Reinforcement Learning and Evolutionary Algorithms for Attitude Control

A comparison for aerial vehicles

Eivind Brastad Dammen

Thesis submitted for the degree of
Master in Robotics and Intelligent Systems
60 credits

Department of Informatics
Faculty of mathematics and natural sciences

UNIVERSITY OF OSLO

Spring 2022

Reinforcement Learning and Evolutionary Algorithms for Attitude Control

A comparison for aerial vehicles

Eivind Brastad Dammen

Reinforcement Learning and Evolutionary Algorithms for Attitude Control

http://www.duo.uio.no/

Printed: Reprosentralen, University of Oslo

Abstract

In recent years, the use of Unmanned Aerial Vehicle (UAV) has increased rapidly because of breakthroughs in relevant technology, such as efficient power sources, software frameworks, smaller form factors and computational power. Quadrotors is a well studied airframe, that is known for its high manoeuvrability and relevancy in multiple tasks, like search, surveillance, monitoring, transport etc. However, this vehicle provide complex dynamics, due to its underactuated nature and complicated aerodynamics. At the same time, research in Artificial Intelligence (AI) has shown impressive capabilities in optimization problems among others.

With increasing complexity and higher demand in UAVs, one could see the need for new control systems that were faster and easier to implement. and this is where AI, or more specific Reinforcement Learning (RL) and Evolutionary Algorithm (EA) comes in, as potential assistants in simplifying and optimizing the development of control systems for complex vehicles, such as the quadrotor.

In this thesis, the performance of RL for attitude control on quadrotor is shown, and its performance is compared to the classical Proportional Integral Derivative (PID) controller. With the help of open source flight controller software, an Evolutionary Algorithm is developed and used to optimize the PID controller using the same environment as the RL model.

The results show that the combination of PID with a effective optimization algorithm, unlocks a great performance from the classical control system, but RL show some benefits in its behaviour, over the PID system.

The tools used in this thesis show how relatively simple it can be to create an accurate model of an aerial vehicle. The performance and behaviour of the different control systems are concluded with a Monte Carlo Simulation.

Acknowledgments

This thesis was written during an odd period to say the least. With restrictions that resulted in mostly digital lectures and exams, to mention some. I would like to thank those that made it possible to finish this thesis. My first thanks goes to my supervisors Kim Mathiassen, Tønnes Frostad Nygaard and Christian Horn, which guided me and inspired me to work hard. Furthermore, I want to thank my family for their support and encouragement throughout these two years.

As I can hear the "Too long"-song starts playing and the bottom of this dedicated page closing in. I would like to give a wholeheartedly shoutout to the bois, which all wrote their thesis partly in the same period as me and experienced the same restrictions.

Contents

List of Figures

List of Tables

Acronyms

AI Artificial Intelligence.

ANN Artificial Neural Networks.

CAD Computer-Aided Design.

COM Center of Mass.

DF Ducted Fan.

DFMAV Ducted Fan Micro Air Vehicle.

DNN Deep Neural Network.

DOF Degrees Of Freedom.

DRL Deep Reinforcement Learning.

DT Digital Twin.

EA Evolutionary Algorithm.

ESC Electronic Speed Control.

FC Flight Controller.

FMU Flight Management Unit.

GPS Global Positioning System.

IMU Inertial Measurement Unit.

KF Kalman Filter.

MAV Micro Aerial Vehicle.

MDP Markov Decision Process.

ML Machine Learning.

NN Neural Network.

PD Proportional Derivative.

PID Proportional Integral Derivative.

PPO Proximal Policy Optimization.

PSO Particle Swarm Optimization.

PWM Pulse-Width Modulation.

RL Reinforcement Learning.

RPM Revolutions Per Minute.

SE(n) Special Euclidean Group.

SITL Software In The Loop.

UAV Unmanned Aerial Vehicle.

VTOL Vertical takeoff and landing.

ZN Ziegler Nichols.

1 Introduction

The drone industry has seen massive growth in the last decade, both for civilian, professional and military use, some of it due to the recent advancements of micro electro mechanical sensors and batteries [1]. This growth has led to the need for research and development in Unmanned Aerial Vehicle (UAV) technology. While applications such as aerial filming, mapping and inspection can be accomplished with the current technology available on the market [2], the industry is pushing to break barriers into the new markets, such as logistics, surveillance in though conditions, e-commerce etc [3] [4]. Unmanned air vehicle with Vertical takeoff and landing (VTOL) provide many advantages to their equivalent fixed-wing counterparts. Being able to move in almost any direction and have a relatively small frame, it can proceed in very challenging areas where humans otherwise would not be able to.

Reinforcement Learning (RL) has gotten a lot of attention lately, with promising results in a broad range of applications. RL has solved very complex tasks in the past, some examples are robotic manipulator grasping and stacking of Lego [5], playing various games like Chess [6], Go [7] and StarCraft [8] with great results. Some of the reasons for this recent rise in popularity has been because of the increasing computational abilities in computers, and their availability to consumers. With increasingly complex environments, the usefulness of this method will rely on the computational power.

With all these previous success stories, one doesn't have to look much further for other interesting problems to tackle. One is if RL can be applied to and be beneficial to the control of Micro Aerial Vehicle (MAV) [9]. RL is an algorithm where an agent learns to perform actions in an environment, to maximize a reward. The agent observes and takes actions in the environment based on its policy, and receives a reward based on

what we want it to achieve in the environment. The simple goal is to find the best action to each state, by improving the policy which in return maximizes the sum of rewards [10].

The control of the different airframes of MAVs is a well studied problem, and is typically achieved by modelling the equations of motion for the aircraft. The challenging parts that remain are due to the complexity of aerodynamics modelling and the many other uncertainties that the aircraft holds [11]. Industry and academia are collaborating on solving the associated challenges, which are all tightly linked to the UAV flight controller hardware and software and has gotten support from foundations like DroneCode, an open-source project under the Linux-Foundation [12]. The quadrotor, which is the airframe of choice in this thesis, is a six Degrees Of Freedom (DOF) aerial vehicle with four lift-and-thrust producing propellers. It shares the hovering and VTOL capabilities of the helicopter.

With the need for more complex UAVs and the demand for these increasing, it is important that the control techniques used are set to a high standard. This is essential since UAVs are almost always safety-critical systems, due to the primary purpose of them being used in environments where humans are present. These vehicles have also very complex dynamics with high non-linearity, underactuation and uncertainties due to the often unmodeled aerodynamic forces and moments in the controller of the vehicle [13]. Reliable and precise controllers which are easily implemented with high availability, is essential.

Deploying neural networks on a variety of problems has become an easy task, with the help of dedicated frameworks [14]. The question is if these algorithms can easily solve the complex control of a quadrotor. By simplifying and speeding up the optimization processes, of the constrained control problem. Ultimately, reducing the development time and cost [15].

1.1 Previous Work

Machine Learning (ML) assisted control theory is a relatively recent subject, even though the word "machine-learning" was invented in 1952 by Arthur Samuel from IBM [16]. The use of ML has gotten a lot of

attention the last decade, due to the new and promising RL algorithms and the increased computational power in commercial computers.

The use of ML on control-problems is not a common subject. The first article about machine learning in position control seems to have been in 1993 [17], but the subject didn't see much development before the 2010s. There has been multiple methods of implementing an ML-assisted control system, one method is by utilizing Particle Swarm Optimization (PSO). As demonstrated in [18], where a pendulum-like MAV-model is used in a simulation, where the control system is using an online PSO-algorithm on the gain-channels, to actively find the best values.

PSO is a computational method that optimizes a problem by iteratively trying to improve a solution by having a population of candidate solutions (particles), and letting them listen to each other, while moving towards a common best solution while also exploring new ones. Imagine a swarm of particles moving on a grid, where each position represents the value for the tuning parameters of a control system. Other bio-inspired algorithms such as genetic algorithm and ant-colony-optimization are other promising methods to find good parameters and solving constrained control problems in robotics [15].

Bio-inspired control has seen use in more than just robotics, [19] built a control model for Maglev Transportations system (magnetic levitation system for trains). Where each gain-channel on a classic Proportional Integral Derivative (PID) control, was controlled by an online PSO algorithm to perfectly balance the train with maglev technology. While the problem here is very different, it's still a type of position controller, so the objective is not entirely different from robotics, from a lower level point of view.

A common method when training models for control in robotics, is by introducing random uncertainties to a dynamic model. This can involve up to 20 percent of randomness in the mass, moment of inertia and the stability and control derivatives. Letting the algorithm learn how to respond to these changes, which results in a reasonably stable control in real world testing [20].

Quadrotors have complex nonlinear behaviours and under-actuated properties, which can make the flight control very challenging. The classical PID feedback control method have been used in many aerial

vehicles and have shown acceptable performance in non demanding scenarios provided that the parameters are sufficiently tuned [21].

With a cascaded controller that controls the attitude and position separately, one can see adequate results with well tuned parameters, but suffers from rapidly changing environments [22]. More modern control methods have been developed through the years, such as Adaptive control, Backstepping Control, Feedback Linearisation and Model Predictive control. These are all nonlinear control methods, but the downside of these is that they require accurate system models, which needs extensive amount of research and tests to obtain a accurate physical model. This gives rise to a limitation where some of these control methods can be tested only in simulation [23].

A geometric method was also tested on a quadrotor MAV [24], but since all these approaches ignore the aerodynamic effects and external disturbances during the controller design, they are unable to be used on MAVs that operate outdoors at high speed.

On more advanced MAVs, like a Ducted Fan Micro Air Vehicle (DFMAV), where the vehicle is controlled by vanes beneath the propeller. These heavily rely on real-time parameter adjustment according to flight conditions when using PID controllers, since this airframe is actively unstable [25]. [26] proposed a back stepping neural adaptive control law to track the DFMAVs nonlinear dynamics. With individual controllers for pitch, roll and yaw axis resulting in a smooth response in simulation for a "bop-up" maneuver (taking off and landing straight down again).

[27] proposed an adaptive neural network approach to MAV tracking control, where an outer loop would directly generate angular velocity commands in the presence of unknown aerodynamics and disturbances, like strong wind. Meanwhile the faster inner loop would handle the angular rate control. The control method showed good performance but suffered greatly in fast changing uncertainties.

Other researchers have also proposed Neural Network (NN) based controllers for these DFMAVs. The same author as the adaptive neural network approach above [26], proposed a Neural-Network-based controller where the controller have been directly tuned on the real aircraft, or pre-tuned on a mathematical model which is acquired through wind tunnel tests or computational fluid dynamics [28]. So the idea with this ap-

proach is to build a reference system, which is easy to tune, and use Neural Network (NN) to learn and compensate for all the uncertainties between the real system and the reference system. First, the system dynamics is divided into two parts. One is completely known and formulated into a reference system containing the reference inputs and the desired dynamics. The other part consists of all the uncertainties. The neural networks are introduced to reconstruct the unknown system dynamics, which are the weight matrices in this instance. They conclude that the controller is capable of steady hover under high-frequency disturbance and good robustness under high speed flight on real tests.

[9] came up with an RL-control approach, where they would use RL to directly map states to the rotor speeds. In their approach, they would ignore all drag forces acting on the body and use a simple floating body model with four thrust forces acting on the body. Even using a simple diagonal inertia matrix, a very simple model yet still achieving great performance in real life experiments on a quadrotor drone.

This way of modeling is sometimes referred to as a "Blackbox"-approach, where the model finds the entire dynamics by optimizing the model to reality. A "Whitebox" model would mean that all the equations are derived from physics and the constants are found through separate tests. "Greybox" is placed somewhere in the middle, where some parts of the dynamics are known while others are found by optimizing the model to real measurements [29].

Two years later, the same author as [9] came up with a new method for training neural net policy in simulations, and transferring to real tests [30]. Their idea is to collect data by trial and error in real tests and auto-tune the controller. These tests were conducted on a quadrupedal legged robot, but the principle can be used on MAVs too, though a little more challenging with flying robots. Their methods uses partly randomized kinematics, this is to insure that the policy will be more robust against system changes and model inaccuracies, since it doesn't solely rely on kinematics. This new method is very cost-effective for data-generation and shows great results with only a few hours of training (depending on the computational hardware).

1.2 Problem statement

The problem in this thesis is mainly addressing the attitude control problem or dynamic positioning of an MAV, with the quadrotor as the airframe of choice. Motivated by training deep neural networks to learn how to fly from simulation. Reducing the cost of developing control systems by simplifying the control problem. Letting the algorithm learn to compensate for the unknown parameters. Finding a cost-effective way for tuning and optimizing the controllers, by using a well designed simulation environment.

To get a better picture of how well the proposed RL controller stack up against the control systems already available. There will be done a comparison to a "conventional" controller, this being the PID controller. This way, one will get an idea if the new controller is adequate, relative to development time and cost. Analyzing the accuracy and potential of the resulting controller under some kind of test.

The quadrotor is probably the most known and used MAV airframe, at least in a consumer setting. There are other interesting airframes that one could study that hasn't seen nearly as much work, like the Ducted Fan Micro Air Vehicle. However these airframes has less open-source contributions available, and one could risk not getting a reasonable result within the timeframe of this thesis. That is why the choice landed on the quadrotor. The controller will be simulated in a modified version of Gazebo through the open source GymFC-framework.

Goals

1. Develop a position controller for a quadrotor by exploring the possibilities of reinforcement learning, with the help of Open Source material.

2. Analyze the stability of the resulting controller and the further potential of this method.

3. See if the RL would act as an adequate controller compared to the existing solutions, and relative to development time and cost.

1.3 Thesis Outline

The rest of this thesis is structures as followed: **Chapter 2** explains the background and theory on UAVs and some of the fundamentals used in the design of an attitude-controller, together with an introduction to AI and the subcategories used in this thesis. **Chapter 3** includes the tools and frameworks used for the implementation of the controllers in **chapter 4**. For **chapter 5**, the experiments and results are shown, which are then discussed in **chapter 6**. The conclusion and future work is found in **chapter 7**, and lastly an appendix with some extra figures and code from the implementation.

