Reinforcement Learning Based Autonomous Quadcopter Control

- Semester 7 Report -

Pankayaraj Pathmanathan
Chandima Samarasinghe
Yuvini Sumanasekera

Department of Computer Engineering
University of Peradeniya

Final Year Project (courses CO421 & CO425) report submitted as a requirement of the degree of
B.Sc.Eng. in Computer Engineering

November 2019

Supervisors: Dr. Dhammika Elkaduwe (University of Peradeniya) and Dr. Upul Jayasinghe (University of Peradeniya)

We would like to dedicate this thesis to our project supervisors, academia and colleagues who guided us in this process. We are grateful for their support, advise and endless encouragement.

Declaration

We hereby declare that except where specific reference is made to the work of others, the contents of this dissertation are original and have not been submitted in whole or in part for consideration for any other degree or qualification in this, or any other university. This dissertation is our own work and contains nothing which is the outcome of work done in collaboration with others, except as specified in the text and Acknowledgments.

Pankayaraj Pathmanathan
Chandima Samarasinghe
Yuvini Sumanasekera
November 2019

Acknowledgements

First and foremost we would like to thank our supervisor, Dr. Dhammika Elkaduwe and our co-supervisors Dr D.H.S Maithripala and Dr. Upul Jayasinghe for their continuous support and guidance in the successful delivery of the project. Their invaluable feedback and constructive criticism ensured that the project progressed in the right direction and its successful completion in Semester 7.

At the same time, we wish to express our appreciation to Prof. Roshan Ragel, Dr. Pradeepa Bandaranayake and Dr.Sampath Deegalla for their guidance and helpful discussions regarding the final year project.

We are also grateful to all members of the academic staff and non-academic staff at the Department of Computer Engineering, University of Peradeniya, for their support extended in numerous ways.

Finally, our heartfelt gratitude goes out to our families who have always been encouraging and supporting us throughout our academic career.

Abstract

In recent years, extensive research has been carried out in the field of autonomous aerial vehicle control, motivated by the rapid advancements in Machine Learning (ML). In particular, Reinforcement Learning (RL) has gained immense interest in developing control algorithms given its ability to learn useful behavior by dynamically interacting with the environment, without the need for an explicit teacher. In this work, we examine the use of RL methods on vision-based quadcopter control in both single-agent and multi-agent simulated environments. Specifically, the DQN algorithm was investigated in the single-agent setting and the MADDPG algorithm in the multi-agent setting. The control task in each of these settings was to navigate through the environment by avoiding obstacles to reach the specified goals. Thus, each of the aforementioned algorithms were evaluated on their ability to perform this control task. Given the relatively short training period (given the time constraints of the current semester), each algorithm was shown to perform considerably well. Existing gaps within current implementations were identified as well as potential improvements that can be carried forward to the next semester. The necessary ground work required to implement these improvements were also completed.

Table of contents

List of figures viii

List of tables ix

Nomenclature x

1 Introduction 1

1.1 Background . . . 2

1.1.1 Reinforcement Learning . . . 2

1.1.2 Multi-Agent Reinforcement Learning (MARL) . . . 4

1.2 Problem Definition . . . 6

1.3 Deliverables and Milestones . . . 7

2 Related work 9

3 Methodology 12

3.1 Architecture . . . 12

3.2 Simulation Platforms For Quadcopters . . . 13

3.2.1 AirSim . . . 14

4 Experimental Setup and Implementation 15

4.1 Simulation Environment . . . 15

4.1.1 Setting up the project environment . . . 15

4.1.2 Building custom maps . . . 16

4.2 Interactive Wrapper Environment . . . 18

4.2.1 Data Manipulation . . . 19

4.3 Algorithms . . . 20

4.3.1 Single-Agent Setting - DQN . . . 20

4.3.2 Multi-Agent Setting - MADDPG . . . 21

4.4 Pitfalls and Workarounds 23

5 Results and Analysis 25

5.1 Results 25

5.2 Analysis 26

6 Conclusions and Future Works 27

References 28

List of figures

1.1 Agent-Environment Interaction in Reinforcement Learning 2
1.2 Gantt Chart (Project Progress of Semester 7) 8

3.1 High-level Architecture 12

4.1 Open environment with goals only 16
4.2 Environment with goals and obstacles 16
4.3 Maze environment with goals and obstacles 17
4.4 Corridor with partitions on the top and bottom 17
4.5 Corridor with partitions on the left and right 17
4.6 Maps in which the algorithm was trained on 18
4.7 Interactive Wrapper Environment 19
4.8 Neural network architecture for the DQN Q-value function 21
4.9 A graphical representation of the MADDPG architecture as given in [1] . 22
4.10 Architecture of an actor network in the MADDPG model 23
4.11 Architecture of the critic network in the MADDPG model 23
4.12 An unclear depth image 24
4.13 A clear depth image 24

5.1 Simple corridor 26
5.2 Pre-built map 26

List of tables

Nomenclature

Acronyms / Abbreviations

MARL	Multi-Agent Reinforcement Learning
MDP	Markov Decision Process
ML	Machine Learning
NN	Neural Network
RL	Reinforcement Learning
UAV	Unmanned Aerial Vehicle

Chapter 1

Introduction

Quadcopters have become increasingly ubiquitous over the recent years. Given their simplified construction, high maneuverability, and operational flexibility, the utilization of quadcopters has diversified into numerous domains including military surveillance, agriculture, geographic mapping, disaster management, wildlife conservation, etc. Such application domains may involve dynamic, unstructured, and complex environment settings. However, for quadcopters operating in such environments, defining which control behaviour is optimal for a given situation, ahead of time, is impossible. Thus, traditional control methodologies are often inadequate in such situations. As a result, the focus of research has shifted to autonomous control as a potential alternative [2] [3].

The application of Machine Learning (ML) techniques for autonomous control has gained significant attention in the research community. Notably, incorporating Reinforcement Learning (RL) for autonomous control of aerial vehicles is currently an active area of research. As opposed to using a predefined controller structure, RL enforces the learning of optimal behaviour through trial and error, by allowing the agent (i.e. the learner) to interact with the environment. Thus, RL is particularly useful in solving autonomous control problems in dynamic, uncertain, and potentially complex environments [4] [5]. Moreover, RL can be applied in circumstances where information of the environment and/or system dynamics is limited or unavailable [6].

As of now, comparatively, research on RL has been widely conducted within single-agent domains. However, the use of multiple agents to improve performance in terms of efficiency, reliability, effectiveness, etc., can be seen in emergent quadcopter applications [7] [8] [9] [10]. In such multi-agent settings, the challenge lies with designing an algorithm that encapsulates the added complexity of dynamic interactions between agents on top of the existing dynamic behaviour seen in single-agent setups. This research has been centered around this interesting challenge of successfully incorporating RL to address the quadcopter control problem in a multi-agent setting. That is, in situations with multiple quadcopters operating within the same environment.

1.1 Background

1.1.1 Reinforcement Learning

In contrast to classical Machine Learning techniques, Reinforcement Learning approaches attempt to learn useful behaviour through dynamic interactions between a "goal-directed" agent (i.e. the learner and decision-maker) and the environment [6]. Reinforcement Learning problems are formally modelled as Markov Decision Processes (MDPs) which can be defined by a tuple $M = \langle S, A, T, R, \gamma \rangle$. Thus, as the agent explores potential behaviour strategies, based on the observed state and the agent's selected action at each time step, it receives a feedback signal from the environment in the form of a numerical reward (positive or negative). Figure 1.1 illustrates this agent-environment interaction.

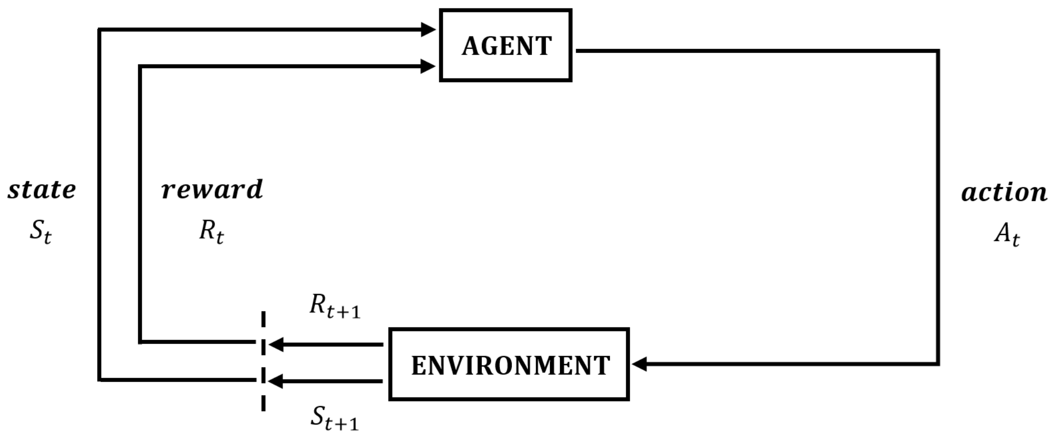


Fig. 1.1 Agent-Environment Interaction in Reinforcement Learning

Assume that the agent's state and action space is denoted by S and A respectively. The probability of reaching the successor state $s' \in S$ by taking action $a \in A$ from a given state $s \in S$, is denoted by $T(s'|s, a)$. $R(s, a)$ denotes the associated reward. A model refers to the agent's representation of the environment. Thus, the model can be defined mathematically in terms of the state transition probability function $T : S \times A \times S \to [0, 1]$ and the reward function $R : S \times A \to \mathbb{R}$.

