Technical University of Crete
School of Electrical and Computer Engineering

Diploma Thesis

Autonomous Drone Navigation using Visual Gate Detection and Reinforcement Learning

Panteleimon Karamailis

Thesis Committee

Professor Michail G. Lagoudakis (School of ECE)
Professor Michalis Zervakis (School of ECE)
Professor Panagiotis Partsinevelos (School of MRE)

Chania, July 2024

Πολυτεχνείο Κρήτης
Σχολή Ηλεκτρολόγων Μηχανικών
και Μηχανικών Υπολογιστών

Διπλωματική Εργασία

Αυτόνομη Πλοήγηση Drone με χρήση Οπτικής Ανίχνευσης Θυρών και Ενισχυτικής Μάθησης

Παντελεήμων Καραμαϊλής

Εξεταστική Επιτροπή

Καθηγητής Μιχαήλ Γ. Λαγουδάκης (Σχολή ΗΜΜΥ)
Καθηγητής Μιχάλης Ζερβάκης (Σχολή ΗΜΜΥ)
Καθηγητής Παναγιώτης Παρτσινέβελος (Σχολή ΜΗΧΟΠ)

Χανιά, Ιούλιος 2024

Acknowledgements

First and foremost, I would like to express my deepest gratitude to my family, who has been my rock throughout this journey. To my parents, Naki and Kwstas, who have instilled in me the importance of hard work and perseverance, your unwavering support has given me the strength to pursue my dreams. Your unconditional love, encouragement, and support—mentally, psychologically, and financially—have been my guiding light, and I am forever grateful for the sacrifices you have made to ensure my success.

To my dear grandmother Theresa, your wisdom and kindness have been an endless source of inspiration. Your stories and experiences have taught me invaluable life lessons, and your warm presence has constantly reminded me that I am never alone in this journey.

To my sister Elli, who has always been by my side, your love, laughter, and understanding have brightened the darkest days. Your belief in me, even when I doubted myself, has driven my determination to succeed.

To my friends, who have supported me in countless ways, I am eternally grateful for your camaraderie and compassion. Your unwavering faith in me and willingness to lend an ear or a shoulder have made all the difference. Your perseverance in the countless hours of reading and providing support, despite the fatigue, has been an inspiration to keep up the hard work.

On the professional side, I would like to express my profound appreciation to the three esteemed professors who have guided me through writing this thesis. To my primary Professor, Michail G. Lagoudakis, your expertise and passion for the subject have shaped my understanding and approach. Your patience and commitment to my growth have not gone unnoticed, and I am truly grateful for your mentorship.

To Professor Panagiotis Partsinevelos, who graciously opened the doors of his laboratory to me, words cannot adequately express my heartfelt gratitude for your immense support and guidance. Your generosity in providing me a nurturing environment to explore and develop my ideas has been truly transformative. Your unwavering belief in my abilities and your tireless dedication to my success has been a constant motivation and inspiration. Your keen insights and constructive feedback have challenged me to think critically and push the boundaries of my research. At the same time, your empathy and understanding have made me feel seen and valued as both a student and an individual. I am deeply honored to have had the opportunity to work under your mentorship, and I am forever indebted to you for the knowledge, skills, and confidence you have instilled in me.

To Professor Michalis Zervakis, your insightful feedback and unwavering support have been invaluable to the completion of this project. Your dedication to your students and willingness to share your knowledge and experience has impacted my academic journey.

In closing, I am forever indebted to all of you—my family, friends, and mentors—for your love, guidance, and support. You have each played an integral role in my accomplishments; I am eternally grateful for that. This thesis reflects my hard work and is a testament to the wonderful people who have believed in me and stood by me every step of the way.

Abstract

In recent decades, drones have gained much recognition for their ability to carry out missions involving areas inaccessible to humans, requiring high costs and long times to approach. Given the ever-increasing complexity of UAV applications, the aim is to minimize their interaction with humans to reduce human error, which consequently may cause material damage and injury. Thus, a particularly desirable direction is the development of fully autonomous navigation systems. Such systems rely on the vehicle's sensors to guide the vehicle in order to reach a target position in unknown and dynamic environments, while avoiding possible collisions. Therefore, novel and innovative approaches are used to develop applications of such complex behavior beyond conventional methodologies. Among these relatively new techniques is Reinforcement Learning (RL), a branch of Machine Learning that has achieved excellent results in many different problems in recent years. This way of learning simulates the learning of living beings, as the agent interacts with its environment and, through constant trial and error, improves its behavior to achieve its goals. Thus, the topic of this diploma thesis concerns the development of a fully operational navigation system for unmanned aircrafts and sets as its ultimate goal the safe navigation through certain gates. The inspiration for this work is the annual AlphaPilot competition of heroX, which invites participants to develop a fully autonomous drone that can beat the best pilot team in a speed race. However, this thesis focuses on sailing the aircraft as safely as possible and not on the speed of flight. Thus, the first and most significant part of this work concerns the autonomous flight of the unmanned aircraft in unknown dynamic environments and the avoidance of possible obstacles using Deep Reinforcement Learning. The proposed approach uses Deep Neural Networks to approximate value functions within RL, addressing the high-dimensional problem that traditionally challenges RL. The second part of this thesis focuses on enhancing an existing optical gate recognition system, operating in real-time through the aircraft's fixed camera. The proposed integrated system, developed and tested in the Gazebo robot simulator using the Robot Operating System (ROS) framework, successfully avoids most obstacles and navigates through gates in numerous randomly-generated instances of variable difficulty, demonstrating significant potential for robust performance.

Περίληψη

Τις τελευταίες δεκαετίες τα μη επανδρωμένα αεροσκάφη τυγχάνουν μεγάλης αναγνώρισης λόγω της ικανότητάς τους να φέρουν εις πέρας αποστολές, οι οποίες λαμβάνουν μέρος σε δυσπρόσιτες για τον άνθρωπο περιοχές, η προσέγγιση των οποίων απαιτεί υψηλό κόστος και εκτενείς χρόνους. Δεδομένης της διαρκώς αυξανόμενης πολυπλοκότητας των εφαρμογών των μη επανδρωμένων αεροσκαφών, επιδιώκεται η όσον το δυνατόν λιγότερη αλληλεπίδρασή τους με τον άνθρωπο, προκειμένου να περιοριστεί το ανθρώπινο λάθος και κατ' επέκταση η πιθανή πρόκληση υλικών ζημιών και τραυματισμών. Έτσι, μία ιδιαίτερα επιθυμητή κατεύθυνση είναι η ανάπτυξη συστημάτων πλήρους αυτόνομης πλοήγησης. Τα συστήματα αυτά, βασίζονται στους αισθητήρες του οχήματος για να το καθοδηγήσουν, ώστε να προσεγγίσει μία θέση στόχου σε άγνωστα και δυναμικά περιβάλλοντα, αποφεύγοντας παράλληλα ενδεχόμενες συγκρούσεις. Ως εκ τούτου, για την ανάπτυξη εφαρμογών τέτοιας περίπλοκης συμπεριφοράς γίνεται επιστράτευση νέων και καινοτόμων προσεγγίσεων που ξεφεύγουν από τις συμβατικές μεθοδολογίες. Ανάμεσα σε αυτές τις σχετικά νέες τεχνικές είναι και η Ενισχυτική Μάθηση (Reinforcement Learning – RL), η οποία αποτελεί κλάδο της Μηχανικής Μάθησης και τα τελευταία χρόνια έχει σημειώσει εξαιρετικά αποτελέσματα σε πολλά διαφορετικά προβλήματα. Αυτός ο τρόπος μάθησης προσομοιώνει τον τρόπο μάθησης των έμβιων όντων, καθώς ο πράκτορας αλληλεπιδρά με το περιβάλλον του και μέσω συνεχών δοκιμών και σφαλμάτων βελτιώνει τη συμπεριφορά του, προκειμένου να πετύχει τους στόχους του. Έτσι, το θέμα της παρούσας διπλωματικής εργασίας αφορά στην ανάπτυξη ενός πλήρους λειτουργικού συστήματος πλοήγησης για μη επανδρωμένα αεροσκάφη και θέτει ως απώτερο στόχο την ασφαλή πλοήγηση μέσω συγκεκριμένων πυλών. Πηγή έμπνευσης αποτελεί ο ετήσιος διαγωνισμός AlphaPilot της heroX, που καλεί τους συμμετέχοντες να αναπτύξουν ένα πλήρως αυτόνομο drone, το οποίο μπορεί να νικήσει την καλύτερη πιλοτική ομάδα σε έναν αγώνα ταχύτητας. Ωστόσο, η διπλωματική εργασία επικεντρώνεται στην όσο το δυνατόν ασφαλέστερη πλοήγηση του σκάφους κι όχι στην ταχύτητα της πτήσης. Έτσι, το πρώτο και μεγαλύτερο μέρος της εργασίας αφορά στην αυτόνομη πτήση τού μη επανδρωμένου αεροσκάφους σε άγνωστα δυναμικά περιβάλλοντα και την αποφυγή πιθανών εμποδίων με χρήση Βαθιάς Ενισχυτικής Μάθησης (Deep Reinforcement Learning). Η προτεινόμενη μέθοδος χρησιμοποιεί Βαθιά Νευρωνικά Δίκτυα (Deep Neural Networks) για την προσεγγιστική αναπαράσταση συναρτήσεων αξίας εντός της Ενισχυτικής Μάθησης, αντιμετωπίζοντας το πρόβλημα υψηλής διαστατικότητας που παραδοσιακά αποτελεί πρόκληση για την Ενισχυτική Μάθηση. Το δεύτερο μέρος αυτής της εργασίας επικεντρώνεται στη βελτίωση ενός υπάρχοντος συστήματος οπτικής αναγνώρισης πυλών, το οποίο λειτουργεί σε πραγματικό χρόνο μέσω της σταθερής κάμερας του αεροσκάφους. Το προτεινόμενο ολοκληρωμένο σύστημα, που αναπτύχθηκε και δοκιμάστηκε στον ρομποτικό προσομοιωτή Gazebo, χρησιμοποιώντας το πλαίσιο Robot Operating System (ROS), αποφεύγει επιτυχώς τα περισσότερα εμπόδια και πλοηγείται μέσα από πύλες σε πολλά τυχαία-δημιουργημένα στιγμιότυπα μεταβλητής δυσκολίας, επιδεικνύοντας σημαντικές δυνατότητες για ισχυρή απόδοση.

Contents

List of Figures

List of Abbreviations

AI . . . . . Artificial Intelligence
ANN . . . . . Artificial Neural Network
CNN . . . . . Convolutional Neural Network
DDQN . . . . Double Deep Q-Network
DP . . . . . Dynamic Programming
DQN . . . . . Deep Q-Network
FG . . . . . FlightGoggles
FoV . . . . . Field of View
GNSS . . . . . Global Navigation Satellite System
GPU . . . . . Graphics Processing Unit
IMU . . . . . Inertial Measurement Unit
JSON . . . . . JavaScript Object Notation
LiDAR . . . . Light Detection And Ranging
ML . . . . . Machine Learning
MC . . . . . Monte Carlo
RGB . . . . . Red, Green, Blue
RL . . . . . Reinforcement Learning
ROS . . . . . Robot Operating System
Rviz . . . . . ROS visualization
SL . . . . . Supervised Learning
SLAM . . . . . Simultaneous Localization and Mapping
TD . . . . . Temporal Difference
UAV . . . . . Unmanned Aerial Vehicle
UL . . . . . Unsupervised Learning
VTOL . . . . . Vertical Take Off and Landing
VFA . . . . . Value Function Approximation
WGS84 . . . . World Geodetic System 1984

1 Introduction

Contents

1.1 Introduction

It is a fact that in recent years the popularity of drones has grown exponentially. Thanks to their technology, equipment, and size, they can perform vital tasks in an increasing number of applications. However, their use remains conservative, and their full capabilities are rarely used in specific scenarios. As a result, one finds that their utilization is not commensurate with their recognition by the public. One of the main reasons for this is that the execution of a more complex mission, requiring faster and more agile flight, requires specially trained pilots. In addition, an imminent collision causes insecurity and leads to conservative use of the aircraft, as it would represent a significant economic loss. Solutions have been proposed and tested in recent years, such as developing semi-autonomous navigation and obstacle avoidance systems. However, these must be established, as currently they could not operate in real conditions.

For this reason, the industry has turned to the search for a more permanent solution, taking advantage of the simultaneous development of artificial intelligence and, in particular, reinforcement learning. The way reinforcement learning creates artificial intelligence is identical to how living beings develop their intelligence, as reward events modify the agent's behavior. Thus, through its sensors and actuators, the agent interacts with its environment through trial and error, typically within a simulation. Then, as feedback for performing its actions, it receives positive or negative rewards, based on which it judges situations and gradually improves on what it does.

Eventually, focusing on our drone navigation problem, after millions of trials, evaluations of its actions, and possible conflicts, the simulated drone becomes capable of avoiding the obstacles it encounters, making its flight safe.

1.2 Thesis Contribution

This thesis presents a new approach to autonomous drone navigation in scenarios of unknown indoor and outdoor environments, which goes beyond the narrow confines of the traditional methods used until recently. One of the most significant challenges in this field is the problem of Simultaneous Localization And Mapping (SLAM). SLAM involves creating a three-dimensional map of the environment and at the same time finding the relative location of the drone within it. The traditional approaches to solving the SLAM problem are computationally heavy and often fail to operate in real time. In contrast, the proposed method requires significantly less computational resources, relying on a Deep Reinforcement Learning (DRL) architecture assisted by optical detection. As part of the approach, the drone participates in three indoor scenarios with several obstacles and a single goal, a gate. The goal is to navigate the drone in each environment until the final goal, avoiding any possible obstacles that may appear. Using deep learning techniques, the agent is in constant trial and error. Thus, without knowing its position and the final goal, it gradually learns through its mistakes to evaluate the situation and choose the best action. The above experiments took place in the Gazebo simulator, while the Robot Operating System (ROS) framework was also used, thus offering activation and interaction of the agent with its environment. As a

learning algorithm, Deep Q-Network (DQN) was used, which stands as a foundational approach in reinforcement learning. At the same time, an effort is being made to enhance the existing visual tracking system with the ultimate goal of achieving even better results in navigation. According to the results of the experimental process, the agent, in the more manageable stages, showcases promising performance, adeptly navigating toward the target while avoiding obstacles. Although the agent's success rate diminishes in more complex environments, the outcome remains satisfactory. Further training of the agent facilitates notable improvements, reinforcing the method's potential. This research is a promising foundation for crafting a navigation system adept at safely steering a drone toward the designated destinations in real-world settings.

1.3 Thesis Outline

The thesis is structured into a total of six chapters. Each chapter is an independent thematic unit that describes each step involved in developing research.

- Chapter 2 contains the essential theoretical background to understand the work better. More specifically, it presents the introductory concepts of Reinforcement learning, the basic mathematical formalism, and the algorithm used to solve the problem. In addition, it includes the tools and frameworks used. Furthermore, it overviews UAVs' basic concepts and technical characteristics.
- Chapter 3 aims to formulate the problem that this thesis is called to solve. Furthermore, it describes the state-of-the-art technology used in obstacle avoidance in UAVs, referring to other traditional approaches and related work.
- Chapter 4 presents the proposed approach as it was developed. The implementation includes the design of the simulated drone model, the proposed agent behavior evaluation environments to address this problem, and the selected Deep Learning Algorithm.
- Chapter 5 describes and analyzes tests performed in simulated environments to evaluate the proposed approach's performance. In addition, it interprets

the results obtained at each stage of the experimental process and explains the advantages and limitations of the methodology.

- Chapter 6 summarizes the present thesis and presents the most important conclusions that emerged. In addition, it offers suggestions for future extensions and ideas for alternative approaches to achieve better results.

2
Background

Contents

This chapter of the thesis provides the necessary theoretical background to understand the contained work. It lays the foundation for the understanding of the concepts, models, and theories that will be used to carry out the research work. The purpose of this chapter is to provide a sound basis for the research work to be conducted and to ensure that the reader has a clear understanding of the concepts and theories that are fundamental to this research.

2.1 Reinforcement Learning

2.1.1 Structural Overview

Artificial intelligence is a field of computer science that has achieved great things for the benefit of humans in recent years. Although there is no exact definition, as several have been given from time to time due to different perspectives, it would be common sense to say that it is intelligence that machines acquire artificially. Artificial intelligence reproduces and imitates human intelligence, thus covering many areas of knowledge related to machine learning.

Machine Learning (ML) is a subset of artificial intelligence and concerns the ability of machines or computers to progressively improve their performance around a given process using given data. In order to achieve a given goal, computers are not always explicitly programmed, i.e., how the final goal is to be achieved may not be specified. Instead, the learning algorithms analyze the data given to them, find patterns and trends in the data, and ultimately create models to predict them. ML is divided into three main categories as shown in Figure 2.1, which are discussed below.

The first of the three categories is Supervised Learning (SL). In this branch, the mathematical model is trained with labeled data, i.e., data that contains the correct prediction answer in advance. Thus, the knowledge provided by the data is gradually generalized and used to make correct predictions on unlabeled data.

In Unsupervised Learning (UL), the datasets provided for training do not carry labels. For this reason, the model tries to discover patterns, hidden structures, and correlations among the data elements to extract the appropriate information and gain knowledge for future predictions.

Reinforcement Learning (RL) is a modern and complex branch of machine learning, distinct from supervised and unsupervised learning. This is a learning technique, in which the agent interacts with its environment, performs actions, and observes their results. In the early stages, it starts by making completely random trials. The agent is rewarded for each desirable action, while the agent receives a penalty for each negative action. Thus, through trial and error, the agent gradually gains experience, learns from it and aims to maximize its total rewards over time.

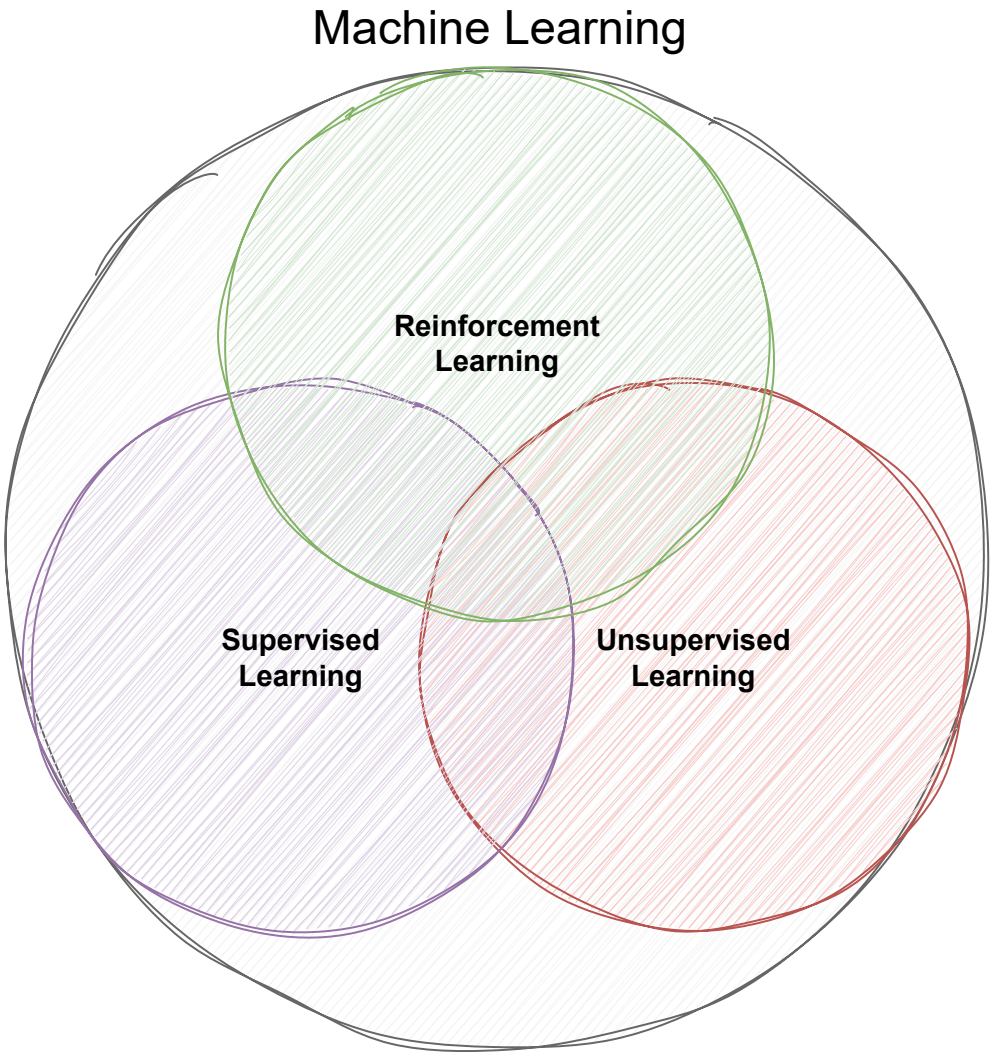

Figure 2.1: Intersection of Machine Learning Methodologies.

2.1.2 Definitions and Terminologies

Agent and Environment

The concepts of agent and environment are perhaps the two most essential elements of Reinforcement Learning.

An agent is an AI algorithm that makes decisions and learns through interacting with the environment. Thus, it is progressively trained to make the right decisions to solve sequential decision-making problems under uncertainty.

The environment is defined as the world, which includes everything outside the agent. The agent is in the environment and constantly interacts with it, performing various actions that respect its physics and dynamics. On the other hand, the environment responds to these actions by presenting the new state that arises after action execution. In addition, the environment returns a numerical value, the so-called reward, whose role is explained in more detail below.

The environments are divided into categories according to some of their characteristics. One important distinction is related to the state space an agent can encounter. An environment is classified as discrete if its state space consists of a finite or countably infinite set of distinct states, indicating each state is separate and countable. On the other hand, an environment is considered continuous if its state space is uncountably infinite, typically representing a continuum of possible states.

Games, such as chess and tic-tac-toe, that take place on grids are characterized as discrete, while autonomous navigation of a UAV takes place in continuous environments.

Finally, another important distinction between environments is between deterministic and stochastic. In a deterministic environment, there is no uncertainty. Therefore, the next state of the environment is determined precisely by taking into account its current state and the agent's action. In contrast, in stochastic environments, the next state cannot be determined with absolute precision due to the randomness involved. The next state's determination rate is directly related to the degree of randomness inherent in the environment and in the execution of some action by the agent.

State

It fully describes the agent's environment at any given time through a set of variables. At each action of the agent, the environment returns a new representation to the agent, i.e., a new state. The variables, their number, and the values they take vary depending on the environment. The agent receives this helpful information and processes it appropriately to decide on its next action.

Observation

This term describes the agent's observation of the state of the environment. The data received by the agent and their possible values are described in a structure called Observation Space. The format varies depending on the environment and the needs of the problem. Thus, it can be anything from a vector to a screenshot of a game screen. The observation is divided into two main categories:

- **Complete Observation**: The decision maker observes all the information on the state of the environment after an action has been performed.
- **Partial Observation**: In this case, the agent indirectly observes the environment and, as a result, does not receive all the information on the state of the environment.

Action

It is any possible action that the agent can perform in the environment. Through actions, the agent interacts with its environment and moves from one state to another.

Model

It is the agent's perspective on the environment. Through it, the agent maps the probability distributions of each state-action pair to the existing states. However, in RL, models are generally not available. When models are used, they typically include the transition model, which describes the probability of transitioning from one state to another given an action, and the reward model, which describes the expected reward for each state-action pair.

Time-Step

To time - step is a specific tick of time. During the interaction of an agent with its environment, each state corresponds to a time-step. The steps required until the agent reaches some terminal state essentially define the duration of an episode.

Episode

An episode is defined as a sequence of situations, actions, and rewards. Each episode consists of several time steps and is completed when the agent reaches a terminal state or the number of time steps defined for that episode is completed.

Task

The term Task is used to describe the instance of a Reinforcement Learning problem. Tasks are divided into two categories.

- **Finite Horizon Task**: A Task is characterized as Finite Horizon, when it has a terminal state and a start point. Thus, an episode is created with a sequence of states, moves and rewards. Furthermore, this implies that the number of states it possesses is finite.

- **Infinite or Unbounded Horizon Task**: In this case there is no terminal state, so the Task continues to run forever unless some external agent decides to terminate its operation. In these Tasks, the agent must gradually learn how to select the best possible moves.

Trajectory

A trajectory is denoted by τ and describes a sequence of states, actions, and rewards of a set of times of a particular episode or a more significant part of the problem. Often the time interval chosen is an entire episode, which is why the episode is often referred to as a Trajectory.

Reward

Reward is defined as the numeric value that the agent receives from the environment after performing an action. The reward acts as feedback for the agent, as it expresses whether the result of each action successfully contributed to the achievement of its final goal. However, this does not imply that the rewards received by an agent are always high. More specifically, rewards can be high or low. High rewards indicate that the action is desirable, because it contributes positively to the completion of the mission. Low rewards reveal that the agent should avoid repeating the action in the future, as it did not contribute significantly to the end goal. The concept of reward is one of the most important in Reinforcement Learning, as it is the only critic the agent has in order to learn and eventually adopt the desired behavior. Therefore, since the reward is the agent's motivation for learning, the goal is to maximize the expected cumulative (aggregate) reward received. Mathematically, the cumulative reward from time step t is the sum of the rewards the agent receives after time step t:

$$G_t = r_{t+1} + r_{t+2} + r_{t+3} + \cdots = \sum_{k=0}^{T} r_{t+k+1} \tag{2.1}$$

However, calculating the cumulative reward in this way is relevant for Episodic Tasks, where the number of interaction steps is finite. In the case of unbounded episode horizons, there is no restriction on the number of interaction steps and hence no bound on the value of the cumulative reward, so that it tends to infinity. In addition, the uncertainty introduced in the environment over time makes immediate rewards more predictable than long-term rewards. These two factors would be particularly destructive to the agent's training and lead to a weak achievement of the desired goal. This problem is solved by a convergent solution, which makes the reward G_t bounded and therefore computable. Thus, the discount factor γ (where $0 \leq \gamma \leq 1$) is used to discount each reward by γ in the exponent of the time step:

$$G_t = \sum_{k=0}^{\infty} \overbrace{\gamma^k}^{\text{discount rate}} \cdot \underbrace{r_{t+k+1}}_{\substack{\text{Rewards received} \\ \text{at each state}}} \tag{2.2}$$

If the factor γ is less than 1, then due to the discount factor applied, more immediate rewards contribute more to the total cumulative reward than future rewards. Indeed, the above equation can be written recursively as follows:

$$\begin{aligned} G_t &= r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \gamma^3 r_{t+4} + \cdots + \gamma^{T-1} r_T \\ G_t &= r_{t+1} + \gamma(r_{t+2} + \gamma r_{t+3} + \gamma^2 r_{t+4} + \cdots + \gamma^{T-2} r_T) \\ G_t &= r_{t+1} + \gamma(G_{t+1}) \end{aligned} \tag{2.3}$$

Policy or Strategy

Policy $\pi(s)$ is the strategy that the agent uses to choose its next action, given its current state. In other words, the policy determines the agent's behavior in the environment. Furthermore, the policy is divided into two types:

- **Deterministic Policy**: The action that the agent will perform in a given state is deterministically-chosen, i.e., it always performs the same action in that state.
- **Stochastic Policy**: A probability distribution over actions for a given state. The agent selects an action according to this distribution.

A policy π_1 is considered better than or equal to a policy π_2 if it leads to better or equivalent outcomes in all states. This concept will be formalized in terms of value functions in the subsequent sections.

Optimal Policy

An optimal policy is defined as a policy that maximizes the total cumulative reward. It is denoted by π^* and is the agent's primary objective. An optimal policy ensures the highest possible reward in every situation the agent encounters. Mathematically, the optimal policy satisfies the condition $\pi^* \geq \pi$ for every other policy π. Furthermore, it is a fact that every problem has at least one deterministic optimal policy, but there can be more optimal policies with the same outcomes. Suppose different optimal policies exist in some time intervals of the same problem. In that case, they can be appropriately combined into one policy, which essentially chooses the policy that offers the highest value for that interval.

A realistic estimate of an optimal policy requires access to the Optimal Action-Value Function Q^* (discussed below), as it contributes to the computation of the optimal policy as follows:

$$\pi^*(a|s) = \begin{cases} 1 & \text{if } a = \arg\max_{a \in A} Q^*(s,a) \\ 0 & \text{otherwise} \end{cases} \tag{2.4}$$

The function essentially states that to find an optimal policy, any action that maximizes its value should be chosen since, as a rule, there is at least one optimal action that gives maximum value. As will be seen later in the chapter, finding the Optimal Value Functions corresponds to finding an agent's optimal policy.

Value Functions and Optimal Value Functions

In the problems that Reinforcement Learning has to deal with, Value Functions are most often used, as they help to find optimal policies for the agent. Generally speaking, these functions express a state or a state-action pair's value for the agent in the long run instead of the immediate reward. Their value is influenced by the agent's environment and the policy (strategy) it follows to achieve its goal. Therefore, the role they play is particularly important, so a specific analysis of their importance is given later in the chapter. There are two types of Value Functions, and each type leads to two equations, the policy-specific form and the optimal-policy form:

- **V-Function or State Value Function**:
  - **On-Policy State Value Function** $V^\pi(s)$: expresses the expected long-term return from the agent's current state s, following a specific policy π. In other words, it expresses the future total return of the agent if it starts from that state and follows a selected policy. Mathematically, it is expressed as:

$$V^\pi(s) = \mathbb{E}_{a \sim \pi}[G_t \mid s_0 = s] \tag{2.5}$$

  - **Optimal State Value Function** $V^*(s)$: similar to the above function except that the policy followed by the agent is an optimal one. That is, it

expresses the maximum return for the states in which the agent will find itself. Mathematically, it is expressed as:

$$V^*(s) = \max_{\pi} \mathbb{E}_{a\sim\pi}[G_t \mid s_0 = s] \tag{2.6}$$

► **Q - Value or Action Value Function**:

- **On-Policy Action-Value Function** $Q^\pi(s, a)$: the value of the function is quite similar to that of the Value Function, except that it takes into account an additional parameter, the current action of the agent. So the function expresses the long-term return from the agent's current state, if it chooses a particular action a in the current state s and uses policy π afterwards. Mathematically, it is expressed as:

$$Q^\pi(s, a) = \mathbb{E}_{\alpha\sim\pi}[G_t \mid s_0 = s, a_0 = a] \tag{2.7}$$

- **Optimal Action-Value Function** $Q^*(s, a)$: similar to the above function except that the policy followed by the agent is an optimal one. It expresses the maximum return the agent will receive, if it starts from the current state s and performs action a in the current step and uses an optimal policy afterwards. Mathematically, it is expressed as:

$$Q^*(s, a) = \max_{\pi} \mathbb{E}_{a\sim\pi}[G_t \mid s_0 = s, a_0 = a] \tag{2.8}$$

At this point, it is worth emphasizing that the Optimal State and Action Value Functions yield the maximum long-term cumulative reward relative to any other Value Function. In other words, they play a crucial role in achieving the agent's ultimate goal, as they contribute to finding an optimal policy. In each problem, Optimal Value Functions are unique, so that, if there is more than one optimal policy, their value functions are shared equally.

