navigate in complex environments. In *ICLR*, 2017.

[101] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937, 2016.

[102] Volodymyr Mnih, Nicolas Heess, Alex Graves, et al. Recurrent models of visual attention. In *Advances in neural information processing systems*, pages 2204–2212, 2014.

[103] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[104] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

[105] Michael Montemerlo, Sebastian Thrun, Daphne Koller, Ben Wegbreit, et al. Fastslam: A factored solution to the simultaneous localization and mapping problem. *Aaai/iaai*, 593598, 2002.

[106] Hadi Moradi, Jongmoo Choi, Eunyoung Kim, and Sukhan Lee. A real-time wall detection method for indoor environments. In *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4551–4557. IEEE, 2006.

[107] Hans Moravec and Alberto Elfes. High resolution maps from wide angle sonar. In *Proceedings. 1985 IEEE international conference on robotics and automation*, volume 2, pages 116–121. IEEE, 1985.

[108] Melissa Mozifian, Juan Camilo Gamboa Higuera, David Meger, and Gregory Dudek. Learning domain randomization distributions for transfer of locomotion policies. *arXiv preprint arXiv:1906.00410*, 2019.

[109] Raul Mur-Artal, Jose Maria Martinez Montiel, and Juan D Tardos. Orb-slam: a versatile and accurate monocular slam system. *IEEE Transactions on Robotics*, 31(5):1147–1163, 2015.

[110] Pushmeet Kohli Nathan Silberman, Derek Hoiem and Rob Fergus. Indoor segmentation and support inference from rgbd images. In *ECCV*, 2012.

[111] Nils J Nilsson. Shakey the robot. Technical report, SRI INTERNATIONAL MENLO PARK CA, 1984.

[112] Deepak Pathak, Parsa Mahmoudieh, Guanghao Luo, Pulkit Agrawal, Dian Chen, Yide Shentu, Evan Shelhamer, Jitendra Malik, Alexei A Efros, and Trevor Darrell. Zero-shot visual imitation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 2050–2053, 2018.

[113] Nick Pears and Bojian Liang. Ground plane segmentation for mobile robot visual navigation. In *Proceedings 2001 IEEE/RSJ International Conference on Intelligent Robots and Systems. Expanding the Societal Role of Robotics in the*

the Next Millennium (Cat. No. 01CH37180), volume 3, pages 1513–1518. IEEE, 2001.

[114] Jan Peters and Stefan Schaal. Policy gradient methods for robotics. In *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2219–2225. IEEE, 2006.

[115] M. Pfeiffer, S. Shukla, M. Turchetta, C. Cadena, A. Krause, R. Siegwart, and J. Nieto. Reinforced imitation: Sample efficient deep reinforcement learning for mapless navigation by leveraging prior demonstrations. *IEEE Robotics and Automation Letters*, 3(4):4423–4430, Oct 2018.

[116] B. Piot, M. Geist, and O. Pietquin. Bridging the gap between imitation learning and inverse reinforcement learning. *IEEE Trans. Neural Netw.*, 28(8):1814–1826, Aug 2017.

[117] Yuan-Qing Qin, De-Bao Sun, Ning Li, and Yi-Gang Cen. 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)*, volume 4, pages 2473–2478. IEEE, 2004.

[118] Purushothaman Raja and Sivagurunathan Pugazhenthi. Optimal path planning of mobile robots: A review. *International journal of physical sciences*, 7(9):1314–1320, 2012.

[119] Anthony Remazeilles, François Chaumette, and Patrick Gros. Robot motion control from a visual memory. In *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA'04. 2004*, volume 5, pages 4695–4700. IEEE, 2004.

[120] Edward Rosten and Tom Drummond. Machine learning for high-speed corner detection. In *European conference on computer vision*, pages 430–443. Springer, 2006.

[121] Daniel J Russo, Benjamin Van Roy, Abbas Kazerouni, Ian Osband, Zheng Wen, et al. A tutorial on thompson sampling. *Foundations and Trends® in Machine Learning*, 11(1):1–96, 2018.

[122] Fereshteh Sadeghi and Sergey Levine. Cad2rl: Real single-image flight without a single real image. *arXiv preprint arXiv:1611.04201*, 2016.

[123] Fereshteh Sadeghi and Sergey Levine. Rl: Real single-image flight without a single real image. arxiv preprint. *arXiv:1611.04201*, 12, 2016.

[124] Fereshteh Sadeghi and Sergey Levine. (cad)2rl: Real single-image flight without a single real image. *Robotics: Science and Systems*, 2017.

[125] José Santos-Victor, Giulio Sandini, Francesca Curotto, and Stefano Garibaldi. Divergent stereo for robot navigation: Learning from bees. In *Proceedings of IEEE conference on computer vision and pattern recognition*, pages 434–439. IEEE, 1993.

[126] Davide Scaramuzza. 1-point-ransac structure from motion for vehicle-mounted cameras by exploiting non-holonomic constraints. *International journal of computer vision*, 95(1):74–85, 2011.

[127] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*, 2015.

[128] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. In *International Conference on Learning Representations*, Puerto Rico, 2016.