In general, the agent seeks to maximize its expected cumulative reward (i.e. the return G_t). A discount rate $\gamma \in [0, 1]$ is introduced, which determines the long/short sightedness of the agent.

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots \tag{1.1}$$

A policy π fully defines the agent's behaviour function. Thus, a stochastic policy $\pi(a|s)$, maps a given state to a probability distribution over the actions. Every state's or state-action pair's quality is valued as an expectation of the return (G_t) starting from that particular state or state-action pair and following a certain policy $\pi(a|s)$ thereafter. These expectations are known as the state-value function $V_\pi(s)$ and action-value function $Q_\pi(s, a)$ respectively.

$$V_\pi(s) = E_\pi \left[G_t | S_t = s\right] \tag{1.2}$$

$$Q_\pi(s, a) = E_\pi \left[G_t | S_t = s, A_t = a\right] \tag{1.3}$$

The cumulative expected reward by following a particular policy π over time can be expressed as:

$$J(\pi) = \sum_t \mathbb{E}_{(\mathbf{s}_t, \mathbf{a}_t) \sim \rho_\pi} \left[R\left(\mathbf{s}_t, \mathbf{a}_t\right)\right] \tag{1.4}$$

Thus, the ultimate goal of RL boils down to learning a policy π that maximizes (1.4). RL algorithms can be classified into the following three categories.

1. Value-based methods
2. Policy-based methods
3. Actor-Critic methods

The following three subsections briefly discusses each of these categories.

Value Based Methods

In value-based approaches, the value function is explicitly modelled and improved, which is then subsequently used to extract the optimal policy for the agent. One popular value-based method is Deep Q-Network (DQN) [11] where an explicit modelling of the Q-function is used.

$$Q^\pi(s, a) = \mathbb{E}_{s'} \left[r(s, a) + \gamma \mathbb{E}_{a' \sim \pi} \left[Q^\pi\left(s', a'\right)\right]\right] \tag{1.5}$$

$$\mathcal{L}(\theta) = \mathbb{E}_{s,a,r,s'} \left[\left(Q^*(s, a|\theta) - y\right)^2\right] \tag{1.6}$$

$$y = r + \gamma \max_{a'} \overline{Q}^{*}(s', a') \tag{1.7}$$

The loss $\mathcal{L}(\theta)$ is minimized to obtain the optimal policy. An experience replay and a separate target network is used alongside Q-learning to improve the stability and efficiency of learning.

Policy Based Methods

In policy-based methods, a parameterized policy $\pi_\theta(a|s)$ is directly modelled and improved over time to obtain the optimal policy. As opposed to value-based techniques, the value function is not explicitly modelled. The objective function (1.4) is maximized using the policy gradient shown in the following equation (1.8).

$$\nabla_\theta J(\theta) = \mathbb{E}_{s\sim p^\pi, a\sim\pi_\theta}\left[\nabla_\theta \log \pi_\theta(a|s) R\right] \tag{1.8}$$

where p^π denotes the state distribution for policy π. Based on the Policy Gradient theorem [6], for any differentiable policy $\pi_\theta(a|s)$, R can be replaced with Q (which is estimated instead of being modelled in purely policy-based methods). An example of this approach is the REINFORCE [12] algorithm where multiple trajectories are sampled while updating the policy using the estimated gradient.

Actor-Critic Methods

Actor-critic methods can be considered as a hybrid approach where both the policy and the value function are explicitly modelled. The *critic* estimates the value function (value-based), which is then used to update the *actor's* policy in the direction of performance improvement (policy-based). These algorithms aim to reduce the variance which thereby results in faster convergence. DDPG [13] and PPO[14] are well-known examples of actor-critic algorithms.

1.1.2 Multi-Agent Reinforcement Learning (MARL)

Definition

In a multi-agent learning setup, multiple adaptive agents co-exist within the same environment. A common classification of multi-agent learning problems is based on the nature of interaction among agents; fully competitive, fully cooperative or mixed [15]. RL can be applied in such settings to develop complex physical and/or communicative behaviour strategies among these agents. One of the most commonly used frameworks;

Markov games [16] is a generalisation of MDPs to a multi-agent learning setup. A stochastic or Markov game is defined by a tuple $G = \langle S, A, T, R, n, \gamma \rangle$, where S denotes the finite set of states of the environment. Each of the n agents chooses actions sequentially. A_i is a finite set of actions available to agent i (where $i = 1, 2 \ldots, n$). Thus, each agent i chooses an action $a \in A_i$ which contributes to the joint action $A = A_1 \times A_2 \times \cdots \times A_n$. This in turn induces a state transition in the environment according to the state transition probability function $T : S \times A \times S \to [0, 1]$.

By definition, a single reward function $R : S \times A \to \mathbb{R}$ is shared among all agents and $\gamma \in [0, 1]$ denotes the discount factor. However, the reward structure can be modified to provide separate rewards to individual agents as well.

Challenges in MARL

RL problems formulated using MDPs are under the assumption that the environment is stationary from the point of view of any given agent. In most single-agent settings, given sufficient exploration, the convergence of the underlying RL algorithm is dependant on this assumption [15]. In a multi-agent setting, if each agent learns its optimal behaviour independently, the remaining agents will be treated as part of the environment. However, as learning occurs concurrently, due to the actions of the other agents, now the environment is no longer stationary from the perspective of any given agent, thus violating the stated assumption. Therefore, the presence of any adaptive agents other than the agent of interest leads to this *non-stationarity* problem [17]. One alternative that has been utilized, is the notion of providing a centralized view of all the other agents' actions and/or state observations to any given agent. As a consequence, as learning progresses, the environment becomes stationary for any given agent.

The *credit assignment* problem is commonly prevalent in cooperative multi-agent learning settings, where the agents coordinate their behaviours to achieve some common goal. Thus, the actions of all the agents contribute towards a single global reward. However, with such a reward structure, it is difficult to determine the direct impact of an individual agent's actions to the overall collective performance [18] [19]. Recently, Foerster et al. proposed a multi-agent actor-critic method known as COMA [20], which addresses this issue. COMA utilizes a counterfactual baseline which allows the global reward to be compared against the reward obtained when the action of a given agent is replaced by some default action.

Often times in practical applications, the agents may have to interact with the environment under *partial observability* [21]. An observation is not a complete representation of the actual state of the environment. Moreover, in a multi-agent setting, uncertainties may arise when predicting the actions of the other agents and their subsequent effect to the environment. In such cases, in order to obtain a centralized view of the state at any given time, a potential solution would be to consider the aggregation of the partial observations of all the agents at that particular time step.

While maintaining a centralized view of all the agents' actions-observations may address the non-stationarity issue, it may lead to the *curse of dimensionality* [17]. This represents the exponential growth of the state-action variables with the number of agents. This may also be the case with continuous and high dimensional state-action spaces [13].

In general, the ideal expectation of a multi-agent learning algorithm is that once it's trained, it allows any agent to act independently of the other agents' actions and/or observations, while delivering the expected results through coordinated behaviour. The idea of a centralized critic solves the issue of obtaining the optimal cooperative or competitive results in accordance with the situation. However, the necessity of a centralized model contradicts the idea of acting independently. In the case of value-based RL methods, there is only a single model of the value-function that is utilized to both evaluate and facilitate in making decisions regarding the actions to take at any given time. Centralizing this function indicates the necessity for a non-ideal centralized view in both training and deployment/execution.

In contrast, actor-critic methods utilize two separate models. Particularly, the actor model is used to make decisions (i.e. action selection) and the critic model is used to evaluate state-action pairs/states in order to guide the policy in an ideal direction during the training phase. At execution, the only model of interest is that of the policy, thus the critic model can be discarded at this stage. This notion of *centralized training and decentralized execution* has been the natural paradigm for multi-agent learning problems. It not only facilitate the learning of complex behaviour in the presence of adaptive agents but also enables any given agent to act independently of the other agents' observations. Thus, actor-critic methods use this idea of centralizing the value function. COMA [20] and MADDPG [1] are two existing algorithmic implementations of this concept.

