Fault Tolerant Quadcopter Control using Reinforcement Learning

MECH0020 Individual Project
AY 2022/23

Student Name: Fidhal Kotta
Student Number: [redacted]
Supervisor: Professor Giles Thomas

May 22, 2023

Abstract

The increasing adoption of Unmanned Aerial Vehicles (UAVs) across various industries has spurred a significant demand for fault-tolerant control schemes in quadcopters. Undetected/uncompensated faults in quadcopters could lead to closed-loop instability, eventually causing harm to people or possessions nearby.

This study employed Reinforcement Learning (RL) to develop a controller capable of maintaining safe flight in both nominal and faulty conditions. To investigate quadcopter faults, this work extended an existing simulation framework by incorporating variable fault injection during training and simulation stages, continuous wind disturbances, and compatibility with Stable-Baselines3. This enhanced environment facilitated the exploration of different reward strategies in the nominal case to identify the most effective method for training quadcopters: one with a negative squared error (Model A), and another with a Gaussian curve-based error (Model B).

Model B outperformed Model A and was thus utilised in a case study examining performance under faulty conditions. The final model's performance was compared to classical control techniques. Model B effectively maintained stability in both nominal and faulty cases, showing a significant improvement over a Proportional Integral Derivative (PID) controller. While both models exhibit closed-loop unstable dynamics when faults are injected, the RL model is shown to tolerate higher values of loss of effectiveness. This study demonstrates the successful implementation of RL for robust Passive Fault Tolerant Control (PFTC) within a simulated environment. Due to the assumptions and simplifications made in the simulation, the current model is not completely representative of a real-world quadcopter implementation. Future research should strive to capture real-world conditions more accurately and transition this model to real-world applications.

Acknowledgements

I would like to thank my academic supervisor, Professor Giles Thomas, for his invaluable guidance, support and feedback throughout the entire project. I would also like to extend my thanks to Davide Grande, who provided me with technical assistance, insightful discussions and a fresh perspective on several aspects of this work. His enthusiasm and willingness to help were truly appreciated. Lastly, I would like to acknowledge the contributions of Dr Andrea Grech La Rosa, another academic who offered his support and insights during the course of this study.

Code

All of the code used for the development of the quadcopter simulation as well as the data processing is publicly available here:

`https://github.com/fidhalkotta/gym-copter-FTC`

Word Count

Total word count is **7473.**

Declaration

I, Fidhal Kotta, confirm that the work presented in this report is my own. Where information has been derived from other sources, I confirm that this has been indicated in the report.

Contents

Acronyms

A2C Advantage Actor Critic.
AFTC Active Fault-Tolerant Control.
AUV Autonomous Underwater Vehicle.

DDPG Deep Deterministic Policy Gradient.
DNN Deep Neural Network.
DQN Deep Q Networks.

FPS Frames Per Second.
FTC Fault-Tolerant Control.

ML Machine Learning.

PFTC Passive Fault-Tolerant Control.
PID Proportional Integral Derivative.
PPO Proximal Policy Optimisation.

RL Reinforcement Learning.
RMSE Root Mean Squared Error.

SAC Soft Actor Critic.
SMC Sliding Mode Control.
SSE Steady State Error.

TD3 Twin Delayed Deep Deterministic Policy Gradient.
TRPO Trust Region Policy Optimisation.

UAVs Unmanned Aerial Vehicles.

List of Figures

List of Tables

Nomenclature

Subscripts

d	Variable target value
e	Error in variable
$RMSE$	Root Mean Squared Error of the variable
SSE	Steady State Error of the variable
T	Total number of timesteps in the episode
t	Variable value at a specific time

Superscripts

$*$	Optimal variable value
$\dot{a}$	Time derivative of variable a
π	Variable value for a specific policy

Symbols

$\dot{\boldsymbol{\eta}}$	Attitude rate vector
$\dot{\boldsymbol{\zeta}}$	Absolute linear velocity vector
$\boldsymbol{\eta}$	Attitude vector error with Euler Angles
$\boldsymbol{\eta}$	Attitude vector in the inertial frame with Euler Angles
$\boldsymbol{\nu}$	Angular velocity in the body frame
$\boldsymbol{\zeta_e}$	Absolute linear position vector error
$\boldsymbol{\zeta}$	Absolute linear position vector
$\boldsymbol{a}$	Action space
$\boldsymbol{f}$	Fault matrix
$\boldsymbol{g_\nu(a)}$	Angular velocity Gaussian transformation function taking in variable a
$\boldsymbol{g_\zeta(a)}$	Positional Gaussian transformation function taking in variable a
$\boldsymbol{m}$	Command values for the quadcopter motors
$\boldsymbol{s_{obs}}$	Observation state space
$\boldsymbol{u}$	Control input to the dynamic model
$\boldsymbol{W_\eta}$	Angular velocity transformation matrix
$\boldsymbol{W}$	Wind force vector
ϕ	Euler roll angle [rad]
$\pi(s, a)$	Policy function
ψ	Euler yaw angle [rad]
σ_ν	Standard deviation for $\boldsymbol{g_\nu(a)}$
σ_ζ	Standard deviation for $\boldsymbol{g_\zeta(a)}$
θ	Euler pitch angle [rad]

θ^{μ}	Policy Deep neural neural network
θ^{Q}	Q Deep neural Neural network
$\theta^{\mu'}$	Target Policy Deep neural neural network
$\theta^{Q'}$	Target Q Deep neural neural network
$\tilde{f}$	Fault magnitude in any one of the four motors
a	Action space
f_i	Effectiveness of motor i
h	Termination boolean
l_x	Limit value in the x position
l_y	Limit value in the y position
l_z	Limit value in the z position
p	Penalty for termination
p	Pitch angular velocity [$\mathrm{rad\,s^{-1}}$]
p	Roll angular velocity [$\mathrm{rad\,s^{-1}}$]
p	Yaw angular velocity [$\mathrm{rad\,s^{-1}}$]
$Q(s, a)$	Action value function
$Q^*(s, a)$	Optimal action value function
$Q^{\pi}(s, a)$	Action value function for policy π
$r(s, a)$	Reward function
r_T	Total cumulative reward up until time T
r_t	Reward at time t
s	State space
T_s	Settling time [s]
w_f	Wind force magnitude
w_k	Change in wind force constant
w_p	Weight for p term
w_q	Weight for q term
w_r	Weight for r term
W_x	Wind force in the x direction
w_x	Weight for x term
W_y	Wind force in the x direction
w_y	Weight for y term
W_z	Wind force in the x direction
w_z	Weight for z term
x	Position in x axis [m]
y	Position in y axis [m]
z	Position in z axis [m]

Chapter 1

Introduction

1.1 Background

Over the past few years, the popularity of Unmanned Aerial Vehicles (UAVs) has risen greatly as the technology has become cheaper and more accessible to the general population. A quadcopter is a robotic platform that uses four thrusters to generate lift and preserve stability during flight. As more research is carried out into quadcopter control, these platforms begin to be employed in applications such as rescue missions [1] or humanitarian aid [2].

Quadcopter controls are usually designed with various forms of Proportional Integral Derivative (PID) architectures [3]. Typically, a double-layered architecture is devised, where an inner-loop is tasked with solving the attitude control problem, whilst the outer-loop is designed to follow mission-level aims, such as path-following or position control. PID controllers are designed on linearised dynamics, that proved to be sufficiently descriptive when the environmental disturbances are bounded and when model uncertainties are low. Due to their linearised nature, PID controllers deliver sub-optimal performance when faced with scenarios deviating from the nominal conditions [4]. One such challenging problem is represented by the loss of one motor or its reduced efficiency as seen in Fig 1.1. Fault-Tolerant Control (FTC) techniques tackle the scenarios encompassing issues arising from the system.[5].

Research into FTC aims to develop control systems either to increase the robustness of the controller against unintended events or to actively compensate for a detected fault. Undetected/uncompensated faults in quadcopters could lead to closed-loop instability, eventually causing harm to people or possessions nearby. FTC techniques allow quadcopters to cope with faults and consequently increase the platform's safety.

Figure 1.1: A quadcopter attempting to keep stable flight with one rotor failure [6]

A potential approach to solve the FTC problem is using Reinforcement Learning (RL). This Machine Learning (ML) approach relies on an agent, which is an intelligent system that can perceive its environment and take decisions to achieve its goal. For every taken action the agent is given a reward, a value quantifying the quality of the selected action. The agent attempts to maximise the reward by iteratively tuning its decision process in a procedure called training. It has been shown that well-optimised networks can outperform the best humans at complex tasks such as the games

of Chess or Go [7]. In this study, RL will be employed to automatically design control policies that can preserve closed-loop stability when a quadcopter loses efficiency in its motors.

1.2 Problem

Designing FTC laws for quadcopters currently relies on the dynamics of the model [8], meaning the control solutions require specific designs for each different quadcopter and fault model. FTC techniques also make the assumption that faults are detected instantaneously and fully known. This assumption is particularly critical for quadcopters, as even small delays in the identification of faults can be the cause of instabilities, with catastrophic outcomes [9]. RL is an efficient tool to systematically develop model-free control laws that can account for uncertainty in the systems, such as a faulty thruster. Specifically, RL can be used to design a robust control law that does not rely on online information about the occurrence of the fault.

1.3 Aim

The project aims to use RL to develop a controller that can maintain the stability of a quadcopter in the presence of actuator faults. Training and testing of the control system shall be carried out in a simulation environment developed for quadcopter control. The response of the devised RL-based FTC system will be compared with traditional techniques both in the nominal and faulty cases.

Chapter 2

Literature Review

This chapter reviews the state-of-the-art RL methods and FTC of quadcopters.

2.1 Reinforcement Learning

This section contains an introduction to RL and a comparison of the state-of-the-art methods.

RL is a ML technique in which an agent learns a policy to optimally react to its environment, a policy being a set of algorithms that map states to actions. RL tackles an optimisation problem in which the optimal solution is the policy which maximises the expected reward from an action in the environment [10]. A Markov decision process models the action selection depending on the action-value function $Q(s_t, a_t)$, which represents an estimate of the future reward. A schematic of RL can be seen in Fig. 2.1 and all algorithms will consist of the following components:

- **State Space** s_t: A set of discrete or continuous multi-variate states observable by the agent at time step t.
- **Action Space** a_t: A set of discrete or continuous actions the agent can take in the environment at time step t.
- **Reward Function** $r(s, a)$: A function that maps a state s_t and action a_t to a reward value.
- **Policy Function** $\pi(s, a)$: A function that determines the likelihood of the agent picking a specific action given a specific state.

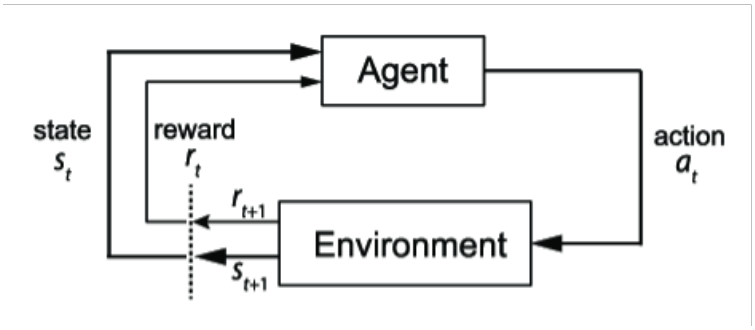

Figure 2.1: RL schematic shown as the process of an agent interacting with the environment to maximise the future reward [11].

2.1.1 Deep Q Networks

Basic Q-learning consists of an action-value function $Q^\pi(s, a)$, a function which estimates the future total reward of taking action a given the state s. The larger this Q-value, the more likely the policy π is to carry out action a. The aim is to minimise the loss function, which is the difference between the best possible Q-function $Q^*(s, a)$ and the current Q-function found $Q^\pi(s_t, a_t)$.

Basic Q-learning uses an n states by m actions matrix to continually update the policy, this however becomes computationally infeasible for more complex problems. Deep Q Networks (DQN) implements a Deep Neural Network (DNN) to approximate the Q-function [12], by taking in the current state as an input and outputting a set of Q-values for each possible action. The DNN is trained to

minimise the difference between the predicted Q-values and the approximated true Q-values. DQN is a powerful algorithm that combines Q-network learning, fixed Q-targets and experience replays, allowing it to handle high dimensional state spaces.

2.1.2 Proximal Policy Optimisation

Proximal Policy Optimisation (PPO) [13] aims to optimise the policy by maximising the expected reward while keeping the policy updates relatively small. PPO is designed to overcome the pitfalls of previous methods such as Policy Gradient [14] and Trust Region Policy Optimisation (TRPO) [15], which were found to get stuck in local optima and have slow convergence rates. TRPO is also more computationally expensive and time-consuming, because of the limits placed on step sizes.

PPO consists of an *Actor*, which represents the policy mapping states to actions, and a *Critic*, which is the value function that estimates the expected reward for a given state or state-action pair. During the learning process, the policy is initially updated using gradient ascent to maximise the expected reward. Before being *clipped* to stop sizeable deviations from the previous policy, effectively discouraging large changes if it is outside the specified zone.

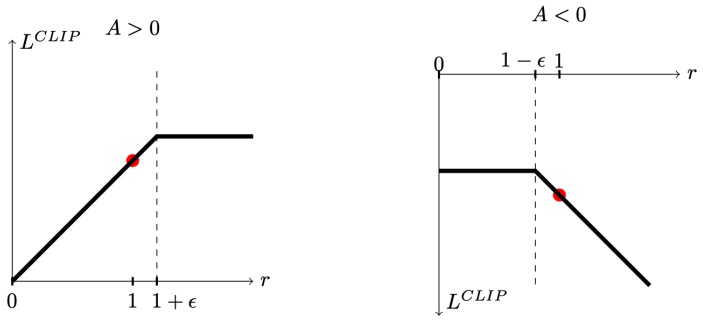

Figure 2.2: Plots showing the loss function being clipped when the ratio between the old policy and new policy (r) is outside of a set range $1 \pm \epsilon$ [13]

2.1.3 Deep Deterministic Policy Gradient

Deep Deterministic Policy Gradient (DDPG) [16] builds on the same Actor-Critic model used in PPO and uses four DNNs:

- Q Network θ^Q
- Policy Network θ^μ
- Target Q Network $\theta^{Q'}$
- Target Policy Network $\theta^{\mu'}$

The θ^Q and θ^μ act as modified Actor and Critic models where the Actor directly maps states to actions instead of outputting the probability of selecting said action. The corresponding target networks for both are time-delayed copies that slowly track the learned networks. Using these target networks improves the stability of the algorithm's learning and reduces the chance of divergence.

DDPG utilises a replay buffer for sampling experiences to update the value and policy networks. The next Q-value is found using the $\theta^{Q'}$ and $\theta^{\mu'}$ to minimise the mean squared error between the new Q-value and the original Q-value. For discrete action space algorithms such as DQN, exploration is done by selecting a random action, but for continuous action spaces, exploration is carried out by adding noise to the selected action.

2.1.4 Comparison

Many studies have compared different RL algorithms for various applications, like the comparison of DQN, PPO, Advantage Actor Critic (A2C), and others in [17]. OpenAI's Gym [18] was chosen

to compare the different algorithms over a set of reproducible scenarios. Results show that as the complexity of the environment increases, PPO has the lowest total training time. A2C algorithms vary drastically with total training time and have high sensitivity to hyperparameter changes.

Larsen et al studied the performance of various RL algorithms for continuous control of ships to navigate challenging waters [19]. A pre-existing simulation environment [20] was used to compare the performances of DDPG, Twin Delayed Deep Deterministic Policy Gradient (TD3) and Soft Actor Critic (SAC) against PPO. They found that DDPG, TD3, and SAC matched the performance of PPO for less difficult environments. As the complexity increases, PPO becomes the best performer overall and produces the most consistent results as seen in Fig. 2.3.