2 Background and Theory

2.1 Unmanned Aerial Vehicle

It was not until World War 1, that Unmanned Aerial Vehicle (UAV) were recognized systems [31]. Charles Kettering from General Motors developed a biplane UAV for the army Signal Corps. It was called the "Kettering Bug" and it took about 3 years to finish. It was designed to carry high explosives to a predestined location and was controlled by preset commands, that would detach the wings over a set location to plunge toward its target.

UAVs was used extensively during the Vietnam-war but for reconnaissance missions only. The vehicle were usually launched from a C-130 bomber airplane and recovered by parachute. Although many UAVs were flown, these vehicles didn't really have a big effect on the war but the military started to think of "what could have been" a potentially key weapon to have in future wars, which assured their continuing development. However, the interest in these vehicles dwindled, as they simply weren't that reliable and the technology was not there yet. The UAVs at the time were still using combustion engines and the data-link transmissions saw lots of interference from other communication-systems. The UAVs built in this era were mostly "fixed-wing", which look more like small airplanes. These have less degrees of freedom compared to the multi-rotor vehicles that are popular today. The war in Afghanistan and Iraq changed the status of the UAVs, where they were valued as a key weapons system. They were especially good at night, locating and keeping track of insurgent forces, which proved UAVs as a valuable weapon for the military [31].

Rotorcraft UAVs are attractive robotic platforms, with their low cost, high maneuverability and simplicity of use. Thus many have contributed

to the field and several industries have taken usage of this type of vehicle. A quadrotor is a subcategory under rotorcraft, where there are four rotors connected to the body of the vehicle. These rotors control the orientation by adjusting the speed of each rotor, usually in pair. Generally, it should be classified as a rotary-wing aircraft according to its capability to hover, horizontal flight and VTOL capabilities [32].

These days, a quadrotors size can vary greatly, from over 5kg to less than 50g, although vehicles with a weight of less than 1kg is most often referred to as a Micro Aerial Vehicle (MAV) [33]. With versatile forms, come versatile application, like search and rescue, aerial photography or even carrying payloads to hazardous areas. Since the quadrotor is inexpensive and easily assembled, as well as complex dynamics which makes for an excellent research platform for aerial robotics, for the problems related to three-dimensional mobility and perception [13].

2.1.1 The control problem

Compared to control for other advanced autonomous manipulators, the control of UAV presents two unique challenges. First, they have constraints in terms of size, weight and power not found to the same extent in robotics. Second, UAV control systems are in practice almost always safety-critical systems, due to the primary purpose of them being used in environments where humans are present.

It is clear that UAVs need a complex control system with high reliability and precision. With its high non-linearity, strong inter-axis vehicles dynamic coupling, under-actuation and uncertainty due to often unmodeled aerodynamic forces and moments on the vehicles body, rotor induced flow, ground effect, rotor-flapping etc [11].

The quadrotor consists of four individual rotors attached to a rigid frame in a cross shape. The control is achieved by differential control of the thrust generated by each rotor [33]. Pitch, roll and yaw control. As shown in figure 2.1, rotor 1 and 3 rotates in the same direction (anticlockwise in this case), while the even numbered rotors rotate the opposite direction. Yaw control is obtained by adjusting the average speed of the clockwise and anticlockwise rotating rotors. The system is underactuated, meaning the remaining degrees of freedom (DOF) in the translational velocity in

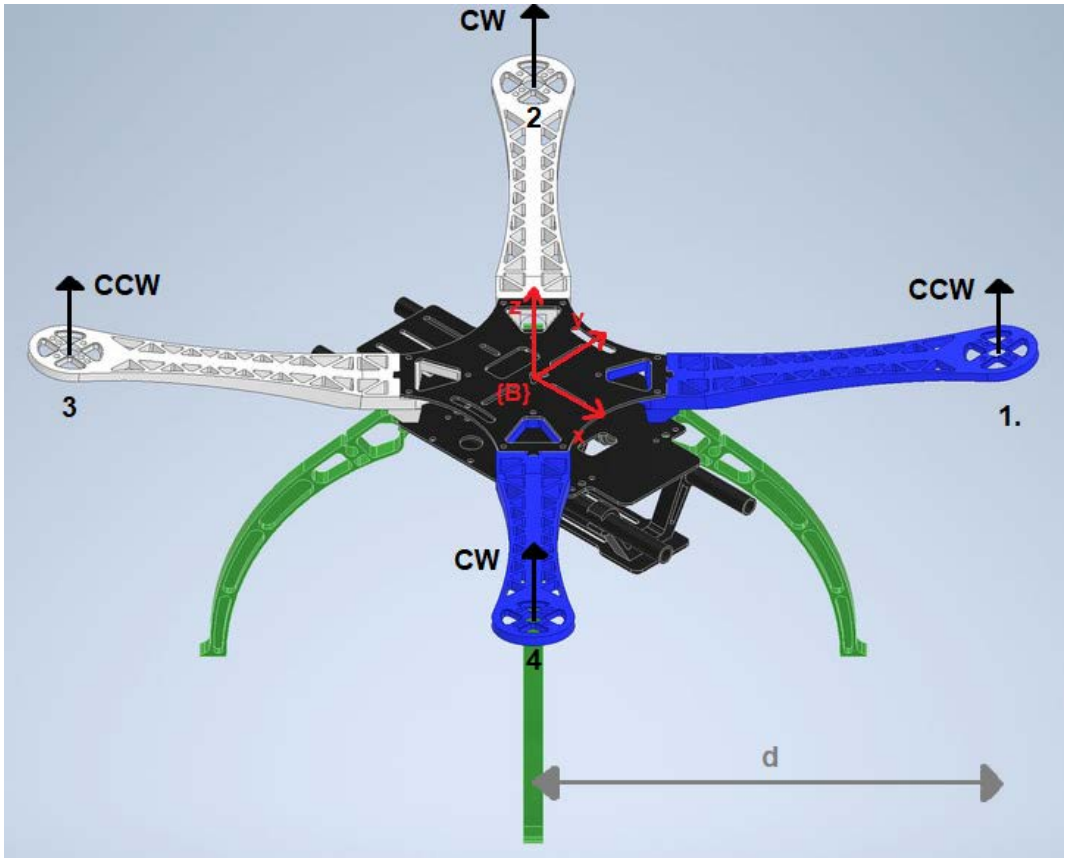

Figure 2.1: Quadrotor body frame

x-y plane must be controlled through the system dynamics. The vehicle uses roll and pitch to point the thrust vector in the desired translational motion. There are several frame-designs for quadrotors, but figure 2.2 - 2.5 covers most of them.

2.1.2 Rigid-body dynamics and scaling laws

Roll angle (ϕ) is around x, pitch angle (θ) around y and yaw angle (ψ) is around the z axis.
Equation 2.1 is the 6 DOF rotational matrix for a quadrotor.

$$R = \begin{pmatrix} c\psi c\theta - s\phi s\psi s\theta & -c\phi s\psi & c\psi s\theta + c\theta s\phi s\psi \\ c\theta s\psi + c\psi s\phi s\theta & c\phi c\psi & s\psi s\theta - c\psi c\theta s\phi \\ -c\phi s\theta & s\phi & c\phi c\theta \end{pmatrix} \tag{2.1}$$

where c and s are short for cosine and sine.
Let $v \in \{A\}$ (the inertial frame), denote the linear velocity of the body

frame *{B}* with respect to *{A}*. $\{A\}$ is denoted by the unit vectors by $\{\vec{a_1}, \vec{a_2}, \vec{a_3}\}$, where $\vec{a_1} = \vec{x}, \vec{a_2} = \vec{y}, \vec{a_3} = \vec{z}$. Ω is the angular velocity of *{B}* with respect to *{A}*. Let m be the mass of the rigid body object, and $\boldsymbol{I} \in \mathbb{R}^{3\times3}$ denote the constant inertia matrix expressed in *{B}*. The rigid body equations of motion for the quadrotor are:

$$\dot{\zeta} = v, \tag{2.2}$$

$$m\dot{v} = mg\vec{a_3} + RF, \tag{2.3}$$

$$\dot{R} = R\Omega_x, \tag{2.4}$$

$$\mathbf{I}\dot{\Omega} = -\omega \times \mathbf{I}\Omega\ \tau \tag{2.5}$$

where Ω_x is the skew symmetric matrix, $\Omega_x v = \Omega \times v$ for the vector cross product $\times$ and any vector $\boldsymbol{v} \in \mathbb{R}^3$. The vectors $\boldsymbol{F}, \tau \in \{B\}$ combine the principal forces and moments applied to the quadrotor airframe [33].

Changing the scale of the quadrotor has an effect on the inertia and the achievable angular and linear acceleration. One might find it useful to find a physics model to analyze a quadrotors ability to produce linear and angular accelerations from steady hovering state. In equation 2.6, length from center of drone to motor is d (as seen in figure 2.1), rotor radius r and rotor speed ϖ. The lift/thrust and drag from the rotors scale with the square of the rotor speed, ϖ^2. Linear acceleration is denoted as $a = \dot{v}$ and angular acceleration as $\alpha = \dot{\Omega}$. Then the moment arm and the moments of inertia, scale as:

$$a \sim \frac{\varpi^2 d^4}{d^3} = \varpi^2 d \tag{2.6}$$

$$\alpha \sim \frac{\varpi^2 d^5}{d^5} = \varpi^2$$

To explore the scaling of the rotor speed with length, one might use two commonly accepted approaches within aerial vehicles. Mach scaling and Froude scaling. Assuming $r \sim d$, we get $\varpi \sim (\frac{1}{\sqrt{r}})$,

Mach scaling predicts

$$a \sim \frac{1}{d}, \tag{2.7}$$

$$\alpha \sim \frac{1}{d^2}.$$

Froude scaling leads to

$$a \sim 1, \tag{2.8}$$

$$\alpha \sim \frac{1}{d}.$$

One of the assumptions here is that the rotor blades are rigid, which might be inaccurate and the blade designs themselves might also be optimized differently and might change the scaling.

The biggest takeaway from Froude and Mach numbers is that smaller quadrotors can produce faster angular accelerations while the linear acceleration is at worst unaffected by scaling. This might not come as a surprise that the smaller quadrotors are more agile than the bigger versions [33].

2.1.3 Aerodynamics

The aerodynamics of rotors has been extensively tested and studied during the mid 1900s with the development of the manned helicopters. The design of these rotors are crucial to the design problem. Typically on a robotic quadrotor, you'll see only a few different rotor designs and only the basic level of the aerodynamic modeling is required.

The thrust in free air is modeled by the rotor disk area A_{r_i}, radius r_i, angular velocity ϖ_i and thrust coefficient that depends on the geometry and profile of the rotor C_T. ρ is the air density. The thrust equation for a rotor:

$$T_i = C_T \rho A_{r_i} r_i^2 \varpi_i^2 \tag{2.9}$$

There are many aerodynamic and gyroscopic effects associated with

every rotor craft, the Thrust force mentioned above is just one of them. Some of the effects that are worth considering are blade flapping and induced drag [33].

Quadrotor vehicles are typically equipped with lightweight, plastic rotors. These are not very rigid, and the aerodynamic and inertial forces applied to a rotor in flight are quite strong, which can cause the rotor to flex. Letting the rotor bend is actually an important property of the design of a quadrotor. A too rigid rotor can lead to damage on the motors through the aerodynamic forces or on the airframe.

> **Rotor flapping**: The high angular momentum of the rotor disk makes it act like a gyroscope, which casues the rotor disk to tilt around the axis. Since the motor shaft is vertical, the blade flaps up as it advances into the wind. When the blades change their angle of attack, it also changes the lift coeffiecient. [34]

For a typical rotor, the flapping dynamics converge to a steady-state after one rotation, so the modeling can be simplified to only the steady-state response of the flapping dynamics need to be considered.

When using a simulator that derive the force and torque for a propeller propulsion system using blade element theory. The performance can be defined by two coefficients C_T and C_Q [35]. These are dimensionless and are for the thrust and torque respectively. Thrust coefficient is defined as:

$$C_T = \frac{T}{\rho n^2 D^4} \tag{2.10}$$

where T is thrust, ρ is air density, n is the propeller speed in Revolutions Per Minute (RPM), and D is the propeller diameter. The torque coefficients is defined as:

$$C_Q = \frac{Q}{\rho n^2 D^5} \tag{2.11}$$

where Q is the torque.

The power coefficient is defined as:

$$C_P = \frac{P}{\rho n^3 D^5} \tag{2.12}$$

These coefficient is needed to predict how the thrust and torque vary with RPM. These coefficients depend on the advance ratio J which quantifies the effects of the propeller in forward motion with relation to its angular velocity:

$$J = \frac{V_\infty}{nD} \tag{2.13}$$

where V_∞ is the freestream fluid velocity.

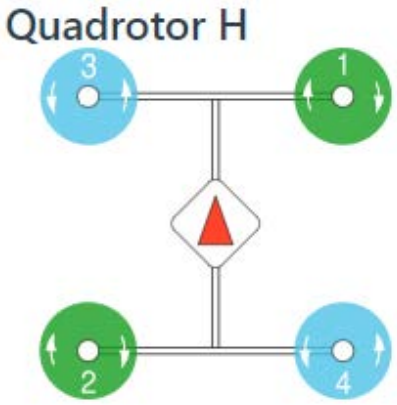

Figure 2.2: H-frame from [36]

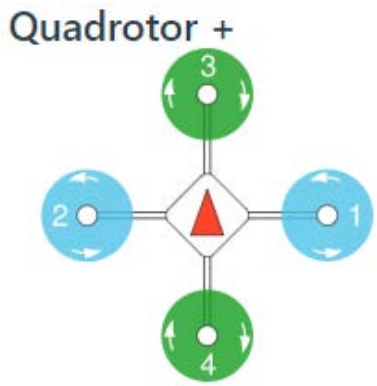

Figure 2.3: Cross-frame from [36]

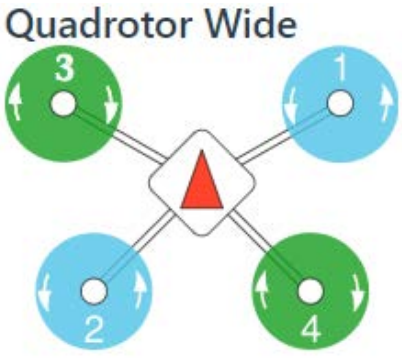

Figure 2.4: Wide-frame from [36]

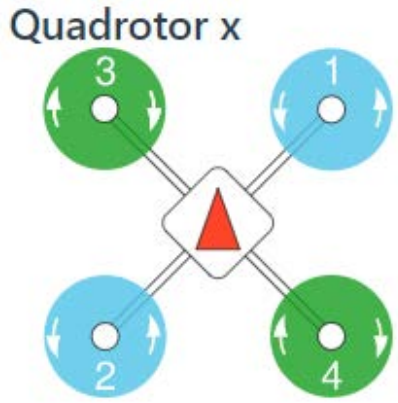

Figure 2.5: X-frame from [36]

2.1.4 Estimating the state

When controlling the quadrotor or any other UAV, the required states for controlling are its height, attitude, angular velocity and linear velocity. The most important instrumentation of the quadrotor is the Inertial Measurement Unit (IMU) with gyro, accelerometer and magnetometer accomponied by a height measurement of some sort, like acoustic, infrared, laser-based or barometric pressure. In some more advanced cases, one might also see the need for cameras or Global Positioning System (GPS) to get a position reference to the ground.