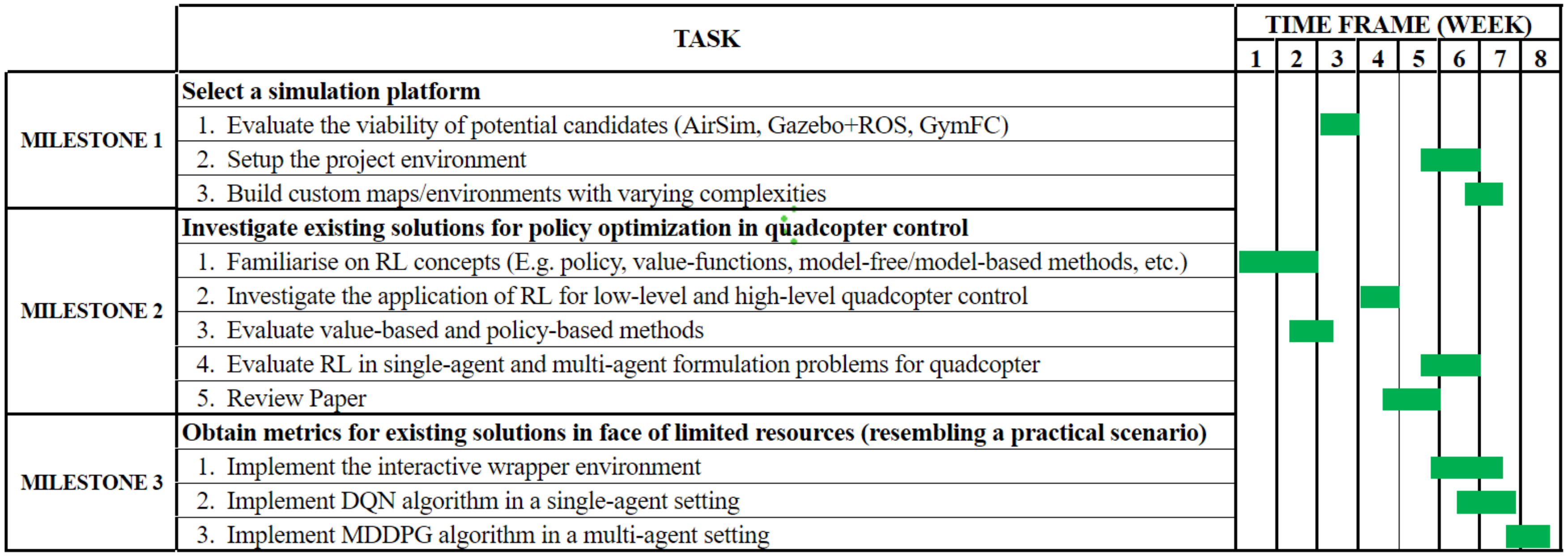
1.2 Problem Definition

The objective of this project is to build on the research of recent years where RL has been utilized in multi-agent domains, so as to effectively coordinate the behaviour of quadcopters such that non-trivial control tasks can be performed. Specifically, the aim is to learn control policies directly from visual-based inputs of the quadcopters to facilitate each quadcopter to successfully navigate to predefined goals by avoiding obstacles. It should be noted that the training will be conducted in simulation. However, the development would be such that it allows the eventual transfer of the model to physical quadcopter hardware in the future as well as ensuring the relevance to real-world applications.

The goal for the current semester (Semester 7) was to identify potential gaps in existing multi-agent RL algorithms such that the learnings can be utilized in the following semester to be integrated into the development of a new MARL algorithm. To facilitate this, the first objective was to investigate the application of RL algorithms in both single-agent and multi-agent settings. Once the selected algorithms in each setting were implemented, the drawbacks/gaps were identified for each algorithm implementation. The second objective was to set up the environment with the necessary requirements for future work. This included implementing an interactive wrapper environment (Section 4.2) and custom building maps (Section 4.1).

1.3 Deliverables and Milestones

The following Gantt chart lists the milestones and associated deliverables of the project pertaining to Semester 7.



	TASK	TIME FRAME (WEEK)							
		1	2	3	4	5	6	7	8
MILESTONE 1	**Select a simulation platform**								
	1. Evaluate the viability of potential candidates (AirSim, Gazebo+ROS, GymFC)			🟩					
	2. Setup the project environment					🟩	🟩		
	3. Build custom maps/environments with varying complexities						🟩	🟩	
MILESTONE 2	**Investigate existing solutions for policy optimization in quadcopter control**								
	1. Familiarise on RL concepts (E.g. policy, value-functions, model-free/model-based methods, etc.)	🟩	🟩						
	2. Investigate the application of RL for low-level and high-level quadcopter control				🟩				
	3. Evaluate value-based and policy-based methods		🟩	🟩					
	4. Evaluate RL in single-agent and multi-agent formulation problems for quadcopter					🟩	🟩		
	5. Review Paper				🟩	🟩			
MILESTONE 3	**Obtain metrics for existing solutions in face of limited resources (resembling a practical scenario)**								
	1. Implement the interactive wrapper environment					🟩	🟩	🟩	
	2. Implement DQN algorithm in a single-agent setting						🟩	🟩	
	3. Implement MDDPG algorithm in a multi-agent setting							🟩	🟩

Fig. 1.2 Gantt Chart (Project Progress of Semester 7)

Chapter 2

Related work

Quadcopters are known to be inherently unstable in nature [22] [23]. This coupled with the uncertainty and complexity of the environment in which the quadcopter operates, makes control a non-trivial task. In the autonomous control paradigm, control laws can be classified based on the level of autonomy they achieve. High-level (i.e. outer-loop) control algorithms address mission-level objectives such as way-point navigation, task allocation, etc., while low-level (i.e. inner-loop) control algorithms govern factors such as stability, control, and signal tracking [24]. Classical UAV (Unmanned Aerial Vehicle) tasks such as taking-off, navigation, hovering, and landing, etc. are governed by different control laws and methods as the control precision of flight parameters required by each task is different. Within the context of autonomous flight control, given the complexity and obscurity of each individual task, previous work has primarily focused on developing and testing control algorithms capable of completing a specific task, in isolation [25]. Moreover, in past research, the application of RL has been mostly confined to higher-level control decisions as opposed to lower-level actuator control.

Within the context of single-agent settings, where only a single quadcopter operates in the environment, Q-Learning based approaches can be seen to have been applied. Moreover, past efforts have examined feasible path planning techniques centered around various objectives such as minimization of the traversed distance, maximization of path safety (E.g. collision avoidance), etc. The control objective of [26] was to determine the shortest possible path to a predefined position starting from an arbitrary position. Model-free Q-learning was adopted to train the quadcopter to navigate to the target point in a closed environment. The actual implementation of the solution verified the simulated results, with the quadcopter successfully learning how to navigate within an unknown environment. However, it was based on a discretized state-action space. Similar to [26], Wu et al. [**?**] addressed the same navigation problem of finding the shortest straight path between two given points. Here, collision avoidance was also taken into consideration. A deep Q-network model was used to determine a route bypassing obstacles in a 3D environment. Their approach was able to reduce the collision rate to 14%. However, the proposed control algorithm focused on maneuvering the quadcopter only in the forward direction. The work [27] used a Q-learning based approach known as Adaptive and Random Exploration (ARE) to accomplish the same control task as in [**?**]; navigation via obstacle avoidance. The paper introduced the triple trap-escape strategy to deal with two major problems that cannot be solved using only a networked Q-learning algorithm. One problem was the possibility of selecting an action that will collide with an obstacle given the action selection probability. The other problem was that it was impossible to explore a trap-escape path if the UAV falls into a local optimum trap. The authors compared three algorithmic models; Q-learning model, Neural Network model, and trap-escape model in simulation. The proposed trap-escape model outperformed the Q-learning model in every map, bypassing all obstacle walls and avoiding falling into wall traps. In contrast, the work of Polvara et al. [28] exhibited the completion of the landing maneuver of quadcopters in the presence of noise using DQN. The control problem was subdivided into two tasks; landmark detection and vertical descent, and a hierarchical arrangement of two Deep Q-Networks (DQNs) was employed. The performance of the proposed solution was similar to that of human pilots and an Augmented Reality (AR) visual tracker. However, both [29] and [28] were carried out in simulated obstacle-free environments. One of the most recent works, [30], also focuses on navigation via obstacle avoidance using raw depth input image by operating in a partially known environment. Seven different environments with varying complexities (i.e. shapes, locations, etc, of obstacles) were used to train the underlying DQN network. It was shown to be able to generalize better in environments the agent has not been trained before. However, in this work a rough path to the goal is known beforehand, although the agents did not have prior knowledge regarding the obstacle locations. The authors of [4] utilized DQN, Double DQN and Dueling architecture for the same control objective as [30]. Similar to [30], visual-based inputs from the cameras were used for learning. The Double-DQN was shown to perform better as opposed to the other two candidates. However, binding to real-time rendering was identified as a bottleneck that affected the training time. Both [30] and [4] were carried out in simulated environments using AirSim to implement the underlying algorithms.

Furthermore, in past research, the application of RL to perform high-level control tasks collaboratively in multi-agent environments has also been investigated. In [31], a Geometric Reinforcement Learning (GRL) algorithm was proposed to resolve the path planning problem in UAVs. This approach incorporates distance information in learning to reduce the time needed for the quadcopter to reach a target. As a single reward matrix is maintained, this approach accommodates both single and multi-quadcopter systems. Through collaboratively sharing information, the GRL algorithm outperformed the Q-learning algorithm by facilitating better path planning in terms of the risk exposure to obstacles and path length. In [32], an online distributed cooperative trajectory planning algorithm for object searching and tracking was proposed. The proposed Quantum Probability Model (QPM) describes the partially observable target positions. This model is used in conjunction with the Upper Confidence Tree (UCT) [33] algorithm to determine the optimal route, where a partner learning model is employed to manage the non-stationary issues in distributed RL. The authors observed the approach to attain a successful tracking ratio. To the authors' knowledge, very few research work has been done as of now, which incorporates cooperative high-level control using RL for quadcopters, particularly for the purpose of navigation.