Now that all the basic concepts have been defined, the interaction between the agent and the environment can be illustrated. The interaction is depicted in Figure 2.2.

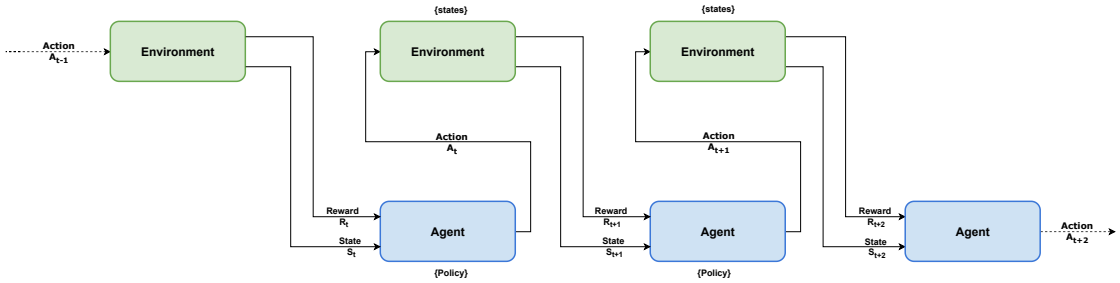

Figure 2.2: Reinforcement Learning Interaction Diagram.

2.1.3 Markov Decision Processes (MDPs)

In Reinforcement Learning, there is often a need to describe decision-making problems mathematically. This is often achieved using a mathematical framework, the so-called Markov Decision Process (MDP). The MDP is a probabilistic model that describes sequential decision problems of an agent interacting with the environment in which it is placed. More specifically, at any given time, the agent is in its current state and takes some available action. Then, depending on the action performed and the currect state, it receives a reward and transitions to the next state. Through this process, it can be seen that only the current state and the chosen action provide enough information to determine the outcome of the action in the environment, i.e., the probability distribution of the various future states. Finally, it should be noted that for the above to hold, Markov's property must be observed. According to it, any state the agent enters after acting depends only on the last state and the action chosen in that state.

The MDP model contains a tuple of five components (S, A, P, R, γ) where:

- **State Space** (S): It is the set of possible states of the environment that the agent may be found while interacting with it. If the environment is fully observable and observations are accurate, the State Space is identical to the Observation Space. This is because the agent observes all the information provided by the state of the environment. Finally, it is worth mentioning that State Space can be encountered either as discrete or continuous.

- **Action Space** (A): It is the set of possible actions that the agent can perform in the various states of the environment. Depending on its environment, each

agent can have either discrete or continuous action space. Each agent's action is a real value in continuous action space. In the case of a discrete action space, the agent is asked to choose an action from a finite set of actions. Due to the above, the discrete set contains actions expressed with less precision and detail. For the same reason, however, the continuous action space is more complex and thus more challenging to manage and exploit by the agent.

- **Probability Transition Model** (P): The state transition model contains the probabilities of the environment, transitioning to a new state, given the current state the agent is in and the action it is about to perform. The state the decision maker will find itself in depends only on the state it was previously in and the action it performed.

- **Reward Model** (R): The reward model represents the reward the agent will receive from the environment, when it transitions from the current state to another after choosing a particular action. The value of the reward received depends only on the current state and the action performed, i.e., they satisfy the Markov property.

- **Reward discount factor** (γ): The value of this parameter determines the importance of the future rewards the agent will receive compared to those received in the current situation. Thus, the rewards received are discounted with time, which allows the agent to perform non-intuitive actions that will yield greater returns in the long term. γ is a real number, taking values between 0 and 1. The smaller the value of the factor, the more the agent places emphasis on the more immediate rewards. Larger values, close to 1, indicate that future rewards are as significant as early ones. The choice of the value of the discount factor depends on the task the agent will perform and how the agent is trained.

Solving a Markov Decision Process aims to maximize the total return of each episode, which makes it not exclusively interested in the immediate reward. Therefore, sometimes the agent chooses to receive a smaller reward at some time step, which eventually leads to a higher final return than it would have received if it had chosen the larger one.

2.1.4 Bellman Equations

Bellman Equations' contribute to the computation of Value Functions and optimal policy. By using them, the agent essentially relates the current state's value to future states without having to obtain all future rewards. To achieve this, the Bellman equations divide the Value Functions into two simpler parts, the immediate reward, and the discounted future ones. Each of these two equations is solved recursively to find their optimal solution quickly.

The Value functions defined above are the State Value Function and the Action Value Function. It is worth noting that both functions use the expectation $\mathbb{E}[.]$, as the environment transition function may be stochastic.

$$V^{\pi}(s) = \mathbb{E}_{\pi}[G_t|S_t = s], \tag{2.9}$$

$$Q^{\pi}(s,a) = \mathbb{E}_{\pi}[G_t|S_t = s, A_t = a]. \tag{2.10}$$

In addition, for the correlation of these two equations, we consider stochastic policies where the sum of the probabilities of all actions $a \in A$ of the agent, given the current state s it is in, is equal to unity:

$$\sum_{a} \pi(a|s) = 1 \tag{2.11}$$

where $\pi(a|s)$ is defined as the probability that some policy π chooses an action a, given a current state s.

For stochastic policies, the State Value Function equals the sum of the probability of choosing each action a in state s multiplied by the Action Value Function of all actions a. That is:

$$V^{\pi}(s) = \sum_{a} \pi(a|s) Q^{\pi}(s,a) \tag{2.12}$$

For deterministic policies, where a specific action a is always chosen in state s, the State Value Function simplifies to:

$$V^{\pi}(s) = Q^{\pi}(s,a) \tag{2.13}$$

Considering the recursive relation of cumulative reward (2.3), then the Bellman Equations for the Value Functions can be defined recursively as follows:

Bellman equation for the State Value function:

$$\begin{aligned} V^{\pi}(s) &= \mathbb{E}^{\pi}[G_t|S_t = s] = \mathbb{E}^{\pi}[R_{t+1} + \gamma G_{t+1}|S_t = s] \\ &= \sum_a \pi(a|s) \sum_{s'} P(s'|s,a)(R(s,a) + \gamma \mathbb{E}^{\pi}[G_{t+1}|S_{t+1} = s']) \\ &= \sum_a \pi(a|s) \sum_{s'} P(s'|s,a)(R(s,a) + \gamma V^{\pi}(s')) \end{aligned} \tag{2.14}$$

Bellman equation for the Action Value function:

$$\begin{aligned} Q^{\pi}(s,a) &= \mathbb{E}^{\pi}[G_t|S_t = s, A_t = a] \\ &= \mathbb{E}^{\pi}[R_{t+1} + \gamma G_{t+1}|S_t = s, A_t = a] \\ &= \sum_{s'} P(s'|s,a)(R(s,a) + \gamma \mathbb{E}^{\pi}[G_{t+1}|S_{t+1} = s', A_{t+1} = a']) \\ &= \sum_{s'} P(s'|s,a) \left(R(s,a) + \gamma \sum_{a'} \mathbb{E}^{\pi}[G_{t+1}|S_{t+1} = s', A_{t+1} = a'] \right) \\ &= \sum_{s'} P(s'|s,a) \left(R(s,a) + \gamma \sum_{a'} \pi(a'|s') Q^{\pi}(s',a') \right) \end{aligned} \tag{2.15}$$

Due to the relationship (2.12), the equation can take the form:

$$Q^{\pi}(s,a) = \sum_{s'} P^a_{ss'}(r(s,a) + \gamma V_{\pi}(s')) \tag{2.16}$$

The two final equations (2.14) and (2.16) make it possible to find the values of a state and a state-action pair, respectively, following a policy π. In addition, the equations are linear systems, allowing their expression as a matrix. Therefore, each equation can be solved in two ways, either as direct or iterative, by taking advantage of the matrix form they can take.

The Bellman equation of optimality

It is clear that Bellman's equations are necessary for finding the values of the Value Functions for any given policy. To find the Value Function of an optimal policy, the Bellman optimality equations can be used, by taking the Bellman equations of the Value Function and replacing policy π with an optimal one. Essentially, for their derivation, the average of the agent's actions is no longer taken, but only the action that offers the maximum value. Thus, the following equations finally emerge:

- **Bellman optimality equation for V^*:**

$$V^*(s) = \max_a \left(\sum_{s'} P(s'|s,a)(R(s,a) + \gamma V_\pi(s')) \right) \tag{2.17}$$

- **Bellman optimality equation for Q^*:**

$$Q^*(s,a) = \sum_{s'} P(s'|s,a) \left(R(s,a) + \gamma \max_{a'} Q_\pi(s',a') \right) \tag{2.18}$$

These Bellman optimality equations form non-linear systems of equations. One common method to solve them is through iterative approaches, such as Value Iteration or Policy Iteration.

Value Iteration involves initializing the value function arbitrarily and then repeatedly updating it using the Bellman optimality equation until the values converge to a fixed point. This method ensures that the values approach the true optimal values.

Policy Iteration alternates between policy evaluation and policy improvement. In policy evaluation, the value function for the current policy is computed, and in policy improvement, the policy is updated to be greedy with respect to the computed value function. This process repeats until the policy converges to the optimal policy.

To extract an optimal policy from an optimal value function, the following rule is applied:

$$\pi^*(s) = \arg\max_a Q^*(s,a) \tag{2.19}$$

This means that, given the optimal action-value function Q^*, the optimal policy π^* at state s is the action a that maximizes $Q^*(s,a)$. Thus, the agent selects the action that provides the highest value according to the optimal value function.

2.1.5 Reinforcement Learning Algorithms Taxonomy

This section summarizes the leading and most popular algorithms of Reinforcement Learning. Each algorithm aims to establish a pattern of behavior that will yield the agent the highest return. This is achieved gradually, through continuous trial and error, so the algorithm can evaluate the actions and come up with the best possible ones based on the reward received and the estimated return. As shown in the Figure 2.3, algorithms are divided into broader categories according to some criteria concerning

their technical characteristics [1]. This division makes selecting an algorithm for a given problem much easier, as it indicates how the algorithm in question handles the problem. Furthermore, it is worth noting that the diagram in the figure does not provide an accurate and complete classification of algorithms, since the representation of algorithms in the form of a tree leads to significant omissions.

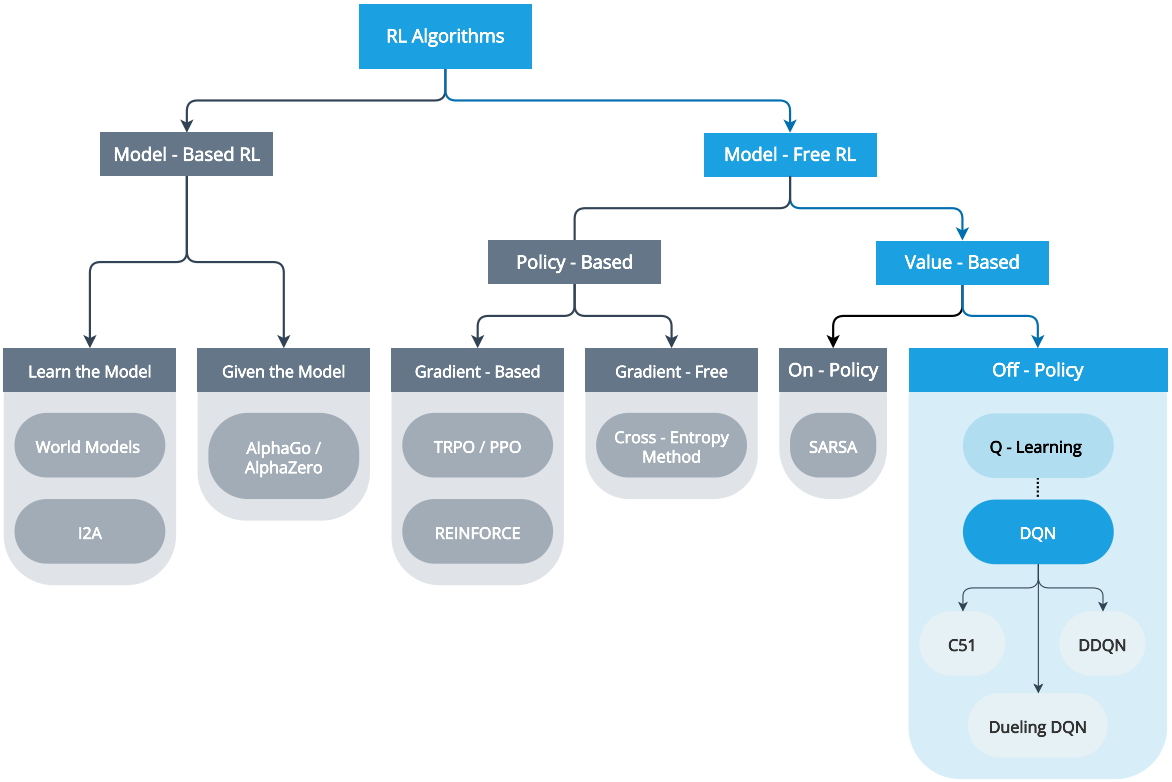

Figure 2.3: Overview of RL algorithm taxonomy, highlighting the DQN method utilized in this research.

Model-based vs Model-free learning algorithms

The first and broadest categorization of algorithms, as shown in the Figure 2.3, is whether they are model-based or not. As discussed, the term model refers to a structure that contains the probabilities of transitioning to a state and the corresponding rewards.

Model-based algorithms branch into two categories, one where the agent builds and learns the model through exploration (e.g., Imagination Augmented Agents or I2A) and one where the model is given (e.g., chess). In the first case, the algorithm models the agent's environment during the learning process. It samples various states the agent

might encounter and observes the actions the agent takes. The algorithm also tracks the rewards the agent receives for these actions. Thus, through the agent's limited interaction with the real environment, a virtual model is built based on the real one. Then, the virtual model is used for the agent to simulate subsequent episodes and plan its next steps, trying to predict the expected future situation and reward given the current situation and action. This approach allows the agent to look ahead, anticipating a set of possible options and choosing the best of them. Moreover, in applications where data and experience collection is demanding and time-consuming, the required learning time is reduced, since the agent interacts with the virtual model rather than the real environment. These algorithms greatly appeal to deterministic problems, where there is full knowledge of the environment and how it responds to the agent's various actions. In contrast, non-deterministic problems, such as real-world scenarios, are challenging to model, as each snapshot of the world requires a new model representation. Another drawback concerns the second case, where the model is given. In this case, if the model is not totally correct, there is a severe risk that the agent exploits the model's bias and performs seemingly well based on the model, but underperforms in the real world.

On the other hand, Model-Free algorithms do not model the environment, nor do they have any transition or reward function to judge the best policy. The optimal policy is judged through experience directly, i.e., the agent interacts with the environment through a trial-and-error approach and thus gradually learns an optimal policy. These algorithms mainly occur in scenarios, where there is little information about the environment and its dynamics constantly changes. Therefore, given that these algorithms perform well in such conditions, it is a fact that they have been extensively tested and show more resonance in real-world problems. As an example, some of the most famous algorithms in this category are Policy gradient and Proximal Policy Optimization (PPO).

The relationship between model-based and model-free approaches is depicted in Figure 2.4. Model-based approaches use the learned or given model to plan and predict future states and rewards, whereas model-free approaches rely on direct experience to learn the optimal policy.

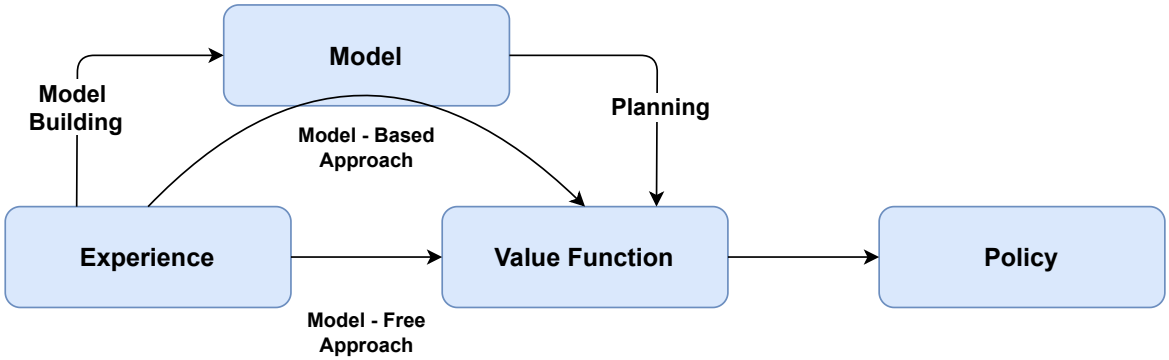

Figure 2.4: Overview of Model-Based and Model-Free Approaches

Value-Based vs Policy-Based Algorithms

The branch of Model-Free algorithms can be further divided into Value-Based and Policy-Based algorithms. As shown below, the basis for this division is the estimation of different elements.

Value-Based algorithms have as their primary objective the optimization of Value Functions, which can guide the selection of the actions that will yield the maximum expected future reward. To do this, the Q-Value Function is selected which is most advantageous for action selection. Then, the selected Value Function is randomly initialized and appropriately approximated by the algorithm, continuously improving its value estimate. After the Value Function is accurately determined, the corresponding policy is easily derived from the greedy actions that the agent will perform in each state based on that Value Function. Thus, it is considered that seeking an optimal policy indirectly requires finding the values of the selected Value Function. It is worth mentioning that the Value-Based approach, although more stable, may not be suitable for use in problems where the action space is continuous, as action selection presents increased complexity. However, in recent years, approaches that discretize the continuous action space have appeared to apply Value-Based algorithms. Finally, two of the most famous algorithms in this category are SARSA and DQN.

In contrast, Policy-Based algorithms optimize a Policy Function without using Value Functions. In this category, a search is done for the policy that maps in each state to a good (or even optimal) action, i.e., an action that, when executed, is expected to yield the maximum reward in the future. Like Value-Based algorithms,

Policy-Based algorithms converge. The policy is redefined at each time step, until this convergence is achieved. The most popular algorithms are the Monte Carlo policy gradient (REINFORCE) and the deterministic policy gradient (DPG). Finally, Policy-Based methods, although theoretically applicable to any RL problem, often require a long training time to reach the desired result, sometimes making their application unrealistic.

On-Policy vs Off-Policy Algorithms

Value-Based algorithms are further categorized into On-Policy, and Off-Policy algorithms. Before analyzing these two approaches, a necessary condition is to clarify that there are two types of policies, Behavior Policy b and Target Policy π. The first describes the agent's behavior when interacting with the environment, i.e., the agent chooses the action based on this. The second policy is the one that the agent tries to learn gradually, learning the Value Function that corresponds to it by extension. Now, having defined the above concepts, it is easier to analyze the two categories of the paragraph. Thus, On-Policy methods seek to evaluate and continuously improve the Behavior Policy the agent already uses. That is, the two policies are identical. This is why this method is used to optimize the value of the explored agent. Some examples of algorithms are Policy Iteration, PPO, and SARSA. On-Policy methods are highly efficient in scenarios where the agent's exploration is directly linked to the improvement of the policy. However, if there is a need for more exploration beyond what the Behavior Policy allows, their performance can be limited due to the lack of flexibility in adjusting the Behavior Policy.

On the other hand, an Off-Policy algorithm does not identify the Target with the Behavior Policy, since it essentially attempts to estimate an optimal policy regardless of the agent's behavior. These algorithms can be highly efficient in real-world scenarios, as the detached Behavior Policy can be made as exploratory as needed without affecting the learning of the Target Policy. One of the most famous examples of an Off-Policy algorithm is the Q-learning algorithm.

Combined Algorithms

Each category of algorithms carries pros and cons, meaning they cannot always be applied. In this context, algorithms have been developed that combine elements from two or more learning approaches. Thus, many of the disadvantages of the algorithms are overcome, making their scope wider. One of the most prominent examples of what is mentioned is the Actor-Critic approach, which combines the Value-Based and Policy-Based approaches.

Exploration - Exploitation Dilemma

In the context of the learning process, choosing the most appropriate approach is demanding and varies depending on the problem. However, equally essential issues arise subsequently that can lead to deadlock and, ultimately, failure. The most common problem is the exploration - exploitation trade-off in scenarios where the agent chooses uncertain payoffs and tries to maximize its expected cumulative reward. Before proceeding to a more extensive analysis of the problem, it is considered appropriate to formulate the concepts of the terms describing the problem:

Exploration: This concept focuses on the long-term benefit of the agent. Through taking random actions, the agent visits unexplored states of the world, collects information, and improves its knowledge to make the best overall decision.

Exploitation: The agent, acting greedily, chooses actions that are expected to yield the greatest reward, based on the current estimated values, which nevertheless rely on the up to now experienced information rather than on true, objective information. As a result, it is possible that the return achieved is less than the maximum possible.

As mentioned above, the agent, in its action, seeks to maximize its expected cumulative reward. However, to achieve this, it is faced with the dilemma of whether to repeat some past action that has yielded a high reward (exploit) or further explore the environment through new actions, hoping to discover rewards of better value (explore). In other words, if the agent, acting greedily, switches to actions that lead to immediate rewards, it will be stuck at some local maximum, since many states and actions that might yield higher rewards will remain hidden. On the other hand,

if it switches to full exploration of the environment, it would require excessive time and information to collect the required data.

This trade-off is visually represented in Figure 2.5. The figure illustrates how the balance between exploration and exploitation changes as the agent gains more understanding of the environment.

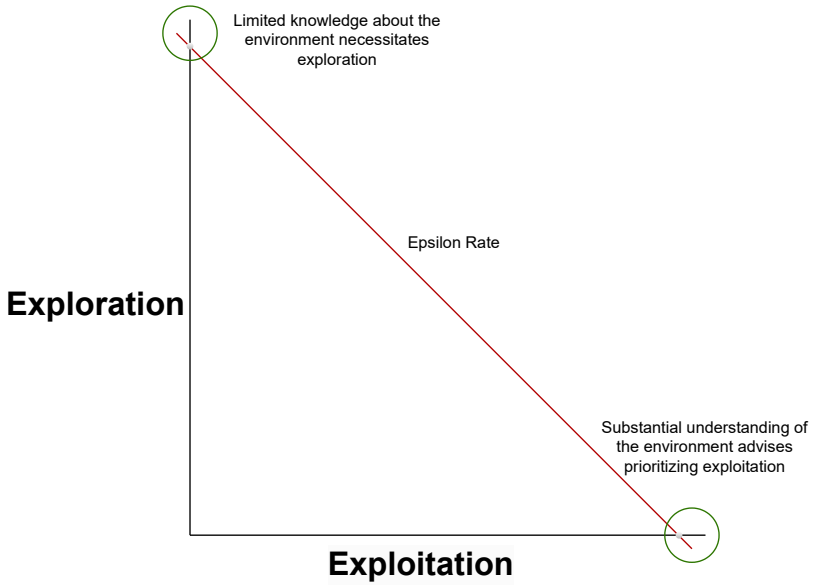

Figure 2.5: Exploration vs Exploitation

It follows from the above that the prevalence of one scenario or the other would lead to failure. Thus, approaches have developed that balance the Exploration-Exploitation Dilemma by encouraging the agent to try a sufficient number of actions and then to prefer the best ones. The main element of these approaches is the Epsilon parameter $\varepsilon \in [0, 1]$, which expresses the probability of taking a random action in an Epsilon-Greedy Policy.

The Epsilon-Greedy Policy is one of the most functional methods. This technique uses the epsilon parameter so that initially, the agent will explore its environment to some extent ($\varepsilon = 1$). Then, having gathered some knowledge about the rewards, Epsilon is reduced to some particular value to set the probabilities of exploration and exploitation. More precisely, the agent $(1 - \epsilon) \times 100\%$ of the time exploits the best option so far, while the remaining ε percent randomly explores for new better options

than the existing ones. After the initial exploration, ε is suggested to take small values so that the agent gives weight to the experience available and explores less frequently.

The decision-making process in an Epsilon-Greedy Strategy is illustrated in Figure 2.6.

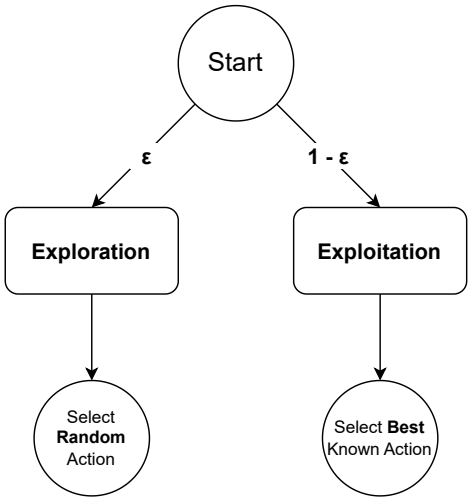

Figure 2.6: Decision-Making Process in Epsilon-Greedy Strategy

Another well-known approach is the Decaying Epsilon-Greedy Policy. Although it is similar to the previous method, the Epsilon gradually decreases over time. Thus, the agent starts its activity by exploring the environment, but then as the parameter value decreases, it limits its exploration and behaves more greedily. What is stated mathematically is expressed through the following relationships:

$$\pi(s) \leftarrow \begin{cases} \arg\max_{a'} Q(s,a') & \text{with probability } 1-\epsilon \\ \text{random action} & \text{with probability } \epsilon \end{cases} \tag{2.20}$$

$$\epsilon = \frac{1}{\text{episode} + 1} \tag{2.21}$$

Temporal Difference Learning

Temporal Difference (TD) Learning belongs to the broader category of Model-Free algorithms and is a combination of Monte Carlo (MC) and Dynamic Programming (DP) methods. It is mainly used in RL, as it solves the problem of predicting rewards without the agent having any prior knowledge of the environment it is situated. In simple terms, this method makes an initial estimate of the Value Function of a policy.

A partial exploration of the environment is carried out, and based on this exploration, it updates the previous estimate. The whole process aims to estimate the total future accumulated reward. This category has a total of three different variations, TD(0), TD(1), and TD(λ), but their further analysis is beyond the scope of this thesis. TD Learning uses the following update equation for the Value Function of a policy:

$$\text{NewEstimate} \leftarrow \text{OldEstimate} + \text{StepSize}\underbrace{(Target - OldEstimate)}_{\text{error}} \tag{2.22}$$

The above equation calculates the difference between the target and the old estimated value. This deviation is known as Temporal Difference, and its decreasing value means the target is getting closer and closer. Finally, the StepSize parameter expresses the learning rate, and estimating its correct value is quite a complex process, as it quickly leads to failures. Moreover, it has been shown that with the successful completion of the learning process and thus the approach to the target, the value of the StepSize should be decreased, since significant corrective moves would cause some possible overshooting of the target.

Q-Learning

Q-Learning is an Off-Policy TD control algorithm widely used for solving RL problems. It is characterized as Off-Policy, because the function is learned independently of the agent's behavior, i.e., the current policy. Using the Q-function, the algorithm seeks to find a good (even optimal) action-selection Policy to maximize the total return. A simple data structure, the so-called Q-Table, is used to make this happen. This table maps each state-action pair to a Q-value, the estimated maximum expected future return. The Q-Values are initialized to zero and then are updated, according to the TD-Update rule, given by Bellman's equation:

$$Q_{\text{new}}(s,a) = Q(s,a) + \alpha\left[R(s,a) + \gamma \max_{a'} Q(s',a') - Q(s,a)\right] \tag{2.23}$$

Thus, as the episodes add up and through the table's incremental update, the Q-Function gradually approaches to the optimal Q-value function Q*, which offers the largest expected return achievable by any policy π for each possible state-action

pair. The most modern examples of Q-Learning methods are DQN and C51 [2]. The first algorithm introduced the concept of Deep RL in predicting the Q-value for each state-action pair. The second is an evolution of the first algorithm and works by predicting a histogram model for the probability distribution of the Q-values.

Algorithm 1: Q-Learning Algorithm

Data: Initial state s, action set A, learning rate α, discount factor γ, exploration rate ϵ

Result: Learned Q-values for each state-action pair

1. Initialize Q-values table, $Q(s, a)$, arbitrarily for all state-action pairs
2. For each episode, do:

begin

3. Initialize the starting state, s

while *episode is not terminated* **do**

4. Choose an action a from the action set A using an ϵ-greedy policy derived from Q
5. Take action a, observe reward r and new state s'
6. Update the Q-value using the equation:

$$Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma \max_{a' \in A} Q(s', a') - Q(s, a)]$$

7. Set $s \leftarrow s'$

end while

end

Value Function Approximation (VFA)

Many algorithms, such as Q-Learning, in simple worlds with few states, use Tabular form (Matrix or Vector) to store the values of Value Functions. In this way, the agent builds a form of memory that stores the value of each possible combination of states and actions to follow trajectories that will yield the maximum return. However, in real-world scenarios, where the state space is continuous and possibly multi-dimensional, using tabular methods is impossible and inefficient, since it requires huge memory space, time, and computational power. This exponential increase in the number of states versus the number of dimensions is known as the curse of dimensionality. Its solution is a significant challenge in the field of Reinforcement Learning. Thus, a new approach was developed, the so-called Value Function Approximation, which groups

the experiences acquired by the agent, generalizing the estimation of value to situations with similar characteristics. This approach seeks an approximation of the value, not the actual value, so convergence to an optimal solution cannot be guaranteed. However, despite any loss of accuracy, ultimately, through its generalizations, it performs faster computations and ensures significantly fewer storage requirements compared to Tabular methods. Several types of function approximators can be divided into linear (e.g., tile coding, polynomials) and non-linear (e.g., neural networks). Of these, neural networks (NNs) are the ones that are now well established and used in most Reinforcement Learning problems. What made them stand out over the other methods was their ability to represent more complex Value Functions with a smaller number of parameters, speeding up agent training and reducing the required memory.

2.1.6 Artificial Neural Networks

Artificial Neural Networks (ANNs), or more simply Neural Networks, is a branch of ML that models how the human brain works. Through their exposure to rich data sets, they gradually generate various identifying features, based on which they then classify and cluster the new data. Moreover, because neural networks have not been pre-programmed to understand the data, they have become flexible enough to adapt quickly to new data sets without further processing. In the early years of their creation, the scientific community did not recognize the value of neural networks. The primary deterrents to their use were the "local minima" problem, insufficient learning data, and the computing power to process it. As a result, RL was quite restrictive, since it used algorithms, such as Q-Learning, which, although quite successful, could only be applied to simple worlds and problems that were far distant from those of the real world. As said above, these algorithms use Tabular Methods, which assume a small number of states and actions for the agent to be able to visit all possible states and calculate the value for each action-state pair, always within the limits of the available memory.