These sensors are selected to be able to estimate the orientation of the drone and its altitude. Both the simulation and the real drones share the same sensors. These sensors are usually placed together inside a unit called the flight controller.These sensors gather info that forms the state vector of the drone:

$$x = \left[x, y, z, \psi, \theta, \phi, u, v, \omega, p, q, r\right]^T \tag{2.14}$$

Which represents, position, attitude, velocity and angular velocity.

2.2 Control Theory

The control problem is challenging for several reason. First, the system is underactuated, four inputs while SE(3) is six dimensional. Second, the aerodynamics are only approximate and finally, the motor controllers must overcome the drag moments to generate the required speed and realize the input thrust and moments.

The most common controller is a hierarchical control approach where the lowest level with the highest bandwidth, controls the rotor rotational speed. The next level controls the vehicle attitude and outer level is in control of the position. The idea of separating the control into two or more cascaded loops is seen in many applications. The benefits of these separated loop structures is to simplify implementations, ease of tuning, and satisfaction on input or state constraints [22]. For quadrotors, the outer loop stabilizes translational variables like position and linear velocity, and generates a reference signal that is further sent to the inner loop, the attitude controller which controls the angle of the vehicle. This

is the part of the controller that we will be looking at in this thesis.

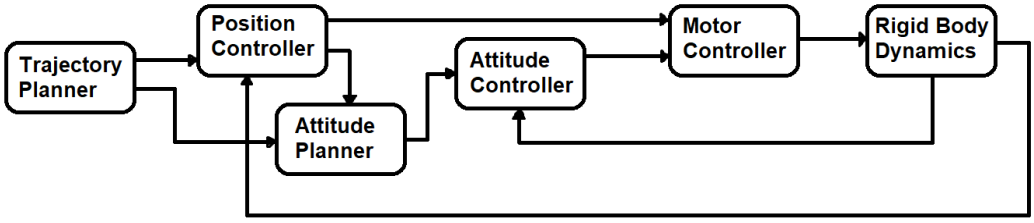

Figure 2.6: Hierarchical controller

2.2.1 Proportional Integral Derivative

Proportional Integral Derivative (PID) controllers are probably the the most widely used industrial controller. It has a long history of use, even surviving the changes of technology from the analogue to the digital era [37].

A visual representation of the three terms of the PID controller is shown in figure 2.7. Proportional control, denoted by the P-term gives the proportional gain to the size of the error signal $e(t) = r(t) - y_m(t)$, where e is error, r is reference and y is the measured output at time t. A Proportional controller can be represented in the time domain as:

$$u_c(t) = k_P e(t)$$

The integral part denoted as the I-term, is used when there is need to correct for a steady offset, from a constant reference signal value. Representation in the time domain is given as:

$$u_c(t) = k_I \int^t e(\tau) d\tau$$

A derivative controller (D-term) uses the rate of change of an error signal as an input. Time domain representations is given as:

$$u_c(t) = k_D \frac{de}{dt}$$

When all the terms is combined, you get a PID controller. The formula for the basic parallel PID controller is:

$$u_c(t) = \left[k_P + k_I \int^t dt + k_D \frac{d}{dt}\right] e(t)$$

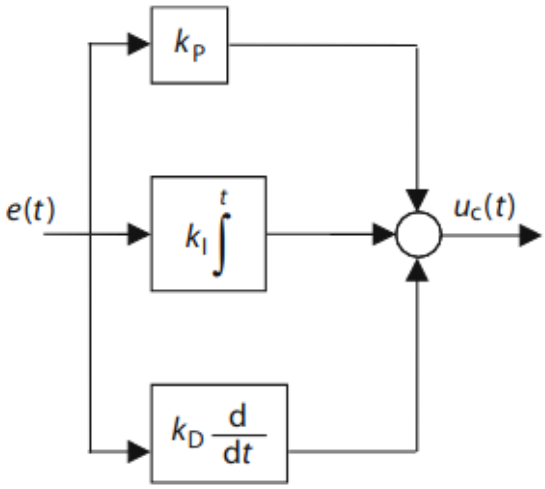

Figure 2.7: Block diagram of PID controller in time domain. Taken from [37]

PID processes the error signal into input signal, with the aid of a proportional factor k_P, an integrative action k_I and differential action k_D. The states are compared with their setpoints, where an error is calculated. The error is amplified with k_P, it is the present error term and will improve the speed of the response. A accumulation of past error is integrated upon with k_I and differentiated with k_D, which is the prediction of future error.

This controller is easy to implement but the tuning process can be time consuming. There are many ways of tuning a PID system, usually by starting with proportional gain, then the derivative gain, and lastly the integral gain. These values are tuned until the user is satisfied with the response from the controller, known as the "trial and error" method [38].

2.2.2 Ziegler Nichols

Ziegler Nichols method, which is a purely mathematical approach to automatic control. The method was used to simplify the mathematics involved when studying automatic control on any application [39]. This method is meant to assist the designer when doing adjustments on existing controller applications and in the design of new installation.

The method is based on finding the lowest Proportional-value K_u where the controller outputs a steady oscillation. Then one would measure the period of the oscillations T_u and insert the two values K_u and T_u in the equations 2.15, 2.16, 2.17.

$$K_p = 0.6K_u \tag{2.15}$$

$$K_i = 1.2\frac{K_u}{T_u} \tag{2.16}$$

$$K_d = 3K_u\frac{T_u}{40} \tag{2.17}$$

2.3 Artificial Intelligence

Artificial Intelligence (AI) has become a broad term that contains a long list of statistical methods on data, so that a computer can "learn" a task or find patterns. Within the subcategory Machine Learning (ML), we have mainly three categories:

- Supervised Learning: **learn with a direction**
  - An agent is given labelled data and is set to try to learn how it's categorized. Example what makes an apple, and how is it different from a pear etc. It tries to generalize a rule by looking at examples.
- Unsupervised Learning: **learn without a direction**
  - Similar to Supervised, except the data is not labelled. It is up to the agent to figure out the patterns for itself. There are several methods within this category, like clustering where groups of data which are similar are labeled together.

    Another method is Principal component analysis, which takes multidimensional data and reduces it. The result is a lower-dimensional representation of the data, where one can see correlations between the important components of the data. This is also results in a visually better representation of data, which can be more easily interpreted.

- Reinforcement Learning: **learn interacting with an environment**
  - An agent is placed inside an environment without any idea of what the task is. It will start by acting randomly on the environment and receive a reward or punishment based on that action. After a series of episodes, the agent will learn what actions to make, based on how the reward-function is set up.

2.3.1 Evolutionary Algorithm

Another part of the Artificial Intelligence field is Evolutionary Algorithm (EA). It is mainly used in tasks where search and optimization is needed. The algorithm iteratively optimizes a problem, also known as a heuristic procedure, designed to find, generate or select a sufficiently good solution to an optimization problem. This algorithm simulates a population of individuals which are the solutions, and uses operators like mutation and recombination on these individuals. There are many different ways of generating new individuals with evolutionary operators, the most common methods are:

Mutation

A common mutation technique is the non-uniform mutation with a fixed distribution. This operator takes one solution as input and generates a new one. The genes of the individuals are in this case, a set of floating point numbers. The genes have a mutation probability and are then mutated by adding a value, from a specific distribution, like Gaussian. This operator will most often let small mutations happen, while large mutations occur less frequent.

Crossover

Crossover or recombination, where the offspring of the chosen parents is given a chance to inherits some of its parents attributes. This operator can take the solution between two individuals and calculate the average value between them, to create a new offspring.

Further on, the algorithm needs a fitness of each individual, which reflects the quality of their solution, and is used when choosing parents for the next generation [40].

For this algorithm to work, one has to model the following:

- a method for representing problems as chromosomes
- a way to calculate the fitness of a solution
- a selection method to choose parents
- a way to generate offspring by breeding the parents

Algorithm 1 Evolutionary Algorithm

INITIALIZE population with random candidate solutions;
EVALUATE each candidate;

while (TERMINATION CONDITION is not satisfied) **do**

1 SELECT parents;

2 RECOMBINE pairs of parents;

3 MUTATE new candidates;

4 EVALUATE new candidates;

5 SELECT individuals for the next generation;

end while

The Evolutionary Algorithm has lots of possibilities, and it works very well most of the time. This algorithm like many others has its drawbacks. One main point is that it can be very slow. If the solutions of the population reaches a local maximum, it can take a very long time before a mutation is generated that can escape this maximum. A mutation great enough to find another higher, local maximum. It is also not guaranteed that the algorithm will always converge towards an optimal solution.

2.3.2 Artifical Neural Networks

Artificial Neural Networks (ANN) are called networks, because they contain multiple neurons that are connected together, which is an attempt to mimic the neurons of a brain. Each node has a number of inputs and an output which can be further connected to a new node.

The node is simple, when a linear combination of the inputs exceeds a value, the node "fires" and produces an output.

$$in_j = \sum_{i=0}^{n} w_{i,j} a_i \tag{2.18}$$

Here the index i are all the nodes connected from the previous layer, w is the weight of each connection between the nodes. There is an activation function g which tells when the neuron should "fire":

$$a_j = g\left(in_j\right) = g\left(\sum_{i=0}^{n} w_{i,j} a_i\right) \tag{2.19}$$

The neurons are arranged in different layers, their categories are "input layer", "hidden layer" and "output layer".

How the network "learns" is by updating its weights between each node, which starts out as one arbitrary value or is randomized. Usually with a method called back-propogation, where one propogates the error from the output layer, back to the hidden layer, then calculates the error in the last hidden layer, then the previous and so on.

The rule for updating a weight at layer i, node j is:

$$w_{i,j} \leftarrow w_{i,j} + \alpha \times a_i \times \Delta_j \tag{2.20}$$

$$\Delta_j = e_k \times g\left(in_j\right) \tag{2.21}$$

where e is error, g is the activation function and in_j is the weighted sum of the inputs.

2.3.3 Deep Learning

Deep Learning is based on ANN but uses multiple layers of nonlinear processing units in order to learn. Most often referred to as Deep Neural Network (DNN). These have become very popular, and are responsible for some very impressive abilities.

The definition of DNNs is simply a ANN which contains multiple hidden layers between the input and the output layer, as long as there is more than one layer. Figure 2.8 shows a deep neural network.

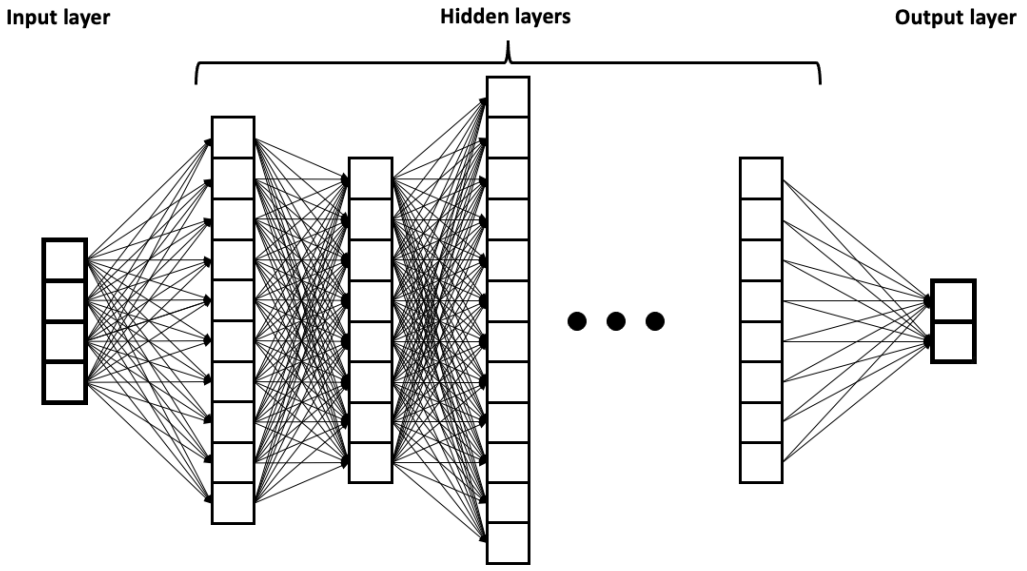

Figure 2.8: Deep Neural Network with n hidden layers from [41]

2.4 Reinforcement Learning

Reinforcement Learning (RL) problems involve learning what to do, how to map situations to actions which maximize a reward. The learner in this problem is called an agent, which is not told what actions to take, as in some of the other forms of ML, but instead must discover which actions gives the most reward by trial and error. An action may affect the immediate reward but also the consequences of the next step. The environment is where the agent makes its decisions, it is what the agent try to manipulate or act upon, and where the goal can be achieved. To be able to make any form of decision, the agent has to sense the environment in some way, and sense that the state has changed after an action. Lastly, it needs to be able to receive rewards based on the state it senses. The formulation includes these three aspects: Sensation, action and goal [10].