[129] Florian Schroff, Dmitry Kalenichenko, and James Philbin. Facenet: A unified embedding for face recognition and clustering. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 815–823, 2015.

[130] Alexander Scott, Lynne E Parker, and Claude Touzet. 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*, volume 2, pages 1422–1427. IEEE, 2000.

[131] Souvik Sen, Jeongkeun Lee, Kyu-Han Kim, and Paul Congdon. Avoiding multipath to revive inbuilding wifi localization. In *Proceeding of the 11th annual international conference on Mobile systems, applications, and services*, pages 249–262. ACM, 2013.

[132] Souvik Sen, Božidar Radunovic, Romit Roy Choudhury, and Tom Minka. You are facing the mona lisa: spot localization using phy layer information. In *Proceedings of the 10th international conference on Mobile systems, applications, and services*, pages 183–196. ACM, 2012.

[133] Rico Sennrich, Barry Haddow, and Alexandra Birch. Improving neural machine translation models with monolingual data. *arXiv preprint arXiv:1511.06709*, 2015.

[134] Jayaram Sethuraman. A constructive definition of dirichlet priors. *Stat. Sin.*, pages 639–650, 1994.

[135] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484, 2016.

[136] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. In *Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32*, ICML'14, pages I–387–I–395. JMLR.org, 2014.

[137] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *Nature*, 550(7676):354, 2017.

[138] Kahlouche Souhila and Achour Karim. Optical flow based robot obstacle avoidance. *IJARS*, 4(1):2, 2007.

[139] Gregory J Stein and Nicholas Roy. Genesis-rt: Generating synthetic images for training secondary real-world tasks. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 7151–7158. IEEE, 2018.

[140] Richard S Sutton. Learning to predict by the methods of temporal differences. *Machine learning*, 3(1):9–44, 1988.

[141] Richard S Sutton, Andrew G Barto, et al. *Introduction to reinforcement learning*, volume 135. MIT press Cambridge, 1998.

[142] Richard S Sutton, David A McAllester, Satinder P Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Advances in neural information processing systems*, pages 1057–1063, 2000.

[143] Tristan Swedish and Ramesh Raskar. Deep visual teach and repeat on path networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 1533–1542, 2018.

[144] Lei Tai, Shaohua Li, and Ming Liu. A deep-network solution towards model-less obstacle avoidance. In *IROS*, pages 2759–2764. IEEE, 2016.

[145] Lei Tai and Ming Liu. Towards cognitive exploration through deep reinforcement learning for mobile robots. *arXiv:1610.01733*, 2016.

[146] Lei Tai, Giuseppe Paolo, and Ming Liu. Virtual-to-real deep reinforcement learning: Continuous control of mobile robots for mapless navigation. In *Intelligent Robots and Systems (IROS), 2017 IEEE/RSJ International Conference on*, pages 31–36. IEEE, 2017.

[147] Ashit Talukder, S Goldberg, Larry Matthies, and Adnan Ansar. Real-time detection of moving objects in a dynamic scene from moving robotic vehicles. In *Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003)(Cat. No. 03CH37453)*, volume 2, pages 1308–1313. IEEE, 2003.

[148] Ashit Talukder and Larry Matthies. Real-time detection of moving objects from moving vehicles using dense stereo and optical flow. In *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)(IEEE Cat. No. 04CH37566)*, volume 4, pages 3718–3725. IEEE, 2004.

[149] Edward L Thorndike. The law of effect. *The American journal of psychology*, 39(1/4):212–222, 1927.

[150] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. A probabilistic approach to concurrent mapping and localization for mobile robots. *Autonomous Robots*, 5(3-4):253–271, 1998.

[151] Sebastian Thrun and Michael Montemerlo. The graph slam algorithm with applications to large-scale mapping of urban structures. *The International Journal of Robotics Research*, 25(5-6):403–429, 2006.

[152] Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 23–30. IEEE, 2017.

[153] X. Truong and T. D. Ngo. Toward socially aware robot navigation in dynamic and crowded environments: A proactive social motion model. *IEEE Trans. Automat. Sci. Eng.*, 14(4):1743–1760, Oct 2017.

[154] Iwan Ulrich and Illah Nourbakhsh. Appearance-based obstacle detection with monocular color vision. In *AAAI/IAAI*, pages 866–871, 2000.

[155] Prahlad Vadakkepat, Kay Chen Tan, and Wang Ming-Liang. 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)*, volume 1, pages 256–263. IEEE, 2000.

[156] Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *AAAI*, pages 2094–2100, 2016.

[157] Ayzaan Wahid, Alexander Toshev, Marek Fiser, and Tsang-Wei Edward Lee. Long range neural navigation policies for the real world. *arXiv preprint arXiv:1903.09870*, 2019.

[158] Sen Wang, Hongkai Wen, Ronald Clark, and Niki Trigoni. Keyframe based large-scale indoor localisation using geomagnetic field and motion pattern. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2016.

[159] Xin Wang, Qiuyuan Huang, Asli Celikyilmaz, Jianfeng Gao, Dinghan Shen, Yuan-Fang Wang, William Yang Wang, and Lei Zhang. Reinforced cross-modal matching and self-supervised imitation learning for vision-language navigation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6629–6638, 2019.