Chapter 3

Methodology

3.1 Architecture

Figure 3.1 provides a high-level overview of the architecture of the overall system. Unreal Engine (UE4) was used to design and simulate 3D environments as well as the underlying quadcopter physics. AirSim acts as an interface which transfers the visual input data and control actions from UE4 to Python and vice versa. The interactive wrapper environment was developed to add a layer of abstraction and it provides a generalized environment for benchmarking future algorithms. PyTorch was utilized to implement DQN and MADDPG algorithms in the single-agent and multi-agent domains respectively.

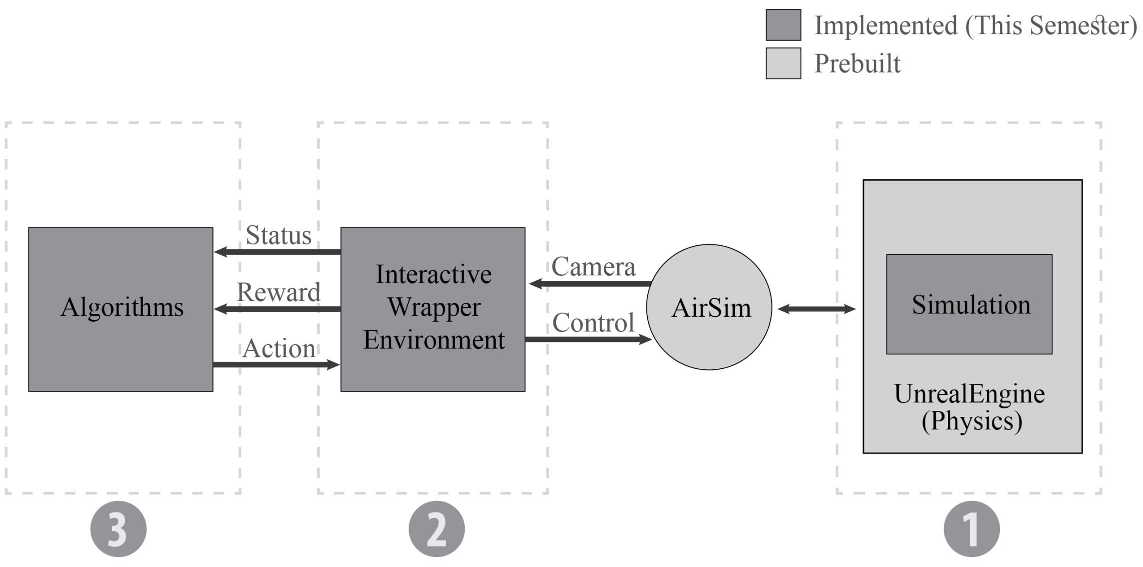


Fig. 3.1 High-level Architecture

3.2 Simulation Platforms For Quadcopters

Several issues exist within the context of addressing autonomous UAV control problems. In order to learn useful behaviours utilizing RL techniques, a substantial amount of training data is required within diverse environment settings and circumstances. Moreover, the training phase is often a costly and time-consuming process intertwined with unpredictability [34] [35].

Thus, simulation provides a feasible alternative to avoid the expensive and time-consuming process of training and testing autonomous quadcopters in the real world. For instance, unlike in the real world, collisions in a simulated environment is neither catastrophic nor costly. Moreover, it can provide useful information to further refine the design of the system. The basic requirements of a simulator for this research comprise of a flight dynamics model, a system model (i.e. a mathematical model of the UAV system), a control system, and other application-specific features (E.g. a module for data analysis) [36].

The credibility of a given simulator can be assessed on the basis of several factors. For instance, how closely the model resembles the physics and dynamics of the real system (which determines the accuracy of the model), whether meaningful and realistic data can be generated and captured, the ability to integrate generic flight control hardware and software protocols within the simulation, etc. As this paper focuses on the use of RL for control, considerable attention must also be given to the feasibility in interfacing the simulation tool with ML models for training or/and control. These factors determine how well all the learnings and inferences that occur in simulation can eventually be transferred to facilitate the real-world operation of a quadcopter [36] [4].

In order to facilitate the project design, three software simulation platforms which accommodate the above mentioned requirements were considered. Subsequently, AirSim [34] was selected over GymFC [37] and Gazebo+ROS [35]. The justification for the selection was as follows.

- GymFC [37] is an environment which employs Gazebo [38] high-fidelity physics simulator and OpenAI Gym [39] interfaces for developing and training controllers for UAVs using RL. However, this environment focuses on the highly specific task of attitude control, with no consideration to guidance and/or navigation control. Moreover, past work carried out in this simulated environment have not validated the behaviour of the designed controllers against real quadcopter hardware [37] [40]. Neuroflight [41], an enhanced version of GymFC addresses some of the limitations of this simulation platform. However, at the time of this research, Neuroflight was currently in its experimental phase.

- Gazebo is a 3D modeling and rendering tool, capable of accurately simulating multi-robot environments including rigid-body physics and generating realistic sensor feedback[42]. ROS (Robot Operating System)[43] is an open-source distributed robot software framework which provides a structured communication layer (middleware) on top of the host OS, to facilitate the development of robot applications. However, within the Gazebo+ROS [35] setup, each implementation remains unique. Thus, benchmarking becomes a potential issue. Moreover, building large-scale, visually rich environments can also be difficult.

3.2.1 AirSim

Microsoft AirSim (Aerial Informatics and Robotics Simulation) [34] is a simulation platform for developing and testing autonomous vehicles. It has been developed as a plugin for Unreal Engine, which acts as the visual rendering platform facilitating the creation of detailed, photo-realistic environments. Through the generation of rich and diverse training datasets, AirSim supports the integration of ML techniques such as RL to develop algorithms aimed at perception and control tasks.

The architecture of AirSim comprises of the following core components; the environment model, vehicle model, physics engine, sensor models, public API layer, rendering interface, and an interface layer for vehicle firmware.

AirSim exposes Application Program Interfaces (APIs) to programmatically interact with the UAV in simulation, thus facilitating the retrieval of data such as state information, output from sensor streams, etc. as well as the transmission of commands for quadcopter control. The design of these APIs is such that the underlying functionality is hidden to ensure that the companion computer is oblivious as to whether it is being run in simulation or real hardware. Thus, the algorithms developed and tested in the simulator can be seamlessly deployed to a physical quadcopter hardware.

The experiments conducted by Shah et al. [34] qualitatively assess how near realistic the simulated aerodynamics behaviour in AirSim is to that of a quadcopter operated in the real world. In recent work, AirSim has been used as the simulation platform to implement control tasks for quadcopters in conjunction with RL, particularly in navigation and obstacle avoidance [44] [45] [46] [47].

Chapter 4

Experimental Setup and Implementation

4.1 Simulation Environment

4.1.1 Setting up the project environment

- *Container setup in a high-performance server.* Once the credentials to a fresh container in a high-performance server were obtained, SSH settings were configured in the local machines. Moreover, GPU drivers and essential packages were installed accordingly.
- *Python development environment.* PyCharm IDE was installed and configured with Python 3.6 on local machines.
- *Unreal Engine 4 (UE4) setup*
  1. Windows: Installed using the provided binary installer from UE4 official website.
  2. Linux: As a binary installer for Linux were not provided by the developers of Unreal Engine (UE4), a binary had to be compiled from the provided source code.
- *AirSim.* AirSim source code was obtained from the GitHub repository and compiled and installed following the instructions provided in the documentation. Thereby the Python API package for AirSim was also installed.

4.1.2 Building custom maps

As mentioned in Section 3.3, in order to ensure the viability of the developed algorithm in practical applications, a substantial amount of data is required within diverse environment settings during the training phase. Therefore, the diversity and size of the training data, directly influences the performance of the algorithm. In order to accommodate this requirement, a few custom maps with varying complexities were built using unreal engine (Figures 4.1 - 4.5).

