

Titre: Deep Reinforcement Learning for Environment-Aware Mapless Navigation
Title:

Auteur: Charles Dansereau
Author:

Date: 2023

Type: Mémoire ou thèse / Dissertation or Thesis

Référence: Dansereau, C. (2023). Deep Reinforcement Learning for Environment-Aware Mapless Navigation [Mémoire de maîtrise, Polytechnique Montréal]. PolyPublie.
Citation: <https://publications.polymtl.ca/57051/>

 **Document en libre accès dans PolyPublie**
Open Access document in PolyPublie

URL de PolyPublie: <https://publications.polymtl.ca/57051/>
PolyPublie URL:

Directeurs de recherche: Gabriela Nicolescu
Advisors:

Programme: Génie informatique
Program:

POLYTECHNIQUE MONTRÉAL

affiliée à l'Université de Montréal

Deep Reinforcement Learning for Environment-Aware Mapless Navigation

CHARLES DANSEREAU

Département de génie informatique et génie logiciel

Mémoire présenté en vue de l'obtention du diplôme de *Maîtrise ès sciences appliquées*
Génie informatique

Décembre 2023

POLYTECHNIQUE MONTRÉAL

affiliée à l'Université de Montréal

Ce mémoire intitulé :

Deep Reinforcement Learning for Environment-Aware Mapless Navigation

présenté par **Charles DANSEREAU**

en vue de l'obtention du diplôme de *Maîtrise ès sciences appliquées*

a été dûment accepté par le jury d'examen constitué de :

Daniel ALOISE, président

Gabriela NICOLESCU, membre et directrice de recherche

Sarath Chandar ANBIL PARTHIPAN, membre

DEDICATION

*To all my friends,
thank you for the good times and for asking me about this project again and again.*

*To my family,
Thank you for all the unconditional support and love. . .*

ACKNOWLEDGEMENTS

First of all, I would like to thank my research director, Gabriela Nicolescu, for her constant support and for giving me the opportunity to have the best experience possible leading this project, as well as Bardienus Pieter Duisterhof, for his help and guidance for a major part of this project.

I would also like to thank the members of the jury Mr. Daniel Aloise and Mr. Sarath Chandar Anbil Parthipan for agreeing to examine my work.

I express my utmost gratitude to CRSNG and FRQNT for funding me, allowing me to concentrate on my research, as well as humanitas, who made sure i had the liberty to work on what passionned me.

Finally, I am eternally grateful to my friends, family, and my partner and accomplice for encouraging me during the dissertation of my thesis.

RÉSUMÉ

La navigation autonome est la capacité d'un robot ou d'un véhicule à se déplacer d'un point à un autre sans intervention humaine. Avec les progrès récents en vision par ordinateur et en robotique, les robots mobiles sont déployés dans de plus en plus de domaines et de scénarios. Cependant, l'état de l'art existant repose sur des méthodes qui peuvent être coûteuses en termes de calcul, nécessitant des processeurs puissants voire même des unités de traitement graphique embarqués, et supposent des environnements statiques. Cela limite ainsi fortement le déploiement de tels robots dans le monde réel, où les environnements complexes contiennent souvent des objets dynamiques. Ceci est d'autant plus vrai pour les véhicules aériens de petite taille qui n'ont pas la puissance de calcul nécessaire ou qui disposent d'une énergie embarquée limitée. Pour les robots plus petits, des méthodes de navigation réactives existent, mais elles peuvent monopoliser une proportion significative des ressources du robot.

Dans des travaux récents, l'apprentissage par renforcement profond (ARP) a été utilisé pour naviguer efficacement dans des environnements dynamiques et a montré des résultats prometteurs. Cependant, cette approche bout-en-bout est beaucoup moins robuste et n'a pas atteint les performances des méthodes classiques. Cette thèse adopte une approche combinant à la fois des méthodes traditionnelles et l'ARP pour permettre à des robots aux ressources limitées de naviguer en évitant les obstacles. L'ARP est appliqué pour apprendre un mécanisme d'attention permettant de sélectionner l'information importante à la navigation et ignorer les données des capteurs non pertinentes pour l'évitement d'obstacles, rendant les algorithmes de navigation plus efficaces et plus rapides. Nous implémentons and et évaluons agent pour l'ARP sur un drone en simulation et sur un micro-rover pour démontrer que l'approche est applicable sur différents robots.

Tout d'abord, plusieurs agents sont conçus avec différentes représentations d'état, espaces d'action et fonctions de récompense, et sont entraînés avec divers algorithmes d'ARP, architectures de politique et hyperparamètres. Ensuite, en utilisant les résultats de l'exploration initiale, les deux agents les plus performants sont sélectionnés et évalués à la fois de manière quantitative et qualitative, et sont comparés à une approche de navigation traditionnelle. Les agents ont développé deux comportements différents. Un agent a appris à limiter de manière agressive la méthode de navigation pour ne traiter que les données de détection et télémétrie par la lumière (lidar) critiques pour l'évitement des collisions. L'autre agent a appris une politique plus conservatrice, permettant à la méthode de navigation d'utiliser en moyenne 16% des données du lidar, en sélectionnant les informations pertinentes à la fois

pour l'évitement des collisions et pour une meilleure recherche de trajectoire. En plus de cette réduction des données utilisées, la méthode proposée a réduit le temps de calcul nécessaire pour la navigation d'un facteur de 2.7x.

ABSTRACT

Autonomous navigation is the ability of a robot or vehicle to move from one point to another without human interference. With the recent progress in computer vision and robotics, mobile robots are deployed in more and more areas and scenarios. However, the existing state-of-the-art consists of methods that can be computationally expensive, requiring powerful CPUs or even GPUs to be embedded, and assume static environments, limiting strongly the deployment of such robots into the real world, where complex environments often contain dynamic objects. This is even more true for micro-aerial vehicles that do not have the computational resources or have limited onboard energy. For smaller robots, map-less reactive navigation methods exist, but they can monopolize a significant proportion of the robot's resources.

In recent works, Deep Reinforcement learning (DRL) was used to perform efficient navigation in dynamic environments and has shown promising results. Still, this end-to-end approach is much less robust and has not reached the performance of classical state-of-the-art methods. This thesis takes the approach of combining both traditional methods and DRL to realize map-less navigation with obstacle-avoidance on resource-constrained robots. DRL is applied to learn a focus mechanism, which aims to select in a smart way which information is relevant for navigation and discard the less important sensory data, making the navigation algorithms more efficient and faster. We implement and evaluate a DRL agent both on a drone in simulation and a micro-rover in the real world to show it can be applied to different robots.

First, multiple agents are designed with different state representations, action spaces and reward functions, and are trained with various DRL algorithms, policy architectures, and hyperparameters. Second, using the results of the initial exploration, the two best-performing agents are selected and evaluated both in a quantitative and qualitative way, and are compared to a traditional navigation approach. The agents evolved two different behaviors. One agent learned to aggressively limit the navigation method to only process LiDAR data which is critical for collision avoidance. The other agent learned a more conservative policy, making the navigation method use 16% of the LiDAR data in average, selecting relevant information for both collision avoidance and better path-finding. In addition to using less sensory data, this method reduced the global processing time of the navigation pipeline by a factor of 2.7x, while maintaining the same navigation capabilities.

TABLE OF CONTENTS

DEDICATION	iii
ACKNOWLEDGEMENTS	iv
RÉSUMÉ	v
ABSTRACT	vii
TABLE OF CONTENTS	viii
LIST OF TABLES	xi
LIST OF FIGURES	xii
LIST OF SYMBOLS AND ACRONYMS	xiii
LIST OF APPENDICES	xv
CHAPTER 1 INTRODUCTION	1
1.1 Context and motivations	1
1.2 Objectives and contributions	2
1.3 Document organisation	3
CHAPTER 2 BASIC CONCEPTS	4
2.1 Reinforcement learning	4
2.1.1 Markov Decision Processes (MDPs)	5
2.1.2 Discounted rewards	6
2.1.3 Policies and Value functions	6
2.1.4 Model-based methods	8
2.1.5 Model-free methods	8
2.2 Deep Reinforcement Learning	11
2.2.1 Value-Based Methods for Deep Reinforcement Learning	12
2.2.2 Policy Gradient Methods for Deep Reinforcement Learning	14
2.3 Map-based Navigation	19
2.3.1 Sensing	19
2.3.2 Localization	20

2.3.3	Simultaneous Localization and Mapping (SLAM)	21
2.3.4	Path Planning	22
2.4	Map-less Navigation	23
2.4.1	Tangent Bug Algorithm	24
2.4.2	Vision-based methods	25
CHAPTER 3 LITERATURE REVIEW		28
3.1	Map-less obstacle avoidance navigation	28
3.1.1	Classical methods	28
3.1.2	Deep reinforcement learning for navigation	30
3.2	Runtime optimization	33
3.2.1	Hardware acceleration	33
3.2.2	Algorithmic optimization	34
3.2.3	Algorithm/hardware codesign	36
3.3	Conclusion	37
CHAPTER 4 RL-BASED ENVIRONMENT-AWARE MAPLESS NAVIGATION		39
4.1	Environment Design	39
4.1.1	Defining the task	39
4.1.2	Reinforcement Learning Environment	45
4.1.3	Environment and MDP modeling	48
4.2	Design of the DRL Agent	49
4.2.1	Action and Observation Space Design	49
4.2.2	Reward Design	52
4.2.3	Agent Architecture	55
CHAPTER 5 RESULTS		58
5.1	Training Results	58
5.1.1	Environment Setup for Evaluation	58
5.1.2	Evaluation Metrics	59
5.1.3	Comparison with other baselines	69
5.2	Qualitative Evaluation	73
5.2.1	Blocking Obstacle	74
5.2.2	Passing by an obstacle	76
5.2.3	Complex environment	78
5.3	Real Robot Experiments	81
5.4	Summary of key results	84

CHAPTER 6 CONCLUSION	86
6.1 Summary of Contributions	86
6.2 Limitations	87
6.3 Future Research	88
REFERENCES	89
APPENDICES	102

LIST OF TABLES

Table 3.1	Comparison of RL approaches for autonomous navigation. . . .	32
Table 3.2	Related work on accelerating efficient navigation pipelines . . .	34
Table 4.1	Comparison of Reinforcement learning Gyms	45
Table 4.2	YDLIDAR X2 performance specifications [1]	48
Table 4.3	Initial neural network architecture of the RL agent	56
Table 4.4	Recurrent neural network architecture of the RL agent with LSTMs	57
Table 5.1	Global agent configurations summary	58
Table 5.2	Configuration of the environment generator parameters	59
Table 5.3	Initial agents configuration summary.	60
Table 5.4	Success rate (%) achieved by the agent for different configura- tions of the observation spaces and reward functions.	60
Table 5.5	Average training time of the agents	64
Table 5.6	Domain of the hyperparameters explored for A2C	64
Table 5.7	Ablation study of the hyperparameters for A2C	66
Table 5.8	Policy architecture exploration results	68
Table 5.9	Final RL model and agent configuration	69
Table 5.10	Comparison of the agent with the baselines	72
Table 5.11	Results of the rover experiments.	82
Table 5.12	Simulation results on different methods.	85
Table A.1	Different network architectures explored for the agents	102

LIST OF FIGURES

Figure 2.1	Agent-Environment interaction	4
Figure 2.2	Monte Carlo policy improvement	9
Figure 2.3	Actor-Critic Architecture	15
Figure 4.1	Different arenas introduced for the task	40
Figure 4.2	Different obstacles spawned in the arenas	42
Figure 4.3	Map-less navigation pipeline	43
Figure 4.4	Velocity discretization for DWA	45
Figure 4.5	Air Learning Simulator	47
Figure 4.6	Vector Field Histogram method	50
Figure 4.7	Reward Function 2 for different r_{cost} values	54
Figure 5.1	Training results of the different reward functions	62
Figure 5.2	Training results with different learning rates	66
Figure 5.3	Comparison of the processing time of the DWA	70
Figure 5.4	Behavior of the agent in front of an obstacle	75
Figure 5.5	Visualizing the decisions of the agent in front of an obstacle	75
Figure 5.6	Behavior of the agent near an obstacle	77
Figure 5.7	Visualizing the decisions of the agent passing near an obstacle	77
Figure 5.8	Behavior of the agent in a complex environment	78
Figure 5.9	Visualizing the decisions of the agent in a complex environment	80
Figure 5.10	Number of sectors used by the agent	81
Figure 5.11	The robot focusing on part of the environment to avoid a human moving in its path.	82
Figure 5.12	The Agilex Limo rover [2].	82
Figure 5.13	Comparison of the processing time and sector usage with Focus Bug on the Limo rover	83
Figure 5.14	Examples of rover trajectories in static environments	84
Figure B.1	Ablation study on the sector cost	103
Figure C.1	Ablation study on the sector discretization	104
Figure D.1	Bug algorithm boundary following	105
Figure E.1	Comparison of the A2C and PPO algorithms in initial experi- ments.	106

LIST OF SYMBOLS AND ACRONYMS

UAV	Unmanned Aerial Vehicle
MAV	Micro Aerial Vehicle
RL	Reinforcement Learning
MDP	Markov Decision Process
SR	Success Rate
DP	Dynamic Programming
GPI	Generalized Policy Iteration
TD	Temporal Difference
DRL	Deep Reinforcement Learning
ARP	Apprentissage par Renforcement Profond
DQN	Deep Q-Networks
DDQN	Double Deep Q-Networks
DDPG	Deep Deterministic Policy Gradient
A2C	Advantage Actor-critic
PPO	Proximal Policy Optimization
TRPO	Trust Region Optimization
NN	Neural Network
LSTM	Long short-Term Memory
FCN	Fully Connected Networks
ReLU	Rectifier Linear Unit
RNN	Recurrent Neural Network
FPGA	Field-Programmable Gate Array
GPU	Graphics Processing Unit
CPU	Central Processing Unit
ASIC	Application-Specified integrated Circuit
DSP	Digital Signal Processor
HIL	Hardware-in-the-Loop
LiDAR	Light Detection and Ranging
MPC	Model Predictive Control
PID	Proportional-Integral-Derivative
SLAM	Simultaneous Localization and Mapping
VIO	Visual Inertial Odometry
DWA	Dynamic Window Approach

VFH	Vector-Fields histograms
APF	Artificial Potential Fields
RRT	Rapidly exploring Random Tree
GPS	Global Positioning System
RGB	Red-Green-Blue
ToF	Time-of-Flight
QoF	Quality-of-Flight
INS	Inertial Navigation System
IMU	Inertial Mesurement Unit
MCL	Monte Carlo Localization
ICP	Iterative Closest Point

LIST OF APPENDICES

Appendix A	Neural network architectures explored	102
Appendix B	Ablation study on the cost of sectors in the reward function	103
Appendix C	Ablation study on the discretization of the surroundings into sectors for the VFH method	104
Appendix D	Bug algorithm: the boundary in front of the bug is not always the best boundary to follow	105
Appendix E	Comparison of PPO and A2C	106

CHAPTER 1 INTRODUCTION

This chapter introduces the topic of the thesis, the challenges that it aimed to solve and the research objectives. In Section 1.1, a motivation for the use of reinforcement learning on intelligent sensing for navigation of resource-constrained robots is given. Section 1.2 lists the objectives and contributions of the thesis, and the document’s organization is summarized in Section 1.3.

1.1 Context and motivations

Energy-constrained mobile robots are a prevalent technology in various domains such as logistics, search and rescue, and environmental monitoring. These robots are typically equipped with a range of sensors, enabling them to perceive their surroundings and make informed decisions to navigate their environment. However, the computational capabilities of these robots are often limited due to power constraints imposed by their energy sources. Consequently, efficient utilization of computational resources becomes paramount to ensure longer missions and better performance [3] [4].

Dynamic environments pose significant challenges for mobile robots, as the surroundings are subject to frequent changes. Traditional map-based navigation methods rely on pre-built maps or build maps assuming static environments. However, in dynamic settings, these methods quickly become obsolete and fail to provide accurate obstacle-avoidance strategies. Thus, there is a pressing need for alternative approaches that can adapt in real-time to changing environments [5].

To address the limitations of map-based navigation, a map-less approach is considered in this research. Map-less approaches are in general less optimal but offer several advantages, including enhanced adaptability, reduced computational load, and improved real-time decision-making in dynamic environments. Moreover, their simplicity eliminates the need for explicit map representation, which can be resource-intensive and challenging to maintain in rapidly changing scenarios. Still, more than 90% of computation time in motion planning is spent on collision detection between the robot and the environment. [6]

Deep Reinforcement learning (DRL) has emerged as a powerful technique for training autonomous agents to make optimal decisions in complex and dynamic environments. The DRL-agents are trained in a designed environment to learn a task on a trial-and-error basis. By employing RL, we can train the mobile robot to learn effective policies that exploit sensory

data efficiently, enabling it to navigate through obstacles while conserving computational resources. Specifically, we aim to reduce the computational load of obstacle avoidance methods by utilizing reinforcement learning to process sensory data and extract which information is critical for obstacle avoidance. The increasing number of available realistic simulators is motivating to apply DRL to tasks in robotics, as DRL can now be applied safely and at a higher scale in simulation.

1.2 Objectives and contributions

The primary objective of this thesis is to show how the existing obstacle-avoidance methods can be improved to be faster and use less computational resources. Reinforcement learning would allow to select the relevant sensory information so that the navigation pipeline can focus its resources only on the information which is critical to navigate safely.

Research questions: How can Deep Reinforcement Learning be used to identify the important sensory information for safe navigation, and reduce the computational load of existing methods?

To answer this question, we target these main objectives:

- Design an environment with dynamic obstacles in a realistic simulator;
- Define the RL task and its dynamics (action and state spaces, rewards, etc.);
- Design and train a DRL agent to reduce the computational load of an obstacle-avoidance pipeline, and;
- Investigate the effectiveness of the proposed approach in reducing the computational load of obstacle avoidance methods, and evaluate the adaptability and real-time decision-making capabilities of the RL-based system in dynamic environments.

Following the previous objectives, the contributions of this thesis are:

1. The design of a DRL agent to learn to identify critical sensory information for obstacle avoidance, demonstrating how navigation can be made more efficient by focusing computational resources on the most relevant parts of the environment;
2. An approach that combines the dynamic window approach, the tangent bug algorithm, and a deep reinforcement learning agent for efficient navigation in dynamic environments without requiring a map, and;

3. Various simulation and real-world experiments show the advantages of the proposed method in terms of processing speed and success rate over classical and DRL baselines.

The research conducted in this thesis holds significant implications for the field of mobile robotics, particularly in the context of energy-constrained robots operating in dynamic environments, and applied RL, in the context of developing methodologies to apply RL to multidisciplinary fields. By utilizing reinforcement learning to process sensory data in a smart way, we aim to reduce the computational load of obstacle avoidance methods, thereby conserving valuable computational resources. The map-less approach allows the robot to adapt to dynamic environments in real-time, making it more versatile and capable of navigating through changing scenarios. These findings will contribute to the advancement of mobile robot navigation techniques and pave the way for more efficient and adaptable systems in practical applications.

1.3 Document organisation

Chapter 2 presents the basics concepts of Reinforcement Learning and robot navigation to fully understand the topic of the thesis. In Chapter 3, a literature review is done, focusing on the research aiming to accelerate map-less obstacle-avoidance methods. This includes different approaches such as other DRL works, hardware acceleration, and algorithmic optimization. Chapter 3 illustrates that DRL-based methods are not yet able to fully solve (end-to-end) the task of navigation with obstacle avoidance in a robust way, and that classical approaches to make navigation more efficient are often not generalizable to other robots or situations. This justifies the combination of DRL and classical approaches in this thesis, to achieve both robust and efficient navigation. The methodology to train an agent for this task is described in Chapter 4. Finally, Chapter 5 presents the results achieved by the trained agent, studies the behavior of the proposed method, and compares it to classical baselines and state-of-the-art methods.

CHAPTER 2 BASIC CONCEPTS

This chapter summarizes the relevant concepts for further understanding of the topic of the thesis. In Section 2.1, the basic concepts and key ideas of traditional Reinforcement Learning, such as the Markov Decision Process framework, policies and value functions, and basic algorithms, are described. Section 2.2 presents how to apply deep learning techniques to Reinforcement Learning, known as Deep Reinforcement Learning (DRL). Four Main DRL algorithms are discussed: Deep Q-Network (DQN), Deep Deterministic Policy Gradient (DDPG), Advantage-Actor Critic (A2C), and Proximal Policy Optimization (PPO). Finally, Section 2.3 covers the fundamentals of autonomous navigation, discussing both map-based and map-less navigation.

2.1 Reinforcement learning

Reinforcement learning [7] is a machine learning paradigm that focuses on developing algorithms capable of learning and making decisions in dynamic environments to maximize some notion of cumulative reward. It draws inspiration from how humans learn from interaction with the world.

In reinforcement learning, an agent interacts with an environment through a series of actions, observations of the state of the environment, and rewards. The agent's goal is to learn a strategy for selecting actions in different states to maximize its expected cumulative reward over time. At each timestep $t = 0, 1, 2, \dots$, the agent chooses an action A_t to move from the state S_t to S_{t+1} , and receives a reward R_t .

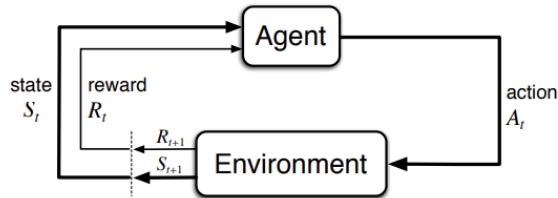


Figure 2.1 Interaction of the agent with its environment in reinforcement learning [7]

The fundamental idea behind reinforcement learning is to learn by trial and error, guided by the feedback provided by the environment. Through a process of exploration and exploitation, the agent learns to associate certain actions with positive or negative outcomes, allowing it to make better decisions to maximize its long-term rewards.

Reinforcement learning has achieved remarkable successes in various domains, ranging from playing complex board games like Go and chess to controlling autonomous vehicles and robots: achievements include AlphaGo defeating world champion Go players [8], DeepMind’s AI agent mastering several Atari 2600 games with only visual input [9], and OpenAI’s Dota 2 bot defeating professional human players [10].

This Section will cover the fundamental concepts and techniques in reinforcement learning: Markov Decision Processes, policies, value functions, and common reinforcement learning methods as presented in *Reinforcement Learning - An Introduction* by Sutton and Barto [7]. They serve as a basis for the more advanced techniques of deep reinforcement learning, presented in Section 2.2

2.1.1 Markov Decision Processes (MDPs)

Reinforcement learning builds on the concept of Markov Decision Processes (MDPs), which provide a mathematical framework to model sequential decision-making problems. By formulating problems as MDPs, reinforcement learning algorithms can learn optimal policies or value functions to guide decision-making. An MDP captures the dynamics of the environment and consists of a set of states, actions, transition probabilities, and reward functions. It is defined as a tuple (S, A, P, R) , where:

- S is the set of all possible states. Each state $s \in S$ represents the current situation or configuration of the environment;
- A is the set of all possible actions. Each action $a \in A$ represents a decision made by the agent;
- $P(s'|s, a)$ is the transition probability. It represents the probability of transitioning from state s to state s' when action a is taken. The transition probabilities capture the dynamics of the environment, and;
- $R(s, a, s')$ is the reward function. It assigns a numerical value to the agent’s experience when transitioning from state s to state s' by taking action a . The goal of the agent is to maximize the cumulative reward it receives over time.

The dynamics of an MDP can be described using the Markov property, which can be expressed as:

$$P(s_{t+1}|s_t, a_t, s_{t-1}, a_{t-1}, \dots, s_1, a_1) = P(s_{t+1}|s_t, a_t) \quad (2.1)$$

where s_t and a_t represent the state of the environment and the action performed at time t , respectively. the equation states that the future state depends only on the current state and action, and not on the past history.

2.1.2 Discounted rewards

In reinforcement learning, the concept of discounted rewards is used to assign a value to future rewards. By applying a discount factor, the agent places more importance on immediate rewards compared to future rewards. This discounting allows the agent to balance immediate gains with potential long-term rewards. The discounted cumulative reward, also known as the return, is defined as the sum of the discounted rewards over a sequence of time steps.

For a sequence of rewards $r_0, r_1, r_2, \dots$ obtained by the agent, the discounted cumulative reward at time step t is calculated as follows:

$$G_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \dots = \sum_{k=0}^{\infty} \gamma^k r_{t+k} \quad (2.2)$$

Here, γ is the discount factor, which determines the importance of future rewards compared to immediate rewards. It typically takes a value between 0 and 1, a value of 1 meaning future rewards are as important as immediate rewards, and a value of 0 meaning future rewards are ignored.

The discount factor ensures that the sum converges to a finite value as long as the rewards are bounded. This convergence property allows for the calculation of the value of each state or action in terms of its expected discounted cumulative reward, even in situations where actions are taken indefinitely.

The discounted rewards formulation accounts for the uncertainty and variability in the environment, as future rewards are inherently less certain than immediate rewards. It also encourages the agent to make decisions that maximize long-term rewards.

2.1.3 Policies and Value functions

As presented earlier, the agent interacts with the MDP through a sequence of actions, observations, and rewards. At each time step t , the agent observes the current state s_t , selects an action a_t based on its policy, and receives a reward r_t and the next state s_{t+1} . The goal is to learn a policy that maximizes the expected cumulative reward.

A policy is defined as a rule or a function that maps states to actions. It helps the agent in

determining the optimal action in a certain state to maximize the cumulative reward. The policy π can either be deterministic or stochastic:

$$\begin{array}{ll} \pi(a|s) : S \rightarrow A & \text{Deterministic} \\ \pi(a|s) = P(A_t = a | S_t = s) & \text{Stochastic} \end{array}$$

Since the policy is an integral part of the agent's learning process, "policy" and "agent" are often substituted with each other. To evaluate and compare different policies and state-action pairs, value functions are used.

State-Value Function

The state-value function for a given policy, denoted as $V^\pi(s)$, measures the expected return starting from state s and following policy π . It represents the long-term value of being in a particular state under a given policy. It is defined as:

$$V^\pi(s) = E_\pi \left[\sum_{t=0}^{\infty} \gamma^t r_t \mid s_0 = s \right] \quad (2.3)$$

Action-Value Function

The action-value function for the same policy, denoted as $Q^\pi(s, a)$, measures the expected return starting from state s , taking action a , and following policy π . It represents the long-term value of choosing a particular action in a given state under a given policy. It is defined as:

$$Q^\pi(s, a) = E_\pi \left[\sum_{t=0}^{\infty} \gamma^t r_t \mid s_0 = s, a_0 = a \right] \quad (2.4)$$

The action-value function is a useful measure for selecting the best action to take in a given state, contrary to the state-value function, which only tells us how good a given state is.

The state-value function and the action-value function are related as follows:

$$V^\pi(s) = \sum_{a \in A} \pi(a|s) Q^\pi(s, a) \quad (2.5)$$

The Bellman optimality equation defines the optimal value functions, which represent the maximum expected return achievable under an optimal policy:

$$V^*(s) = \max_{a \in A} \left[R(s, a) + \gamma \sum_{s' \in S} P(s'|s, a) V^*(s') \right] \quad (2.6)$$

$$Q^*(s, a) = R(s, a) + \gamma \sum_{s' \in S} P(s'|s, a) \max_{a' \in A} Q^*(s', a') \quad (2.7)$$

where $V^*(s)$ and $Q^*(s, a)$ represent the optimal state-value and action-value functions, respectively. A policy π is considered optimal if it is better than any other policy π' , meaning its value function is greater or equal to the value function of the other policies for all possible states :

$$V^*(s) = V_\pi(s) \geq V_{\pi'}(s) \quad \forall s \in S \quad (2.8)$$

There can be multiple optimal policies that lead to the same optimal value function.

2.1.4 Model-based methods

Model-based methods try to solve the reinforcement learning problem by using knowledge of the environment for planning. They use a model of the environment to predict the transitions from one state of the MDP to another, according to the action considered. This allows one to perform optimal planning by directly solving the Bellman optimality equations in an iterative way, an approach known as Dynamic Programming (DP) [11]. A model of the environment is derived from first principles or gathered data, and the model is used to predict the dynamics of the environment and learn a policy.

However, if the model of the environment is not accurate, the policy learned will most likely make wrong decisions when deployed in the real world. Since reinforcement learning tasks in the real world often involve very complex environments, a more popular approach is to try to learn the model, with methods such as Dyna-Q [12] or Real-time model-based architecture [13]. Since the environments and dynamics of the vehicles for autonomous obstacle avoidance in hard environments are highly complex, this thesis will be taking the approach of model-free RL, presented in Section 2.1.5.

2.1.5 Model-free methods

Model-free methods have the advantage of being able to be applied to any reinforcement learning problem, as they do not require a model of the environment [7]. The main approaches to model-free reinforcement learning try to learn a value function (value-based methods, presented in Subsection 2.2.1) or directly optimize a policy (policy gradient methods, presented

in Subsection 2.2.2). The following subsections will cover some well-established model-free algorithms, upon which many deep reinforcement learning algorithms are based.

Monte Carlo methods

Monte Carlo methods make use of the Generalized Policy Iteration (GPI) idea. It is an iterative method of two steps. The first step, known as the policy evaluation step, tries to approximate the action-value table Q based on the current policy π : $Q \approx q^\pi$. The second step, known as the policy improvement step, aims to improve the policy with the most recent approximation of Q . Figure 2.2 illustrates the iterative process.

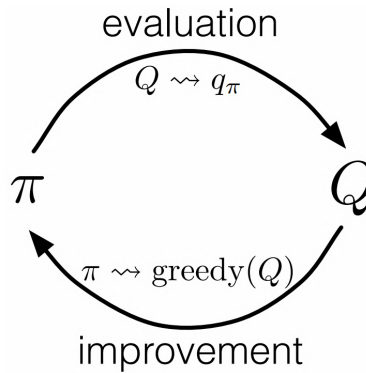


Figure 2.2 GPI for Monte Carlo control. **Evaluation:** The agent collects experiences to update the action-value function. **Improvement:** The policy is improved according to the updated Q-table. [7]

To estimate the Q-table in the first step, the agent follows the current policy in the environment, and the accumulated reward over the entire episode is used to form an estimate. To improve the policy in the second step, the policy is updated using a greedy strategy, where the action with the highest value in each state is updated in the policy. It can be shown that GPI converges to the optimal value function and policy, although Monte Carlo methods typically require a large number of iteration for their convergence and suffer from a large variance in their value function estimation. [7]

Temporal Difference methods

In temporal difference learning (TD), the policy is updated in each timestep instead of using the accumulated reward, so that there is no need for an entire episode to be completed before improving the policy. To do this, a temporal error (TD-error), defined as the difference between the new estimate and the old estimate of the value function, is calculated:

$$V(s) \leftarrow V(s) + \alpha \underbrace{[r + \gamma V(s') - V(s)]}_{\text{TD-error}} \quad (2.9)$$

where α is the learning rate, r is the reward obtained in the last timestep, s' is the new state and s is the last state. This kind of update reduces variance but increases bias in the estimate of the value function, as it is approximated from the reward of a single timestep instead of a whole episode. Two TD algorithms that have been widely used in reinforcement learning are SARSA and Q-learning, presented below.