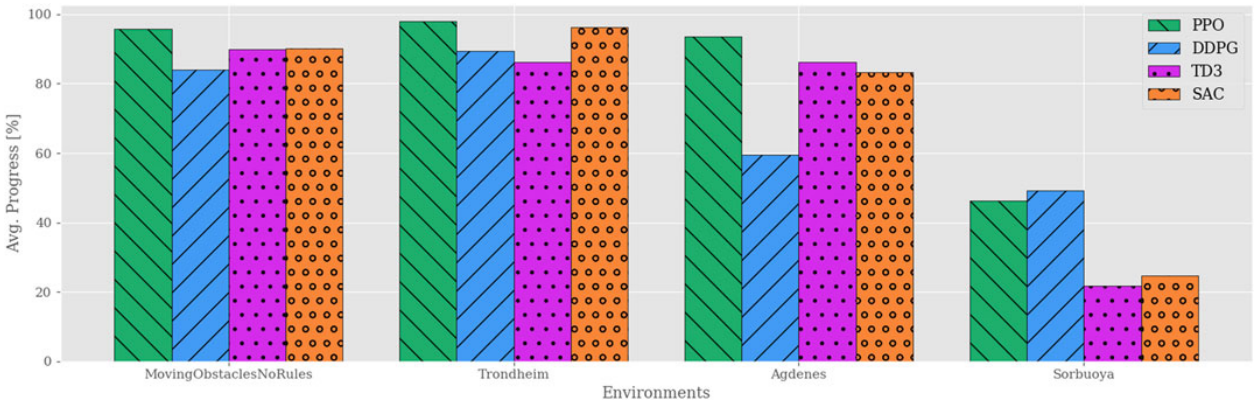

Figure 2.3: Path progression performance comparison between various RL algorithms navigating different shipping routes with varying difficulties [19]. Increasing in difficulty towards the right.

The advantages and disadvantages of the previously discussed algorithms have been compiled below in Table 2.1.

Table 2.1: Comparison of various reinforcement learning methods.

	Advantages	Disadvantages
DQN	Simple to implement	High convergence rate
	Low computational resource use	Only for discrete action spaces
	-	Possible overestimation bias
PPO	Works with continuous action spaces	Requires on-policy learning
	Converges quickly (on average)	More complex implementation than DQN
	Easy hyperparameter tuning	-
DDPG	Works with continuous action spaces	Most complex implementation
	Can use off-policy data	High hyperparameter sensitivity
	-	Possible stability Issues

The implementation complexity and resource use are important factors as the training of the RL models will be conducted on an office laptop. The lack of high computational power could lead to extended training times or process crashes from memory losses. High sensitivity to hyperparameters means more training iterations are needed to find an appropriate combination of hyperparameters, further straining available resources. As the quadcopter actuator outputs are continuous, the signals for DQN would have to be discretised, introducing a discretisation error [21] and reducing the validity of the study. From this analysis, **PPO** is chosen as the algorithm for future experimentation.

2.2 Quadcopter Fault-Tolerant Control

Quadcopters can be subjected to various types of faults such as sensor, actuator, and signal communication faults; which makes FTC an integral aspect of quadcopter control. When a complete loss of one thruster occurs, such as when a propeller blade falls off, the system becomes uncontrollable,

and the control objectives need to be re-defined. Therefore, this study focuses on the partial loss of actuator effectiveness, such as the one occurring following the loss of a portion of a blade.

Current research into quadcopter FTC can be broadly split into two categories, Active Fault-Tolerant Control (AFTC) and Passive Fault-Tolerant Control (PFTC). PFTC relies on the natural robustness of the system to handle faults and requires no knowledge of the specific type of fault. Instead, the system is designed with a focus on in-built redundancy [22].

One study provided a uniform fault-tolerant controller which adapts its control based on the magnitude of the fault, for example, abandoning the yaw channel when a rotor is completely lost [8]. A variety of non-linear strategies such as sliding mode control, adaptive sliding mode control [23], fuzzy control [24] and backstepping control have been proposed. These PFTC methods address the nominal fault-free case as well as faulty cases, but the capability to design a unique control law is limited by the magnitude of the fault. AFTC comprises strategies that actively attempt to prevent the effects of faults. This is achieved by utilising redundant control systems, or controller switching, where a separate integrated system is used to handle faults. A 3-loop non-linear controller was designed and tested on a quadcopter with one rotor removed, inside of a wind tunnel [25]. This research showed that despite severe actuator damage, quadcopters can continue high-speed missions. A method of switching from PFTC to AFTC was also implemented and tested [26].

2.3 Reinforcement Learning in Quadcopter Fault-Tolerant Control

Recent research in quadcopter FTC using RL has shown promising results. One study implemented a Fault-Tolerant Bio-inspired Flight Controller to fly quadcopters and showed success even with two faulty actuators [27].

Another study took a novel hybrid approach by using a weighted blend of both PID and RL for the low-level control [28]. Unlike traditional FTC, one study developed a policy that did not require fault detection and diagnosis, nor tailoring the controller for specific attack scenarios [29]. Instead, the policy was run simultaneously alongside the stabilising controller without the need for on-detection activation. This was designed with various cyber-attacks in mind such as GPS signal spoofing, IMU signal replacement and more.

2.4 Summary

Current research in quadcopter FTC has shown promising results with approaches using RL paired with traditional control schemes. Novel algorithms have been investigated for PFTC, and the state of the art algorithms have been implemented with AFTC. There have been studies regarding injecting faults in models that have only been trained on the non-faulty case [30].

Currently, there is a lack of research for using state-of-the-art RL algorithms in PFTC. This work aims to learn one flight policy to control both the nominal case (no faults) as well as the faulty case. A key advantage of the proposed approach is that the control system does not rely on fault information. Additionally, the control law is static, i.e. it is designed with a unique set of gains, avoiding issues linked to switching control systems. This study will be conducted in a simulated environment, laying the groundwork for future work to implement the model in real-life.

Chapter 3

Research Approach

This chapter outlines the series of steps that will be carried out to achieve the aim of the study and will define some aspects of the research.

3.1 Simulation Environment

Simulation over real-life testing was selected due to several factors:

- **Safety**: Physical experiments can be dangerous and harmful to humans, especially as faults in the quadcopter are being tested. A simulated environment would help avoid these risks.
- **Costs**: There are additional costs to physical testing such as purchasing and maintenance of the quadcopter. The time and cost associated with developing a safe test rig should also be considered.
- **Control**: Testing different parameters or conditions of the experiment is easier in a simulation and allows for faster iteration. The engineer has full control over the entire system and all parameters.
- **Scalability**: Simulated environments can be scaled and modified easily to increase efficiency. Advancements in computer architecture allow for a high level of parallelisation, increasing the overall technical output.
- **Reproducibility**: The experiments can be easily reproduced, allowing for fast verification and validation by other researchers.

Stable Baselines3 [31], an open-source package containing many state-of-the-art algorithms, was used for the implementation of the RL algorithm discussed in Section 2.1. **OpenAI's Gym** [18] was employed as the framework for all of the RL environments.

Several existing quadcopter simulation infrastructures were researched, with some important decision factors being: use in recent research, current maintenance status, code accessibility and resource consumption.

GymFC [32] was one such quadcopter environment, with detailed documentation and many features to aid in the RL of quadcopters. Unfortunately, it is not compatible with OpenAI Gym, instead using ROS and Gazebo [33]. *QuadGym* [34] is based on OpenAI Gym but has not been updated in four years and has very little documentation. *GymRotor* has detailed documentation and uses MuJoCo [35], a 3D physics engine that can render realistic multi-joint models. This is a large package that would be heavily resource intensive, making it suboptimal to be run over unassuming office laptops.

The quadcopter simulation engine used was ***GymCopter*** [36] which has thorough documentation with a versatile code-base that can be modified to fit the needs of this study. This allowed for the addition of features such as fault injection, random wind perturbations and training using stable baselines. It uses realistic multirotor dynamics [37] to ensure accurate and consistent results.

GymCopter comes with four supported environments; Lander2D, Lander3D, Hover2D and Hover3D. These environments can be rendered in real-time to the user as seen in Fig. 3.1. To allow for future work based on this study to transfer the model to real life, all of the physical parameters of the quadcopter have been based on the popular DJI Phantom 4 quadcopter.

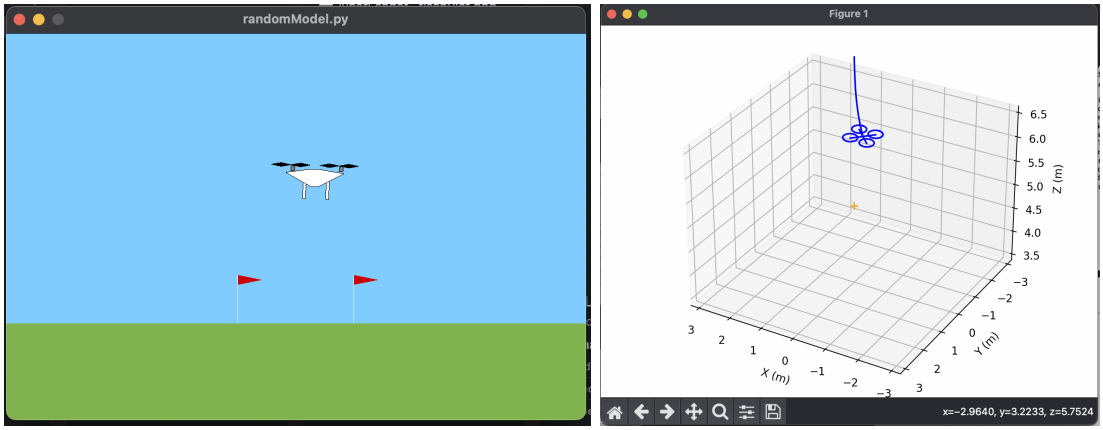

Figure 3.1: Renders of **a)** Lander2D and **b)** Hover3D environments.

3.2 Fault Type

There are a variety of fault types [29] that affect the performance of the quadcopter in different ways. There can be faults in the actuators, sensors or general signal transfer. The following are the main types:

- **Total Failure**: The sensor or motor has broken and the signal is zero.
- **Noisy Signal**: The state signal being detected by the sensor or the signal sent to the motor is noisy and there is a loss of information.
- **Multiplicative**: The signal is being multiplied representing an actuator loss of effectiveness or sensor resonating.

In this study, only the **multiplicative** faults will be investigated as they are the most representative of the loss of effectiveness of a motor. Flight in harsh environments can also lead to sensor scaling or calibration errors which can be modelled as a multiplicative fault. Specifically **actuator** multiplicative faults will be considered, while sensor faults can be considered in future work.

3.3 Summary of Steps

The complexity of the work being conducted will be slowly increased one element at a time, and the steps are summarised in Fig. 3.2.

First, an existing quadcopter environment will be used alongside a state-of-the-art library with RL algorithms already implemented. The base model will be built on top of this, with simulation parameters and state variables defined.

This allows for the development of the reward function which is important as it dictates the general behaviour and performance of the agent. The study will then use the chosen RL algorithm to train a controller that can maintain stable flight in the nominal case (faultless). This will allow for a comparison of the effectiveness of the different reward methods investigated.

Following this, the development of the fault model feature will begin allowing the environment to handle loss in actuator output. This leads to the development of a new RL model that has faults injected throughout the training process, aiming to create one controller that can maintain flight in the nominal case, as well as a faulty case, resulting in PFTC. This final model will be evaluated using various performance criteria against traditional control methods such as a PID controller.

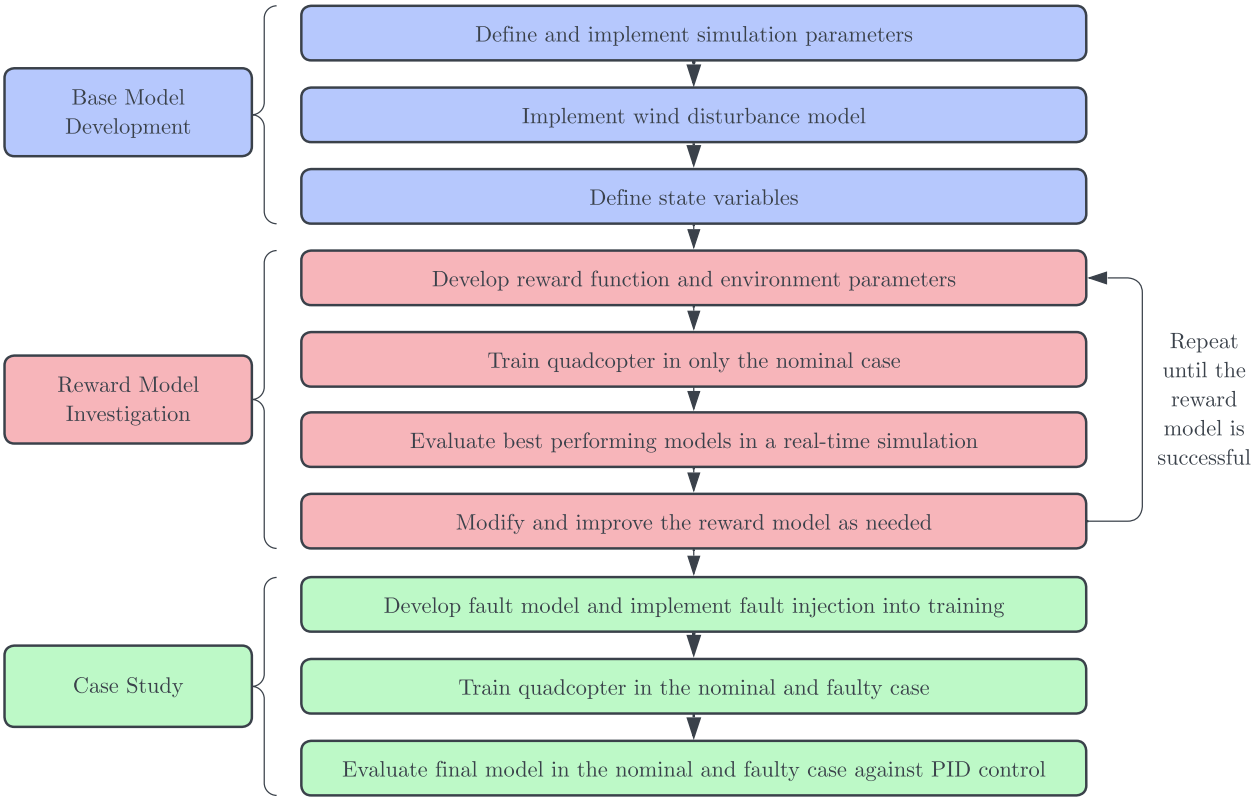

Figure 3.2: Overall Study Plan

Chapter 4

Base Model Development - Faultless

This chapter discusses the development process of the base model, which is trained with no faults. These steps aim to design the optimal environment for quadcopter RL, before introducing faults in Chapter 5 as well as developing a wind disturbance model for a more realistic environment.

To validate the choices made in Section 2.1.4, a series of tests were carried out as detailed in Appendix B. This confirmed that PPO was the correct choice and that due to the impact of randomness, multiple iterations of the training will be conducted for each model.

4.1 Simulation Parameters

This section covers the problem formulation and design of the environment states of the quadcopter.

The existing *GymCopter* environment has no continuous wind disturbance model, and so one has been implemented from scratch as detailed in Appendix A.

The desired behaviour of the quadcopter must be defined and formulated to allow for accurate design of the environment. To show that a model is fault-tolerant, it will be tasked with flying safely from its initialisation point and hovering around a target location for extended periods of time. This behaviour would show that the model can still execute a stable flight to a desired point even when faulty.