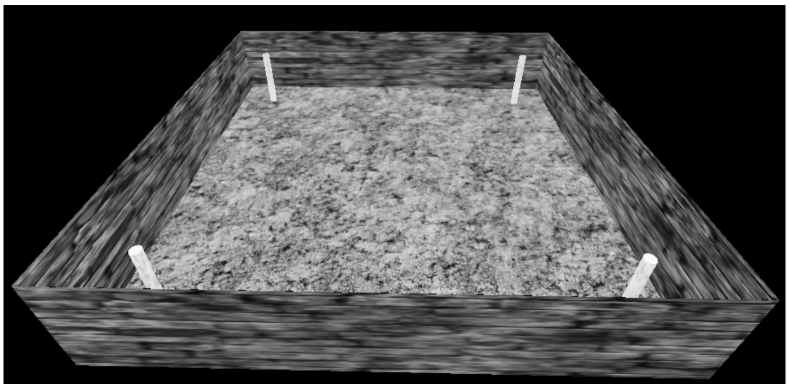


Fig. 4.1 Open environment with goals only

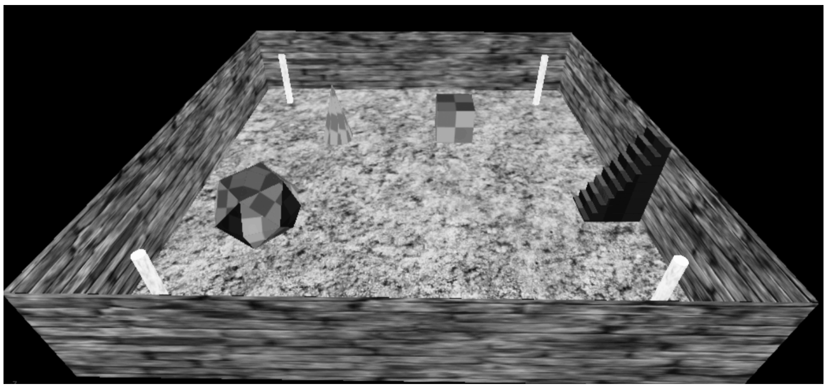


Fig. 4.2 Environment with goals and obstacles

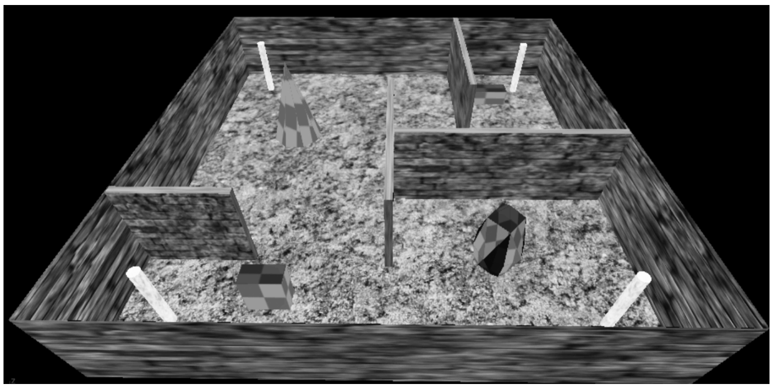


Fig. 4.3 Maze environment with goals and obstacles

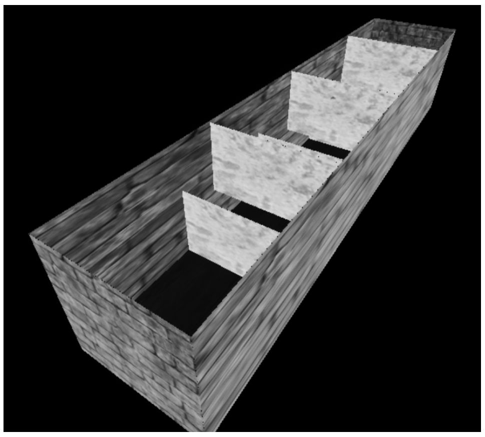


Fig. 4.4 Corridor with partitions on the top and bottom

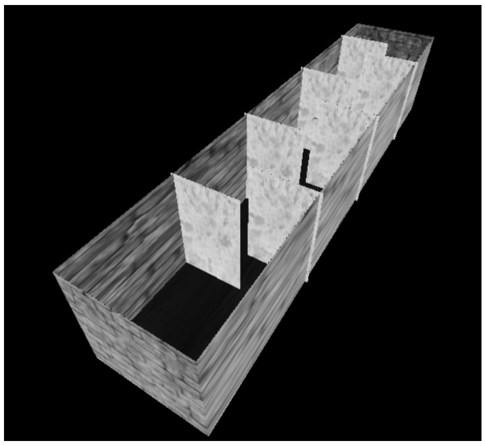


Fig. 4.5 Corridor with partitions on the left and right

However, given the limited time duration (Semester 7), only two custom maps were used to train the algorithm in addition to a pre-built map (Figure 4.6).

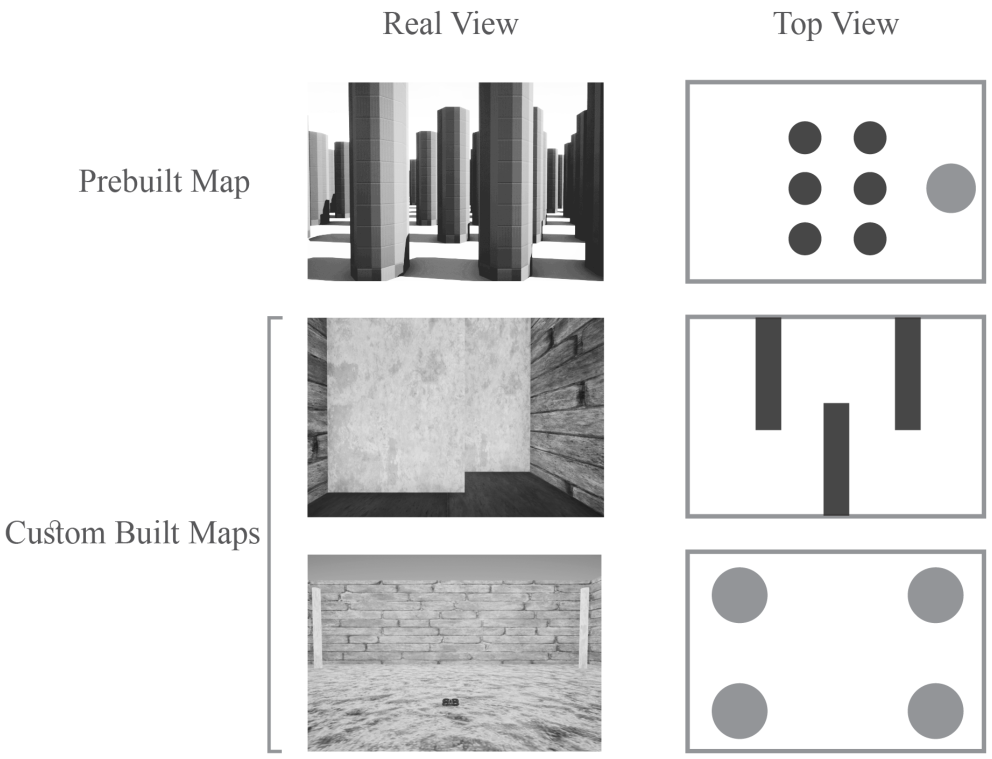


Fig. 4.6 Maps in which the algorithm was trained on

4.2 Interactive Wrapper Environment

In RL the simulating environment's complexity plays a significant role in evaluating the environment. The more complex an environment becomes, the more knowledgeable the evaluator of an algorithm is expected to be regarding the dynamics of the environment. When testing over a variety of environments, the necessity to have an understanding of the dynamics of any given environment becomes a bottleneck. Thus, to avoid such issues, wrappers such as OpenAI Gym [39] has integrated generalized and relatively simplified wrapper formats in order to black box the dynamics of the associated environments. For instance, the use of attributes such as state, action reward, and their dimensions as opposed to specifying their nature. By following a well-known standard of OpenAI Gym, several wrappers were developed to add a layer of abstraction to the environment so that it can be used within the context of the current problem as well as enabling it to be utilized as an environment to benchmark for future settings and algorithms.

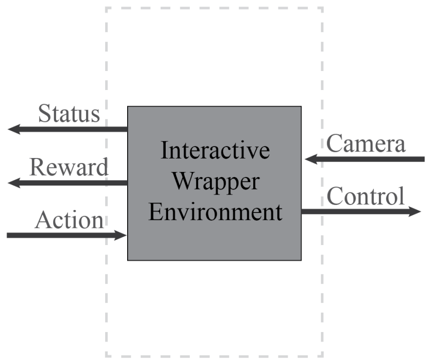


Fig. 4.7 Interactive Wrapper Environment

4.2.1 Data Manipulation

Corridor with partitions on the left and right (Figure 4.5)

In this particular setting, only the depth visual from the front camera was pre-possessed by the wrapper. This was carried out by increasing the quality of the image using a PSNR (Peak Signal to Noise Ratio). Thereafter, it was transformed into an 84×84 image with values ranging between 0-255 and a state matrix was returned (i.e. a Gym data type; Box of size (84, 84)). The actions allowed for the quadcopter were defined as follows. Moving forward was considered as1, turning left by 30°as 2 and turning right by 30°as 3. Thus, for each action, a single numerical digit among the three discrete values; 0, 1, 2 (Discrete(3) in terms of the Gym data type) was returned. Rewards were designed such that in the case of a collision, the agent would receive a -100 reward and a +100 reward for reaching the goal. In addition, the agent would get a -1 reward for each time step and a positive reward proportional to how close the agent is getting towards the goal. The optimal strategy of this environment is to reach the goal without colliding with the walls/partitions.