Furthermore, knowledge about similar states is not shared, so at least some cannot be grouped. Thus, up to that point, trying to solve more complicated problems with more complex worlds either led to a dead end or required a turn to Linear Function

Approximators. The latter requires deep knowledge of the domain in question in order to make a correct choice of features. Although neural networks did not initially have a broad appeal, eventually, the stagnation in the RL industry combined with the development of technology played a crucial role in them to prevail. More specifically, after the emergence of powerful graphics processing units (GPUs) and the creation of ImageNet, the first database with vast volumes of labeled images, scientists could now train their models on rich training material faster than before. The above facts encouraged the replacement of Tabular representations with Neural Networks as a function approximation approach when necessary and feasible.

Components of the Neural Network Architecture

Neural Networks are inspired by how neurons in the human brain work, when data arrive through the various senses. In this part, an attempt is made to analyze Neural Networks, explaining the individual elements that make them up.

The fundamental building block of a Neural Network architecture is the neuron. Each Neuron consists of some individual components, which are:

Input: Each Neural Network receives one or more inputs, depending on the problem it is called upon to carry out. As input, it accepts the measures of the features used for learning.

Weight: Through graded multiplications, they determine the importance of the inputs that the Neural Network receives. Thus, depending on the weight of each input, the degree to which it will contribute to learning is decided.

Transfer Function: Combines the multiple inputs received into one output value, usually by summing them up, to enable the Activation Function.

Activation Function: Introduces nonlinearity into the network, allowing for greater complexity in the model. In addition, this function determines what information will be passed to the following layers.

Bias: This is an auxiliary constant that provides the model with flexibility. Thus, it adapts to the unseen inputs it receives and laterally shifts the value resulting from the Activation Function.

The Neural Network that carries only one artificial neuron is called a perceptron, the simplest form of Neural Network. Figure 2.7 below shows that the perceptron receives one or more inputs, and each input is associated with a weight. Additionally, there is a bias term involved in the computation. It then processes the inputs appropriately by multiplying them with their corresponding weights. Finally, the resulting products are added together with the bias and passed through the Activation Function, which produces the final output. Several perceptrons can be connected in a chain form to create a more complex Artificial Neural Network architecture.

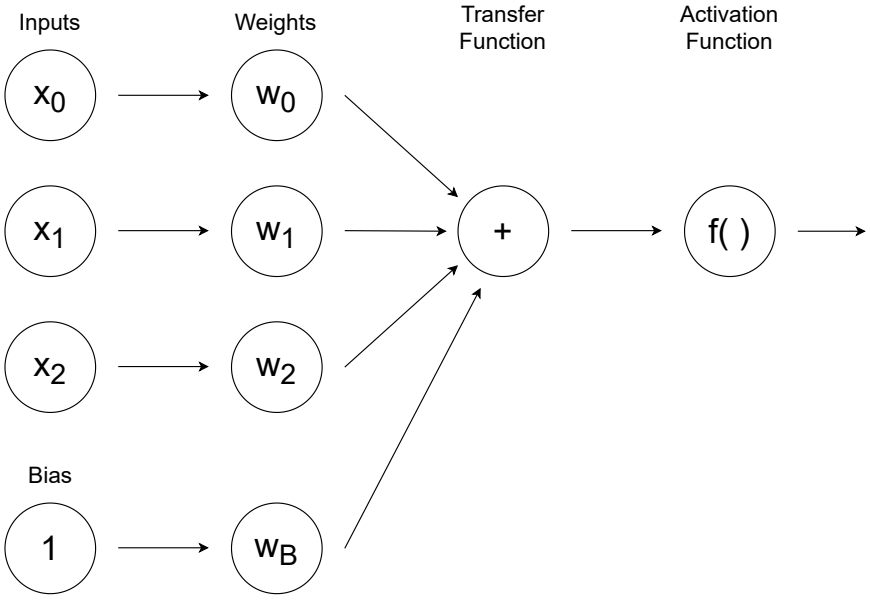

Figure 2.7: The structure of a perceptron

Types of Neural Networks

Neural networks, from time to time, have been classified in different types, based on different criteria regarding their structure, density, layers, and more. Of these, the most prevalent classification criteria are the layers and how they are connected. Each layer consists of groups of neurons located in the same layer. The layers accommodate different ways of connection, suitable for different processes. There are various types of connection of layers, but the four most widespread ones are discussed below.

- **Fully Connected Layers (FC Layers):** Fully connected layers are the most basic type of layer in a neural network. They consist of a set of neurons, each of

which is connected to every neuron in the previous layer and has its weight. The input to a fully connected layer is a flat vector, and the output is a flat vector. Fully connected layers are used to learn complex nonlinear relationships between the input and output of the network. They are typically used as the final layer in a classification network, where the output represents the class probabilities.

- **Convolutional Layer:** Convolutional layers are used in convolutional neural networks (CNNs), designed to process data with a grid-like topology, like an image. Each neuron in a convolutional layer is connected to a small input region, called a receptive field. The weights of these connections are shared across the entire input, forming a convolutional kernel or filter. The input to a convolutional layer is a multi-channel image and the output is a feature map, representing the convolutional kernel's responses at each location in the input. Convolutional layers are used to learn spatial hierarchies of features in the input data, such as edges, corners, and patterns. They are typically followed by pooling layers, which downsample the feature map by taking the maximum or average value over a small region, called a pooling window.

- **Deconvolutional Layer:** Deconvolutional layers are the reverse of convolutional layers and are used for upsampling the input data. They are also known as transposed convolutional layers or fractionally strided convolutional layers. Deconvolutional layers are used in generative models, such as generative adversarial networks (GANs), to generate high-resolution images from low-resolution input. They apply a convolutional kernel to the input and then insert zeros between the output values to increase the resolution. Deconvolutional layers can also learn to restore lost information in the input, such as in image denoising or inpainting.

- **Recurrent Layer:** Recurrent layers are used in recurrent neural networks (RNNs) to process sequential data, such as time series or natural language. Each neuron in a recurrent layer has a memory or state updated at each time step, allowing the network to capture temporal dependencies in the data. The input to a recurrent layer is a sequence of vectors and the output is also a sequence

of vectors. Recurrent layers can be unidirectional, where the output at each time step depends only on the input at that time step and the previous state, or bidirectional, where the output at each time step depends on the input at that time step and the previous and future states. Recurrent layers are used to learn long-term dependencies and patterns in sequential data. They are typically used as the first layer in a sequence modeling network, where the output at each time step represents a prediction for the next time step.

In summary, understanding the different types of layers and how they work is essential for designing and building effective neural networks. The right choice of layers can significantly impact the performance and efficiency of a neural network, so it is essential to carefully consider the characteristics of the data and the desired output, when selecting the layers for a specific task. By selecting the appropriate combination of layers, it is possible to build a neural network that can learn effectively from the input data and produce accurate and valuable output.

2.1.7 Deep Reinforcement Learning

Reinforcement learning (RL) involves training agents to make a series of decisions in an environment to maximize their reward. Agents receive rewards based on their actions and use this feedback to update their decision-making process. To achieve this, Q-learning and other algorithms often use tabular techniques, common approaches in small state-space environments. However, as environments and tasks become more complex, traditional RL methods are limited by their inability to generalize past experiences and identify patterns in data. In particular, it is difficult to scale to high-dimensional state spaces with many possible states, making it challenging to learn effective policies in these environments. Researchers have developed alternative methods to address this issue, such as function approximation and Deep Learning. Deep Reinforcement Learning (DRL) combines the representation learning capabilities of deep neural networks with RL, allowing agents to use the powerful representation learning capabilities of deep neural networks to learn more complex policies. The main difference between traditional RL and DRL is using function approximations to

represent values or policy functions. In traditional RL, agents typically represent their state space using a table or a few hand-designed features, but the need to discretize the state space can lead to a loss of information. In contrast, DRL uses deep neural networks as function approximators. This allows the agent to learn a continuous representation of the state space, allowing it to handle high-dimensional state spaces more effectively. Overall, the move from traditional RL to DRL was driven by the need to handle more complex environments and tasks. DRL has proven to be a powerful tool for solving many complex problems, such as control tasks, games, and natural language processing.

Deep Q-Network (DQN) algorithm

Methodology

Deep Q-Networks (DQN) are a class of reinforcement learning algorithms that use neural networks to approximate the optimal action-value function. DQN aims to learn a policy that maximizes the cumulative reward over time, given a set of states and actions. The algorithm was introduced in a 2013 paper by Google DeepMind [3], and has since been widely used and extended to solve various tasks in different domains.

One of the main advantages of the DQN is its ability to learn directly from high-dimensional sensory inputs, such as raw pixel values from a video game screen. This is made possible using a deep CNN as the function approximator for the Q-function, which automatically allows the DQN to learn features from the raw sensory inputs. It is beneficial in tasks where the features take time to be apparent or are too complex to design manually.

Architecture

The architecture of the DQN algorithm consists of three main components: a neural network, a replay memory, and a target network.

The neural network, also known as the online network, estimates the Q-function for a given state-action pair. It consists of an input layer that receives the state as input, one or more hidden layers, and an output layer that produces the Q-values for each action. The hidden layers can be any neural network layer, such as fully connected, convolutional, or recurrent, depending on the nature of the input state. The output

layer is a fully connected layer with one unit per action. The Q-value for each action is obtained by forward propagating the input state through the network, and each output unit provides the Q-value for a respective action.

The replay memory is a data structure that stores a fixed-size buffer of past transitions or experiences in the form of tuples $(s, a, r, s', \text{done})$, where s and s' are the states, a is the action, r is the reward, and done is a boolean indicating whether the episode has terminated. The purpose of the replay memory is to store and decouple the experiences, making the learning process more stable by sampling from a diverse set of transitions.

The target network is a copy of the online network used to compute the target Q - values for training the online network. The target network is updated periodically to the weights of the online network. This update is performed using a soft update rule, which allows the target network to converge slowly to the online network and avoids oscillations in the learning process.

Training procedure

The training procedure of the DQN algorithm can be divided into two main phases: exploration and exploitation. During the exploration phase, the agent takes random actions to explore the environment and collect experiences in the form of transitions, or tuples $(s, a, r, s', \text{done})$. These transitions are stored in the replay memory. By sampling from this memory, the learning process is stabilized and the updates' variance is reduced.

During the exploitation phase, the agent uses the online network to select the best action for each state based on the predicted Q-values. To improve the accuracy of the Q-values, the online network is trained using a batch of transitions sampled from the replay memory. Specifically, the target Q-values for each transition are computed using the target network, which is a periodically updated copy of the online network. The target Q-values are compared to the predicted Q-values produced by the online network using a suitable loss function, such as the mean squared error (MSE). The loss is then backpropagated through the online network to update the weights and improve the accuracy of the Q-values.

The exploration-exploitation trade-off is achieved using an epsilon-greedy policy, which determines the probability of selecting a random action versus the best action based on the predicted Q-values. The probability of selecting a random action, epsilon, gradually decreases during training. So, the agent initially explores the environment extensively and then exploits the learned policy more and more, as it becomes more confident in its predictions.

In addition to the exploration-exploitation trade-off, the DQN algorithm uses several techniques to stabilize the learning process and accelerate convergence. One such technique is using a target network, which is updated slowly to the weights of the online network using a soft update rule, such as an exponentially moving average (EMA). This technique helps to reduce the oscillations in the Q-values and to smooth the learning process. Another technique uses a replay memory, which stores a fixed-size buffer of past transitions and random samples to break the correlations between the experiences. This technique helps stabilize the learning process and reduce the updates' variance.

The training procedure is repeated for a fixed number of iterations or until a certain performance threshold is reached. During training, the online network is updated with the latest experiences, and the target network is updated periodically to stabilize the learning process.

In summary, the training procedure of the DQN algorithm is shown in Algorithm 2:

Algorithm 2: Deep Q-Network (DQN) Algorithm

Data: Initial state s, replay memory with capacity M, batch size N, discount factor γ, online network Q, target network Q'

Result: Trained online network

1. Initialize the online network Q with random weights
2. Initialize the target network Q' with the weights of the online network
3. Initialize the replay memory to capacity M
4. Initialize the state s and select initial action a using an epsilon-greedy policy based on Q

while *training is not converged* **do**

5. Execute action a and observe reward r and next state s'
6. Store the transition tuple (s, a, r, s') in the replay memory
7. Sample a mini-batch of N transitions from the replay memory
8. For each transition tuple (s_i, a_i, r_i, s'_i) in the mini-batch, compute the target Q-value, y_i using the equation:

$$y_i = \begin{cases} r_i + \gamma \max_{a'} Q'(s'_i, a') & \text{if episode is not terminated at step } i \\ r_i & \text{if episode is terminated at step } i \end{cases}$$

9. Compute the loss using the mean squared error loss function:

$$\text{Loss} = \frac{1}{N} \sum_i \left(y_i - Q(s_i, a_i)\right)^2$$

10. Perform a gradient descent step to minimize the loss and update the weights of Q
11. Every C steps, update the weights of Q' to the weights of Q using soft update rule
12. **if** *the episode has terminated* **then**
    Reset the state s to a starting state for a new episode
    else
    Update the state s to s' and select a new action a using epsilon-greedy policy based on Q
    end if

end while

Hyperparameters

The DQN algorithm has several hyperparameters that affect its performance, such as the learning rate, the batch size, and the update frequency of the target network.

The learning rate determines the step size at which the weights of the online network are updated during training. A larger learning rate can lead to faster convergence

but also instability, while a lower learning rate can lead to slower convergence and more stable learning. The learning rate is usually set using a schedule, such as a constant or an exponentially decaying schedule.

The batch size determines the number of transitions sampled from the replay memory and is used to compute the loss and update the weights of the online network. The batch size is usually set to a value between 32 and 1024, depending on the size of the replay memory and the computational resources available. A larger batch size can lead to more stable learning but also slower convergence, while a smaller batch size can lead to faster convergence and more noisy learning.

The update frequency of the target network determines the rate at which the target network is updated to the weights of the online network. The update frequency is usually set to a value between 1,000 and 100,000, depending on the complexity of the task and the learning rate. A higher update frequency can lead to faster convergence but may result in more unstable learning, while a lower update frequency can lead to slower convergence, but more stable learning.

In addition to these hyperparameters, the DQN algorithm also relies on several implementation details, such as the choice of the neural network architecture, the choice of the optimizer, and the choice of the loss function. These details can affect the performance of the DQN algorithm and should be carefully considered, when applying it to a specific task.

The DQN algorithm is a powerful method for learning policies from raw sensory inputs in complex environments using deep neural networks. Its architecture consists of an online network, a replay memory, and a target network. Its training procedure involves sampling transitions from the replay memory, computing the loss between the predicted and target Q-values, and updating the weights of the online network using back-propagation.

Limitations and Extensions

Despite its success, the DQN algorithm has several limitations that can make it challenging to apply to specific tasks. One limitation is the need for many samples

to learn effective policies, which can be resource-intensive or time-consuming in some domains. Additionally, like many RL methods, DQN assumes a stationary environment, which may not hold in real-world settings.

Several extensions of the DQN algorithm have been proposed to address these limitations. One such extension is the Double DQN algorithm [4], which decouples the action selected from the action evaluation to reduce the overestimation bias of the Q-function. Another extension is the Dueling DQN algorithm [5], which decomposes the Q-function into a value function and an advantage function to improve learning efficiency.

Other extensions of the DQN algorithm include the Prioritized Experience Replay [6], the Noisy DQN [7], and the Rainbow DQN [**hessel2018**], which combine multiple extensions with improving the performance of the algorithm further.

2.2 Tools

In the rapidly-evolving landscape of autonomous drone navigation, the choice of tools stands central to innovative breakthroughs and the successful implementation of sophisticated systems. Tools serve as the linchpins, orchestrating the harmonious functioning of various elements involved in autonomous navigation through visual gate detection and reinforcement learning. This section highlights the tools handpicked for this endeavor, each for its remarkable capabilities and contributions to robotics programming. From offering foundational frameworks, like the Robot Operating System (ROS), to visualization tools, such as Rviz, we will embark on a detailed exploration of these tools, unfurling their vital roles and functionalities in the grand scheme of the project.

2.2.1 Robot Operating System (ROS)

The Robot Operating System (ROS) [8] is becoming the standard in robotics programming. ROS is a flexible framework that provides a set of tools, libraries, and conventions. ROS facilitates message passing between nodes, making it easier to manage hardware-software interactions. It enables executables to be individually designed and easily

coupled at runtime, thus achieving robotics software collaboration. To achieve this, ROS provides a network with a central hub (Master node) and a set of processes (nodes).

- **Nodes**

  Nodes are executable programs that perform some computation or task, organized in ROS packages in every robotic application. In addition, they are combined into a graph and communicate with other nodes through ROS communication tools (topics, services, actions). Thus, nodes use these tools to send or receive specified message types.

- **Messages**

  Messages are simple or complex data structures that contain helpful information, such as sensor measurements. The type and the data format define each ROS message. So, messages are an essential part of robotics applications, as they state what information all nodes need to produce to communicate with each other.

- **Topics**

  As mentioned above, nodes use communication tools to exchange messages with each other. A way to make this happen is through topics that implement a publish/subscribe communication mechanism. Thus, topics behave as buses, where each node, depending on its function, can publish or receive data. Finally, topics are designed to support unidirectional, streaming communication, allowing one node to continuously update others with new information.

- **Services**

  Services are an alternative way of communication between nodes. Services are synchronous and bi-directional remote procedure calls that allow each node to send a request and receive a response. Furthermore, the term “synchronous” means that the client sends a request and blocks until a response is received. Finally, a service can be addressed to many clients and is used only for simple computations and quick actions.

- **Actions**

  Actions are more complex, as they are implemented internally using topics; nevertheless, their architecture resembles services. So, similar to the request and response of a service, the client can send a request that takes a long time, but can cancel it anytime. This can be accomplished, as actions are asynchronous, resulting in the client asynchronously monitoring the server's state.

2.2.2 ROS Visualization (Rviz)

The ROS visualization or Rviz provides various tools that support introspecting, debugging, plotting, and visualizing the state of the system being developed. The most well-known tool, but also the most used in this research, is Rviz. Rviz is a 3D visualization interface that allows visualizing a lot of information, such as captured data from the robot sensors. Using ROS in conjunction with the Gazebo Simulator and Rviz can make troubleshooting more manageable, such as spotting misinterpretation of sensor data or robot model inaccuracies. The first shows what the robot thinks is happening, while the second shows what is actually happening; therefore, comparing their results, some potential mismatching might be found.

2.2.3 OpenAI Gym

OpenAI Gym is a popular library for writing, comparing, and testing reinforcement learning algorithms. Gym includes various simulated environments of different types and difficulties, where RL agents are trained and tested. However, it is not limited to this, as it allows the creation of a custom environment for a specific task. Thus, it is understood that in projects of increased difficulty, the use of Gym is quite helpful, as the modular structure it provides, in combination with the creation of specific environments, can contribute to creating a powerful agent.

2.2.4 Blender

Blender is an open-source platform that specializes in 3D computer graphics creation. This tool set supports almost every aspect of 3D development, with 3D modeling and

sculpting tools being its most powerful and valuable features. Depending on how the tool is used, it can be applied from a simple task to a very complex one. As a result, its services are used by both novice users and companies and associations of global renown, such as NASA, which has built important models occasionally.

2.3 Frameworks

This section meticulously unravels how the frameworks, each with unique attributes, came together to forge the backbone of the Deep Q-network (DQN) algorithm central to this study. It explains the collaborative relationship between TensorFlow and Keras, showcasing how they work together to streamline the development process. As one navigates through this section, the indispensable role of these frameworks in piloting the project to its goal becomes progressively evident.

2.3.1 Tensorflow

Tensorflow is an open-source artificial intelligence library that is used for creating Machine Learning applications. Its popularity is based on its ability to simplify the process of building and training models. In addition, it provides more flexibility and network control, as it is a low-level library.

2.3.2 Keras

Keras is a high-level, deep-learning API designed to operate on top of other open-source machine-learning libraries, such as TensorFlow. Keras stands out for its ease of use and flexibility compared to TensorFlow, facilitating the straightforward implementation of basic neural networks. Leveraging the strengths of both TensorFlow and Keras, the DQN algorithm was developed and applied in the present study, forming the bedrock of the machine learning component of this research.

2.4 Simulators

The construction of UAVs and robotic systems, is getting more and more complex. So, it is pretty hard to develop such systems, often carrying quite expensive equipment,

which can be easily damaged or destroyed. Therefore, the need to use a simulator is commonly accepted, where robots and algorithms can be tested and evaluated easier and faster without material damage. Thus, many simulators have been developed, representing the real world and the prevailing conditions fairly well.

2.4.1 Gazebo Simulator

Gazebo simulator is a well-designed 3D robot simulator that can accurately and efficiently simulate every type of robot. What makes Gazebo unique is that it provides a complete world with built-in dynamic and kinematic physics. Additionally, integration between ROS and Gazebo is provided. Thus, the robots' data and sensors can be appropriately processed, as the nodes created are perfectly compatible with the simulator. With Gazebo, therefore, it is possible to test both robotic applications and algorithms and train AI systems using realistic scenarios. After completing the process and the desired results for the robot's behavior in a simulated environment, the robot's brain can easily be transferred to the physical robot with just a few changes. An example of a simulation environment in the Gazebo simulator is shown in Figure 2.8.

Figure 2.8: A Simulation Environment in the Gazebo Simulator [9].

2.4.2 FlightGoggles Simulator

FlightGoggles Simulator is a modern simulator created by the Massachusetts Institute of Technology (MIT) for perception-driven robotic vehicles. What sets FlightGoggles apart from the rest of the simulators is that it combines real physics and data-driven exteroceptive sensors. In addition, it uses graphics assets generated with photogrammetry to provide photorealistic simulation. Thus, the simulator renders photorealistic camera streams, while vehicles experience real dynamics, inertial sensing, and human behavior. Finally, it is worth noting that just like Gazebo, FlightGoggles works perfectly with ROS for the flexibility of any robotic system being developed. Figure 2.9 depicts the FlightGoggles simulator environment.

Figure 2.9: The FlightGoggles Simulator Environment

2.5 Sensors

For the development of a visual gate detection and collision avoidance system, the perception of the environment by the drone itself is necessary. Considering that the drone is equipped with one or more sensors working as a unified world perception system, their use varies according to the needs. However, it is well-known that existing

measurement and sensor technologies fail to provide optimal outcomes; as a result, they often do not meet the industry's needs, requirements, and expectations. A typical example of this is the ultrasound sensors (sonars), which provide limited distance information with just one measurement. In the same context, higher-definition Lidars have high purchase costs, are bulky, and are relatively easily damaged in case of collision. In addition, the camera technology, although quite promising, is still at an early stage, and its use is relatively complicated. Therefore, to provide complete protection of the unmanned vehicle from impending collisions and the detection of gates, the combination of multiple sensors is required to achieve a balance between the required cost and the information needed from the environment.

The sensors carried by a drone are divided into two broad categories, passive and active sensors [10], as shown in Figure 2.10.

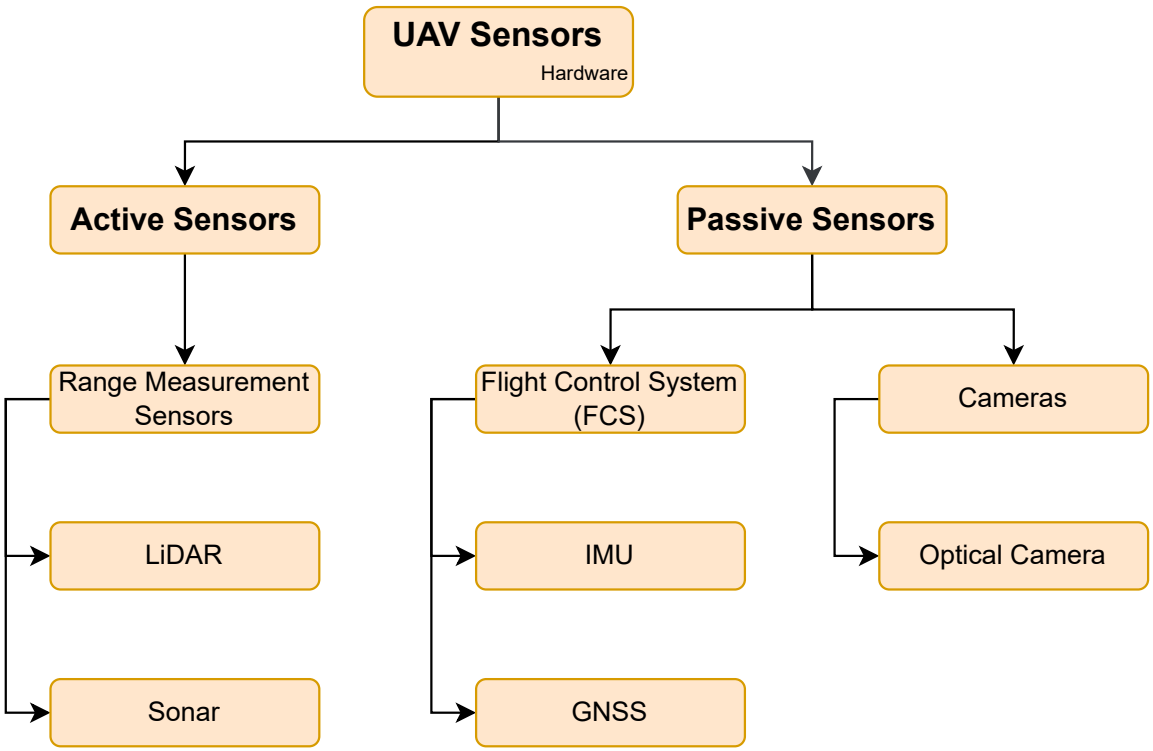

Figure 2.10: UAV Sensing Architecture: Active and Passive Systems

2.5.1 Active Sensors

Active sensors both emit a pulse of energy and detect the reflected energy, and hence they have their transmitter and detector. Thus, the sensor's transmitter emits a signal in the form of, e.g. a light wave, an electrical signal, or an acoustic signal, which then bounces off an object, and the receiver reads the reflected signal.

Light Detection and Ranging (LiDAR)

The LiDAR Sensor is arguably one of the most common distance measuring sensors, known as Light Detection And Ranging. It is a remote sensing method that measures the exact distance of an object using light in the form of a pulsed laser. More specifically, the sensor emits pulsed light waves into the surrounding environment; the pulses bounce off surrounding objects and return to the sensor. Thus, depending on the travel time it took for each pulse to return to the sensor, the distance of each object around it is calculated. This process is repeated millions of times every second, resulting in 3D models and maps of objects and environments. In recent years, the use of the LiDAR has become necessary in cases involving obstacle detection, avoidance, and safe navigation through various environments, and this is why it is used in the majority of autonomous vehicles.

Sound navigation and ranging (Sonar)

The Sonar, which stands for Sound navigation and ranging, is also included in the distance measuring sensors. Ultrasonic sensors use high-frequency sound waves as emitted signals to detect objects in the environment. Thus, if an object is in the sensor range, the sound wave is reflected and, based on the time it takes to return and the speed of sound, the distance from the source is calculated. Such sensors were primarily used in oceans and seawater for exploration and mapping. However, in recent years, they have become increasingly popular in autonomous vehicles.

Comparing LiDAR and Sonar

Table 2.1 shows where each measurement sensor is superior:

Quality/ Device	LiDAR	Sonar
Accuracy	×	
Cost		×
Size		×
Data Update Speed	×	
Range (min)	0.3 meters	0.03 meters
Range (max)	30 meters	5 meters

Table 2.1: Comparative Analysis of LiDAR and Sonar Characteristics

LiDAR sensors are incredibly accurate, and their fast update speed allows them to detect fast-moving objects. Nevertheless, their high cost limits their frequent use. Hence, Ultrasonic sensors are used complementary, as it is a cost-effective solution, but with less sensing accuracy.

2.5.2 Passive Sensors

Passive sensors are microwave instruments that detect the energy discharged by the objects of the scenery under observation. To achieve this, they include radiometers and spectrometers, the first being used to measure the radiant flux of electromagnetic radiation and the latter to see spectral lines and measure their wavelength and intensity. The most well-known passive sensors employed in sensing applications are optical or visual cameras and thermal or infrared (IR) cameras. These different types of cameras operate in the electromagnetic spectrum's visible, infrared, thermal, and microwave segments.

Optical Camera

While newer imaging technologies have emerged, optical cameras continue to dominate, as they enable varied applications from real-time guidance, navigation, and control to machine learning and object detection. In addition, all cameras rely on heavy image processing to extract useful information, but are highly preferred as they are easily affordable. Thus, most drones today are equipped with one or more optical cameras. The camera's operation is relatively simple, since the camera lens absorbs all the light

rays and, through a glass, directs them to a focal point. Afterward, the camera uses electronic sensors on the back to capture light and create a sharp image.

Global Navigation Satellite System (GNSS)

Global Navigation Satellite System (GNSS) refers to a network of satellites providing timing and orbital information to GNSS receivers. Then these receivers use the data for navigation and positioning measurements, when operating in outdoor environments. In recent years, the GNSS device has been necessary for the safe and reliable navigation of drones. Furthermore, the sensor is often connected to other sensors, such as the camera, and geotags are generated with 1 cm accuracy for each image.

Internal Measuring Unit (IMU)

An Inertial Measurement Unit (IMU) is a sensor device that provides data used in Activity Recognition problems or as components of navigational equipment, such as GPS positioning systems, in manned and unmanned aircrafts. An IMU combines input from several different sensor types in order to output movement accurately. These sensors are gyroscopes, accelerometers, and more rarely, depending on the requirements, magnetometers and barometers. An accelerometer is a sensor that measures inertial acceleration, which is the rate of change of velocity. The gyroscope measures angular velocity around three axes: pitch pitch (x-axis), roll (y-axis), and yaw (z-axis). A magnetometer device provides a measurement of the magnetic field surrounding each system. Finally, the barometer is a sensor that measures air pressure and can provide altitude. The main disadvantage of the IMU is known as "drift" and defines the accumulated error over time. Hence, many applications use additional sensors, such as GNSS receivers and cameras, constantly correcting accumulated errors.

Infrared beacon sensor

The infrared beacon sensor, functioning as an infrared receiver, is used alongside one or more beacons. It detects infrared light emitted by these beacons, enabling the calculation of their direction and distance. This feature classifies it as a passive sensor, since it does not emit infrared light, but only receives it. Such sensors are extensively

used in autonomous robots, often in pairs to detect each other, and are increasingly employed in drones for high-precision landings. Although primarily passive, infrared technology can also be adapted for active sensing in different applications.

Table 2.2 summarizes the sensor specifications and their operational parameters, including weather dependency and processing requirements.