4.1.1 State Variables

Dynamic Model States

The dynamics model in *GymCopter* is based on Bouabdallah et al [37] and returns the following observation state space;

$$\boldsymbol{s_{obs}} = \begin{bmatrix} \boldsymbol{\zeta} & \boldsymbol{\eta} & \dot{\boldsymbol{\zeta}} & \dot{\boldsymbol{\eta}} \end{bmatrix}^T \tag{4.1}$$

$$\boldsymbol{\zeta} = \begin{bmatrix} x & y & z \end{bmatrix}^T \qquad \boldsymbol{\eta} = \begin{bmatrix} \phi & \theta & \psi \end{bmatrix}^T. \tag{4.2}$$

Where ζ is the absolute linear position vector in the inertial body frame. The attitude η, is defined in the inertial frame with three Euler angles. The roll angle ϕ, pitch angle θ and yaw angle ψ determine the rotation of the quadcopter around the x, y and z axes respectively. The dot notation represents the time derivative of that variable. This leaves the final state space returned by *GymCopter* to be

$$\boldsymbol{s_{obs}} = \begin{bmatrix} x & \dot{x} & y & \dot{y} & z & \dot{z} & \phi & \dot{\phi} & \theta & \dot{\theta} & \psi & \dot{\psi} \end{bmatrix}^T. \tag{4.3}$$

Reinforcement Learning Model States

To aid the RL agent in reaching the objective, the RL state space $\boldsymbol{s_{RL}} = \boldsymbol{s}$ must be defined. For the quadcopter to learn to fly towards the target, it must have some indication of positional error $\boldsymbol{\zeta_e}$ (position respective to the target). This can be defined as

$$\boldsymbol{\zeta_e} = \boldsymbol{\zeta} - \boldsymbol{\zeta_d} \tag{4.4}$$

where $\boldsymbol{\zeta_d}$ is the target position and in this study is set to

$$\boldsymbol{\zeta_d} = \begin{bmatrix} 0 & 0 & 5 \end{bmatrix}^T. \tag{4.5}$$

The desired behaviour is to fly in a stable manner, keeping the pitch and roll angles bounded; thus the agent will be given a representation of its attitude. This will also guide the agent away from undesirable behaviours such as flying while spinning on its principal axes. Instead of the attitude $\boldsymbol{\eta}$ or its derivatives $\dot{\boldsymbol{\eta}}$ that are returned by $\boldsymbol{s_{OBS}}$, this study will use angular velocities

$$\boldsymbol{\nu} = \begin{bmatrix} p & q & r \end{bmatrix}^T. \tag{4.6}$$

The angular velocities $\boldsymbol{\nu}$ give a better representation of the rotational motion of the quadcopter as they are in the frame of the quadcopter itself, whereas the attitude rates $\dot{\boldsymbol{\eta}}$ indirectly represent this motion through the Euler Angles derivatives. To convert from the attitude rates outputted from $\boldsymbol{s_{OBS}}$, to angular velocities, the following equation [38] can be used,

$$\boldsymbol{\nu} = \boldsymbol{W_\eta}\dot{\boldsymbol{\eta}} \tag{4.7}$$

where $\boldsymbol{W_\eta}$ is the transformation matrix for angular velocities from the inertial frame to the body frame. This equation is thus

$$\begin{bmatrix} p \\ q \\ r \end{bmatrix} = \begin{bmatrix} 1 & 0 & -S\theta \\ 0 & C\phi & C\theta S\phi \\ 0 & -S\phi & C\theta C\phi \end{bmatrix} \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} \tag{4.8}$$

in which $Sx = \sin(x)$ and $Cx = \cos(x)$. The quadcopter will aim to keep the angular velocities to a minimum meaning that the angular velocity error will be

$$\boldsymbol{\nu_e} = \boldsymbol{\nu} - \boldsymbol{\nu_d} \tag{4.9}$$

where $\boldsymbol{\nu}$ is the target angular velocities and is set to

$$\boldsymbol{\nu_d} = \begin{bmatrix} 0 & 0 & 0 \end{bmatrix}^T. \tag{4.10}$$

The final state space used by the RL agent is thus

$$\boldsymbol{s} = \begin{bmatrix} \boldsymbol{\zeta_e} & \boldsymbol{\nu_e} \end{bmatrix}^T = \begin{bmatrix} x_e & y_e & z_e & p_e & q_e & r_e \end{bmatrix}^T. \tag{4.11}$$

Other Considerations

Other state variables such as energy and fuel use have been incorporated in other works [39], but were deemed to be irrelevant to the main objective of solving the FTC problem. Inclusion into the state space would increase the complexity without aiding in the desired behaviour.

4.1.2 Action Space

The action space returned by the RL agent is composed of;

$$\boldsymbol{a} = \boldsymbol{m} = \begin{bmatrix} m_1 & m_2 & m_3 & m_4 \end{bmatrix}^T \tag{4.12}$$

which are the command values for each of the four motors.

4.1.3 Termination and Penalty States

In the RL environment, termination states must be defined to mark the end of an episode. There are four main types of termination:

- Time-based: Termination after a fixed number of timesteps.
- Goal-based: Termination when the agent reaches a certain goal state.
- Error-based: Termination when the agent exceeds a certain error threshold.
- Natural Termination: Termination when the agent crashes or comes to a natural final state.

Termination states are a critical part of designing a successful RL environment as they can heavily impact the agent's ability to learn [40]. This study uses both **time-based** and **natural termination states** to guide the learning process. The episode will terminate when:

- The quadcopter has crashed on the ground i.e. $z \leq 0$.
- The total number of timesteps has exceeded $20,000$, which at $100\,\text{Hz}$ is 2000 seconds.

Depending on the reward model, natural termination may have a penalty p like other works [39], which will adjust how important the agent deems termination avoidance to be. Other factors such as imposing additional penalties/ terminal states for being more than a certain distance away from the target, or for having a large pitch/roll angle were considered, but not implemented. To keep the RL model simple and allow for adequate exploration of the state space.

4.2 Model B.1: Gaussian Curve

All RL agents aim for maximum reward and ideally, the agent would be rewarded after the quadcopter has hovered around the target for the desired time, or would be penalised after crashing [41]. Such a reward function would be extremely sparse and would be an almost impossible task to learn, due to the large state space. To address this, a continuous reward signal r can be designed to guide the agent towards the desired behaviour, increasing the chances of successful training.

The following section formulates the Gaussian Curve-based reward model, which was chosen after a systematic investigation of two models. The analysis of the first model (Model A - negative squared error) is detailed in the appendix (Section C).

4.2.1 Reward Model

Martinsen and Lekkas used a reward-based strategy, consisting of a Gaussian curve with a maximum value of 1 and a standard deviation of σ [42]. Model B.1 is inspired by this function and is adapted as follows:

$$r = \omega_x \boldsymbol{g_\zeta}(x_e) + \omega_y \boldsymbol{g_\zeta}(y_e) + \omega_z \boldsymbol{g_\zeta}(z_e) + \omega_p \boldsymbol{g_\nu}(p_e) + \omega_q \boldsymbol{g_\nu}(q_e) + \omega_r \boldsymbol{g_\nu}(r_e) \tag{4.13}$$

where $\boldsymbol{g_\zeta}$ and $\boldsymbol{g_\nu}$ are functions that transform the positional or angular velocity error to be between 0 and 1, calculated as:

$$\begin{aligned} \boldsymbol{g_\zeta}(a) &= e^{-\frac{a^2}{2\sigma_\zeta}}, \\ \boldsymbol{g_\nu}(a) &= e^{-\frac{a^2}{2\sigma_\nu}}. \end{aligned} \tag{4.14}$$

ω_i indicates the i-th weights and j_e represents the j-th state variable error. The reward coefficients are linked to the desired priority of each variable. For instance, to get the agent to prioritise maintaining position in the z-axis over the other parameters, ω_z can be chosen higher than the other values. The selected values are summarised in Table 4.1.

Table 4.1: Weights used by Model B.1 reward function.

ω_x	ω_y	ω_z	ω_p	ω_q	ω_r
0.1	0.1	0.5	0.1	0.1	0.1

The environment will not have the crash penalty p and additional termination states used in Model A and defined in Eq. C.4, with the aim of simplifying the reward model.

The value of σ in the Gaussian curve is important to determine how sparse/dense the reward will be, which is known to have a significant impact on the performance of a model [43]. Fig. 4.1 shows how different tuning of the σ value affects the reward function: low values of sigma lead to a more accurate controller, as the quadcopter needs to stay closer to the target to receive high reward values.

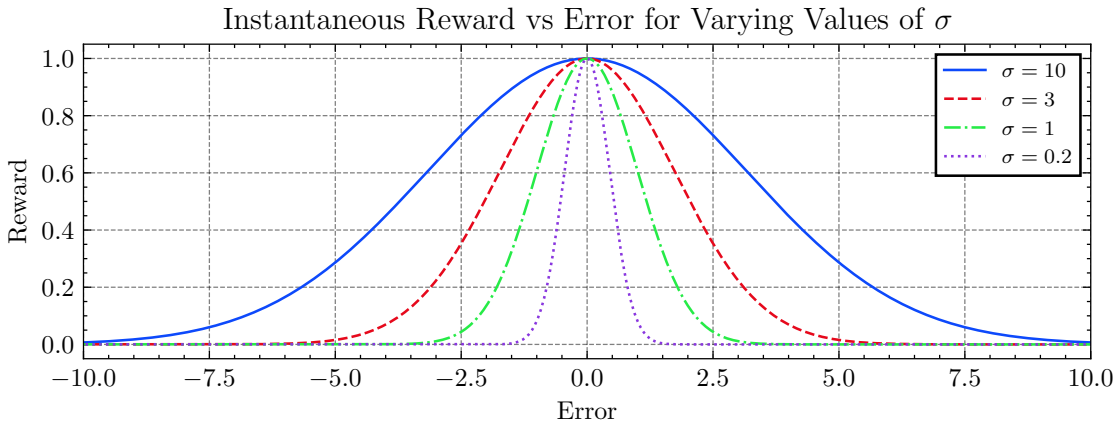

Figure 4.1: Instantaneous reward in Model B.1, where *Error* denotes both Position Error [m] and Angular Velocity Error [rad].

4.2.2 Standard Deviation Analysis

To find suitable values of σ_ζ and σ_ν, a systematic simulation analysis was conducted, with every combination of the following parameter ranges being used for a total of 40 training iterations:

$$(\sigma_\zeta, \sigma_\nu) \mid \sigma_\zeta \in \{0.25, 0.5, 0.75, 1, 1.5, 2, 3, 5, 7, 10\} \text{ and } \sigma_\nu \in \{\frac{\pi}{5}, \frac{\pi}{3}, \frac{\pi}{2}, \pi\} \ . \tag{4.15}$$

For each iteration, the maximum reward value throughout the training was recorded to give a measure of which combination of σ_ζ and σ_ν leads to a trained model with the best performance. The results of this systematic analysis are shown in Fig. 4.2, comparing the maximum reward value of each iteration.

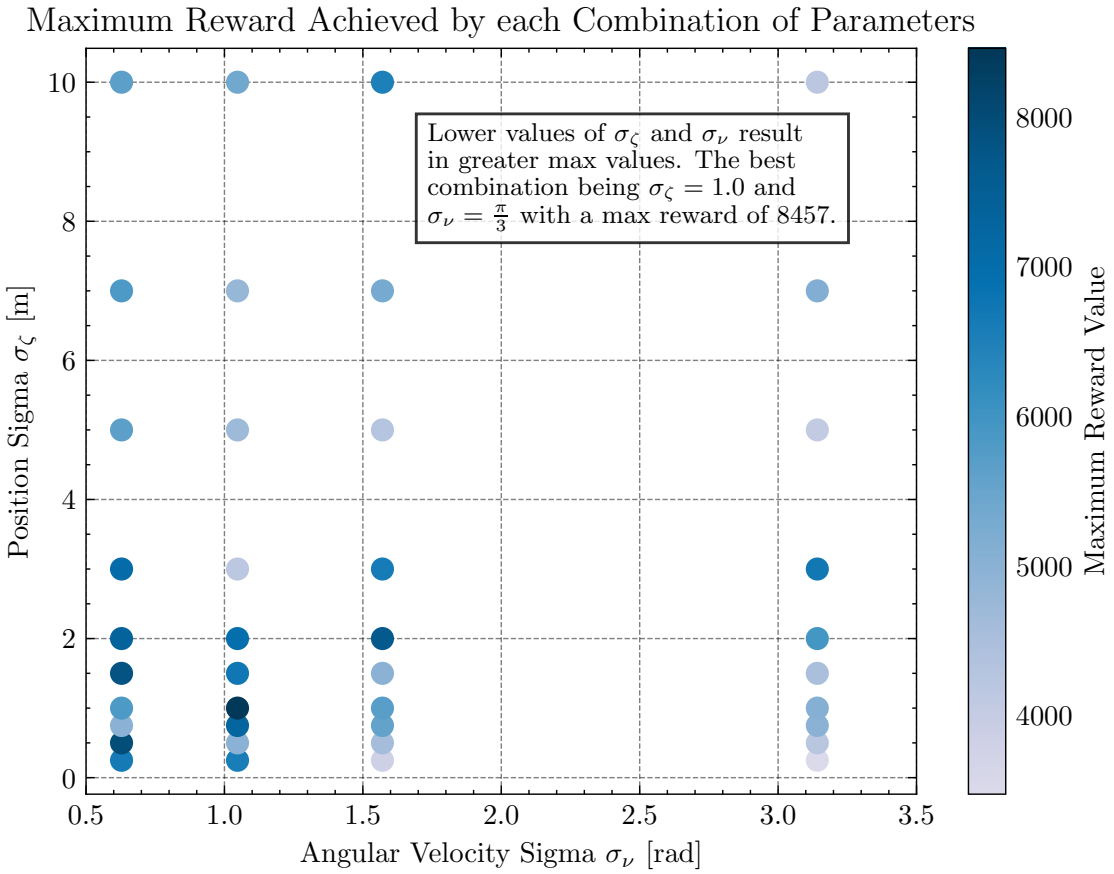

Figure 4.2: Systematic simulation analysis of σ_ζ and σ_ν combinations.

The analysis shows that with higher σ_ζ values (≥ 5), the maximum reward achieved is generally lower, ranging from 4050 to only 6500. This pattern is also prevalent in the σ_ν values, which show that larger values ($\geq \frac{\pi}{2}$), range from 4050 to 7300 in maximum reward. Contrastingly, the following combinations:

$$(\sigma_\zeta, \sigma_\nu) \mid \sigma_\zeta \in \{0.25, 0.5, 0.75, 1, 1.5, 2\} \text{ and } \sigma_\nu \in \{\frac{\pi}{5}, \frac{\pi}{3}\} \ , \tag{4.16}$$

have much higher maximum reward achieved, ranging from 5000, to the best result of 8450. To find which trained agent has the best performance, the best-saved model of each of the top five combinations was simulated in real-time [1], to allow for qualitative and quantitative analysis of each model.

4.2.3 Results

The simulation testing on the previous top five models resulted in the most stable iteration having the following parameters;

$$\sigma_\zeta = 0.25, \quad \sigma_\nu = \frac{\pi}{5}, \tag{4.17}$$

and henceforth this saved model will be referred to as *Model B.1.* Fig. 4.3 illustrates the Position and Euler Angles of Model B.1, for the first 60 seconds of the simulation. Showing that the quadcopter successfully maintains position around the target point, unlike previous models. All of the subsequent

[1]For the sake of brevity, the results of every simulation have been omitted in favour of only the most stable model.

plots show that similar to the PID controller, the closed-loop response of this model is marginally stable around the target position.