SARSA: SARSA (state-action-reward-state-action) is a TD control algorithm which learns the action-value function instead of the value function. The policy evaluation step adapts the TD-error of the general TD algorithm for the action-value function. Algorithm 1 summarizes the the main steps of SARSA. First, the action-value function is initialized arbitrarily. Then, an initial state and action are selected following an exploration policy, such as ϵ -greedy. In brief, this policy selects either the best action according to the action-value estimate, or a random action according to a probability ϵ . The action is performed on the environment to observe the corresponding new state and reward, and a new action is chosen from the new state using the ϵ -greedy policy. Finally, As shown on Step 9, the difference between the new estimate $r + \gamma Q(s', a')$ and the old estimate $Q(s, a)$ is used to update the action-value function, and the process is repeated until convergence of the action-value function.

Algorithm 1 SARSA [7]

```

1: Initialize action-value function  $Q(s, a)$  arbitrarily
2: for Each Episode do
3:   Initialize state  $s$ 
4:   Choose action  $a$  from  $s$  using an exploration policy (e.g.,  $\epsilon$ -greedy)
5:   while  $s$  is not terminal do
6:     Take action  $a$ , observe reward  $r$ , and new state  $s'$ 
7:     Choose action  $a'$  from  $s'$  using the exploration policy
8:     Update the action-value function:
9:        $Q(s, a) \leftarrow Q(s, a) + \alpha [r + \gamma Q(s', a') - Q(s, a)]$ 
10:     $s \leftarrow s'$ 
11:     $a \leftarrow a'$ 
12:   end while
13: end for
```

Q-learning: Q-learning was introduced by Watkins [14]. Contrary to SARSA, it is an off-policy algorithm, meaning the Q-table update rule is different: as shown in Algorithm 2, instead of using the Q-value of the next action pair $Q(s', a')$ by following the current policy,

the maximum value of the next state is taken (regardless of what action the ϵ -greedy policy would take). This means the action that is used to update the action-value function is not necessarily the same as what the current policy would have chosen. The Q-learning update rule greatly simplified early convergence proofs [7].

Algorithm 2 Q-learning [7]

```

1: Initialize action-value function  $Q(s, a)$  arbitrarily
2: for Each Episode do
3:   Initialize state  $s$ 
4:   while  $s$  is not terminal do
5:     Choose action  $a$  from  $s$  using an exploration policy (e.g.,  $\epsilon$ -greedy)
6:     Take action  $a$ , observe reward  $r$ , and new state  $s'$ 
7:     Update the action-value function:
8:        $Q(s, a) \leftarrow Q(s, a) + \alpha [r + \gamma \max_{a'} Q(s', a') - Q(s, a)]$ 
9:      $s \leftarrow s'$ 
10:  end while
11: end for

```

2.2 Deep Reinforcement Learning

Deep reinforcement learning provides a flexible framework for modeling complex environments. Neural networks are capable of approximating value or policy functions, allowing agents to learn from high-dimensional state spaces and handle large action spaces. In real-world problems, the state or action spaces can get extremely large. It is not feasible to visit all possible states to retrieve the value for all action-state pairs. Deep reinforcement learning helps solve this problem by approximating the value function and policy, enabling to infer the value of a given state by generalizing with other similar states that were encountered. This is also important as the tabular methods of classical RL are also limited by the physical memory required to store all the values for each state, which becomes prohibitive for large state spaces.

Furthermore, in complex environments, traditional reinforcement learning methods often require manual feature engineering, where domain-specific knowledge is utilized to design meaningful representations of the environment. In deep reinforcement learning, the agent autonomously learns to extract the relevant features from the raw inputs, eliminating the need for explicit feature engineering. This learning approach enables the agent to automatically discover meaningful state representations, making it applicable to a wider range of domains and reducing the need for human expertise to design features.

The integration of deep learning techniques with reinforcement learning has led to remark-

able achievements across various domains. Deep reinforcement learning algorithms have demonstrated exceptional performance in complex video games, surpassing human-level performance in challenging environments like Dota 2 [10] and StarCraft II, which was considered an important challenge due to its raw complexity and multi-agent challenges [15]. They have also made significant advancements in robotics control, autonomous driving, natural language processing, and numerous other fields.

2.2.1 Value-Based Methods for Deep Reinforcement Learning

Value-based methods adapted to deep reinforcement learning aim to estimate the action-value function or the state-value function with neural networks. They learn the optimal value function and derive a classical policy (without neural networks) from it. One of the most influential value-based methods in deep reinforcement learning is the Deep Q-Network (DQN) and its improvements.

Deep Q-Network (DQN)

Deep Q-Network (DQN) [16] is a well-established algorithm in deep reinforcement learning that combines deep neural networks with the Q-learning algorithm to approximate action-value functions. DQN introduces two key mechanisms: the frozen target network and experience replay, which contribute to the stability and efficiency of learning.

To approximate the action-value function $Q(s, a)$, DQN employs a neural network with weights θ . The Q-network takes the state s as input and outputs the estimated Q-values for all possible actions. The Q-value for a specific action a is denoted as $Q(s, a; \theta)$.

During the training process, DQN uses a variant of the Q-learning update rule to update θ . The update rule minimizes the following loss function:

$$L(\theta) = E \left[\left(r + \gamma \max_{a'} Q(s', a'; \theta^-) - Q(s, a; \theta) \right)^2 \right] \quad (2.10)$$

where θ^- represents the weights of the frozen target network. The frozen target network is a copy of the Q-network with fixed weights that are updated less frequently. It provides a stable target for the Q-learning update, reducing the divergence and improving the convergence of the training process.

In addition to the frozen target network, DQN employs experience replay to improve learning efficiency and stability. Experience replay stores the agent's experiences, including the state, action, reward, and next state, in a replay buffer. During training, a set, often called a

batch, of experiences is randomly sampled from the replay buffer, allowing the agent to learn from experiences with diverse situations and reducing the correlation between consecutive samples.

The DQN algorithm updates the Q-network’s weights by minimizing the loss function over the batch of experiences, using gradient descent methods such as stochastic gradient descent (SGD) [17]. This process iteratively improves the Q-network’s approximations of the action-value function.

In short, the combination of the frozen target network and experience replay in DQN addresses the issue of instability and divergence that can occur when training deep neural networks using reinforcement learning. The frozen target network provides a consistent target for training, and experience replay decorrelates the training data, making the learning process more stable and efficient.

Improvements to DQN

Several improvements have been proposed to increase the performance and stability of the original DQN algorithm. These advancements address some of the limitations and challenges faced by DQN, leading to more effective and efficient learning. Some notable improvements include Double DQN and Dueling DQN.

Double DQN: Double DQN addresses the issue of overestimation bias in the original DQN algorithm [18]. In DQN, the maximum Q-value for the next state is used to estimate the target Q-value. However, this can lead to over-optimistic value estimates, as the same set of weights is used for both selecting and evaluating actions. Double DQN introduces a decoupling of action selection and evaluation by using the Q-network to select the action and the frozen target network to evaluate the Q-value. By mitigating the overestimation bias, Double DQN achieves more accurate value estimates and generally improves the overall performance.

Dueling DQN: Dueling DQN introduces a new architecture that separately estimates the value and advantage functions [19]. The value function estimates the value of being in a particular state, while the advantage function quantifies the importance of each action compared to the average action value. By decoupling the estimation of value and advantage, Dueling DQN improves the learning efficiency, and most importantly, stability. It allows the agent to focus on states with high potential and prioritize actions based on their advantage values.

Prioritized Experience Replay: Prioritized Experience Replay modifies the original ex-

perience replay mechanism of DQN by assigning priorities to experiences based on their TD errors [20]. Experiences with higher TD errors, which means a higher difference between predicted and target Q-values, are sampled with higher probability. This approach emphasizes the replay of experiences that are most informative or challenging for the agent, leading again to improved learning efficiency and faster convergence.

2.2.2 Policy Gradient Methods for Deep Reinforcement Learning

Unlike value-based methods that focus on estimating the action-value function, policy gradient methods aim to find the optimal policy by directly optimizing the policy parameters. In policy gradient methods, the policy is typically represented by a parametric function $\pi(a|s; \theta)$, which outputs the probability of taking action a in state s with parameters θ . In the case of deep reinforcement learning, the function is approximated by a neural network with weights θ . The goal is to find the policy parameters that maximize the expected cumulative reward, also known as the objective function $J(\theta)$.

Policy gradient methods employ gradient ascent to update the policy parameters in the direction of increasing expected reward. The update rule can be expressed as:

$$\theta_{t+1} = \theta_t + \alpha \nabla_{\theta} J(\theta) \quad (2.11)$$

where α is the learning rate and $\nabla_{\theta} J(\theta)$ is the gradient of the objective function with respect to θ .

Value-based methods often struggle with issues like overestimation bias and instability in the estimation of action values: a small change in the value function can have a great impact on the policy output, resulting in oscillations during training. Policy gradient methods, on the other hand, avoid the complexities associated with estimating value functions since they directly optimize the policy function.

Policy gradient methods can also learn stochastic policies, which are valuable in scenarios where exploring different actions is necessary to discover optimal strategies. Stochastic policies allow for exploration, helping the agent to escape from local optima and find more diverse and potentially better solutions. This capability is especially important in environments with inherent uncertainty or when dealing with partial observability.

Further, those methods naturally handle continuous and high-dimensional state spaces, making them well-suited for real-world problems with complex observations. Unlike value-based methods that require function approximation techniques or discretization, policy gradient

methods can directly parameterize policies and learn to map states to actions without the need for extensive preprocessing or feature engineering.

One disadvantage of policy gradient methods is the possibility of getting stuck in local optima. Since they optimize the policy function directly, there is a risk of converging to suboptimal policies, particularly in complex and high-dimensional spaces. [7]

Actor-Critic Architecture

The Actor-Critic architecture combines elements of both value-based and policy-based methods by maintaining two separate components: an actor network and a critic network. The actor network parameterizes the policy and is responsible for exploring the environment and selecting actions based on the learned policy. The critic network estimates the value function and provides feedback to the actor network by evaluating the quality of the chosen actions.

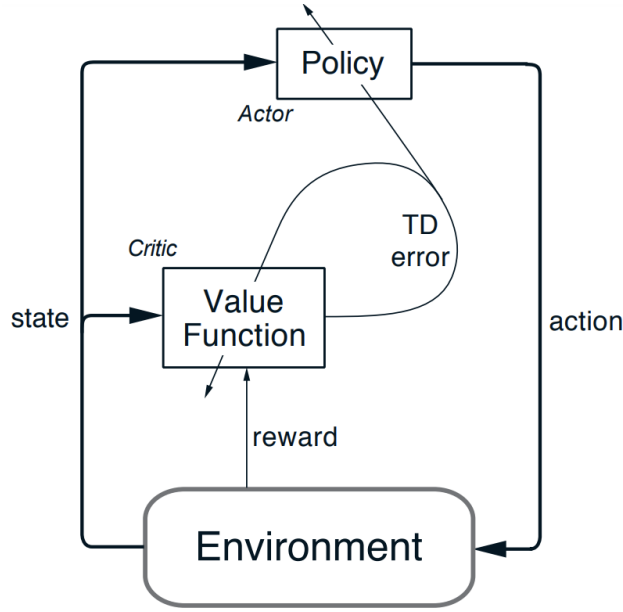


Figure 2.3 Interaction of the agent with its environment in the Actor-Critic architecture. the actor acts as the policy and takes the actions, interacting with the environment. The critic acts as the value function and helps the actor improve its policy. The TD error can be used to update both networks. [7]

As shown in Figure 2.3, the actor interacts with the environment, and the reward is given to the critic. Then, the networks are updated using temporal difference (TD) learning methods, to minimize the TD error between estimated and target values:

$$\delta_t = r_{t+1} + \gamma V(s_{t+1}; \theta_V) - V(s_t; \theta_V) \quad (2.12)$$

One of the first policy gradient methods is REINFORCE [21], proposed in 1992. It is now outperformed by multiple more recent DRL algorithms such as DDPG [22], TRPO [23], A3C [24] and PPO [25].

Deep Deterministic Policy Gradient (DDPG)

Deep Deterministic Policy Gradient (DDPG) [22] is an algorithm specifically designed for continuous action spaces. It combines elements of the actor-critic architecture with the mechanisms of DQNs [16] to learn deterministic policies in continuous control problems. It can be thought of as applying Q-learning to continuous action spaces, because of how the max over the possible actions is computed. Since the action space is continuous, it is impossible or extremely costly to find the optimal action in a given state by evaluating all the actions ($\max_a Q(s, a)$). Instead, the action value function is differentiated with respect to the actions, allowing to learn a policy $\mu(s)$ with deterministic policy gradient algorithms [26], which can be used to approximate $\max_a Q(s, a) \approx Q(s, \mu(s))$. Two main tricks introduced by DQN are employed to help learning the policy in a stable way: Replay buffers and Target networks.

Replay buffers. DDPG utilizes an actor network $\mu(s; \theta_\mu)$ which directly outputs the deterministic actions given states. The critic network $Q(s, a; \theta_Q)$ estimates the action-value function and provides feedback on the chosen actions. During training, two separate replay buffers are used: one for the actor network to store the transitions experienced by the agent, and another for the critic network to sample batches of transitions for updates. This replay buffer allows for more efficient use of past experiences, reducing the correlation between consecutive trajectories sampled by the agent.

Target networks. DDPG also applies a variant of the frozen target networks to stabilize the learning process. It uses two sets of target networks, one for the actor and one for the critic, which have their weights updated slowly through a soft update mechanism as shown below.

$$\theta_{Q_{\text{target}}} \leftarrow \tau \theta_Q + (1 - \tau) \theta_{Q_{\text{target}}} \quad (2.13)$$

$$\theta_{\mu_{\text{target}}} \leftarrow \tau \theta_\mu + (1 - \tau) \theta_{\mu_{\text{target}}} \quad (2.14)$$

The soft update parameter τ determines the rate at which the target network is updated. It is typically set to a small value (e.g., $\tau = 0.001$) to introduce delayed updates to the target networks, providing more stable and reliable target values during training.

Advantage Actor-Critic (A2C)

Advantage Actor-Critic (A2C) is a policy gradient algorithm that combines the benefits of both value-based and policy-based methods. It is a synchronous version of Asynchronous Advantage Actor Critic (A3C) [24], which makes use of an advantage function and has surpassed the state of the art on the Atari domain, while training for half the time on a single multi-core CPU instead of on a GPU. OpenAI showed that A2C performs better than their asynchronous implementations and that it is more cost-effective than A3C when using single-GPU machines [15].

The advantage function measures the advantage of taking a specific action compared to the average action value. The key idea is that it is better to choose an action considering its value relative to the other options, instead of its absolute value. It can be defined as the difference between the estimated action-value and the state value:

$$A(s, a) = Q(s, a) - V(s) \quad (2.15)$$

During training, A2C aims to maximize the expected return by updating both the actor and critic networks simultaneously. The actor network is trained using policy gradients which can be computed as:

$$\nabla_{\theta_\pi} J \approx \frac{1}{N} \sum_{i=1}^N \nabla_{\theta_\pi} \log \pi(a_i | s_i; \theta_\pi) A(s_i, a_i) \quad (2.16)$$

while the critic network is trained using the mean squared error loss between the estimated state value and the target value:

$$L(\theta_v) = \frac{1}{N} \sum_{i=1}^N (V(s_i) - V_{\text{target}})^2 \quad (2.17)$$

A2C offers several advantages compared to classical reinforcement learning algorithms like DQN. First, it combines the benefits of value-based and policy-based methods by estimating the value function and learning a policy directly. This allows for more stable and efficient learning. Second, A2C is an on-policy algorithm, which means it updates its policy using the most recent experiences. This leads to more sample-efficient learning. Finally, the advantage function guides the policy updates, allowing A2C to focus on actions with higher advantage values, which means the policy updates are in general more effective.

Proximal Policy Optimization (PPO)

Proximal Policy Optimization (PPO) [25] is a state-of-the-art policy gradient method that addresses some of the limitations, mainly stability during training, of traditional policy gradient algorithms like DDPG [22], TRPO [23], and A3C [24], by aiming to be much simpler to tune. To do this, PPO uses importance sampling, introduced by TRPO, and a surrogate objective function.

Importance sampling. Importance sampling improves the data efficiency of the learning algorithm by allowing the use of trajectories collected from previous policies to estimate the performance of the new policy. It estimates the expected value of a function $f(x)$ under the target distribution $p(x)$ (distribution of the new policy), with samples collected from the behavior distribution $q(x)$ (old policy) with the following equation:

$$\hat{E}_p[f(x)] = \frac{1}{N} \sum_{i=1}^N \frac{p(x_i)}{q(x_i)} f(x_i) \quad (2.18)$$

where x_i is sampled from $q(x)$. This leads to a new objective function also called the surrogate objective function:

$$L(\theta) = \hat{E}_t \left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)} A^{\pi_{\theta_{\text{old}}}}(s_t, a_t) \right] \quad (2.19)$$

where $A^{\pi_{\theta_{\text{old}}}}(s_t, a_t) = \sum_{t'=t}^T \gamma^{t'-t} R_{t'} - V^{\pi_{\theta_{\text{old}}}}(s_t)$ is the advantage estimate, similar to the advantage function used by A3C.

Surrogate objective function. The surrogate objective function ensures stable policy updates while maintaining a trust region [23]. The objective function tries to maximize the policy's performance while minimizing the deviation from the previous policy. By limiting the policy update step sizes, PPO prevents big policy changes that could lead to instability or divergence.

More specifically, it uses a clipped surrogate objective, which bounds the policy update based on a threshold, again to prevent too large policy updates and stabilize the learning process. The clipped surrogate objective can be expressed as a combination of the probability ratio between the new and old policies, and the clipped version of the ratio:

$$L^{\text{clipped}}(\theta) = E [\min(r_t(\theta) A^{\pi_{\theta_{\text{old}}}}(s_t, a_t), \text{clip}(r_t(\theta), 1 - \epsilon, 1 + \epsilon) A^{\pi_{\theta_{\text{old}}}}(s_t, a_t))] \quad (2.20)$$

where $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$ is the probability ratio, and ϵ is a hyperparameter determining the size of the trust region.

2.3 Map-based Navigation

Autonomous navigation [27] is an essential task in robotics allowing a robot to move in an environment without the help of a human pilot, and sometimes without GPS. It is often necessary in various fields from autonomous driving cars to Micro Aerial vehicles performing search and rescue, and from widespread automated delivery to domestic use for cleaning. This work focuses on the smaller robots with limited resources, that need to be efficient in their decision-making in multiple aspects, navigating hard environments with both static and dynamic obstacles. Because of this, it is a requirement that the robots navigating autonomously are able to perform obstacle avoidance.

In traditional map-based navigation, three main components allow the robot to move while avoiding collisions: Perception, planning, and control, also known as sensing, planning, and acting. Sensing enables the robot to get information on its environment to detect obstacles and know its position and orientation. To do this, sensors and localization methods are used. Planning involves using algorithms to find a path between the robot's location and where it wants to go. Finally, control systems take the produced paths of the planner and convert them into motor and actuator commands to navigate while performing collision avoidance.

2.3.1 Sensing

The first task of perception is to use sensors to gain information about the environment of the robot, allowing it to do map-based navigation. To do this, the robot extracts geometrical information using different sensors such as depth cameras, LiDAR, ultrasonic sensors, or RGB cameras. A map can then be built with the gathered information to perform obstacle avoidance.

Range sensors measure the distance to the objects without physical contact by emitting waves and measuring the delay before the signal returns. There are multiple methods such as laser range sensors and infrared sensors, which emit electromagnetic waves, and ultrasonic sensors, which emit sound waves [28]. LiDARs work by using light waves, which bounce off objects and return to the sensor. The time between the emission from the sensor and when it comes back is then measured to compute the distance, sometimes called Time-of-flight (ToF). All the distances can then be processed to generate a 3D representation of the surroundings, known as a *point cloud* [29].

Vision-based sensing with cameras allows robots to capture and process visual information, including object detection, recognition, and tracking. Further, visual sensors such as monocular cameras are cheap, lightweight and easy to use. However, this requires a lot more

processing to infer geometric information from the gathered images and was considered a major limitation of vision-based methods, since monocular cameras cannot perform triangulation to produce a precise depth map [30]. With the recent developments in computer vision and the achievements of deep learning, it is now an active field of research [31] [32].

2.3.2 Localization

The second task of perception in autonomous navigation is localization. It refers to the ability of a robot to determine its own position and orientation within its environment. It enables the robot to understand where it is relative to a map or its surroundings, which is essential for effective navigation and path planning.

The key concept of localization is to take information gathered by the sensors to estimate the robot's pose (position and orientation). This estimation is typically performed in real-time as the robot moves and perceives its surroundings. The main challenges in localization include dealing with uncertainty, handling sensor noise and errors, and addressing environmental changes (in dynamic environments) [33]. Multiple approaches for localization exist and are used depending on the environment and sensors available:

1. **Odometry:** Odometry relies on measuring the movement of the robot's wheels or other proprioceptive sensors (inertial navigation system (INS), accelerometers) to estimate its position. By keeping track of the distance traveled and changes in orientation, odometry provides a relative pose estimate. However, odometry is prone to accumulating errors over time and is not reliable for long-term localization without external corrections [34].
2. **Exteroceptive sensors:** in addition to the proprioceptive odometry, exteroceptive sensors can be used to use external information to help the robot localize itself. A simple method is to use a Global Positioning System (GPS) [35], however, this is limited to environments where line of sight with satellites can be maintained, and suffers from low precision. Alternatively, some indoors localization sensors include WIFI, RFID, Ultra-WideBand (UWB), Bluetooth and visible light communication (VLC) [36]. Exteroceptive sensors also include ranging sensors discussed in 2.3.1.
3. **Landmark-based Methods:** Landmark-based methods involve identifying and recognizing distinctive features or landmarks in the environment, such as visual markers or beacons. This can be achieved with range sensors [37] or vision-based techniques [31]. The robot can estimate its position by comparing the observed landmarks with a pre-existing map. However, this approach requires the presence and accurate detection of landmarks.

4. Kalman or Particle Filters: Particle filters, also known as Monte Carlo Localization (MCL) [38], are probabilistic algorithms used for localization. They maintain a set of particles, where each particle represents a potential pose of the robot. By updating and resampling these particles based on sensor measurements and motion models, particle filters provide an estimation of the robot's pose. Particle filters can handle uncertainty and work well in non-linear and multi-modal situations. The Kalman filter accomplishes this goal by linear projections, so they work best in linear systems [39]. Even though particle filters are more flexible and precise, both methods are often adapted for the Simultaneous Localization and Mapping (SLAM) problem, presented in 2.3.3, as Kalman filters are less computationally expensive.

2.3.3 Simultaneous Localization and Mapping (SLAM)

Simultaneous Localization and Mapping considers a class of problems where the robot doesn't have any prior information on its environment, and needs to both create a map and localize itself within the map in real-time, during the navigation. The map can then be used to navigate while avoiding obstacles. SLAM has been the focus of many works in recent years, and the increasing computing power available on embedded systems enables more and more practical applications [40] [41] [42]. An intuitive example for SLAM is the task of a robot vacuum cleaner. By mapping the environment and localizing itself, the robot can have a clear understanding of its surroundings and compute efficient paths that will ensure all the area is covered while avoiding to pass multiple times on the same zone. Compared to a robot without SLAM, this will result in much better cleaning, less battery usage and mission time, and less risk of bumping into things.

The two most studied methods to perform SLAM are Visual SLAM and LiDAR-based SLAM. Visual SLAM [42] uses cameras as the primary sensor. It extracts visual features from camera images, matches them across frames, estimates the camera's pose, and constructs a map of the environment. It benefits from rich visual information but can be challenging in low-texture or fast-motion scenarios. LiDAR-based SLAM [43] uses LiDAR sensors to generate dense 2D or 3D point clouds of the environment. It processes the point cloud data to extract features, and estimates the robot's pose by aligning the observed point cloud with previous point clouds using algorithms like iterative closest point (ICP) [44] and normal distributions transform (NDT) [45]. LiDAR-based SLAM provides precise information and are well suited to high-speed vehicles such as drones, but can be computationally demanding and expensive. Also, point clouds are not as detailed as images and sometimes lack the necessary features to be matched. Both methods have their advantages and limitations, and the choice depends on

factors such as the environment, desired level of detail, and specific application requirements. Finally, LiDAR-based SLAM is in general more robust than visual-based SLAM because the latter will often fail to measure a correct distance in the absence of features. For example, while staring at a blank wall, cameras will fail to measure the distance to the wall and fail to localize, whereas the LiDAR sensor will at least measure the correct distance to the wall.

There are still significant challenges to be addressed in SLAM. First, localization errors accumulate during navigation [46]. The longer the mission, the higher the risk of substantial error in localization, meaning the robot can fail to find valid paths to navigate its environment. A lot of work was done to try to mitigate this limitation by performing what is known as *loop closure* [47]. The idea is to use information from the map to detect when the robot visits a place where it already has been (closed a loop), which helps to localize in a much more robust way. However, this is still an active research area, as the remaining challenges include proposing solutions that are more robust, efficient, and scalable within limited resources. Second, the SLAM methods requires high computational resources [48], which can be a major limitation when trying to apply it to vehicles with limited onboard energy. Finally, another challenge is performing SLAM in dynamic environments. Moving obstacles or changing conditions (e.g. light) can result in conflict when updating the map and hinder the ability of the robot to find paths or even localize itself.

2.3.4 Path Planning

Once the robot can perceive its environment, the next step is to plan a trajectory, or path, to navigate around obstacles and reach the goal. Path planning algorithms are used to compute these paths to be the most efficient possible. A common target for this is to find the shortest path, but there are a lot of cases where other metrics like energy, path smoothness, and time can be more important, which means the shortest path is not necessarily the optimal path. An example of methods for such path planning problems includes trajectory optimization algorithms, where the path planning problem is formulated as an optimization problem that considers the desired vehicle performance, relevant constraints, and vehicle dynamics to optimize energy, time or distance to obstacles. However, this technique can be computationally expensive, making it hard for real-time applications in high-complexity problems [49]. Path planning algorithms can be divided into 2 categories: global planning and local planning [50].

Global path planning involves generating a high-level, long-term path from the robot's current position to its goal location with the use of a map, and often assuming a static environment. Common algorithms like A* [51], Rapidly exploring Random Tree (RRT) [52] and their

variants, are well established. As the full map is assumed to be static, global planning works best in static environments, as recomputing the path every timestep in dynamic environments can be computationally expensive. This type of path planning also often includes constraints to output feasible trajectories for the robot, such as making sure the maximum velocity is not exceeded, the angle of the terrain is not too steep, or enough energy is available for the whole trajectory.

Local path planning focuses on generating short-term, low-level paths or trajectories for the robot to follow based on its current sensor readings and immediate surroundings. It is a reactive method and is responsible for collision avoidance, obstacle evasion, and adapting the robot's motion to handle dynamic or unexpected situations. Because of this, Local path planning is often performed online and at a faster rate than global path planning, and will try to follow the high-level path generated by the global planning algorithms. Common approaches to local path planning include the dynamic window approach (DWA) [53], artificial potential fields (APF) [54], vector field histogram (VFH) [55], and more. They are discussed in more detail in Chapter 3.

In map-based navigation, both methods are used in combination. Global planning finds the overall optimal approach, and local planning reacts to unpredictable situations. The main limitation of these traditional approaches is that the effectiveness of the path planning is limited by the quality of the map representation, the space required to store it, and the intensive computation required to recompute the paths in dynamic environments.

2.4 Map-less Navigation

In map-less navigation, the robot navigates without using an existing map, or without creating one (SLAM). Instead, the robot directly uses its sensor observations and navigates by making reactive decisions. Local planning methods are especially well suited for this as they are fast, reactive methods, designed to adapt to unexpected situations and avoid obstacles. However, since no map is available, classical global planners cannot be used to compute a path, and local planners can get stuck in local minima, failing to reach the goal [56].

To solve this problem, bio-inspired algorithms can be used to complement local, reactive methods by acting as a global planner, making sure navigation can be performed without getting stuck. These planners are known as Bug Algorithms. Bug Algorithms are inspired by the navigation behavior of insects and other animals that use simple rules to navigate in their environment without a global map [57]. The basic idea behind bug algorithms is to use a combination of wall-following and path-searching techniques to navigate around

obstacles and reach a goal [58]. A classical bug algorithm is the Tangent Bug Algorithm, which will be used in this thesis. Alternatively, other approaches to map-less navigation that were proposed are vision-based methods, such as optical flow methods, appearance-based methods, and recent deep learning approaches, described below.

2.4.1 Tangent Bug Algorithm

The Tangent Bug algorithm [59] operates by imitating a bug that moves along the boundary of obstacles until it finds a tangent line to the goal. As shown in algorithm 3, it starts by initializing the bug position x to the robot's initial position, defining the goal position q_{goal} , and setting the minimum distance between the sensed boundary and the goal $d_{followed}$ to a large value. The shortest distance between the blocking obstacle and the goal d_{reach} (or the shortest reachable distance to the goal if no obstacles are visible) is also initialized. The bug tries to move towards the goal as long as it can reduce its heuristic distance $h(x, O_i, q_{goal})$, defined as :

$$h(x, O_i, q_{goal}) = d(x, O_i) + d(O_i, q_{goal}) \quad (2.21)$$

where $d(x, O_i)$ and $d(O_i, q_{goal})$ are respectively the distance from the robot to the sensed boundary O_i and the distance from the boundary to the goal. It can be the edge of an obstacle or free space if the range of the sensor is reached.

When the robot encounters an obstacle and can no longer reduce its heuristic, it follows the obstacle boundary, continually updating the set of new points in its environments $\{O_i\}$, d_{reach} and $d_{followed}$. The boundary following continues until of the following happens:

- $d_{reach} < d_{followed}$, meaning the robot found a direction to evade the obstacle, at which point the bug returns to directly moving towards the goal
- the goal is reached, and the algorithm stops.
- complete cycle is performed, meaning the goal is unreachable (even for map-based planners).