Sensor	Weather Condition	Processing Requirement
Optical Camera	High Dependency	High
GNSS	Partial Dependency	Low
IMU	Not Dependent	Low
Sonar	Low Dependency	Low
LiDAR	Low Dependency	Low

Table 2.2: Sensor Specifications and Operational Parameters

2.5.3 Unmanned Aerial Vehicle (UAV)

Introducing quadrotors

The acronym UAV stands for Unmanned Aerial Vehicle, which refers to a class of aircraft with no human on board guided remotely or autonomously. Quadrotors are one of the best-known and most widely used models, often called quadcopters. As its name implies, quadcopters have four motors and propellers whose size and formation vary. The motors are divided into pairs; in each pair, one rotates clockwise, while the other rotates counterclockwise. This family of UAVs is famous, as they were the first successful vertical take-off and landing (VTOL) vehicles, simulating how a typical helicopter flies. Furthermore, they have a relatively low cost of construction compared to other aerial craft, without considering the various sensors they can carry. Finally, Quadrotors offer electronic stabilization during the flight, making it easier to fly indoors and outdoors.

Defining roll, pitch, and yaw

UAVs that fly in the air have more difficult control, less stability, and more complex movement than those vehicles that move on the ground or on the sea. The difference is that the aerial vehicles move freely in three dimensions, while the others in two.

The vehicle's movements in the three dimensions are described by pitch, roll, and yaw, which are rotational movements around the x (Longitudinal), y (Lateral), and z (Vertical) axes, respectively. Understanding these concepts is especially important, as this is the only way to understand the basic principles of aircraft flight.

Pitch: The pitch movement is a rotation that attempts to move the nose of the aircraft either upwards or downwards.

Yaw: The yaw expresses the system's rotation about the z-axis, i.e., it indicates whether the aircraft's nose turned right or left. Therefore, if we assume that the aircraft remains stable at the same level, a change in yaw implies that it rotates around either clockwise or counterclockwise.

Roll: The roll movement is a rotation that attempts to make the aircraft tip to the left or right along the axis that runs from the nose to the back of the aircraft.

As shown in Figure 2.11, the axes of rotation for a UAV include roll (x-axis), pitch (y-axis), and yaw (z-axis).

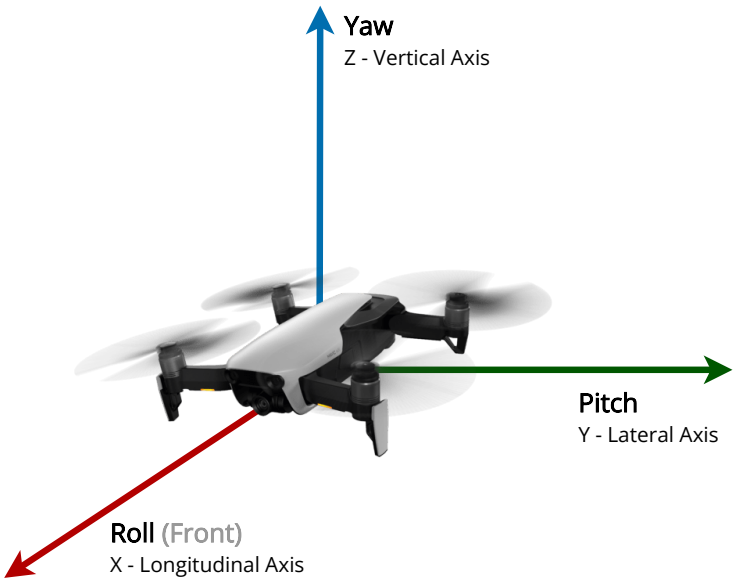

Figure 2.11: Axes of rotation for a UAV: roll (x-axis), pitch (y-axis), and yaw (z-axis).

3
Problem Statement

Contents

This chapter describes the problem and presents related tasks. Specifically, the problem statement outlines the challenges this research should face, and the related work part provides an overview of the current state of knowledge. This will help to understand how our work fits into the broader research context in this area.

3.1 Problem Definition

This thesis focuses on a Deep Reinforcement Learning (DRL) approach, which seeks to teach the agent (UAV) to learn how to move autonomously in an unknown 3D simulated environment in a trial-and-error manner. If learning is successful, the learned model can be transferred to a real UAV, being able to be safely used in the real world. Three different virtual worlds have been created, each with multiple difficulty levels characterized by the number and complexity of obstacles, as well as the intricacy of gate placements. Each world contains differently-shaped obstacles and one gate placed randomly in the world. The agent's goal is to detect the gate and approach it

optimally (i.e. via the shortest path), while avoiding any potential obstacles that may be encountered along the way. To achieve its goal, the agent utilizes the information it receives from the built-in sensors about the state of the environment. Based on this information, it makes decisions and performs actions. The environment generates a reward for each action, expressing whether the action contributed positively or negatively to the agent's task fulfillment. In the early phases of its training, the agent makes random actions, resulting in the agent moving randomly in space and often colliding with obstacles. Gradually, due to the optimization algorithms driving its learning process to maximize the final reward it receives, the agent improves its performance and learns to choose the best possible action for each situation. In this way, the drone is trained to fly in space and approach its targets safely, avoiding any imminent collisions with obstacles [11, 12, 13].

3.2 Related Work

Autonomous navigation of UAVs and obstacle avoidance are problems of considerable concern for the scientific community and industry in recent years. Over the years, many methods have been proposed to address the problem, but none has been widely established. In more detail, the first and most simplistic approach is not based on learning, but on sensing, as obstacle avoidance and path planning are done through traditional path planning algorithms [14]. Although these methods are relatively effective in sparse obstacle scenarios, in more crowded conditions, their performance drops sharply. Another navigation approach relies on retracing pre-planned trajectories using various techniques that minimize the error between the estimated and pre-planned positions. In this case, it is impossible to deal with dynamic environments, exactly because of the pre-planned trajectories [15]. Finally, a reasonably widespread technique used in the past was Simultaneous Localization And Mapping (SLAM). Here, the agent generates a real-time 3D map of its environment through its sensors [16]. Then, using the constructed map, the relative position of the UAV is estimated, and the route it will follow is being planned. This process achieves excellent results in situations where the world is small, since it requires full exploration which is not

always practical or possible, especially in dynamic or large-scale environments, as it relies on a complete understanding of the environment to generate accurate maps and plan efficient routes. [17]

On the contrary, even partial mapping is impossible in complex, dynamic, large-scale environments, as it requires complex mathematical calculations and massive memory. Therefore, the computational burden and cost make applying the method in real-time flight impossible. From the above, it can be concluded that traditional approach methods' inherent shortcomings make them insufficient for autonomous navigation and obstacle avoidance of UAVs in complex environments [18]. Thus, more recent research has turned to methods, such as Imitation Learning and Reinforcement Learning (RL), whose implementation does not require knowledge of the environment [19]. Compared to previous approaches, the results were satisfactory in dynamic environments, but in cases of increasing scale and complexity, they face challenges associated with the curse of dimensionality, where the state and action spaces become too large for the algorithms to handle effectively [20]. Finally, the challenges associated with scaling RL algorithms in high-dimensional spaces necessitated the development of more robust methods, capable of addressing previously intractable problems. A prominent example of this advancement is the fusion of Deep Learning and RL, known as DRL (Deep Reinforcement Learning), which has demonstrated promising outcomes and paved the way for further improvements in this domain [21].

4
Approach

Contents

This chapter presents a proposed approach for autonomous UAV navigation. This approach integrates several critical components for efficient and effective navigation, such as UAV configuration and operating environment. This

chapter provides a comprehensive overview of the proposed approach. It is intended to provide a detailed description of the implementation and a clear understanding of the system. The proposed approach was developed through an extensive review of relevant literature and practical considerations to improve the performance of autonomous UAV navigation.

4.1 UAV Setup

This section presents the package drone model, the sensors, and the reasons they were utilized in the experimental process.

4.1.1 Introducing Hector Quadrotor

As mentioned above, drones often bring high-value equipment and a possible crash would have significant financial consequences. Because of this, a simulated air vehicle was used for the agent's training and testing. More specifically, the hector_quadrotor meta-package was used, which includes packages used for modeling, control, and simulation of quadrotor UAV systems (Figure 4.1). This flying robot is designed to be fully compatible with the ROS Gazebo environment, as it offers many benefits. The basic package provided was modified to equip the drone with the necessary sensors. Thus, the flight algorithm was tested and improved several times during the experiments, without worries about any possible damage to the equipment.

Figure 4.1: The Hector Quadrotor in the Gazebo Simulation Environment

4.1.2 Sensor Modules

In the context of the needs and requirements of this thesis, a set of sensors was selected which ensure satisfactory results in the appropriate time frames and without skyrocketing the financial cost.
The selected sensors are as follows:

- Optical Camera
- GNSS
- IMU
- LiDAR
- Sonar

Optical Camera Module

The central front-facing camera is used only for gate detection, as distance measurement sensors completely cover obstacle avoidance. The vision sensor was first used in this research to enrich the dataset and then identify the gates in real-time during the drone flight. The camera returns a 3-channel (24-bit) RGB image with resolution 1280 px × 768 px.

GNSS Module

The simulated GNSS module consists of a 3-axis digital compass and a satellite receiver. It publishes messages with the robot's position and altitude in WGS84 (World Geodetic System 1984) coordinates. It is worth noting that the drone is trained in ideal weather conditions and the sensors do not exhibit start or stop delays. Therefore, it achieves optimal compass direction accuracy, as well as absolute horizontal dilution.

IMU Module

The hector_quadrotor meta-package includes an IMU sensor plugin, which combines a gyroscope and an accelerometer. The existence of this unit is vital, as it provides information on the acceleration and orientation of the aircraft, i.e. essential characteristics of its flight. The simulated sensor is very close to the real one, as

it is affected by Gaussian noise and low-frequency random drift, which are the two most common errors during flight.

LiDAR Module

The quadrotor is equipped with a 2D LiDAR sensor, called "Hokuyo UTM-30LX", to sense the surrounding environment and avoid obstacles on the horizontal axis. The laser range finder offers a 270° field of view up to 30 meters, as shown in Figure 4.2. Thus, the device measures the distance to nearby objects, Nevertheless only in a two-dimensional plane, for the drone to respond to many realistic scenarios related to safe navigation indoors and outdoors.

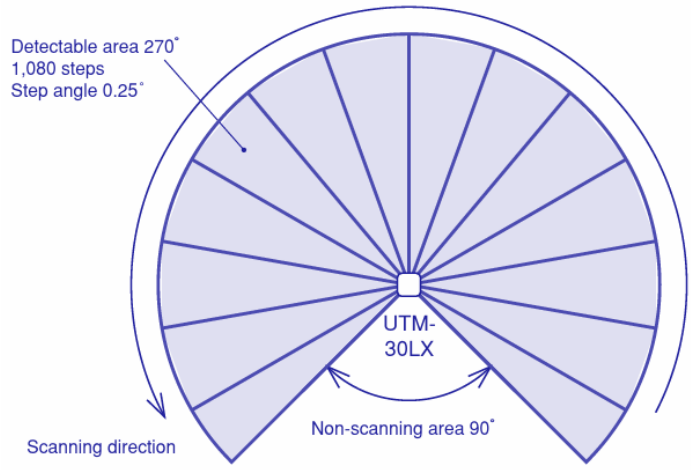

Figure 4.2: Hokuyo UTM-30LX

Sonar Module

In the reinforcement learning domain, the setup of the training environment stands as a cornerstone that governs the safety and efficacy of the model training, testing, and evaluation process. The essential predicate for ensuring a constructive learning pathway for drone navigation is delineating a controlled environment that caters to all the critical requisites and adheres to specified bounds.

Within this purview, a three-dimensional, cube-shaped space with dimensions of $10 \times 10 \times 10$ meters has been delineated to demarcate the operational boundaries for drone flight. This judicious setup provides a clearly defined theater of operations,

eliminating any uncertainties regarding the flight limits and ensuring a structured framework for the drone.

The delineation of these boundaries is not just a functional requisite, but also a safety imperative, as it aids in preventing unwanted excursions beyond the permissible operational sphere, thus averting any unforeseen circumstances that might arise from an unbounded operational scope. If the drone reaches a boundary, it will be treated as a collision, similar to encountering an obstacle. These borders effectively act as walls, ensuring that the drone remains within the defined area. By defining these borders, we confer a degree of predictability to the training process, facilitating the development of a robust drone navigation model that is cognizant of its operating limits and can maneuver adeptly within them.

In essence, establishing this well-defined operational environment is a preliminary, yet pivotal, stride in developing a nuanced reinforcement learning solution for drone navigation.

4.2 Simulation and Environment Setup

4.2.1 Dynamic Environments

General Characteristics

A quintessential feature of the simulation setup is the inclusion of dynamic environments, which have been conceptualized to mirror real-world complexities to the greatest extent possible. These environments are devised to nurture a model adept at navigating many scenarios with varying complexity.

One of the standout features of these environments is the degree of customization they offer. Central to this is the random initialization procedure, which algorithmically generates a new arrangement of obstacles and environmental conditions in each instance of the simulation, ensuring that each session presents a fresh array of challenges for the drone to navigate. This setup is pivotal in avoiding a scenario, where the model might become too accustomed to a static setup, commonly called over-fitting. By constantly altering the structural dynamics, the system compels the algorithm to remain flexible

and adaptive, catering to a broad spectrum of unforeseeable scenarios and promoting a more generalizable understanding of navigation dynamics.

The dynamic environments are delineated into three distinct worlds: block world, sugar cane world, and slat fence world, each nurturing a unique set of obstacles and challenges, thereby ensuring a rich and diverse learning palette. The worlds embody a broad spectrum of geometric structures, ranging from various shapes in the block world to the more-defined cylindrical constructs in the sugar cane and slat fence worlds. The design emphasizes random placement and random characterization of these elements, thus harboring a ground rich for holistic learning [12, 22].

Levels of Difficulty and Complexity

The stratification of these dynamic environments into different difficulty and complexity levels adds another layer of depth to the training process. It serves as a graded pathway, guiding the drone through an ascending series of challenges, each presenting a higher degree of difficulty than the last.

At the foundational level, the system acquaints the drone with a more straightforward setup, facilitating easy navigation with fewer obstacles. As the drone advances through the levels, the complexity multiplies, presenting a denser array of hurdles, characterized by increased obstacles with varying dimensions and orientations.

The distinguishing factor between these levels is primarily the number and configuration of the obstacles that populate the environment. This graded approach facilitates a structured learning pathway. It ensures the drone is progressively equipped to handle more complex navigation scenarios, seamlessly adapting from a novice to a proficient navigator, as it advances through the levels.

Each level serves as a stepping stone in this hierarchical setup, gradually escalating the drone's capability and fostering a deeper understanding of navigational nuances. By training the drone in this manner, it is sought to develop a robust navigational model capable of adeptly handling real-world complexities.

4.2.2 Individual World Descriptions

This section goes deeper into a detailed description of each of the worlds formulated for the training and evaluation of the drone in our setup. These worlds are crafted to cater to different facets of learning for the drone, fostering a comprehensive training regimen.

Block World

The "Block World" emerges as the first in this series, presenting a landscape dotted with diverse geometric shapes. These shapes come in a collection of four distinct types, each offering variations in size, manifested in two or three different dimensions.

An intriguing aspect of this world is the dynamic nature of the environment, where each run witnesses a fresh setup. The system randomly allocates attributes, such as color, size, position, and orientation to the shapes, paving the way for an expansive array of potential scenarios. This randomness instills an essence of unpredictability in the training process, prompting the drone to adapt and learn in a continually changing landscape.

Figure 4.3 illustrates the Block World Simulation Environment, showcasing the variety and randomness of the geometric shapes and their placements.

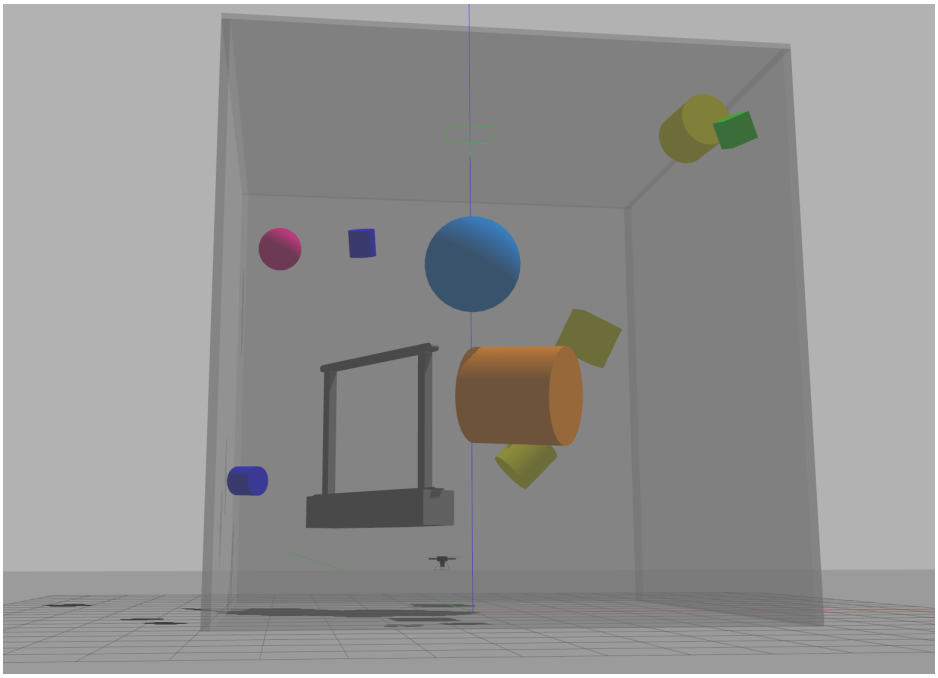

Figure 4.3: The Block World Simulation Environment

Sugar Cane World

Proceeding to the "Sugar Cane World", we find a significant shift in the environmental setup. This world distinguishes itself through a predominant presence of vertical cylindrical obstacles, standing tall, creating a maze for the drone to navigate.

While the structures themselves maintain a constant dimension and predetermined orientation, what brings in dynamism is their random placement in every session. This world, therefore, nurtures precision in navigation, where the drone learns to maneuver through narrowly spaced vertical barriers, enhancing its skills in avoiding collisions and adhering to a safe pathway.

Figure 4.4 illustrates the Sugar Cane World Simulation Environment, showcasing the vertical cylindrical obstacles and their random placements.

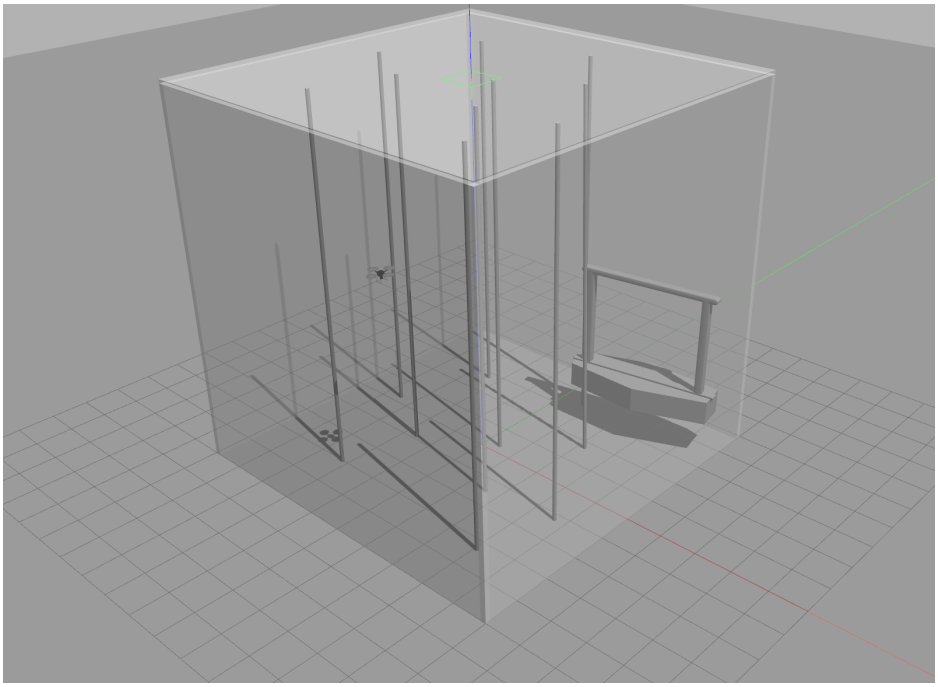

Figure 4.4: The Sugar Cane World Simulation Environment

Slat Fence World

The final stage of this training journey finds its setting in the "Slat Fence World". Here, the environment mirrors the Sugar Cane World, but shifts from vertical to horizontal cylinders, creating a unique lattice of barriers that the drone must traverse.

Similar to the sugar cane setup, the dimensions of these cylindrical obstacles remain constant and their orientation is predetermined. The only variable in this scenario is the random determination of their positions within the world, presenting the drone with a new challenge, every time it embarks on a training session in this world.

By acquainting the drone with horizontal barriers, it is trained to undertake vertical maneuvers effectively, elevating its capability to navigate more complex real-world scenarios, characterized by vertical and horizontal obstructions.

Figure 4.5 illustrates the Slat Fence World Simulation Environment, showcasing the horizontal cylindrical obstacles and their random placements.

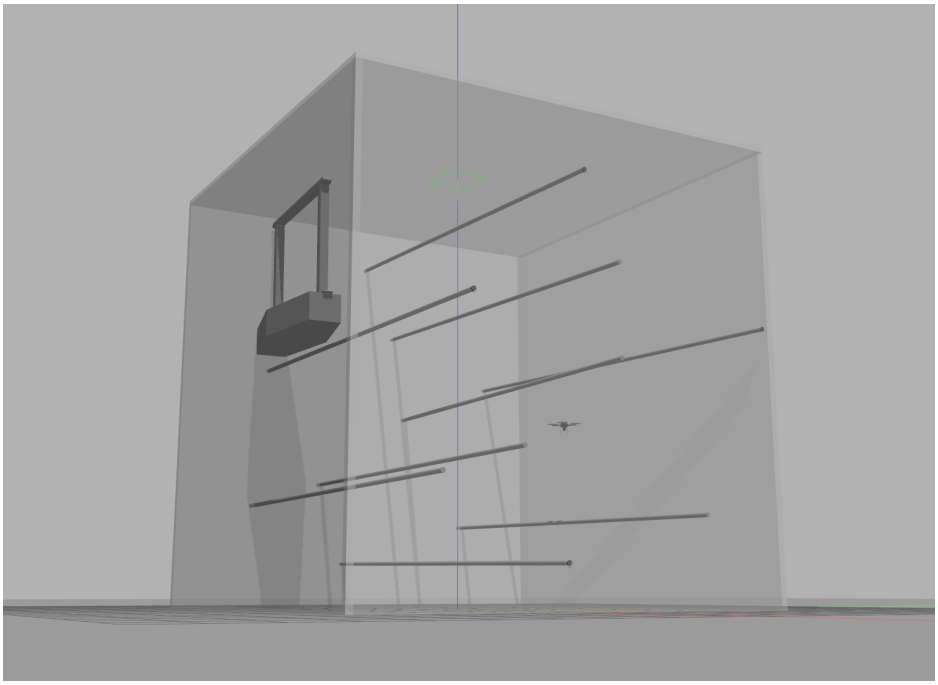

Figure 4.5: The Slat Fence World Simulation Environment

A reasonable question that the reader can raise is why not have a more complex world that combines features of the aforementioned worlds (e.g., vertical and horizontal obstacles). Initially training the agent in simpler environments is a strategic choice based on principles of reinforcement learning. The DQN algorithm benefits from a progressive complexity approach, allowing the agent to build foundational skills effectively. Introducing both vertical and horizontal obstacles simultaneously could overwhelm the agent, leading to instability and inefficient learning. Mathematically, this

complexity increases the variance in the agent's learning process, causing oscillations and difficulty in converging to an optimal policy. Simplified environments ensure efficient data collection and utilization, enabling the agent to learn robust policies within a reasonable training timeframe. This phased approach aligns with curriculum learning principles, where the agent progresses through increasingly difficult tasks, ensuring stable and effective learning. By mastering individual components sequentially, the agent better generalizes its learned policies, preparing it for future expansions into more intricate environments, thus reducing the risk of catastrophic failures and enhancing its overall learning efficiency.

4.2.3 Gate Model

Central to the simulation environment is the three-dimensional gate model, which serves as the ultimate destination in each flight pathway. This model has been meticulously designed to adhere to the specifications regarding dimensions, body structure, and colors, as mandated by the HeroX competition guidelines.

The Blender platform on which it was created facilitates the smooth integration of this intricate model into the simulation setup. This choice of platform ensured that the gate model retained a high degree of fidelity to the structure envisaged for the competition and guaranteed compatibility with the Gazebo simulator used in the current project. Figure 4.6 illustrates the Gate Model in the Gazebo Simulator Environment, showcasing the intricate design and its seamless integration.The symbiosis between the Blender platform and the Gazebo simulator eased the introduction of the gate model into the simulation environment, fostering a seamless simulation process and ensuring the drone had a realistic target to orient towards in each training session.

The model's adherence to the standards stipulated by the HeroX competition accentuates its relevance, offering a realistic and applicable target for the drone to navigate towards. The gate, therefore, stands as more than just a static structure in the simulation; it emerges as a yardstick of success, representing the drone's ability to successfully navigate through the complex pathways, delineated by the various worlds, while avoiding collisions and maintaining a safe course.

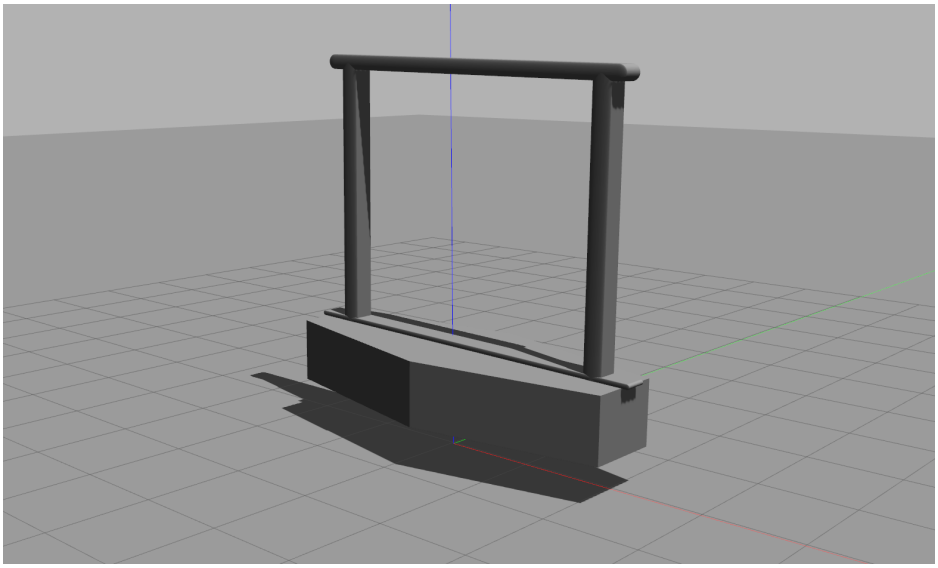

Figure 4.6: Gate Model in Gazebo Simulator Environment

4.2.4 Objective of Using Multiple Worlds

In pursuing a robust and adaptable drone navigation solution via the DQN algorithm, using multiple dynamic worlds is a cardinal strategy to avoid over-fitting. This approach engenders a rich and diverse training ground, where the algorithm is exposed to myriad scenarios, each presenting unique challenges in terms of obstacle configurations and complexity.

A notable benefit of this methodology is the prevention of over-fitting, a common pitfall in machine learning, where the model learns to perform exceedingly well on the training data, but poorly on unseen or new data. The worlds designed in this project, owing to their dynamic nature and random initialization features, mitigate this risk significantly. By ensuring that the drone encounters varied environments, each with its distinct layout and obstacle characteristics, the model is deterred from focusing on too specific patterns, fostering a more generalized understanding that holds steady across training and test datasets.

Employing a plethora of worlds, each distinct in its structural composition and obstacle presentation, not only augments the richness of the training environment, but also offers a fertile ground to gauge the effectiveness of the obstacle and collision avoidance system crafted through the DQN algorithm.

In addition, it allows for an in-depth study of how varying environments and external geometries can influence the RL algorithm, essentially creating a sandbox, where the algorithm's adaptability and resilience.

4.3 Reinforcement Learning Framework

In this part of the thesis, the focal point shifts to delineating the fundamental aspects that structure the reinforcement learning (RL) framework vital to UAV navigation. This section seeks to elucidate the distinct configurations constituted in the RL, inclusive of the action and observation spaces, the reward system, and the terminal states. These elements play a pivotal role in shaping the learning trajectory of the UAV, thereby facilitating its adept navigation in the specified environment. Figure 4.7 illustrates the overall workflow of the reinforcement learning process.

Figure 4.7: Reinforcement Learning Workflow.

4.3.1 Action Space and Actions

In the context of drone navigation, the drone's motion can be intricately manipulated through four basic movements: pitch (forward or backward movement), roll (left or right movement), throttle (up or down movement), and yaw (left or right rotation). These movements, inherently continuous due to their wide range of values, can be controlled independently. However, discretizing this action space becomes essential, when employing Deep Q-Learning (DQN).

To this end, a specific set of discrete actions is defined to represent varied changes in the drone's motion. In the current implementation, eight actions have been chosen to represent distinct maneuvers: moving forward, moving backward, moving right, moving left, moving higher, moving lower, rotating clockwise, and rotating anti-clockwise. Each

action is associated with a specific velocity increment or decrement, which has been fine-tuned to ensure precise control during navigation. Each action corresponds to an alteration in only one of the four basic movements. For example, the "move forward" action increases the pitch, but has no effect on roll, yaw and height, while the "move right" action increases the roll, but has no effect on pitch, yaw and height. Furthermore, each action is tied to a particular velocity increment or decrement to maintain precision and control of the drone's movements. When an action, such as "move forward" is chosen, the drone's forward velocity increases by a predetermined amount. This increase is capped at a maximum value to prevent uncontrolled acceleration, if the drone is already moving forward. Conversely, if the drone moves backward, selecting "move forward" gradually slows it down, before reversing direction.

Table 4.1 provides a concise overview of the drone's actions and the specific movements they affect, ensuring a clear understanding of how each action influences the drone's navigation.

Action Name	**Movement Affected**	**Description/Notes**
Move forward	Increase Pitch	Increases velocity
Move backward	Decrease Pitch	Decreases velocity
Move right	Increase Roll	Increases rightward motion
Move left	Decrease Roll	Increases leftward motion
Move higher	Increase Throttle	Raises the drone
Move lower	Decrease Throttle	Lowers the drone
Rotate clockwise	Increase Yaw	Rotates the drone clockwise
Rotate anti-clockwise	DecreaseYaw	Rotates the drone anti-clockwise

Table 4.1: Concise Overview of Drone's Actions and Movements

Utilizing a discrete action space in this manner enables the application of DQN to learn a drone navigation policy. Each discrete action corresponds to a specific change in the drone's motion, and the DQN algorithm learns to choose the appropriate action based on the environment's current state. This implementation trades off the granularity of control for computational feasibility, while still harnessing DQN's power. Though the drone's natural movements are continuous, a discrete action space allows for the practical and successful application of DQN.