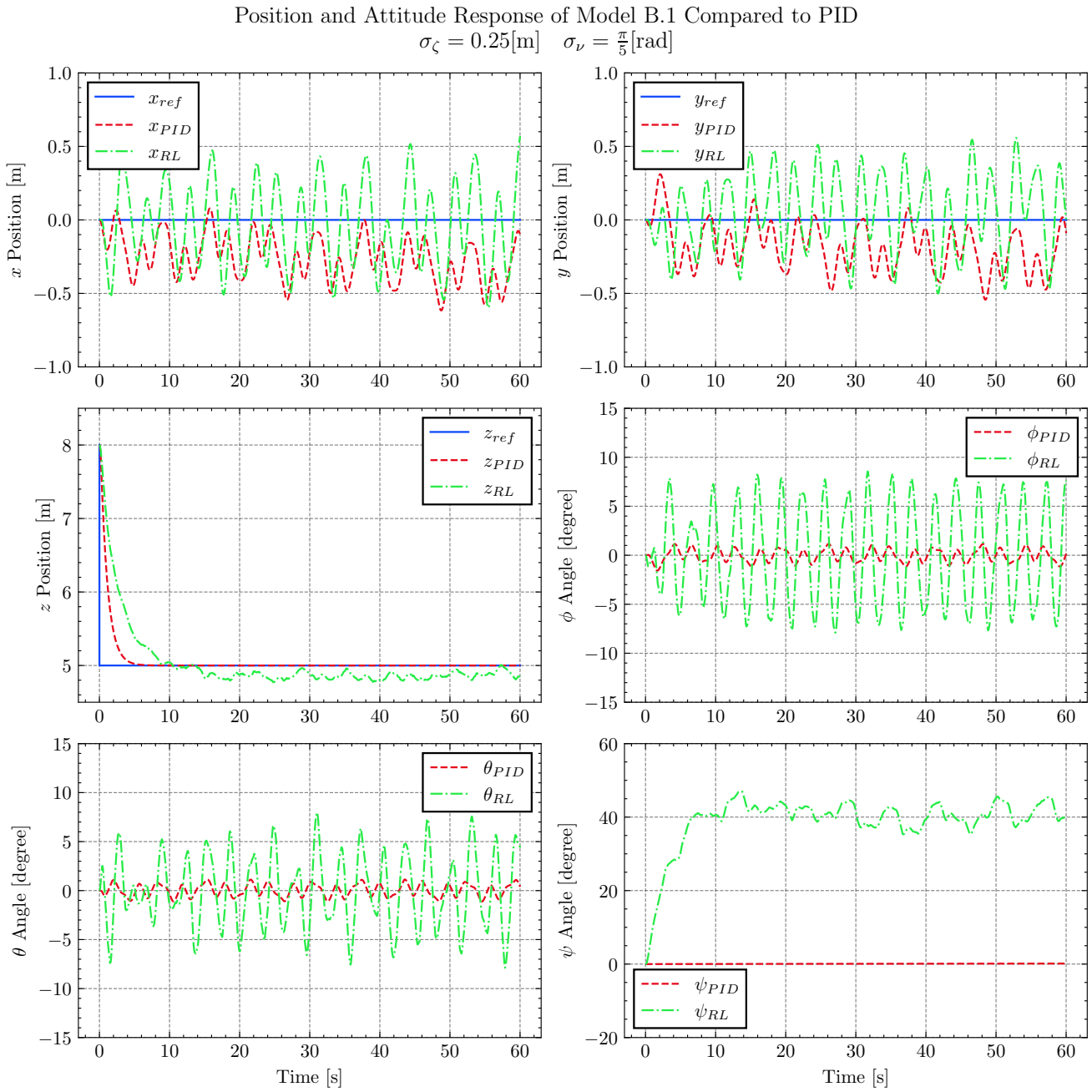

Figure 4.3: Model B.1 real-time simulation performance compared to PID in the nominal case.

The x and y Position plots show that Model B.1 oscillates about the target due to the random wind effects and has a Root Mean Squared Error (RMSE) of ≈ 0.3 m, which is similar in magnitude to the PID controller. The ϕ and θ Attitude plots show that the model has learnt to maintain a pitch ϕ and roll θ angle of 0°, with a range of $\approx 10°$, which is ≈ 10 times larger than the PID law. The z-axis Position plot indicates that the quadcopter learns to descend from the initialisation point $(0, 0, 8)$ to the target $(0, 0, 5)$, before maintaining marginal stability. The settling time $T_{s,B.1}$ within a 5% tolerance band for the quadcopter is 6.63 s which is almost double that of the PID controller 3.13 s. Model B.1 has a Steady State Error (SSE) in the z-axis of ≈ -0.1 and all of the closed-loop response characteristics of both the PID law and Model B.1 are summarised in Table 4.2.

Table 4.2: Table detailing the response characteristics of both the PID law and Model B.1 in the nominal case.

Parameter	Characteristic	Units	Symbol	Model B.1	PID
x Response	RMSE	m	x_{RMSE}	0.28235	0.30494
y Response	RMSE	m	y_{RMSE}	0.26249	0.24848
z Response	RMSE	m	z_{RMSE}	0.14046	0.01412
	Settling Time (5%)	s	T_s	6.63	3.13
	SSE	m	z_{SSE}	−0.11372	0.00264

4.2.4 Conclusions

Section 4.2.2 proved that this model has a high sensitivity to the standard deviation values σ_ζ and σ_ν, meaning that the systematic analysis was necessary to find the best combination of parameter values. The best values were then investigated to find Model B.1, which was successful in maintaining position and hovering around the target point while having similar closed-loop performance characteristics to the PID controller. Unlike the negative squared error model (Section C.1), this model portrays the desired behaviour and so Model B.1 (with $\sigma_\zeta = 0.25$ and $\sigma_\nu = \frac{\pi}{5}$), will be used in Section 5 to investigate faults.

Chapter 5

Case Study - Fault Injection

This chapter investigates faults and their implementation, as well as the limit of robustness for the previous model (B.1). This analysis is used to determine the severity of the faults injected throughout the training process of the new model (B.2). This final model is then evaluated against the traditional PID controller and Model B.1.

5.1 Fault Model

5.1.1 Implementation

To implement the multiplicative actuator faults discussed in Section 3.2, the action space outputted by the RL model (Section 4.1.2) must be modified before being processed by the dynamic model. To simulate the fault, the command values from the action space are pre-multiplied by the fault matrix $\boldsymbol{f}$;

$$\boldsymbol{f} = \begin{bmatrix} f_1 & 0 & 0 & 0 \\ 0 & f_2 & 0 & 0 \\ 0 & 0 & f_3 & 0 \\ 0 & 0 & 0 & f_4 \end{bmatrix} \tag{5.1}$$

where f_i represents the effectiveness of the respective motor, i.e. $f_i = 1$ in the nominal case and $f_i = 0.8$ for a loss of effectiveness of 20%. This leads the final control input $\boldsymbol{u}$ for the quadcopter dynamic model to be

$$\boldsymbol{u} = \begin{bmatrix} f_1 & 0 & 0 & 0 \\ 0 & f_2 & 0 & 0 \\ 0 & 0 & f_3 & 0 \\ 0 & 0 & 0 & f_4 \end{bmatrix} \begin{bmatrix} m_1 \\ m_2 \\ m_3 \\ m_4 \end{bmatrix} = \begin{bmatrix} f_1 m_1 \\ f_2 m_2 \\ f_3 m_3 \\ f_4 m_4 \end{bmatrix}. \tag{5.2}$$

5.1.2 Training

For the RL agent to have a robust control scheme that can fly in both the nominal (faultless) and faulty case, the training process must be modified to allow for different fault cases. Each episode in the training process will be randomly set to one of the fault cases or the nominal case ensuring that the controller will also learn to fly even in the nominal case. This study focuses on only one actuator fault at any given time but the actuator can be any one of the four motors, ensuring a generalised model is developed. This results in the following five equally likely cases for the values of $\boldsymbol{f}$:

$$f_1, f_2, f_3, f_4 = \begin{cases} \text{Case 1} & f_1 = f_2 = f_3 = f_4 = 1, \\ \text{Case 2} & f_1 = \tilde{f} \quad f_2 = f_3 = f_4 = 1, \\ \text{Case 3} & f_2 = \tilde{f} \quad f_1 = f_3 = f_4 = 1, \\ \text{Case 4} & f_3 = \tilde{f} \quad f_1 = f_2 = f_4 = 1, \\ \text{Case 5} & f_4 = \tilde{f} \quad f_1 = f_2 = f_3 = 1, \end{cases} \tag{5.3}$$

$$0 \leq \tilde{f} \leq 1 \tag{5.4}$$

where $\tilde{f}$ represents the fault magnitude for that particular case, i.e. $\tilde{f} = 0.8$ corresponds to 20% loss of effectiveness. This model uses the same reward strategy as Model B.1, but with multiple fault cases and will be henceforth referred to as *Model B.2*.

5.2 Model B.1: Inherent Robustness Limit

To determine the severity of the fault that Model B.2 should be trained on, first Model B.1 must be analysed at varying fault magnitudes, to analyse the decrease in performance. Every control system will have *inherent robustness*, which refers to the tolerance the system has to arbitrarily minor variations in the problem. This means that despite Model B.1 never having encountered a fault before, it may still maintain stable flight around the target point up until an unknown *limit of robustness*. After this, the problem parameters have changed significantly enough to cause instability and crash. Finding the limit of robustness for Model B.1 will thus define at what fault magnitude $\tilde{f}$ Model B.2 should be trained at.

5.2.1 Results

Fig. 5.1 shows the Position and Euler Angles of Model B.1 when injected with varying magnitudes of faults. Looking initially at the results with $\tilde{f} \geq 0.85$, the x, y, ϕ and θ plots show no significant difference in performance, as the responses are all similar. The z Position plot confirms that with $\tilde{f} \geq 0.85$, the quadcopter successfully hovers around the target point and does not crash. However, as you decrease the value of $\tilde{f}$, the steady-state error in the z does increase from -0.1 at $\tilde{f} = 1.00$, to -0.3 and -0.6 at $\tilde{f} = 0.95$ and $\tilde{f} = 0.85$ respectively.

The results at $\tilde{f} = 0.75$ for the x, y, ϕ and θ plots, show oscillations around the target values that grow in amplitude until eventually, the quadcopter crashes. This indicates that a fault of 0.75 causes the system to go just beyond the limit of robustness and become unstable, with the errors growing exponentially until failure. The z Position plot shows that this quadcopter initially descended in a similar manner to the simulations with $\tilde{f} \geq 0.85$. But then continued to drop further, and despite attempting to slow down the fall, eventually loses control and crashes at 19.59 s. Finally, the z plot of $\tilde{f} = 0.65$ shows that this model falls rapidly from the initialisation point before crashing in only 4.40 s.

5.2.2 Conclusions

The results from tests at $\tilde{f} \geq 0.85$ show that 0.85 is before the limit of robustness, as the quadcopter even with those faults injected, managed to successfully hover around the target point. The increasing z_{SSE} for these simulations also indicate that as the fault increases, the quadcopter loses performance in the z axis, hovering further and further away. The results for when $\tilde{f} = 0.75$ suggest that it is just after the limit of robustness, leading to exponentially increasing oscillations causing a crash. The limit of robustness for this model will therefore be assumed to be $\tilde{f} = 0.75$. Model B.2 will aim to improve performance at this fault magnitude to investigate if a model trained on faults, can improve upon the inherent robustness of a model that had never seen faults before.

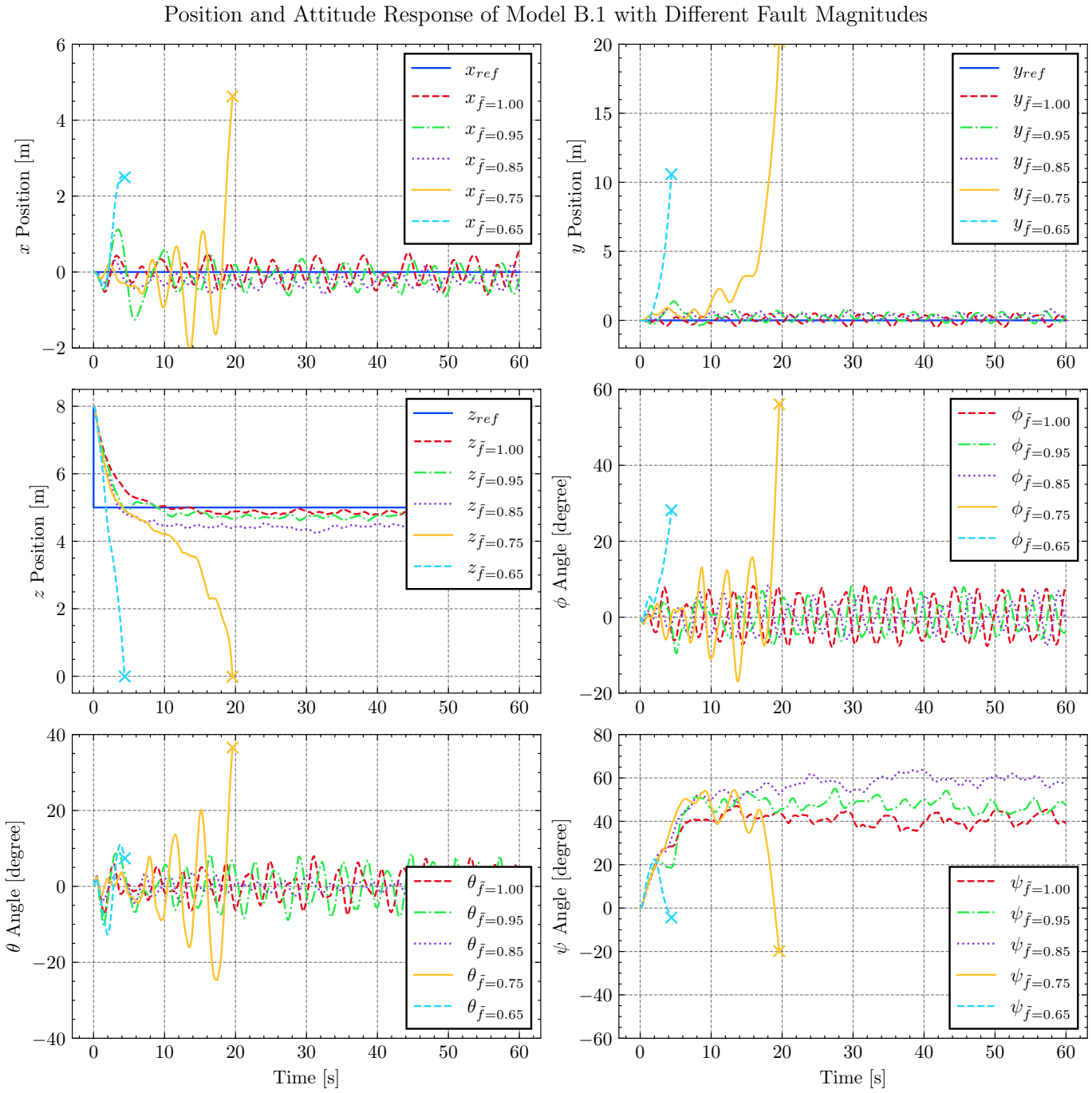

Figure 5.1: Model B.1 real-time simulation performance with different fault magnitudes injected.

5.3 Model B.2: Fault Injected Training

Model B.2 was trained using the different fault cases discussed in Section 5.1.2 with the fault magnitude $\tilde{f}$ set to 0.75 to improve upon the performance of Model B.1 as per the findings in Section 5.2. The training spanned 5,000,000 episodes and the final saved models were tested in real-time simulations and evaluated against Model B.1 and the PID Controller. This comparison includes Model B.1 which was an RL model that was trained with no knowledge of faults, as every episode during training had full effectiveness in all four motors. Thus showing the increase in performance when injecting faults throughout the training processes.

5.3.1 Nominal Case

Results

Fig. 5.2 shows the Position and Euler Angles of Models B.2, B.1 and PID controller in the nominal case, for the first 60 seconds of the simulation. This proves that this new model, trained with faults as well as the nominal case, successfully maintains position around the target, with a marginally stable response.