Algorithm 3 Tangent Bug Algorithm [59]

- 1: Initialize the minimum distance to the goal d_{reach} to a large value
 - 2: Initialize the current position x to the robot's initial position
 - 3: Initialize the current distance to the goal d_{reach} to the distance between x and the goal q_{goal}
 - 4: **repeat**
 - 5: Move towards goal in local direction that minimizes $h(x, n) = d(x, n) + d(n, q_{\text{goal}})$
 - 6: **until**
 - The goal is encountered, or
 - The value of $h(x, n)$ begins to increase
 - 7: **repeat**
 - 8: Follow the boundary of the obstacle, continuing in the same direction (tangent) as before.
 - 9: Update $\{O_i\}$, d_{reach} , and d_{followed}
 - 10: **until**
 - The goal is reached, or
 - A complete cycle is performed (goal is unreachable), or
 - $d_{\text{reach}} < d_{\text{followed}}$
-

Kamon et al [59] showed that the tangent bug algorithm is complete, meaning that if there exists a path between the robot and the goal, the algorithm is guaranteed to find a valid path. This guarantee is really useful and can be combined with local reactive planners to navigate in unknown environments without a map, while performing obstacle avoidance and without the risk of getting stuck in local minima. However, the major limitation of tangent bug algorithm is that there is no guarantee that the shortest path will be found, so the trajectories can be less efficient than those of map-based global planners.

2.4.2 Vision-based methods

With the increasing computational power available in small chips, vision-based navigation is becoming a more active field of research: the new available computational power mitigates the limitation of computer vision techniques regarding computing resources. Moreover, visual sensors like monocular cameras are cheap and lightweight, making them a good option for small mobile robots. Contrary to ranging sensors which emit waves, visual sensors are passive and don't affect the environment they're measuring. Cameras are also able to capture rich information in the images that wouldn't be available from a point cloud or depth

map, even though this requires complex computer-vision techniques, as no explicit geometric information is available [60]. Classical vision-based methods like the Optical flow method and appearance-based methods were proposed, and the advances of deep learning made it an attractive approach to perform such tasks [61]. Deep reinforcement learning for autonomous navigation will be discussed in Section 3.1.2

Optical flow method

Optical flow methods are techniques used in computer vision to estimate the motion of objects or pixels between consecutive frames of an image sequence. They are based on the assumption that pixels in an image sequence move smoothly from one frame to the next, meaning they don't drastically change in intensity, and the motion of near pixels is similar [62]. The goal is to compute the displacement vector for each pixel, representing its apparent motion. These displacement vectors form the optical flow field, which describes the overall motion pattern in the scene. The robot can then move toward parts of the image with lower relative velocity, as they indicate far objects, while close objects will move fast relative to the robot's field of view.

It is worth noting that, while optical flow provides valuable information about motion, it is a 2D approximation and cannot capture depth or 3D structures. To address this limitation, techniques such as stereo vision or combining optical flow with other range sensing methods (LiDAR, ultrasonic sensors, etc.) can be employed for a more accurate understanding of the environment [60].

Appearance based method

Appearance-based methods instead take the approach of memorizing the working environment. The key idea is to store visual information about the environment, and associate the series of commands that will successfully drive the robot to the goal when it encounters similar conditions [63]. For this, two phases are needed: the training phase allows to gather images or features of the environment and store them as model templates, which are labeled with localization information and necessary control commands. Then, in the navigation phase, the robot has to match its sensed images with the memorized information to localize itself and identify the correct commands to use. The main challenge for this is to find a good representation of the environment and correctly match the images with the memorized ones to find the appropriate commands [31]. However, these methods suffer from the same limitation that applying it in dynamic environments might lead to confusion in localizing itself in its memorized representation of the environment. For example, the robot might not be able to

match its perceived images if the light conditions have changed or multiple obstacles have moved.

CHAPTER 3 LITERATURE REVIEW

This chapter presents related work on autonomous navigation methods applicable to UAVs in changing environments or with limited onboard energy. We also present the state-of-the-art on research aiming to reduce the computational load and accelerate the obstacle avoidance methods. This chapter also discusses the limitations of the existing solutions and research that has attempted to mitigate them.

3.1 Map-less obstacle avoidance navigation

In this section, we summarize the scientific advances that allow more robust and safe autonomous navigation with obstacle avoidance. We present existing solutions to do obstacle avoidance navigation with limited onboard computing and their limitations.

3.1.1 Classical methods

One approach to map-less navigation is to use methods with reactive obstacle avoidance, often used in local planners, that rely on local sensor information to adjust the robot's speed and direction in real-time [64]. Multiple methods were proposed:

1. The Artificial Potential Fields (APF) method works by creating a virtual force field around the robot, where the repulsive forces from obstacles in the environment push the robot away and the attractive forces from the goal location pull the robot towards it. The resulting net force vector directs the robot towards the goal while avoiding obstacles in its path [54].

One advantage of the APF method is that it does not require prior knowledge of the environment so it can be used for map-free navigation, making it computationally efficient and a viable choice for real-time applications. However, the APF method has some limitations. For example, it can get stuck in local minima where the robot is unable to reach the goal due to a given configuration of obstacles. Additionally, even though the method is adaptable to new environments, it may not work well in complex environments with dynamic obstacles. Finally, the APF method can generate highly oscillatory or chattering control inputs which can lead to wear and tear on the hardware. It is still an active research field with many works trying to circumvent its limitations, such as [65], [66], [67].

2. The Vector Field Histogram (VFH) method works by dividing the robot's sensor data into a set of histograms, each representing a different angle range around the robot. The histograms are then used to generate a polar occupancy grid map that describes the presence of obstacles in each direction. The robot then chooses a path that maximizes the free space in the direction of the goal while avoiding obstacles [55].

It has the advantage of handling complex environments with irregular or dynamic obstacles. It is also relatively robust to sensor noise and can be used with a variety of sensor types. Additionally, the method can be extended to 3D environments, making it useful for navigation in complex indoor or outdoor environments. However, it requires a relatively high amount of computation, making it less suitable for resource-constrained robots. Additionally, it may not work well in very cluttered environments with many obstacles in close proximity to the robot [68].

3. The Dynamic Window Approach (DWA) uses a window of achievable velocities to determine the best velocity command for a robot to reach its goal while avoiding obstacles. it works by evaluating the robot's achievable velocities based on its current velocity and the maximum acceleration and deceleration capabilities of the robot. A subset of these achievable velocities is then selected based on the proximity to obstacles in the environment, resulting in a smaller subset of "safe" velocities. From this set, the DWA method selects the velocity that maximizes the distance to obstacles while moving towards the goal [53].

One advantage of the DWA method is that, similarly to the APF method, it is relatively simple to implement and computationally efficient, making it a popular choice for real-time applications, and often used as the local planner of state-of-the-art SLAM navigation systems [69]. The DWA method is also relatively robust to sensor noise and can be used with a variety of sensor types. some of the limitations include an assumption that the robot has precise control over its motion, which may not be true for all robot platforms. Additionally, the method may struggle in very cluttered or complex environments with many obstacles in close proximity to the robot, where there are few "safe" velocities to choose from. Finally, only the position of the obstacles is taken into account, not their velocity. This makes navigation in dynamic environments harder. Some work has been done to mitigate that limitation by predicting possible future trajectories of the obstacles [70], [69], [71].

4. Velocity obstacles is a method that assumes a priori knowledge of the motion of mobile obstacles. The approach consists of characterizing, among the permissible velocities, respecting the kinematic and dynamic constraints of the system, those leading to a

potential collision with the obstacles in the future up to a certain time horizon [72]. Several extensions of this work have emerged, such as the consideration of more complex approximations of the future model of obstacles [73], an attempt to define the "right time horizon" necessary to guarantee the safety of motion [74], and combining it with other methods [75]. However, the velocities of the obstacles can be hard to predict in highly dynamic environments.

One common limitation of all the methods presented above is that, since they are local planners, none offer a guarantee that the goal will be reached. They offer pretty good approximations, but navigating through complicated environments might require a global planner to avoid local minima. map-less solutions for global planners exist, but are more limited. As presented in section 2.4.1, Bug Algorithms can be used for map-less global path planning. They have the advantage of offering the same guarantee of finding the path, if it exists, as the current state-of-the-art planners like A* [51], RRT [52], and their modern variants. The major limitation is that they have no guarantee to find the optimal path (the shortest). This is a whole field of research, with recent works trying to make use of swarms [76] [77] or find better heuristics [78] [79] to improve towards more optimal trajectories.

A combination of the classical methods and Bug Algorithms presented above can be used to perform navigation with obstacle avoidance while having a stronger guarantee to reach the goal. Since the present work addresses the problem of map-less navigation with obstacle avoidance and aims to reduce the computational load of such methods, a more exhaustive literature review is done in section 3.2 to cover research focusing on achieving this goal.

3.1.2 Deep reinforcement learning for navigation

Another approach to perform navigation without a global map, which has been used increasingly in recent works, is deep reinforcement learning [61]. Deep neural networks can be used to learn a policy for navigating a robot to a goal while avoiding obstacles, relying only on the robot's sensors to perceive the environment. There are multiple approaches with different learning algorithms that were proposed:

1. A popular learning algorithm for UAV autonomous navigation is Deep Deterministic Policy Gradient (DDPG), a model-free off-policy actor-critic algorithm, because it is well adapted to control tasks, allowing for continuous state spaces and offering greater stability for learning [80]. In 2020, Bohao and Yunjie proposed an improved DDPG algorithm to perform path planning for UAV ground target tracking [81]. They showed in simulation that their method can allow the UAV to keep tracking and avoid obstacles

efficiently. However, when put in environments with more obstacles, the success rate of their algorithm dropped to 67.5%, and it would be expected to be even lower with dynamic obstacles. In [82] DDPG was combined with imitation learning and achieved a success rate close to that of a human pilot, but the controller’s success rate in obstacle avoidance also dropped to 72% when tested in new environments.

2. Deep Q-Networks (DQN) require discrete state and action spaces, but also demonstrated a high stability in learning various control tasks [16]. Other work proposed in [83] a framework to locate a missing human after a natural disaster using a DQN to navigate. They implemented it on a real drone to test it. However, they assumed the drone had perfect knowledge of its environment and performed the evaluation on a chess-board-like environment, with discretized possible movements and obstacle positions known in advance. In [84], the authors studied multiple variants of DQN, such as double DQN [18], dueling DQN [19], and Double Dueling DQN, and continuous actor-critic algorithms like TRPO, PPO and ACKTR. They trained the RL algorithms in 3 environments and compared them with a human pilot. All the algorithms learned to navigate, although only the continuous were able to reach the goal in harder environments. Most importantly, all the algorithms failed to reach the goal when the environment was too different from the training conditions (81% more obstacles in this case).
3. Another approach makes use of Recurrent Neural Networks (RNNs) and Long Short-Term Memory (LSTM) to build a representation of the UAV’s surroundings. This allows the UAV to keep in memory the previous states. In 2021, a DRL method was proposed for enabling a UAV quadrotor, equipped with a monocular camera, to autonomously avoid collisions with obstacles in unstructured and unknown indoor environments [85]. The method is based on RNN architecture and Temporal Attention, which allows the UAV controller to store relevant observations gathered over time to make better obstacle avoidance decisions. The authors were able to show obstacle avoidance behavior in difficult environments (with dynamic obstacles) in simulation. However, the controller wasn’t able to perform obstacle avoidance indefinitely as the principal metric they use to evaluate their method is the number of steps taken before a collision happens.

There have been works achieving performance closer to classical state-of-the-art methods, but they often assume perfect knowledge of the obstacles [83], are evaluated in simulators that don’t necessarily simulate the complete drone dynamics [81] [86] or in environments with

Table 3.1 Comparison of the RL approaches with state-of-the-art classical methods for autonomous navigation. ✓ means the proposed approach achieved or exceeded state-of-the-art results with classical solutions. ✗ means it didn't reach state-of-the-art results in that aspect (collisions still happening, simplifying assumptions or environment with sparse obstacles for instance).

Reference	High Success Rate	Low Computational Load	Realistic Simulation / Real Robot Experiments	Without Strong Simplifying Assumptions	Difficulty of Environment
(Li, 2020)	✓	✓	✗	✗	✗
(He, 2020)	✓	✓	✓	✓	✗
(Singla, 2021)	✗	✗	✓	✓	✓
(Bouhamed, 2020)	✓	✓	✗	✗	✗
(Shin, 2019)	✗	✓	✓	✓	✓
(Pham, 2018)	✓	✓	✓	✗	✗
(Salvatore, 2020)	✗	✓	✗	✗	✓
Ours	✓	✓	✓	✓	✓

sparse obstacles [82]. Further, another major limitation of RL methods is that they are a lot less robust than classical methods, as it is hard to evaluate them in all possible conditions, they are often tested in environments too similar to their training data [84] or simply fail to be robust in obstacle avoidance in harder scenarios [81] [82] [85]. Finally, other works have used DRL in combination with classical methods and have shown promising results. Greatwood C. and Richards A. G. proposed to combine RL and model predictive control (MPC). They implemented MPC, which performs the navigation and obstacle avoidance, along with RL, which is used to guide the UAV through complex environments [87], showing both modest computing requirements and robust navigation. In [88], the authors proposed to use DRL to learn motion planning, and integrate their algorithm with an existing obstacle avoidance pipeline to navigate, with a goal to reduce the need for a high-precision map in current grid-based planners. Their method can find paths with less computing cost than their baseline but similar performance, with a 2x reduction in CPU consumption and 7x increase in path planning frequency.

In brief, Table 3.1 compares the presented work to classical state-of-the-art methods and shows that there are still improvements needed for DRL to surpass the classical obstacle avoidance methods in autonomous navigation. Still, it is an active research field with increasingly better results, and recent work shows potential for combining it with state-of-the-art solutions for more robust and efficient navigation.

3.2 Runtime optimization

In this section, we go over the state-of-the-art research that aims to reduce the computational resources needed to apply the previously presented methods to real-world robots. There has been progress in both hardware acceleration and runtime optimization. Many SLAM and obstacle avoidance algorithms have been proposed, as presented in the previous sections. However, running these algorithms in real-time can be computationally intensive, requiring powerful CPUs and/or GPUs, or even needing multiple seconds to compute (for e.g., in the case of trajectory optimization) [89]. Mounting such CPUs and GPUs on a big drone such as the Skydio R1 is feasible [90]. However, these solutions cannot be applied to very small drones/UAVs as those presented in [91] or [92]. This is because of the constraints on the integration as well as the extremely limited power budget available on these miniature vehicles. For example, the budget for a stable flight in nano/pico drone is around 100 mW [92], which is an order of magnitude lower than the power consumption of embedded CPUs.

3.2.1 Hardware acceleration

A popular method to improve the computing speed of the navigation methods is to implement them into specialized hardware or even design context-specific chips for the tasks [93]. Different computing platforms are used to integrate methods such as path planning, motion planning or Visual Inertial Odometry (VIO):

1. **FPGA-based motion planners:** A FPGA is a chip programmed by the customer, after it has been produced. In [94], the authors studied motion planning on a chip, describing a process for constructing robot-specific circuitry capable of generating motion plans for a robotic arm up to three orders of magnitude faster than the existing methods at the time. They then extended their work by implementing it on an FPGA in [95], where they showed that the accelerator improved performance by the amount predicted in their previous work. However, the flexibility of this is limited, as it would require the process to be done again when applying it to other robots or UAVs with different motion equations and hardware.
2. **Application-Specified Integrated Circuit (ASIC):** Another method to reduce energy conception and footprint is to fully integrate the navigation system on a custom chip to avoid off-chip processing and storage. In [96], an energy-efficient accelerator for VIO that enables autonomous navigation of miniaturized robots was proposed. The entire VIO system was integrated on-chip, able to process 752 x 480 stereo images at 20 frames per second (fps) with an average power consumption of 2mW.

3. Digital Signal processor (DSP): [97] demonstrated the possibility of using a single DSP core to implement a real-time flight controller software (FCS) in a real-time Hardware-In-the-Loop (HIL) simulation. in [98], the authors used a DSP to perform autonomous navigation of agricultural mini-unmanned vehicles. To the best of our knowledge, no recent work has been done to design accelerators for modern obstacle avoidance methods using DSPs.

Deploying single-accelerator-based computing platforms to address the high workload of complex robotic applications remains a challenge mainly for two reasons: different robotic tasks require different accelerators, and managing multiple accelerators simultaneously is difficult [99]. Authors in [99] proposed a robotic runtime framework to manage dynamic task executions on mobile systems with multiple accelerators. They successfully performed autonomous navigation with localization (25 Hz), obstacle detection (3 Hz), path planning, large map generation and scene understanding within an 11W computing power envelope. However, since models and objective functions differ between robotic systems and tasks, the range of applicability of the contribution is still limited to the specific robot used in this work.

3.2.2 Algorithmic optimization

Another approach to reduce the workload of autonomous navigation is to design efficient, faster or adapting algorithms. This section will go over the research that tries to accelerate the processing, or reduce the computing resources, of obstacle avoidance pipelines using algorithmic optimization. The presented work is summarized in Table 3.2.

Table 3.2 Comparison of the research reducing the resources needed for an autonomous navigation pipeline

Contributions	Approach	Targeted step in the navigation	Outcome
[100]	Local planning	Path planning	Performed obstacle avoidance on a real drone within a 11W computing envelope.
[101]	Limiting computing time	Path planning	Between 8.9x less to 2.1x less energy consumed on Micro-Aerial Vehicles (MAV).
[102]	Dynamically adjusting resources and precision	Perception and path planning	Reduced mission time by 4.5x, energy consumption by 4x and CPU utilization by 36%.
[103]	Efficient heuristic for graph-based path planning	Path planning	Reduced path planning time by 15x.

in [100], a dual-range planning horizon method is proposed to address the problem of high-speed autonomous navigation with limited sensing and computation. The contribution relies

on accelerating the computation of the trajectory at a local level. Current state-of-the-art SLAM methods construct a global map as they explore their environment, and perform their path planning algorithm on the entire map, which requires a lot of computation [42]. In [100], rather than accumulating consecutive sensor measurements to form a global map, only the latest sensor measurement is used to create a local map, on which the local planning is performed. However, no comparison with a baseline is done to quantify the improvement in terms of computation time for the trajectory or the entire pipeline. A global planner is also assumed in case the proposed local planner fails to find a valid path to the goal.

There is also work that has been done for situations with ultra-low energy vehicles, where the energy needed for the computing resource is in the same order of the actuation energy needed to perform navigation. In [101], the authors studied this class of problems with vehicles such as insect-size flyers or high-endurance autonomous blimps. They proposed the Compute Energy Included Motion Planning (CEIMP) algorithm, which is an anytime planning algorithm that stops when it estimates further computing will require more computing energy than potential savings in actuation energy. When the energy to compute during one more second is equal to the energy to move 3 more meters, their proposed algorithm outperforms their average baseline 99.7% of the time. However, the expected gain gets smaller when the energy needed to move increases. CEIMP outperformed their baseline 90.6% of the time when the energy to compute one more second is equal to moving 1 meter, and 74% when it is 10 times less. In different test cases, the energy used went from 8.9x less to 2.1x less than the total energy used by the baseline.

Another approach proposed by [102] in 2021 is to exploit the environment heterogeneity to dynamically adapt the precision of the navigation pipeline. Current SLAM methods make static, worst-case assumptions to ensure obstacle avoidance, which results in lower velocity throughout the mission, even if parts of the environment are obstacle-free [104] [100]. The authors propose a spatial-aware algorithm, RoboRun, to dynamically adjust the computational resources of the robot depending on its environment. Based on a time budgeting algorithm and constraint solver, RoboRun chooses a combination of values for parameters such as the LiDAR's pointcloud precision, the path planner precision or the volume considered for the path planning, to reduce the computational load when possible, and increase precision when needed. The algorithm achieved a reduction of the energy consumption by 4x, mainly by reducing mission time, since the power for actuation is much greater than computing power for drones able to perform SLAM [105]. It also achieved a reduction of 36% in CPU utilization freeing up computational resources for other tasks. One of the limitations is that the time budgeting algorithm and the constraints of the solver are highly dependent on the dynamics of the vehicle and need to be redesigned to be applied to a different robot. In 2022, authors

in [103] proposed a fast motion planning method for UAVs to navigate through unknown environments cluttered with obstacles. The authors suggest a graphical representation of the planning problem, where the input space is converted into a lattice input set and the corresponding UAV system states are represented as vertices in a directed tree. An off-the-shelf path planner was then deployed onto this framework for optimal path search towards the goal. The paper also proposed an efficient heuristic calculation technique through model reduction by relaxing the dynamics, along with a heuristic learning technique that adjusts the heuristic function with each new vertex visit towards the goal. The proposed technique was able to reduce the planning time by a factor of 15 when compared to solving the optimization problem with an off-the-shelf solver. However, the work has the same limitation as in [102] that the solution is dependent on the dynamics of the vehicle. The research also only considered the planning time, which means the reduction in computing time will be less than what is obtained when we consider the entire navigation pipeline. Finally, the authors acknowledge that the method needs to be evaluated in the presence of disturbances and model uncertainties.

3.2.3 Algorithm/hardware codesign

A third method to accelerate or reduce the impact of the intensive navigation pipelines is to take both approaches and combine them. In this subsection will be presented work that has been done on Algorithm and hardware Co-design.

One of the critical part of path planning is Collision detection, which can take up to 99% of the entire planning time [106] [107]. There is a lot to gain by parallelizing this part of the process and using specialized hardware as presented in 3.2.1. Another bottleneck is that, while individual collision checks can be parallelized, the end-to-end speedup of hardware acceleration is limited due to the difficulty in parallelizing the path search algorithms. Search algorithms like Dijkstra, A*, and their variants and extensions exhibit little to no parallelism for path planning [108]. This is where algorithm and hardware Co-design can be extremely useful, as it can act on both of those limitations. In [109], a hardware-directed algorithm for finding a path for a mobile robot is presented. The algorithm was mapped to a Xilinx FPGA device to show that it could be operated to use the Euclidean distance transform of the binary image of an environment and compute a collision-free path in less than 3ms, meaning it can process images at a video rate, in real-time. In 2022, [110] proposed RACOD, an algorithm/hardware co-design for mobile robot path planning. It consists of a hardware accelerator for collision detection and an algorithm extension to perform runahead path exploration, which allows them to predict future path explorations and proactively computes

its collision status ahead of time. They evaluated RACOD on the *freiburg campus*, which is a 3D environment available in the OctoMap 3D scan dataset [111], achieving a speed-up of 31.3x for UAVs and up to 41.4x for self-driving cars, which is the highest speedup in the research presented above. They also studied the effects of using the hardware accelerator or the algorithm alone on a stationary robotic arm, and showed a 3.4x increase for the hardware, and an 8.6X/2.9x increase with 32/128 CPU/GPU threads for the algorithm. Those results are closer to what was achieved in other related works [95] [99] [96] [112], and it is the combination of both that outperforms other approaches. One of the limitations of the paper is that the acceleration algorithm was implemented on powerful CPUs or GPUs, which might not be available on a smaller vehicle, where the problem of computing a path in real time is most critical. Additionally, the gain of the hardware is limited compared to other works when it is not used with the accelerating algorithm.

Recent work has also been done to facilitate the hardware acceleration of autonomous control. In [113], the authors created RoboX, an end-to-end solution which exposes a high-level language to roboticists, to allow them to express the physics of the robot and its task in a form close to its mathematical expressions. Their solution then automatically maps the specification to a programmable architecture. The authors showed a speedup of 29.4x using their accelerator over an ARM Cortex A57. Compared to a Tegra X2 GPU, they showed a 7.8x higher performance-per-watt. Their contributions rely on the fact that many robotic applications are formulated as constrained optimization problems, which can be solved online using Model Predictive Control (MPC), but this is not always the case, and a lot of off-the-shelf drones still use PID controllers since MPC requires very high expertise and problem-specific engineering [114].

3.3 Conclusion

In conclusion, autonomous drones have the potential to revolutionize various industries such as package delivery, search and rescue, and surveillance. However, their limited onboard energy remains a significant challenge that has been addressed through the development of faster algorithms to improve robot navigation. Although hardware-based acceleration techniques are available, they are typically applicable to specific robots and tasks and are not very flexible. Algorithm-based acceleration techniques proposed by the literature are highly dependent on the dynamics of the specified robot, making them also not widely applicable.

In recent years, work has been done to combine both hardware and algorithm acceleration techniques, but these often use power-consuming GPUs that are not available in smaller drones or target specific control methods like model predictive control, which isn't always used

on off-the-shelf drones. Another potential solution to accelerate the autonomous navigation pipeline of drones is through offloading computing tasks to more powerful hardware. However, this approach can be unpractical in certain situations, particularly if the latency of the connection between the drone and the offloaded computing hardware is unreliable. Any delays in the processing of information could lead to slower reaction time in the drone, which could have serious safety implications. Therefore, while offloading computing could be a viable option in some cases, it should be approached with caution and careful consideration of the potential risks and limitations.

To address these limitations, reinforcement learning can be used to accelerate the obstacle avoidance pipeline, which has the advantage of being agnostic to the dynamics, hardware, control method, and task. Overall, this literature review highlights the need for flexible and scalable techniques that can be applied to a wide range of drones and tasks, and reinforcement learning appears to hold great promise in this regard. In the following chapters, we present how to use reinforcement learning to dynamically adapt to the heterogeneity of the environment, and accelerate an obstacle avoidance pipeline. We propose a methodology to train an agent in a simulated environment using a realistic simulator in chapter 4. In chapter 5, we evaluate the proposed agent to show its effectiveness.

CHAPTER 4 RL-BASED ENVIRONMENT-AWARE MAPLESS NAVIGATION

This chapter presents the methodology developed to design and train a DRL agent to reduce the processing time of mapless navigation with obstacle avoidance. For this, the project was divided into two main parts. Section 4.1 presents the environment design that was done to train a reinforcement learning algorithm for smart sensor selection for more efficient computing. This includes the decisions made to choose the learning environment and the simulator. Two maps of different complexities are introduced. Then, the modeling of the environment components such as the UAV, the sensors, and the control frequency is described. Section 4.2 will go over the methods used to design, train, and evaluate the agent. The design of the observation and action spaces, reward functions, and agent architecture will be presented.

4.1 Environment Design

4.1.1 Defining the task

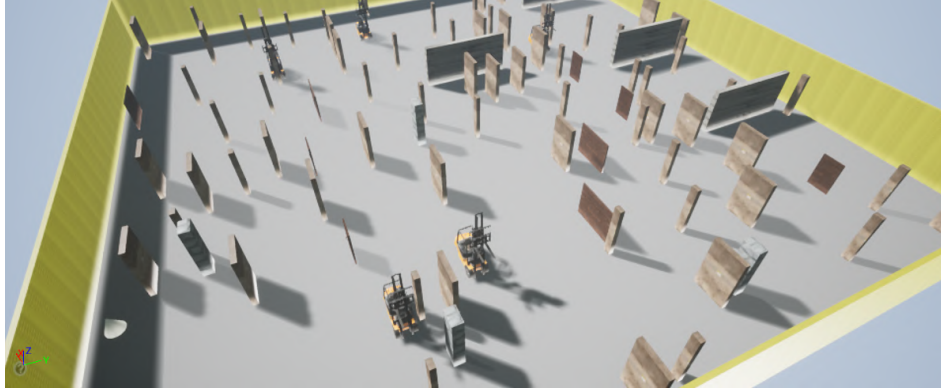
The task is for a small UAV to navigate in an environment from a starting position to a target position, or *goal*. The drone will need to avoid obstacles as it navigates without the use of a map. The objective of the RL agent is to choose in a smart way which data from the sensors is important, and which data can be safely ignored for this task to make the navigation pipeline as efficient as possible, in terms of computational resources and processing speed. Traditional navigation methods presented in the literature review ignore the environment heterogeneity and do not have a high-level understanding of the situation, making worst-case assumptions to navigate safely. For this task, a well designed RL agent will be able to choose what information from the sensors is important and concentrate its computational resources accordingly.

To perform autonomous navigation in realistic environments, two different arenas are introduced: An arena with a dense distribution of static obstacles with different shapes, and another with both static and dynamic obstacles, placed in a way so that the environment is heterogeneous (easier and harder zones). In both arenas, the obstacles can take multiple shapes to avoid overfitting, and different shapes are used to put the agent in situations where its sensors might be less reliable (narrow or complex shapes like the front of a forklift, as shown in Figure 4.2) to make it more robust. In Arena 2, the dynamic obstacles move

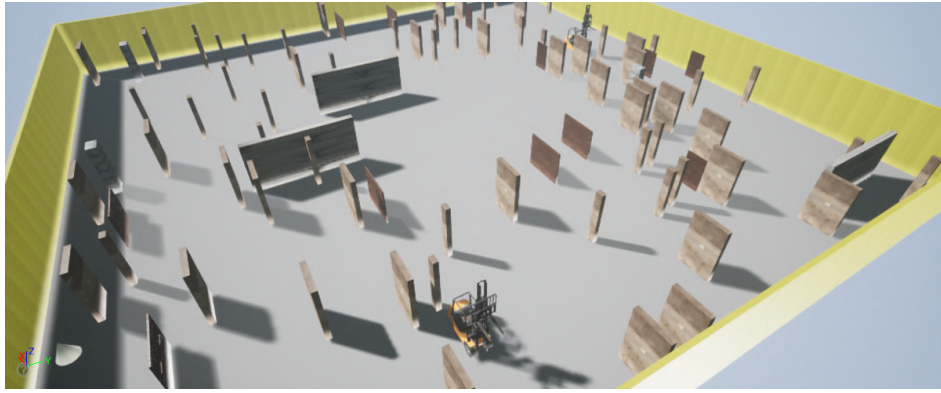
randomly with a velocity changing over time in the interval:

$$v \in \{\mathbb{R}^2 \mid v_1 \in [-v_x, v_x], v_2 \in [-v_y, v_y]\} \quad (4.1)$$

Where v_x and v_y can be set depending on the target max velocities. Note that there is no vertical velocity, as the drone will be forbidden to just fly over the obstacles, which would make the obstacle avoidance task too easy. This can often be the case in indoor environments, where the vertical space is limited.



(a)



(b)

Figure 4.1 Different arenas introduced for the task

To navigate, the drone will receive information about the environment from its sensors (2D LiDAR, IMU, etc.) and send velocity commands to the flight controller simulated by AirSim [115], which then sends motor commands to reach the target velocities. The sensors used are described in more detail in Section 4.1.3. In each episode, a new random goal position is generated in the 100x100 meters arena, and obstacles are spawned randomly with some rules introduced to have control over the density and heterogeneity of the environment. First,

parameters for the minimal distance between two obstacles and the amount of total obstacles can be set. Second, regions of variable size can be defined so that no obstacles spawn in them, creating easier zones, and harder zones where the obstacles are more cluttered. The agent successfully fulfills its task if it reaches the goal within 1 meter. It is not expected to reach the goal exactly, simply because the flight controller of the drone is not precise enough to reliably reach a target position with an error of less than 1 m. The agent fails its task if the drone collides with an obstacle, which is registered by the UE4 physics engine, or exceeds the time limit, which is set to a fixed number of steps, discussed in Subsection 4.1.3.