However, discretizing the action space raises the critical consideration of action granularity. Essentially, this involves deciding the number of discrete actions that will represent the drone's movements. The choice here is pivotal, as it directly impacts DQN's learning efficiency. More actions afford finer control, but increase the complexity of Q-value estimation. DQN must estimate Q-values for each state-action pair; having more actions means more pairings. This can prolong the learning process, as more Q-values are estimated; in certain cases, it may even destabilize the learning process.

In contrast, fewer actions mean DQN has fewer Q-values to estimate, which can expedite the learning process. However, this comes at the cost of limiting the drone's navigation capabilities due to a reduced set of responses to environmental states. This could be particularly restrictive, when nuanced adjustments are required for precise navigation.

In conclusion, balancing action granularity and learning efficiency is crucial, when designing a discrete action space for drone navigation with DQN. This balance is contingent on the specifics of the navigation task and drone performance requirements. Determining the optimal set of discrete actions might necessitate a phase of experimentation or systematic fine-tuning.

4.3.2 Observation Space

In this thesis, developing an autonomous drone navigation system necessitates carefully selecting and organizing information that the drone can use to make decisions. To this end, the observation space (or state space, for our purposes) plays a pivotal role, as it comprises the sensory input and derived information that the drone utilizes to perceive its environment and decide its actions. The observation space is structured as a 1D array of features, passed to the Deep Reinforcement Learning (DRL) agent. For computational efficiency and to facilitate the learning process, it's essential to present this information in a structured and meaningful way.

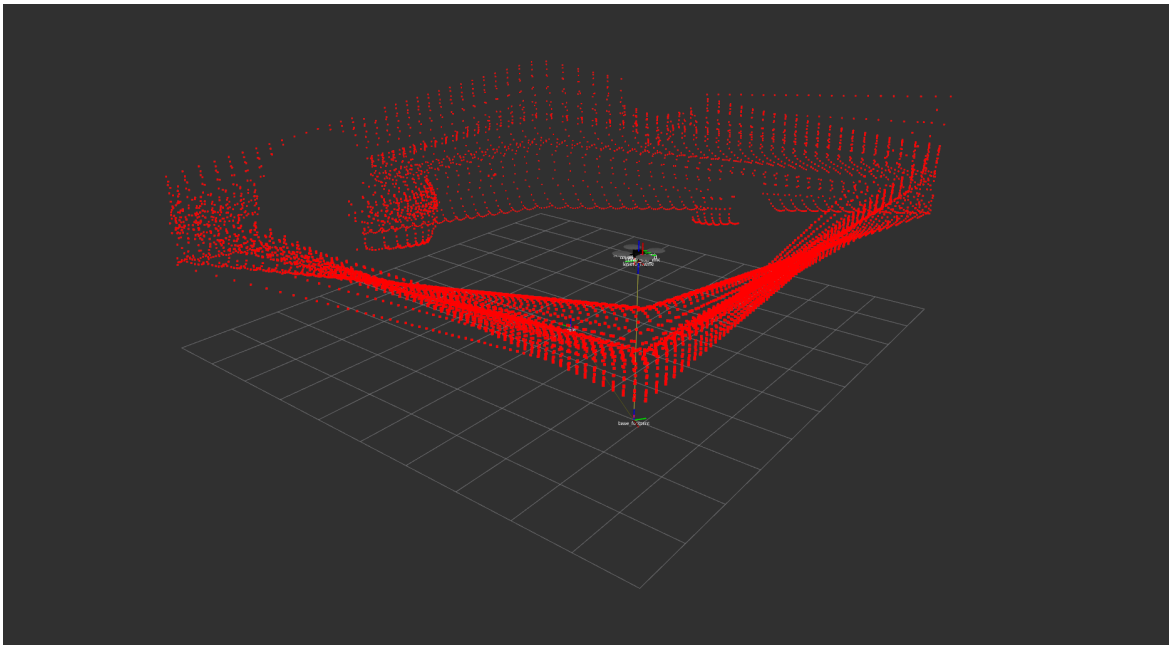

Figure 4.8: Visualization of Lidar Data in RViz

The observation space is divided into three principal categories: Distance Measurements, Orientation Measurements, and Goal-Oriented Measurements. This categorization is motivated by the nature of the information and its utility in the drone's navigation tasks.

1. **Distance Measurements:** This category encompasses the drone's sensory information regarding its immediate surroundings. It includes the Lidar measurements, averaged over six sectors covering a 270-degree field and the normalized altitude. These measurements are crucial for collision avoidance and safe navigation through unknown environments. The Lidar data is reduced to six sectors to balance the richness of data and computational efficiency.

2. **Orientation Measurements:** Orientation Measurements is essential for controlling the drone's orientation and ensuring a stable flight. It includes the tilt and yaw angles represented in their cosine and sine. This representation is chosen, as it provides continuity, when angles wrap around, which benefits the learning process.

3. **Goal-Oriented Measurements:** This category is central to the drone's ability to navigate towards its target successfully. It includes information about the relative position and orientation of the target, represented in polar coordinates, and the rate at which the distance to the target changes. This temporal information can be crucial for evaluating the efficiency of the drone's trajectory toward the goal.

Organizing the observation space into these three categories provides the drone with a structured and efficient representation of the environment. The processing of these observations is handled by the DRL agent, which analyzes the data to make informed navigation decisions dynamically during each training session. This helps the DRL agent more effectively learn the relationships between its sensor data, the state of the environment, and the actions needed to navigate complex indoor and outdoor scenarios successfully. Moreover, this structure helps modularize the approach, which can be beneficial for troubleshooting, fine-tuning, and possible future system extensions.

Distance Measurements

In autonomous drone navigation, understanding the surrounding environment is crucial for safely avoiding obstacles and efficiently navigating toward the target. The Distance Measurements category in the observation space is designed to provide the drone with essential information regarding its proximity to objects and surfaces and its distance from the goal. This category includes Lidar measurements averaged across six sectors, normalized altitude, and relative distance to the gate. An analysis of each component follows:

1. **Lidar Average Distances (Six Sectors)**

   Purpose: Lidar sensors provide extensive data, which can be computationally expensive to process in real-time. By dividing the 270-degree field of view into six equally-sized sectors of 45 degrees each and calculating the average distance in each sector, the data's dimensionality is reduced, while

maintaining crucial spatial information. The formula for calculating the average distance in each sector is as follows:

$$\text{Average Distance} = \frac{1}{N} \sum_{i=1}^{N} \text{Distance}_i \tag{4.1}$$

where N is the number of data points in each sector. This summarized representation enables the drone to make rapid navigation decisions, especially when swift responses are necessary.

Benefits: Simplifying Lidar data into six sectors allows the drone to promptly identify directions that are relatively free of obstacles, assisting in path planning and collision avoidance. This reduction also decreases the computational load, enabling real-time processing and faster decision-making.

2. **Sonar Readings**

Purpose: In both indoor and outdoor environments, the ability to sense proximity to obstacles above and below the drone is crucial. The sonar readings provide real-time data about the distance to the nearest object or surface detected above and below the drone. These readings are directly used to inform the drone's vertical navigation decisions.

Details: Upward-facing Sonar: Measures the distance to overhead obstacles, helping the drone to avoid collisions when ascending or when flying under structures. Downward-facing Sonar: Provides the distance to the ground or objects below the drone, which is vital for maintaining a safe altitude and avoiding crashes into the ground.

Benefits: Utilizing these sonar readings allows the drone to maintain a safe operating altitude, adjust its flight path to avoid obstacles, and ensure efficient navigation. This approach simplifies the observation space while providing all necessary information for safe and effective drone operation.

The Distance Measurements category ensures the drone can safely and efficiently navigate its environment. By effectively summarizing spatial data and presenting it in a format amenable to real-time processing, this category allows the drone to make well-informed decisions on its movement strategies. This becomes increasingly critical in unknown environments, where the drone must rely heavily on sensor data to understand and adapt to its surroundings.

Orientation Measurements

The drone's orientation in three-dimensional space is paramount for efficient and safe flight. Orientation Measurements refers to the orientation and angular positioning of the drone relative to its environment. Correctly understanding and adjusting its orientation is crucial for the drone to maintain stability, avoid obstacles, and move in the desired direction. In the observation space, the Orientation Measurements category is represented by the cosine and sine of the tilt angle and the cosine and sine of the yaw angle. Using sine and cosine values to represent angles helps handle the wraparound of angles, providing a smooth representation for algorithms and essentially providing a unit vector representation of direction. Here is a deeper analysis of each component:

1. **Cosine and Sine of the Tilt Angle** cos(**Tilt**)**,** sin(**Tilt**)

   Purpose: The tilt angle represents the drone's inclination with respect to the horizontal plane. In other words, it is the angle between the drone's longitudinal axis (front to back) and the horizontal axis. By representing the tilt angle in terms of its cosine and sine, the drone can understand its tilt in a format more conducive to mathematical processing and control algorithms.

   Benefits: Knowing the tilt angle is essential for the drone's stability and control, especially in an indoor environment with obstacles. By actively monitoring and adjusting the tilt, the drone can ensure that it doesn't inadvertently ascend or descend and can make finer adjustments to its path, which is

especially crucial, when navigating through narrow passages or avoiding obstacles.

2. **Cosine and Sine of the Yaw Angle** $\cos(\mathbf{Y})$**,** $\sin(\mathbf{Y})$

   Purpose: The yaw angle represents the drone's rotation around about the vertical axis, indicating which direction the drone faces. Like the tilt angle, the yaw angle, in terms of its cosine and sine, is advantageous for mathematical processing and control algorithms. This representation is beneficial for understanding the drone's heading in a circular manner, rather than a linear scale, which is vital for rotation.

   Benefits: Knowing the yaw angle is critical for direction control. The drone must know which way it is facing to effectively navigate toward its goal or away from obstacles. The drone can make rotational adjustments by monitoring the yaw angle to align itself with the desired path.

The Orientation Measurements is vital for the drone's ability to control its orientation in three-dimensional space. Proper orientation control is fundamental to stable flight and efficient navigation, especially in environments with various obstacles. By representing the tilt and yaw angles using their sine and cosine values, the drone can process its orientation information in a manner that is well-suited for the trigonometric calculations often used in control algorithms. Additionally, using sine and cosine allows for handling the cyclic nature of angles, which is particularly beneficial in navigation tasks. This, in turn, contributes to more precise and responsive adjustments to the drone's orientation during flight.

Goal-Oriented Measurements

This category contains information crucial for the drone to navigate toward its target efficiently. This includes the distance, azimuth, and elevation angles relative to the gate and how fast the drone is approaching or moving away from the gate. Additionally, using polar coordinates for representing relative position to the gate is essential to

provide directional context in 3D space, which is more effective and intuitive for navigation than Cartesian coordinates.

1. **Relative Distance to the Gate (RDG)**

   Purpose: The relative distance to the gate gives the drone a direct measurement of its distance from its target in Euclidean space. The drone needs to gauge its proximity to the goal to evaluate the efficiency of its path and make necessary adjustments.

   Benefits: Knowing the relative distance to the gate helps the drone in several ways. It can estimate how effectively it is approaching the goal and use this measurement as part of the reward function in the reinforcement learning algorithm. This distance serves as an indicator of progress and reducing this distance can be a positive signal for the learning agent.

   Analysis of Choice of Euclidean over Manhattan Distance

   Nature of Movement: Drones can move freely in three-dimensional space and take a direct path between two points. Here, (x_1, y_1, z_1) and (x_2, y_2, z_2) are the coordinates of the two points in 3D space. The Euclidean distance, represented as:

$$d_{\text{euclidean}} = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2}$$

   accurately represents this straight-line distance, making it more suitable for drone navigation in 3D space compared to Manhattan distance, which is better suited for grid-based pathfinding.

   On the other hand, Manhattan distance is represented as:

$$d_{\text{manhattan}} = |x_2 - x_1| + |y_2 - y_1| + |z_2 - z_1|$$

   This is more relevant in cases, where movements are constrained to grid paths, as it sums the absolute differences along each axis.

Physical Relevance: Euclidean distance accurately reflects the direct path a drone can take in three-dimensional space. For meaningful navigation, distance measurements must mirror the physical characteristics of the environment. The Manhattan distance would not accurately represent the distance the drone would need to travel in 3D space.

Computational Considerations: While Euclidean distance involves the square root operation, which is computationally more expensive than the simple addition in Manhattan distance, it is more representative of the physical path. This representation is more meaningful for the drone's navigation algorithm and potentially leads to more efficient learning and decision-making processes.

In summary, the choice of Euclidean distance in measuring the relative distance to the gate is grounded in its alignment with the drone's three-dimensional movements and its physical accuracy in representing distances in such a space. This ensures more meaningful data for decision-making and learning within the drone's navigation system.

2. **Azimuth and Elevation**

Purpose: To inform the drone of the direction in which the gate lies in 3D space relative to its current position and orientation. This is done using polar coordinates, which are inherently directional and provide a more intuitive representation for navigation in 3D space compared to Cartesian coordinates (ΔX, ΔY, ΔZ).

Benefits: With this information, the drone can adjust its orientation and make directional changes to move toward the gate. Polar coordinates capture the essence of the drone's relative position to the gate in terms of angles, which is more aligned with the way the drone moves and perceives its environment.

Calculation: Azimuth is the angle measured in the horizontal plane between a reference direction (usually north or the drone's current direction in our

case) and the projection of the point onto the horizontal plane. Elevation is the angle measured in the vertical plane between the reference (horizontal) direction and the projection of the point onto the vertical plane.

Here, (x_g, y_g, z_g) are the coordinates of the gate, and (x_d, y_d, z_d) are the coordinates of the drone.

$$\text{Azimuth} = \text{atan2}\left(y_g - y_d, x_g - x_d\right)$$

$$\text{Elevation} = \text{atan2}\left(z_g - z_d, \sqrt{\left(x_g - x_d\right)^2 + \left(y_g - y_d\right)^2}\right)$$

Here, $\text{atan2}(y, x)$ is the arctangent function that gives the angle in radians from the x-axis to a point (y, x).

3. **Rate of Change in Distance to Gate**

Purpose: This represents how fast the distance to the gate changes over time. Since the time steps are small, the change in distance can be minimal and noisy. A rolling average over several time steps is used to smooth out this measurement and make it more representative of the drone's movement.

$$\text{Rate of Change in Distance to Gate} = \frac{1}{N \times \Delta t} \sum_{i=1}^{N} (\text{Current Distance}_{\text{gate},i} - \text{Previous Distance}_{\text{gate},i})$$

where N denotes the number of time steps over which the averaging of the distance change is executed.

Benefits: Smoothing the rate of change in distance over multiple time steps reduces the impact of noise and provides a more reliable indicator of the drone's movement toward or away from the gate. This information is critical for the drone to evaluate and adapt its strategy in real time and ensures a more stable learning process. The modification in the calculation of the rate of change in distance to the gate will help in capturing a more meaningful

representation of the drone's movement, especially when the time steps are small.

In summary, the Goal-Oriented Measurements category aims to equip the drone with the necessary information to efficiently navigate towards its target. The use of polar coordinates provides a more intuitive and effective representation for navigation in three-dimensional space. These coordinates coupled with information about the rate at which the drone is approaching or receding from the target allow for more informed and dynamic decision-making.

4.3.3 Reward System

Navigating the multifaceted realm of autonomous drone navigation, especially in intricate and dynamic settings, requires a comprehension of its inherent challenges and the application of robust methodologies to surmount them. The reward system is central to these methodologies and the bedrock of Deep Reinforcement Learning (DRL) - a meticulously structured feedback mechanism that steers our autonomous drone's real-time decisions.

Imagine the reward system as an astute instructor guiding a diligent pupil. Upon demonstrating proficiency, this pupil earns accolades (a positive reward) and, upon faltering, receives constructive critiques (a negative reward or penalty). The goal is unambiguous: guide this pupil—or, in the context of this study, the drone—towards choices that resolutely conform to the task's objectives and stringent constraints.

Nevertheless, tailoring a reward function to facilitate drone navigation seamlessly is no trivial pursuit. It should encapsulate an intricate dance of priorities: safeguarding the drone, fostering adept navigation, aligning accurately with the target, and refining the flight speed. Each element bears significance, and their collective oversight could spell mediocrity or disasters.

Deep Q-Networks (DQNs), an exemplar of reinforcement learning, crystallize this challenge. Their primary purpose is the maximization of cumulative rewards over time, with the reward function acting as the linchpin. The stakes are palpable: a mal-designed reward function could misdirect the DQN agent toward undesired

or detrimental policies. Conversely, with a reward function exuding precision, the agent can soar toward task mastery.

A unique conundrum this drone faces is discerning between obstructions—requiring circumvention—and gates—demanding passage. Given the gate's restrictive width, akin to being in close quarters with obstructions, such discernment is not intrinsically evident from raw sensor data. The agent's learning curve, thus, hinges heavily on the reward function's calibration to simultaneously champion safe traversal and successful gate navigation.

The drone's real-time orientation adjustments, specifically its yaw and tilt, cannot be understated in this intricate dance. An adept reward function accounts for this, ensuring holistic 3D navigational learning. Furthermore, the drone's navigation should epitomize efficiency, prompting the reward function to be attuned to the drone's instantaneous state and its temporal evolution. This dynamism is manifested in terms that evaluate proximity to the gate and its alterations over successive timesteps.

Infusing this reward function with nuances are the plethora of observational space elements, from LIDAR and SONAR sensor readings to relative distances and angular orientations. Each component furnishes the agent with granular insights into its environment and mission, rendering the reward function comprehensive and instructive.

Starting this chapter, one endeavors to dissect and illuminate the intricacies of the drone's reward system. One will traverse the motivations, the mathematical scaffolding, and the strategic imperatives that craft a reward system primed for success in the challenging world of autonomous drone navigation.

Collision Avoidance Reward

One of the primal fears of any drone operator or autonomous system developer is the dreaded "collision". Not only does a collision spell an end to the drone's mission, but it can also significantly damage the drone itself and the environment or even pose risks to human safety. In the realm of DRL, this fear is encapsulated and addressed using the collision avoidance reward.

The main objective behind the collision avoidance reward is to ensure that the drone maintains a safe distance from potential obstacles. The reward function is designed to recognize imminent collisions based on sensor readings and penalize or discourage such scenarios from achieving this. By doing this, the DRL agent learns to associate near-collision states with negative feedback, navigating away from them in future iterations.

Mathematical Formulation

The reward function for collision avoidance is given by:

$$R_t^{\text{collision}}(d) = \frac{1}{1 + e^{10 \cdot D_{\text{col}} - 5d}}$$

Here, d represents the minimum distance of any of the sensors, which include six LIDAR values and two SONAR values. The term D_{col} is a constant that denotes the threshold distance at which a collision is considered imminent. D_{col} can take values in the range $0 < D_{\text{col}} \leq 5$ m.

The function leverages the sigmoid activation function, characterized by its S-shaped curve, to smoothly transition the reward values based on the distance to obstacles. As d approaches D_{col}, the reward value approaches 0, indicating a higher collision probability. On the other hand, as d increases well beyond D_{col}, the reward value gets closer to 1, indicating a safer scenario.

Significance and Benefits

Unlike binary feedback mechanisms, this function offers a gradient of values, providing nuanced feedback to the agent, based on the severity of the impending collision threat. In addition, given its mathematical formulation, the reward rapidly changes as the drone gets perilously close to obstacles. This promotes quick evasive actions, essential for safe navigation in dynamic environments. The function ensures comprehensive

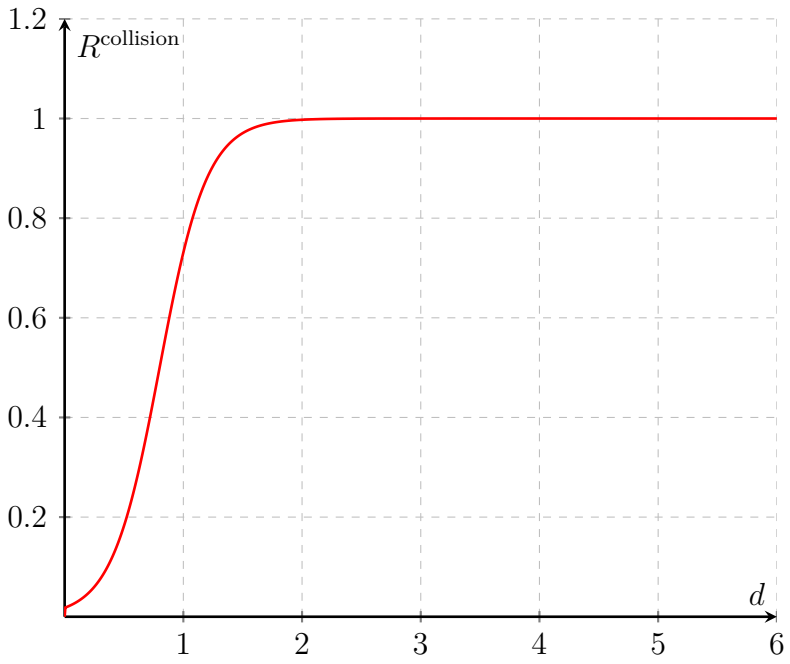

Figure 4.9: Collision Avoidance Reward with $D_{\text{col}} = 0.4$ m.

spatial awareness by taking the minimum value from collecting sensors. This way, the agent remains vigilant about obstacles from all directions.

In conclusion, the collision avoidance reward, $R_t^{\text{collision}}(d)$, serves as the guardian of the drone's safety. It steers the agent away from potential dangers, ensuring the primary goal: to keep the drone unharmed and functional throughout its mission. As we progress, we will see how this foundational reward integrates with others to create a cohesive learning environment for the agent.

Distance to Gate Reward

Navigating a drone through a vast space is not solely about avoiding obstacles; reaching specific waypoints or objectives is equally essential. In this case, the most prominent objective is the "gate". Ensuring the drone efficiently and consistently approaches and passes through this gate is fundamental, and this purpose is achieved using the distance-to-gate reward.

The gate represents a critical waypoint or destination for the drone on a mission. The primary aim behind the distance-to-gate reward is to ensure that as the drone navigates,

it constantly aligns its trajectory to approach the gate. The closer the drone gets to the gate, the higher the reward, instilling positive reinforcement toward the desired behavior.

Mathematical Formulation

The reward function for distance to the gate is defined as:

$$R_t^{\text{gate}}(d) = \frac{1}{1 + e^{\frac{20d}{d_{\text{cosmos}}} - 5}}$$

Here, d signifies the current distance of the drone from the gate. The term d_{cosmos} is a constant that represents the dimensions of the operational environment along any given axis in meters. d_{cosmos} can take values in the range $0 < d_{\text{cosmos}} \leq 100$ m. Similar to the collision avoidance reward, the sigmoid function also inspires this function. As d gets smaller (indicating proximity to the gate), the value of the reward approaches 1. Conversely, as d increases (signifying the drone moving away from the gate), the reward dwindles towards 0.

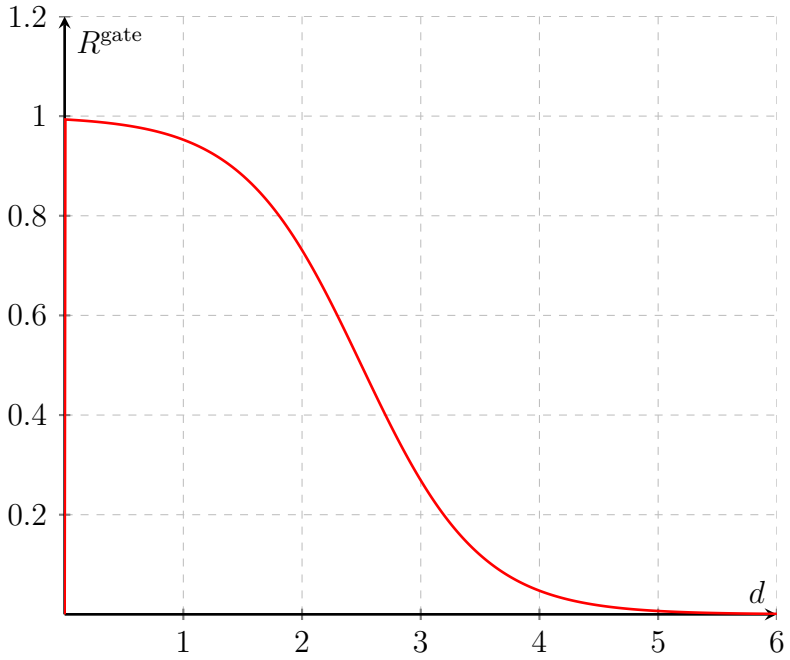

Figure 4.10: Distance to Gate Reward with $d_{\text{cosmos}} = 10$ m.

Significance and Benefits

This reward acts as a continual incentive for the drone to progress towards the gate. Regardless of where the drone is, this reward always points it in the right direction. Moreover, by factoring in d_{cosmos}, the reward scales itself according to the size of the operational environment. This ensures that the function remains practical and relevant regardless of the mission's spatial context. The sigmoid-based approach ensures a smooth gradient in rewards as the drone gets closer or farther from the gate. This promotes more seamless and gradual trajectory adjustments than abrupt course corrections.

In conclusion, the distance to gate reward, $R_t^{\text{gate}}(d)$, is integral for mission success. While the collision reward ensures the drone's safety, the gate distance reward ensures its mission efficacy, guiding it toward its primary objective.

Approach Reward

The complex environment in which drones operate is about more than just static positions or fixed distances. It also revolves around the motion and the rate at which changes occur. The Approach Reward captures this essence by focusing on changing the drone's distance from the gate between consecutive time steps. It provides an insight into the drone's motion direction. The approach reward focuses on the drone's momentum towards the gate. The principle here is to motivate the drone to be near the gate and encourage it to make consistent progress toward it. Even if the drone is initially far from the gate, as long as it moves closer, this reward acknowledges and encourages that effort.

Mathematical Formulation

A piecewise function defines the approach reward:

$$R_t^{\text{approach}}(\Delta d) = \begin{cases} 1, & \text{if } \Delta d \geq +K \\ 0, & \text{if } \Delta d < -K \\ F \cdot \frac{\Delta d + K}{2}, & \text{otherwise} \end{cases} \tag{4.2}$$

Where:

Δd is the change in distance between two consecutive time steps, defined as $\Delta d = d - d_{\text{previous}}$.

K is the maximum expected travel distance between two-time steps, determined empirically. The range for K is $0 < K \leq 0.1$ meters.

F is the agent's refresh rate, with a range of $30 \leq F \leq 60$ Hz.

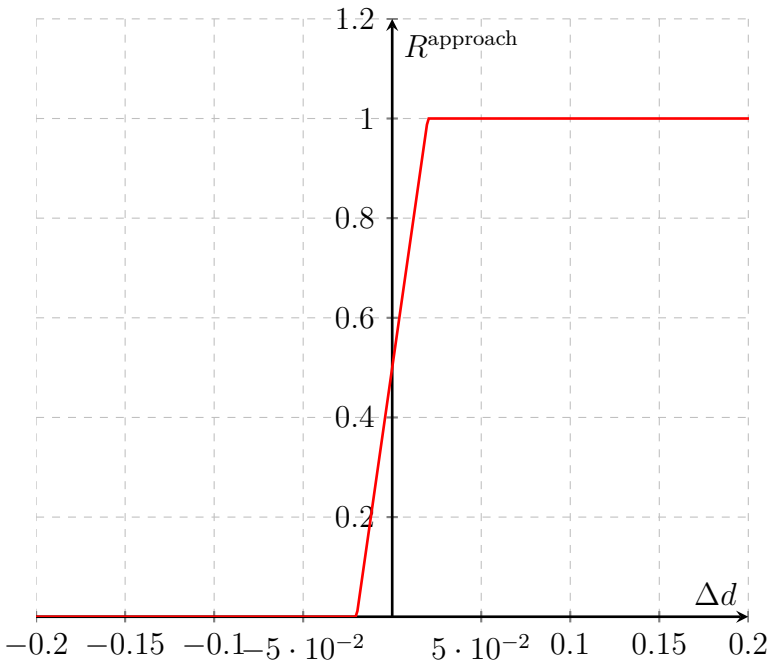

Figure 4.11: Approach Reward with $K = 0.02$ meters and $F = 50$ Hz.

Significance and Benefits

Unlike passive rewards that react to the current state, the approaching reward is forward-looking. It anticipates future actions and encourages the drone to move proactively toward the gate. The reward structure discourages the drone from hovering at a spot or moving back and forth without progressing toward the gate. A drone that stalls or oscillates will receive a diminished reward, motivating it to progress instead. Drones are subjected to various disturbances during flight and might not

always move at an ideal rate. This clause ensures that even if the drone's progress is imperfect, its efforts are rewarded proportionally. The value of K, derived from empirical observations, provides adaptability to the drone's operational conditions. Different missions or drones with varied capabilities might have different ideal travel distances, and by adjusting K, this reward can be suitable for varied contexts.

To sum up, the Approach Reward, $R_t^{\text{approach}}(\Delta d)$, serves as a dynamic counterpart to the more static distance to gate reward. Considering the drone's trajectory and pace ensures that it is not just near the gate, but actively approaching it, fostering proactive and consistent behavior throughout its mission.

Alignment Reward

In the multifaceted world of drone navigation, alignment stands out as one of the crucial factors determining the success of a drone's mission, especially when it has to pass through specific structures, like gates. The Alignment Reward captures this notion, focusing on the drone's orientation, particularly its yaw, concerning the target gate. This encourages the drone to be proximal to the gate and well-aligned for a smooth passage.

Mere proximity is insufficient when a drone aims to fly through a gate or any structured opening. The drone must ensure its orientation to easily pass through without colliding with the gate edges. An improper alignment, even if the drone is close to the center of the gate, could result in collisions with the gate sides or top.

Mathematical Formulation

The reward formula for alignment based on the yaw difference between the drone and the gate is given by:

$$R_t^{\text{align}}(\Delta\theta) = \frac{\cos(\Delta\theta) + 1}{2}$$

Where:

$\Delta\theta$ represents the angle difference between the drone's yaw and the gate's yaw orientation.

The cosine function offers a smooth gradient of values, ensuring that the reward is highest when the drone's yaw is perfectly aligned with the gate ($\Delta\theta = 0$) and decreases as the alignment deviates.