The agent

The agent is the "player" that interacts with the environment in the hopes of getting a reward. It can sense the environment in some way, this is usually the inputs of the environment, also called observer-space, and the agent can interact in some way with its outputs or "the action-space".

Reward-function

The reward or fitness defines the goal that an agent is trying to solve. At each time step, the agent receives a reward from the environment. The only objective for the agent, is to maximize the total reward over the long run. So the agent also needs to learn actions that give a greater total reward, and not just actions that give immediate reward, actions that give direct and indirect effect on the reward.

Markov Decision Process

An RL problem which only uses the current action and state to choose its next move, has the Markov Property. Markov Decision Process (MDP) is a way to compute the next reward, and the next state, from only the current state and action, based on previous experience [40]. The equation that computes the probability of the next reward r' and the next state s' at time t is shown in 2.22.

$$Pr\left(r_t = r', s_{t+1} = s' | s_t, a_t\right) \tag{2.22}$$

There is a set of states, which contains all the states of the environment, and a set of actions which can be taken in each state. Pr denotes the probability of reaching the next state s' when performing action a_t in state s_t. The amount of states available depends on how much the agent can observe. It can be *partially observable* through the sensors that the robot is given or it can be *fully observable*, like all the squares on a board of chess. The action space are all the given action the agent can take. In chess, this would include all the valid moves, at a specific state of the game.

Gradient Descent

If we have a function-value that we want to minimize, gradient descent will help to find in which direction the variable should go. Meaning if we should increase or decrease a variable x. In the context of neural networks, this would be used to gradually reduce the error on the weights. Gradient descent uses the gradient of the loss function of the neural network, to progressively search for a minimum.

Value Function

The expected reward at the next action, is called the *value*. There are two ways of computing this. We can consider the current state, and take the average of all valid actions, leaving the policy to sort it out (*state-value function* V(s)). Or we can take each action that can be taken separately, the *action-value function* $Q(s, a)$.

$$V(s) = E(r_t|s_t = s) \tag{2.23}$$

$$Q(s, a) = E(r_t|s_t = s, a_t = a) \tag{2.24}$$

where $E(\cdot)$ is the statistical expectation.

The second estimate is more accurate in the long run, since it has more information.

The optimal value function when following policy π is:

$$V'(s) = max_\pi V_\pi(s) = max_\pi E(r_t|s_t = s) \tag{2.25}$$

The optimal action-value function when following a policy π is:

$$Q'(s, a) = max_\pi Q_\pi(s, a) = max_\pi E(r_t|s_t = s, a_t = a) \tag{2.26}$$

These two functions can be linked. The first considers taking the optimal action at each case, the second considers action a at this time, an then following the optimal policy. Now we only considers the current reward and the discounted γ estimate of the future rewards:

$$Q'(s, a) = E(r_{t+1}) + \gamma max_{a_{t+1}} Q(s_{t+1}, a_{t+1})$$

$$= E(r_{t+1}) + \gamma V'(s_{t+1}|s_t = s, a_t = a) \tag{2.27}$$

Exploration vs. Exploitation

At each stage of the learning process, the algorithm looks at the available actions and computes a value for each. If the algorithm is set to always pick the one with the highest value, we say it's *greedy*. It chooses to always

exploit the current knowledge. If the algorithm is given a probability to pick some other action at random, we say that the policy uses exploration to gather more information about the environment.

Policy

Action selection is aiming maximize the expected reward by finding a balance between exploration and exploitation. If a decision is made to always pick the optimal choice at each stage, and not do exploration. This is called a policy π. The agent wants to learn the best policy that is specific to each state s_t. The essence of RL is to learn a policy from state to action. In order to find this policy, one needs to know how much information is needed for how we got to the current state. Second is how we give value to the current state [40].

2.5 Proximal Policy Optimization

In Reinforcement Learning (RL), the algorithms have many moving parts that may be hard to debug, and require high effort in tuning in order to get good results. Proximal Policy Optimization (PPO) is built to be a balance between ease of implementation, sample complexity and ease of tuning. It was built when Deep Q-learning and Trust region Policy Optimization (TRPO) were the leading contenders, were PPO was designed to fill the gap were these algorithms were lacking. It is set to compute an update at each step that minimizes the cost function while not deviating too far from the previous policy [42].

The PPO algorithm is a state-of-the-art RL-algorithm based on an actor-critic architecture. This means that an agent set doing actions in an environment, receiving rewards based on how well it performed. In addition to this actor, a critic is implemented, which calculates what other actor's action would have given as a reward, thereby showing what the actor should have done and comparing it to what the actor did. This makes the PPO algorithm learn quickly from its actions and adapts the gradient descent based on how much better the critic calculated that the agent could have done.

Policy gradient methods work by computing an estimator of the policy

gradient and putting it into a stochastic gradient ascent algorithm. This has the following form:

$$\hat{g} = \hat{E}_t \left[\nabla_\theta log \pi_\theta(a_t|s_t) \hat{A}_t \right] \tag{2.28}$$

π_{theta} is a stochastic policy, $\hat{A}_t$ is the estimator for the advantage function at t time step. $\hat{E}_t$ is the expectation, the average over a given batch of samples.

To make the gradients adjustable, a clipping method is used on the advantage estimation function, thereby hindering too large optimization steps. The clipping method is used to remove incentives for the policy to get too far from the old policy. The epsilon used is a hyperparameter determining how far away from the old policy a new policy is allowed to be and is generally set to 0.1-0.2 or 10-20%.

$$L^{CLIP}(\theta) = \hat{E}_t \left[min(r_t(\theta)\hat{A}_t, clip\left(r_t(\theta), 1-\epsilon, 1+\epsilon\right)\hat{A}_t) \right] \tag{2.29}$$

The first term inside the min is the conservative policy iteration. The second term clip, modifies the surrogate objective by clipping the probability ratio.This ensures that r_t won't move outside of the interval $[1-\epsilon, 1+\epsilon]$. The min of the clipped and unclipped objective, which finds a lower bound on the unclipped objective. By clipping, we ensure that results that are far away from the current policy are held within this threshold, making sure that the algorithm is not too sensitive to precise hyperparameter tuning and outlier data in training. The PPO seeks to maintain smooth, gradual gradient updates to get continuous improvement and avoid unrecoverable crashes.

The primary function that makes the PPO able to do all this improved learning is the surrogate policy loss function. It is a ratio between the new probabilities and the old probabilities times the advantage given by the advantage function. This advantage function estimates the relative value of a selected action by taking the discounted sum of rewards and subtracting a baseline estimate (estimation of future reward) given the state that the actor is in.

Algorithm 2 Proximal Policy Optimization

for iteration=1,2,... **do**

  for actor=1,2,...,N **do**

    Run policy $\pi_{\theta_{old}}$ in environment for T time steps

    Compute advantage estimates A_1,...,A_T

  end for

  Optimize surrogate loss L wrt θ,
with K epochs and minibatch size M $\leq NT$

  $\theta_{old} \longleftarrow \theta$

end for

2.6 Digital Twin

A Digital Twin (DT) is the combination of physical entities in the physical world, the virtual models in the virtual world and the connected data that the two together. There is no unique definition of a DT in literature, but in present study, the DT model is considered as a detailed digital representation of the physical components, in this case being an aircraft-system. The physical data is taken from various sources like real sensor-data for the propeller/motor combo, physical measurements on the aircraft etc. This data can be combined and optimized to predict the behavior of the system [43].

A common use of DT is to predict the life-span or maintenance-schedule or degradation of a system. As in [43] where it is being used to model the behaviour of a aircraft-engine over its lifespan. Taking data from historical maintenance, and a wide range of real-time sensors. The idea is to verify what a normal behaviour of each part of the engine is, so that they can detect outliers associated to known failures. Thereby preventing real failures by catching these abnormal behaviours ahead of time.

NASA has developed a similar system for their vehicles. A ultra-high fidelity simulation that mirrors the health management system of the real rocket, which gives a major increase in levels of safety and reliability [44].

An important part of DT is ML, as it helps with finding correlations in high-dimensional data, usually with unsupervised learning algorithms [43].

2.7 Monte Carlo Simulation

Monte Carlo simulation is a type of computational algorithm that uses repeated random sampling to estimate the possible outcomes of an uncertain event. This Method was invented by John von Neumann and Stanislaw Ulam during World War II to improve decision making under uncertain conditions. It is one of the oldest and most widely used statistical procedures.

In Monte Carlo computing, a random number generator is repeatedly called which returns a real number within a set range, and the results are used to generate a distribution of samples that gives a representation of the target probability distribution [45].

An example for a Monte Carlo simulation in a game of dice at the casino can be seen in figure 2.9. The first figure shows 500 outcomes from a game after 30 dice rolls. The second figure 2.10 shows after throwing 800 dices. The game simulated is simple, a player starts out with a balance of 1000$. The player rolls a 6 sided dice and the house rolls another one. The player has to give 1$ for each round but if he wins, the house pays out 4$. In the first simulation, it is not obvious if the the casinos business model is working. After many rounds with a game of dice however, one can clearly see that the house always wins.

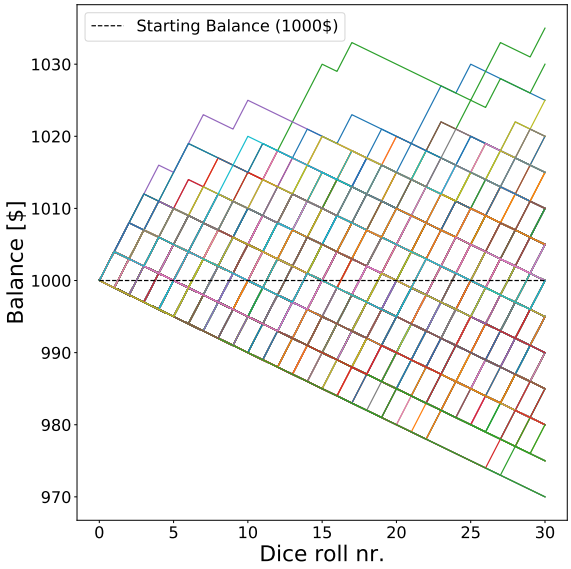

Figure 2.9: 500 players with an ave. balance of 995$ after 30 dice rolls.

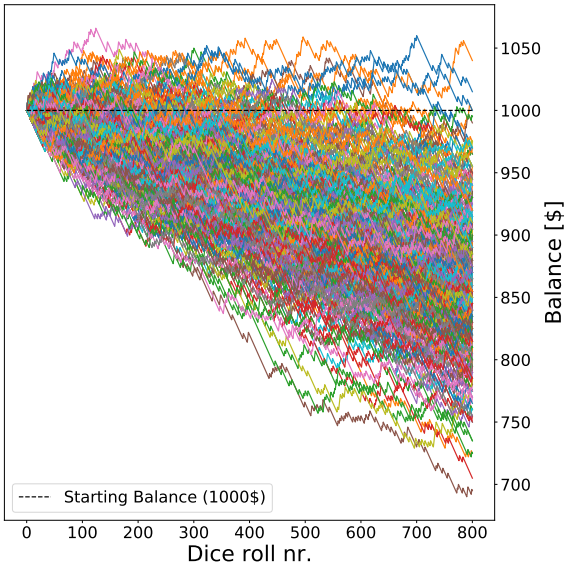

Figure 2.10: 500 players with an ave. balance of 862$ after 800 dice rolls.

3 Tools and Framework

3.1 GymFC

GymFC is a tool for flight control development, introduced in [46]. It is a generic tool for flight controller development, not necessarily for Neural Network-based flight but also traditional controllers like PID. To synthesize a good flight controller, each controller must be trained for its unique digital twin.

3.1.1 PX4 Gazebo SITL motor model plugin

The PX4 Gazebo motor model plugin is a flight simulator for multirotors, VTOL, fixed wing and more. It is based on the RotorS simulator, but in contrast to RotorS, PX4 is not dependent on Robot Operating System (ROS).

The framework is designed to enable a quick start to perform research on MAVs. The simulator includes control and state estimation and is designed in a modular way, such that different controllers and state estimators can be used. It provides controllers which can be adapted to a custom vehicle by only changing a parameter file [47].

The simulator was developed with an emphasis on keeping the structure as close as possible to the real system. This is to make the transition of the code from the simulator to the code running on the actual hardware as simple as possible.

The forces and moments acting on the MAV is split up into forces and moments acting on each rotor, and the gravitational force acting on the center of gravity of an MAV. The forces modeled on each rotor in this simulator, is the thrust force F_T, drag force F_D, rolling moment M_R, and the moment originating from the drag of a rotor blade M_D.

$$F_T = \omega^2 C_T \cdot e_{z_B} \tag{3.1}$$

$$F_D = -\omega C_D \cdot v_A^{\perp} \tag{3.2}$$

$$M_R = \omega C_R \cdot v_A^{\perp} \tag{3.3}$$

$$M_D = -\epsilon C_M \cdot F_T \tag{3.4}$$

where ω is the positive angular velocity of the rotor blade, C_T is the thrust constant, C_D is the drag constant, C_R is the rolling moment constant and C_M is moment constant of the rotor. ϵ tells the direction which the rotor is turning (1 for counter clockwise and -1 for clockwise). e_{Z_B} is the unit vector in the z-direction. $\mathbf{v}^{\perp}$ denotes the projection of a vector $\mathbf{v}$ onto the rotor plane.

3.1.2 Simulation environment

The environment in GymFC is different from other simulation-environments because it uses a Digital Twin (DT), opposed to simulating only the important parts of the agent, GymFC simulates the whole system. Everything from the electrical noise from IMU inside the flight controller, to the ramp-up speed of the propellers after it has gotten its PWM signals. The simulation environment is specifically built for tuning of flight controllers. It supports attitude control tuning and motor modelling. Upon launch, the environment reads the XML-file which includes the configuration for the aircraft. The environment then dynamically loads the aircraft model into the simulator. From here, it waits on motor control messages from the step_sim function. A visual representation of GymFC's architecture for training a model with RL is shown in figure 3.1. The step function calls four functions: *transform_input*, *transform_output*, *generate_command* and *compute_reward*. The first two functions support transforming the aircraft state to the input of the neural network and from the network to the control signal. *Generate_command* function generates the angular velocity setpoints, that the agent must achieve. The *compute_reward* function calcu-

lates the reward at each time step for the agent.