Arena Setup

As discussed above, a random generator is used for each episode to generate a new map. Here is the summarized generating process:

- The number of static and dynamic objects is determined randomly between a given interval $[n_{min}, n_{max}]$;
- The object type is selected randomly from all the different possible objects (see Figure 4.2 for examples);
- The position of each object is generated by selecting a random valid position within the arena, re-sampling if there is already an obstacle within the set minimal distance. The valid positions within the arena are all positions within 100 x 100 m, except the easier zones set in the heterogeneous environment, and;
- A random position is sampled from the remaining free space as the goal.



Figure 4.2 Different obstacles spawned in the arenas

Navigation Stack

To perform map-less navigation, two algorithms were implemented: The Tangent Bug Algorithm, presented in Section 2.4.1, and the Dynamic Window Approach (DWA), presented in Section 3.1.1. The architecture for the whole pipeline is illustrated in Figure 4.3. The Tangent Bug is responsible to act as the global planner and produce a closer, local goal to reach. The DWA will then take that goal as input and locally plan a trajectory to reach it while avoiding obstacles. It outputs velocity commands that are sent to the drone controller in AirSim.

The DWA was initially proposed for wheeled robots, and thus originally considered curved trajectories with fixed linear and angular velocities (v, ω) . This is because those trajectories are the easiest to reproduce with non-holonomic wheeled robots, by simply fixing the wheels to different velocities. However, since drones move in a very different way than wheeled robots (6 degrees of freedom) the method can be modified to be better adapted to the drone dynamics.

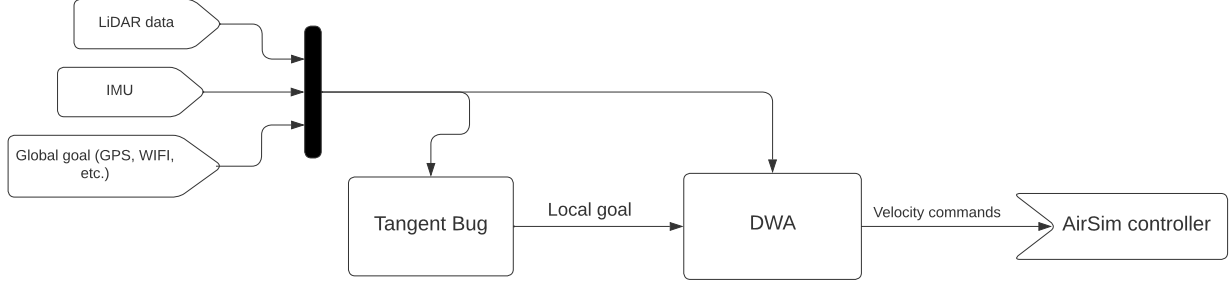


Figure 4.3 Map-less navigation pipeline

For this task, we propose to modify the original dynamic window approach by using a different velocity space, so that the possible trajectories were closer to those of a UAV. The velocity space is instead the set of velocities (v_x, v_y) to consider not only the velocities in front of the UAV but in all directions. As mentioned earlier, the vertical velocity will not be controlled by the navigation pipeline to force the drone to avoid obstacles and not fly over them. The modified algorithm is presented in Algorithm 4.

Algorithm 4 Modified Dynamic Window Approach

Input: Goal: relative coordinates computed by the *tangent bug*
 robotVelocities: current velocity

```

1: bestCost =  $\infty$ 
2: desiredV = calculateV(Goal)
3: laserscan = readScanner()
4: allowable $_{v_x}$  = generateWindow(robotvelocities, Goal)
5: allowable $_{v_y}$  = generateWindow(robotvelocities, Goal)
6: for  $v_x \in \text{allowable}_{v_x}$  do
7:   for  $v_y \in \text{allowable}_{v_y}$  do
8:     headingTerm = progression( $v_x, v_y$ , goal)
9:     safetyTerm = closestObstacle( $v_x, v_y$ , laserscan, goal)
10:    cost = costFunction(headingterm, safetyTerm)
11:    if cost < optimal then
12:      best $_{V_x}$  =  $v_x$ 
13:      best $_{V_y}$  =  $v_y$ 
14:      optimal = cost
15:    end if
16:  end for
17: end for
18: return best $_{V_x}$ , best $_{V_y}$ 
  
```

The functions of the pseudocode do the following:

- **calculateV()** Takes the goal as input and returns the required velocities to go directly towards it. It can be done in constant time: $\mathcal{O}(1)$.
- **readScanner()** Returns the point cloud of the LiDAR, also in $\mathcal{O}(1)$.
- **progression()** Simulates the dynamics of the UAV with the given velocities to estimate the real progress toward the goal in the next timestep, also in $\mathcal{O}(1)$.
- **closestObstacle()** Computes the minimal distance to any obstacle that will be reached following the given velocities. This is proportional to the amount of points p in the point cloud to evaluate: $\mathcal{O}(p)$.
- **costFunction()** Computes the cost of the considered velocities with the previously determined progress and distance to obstacles, in $\mathcal{O}(1)$. The two terms are weighed by parameters that need to be tuned to achieve the correct behavior (value progress and safety correctly).
- **generateWindow()** returns the velocities that are achievable with the current velocity of the drone, in $\mathcal{O}(nm)$, where n and m are determined with the discretization methods described below.

Since the proposed velocity space is continuous, the searched velocities need to be discretized to make sure the algorithm can run in real-time. The method used for discretization needs to systematically cover the space in $\mathbb{R} \times \mathbb{R}$. To do this, we consider a maximum velocity v_{max} that the UAV can safely reach. Since it can go in any direction, this gives us a circle with radius v_{max} to cover. We can then discretize all the directions by dividing this circle in n equal regions, where the angle of each region represents a possible direction. Finally, to cover the area of the circle, we can consider m different intermediate velocities between $[0, v_{max}]$. An illustration of this discretization is done in Figure 4.4.

If we consider a number of points in the point cloud p , this gives us a time complexity of $\mathcal{O}(nmp)$, meaning the number grows polynomially, instead of the exponential amount of possible trajectories in global planning. However, quadratic growth (with n and m) can still be limiting when the process needs to be executed in real-time. The more precisely the velocities are discretized, the more likely the DWA is to find smoother trajectories to avoid the obstacles, but the lower its reaction time gets. A compromise needs to be found. For this, the n and m values will be gradually increased for more precise control until the reaction time affects the behavior (i.e. the drone starts to collide with obstacles).

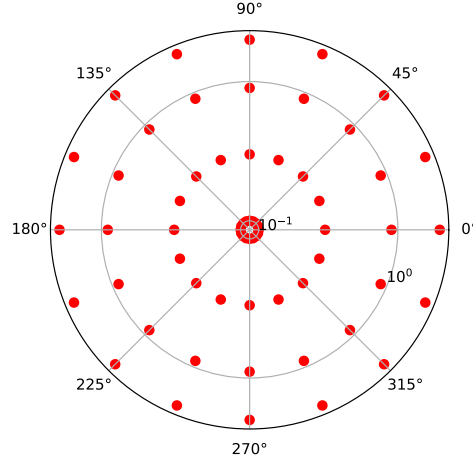


Figure 4.4 Example of discretization of the velocities with $n = 16$ and $m = 4$ for the modified Dynamic Window Approach. The drone is in the center and can go in 16 directions at 4 different speeds.

4.1.2 Reinforcement Learning Environment

Training an agent to perform a task usually involves a lot of trial and error. A well-established way to apply reinforcement learning is to use a simulator: this lets the agent explore different strategies and try them in a huge amount of situations, something that can be really expensive, but also dangerous, in the real world. Multiple simulators exist for different robotics tasks (e.g. Gazebo [116], CARLA [117], Airsim [115]), and a lot of learning environments were proposed to use these simulators for reinforcement learning: GymFC [118], PyRobot [119], Air Learning [120] and more.

Table 4.1 Comparison of features present in RL research infrastructures. ✓ means the feature exists. ✗ means missing feature or requires significant effort for the end-user to enable that feature. [120]

Features	UAV Specific			UAV Agnostic					
	Air Learning	AirSim	GymFC	CARLA	Gazebo	PyRobot	Robot-Grasping	ROBEL	SenseAct
Photorealism	✓	✓	✗	✓	✗	✓	✓	✗	✗
Domain Randomization	✓	✗	✗	✓	✗	✓	✓	✗	✗
OpenAI Gym Interface	✓	✓	✓	✓	✓	✓	✓	✓	✓
RL Algorithm Exploration	✓	✓	✓	✓	✓	✓	✓	✓	✓
ML Backend Integration	✓	✓	✓	✓	✓	✓	✓	✓	✓
UAV Physics	✓	✓	✓	✗	✓	✗	✗	✗	✗
Energy Modelling	✓	✗	✗	✗	✗	✗	✗	✗	✗
Compute Benchmarking	✓	✗	✗	✗	✗	✗	✗	✓	✓
RL policy Deployment on Robot	✓	✓	✓	✓	✓	✓	✓	✓	✓

Multiple components play a part in making a good environment for the task considered in

this work. First, a UAV physics model is necessary to make the simulator realistic, allowing for a smaller gap between simulation and reality. In fact, a simulator with incorrect modeling of the dynamics of the drone might cause a large gap between the state transitions when the agent is learning and when it is deployed, which can lower significantly its performance. Regarding this, UAV-specific environments like AirSim are better suited than more general simulators, although some of them (e.g. Gazebo) do model UAV physics. Second, good metrics are desirable, as this makes it easier to evaluate the methods, in a quantitative way. In the context of performing efficient map-less navigation with obstacle avoidance, some good metrics to measure could be the computational load and energy spent. Third, domain randomization is also a desirable feature, making the agent much more robust by exposing it to different environments at every episode to avoid overfitting a particular set of situations. Finally, an important component to consider when choosing the environment is how the reinforcement learning is exposed to the user. For example, having a *gym* interface [121] makes it easier to design the environment dynamics, and RL algorithm exploration such as *stable-baselines* [122] makes it much faster to experiment with different learning algorithms and policy architectures.

Gym interface: The gym interface, proposed by OpenAI in 2016 [121], is a toolkit for reinforcement learning research. The basic idea is to expose a common interface for a collection of benchmark problems. It includes multiple environments recognized for benchmarking, but also a standardized interface to create custom environments for any task. This allows to concentrate the research on designing and training the algorithms without spending too much time on low-level details of environment implementation.

Stable-Baselines3: Stable-Baselines3 [122] is a Python library implementing DRL algorithms that follow a consistent interface and are accompanied by their respective documentation. This makes it much faster to train and compare different RL algorithms, and find the one most adapted to the learning task. The library also allows to easily change the neural networks used for the policies (and the value functions in actor-critic architectures). Using standardized implementations of RL algorithms is especially important since even small implementation details can have a great effect on performance [123], making it much harder to compare with the literature or reproduce results.

The choice of the Reinforcement learning environment was based on the criteria presented previously, and the simplicity of using the environment itself. Table 4.1 shows a comparison of the different simulators/RL environments considered. One that is worth mentioning is Gazebo, which is well-established as a realistic simulator for robot tasks. However, a lot of the mentioned criteria, such as RL interfaces and energy modeling, are not immediately

available and require implementation. A good choice that emerges from the comparison based on our criteria is Air Learning, which is an environment built using the AirSim simulator and has all the desirable features for the task of this thesis. One important thing to note here, is that while Air Learning is a good choice at the time of beginning the project, simulation for RL is a very active domain, benefiting from contributions on a regular basis, making it a rapidly changing field. This is why some newly proposed simulators that are promising and might check all the mentioned criteria, such as Isaac Gym and Aerial Gym [124], might be missing in the comparison for the RL environment choice.

Air Learning: Air Learning [120] is an open-source simulator and gym environment for DRL research, specifically made for resource-constrained aerial robots. It is based on the AirSim [115] simulator, that runs using Unreal Engine 4 (UE4) [125], to provide a high-fidelity simulation and dynamics of the UAVs. The Air Learning tool-set can be divided in four main components: first, it offers off-the-shelf domain randomization with multiple configurable parameters (arena size, type, number and density of obstacles, goal positions, textures, etc.) with an environment generator built on UE4, which is a photo-realistic game engine. Second, the environment generator is integrated with AirSim, OpenAI gym, and stable-baselines to quickly prototype different RL algorithms. Third, its back-end uses tools like Keras/Tensorflow to enable easier policy exploration and design. Finally it offers quality-of-flight (QoF) metrics such as energy consumed or average trajectory length. The different components and features explained above are summarized in 4.5.

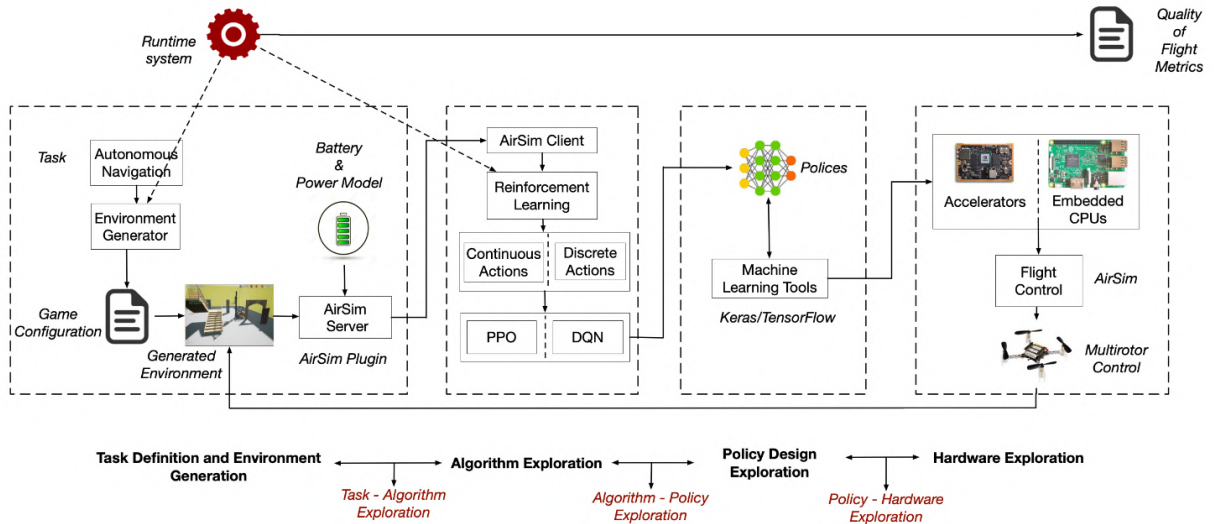


Figure 4.5 Main components of Air Learning [120].

4.1.3 Environment and MDP modeling

To ensure that the agent can learn a strategy that can be potentially transferred into the real world, the simulated environment needs to be as close to reality as possible, and the configured sensors and drone dynamics need to mimic existing hardware. Fortunately, AirLearning offers realistic sensors with artificial noise by using an AirSim plugin to simulate them. Those sensors can be configured with the specifications of real sensors. This includes cameras, LiDARs, GPS, IMU, and more. The model of the drone is based on the DJI F450 [126], designed for aerial photography. It has an IMU, a GPS and a lightweight LiDAR. To have realistic data for the LiDAR, its parameters are configured to model the YDLIDAR X2 [1], a lightweight 2D laser scanner with the following specifications:

Table 4.2 YDLIDAR X2 performance specifications [1]

Type	Min	Typical	Max	Unit	Remarks
Ranging frequency	-	3000	-	Hz	Ranging 3000 times per second
Motor frequency	5	6	8	Hz	
Ranging distance	0.12	-	8	m	Indoor environments with 80% Reflectivity
Field of view	-	0-360	-	Deg	
Systematic error	-	2	-	cm	Range $\leq 1\text{m}$
Relative error	-	3.5%	-	(-)	$1\text{m} < \text{Range} \leq 6\text{m}$
Tilt angle	0.25	1	1.75	Deg	
Angle resolution	0.60 (frequency at 5Hz)	0.72 (frequency at 6Hz)	0.96 (frequency at 8Hz)	Deg	Different motor frequency affects angle resolution

The control frequency, at which the agent will make decisions, is also a design choice that can depend on different factors. Since the maximum frequency of the LiDAR motor is 8 Hz, the rate at which the drone will receive sensory input has an upper boundary. In general, a higher frequency would mean a faster reaction time, but the process needs to run inside the time interval of the control frequency, which gets shorter as frequency increases, to compute a velocity command in time. This is why some computationally expensive processes, like global planning, are typically done at a much lower frequency than low-level control. In map-less navigation, the most complex part of the navigation pipeline is the obstacle avoidance stack (complexity of $\mathcal{O}(nmp)$). For example, the dynamic window approach implementation on the Robot Operating System (ROS) has a default control frequency of 20 Hz [127]. Since this value is still higher than the sensor’s frequency, a sensible choice of control for the MDP of this task is to take the maximum frequency of the LiDAR, at around 8 Hz.

POMDP

Based on the previously described task and modeling of the environment, we can define the Partially Observed Markov-Decision-Process (POMDP) for this task:

- **O:** The observations are represented by all the sensory inputs of the drone (LiDAR, IMU, and goal data). This means the agent will have access to a partial observation, and will not know all the information about the state (e.g. positions of all the obstacles), which is more realistic.
- **A:** The actions the agent will be choosing are subsets of all the sensory Inputs of the drone. Since considering all the points of the point clouds is the critical operation in the DWA, the goal of the agent is to find the important points to consider and the ones we can ignore to still perform obstacle avoidance. The action space design will be discussed in more details in Section 4.2.
- **R(s,a,s'):** In essence, the reward function will reward behaviors that lead to obstacle avoidance while using the minimum amount of points in the point cloud. Reward engineering is often one of the most important parts of training an agent and is discussed in more detail in Section 4.2.
- **P(s'|s,a):** The transition probability function captures the dynamics of the environment, and is modeled by the AirSim simulator, which will handle the complete state (not just the observations) to simulate the dynamics and produce the state transitions.

4.2 Design of the DRL Agent

Multiple RL algorithms can be used to learn the task, as presented in Section 2.2. Two algorithms were selected to be evaluated in this thesis. First, PPO, as it is a well-established algorithm known to be applicable to multiple problems and generally requires less tuning. Second, A2C, as it was shown to surpass the state-of-the-art for a lot of games in the Atari Learning Environment (ALE) [24] [15]. The implementations of the RL algorithms from stables-baselines are used, to help with quick architecture exploration and use standardized implementations to compare with the literature.

4.2.1 Action and Observation Space Design

The goal of the agent is to choose the set of relevant points from a point cloud and only give those as input for the DWA, to reduce its computational payload. This means the other

points will be discarded. For example, it is crucial to take incoming obstacles into account to react in time, while objects that were just passed but are still detected by the LiDAR will likely not cause a collision anymore. Thus, an intuitive action space for the agent might be to make a decision for each individual point if it is to be used. However, training an agent to evaluate a single point and perform inference for each point of the point cloud is not feasible, as point clouds can contain thousands of points, meaning the time saved by the DWA is far exceeded by the inference time of the agent.

Instead, the whole point cloud could be considered all at once, and the action would be a binary vector $\vec{a} = (a_0, a_1, \dots, a_i, \dots, a_p)$ of length p where a_i can be 0 to ignore the point, or 1 to keep it, and p is the size of the point cloud. This also allows the agent to use knowledge about other points when deciding if a point is worth keeping. However, the huge size of the point clouds introduce another problem: the action space becomes extremely large, as the number of actions grows exponentially with the number of points. This makes exploration extremely hard, and makes it even harder to correctly assign which of the chosen sensors were critical and which were not useful. Early experiments, presented in Chapter 5, showed that using this action space resulted in little to no learning.

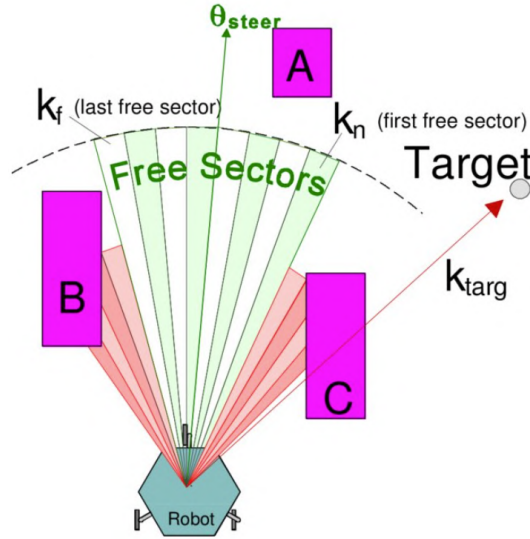


Figure 4.6 The Vector Field Histogram method [55]

To solve this problem, the action space needs to be reduced, while still letting the agent to act on the whole point cloud. The solution proposed draws inspiration from the Vector field histogram (VFH) navigation method, presented in Section 3.1.1. As shown in Figure 4.6, the VFH method works by dividing the robot's sensor data into a set of histograms, each representing a different angle range around the robot. The histograms are then used

to generate a polar occupancy grid map that describes the presence of obstacles in each direction. We can use this to separate the point cloud into regions and let the agent choose the regions instead, which drastically reduces the action space, although it also reduces the agent’s precision in its choices. Regarding this, one advantage of this method is that the number of sections can be varied to find a compromise between the size of the action space (or learning efficiency), and the granularity of the agent’s choices over the point cloud (policy performance). Multiple experiments are done with different number of regions to evaluate the impact of the number of regions and pick the best value, as shown in Appendix C. The final action space will then be a vector $\vec{a} = (a_0, a_1, \dots, a_k)$ where k is the number of regions considered.

Observation space

For the observation space, the agent will have access to the same information the drone has from its sensors. The drone’s velocity, the point cloud measured by de LiDAR, and the relative position of the goal are concatenated to create a one-dimensional vector for the observation, leading to a first representation:

Observation Space 1: Raw Data representation The first approach is to feed the raw data directly. More specifically, the distance and orientation to the goal $[d_{goal}, \theta_{goal}]$, the velocities of the UAV $[v_x, v_y]$ and the points of the point cloud with size p , which is an array of polar coordinates $[\mathbf{d}_i, \theta_i]$ of length p .

Similarly to the action space, the observation space can be reduced by using the VFH method to separate the point cloud into sectors. Another advantage of doing this is that Vector field Histograms are more robust to noise: Even if the LiDAR can get bad readings for a few points, the probability that every point in a given sector is erroneous is much lower. This makes the observation more robust and the learning easier, again at the cost of giving less low-level information to the agent. Experiments will be done to find the impact of varying the number of regions in the observation space on the performance of the agent. This leads to a second representation:

Observation Space 2: VFH representation It is assumed that transforming the point cloud into a Vector Field Histogram with k sectors will reduce the dimensionality of the problem while conserving a maximum of information. Again, the distance and orientation to the goal $[d_{goal}, \theta_{goal}]$, the velocities of the UAV $[v_x, v_y]$ are used, and the points of the point cloud are transformed into k sectors with a similar representation: an array of polar coordinates $[d_{sector}, \theta_{sector}]$ of length k where d_{sector} is the distance to the closest obstacle in that sector.

Neural networks benefit from easier gradient descent when their inputs are between $[-1, 1]$. Following this, the normalization of the observation is common practice in DRL. However, normalization has to be done carefully to not lose information, and different normalization methods need to be applied to different inputs. First, the velocities in the drone, which are in the interval $[-v_{max}, v_{max}]$ for both x and y coordinates, can simply be set between $[-1, 1]$ by dividing by the maximal velocity: $[\frac{v_x}{v_{max}}, \frac{v_y}{v_{max}}]$. Second, the goal direction, which is a distance and orientation $[d_{goal}, \theta_{goal}]$ can be normalized with the following transformation:

$$[d_{goal}, \theta_{goal}] \leftarrow \left[\frac{\log(d_{goal} + \epsilon)}{\log(d_{range})}, \frac{\theta_{goal}}{180} \right] \quad (4.2)$$

Where $\epsilon > 0$ is a small value to make sure the log will not diverge, and d_{range} is the maximum distance that can be measured by the YDLIDAR X2. The values are then clipped to $[-1, 1]$. Finally, the same technique is applied to the sectors computed by the VFH method. Note that the log scale helps give more importance to small values, giving the agent more precise information for closer obstacles:

$$d_{sector} \leftarrow \left\lfloor \frac{\log(d_{sector} + \epsilon)}{\log(d_{range})} \right\rfloor \quad \forall \text{ sectors} \in \text{histogram} \quad (4.3)$$

4.2.2 Reward Design

Reward design is a critical step in Reinforcement Learning, as it is one of the first factors in whether an algorithm can successfully learn a task. There are two perspectives when designing reward functions. On the one hand, RL algorithms should be general and learn with minimal knowledge, leading to minimal reward design to avoid introducing too much bias in the rewards. On the other hand, sparse rewards make for a very hard learning task, and will often require much more data before learning an effective strategy. In practice, some amount of reward shaping is common to increase the learning efficiency, especially when trying to apply RL to solve a specific problem. Multiple reward functions have been designed and evaluated during this thesis, and the ones explored in more detail will be presented in this section.

Sparse rewards

First, a few sparse rewards were designed as the basis of the reward function. They aim to reward high-level behaviors (e.g. moving from point A to point B safely), so they give a less specific learning signal to the agents, but are easier to define so that they encourage the

wanted behavior without rewarding unexpected strategies. For this, terminal rewards, given to the agent at the last step T of the episode, are defined as follows:

$$r_T = \begin{cases} 1 & \text{if the agent reaches the goal} \\ 0 & \text{if the maximum amount of time steps is reached} \\ -10 & \text{if the UAV collides with an obstacle} \end{cases} \quad (4.4)$$

Reward function 1

This reward function is based on two key ideas: first, if the agent is not able to reach a target success rate SR_{target} , it probably means it is not choosing sectors that are important to avoid collisions. Thus, we want to encourage the use of more sensors. Second, if the agent does reach the target success rate and exceeds it, it means all the important sectors are chosen by the agent, and we can apply a penalty to discourage using more sensors, and pressure to not choose useless sectors.

$$r_{t,1} = r_t(sectors) * r_t(SR) \quad (4.5)$$

Equation 4.5 models this behavior with two terms. The first term, $r_t(sectors)$, is the difference between the total number of sectors k_{max} , and the number of sectors chosen by the agent k_{chosen} . It is maximal when no sectors are used and 0 when all of them are used. When the success ratio of the agent SR , computed over a window of the last 100 episodes, is over a chosen threshold SR_{target} , the term $r_t(SR)$ is positive. This means the agent gets more reward when choosing fewer sectors, and is encouraged to choose the lowest possible amount of sectors needed to avoid obstacles. However, when the success ratio is under the wanted threshold, $r_t(SR)$ becomes negative, and the agent now gets more rewards by using more sectors. This is especially useful to help the agent find the critical sectors, as it might be hard to discover them by luck otherwise: most of the time, the majority of the surroundings of the drone are less important for obstacle avoidance (e.g. far obstacles, objects behind the drone) and the relevant obstacle might be small enough so that only it occupies only a few sectors.

$$r_t(sectors) = k_{max} - k_{chosen} \quad (4.6)$$

$$r_r(SR) = SR_w - SR_{target} \quad (4.7)$$

As mentioned earlier, the success ratio SR is the ratio of successful runs over the unsuccessful ones (collisions or running out of time), for a window of the last 100 episodes: $SR_w = \frac{\sum Success}{\sum Fails}$

Reward function 2

A second reward function is designed with the idea that, in general, closer obstacles represent a higher risk for collisions. Moreover, the difference between an obstacle at 0.5 m and one at 1 m is more critical than the difference between 5.5 m and 6 m. To model this, reward function 2 is proposed in Equation 4.11:

$$r_{t,2} = \frac{1}{k_{max}} \sum_{i=1}^{k_{max}} \begin{cases} r_{t,dist}(d_i) & \text{if } i \in k_{chosen} \\ 0 & \text{otherwise} \end{cases} - r_{cost} * \|k_{chosen}\| \quad (4.8)$$

where k_{chosen} is the set of sectors selected by the agent, r_{cost} is a parameter for the cost of each sector that can be tuned, d_i is the distance to the closest obstacle in sector i , and $r_{t,dist}(d_i)$ is defined as:

$$r_{t,dist} = \begin{cases} 10 & \text{if } d_i < d_{safety} \\ \frac{3}{(d_i - d_{safety})} & \text{otherwise} \end{cases} \quad (4.9)$$

This creates a zone of radius d_{safety} meters around the drone which gives high reward, with diminishing value as the distance to the drone increases. The d_{safety} parameter can be tuned to increase the safety zone to any wanted value. For $d_{safety} = 0.3$ m, Figure 4.7 shows how closer obstacles give more reward, and how the differences for small distances are more significant than those at long distances. The r_{cost} parameter can also be tuned to create a baseline for each sector. With a very small cost, almost all sectors will give positive rewards. For a very high cost, only the sectors with extremely small d_i will have positive rewards, so the agent will be more picky.