The x and y Position plots indicate that Model B.2 oscillates around the target point similarly to Model B.1 and the PID controller, suggesting these disturbances are just due to the wind. This finding is reinforced by the ϕ and θ Attitude plots, which show both RL-trained models having similar responses in the attitude while maintaining position. Interestingly, the z axis plot indicates that Model B.2 has $z_{SSE} \approx 0.03\,\text{m}$, a significant improvement to Model B.1, which was oscillating $z \approx 4.89\,\text{m}$.

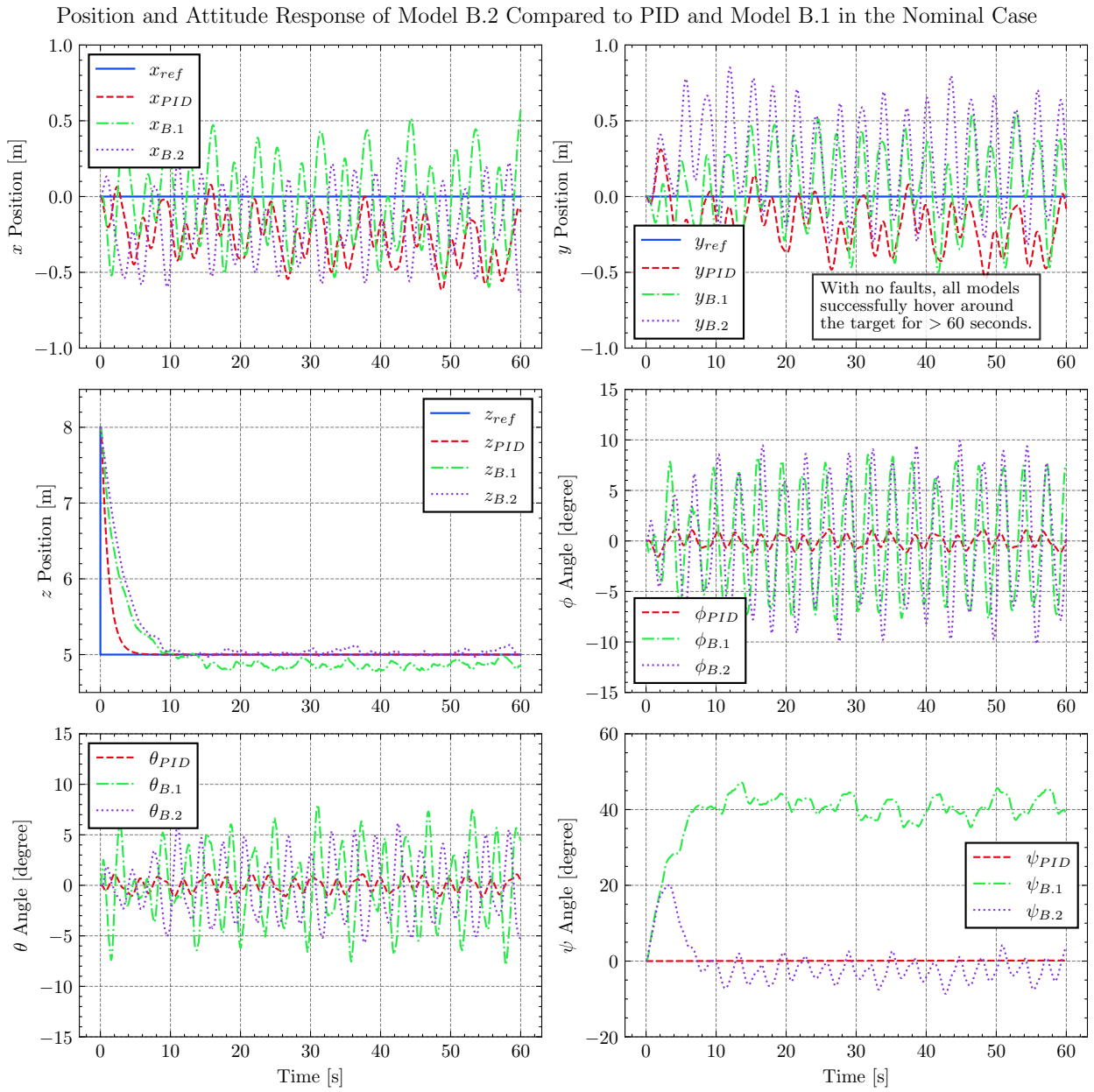

Figure 5.2: Model B.2 real-time simulation performance compared to PID and Model B.1 in the nominal case.

All of the performance characteristics of Model B.2 in the nominal case are listed in Table 5.1. The Model B.1 and PID controller characteristics are the same as Table 4.2.

Table 5.1: Response characteristics of Model B.2 in the nominal case.

Parameter	Characteristic	Units	Symbol	Model B.2
x Response	RMSE	m	x_{RMSE}	0.29404
y Response	RMSE	m	y_{RMSE}	0.39317
z Response	RMSE	m	z_{RMSE}	0.05626
	Settling Time (5%)	s	T_s	7.35
	SSE	m	z_{SSE}	0.03370

Discussion

These results indicate that despite Model B.2 encountering one-fifth of the nominal cases as Model B.1, it has still learnt to hover with full effectiveness in all four motors. This meets one of the goals of this study which was to develop a single controller that can maintain stability in the nominal case.

This model also indicates better performance in the z axis which could be due to Model B.1 overfitting the environment. Overfitting is when an RL model learns the training data too well and may perform poorly when tested. Kang et al suggest that increasing the generalisation of a quadcopter RL model decreases the impact of overfitting and increases performance [44], and in this case, Model B.2 is more generalised as it experiences different fault cases. This could also be due to randomness within the algorithm and in future work, more training runs shall be investigated to determine if Model B.1 could match the performance of Model B.2 in the z axis.

5.3.2 Faulty Case

Results

Fig. 5.3 shows the performance of Model B.1, B.2 and PID law in a real-time simulation when subjected to a fault magnitude of 0.75. This clearly shows both Model B.1 and the PID controller failing within 60 s, albeit in different ways.

The response of Model B.1 is the same as in Section 5.2.1, where it fails around 20 s in. The PID controller is seen to fail after 35.34 s and the x and y Position plots show the immediate increase in error across both of these axes, as the quadcopter flies away from the desired point. These errors are extremely large, with the quadcopter at certain points being 65 m away from the desired position. The ϕ and θ plots show that this model is unstable, as it slowly oscillates about its pitch and roll axis, attempting to correct itself until the eventual failure. Interestingly, the z axis plot shows that the quadcopter maintains altitude perfectly, just as in the nominal case, until ≈ 34.34 s, when it drops rapidly and crashes in the next second.

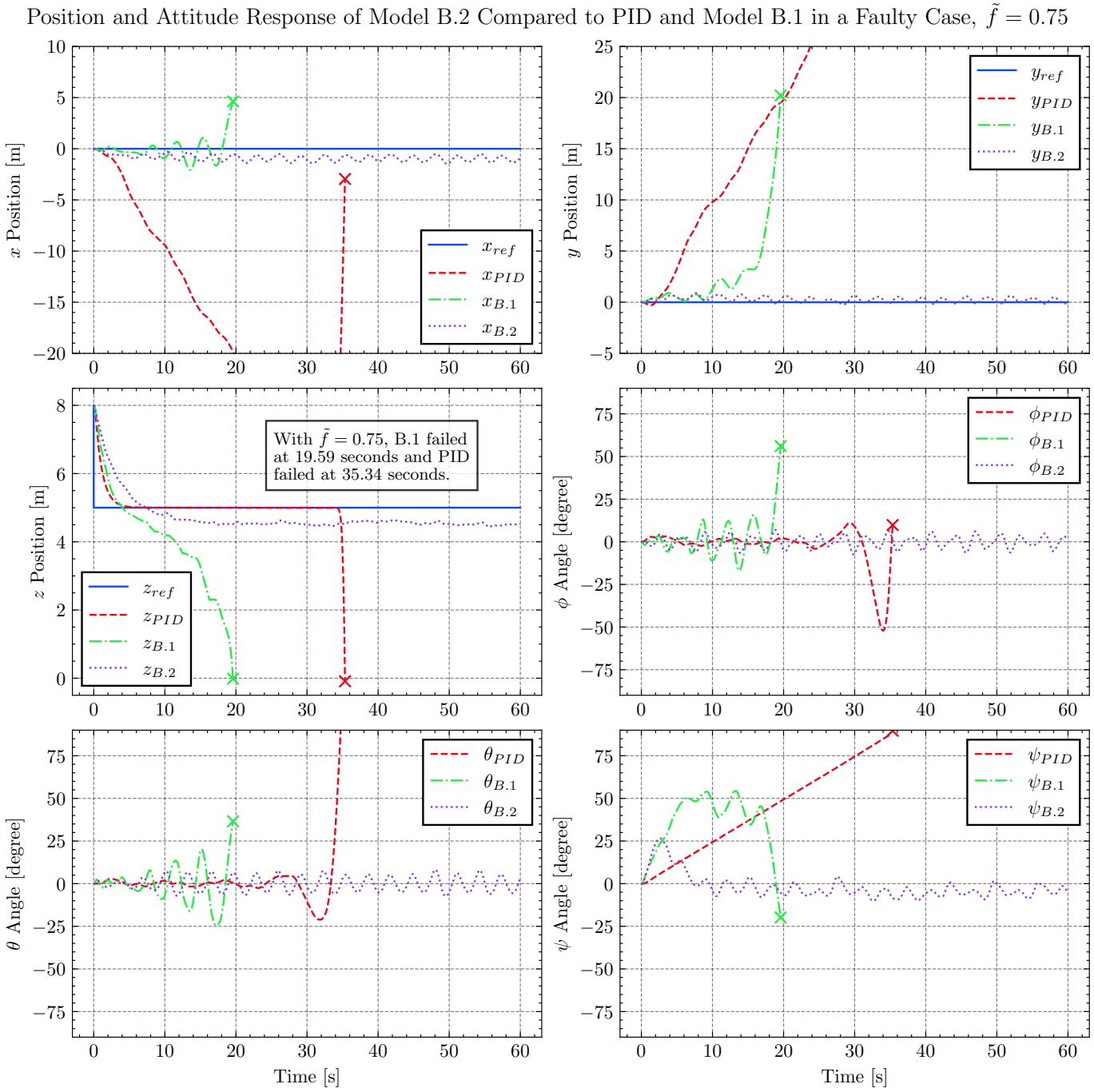

Figure 5.3: Model B.2 real-time simulation performance compared to PID and Model B.1 in the faulty case.

Model B.2 is seen to successfully complete the objective by hovering around the target point for $> 60\,\text{s}$. The x and y Position plots show the quadcopter hovering with an $x_{RMSE} \approx 1.0\,\text{m}$, which is approximately three times as large as the nominal case. The ϕ and θ Attitude plots show similar oscillations and ranges as the nominal case ($\approx 7°$). Inspecting the z plot shows that the quadcopter settles at a final altitude of $z = 4.5\,\text{m}$ and takes $11.05\,\text{s}$ to get within 5% of that final value. All of the performance characteristics are listed in Table 5.2.

Table 5.2: Response characteristics of Model B.2 in the faulty case.

Parameter	Characteristic	Units	Symbol	Model B.2
x Response	RMSE	m	x_{RMSE}	0.93487
y Response	RMSE	m	y_{RMSE}	0.35912
z Response	RMSE	m	z_{RMSE}	0.45631
	Settling Time (5%)	s	T_s	11.05
	SSE	m	z_{SSE}	−0.45349

Discussion

Assuming the PID controller to be a closed-loop control system allows for the poles of the system to be considered, which determine the system's response to disturbances. Injecting a fault into the quadcopter leads to the outputs of the system changing as they are now being multiplied by the fault matrix $\boldsymbol{f}$. This changes the transfer function of the quadcopter plant, but as the PID controller was designed and tuned for the original transfer function, the model is no longer representative of the true system. The fault would decrease the output of the quadcopter, meaning a decrease in the system gain, which will move the poles of the system on the s-plane into the positive section of the real axis. The results show that the quadcopter's dynamics increase exponentially before eventually crashing, meaning the poles have crossed the imaginary axis of the s-plane and thus became unstable. These issues originate from the model-based nature of a strategy such as PID, which must have accurate knowledge of the system's dynamics to perform.

Model B.2's performance is evidence that the model has achieved the objective of this study, as this one control scheme has shown stability in both the nominal and faulty case, with no active control switching. Comparing the PID law and Model B.2 shows that traditional control strategies lead to failure whereas the RL-trained model has learnt to deal with fault injection.

5.3.3 Varying Fault Magnitudes

To determine how robust Model B.2 is the quadcopter will be injected with varying magnitudes of faults and simulated in real time. This study will determine how much inherent robustness is in this model, as well as how the model performs with more severe faults.

Results

Fig. 5.4 shows the Position and Attitude response of Model B.2 when injected with various fault magnitudes;

$$\tilde{f} \in \{0.75, 0.70, 0.60, 0.25, 0.00\} \ . \tag{5.5}$$

The results for $\tilde{f} = 0.70$ show that the quadcopter successfully hovers around the target point for the targeted time and performs similarly to the response at $\tilde{f} = 0.75$. The x, y, ϕ and θ plots for $\tilde{f} = 0.60$ all show the quadcopter initially maintaining position and attitude around the target for the first 13 s before losing control and crashing. This is reinforced when looking at the z Position plot which shows a faster descent than the two successful runs, and it seems that the quadcopter falls too far ($z \approx 3.5$) before being unable to control itself and crashing at 15 s. At $\tilde{f} \leq 0.25$, all of the position and Euler Angle plots display rapidly increasing errors, as the quadcopter falls immediately out of the air with no ability to control itself and crashes in 3.58 s.

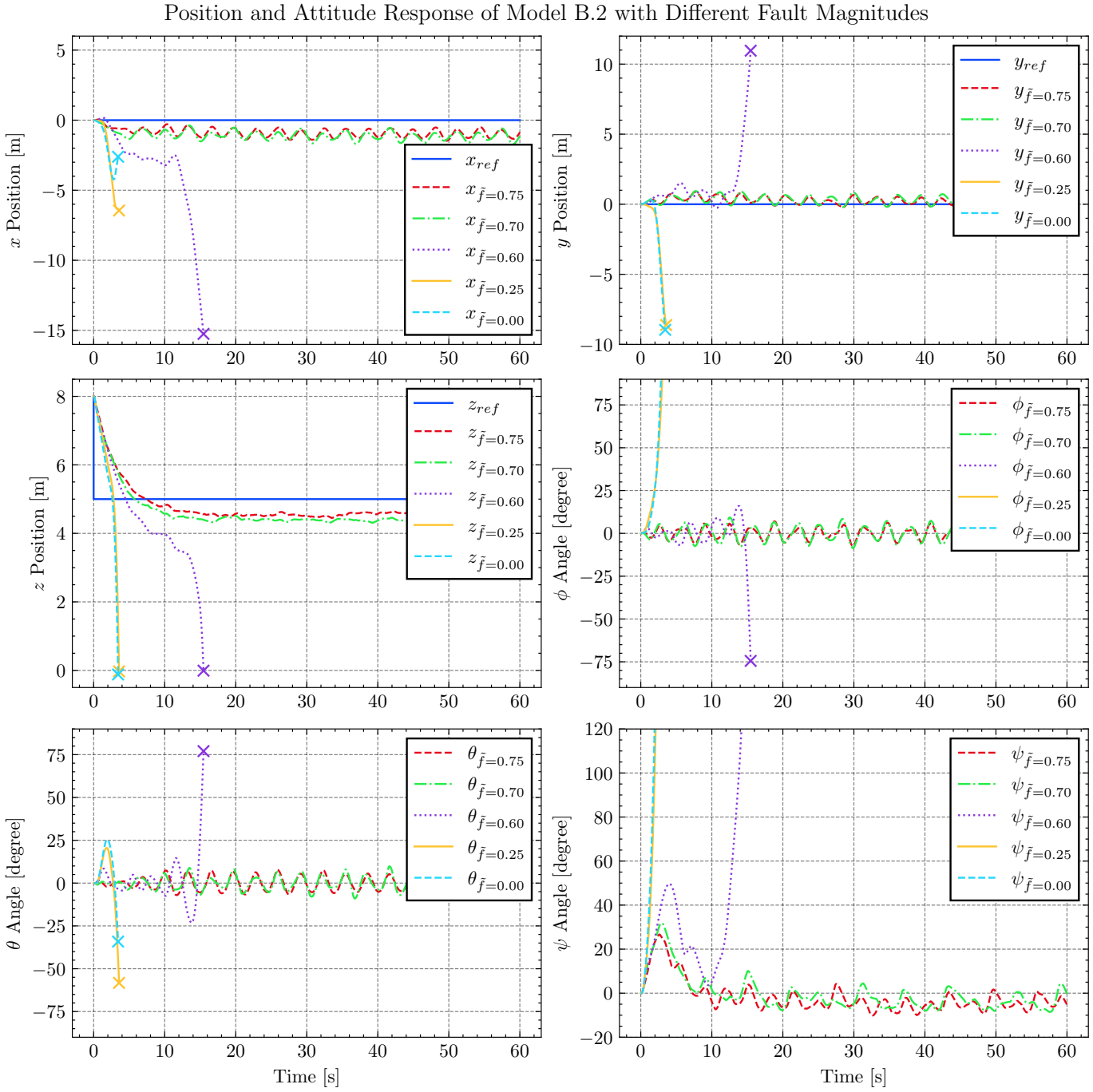

Figure 5.4: Model B.2 real-time simulation performance with various fault magnitudes.