[160] Ziyu Wang, Nando de Freitas, and Marc Lanctot. Dueling network architectures for deep reinforcement learning. *CoRR*, abs/1511.06581, 2015.

[161] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado Hasselt, Marc Lanctot, and Nando Freitas. Dueling network architectures for deep reinforcement learning. In Maria Florina Balcan and Kilian Q. Weinberger, editors, *Proceedings of The*

33rd International Conference on Machine Learning, volume 48 of *Proceedings of Machine Learning Research*, pages 1995–2003, New York, New York, USA, 20–22 Jun 2016. PMLR.

[162] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado Van Hasselt, Marc Lanctot, and Nando De Freitas. Dueling network architectures for deep reinforcement learning. *arXiv preprint arXiv:1511.06581*, 2015.

[163] Christopher John Cornish Hellaby Watkins. *Learning from delayed rewards*. PhD thesis, King's College, Cambridge, 1989.

[164] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.

[165] Oliver Wulf and Bernardo Wagner. Fast 3d scanning methods for laser measurement systems. In *International conference on control systems and computer science (CSCS14)*, pages 2–5. Citeseer, 2003.

[166] Jing Xiao, Zbigniew Michalewicz, Lixin Zhang, and Krzysztof Trojanowski. Adaptive evolutionary planner/navigator for mobile robots. *IEEE transactions on evolutionary computation*, 1(1):18–28, 1997.

[167] Zhuoling Xiao, Hongkai Wen, Andrew Markham, and Niki Trigoni. Robust pedestrian dead reckoning (r-pdr) for arbitrary mobile device placement. In *Indoor Positioning and Indoor Navigation (IPIN), 2014 International Conference on*, pages 187–196. IEEE, 2014.

[168] Linhai Xie, Sen Wang, Andrew Markham, and Niki Trigoni. Towards monocular vision based obstacle avoidance through deep reinforcement learning. *arXiv preprint arXiv:1706.09829*, 2017.

[169] Linhai Xie, Sen Wang, Stefano Rosa, Andrew Markham, and Niki Trigoni. Learning with training wheels: speeding up training with a simple controller for deep reinforcement learning. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6276–6283. IEEE, 2018.

[170] Zhaoming Xie, Glen Berseth, Patrick Clary, Jonathan Hurst, and Michiel van de Panne. Feedback control for cassie with deep reinforcement learning. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1241–1246. IEEE, 2018.

[171] Ali Yahya, Adrian Li, Mrinal Kalakrishnan, Yevgen Chebotar, and Sergey Levine. Collective robot reinforcement learning with distributed asynchronous guided policy search. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 79–86. IEEE, 2017.

[172] Shichao Yang, Sandeep Konam, Chen Ma, Stephanie Rosenthal, Manuela Veloso, and Sebastian Scherer. Obstacle avoidance through deep networks based intermediate perception. *arXiv:1704.08759*, 2017.

[173] Z. Yang, K. Merrick, L. Jin, and H. A. Abbass. Hierarchical deep reinforcement learning for continuous action control. *IEEE Trans. Neural Netw.*, 29(11):5174–5184, Nov 2018.

[174] Cang Ye, N. H. C. Yung, and Danwei Wang. A fuzzy controller with supervised learning assisted reinforcement learning algorithm for obstacle avoidance. *IEEE Trans. Syst., Man, Cybern. B*, 33(1):17–27, Feb 2003.

[175] Jingwei Zhang, Lei Tai, Peng Yun, Yufeng Xiong, Ming Liu, Joschka Boedecker, and Wolfram Burgard. Vr-goggles for robots: Real-to-sim domain adaptation for visual control. *IEEE Robotics and Automation Letters*, 4(2):1148–1155, 2019.

[176] Li Zhang and Bijoy K Ghosh. Line segment based map building and localization using 2d laser rangefinder. In *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No. 00CH37065)*, volume 3, pages 2538–2543. IEEE, 2000.

[177] Zhengyou Zhang. Microsoft kinect sensor and its effect. *IEEE multimedia*, 19(2):4–10, 2012.

[178] Oleksii Zhelo, Jingwei Zhang, Lei Tai, Ming Liu, and Wolfram Burgard. Curiosity-driven exploration for mapless navigation with deep reinforcement learning. *arXiv preprint arXiv:1804.00456*, 2018.

[179] Chao Zhou, Yucheng Wei, and Tieniu Tan. Mobile robot self-localization based on global visual appearance features. In *2003 IEEE International Conference on Robotics and Automation (Cat. No. 03CH37422)*, volume 1, pages 1271–1276. IEEE, 2003.

[180] Jin Zhou and Baoxin Li. Homography-based ground detection for a mobile robot platform using a single camera. In *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.*, pages 4100–4105. IEEE, 2006.

[181] Yuke Zhu, Roozbeh Mottaghi, Eric Kolve, Joseph J Lim, Abhinav Gupta, Li Fei-Fei, and Ali Farhadi. Target-driven visual navigation in indoor scenes using deep reinforcement learning. In *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, pages 3357–3364. IEEE, 2017.