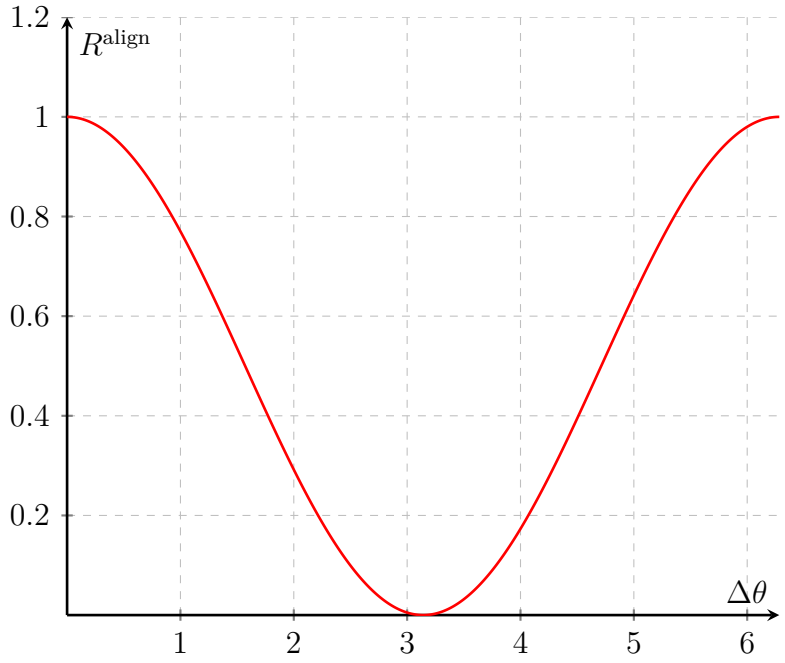

Figure 4.12: Alignment Reward

Insights and Implications

The formula normalizes the cosine value to lie between 0 and 1, ensuring the reward's consistency with other components of the overall reward system. The highest reward is attained when the yaw alignment ($\Delta\theta = 0$) is perfect, meaning the drone directly faces the gate. This promotes a straight pass-through, reducing the chances of edge collisions. Even if the drone is not perfectly aligned, it still receives a proportion of the reward, encouraging it to correct its course over time, instead of aiming for perfection at every instant.

In conclusion, $R_t^{\text{align}}(\Delta\theta)$ serves as a critical tool in refining the drone's navigation through structured environments. While other rewards ensure the drone gets to the gate,

the alignment reward ensures it passes efficiently. This reward emphasizes precision, an attribute crucial for drones operating in constrained or precision-required scenarios.

Composition of the Total Reward

One of the core principles of reinforcement learning is that an agent learns to make decisions based on the reward signals it receives in response to its actions. As such, the reward function is indispensable in shaping the agent's behavior. In the context of our drone navigation system, the reward function is not just a single, monolithic entity; it is a carefully constructed combination of different reward components, each addressing specific aspects of the drone's mission. When navigating in a three-dimensional space with numerous challenges, a singular reward signal might not sufficiently capture the nuances of the task. The drone's mission is not just about avoiding collisions or aligning with the gate. It is about doing all these things simultaneously and optimally. Therefore, the reward signal must reflect the multi-faceted nature of this task, encouraging the drone to achieve a balance between different, possibly competing objectives.

The total reward function is an aggregate of the following individual rewards:

Collision Reward: It penalizes or rewards the drone based on how close it gets to an obstacle. This encourages the drone to maintain safe distances from obstacles and avoid collisions.

Distance to Gate Reward: This reward ensures the drone is motivated to reach its target gate. It rewards the drone based on its proximity to the gate, thus acting as a guiding beacon.

Approach Reward: Navigation is about more than just reaching the target, but doing so efficiently. The approach reward ensures that the drone is not merely wandering around, but is making meaningful progress towards the gate.

Alignment Reward: Lastly, the drone is rewarded for its proximity to the target, ensuring smooth passage through gates or structured openings.

Mathematical Formulation

The total reward, R_t, at any time step t is a simple summation of the individual rewards:

$$R_t = R_t^{\text{collision}} + R_t^{\text{gate}} + R_t^{\text{approach}} + R_t^{\text{align}}$$

All the reward components have equal importance and are normalized to [0,1], ensuring a balanced contribution to the total reward signal.

Considerations and Implications

By combining rewards, the drone does not over-optimize for a single objective. Instead, it learns to navigate while balancing safety, efficiency, and precision. This composite reward system provides the drone with continuous feedback on multiple fronts. This is vital in large state spaces, where sparse rewards could lead to slow or suboptimal learning.

The total reward reflects the mission objectives and the drone's challenges. It encapsulates the complexity of the task, providing the drone with a clear and comprehensive feedback mechanism to learn and adapt its navigation strategies.

Dense Rewards in Large State Spaces

In reinforcement learning, the choice between dense and sparse rewards is crucial, driving how an agent learns and adapts its behavior. This decision becomes even more consequential, when the agent operates in a vast state space, where the sheer number of possible states makes the learning task daunting. This section will delve into the importance of dense rewards, especially in large state spaces, and highlight why such a reward mechanism was adopted for our drone navigation system.

Sparse vs Dense Rewards

Sparse Rewards: Sparse rewards are occasional feedback signals given to the agent. In a navigation task, a typical sparse reward might only be granted when the drone

successfully reaches its destination or hits an obstacle. The intervening states offer no reward signal, leaving the agent largely "in the dark" about its performance.

Dense Rewards: Contrarily, dense rewards provide the agent with frequent feedback. As the drone navigates, it receives reward signals guiding its learning at nearly every step.

Advantages of Dense Rewards in Large State Spaces

Dense rewards ensure the agent is always provided with feedback for long periods. This consistent guidance is essential in vast state spaces, where agents can quickly get "lost" without frequent course corrections. So, with more frequent feedback, the agent can regularly update its policies, leading to faster convergence towards optimal strategies. Moreover, the exploration-exploitation dilemma becomes particularly pronounced in expansive state spaces. Dense rewards offer the agent constant nudges, assisting it in navigating the exploration phase effectively. They also reduce the chances of the agent getting stuck in local minima, since the agent constantly receives feedback about its actions, enabling it to maneuver out of sub-optimal strategies.

Sparse rewards could have left the drone floundering in this vastness, unsure of its performance, until it reached a gate or collided. Choosing a dense reward mechanism ensures that the drone always has a sense of direction. Every action, be it a slight change in altitude or a minute turn, results in feedback, ensuring the learning process is continuous, dynamic, and effective.

While dense rewards offer numerous advantages in large state spaces, they are not without challenges. There is a risk of overwhelming the agent with too much information, and the quality of the dense rewards is paramount. If not designed thoughtfully, dense rewards might even mislead the agent, for instance, by over-emphasizing less critical aspects of the task, while overshadowing more important objectives.

In summary, the vastness and complexity of our drone's operational environment warranted dense rewards. This decision, rooted in the challenges and intricacies of

large state spaces, has been foundational in guiding our drone's learning process and ensuring its effective navigation.

4.3.4 Terminal States

In Reinforcement Learning (RL), an agent interacts with an environment by taking actions, receiving rewards, and transitioning between states. A significant aspect of this interaction, often overlooked in initial discussions, but central to the agent's learning cycle, is the concept of terminal states. Understanding the terminal state allows someone to comprehend how episodes or sequences conclude, setting the stage for a new learning experience or iteration.

A terminal state, as the name suggests, is a definitive endpoint in the trajectory of an agent within a particular episode. In essence, when an agent reaches a terminal state, it signifies the termination of the current episode, after which the agent is typically reset to a starting position or state, ready to embark on a new episode, thus facilitating the learning process over multiple iterations.

Objective Achievement: Terminal states often correspond to the achievement or failure of the agent's primary objective. In the drone's context, successfully navigating through a gate could be seen as achieving the objective.

Safety and Failures: In scenarios where the agent might incur damage or jeopardize its environment, terminal states act as a safeguard. By recognizing hazardous situations as terminal, we can prevent potential mishaps. In the drone scenario, a collision might lead to a terminal state, ensuring the drone's safety and that of its surroundings.

Efficient Learning: Continuously letting an agent wander without purpose is not productive. Terminal states, especially in cases like stagnation, ensure that episodes that are not contributing to learning are terminated, paving the way for a fresh start.

Resource Management: In practical implementations, computational resources, energy, and time are finite. Using terminal states effectively ensures that these resources are utilized optimally.

Collision as a Terminal State

The dynamic nature of drone navigation, especially in intricate environments, necessitates an acute focus on safety. This becomes even more pertinent, when learning algorithms are introduced that have the potential to explore actions leading to unsafe maneuvers. The classification of certain states as "unsafe" or indicative of a "collision" thus acts as a guiding light, ensuring the drone's operations remain within safety boundaries.

Collision: A collision is deemed to have occurred when the drone's sensors register a distance less than a predetermined safety threshold

$$d_{\text{sensor}} \leq d_{\text{threshold}} \Rightarrow \text{Collision State}$$

Where:

$d_{\textbf{sensor}}$ is the distance reading from the drone's sensors.

$d_{\textbf{threshold}}$ is the predetermined safety threshold.

Importance

The emphasis on safety is multi-faceted. Drones, especially sophisticated ones, equipped with various sensors and computing modules, represent a significant investment. Protecting this hardware from potential damages arising from collisions is paramount. As the learning agent explores the action space to derive an optimal policy, this exploration must not lead to risky maneuvers. Defining and respecting safety boundaries ensures this.

Practical Implications

Defining and adhering to safety thresholds has several practical implications. Episodes that end prematurely due to collisions can be seen as "failed attempts", which are

crucial for the learning algorithm. These episodes provide negative reinforcement, guiding the agent to better policies over time. So, by understanding the consequences of unsafe actions, the learning agent can steer its exploration towards more promising and safe regions of the action space.

In summary, safety and collision-centric terminal states are pivotal in guiding the learning process, ensuring efficient and risk-averse navigation policies. The mathematical underpinnings provide a quantifiable measure, offering clarity and precision to the agent's operational boundaries.

Goal Achievement as a Terminal State

In the context of navigation, drones are often seen as vehicles with a destination. For reconnaissance, delivery, or any other application, reaching a specified target (in this case, a gate) is typically the primary objective. By defining a terminal state associated with goal achievement, we structure the drone's learning journey, culminating in successful navigation. This understanding is crucial for both simulation and real-world applications.

A gate can be considered "reached" when the center of the drone lies within a specified distance d_{goal} from the center of the gate. For success, the distance between the drone and the gate has to satisfy the following condition:

$$\sqrt{(x_{\text{drone}} - x_{\text{gate}})^2 + (y_{\text{drone}} - y_{\text{gate}})^2 + (z_{\text{drone}} - z_{\text{gate}})^2} \leq d_{\text{goal}}$$

Where:

$(x_{\mathbf{drone}}, y_{\mathbf{drone}}, z_{\mathbf{drone}})$ are the drone's coordinates.

$(x_{\mathbf{gate}}, y_{\mathbf{gate}}, z_{\mathbf{gate}})$ are the gate's center coordinates.

Importance

Identifying and reinforcing goal achievement serves multiple purposes. The essence of drone navigation in this context is to reach the gate. By identifying and rewarding this action, we reinforce the primary objective of the drone's task. The frequency and efficiency with which the drone reaches its goal can serve as a key performance indicator, shedding light on the efficiency of the learning process.

Feedback Loop

The reward mechanism and the terminal states in reinforcement learning form a crucial feedback loop. This loop is fundamental for the drone's learning algorithm to understand the ramifications of its actions and refine its policy accordingly. Successfully navigating to the gate and achieving the goal provides a significant positive reward. This not only marks the successful end of an episode, but also serves as a strong incentive for the agent to replicate such successful actions in the future. With each successful navigation, the learning algorithm updates its policy, placing higher probabilities on action sequences that lead to goal achievement. Over time, this iterative feedback loop steers the drone towards more efficient and reliable navigation paths. In cases where the drone receives partial rewards for nearing the gate, but does not successfully navigate through it, the agent learns to modify its approach, avoiding paths and maneuvers that do not culminate in goal achievement.

Concluding, the definition of goal achievement as a terminal state, not only aligns the drone's actions with the primary navigation objective, but also ensures that the learning algorithm is constantly steered toward optimal performance. The mathematical criterion ensures precision, while the reward feedback loop ensures adaptability and iterative improvement.

Time Expired as a Terminal State

In reinforcement learning, especially in environments with specific objectives like drone navigation, a terminal state based on time expiration is crucial. This terminal state

activates when the drone fails to reach its goal or avoid collisions within a predefined number of steps. It ensures that the agent's learning process is efficient and that episodes do not extend unnecessarily, which could waste computational resources and time.

Criteria The episode ends if the drone does not reach the target gate or encounter a collision within N steps:

$$\text{if } k \geq N, \text{ then terminate episode}$$

Where:

N is the maximum number of steps allowed per episode.

k is the current step count within the episode.

Importance

This terminal state ensures that the training process remains focused and time-efficient. By limiting the duration of each episode, we prevent the agent from becoming stuck in unproductive states, such as endlessly exploring or failing to make progress toward the goal. This mechanism helps concentrate learning efforts on more critical scenarios, like reaching the gate or effectively avoiding obstacles.

Benefits

Efficiency: Time-based termination prevents the waste of computational resources by limiting the duration of non-productive episodes.

Focused Learning: Encourages the agent to develop strategies for reaching the goal or avoiding collisions within the time frame, enhancing the effectiveness of the training.

Policy Improvement: Provides clear feedback on the agent's performance, helping to refine policies that are more time-sensitive and goal-oriented.

In summary, incorporating a time-expired terminal state is vital for maintaining an efficient and effective learning process. It helps streamline the training, ensuring

that the agent focuses on achieving objectives within a reasonable time frame, thereby optimizing resource use and enhancing overall performance.

Operational Limits as Terminal States

Operational limits, often called constraints arising from the physical and environmental conditions where drones operate, can be critical determinants in establishing terminal states in reinforcement learning for drone navigation. However, it is essential to underscore that in the context of this project, which operates within a simulation environment, these operational limits do not apply, since the simulation does not incorporate battery depletion or operational boundary conditions. Despite their non-application in the present scope, they are delineated here for potential future applicability and to emphasize the importance of real-world constraints in drone operations.

Battery Constraints

In real-world applications, the battery life of a drone is a critical factor in determining its operational time, setting a terminal state conditioned by the battery life. The episode ends, when the drone's battery level reaches a predetermined critically low threshold.

Let $B_{\min}$ represent the minimum battery level below which the episode terminates. The episode ends, when the drone's battery level reaches a predetermined critically low threshold, expressed as:

$$\text{if } B_t < B_{\min}, \text{ then terminate episode}$$

where B_t is the battery level at time t.

Boundary Conditions

When drones are utilized in real environments, they are often restricted to a specific operational area to ensure safety and compliance with regulatory norms. The episode ends, when the drone moves outside a predefined operational zone delineated by spatial coordinates.

Let the operational zone be defined by a set of boundary coordinates $(x_{\min}, x_{\max}, y_{\min}, y_{\max}, z_{\min}, z_{\max})$.

The episode terminates if the drone exits the predefined operational zone, as expressed by the following conditions:

$$x_{\text{drone}} < x_{\min} \text{ or}$$

$$x_{\text{drone}} > x_{\max} \text{ or}$$

$$y_{\text{drone}} < y_{\min} \text{ or}$$

$$y_{\text{drone}} > y_{\max} \text{ or}$$

$$z_{\text{drone}} < z_{\min} \text{ or}$$

$$z_{\text{drone}} > z_{\max},$$

if any of these conditions are true, then terminate the episode. The drone's coordinates at time t are denoted as $(x_{\text{drone}}, y_{\text{drone}}, z_{\text{drone}})$.

Importance

Despite the non-applicability in the simulated environment, understanding these operational limits bears importance, because it enhances the transition of the learned behavior from simulation to real-world scenarios, by accounting for physical and environmental constraints. Moreover, it ensures the drone operates within a safe and regulated framework, preventing mishaps due to battery depletion or straying out of the operational boundaries.

In conclusion, while operational limits as terminal states hold no bearing in the current simulated setup, they remain a vital area of concern for real-world implementations, underscoring the necessity to integrate these aspects in future renditions of the project.

4.4 Evaluation Metrics

Evaluation metrics are the bedrock upon which the effectiveness and efficiency of Reinforcement Learning algorithms, such as the DQN employed in this project, can be

gauged. For our drone navigation task, metrics are particularly pivotal, translating the nuanced workings of the algorithm into quantifiable measures, thus allowing for a more systematic assessment of performance, comparison across iterations, and iterative refinement.

4.4.1 Success Rate

Definition

The success rate stands as a paramount metric in our evaluation arsenal. At its core, the success rate captures the proportion of episodes, where the drone successfully navigates through the designated gate, without any collisions or other terminal state occurrences. Mathematically, it can be represented as:

$$\text{Success Rate} = \frac{\text{Number of Successful Navigations}}{\text{Total Number of Episodes}} \times 100\%$$

Relevance

Given the primary objective of the drone navigation task—to successfully transit through gates without collisions—the success rate emerges as an intuitive and direct measure of the effectiveness of the training and the RL algorithm. A high success rate indicates that the drone is proficiently avoiding obstacles, making optimal navigation choices, and consistently reaching its target, which, in this context, is the gate. Conversely, a lower success rate might signal the need for further refinement in the algorithm, adjustment in the training regimen, or tweaks in the model parameters.

Applications

The success rate can be utilized in various facets of the project:

Training Evaluation: Monitoring the success rate across different training epochs can provide insights into the learning curve of the model, indicating the epochs where the model starts to converge or if there are oscillations in performance.

Algorithm Comparison: If multiple versions or variants of algorithms are deployed in the future, the success rate can serve as a common ground for comparison, helping select the most effective one.

Hyperparameter Tuning: One can optimize the model to achieve the highest possible success rate by assessing the success rate under varied hyperparameter configurations.

In summary, the success rate crystallizes the efficacy of the drone navigation system into a single, comprehensible metric, acting as both a beacon for current performance and a compass for future improvements.

4.4.2 Computational Efficiency

Computational efficiency pertains to the ability of the algorithm—in this case, DQN—to provide optimal or near-optimal solutions within an acceptable time frame and using minimal computational resources. This encompasses both the time complexity (how the algorithm's running time scales with the input's size) and the space complexity (how the memory usage grows with input size). Furthermore, drones typically operate on embedded systems with limited computational resources. Thus, an efficient algorithm is not just a desire, but a necessity.

Key Factors Influencing Computational Efficiency in DQN

Network Architecture: The depth and breadth of the neural network used in DQN can significantly influence its computational demands. Deeper networks might capture complex features, but can be more resource-intensive.

Batch Size: During training, the batch size used for updates can affect training speed and memory usage. Larger batches might stabilize training, but can be slower and more memory-demanding.

Experience Replay: DQN employs experience replay, storing and sampling past experiences. The size of this replay buffer and the frequency of its utilization can have memory and computational implications.

Exploration Strategy: Methods like epsilon-greedy, which influence how often the agent explores versus exploits, can impact the number of computations, if the exploration involves complex computations or simulations.

Evaluation of DQN Algorithm

In the context of this project, the DQN algorithm has been designed to keep computational efficiency at the forefront. While it is adept at learning intricate navigation strategies, careful considerations were made to ensure it operates seamlessly within the constraints of drone-embedded systems. Benchmarks in various environments, ranging from the simpler block world to more complex ones, demonstrated that the DQN consistently delivered results within acceptable time frames, ensuring timely responses critical for drone operations.

In essence, computational efficiency is a testament to the algorithm's practical viability. A highly effective, yet computationally burdensome, algorithm might be of limited real-world utility. On the other hand, as attempted in this project, a balanced approach that marries efficacy with efficiency lays the foundation for practical and scalable real-world deployments.

4.5 Visual Gate Detection

4.5.1 Selection of Detector System

Besides developing the obstacle avoidance system, the correct detection of the gates is essential to the project. An initial concern was whether to create a new detection system or utilize an existing one [23, 9]. Following a thorough review, an existing gate detection system was chosen, which segments the flyable area of an Unmanned Aerial Vehicle in real-time based on the captured image frame. This system, developed in a previous master's thesis at the Technical University of Crete, was selected for two main reasons [9]. Initially, the system achieves one of the highest, if not the highest, Intersection over Union (IoU) accuracy rate compared to the rest of the market. The percentage is 91.2%, based on the official heroX team's Leaderboard testing dataset. In addition, the system has been tested in many different scenarios of the Gazebo

simulator, which is a simulator used in the present dissertation. As for the technical part, the detector system combines a custom convolutional neural network and an attached deconvolutional network. The model consists of 16 weight layers, of which 9 belong to the convolutional network. The filters it uses have a kernel size of 3 by 3 and are fully connected to the 3 final layers of the network.

4.5.2 Boost Gate Detection Accuracy

Although the selected system achieves satisfactory detection rates, it was deemed appropriate to improve it further, as it would positively impact the evolving navigation system. However, it was considered that the change of the network architecture would be beyond the scope of the work. Thus, only some peripheral areas of the system that could be improved were considered. Research has shown that requiring more training data can generally ensure better system performance. However, no fixed rule can be applied here, and case by case, it could be different. The number of training data needed depends on many different aspects, such as the nature of the problem, the number of features, and the complexity of the network architecture. For this reason, it was empirically proposed to use more data for the training of the model, as the AlphaPilot training dataset contains roughly 9,300 images. Therefore, ways to enrich the dataset were sought. The first solution was to search for available public datasets; however, as far as is known, no other data is currently provided other than those of the tender. Therefore, an effort was made to create and utilize an improved dataset for better detection results. Thus, data of wide variety and extensive content were collected for the model's training, including images of the gates from different angles, different lighting conditions (brightness or contrast), and different image quality, as some contain motion blur (see Figure 4.13).

Figure 4.13: Alpha Pilot Training Gate - Essential for Autonomous Drone Racing Dataset

4.5.3 Dataset Enrichment

The data collection was carried out with the FlightGoggles Simulator, in order to take advantage of the fact that the simulated quadcopter is equipped with an infrared beacon sensor and the eleven gates with infrared beacons at each corner.

Thus, the drone moves freely within the space of the simulator and, through the appropriate code, can at any time manually take photos stored locally. At the same time, information from each image is stored in a .json file, which has the same format as that provided for the contest dataset. If within a photo all four inner corners of the gate are visible, regardless of whether the flyable region is partially obstructed, then the data of the infrared beacon sensor is utilized. This data refers to the 2D coordinate system of the image plane recorded by the UAV camera and essentially represents the x-y pixel coordinates of the corners. The same happens in the case that all the corners of two or more gates are detected in an image. The coordinates are then recorded in the JSON file and constitute the Ground-Truth Labels for each image provided. So eventually, the JSON file is structured so that it contains an object that corresponds to the label of each image that has been captured. For each gate in an image, there is a set of values corresponding to the four (x, y) coordinates of the corners of the quadrilateral, while if there is no gate in the image, the values are empty. The four coordinates of the four corners of the quadrilateral are in a clockwise direction, starting from the upper left corner of the gate. Indicatively, the '.json' file has the following format:

```
{
    "IMG_0001.png": [[320, 81, 393, 94, 393, 160, 322, 152]],
    "IMG_0002.png": [[130, 177, 212, 214, 208, 313, 126, 299]],
    "IMG_0003.png": [[581, 177, 669, 184, 669, 280, 579, 277]],
    "IMG_0004.png": [[275, 183, 362, 210, 358, 300, 271, 288]]
}
```


The images have a size of 1024×768 pixels. Eventually, a dataset of a few thousand photos was created from the environment of the FlightGoggles simulator, and it has a greater variety than what was initially given. The photos contain the gates from different heights, angles, lighting conditions, and quality to better train the network and achieve even higher percentages of detection accuracy.

Figure 4.14 shows a grid of sample photos from the dataset, showcasing the variety in perspectives and conditions.

IMG_0001 IMG_0002

IMG_0003 IMG_0004

Figure 4.14: Sample images from the enriched dataset showing gates from different heights, angles, lighting conditions, and quality.

5

Experiments, Results, and Observations

Contents

This chapter presents the results of experiments conducted to evaluate the UAV's navigation capabilities across various simulated environments within the scope of this thesis. It assesses the drone's adaptability and learning progress in dynamic settings of increasing complexity, emphasizing the impact of environmental challenges and training configurations on performance. These insights are crucial for understanding the effectiveness of the applied algorithm in enhancing autonomous drone navigation.

5.1 Experimental Setup

5.1.1 Overview

The objective was to simulate real-world complexities through various virtual environments, each presenting unique challenges and obstacles. These simulations aim to train the UAV to adapt and navigate through unfamiliar settings, thereby enhancing its autonomous navigation skills. The rationale behind selecting dynamic environments is to prevent overfitting to static conditions, ensuring the UAV's algorithm remains flexible and capable of generalizing its navigation capabilities across diverse scenarios.

5.1.2 Experimental Environments

The cornerstone of the experimental setup is the deployment of dynamic environments, each meticulously designed to simulate the intricacies and unpredictability of real-world navigation. These environments act as the battlegrounds, where the UAV's ability to adapt, learn, and navigate through unknown terrains is rigorously tested. The specifics of each environment, highlighting the unique challenges they present, are detailed below.

Block World

Block World is the inaugural environment in our series, characterized by its assortment of geometric shapes scattered across the landscape. This world is a conglomerate of four distinct shape types, each manifesting in varying sizes and dimensions. The dynamic essence of Block World is captured through its procedural generation of each session, where shapes are randomly assigned attributes, such as color, size, position, and orientation. This randomness ensures no two simulation runs are identical, compelling the UAV to confront and adapt to a new set of obstacles with every iteration. The primary objective within Block World is to foster an adaptive learning process, encouraging the UAV to develop a versatile navigation strategy that can accommodate a wide range of spatial configurations.

Sugar Cane World

Transitioning from the geometric diversity of Block World, Sugar Cane World introduces the UAV to a more specialized scenario. This environment is dominated by vertical cylindrical obstacles, reminiscent of a densely packed sugar cane field. These obstacles maintain constant dimensions and a fixed orientation, but are randomly placed throughout the environment in each new session. The challenge here pivots towards precision navigation, as the UAV is required to thread its way through the narrow gaps between these vertical barriers. Sugar Cane World is instrumental in honing the UAV's ability to perform delicate maneuvers, enhancing its dexterity in avoiding collisions, and maintaining a safe trajectory through tight spaces.

Slat Fence World

The final environment, Slat Fence World, builds upon the complexity introduced in Sugar Cane World, but with a significant twist. Here, the vertical obstacles of the previous world are replaced with horizontal cylinders, creating a lattice of barriers that the UAV must navigate. Despite the constant dimensions and predetermined orientation of these cylinders, their random placement across sessions introduces a new dimension of challenge. This environment is specifically designed to test the UAV's vertical maneuvering capabilities, requiring it to demonstrate agility and precision in navigating over or under the slat fences. Slat Fence World is crucial for developing the UAV's competency in handling scenarios that demand complex three-dimensional navigation strategies.

Each of these dynamic environments plays a pivotal role in our experimental framework, providing a comprehensive platform for testing and improving the UAV's autonomous navigation capabilities. By exposing the UAV to a variety of obstacles and configurations, we aim to cultivate a navigation model that is not only effective in specific scenarios, but is robust and adaptable enough to tackle the unpredictable nature of real-world environments.

5.1.3 Training Configurations

The comprehensive training regimen for the UAV across the dynamic environments was meticulously designed to span various levels of difficulty and durations, aiming to rigorously assess and enhance the drone's navigational capabilities. These configurations are pivotal for understanding the adaptability and learning progression of the UAV in responding to different challenges and complexities presented by each simulated world. Here, we delve into the specific training configurations employed in the experiments, delineated by the levels of difficulty and the training durations.

Levels of Difficulty

The simulated environments were structured into three distinct levels of difficulty, namely Easy, Medium, and Hard, to systematically challenge and evaluate the UAV's ability to navigate and adapt. The classification of these levels was primarily based on the number of obstacles present within the environment:

- **Easy:** This level was designed as an introductory phase for the UAV, featuring no obstacles (0 obstacles) within the simulation. It served to acclimate the UAV to the basic mechanics of flight and navigation within the simulated worlds, without the immediate challenge of obstacle avoidance.
- **Medium:** Representing a moderate increase in complexity, this level introduced a total of 6 obstacles into the simulation. The inclusion of these obstacles aimed to test the UAV's initial collision avoidance capabilities and its ability to make more complex navigational decisions.
- **Hard:** The highest level of difficulty presented the UAV with a dense array of 12 obstacles, significantly elevating the navigational challenge. This level was designed to push the UAV's limits in dynamic pathfinding, requiring advanced strategies for maneuvering through tightly spaced obstacles and adapting to rapidly changing environments.

Training Durations

To further explore the impact of training intensity on the UAV's learning outcomes, two distinct training durations were implemented:

- **Large:** A substantial training period of 8 hours was designated as the Large duration. This period allowed for a deep, but manageable, exploration of the environments, providing the UAV ample opportunity to learn from its interactions and gradually improve its navigation strategies.
- **Marathon:** Extending significantly beyond the Large duration, the Marathon training sessions lasted for 32 hours. This intensive training regime was aimed at deeply embedding the learned navigation skills, allowing the UAV to refine its strategies and behaviors through prolonged exposure to the simulation challenges.

Configurations Across Worlds

Given the unique characteristics and objectives of each simulated world, training was not uniformly applied across all difficulty levels and durations for each environment. Instead, strategic choices were made to optimize learning outcomes and resource utilization:

- **Block World:** Training encompassed a broad spectrum of configurations, with sessions conducted at all levels of difficulty (Easy, Medium, Hard) and both durations (Large, Marathon), except for the Easy level, which was limited to Large duration. This approach facilitated a comprehensive exploration of the UAV's learning capacity in a geometrically diverse setting.
- **Slat Fence World and Sugar Cane World:** For these environments, training focused on the more challenging Medium and Hard difficulty levels, exclusively employing the Marathon duration. This decision was based on the anticipation that the increased complexity and obstacle density in these worlds would benefit from extended training periods, enabling the UAV to better adapt to the intricate navigation tasks presented.

These configurations were carefully selected to balance the UAV's exposure to a variety of learning scenarios against the practical constraints of time and computing resources. The aim was to cultivate a robust and versatile navigational model, capable of adeptly handling a wide range of real-world autonomous flight challenges.

5.2 Results

5.2.1 Block World Results

To provide a clearer understanding of the UAV's navigation capabilities, we present an example of the drone's trajectory in the Block World under the Hard difficulty level. This visual example, shown in Figure 5.1, illustrates the path taken by the UAV from start to the goal (gate), demonstrating its ability to navigate through complex environments.