Discussion

Despite Model B.2 being trained with $\tilde{f} = 0.75$, the results show it has inherent robustness as it can still maintain stability at $\tilde{f} = 0.70$, meaning that it is the upper limit of the inherent robustness limit. The quadcopter with $\tilde{f} = 0.60$ has some control before crashing, meaning that this is the lower bound for the inherent robustness limit.

Overall this means that Model B.2 is generalised in nature as it can successfully control a quadcopter when any one of its four motors has a fault magnitude ranging from $0.70 \leq \tilde{f}$ and in the nominal case with all four motors being fully functional.

Chapter 6

Evaluation

Model B.2 was developed which was one control policy that can control the simulated quadcopter in both the nominal case, as well as the faulty case, with no control switching. Thus achieving the aim from Section 1.3.

6.1 Strengths

A major strength of this study is the **depth in which the various reward strategies were investigated**. Sections C.1, C.2 and 4.2 investigated different reward strategies, with extensive testing on each. The results were analysed and used as the bases for the following Case Study (Section 5), leading to high confidence that the final model used (Model B.2), was the optimal choice.

Another achievement is the **generalised design of the final model**. This work added non-repeating, continuous wind disturbances (Section A) to the quadcopter environment, proving the final resulting model can still perform as intended even with realistic disturbances. Additionally, the fault injection during training consisted of an equal chance between any of the four motors on the quadcopter, as described in Section 5.1.2. This allows for the quadcopter to successfully hover around the target position with the fault being injected in any of the four motors, making the model more effective. Finally, this study chooses to focus on PTFC instead of AFTC, removing the need for additional fault identification systems or control switching schemes, as discussed in Section 2.4.

6.2 Weaknesses

A major issue that slowed the study's progress was the **existence of bugs in the code** written over the entire project. A simple typo in the Stable-Baselines3 integration code incorrect simulations to be shown, and consequently lead to weeks of debugging. This type of error is unfortunately a part of the engineering process but may have been more likely due to a lack of experience with the different technologies. To prevent this type of issue in the future, a well-devised plan for each script should be made, with more comments and debug outputs written.

The **sequential nature of the training process** was another weakness of the work done. The training of all the models in this study was conducted on an office laptop, sequentially on a CPU, limiting the training times and types of RL algorithms used. Parallelisation could have overcome the long training times detailed in Sections 2.1.4 and B.2, allowing for a more thorough investigation. This would allow for training to be conducted concurrently, reducing the overall time for the same amount of experiments, or allowing for more experimentation in the same amount of time.

Another issue encountered was the multiple **assumptions and unclear code in the existing quadcopter simulation software** (GymCopter). There were substantial delays in the project due to bugs/assumptions in the existing code-base. For example, the quadcopter dynamics processing script assumed small angles of pitch and roll to speed up calculations, which is only valid for $\lesssim 10°$. These errors can be avoided in the future by either using software that is better documented, with assumptions and ongoing bugs listed clearly or by conducting a thorough investigation into the code-base before using it, to uncover such bugs/assumptions.

Chapter 7

Conclusion

7.1 Summary

In this study, RL techniques were applied to the flight control of a quadcopter both in the nominal and faulty case. Specifically, traditional and RL strategies were evaluated for the task of hovering around a target point without crashing, when one of the motors in the quadcopter loses partial effectiveness and with wind disturbances present.

The RL algorithm used in this study was PPO, which utilises the actor-critic deep neural network architecture as well as a *clipping* function that stops the policy updates from being too large. Two different reward strategies were investigated, with one using the negative squared error value as the basis of the reward (Section C.1) and the other using a Gaussian curve on the error value (Section 4.2). These analyses consisted of multiple training runs with varying hyperparameters and modified environments. The runs with the highest rewards were then simulated in real-time to assess the Position and Euler angles, better indicating that particular model's performance, with the Gaussian curve model resulting in the best end result. Multiplicative actuator faults were then defined and implemented into the environment (Section 5.1), before training a new model with faults injected throughout the training process. This final model (Model B.2) was then evaluated against previous models and a PID controller in both the nominal and faulty cases, to determine the performances with varying fault cases and magnitudes.

Model B.2 successfully hovered around the target position in both the nominal case (all motors with full effectiveness) and the faulty cases (one of the four motors has a loss of effectiveness). The PID controller's performance showed how it crashed in $\approx 30\,\mathrm{s}$ and was drifting away in the x and y the entire time. Model B.2 met the objective and showed PFTC of a quadcopter with a multiplicative actuator fault. This was achieved with the Gaussian Curve-based reward strategy for the RL training, with small σ_ζ and σ_ν values leading to a dense reward distribution. This work proves that RL can be used to develop a safe and consistent control scheme for quadcopter FTC concerning multiplicative actuator faults.

7.2 Implications

The developed control scheme can maintain stable flight in the presence of motor faults, which was shown to be unachievable with existing traditional control techniques like PID. This is a significant achievement for industries which rely on quadcopters, such as aerial mapping, inspection or search and rescue operations. This study shows that after more development of this model and implementation into a physical quadcopter, the **RL-trained model could be used in place of the current model-based controllers to improve safety and reliability**.

This study also has significant implications for the academic community as this work could serve as a baseline for evaluating future RL algorithms or testing this model against other traditional control schemes like Sliding Mode Control (SMC). This work could be **extended in academia to other complex control problems requiring FTC** like robotics [45] or power distribution systems [46].

7.3 Limitations

This study has some limitations such as the **wind disturbance model** which is based on a non-periodic, continuous function for the wind's x, y and z components. Instead, more complex

and realistic wind models could be used such as the Dryden spectral representation [47] or the Von-Karman Wind Turbulence Model [48]. The simulation **fails to account for processing time and resources required** by the RL model at each timestep, leading to slight inaccuracies as real-time quadcopter dynamics evolve concurrently with data processing. Finally, the model assumes an **infinite power supply**, simplifying the RL process but neglecting real-world considerations such as power-intensive manoeuvres and battery usage.

7.4 Future Work

Future studies could focus on **implementing the final RL model into a physical quadcopter** and evaluate the performance of the model, as well as the validity of the simulated environment used in this study. **Further investigation should be conducted on aspects of the environment and dynamics that were found to be unrealistic** such as computational power, sensor ranges or noise.

Other types of faults discussed in Section 3.2 could be investigated using this study as a base framework. More complex disturbance rejection could be explored with more realistic wind and weather modelling. This study establishes that RL algorithms offer fault tolerance without a model, though the policy, encoded in a neural network, behaves as a *black box*, making its decisions unpredictable. Future research could explore **Explainable AI** [49] to identify potential issues, enhancing quadcopter control system safety and reliability.

Bibliography

1. Ononiwu G, Okoye A, Onojo J, and Onuekwusi N. Design and implementation of a real time wireless quadcopter for rescue operations. American Journal of Engineering Research 2016; 5:130–8

2. Boehm D, Chen A, Chung N, Malik R, Model B, and Kantesaria P. Designing an unmanned aerial vehicle (UAV) for humanitarian aid. Tech. rep. Technical report, Rutgers School of Engineering, 2017

3. Praveen V and Pillai S. Modeling and simulation of quadcopter using PID controller. International Journal of Control Theory and Applications 2016; 9:7151–8

4. Maleki KN, Ashenayi K, Hook LR, Fuller JG, and Hutchins N. A reliable system design for nondeterministic adaptive controllers in small UAV autopilots. *2016 IEEE/AIAA 35th Digital Avionics Systems Conference (DASC)*. IEEE. 2016 :1–5

5. Blanke M, Christian Frei W, Kraus F, Ron Patton J, and Staroswiecki M. What is Fault-Tolerant Control? IFAC Proceedings Volumes 2000; 33:41–52. DOI: 10.1016/s1474-6670(17)37338-x

6. Sun S, Cioffi G, De Visser C, and Scaramuzza D. Autonomous quadrotor flight despite rotor failure with onboard vision sensors: Frames vs. events. IEEE Robotics and Automation Letters 2021; 6:580–7

7. Silver D, Huang A, Maddison CJ, Guez A, and Sifre L. Mastering the game of Go with deep neural networks and tree search. Nature 2016; 529:484–9

8. Ke C, Cai KY, and Quan Q. Uniform Fault-Tolerant Control of a Quadcopter With Rotor Failure. IEEE/ASME Transactions on Mechatronics 2022

9. Kargar S, Salahshoor K, and Yazdanpanah M. Integrated nonlinear model predictive fault tolerant control and multiple model based fault detection and diagnosis. Chemical Engineering Research and Design 2014; 92:340–9

10. Sutton RS and Barto AG. Reinforcement Learning: An Introduction. Second. The MIT Press, 2018

11. Galatzer-Levy I, Ruggles K, and Chen Z. Data Science in the Research Domain Criteria Era: Relevance of Machine Learning to the Study of Stress Pathology, Recovery, and Resilience. Chronic Stress 2018 Jan; 2:247054701774755. DOI: 10.1177/2470547017747553

12. Mnih V, Kavukcuoglu K, Silver D, Graves A, Antonoglou I, Wierstra D, and Riedmiller M. Playing Atari with Deep Reinforcement Learning. 2013. DOI: 10.48550/ARXIV.1312.5602

13. Schulman J, Wolski F, Dhariwal P, Radford A, and Klimov O. Proximal Policy Optimization Algorithms. 2017. DOI: 10.48550/ARXIV.1707.06347

14. Mnih V, Badia AP, Mirza M, Graves A, Lillicrap T, Harley T, Silver D, and Kavukcuoglu K. Asynchronous methods for deep reinforcement learning. *International conference on machine learning*. PMLR. 2016 :1928–37

15. Schulman J, Levine S, Moritz P, Jordan MI, and Abbeel P. Trust Region Policy Optimization (TRPO). CoRR abs/1502.05477 2015

16. Lillicrap TP, Hunt JJ, Pritzel A, Heess N, Erez T, Tassa Y, Silver D, and Wierstra D. Continuous control with deep reinforcement learning. 2015. DOI: 10.48550/ARXIV.1509.02971

17. Krishna V and Yarram S. Course: CSE 546: Introduction to Reinforcement Learning COMPARISON OF REINFORCEMENT LEARNING ALGORITHMS. 2020

18. Brockman G, Cheung V, Pettersson L, Schneider J, Schulman J, Tang J, and Zaremba W. Openai gym. arXiv preprint arXiv:1606.01540 2016

19. Larsen TN, Teigen HØ, Laache T, Varagnolo D, and Rasheed A. Comparing Deep Reinforcement Learning Algorithms' Ability to Safely Navigate Challenging Waters. Frontiers in Robotics and AI 2021; 8. DOI: 10.3389/frobt.2021.738113

20. Meyer E. On Course Towards Model-Free Guidance: A Self-Learning Approach To Dynamic Collision Avoidance for Autonomous Surface Vehicles. Ntnu.no 2020. DOI: no.ntnu:inspera:56990118:18201685

21. Sinclair S, Wang T, Jain G, Banerjee S, and Yu C. Adaptive discretization for model-based reinforcement learning. Advances in Neural Information Processing Systems 2020; 33:3858–71

22. Gao Z, Cecati C, and Ding SX. A Survey of Fault Diagnosis and Fault-Tolerant Techniques—Part I: Fault Diagnosis With Model-Based and Signal-Based Approaches. IEEE Transactions on Industrial Electronics 2015; 62:3757–67. DOI: 10.1109/TIE.2015.2417501

23. Nguyen NP and Hong SK. Fault-Tolerant Control of Quadcopter UAVs Using Robust Adaptive Sliding Mode Approach. Energies 2018; 12. DOI: 10.3390/en12010095

24. Unal G. Integrated design of fault-tolerant control for flight control systems using observer and fuzzy logic. Aircraft Engineering and Aerospace Technology 2021

25. Sun S, Sijbers L, Wang X, and Visser C de. High-Speed Flight of Quadrotor Despite Loss of Single Rotor. IEEE Robotics and Automation Letters 2018; 3:3201–7. DOI: 10.1109/LRA.2018.2851028

26. Nguyen NP and Hong SK. Active Fault-Tolerant Control of a Quadcopter against Time-Varying Actuator Faults and Saturations Using Sliding Mode Backstepping Approach. Applied Sciences 2019; 9. DOI: 10.3390/app9194010

27. Dooraki AR and Lee DJ. Reinforcement learning based flight controller capable of controlling a quadcopter with four, three and two working motors. *2020 20th International Conference on Control, Automation and Systems (ICCAS).* 2020 :161–6. DOI: 10.23919/ICCAS50221.2020.9268270

28. Sohège Y, Quiñones-Grueiro M, and Provan G. A Novel Hybrid Approach for Fault-Tolerant Control of UAVs based on Robust Reinforcement Learning. *2021 IEEE International Conference on Robotics and Automation (ICRA).* 2021 :10719–25. DOI: 10.1109/ICRA48506.2021.9562097

29. Fei F, Tu Z, Xu D, and Deng X. Learn-to-Recover: Retrofitting UAVs with Reinforcement Learning-Assisted Flight Control Under Cyber-Physical Attacks. *2020 IEEE International Conference on Robotics and Automation (ICRA).* 2020 :7358–64. DOI: 10.1109/ICRA40945.2020.9196611

30. Bernini N, Bessa M, Delmas R, Gold A, Goubault E, Pennec R, Putot S, and Sillion F. A few lessons learned in reinforcement learning for quadcopter attitude control. *Proceedings of the 24th International Conference on Hybrid Systems: Computation and Control.* 2021 :1–11

31. Raffin A, Hill A, Gleave A, Kanervisto A, Ernestus M, and Dormann N. Stable-Baselines3: Reliable Reinforcement Learning Implementations. Journal of Machine Learning Research 2021; 22:1–8. Available from: http://jmlr.org/papers/v22/20-1364.html

32. Koch W, Mancuso R, West R, and Bestavros A. Reinforcement learning for UAV attitude control. ACM Transactions on Cyber-Physical Systems 2019; 3:22

