Implementation of Deep Reinforcement Learning for Collision Prevention in Quadrotor

Dissertation Report

submitted in fulfilment of the requirements

for the award of degree

Integrated B.Tech + M.Tech

in

Computer Science and Engineering

(Specialization: Artificial Intelligence and Robotics)

Submitted By

Rishabh Goel

17/ICS/073

Under the supervision of

Dr. Navaid Z. Rizvi

To

Department of Computer Science

School of Information and Communication Technology

Gautam Buddha University

June 2022

Declaration

I hereby declare that the dissertation work titled **"Implementation of Deep Reinforcement Learning for Collision Prevention in Quadrotor,"** submitted to Gautam Buddha University, is a record of an original work done by me under the guidance of **Dr. Navaid Z. Rizvi**. This dissertation work is submitted under the fulfilment of the award of the degree of Integrated B.Tech + M.Tech in Computer Science and Engineering with specialization in Artificial Intelligence and Robotics.

Name of student: Rishabh Goel
Roll Number: 17/ICS/073
Signature: .....
Date: .....
Place: Greater Noida

Certificate

This is to certify that **Rishabh Goel** has carried out the Dissertation Thesis titled, **"Implementation of Deep Reinforcement Learning for Collision Prevention in Quadrotor,"** for the award of Integrated B.Tech + M.Tech in Computer Science and Engineering with specialization in Artificial Intelligence and Robotics, from Gautam Buddha University, Greater Noida(UP), under my supervision. The thesis embodies result of original work and studies carried out by the student himself and the contents of this project do not form the basis of the award of any other degree of the candidate to anybody else.

. . . . . Date & Place . . . . . .

Dr. Navaid Z. Rizvi
Department of Computer Science,
School of Information and Communication Technology,
Gautam Buddha University,
Greater Noida(UP)

Acknowledgement

First and foremost, I would like to thank my supervisor, Dr. Navaid Z. Rizvi, for his trust and unwavering support. I am grateful for his kind assistance towards the difficulties that I have faced. His infinite passion in research work inspired me to work hard. I also extend by gratitude towards Dr. Vimlesh K. Ray, Dr. Ana Kumar and Mrs. Snehlata Gautam for their support towards refining my dissertation topic.

I am also grateful to my friends for their support. I am also extremely grateful to various people in various community forums who were able to provide me assistance to never ending questions that arose during this thesis work. Finally, a special thanks to my family members for their endless morale boost. This thesis wouldn't be possible without all of their support.

Abstract

Reinforcement Learning has been implemented on diverse tasks and has proved itself as a possible contender for developing potential future applications using machine learning. Machine learning is categorised as follows; Supervised Learning, Unsupervised Learning and Semi-supervised Learning also known as Reinforcement Learning. In Reinforcement Learning, agents are designed to interact with the environment by choosing appropriate actions and there is an associated reward with each action. This interaction is what enables the agent to learn *correct* behaviour. Here, the *correct* behaviour is defined by the researcher to meet their requirements. The agent does not know right from wrong and relies on the provided reward to optimize its performance.This agent-environment interaction has been pursued to develop an autonomous Quadrotor that should be able to prevent collisions with the surroundings.

A Quadrotor, as the name suggests, consists of four propellers/rotors placed symmetrically at four corners of an imaginary square frame. A Quadrotor is controlled by applying individual torque to each rotor. In the proposed research, the quadrotor is being controlled by the reinforcement learning or more specifically deep reinforcement learning agent. The quadrotor contains an Inertial Measurement Unit and multiple proximity sensors, also known as rangefinders, which help in detecting the distance from nearby walls. The sensor data is processed accordingly and is passed on to the Deep Reinforcement Learning agent, whose output contains four values, where each value represents a rotor thrust in the range of 0 to 1. The proposed solution is a low-level control structure where the output rotor thrust is directly mapped to each rotor. The performance measure of the trained agent has been quantified with the amount of time it stays online, that is, manages a collision-free flight.

List of Abbreviations

RL . . . . . Reinforcement Learning

DRL . . . . . Deep Reinforcement Learning

PID . . . . . Proportional-Integral-Derivative

MPC . . . . . Model Predictive Control

DDPG . . . . . Deep Deterministic Policy Gradient

PPO . . . . . Proximal Policy Optimization

SLAM . . . . . Simultaneous Localization and Mapping

TRPO . . . . . Trust Region Policy Optimization

LSTM . . . . . Long Short-Term Memory

CW . . . . . Clockwise

CCW . . . . . Counter Clockwise

VTOL . . . . . Vertical Take-off and Landing

AI . . . . . Artificial Intelligence

GPU . . . . . Graphical Processing Unit

RAM . . . . . Random Access Memory

NN . . . . . Neural Network

ANN . . . . . Artificial Neural Network

IMU . . . . . Inertial Measurement Unit

RF . . . . . Rangefinder

LiDAR . . . . . Laser Interferometry Detection and Ranging

List of Figures

List of Tables

Contents

Chapter 1

Introduction

This chapter briefly discusses UAV, its components and real world applications. Then the research motivation, objective, methodology has been discussed. Finally, the organisation of this thesis is detailed.

1.1 Unmanned Aerial Vehicle

Any aerial vehicle which is not manned (no person onboard) during operation is referred to as Unmanned Aerial Vehicle [1]. UAV are also referred to as Remotely Operated Aircraft due to the fact that the mere absence of a person onboard does not mean it is an autonomous system, and might still be operated from a remote setting. There is a clear distinction between UAV and an autonomous UAV. An autonomous system does not require any input from a person during operation. It is capable of making decisions on its own.

Absence of any crew members only adds to the operational capability of such a system. It is capable of carrying more payload than a similar manned aerial vehicle. A UAV is capable of making extreme manoeuvres which generate very high g-forces, which is impossible for a human to sustain. A trained pilot is capable of withstanding high g-forces (equivalent to 9 g) [2].

Category Name	**Mass (kg)**	**Range (km)**	**Flight Altitude (m)**	**Endurance (hours)**
Micro	< 5	< 10	250	1
Mini	$< 25/20/150$	< 10	150/250/300	< 2
Close range	25-150	10-30	3000	2-4
Medium range	50-250	30-70	3000	3-6
High Alt. long endurance	> 250	> 70	> 3000	> 6

Table 1.1: UAV categories

UAVs have long been used in military operations. Their common use cases in military includes

reconnaissance, ordnance mounted UAV for use in battlefield and light handheld versions to help with planning by troops [3]. Other use cases include remote sensing [4] and search and rescue deployment and many more.

Table 1.1 shows a small extract of various categories of UAVs [1]. This study deals with Micro UAV with mass less 5kg. Various sensor types [4, 5] used on UAV are:

- RGB Camera
- Multi-spectral Camera
- Hyper spectral Camera
- Thermal Infrared Camera
- LiDAR
- Ultrasonic
- Infrared

1.2 Components of Micro Aerial Vehicle

Two complicated components of developing a MAV is the position and attitude control [6]. An attitude controller is responsible for the orientation/state of the MAV in the 3D space. A quadrotor is capable of moving around by varying its roll, pitch and yaw. This variation of is enacted by applying different rotor commands to each rotor. An onboard Inertial Measurement Unit helps provide the necessary data for the controller to estimate its state and provide necessary correcting changes.

A position controller is responsible for reaching the goal location in the 3D space. It outputs the necessary commands for the quadrotor to reach its goal state. It is possible that the goal state is dynamic and therefore the controller has to adapt to the uncertainty. Current position of the MAV is estimated with the help of Simultaneous Localization and Mapping (SLAM) based algorithms. These two types of controllers only output the necessary changes to rotors. However, this change is actually implemented by a platform controller.

A platform controller is responsible for executing the commands it receives from attitude controller. It varies the power output to each rotor to achieve the desired result. This is a continuous operation where the desired change is accepted from the attitude controller and is enacted by the platform controller. Each of these controllers has a frequency associated with them. A higher frequency means the controller is making from decisions per second over a lower frequency controller. This is a low-level controller, that is, it interacts with the hardware directly. Whereas, the other two controllers do not.

A variety of sensors, as mentioned previously, help in achieving the task at hand and the data from these sensors is included in input of the above mentioned controllers. The Fig. 1.1 shows the association of these three controllers [6].

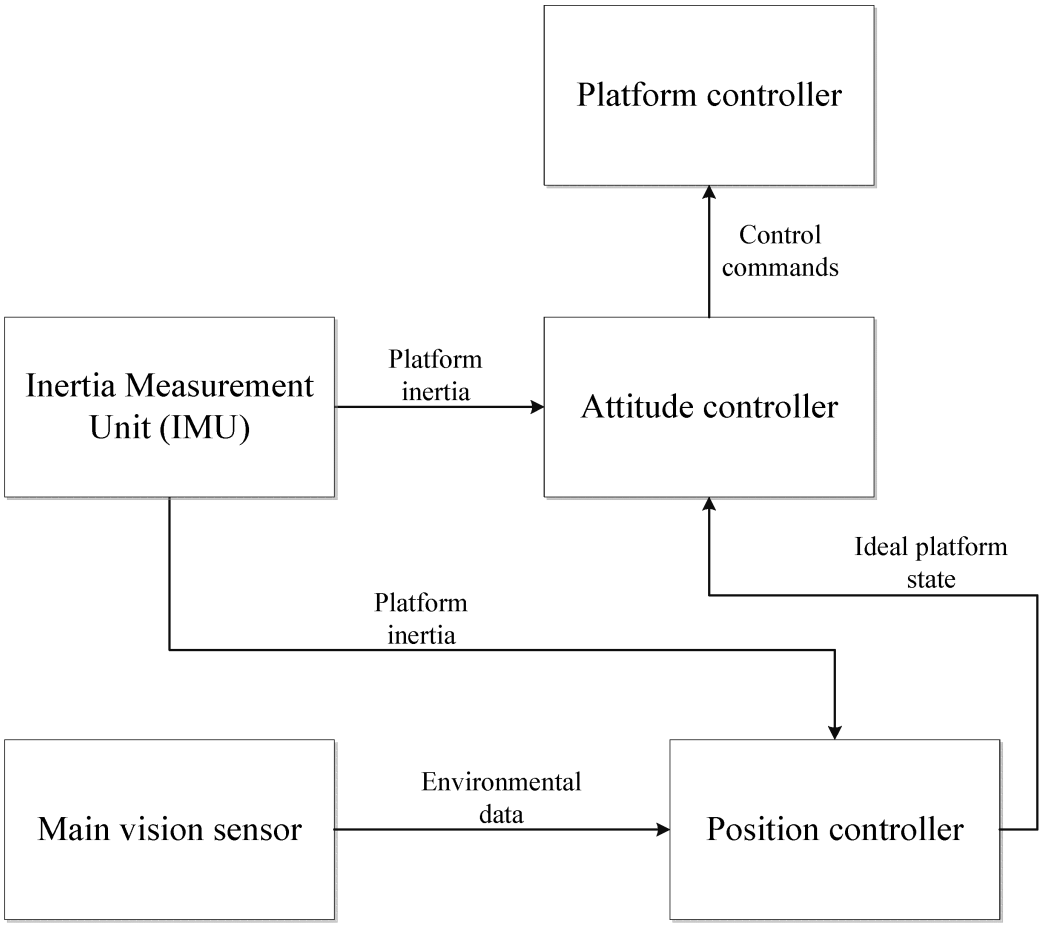


Figure 1.1: MAV block diagram

1.3 Applications of UAV in India

A hex-copter, a type of UAV with six propellers, was deployed by North Eastern Space Applications Center (NE-SAC) [7] for mapping and other tasks. Since it was deployed with various use cases in mind, it can carry a maximum payload/weight of 2.5 Kg. This payload is either a single camera or multiple sensors for the task at hand. A single UAV serves multiples purpose in this case.
The various applications of NE-SAC UAVs are as follows [8]:

- **Mapping of Landslide Affected area in Nongpoh, Meghalaya** An UAV was used to map the area around National Highway 40, which is prone to landslides.
- **Crop Damage Assessment** Crop Damage Assessment due to infestation was surveyed using UAV In Morigaon, Assam.
- **Large Scale Mapping** Nongpoh town, situated along the National Highway 40 was mapped from an altitude of 120m.
- **3D Modelling of terrain** A 3D model of MUDA shopping complex in Nongpoh was created with the help of UAV imaging.

1.4 Research Motivation

There are two reasons at play for the reason to pursue this specific topic. First, there has been a surge in recent years over the use of drones for meeting various requirements, for example drone racing [9], food delivery using drones [10] and much more. The personal use of drones has increased many fold and there has been an increase in the researchers who are interested in building ingenious solutions using such fascinating machinery.

Drones are available in various sizes, big like those used by militaries around the world for reconnaissance and small, which are barely bigger than the palm. People around the world have developed various solutions but they have barely scratched the surface of the enormous potential. Second, reinforcement learning has made a name for itself after its successful deployment by DeepMind researchers in tasks like playing the Atari games and the more famous AlphaGo, which is a computer program trained to play *Go* and which was able to beat a professional player, a feat achieved by no other computer program for this game. Reinforcement learning allows training an agent with the help of positive or negative reinforcement. Giving a positive reward for good actions and penalising for bad ones helps the agent to conform to the good behaviour. This is a simple idea but it is rather difficult to implement RL to a wide variety of tasks due to a lack of research in progressing this field.

That being said, this thesis project is not intended to be revolutionary. However, it does intend to progress the field of reinforcement learning by applying its concepts to quadrotors which have been tacked in the past, albeit with a different objective. The reason of developing a quadrotor which is capable of flying indoors, is its application in military and non-military needs. It will be an ideal candidate for Search and Rescue missions, which are often set indoors, and where the entry/exit of rescue personnel might be dangerous.

1.5 Research Objectives

The objective of this thesis are as follows:

- To propose a quadrotor system with low cost sensors for autonomous flight.
- To propose an efficient sensor layout.
- To design a control system for the quadrotor capable of flying indoors.
- To use Reinforcement Learning for training an agent capable of autonomous flight.

1.6 Research Methodology

The methodology followed during this research has been compiled in fig. 1.2. It was imperative that all steps were followed. Without the proper testing of the quadrotor in training environment, it wouldn't

have been possible to address the limitations of the simulation, which might have resulted in learning an incorrect control policy.

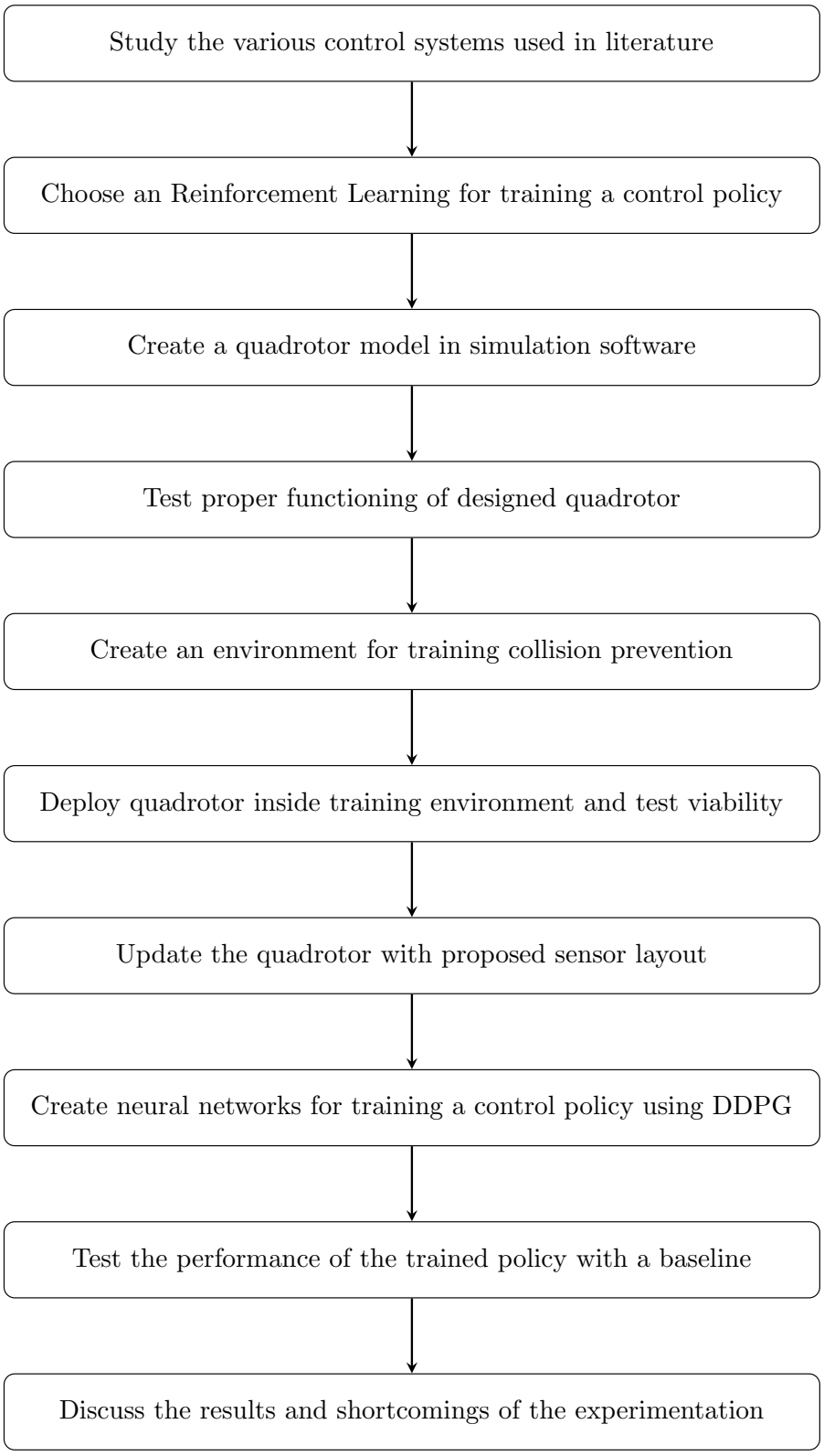


Figure 1.2: Research methodology

1.7 Dissertation Organization

This thesis has been compiled in various chapters, where each chapter touches upon any and all relevant material, and is as follows:

Chapter 2 begins by studying the literature and analysing the various solutions proposed by other researchers. Once the necessary information is extracted from the literature, then the challenges faced by the researchers is organised. Finally, a summary has been provided.