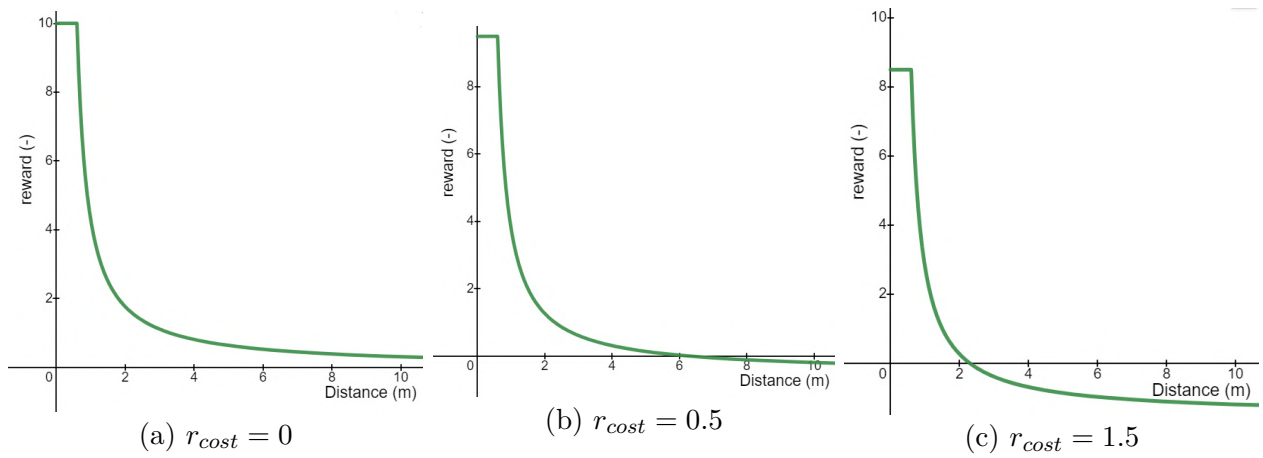


Figure 4.7 Reward Function 2 for different r_{cost} values

Reward function 3

A third reward function was investigated in this thesis. It aims to encourage choosing sectors that are generally towards the direction of the velocity. The basic idea is that, when we move, we look forward up to large distances while only keeping track of our close surroundings in the other directions. Therefore, reward function 3 is defined by:

$$r_{t,3} = \frac{\|v\|}{k_{max}} \sum_{i=1}^{k_{max}} \begin{cases} r_{t,angle}(\theta_i) & \text{if } i \in k_{chosen} \\ 0 & \text{otherwise} \end{cases} - r_{cost} * \|k_{chosen}\| \quad (4.10)$$

Where $\|v\|$ is the norm of the velocity vector of the drone, θ_i is the angle between the velocity and the sector i , and $r_{t,angle}(\theta_i)$ is defined as:

$$r_{t,angle}(\theta_i) = r_{orientation} * \cos(\theta_i) \quad (4.11)$$

$r_{orientation}$ is a parameter that can be tuned to give more or less importance to the orientation of the sectors. In summary, sectors with objects that are in front ($\theta_i < \pi/3$) will be rewarded, sectors in the rear (meaning the UAV is already going away from them) will be penalized, and sectors on the sides will have a somewhat positive reward for $\theta_i < \pi/2$ and somewhat negative reward for $\theta_i > \pi/2$. The idea behind multiplying the term with the norm of the velocity is that it becomes increasingly important to look forward the faster we go, so the rewards are scaled with the speed of the UAV. Similarly to the reward function 2, the parameter r_{cost} can be tuned to introduce a cost for using a sector, creating a baseline and incentivizing the drone to only pick the sectors that give high rewards.

4.2.3 Agent Architecture

The design of neural networks is a research field in and of itself, with a vast amount of network architectures proposed in the literature depending on the task (fully-connected, convolutional, recurrent, etc.) For this thesis, a primary objective of the neural network, apart from approximating complex functions for the policy and value functions, is to have as short of an inference time as possible. In fact, since the goal of applying reinforcement learning to this problem is to make the navigation pipeline as efficient as possible, the inference time will directly influence the amount of time that can be saved processing the modified DWA. If the inference time is similar to the processing time of the obstacle avoidance method, accelerating it will not lead to more time available for other tasks, because it will be used performing inference. Because of this, the policies explored will be shallow (fewer layers) and

multiple architectures will be explored to find a good compromise between performance and inference time.

Furthermore, since PPO and A2C are actor-critic learning algorithms, there are different neural networks that can be designed for the actor and the critic, and as such, different combinations of architectures will be investigated. In the literature, a shared architecture is often used for the two networks [128] [129], but Mysore et al. showed in 2021 that the actor’s neural network can be reduced by up to 99% (in the number of learnable parameters) while achieving comparable policy performance [130]. This is particularly relevant to this work as the size of the network directly affects the inference time. The design choices are on the number of layers, the different layer sizes, and the types of layers (fully connected, recurrent, etc).

Fully Connected Neural Networks (FCNs)

The first architecture investigated in this thesis is the fully connected base architectures. To reduce the search space, the overall methodology is to start from a network complex enough to learn the task, then decrease the actor’s size to reduce the inference time, until it can no longer approximate a good policy. The networks are constructed using the stable-baselines custom policies tool-set, which provides an easy way to specify the architecture with a Python dictionary. It also supports using shared or separate layers for the actor and critic in an actor-critic framework. Table 4.3 shows the network architecture of both the actor and critic of the more complex architecture, to have enough hidden layers and neurons to be able to solve the task. The Tanh activation function is used, as recommended with the Stable-baselines implementations of the RL algorithms [121].

Table 4.3 Initial neural network architecture of the RL agent

Layer	Type	Activation	Size
1	Input	-	Observation Space
2	Dense	tanh	128
3	Dense	tanh	64
4	Dense	tanh	64
5	Dense	tanh	64
6	Output	Softmax	Action Space

(a) Actor architecture

Layer	Type	Activation	Size
1	Input	-	Observation Space
2	Dense	tanh	256
3	Dense	tanh	128
4	Dense	tanh	64
5	Dense	tanh	128
6	Dense	tanh	128
7	Output	Softmax	Action Space

(b) Critic architecture

Multiple architectures will be investigated in this thesis to find a minimal policy size, therefore a minimal inference time. Not all of them are predetermined in advance, as the results of

evaluating those policies will be used to guide the exploration (e.g. if reducing hidden neurons causes the performance to drop, we will instead switch to changing the number of layers). All the architectures trained in this thesis are presented in Appendix A.

Recurrent Neural Networks (RNNs)

The second network architecture that is investigated in this thesis is the Recurrent Neural Network (RNN) with Long Short-Term Memory (LSTM) networks [131]. Those networks are particularly well suited for tasks where data is sequential, and since the measurements of the sensors of the UAV are also sequential, they seem well adapted for this task. The idea is to make use of previous information about the environment to make better decisions about which sectors are important for the obstacle avoidance algorithm.

Similarly to FCNs, the methodology is to start with an architecture complex enough to have high performance, and reduce the size of the networks, using the results of the trainings to guide architecture exploration. Stable-baselines provides a tool set to quickly construct LSTM networks with any number of layers, neurons, and LSTM cells. One challenge is that, before exploring LSTM architectures, a good network architecture needs to be found to efficiently encode the observation of the agent. If the features are not encoded before they are fed into the LSTM cells, the inference time might increase, and if they are too compressed, important information might be lost. Fortunately, the results from the exploration with FCNs can be used to have a starting point. Table 4.4 shows the initial architecture of the LSTM networks, and all the architectures that were explored are presented in Annex A.

Table 4.4 Recurrent neural network architecture of the RL agent with LSTMs

Layer	Type	Activation	Size
1	Input	-	Observation space
2	Fully connected	Tanh	128
3	Fully connected	Tanh	64
4	LSTM	Tanh	256
5	Fully connected	Tanh	128
6	Fully connected	Tanh	128
7	Output	Softmax	Action space

(a) Actor architecture

Layer	Type	Activation	Size
1	Input	-	Observation space
2	Fully connected	Tanh	128
3	Fully connected	Tanh	64
4	LSTM	Tanh	256
5	Fully connected	Tanh	128
6	Fully connected	Tanh	128
7	Fully connected	Tanh	128
8	Output	Softmax	Action space

(b) Critic architecture

CHAPTER 5 RESULTS

In this chapter, we present the results obtained in the two introduced arenas with the different training setups and architectures explored, and the experiments performed on a real rover. First, the training results for the architecture explorations will be presented. The agent is evaluated in a quantitative (metrics like success ratio, mission time, etc.) as well as a qualitative (behavior in different situations) way. Then, a selection of the best-performing agents is evaluated to compare them with different baselines, and an agent is implemented on a rover to evaluate its impact on mapless navigation in real scenarios. In each section, the results are presented, then they are discussed along with their limitations and potential for future research.

5.1 Training Results

Table 5.1 gives an overview of the different agent configurations that were explored. Since the number of variations of these configurations is too big to explore exhaustively, a funnel approach is used: starting with more fundamental changes in the configurations (Reward functions, observation space) and refining down to more specific parameters (e.g. policy architecture, learning hyperparameters).

Table 5.1 Global agent configurations summary

Observation space	Raw data representation		VFH representation	
Reward function	Reward Function 1	Reward Function 2	Reward Function 3	
Learning algorithm	PPO		A2C	
Learning algorithm hyperparameters	Table 5.6			
Policy architectures	Table A			

5.1.1 Environment Setup for Evaluation

To create the dense and heterogeneous arenas, multiple parameters were configured using the environment generator described in Chapter 4. They are presented in Table 5.2.

The evaluation consists of running the agent in the arenas for 100 episodes with randomized goal and obstacle positions. All the environment generation is seeded so that the conditions are reproducible and all the methods can be evaluated.

Table 5.2 Configuration of the different parameters of the environment generator for the two introduced arenas.

Parameter	Static Arena	Dynamic arena
Arena size	100 m x 100 m	100 m x 100 m
Number of static obstacles	75	25
Number of dynamic obstacles	0	75
Minimum distance between two obstacles	3 m	n/a
Velocity interval of the dynamic obstacles	n/a	[-1,1] m/s

5.1.2 Evaluation Metrics

For the quantitative evaluation, multiple metrics will be monitored:

- **Success Rate:** The most important metric is the success rate, defined previously as the ratio of successful runs over the unsuccessful ones (collisions or running out of time), for a window of the last 100 episodes: $SR_w = \frac{\sum Success}{\sum Fails}$. A successful run is completed by reaching the goal from a starting position within 1 meter without colliding with the obstacles. Any collision is counted as a fail, and episodes have a limit of 500 timesteps to allow a conservative amount of time to reach the goal. There will be a distinction between the fails caused by collision or running out of time, as a collision is usually worse than taking too much time.
- **Processing time:** Since the goal of this work is to enable faster collision avoidance algorithms with DRL, another main metric will be the processing time of the obstacle avoidance pipeline. This will include all its steps from performing inference, sending the sensor data to the modified DWA, computing a local path, and outputting the corresponding velocity commands.
- **Sector usage:** The number of sectors chosen by the agent will also be monitored to visualize the behavior and compare with both the different learning setups considered, and baselines introduced for this task. It is also a more objective measure of the difference in computational workload of the obstacle avoidance pipeline, as it does not depend on the hardware used.
- **Mission time and trajectory length:** The mission time and trajectory length will be also measured to monitor if the proposed method affects the overall trajectory and navigation speed of the obstacle avoidance method.

Initially, The architecture exploration is based on the success rate metric, as we want to first successfully learn the task. Success rate gives a better idea of whether the task is correctly

learned, compared to presenting the reward over time, which is more abstract. This also helps to compare with other agents that will have different reward functions, as multiple ones were investigated in this thesis.

Table 5.3 Initial agents configuration summary.

Learning Algorithm	A2C
	Default hyperparameters from Stable-baselines
Actor Policy	Multi-layered Perceptron (FCN)
	4 layers of 64 neurons
	<i>tanh</i> activation function
Critic Policy	Multi-layered Perceptron (FCN)
	6 layers of 128 neurons
	<i>tanh</i> activation function

Observation Space

For the first experiment, 6 agents were trained to explore the 2 proposed observation spaces (raw sensor data and VFH representation) and the different reward functions introduced in chapter 4. The trainings were limited to 6 hours, as the goal of the experiment is to identify the configurations that can lead to learning the task, and eliminate those where the agents were not able to learn. The other aspects of the agent’s architectures are kept constant and summarized in Table 5.3. Note that A2C was used because of its faster iteration time, allowing a faster exploration of configurations, and because of early results indicating better performance than PPO, presented in annex E. Also, the policy architectures are chosen to be complex enough to learn a representation, but not too big to keep the training time reasonable, as the optimal architecture will likely be very different. Still, this allows us to use the obtained results to guide future exploration. The results obtained are summarized in Table 5.4, which shows the success rate of the agent obtained with a given combination of a reward function and an observation space.

Table 5.4 Success rate (%) achieved by the agent for different configurations of the observation spaces and reward functions.

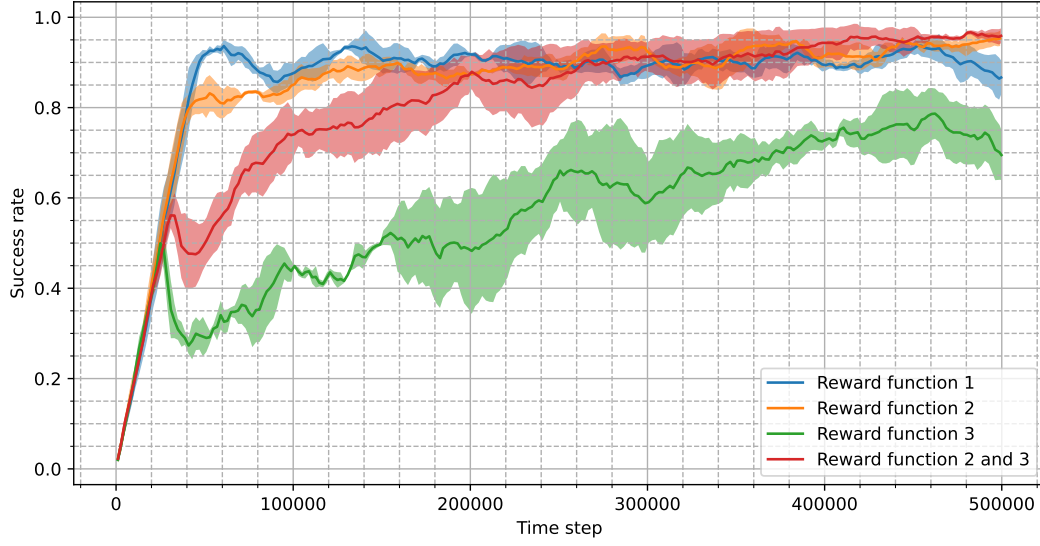
Configurations	Reward Function 1	Reward Function 2	Reward Function 3
Raw data representation	64%	33%	35%
VFH representation	77%	65%	80%

As expected, the VFH representation outperforms the Raw data representation for all reward functions, as the state space is much larger, which makes it harder for the agents to learn which information is relevant. In fact, We can see that the largest differences between the two

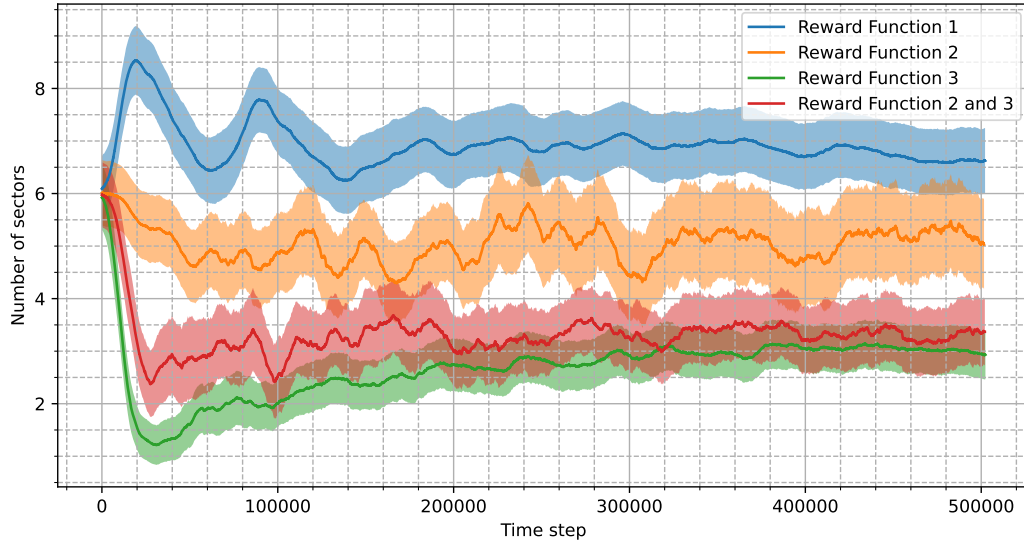
observation spaces happen for reward functions 2 and 3: for reward function 2, 65% success rate for the VFH representation versus 33% for the Raw data representation, and 80% success rate versus 35% respectively for reward function 3. This can be explained by the fact that those reward functions are directly based on information about the sectors that the agent can choose, so a lot of information in the raw data needs to be processed into features before becoming useful for the action space. In contrast, the VFH representation already exposes the relevant information in each sector to the agent. Note that since all the information is available in the Raw data representation, it would be expected that this observation space eventually reaches the performance of the VFH representation, but the amount of compute required could be prohibitively large. For the remaining experiments, the VFH observation space was selected as it was performing much better than the raw data representation for all the reward functions. As for the reward functions, the first one proposed seemed the easiest to learn, as it achieved the best success rate with the first observation space and 77% with the second. However, reward function 3 seemed to achieve the best success rate with 80%. A more thorough study of the reward functions is done next.

Reward Function

To better identify which reward function performed better, a longer training was done with four agents: 3 for the 3 proposed reward functions, and one combining reward functions 2 and 3, as they represent different ways to identify dangerous obstacles. For this experiment, both success rate and sector usage will be monitored, to get a better understanding of the performance of the agents. In fact, the success rate alone is not sufficient to tell if the agent developed a good strategy for this task, and this is because it does not only involve navigating safely, but also doing it more efficiently. For example, using all the sectors (which is equivalent to the classical solution) will result in very few collisions and a high success rate while no actual strategy is learned. To solve the task, the agents need to actually reduce the computational load of the DWA without hindering its ability to avoid obstacles. Thus, in the following experiment, the agents were trained to evaluate which reward function leads to high success rate and low sector usage.



(a) Success rate over a 100 episode moving window



(b) Average number of sectors chosen by the agent

Figure 5.1 Training results of the different reward functions. For each reward function, 5 agents are trained and the average success rate and sector usage are monitored. The moving average over 100 consecutive episodes is plotted, and the variance over the 5 agents is represented by the colored area.

Each agent was trained for 500 thousand timesteps on CPU and GPU, and 5 trainings were done for each presented configuration to have an idea of the variance. Ideally, more agents

would need to be trained to better characterize the variance, but since the training time for one agent is about 12 hours, training more agents would mean less time to explore different configurations. The VFH representation is used and the initial configuration divides the surroundings of the UAV into 12 sectors. A study of the impact of the number of sectors in the VFH methods is done in Annex C. The total training time varies depending on the performance of the agent (e.g. crashes cause shorter episodes, which means more episodes for the same amount of training steps, and more simulation work to reset the environment), and the average training time for each agent is presented in Table 5.5.

As shown in Figure 5.1, reward function 1 is the fastest to converge to a high success rate with very low variance, but it is then surpassed by reward function 2 and the combination of both 2 and 3. We can see that it oscillates at a success rate of around 90%. This can be explained by the behavior shown in Figure 5.9b: Since the cost of using a sensor varies as a function of the success rate, this introduces oscillations in the number of sectors used. When the success rate is lower than the target, sensor usage rapidly increases, and inversely when the success rate is high. The agent moves from one state to the other, and the oscillations get smaller and smaller until the agent converges to about 7 sectors over 12. This method illustrates the need to monitor sector usage, as its performance is explained by the fact that it uses much more sectors than the other agents, making it less efficient: the agent is too conservative, and gives more information to the DWA than needed. In contrast, reward function 3 achieves the lowest amount of sector usage, with an average of only 2.4 sectors used for navigation. However, we can also see that the success rate is lower, meaning the DWA is missing some important information for obstacle avoidance and the drone still collides often. The agent is too aggressive in trying to reduce the amount of data processed. Reward function 2 achieves a high success rate (97%) while using fewer sectors than reward function 1, with an average of about 5. This is expected because the distances to the obstacles give a more direct sense of danger than their angle relative to the drone movement, which makes the agent more robust. Also, contrary to reward function 2, this method is directly affected by the difficulty of the environment (denser obstacles will require more sectors to be used), and the agent adapted its behavior to the hard training environments by using more sectors. Finally, the combination of both reward functions 2 and 3 achieves the best success rate (98%) and a significantly lower sector usage with an average of 3.2 over 12.

Considering those results, it can be assumed that the combination of both allows the agent to be more efficient while staying robust. We can see that the success rate initially increases rapidly like for the other methods, and that the sector usage decreases rapidly similar to reward function 3. The performance then drops at around 50 000 time steps, meaning the agent is starting to discard important information. However, the rewards related to the

proximity of the obstacles become more important now that the agent collides more often and sector usage increases slightly. This is then followed by an increase in the success rate until convergence and a slight decrease in sensor usage.

Table 5.5 Average training time of the agents

Agent	Reward function	Average training time
1	1	10h 53m
2	2	11h 17m
3	3	13h 45m
4	2 and 3	12h 01m

Hyperparameter Tuning

Once configurations that can learn the task are identified, the next objective was to explore the hyperparameters of the learning algorithm. Tuning the hyperparameters can help improve performance and greatly accelerate training. This is helpful as shorter training time can enable us to explore more configurations. For the rest of the experiments, the combination of reward functions 2 and 3 was used. Multiple agents were trained to explore different values for the learning rate, discount factor, maximum gradient norm for the policy update, etc. The ranges to explore were based on the suggested hyperparameter ranges from the Stable-Baselines documentation. The hyperparameter ranges explored are listed in Table 5.6, and the results are presented in Table 5.7. As explained in Chapter 2, the learning rate represents the size of the update of the value function, the discount factor represents the weight that is given to future rewards (a value of 1 will give as much value for future rewards as for immediate rewards, and a value of 0 will completely ignore future rewards), the Max_gradient_norm will limit the updates of the actor and critic neural networks, and the activation function will affect the output of each neuron of the neural network.

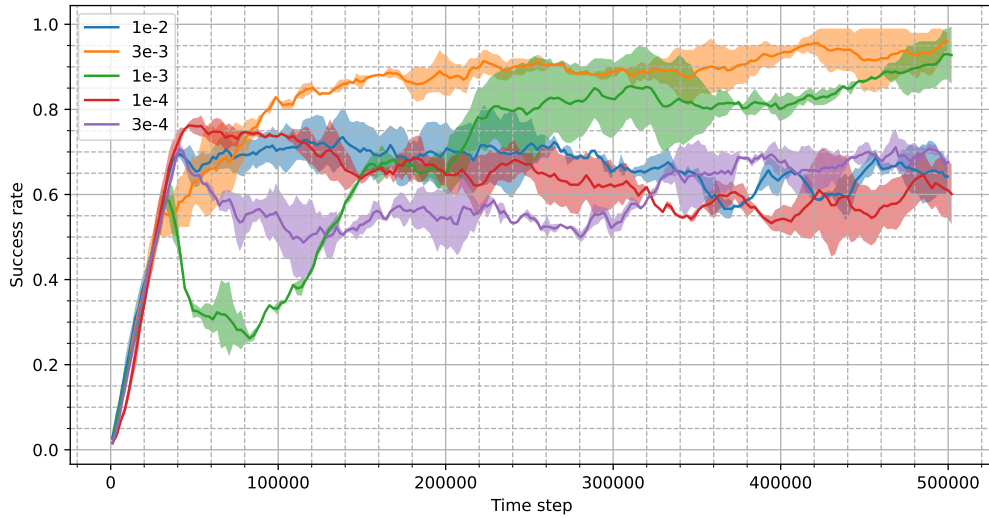
Table 5.6 Domain of the hyperparameters explored for A2C

Hyperparameters	Range of values to test
Learning Rate	[1e-4, 1e-2]
Discount Factor	[0.9, 0.999]
Max_gradient_norm	[0.3, 5]
Activation Function	tanh, Relu

Because of the large number of possible combinations, the hyperparameters were explored sequentially (changing only 1 at a time). This makes it easier to evaluate the impact of the individual hyperparameters. However, it is important to note that the optimal configuration might have been missed, as some of the values are close, and one hyperparameter can have

an impact on the optimal values of the other hyperparameters. However, since the different hyperparameters are highly dependent with each other, more time would be needed to explore configurations and find the optimal one.

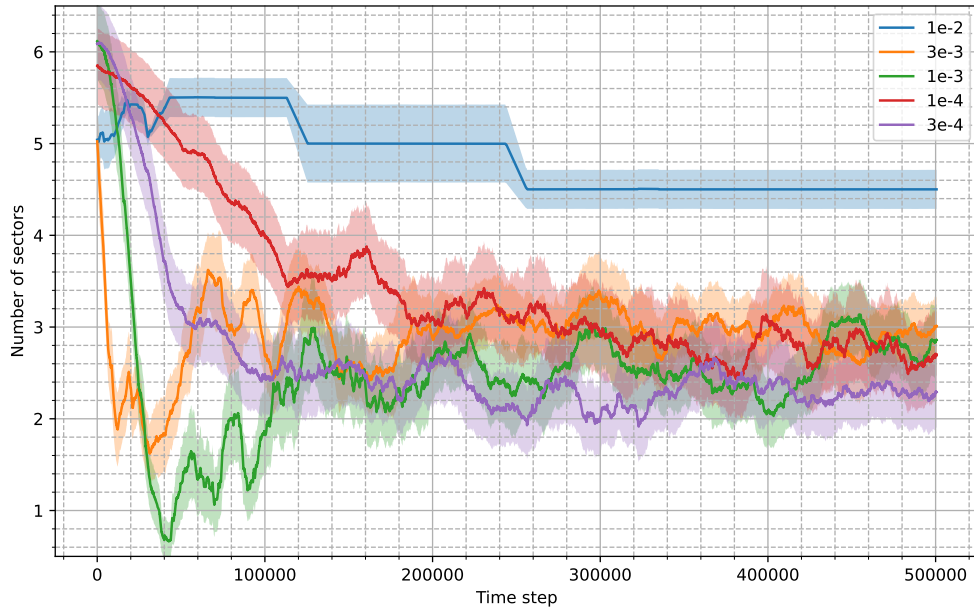
Using values at the extremes of the explored ranges for the hyperparameters often caused the algorithm to fail to learn or drastically reduced its stability. For example, a value of 5 for the `Max_gradient_norm` in the policy updates caused oscillations, as the updates were too large and caused drastic changes in the behavior. We can see a slight decrease in the performance for high values of the discount factor with a success rate of 90% with $\gamma = 0.999$ compared to 98% for $\gamma = 0.9$. One hypothesis for this is that since the agent has access to very limited information about the state of the environment, it is almost impossible to predict the future states up to a few time steps, so less importance can be given to future rewards (smaller γ). The hyperparameter that had the most impact was the learning rate, as shown in Figure 5.2.



(a) Success rate over a 100-episode moving window

Table 5.7 Hyperparameter study for A2C. When the learning rate is varied, the others hyperparameters remain fixed at the default value from the stable baselines library. Then, the best value is chosen and the discount factor is varied in a similar way. This is repeated for the remaining hyperparameters.

Hyperparameter	Value	Success Rate		Sector usage	
		Mean	Standard Deviation	Mean	Standard Deviation
Learning Rate	1e-2	65%	4.9%	4.5	0.36
	3e-3	96%	4.7%	3.02	0.43
	1e-3	93%	6.6%	2.83	0.61
	3e-4	67%	2.3%	2.27	0.51
	1e-4	60%	6.1%	2.70	0.48
Discount Factor	0.999	90%	2.1%	3.91	0.45
	0.99	97%	3.2%	3.65	0.59
	0.9	98%	1.3%	3.77	0.65
Max_gradient_norm	5	77%	3.5%	3.45	0.33
	1.5	85%	2.9%	3.94	0.42
	0.3	97%	2.3%	3.82	0.35
Activation Function	tanh	96%	2.4%	3.74	0.36
	Relu	94%	3.3%	3.40	0.40



(b) Average number of sectors chosen by the agent

Figure 5.2 Training results with different learning rates. For each value of the hyperparameter, 2 agents are trained and the average success rate and sector usage are monitored. The moving average over 100 consecutive episodes is plotted, and the variance over the agents is represented by the colored area. The variance for the number of sectors is computed over a window of 4000 timesteps

We can see that the smallest learning rate ($1e-4$) fails to learn the task in less than 500 000 time steps. It is expected to still learn, but it would take much more time. Note that all the different learning rates seem to learn at the same speed at the beginning of the training, but this happens due to the fact that random actions actually achieve a performance of around 60%. In fact, choosing each sector with a probability of 0.5 means the average number of sectors used is pretty high, and most of the obstacles will be seen by chance. The second smallest learning rate ($3e-4$) has an initial drop in performance, then gradually gets a better success rate than $1e-4$. This can be explained by the fact that the higher learning rate causes a more rapid reduction in sector usage at the beginning of the training, before the agent discovers which sectors are important and which are not. This effect is even more present with the learning rate of $3e-3$, which drops to a success rate of 26%, but achieves close to the highest performance. The best results were obtained with a learning rate of $3e-3$. In this case, the learning rate is high enough so that the agent starts to learn which sectors are important as fast as it learns to reduce its sector usage, explaining why the initial drop in performance is smaller. This is also shown in C.1b: the number of sectors used rapidly drops, but is corrected faster with a learning rate of $3e-3$ (compared to $1e-3$). As for the learning rate of $1e-2$, we can see that the very high learning rate causes instability and the agent fails to learn. The number of sectors stays the same for a long time then changes drastically. Because of this, the lower learning rate of $3e-3$ seems to be more appropriate for the task.

Neural Network Architectures

The next experiment explored multiple network architectures for both the actor (policy) and the critic (value function). The primary objective is to find the smallest architecture that can learn the task, so that it performs inference as fast as possible. For this, More metrics are considered: first, the success rate is still the primary metric, as we want to maintain navigation performance, and needs to be maximized. Second, the inference time is now considered, as the size of the neural networks directly impacts it (i.e. smaller policies with fewer parameters require less inference time). The lowest possible inference time while keeping a high success rate is desirable. Third, Sector usage gives an idea of how efficient is the learned strategy and needs to be minimized. The training results are presented in Table 5.8, and all the architectures of the policies explored in this thesis are listed in more detail in Annex A. In the Network architectures, the tuples represent the number of neurons in each layer (e.g. a network of 2 layers of 64 neurons each is represented by $[64, 64]$). When a layer consists of LSTM cells, it is specified after the number of neurons (e.g. $[64, 64 \text{ (LSTM)}]$).

All the policies achieve more than 90% of success rate, indicating that A2C can pretty reliably

Table 5.8 Policy architecture exploration results.