33. Koenig N and Howard A. Design and use paradigms for Gazebo, an open-source multi-robot simulator. 2004 Apr :2149–2154 vol.3. DOI: 10.1109/IROS.2004.1389727

34. ngc92. quadgym: OpenAI gym Environments for Quadrotor Control. 2018. Available from: https://github.com/ngc92/quadgym

35. Todorov E, Erez T, and Tassa Y. MuJoCo: A physics engine for model-based control. *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2012 :5026–33. DOI: 10.1109/IROS.2012.6386109

36. Levy S. A Simple Platform for Reinforcement Learning of Simulated Flight Behaviors. 2021

37. Bouabdallah S, Murrieri P, and Siegwart R. Design and control of an indoor micro quadrotor. IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA '04. 2004 2004. DOI: 10.1109/robot.2004.1302409

38. Luukkonen T. Modelling and control of quadcopter. Independent research project in applied mathematics, Espoo 2011; 22

39. Anderlini E, Parker GG, and Thomas G. Docking Control of an Autonomous Underwater Vehicle Using Reinforcement Learning. Applied Sciences 2019; 9. DOI: 10.3390/app9173456. Available from: https://www.mdpi.com/2076-3417/9/17/3456

40. Reda D, Tao T, and Panne M van de. Learning to Locomote: Understanding how environment design matters for deep reinforcement learning. *Proceedings of the 13th ACM SIGGRAPH Conference on Motion, Interaction and Games*. 2020 :1–10

41. Meyer E, Robinson H, Rasheed A, and San O. Taming an Autonomous Surface Vehicle for Path Following and Collision Avoidance Using Deep Reinforcement Learning. IEEE Access 2020; 8:41466–81. DOI: https://doi.org/10.1109/access.2020.2976586

42. Martinsen AB and Lekkas AM. Straight-Path Following for Underactuated Marine Vessels using Deep Reinforcement Learning. IFAC-PapersOnLine 2018; 51:329–34. DOI: https://doi.org/10.1016/j.ifacol.2018.09.502

43. Memarian F, Goo W, Lioutikov R, Niekum S, and Topcu U. Self-supervised online reward shaping in sparse-reward environments. *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2021 :2369–75

44. Kang K, Belkhale S, Kahn G, Abbeel P, and Levine S. Generalization through simulation: Integrating simulated and real data into deep reinforcement learning for vision-based autonomous flight. *2019 international conference on robotics and automation (ICRA)*. IEEE. 2019 :6008–14

45. Chen G, Song Y, and Lewis FL. Distributed fault-tolerant control of networked uncertain Euler–Lagrange systems under actuator faults. IEEE transactions on cybernetics 2016; 47:1706–18

46. Su X, Liu X, and Song YD. Fault-tolerant control of multiarea power systems via a sliding-mode observer technique. IEEE/ASME Transactions on Mechatronics 2017; 23:38–47

47. Liepmann HW. On the application of statistical concepts to the buffeting problem. Journal of the Aeronautical Sciences 1952; 19:793–800

48. Diederich FW and Drischler JA. Effect of spanwise variations in gust intensity on the lift due to atmospheric turbulence. Tech. rep. 1957

49. Adadi A and Berrada M. Peeking inside the black-box: a survey on explainable artificial intelligence (XAI). IEEE access 2018; 6:52138–60

50. Lowe R and Ziemke T. Exploring the relationship of reward and punishment in reinforcement learning. *2013 IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning (ADPRL)*. IEEE. 2013 :140–7

51. Elfwing S and Seymour B. Parallel reward and punishment control in humans and robots: Safe reinforcement learning using the MaxPain algorithm. *2017 Joint IEEE International Conference on Development and Learning and Epigenetic Robotics (ICDL-EpiRob)*. IEEE. 2017 :140–7

Appendix A

Wind Disturbance Model

As *GymCopter* had no wind disturbances implemented in the environment, this study developed a simple continuous model for wind perturbations to be constantly applied throughout the simulation. The implementation was inspired by the wind model used in OpenAI's Lunar Lander environment [18]. The following non-periodic function is thus used to generate a random force value;

$$W_i = w_f \tanh(\sin(2w_k(t+C)) + \sin(\pi w_k(t+C))), \tag{A.1}$$

$$\begin{aligned} w_k &= 0.01 \\ w_f &= 1.0 \end{aligned} \tag{A.2}$$

where w_k is a constant that controls how gentle the change in force is. t is the current timestep of the simulation and C is a constant sampled randomly between -9999 and 9999 at the start of the simulation. w_f is the magnitude of the wind, and i represents the wind force in any of the x, y or z directions. The force vector $\boldsymbol{W}$ is then applied to the dynamics of the quadcopter, simulating a random wind perturbation at that timestep.

$$\boldsymbol{W} = \begin{bmatrix} W_x & W_y & W_z \end{bmatrix}^T. \tag{A.3}$$

Appendix B

Algorithm Validation

B.1 Randomness Variation

To determine how much randomness impacts the performance of RL algorithms, the training process was completed multiple times on a pre-existing basic environment called the ***Lunar Lander*** from OpenAI Gym. As seen in Fig. B.1, it is a simple rocket trajectory optimisation problem, where the agent is rewarded for landing close to the target.

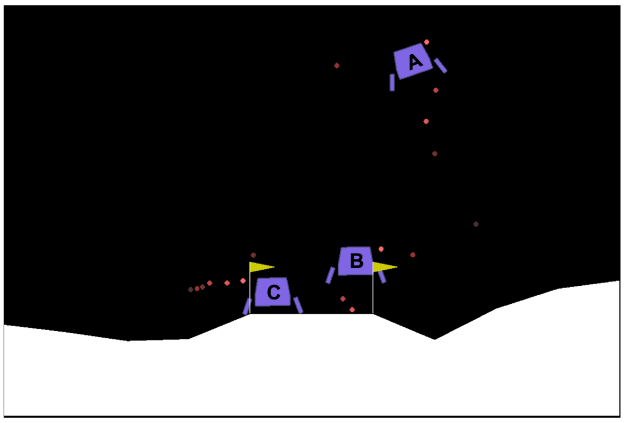

Figure B.1: Lunar Lander Environment

The PPO algorithm was used to train the agent over a total of $500,000$ episodes with the default hyperparameters and Fig. B.2a shows how the reward(loss function), evolves as the training progresses. The reward is a scalar value that gives an indication of how well the agent is doing. It is a measure of how different the current state is from the optimal/desired state, and is what the RL algorithm is attempting to minimise/maximise. The loss function of the Lunar Lander is a weighted sum of the following:

- Negative reward for crashing
- Negative reward for fuel usage
- Positive reward for landing orientated correctly
- Negative reward for distance from landing target

The overall objective of the agent is to land safely on the target while using as little fuel as possible and Fig. B.2a shows the reward progression for five training runs of PPO in the Lunar Lander environment. The initial increase in reward was consistent for all runs, but as the training progresses, significant deviations in reward are present. Fig. B.2b reinforces this idea by showing that the standard deviation of the reward increases, as the number of episodes in training grows.

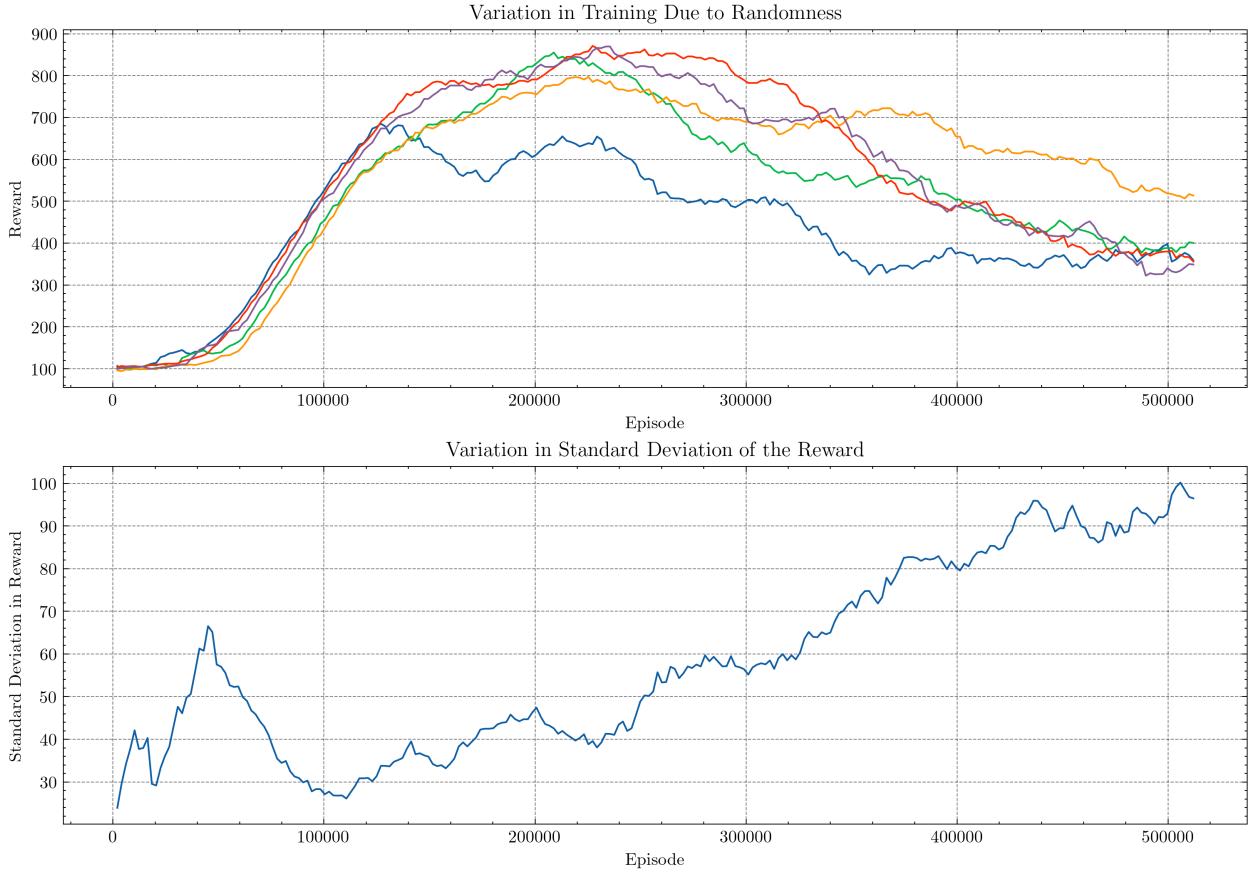

Figure B.2: Plot showing **a)** reward and **b)** standard deviation of reward throughout the training process due to randomness.

These tests show that for all further experimentation, multiple runs must be done to reduce the impact randomness will have on the results. Otherwise, the results may be unrepresentative of the model's true potential performance. Each model henceforth will be trained three times to have a balance of time and performance.

B.2 Algorithm Comparison

To validate the decisions made in Section 2.1.4, various algorithms from Stable Baselines3 were trained on the Lunar Lander environment for $500,000$ episodes, all using the default hyperparameters. PPO, DDPG and TD3 were tested and Fig. B.3 shows that all three have similar growth of reward, with PPO ending training on $\approx -8.12\%$ reward compared to DDPG and TD3. This highlights that there is no significant drop in performance when using PPO over other algorithms, confirming the findings of Section 2.1.4.

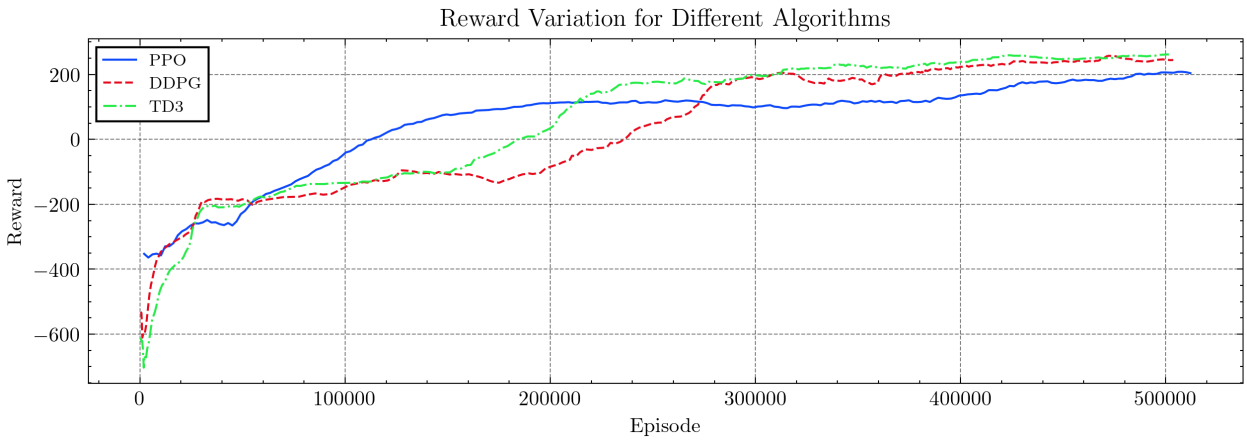

Figure B.3: Plots showing variation in mean reward of multiple RL algorithms.

Fig. B.4 shows how many Frames Per Second (FPS) the training was running at throughout the training. This is a measure of how many timesteps of the quadcopter environment and iterations of the respective algorithm are able to be processed in any given second. Higher FPS leads to faster total training times which is advantageous when running a multitude of experiments on an office laptop. It can be seen that PPO averages around 1000 FPS compared to 200 FPS for DDPG and TD3, so PPO is five times faster. This confirms the findings from Section 2.1.4 and due to this analysis, PPO will be used for all further experiments.

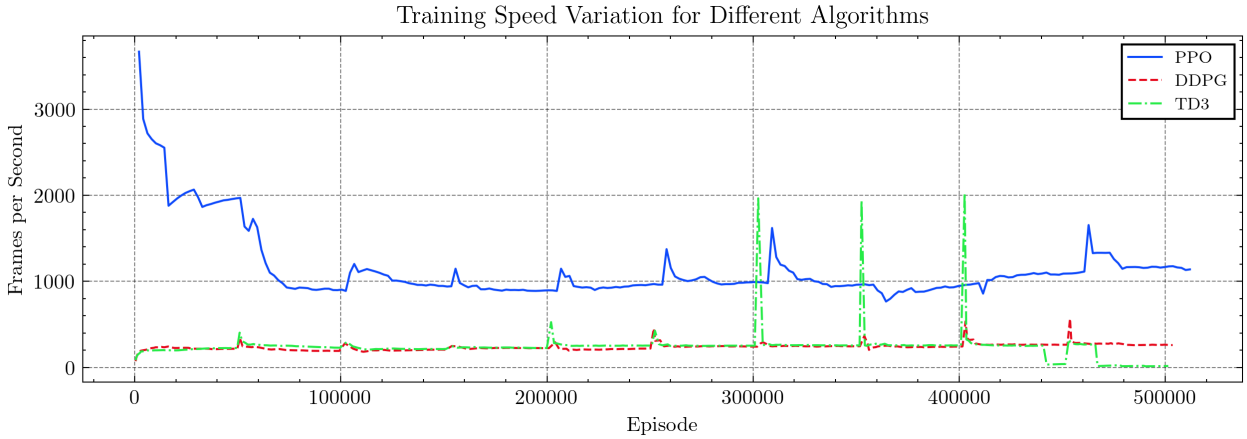

Figure B.4: Plot showing variation in FPS during training of multiple RL algorithms.

Appendix C

Reward Function Analysis

This chapter contains the initial reward functions that were analysed, which were based on a negative squared error.