Chapter 3 compiles the theoretical and mathematical background needed. Theory about quadrotors and reinforcement learning is discussed. Furthermore, the mathematical equations for quadrotor dynamics and DDPG is provided. Moreover, the various orientation representation in 3D space are provided. This orientation is referred to as attitude in quadrotor setting.

Chapter 4 proposes the model used in this thesis and compares it with another similar model. The drawback of the proposed system is also identified.

Chapter 5 encompasses the design and implementation details. Starting by defining the hardware and software requirement specification before moving on to the implementation of the quadrotor and training environment in physics simulator. The chapter then defines the neural network architecture and MDP formulation, necessary for RL learning. All the above information is summarised at the end.

Once the implementation is complete, **Chapter 6** lays out the training parameters and then compares the performance of the proposed approach against random controller with the help of various graphs. Furthermore, the results are discussed and some performance gaps identified.

Finally, **Chapter 7** concludes the thesis and mentions some future approaches that might evolve from this work.

Chapter 2

Literature Survey and Theoretical Background

In this chapter, the previous work in the field of UAV will be surveyed and the various challenges be extracted. A complete background on theoretical and mathematical level has been provided. The quadrotor model, its functioning to vary roll, pitch and yaw is also explained using diagrams. The various attitude representation methods are mentioned with a brief overview on two select methods commonly used in research. The chosen method is also reasoned.

2.1 Literature Review

A control system acts as the central brain of the entity under consideration. It makes decisions based on various inputs it receives from sensors and interprets this information to provide control signals to other connected components like actuators, for example. It is possible to have multiple control systems on a single entity. This usually happens when the entity under consideration is complex enough for a single control system to handle.

In [5], the authors designed a collision avoidance system using ultrasonic and infrared sensors. Their choice was based on the fact that laser sensors are expensive and a low cost approach was warranted. However, the use of aforementioned sensors also has an inherent flaw that they suffer from noisy reading and cannot operate under certain circumstances. Their control system included the use of a modified Proportional-Integral-Derivative (PID) controller. Their system demonstrated a successful implementation of autonomous flight with collision avoidance, including the presence of an individual.

The authors in [11], used Model Predictive Control (MPC) for controlling the quadrotor, while also applying Reinforcement Learning for path-planning. They also applied "convexification" to calculate the obstacle-free region of operation, in order to avoid collisions. RL was used for determining the target location for the quadrotor and MPC provided the control inputs. The dynamic target location was calculated during flight and served as checkpoints in order to reach the final destination. Their test results showed success in traversing the environment.

The researchers in [12] presented an imitation learning based approach for learning a control policy for obstacle avoidance. They represented the policy using a neural network, which was trained on samples generated using time-free MPC known as Model Predictive Contouring Control (MPCC). Their experiment results demonstrated the viability of using this approach to learn a policy, which successfully generalises to the training data and is able to show satisfactory performance in unseen environments. Vision based control system have also been designed. The inclusion of an onboard camera helps the agent actually view its surroundings and react. In [13], the authors designed a controller for avoiding obstacle in its path. The system included an onboard camera, which was used to take images and detect the obstacles. A possible path was generated for the quadrotor to use. Their system only focused on the path-planning as the control system was the one available with the test drone. Moreover, in [14], researchers proposed a curriculum learning framework (branch of reinforcement learning) for visual navigation with successful result in real-world.

Deep Reinforcement Learning has been gaining significance in designing control system for quadrotors, as it has demonstrated time and again of its capability to find a policy suitable to the task. However, RL also has the potential to be used for training efficient and reliably performing control systems. In [15], the authors designed a control system capable of stabilizing the quadrotor under extreme conditions; stabilization when initialized upside-down, for example. They propose a novel deterministic policy optimization using natural gradient descent. Their experimental results consisted of two tasks: way-point tracking and recovery tests. Their approach demonstrated significant performance capability. Their implementation did produce a system that was truly capable of stabilizing itself, even under harsh conditions and initializations.

In [16], an obstacle avoidance framework was proposed. They predefined a set of actions which the agent was responsible to choose from according to its obstacle avoidance requirement. Their use of a predefined action set rather than training a control policy over the whole action space, means the performance is only limited to environments similar to their own.

In [17], the authors designed an adaptive trajectory tracking controller. Their work includes learning a policy which can effectively minimise the external disturbances and other inaccuracies that degrade the performance of the quadrotor. The authors of [18] designed a novel policy gradient based method for hovering and trajectory tracking. Their method is along the lines of Actor-Critic architecture. For the path following problem, in [19], the authors used three approaches to handle this issue at hand. In first approach, the controller has only the instantaneous state of the quadrotor as input. This means the controller has no information of further path shape. This problem is fixed in the second approach where the yaw error is added to the state vector. This addition helps improve the average distance error from the reference path, but only for fixed velocity. Third approach includes calculating a velocity command that will help in following the shape of the curve with optimal velocity. Deep Deterministic Policy Gradient (DDPG) is used for training the neural nets. Results show impressive performance in simulation and more importantly, satisfactory results in real world with wind disturbance.

In [18], the authors made a few changes to the underlying policy gradient function for training and also used hinge loss over PPOs prescribed clipping function for their training algorithm. Their

proposed method achieved better performance over [15] and at considerably lower time steps. They also simulated trajectory tracking with successful results. Their work also consisted of real world testing. The performance was similar to simulation results, which is commendable as the controller was trained in ideal conditions. The only significant deviation from simulation results was the apparent offset of 2 $\pm$0.25m in the z-axis.

Since most control systems are trained in simulation, they do not learn handling the quadrotor in some of the very basic conditions, like wind. In [20], the authors specifically designed a controller that is able to handle disturbances due to external factors. They used an Actor-Critic Architecture, which is trained using Proximal Policy Optimization (PPO) and off-policy training method. The performance of their controller showed similar behaviour in windy and calm environments. Whereas, a model trained without any disturbance compensator showed a slight offset from the desired position. In [21], ANN was used to learn an adaptive control policy with PID.

In [22], researchers presented an end-to-end approach to fly a quadrotor through a narrow rectangular gap. Their approach included the use of curriculum learning during training the control system in two phases. This included training the controller by gradually shrinking the gap dimensions. Soft Actor Critic (SAC) is used for training. Their simulation results show satisfactory performance for large dimensions with understandable poor performance on smaller dimensions. They also proposed a Sim2Real framework which does not require the use of real-world data. This is especially important because deploying a policy, trained in simulation, on a real quadrotor presents its own challenges; for example, inaccurate modelling of the simulation framework which may lead to dangerous manoeuvrers in real world. Their proposed framework showed similar performance as simulation results.

Attitude control performance for three RL algorithms (DDPG, TRPO and PPO) are presented in [23] with PID as baseline. PPO demonstrated better performance among the three algorithms. Simulation results showed that PPO performed better than PID in regard to total error, rise time and peak velocity achieved. For random sampled simulation, PPO and PID performed similarly. TRPO and DDPG had experienced oscillations in roll and yaw, which are not desired from a controller as it may lead to instability and unrecoverable state.

While most papers relied on model-free RL algorithms, authors in [24] designed a low-level model-based control system. In model-based system, the aim is the learn the dynamics of the environment. However, this requires excellent training data over ample training time and might lead to a situation when the performance is unsatisfactory, if the dynamics learned is offset from the truth. This is generally noticed in simulation vs real-life test, since simulation is not able to create such complex environment. The dynamics of the system were approximated using Neural Networks and the MPC controller used the dynamics model, which predicted the state evolution, to simulate simultaneous evaluation of actions. Trained on 3 minutes of training data, the quadrotor was able to hover for 6s.

Linear and Non-linear controllers are capable of adequate performance in stabilizing a quadrotor. However, it is possible to augment the performance of the traditional controllers with the help of a supplementary controller, as demonstrated in [25]. Their method involved training a DRL based controller that works in tandem with a 'basic' controller to optimize the performance index. Their eventual con-

troller consisted of cascade PD controller and RL based controller. Simulation results for experiments with wind disturbance showcase the better performance as compared to dynamic surface control method.

2.2 Autonomous Quadrotor Challenges

- Most researchers initially train their systems in simulations. This means the modelling constraints are heavily relaxed, given the high computation requirement for simulating the exact dynamics of real world [12, 15].
- Embedded hardware does not offer much compute power and thus the developed systems need to be efficient.
- The compute power required for training a control policy is unprecedented.
- Simulation to real-world transfer is not easy and requires extra training [15, 17, 18, 19].
- LSTMs were unable to generalize to the training dataset [14]. LSTMs act as memories and are used in ML based applications extensively.
- Attitude representation using quaternions take longer to learn as compared to rotation matrix in the beginning [18].
- RL based methods are highly inefficient [12, 22].
- Systems that do not implement sensor for collision avoidance tend to show failures [24].
- Quadrotor dynamics changes during flight due to various factors [24], eg battery drainage leads to low voltage.

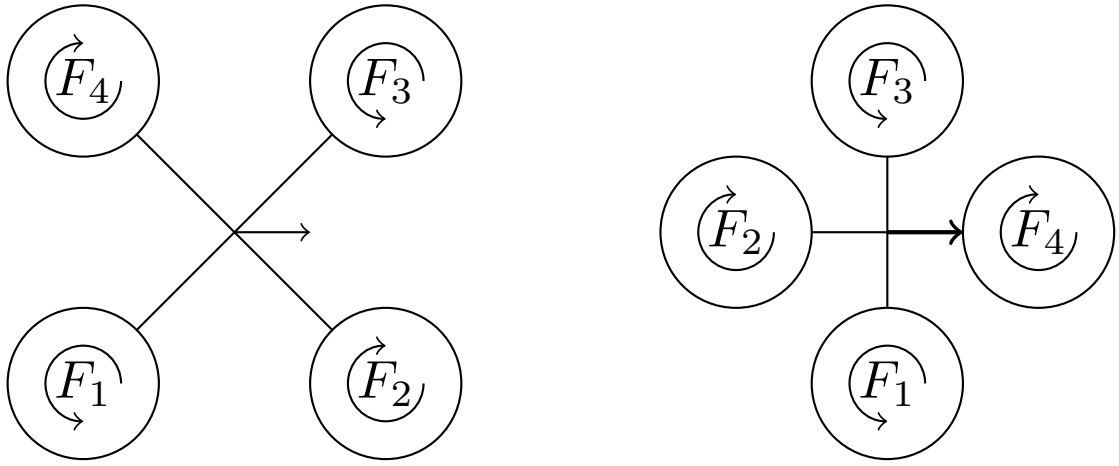
2.3 Theoretical Background

This section details the relevant material required for understanding quadrotor (section 2.3.1) and reinforcement learning (section 2.3.2. A quadrotor model was created in simulation and reinforcement learning was used to learn a control policy.

2.3.1 Quadrotor

A quadrotor, or quadcopter, is non-linear, underactuated, vertical take-off and landing (VTOL) system. As the name implies, it consists of four rotors/propellers. They are generally found in two configurations: $+$ (plus) and $\times$ (cross). Henceforth, the term rotors will be used to imply the same meaning as propellers and to remove any possibility of ambiguity.

The fig. 2.1 illustrates the two frames commonly used in quadrotors. Both these configurations have different dynamics and operation mechanism. The preferred approach for this thesis is shown in



(a) Cross($\times$) configuration (b) Plus($+$) configuration

Figure 2.1: Two different configuration of rotor setup

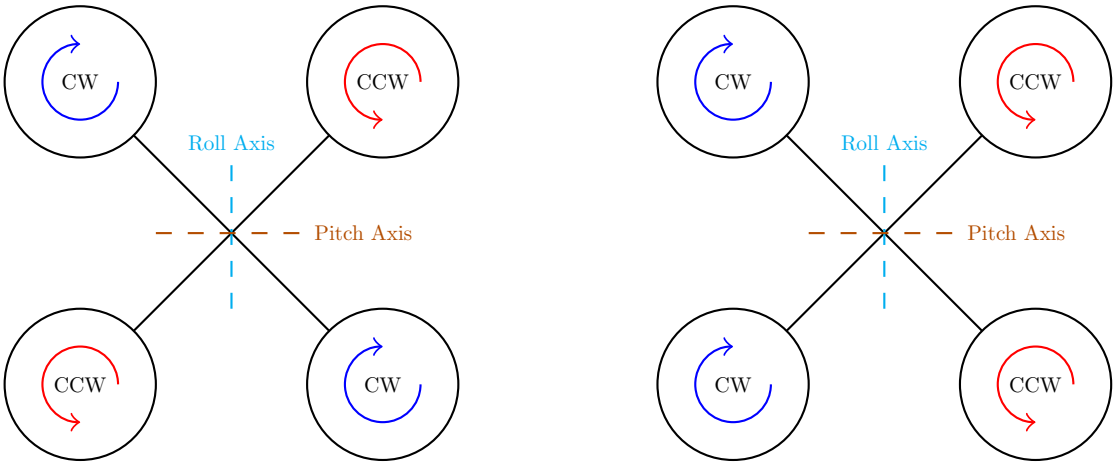
fig. 2.1(a). The cross type configuration may use all four rotors to induce roll/pitch. However, the plus configuration can only use two rotor for the same action [26]. This indicates a more agile nature of the cross configuration. It is faster to increase a rotor spin than to decrease, due to inertia. This is the reason that, when the plus configuration induces roll, there is a slight yaw induction as well, since the speed up of rotor and the slow down of the rotor does not happen at the same rate.

A plus configuration induces roll/pitch by simultaneously increasing and decreasing two rotors that effect the desired change. For example, to induce roll in quadrotor shown in fig. 2.1(b), F_3 will need to be greater than F_1. This will require increasing the spin in rotor M_3 (M_3 produces F_3) and decreasing the spin in rotor M_1. Since, the rate of change is not equivalent for both the actions, a slight yaw is induced, which in turn is corrected by rotor pair M_2 and M_4. Another difference between the two configurations is the production of side force in plus configuration and absence in cross configuration [26]. This side force is a lateral force that deviates the quadrotor from its designated path.

The more general approach to building a quadrotor is shown in fig. 2.2(a). In this configuration, the diagonal rotors spin in the same direction. Fig. 2.2(b) implements a system in which the diagonal rotors spin in different direction. At first glance, its not obvious what is an inherent limitation of such a setup. When the two rotors spinning in the same direction start spinning faster, then there will be an increase in yaw, however, roll is induced too. Which means to induce roll there is an induction of yaw and vice-verse. This is why, the first approach is deemed acceptable and can be found in a frequent use over the latter. The section 2.4.1 takes a closer look at the dynamics of the quadrotor.

2.3.2 Reinforcement Learning

Reinforcement is its own class of algorithms under machine learning, where rather than providing the absolute truth (target value), as in supervised learning, or providing no truth at all, as in unsupervised learning, an agent is either rewarded or penalised for its actions. It is also referred to as semi-supervised



(a) Diagonal rotors same spin

(b) Adjacent rotors same spin

Figure 2.2: Two different configuration of rotor spin

learning because of the system of rewards. Fig. 2.3 shows the machine learning hierarchy. All algorithms in machine learning are categorised accordingly.