Configuration	Network Architectures	Success Ratio	Inference time	Sector usage
1	Actor [128, 64, 64, 64] Critic [256, 128, 64, 128, 128]	97 %	0.24 ms	2.92
2	Actor [64, 32, 32, 32] Critic [128, 64, 64, 64]	96 %	0.15 ms	2.67
3	Actor [128, 128] Critic [128, 128, 128]	92 %	0.19 ms	3.04
4	Actor [64, 64] Critic [128, 64, 128]	94 %	0.16 ms	2.74
5	Actor [32, 32] Critic [64, 64, 64]	96 %	0.13 ms	2.94
6	Actor [64] Critic [64, 64, 64]	95%	0.12 ms	2.86
7	Actor [128, 64, 256 (LSTM), 128, 128] Critic [128, 64, 256 (LSTM), 128, 128, 128]	97 %	0.41 ms	3.15
8	Actor [128, 64 (LSTM), 64, 64] Critic [128, 64 (LSTM), 128, 128]	95 %	0.26 ms	3.14
9	Actor [64, 32 (LSTM), 32] Critic [64, 32 (LSTM), 32, 32]	96 %	0.18	3.28

solve the task even with a very simple policy. For example, the smallest policy, policy 6, still achieves a success rate of 96% and uses an average of 2.86 sectors per time step, showing that it did not learn to just give all the sectors to the DWA. By looking at the inference time of the different architectures explored, we can see that the size of the network affects the inference time, as expected. The biggest policy, policy 7, has the highest inference time with 0.41 ms of CPU time. Note that the time was measured on a separate test focusing on the inference, and measuring CPU time instead of real-time, as real-time could be affected by factors like hardware heat and other processes running in parallel. This means the measured values can still change depending on the hardware used for the inference, but it is a more consistent way to compare them with each other. The inference time is faster with smaller actors, but reducing the policy size has diminishing returns, with a 15% difference between policy 5 and policy 6. The fastest policy has an inference time of 0.11 ms. Another way to consider which policy architecture to choose is to look at the sector usage of that agent. This effectively tells us how much it is able to reduce the computational load of the DWA, which means the best agent in Table 5.8 is not necessarily the one with the fastest inference time. Policy number 5 seems to make a good compromise between a high success rate (reaching the goal safely), having a fast inference time, and having one of the lowest averages of sectors used, but all the policies seem to have the capacity to approximate a correct strategy.

As for the two architectures considered (fully connected and LSTM networks) both achieve similar performance, but the fully connected are slightly faster for a similar number of parameters. This can be explained by the fact that the LSTM cells need more processing than the basic neurons of the FCNs, and is shown with the inference time of policy 7, which has

the most LSTM cells.

Finally, the critic architecture seems to have a lesser impact, as the task can be solved even by the smallest critic network, so the bigger critic architectures are not needed. Note however that the number of parameters in the critic does not affect inference time, so only the success rate and average sector usage should be considered for the critic’s architecture. The next thing to evaluate is the impact of the RL agent on the DWA processing time. Further experiments in the qualitative evaluation will also help better understand the learned behavior.

5.1.3 Comparison with other baselines

Acceleration of the navigation pipeline

To measure the impact of the proposed method on the efficiency of the modified DWA, the test setup presented in 5.1.1 (dense arena and heterogeneous arena) is used and the CPU processing time is monitored. Two final agents will be trained and evaluated with the configurations that seemed to give better results in the experiments above. The first agent we propose, named Safety Bug, aims to achieve the highest success rate, while the second agent, named Focus Bug, aims at a compromise between safety and processing speed, and has a higher cost for using sectors. The final configurations for the trained agents are summarized in Table 5.9. The agents were trained for 1 million time steps using curriculum learning: the cost of the sectors in the reward function is gradually increased as the agent learns its strategy. The major difference between the agents is the final cost of the sectors in the reward function. A study was also performed to explore the impact of different costs on the final performance and sector usage in Annex B. The training time of the first agent was 20 hours and 32 minutes, and the second agent was 21 hours and 07 minutes.

Table 5.9 Final RL model and agent configuration

Item	Agent 1 (Safety Bug)	Agent 2 (Focus Bug)
Observation Space	VFH representation with 12 sectors	
Reward function	Combination of reward functions 2 and 3 (proximity of obstacles and direction of velocity) final sector cost = 1 final sector cost = 1.3	
Learning Algorithm	A2C	
Hyperparameters	Learning rate: 3e-3 Discount factor: 0.9 Max gradient norm: 0.5 Activation function: tanh	
Agent architecture (Fully connected)	Actor: [32, 32] Critic: [64, 64, 64]	Actor: [64] Critic: [64, 64, 64]

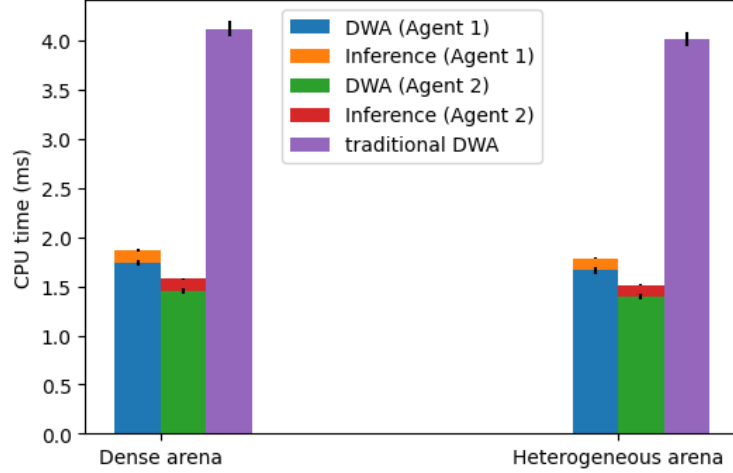


Figure 5.3 Comparison of the processing time of the DWA

Figure 5.3 presents the total CPU processing time for the obstacle avoidance pipeline. The time interval measured goes from receiving the sensory inputs to outputting the wanted velocity commands to the UAV flight controller. The orange and red bars are the inference time of the agents, which needs to be taken into account for a fair comparison with the baseline. The navigation pipelines were tested for 100 episodes, in a fixed environment with predetermined goal positions, so that they encounter the same conditions.

Our method achieves a 2.72x acceleration (with Focus Bug) of the DWA pipeline by using on average 13% of the sectors, meaning most of the information that is processed in the classical obstacle avoidance method is not critical to avoid collisions. Safety Bug achieves a 2.3x acceleration using on average 16% of the sectors. This is not surprising since state-of-the-art methods often make conservative assumptions to simplify the navigation problem while making sure the robot can navigate safely, and all the geometric information gathered by the LiDAR is processed, regardless of its position in the drone’s surroundings. The biggest advantage of the agents is to be able to keep the same robustness in obstacle avoidance while removing those conservative assumptions and making navigation more efficient. Even though the amount of sectors is reduced by a factor of up to 8, we do not observe an 8x speedup since other operations take the same amount of time both in the baseline and our approach, and the inference of the network takes a small, although non-negligible time.

More time to explore configurations and train longer could lead the agents to become even more efficient, but up to a certain point only. With the chosen policy, inference time is still much lower than the obstacle avoidance pipeline, but if the agent gets more complex and makes the DWA even more efficient, we might reach a point where inference time is

in the same order as obstacle-avoidance processing time. Further work would be needed to explore this boundary and try to learn where it makes more sense to spend more resources on selecting relevant sensory input or simply spend those resources to process more information. As shown in Annex B, the sector usage can still be reduced by increasing the sector costs in the reward function although this will make the agent less robust and increase the chances of a collision. Further work would be needed to optimize this value and find a good compromise between efficiency (Focus Bug) and robustness (Safety Bug). This compromise is illustrated in the following results.

Robustness of the agent

The agents' impact on the robustness of the obstacle-avoidance pipeline was measured and compared with the traditional approach and two baselines. Those were introduced to give a better idea of the quality of the strategy learned by the agents. The baselines are defined as:

- **Random actions:** This baseline will simply choose a random set of sectors at each time step. It is often used in RL to verify if the policy has learned something or is just acting randomly.
- **Greedy policy:** This baseline will process the LiDAR data to find the n sectors with the closest obstacles. n will be chosen to be equal to the average number of sectors used by the agent. This allows to know if the learned strategy is better than always taking the n more dangerous sectors.

Note that the introduced baselines do not aim to reproduce the state-of-the-art but are rather used to know if using RL to solve this problem is justified. If *Greedy policy* performs better than the RL agents, we should simply replace the agent with this strategy to make the DWA approach more efficient. To compare with the state-of-the-art, we need to look at the differences between the agents and the traditional approach.

To compare all the methods, the same testing setup is reproduced and the metrics that were defined in Subsection 5.1.2 (Success rate, sector usage, mission time, and trajectory length) are used. The results of the evaluation and the comparison of the RL agent with the other baselines are presented in Table 5.10.

Table 5.10 Comparison of the agent with the traditional obstacle-avoidance method and the 2 introduced baselines. *The traditional method does not have an average sector usage since it directly takes the LiDAR pointcloud as an input to compute the local path.

Agent/Baseline	Avg. sector usage	Success rate	Collision frequency	Total mission time (s)	Traveled distance (m)
Safety Bug (ours)	1.85	98%	0%	1275.2	1741.2
Focus Bug (ours)	1.48	93%	7%	1243.5	1726.1
Traditional DWA	N/A*	91%	0%	1497.3	1634.6
Random actions	6	72%	16%	1162.7	1696.3
Greedy policy	2	92%	6%	1349.5	1782.4

(a) Heterogeneous environment

Agent/Baseline	Avg. sector usage	Success rate	Collision frequency	Total mission time (s)	Traveled distance (m)
Safety Bug (ours)	2.1	99%	1%	1304.8	1713.4
Focus Bug (ours)	1.64	91%	9%	1318.9	1700.1
Traditional DWA	N/A*	99%	0%	1646.2	1742.1
Random actions	6	65%	33%	1289.7	1814.0
Greedy policy	2	93%	7%	1455.7	1832.1

(b) Dense environment

The agents seem to perform best in hard, heterogeneous environments, where the conservative assumptions of the classical methods fail to adapt to the changing conditions. In the heterogeneous environment, Safety Bug (SB) selected just under 2 sectors per time step (1.85), so the *Greedy policy* is set to use the 2 sectors with the closest measurements at all times. SB achieved the highest success rate (98%) of all the methods. Note that the traditional DWA did not achieve a success rate of 100% even though it is more robust, because there are two possible ways for the drone to fail an episode: a collision, or running out of time. This means the traditional method, being safer to avoid all the collisions, was too conservative and failed to reach the goal in 9% of the episodes. Those failures occurred when there was an obstacle very close to the goal or a passage too narrow for the DWA to find a path through. Note that the method could be tuned to be able to reach the goal in those situations, but it would become less robust and collisions would start to appear. In contrast, the agent was able to identify which obstacles could be safely ignored and was able to reach the goal more reliably, with one episode ending with a collision. Again, the same trade-off exists here: the agent could be trained to be more robust and have no collisions at all, at the cost of being slightly less efficient. For example, we can see that Focus Bug (FB) used even fewer sectors (1.48 on average in the heterogeneous environment), but had only a 93% success rate. This trade-off is illustrated in the study of the cost of sectors in the reward function, in Annex B.

We can also see that the agents outperformed the 2 introduced baselines, confirming that the learned strategy is better than simply selecting the sectors with the closest obstacles or acting

randomly. Interestingly, the smallest mission time is achieved by *Random actions*, even when correcting by only counting the successful episodes. This can be explained by the fact that, on average, *Random actions* had more chance to succeed in shorter episodes with closer goals. Similarly, the traditional method achieved the smallest traveled distance, which makes sense since all the information is available to the local planner to find the shortest path, but this result is biased because the DWA had more chance to get stuck in episodes where the goal was very far. Because of this and the small differences in the results, the mission time and traveled distance metrics do not indicate that the agents had a great impact on them. This can be expected, as the agents only act on the data given to the local planner, and remove information that is not critical to obstacle avoidance, so the paths produced by the modified DWA are similar to the traditional approach.

In the dense environment, we can see that the sector usage of both agents is slightly higher, being constantly near obstacles. Focus Bug collided in 9% of the episodes. Safety Bug still outperforms the baselines, but is on a similar level as the traditional method. This time, the agent achieves a shorter traveled distance of 1713.4 meters, slightly less than the classical DWA (1742.1 m). This can be explained by the fact that the agent can identify which obstacles can be ignored, meaning the drone does not need to modify its path to keep a higher distance from the obstacles. In contrast, the traditional approach aims to keep a fixed safety distance between all the obstacles, resulting in paths with more deviations. This is investigated in more detail in the qualitative evaluation.

In conclusion, the agents are able to maintain the high navigation performance of the state-of-the-art, even in environments with a high density of obstacles, where the conservative, worse-case assumptions of the traditional methods apply. The trained agents can even outperform it by exploiting the heterogeneity of the environment when applicable. The proposed method achieved a maximum of 2.72x speedup of the obstacle-avoidance pipeline by selecting on average 13% of the LiDAR data, considered relevant to avoid obstacles. SB and FB show the trade-off that exists between making the drone more efficient or more robust, and the reward function can be optimized to find a compromise by changing the value of only one parameter.

5.2 Qualitative Evaluation

To fully evaluate the results of the agents, a qualitative study gives more hindsight on how to improve the agents. It is also needed to get an idea of the strategy that was learned. It can help identify situations where the agents underperform and adapt the reward function, action space, or observation space as needed. Three scenarios are introduced to put the

agents in different situations and visualize how it compares to the baseline and other crafted strategies:

1. Blocking obstacle: a situation that happens very often in hard environments is simply when an obstacle is directly in the way of the drone to reach the goal. The drone needs to avoid it in a robust way (with a big enough margin).
2. Passing by an obstacle: another situation that is even more common is the presence of obstacles that do not necessarily obstruct the path to the goal, but require the drone to pass near them. The drone does not need to change its course, but traditional methods will slightly deviate to respect a minimum distance.
3. Complex environments: one of the hardest situations that the drone can encounter is when it needs to avoid multiple obstacles at once and go through a very narrow passage. The drone needs to correctly identify the obstacles that may cause collisions and ignores the obstacles that do not pose a danger.

5.2.1 Blocking Obstacle

For the first scenario, the agent was put in an environment with an obstacle in the way of the goal. The agent needs to give the correct information to the modified DWA to avoid the obstacle, but avoid selecting irrelevant information when the drone is in a safe situation. To Analyze the behavior of the agent, its decision-making (action with the corresponding observation) is visualized at key points in the scenario. To better understand the impact of the agent on the navigation pipeline, the traditional method was also simulated and the trajectories are compared.

Figure 5.4 shows the trajectories obtained by both the agent and the traditional approach (baseline). The starting point of the drone is at point A, and the goal is the green circle at the right of the 20 x 8 meters arena. we can see that both methods easily avoid the obstacle (green rectangle in the center), but their trajectories are slightly different: the agent starts avoiding the obstacle later and gets closer than the baseline, resulting in a 5% shorter distance traveled. This can be explained by the actions taken by the agents at the points B, C, D and E, illustrated in Figure 5.5.

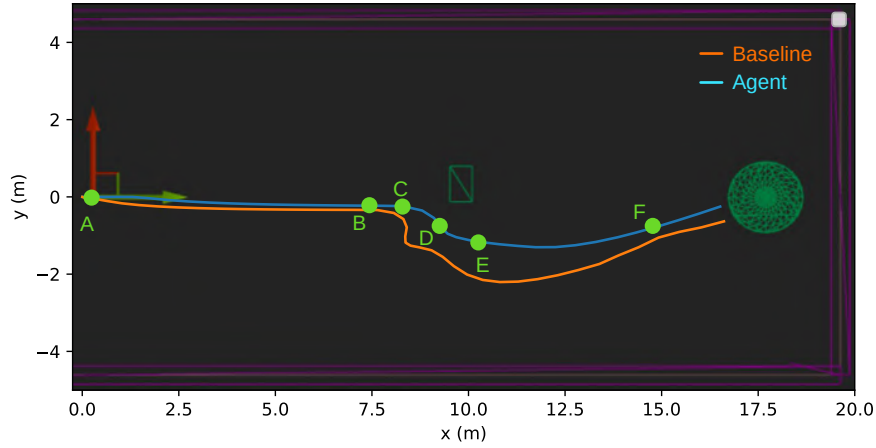


Figure 5.4 Trajectories of the agent and the baseline (traditional approach) in a scenario with an obstacle. The green rectangle is the obstacle, and the green circle is the goal. The drone starts at (0,0) and its trajectory is drawn for both methods. The decisions of the agent at different points are shown in Figure 5.5.

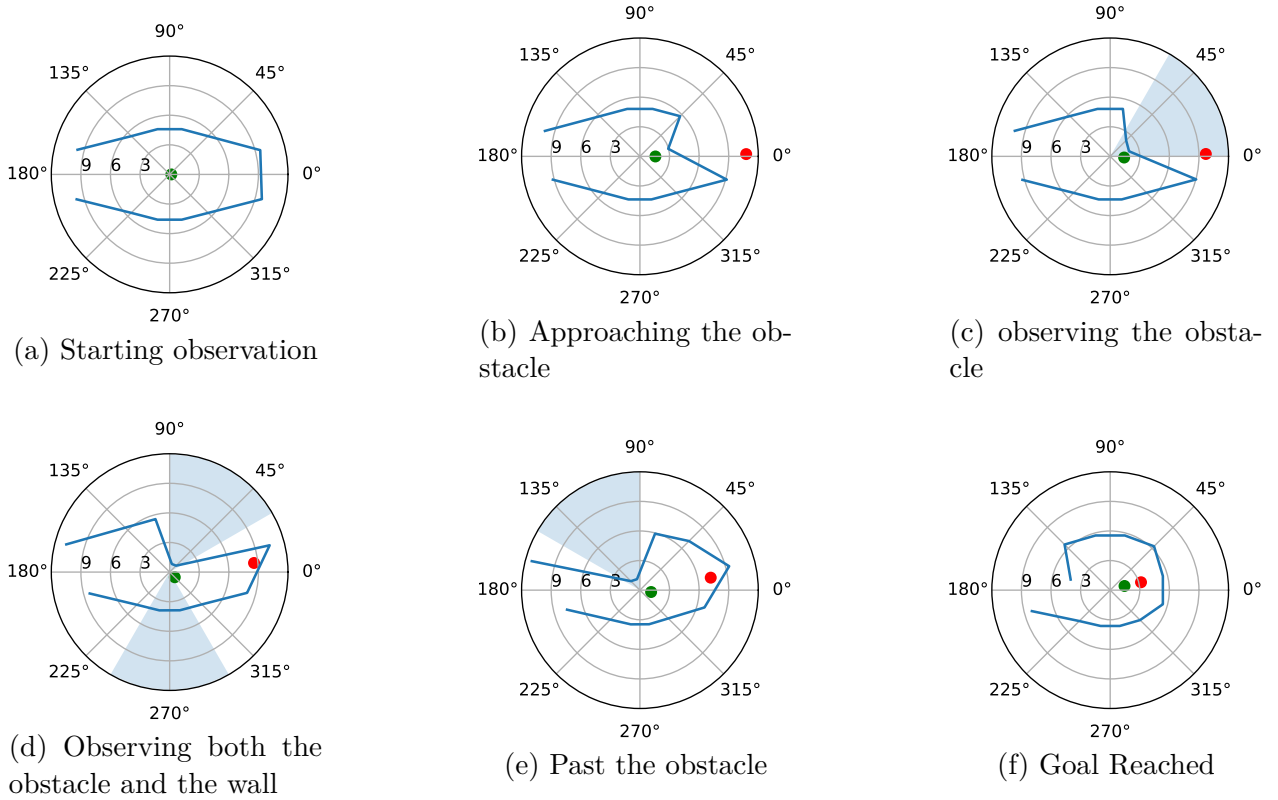


Figure 5.5 Decisions made by the agent at the points corresponding to the ones in Figure 5.5. The observation for one time step is shown in a polar graph, and the blue line represents the distances in all the sectors of the VFH representation, at their corresponding angle (e.g. in (a) the drone sees about 10 meters to the right, and sees walls at 5 meters at 90 and 270 degrees). The sectors chosen by the agent in each time step are highlighted with a light blue color. Finally, the green dot is the predicted position in the next timestep, calculated by the modified DWA, and the red dot is the goal position.

In the starting position A, the drone is in a safe position, with no close obstacles. Consequently, the agent chooses to use no sectors, meaning the navigation pipeline is essentially skipping the obstacle-avoidance process in that time step, making it much faster (about 80% of the processing time is spent on the obstacle-avoidance pipeline). At point B, the UAV approaches the obstacle, but the agent still does not select the corresponding sector, and the modified DWA does not consider the obstacle yet, while the baseline starts a maneuver to avoid the obstacle. Then, at point C, when the obstacle gets closer than 2 meters from the drone, the agent selects the corresponding sector and the drone starts to avoid it, passing closer to the obstacle. When the drone deflects from its course to avoid the obstacle (D), it goes toward the boundary of the arena, and the agent adds the sectors on the bottom to not crash into the wall. 1 meter further (E), the agent seems to judge it is safe and selected no sectors, even though the obstacle is still very close (~ 1 m). The obstacle is not in the way of the goal anymore, resulting in no collisions.

Note that, even though the agent was more efficient than the baseline, the traditional approach could be tuned to achieve the same results. However, this would require modifying the parameters of the DWA, such as the safety distance and weights of the different terms of algorithm 4, which can be time-consuming and requires expert knowledge. Similarly, The agent can be made to take the safer trajectory, depending on what compromise we want to make between efficiency and robustness, by modifying the cost of the sectors in the reward function.

5.2.2 Passing by an obstacle

In this scenario, the UAV will pass near an obstacle that is not blocking the path to the goal. The expected behavior of the traditional approach is to deviate from the path because of its conservative assumptions that remain regardless of the surroundings of the UAV. Again, both the agent and the baseline are put into the same scenario to compare their trajectory and understand the learned behavior of the agent.

In Figure 5.6, we can see that the baseline adjusted its course to maintain a fixed distance between the drone and the obstacle, as expected. Note that the traditional approach found the middle point between the obstacle and the limit of the arena on the bottom, maximizing its distance from any obstacle. In contrast, the agent almost kept a straight course, ignoring the obstacle on its left, resulting in a 4% shorter trajectory. By taking into account the results of scenarios 1 and 2, it can be assumed that the agent learned to take more risks and learned a lower safety margin than the traditional approach. It seems to prioritize the sectors where there are obstacles closer than 2 meters that also are in the way of the drone's velocity.

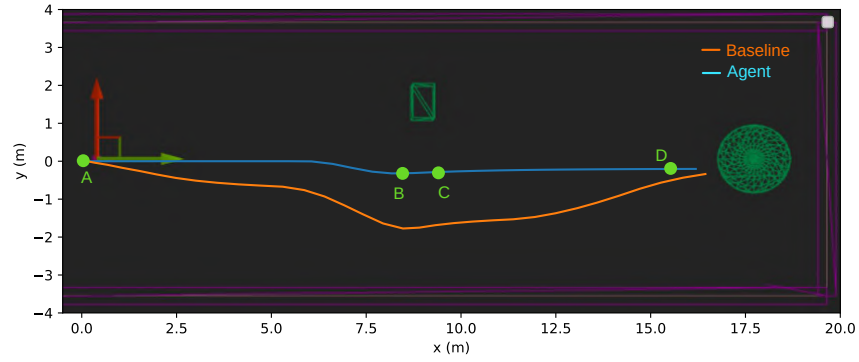


Figure 5.6 Trajectories of the agent and the baseline (traditional approach) in a scenario with an obstacle that is not blocking the way. The green rectangle is the obstacle, and the green circle is the goal. The drone starts at (0,0) and its trajectory is drawn for both methods. The decisions of the agent at different points are shown in Figure 5.5.

This is not surprising since these two aspects are present in reward functions 2 and 3. The behavior is shown in points B and C of Figure 5.7, where the agent ignores the obstacle, since it is not dangerous for the current trajectory, only observing one sector in the direction of the goal.

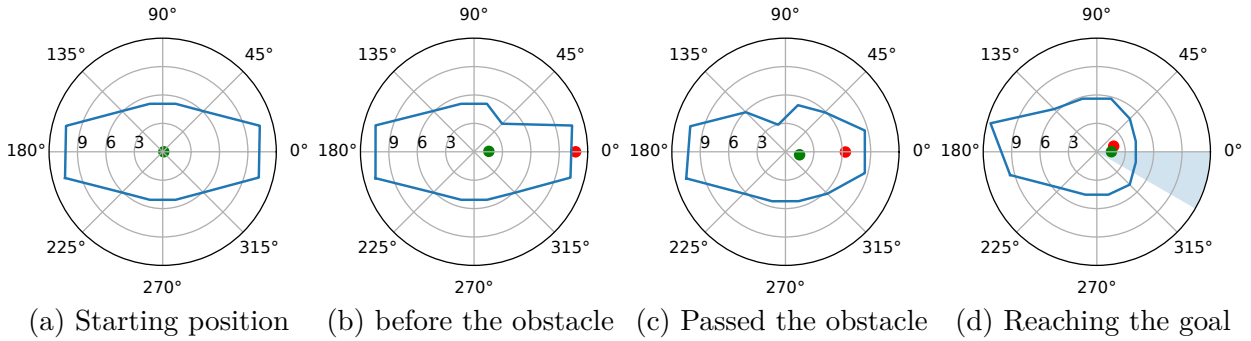


Figure 5.7 Decisions made by the agent at the points corresponding to the ones in Figure 5.6. The observation for one time step is shown in a polar graph, and the blue line represents the distances in all the sectors of the VFH representation, at their corresponding angle (e.g. in (a) the drone sees about 10 meters to the right, and sees walls at 5 meters at 90 and 270 degrees). The sectors chosen by the agent in each time step are highlighted with a light blue color. Finally, the green dot is the predicted position in the next timestep, calculated by the modified DWA, and the red dot is the goal position.

5.2.3 Complex environment

This scenario aimed to put the agent in a much more complex environment, with a high density of obstacles, to test the robustness of the agent and put it in situations where it is harder to determine which information will be important for the drone and which information can be discarded (as opposed to a single obstacle). The drone needs to adapt its path to the multiple obstacles and the global planner (tangent bug) is used to escape local minimums when the DWA fails to progress. Six key points are selected to understand the behavior of the agent. They were selected according to two main criteria: cover different situations to see how the agent reacts (e.g. more vs less obstacles), and cover points where the trajectory of the agent differs significantly from the baseline.

Figure 5.8 shows the trajectory of both the agent and the baseline for the proposed scenario, and the 6 time steps selected to understand how the agent reacts to different situations and explain the differences with the traditional planner. The major difference in the trajectories happens at point E, where the baseline goes back to avoid an obstacle, whereas the agent successfully finds the shorter path, achieving a 16% shorter distance traveled. Point E was chosen to explain the difference between the two methods. At this moment, the navigation pipeline begins a boundary following process, since the drone wasn't able to progress (get closer to the goal) for the last 5 time steps.

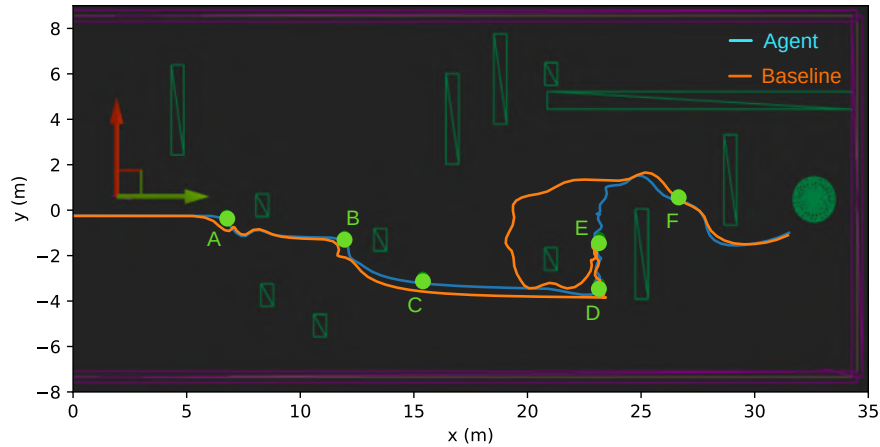


Figure 5.8 Trajectories of the agent and the baseline (traditional approach) in a scenario with multiple obstacles. The green rectangles are the obstacles, and the green circle is the goal. The drone starts at (0,0) and its trajectory is drawn for both methods. The decisions of the agent at different points are shown in Figure 5.9.

As described in Algorithm 3 in Subsection 2.4.1, the drone will follow the boundary of the

closest obstacle until it finds an escape path. However, since the closest obstacle to the UAV is actually the obstacle behind at coordinates $(21, -2)$, the traditional method starts to follow the boundary of that obstacle, going back around the obstacle until it finds an escape. This is a limitation of bug algorithms applied to real-world robots, because the proof of completeness assumes the bug will follow the boundary of blocking obstacles within an infinitesimal distance. In the real world, this is not possible, as the smallest of perturbations will cause collisions, so a safety distance is maintained. Because of this, the bug algorithm will not necessarily follow the correct boundary. In contrast, the agent follows the boundary of the other obstacle, leading to a shorter path. We can see the reason why the agent did not suffer from this problem in Figure 5.9e: once the drone went past the obstacle at coordinates $(21, -2)$, the agent did not select the sectors containing the obstacle, meaning the only possible boundary for the bug algorithm to follow was the obstacle in front, and resulting in a more efficient maneuver to find an escape path.

Note that it is not always the case that the boundary to follow is in front of the bug (a counter-example is shown in Annex D), so the selection of the boundary to follow when there are multiple obstacles at similar distances is not trivial. This means a systematic rule for the bug algorithm would not be sufficient to select the right boundary every time, and the agent will not necessarily always pick the best obstacle at all times, as it does not have access to global information about the environment. To conclude, the agent achieved a better boundary-following behavior by selecting which obstacles are relevant for the bug algorithm, although it is not guaranteed that it will systematically find the shortest path.