C.1 Model A.1: Negative Squared Error

C.1.1 Reward Model

One study used RL for the docking of an Autonomous Underwater Vehicle (AUV) [39], employing a negatively weighted sum of error in the x, z and pitch angle, paired with a penalty for termination and a few additional terms. Model A in this study is inspired by this strategy and is adapted as follows:

$$c = -\omega_x x_e^2 - \omega_y y_e^2 - \omega_z z_e^2 - \omega_p p_e^2 - \omega_q q_e^2 - \omega_r r_e^2 \tag{C.1}$$

$$r = \begin{cases} c & \text{if } h = 0, \\ c - p & \text{if } h = 1. \end{cases} \tag{C.2}$$

Where ω_i indicates the i-th weights, p is the penalty for crashing, and j_e represents the j-th state variable error. The boolean h describes whether the episode should be terminated because the quadcopter has crashed, namely:

$$h = \begin{cases} 1 & \text{if } z \leq 0, \\ 0 & \text{otherwise.} \end{cases} \tag{C.3}$$

The reward coefficients are linked to the desired priority of each variable. For instance, to get the agent to prioritise maintaining position in the z-axis over the other parameters, ω_z can be chosen higher than the other values. To select the penalty term p, typical reward values are calculated to gauge how much more the agent should prioritise crash avoidance. The quadcopter just falling from its initialisation position of $(0, 0, 8)$ and crashing leads to a total reward at the final timestep to be $R_T \approx -250$. Setting the penalty to a significantly larger value aims to heavily discourage the agent from crashing and instead should learn to hover to avoid termination. The selected values are summarised in Table C.1.

Table C.1: Weights and crash penalty used by Model A.1 reward function.

ω_x	ω_y	ω_z	ω_p	ω_q	ω_r	p
0.1	0.1	0.5	0.1	0.1	0.1	5000

C.1.2 Results

Model A.1 was set up to train three times, each initialised with different seeds and each trained for 5,000,000 episodes. As seen in Fig. C.1, two out of three training runs crashed, with the reward in both cases decreasing rapidly, causing overflow when the value maximum storable value within a 32-bit Float format ($-3.4028 * 10^{38}$) is reached.

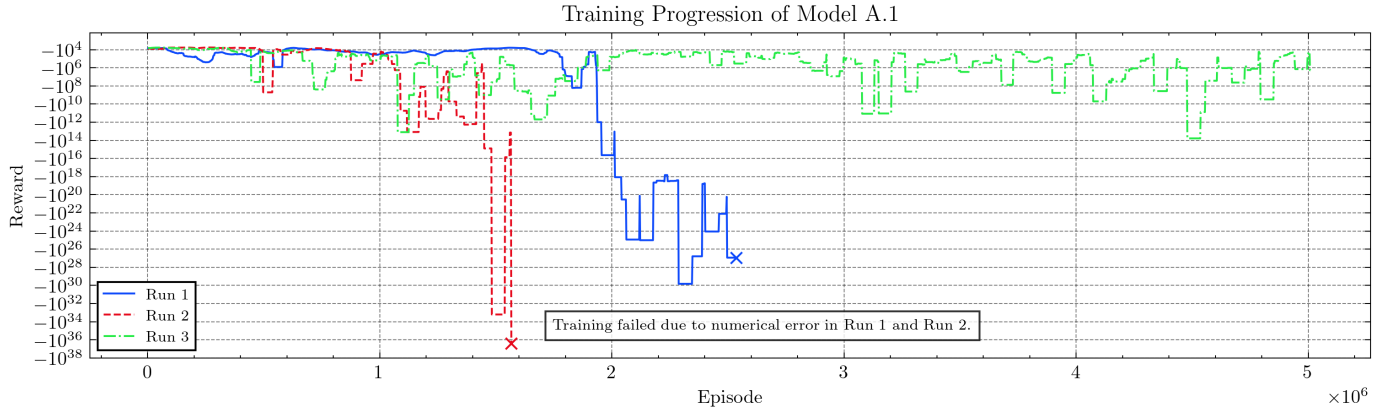

Figure C.1: Three runs of Model A.1 training.

During training, the model is saved periodically to allow for testing of the closed-loop behaviour following the training. Inspecting Model A.1 Run 1 at the last saved iteration before the numerical crash (Episode 2,400,000), shows that it learns to fly away from the ground quickly. In Fig. C.2, the Position and Euler Angles of the quadcopter are illustrated, both for the trained RL controller and for a PID law. The z-axis Position plot shows that during the first 3 s, the quadcopter, initialised at 8 m, flies far away from the target, chosen at 5 m. Simultaneously, the quadcopter begins to spin on all three axes, conveyed by the diverging Euler Angles. With both the position and the angular velocity errors growing rapidly, the reward decreases too. In the case illustrated, the penultimate reward (at $T-1$, with T is the total number of timesteps in the episode), is $r_{T-1} = -1.916 \times 10^6$. The diverging values eventually lead to closed-loop instability and cause a crash after 5 seconds.

C.1.3 Conclusions

Model A.1 struggles to learn the desired behaviour as the agent is given a negative reward every timestep it survives. This could mean that the agent prioritises crashing and getting a one-off large negative reward of $p = 5000$, instead of surviving and getting continuous negative rewards that will continue to accumulate. This could cause the agent to struggle to explore the state space and instead only carry out the undesirable behaviour.

The model performance is also very sensitive to the value of p. Low values could lead to the agent opting to always crash as seen in this model. On the other hand, high p leads to the other objectives of the model being overshadowed, i.e. the agent would have to prioritise not crashing much more than hovering close to the target position.

Finally, the exponential nature of the reward function reported in Eq. (C.1), could be disadvantageous as the penalties may be too harsh for the agent to learn anything of substance. This paired with the quadcopter learning to fly away and spin uncontrollably would cause the extreme negative rewards seen previously in Fig. C.1 that cause the agent to learn nothing.

A potential reason for the reward strategy performing worse than in [39], is due to the differences in the environment. The objective of this study is to hover around a target point for as long as possible, which can be classified as being a *continuous* task, instead of an episodic task. Previous work had a fixed success state that would terminate the episode, such as the successful docking of the vehicle.

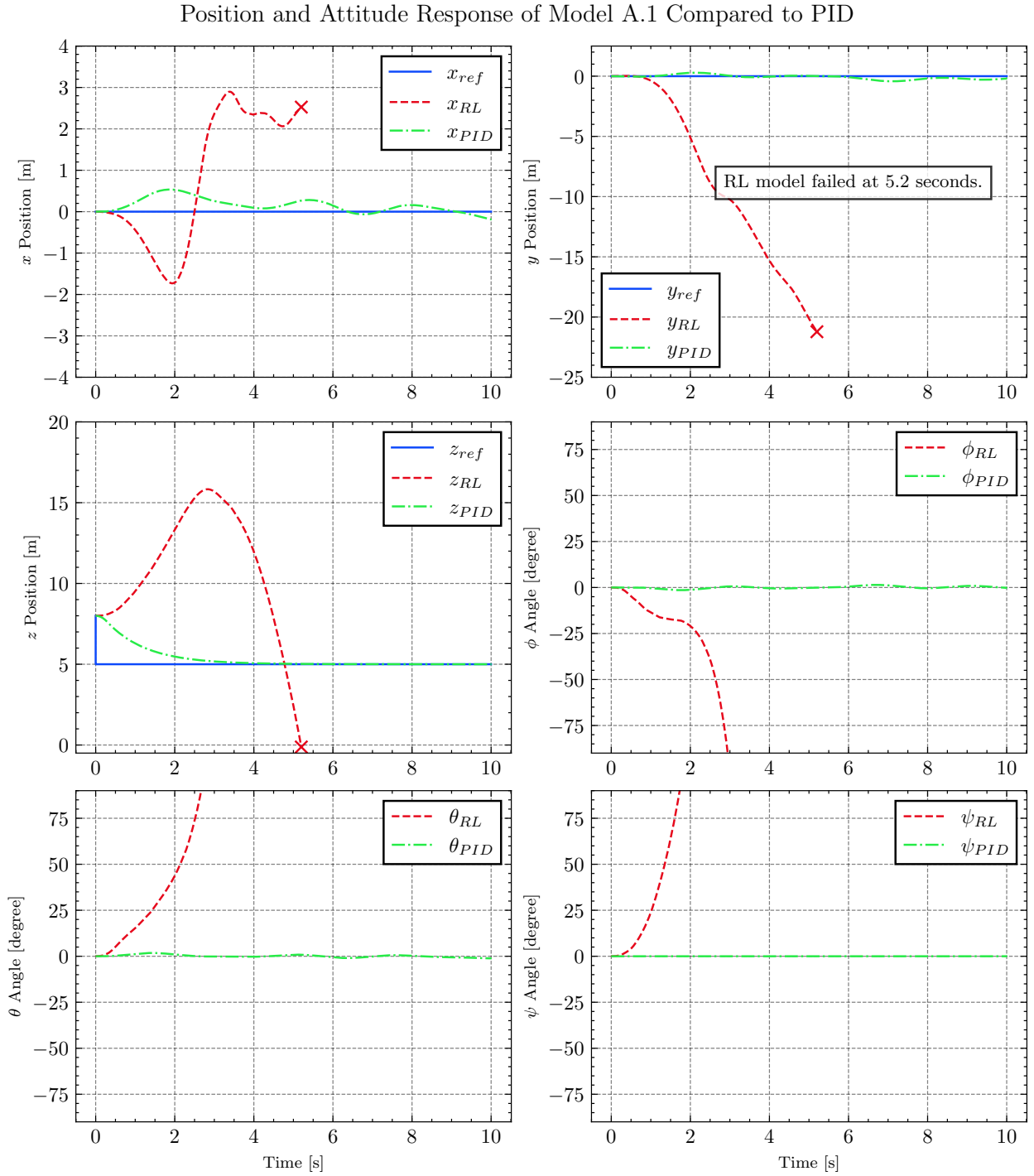

Figure C.2: Model A.1 real-time simulation performance compared to PID.

C.2 Model A.2: Additional Termination States

C.2.1 Modifications

To improve the training results discussed so far, the environment can be modified to prevent the overflowing reward error. This can be achieved by terminating the episode if the agent exceeds pre-defined position bounds, and so the definition of h from Eq. (C.3) can be modified as:

$$h = \begin{cases} 1 & \text{if } x_e < l_{x,min} \text{ or } x_e > l_{x,max} \text{ or } y_e < l_{y,min} \text{ or } y_e > l_{y,max} \text{ or } z \leq 0 \text{ or } z > l_{z,max}, \\ 0 & \text{otherwise.} \end{cases} \tag{C.4}$$

where l_i represents the i-th bounding value. These bounds stop the agent from being able to fly far away from the target by interrupting the simulation when the error exceeds the l_i value. By preserving the same initial conditions [0,0,8] m and the same target value [0,0,5] m, the limit values are selected as reported in Table C.2.

Table C.2: Limits used by Model A.2 reward function.

$l_{x,min}$	$l_{x,max}$	$l_{y,min}$	$l_{y,max}$	$l_{z,max}$
-10	10	-10	10	10

C.2.2 Results

Three successful training runs with Model A.2 were conducted for 5,000,000 episodes each. Fig. C.3 shows how Model A.2 performs in a real-time simulation, and it is reported as compared to Model A.1 and the PID controller. Looking at the x and y Position plots, it is clear that Model A.2 maintains position for significantly longer than A.1 before diverging in a similar manner. The response in the z shows how Model A.2 learns to descend to the target in a similar trajectory to the PID controller, before losing control and eventually crashing. The Euler Angle plots illustrate that the attitude relatively constant for two seconds before starting to spin out of control, a significant improvement on Model A.1 which diverged immediately.

C.2.3 Conclusions

The modifications made to Model A.2 is shown to positively affect the closed-loop performance when compared to Model A.1. Despite the quadcopter learning to initially approach the target, it then loses control and flies away, so more experimentation is needed with different reward strategies to achieve the desired behaviour.

A potential problem with Model A is that the reward is always negative, which punishes the agent for surviving and exploring different control strategies. This model can be classed as *punishment-based* based which is separate to *reward-based*. Studies show that these two methods of RL reward models must be treated differently and perform best in different systems [50] [51]. This may be a potential reason for sub-optimal behaviour, and a *reward-based* strategy will be investigated next.

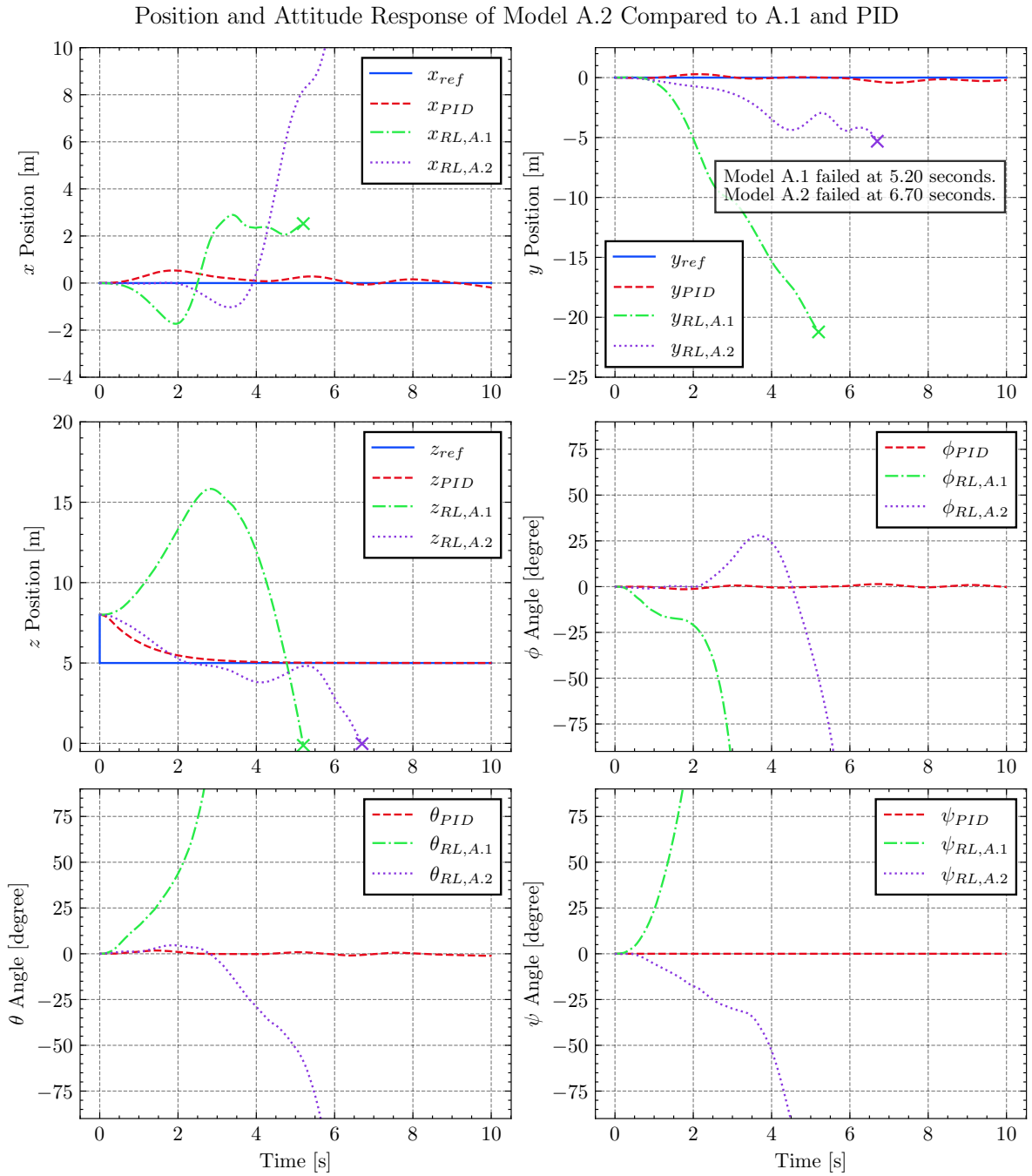

Figure C.3: Model A.2 real-time simulation performance compared to Model A.1 and PID.