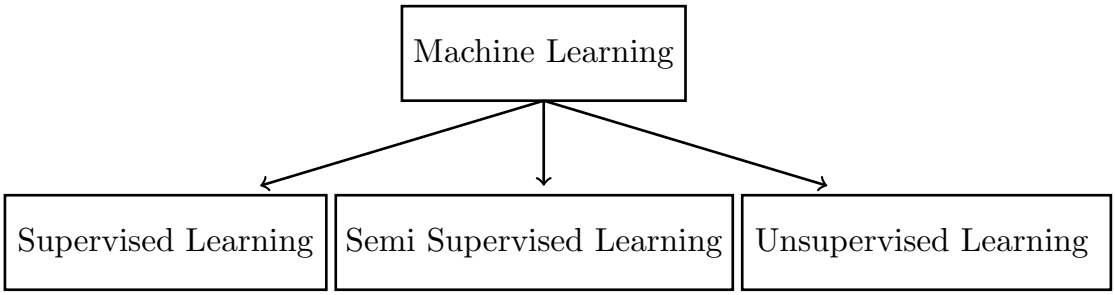


Figure 2.3: Machine Learning Hierarchy

Reinforcement Learning algorithms provide a method of learning in situations where it is not possible to provide the target value. The system of rewards and penalties help in guiding the learner towards the desired result. This becomes more evident in complex environments like training autonomous quadrotor, where it is not possible to provide an annotated dataset due to the large state and action spaces. Any algorithm that is able to search these large spaces, without much human labour, are preferred over others.

Markov Decision Process

The basic requirement of RL is the declaration of an environment containing the Markov property. Therefore, all environments in RL are defined as a Markov Decision Process. The definition of an MDP is as follows: $(\mathbb{S}, \mathbb{A}, P^a_{ss'}, r)$, where $\mathbb{S}$ is the state space, $\mathbb{A}$ is the action space, $P^a_{ss'}$ is the state transition probability, r is the reward[27, 13]. An action is chosen by the agent according to a stochastic policy, $\pi(a|s;\theta)$ or a deterministic policy, $\mu(a|s;\theta)$ [28] [1]. Here θ is used to represent a parametrised policy.

[1]The policy here is represented with π only to follow the general notation of representing a policy irrespective of its type.

State value function, $V^{\pi}(s)$, estimates the expected reward from current state, such that the policy is followed henceforth, is given as [27]:

$$V^{\pi}(s) = \mathbb{E}\left[\sum_{l\geq 0} \gamma^{l} r_{t+1} \,\middle|\, s_t = s, \pi\right] \tag{2.1}$$

and the action value function, which estimates the expected reward from current state, taking action 'a' and following the policy thereafter, is [27]:

$$Q^{\pi}(s,a) = \mathbb{E}\left[\sum_{l\geq 0} \gamma^{l} r_{t+1} \,\middle|\, s_t = s, a_t = a, \pi\right] \tag{2.2}$$

2.4 Mathematical Background

This section compiles the relevant mathematical background for implementation purposes. The most important topic of attitude representation is discussed and the approach for this thesis is reasoned.

2.4.1 Quadrotor Dynamics

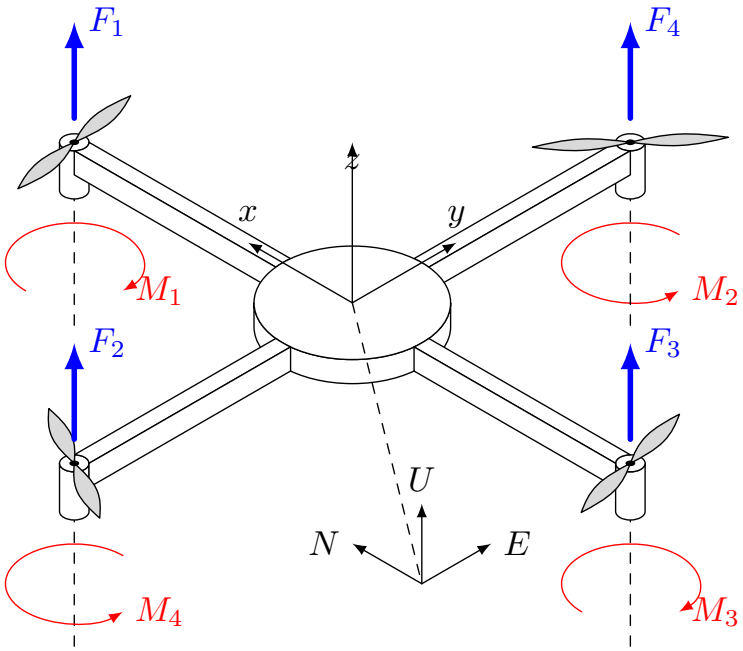


Figure 2.4: Quadrotor Model

A Quadrotor is able to produce translational motion by tilting in the direction of desired travel. The three rotation parameters that represent tilting are roll θ, pitch ϕ and yaw ψ. The desired change is attained by varying the thrust of a pair of rotors. The chosen pair allows the quadrotor to alter either of these

parameters. When the individual thrust of each rotor is changed proportionally, then the effective result leads to change in vertical thrust. The figure 2.4 shows a model of a quadrotor [29].

- Vertical thrust is the total force generated by the four rotors in the z-axis (body frame), as shown in fig. 2.4.
- *Roll* θ is the rotational motion about the x-axis (refer to fig. 2.5(a)). This is essential in translational motion along the y-axis.
- *Pitch* ϕ is the rotational motion about the y-axis (refer to fig. 2.5(b)). This is essential in translational motion along the x-axis.
- *Yaw* ψ is the rotational motion about the z-axis (refer to fig. 2.5(c)). This is achieved when the diagonally opposite pair of rotors are spinning at a rate higher/lower than the opposite pair. Yaw does not result in translational motion. It does, however, spin the quadrotor to point in a certain direction.

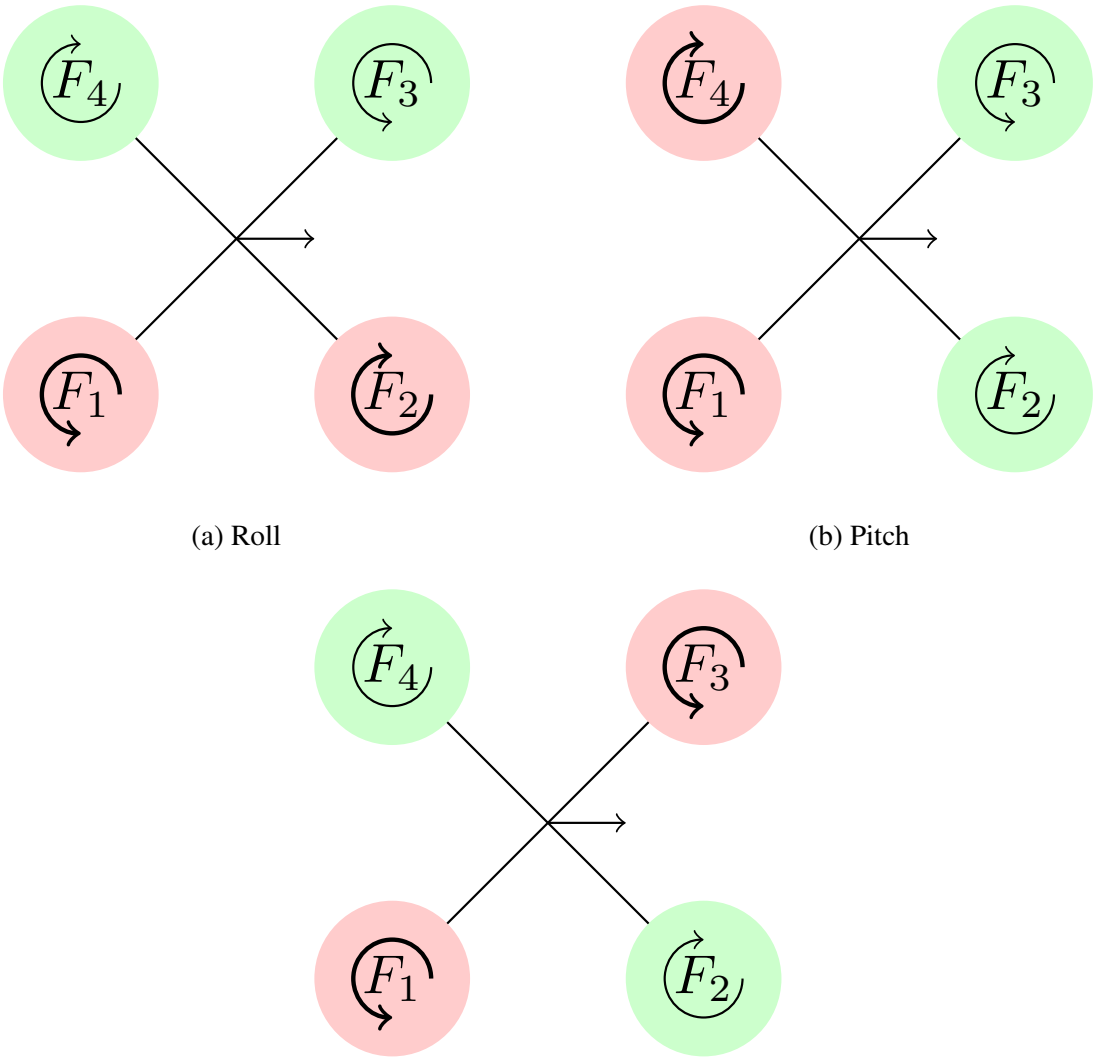
Equations of motion are given as [30]:

$$\begin{cases} \dot{\zeta} & = v \\ m\dot{v} & = RF_b \\ \dot{R} & = \hat{\omega} \\ J\dot{\omega} & = -\omega \times J\omega + \tau_a \end{cases} \tag{2.3}$$

where,

$$\begin{aligned} \zeta &= \text{Position vector} \\ R &= \text{Rotation Matrix} \\ F_b &= \text{Force in body frame} \\ \hat{\omega} &= \text{Skew Symmetric Matrix} \\ J &= \text{Rotor Inertia} \end{aligned}$$

The fig. 2.5 shows the rotor combinations that induce roll, pitch and yaw in the quadrotor. Red rotors represent higher thrust over green rotors. The small arrow at the cross-section shows the direction it is facing. It is important to mention the direction in which the quadrotor is facing, as this coincides with the roll axis.



(a) Roll

(b) Pitch

(c) Yaw

Figure 2.5: Varying roll, pitch and yaw in a quadrotor through different rotor combinations

2.4.2 Deep Deterministic Policy Gradient

DDPG in an *off*-policy actor-critic algorithm. It consists of four neural networks: *actor*, *critic*, *target actor* and *target critic*. The *actor* and *critic* networks are updated using gradient descent, where the critic network is updated by minimising the loss [28]:

$$L(\phi, \mathscr{D}) = \underset{(s,a,r,s',d)\sim\mathscr{D}}{\mathbb{E}} \left[\left(Q_\phi(s,a) - \underbrace{y(r,s',d)}_{target} \right)^2 \right] \tag{2.4}$$

where *target* is calculated with the following formula:

$$r + \gamma Q_{\phi_{\text{targ}}}(s', \mu_{\theta_{\text{targ}}}(s'))$$

and the actor is updated using:

$$\max_{\theta} \underset{s\sim\mathscr{D}}{\mathbb{E}} [Q^{\pi}(s, \mu_{\theta}(s))] \tag{2.5}$$

Target network weights are updated as follows,

$$\phi_{\text{targ}} \leftarrow \tau\phi_{\text{targ}} + (1-\tau)\phi \tag{2.6}$$

$$\theta_{\text{targ}} \leftarrow \tau\theta_{\text{targ}} + (1-\tau)\theta \tag{2.7}$$

τ regulates the update step. A large update step means the target network would resemble the primary networks in lower number of iterations. A small update step ($\sim$0.995) has been used in this study. Advantages of DDPG are as follows:

- Being an *off*-policy algorithm, the training data does not require generation from the latest policy.
- Target networks help stabilizing the learning process [31], since the loss is computed using target networks which do not change at the same rate as primary networks.
- Use of replay buffers also helps in stabilizing training [31].

Disadvantages are mentioned below:

- Requires longer to converge [23].
- Requires four NN networks during training.

Above algorithm has been used for finding optimal control policy in this study.

2.4.3 Attitude Representation

Attitude, also called as rotation in 3D, refers to the orientation of the quadrotor in inertial frame. There are four ways of representing rotation in three dimensions: unit quaternion, rotation matrix, euler angles, axis-angle [32]. Euler angles are more easier to understand and implement, however they do suffer from singularity, also known as gimbal lock. Axis-angle representation is also easier to understand but suffers from the issue of similar representations when the rotation difference is exactly 2π. The more common approaches are rotation matrix and unit quaternion. Thus, they have been discussed in the following sections. Both have their strengths and weakness and are thus used according to one's use case. Researchers have used both these approaches before deciding on a single approach [18].

Quaternion

A quaternion is formally written as $a_0 + a_1 i + a_2 j + a_3 k$, where a_i are real numbers and the following conditions are satisfied [33]:

$$i^2 = j^2 = k^2 = ijk = -1$$
$$ij = k, ji = -k$$
$$jk = i, kj = -i$$
$$ki = j, ik = -j$$

A unit quaternion, represented as q_0 here, is calculated with the formula given below, where $\|q\|$ is the norm of q:

$$q_0 = \frac{q}{\|q\|} \qquad where, \|q\| = \sqrt{{a_0}^2 + {a_1}^2 + {a_3}^2 + {a_4}^2}$$

Talking points of unit quaternions are as follows: First, the big drawback of quaternions is that q and -q represent the same orientation [15, 32]. Second, quaternions are less computationally expensive as compared to rotation matrix. Third, they do have a steep learning curve, whereas rotation matrix are extremely easy to understand. Each use case needs to weigh these pros and cons before settling on a preferred representation technique.

Rotation Matrix

A rotation matrix uses 9 values to represent 3 DOF. This leads to redundancy and is the reason it does not suffer from ambiguous representation issue as the other representations. Before defining the rotation matrix, it is rather easier to state the rotation about each axis in isolation. This helps in a better understanding of the rotation matrix formula.
Rotation about z-axis by an angle ψ, is given as:

$$R_{z(\psi)} = \begin{bmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Rotation about y-axis by an angle ϕ, is given as:

$$R_{y(\phi)} = \begin{bmatrix} \cos\phi & 0 & \sin\phi \\ 0 & 1 & 0 \\ -\sin\phi & 0 & \cos\phi \end{bmatrix}$$

Rotation about x-axis by an angle θ, is given as:

$$R_{x(\theta)} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix}$$

There are 6 possible orders of rotation about the three axes: $xyz, xzy, yxz, yzx, zxy, zyx$. It is certainly possible to rotate about any axis multiple times, which means total 3^9 orders of rotation. However, since we are only concerned with the orientation in 3D space, rotation about each axis only once is considered. For rotation in the order xyz, first multiply $R_{y(\phi)}R_{z(\psi)}$,

$$R_{y(\phi)}R_{z(\psi)} = \begin{bmatrix} \cos\phi & 0 & \sin\phi \\ 0 & 1 & 0 \\ -\sin\phi & 0 & \cos\phi \end{bmatrix} \begin{bmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Next, the result above is further multiplied with $R_{x(\theta)}$:

$$R_{x(\theta)}(R_{y(\phi)}R_{z(\psi)}) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} \cos\psi\cos\phi & -\cos\phi\sin\psi & \sin\phi \\ \sin\psi & \cos\psi & 0 \\ -\sin\phi\cos\psi & \sin\phi\sin\psi & \cos\phi \end{bmatrix}$$

Solving above matrix multiplication leads us to the final form, $R_{x(\theta)y(\phi)z(\psi)}$:

$$\begin{bmatrix} \cos\phi\cos\psi & -\cos\phi\sin\psi & \sin\phi \\ \cos\theta\sin\psi + \sin\theta\sin\phi\cos\psi & \cos\theta\cos\psi - \sin\theta\sin\phi\sin\psi & -\cos\psi\sin\theta \\ \sin\theta\sin\psi - \cos\theta\sin\phi\cos\psi & \sin\theta\cos\psi + \cos\theta\sin\phi\sin\psi & \cos\phi\cos\theta \end{bmatrix}$$

Rotation matrix representation has been used in this study, keeping in mind the drawback mentioned in [15].

2.5 Intrinsic and Extrinsic Rotation

There are two types of rotations: extrinsic and intrinsic. In intrinsic rotation, the principal axis rotate with the rotations performed on the object. On the contrary, the rotation happens about fixed principal axis in extrinsic rotation [34]. In MuJoCo, both rotations may be defined, however the default option is the intrinsic rotation which has been left unchanged.

Chapter 3

Proposed Model

This chapter compares two proposed layouts for sensors. The core quadrotor model was left unchanged, in terms of any significant changes. Therefore the core model is not discussed in this chapter. The following section first defines the meaning of sensor layout in this thesis and will then propose the layouts with proper diagrams. Only one layout was finalised for implementation and this will be reasoned as well. Furthermore, the drawbacks of this model have been discussed.

3.1 Sensor Layout

Sensor layout refers to the placement of sensors, either ultrasonic or infrared, on the body of the quadrotor to achieve maximum coverage. It is important to mention that the layout design was only in consideration of the previously mentioned sensor types. Other type of sensor like LiDAR is generally very heavy and require bigger area to implement.