Environment with goals only (Figure 4.1)

This setting accommodated the operation of multiple agents within the same environment. Here, the issue of partial observability is high due to the limited view of each agent. Thus, in order to mitigate this problem to a certain extent, the depth visuals from all four cameras (i.e. the front, left, right and back) were obtained and subjected to the same transformation as aforementioned in the corridor scenario. Thereafter, a state was represented as a 3D matrix of shape (4, 84, 84) (in terms of the Gym data type, this is a Box input of shape (4, 84, 84)). As the aim was to test the multi-agent cases using the MADDPG algorithm (further details mentioned in Section 4.3) and as DDPG (Deep Deterministic Policy Gradients) can only work with continuous action spaces, the action space was designed as follows. The action space is returned as a vector of shape (2,1) (i.e. Gym Box data type of size (2,1)) with both elements taking a real number between (-1,1). The first element corresponds proportionally to 360 which indicates the angle the agent must rotate by. The second served as a threshold to indicate whether the agent should move forward by a certain distance for one second or not. The rewards were designed to give negative values inversely proportional to the distance to the other agents. A reward of -1 was given for every time step and a +100 of reward was given upon reaching any one of the four goals. The optimal strategy of this environment is to reach one of the four goals while maintaining as much distance as possible from the other agents.

4.3 Algorithms

4.3.1 Single-Agent Setting - DQN

Deep Q-learning is a modification of Q-learning where instead of storing the Q-value in a tabular form as a running average for each action-state pair, a Neural Network-based model is used to learn a function representation given a particular state-action pair. A notable disadvantage of a tabular Q-learning model is that as the state space grows, the memory needed to store these values grows as well. Thus, the use of a neural network enables the prediction of the Q-values corresponding to all action values given a single state. This removes the need for several passes through the neural network. As the objective is to learn a model that generalizes the value function, data from each step is stored in a queue-structured memory (known as the experience replay) and a batch is sampled uniformly at random and used to train the network. This minimizes the possibility of overfitting to a local optimum. Furthermore, as in Equation 1.5, the error function that optimizes the Q-value function's parameters depends directly on the same Q-value function. This causes a non-stationary issue in learning (due to non-stationary learning targets). In order to avoid this, two networks are used. Namely, the target network and the current network, where the current network is updated at each time step with the values from the target network. The target network is either adjusted to match the current network once every certain number of steps (i.e. a hard update) or is made to follow the current network by some soft update rule.

In the single-agent case, the DQN algorithm [11] was used to address the problem of quadcopter navigation. A single depth visual was obtained from the front camera of the quadcopter. Once the intensity of the image was increased, it was re-scaled to an 84×84 pixel image and was fed into the network. The network as in Figure 4.8 consisted of several convolution layers prior to the feed-forward layers. In the case of navigation, the possible actions that were allowed for the quadcopter were moving straight, turning left by 30°and turning right by 30°. DQN was implemented using the Keras-RL library with a TensorFlow backend. The parallelization feature of the TensorFlow was utilized to speed up the training. However, as mentioned Section 4.4, one of the largest bottlenecks was the time consumed for a single sample of the simulation to play out. DQN was tried on both the custom made simple corridor enviornment and the pre-build map of several poles with their pre-built wrapper who's ideal strategy would be to navigate to a goal without colliding with any of the intermediate poles.

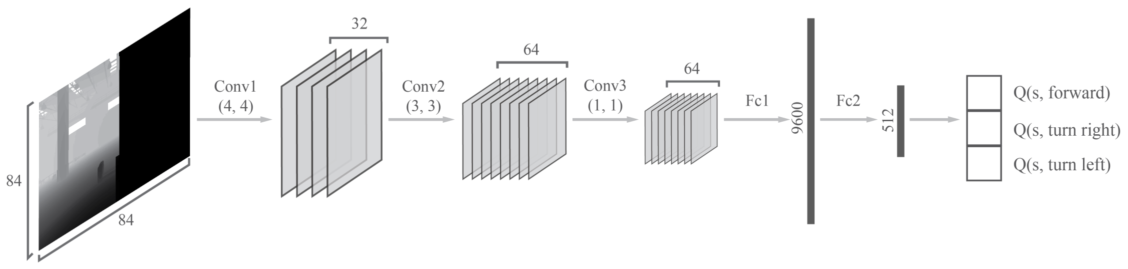


Fig. 4.8 Neural network architecture for the DQN Q-value function

4.3.2 Multi-Agent Setting - MADDPG

As mentioned in Section 1.2, two existing algorithms for multi-agent RL through the use of actor-critic methods are COMA and MADDPG. The major difference in COMA, when compared to MADDPG, is its structure to infer an agent's contribution towards the global reward, obtained within a global reward setting, based on the notion of a difference reward. In MADDPG there is no explicit strategy to infer the credit of a certain agent given a particular reward. For this particular phase of the project (Semester 7), given the simplicity of its reward structure, MADDPG was chosen as a starting point for the exploration of multi-agent scenarios using RL.

MADDPG [1] is an extension of the Deep Deterministic Policy Gradient [13], which in turn is an extension of the Deterministic Policy Gradient [48], with the use of deep neural networks. Here, the notation of $\mathbf{Q_i}\left(\mathbf{x}, \mathbf{a_1}, \ldots, \mathbf{a_N}\right)$ is used to denote a centralised critic. The formula for the policy gradient denoted by Equation 1.4 can be modified as following with this centralized critic for an agent i in the context of MADDPG.

$$\nabla_{\theta_i} J(\theta_i) = \mathbb{E}_{s\sim p^{\mu}, a_i \sim \pi_i} \left[\nabla_{\theta_i} \log \pi_i (a_i | o_i) \, Q_i^{\pi} (\mathbf{x}, a_1, \ldots, a_N) \right] \tag{4.1}$$

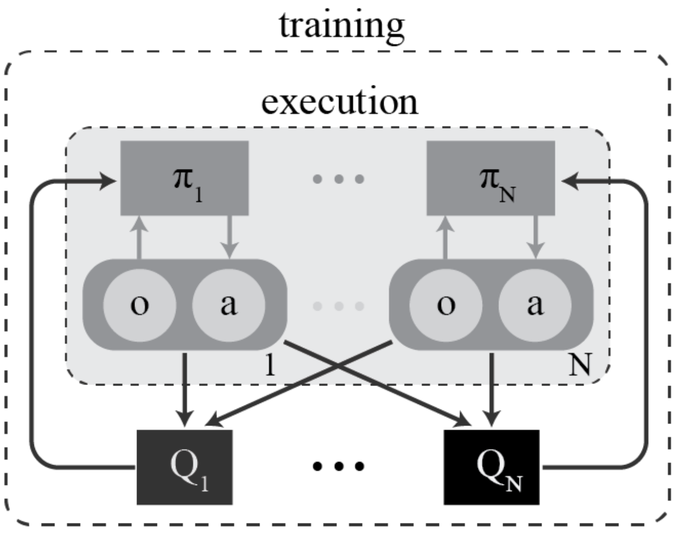


Fig. 4.9 A graphical representation of the MADDPG architecture as given in [1]

DDPG can only be applied in situations with continuous action spaces. As far as the action space is concerned, a 1-dimensional vector with two elements was used. One element denoted the angular movement while the other denoted the transitional movement of the given quadcopter. As the partial observability of the inputs from the environment was high, four depth visuals from cameras of all four directions (i.e. front, left, right and back) were used. These images were intensified and then transformed into an 84×84 image and were fed as observations into both the actor and critic networks. An additional information of the action was fed into the actor-network and the agents were allowed to learn with independent rewards for each agent. The implementation was carried out from scratch using PyTorch and its parallelization feature was leveraged. MADDPG was trained on the Poles-Only environment (Figure 4.1).

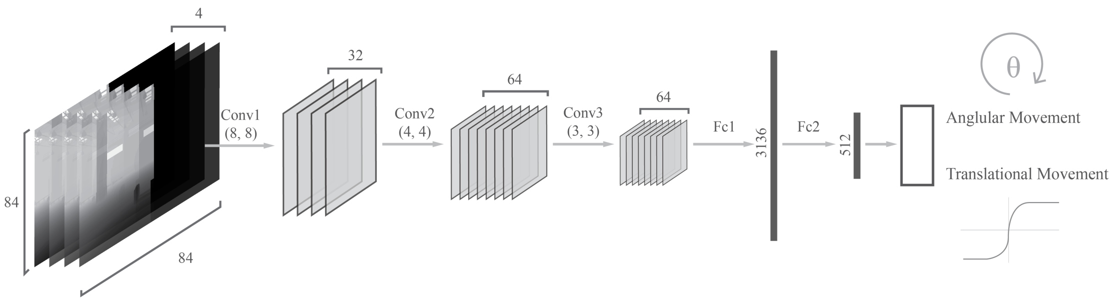


Fig. 4.10 Architecture of an actor network in the MADDPG model

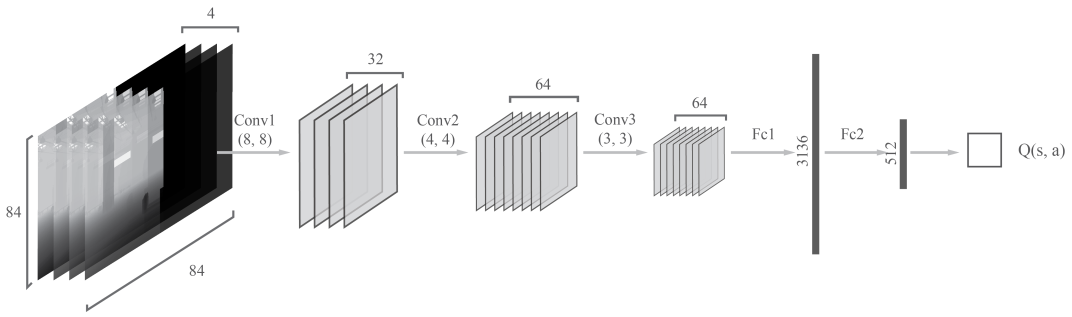


Fig. 4.11 Architecture of the critic network in the MADDPG model

4.4 Pitfalls and Workarounds

- *Environment setup.* Although the necessary instructions were followed to obtain the source code of UE4, there was an issue with obtaining it from the Epic Games official GitHub repository due to a technical fault in their permission procedure. Thus, the source code of Unreal Engine 4.18 had to be obtained from another contributor's private repository. Moreover, certain issues related to building due to missing files were resolved by creating symbolic links wherever necessary. Once the binary was successfully built from the source code, a segmentation fault occurred upon trying to run Unreal Engine on the server. The binary was built multiple times and considerable time was spent on trying to identify the underlying issue. Eventually, the root cause was identified to be a GPU driver mismatch on the container in the high-performance server.