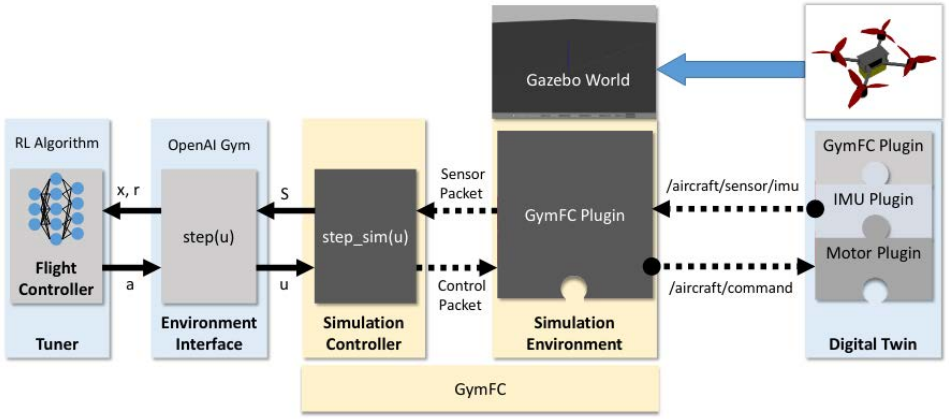

Figure 3.1: GymFC architecture for RL-based flight controller (taken from [46])

3.1.3 Reward functions

GymFC has some reward-functions developed that the user can choose to use or build upon, for their fitness-function. The basic reward function is:

$$r = r_e + r_y + r_\Delta \tag{3.5}$$

where r_y is the reward for minimizing the control output, and r_Δ is the reward for minimizing oscillations and r_e is:

$$r_e = -(e_\phi^2 + e_\theta^2 + e_\psi^2) \tag{3.6}$$

where e_ϕ is the roll error on each axis.

The next functions are present to help the agent learn how to fly more stable and efficiently. These reward functions are:

Output Oscillation penalty:

- A penalty is given for high changes in control output.

  $r_{\Delta y} = \beta \sum_{i=0}^{N-1} max\{0, \Delta y_{max} - (\Delta y_i)^2\}$

Minimizing signal output reward:

- A reward is given for reducing their average control signal output if they are in the error band, defined by the percent of the target angular velocity. This reward helps with minimizing error and output, which reduces oscillations. This is the main source of reward.

  $r_y = \alpha(1 - \bar{y})$

  where $\bar{y}$ is the average output $\bar{y} \in [0, 1]$ and α is a scaling constant.

Change in error:

- Reward is given when error is reduced, while a penalty is given if the error increases from the last timestep. r_e is from equation 3.6.

  $r_{\Delta e} = r_{e_t} - r_{e_{t-1}}$

Over saturation penalty:

- Penalizes the agent if they have saturated the control outputs.

  $r_s = -\beta \left[\sum_{i=1}^{4} y_i\right]$

  where r_s is denotation for saturation, y_i are motor outputs with values above 1 and β is a scaling constant.

Doing nothing penalty:

- If more than two motor-outputs are zero and the target angular velocity on all axes is not zero, a penalty is applied.

  $$r_{y,null} = \begin{cases} -\beta, & \text{if } \left[\sum_{i=1}^{4} y_{i,null} >= 2\right] \text{ AND setpoint} \neq 0 \\ 0, & \text{otherwise} \end{cases}$$

  where β is a scaling constant, $y_{i,null}$ is motor output with value = 0.

All rewards and penalties are summed up and returned to the neural network for each timestep.

3.1.4 OpenAI Baselines

GymFC uses OpenAI Baselines for its RL framework. Baselines is a set of high-quality implementations of reinforcement learning algorithms [14]. The algorithms are meant for the research community, to further identify new ideas and to build research on top of. The Baselines-version used in this thesis was last updated in July 2018. Newer versions are available, but GymFC is specifically made to work with this version of Baselines.

OpenAI Baselines were open-sourced in May of 2017. The initial release included Deep Q-Network, a RL algorithm which combines Q-learning and DNN. Resulting in a algorithm that can solve complex tasks, such as video games and robotics. One of the the algorithms available in Baselines is Proximal Policy Optimization (PPO), which is the one used in this thesis.

3.2 Computer-aided Design

CAD is a technology for design and technical documentation. With this, it's possible to build an entire model in an imaginary space, letting you view and measure properties like size, material, mass and more. Fusion360 by Autodesk is a cloud-based CAD software for product design and production. With plenty of features such as 3D design, electronics, generative design and simulation. It is a popular CAD tool with a free version for students [48]. Fusion360 was used in this thesis for modeling the vehicles components and measurement of the inertial matrices. Extracting the mass properties for a motor is seen in figure 3.2.

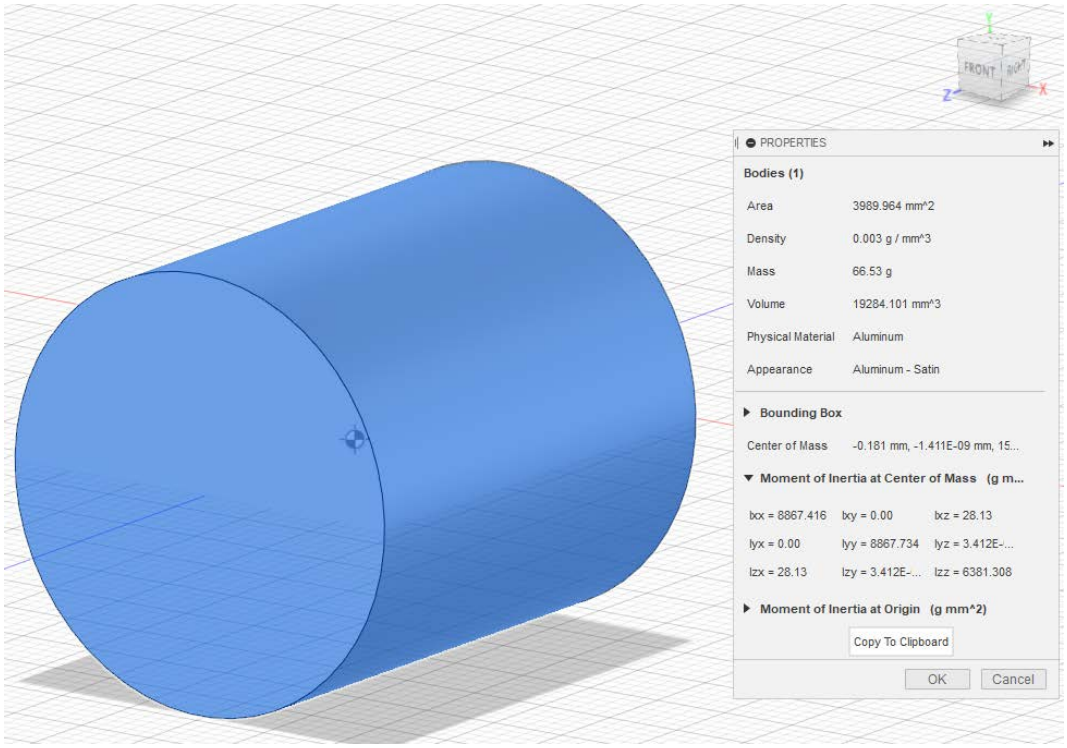

Figure 3.2: Shows a simple model of a motor in CAD. Center of gravity is seen in the center, and the properties with mass and moment of inertia, is seen on the right.

3.3 Statistical analysis and visualization

Seaborn is a library for making statistical graphics in python [49]. It is built upon Matplotlib and integrates closely with pandas data structures. Seaborn is designed to be useful throughout the lifecycle of a scientific project. The library offers an interface to matplotlib that gives rapid data exploration and prototyping of different plots, while having the necessaries for producing publication-quality figures.

To visualize the data from this thesis, Boxplots and "Rain Cloud Plots" are used with the help of Seaborn. The latter is a combination of "Violin plots", "Jittered data points" and Boxplots. This gives an overview of the raw data, the probability distribution, and statistical inference. The vilolin plot, which in this case is split in half to give more room (and makes it look like a cloud), shows the density of the data points. The jittered data points is the raw data (rain), and lastly we have the boxplots [50]. The boxplot shows a summary of the data including the minimum-value, the

lower quartile, median value, upper quartile and max value.

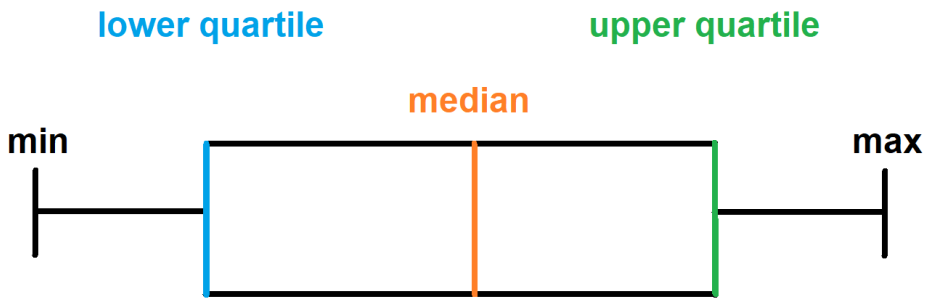

Figure 3.3: The box plot

- Minimum value
  - The lowest value, excluding outliers
- Lower Quartile
  - 25% of scores fall below the lower quartile value
- Median
  - Marks the mid-point of the data. Half of the points are greater than or equal to this value and half are less
- Upper Quartile
  - 75% of the scores fall below the upper quartile.
- Max value
  - The highest value, excluding outliers
- Whiskers
  - The upper and lower whiskers represent values outside the middle 50%.

Another statistical analysis used is Mann Whitney U test, also called Wilcoxon Rank Sum test. It's a technique for testing the equality of mean in independent samples. A necessary assumption when using this test, is

that the continuous outcome was approximately normally distributed or the samples were sufficiently large [51]. The Mann Whitney U test is used to test the probability of two samples have been derived from the same population. We have two hypotheses:

H_0: The two populations are equal.

H_1: The two populations are not equal.

Since we will use multiple comparisons, we should use the Bonferroni method when choosing significance level. This method says to lower the significance when there are multiple comparisons [52]. In a multiple hypothesis testing, an increased number of samples, n, in a given family increases the probability that false positives will arise. Therefor the threshold α is lowered by the equation α/n.

4 Method and Implementation

This chapter explains the method for modeling a quadrotor for the GymFC environment. The changes to the training of the Reinforcement Learning. The methods for the tuning of PID gains with Ziegler Nichols and Evolutionary Algorithm is also explained, and how the Monte Carlo test was implemented.

4.1 Motor-modeling

Modeling the behaviour of the aircraft usually starts by figuring out the characteristics of the motors. How much thrust, torque and how they scale with RPM. To calculate this, one would need to find data of the motor/propeller combo that is being used. The only way to obtain this data is to use a testbench where one can measure the thrust and torque at different speeds. Fortunately there are researchers that has already obtained this data for the most common motor/propeller combos on the market and created a database [53].

GymFC uses PX4 Gazebo Software In The Loop (SITL) motor model plugins. This motor model need the following parameters for the aircraft configuration:

- Thrust constant (motor constant), K_T
- Torque constant, K_Q
- Motor response

We need to find the dimensionless parameters to be able to simulate the varying forces and moments of the quadrotor. Propellers are commonly designated by two numbers $D_{in} \times p_{in}$, where D_{in} is the diameter in inches

and p_{in} is the pitch in inches. The pitch angle β is the angle between the zero-lift of the propeller blade section and the plane of rotation. $p = \pi D tan\beta$. We use the database to look up the needed coefficients for the motor/propeller combo used in this thesis.

- $C_T = T/\rho n^2 D^4$, thrust coefficient
- $C_Q = Q/\rho n^2 D^5$, torque coefficient
- $C_P = P/\rho n^3 D^5$, power coefficient

With these we can calculate the needed parameters. The power and torque coefficients are related by the equation $C_Q = C_P/2\pi$,

Proof:

$$P = Q\omega = Q(2\pi n)$$

$$C_Q = \frac{Q}{\rho n^2 D^5}\left(\frac{n}{n}\right) = \frac{P}{2\pi\rho n^3 D^5}$$

$$C_Q = \frac{C_P}{2\pi} \tag{4.1}$$

So if we have C_P, we can calculate C_Q:

$$C_Q = \frac{C_P}{2\pi} = \frac{0.0352}{2\pi} = 5.61 \cdot 10^{-3} \tag{4.2}$$

Motor Constant:

$$K_T = \frac{C_{T_0}\rho D^4}{(2pi)^2} = \frac{0.098 \cdot 1.2041 \cdot 0.23^4}{(2\pi)^2} = 8.365 \cdot 10^{-6} \tag{4.3}$$

Moment Constant:

$$K_Q = \frac{C_Q}{C_T}D = \frac{0.00561}{0.098}0.254 = 1.40 \cdot 10^{-2} \tag{4.4}$$

The last part needed is the step response. To find this the RPM is plotted over different throttle percentages, to obtain the two degree

polynomial function. A simple way of doing this is making a python-script and using Numpy "PolyFit" function on the motor-data, giving a plot and second degree polynomial of the RPM over PWM output.