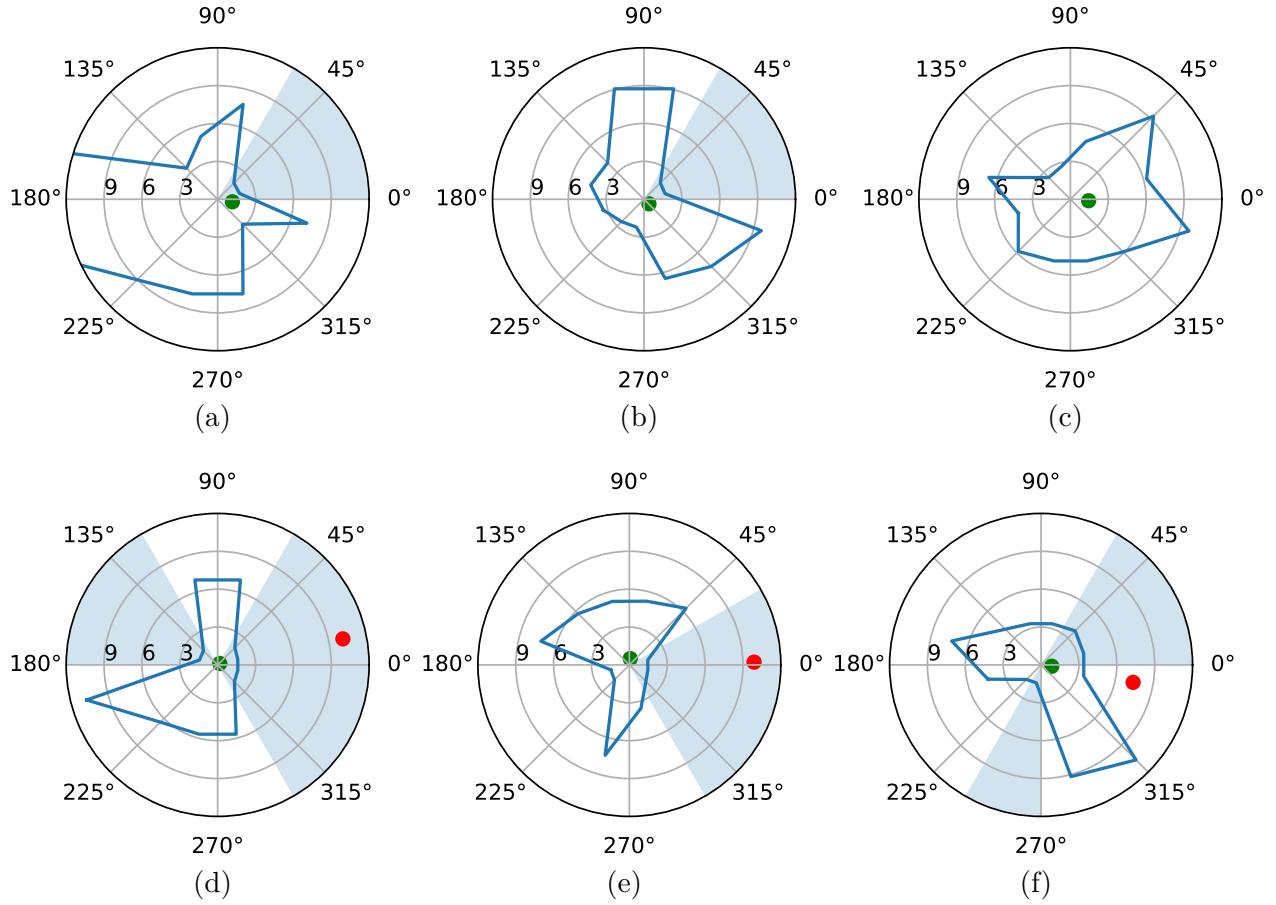


Figure 5.9 Decisions made by the agent at the points corresponding to the ones in Figure 5.8. The observation for one time step is shown in a polar graph, and the blue line represents the distances in all the sectors of the VFH representation, at their corresponding angle (e.g. in (a) the drone sees about 10 meters to the right, and sees walls at 5 meters at 90 and 270 degrees). The sectors chosen by the agent in each time step are highlighted with a light blue color. Finally, the green dot is the predicted position in the next timestep, calculated by the modified DWA, and the red dot is the goal position.

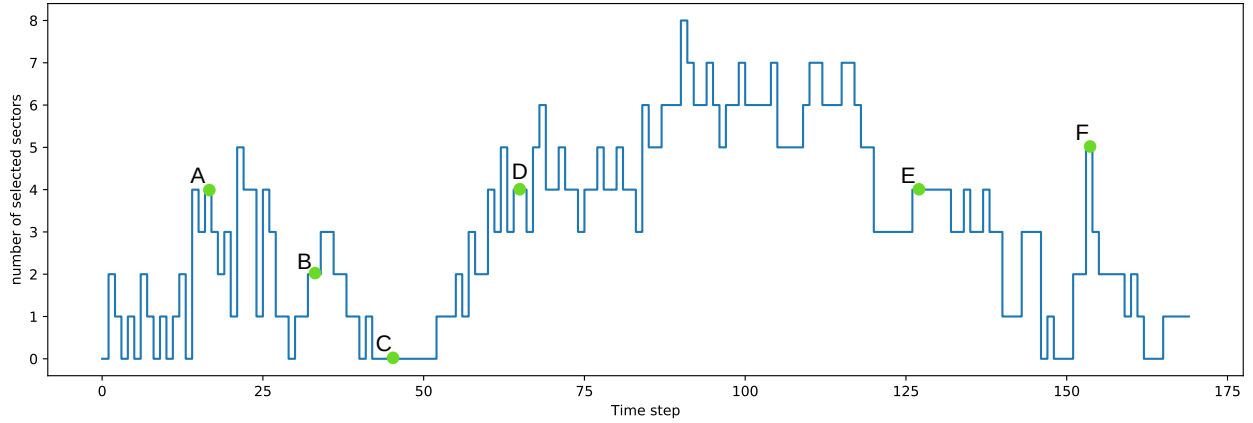


Figure 5.10 Number of sectors used by the agent at each time step of scenario 4. The discussed points A, B, C, D, E, and F shown in Figure 5.9 are labeled at the corresponding time step.

It can be observed in points A and B that the agent focuses on the closest obstacles. However, at point C, the agent judges its position safe enough to select no range data, meaning the obstacle avoidance pipeline is skipped for this time step. At point D, the agent correctly identifies the two walls of the narrow passage. The different states evaluated have shown that the agent will significantly change its sector usage, adapting to the situation. This is illustrated in Figure 5.10, which shows the average number of sectors used during the scenario. Thus, the agent is most effective in heterogeneous environments, where there might be a combination of easier and harder segments during navigation (e.g. going to a location at a high altitude first, then investigating the location at a much closer level). This is also reflected in the quantitative results of the previous evaluation, where the agent achieved a better acceleration of the obstacle-avoidance pipeline (2.7x) compared to the homogeneous, dense environment (2.3x).

5.3 Real Robot Experiments

We evaluate the proposed approach on an Agilex Limo Rover. It is equipped with a 2D LiDAR (EAI X2L LiDAR), an NVIDIA Jetson Nano, and a Cortex A-57 CPU. To show navigation can be performed with constrained resources, all the processing for obstacle detection, inference and navigation is done on the CPU. The simulated methods are implemented on the rover using ROS. As the rover navigates at lower velocities than the simulated drone, Focus bug is selected for the experiments to maximize the reduction in sector usage. The tests are done in 5 static scenarios and 3 dynamic scenarios. Static environments include

difficult features such as U-shaped obstacles (dead-ends) or narrow passages. Dynamic scenarios involve people crossing the path of the robot in three ways: moving out of its way, ignoring it, and actively moving in its way.

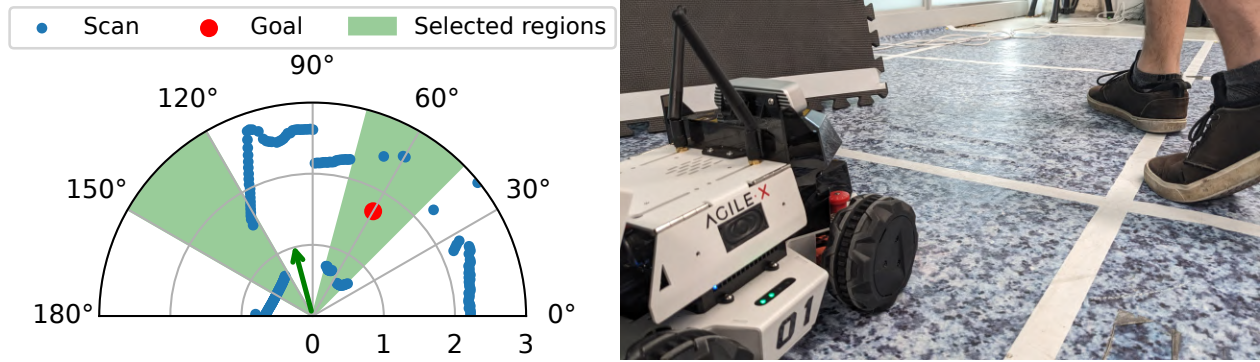


Figure 5.11 The robot focusing on part of the environment to avoid a human moving in its path. The left Figure presents the current observation in blue and the action of the agent in light green. The goal to reach is the red dot and the computed velocity by the DWA is the green arrow.

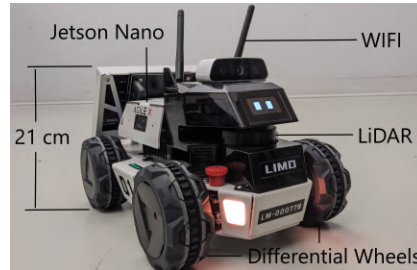


Figure 5.12 The AgileX Limo rover [2].

Table 5.11 Results of the rover experiments.

Method	Static environments			Dynamic environments		
	Success Rate (%)	Mission Time (s)	Path (m) Length	Success Rate (%)	Mission Time (s)	Path (m) Length
FB+DWA	97	22.3	7.32	100	14.6	5.74
DWA	95	24.7	7.25	100	14.8	5.81
Greedy+DWA	86	21.0	6.93	95	14.6	5.19

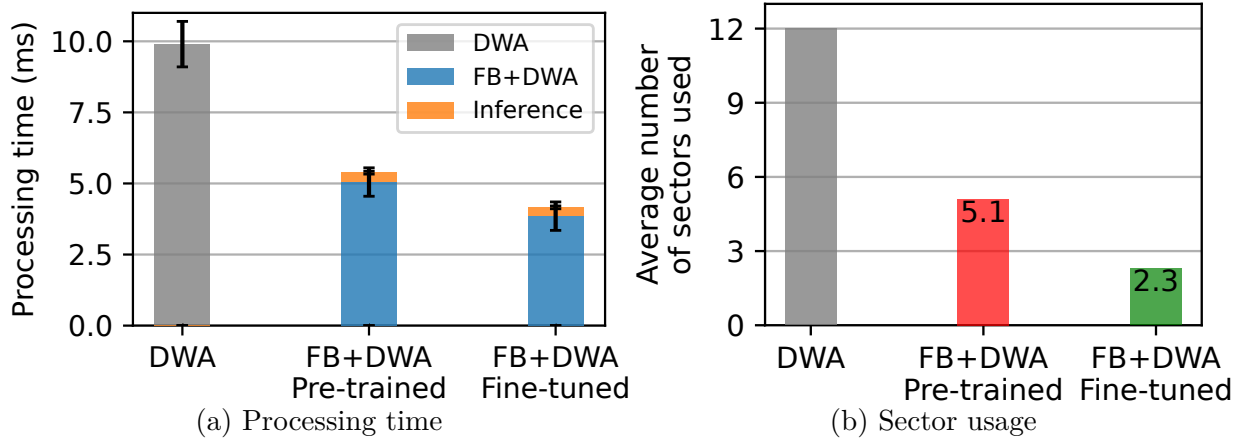


Figure 5.13 Comparison of the processing time and sector usage with Focus Bug on the Limo rover. The pre-trained model is directly applied from the simulation experiments and the fine-tuned model had additional training with a maximum velocity closer to the rover specifications. The initial training took around 21h07 minutes, and the training for the fine-tuning took 3h27 minutes.

As shown in Table 5.11, FB is able to generalize to the new robot and maintain high success rate. Note that generalization is not guaranteed, as robots with much higher speeds need to know more in advance about existing obstacles to react in time, but we show that Focus Bug can be applied to different robots as long as the navigating speeds are similar. It can also be observed in Fig. 5.13a that the average processing time is reduced by 2.6X with some additional training. The differences in robot speeds explain the improvement with fine-tuning: the simulated drone navigated at higher speeds, meaning the agent learned a strategy that becomes more conservative when applied to the rover. When fine-tuning the agent in simulation with the speed of the rover, Focus bug adapted to the new situation and was able to reduce sector usage from 5.1 down to 2.3 on average.

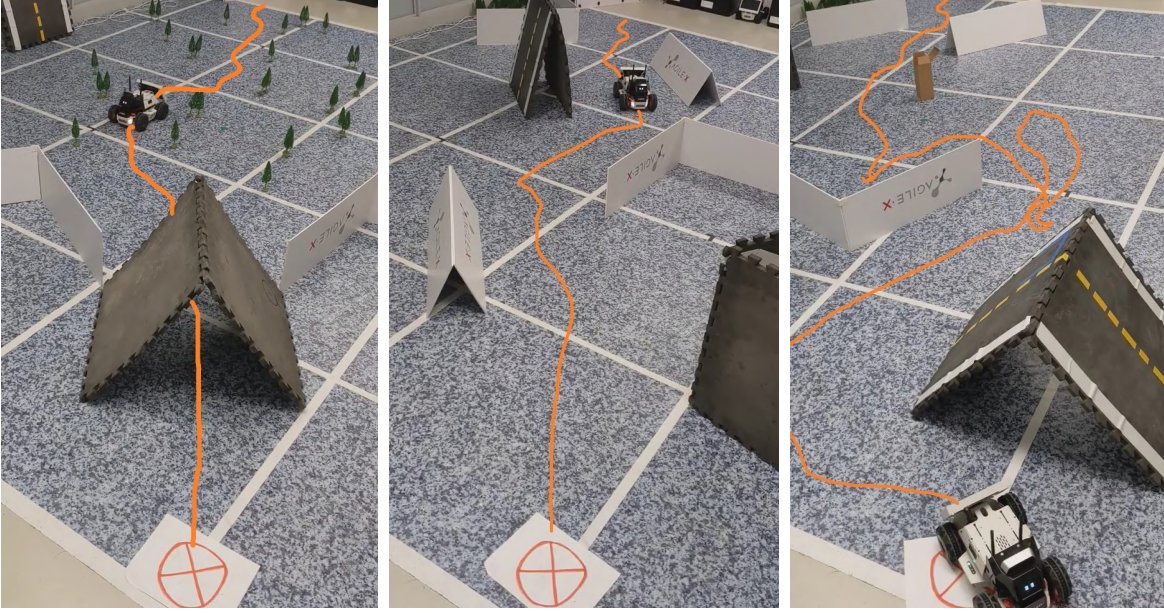


Figure 5.14 Examples of rover trajectories in static environments

Some examples of trajectories in static environments are shown in Fig. 5.14. Focus bug is able to navigate in dense environments and avoid U-shaped obstacles (right image) that are prone to cause local minima in classical mapless methods. A failure case for our approach is when the tangent bug incorrectly follows boundaries in the presence of too many obstacles. This was addressed by tuning DWA to allow closer boundary following.

5.4 Summary of key results

We do a summary of the key results presented in Chapter 5, comparing the proposed method with related work using reinforcement learning for navigation, and list the main contributions of this thesis.

The main achievement of the proposed method is the reduction of the processing time of the navigation pipeline, reducing it by a factor of 2.7X compared to the state-of-the-art, as shown in Fig. 5.3, while maintaining a high success rate. Table 5.12 summarizes simulation results obtained with classical DWA [53] and APF [54] methods, and one end-to-end RL method that is similar to our approach, using a 2D LiDAR scanner [132]. The results on the rover experiments are summarized in Table 5.11. They demonstrate that our method maintained the high success rate of classical methods, even outperforming them in the dynamic environments, because of its faster reaction time. As shown in Tables 5.11 and 5.12, even when applied to a rover with different dynamics than in training, the agent is able to

generalize and achieves a success rate of 99% in simulation and 97% on the rover. Moreover, the approach reduced the amount of processed data by 84%, improving the efficiency of the navigation algorithm. Other key results that were found in ablation studies are the trade-offs between the penalty for using sectors and the success rate, or the number of sectors used in the discretization and the success rate, which is illustrated in Appendices B and C. In brief, a discretization into more sectors and a lower cost for using them results in a higher success rate (fewer collisions), but at the cost of a lower reduction of the processing time.

Table 5.12 Simulation results on different methods.

Method	Static environments			Dynamic environments		
	Success Rate (%)	Mission Time (s)	Path (m) Length	Success Rate (%)	Mission Time (s)	Path (m) Length
FB+DWA	99	13.04	17.13	97	12.75	17.41
DWA [53]	99	13.96	17.42	93	12.89	16.34
Greedy+DWA	92	14.55	18.32	93	13.49	17.82
APF [54]	67	16.98	20.01	81	17.15	19.13
E2E RL [132]	52	12.44	18.62	73	13.03	18.70

When compared with other end-to-end RL approaches, this work achieves a higher success rate and fewer collisions, benefiting from the robustness of the well-established classical navigation methods. There are works presented in the literature review that have also achieved very high success rate with end-to-end RL, but they often make simplifying assumptions or are only tested in simulation [88] [81] [82] [85].

CHAPTER 6 CONCLUSION

This chapter concludes the thesis with a short summary of the work done and a discussion of its limitations. It finishes with a discussion of potential future research on the subject of the thesis.

6.1 Summary of Contributions

This thesis presented a study on the viability of applying deep reinforcement learning to make mapless navigation faster and more efficient, as opposed to traditional acceleration methods like hardware acceleration and algorithmic optimization. We designed an RL agent and trained it to reduce the amount of data processed by the obstacle-avoidance pipeline to only the information critical for safe navigation. To do this, we first modeled an environment to simulate a drone in an arena with obstacles, and proposed a methodology to design and train an agent to learn the task considered in this thesis in Chapter 4. Then, we specified multiple metrics and designed scenarios to evaluate the agent, compare the new method with baselines, and investigate the learned strategies in Chapter 5.

In the simulation experiments, the agent learned to efficiently select the relevant information for obstacle-avoidance, reducing the amount of data processed by the Dynamic-Window Approach by a factor of 7.7x, and the total processing time of the navigation algorithm by 2.7x. The agent achieved the same success rate as the baseline, which had access to all the LiDAR data, (99%) in the dense environment, and even surpassed the traditional approach in the heterogeneous environment, with a success rate of 98% (vs 93% for the traditional approach). The agent achieved this by exploiting the heterogeneity of the environment to adapt the behavior of the drone around obstacles. However, the agent was found to be less robust when the rewards penalizing sector usage were too important, as it was taking more risks by getting closer to the obstacles, causing 2-3% collisions. Still, this outperforms the state-of-the-art for works that tried to use an end-to-end RL approach [88] [81] [82] [85]. Another agent trained with smaller incentives for efficiency achieved more robustness (no collisions) with a slightly lower acceleration (2.3x). This is in general lower than hardware acceleration methods that achieved speedups from 1.6x to 31.3x [102] [110] [113]. However, those methods require specialized hardware that is designed with a specific robot and task in mind and are much less transferable than the proposed method. Moreover, those 2 acceleration techniques are not mutually exclusive, and an even higher acceleration could be achieved by combining the two.

We then designed 3 scenarios to explain the decisions of the agent and showed that the learned strategy involved focusing generally on close obstacles that are either between the drone and its goal, or in the direction of its velocity. Further, the agent learned to skip the obstacle-avoidance pipeline completely when the drone was in a safe position. Finally, the agent learned to prioritize certain obstacles depending on the situation to help the bug algorithm find more efficient paths toward the goal.

In the rover experiments, the agent was able to generalize and maintain a success rate of 97%, while still reducing the processing time by a factor of 2.7X. Note that this required fine-tuning the agent to the different dynamics of the rover, but the additional training time was 7 times lower than the initial training time. The agent outperformed the greedy baseline introduced in this thesis and maintained the high success rate of the traditional navigation pipeline (DWA).

6.2 Limitations

The first limitation of our approach is the robustness of the agents. As discussed in Chapter 5, there is a trade-off between the efficiency of the agent and its robustness to collisions. A value too high for the cost of sectors in the rewards function led to the agent discarding information too aggressively, and the drone started to collide. Fortunately, the value can be tuned and multiple experiments allowed to find a good compromise between efficiency and safety. Still, it is a time consuming process and, worse than that, it is impossible to explore every possible state while testing the agent to make sure the drone will never collide. Depending on the task, achieving a comparable success rate to the traditional approaches might be sufficient, but RL might not be applicable to situations where the behavior needs to be specified for all situations (e.g. carrying precious equipment or rescue missions). To mitigate this, evaluating the agent in real-world conditions for a long enough period of time might build a higher confidence in the robustness. Further, a more conservative reward function (like Safety Bug) can be set when safety is more critical.

Another limitation is the time, knowledge, and exploration needed to successfully design an agent to learn the task and train it. In this thesis, a lot of different designs were explored with different action spaces, observation spaces, reward functions, agent architectures, learning hyperparameters, etc. before having a performance on the same level as the state-of-the-art. All of this required a lot of time and knowledge on both DRL and navigation, so it cannot be claimed that using reinforcement learning allowed to solve this problem faster or was better than designing an algorithm to perform the same task. However, it makes a case for the applicability of RL for similar tasks by doing a proof of concept and reaching state-of-the-art

performance. The final method is also more easily applicable to different robots as tasks, as the real robot experiments demonstrated

Finally, another limitation of the designed approach for sensor selection is that it only considers instantaneous sensory data, so it is most suited for mapless navigation. Significant work would be needed to apply it to map-based navigation methods, as the range data selection would lead to incomplete maps or more difficult localization.

6.3 Future Research

Since the proposed approach is restricted to only mapless navigation, one direction of future research emerging from this is to explore the possibility of extending the scope of the agent to map-based navigation. For example, making the agent select parts of an entire map to help a map-based global planner reduce its search space. The agent could even be trained to act on the various parameters of navigation methods and adapt them dynamically, as explored in [32] and [33].

Also, if more time was available for training, more reward functions could be explored to give the agent more fundamental rewards, instead of explicitly encouraging some behaviors, which might lead to the discovery of more efficient strategies. For this, it would be interesting to explore more recent simulators such as NVIDIA’s Isaac Gym [124]. It allows parallelization of the environments, which can get multiple orders of magnitude more training data, depending on the available computing resources.

Further, making smart decisions to improve the efficiency of map-less navigation is not limited to identifying the relevant data, but could be extended to much more aspects: control frequency (or when is the next important moment to make a decision), maximum velocity of the drone, or even dynamically change the parameters of the obstacle-avoidance pipeline. It would be interesting to see how much more efficient navigation can become by expanding the action space of the agents in that direction, although it would extend the training time significantly and pose a new challenge to design a good reward function.

In the long run, having access to realistic simulators that can gather data much more quickly, it would be interesting to investigate more end-to-end RL approaches to remove the need for conservative, traditional navigation methods. Such works already exist, but are not yet as robust as the traditional approaches. However, it is an active field of research, and solving this problem would allow for a much bigger democratization and ease of development of autonomous robots.

REFERENCES

- [1] YLIDAR. (2018) Ydlidar x2. [Online]. Available: <https://www.ydlidar.com/products/view/6.html>
- [2] “LIMO.” [Online]. Available: <https://global.agilex.ai/products/limo>
- [3] M. Ben Chaim, “Energy Limitations and Energetic Efficiency of Mobile Robots,” in *Autonomous Mobile Robots and Multi-Robot Systems*. John Wiley & Sons, Ltd, 2019, pp. 183–197, section: 8 _eprint: <https://onlinelibrary.wiley.com/doi/pdf/10.1002/9781119213154.ch8>. [Online]. Available: <https://onlinelibrary.wiley.com/doi/abs/10.1002/9781119213154.ch8>
- [4] Y. Mei, Y.-H. Lu, Y. Hu, and C. Lee, “Deployment of mobile robots with energy and timing constraints,” *IEEE Transactions on Robotics*, vol. 22, no. 3, pp. 507–522, 2006.
- [5] D. Filliat and J.-A. Meyer, “Map-based navigation in mobile robots:: I. a review of localization strategies,” *Cognitive Systems Research*, vol. 4, no. 4, pp. 243–282, 2003. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S1389041703000081>
- [6] D. Shah, N. Yang, and T. M. Aamodt, “Energy-efficient realtime motion planning,” in *Proceedings of the 50th Annual International Symposium on Computer Architecture*, ser. ISCA ’23. New York, NY, USA: Association for Computing Machinery, 2023. [Online]. Available: <https://doi.org/10.1145/3579371.3589092>
- [7] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.
- [8] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, “Mastering the game of go with deep neural networks and tree search,” vol. 529, no. 7587, pp. 484–489. [Online]. Available: <https://doi.org/10.1038/nature16961>
- [9] Agent57: Outperforming the human atari benchmark. [Online]. Available: <https://www.deepmind.com/blog/agent57-outperforming-the-human-atari-benchmark>

- [10] OpenAI, C. Berner, G. Brockman, B. Chan, V. Cheung, P. Dębiak, C. Dennison, D. Farhi, Q. Fischer, S. Hashme, C. Hesse, R. Józefowicz, S. Gray, C. Olsson, J. Pachocki, M. Petrov, H. P. d. O. Pinto, J. Raiman, T. Salimans, J. Schlatter, J. Schneider, S. Sidor, I. Sutskever, J. Tang, F. Wolski, and S. Zhang, “Dota 2 with large scale deep reinforcement learning.” [Online]. Available: <http://arxiv.org/abs/1912.06680>
- [11] R. Bellman, *Dynamic Programming*. Dover Publications, 1957.
- [12] R. S. Sutton, “Dyna, an integrated architecture for learning, planning, and reacting,” *SIGART Bull.*, vol. 2, pp. 160–163, 1990.
- [13] T. Hester, M. Quinlan, and P. Stone, “Rtmba: A real-time model-based reinforcement learning architecture for robot control,” in *2012 IEEE International Conference on Robotics and Automation*, 2012, pp. 85–90.
- [14] C. J. C. H. Watkins and P. Dayan, “Q-learning,” *Machine Learning*, vol. 8, no. 3, pp. 279–292, May 1992. [Online]. Available: <https://doi.org/10.1007/BF00992698>
- [15] OpenAI baselines: ACKTR & a2c. [Online]. Available: <https://openai.com/research/openai-baselines-acktr-a2c>
- [16] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, “Playing atari with deep reinforcement learning.” [Online]. Available: <http://arxiv.org/abs/1312.5602>
- [17] S. Ruder, “An overview of gradient descent optimization algorithms.” [Online]. Available: <http://arxiv.org/abs/1609.04747>
- [18] H. v. Hasselt, A. Guez, and D. Silver, “Deep reinforcement learning with double q-learning,” in *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, ser. AAAI’16. AAAI Press, 2016, p. 2094–2100.
- [19] Z. Wang, T. Schaul, M. Hessel, H. Van Hasselt, M. Lanctot, and N. De Freitas, “Dueling network architectures for deep reinforcement learning,” in *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, ser. ICML’16. JMLR.org, 2016, p. 1995–2003.
- [20] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, “Prioritized experience replay,” 2016.

- [21] R. J. Williams, “Simple statistical gradient-following algorithms for connectionist reinforcement learning,” vol. 8, no. 3, pp. 229–256. [Online]. Available: <https://doi.org/10.1007/BF00992696>
- [22] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, “Continuous control with deep reinforcement learning.” [Online]. Available: <http://arxiv.org/abs/1509.02971>
- [23] J. Schulman, S. Levine, P. Moritz, M. I. Jordan, and P. Abbeel, “Trust region policy optimization.” [Online]. Available: <http://arxiv.org/abs/1502.05477>
- [24] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, “Asynchronous methods for deep reinforcement learning.” [Online]. Available: <http://arxiv.org/abs/1602.01783>
- [25] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, “Proximal policy optimization algorithms.” [Online]. Available: <http://arxiv.org/abs/1707.06347>
- [26] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, “Deterministic policy gradient algorithms.”
- [27] R. Siegwart, I. R. Nourbakhsh, and D. Scaramuzza, *Introduction to Autonomous Mobile Robots*, 2nd ed. The MIT Press, 2011.
- [28] A. Zhang, R. L. Truby, L. Chin, S. Li, and D. Rus, “Vision-based sensing for electrically-driven soft actuators,” vol. 7, no. 4, pp. 11 509–11 516, conference Name: IEEE Robotics and Automation Letters.
- [29] F. SA. What is a LiDAR drone? [Online]. Available: <https://www.flyability.com/lidar-drone>
- [30] B. Rogers and M. Graham, “Motion parallax as an independent cue for depth perception,” *Perception*, vol. 8, no. 2, pp. 125–134, 1979, pMID: 471676. [Online]. Available: <https://doi.org/10.1068/p080125>
- [31] M. S. Güzel, “Autonomous vehicle navigation using vision and mapless strategies: A survey,” *Advances in Mechanical Engineering*, vol. 5, p. 234747, 2013. [Online]. Available: <https://doi.org/10.1155/2013/234747>
- [32] Z. Li, X. Wang, X. Liu, and J. Jiang, “BinsFormer: Revisiting adaptive bins for monocular depth estimation.” [Online]. Available: <http://arxiv.org/abs/2204.00987>

- [33] Editorial, “Mobile robot localization - Challenges and various methods,” Jan. 2023. [Online]. Available: <https://roboticsbiz.com/mobile-robot-localization-challenges-and-various-methods/>
- [34] J. Kunhoth, A. Karkar, S. Al-Maadeed, and A. Al-Ali, “Indoor positioning and wayfinding systems: a survey,” vol. 10, no. 1, p. 18. [Online]. Available: <https://doi.org/10.1186/s13673-020-00222-0>
- [35] “GPS based Localization for Self-Driving Robots.” [Online]. Available: <https://www.blackcoffeerobotics.com/blog/gps-based-localization-for-self-driving-robots>
- [36] P. Panigrahi and S. Bisoy, “Localization strategies for autonomous mobile robots: A review,” *Journal of King Saud University - Computer and Information Sciences*, vol. 34, 03 2021.
- [37] S. Huang and G. Dissanayake, “A critique of current developments in simultaneous localization and mapping,” *International Journal of Advanced Robotic Systems*, vol. 13, no. 5, p. 1729881416669482, 2016. [Online]. Available: <https://doi.org/10.1177/1729881416669482>
- [38] N. Yatim and N. Buniyamin, “Particle filter in simultaneous localization and mapping (slam) using differential drive mobile robot,” *Jurnal Teknologi*, vol. 77, 12 2015.
- [39] Q. Li, R. Li, K. Ji, and W. Dai, “Kalman filter and its application,” in *2015 8th International Conference on Intelligent Networks and Intelligent Systems (ICINIS)*, 2015, pp. 74–77.
- [40] A. R. Khairuddin, M. S. Talib, and H. Haron, “Review on simultaneous localization and mapping (slam),” in *2015 IEEE International Conference on Control System, Computing and Engineering (ICCSCE)*, 2015, pp. 85–90.
- [41] M. F. Ahmed, K. Masood, and V. Fremont, “Active SLAM: A Review On Last Decade,” May 2023, arXiv:2212.11654 [cs]. [Online]. Available: <http://arxiv.org/abs/2212.11654>
- [42] I. Abaspor Kazerouni, L. Fitzgerald, G. Dooly, and D. Toal, “A survey of state-of-the-art on visual SLAM,” vol. 205, p. 117734. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0957417422010156>
- [43] X. Fan, Y. Wang, and Z. Zhang, “An evaluation of lidar-based 2d slam techniques with an exploration mode,” *Journal of Physics: Conference Series*, vol. 1905, p. 012021, 05 2021.