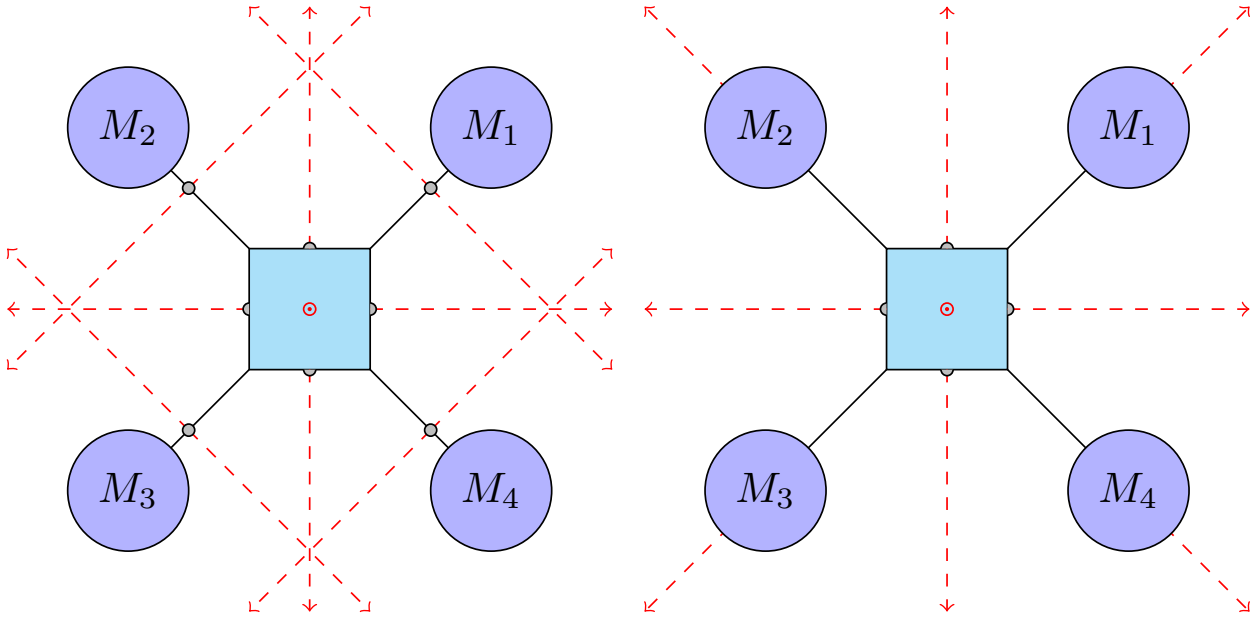
There might be better layouts possible but the two proposed layouts in the next section were the only ones considered and eventually one was deemed superior. Section 3.1.1 shows the two sensor layouts on the quadrotor model implemented in this thesis.

3.1.1 Proposed Layouts

Infrared and Ultrasonic sensors are two low-cost implementations as RF. However, they do tend to be noisy when the object is far from the sensor or if the object may absorb these waves, like clothes [5]. Consequently, the rangefinders in this model have a cut-off at 3m, i.e. all objects further than 3m would still show a sensor reading of 3m. While the quadrotor model is very basic, the sensor layout needed some deliberation. The following criteria had to be fulfilled by any proposed sensor layout.

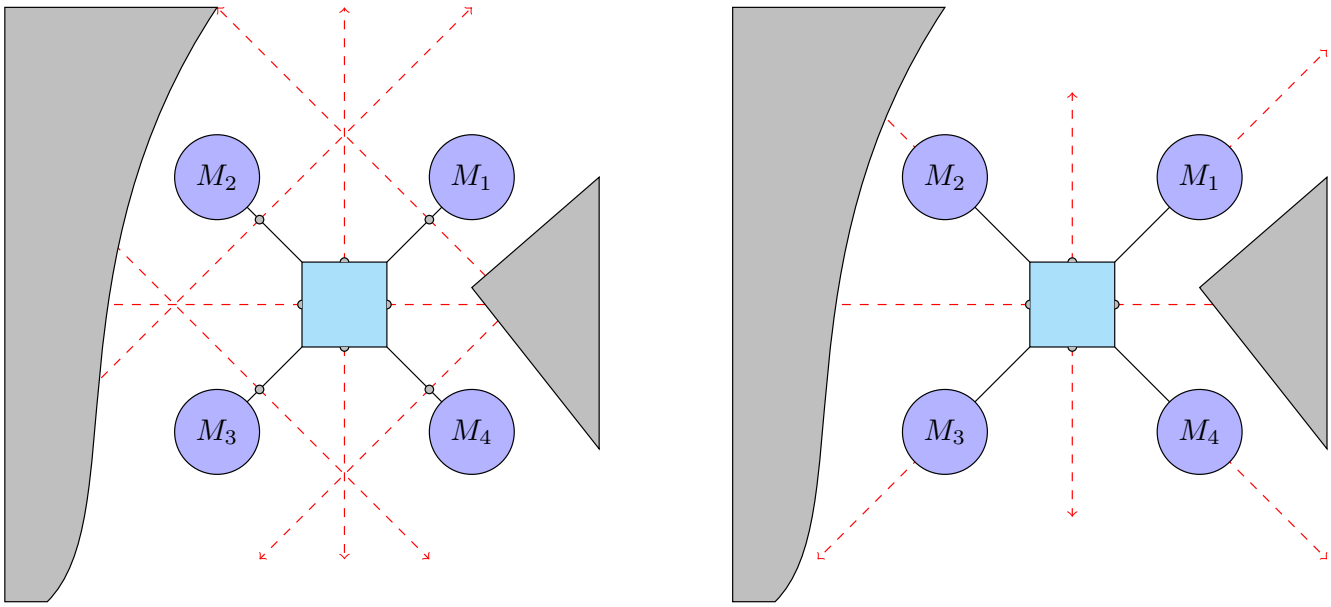
- The sensor count should be as low as possible to prevent the system from being expensive.
- The supposed layout should give maximum coverage.
- It should be possible to retrofit the sensor on existing models.

- The chosen sensors should be safe to use in the presence of humans.



(a) 12 sensor layout (b) 8 sensor layout

Figure 3.1: Potential sensor layouts for implementation



(a) 12 sensor layout near wall corners (b) 8 sensor layout near wall corners

Figure 3.2: Potential sensor layouts coverage near walls

Two proposed layout have been shown in fig. 3.1. Both layouts have common sensor placements on top, bottom and edge of square body. A single sensor on top and bottom was deemed acceptable because the quadrotor is required to perform with very less roll and pitch. If the quadrotor is stable for most of the flight duration, a single sensor shall be adequate for measuring distance from above and below.

The change in layout is noticed in on the quadarms(frame to which the rotors are attached, placed at 45° from the horizontal). In fig. 3.1(a), the RFs are attached at an angle of 90° to the quadarm. This mainly helps near wall corners. Fig. 3.1(b) shows the second layout in which the additional sensors are attached at the end of the quadarm. This layout requires fewer sensors. Fig. 3.2(a) and fig. 3.2(b) demonstrate both the layouts near walls. A better coverage is achieved in the former.

Even though second layout (Fig. 3.1(b)) uses fewer sensors, the first layout (Fig. 3.1(a)) was adopted for the final model due to better coverage. All the experiments have been run with the first layout, in simulation.

3.2 Drawbacks of the proposed model

- The first layout requires more sensors than the second layout.
- These sensor layouts presume ideal working environment.
- Neither Ultrasonic nor Infrared sensor has been assigned a specific spot in the above implementation. It is possible one sensor might perform better at some locations over others.
- The first layout has criss-crossing which might lead to interference and noisy readings.
- Infrared sensors return noisy reading under sunlight and Ultrasonic sensors perform poorly in low-light conditions [35].

Chapter 4

Design and Implementation

The software and hardware requirements are listed along with the design and implementation.

4.1 Hardware Requirement

The simulations were performed on M1 ARM chip with 7-core integrated Graphical Processing Unit (GPU) and 8GB RAM. These above mentioned specifications are the bare minimum. It takes about 5 hours to train for 10,000 episodes with the bare minimum requirement. A more powerful machine (external GPU) will be needed for training for millions of timesteps, at a substantially faster pace.

4.2 Software Requirement

Name	Description
Python(v3.8.9)	All the code is written in Python
MuJoCo(v2.1.1)[36]	Physics engine for simulations.
PyOpenGL(v3.1.5)	Renders simulation for viewing on screen
Tensorflow-macos(v2.8.9)	Required for Neural network training
Tensorflow-metal(v0.4.0)	Enables GPU training on M1 Mac
h5py(v3.6.0)	Saving neural networks in h5 format
dm_control(v0.0.4169)[37]	Provides Python bindings for mujoco
Pillow(v9.0.1)	Used to save simulation images
Numpy(v1.21.5	Needed for array manipulation
Matplotlib(v3.5.1)	Required for plotting graphs

Table 4.1: Software used with brief description

Table 4.1 lists the various software required during this thesis. The mentioned software frameworks were instrumental in developing the simulations and also for result analysis. Two software mentioned, *tensorflow-macos* and *tensorflow-metal*, are only valid for Mac devices containing apple silicon. For implementation on other machines, only $tensorflow >= 2.1$ will be needed for running these experiments.

4.3 Quadrotor Model and Training Environment

The simulation software, MuJoCo, was chosen due to its previous use in other RL agent training (examples shown in [37]). It has been used by other researchers as well and is a fast and reliable simulation software.

4.3.1 Environment setup in MuJoCo

The quadrotor and the training environment were created in MuJoCo and simulations were performed with *dm_control* as an interface between Python and MuJoCo. The fig 4.1 specifies the training environment dimensions. The system of units in mujoco is undefined [38], therefore, MKS unit system will be followed for the rest of the paper. Thus, the environment is a 9m×9m×6m cuboid.

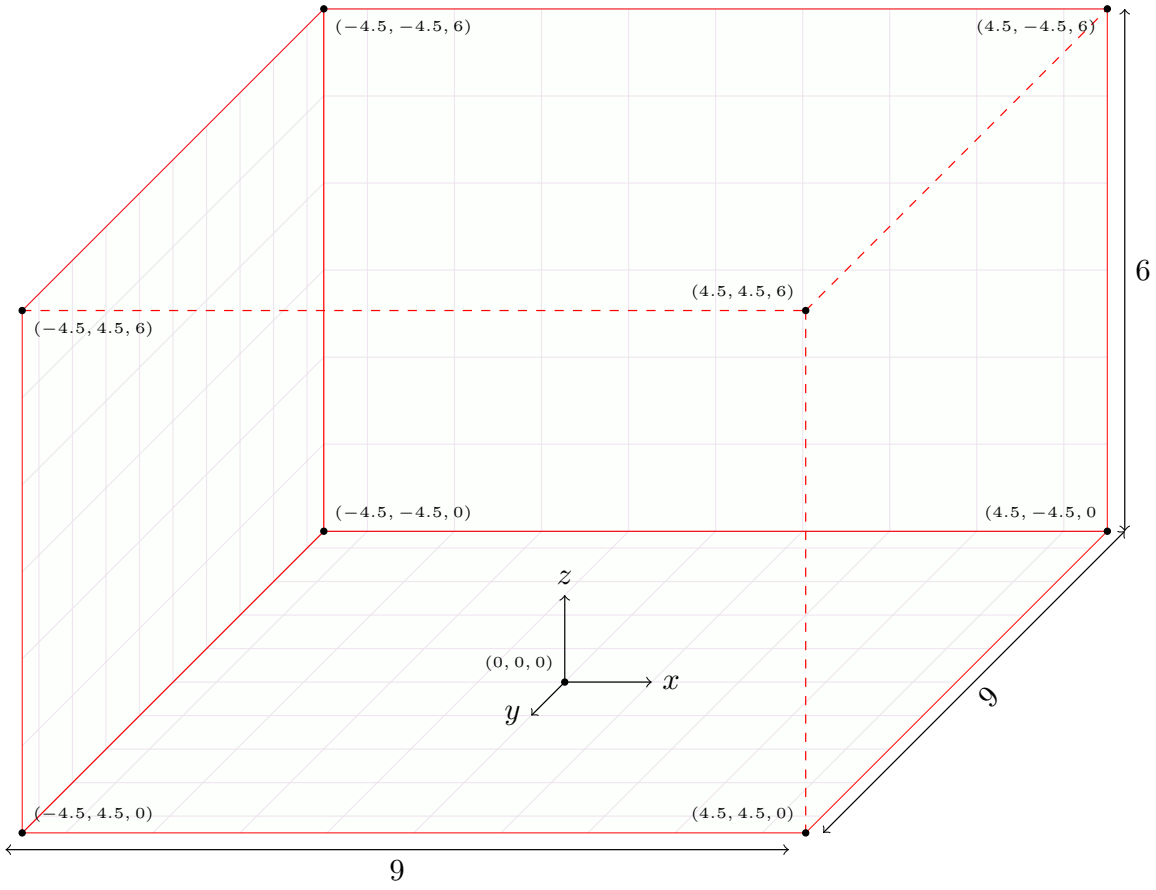


Figure 4.1: Mujoco Environment specification

4.3.2 Quadrotor setup in MuJoCo

Next, a quadrotor model was developed with rangefinder sensors attached at various locations as shown in section 3.1.1. Some minor implementation details are as follows:

- The central quadrotor body has been designed as a square frame with miniscule height (0.01m) and a mass of 0.1kg or 100g.
- Each quadrotor arm(arm is the diagonal frame on which rotors are attached) is situated at an angle of 45° and is assigned a mass of 0.01kg or 10g.
- Each rotor is designed as a cylinder with negligible height (0.002m) and a mass of 25×10^{-4} kg or 2.5g.
- Sensor module do not posses any mass.
- The control input of each rotor is a real value in the range [0-1]. With 1 being full thrust and 0 being none.

The fig 4.2 shows the final quadrotor model implemented using Mujoco. Here the blue cylinders are meant to represent rotor locations.