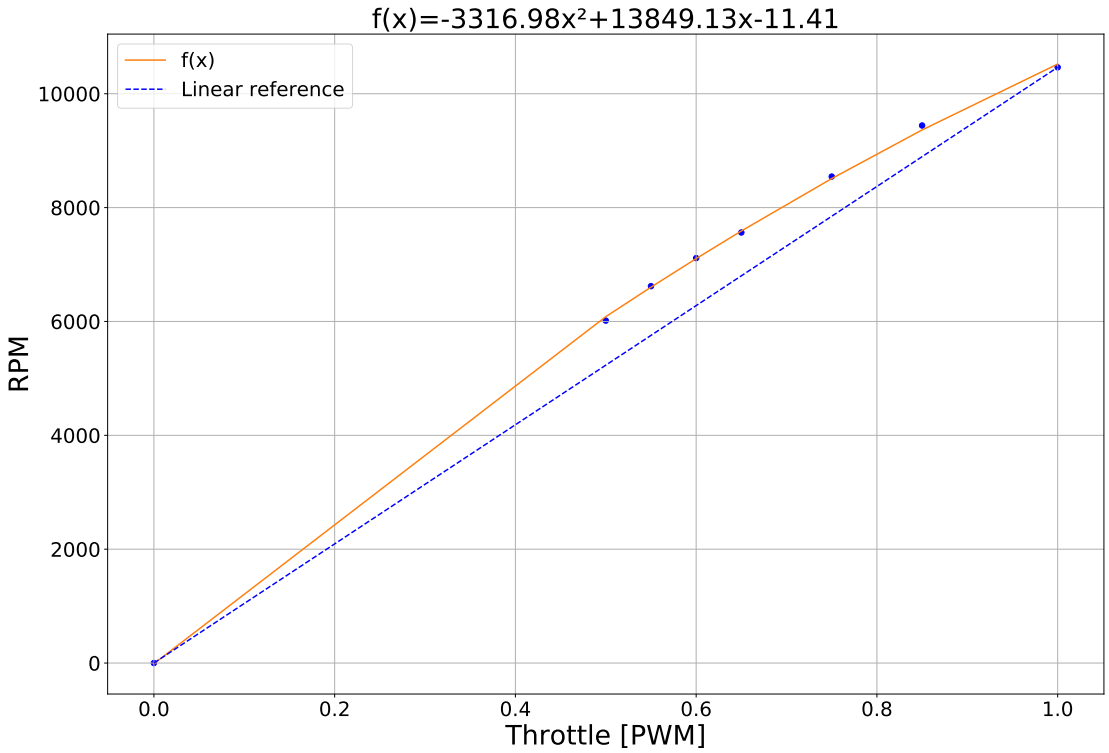

Figure 4.1: Motor Response

When everything is calculated, the values are placed inside the aircraft configuration file. This file follows the gazebo format called *Simulation Descriptive Format* or SDF. Table 4.1 shows the parameters names and their values, that have been obtained. The full XML-code for the sdf-file can be found in the appendix.

Parameter:	Value:
maxRotVelocity	1096
motorConstant	8.365e-6
momentConstant	0.0140
rotorDragCoeff	1.75e-4
term1Coeff	-3316.98
term2Coeff	13849.13
term3Coeff	-11.41

Table 4.1: Motor/Propeller parameters for model.sdf file

4.2 Rigid body modeling

An important part of the behaviour of the aircraft is the inertia. How the object turns in the different directions and the force needed to change attitude. A good approach is by computing the moments of inertia using a computer model of the object. A common way is to assume a uniform mass distribution, but for a quadcopter, most of the mass is located in the battery and in the motors at the end of each arm. To model a more accurate aircraft, we will have do decompose the components of the aircraft and create a rigid body for each. The downside to this is mainly the added work and a higher need for computational power for the simulation.

The way that the aircraft-model in gymFC is set up is by dividing the main components. This being the frame, battery, motors, propellers and flight controller. The user can choose how many components that should be modeled. One would have to either find the CAD files for each part or model it yourselves. Then use the tools in the CAD software to find the inertia-matrix, after choosing correct material and scaling the mass-density, to get the correct weight of each component. The inertia-values and measurements of each component is put into the aircraft configuration file. Table 4.2 shows the parameters for configuring the frame of the aircraft. How this looks like in XML-code for the configuration file, is shown in the appendix.

The S500 quadrotor and 10x45 propeller was found on GrabCad [54] and imported into the CAD tool Fusion360 as seen in figure 4.2, 4.3. From there the real version of the vehicle is dismantled into the main components. Now the weights can be measured individually. In CAD, the correct material is chosen for the parts and the mass density is scaled

Parameter:	Value (kg):
mass	0.514
Ixx	3.260e-3
Ixy	0.000
Ixz	1.819e-6
Iyy	3.632e-3
Iyz	2.050e-7
Izz	5.573e-3
Position:	Value (m):
x	0.000
y	0.000
z	0.039

Table 4.2: Frame inertia

Parameter:	Value (kg):
mass	0.490
Ixx	1.501e-4
Ixy	0.000
Ixz	0.000
Iyy	8.416e-04
Iyz	0.000
Izz	8.730e-04
Position:	Value (m):
x	0.000
y	0.000
z	0.000

Table 4.3: Battery inertia

according to the measured weight. Lastly the inertial matrix of the main components were extracted. One can place the origin in the Center of Mass (COM) of the model, or specify where the COM is relative to the center of the model, inside the aircraft configuration file. The battery and motors were modeled by myself in CAD, as they have a very simple shape. Their inertia-parameters can be seen in table 4.3 and 4.5.

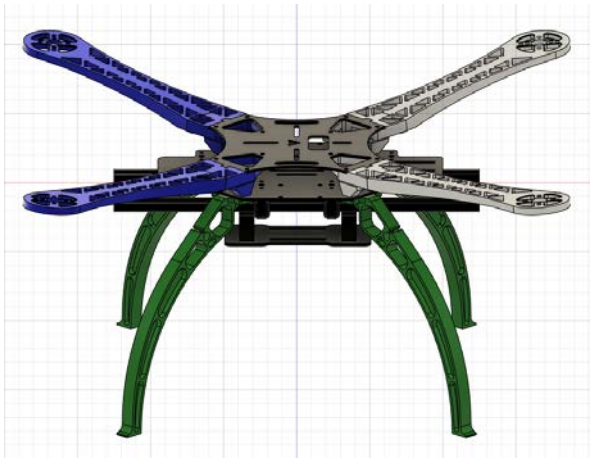

Figure 4.2: S500 Frame

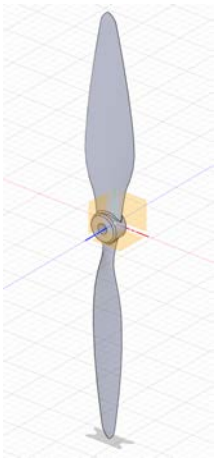

Figure 4.3: Propeller

When every object is modeled, their position has to be specified in the model.sdf file. This was done by using the measuring-tool in CAD and on the real quad, to make sure everything is correct. The position of the individual parts are measured from the center of thrust on the aircraft, which is located in the center of the battery. Since our aircraft is a mirrored

Parameter:	Value (kg):
mass	0.012
Ixx	4.0426-5
Ixy	5.1814e-7
Ixz	0.000
Iyy	3.3892e-7
Iyz	0.000
Izz	4.0665e-5
Position:	Value (m):
x	-0.16500
y	-0.16500
z	0.12640

Table 4.4: Propeller nr. 1 inertia

Parameter:	Value (kg):
mass	0.066
Ixx	8.8677e-06
Ixy	0.000
Ixz	0.000
Iyy	8.8677e-06
Iyz	0.000
Izz	6.3810e-06
Position:	Value (m):
x	-0.16500
y	-0.16500
z	0.12465

Table 4.5: Motor nr. 1 inertia

X-frame, the placement of each motor is symmetric on each side of the x and y-axis. The XML code for the model.sdf file can be found in the appendix.

4.2.1 Airframe

The airframe of choice is the open source "S500" airframe. The specific airframe in this thesis is made out of plastic, although a carbonfiber version is also available. The airframe is a symmetrical x-frame quadrotor as shown in figure 2.5 with a wheelbase of 480mm and a weight of 935g without battery. Recommended specs are 880kv motors paired with a 4S 5000mAh LiPo battery, this configuration can hold a payload of up to 1.8kg [55]. Choosing this frame came down to the design being open source, making it easy to find a 3D model online and import it into the Computer-Aided Design software for measuring and modeling. The size is also attractive, as it isn't overly compact which makes assembly, troubleshooting and placing of components much more forgiving, if chosen to put into the real world.

4.2.2 Mixing

Mixer configurations determine how the motors work together to control the aircraft. The mixer used in the tuning platform of GymFC is ported from BetaFlight, so to figure out what mixing configuration to use, we

Position	Active	Roll	Pitch	Direction
FL	1	1	-1	-1 (CW)
FR	1	-1	-1	1 (CCW)
RR	1	-1	1	-1 (CW)
RL	1	1	1	1 (CCW)

Table 4.6: Motor Mixing

look up their description on their GitHub Repository [56]. The vehicle modeled in this thesis is a symmetrical X-frame, resulting in the mixing seen in table 4.6.

The way the mixing-values are calculated is by measuring the distance from motors to roll and pitch axis. We start with motor nr.1 (Front left) and measure to the roll axis. This is the distance marked as *C* in figure 4.4 and measures 161mm. *A* measures also 161mm, which is the distance to the pitch axis. The longest of the two distances is then set as denominator, so the results are within the range of $[-1.0, 1.0]$. This is not something we need to worry about since this is a symmetrical x-frame where B = D, B = 2A and A = C. The roll mixing for front left is calculated by $161/161 = 1.0$ and pitch as $-161/161 = -1.0$. The other motors have the same calculations except for the +/- sign [57].

From table 4.6, the first motor is the front left. We tell the flight controller that the motor is in use by giving it a non-zero value *1.0*, then we set roll axis to be positive *1.0*, pitch axis is negative *-1.0* and the motor turns clockwise *-1.0*. When training with RL, the agent learns the mixing itself. So configuration of the mixing is only used for the PID controller.

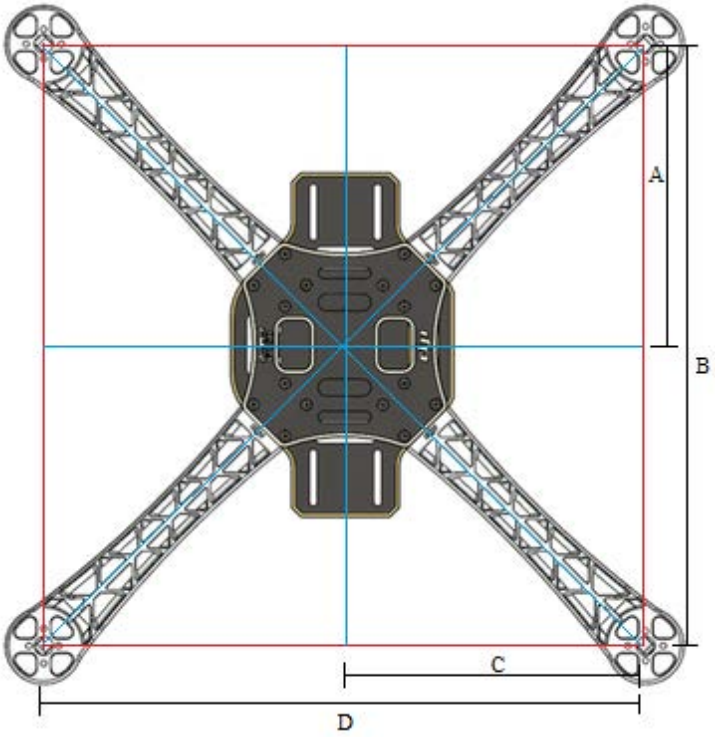

Figure 4.4: Top view of S500 x-frame

4.3 Training

The GymFC repository has a finished script one can use to train a model with Proximal Policy Optimization in the Gazebo Environment. The only change one have to make is to change the string that points to the folder for the new digital twin if desired. If left standard, the example drone called "NF1" will be used for the simulation.

This script works well out of the box on the example vehicle, without any modifications. The rewards provided are tuned after a 250mm wingspan racing drone, training with the same reward function resulted in a very overdamped system on the S500 quadrotor. The characteristics of an overdamped system is a high rise time, defined as the time it takes to reach the desired value, it also never reaches the desired value, but settles below. An example of an overdamped system is shown in figure 4.5. One of the reasons for this was the very aggressive penalty for oscillations and ESC-output in the reward-function. Since these penalties were so excessive, the agent would settle below the desired velocity. Since the

rewards are optimized to be as smooth as possible, the agent would rather loose some reward to velocity-error, than receiving the penalty for further adjusting the motor velocities. After decreasing the penalty for oscillations and motor-output, the agent would occasionally become underdamped. This was compensated by implementing an overshoot-penalty.

ESC noise was also simulated while training. The sample noise was taken from the example-vehicle in the GymFC repository. While it does not use the same Flight Controller, it is still based on the same STM32F7 microcontroller with ARM Cortex M7 core. They also share the same IMU, so it was assumed that the FC provided with the S500 would have similar noise. The values used for noise on each axis is shown in table 4.7.

Axis:	Mean:	Variance:
Roll (ϕ)	-0.2546	1.3373
Pitch (θ)	0.2419	0.9990
Yaw (ψ)	0.079	1.4516

Table 4.7: Gyro noise (deg/s)

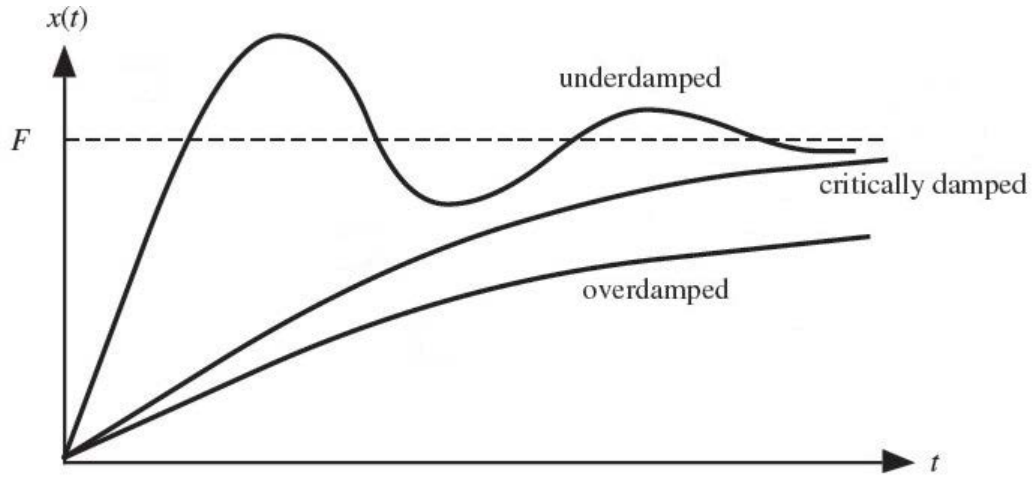

Figure 4.5: Showing different responses

In the training environment, the gravity was disabled focusing on the attitude of the vehicle. The agent would learn how to idle by starting every episode with a desired angular velocity of 0 for a given period. There is also set a minimal idle speed into the mixing, which should make the net force greater or equal to zero if there was gravity present. After the vehicle

has settled, the desired angular velocity is given for each axis. The vehicle will rotate about its center of mass for a given period, before settling back to idle. The results for this method of training for attitude control, was tested on real life quad vehicle and confirmed to be effective [46].