- [44] H. Bai, “ICP Algorithm: Theory, Practice And Its SLAM-oriented Taxonomy,” *Applied and Computational Engineering*, vol. 2, no. 1, pp. 10–21, Mar. 2023, arXiv:2206.06435 [cs]. [Online]. Available: <http://arxiv.org/abs/2206.06435>
- [45] P. Biber and W. Straßer, “The normal distributions transform: A new approach to laser scan matching,” vol. 3, 11 2003, pp. 2743 – 2748 vol.3.
- [46] H. Zhang, N. Chen, G. Fan, and D. Yang, “An improved scan matching algorithm in slam,” in *2019 6th International Conference on Systems and Informatics (ICSAI)*, 2019, pp. 160–164.
- [47] K. A. Tsintotas, L. Bampis, and A. Gasteratos, “The revisiting problem in simultaneous localization and mapping: A survey on visual loop closure detection,” *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, no. 11, pp. 19 929–19 953, nov 2022. [Online]. Available: <https://doi.org/10.1109%2Ftits.2022.3175656>
- [48] T. T. Omar Takleh, N. abu bakar, S. Rahman, R. Hamzah, and Z. Abd Aziz, “A brief survey on slam methods in autonomous vehicle,” *International Journal of Engineering and Technology(UAE)*, vol. 7, pp. 38–43, 11 2018.
- [49] A. Rao, “Trajectory optimization: A survey,” *Lecture Notes in Control and Information Sciences*, vol. 455, pp. 3–21, 03 2014.
- [50] G. Klančar, A. Zdešar, S. Blažič, and I. Škrjanc, “Chapter 7 - autonomous guided vehicles,” in *Wheeled Mobile Robotics*, G. Klančar, A. Zdešar, S. Blažič, and I. Škrjanc, Eds. Butterworth-Heinemann, 2017, pp. 387–418. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/B978012804204500007X>
- [51] D. Foead, A. Ghifari, M. B. Kusuma, N. Hanafiah, and E. Gunawan, “A systematic literature review of a* pathfinding,” *Procedia Computer Science*, vol. 179, pp. 507–514, 2021, 5th International Conference on Computer Science and Computational Intelligence 2020. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S1877050921000399>
- [52] S. M. LaValle, “Rapidly-exploring random trees : a new tool for path planning,” *The annual research report*, 1998.
- [53] D. Fox, W. Burgard, and S. Thrun, “The dynamic window approach to collision avoidance,” vol. 4, no. 1, pp. 23–33, conference Name: IEEE Robotics & Automation Magazine.

- [54] O. Khatib, “Real-time obstacle avoidance for manipulators and mobile robots,” in *1985 IEEE International Conference on Robotics and Automation Proceedings*, vol. 2, pp. 500–505.
- [55] J. Borenstein and Y. Koren, “The vector field histogram-fast obstacle avoidance for mobile robots,” vol. 7, no. 3, pp. 278–288, conference Name: IEEE Transactions on Robotics and Automation.
- [56] A. Shitsukane, W. Cheriuyot, and M. Mgala, “A survey on obstacles avoidance mobile robot in static unknown environment,” *International Journal of Computer (IJC)*, 03 2018.
- [57] H. Choset, K. M. Lynch, S. Hutchinson, G. A. Kantor, W. Burgard, L. E. Kavraki, and S. Thrun, *Bug Algorithms*, 2005, pp. 17–38.
- [58] K. N. McGuire, G. C. H. E. de Croon, and K. Tuyls, “A comparative study of bug algorithms for robot navigation,” vol. 121, p. 103261. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0921889018306687>
- [59] I. Kamon, E. Rivlin, and E. Rimon, “A new range-sensor based globally convergent navigation algorithm for mobile robots,” in *Proceedings of IEEE International Conference on Robotics and Automation*, vol. 1, 1996, pp. 429–435 vol.1.
- [60] A. Chatterjee, A. Rakshit, and N. N. Singh, “Mobile Robot Navigation,” in *Vision Based Autonomous Robot Navigation: Algorithms and Implementations*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 1–20. [Online]. Available: https://doi.org/10.1007/978-3-642-33965-3_1
- [61] F. AlMahamid and K. Grolinger, “Autonomous unmanned aerial vehicle navigation using reinforcement learning: A systematic review,” vol. 115, p. 105321. [Online]. Available: <http://arxiv.org/abs/2208.12328>
- [62] K. Norberg, *Audio Visual Communication Review*, vol. 1, no. 3, pp. 190–194, 1953. [Online]. Available: <http://www.jstor.org/stable/30216658>
- [63] G. Desouza and A. Kak, “Vision for mobile robot navigation: a survey,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 24, no. 2, pp. 237–267, 2002.
- [64] V. Delsart, “Navigation autonome en environnement dynamique: Une approche par déformation de trajectoire,” Ph.D. dissertation, Université de Grenoble, 2010.

- [65] Q. Yao, Z. Zheng, L. Qi, H. Yuan, X. Guo, M. Zhao, Z. Liu, and T. Yang, "Path planning method with improved artificial potential field—a reinforcement learning perspective," vol. 8, pp. 135 513–135 523, conference Name: IEEE Access.
- [66] J. Sfeir, M. Saad, and H. Saliyah-Hassane, "An improved artificial potential field approach to real-time mobile robot path planning in an unknown environment," in *2011 IEEE International Symposium on Robotic and Sensors Environments (ROSE)*, pp. 208–213.
- [67] U. Orozco-Rosas, O. Montiel, and R. Sepúlveda, "Mobile robot path planning using membrane evolutionary artificial potential field," vol. 77, pp. 236–251. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S1568494619300420>
- [68] E. Alagić, J. Velagić, and A. Osmanović, "Design of mobile robot motion framework based on modified vector field histogram," in *2019 International Symposium ELMAR*, pp. 135–138, ISSN: 1334-2630.
- [69] M. Missura and M. Bennewitz, "Predictive collision avoidance for the dynamic window approach," in *2019 International Conference on Robotics and Automation (ICRA)*, pp. 8620–8626, ISSN: 2577-087X.
- [70] M. Seder and I. Petrovic, "Dynamic window based approach to mobile robot motion control in the presence of moving obstacles," in *Proceedings 2007 IEEE International Conference on Robotics and Automation*, pp. 1986–1991, ISSN: 1050-4729.
- [71] D. H. Lee, S. S. Lee, C. K. Ahn, P. Shi, and C.-C. Lim, "Finite distribution estimation-based dynamic window approach to reliable obstacle avoidance of mobile robot," vol. 68, no. 10, pp. 9998–10 006, conference Name: IEEE Transactions on Industrial Electronics.
- [72] P. Fiorini and Z. Shiller, "Motion planning in dynamic environments using velocity obstacles," *The International Journal of Robotics Research*, vol. 17, no. 7, pp. 760–772, 1998. [Online]. Available: <https://doi.org/10.1177/027836499801700706>
- [73] F. Large, C. Laugier, and Z. Shiller, "Navigation among moving obstacles using the NLVO: Principles and applications to intelligent vehicles," vol. 19, no. 2, pp. 159–171. [Online]. Available: <https://doi.org/10.1007/s10514-005-0610-8>
- [74] O. Gal, Z. Shiller, and E. Rimon, "Efficient and safe on-line motion planning in dynamic environments," in *2009 IEEE International Conference on Robotics and Automation*, pp. 88–93, ISSN: 1050-4729.

- [75] P. Chen, Y. Huang, E. Papadimitriou, J. Mou, and P. van Gelder, "Global path planning for autonomous ship: A hybrid approach of fast marching square and velocity obstacles methods," vol. 214, p. 107793. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0029801820307691>
- [76] Minimal navigation solution for a swarm of tiny flying robots to explore an unknown environment | science robotics. [Online]. Available: <https://www.science.org/doi/10.1126/scirobotics.aaw9710>
- [77] B. P. Duisterhof, S. Li, J. Burgués, V. J. Reddi, and G. C. H. E. de Croon, "Sniffy bug: A fully autonomous swarm of gas-seeking nano quadcopters in cluttered environments." [Online]. Available: <http://arxiv.org/abs/2107.05490>
- [78] M. M. Gulzar, Q. Ling, M. Yaqoob, and S. Iqbal, "Realization of an improved path planning strategy," in *2015 International Conference on Control, Automation and Information Sciences (ICCAIS)*, pp. 384–389.
- [79] R. Marino, F. Mastrogiovanni, A. Sgorbissa, and R. Zaccaria, "A minimalistic quadro- tor navigation strategy for indoor multi-floor scenarios," in *Intelligent Autonomous Systems 13*, ser. Advances in Intelligent Systems and Computing, E. Menegatti, N. Michael, K. Berns, and H. Yamaguchi, Eds. Springer International Publishing, pp. 1561–1570.
- [80] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning." [Online]. Available: <http://arxiv.org/abs/1509.02971>
- [81] B. Li and Y. Wu, "Path planning for UAV ground target tracking via deep reinforcement learning," vol. 8, pp. 29 064–29 074, conference Name: IEEE Access.
- [82] L. He, N. Aouf, J. F. Whidborne, and B. Song, "Deep reinforcement learning based local planner for UAV obstacle avoidance using demonstration data." [Online]. Available: <http://arxiv.org/abs/2008.02521>
- [83] H. X. Pham, H. M. La, D. Feil-Seifer, and L. Van Nguyen, "Reinforcement learning for autonomous UAV navigation using function approximation," in *2018 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*, pp. 1–6, ISSN: 2475-8426.
- [84] S.-Y. Shin, Y.-W. Kang, and Y.-G. Kim, "Obstacle avoidance drone by deep reinforcement learning and its racing with human pilot," vol. 9, no. 24, p. 5571,

- number: 24 Publisher: Multidisciplinary Digital Publishing Institute. [Online]. Available: <https://www.mdpi.com/2076-3417/9/24/5571>
- [85] A. Singla, S. Padakandla, and S. Bhatnagar, “Memory-based deep reinforcement learning for obstacle avoidance in UAV with limited environment knowledge,” vol. 22, no. 1, pp. 107–118, conference Name: IEEE Transactions on Intelligent Transportation Systems.
 - [86] O. Bouhamed, H. Ghazzai, H. Besbes, and Y. Massoud, “Autonomous UAV navigation: A DDPG-based deep reinforcement learning approach.” [Online]. Available: <http://arxiv.org/abs/2003.10923>
 - [87] C. Greatwood and A. G. Richards, “Reinforcement learning and model predictive control for robust embedded quadrotor guidance and control,” vol. 43, no. 7, pp. 1681–1693. [Online]. Available: <https://doi.org/10.1007/s10514-019-09829-4>
 - [88] S. Jung and D. H. Shim, “Mapless navigation: Learning UAVs motion for Exploration of unknown environments.” [Online]. Available: <http://arxiv.org/abs/2110.01747>
 - [89] K. Dalamagkidis, “Classification of UAVs,” in *Handbook of Unmanned Aerial Vehicles*, K. P. Valavanis and G. J. Vachtsevanos, Eds. Springer Netherlands, pp. 83–91. [Online]. Available: https://doi.org/10.1007/978-90-481-9707-1_94
 - [90] Blog. [Online]. Available: <https://www.skydio.com/blog/>
 - [91] R. Wood, B. Finio, M. Karpelson, K. Ma, N. Pérez-Arancibia, P. Sreetharan, H. Tanaka, and J. Whitney, “Progress on ‘pico’ air vehicles,” *The International Journal of Robotics Research*, vol. 31, no. 11, pp. 1292–1302, 2012. [Online]. Available: <https://doi.org/10.1177/0278364912455073>
 - [92] K. Y. Ma, P. Chirarattananon, S. B. Fuller, and R. J. Wood, “Controlled flight of a biologically inspired, insect-scale robot,” *Science*, vol. 340, no. 6132, pp. 603–607, 2013. [Online]. Available: <https://www.science.org/doi/abs/10.1126/science.1231806>
 - [93] Z. Wan, A. S. Lele, and A. Raychowdhury, “Circuit and system technologies for energy-efficient edge robotics: (invited paper),” in *2022 27th Asia and South Pacific Design Automation Conference (ASP-DAC)*, pp. 275–280, ISSN: 2153-697X.
 - [94] S. Murray, W. Floyd-Jones, Y. Qi, D. Sorin, and G. Konidaris, *Robot Motion Planning on a Chip*.

- [95] S. Murray, W. Floyd-Jones, Y. Qi, G. Konidaris, and D. J. Sorin, “The microarchitecture of a real-time robot motion planning accelerator,” in *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pp. 1–12.
- [96] A. Suleiman, Z. Zhang, L. Carlone, S. Karaman, and V. Sze, “Navion: A 2-mW fully integrated real-time visual-inertial odometry accelerator for autonomous navigation of nano drones,” vol. 54, no. 4, pp. 1106–1119, conference Name: IEEE Journal of Solid-State Circuits.
- [97] M. Bilal, K. Munawar, M. S. Shaikh, and U. M. Al-Saggaf, “Resource-optimized bare-metal DSP implementation of UAV flight control software,” in *6th International Conference on Intelligent and Advanced Systems, ICIAS 2016, August 15, 2016 - August 17, 2016*, ser. International Conference on Intelligent and Advanced Systems, ICIAS 2016. Institute of Electrical and Electronics Engineers Inc., p. TECHSOURCE.
- [98] T. Chen and S. Lu, “Autonomous navigation control system of agricultural mini-unmanned aerial vehicles based on DSP,” vol. 28, no. 21, pp. 164–169, publisher: Chinese Society of Agricultural Engineering.
- [99] L. Liu, S. Liu, Z. Zhang, B. Yu, J. Tang, and Y. Xie, “PIRT: A runtime framework to enable energy-efficient real-time robotic applications on heterogeneous architectures.” [Online]. Available: <http://arxiv.org/abs/1802.08359>
- [100] S. Liu, M. Watterson, S. Tang, and V. Kumar, “High speed navigation for quadrotors with limited onboard sensing,” in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1484–1491.
- [101] S. Sudhakar, S. Karaman, and V. Sze, “Balancing actuation and computing energy in motion planning,” in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 4259–4265, ISSN: 2577-087X.
- [102] B. Boroujerdian, R. Ghosal, J. Cruz, B. Plancher, and V. J. Reddi, “RoboRun: A robot runtime to exploit spatial heterogeneity,” in *2021 58th ACM/IEEE Design Automation Conference (DAC)*, pp. 829–834, ISSN: 0738-100X.
- [103] N. Thelasingha, K. Nallan, A. A. Julius, and S. Mishra, “Acceleration of heuristic motion planning for unmanned aerial vehicles,” in *78th Vertical Flight Society Annual Forum and Technology Display, FORUM 2022, May 10, 2022 - May 12, 2022*, ser. 78th Vertical Flight Society Annual Forum and Technology Display, FORUM 2022. Vertical Flight Society.

- [104] B. Boroujerdian, H. Genc, S. Krishnan, W. Cui, A. Faust, and V. J. Reddi, “MAVBench: Micro aerial vehicle benchmarking.” [Online]. Available: <http://arxiv.org/abs/1905.06388>
- [105] P. Beigi, M. S. Rajabi, and S. Aghakhani, “An overview of drone energy consumption factors and models.” [Online]. Available: <http://arxiv.org/abs/2206.10775>
- [106] S. Murray, W. Floyd-Jones, G. Konidaris, and D. J. Sorin, “A programmable architecture for robot motion planning acceleration,” in *2019 IEEE 30th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, vol. 2160-052X, 2019, pp. 185–188.
- [107] J. Bialkowski, S. Karaman, and E. Frazzoli, “Massively parallelizing the RRT and the RRT*,” accepted: 2013-10-21T14:44:56Z ISBN: 9781612844565 Publisher: Institute of Electrical and Electronics Engineers (IEEE). [Online]. Available: <https://dspace.mit.edu/handle/1721.1/81448>
- [108] Y. Huang, J. Xu, M. Shi, and L. Liu, “Time-efficient coverage path planning for energy-constrained UAV,” vol. 2022, publisher: Hindawi Limited.
- [109] N. Sudha and A. R. Mohan, “Design of a hardware accelerator for path planning on the euclidean distance transform,” vol. 54, no. 1, pp. 253–264. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S1383762107000951>
- [110] M. Bakhshalipour, S. B. Ehsani, M. Qadri, D. Guri, M. Likhachev, and P. B. Gibbons, “RACOD: Algorithm/hardware co-design for mobile robot path planning,” in *49th IEEE/ACM International Symposium on Computer Architecture, ISCA 2022, June 18, 2022 - June 22, 2022*, ser. Proceedings - International Symposium on Computer Architecture. Institute of Electrical and Electronics Engineers Inc., pp. 597–609.
- [111] K. Wurm, A. Hornung, M. Bennewitz, C. Stachniss, and W. Burgard, “Octomap: A probabilistic, flexible, and compact 3d map representation for robotic systems,” vol. 2, 01 2010.
- [112] S. Lian, Y. Han, X. Chen, Y. Wang, and H. Xiao, “Dadu-p: A scalable accelerator for robot motion planning in a dynamic environment,” in *2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)*, 2018, pp. 1–6.
- [113] J. Sacks, D. Mahajan, R. C. Lawson, H. Esmailzadeh, and B. Khaleghi, “RoboX: An end-to-end solution to accelerate autonomous control in robotics,” in *2018 ACM/IEEE*

- 45th Annual International Symposium on Computer Architecture (ISCA)*, pp. 479–490, ISSN: 2575-713X.
- [114] V. Kangunde, R. S. Jamisola, and E. K. Theophilus, “A review on drones controlled in real-time,” vol. 9, no. 4, pp. 1832–1846. [Online]. Available: <https://doi.org/10.1007/s40435-020-00737-5>
 - [115] S. Shah, D. Dey, C. Lovett, and A. Kapoor, “AirSim: High-Fidelity Visual and Physical Simulation for Autonomous Vehicles,” Jul. 2017, arXiv:1705.05065 [cs]. [Online]. Available: <http://arxiv.org/abs/1705.05065>
 - [116] N. Koenig and A. Howard, “Design and use paradigms for gazebo, an open-source multi-robot simulator,” in *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*, vol. 3, 2004, pp. 2149–2154 vol.3.
 - [117] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, “CARLA: An Open Urban Driving Simulator,” Nov. 2017, arXiv:1711.03938 [cs]. [Online]. Available: <http://arxiv.org/abs/1711.03938>
 - [118] W. Koch, “Flight Controller Synthesis Via Deep Reinforcement Learning,” Sep. 2019, arXiv:1909.06493 [cs, eess, stat]. [Online]. Available: <http://arxiv.org/abs/1909.06493>
 - [119] “PyRobot · An Open Source Robotics Research Platform.” [Online]. Available: <https://pyrobot.org/>
 - [120] S. Krishnan, B. Boroujerdian, W. Fu, A. Faust, and V. J. Reddi, “Air Learning: An AI Research Platform for Algorithm-Hardware Benchmarking of Autonomous Aerial Robots,” Apr. 2021, arXiv:1906.00421 [cs]. [Online]. Available: <http://arxiv.org/abs/1906.00421>
 - [121] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, “OpenAI Gym,” Jun. 2016, arXiv:1606.01540 [cs]. [Online]. Available: <http://arxiv.org/abs/1606.01540>
 - [122] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann, “Stable-Baselines3: Reliable Reinforcement Learning Implementations,” *Journal of Machine Learning Research*, vol. 22, no. 268, pp. 1–8, 2021. [Online]. Available: <http://jmlr.org/papers/v22/20-1364.html>
 - [123] L. Engstrom, A. Ilyas, S. Santurkar, D. Tsipras, F. Janoos, L. Rudolph, and A. Madry, “Implementation matters in deep policy gradients: A case study on ppo and trpo,” 2020.

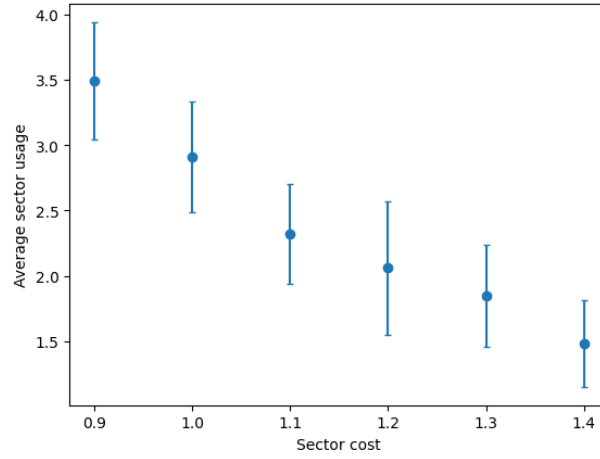
- [124] M. Kulkarni, T. J. L. Forgaard, and K. Alexis, “Aerial gym – isaac gym simulator for aerial robots,” 2023.
- [125] Epic Games, “Unreal engine.” [Online]. Available: <https://www.unrealengine.com>
- [126] DroneCode. (2020) Dji flame wheel 450 with distance sensor and rtk gps (pixhawk 3 pro). [Online]. Available: https://docs.px4.io/v1.9.0/en/frames_multicopter/dji_flamewheel_450.html
- [127] “dwa_local_planner - ROS Wiki.” [Online]. Available: http://wiki.ros.org/dwa_local_planner
- [128] C. Zhang, O. Vinyals, R. Munos, and S. Bengio, “A study on overfitting in deep reinforcement learning,” 2018.
- [129] A. Zhang, N. Ballas, and J. Pineau, “A dissection of overfitting and generalization in continuous reinforcement learning,” 2018.
- [130] S. Mysore, B. Mabsout, R. Mancuso, and K. Saenko, “Honey, i shrunk the actor: A case study on preserving performance with smaller actors in actor-critic rl,” 2021.
- [131] A. Sherstinsky, “Fundamentals of recurrent neural network (RNN) and long short-term memory (LSTM) network,” *Physica D: Nonlinear Phenomena*, vol. 404, p. 132306, mar 2020. [Online]. Available: <https://doi.org/10.1016%2Fj.physd.2019.132306>
- [132] L. Tai, G. Paolo, and M. Liu, “Virtual-to-real deep reinforcement learning: Continuous control of mobile robots for mapless navigation,” in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Vancouver, BC, Canada: IEEE Press, Sep. 2017, pp. 31–36. [Online]. Available: <https://doi.org/10.1109/IROS.2017.8202134>

APPENDIX A NEURAL NETWORK ARCHITECTURES EXPLORED

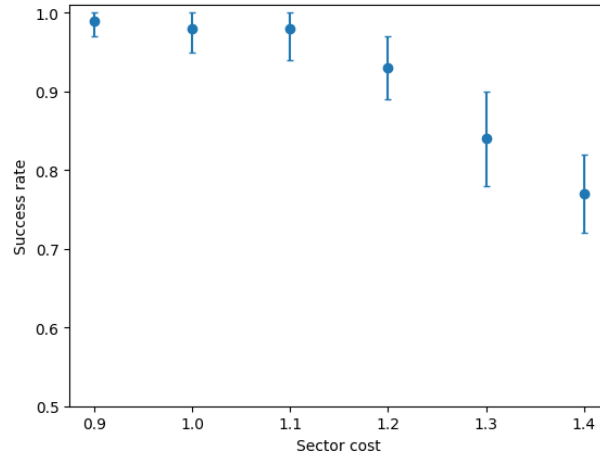
Configuration	Activation function		Network Architectures	Type
1	tanh	Actor	[128, 64, 64, 64]	FCN
		Critic	[256, 128, 64, 128, 128]	
2	tanh	Actor	[64, 32, 32, 32]	FCN
		Critic	[128, 64, 64, 64]	
3	tanh	Actor	[128,128]	FCN
		Critic	[128, 128, 128]	
4	tanh	Actor	[64,64]	FCN
		Critic	[128, 64, 128]	
5	tanh	Actor	[32,32]	FCN
		Critic	[64, 46, 46]	
6	tanh	Actor	[64]	FCN
		Critic	[64, 64, 64]	
7	tanh	Actor	[128, 64, 256 (LSTM), 128, 128]	LSTM
		Critic	[128, 64, 256 (LSTM), 128, 128, 128]	
8	tanh	Actor	[128, 64 (LSTM), 64, 64]	LSTM
		Critic	[128, 64 (LSTM), 128, 128]	
9	tanh	Actor	[64,32 (LSTM), 32]	LSTM
		Critic	[64, 32 (LSTM), 32, 32]	

Table A.1 Different network architectures explored for the agents. Each number in the architectures corresponds to a hidden layer with that many neurons (eg. [32,32] means two layers of 30 neurons). In the LSTM networks, the long short-term memory cells are the ones with the (LSTM) note.

APPENDIX B ABLATION STUDY ON THE COST OF SECTORS IN THE REWARD FUNCTION



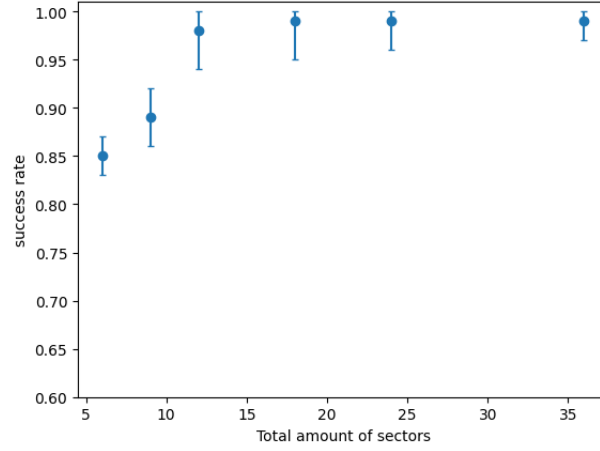
(a) sector usage of the agent for various sector costs



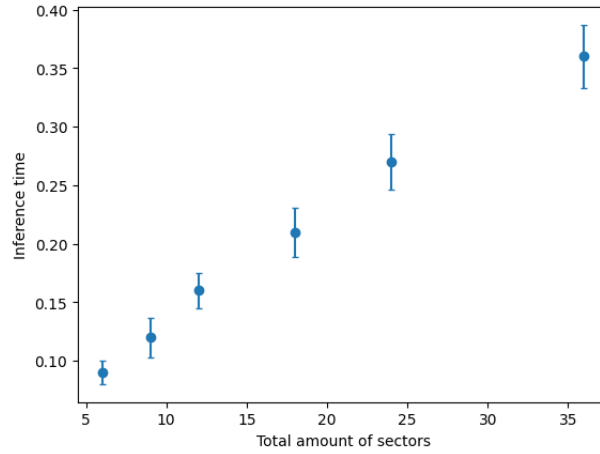
(b) Success rate of the agent for various sector costs

Figure B.1 Trade-off between efficiency and robustness of the agent. By changing the cost of sectors in the reward function, the agent gets more or less picky, changing its average of sectors used but also its success rate.

APPENDIX C ABLATION STUDY ON THE DISCRETIZATION OF THE SURROUNDINGS INTO SECTORS FOR THE VFH METHOD



(a) Success rate of the agent for various amount of sectors



(b) Inference time of the agent for various amount of sectors

Figure C.1 impact of the discretization of the drone's surroundings into n sectors for the VFH method. by increasing the number of sectors, they become more precise (narrow) and the agent gets more robust, at the cost of a higher inference time (bigger networks, observation space, and action space).

APPENDIX D BUG ALGORITHM: THE BOUNDARY IN FRONT OF THE BUG IS NOT ALWAYS THE BEST BOUNDARY TO FOLLOW

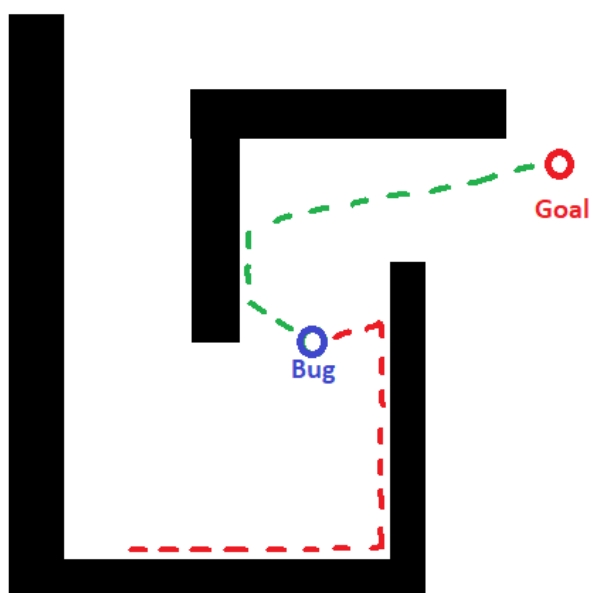


Figure D.1 When switching to boundary following, the Bug algorithm chooses one direction depending on which obstacle is closest. In this example, following the boundary of the obstacle in front of the goal to the right will result in the longest path. Global information is required to pick the right direction.

APPENDIX E COMPARISON OF PPO AND A2C

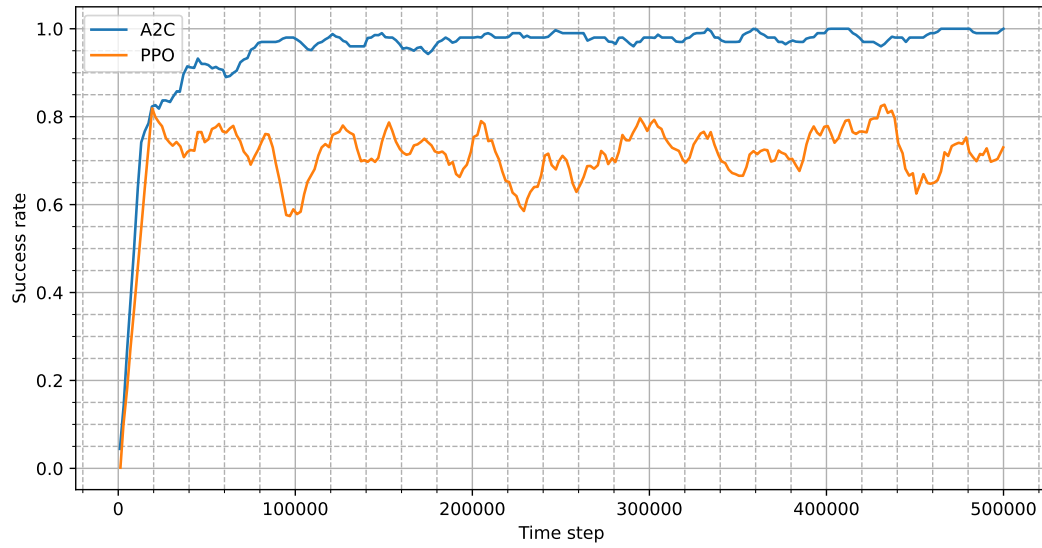


Figure E.1 Comparison of the A2C and PPO algorithms in initial experiments.