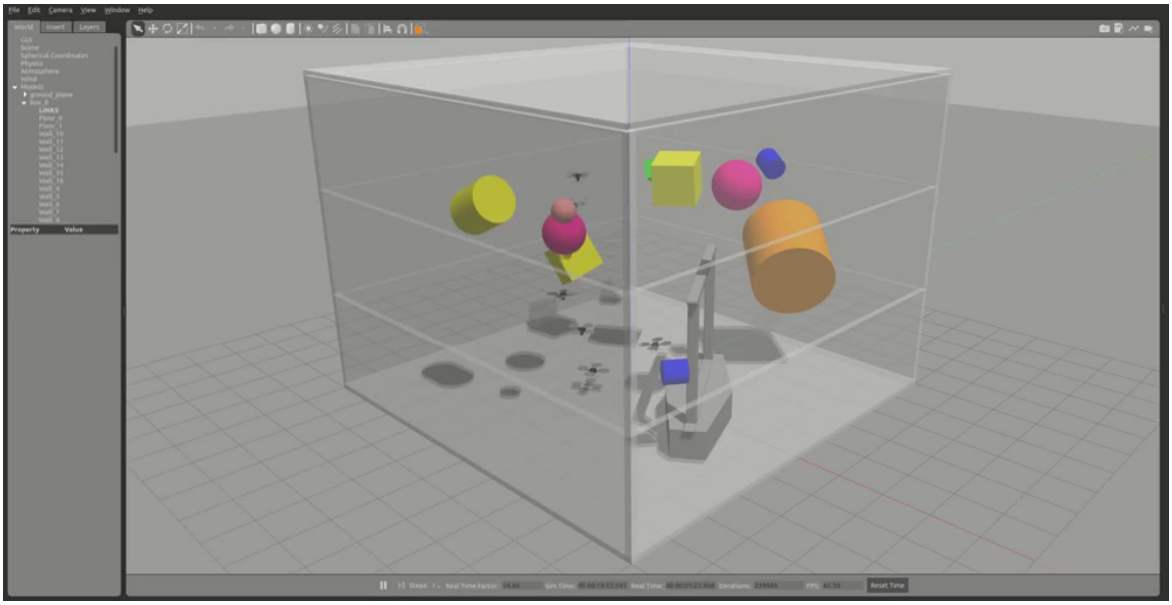

Figure 5.1: Example of UAV trajectory in Block World under Hard difficulty level.

The Block World served as the initial testing ground for evaluating the UAV's ability to navigate through environments populated with geometric obstacles. This section presents and analyzes the outcomes of the UAV's training sessions within the Block World, scrutinizing the effects of varying difficulty levels and training durations on its performance.

Training Outcomes Across Difficulty Levels

- **Easy (0 Obstacles, Large Duration):** As anticipated, the UAV showcased a relatively high success rate in navigating the Block World under the Easy difficulty setting. Given the absence of obstacles, basic flight control, and endurance are the primary challenge. The UAV's performance metrics indicated rapid adaptation to the environment, achieving good success rates in completing the navigation tasks.

- **Medium (6 Obstacles, Large, and Marathon Durations):** Introduction of obstacles under the Medium difficulty presented a notable challenge, requiring the UAV to develop and refine its obstacle avoidance strategies. While the transition to Medium difficulty resulted in an initial dip in success rates, the UAV exhibited improvement over time. A comparative analysis between the Large and Marathon training durations revealed enhanced performance with extended training. The Marathon duration enabled the UAV to engage in more iterative learning cycles, resulting in a more nuanced understanding of obstacle navigation and a higher success rate.

- **Hard (12 Obstacles, Large and Marathon Durations):** The Hard difficulty setting, characterized by a dense array of obstacles, tested the limits of the UAV's navigational capabilities. The initial performance metrics under this setting indicated a substantial challenge, with lower success rates compared to the Medium difficulty. However, akin to the Medium difficulty level, an extension from Large to Marathon duration yielded improvements in the UAV's performance. The intensive training facilitated a deeper learning experience, allowing the UAV to better anticipate and navigate through the complex obstacle configurations, though the success rate remained below that of the less challenging environments.

Comparison within Block World

The graph in Figure 5.2 illustrates the UAV's learning efficiency within the Block World during Large training sessions. The Easy difficulty level reveals the highest gate

found rate, underscoring the UAV's ability to quickly master navigation and objective identification, when free from obstacles. The Medium difficulty shows substantial learning with the presence of obstacles, while the Hard difficulty underscores the challenge that a greater number of obstacles pose to the UAV's learning trajectory.

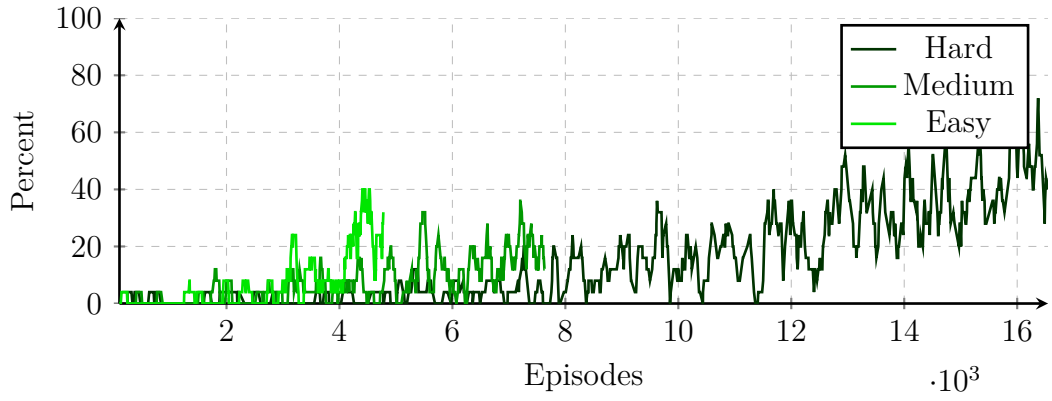

Figure 5.2: Comparison of Gate Found Rates in the Block World Across Easy, Medium, and Hard Difficulty Levels During Large Training Sessions.

During Marathon sessions in the Block World (Figure 5.3), both Medium and Hard difficulties show learning improvements, yet Medium difficulty consistently outperforms Hard. This trend underscores the impact of obstacle density on learning efficacy, with fewer obstacles enabling more efficient navigation and learning.

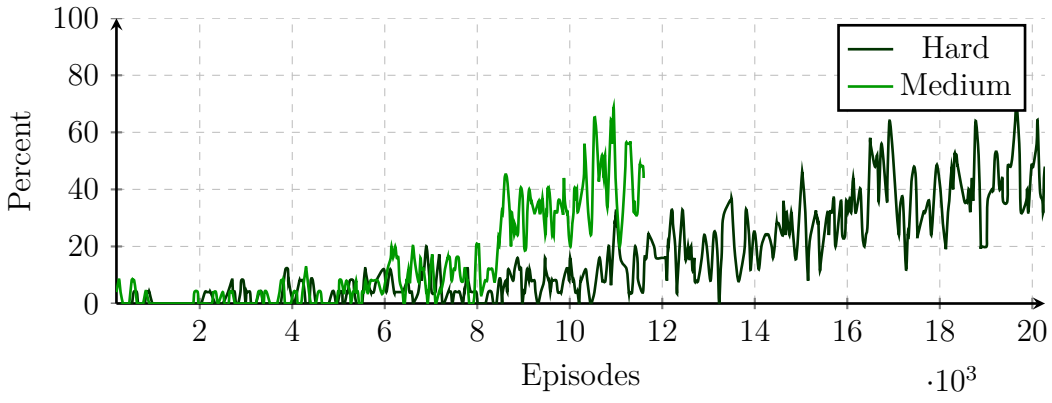

Figure 5.3: Comparison of Gate Found Rates in the Block World Across Medium and Hard Difficulty Levels During Marathon Training Sessions.

In Large duration sessions within the Block World (Figure 5.4), collision rates decreased across all difficulty levels, highlighting the UAV's growing proficiency

in obstacle avoidance. Notably, the Easy difficulty saw the lowest collision rates, reflecting the minimal navigational challenges it presents. The trend illustrates a direct correlation between obstacle density and collision rates, emphasizing the UAV's learning adaptability in increasingly complex environments.

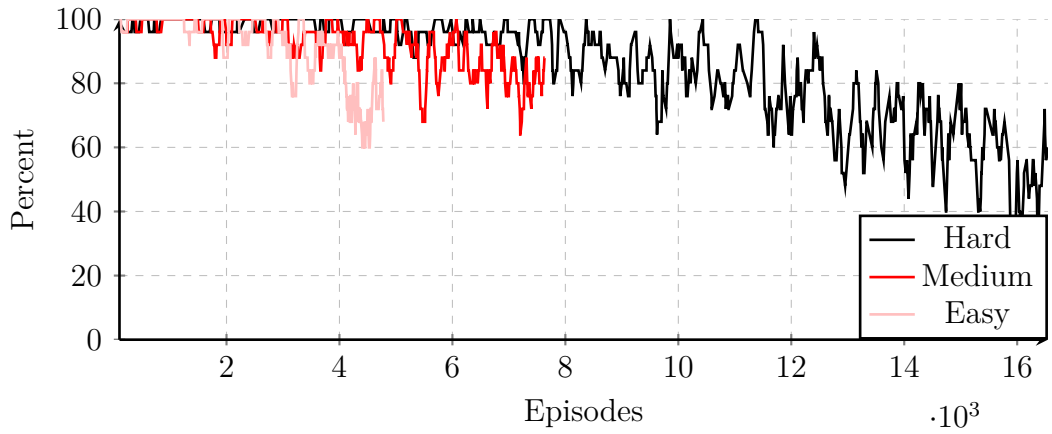

Figure 5.4: Comparison of Collision Rates in the Block World Across Easy, Medium, and Hard Difficulty Levels During Large Training Sessions.

During Marathon duration training (Figure 5.5), both Medium and Hard difficulties witnessed a reduction in collision rates, with Medium difficulty showcasing a more pronounced decrease. This outcome suggests that prolonged exposure to complex environments enhances the UAV's obstacle avoidance capabilities, particularly in scenarios with moderate obstacle density. The data reiterates the importance of training duration and environment complexity in refining the UAV's navigational strategies.

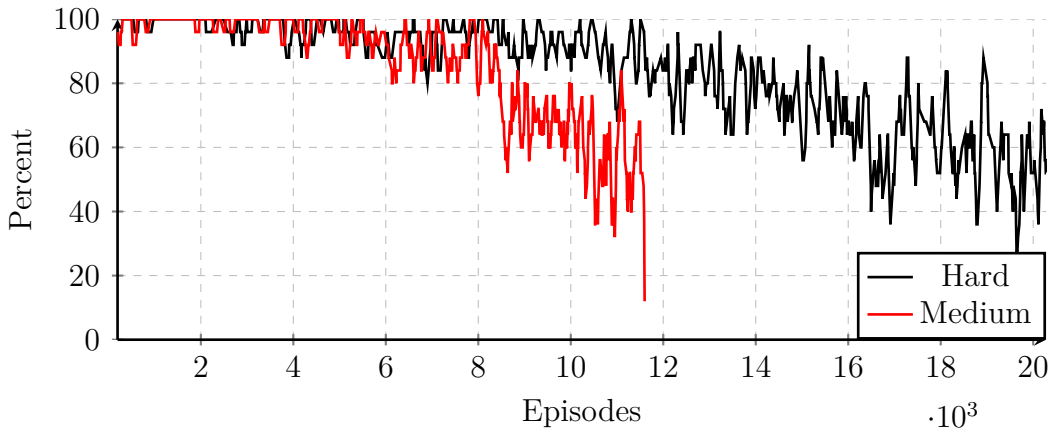

Figure 5.5: Comparison of Collision Rates in the Block World Across Medium and Hard Difficulty Levels During Marathon Training Sessions.

Time expired rates are slightly higher in the Easy difficulty during Large training sessions, indicating that the absence of obstacles allows the UAV to roam more freely, occasionally leading to longer task completion times. In the Easy difficulty, the UAV has more freedom to explore, which can result in episodes lasting longer before the gate is found or the episode terminates due to time expiration. This contrasts with Medium and Hard difficulties, where the presence of obstacles provides more direct feedback to the UAV, leading to more efficient task completion within the time limits due to quicker episode termination either by finding the gate or colliding with obstacles (Figure 5.6).

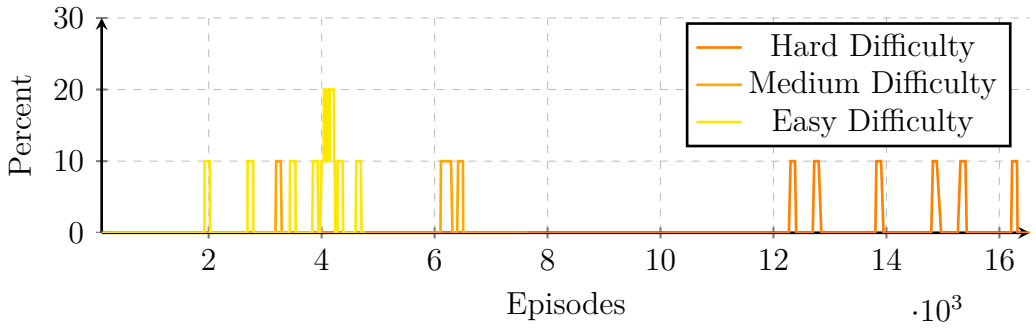

Figure 5.6: Comparison of Time Expired Rates in the Block World Across Easy, Medium, and Hard Difficulty Levels During Large Training Sessions.

In Marathon sessions within the Block World, time expirations are rare for Medium and Hard difficulties, mainly because collisions with obstacles end episodes before time can expire. The presence of obstacles prompts quicker episode termination, making time expiration less of a concern (Figure 5.7).

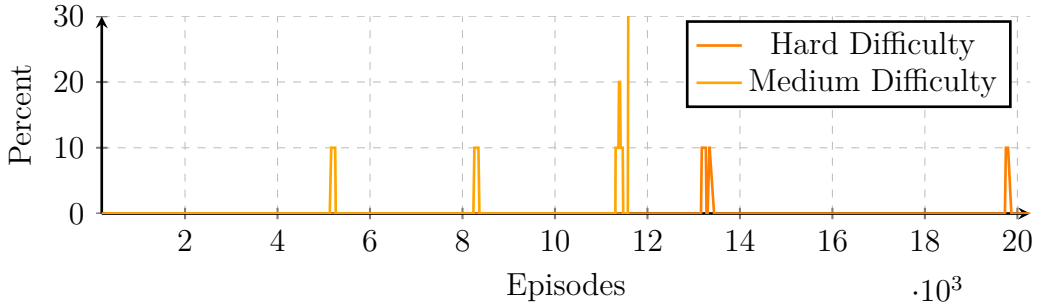

Figure 5.7: Comparison of Time Expired Rates in the Block World Across Medium and Hard Difficulty Levels During Marathon Training Sessions.

In the Block World during Large training sessions, the UAV's total reward accumulation provides insight into its overall performance and learning efficiency. Notably, the Easy level yields the highest rewards, indicative of the UAV's proficiency in unobstructed environments. The progression to Medium and Hard levels sees a decrease in total rewards, reflecting the increased complexity and navigational challenges presented by obstacles (Figure 5.8).

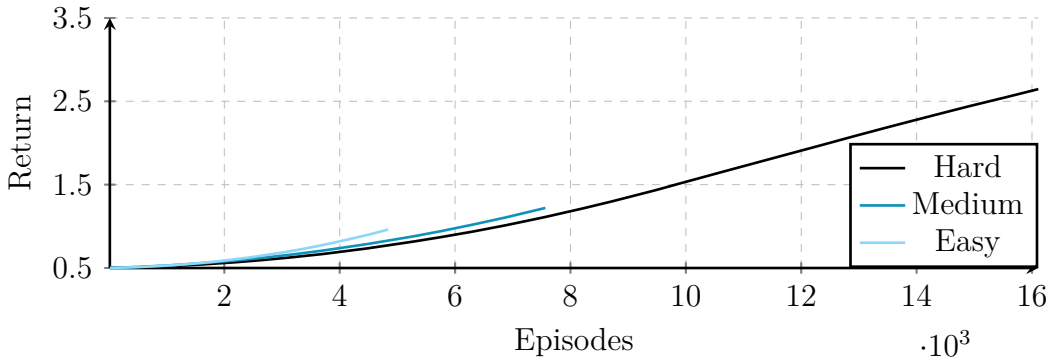

Figure 5.8: Comparison of General Total Rewards in the Block World Across Medium and Hard Difficulty Levels During Marathon Training Sessions.

The Marathon sessions in the Block World reveal the UAV's adaptability and learning growth over extended periods. The Medium difficulty demonstrates a higher total reward compared to the Hard difficulty, suggesting that while the UAV faces challenges in more complex environments, prolonged training allows for significant learning and performance improvements. The gap in total rewards between the Medium and Hard levels underscores the impact of environmental complexity on the UAV's learning progress (Figure 5.9).

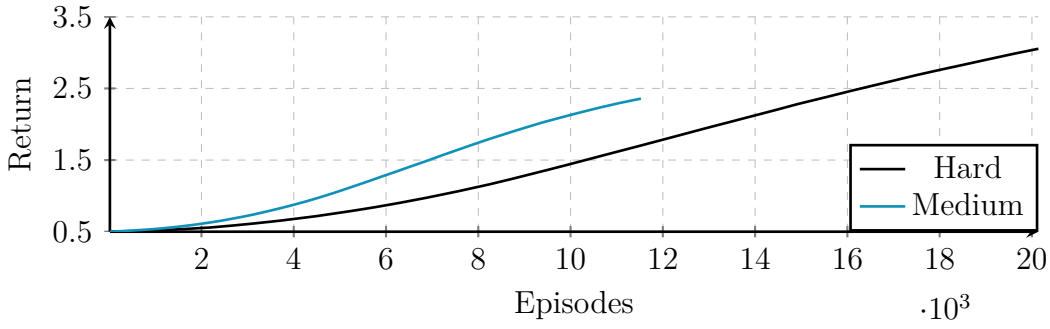

Figure 5.9: Comparison of General Total Rewards in the Block World Across Medium and Hard Difficulty Levels During Marathon Training Sessions.

Performance Overview

The analysis within the Block World underscores the critical impact of both difficulty level and training duration on the UAV's learning and performance. A clear trend emerged from the data: as the difficulty level increased, the initial success rates decreased, highlighting the added complexity and challenge in navigating through more densely populated obstacle environments. However, extending the training duration from Large to Marathon consistently improved performance across difficulty levels, emphasizing the value of prolonged exposure and iterative learning in enhancing the UAV's navigational abilities.

Notably, the improvement in success rates from Large to Marathon duration was more pronounced at higher difficulty levels. This observation suggests that, while longer training durations contribute to improved performance across all settings, their impact is particularly significant, when the UAV is confronted with more complex navigation tasks. This trend reinforces the notion that deep, extended learning experiences are crucial for mastering more challenging autonomous navigation scenarios.

In summary, the Block World results reveal a direct correlation between the UAV's performance and both the difficulty of the environment and the length of training. These findings highlight the importance of tailored training approaches, suggesting that optimal learning outcomes are achieved by adjusting the training duration in response to the complexity of the navigation task at hand.

5.2.2 Sugar Cane World Results

This section delves into the outcomes of the UAV's training sessions within the Sugar Cane World, focusing exclusively on the Medium and Hard difficulty levels, both subjected to Marathon training durations. To provide a concrete example of the UAV's navigation capabilities, Figure 5.10 illustrates a typical trajectory taken by the UAV in the Hard difficulty level. This trajectory exemplifies the challenges and navigation strategies employed by the UAV in a densely packed vertical obstacle environment.

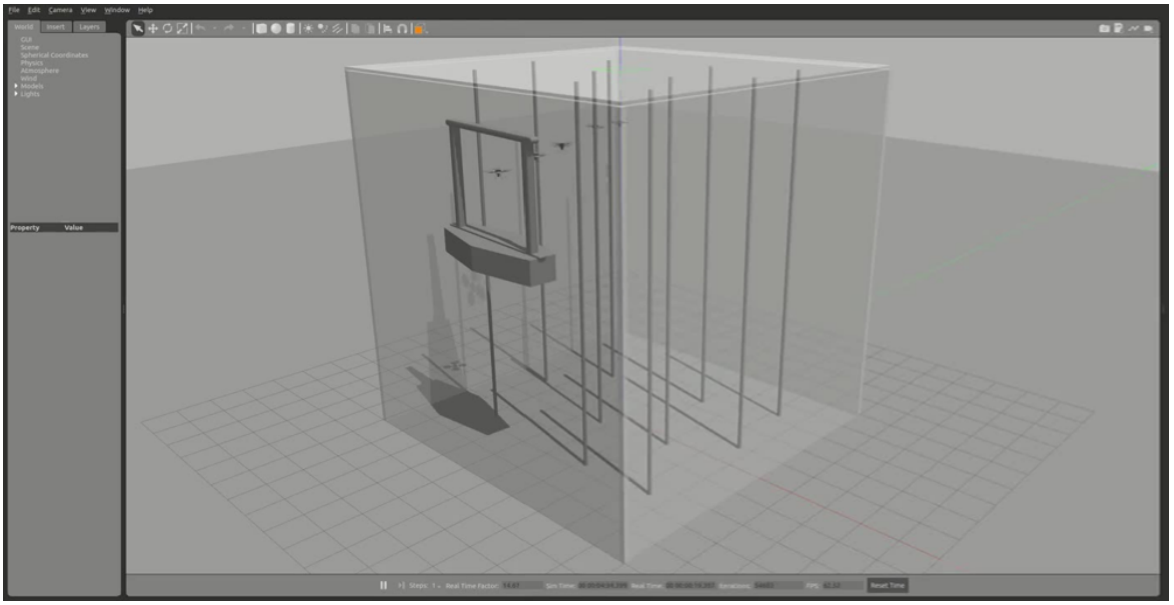

Figure 5.10: Example of UAV trajectory in Sugar Cane World under Hard difficulty level.

Training Outcomes Across Difficulty Levels

- **Medium (6 Obstacles, Marathon Duration):** Training in the Sugar Cane World at Medium difficulty level with Marathon training sessions revealed a discernible, yet modest, improvement in the UAV's navigational capabilities. The dense arrangement of vertical obstacles necessitated precise maneuvering, a skill that proved challenging, yet gradually attainable, over the course of the training. The results indicated that, while the UAV was capable of learning and improving, the inherent complexity of navigating through narrowly spaced obstacles imposed a significant challenge.

- **Hard (12 Obstacles, Marathon Duration):** The Hard difficulty setting pushed the UAV's capabilities to their limits, with the dense obstacle environment significantly impacting success rates. The performance metrics under this setting highlighted the substantial difficulty the UAV faced in adapting to the complex navigation required. Although there was a slight improvement over the training period, the overall success rate remained comparatively low, underscoring the challenges of mastering navigation in such a constrained and obstacle-rich environment.

Comparison within Sugar Cane World

The graph (Figure 5.11) illustrates that the UAV achieved a higher gate found rate in the Medium difficulty compared to the Hard difficulty within the Sugar Cane World. This suggests that the UAV's navigation system responds better to environments with moderately spaced obstacles.

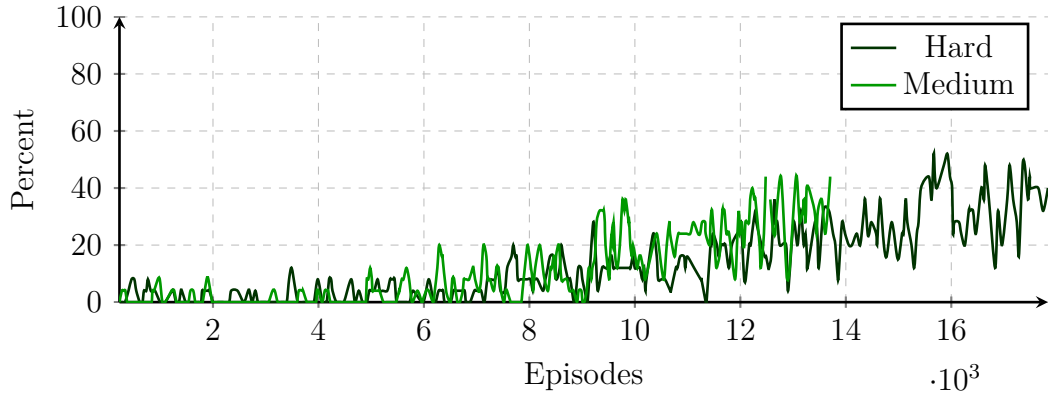

Figure 5.11: Comparison of Gate Found Rates in the Sugar Cane World Across Medium and Hard Difficulty Levels During Marathon Training Sessions.

The collision rates, as presented in the graph (Figure 5.12), are notably higher in the Hard difficulty setting. This demonstrates that the UAV struggles more with collision avoidance, as the complexity and density of vertical obstacles increase.

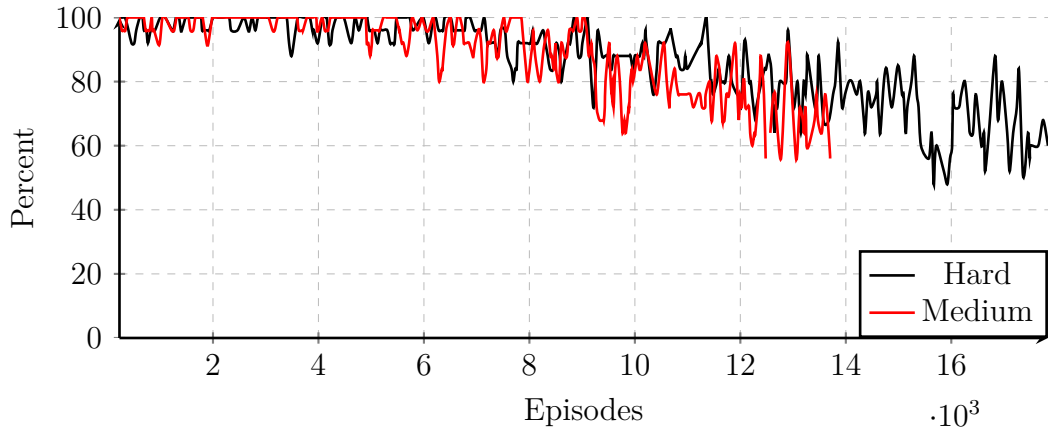

Figure 5.12: Comparison of Collision Rates in the Sugar Cane World Across Medium and Hard Difficulty Levels During Marathon Training Sessions.

For Marathon sessions in the Sugar Cane World, the likelihood of episodes concluding due to time expiration is minimal across both Medium and Hard difficulties. This

outcome arises from the UAV's encounters with obstacles, which typically result in collisions that conclude episodes, well before the risk of time expiration. This pattern suggests that, within complex environments, navigational challenges from obstacles take precedence over the duration of exploration (Figure 5.13).

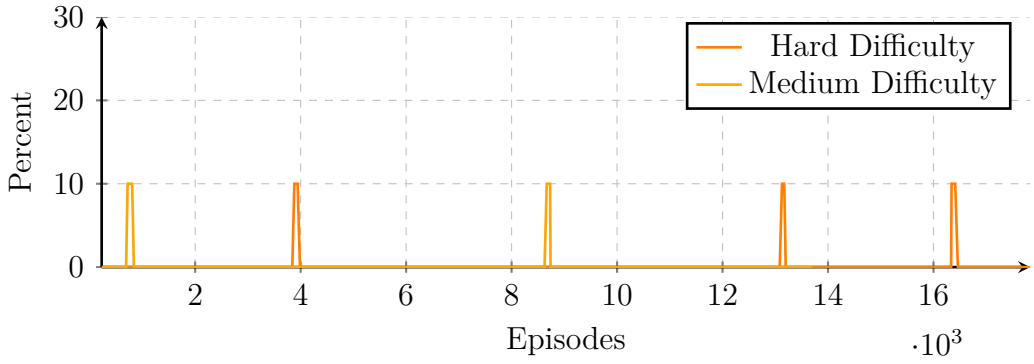

Figure 5.13: Comparison of Time Expired Rates in the Sugar Cane World Across Medium and Hard Difficulty Levels During Marathon Training Sessions.

Reflecting on the UAV's reward accumulation, the graph (Figure 5.14) signifies that Medium difficulty scenarios are more conducive to effective learning, with higher total rewards signifying better overall performance.

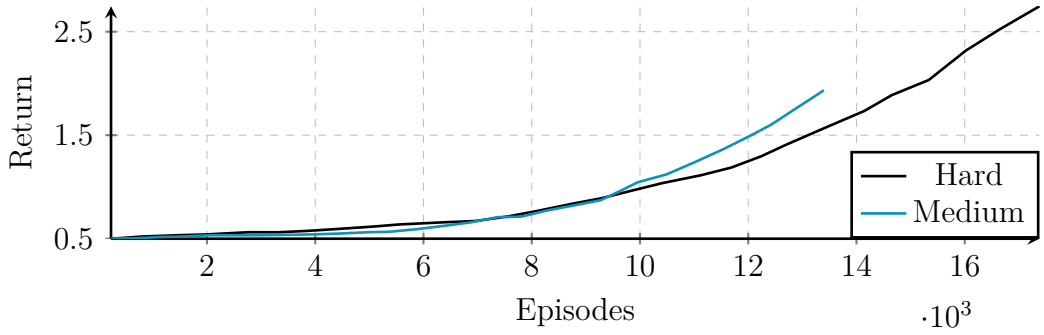

Figure 5.14: Comparison of General Total Rewards in the Sugar Cane World Across Medium and Hard Difficulty Levels During Marathon Training Sessions

Performance Overview

A comparative analysis within the Sugar Cane World across the Medium and Hard difficulty levels reveals a clear pattern: the UAV's performance inversely correlated with the complexity of the obstacle environment. The environment's complexity, characterized by vertical obstacles requiring precise navigation, poses a significant

challenge to the UAV's learning algorithms. Despite the Marathon training duration providing ample opportunity for learning and adaptation, the UAV struggled to achieve significant success rates, particularly in the Hard difficulty setting. This observation is indicative of the substantial impact that obstacle density and configuration have on the UAV's learning efficiency and performance.

Furthermore, the slight improvements observed with prolonged training suggest a nuanced learning curve, where extended exposure leads to incremental gains in navigational proficiency. However, these gains were modest and suggest diminishing returns as complexity increases, potentially indicating a plateau in the learning trajectory within the highly challenging environments of the Sugar Cane World. When compared to other environments, such as the Block World, these results underscore the importance of environment-specific strategies and the inherent difficulties in adapting to different types of obstacles and spatial challenges.

5.2.3 Slat Fence World Results

The Slat Fence World, with its unique configuration of horizontal cylindrical obstacles, was designed to test the UAV's ability to navigate through environments that significantly differ from traditional vertical obstacle layouts. This section focuses on analyzing the UAV's performance in the Slat Fence World, specifically under Medium and Hard difficulties over Marathon training sessions. The aim is to understand the challenges and learning outcomes within this distinctive setup.

Training Outcomes Across Difficulty Levels

- **Medium (6 Obstacles, Marathon Duration):** The UAV's training sessions in the Slat Fence World at Medium difficulty presented a notable challenge. The reliance on vertical sonars, due to the limitations of the lidar sensor in detecting horizontal obstacles, introduced a significant hurdle in the UAV's navigation capabilities. Despite the Marathon duration providing extensive training time, the improvement in success rates was modest. The results indicated a struggle to adapt to the environment, where traditional sensors, like lidar, offered limited

assistance, and the sparse information from vertical sonars proved insufficient for effective obstacle avoidance and navigation.

- **Hard (12 Obstacles, Marathon Duration):** Escalating the challenge to Hard difficulty exacerbated the UAV's navigational challenges in the Slat Fence World. The dense configuration of horizontal obstacles further limited the effectiveness of the UAV's sensor suite, particularly the lidar, in providing actionable environmental data. The performance metrics reflected a low success rate, with only marginal improvements over the course of the Marathon training. This outcome highlights the significant impact that sensor limitations and obstacle orientation have on the UAV's ability to learn and navigate effectively.