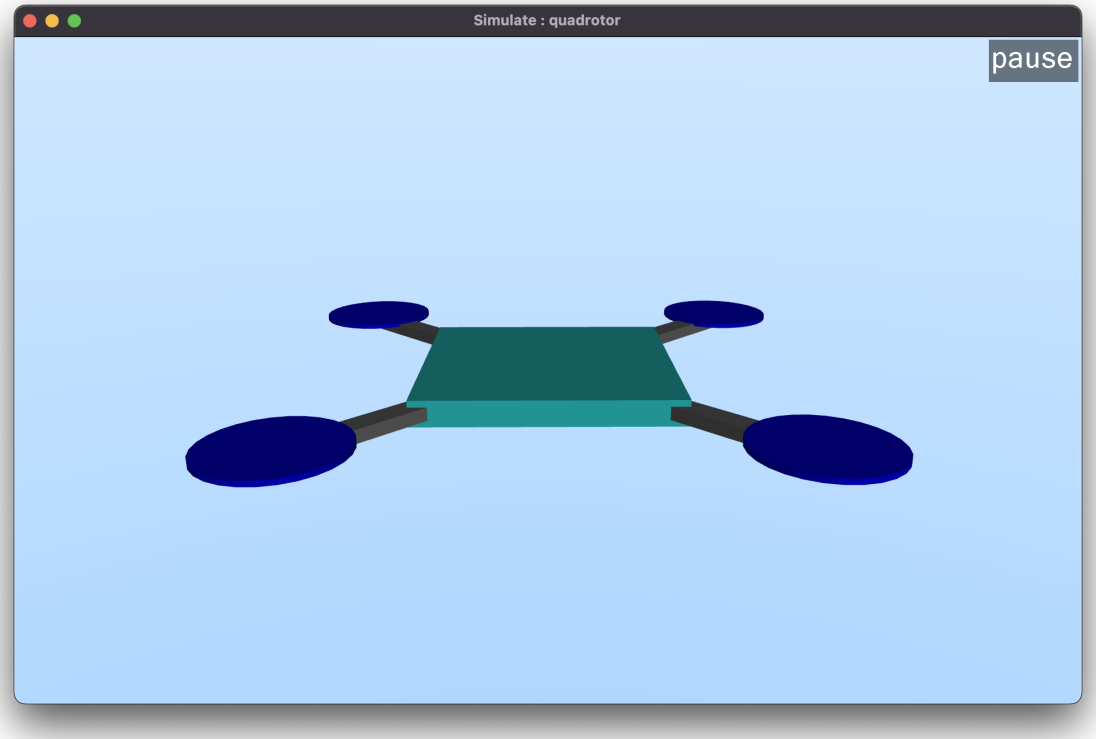


Figure 4.2: Quadrotor implementation in MuJoCo

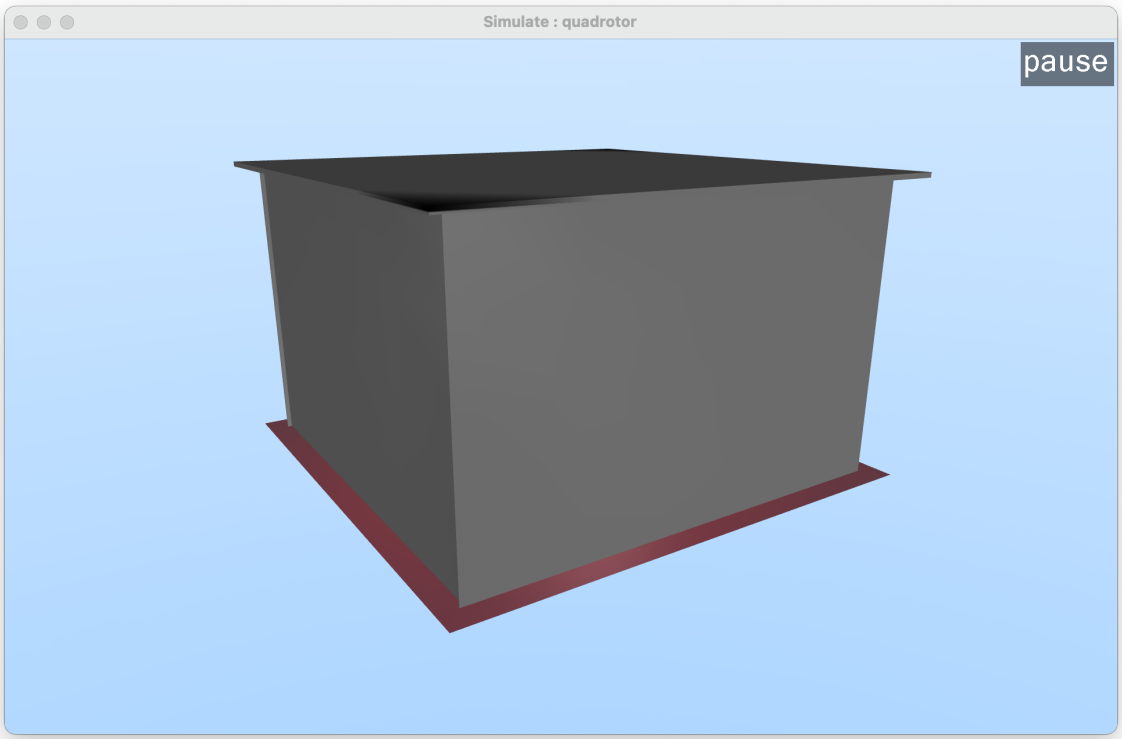


Figure 4.3: Training environment in MuJoCo

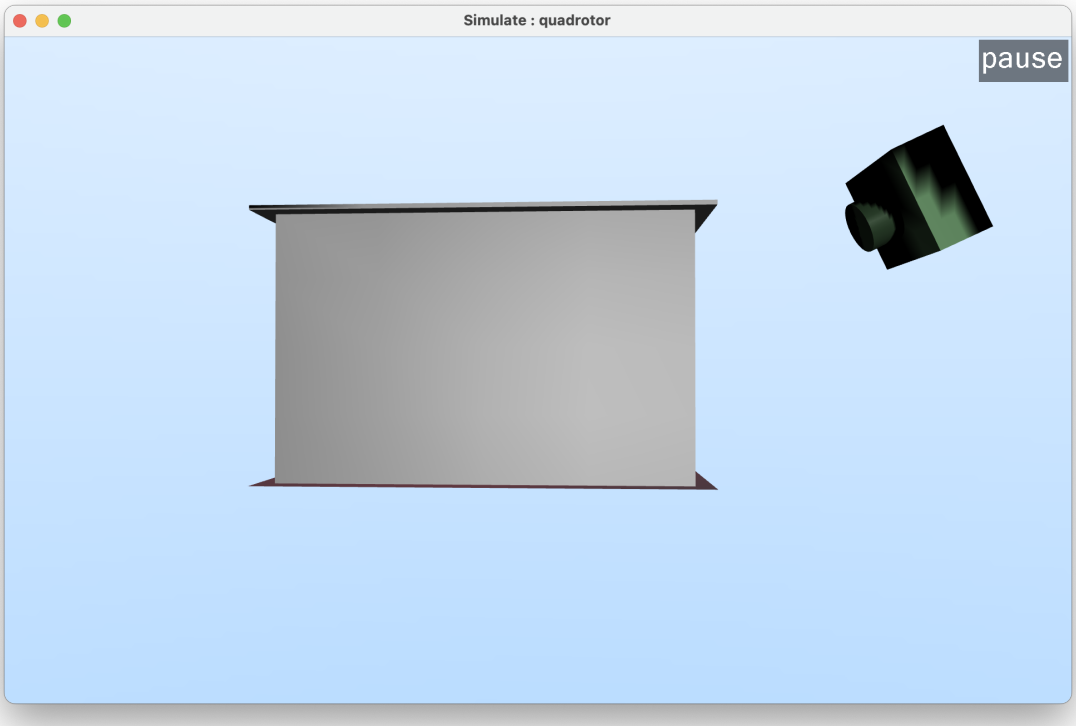


Figure 4.4: Camera setup used during rendering

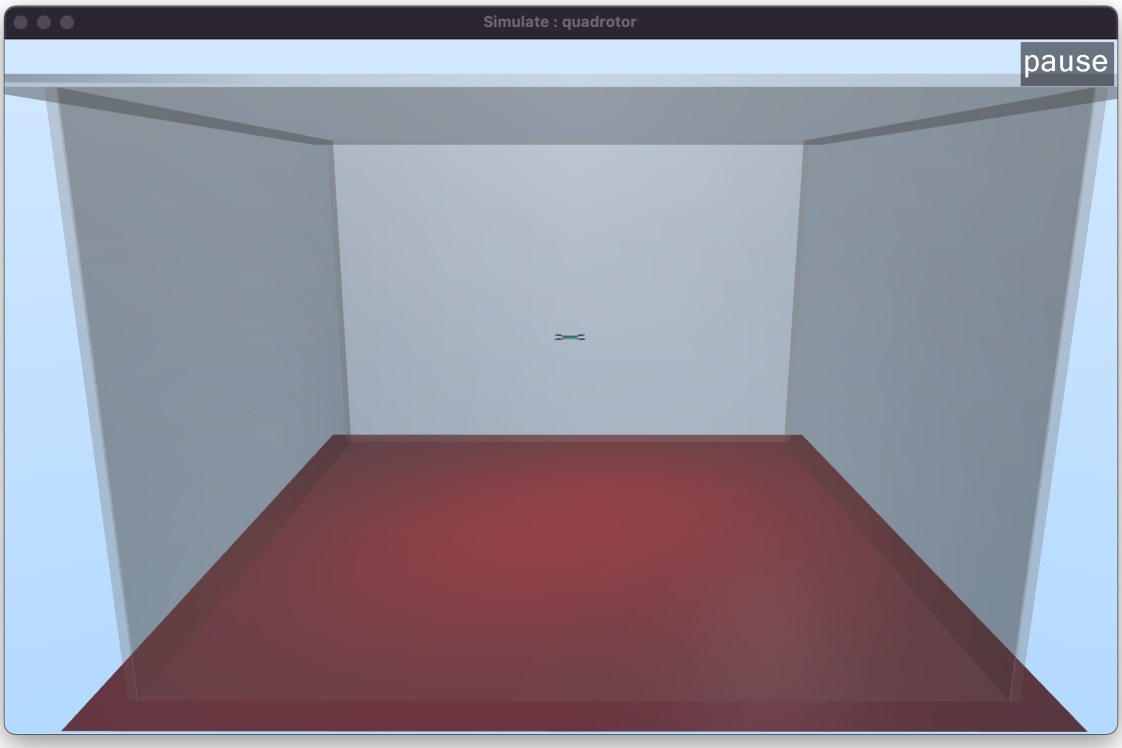


Figure 4.5: Environment implementation in MuJoCo

The fig. 4.3 shows the implementation of fig. 4.1 in MuJoCo. The fig. 4.4 shows the camera implementation alongside the training environment. MuJoCo consists of a *free* camera by default but it did not capture the correct posture of the environment. Therefore, a separate camera module was implemented outside the environment to capture any images. The walls are not as opaque as shown in this figure and is therefore able to capture proper images.

The fig 4.5 shows the training environment. The simulation environment consists of a quadrotor placed inside a room. Walls and the roof in all generated images are translucent so as to facilitate non-obfuscated quadrotor image from the camera placed near one of the walls. It also shows that the walls are extend slightly outward. During testing of the environment, it was noticed that creating exact size walls allowed the quadrotor to escape, even though it shouldn't be allowed. Another important observation is that if the timestep size was set to 0.1, the quadrotor passed through the floor unlike a solid object.

The fig. 4.6 shows a top-down view of the proposed sensor layout implementation in MuJoCo. Since a cut-off value is assigned for each sensor, its effect is clearly visible in the range (represented in yellow colour). The fig. 4.7 shows a sideways view. The top and bottom sensors are making contact with the roof and wall, respectively. This means with each episode initialization, the RL agent is aware of its proximity from above and below, but not on its sides. This should not make much difference, with respect to the performance. Since, the reading from top and bottom sensors would still report 3m at its reading at the very beginning.

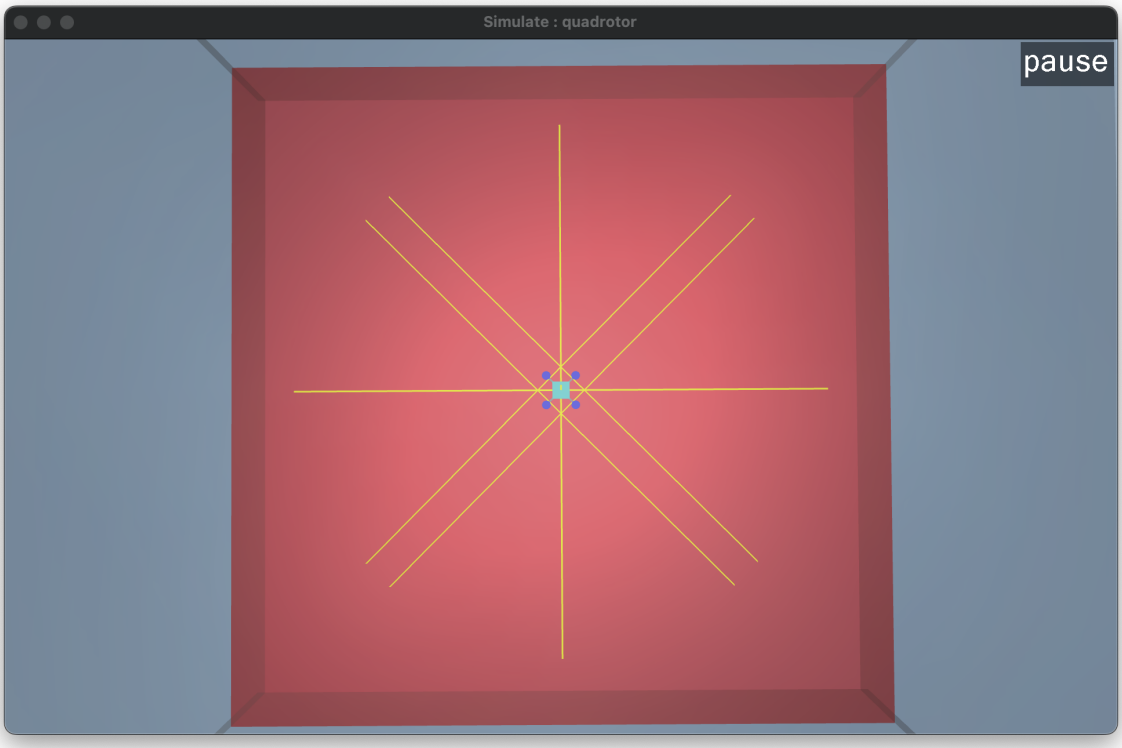


Figure 4.6: Top-down view of proposed sensor layout in MuJoCo

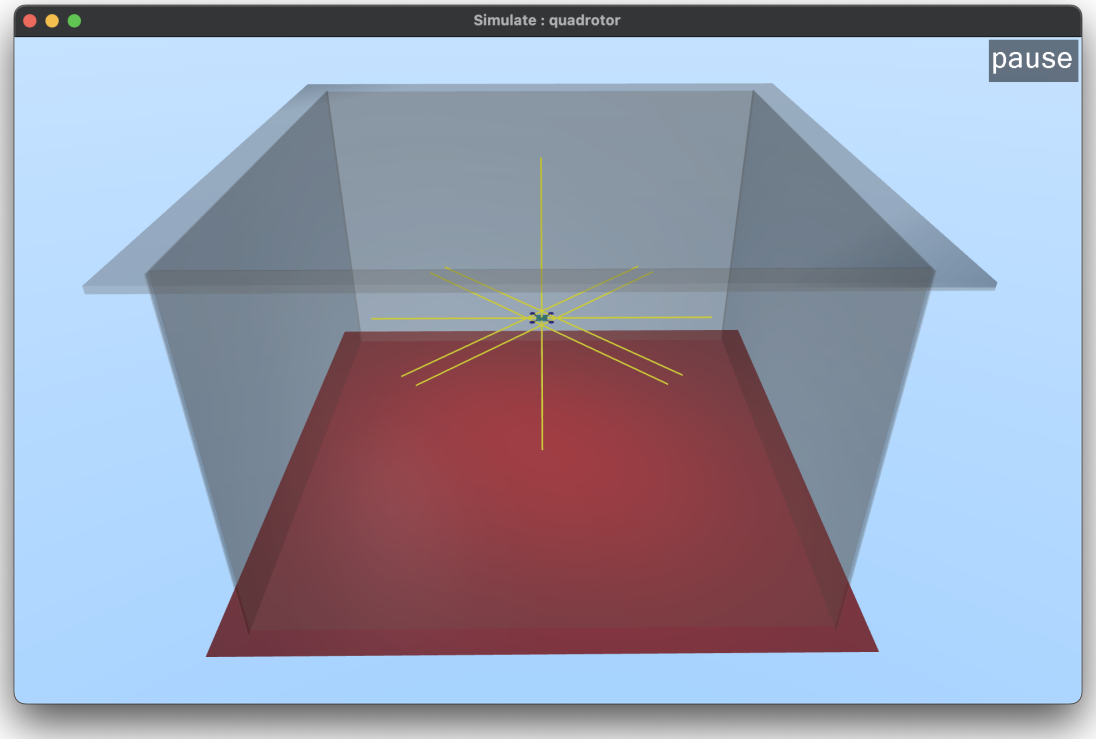


Figure 4.7: Sideways view of proposed sensor layout in MuJoCo

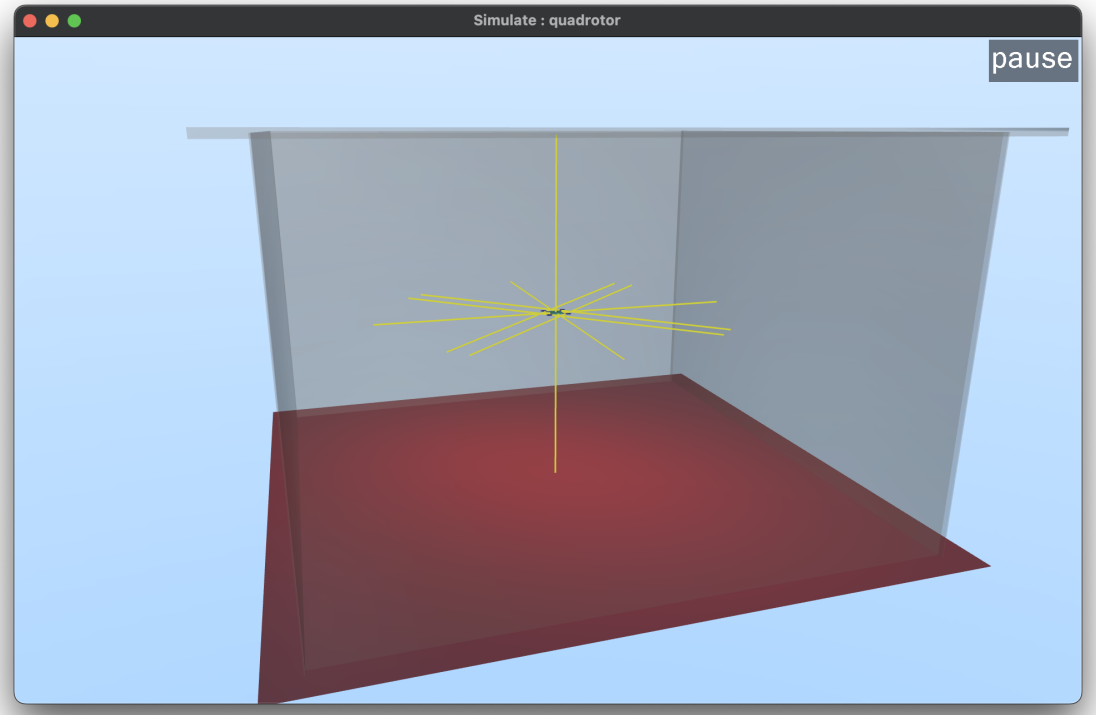


Figure 4.8: 3D view of proposed sensor layout in MuJoCo

Fig. 4.8 demonstrated the complete sensor layout in 3-dimensional space. It this figure, it is clearly visible which object is at a distance of greater than 3m. Since most episodes during learning are initialized in a similar position, none of the walls are ever at a distance of less than 3m. A few of these episodes are initialized with the quadrotor at close proximity to the walls, therefore the RL agent does not have to take evasive action from the nearby walls.