- *Building custom maps.*

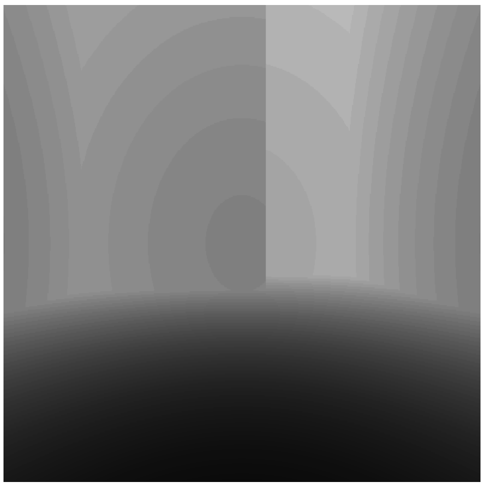


Fig. 4.12 An unclear depth image

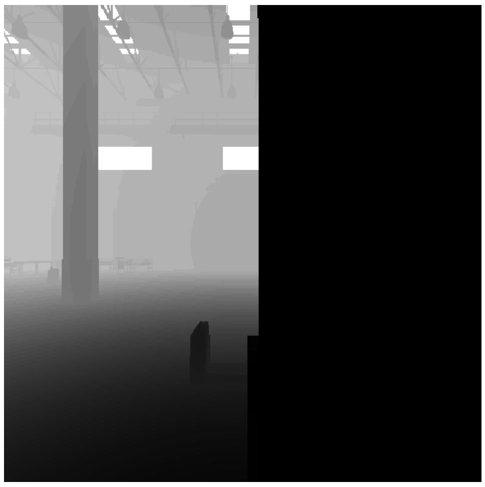


Fig. 4.13 A clear depth image

Unreal Engine is a commercial-level game engine consisting of complex materials and lightning configurations to obtain photo-realistic environments. However, during the initial design of the custom maps, due to the lack of experience with the tool, considerable attention was not given to configure complex parameters that dictate the surface materials, textures, etc. This resulted in faulty depth images (Figure 4.12) (which were taken as visual input by the agent). It was observed to have a negative impact on the accuracy and the training time of the model. Thus, the environment properties of the maps had to be fine-tuned accordingly (Figure 4.13). This was a crucial learning point which should be taken into account during future development work.

- *AirSim API.* During the training phase, the altitude of the quadcopter was observed to decrease with each time step. Thus, episodes had to be terminated prematurely. This adversely affected the performance of the underlying algorithm. Further investigation is required to identify a potential solution for this issue.

Chapter 5

Results and Analysis

5.1 Results

Autonomous control of quadcopter was attempted in three separate settings when using DQN in the single-agent domain; two occasions in the pre-built map (as shown in Figure 4.6) and one in the simple corridor map (Figure 4.5). In the case of the multi-agent domain, MADDPG was used in the open environment with only goals (Figure 4.1). The following section discusses the observations that were obtained upon investigating each of these cases.

1. In the pre-built map scenario with multiple poles, the learning was relatively faster compared to that in the simple corridor setting. Moreover, it was observed that the policy learned by the agent in a smaller area of this map was sufficient to generalize within most of the larger area as opposed to that in the simple corridor setting.

2. In the simple corridor scenario, the quadcopter was observed to spend more time on turning/yawing relative to that in the pre-built map. A visual depiction of the three actions taken by the agent in each map is shown in the following figures (Figure 5.1 - 5.2)

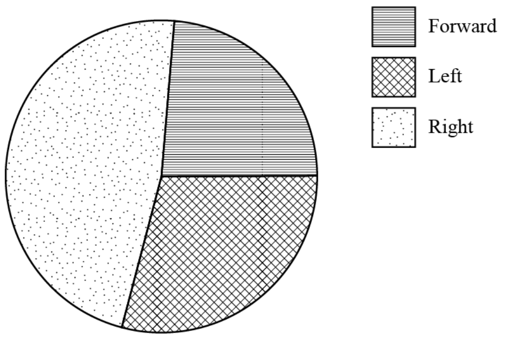


Fig. 5.1 Simple corridor

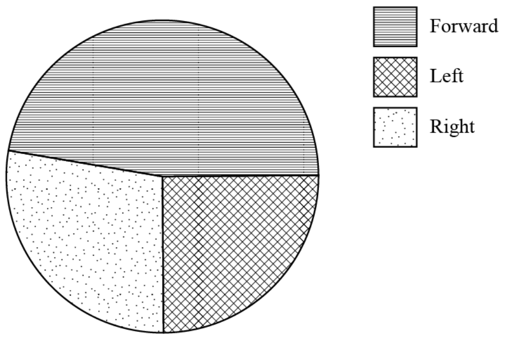


Fig. 5.2 Pre-built map

3. The variance of actions taken by the agent in all three cases did not show a significant difference.

5.2 Analysis

The following section analyses the respective results obtained in the given order.

1. The pre-built map was mostly homogeneous in nature. That is, a certain section could be utilized to generalize most of the map. For instance, the gaps between any two poles do not provide room for a potential collision. However, in comparison, the simple corridor had some diversity to it. For instance, the partitions along the corridor were either to the left or right and there was a more restrictive border for the quadcopter. These factors can be attributed to the ability of the agent to learn much faster to navigate in the pre-built environment setting as opposed to the simple corridor setting.

2. The fact that the agent spent a considerable amount of time on yawing (as opposed to the other actions) in the simple corridor map may be due to the complexity and restrictive nature of the map compared to the pre-built map.

3. As an actor-critic method was considered for the case of the multi-agent domain, compared to high variance policy gradient methods, the observed variance was not significantly high (i.e. theoretically using function approximation to determine the value function as opposed to equating it to an average of several observed rewards, is proven to reduce the associated variance).

Chapter 6

Conclusions and Future Works

This study has explored the application of Reinforcement Learning in the context of autonomous quadcopter control. Particularly, the implementation of the DQN algorithm in a single-agent setting and MADDPG algorithm in a multi-agent setting. In multi-agent settings, where multiple quadcopters can be found to be operating within the same environment, there exist scenarios with individualized reward structures as well a single global reward structure. Upon using a single global reward structure, it is difficult to explicitly determine how each agent's actions contribute to the global reward. Thus, MADDPG is only compatible with an individual reward structure as it fails to identify individual agent contributions in single global reward setting. Existing algorithms such as COMA address this issue to a certain extent, by using a relatively complex global reward inference structure.

In terms of future work, one potential improvement that can be done would be to extend on COMA to facilitate efficient credit assignments to individual agents in a single global reward setting. In addition, as of now, the algorithm was validated against simple and primitive environment settings. A potential avenue would be to further optimize the algorithm in a large, complex and diversified environment such that it can be applied to a real world application. One such application can be in wildlife surveillance, where Supervised Learning can integrated, so as to accommodate the additional functionality of object detection (E.g. poachers, animals, etc.).

References

[1] R. Lowe, Y. Wu, A. Tamar, J. Harb, P. Abbeel, and I. Mordatch, "Multi-Agent Actor-Critic for Mixed Cooperative-Competitive Environments," *arXiv e-prints*, p. arXiv:1706.02275, Jun 2017.

[2] P. J. Antsaklis, K. M. Passino, and S. Wang, "An introduction to autonomous control systems," *IEEE Control Systems Magazine*, vol. 11, no. 4, pp. 5–13, 1991.

[3] K. G. Vamvoudakis, P. J. Antsaklis, W. E. Dixon, J. P. Hespanha, F. L. Lewis, H. Modares, and B. Kiumarsi, "Autonomy and machine intelligence in complex systems: A tutorial," in *2015 American Control Conference (ACC)*, pp. 5062–5079, IEEE, 2015.

[4] K. Kersandt, "Deep reinforcement learning as control method for autonomous uavs," Master's thesis, Universitat Politècnica de Catalunya, 2018.

[5] B.-Q. Huang, G.-Y. Cao, and M. Guo, "Reinforcement learning neural network to the problem of autonomous mobile robot obstacle avoidance," in *2005 International Conference on Machine Learning and Cybernetics*, vol. 1, pp. 85–89, IEEE, 2005.