4.4 Software

Several scripts were made to be able to create plots from the tests, response-comparisons etc. but the new features that were implemented, were an Evolutionary Algorithm for optimizing of PID-gains and a Monte Carlo simulation for testing the controllers. The EA optimization script assists for an unbiased comparison of the potential of the RL-trained model versus the PID-control system, since they are following the same reward function. The two systems are tuned after a reward-function with its preference of what a good response looks like. The advantage to this is that one will get an objective comparison between the two. Since the human factor of tuning a control system is eliminated. The "agent" that control both systems has the same intuition of what a good response looks like, because of the shared reward function. The disadvantage is that the performance solely depends of how well the reward-system is tuned. A poorly tuned reward-system will result in a poorly tuned control-system.

The Monte Carlo simulation is another method to get an objective answer to how well each system behaves. It does not tell which system is the fastest but it tells which had the least mean error to the desired velocity, on each axis. By testing a large range of setpoints, one can study the reliability of the controller. The control systems behave differently to low and high velocities, so by looking at the spread of the results, one can see how often a system would respond poorly and get an idea of its overall accuracy.

4.5 Reward-function

The reward-function in use for the RL flight controller and Evolutionary Algorithm is built up by several functions. The main reward comes from the agent accurately following the desired angular velocity by changing

the motor-velocities. The inputs of the NN are the current and desired angular velocities in 3 axis (roll, pitch yaw), while the outputs are the ESC-signals that power the motors (4 outputs, one for each motor). But the agent needs a more complex reward-system to learn how to behave well in flight. One would also need to teach the agent how to responds smoothly and minimize overshoot and oscillations.

The reward is built up by the following functions:

- Output oscillation penalty
- Minimizing signal output
- Change in Error
- Over saturation penalty
- Doing nothing penalty
- Overshoot penalty

The overshoot penalty is the only part of the reward, that were not implemented. The rest was already available in the GymFC repository, although the scaling of reward/penalty for each of them were tuned after several training-sessions, to obtain a better response behaviour. The overshoot penalty works by giving a penalty for each axis experiencing overshoot, which is a set amount of degrees above the desired angular velocity. The penalty increases by how many of the axis that are experiencing overshoot simultaneously.

Overshoot penalty:

- Penalizes the agent for going above the desired velocity, if the setpoint on all axes is not zero. The penalty increases by the number of axis with overshoot.

  $$r_O = \begin{cases} -\beta\left((O_\phi + O_\theta + O_\psi)/3\right), & \text{if setpoint} \neq 0 \\ 0, & \text{otherwise} \end{cases}$$

  where O is an axis experiencing overshoot $O \in [0, 1]$ and β is a scaling constant.

4.6 Ziegler Nichols optimized PID

To evaluate the performance of the RL-trained Flight Controller (FC), a well known control design is chosen, the Proportional Integral Derivative (PID). This control system has great potential as long as it's tuned correctly.

GymFC can be used as a tuning platform for the PID-parameters, the architecture of the tuning platform is shown in figure 4.6.

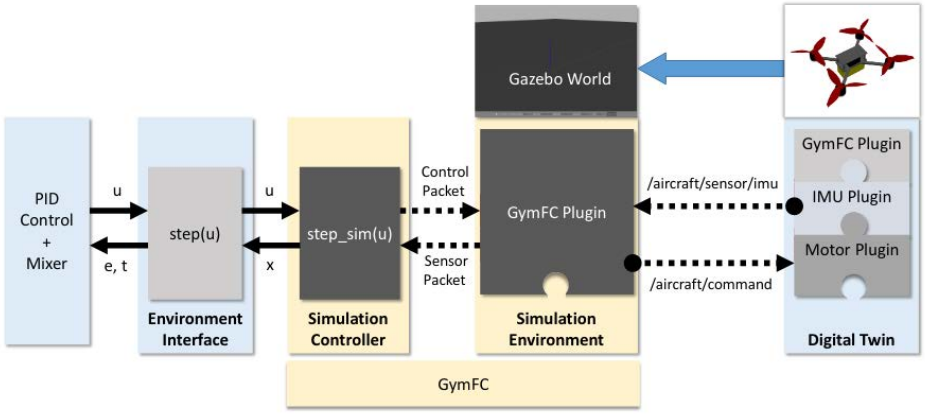

Figure 4.6: GymFC PID control tuning and SITL (taken from [46])

The first PID controller to be used for comparison, is tuned after the classical Ziegler Nichols method. This is a well known and purely mathematical approach to automatic control. Although this method is known for not providing the greatest results, it is used as a baseline. The calculations are executed for each axis independently, but since the quadrotor in use is a symmetrical X-frame, we can use the same gains for roll and pitch.

4.7 Optimizing PID-values with Evolutionary Algorithm

Another method for finding PID gain values, is by introducing optimization algorithms. For example an Evolutionary Algorithm (EA). A biologically inspired algorithm that mimics a population that slowly adapts to the environment, by producing offspring with slightly mutated properties from the previous best individuals, until the population has become well

fitted to the environment. An individual in this context, would be a set of PID-gain values. The algorithm starts out with an all random population. The top individuals with the highest score, meaning the gain-values that had the best response in a PID-control test, is picked as the parents of the next generation. A determined amount of offspring is made from the best individuals of the last generation, but slightly mutated in hope that these will be better than the last generation.

There are many ways of setting up an EA, but the one used here starts with a population of 32 with randomly generated sets of PID gain values ranging [2,70]. These values where chosen from simulation, where one could see that the lowest Proportional-value that could make the drone to change velocities in a reasonable way. The sets have 9 values, 3 PID-gains for each axis $[\phi, \theta, \psi]$.

$$\left[P_\phi I_\phi D_\phi, P_\theta I_\theta D_\theta, P_\psi I_\psi D_\psi\right]$$

After each individual is tested and calculated its fitness-value from the reward-function, the top 3 individuals are picked for the next generation. The top 2 are the "elites" and are kept until the offspring gets a better fitness. This is to make sure that the fitness always stays equal or better from one generation to the next. This can be seen in figure 5.2. The next generation contains 7 offspring from each of the previous top 3. The gain-values are mutated by by adding a random float-value in the range [-2,2], to each gain parameter. The last 9 individuals of the new generation are made with recombination of the attributes from the top 3 parents. The recombination showed great ability in increasing the exploration of the algorithm. As the mutation with float-values alone, did not add enough exploration to leave most of the local maximums.

- 2 Elites
- 21 mutated offspring
- 9 recombination offspring
- Results in a population of 32

The fitness of the individual is calculated from the same fitness function used for RL. The sum from the reward-functions is sampled

from each time step in the simulation, same as for RL-training and an average reward is calculated. The fitness of the best agents found through a number of training sessions is plotted in figure 5.2.These parameters for the EA where found through many tests, studying how often the algorithm would converge and find great results.

Algorithm 3 GymFC EA tuning algorithm

INITIALIZE 32 agents with 3x3 random PID-values
while (TERMINATION CONDITION is not satisfied) **do**
    1 **SIMULATE** agents on specified inputs
    2 **PICK** the 3 best agents
    if (No candidate is better then the best parent) **then**
        Keep best parent for next generation (Elitism)
    end if
    3 **LOG** the current best agent
    4 **MUTATE** 30 new candidates from the top 3
    for (21 agents) **do return** Agents by mutating PID +/- 2.0
    end for
    for (9 agents) **do return** Agents by Recombining PID values from top 3
    end for
end while
return best PID-values with reward and mean error

4.8 Monte Carlo Simulation

To get a good idea how well the different control systems responds, a Monte Carlo simulation was made. 200 sets of random inputs is created that later is tested on the PID control systems and the RL model. The list of desired values that the systems is tested on are identical, meaning the random-function that generated the setpoints uses the same seed. The test calculates the mean error that the episode had for each setpoint. The error is measured on each axis independently. This gives a good measure of how well each control system behaves on all three axis with a variety of angular velocities. The test-environment is the same environment as the training took place, only this time it uses setpoints that it has not seen be-

fore.

The test starts out with an angular velocity of zero, this is to show how well the vehicle can idle. Random setpoints as angular velocities are then given on each axis at the same time. This is a worst case scenario for the quadrotor, as there is no ramp-up period, and the fact that they are given for all three axis simultaneously. The setpoint is then changed back to zero, after the vehicle has held the desired velocity for a set period. A plot of how the desired and actual velocities respond to each other in the environment.

Algorithm 4 GymFC Monte Carlo simulation

GENERATE 200 random sets of setpoints for roll, pitch and yaw
CHOOSE controller for testing

for 200 number of tests **do**

1 **SIMULATE** roll, pitch and yaw control with random setpoints

2 **LOG** error for each axis from episode

end for
return Data from 200 tests

5 Experiments and Results

5.1 Training

In this chapter, we go through the process of training a RL-model, using the PPO algorithm on the Digital Twin (DT) that were modeled for the S500 quadrotor. An approach for acquiring a well performing model and comparing its performance with other controllers, is shown. We validate the results and get an understanding of the behaviour of the RL-controller. Lastly, the different controllers are tested with the Monte Carlo simulation.

Due to the nature of randomness in RL, one does not always get a good resulting controller nor does it always converge. When testing changes in the reward-function, it was made sure to get at least 3 models trained for $3.0 \cdot 10^6$ time steps, to validate the behaviour of the reward-function before concluding if the changes where beneficial or not. A time step takes place every 1ms in the simulation, if it runs in real time. One episode is one simulation of the vehicle, where the vehicle starts out with a angular velocity of 0 on all axes, then gets a set of desired inputs and goes back to idle. An episode consists of 4608 time steps.

Each model is validated by running 3 different inputs-sets of rotational velocities on each axis and receiving a reward and mean error for each episode. This is done on every checkpoint. A checkpoint is made on every $3.0 \cdot 10^3$ time step, resulting in 100 checkpoints. After the evaluation, one can plot response-graphs from the checkpoints that received the largest reward, to visually see if the behaviour is adequate. If a specific checkpoint shows great response, it can be extracted and further tested in a Monte Carlo simulation. This is to further validate its performance on a broader variety of inputs that the model has never seen before. The results from

the Monte Carlo simulation is what gives the box plots its data, which is the chosen method to determine which model had the best performance.

PPO is known for not needing that much tuning of the hyperparameters [42]. The default settings were able to train a working model on the NF1 quadrotor given as example in the GymFC repository, so the hyperparameters were left as is and rather spend time on further optimizing the reward-function for the quadrotor used in this thesis. A training session on the S500 quadrotor is shown in figure 5.1. The plot is showing 100 checkpoints from $4.0 \cdot 10^6$ time steps (868 episodes). The figure shows that the model starts to converge after approximately $1.0 \cdot 10^6$ time steps. The other metrics show the mean error, mean output to the motors, mean delta output and the reward at each checkpoint.

At first, the agent is not responding to the inputs, as shown by the flat metrics of the motor output u. The *doing-nothing-penalty* makes sure that any output to the motors, gives a significantly better reward than no outputs at all. As time goes on, the agent starts to use its motors very frequently. This also increases the mean error, since the vehicle is at this point just spinning out of control. The minimal output penalty and oscillation penalty makes sure that the agent learns to minimize its outputs, to get a more stable flight. By the time of about $2.0 \cdot 10^6$ time steps, the change in motor output declines, which coincides with the increasing reward. The hyperparameters used are shown in table 5.1. The best checkpoint was extracted at approximately $2.8 \cdot 10^6$ timesteps and is labeled as *PPO 0* in the response graph of figure 5.5 and box plots. Training was conducted on a desktop computer running Ubuntu 18.04 with an I5 4670K at 4GHz CPU and an Nvidia GTX 780 GPU.