4.4 Agent setup in TensorFlow

Tensorflow and *Keras* were used for neural network creation and training. They provide a high level API for a quick ML workflow. These software have been extensively tested and are a go-to tech stack in the ML community, including researchers and industry applications.

4.4.1 Neural Net Architectures

The actor network architecture is shown in fig. 4.9 and the critic network as shown in fig. 4.10. All nodes are fully connected ANNs. Hidden layer nodes in both networks use tanh activation function and contain 64 nodes in each layer. Moreover, in the actor network, the output layer has a linear activation. Furthermore, output, O_i. is clipped to be within the range [0,1] (refer to subsection 4.3.2). The critic network contains 4 additional nodes in input layer, to facilitate the input of action commands. Output

layer of the critic contains only a single node, with linear activation, whose value represents how good the current state and action pair is. *Target actor* and *Target critic* follow the same architectures.

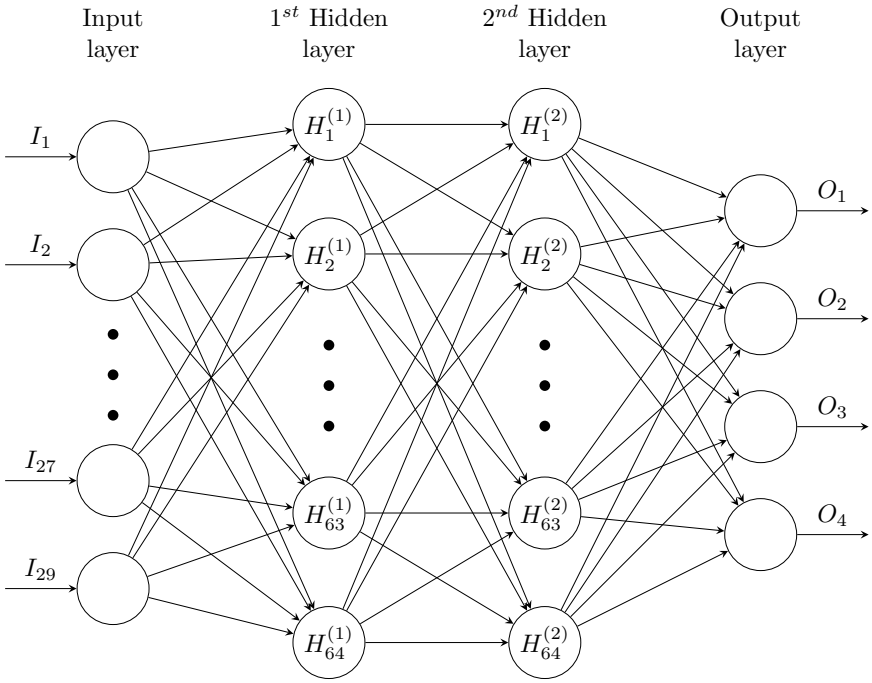


Figure 4.9: Actor network architecture

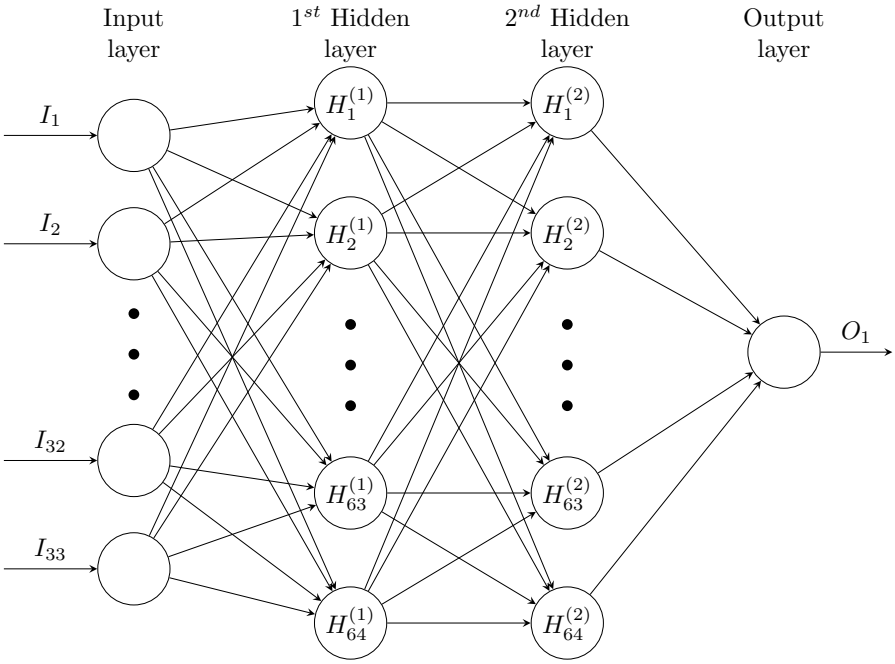


Figure 4.10: Critic network architecture

4.5 MDP formulation

The MDP in this study is partially observable, i.e. the agent does not have complete knowledge of the environment [28]. Observability is key to faster learning in RL. If the agent has entire knowledge of the environment, it is able to exhaustively decide its actions. Now, the definition of state space, action space and reward function is as given below.

The state space (s) is composed of: orientation (represented using rotation matrix R), Inertial Measurement Unit (measures linear acceleration v and angular velocity ω) data and rangefinders (d_r) data. The action space (a) consists of four values which represent the rotor commands.

$$s = \{R, v, \omega, d_r\} \qquad a = \{a_1, a_2, a_3, a_4\} \tag{4.1}$$

The reward function plays a significant role in accelerating learning. Consequently, the reward function is composed of three elements: penalty for orientation, reward/penalty for proximity to walls and penalty for collision.

Let $\{\angle_r, \angle_p\}$ be the angles along the roll and pitch axes, then the agent will be penalised according to euclidean norm. Therefore, if $\angle_r$ and $\angle_p$ are anything other than 0, then the agent will be penalized. Yaw is not considered in this formulation and its consequences will be evident later on, in the experiment result discussion.

$$\begin{aligned} \angle &= \{\angle_r, \angle_p\} \\ Rew(\angle) &= -\|\angle\|_2 \end{aligned} \tag{4.2}$$

The reward formulation for rangefinders is more straightforward. First, all sensor readings are divided by 3, so as to normalize within the range [0,1]. Then, if any sensor reading is less than $\frac{1}{3}$ ($\approx$0.34), then the reading is substituted with -14. This is to present a net negative reward, in case a single sensor has distance of less than 1m from the nearby wall and rest all sensors give a reading of 3m.

$$\begin{aligned} d_r &= \{d_{r_1}, d_{r_2}, \ldots d_{r_{14}}\} \\ Rew(d_r) &= \sum_{1}^{14} d_r \end{aligned} \tag{4.3}$$

The number of collisions in dm_control are accessed using *ncon* attribute in simulation data memory. Since there are no moving objects other than the quadrotor, if *ncon* is anything other than 0, the agent is heavily penalised and the episode in terminated.

$$Rew(ncon) = \begin{cases} 0 & ncon = 0 \\ -5000 & ncon > 0 \end{cases} \tag{4.4}$$

The final reward function (4.5) is formed with the composition of equations (4.2), (4.3) and (4.4).

$$r = Rew(\angle) + Rew(d_r) + Rew(ncon) \tag{4.5}$$

$$r = -\|\angle\|_2 + \sum_{1}^{14} d_r + Rew(ncon) \tag{4.6}$$

Chapter 5

Results and Discussion

After implementation of the quadrotor, training environment and the policy, its training succeeds. This chapter first mentions the parameters used for training. Then with the help of a few graphs, the performance of the agent has been depicted. Furthermore, random initialization was used at the beginning of each episode and has been shown with respective tables for training and testing. Furthermore, the performance of the agent was tested and compared to a random controller. Finally, a few points of discussion have been put forward which may be used by others that build upon this research.

5.1 Training Parameters

Name	Value
State space	29
Action space	4
Hidden layer nodes	64
Actor learning rate	0.001
Critic learning rate	0.002
Batch size	64
Discount factor (γ)	0.99
Timestep (t)	0.01
Number of Episodes	20000
Maximum Episode length	2000
Buffer Size	50000
target network update factor (τ)	0.995

Table 5.1: Parameters and their values

Parameters used during training are shown in table 5.1. The state space and action space were discussed in section 4.5 and thus don't warrant any further explanation.

The NN architectures have been formulated with reference to literature. Multiple combinations of number of nodes in hidden layers were not tested. It is possible that increasing or decreasing the hidden layer nodes might improve performance without varying any other parameter.

The learning rates for actor and critic networks used in this research are very standard values. Again it needs to be mentioned, that other values were not tested and there might be some scope to improve the learning experience of the agent by varying these values.

Initially, the batch size was set to 64 for first 10,000 episodes. However, it was later changed to 128 to help the agent learn over a bigger pool which did help in increasing the performance. The above mentioned parameters all have a possibility of improving performance. However, due to complex nature of DRL, the maximum amount of time was allocated to formulating the correct state and action spaces.

Initial experiments were run with a timestep of 0.001 but changing it to 0.01 did not make any difference. Thus the latter was used for training performance shown in section 5.2.

The total number of episodes for training is 20,000. However, it was trained in parts as is shown in later sections. the maximum episode length was chosen randomly, with 2000 timesteps equivalent to 20 s flight time. Critic and actor learning rates were experimented but were reverted to above mentioned values.

5.2 Training Performance

Parameter	Random function
Position x	$\mathcal{N}(0.0, 1.0)$
Position y	$\mathcal{N}(0.0, 1.0)$
Position z	$\mathcal{U}(2, 4)$
Linear velocity	$\mathcal{N}(0.0, 1.0)$
Angular velocity	$\mathcal{N}(0.0, 1.0)$

Table 5.2: Random initialization settings during training

Quadrotor was initialised randomly in the 3D space. Table 5.2 lists the various distributions used for random initialization for each parameter in position and velocity of the quadrotor. $\mathcal{N}(\mu, \sigma)$ denotes sampling from Gaussian distribution with mean μ and standard deviation σ. $\mathcal{U}(a, b)$ denotes sampling from uniform distribution with lower range a and upper range b(inclusive).

Figure 5.1: Average reward during training for 10,000 episodes

The first 10,000 episodes were in multiple sessions. This is mainly attributed to the under-powered machinery for such a gigantic task. Training for even 1000 episodes increased the processor temperature to very high levels. Thus, to prevent any damage to the underlying hardware, training for the first 10,000 episodes was not continuous.

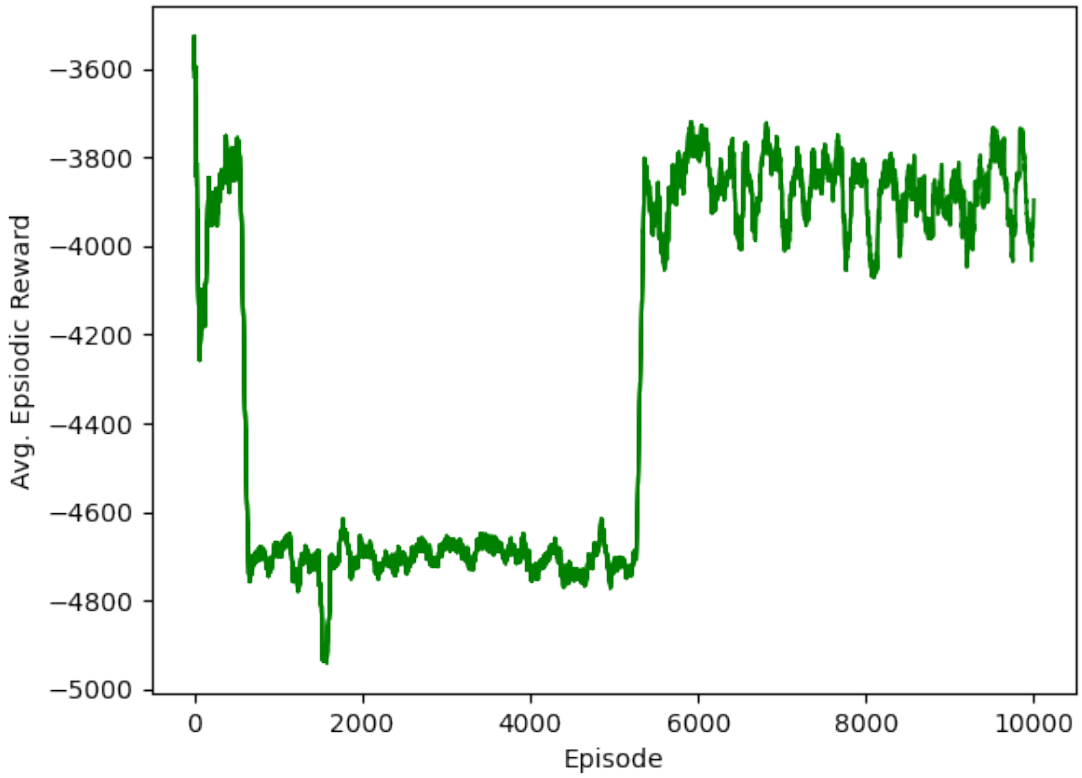


Figure 5.2: Average reward during training from 10,000 episodes to 20,000 episodes

All four NN were saved after each session and reloaded before each session. This helped in training the policy for hundreds of timesteps. Fig. 5.1(a) shows the first thousand episode training performance. For the initial 8,000 episodes, the penalty for collision was very high at -10,000. This lead to the quadrotor learning that it is best to not do anything and thus, free fall under gravity leads to the optimal average reward. This is clearly visible in fig. 5.1(a) to fig. 5.1(f).

After 8,000 episodes some different approach was required. Before continuing training from 8000 episodes, the collision reward was changed from -10,000 to -5,000. The fig. 5.1(f) shows the effect this small change had on the performance. Previously, it wasn't even able to get an average reward over 1000 (before collision) and now, after training for a mere 2000 episodes, it grossed an average reward of 1200 (before collision).

Training was resumed for another 10,000 episodes, totalling 20,000. The fig. 5.2 shows the average reward graph for the whole 10k episodes. It is important to mention that this was the first time, the agent had an average reward above 1450. However, quickly after this achievement, there was a drop in performance. This trend continued for the next 3500 episodes. Even though the agent was not performing at par with its previous achievement, it was learning from all the experiences. After about 5700 episodes since the start, there was a shift in actions this quadrotor was taking. It was back to achieving an average reward of 1200. The performance of the quadrotor is very similar in 1500-5600 episodes and for 6000-10000 episodes. This is down to how much exploration was performed by the agent. Since this is very

early in simulations, the agent does not know well the best actions and therefore, due to the closed space of operation, collision is imminent.

The problem for exploration was addressed by adding some noise in the output of the actor network. Initially it was set to $\mathcal{N}(0.5, 0.3)$ for the first 10,000 episodes. However, from 10,000 episode onwards, the noise was changed to $\mathcal{N}(0.0, 0.3)$.

Changing the collision penalty did improve the performance as the quadrotor started learning to fly and it also revealed a new tendency of the agent's actions, uncontrollable spin.