[6] R. S. Sutton, A. G. Barto, *et al.*, *Introduction to reinforcement learning*, vol. 2. MIT press Cambridge, 1998.

[7] Z. Ziyang, X. Dongjing, and G. Chen, "Cooperative search-attack mission planning for multi-uav based on intelligent self-organized algorithm," *Aerospace Science and Technology*, vol. 76, 02 2018.

[8] J. Modares, F. Ghanei, N. Mastronarde, and K. Dantu, "Ub-anc planner: Energy efficient coverage path planning with multiple drones," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 6182–6189, IEEE, 2017.

[9] Y. Liu, H. Liu, Y. Tian, and C. Sun, "Reinforcement learning based two-level control framework of uav swarm for cooperative persistent surveillance in an unknown urban area," *Aerospace Science and Technology*, p. 105671, 2020.

[10] R. W. Beard and T. W. McLain, "Multiple uav cooperative search under collision avoidance and limited range communication constraints," in *42nd IEEE International Conference on Decision and Control (IEEE Cat. No. 03CH37475)*, vol. 1, pp. 25–30, IEEE, 2003.

[11] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[12] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine learning*, vol. 8, no. 3-4, pp. 229–256, 1992.

[13] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. M. O. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *CoRR*, vol. abs/1509.02971, 2015.

[14] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *ArXiv*, vol. abs/1707.06347, 2017.

[15] K. Tuyls and G. Weiss, "Multiagent learning: Basics, challenges, and prospects," *Ai Magazine*, vol. 33, no. 3, pp. 41–41, 2012.

[16] J. van der Wal, *Stochastic Dynamic Programming: Successive Approximations and Nearly Optimal Strategies for Markov Decision Processes and Markov Games*. Mathematical Centre tracts, Mathematisch Centrum, 1981.

[17] L. Bu, R. Babu, B. De Schutter, *et al.*, "A comprehensive survey of multiagent reinforcement learning," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 38, no. 2, pp. 156–172, 2008.

[18] D. T. Nguyen, A. Kumar, and H. C. Lau, "Credit assignment for collective multiagent rl with global rewards," in *Advances in Neural Information Processing Systems*, pp. 8102–8113, 2018.

[19] Y.-H. Chang, T. Ho, and L. P. Kaelbling, "All learning is local: Multi-agent learning in global reward games," in *Advances in neural information processing systems*, pp. 807–814, 2004.

[20] J. Foerster, G. Farquhar, T. Afouras, N. Nardelli, and S. Whiteson, "Counterfactual Multi-Agent Policy Gradients," *arXiv e-prints*, p. arXiv:1705.08926, May 2017.

[21] T. T. Nguyen, N. D. Nguyen, and S. Nahavandi, "Deep reinforcement learning for multi-agent systems: A review of challenges, solutions and applications," *arXiv preprint arXiv:1812.11794*, 2018.

[22] G. V. Raffo, M. G. Ortega, and F. R. Rubio, "An integral predictive/nonlinear h control structure for a quadrotor helicopter," *Automatica*, vol. 46, no. 1, pp. 29–39, 2010.

[23] H. Bolandi, M. Rezaei, R. Mohsenipour, H. Nemati, and S. M. Smailzadeh, "Attitude control of a quadrotor with optimized pid controller," *Intelligent Control and Automation*, vol. 4, no. 03, p. 335, 2013.

[24] G. H. Elkaim, F. A. P. Lie, and D. Gebre-Egziabher, *Principles of guidance, navigation, and control of UAVs*, pp. 347–380. Springer, 2015.

[25] A. Rodriguez-Ramos, C. Sampedro, H. Bavle, P. De La Puente, and P. Campoy, "A deep reinforcement learning strategy for uav autonomous landing on a moving platform," *Journal of Intelligent & Robotic Systems*, vol. 93, no. 1-2, pp. 351–366, 2019.

[26] H. X. Pham, H. M. La, D. Feil-Seifer, and L. V. Nguyen, "Autonomous uav navigation using reinforcement learning," *arXiv preprint arXiv:1801.05086*, 2018.

[27] Z. Yijing, Z. Zheng, Z. Xiaoyi, and L. Yang, "Q learning algorithm based uav path learning and obstacle avoidence approach," in *2017 36th Chinese Control Conference (CCC)*, pp. 3397–3402, IEEE, 2017.

[28] R. Polvara, M. Patacchiola, S. Sharma, J. Wan, A. Manning, R. Sutton, and A. Cangelosi, "Toward end-to-end control for uav autonomous landing via deep reinforcement learning," in *2018 International Conference on Unmanned Aircraft Systems (ICUAS)*, pp. 115–123, IEEE, 2018.

[29] M. Shaker, M. N. Smith, S. Yue, and T. Duckett, "Vision-based landing of a simulated unmanned aerial vehicle with fast reinforcement learning," in *2010 International Conference on Emerging Security Technologies*, pp. 183–188, IEEE, 2010.

[30] E. Camci and E. Kayacan, "End-to-end motion planning of quadrotors using deep reinforcement learning," *arXiv preprint arXiv:1909.13599*, 2019.

[31] B. Zhang, Z. Mao, W. Liu, and J. Liu, "Geometric reinforcement learning for path planning of uavs," *Journal of Intelligent & Robotic Systems*, vol. 77, no. 2, pp. 391–409, 2015.

[32] T. Wang, R. Qin, Y. Chen, H. Snoussi, and C. Choi, "A reinforcement learning approach for uav target searching and tracking," *Multimedia Tools and Applications*, vol. 78, no. 4, pp. 4347–4364, 2019.

[33] L. Kocsis and C. Szepesvári, "Bandit based monte-carlo planning," in *European conference on machine learning*, pp. 282–293, Springer, 2006.

[34] S. Shah, D. Dey, C. Lovett, and A. Kapoor, "Airsim: High-fidelity visual and physical simulation for autonomous vehicles," in *Field and service robotics*, pp. 621–635, Springer, 2018.

[35] J. Meyer, A. Sendobry, S. Kohlbrecher, U. Klingauf, and O. Von Stryk, "Comprehensive simulation of quadrotor uavs using ros and gazebo," in *International conference on simulation, modeling, and programming for autonomous robots*, pp. 400–411, Springer, 2012.

[36] A. Mairaj, A. I. Baba, and A. Y. Javaid, "Application specific drone simulators: Recent advances and challenges," *Simulation Modelling Practice and Theory*, 2019.

[37] W. Koch, R. Mancuso, R. West, and A. Bestavros, "Reinforcement learning for uav attitude control," *ACM Transactions on Cyber-Physical Systems*, vol. 3, no. 2, p. 22, 2019.

[38] N. Koenig and A. Howard, "Design and use paradigms for gazebo, an open-source multi-robot simulator," in *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)(IEEE Cat. No. 04CH37566)*, vol. 3, pp. 2149–2154, IEEE, 2004.

[39] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," *arXiv preprint arXiv:1606.01540*, 2016.

[40] U. H. Ghouri, M. U. Zafar, S. Bari, H. Khan, and M. U. Khan, "Attitude control of quad-copter using deterministic policy gradient algorithms (dpga)," in *2019 2nd International Conference on Communication, Computing and Digital systems (C-CODE)*, pp. 149–153, IEEE, 2019.

[41] W. Koch, R. Mancuso, and A. Bestavros, “Neuroflight: Next generation flight control firmware,” *arXiv preprint arXiv:1901.06553*, 2019.

[42] W. Qian, Z. Xia, J. Xiong, Y. Gan, Y. Guo, S. Weng, H. Deng, Y. Hu, and J. Zhang, “Manipulation task simulation using ros and gazebo,” in *2014 IEEE International Conference on Robotics and Biomimetics (ROBIO 2014)*, pp. 2594–2598, IEEE, 2014.

[43] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, “Ros: an open-source robot operating system,” in *ICRA workshop on open source software*, vol. 3, p. 5, Kobe, Japan, 2009.

[44] J. Luo, S. Green, P. Feghali, G. Legrady, and Ç. K. Koç, “Reinforcement learning and trustworthy autonomy,” in *Cyber-Physical Systems Security*, pp. 191–217, Springer, 2018.

[45] T.-C. Wu, S.-Y. Tseng, C.-F. Lai, C.-Y. Ho, and Y.-H. Lai, “Navigating assistance system for quadcopter with deep reinforcement learning,” in *2018 1st International Cognitive Cities Conference (IC3)*, pp. 16–19, IEEE, 2018.

[46] S.-Y. Shin, Y.-W. Kang, and Y.-G. Kim, “Automatic drone navigation in realistic 3d landscapes using deep reinforcement learning,” in *2019 6th International Conference on Control, Decision and Information Technologies (CoDIT)*, pp. 1072–1077, IEEE, 2019.

[47] E. Bondi, D. Dey, A. Kapoor, J. Piavis, S. Shah, F. Fang, B. Dilkina, R. Hannaford, A. Iyer, L. Joppa, *et al.*, “Airsim-w: A simulation environment for wildlife conservation with uavs,” in *Proceedings of the 1st ACM SIGCAS Conference on Computing and Sustainable Societies*, p. 40, ACM, 2018.

[48] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, “Deterministic policy gradient algorithms,” in *International Conference on Machine Learning*, pp. 387–395, 2014.