Comparison within Slat Fence World

The Medium difficulty demonstrates a higher gate found rate, as shown in the graph (Figure 5.15). This suggests that the UAV is better suited to navigating environments with fewer obstacles, where sensor data is more manageable and informative.

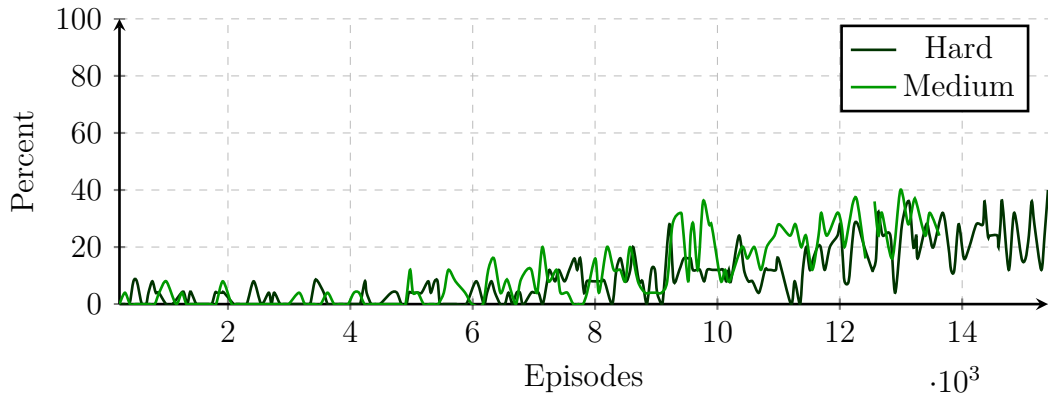

Figure 5.15: Comparison of Gate Found Rates in the Slat Fence World Across Medium and Hard Difficulty Levels During Marathon Training Sessions.

The graph reflects higher collision rates in the Hard difficulty setting (Figure 5.16), indicating the UAV's difficulty in handling dense, horizontal obstacle configurations that challenge its sensor capabilities.

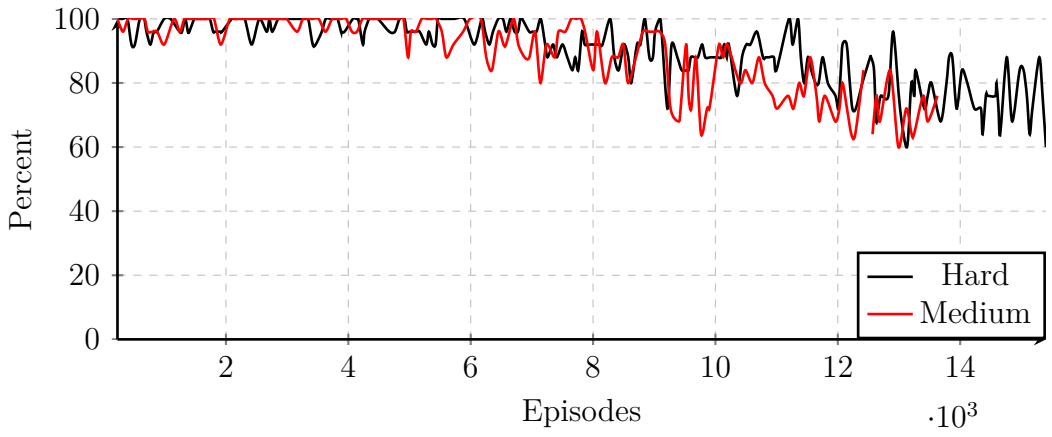

Figure 5.16: Comparison of Collision Rates in the Slat Fence World Across Medium and Hard Difficulty Levels During Marathon Training Sessions.

For the Marathon sessions in the Slat Fence World, time expiration events are uncommon across both Medium and Hard difficulties (Figure 5.17). This pattern emerges as episodes often conclude with collisions, preempting any concern of surpassing the allotted time for a task. This dynamic underscores how navigating through obstacles is the primary challenge, with the UAV's episodes frequently ending upon obstacle encounters.

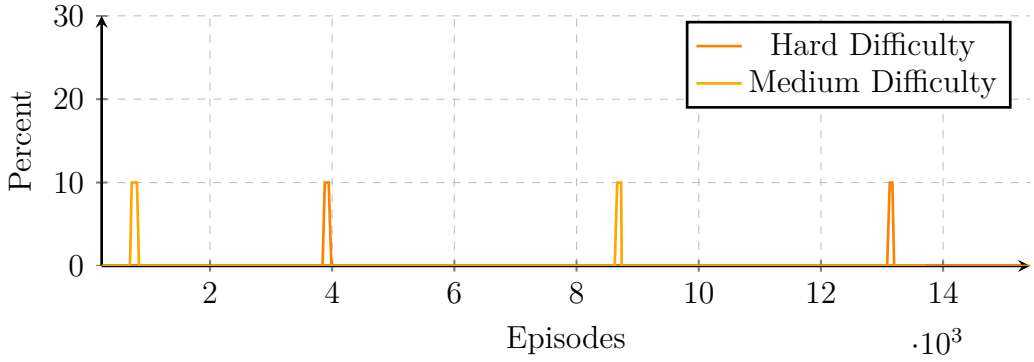

Figure 5.17: Comparison of Time Expired Rates in the Slat Fence World Across Medium and Hard Difficulty Levels During Marathon Training Sessions.

The total reward graph clearly differentiates the UAV's performance, with Medium difficulty yielding better reward outcomes (Figure 5.18), highlighting the challenges posed by the Hard difficulty level.

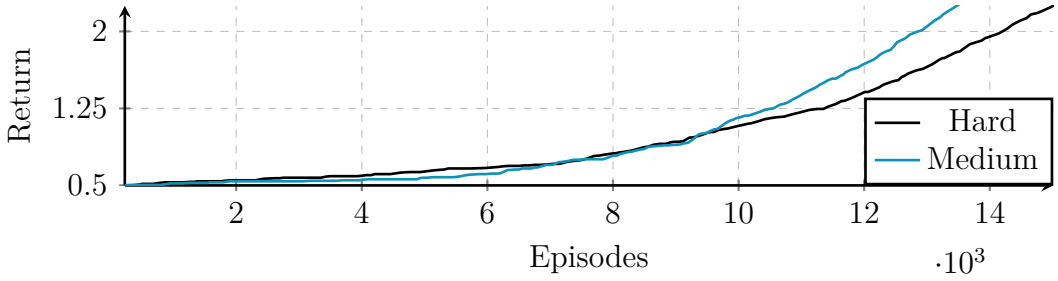

Figure 5.18: Comparison of General Total Rewards in the Slat Fence World Across Medium and Hard Difficulty Levels During Marathon Training Sessions

Performance Overview

The results from the Slat Fence World highlight the intricate challenges involved in UAV navigation, especially when traditional sensing modalities face limitations due to environmental configurations. The specific orientation of obstacles in the Slat Fence World, combined with the UAV's reliance on less informative vertical sonars, underscores a consistent trend: easier difficulty levels generally yield better performance outcomes.

Despite extended Marathon training sessions, performance improvements remained limited. The difficulty in detecting horizontal obstacles with the predominantly used lidar sensor, coupled with the high obstacle density in the Hard difficulty level, significantly hindered the UAV's ability to adapt and navigate effectively. This suggests a plateau in learning efficiency under the constraints imposed by the environment and sensor capabilities.

Compared to the Block World and Sugar Cane World, the Slat Fence World presented a distinct set of challenges that led to comparatively lower performance outcomes. These findings emphasize the need for tailored navigation strategies and possibly sensor augmentation or algorithmic adjustments to improve performance in such unique environments.

5.3 Comparative Analysis

5.3.1 World Comparison Based on Best Results

The comprehensive experimentation across Block World, Sugar Cane World, and Slat Fence World provides a rich dataset for evaluating the UAV's navigational

capabilities under various conditions. This section focuses on comparing the best results obtained from each world to delineate their impacts on the UAV's learning efficiency and adaptability. The objective is to identify which environment fosters the most effective learning and why.

Block World: The Block World, characterized by its geometric diversity and procedural generation of obstacles, served as a foundational environment for the UAV's training. The best results were observed under Medium difficulty in Marathon sessions, where the UAV demonstrated significant improvements in navigation and obstacle avoidance. This suggests that the geometric variability and moderate obstacle density in Block World are conducive to balanced learning, offering enough challenge to stimulate learning without overwhelming the UAV.

Sugar Cane World: In contrast, the Sugar Cane World's vertical obstacles presented a different challenge, emphasizing precision navigation through tightly spaced barriers. The Medium difficulty during Marathon sessions yielded the best results, indicating that the UAV effectively adapted its navigational strategies to maneuver through narrow gaps. The focused nature of this challenge appears to enhance the UAV's precision control and decision-making capabilities.

Slat Fence World: The Slat Fence World, with its horizontal obstacles, tested the UAV's vertical navigation skills. Like the other worlds, the Medium difficulty during Marathon sessions provided the most conducive setting for learning. The environment forced the UAV to adapt to three-dimensional navigation challenges, improving its ability to negotiate obstacles, not just horizontally, but also vertically.

Comparison Summary: Across all environments, Medium difficulty paired with Marathon training sessions emerged as the best configuration for UAV learning. This consistency underscores the importance of a balanced challenge—sufficiently difficult to push the UAV's learning boundaries, but not so dense as to inhibit incremental learning progress. Each world, with its unique obstacle configurations, contributed differently to the UAV's skill set, with Block World enhancing adaptability, Sugar Cane World improving precision, and Slat Fence World broadening navigational strategies.

In conclusion, the comparative analysis reveals that, while each world excels in teaching specific navigational skills, the Block World's diverse obstacle configuration and the balanced challenge it presents make it the most effective environment for generalized UAV training. However, the specialized skills honed in the Sugar Cane and Slat Fence Worlds are indispensable for comprehensive UAV navigational proficiency, emphasizing the value of varied training environments.

5.3.2 Difficulty Level Comparison Across Worlds

This section delves into how varying difficulty levels—Easy, Medium, and Hard—impacted the UAV's learning and performance across the three distinct environments of Block World, Sugar Cane World, and Slat Fence World. The comparison aims to discern patterns in the UAV's adaptability and performance improvements, when faced with increasing levels of navigational challenges.

Block World

- **Easy Difficulty:** Provided a baseline for the UAV's navigational capabilities without the pressure of obstacle avoidance, primarily focusing on flight control and basic maneuvering. The UAV's success rates were predictably high, but the learning potential was limited.
- **Medium Difficulty:** Introduced a moderate challenge with a balanced mix of obstacles, significantly enhancing the UAV's learning curve. This difficulty level was instrumental in teaching the UAV to negotiate a variety of obstacles, demonstrating a substantial increase in adaptability and strategic navigation.
- **Hard Difficulty:** Tested the UAV's limits with dense obstacle configurations. While the UAV showed learning progress, the dense environment often resulted in lower success rates compared to Medium difficulty, indicating a point of diminishing returns, where excessive challenge could impede learning efficiency.

Sugar Cane World

The absence of an Easy difficulty level meant the UAV was immediately subjected to significant navigational challenges. The Medium difficulty proved to be the sweet spot for learning, optimizing the balance between challenge and navigational success. Hard difficulty, while beneficial for honing precision skills, highlighted limitations in adaptability under extreme spatial constraints.

Slat Fence World

Similar to the Sugar Cane World, the focus was on Medium and Hard difficulties, with the UAV confronting unique three-dimensional navigation challenges. The Medium difficulty facilitated more effective learning compared to Hard, suggesting that while the UAV could adapt to novel navigation challenges, overly dense environments hindered its ability to generalize and apply learned strategies effectively.

Across Worlds

Across all environments, the Medium difficulty level consistently emerged as the most effective for UAV training, striking an optimal balance between challenge and learning opportunity. The Easy level, where applicable, offered limited learning potential due to the lack of obstacles, whereas the Hard level often presented challenges that exceeded the UAV's current adaptation capabilities, leading to reduced learning efficiency.

5.3.3 Training Duration Impact Analysis

This segment explores the influence of training duration on the UAV's performance and learning outcomes within the same difficulty levels across the Block World, Sugar Cane World, and Slat Fence World. By comparing the results from Large (8 hours) and Marathon (32 hours) training sessions, we aim to elucidate how extended exposure to training environments affects the UAV's adaptability and proficiency in navigating through obstacles.

Block World Analysis

Large vs. Marathon Duration: In the Block World, both Medium and Hard difficulties were subjected to Large and Marathon durations. The extension from Large to Marathon training sessions consistently showed improvements in the UAV's performance metrics across difficulties. This trend was particularly pronounced in the Hard difficulty, where the extended duration allowed for a more nuanced understanding and maneuvering through dense obstacle arrangements, albeit with diminishing returns, as the complexity increased.

Sugar Cane World Analysis

Marathon Duration Focus: Given the Marathon duration was exclusively used for the Sugar Cane World, the comparison relies on the differential performance between Medium and Hard difficulties over this extended period. The UAV's learning curve indicated that, while progress was made in both difficulties, the Marathon duration's benefits were more observable in the Medium difficulty, where incremental learning and adaptation to narrowly spaced obstacles were more pronounced.

Slat Fence World Analysis

Marathon Duration Focus: Similar to the Sugar Cane World, the Slat Fence World training exclusively utilized the Marathon duration for Medium and Hard difficulties. The extended training period revealed that horizontal obstacles pose unique challenges, with the UAV showing modest performance improvements. The Marathon duration's impact was evident in the Medium difficulty, suggesting a more effective adaptation to horizontal obstacles than in the denser Hard difficulty setup.

Cross-World Implications

Across all environments, the extended Marathon duration facilitated deeper learning and adaptation, especially in Medium difficulty levels. This suggests that, while prolonged exposure to the training environment enhances learning, the effectiveness

of this extended training varies with the complexity of the environment and the nature of obstacles presented.

The Marathon duration's impact highlights the importance of sustained and iterative learning processes, particularly in complex navigation tasks. However, the benefits of extended training must be weighed against the potential for diminishing returns in environments where obstacle density or complexity exceeds a certain threshold.

5.4 Discussion and Observations

5.4.1 Interpretation of Results

The comprehensive experiments conducted across the Block World, Sugar Cane World, and Slat Fence World have provided a deep dive into the effectiveness of the Deep Q-Network (DQN) algorithm in navigating dynamic and challenging environments. This exploration has shed light on the nuanced relationship between the complexity of the environment, the duration of the training, and the UAV's learning progression and performance, highlighting the adaptability and limitations of the DQN algorithm.

Learning Efficiency Across Environments

The results underscore the impact of environmental complexity on the DQN algorithm's learning efficiency. In environments with simpler dynamics, such as the Block World under Easy difficulty, the UAV, showcased rapid adaptability, achieving impressive success rates and gate discovery metrics. Conversely, environments with higher complexity levels, notably in the Hard difficulties of the Sugar Cane World and the Slat Fence World, revealed the challenges DQN faces in optimizing actions in spaces with dense obstacles, evidenced by a more gradual learning progression.

Adaptability to Environment Complexity

The UAV's transition from simpler to more complex environments highlighted the DQN algorithm's potential for adaptability through iterative learning cycles and extended training durations. This adaptability is key to refining the decision-making process in the face of novel and challenging scenarios. However, the variable adaptability across

environments points to the need for further algorithmic enhancements to improve navigation through environments with complex obstacle configurations.

Performance Improvements Through Training:

The positive correlation between extended Marathon training sessions and performance improvements underscores the importance of prolonged exposure for the DQN algorithm to assimilate complex spatial information and refine its policy. This finding speaks to the inherent capacity of DQN for incremental learning, where continued interaction with the environment leads to progressively optimized navigational strategies.

Impact of Obstacle Density and Configuration

The differential impact of obstacle density and configuration on the UAV's performance became evident, when comparing results across the three worlds. High obstacle densities and complex configurations, particularly in the Slat Fence World and the Hard difficulty levels of the Sugar Cane World, emphasized the limitations of the UAV's sensor suite and processing algorithms in efficiently navigating through such environments.

Conclusion

This in-depth analysis reveals the strengths and areas for improvement of the DQN algorithm in autonomous UAV navigation across varied and complex environments. The UAV's performance, guided by DQN, showcases significant learning and adaptability, but also highlights the necessity for algorithmic advancements to tackle environments of elevated complexity. Future work should focus on enhancing the DQN framework to better handle complex spatial configurations and integrate more sophisticated sensor data, paving the way for more adept and versatile autonomous navigation systems.

6 Conclusion

Contents

THIS concluding chapter provides a comprehensive summary of the research journey undertaken in this thesis. It begins by encapsulating the critical aspects of the project, from the problem statement to the proposed solution and its experimental evaluation. Looking ahead, the chapter then outlines potential avenues for future research and improvements inspired by the findings of this work. The final section of this chapter offers reflective remarks on the research process and contemplates the potential impact of this work in the field of autonomous drone navigation. The objective is to provide a clear, concise conclusion to the thesis, while highlighting its implications and opportunities for future exploration.

6.1 Summary of Research

This research aimed to develop a system for autonomous drone navigation in unknown 3D simulated environments, aiming to eventually transfer the learned model to a real UAV. This approach employed Deep Reinforcement Learning (DRL) to enable the drone to learn effective movement patterns in a trial-and-error manner within three distinct types of virtual environments. Each type of environment presented multiple difficulty levels and unique obstacle configurations, containing obstacles of varying shapes and one gate randomly placed within the world. The drone's primary task was to detect the gate and navigate towards it, while avoiding any obstacles.

The drone relied on sensor data about its surrounding environment to inform its choice of actions to fulfill its duty. The drone received feedback in the form of rewards for its actions, incentivizing it to take actions that contributed best to completing its task. Over time, the drone learned to make better decisions, improving its navigation performance.

The drone was equipped with a variety of sensors, including an optical camera, a Global Navigation Satellite System (GNSS) module, an Inertial Measurement Unit (IMU), a 2D Light Detection and Ranging (LiDAR) sensor, and two sonar modules. The camera was primarily used for gate detection, while the other sensors collected crucial data for obstacle avoidance and navigation.

The drone and its environment were simulated using the Robot Operating System (ROS) Gazebo simulator. The Deep Q-Network (DQN) algorithm was used for learning. DQN is a variant of Q-learning, where a deep neural network is used to approximate the Q-value function. Given its current state, the Q-value function helps the agent determine the best action to take. DQN introduces the concept of experience replay and target network to stabilize the learning process, enhancing the efficiency of reinforcement learning.

An existing gate detection system, known for its high Intersection over Union (IoU) accuracy rate, was integrated into the proposed system to improve gate detection accuracy.

The experimental results showed promising performance from the proposed system. The drone successfully navigated towards its goals, while avoiding obstacles, particularly in simpler environments. While performance decreased in more complex environments, the results were still satisfactory, suggesting that the system's performance could continue to improve with additional training. This research lays a strong foundation for developing a navigation system capable of safely directing drones toward desired targets in real-world scenarios.

6.2 Limitations

In summary, the findings of this thesis contribute significantly to the field of autonomous drone navigation and present a host of promising directions for future exploration and development.

Every research endeavor carries its own set of limitations, and this thesis is no exception. The approach and methodology adopted in this thesis have led to promising results, yet they also present certain constraints that must be acknowledged.

6.2.1 Sensor Limitations

A significant challenge highlighted by the experiments is the limitation of the UAV's sensor suite, particularly in environments, like the Slat Fence World, where horizontal obstacles are prevalent. The reliance on lidar and vertical sonars, while effective in certain configurations, proved insufficient for comprehensive environmental perception in others. This limitation suggests a need for exploring advanced sensor technologies or multi-sensor fusion techniques to enhance obstacle detection and environmental awareness.

6.2.2 Algorithmic Constraints

While the DQN algorithm demonstrated a capacity for learning and adaptation, its performance varied notably across different environments and complexity levels. The algorithm struggled with optimizing decision-making in scenarios with high obstacle density and complex spatial arrangements, indicating potential areas for improvement in its architecture or learning process. Incorporating techniques, such as prioritized

experience replay, multi-step learning, or even transitioning to more sophisticated algorithms, like Deep Deterministic Policy Gradients (DDPG), might offer pathways to overcoming these challenges.

6.2.3 Environmental and Real-World Application Constraints

The use of simulated environments, although beneficial for controlled testing, does not entirely capture the unpredictability and complexities of real-world scenarios. Factors, such as weather conditions, GPS signal loss, and sensor noise, pose challenges not fully represented in simulations. Additionally, the research environments were limited to static obstacle configurations, without dynamic or moving challenges, which does not fully test the system's capabilities in more volatile real-world conditions.

6.2.4 Training Duration and Computational Resources

The distinction between Large and Marathon training durations brings to light the computational resource intensity, required for deep learning models to achieve significant learning outcomes. Prolonged training sessions, although beneficial for learning, demand substantial computational power and time, which may not always be feasible. This limitation points to the need for more efficient training methodologies or the exploration of transfer learning techniques to reduce training time, without compromising learning efficacy.

6.2.5 Scalability and Generalization

The system's scalability and generalization across different environmental settings remain under-explored. This research focused on single-agent scenarios, and the performance in multi-agent situations, where multiple drones operate simultaneously, is yet to be tested. Future work needs to address the scalability and flexibility of the UAV's learning algorithm to ensure its applicability across a broader range of scenarios.

The challenges and limitations encountered in this study highlight crucial areas for future research in the field of autonomous UAV navigation. Addressing these challenges requires a multi-faceted approach, including enhancements to sensory

processing, algorithmic improvements, and the exploration of more efficient training methodologies. Overcoming these hurdles is essential for advancing the capabilities of UAVs in navigating through dynamically complex environments with higher degrees of autonomy and reliability.

6.3 Future Work

While providing substantial advancements in autonomous drone navigation, this research also opens the door to numerous future research opportunities. One key area that emerges for further exploration is the challenge of navigating complex environments. While the drone showed adeptness in more straightforward settings, there was a marked decrease in performance within more complex surroundings. Future endeavors could, therefore, focus on augmenting the drone's navigation ability in these intricate environments. This might include the development of more complex simulated environments with a higher density and diversity of obstacles or the introduction of dynamic obstacles.

Regarding algorithmic enhancements, the Deep Q-Network (DQN) algorithm served as the backbone of this project. However, exploring more advanced reinforcement learning algorithms may lead to significant improvements in performance. Techniques, such as Advantage Actor-Critic (A2C) and Proximal Policy Optimization (PPO), or integrating DQN with advanced strategies like Double DQN or Dueling DQN, could be the focus of future studies.

The sensor setup of the drone, which was instrumental in obstacle detection and avoidance, can also be a subject of future optimization. Including other sensor types or configurations could potentially enhance the drone's obstacle detection and avoidance abilities. Moreover, integrating more sophisticated gate detection systems could augment goal recognition.

While this study was conducted within a simulated environment, future work could extend to real-world testing of the proposed system. This would provide a more rigorous and robust evaluation of the system's efficacy and may unearth new challenges and areas for improvement.

The applicability of the proposed system also extends beyond drones. The principles of deep reinforcement learning and goal detection could be adapted for other autonomous vehicles, such as self-driving cars or autonomous underwater vehicles. The potential for navigating through various environments to reach a specific goal presents a broad domain of possibilities.

Lastly, an exciting avenue for future research could be exploring multi-agent systems. Although the current research focused on a single agent, multiple drones collaborating to achieve a shared goal could introduce new complexities, such as coordination and communication between agents.

6.4 Final Thoughts

Reflecting on the journey of this research, it has been a fascinating and challenging endeavor. The goal of enabling a drone to autonomously navigate unknown environments using Deep Reinforcement Learning (DRL) and visual gate detection was ambitious. It presented numerous challenges, from understanding the intricacies of reinforcement learning algorithms to dealing with the complexities of simulated environments. Nevertheless, progressing from the initial concept to the final results was deeply rewarding.

The research revealed the potency and versatility of reinforcement learning, specifically the Deep Q-Network algorithm, in addressing complex navigation problems. Despite its limitations, DRL demonstrated a remarkable ability to learn from trial and error, gradually improving the drone's navigation performance through iterative learning.

However, it is essential to remember that this research, like any scientific endeavor, is a stepping stone rather than a final destination. While significant strides were made, the limitations identified provide a roadmap for future exploration and development. There is potential for further optimization, from using a different algorithm and refining sensor setup to expanding the system to multi-agent scenarios.

The potential impact of this research is both exciting and profound. With further development and refinement, the system could have significant real-world applications. From search and rescue missions in difficult terrains to package delivery

in urban environments, autonomous drone navigation could revolutionize how we approach these tasks.

Overall, the journey of this research has been an enriching learning experience. It is a testament to the power of machine learning and its potential to create solutions, once considered the realm of science fiction. Like the drone in this research, the journey will continue navigating through unknown environments, learning, adapting, and improving along the way. The future of autonomous drone navigation is promising, and this research is a proud contributor.

References

[1] Ahmad Taher Azar et al. “Drone Deep Reinforcement Learning: A Review”. In: *Electronics* 10.999 (2021). DOI: 10.3390/electronics10090999. URL: https://doi.org/10.3390/electronics10090999.

[2] Clare Lyle, Marc G Bellemare, and Pablo Samuel Castro. “A Comparative Analysis of Expected and Distributional Reinforcement Learning”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 33. 01. 2019, pp. 4504–4511.

[3] Volodymyr Mnih et al. “Playing Atari with Deep Reinforcement Learning”. In: *Proceedings of the 26th International Conference on Neural Information Processing Systems (NIPS)*. Vol. 2. Curran Associates Inc. 2013, pp. 2188–2196.

[4] Hado Van Hasselt, Arthur Guez, and David Silver. “Deep Reinforcement Learning with Double Q-learning”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 30. 1. 2016.

[5] Ziyu Wang et al. “Dueling Network Architectures for Deep Reinforcement Learning”. In: *International Conference on Machine Learning*. PMLR. 2016, pp. 1995–2003.

[6] Tom Schaul et al. “Prioritized Experience Replay”. In: *ResearchGate* (2016). URL: https://www.researchgate.net/publication/284219262_Prioritized_Experience_Replay.

[7] Meire Fortunato et al. “Noisy Networks for Exploration”. In: *arXiv preprint arXiv:1706.10295* (2017). URL: https://www.researchgate.net/publication/318107055_Noisy_Networks_for_Exploration.

[8] Carol Fairchild and Thomas L. Harman. *ROS Robotics By Example*. Second Edition. Birmingham - Mumbai: Packt Publishing, 2017.

[9] Dimitrios Chatziparaschis. “Machine Learning for Enhancing Robotic Perception and Control”. Master’s Thesis. Technical University of Crete, 73100: School of Electrical and Computer Engineering, Technical University of Crete, Nov. 2020.

[10] Iris Automation. *Drone Sensors: The Different Types You Should Know About*. Web Article. San Francisco, USA, July 2020. URL: https://www.irisonboard.com/drone-sensors-the-different-types-you-should-know-about/.

[11] Ender Cetin et al. “Drone Navigation and Avoidance of Obstacles Through Deep Reinforcement Learning”. In: *2019 IEEE/AIAA 38th Digital Avionics Systems Conference (DASC)*. IEEE. 2019. URL: https://doi.org/10.1109/DASC43569.2019.9081725.

[12] S.Y. Shin, Y.W. Kang, and Y.G. Kim. “Obstacle Avoidance Drone by Deep Reinforcement Learning and Its Racing with Human Pilot”. In: *Applied Sciences* 9.24 (2019), p. 5571. URL: https://doi.org/10.3390/app9245571.

[13] Michalis Galanis. "Autonomous Drone Navigation for Landmark Position Estimation using Reinforcement Learning". Meng Thesis. Chania, Greece: Technical University of Crete, School of Electrical and Computer Engineering, Sept. 2021.

[14] Chenglu Wen et al. "Three-Dimensional Indoor Mobile Mapping With Fusion of Two-Dimensional Laser Scanner and RGB-D Camera Data". In: *IEEE Geoscience and Remote Sensing Letters* 11.4 (Apr. 2014), pp. 843–847. DOI: 10.1109/LGRS.2013.2279872.

[15] Anna Guerra et al. "Reinforcement Learning for UAV Autonomous Navigation, Mapping and Target Detection". In: *2020 IEEE/ION Position, Location and Navigation Symposium (PLANS)* (Apr. 2020). DOI: 10.1109/plans46316.2020.9110163. URL: http://dx.doi.org/10.1109/PLANS46316.2020.9110163.

[16] Taha Elmokadem and Andrey V. Savkin. "Towards Fully Autonomous UAVs: A Survey". In: *Sensors* 21.21 (2021), p. 6223. DOI: 10.3390/s21186223. URL: https://doi.org/10.3390/s21186223.

[17] Fadi AlMahamid and Katarina Grolinger. "Autonomous Unmanned Aerial Vehicle Navigation Using Reinforcement Learning: A Systematic Review". In: *Engineering Applications of Artificial Intelligence* 115 (2022), p. 105321. DOI: 10.1016/j.engappai.2022.105321. URL: https://doi.org/10.1016/j.engappai.2022.105321.

[18] Yupeng Yang et al. "Autonomous UAV Navigation in Dynamic Environments with Double Deep Q-Networks". In: *2020 AIAA/IEEE 39th Digital Avionics Systems Conference (DASC)*. IEEE. 2020. DOI: 10.1109/DASC50938.2020.9256634. URL: https://doi.org/10.1109/DASC50938.2020.9256634.

[19] Amudhini P. Kalidas et al. "Deep Reinforcement Learning for Vision-Based Navigation of UAVs in Avoiding Stationary and Mobile Obstacles". In: *Drones* 7.4 (2023), p. 245. DOI: 10.3390/drones7040245. URL: https://doi.org/10.3390/drones7040245.

[20] Chao Wang et al. "Autonomous Navigation of UAVs in Large-Scale Complex Environments: A Deep Reinforcement Learning Approach". In: *IEEE Transactions on Vehicular Technology* 68.3 (Mar. 2019), pp. 2124–2136. DOI: 10.1109/TVT.2018.2890773.

[21] C. Sampedro et al. "Laser-Based Reactive Navigation for Multirotor Aerial Robots using Deep Reinforcement Learning". In: *Proceedings of the 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Madrid, Spain, Oct. 2018, pp. 1024–1031. DOI: 10.1109/IROS.2018.8593578. URL: https://doi.org/10.1109/IROS.2018.8593578.

[22] Liang Lu et al. "A Robust and Fast Collision-Avoidance Approach for Micro Aerial Vehicles Using a Depth Sensor". In: *Remote Sensing* 13.9 (2021), p. 1796. DOI: 10.3390/rs13091796. URL: https://doi.org/10.3390/rs13091796.

[23] Philipp Foehn et al. "AlphaPilot: Autonomous Drone Racing". In: *Robotics: Science and Systems* (2020). URL: https://arxiv.org/abs/2005.12813.

[24] Matteo Hessel et al. "Rainbow: Combining Improvements in Deep Reinforcement Learning". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 32. 1. 2018.