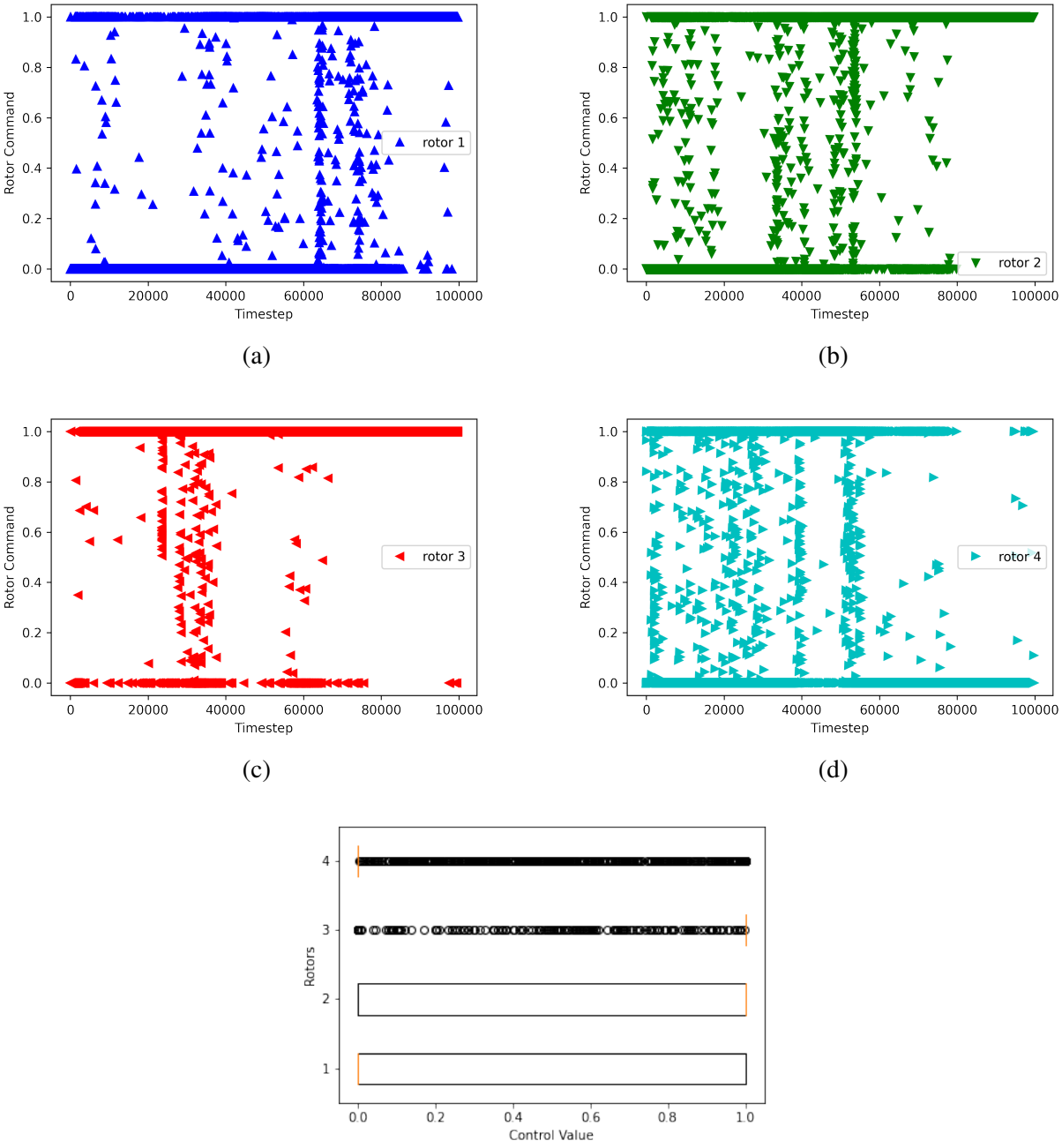


Figure 5.3: Rotor command value[0-1] after 9k episodes

The fig. 5.3 shows the rotor commands between the episodes ranging from 8000 to 9000. The figures 5.3(a) to 5.3(d) show the various actions the agent was outputting. Fig. 5.3(e) shows the box plot of these actions. This is graph is close to the ideal condition where the agent is varying its rotor commands to stabilize itself and prevent collisions. These 1000 episodes lasted for ≈ 1,00,00 timesteps or 1000 s or 16 min.

The fig. 5.4 shows the rotor commands between the episodes ranging from 19,000 to 20,000. The fig. 5.4(a) and fig. 5.4(c) have the rotor command set to 1 whereas, fig. 5.4(b) and 5.4(d) have the rotor command set to 0. The fig. 5.4(e) shows a box plot of the actions taken.

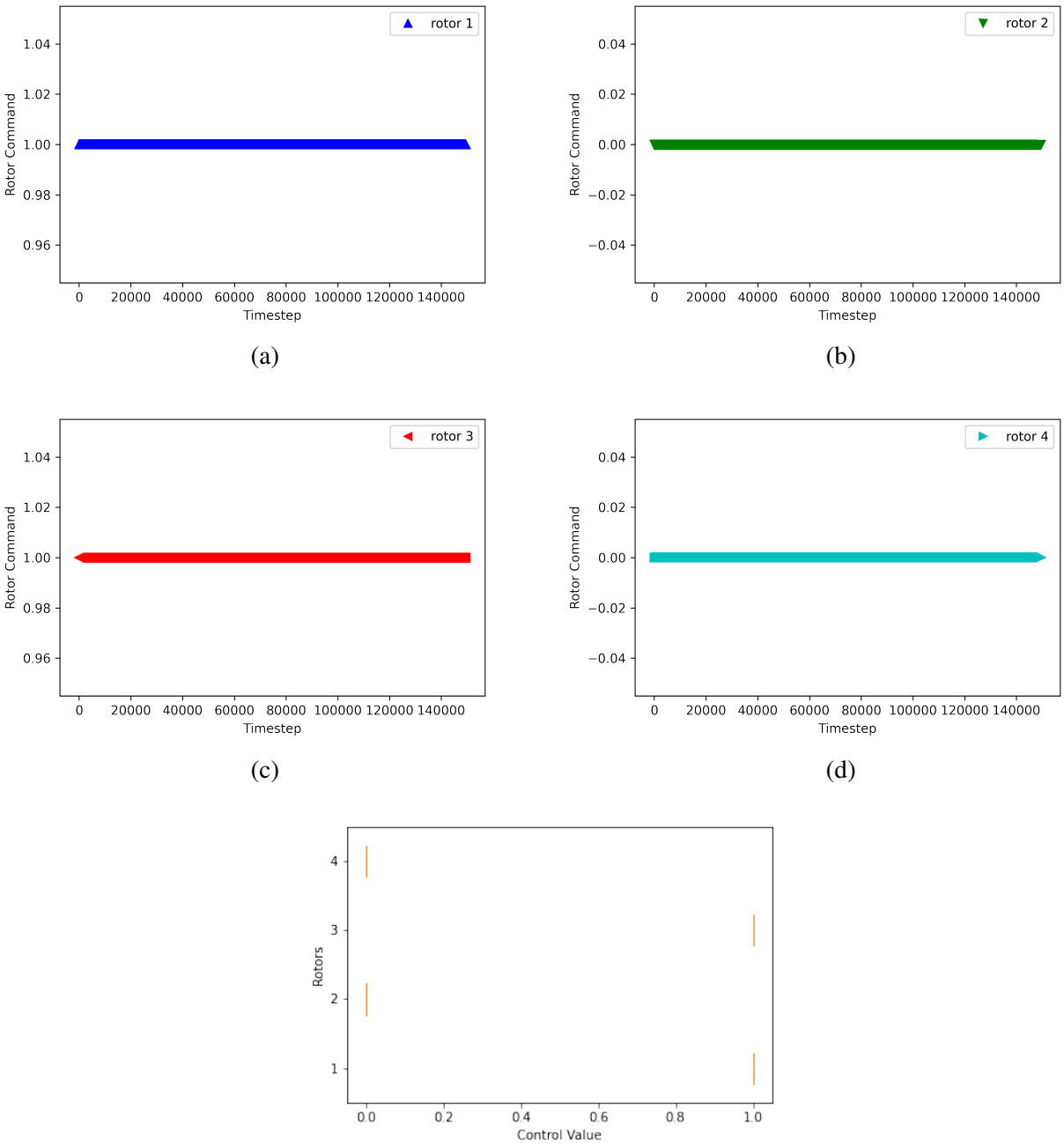
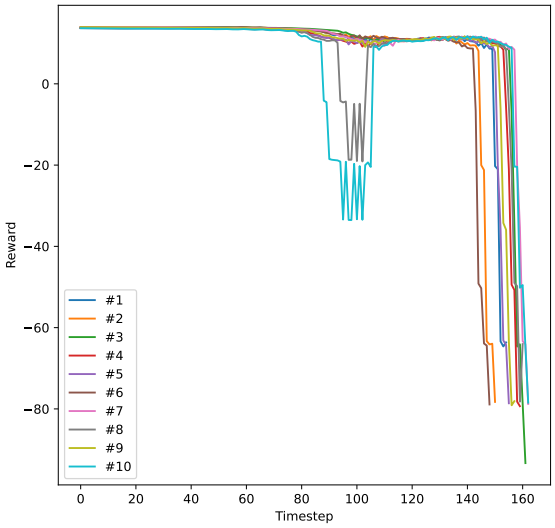
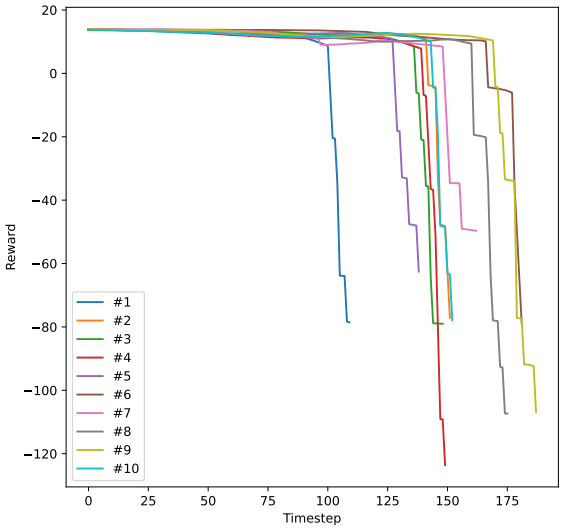


Figure 5.4: Rotor command value[0-1] after 20k episodes

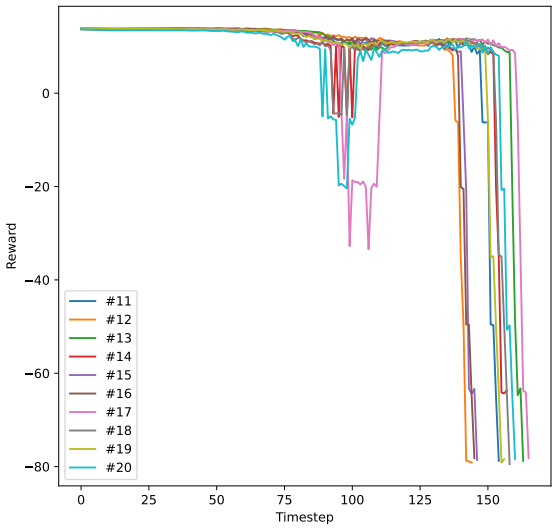
5.3 Testing Performance



(a) DRL controller after 10k



(b) Random controller

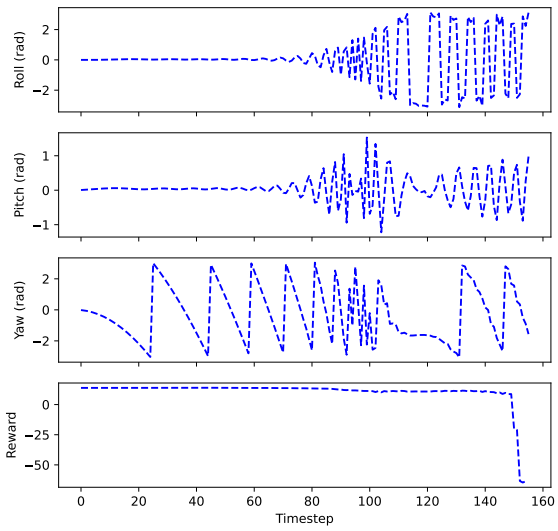


(c) DRL controller after 20k

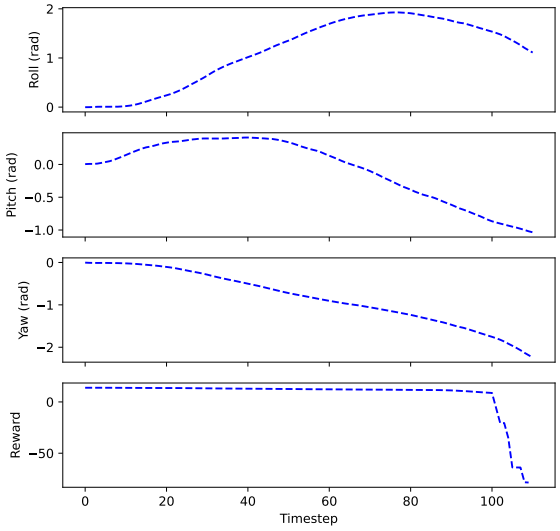
Figure 5.5: Performance over 10 test runs

Fig. 5.5(a) and fig. 5.5(c) showcase the performance of proposed controller over 10 test runs after training for 10,000 episodes and 20,000 episodes respectively. Fig. 5.5(b) shows the performance of random controller over 10 test runs. The actions were sampled from a Gaussian distribution denoted as $\mathcal{N}(0.5, 0.3)$. The mean of previously mentioned distribution is not zero, which has a valid reason. The control input range of each rotors is [0-1] and from manual testing in simulation, it was found that the quadrotor re-

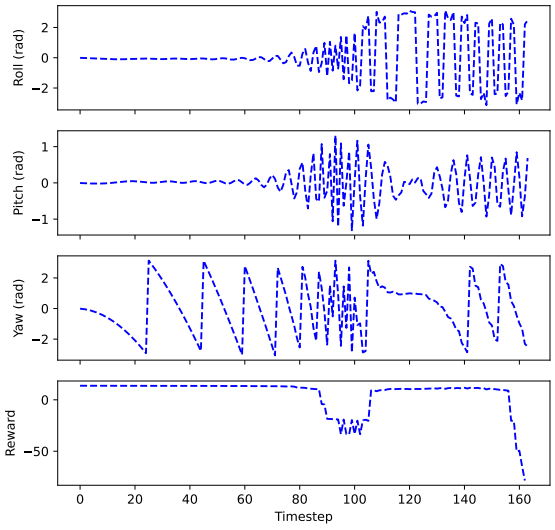
quires a value of 0.34 and above to create a positive thrust in the z-direction. Otherwise, the quadrotor starts to fall under gravity. Therefore, the input noise of each rotor will be between [0.3-0.8] for each rotor with 68.26% probability, according to $\mathcal{N}(0.5, 0.3)$.



(a) DRL controller after 10k



(b) Random controller



(c) DRL controller after 20k

Figure 5.6: Analysis of single testcase for roll, pitch, yaw, and reward

Testing results show a similar performance over 10 test runs for both the trained policy and the random policy. This is not a satisfactory result because a random controller survives for the same amount of time as that of the learned policy. This is down to the fact that the proposed model has not been trained for even a million timesteps. Another reason is DDPG takes longer than algorithms like PPO and TRPO

to converge [23].

Fig. 5.6(a) and Fig. 5.6(c) show the performance of the first test run of DRL controller at each timestep. Initially, the roll and pitch angle are near-perfect zero but as soon as the quadrotor closes in on any walls, it takes evasive action from which it cannot recover and crashes. Fig. 5.6(b) shows a more stable performance. However, due to being randomly controlled, it does eventually crash.

The initialization parameter for testing is shown in table 5.3.

Parameter	Random function
Position x	$\mathscr{N}(0.0, 0.5)$
Position y	$\mathscr{N}(0.0, 0.5)$
Position z	$\mathscr{U}(2, 4)$
Linear velocity	$\mathscr{N}(0.0, 0.5)$
Angular velocity	$\mathscr{N}(0.0, 0.5)$

Table 5.3: Random initialization settings during testing

5.4 Discussion

There were various factors for the insufficient performance in above graphs. These are:

- The agent was trained for insufficient amount. Rather than performing an analysis after training for 10^7 timesteps, the graphs only show the performance for 20,000 episodes, which equates to $\approx 3 \times 10^5$ timesteps.
- The reward function also has a role to play in this misfortune. In a complete oversight, the yaw angle was not included in the reward calculation. It was assumed that the direction of facing bears no significance to the overall problem and thus the exclusion from the reward function. During the learning phase, however, the agent learned that it necessary to stabilize rotation about roll and pitch axes. This lead to the agent setting two rotors(on the same diagonal) to 1 and the other two rotors to 0. This action meant the quadrotor was able to stabilize the roll and pitch axes but induced an uncontrollable spin about the yaw axis.
- It is prudent to say that DDPG is not the preferred choice for training the proposed system on an extremely underpowered machine. During training, the machine reached alarming temperatures and to prevent any damage, the system was trained in batches.
- Initially it was thought that maybe there was an issue in the implementation of DDPG. However, after noticing the behaviour the agent exhibited with regards to using only two diagonal rotors for thrust, it was clear that the agent was learning just not what was expected.

- Even though the results don't show this, there is still faith in the proposed system that it might work, albeit after training rigorously. The reward function shown in section 4.4 was reached after a few iterations. The earliest iteration rewarded the agent for each timestep that it survived. This meant there was no weightage to the actual orientation. An upside-down configuration earned the same reward as a perfectly stable orientation.
- One definite way to improve the training is to introduce a penalty on the basis of linear and angular velocities. If the agent is penalised for high velocities, then in theory it should be able to take precautionary actions to prevent such states. It was decided that the quadrotor should be allowed to achieve high velocities as long as it is able to prevent collisions. This theory is still not proven.

Chapter 6

Conclusion

6.1 Conclusion

This research work mainly focused on developing an autonomous quadrotor capable of flying in indoor environment. A model of quadrotor and the training environment was created in MuJoCo. A control policy was created with tensorflow, keras and trained using off-policy actor critic algorithm, DDPG. Various reward functions were designed along the way and trained on. However, the one mentioned in this thesis was derived too late to train for significant amount of timesteps. The performance of the agent has been shown using various graphs and also compared to a benchmark, that is, a random controller. Both the random policy and the learned policy survive for similar amount of timesteps. This agent, in its current state, is note ready to be tested on a real quadrotor, nor is it capable of being compared with any of the previous works by other researchers.

6.2 Future Work

Future work may include changing the reward function to include linear velocity, angular velocity, a penalty for yaw, training for significant amount of time or changing the learning algorithm to PPO or TRPO. These algorithms converge faster than DDPG. The parameters of the actor and critic networks may be tested with different amount of nodes and activation function or completely different architecture.

References

[1] H. Eisenbeiss *et al.*, "A mini unmanned aerial vehicle (uav): system overview and image acquisition," *International Archives of Photogrammetry. Remote Sensing and Spatial Information Sciences*, vol. 36, no. 5/W1, pp. 1–7, 2004.

[2] "The jump to light speed is a real killer." https://www.scientificamerican.com/article/star-wars-science-light-speed/, August 2018. Accessed: 27-May-2022.

[3] J. F. Guilmartin, "unmanned aerial vehicle." https://www.britannica.com/technology/unmanned-aerial-vehicle. Accessed: 27-May-2022.

[4] H. Yao, R. Qin, and X. Chen, "Unmanned aerial vehicle for remote sensing applications—a review," *Remote Sensing*, vol. 11, no. 12, p. 1443, 2019.

[5] N. Gageik, P. Benz, and S. Montenegro, "Obstacle detection and collision avoidance for a uav with complementary low-cost sensors," *IEEE Access*, vol. 3, pp. 599–609, 2015.

[6] H. H. G. Coppejans and H. C. Myburgh, "A primer on autonomous aerial vehicle design," *Sensors*, vol. 15, no. 12, pp. 30033–30061, 2015.

[7] "Uav applications." https://nesac.gov.in/scientific-programmes/remote-sensing-and-gis/uav-applications/. Accessed: 27-May-2022.

[8] ISRO, "Applications of unmanned aerial vehicle (uav) based remote sensing in ne region." https://www.isro.gov.in/applications-of-unmanned-aerial-vehicle-uav-based-remote-sensing-ne-region. Accessed: 27-May-2022.

[9] G. Dean, "Drone racing: Everything you need to know." https://www.space.com/drone-racing-explained, December 2021. Accessed: 08-July-2022.

[10] CNBCTV18.com, "Drones may soon deliver food, medicines and groceries in india." https://www.cnbctv18.com/technology/drones-may-soon-deliver-food-medicines-and-groceries-in-india-12613082.htm, Feb 2022. Accessed: 08-July-2022.

[11] C. Greatwood and A. G. Richards, "Reinforcement learning and model predictive control for robust embedded quadrotor guidance and control," *Autonomous Robots*, vol. 43, no. 7, pp. 1681–1693, 2019.

[12] S. Stevšić, T. Nägeli, J. Alonso-Mora, and O. Hilliges, "Sample efficient learning of path following and obstacle avoidance behavior for quadrotors," *IEEE Robotics and Automation Letters*, vol. 3, no. 4, pp. 3852–3859, 2018.

[13] H.-Y. Lin and X.-Z. Peng, "Autonomous quadrotor navigation with vision based obstacle avoidance and path planning," *IEEE Access*, vol. 9, pp. 102450–102459, 2021.

[14] S. D. Morad, R. Mecca, R. P. K. Poudel, S. Liwicki, and R. Cipolla, "Embodied visual navigation with automatic curriculum learning in real environments," *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 683–690, 2021.

[15] J. Hwangbo, I. Sa, R. Siegwart, and M. Hutter, "Control of a quadrotor with reinforcement learning," *IEEE Robotics and Automation Letters*, vol. 2, no. 4, pp. 2096–2103, 2017.

[16] J. Ou, X. Guo, M. Zhu, and W. Lou, "Autonomous quadrotor obstacle avoidance based on dueling double deep recurrent q-learning with monocular vision," *Neurocomputing*, vol. 441, pp. 300–310, 2021.

[17] W. Lou and X. Guo, "Adaptive trajectory tracking control using reinforcement learning for quadrotor," *International Journal of Advanced Robotic Systems*, vol. 13, no. 1, p. 38, 2016.

[18] C.-H. Pi, K.-C. Hu, S. Cheng, and I.-C. Wu, "Low-level autonomous control and tracking of quadrotor using reinforcement learning," *Control Engineering Practice*, vol. 95, p. 104222, 2020.

[19] B. Rubí, B. Morcego, and R. Pérez, "Deep reinforcement learning for quadrotor path following with adaptive velocity," *Autonomous Robots*, vol. 45, no. 1, pp. 119–134, 2021.

[20] C.-H. Pi, W.-Y. Ye, and S. Cheng, "Robust quadrotor control through reinforcement learning with disturbance compensation," *Applied Sciences*, vol. 11, no. 7, p. 3257, 2021.

[21] M. Jafari and H. Xu, "Intelligent control for unmanned aerial systems with system uncertainties and disturbances using artificial neural network," *Drones*, vol. 2, no. 3, 2018.

[22] C. Xiao, P. Lu, and Q. He, "Flying through a narrow gap using end-to-end deep reinforcement learning augmented with curriculum learning and sim2real," *IEEE Transactions on Neural Networks and Learning Systems*, 2021.

[23] W. Koch, R. Mancuso, R. West, and A. Bestavros, "Reinforcement learning for uav attitude control," *ACM Trans. Cyber-Phys. Syst.*, vol. 3, feb 2019.

[24] N. O. Lambert, D. S. Drew, J. Yaconelli, S. Levine, R. Calandra, and K. S. Pister, "Low-level control of a quadrotor with deep model-based reinforcement learning," *IEEE Robotics and Automation Letters*, vol. 4, no. 4, pp. 4224–4230, 2019.

[25] X. Lin, Y. Yu, and C. Sun, "Supplementary reinforcement learning controller designed for quadrotor uavs," *IEEE Access*, vol. 7, pp. 26422–26431, 2019.

[26] R. Niemiec and F. Gandhi, "A comparison between quadrotor flight configurations," 2016.

[27] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[28] "Key concepts in rl." `https://spinningup.openai.com/en/latest/spinningup/rl_intro.html`, March 2018. Accessed: 26-May-2022.

[29] J. Castaño, "Making a quadcopter axis system in tikz." https://tex.stackexchange.com/questions/637627/making-a-quadcopter-axis-system-in-tikz, March 2022. Accessed: 18-May-2022.

[30] S. Bouabdallah, P. Murrieri, and R. Siegwart, "Design and control of an indoor micro quadrotor," in *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA '04. 2004*, vol. 5, pp. 4393–4398 Vol.5, 2004.

[31] "Deep deterministic policy gradient." `https://spinningup.openai.com/en/latest/algorithms/ddpg.html`, March 2018. Accessed: 18-May-2022.

[32] H. Parwana and M. Kothari, "Quaternions and attitude representation," *arXiv preprint arXiv:1708.08680*, 2017.

[33] M. Ben-Ari, "A tutorial on euler angles and quaternions." `https://www.weizmann.ac.il/sci-tea/benari/sites/sci-tea.benari/files/uploads/softwareAndLearningMaterials/quaternion-tutorial-2-0-1.pdf`, 2014. Accessed: 25-May-2022.

[34] M. Cheng, Y. Chen, and M. Humphreys, "Rotation matrix and tilt about x/y/z in opticstudio." `https://support.zemax.com/hc/en-us/articles/1500005576822-Rotation-Matrix-and-Tilt-About-X-Y-Z-in-OpticStudio`, March 2021. Accessed: 26-May-2022.

[35] "Interference in ultrasonic and ir range sensors." `https://connectivity.unh.edu/blog/interference-in-ultrasonic-and-ir-range-sensors.html`, April 2017. Accessed: 02-June-2022.

[36] E. Todorov, T. Erez, and Y. Tassa, "Mujoco: A physics engine for model-based control," in *2012 IEEE/RSJ international conference on intelligent robots and systems*, pp. 5026–5033, IEEE, 2012.

[37] S. Tunyasuvunakool, A. Muldal, Y. Doron, S. Liu, S. Bohez, J. Merel, T. Erez, T. Lillicrap, N. Heess, and Y. Tassa, "dm_control: Software and tasks for continuous control," *Software Impacts*, vol. 6, p. 100022, 2020.

[38] "Overview." https://mujoco.readthedocs.io/en/latest/overview.html#units-are-undefined, March 2022. Accessed: 23-May-2022.

Plagiarism Report

Annexure-

Gautam Buddha University
Plagiarism Verification

Date. 21/06/2022

Title of the Thesis: Implementation of Deep Reinforcement Learning for Collision Prevention in Quadrotor

Researcher: Rishabh Goel

Supervisor: Dr. Nawaid Z. Rizvi

Department: CS

School: ICT

Email ID (Researcher): 17ics073@gbu.ac.in

Email ID (Supervisor): nawaid@gbu.ac.in

This is to report that the above thesis was scanned for similarity detection. Process and outcome are given below:

Software used: Turnitin Date: 17.06.2022 Total Page: 42

Similarity Index: 3% Submission ID: 1858472310 Total Word count: 10,451

The complete Similarity Report has been sent to the supervisor and researcher on the above email.

21/6/2022
Deputy Librarian
Bodhisattva Dr. B.R. Ambedkar Library
Gautam Buddha University,
Greater Noida

Deputy Librarian

The complete report of the above thesis has been reviewed by the undersigned. (Check box √)

	Similarity index is below accepted norms.
	The similarity index is above accepted norms, because of the following reasons:

1.....

2.....

3.....

4.....

5.....

The thesis may be considered for the award of degree. (Relevant documents attached.)

Rishabh Goel
21/06/2022
Signature of Student

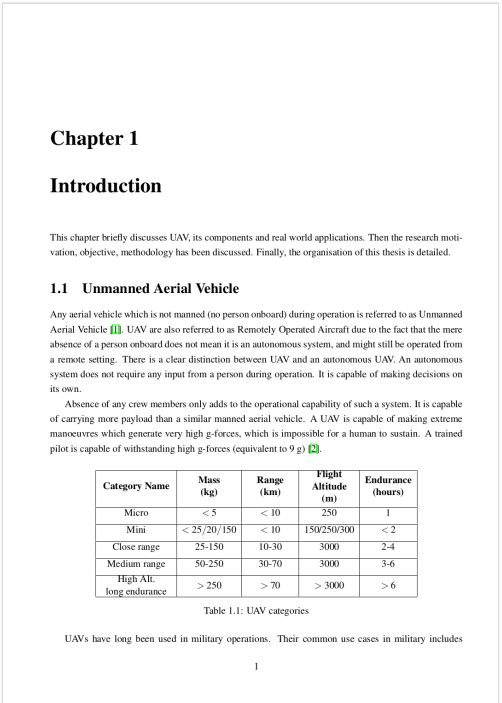
Signature of Supervisor

Digital Receipt

This receipt acknowledges that Turnitin received your paper. Below you will find the receipt information regarding your submission.

The first page of your submissions is displayed below.

Submission author:	Rishabh Goel ...
Assignment title:	Information Communication Technology
Submission title:	Implementation of Deep Reinforcement Learning for Collisio...
File name:	Report.pdf
File size:	7.97M
Page count:	42
Word count:	10,451
Character count:	52,101
Submission date:	17-Jun-2022 04:36PM (UTC+0530)
Submission ID:	1858472310



Chapter 1

Introduction

This chapter briefly discusses UAV, its components and real world applications. Then the research motivation, objective, methodology has been discussed. Finally, the organisation of this thesis is detailed.

1.1 Unmanned Aerial Vehicle

Any aerial vehicle which is not manned (no person onboard) during operation is referred to as Unmanned Aerial Vehicle [1]. UAV are also referred to as Remotely Operated Aircraft due to the fact that the mere absence of a person onboard does not mean it is an autonomous system, and might still be operated from a remote setting. There is a clear distinction between UAV and an autonomous UAV. An autonomous system does not require any input from a person during operation. It is capable of making decisions on its own.

Absence of any crew members only adds to the operational capability of such a system. It is capable of carrying more payload than a similar manned aerial vehicle. A UAV is capable of making extreme manoeuvres which generate very high g-forces, which is impossible for a human to sustain. A trained pilot is capable of withstanding high g-forces (equivalent to 9 g) [2].

Category Name	Mass (kg)	Range (km)	Flight Altitude (m)	Endurance (hours)
Micro	< 5	< 10	250	1
Mini	< 25/20/150	< 10	150/250/300	< 2
Close range	25-150	10-30	3000	2-4
Medium range	50-250	30-70	3000	3-6
High Alt. long endurance	> 250	> 70	> 3000	> 6

Table 1.1: UAV categories

UAVs have long been used in military operations. Their common use cases in military includes

1

Document Viewer

Turnitin Originality Report

Processed on: 17-Jun-2022 4:37 PM IST
ID: 1858472310
Word Count: 10451
Submitted: 1

Implementation of Deep Reinforcement Learning... By Rishabh Goel ...

Similarity Index	Similarity by Source	
3%	Internet Sources:	3%
	Publications:	2%
	Student Papers:	2%

include quoted include bibliography excluding matches < 14 words mode: quickview (classic) report

Change mode print refresh download

1% match (student papers from 17-Jun-2016)
Submitted to University of Auckland on 2016-06-17

<1% match (Internet from 15-Oct-2021)
https://coek.info/pdf-investigation-on-a-two-part-underwater-manoeuvrable-towed-system-.html

<1% match (student papers from 12-Oct-2017)
Submitted to University of Southern Queensland on 2017-10-12

<1% match (publications)
F. El-Hawary. "Compensation of vertical displacement components in marine seismic applications using the coupled heave and pitch model", OCEANS 88 A Partnership of Marine Interests Proceedings, 1988

<1% match (publications)
"Collisions", Physics for Game Programmers, 2005

<1% match (publications)
Industrial Robot: An International Journal, Volume 43, Issue 1 (2016)

<1% match ()
Kokossalakis, George, 1976-. "Neural network based active structural control", Massachusetts Institute of Technology, 2000

<1% match ()
Dzwonkowski, Mariusz, Rykaczewski, Roman. "Quaternion Feistel Cipher with an In nite Key SpaceBased on Quaterni on Julia Sets, Journal of Telecommunications and Information Technology, 2015, nr 4", National Institute of Telecommunications, 2015

<1% match (Internet from 28-Oct-2021)
https://ebin.pub/digital-cities-roadmap-9781119791591.html

<1% match (Internet from 01-Dec-2021)
https://upcommons.upc.edu/bitstream/handle/2117/355310/161112.pdf?isAllowed=y&sequence=1

<1% match (Internet from 27-Mar-2019)
https://epdf.tips/handbook-of-medical-imaging-processing-and-analysis-management.html

<1% match ()
Harwani, Yuli, Suharjo, Budi, Nurmalina, Rita, Suprayitno, Gendut. "MINAT PEMILIHAN PERGURUAN TINGGI DAN PERAN KELOMPOK REFERENSI SERTA KOMUNIKASI PEMASARAN TERINTEGRASI", 'Universitas Mercu Buana', 2018

Chapter 1 Introduction This chapter briefly discusses UAV, its components and real world applications. Then the research moti- vation, objective, methodology has been discussed. Finally, the organisation of this thesis is detailed. 1.1 Unmanned Aerial Vehicle Any aerial vehicle which is not manned (no person onboard) during operation is referred to as Unmanned Aerial Vehicle [1]. UAV are also referred to as Remotely Operated Aircraft due to the fact that the mere absence of a person onboard does not mean it is an autonomous system, and might still be operated from a remote setting. There is a clear distinction between UAV and an autonomous UAV. An autonomous system does not require any input from a person during operation. It is capable of making decisions on its own. Absence of any crew members only adds to the operational capability of such a system. It is capable of carrying more payload than a similar manned aerial vehicle. A UAV is capable of making extreme manoeuvres which generate very high g-forces, which is impossible for a human to sustain. A trained pilot is capable of withstanding high g-forces (equivalent to 9 g) [2]. Category Name Mass (kg) Range (km) Altitude (m) Endurance (hours) Flight Micro < 5 < 10 250 1 Mini < 25/20/150 < 10 150/250/300 < 2 Close range 25-150 10-30 3000 2-4 long endurance > 250 > 70 > 3000 > 6 Medium range High Alt. 50-250 30-70 3000 3-6 Table 1.1: UAV categories UAVs have long been used in military operations. Their common use cases in military includes 1 reconnaissance, ordnance mounted UAV for use in battlefield and light handheld versions to help with planning by troops [3]. Other use cases include remote sensing [4] and search and rescue deployment and many more. Table 1.1 shows a small extract of various categories of UAVs [1]. This study deals with Micro UAV with mass less 5kg. Various sensor types [4, 5] used on UAV are: • RGB Camera • Multi-spectral Camera • Hyper spectral Camera • Thermal Infrared Camera • LiDAR • Ultrasonic • Infrared 1.2 Components of Micro Aerial Vehicle Two complicated components of developing a MAV is the position and attitude control [6]. An attitude controller is responsible for the orientation/state of the MAV in the 3D space. A quadrotor is capable of moving around by varying its roll, pitch and yaw. This variation of is enacted by applying different rotor commands to each rotor. An onboard Inertial Measurement Unit helps provide the necessary data for the controller to estimate its state and provide necessary correcting changes. A position controller is responsible for reaching the goal location in the 3D space. It outputs the necessary commands for the quadrotor to reach its goal state. It is possible that the goal state is dynamic and therefore the controller has to adapt to the uncertainty. Current position of the MAV is estimated with the help of Simultaneous Localization and Mapping (SLAM) based algorithms.