Parameter:	Value:
step size	1e-4
horizon	512
clip	0.2
lambda	0.95
batchsize	64
epochs	5
gamma	0.99
hidden size	32
hidden layers	2
action space	4
observ. space	6

Table 5.1: Training Parameters

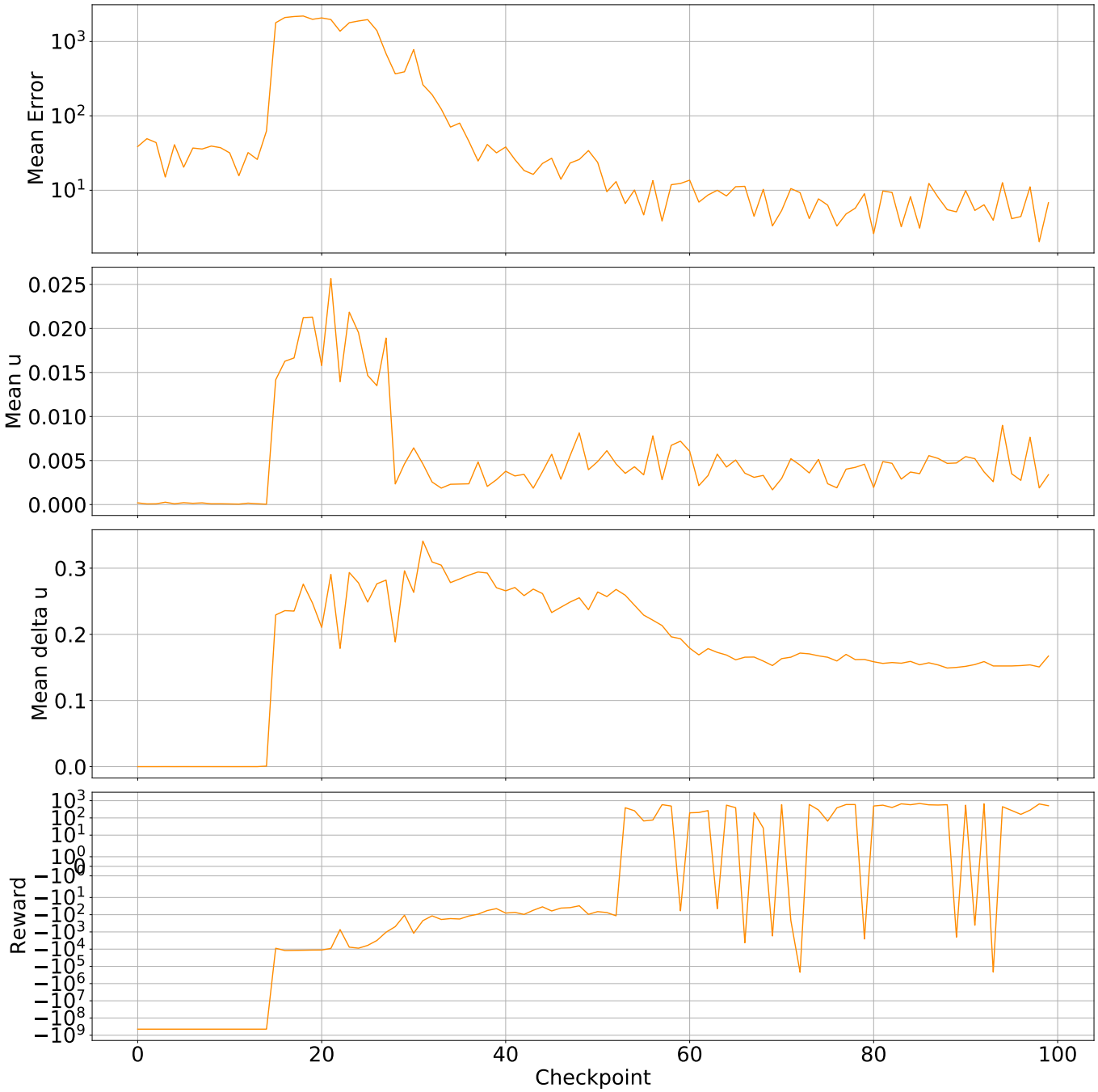

Figure 5.1: Training session with S500 quadrotor

5.2 Comparison and Evaluation

To find out how Reinforcement Learning (RL) is performing on drone-control, we will compare it to the well known Proportional Integral Derivative (PID) controller. First the results and the performance of the PID controller, where the gain values have been tuned using EA. The second PID controller will be with the classical Ziegler Nichols method.

5.2.1 Evolutionary Algorithm optimized PID

The Evolutionary Algorithm (EA) optimized PID controller, uses the same fitness-function as in RL. This makes the comparisons easier, as they value the same type of behaviour, which is low oscillations, minimal and smooth motor-outputs and good tracking of the desired velocity. The best PID-values that the algorithm could acquire were:

- Roll $[P_\phi, I_\phi, D_\phi]$ = [15.84465486, 12.73278438, 0.84717646]
- Pitch $[P_\theta, I_\theta, D_\theta]$ = [40.37873658, 1.12492009, 0.79367289]
- Yaw $[P_\psi, I_\psi, D_\psi]$ = [47.64914102, 4.69741626, 1.46417112]

Response graph of the best individual can be seen in figure 5.5 together with ZN PID and PPO. Data from the EA training session with the best results can be seen in figure 5.2. The result from the EA-algorithm shows a controller that outperforms the ZN method on all axes. Even though the Monte Carlo test shows that it has some issues with tracking the desired velocity for roll-control, with a slightly higher median than the ZN-method. The spread however is much less on all axes, shown by tighter spacing between the lower and upper quartile of the boxplot. This represents 50% of the datapoints, while the whiskers represent the upper and lower 25%. Overshoot is also almost eliminated, while still being just as fast as the aggressive ZN-controller.

5.2.2 Ziegler Nichols optimized PID

The resulting PID-gains from the calculation for roll, pitch and yaw with the ZN method were:

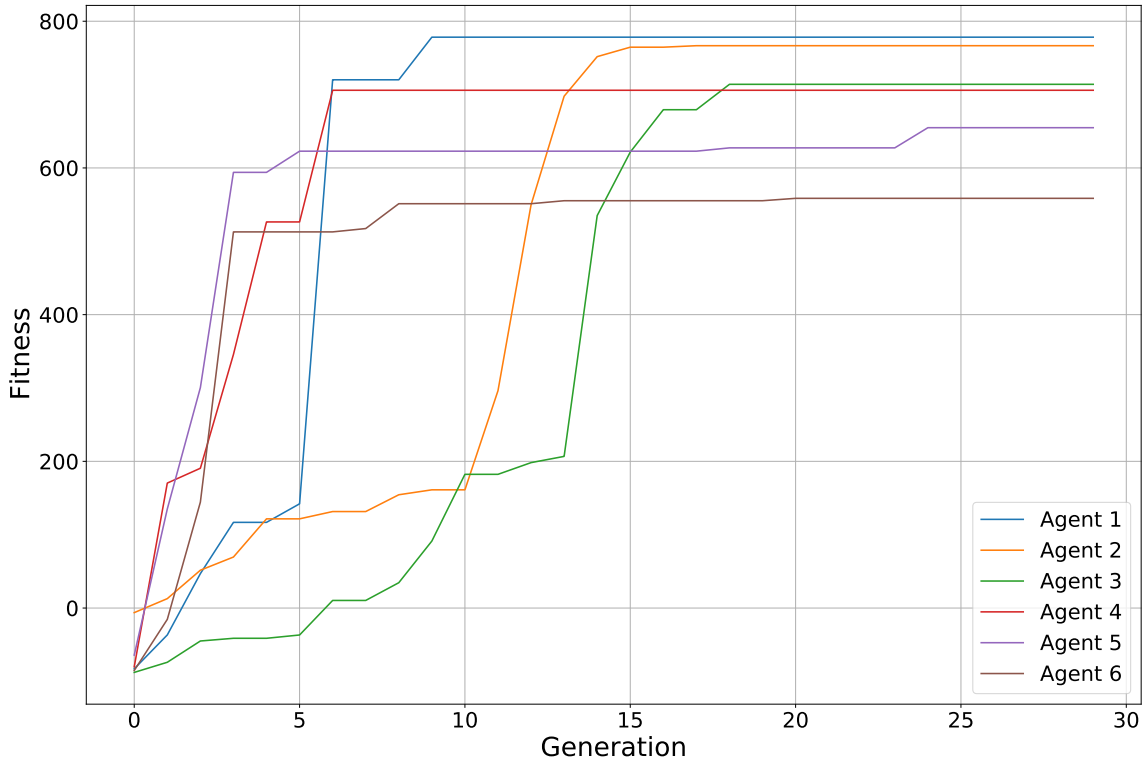

Figure 5.2: Fitness of the best agents after each generation

- Roll $[P_\phi, I_\phi, D_\phi]$ = [21, 233.3, 0.4725]
- Pitch $[P_\theta, I_\theta, D_\theta]$ = [21, 233.3, 0.4725]
- Yaw $[P_\psi, I_\psi, D_\psi]$ = [40, 1, 1]

Since the vehicle is a symmetrical x-frame, the same values for roll and pitch were used. The ZN controller is performing significantly worse than the controller from the EA-algorithm, but it is very often used as comparison in other work [46] as a baseline. Response of Ziegler-Nichols PID is shown with the other controllers in figure 5.5.

The controller does not show any large oscillations while maintaining the setpoints but suffers from violent overshoots. The results from the Monte Carlo simulation shows that it's performing the best on the yaw axis but the median of the data is still worse than PPO and EA. The lower

Model	Roll error	Pitch error	Yaw error
PPO 0	3.2722	4.6958	5.4102
PPO 1	2.3733	9.6180	3.9309
EA PID	8.3615	4.2250	4.8869
ZN PID	7.3249	7.5597	5.7613

Table 5.2: **Results from Monte Carlo simulation**, showing median error in deg/s. These are the same results as in figure 5.3

whisker of the box plot, is on par with the other models for yaw control. Roll-control is showing slightly lower mean error than the best EA-model, which lacked some performance on this axis, but performed well on pitch and yaw. Data from the Monte Carlo tests can be studied closer in the Rain Cloud Plots in the appendix.

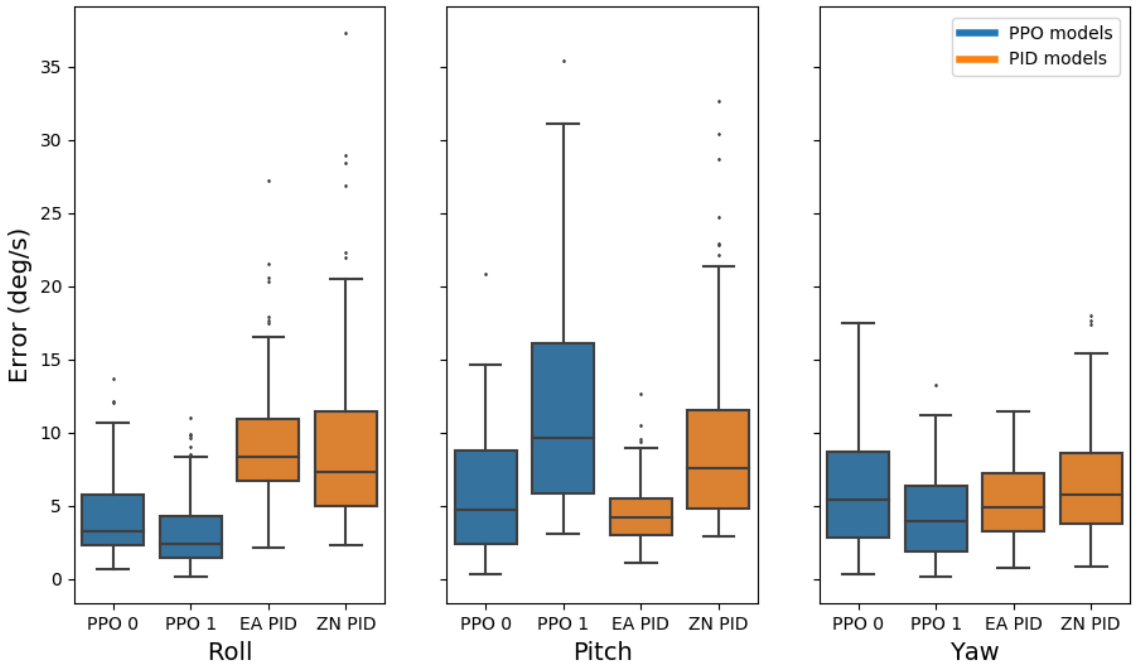

Figure 5.3: Boxplot from Monte Carlo simulation

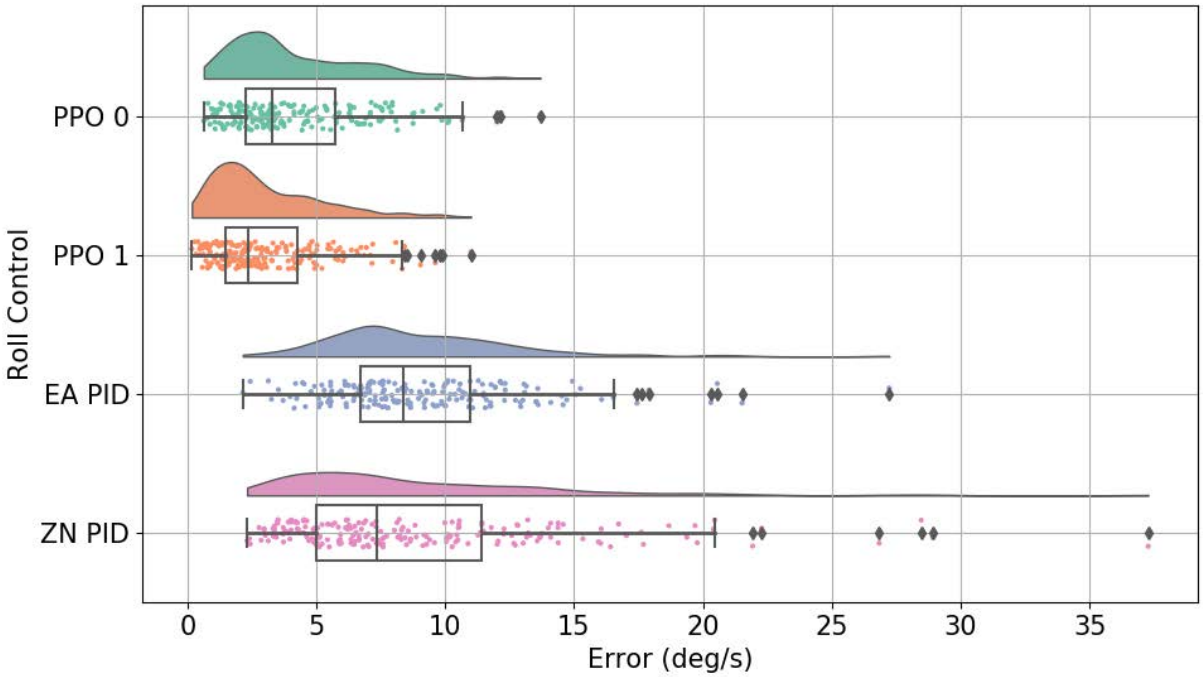

Figure 5.4: Rain Cloud Plot, Roll comparison.

5.2.3 PPO Response and Error comparison

There are two RL models used in these plots. One with the overall best response on all axes labeled as *PPO 0*. A second model that were faster and more precise on some axes but suffered in performance on one axis. This model is labeled as *PPO 1*. Both models can be seen in figure 5.5, which shows their response compared to PID control. The two models are trained with the same hyperparameters and reward-function, only the random seed is different. The results from the Monte Carlo simulation of all the models are shown in figure 5.3 with table 5.2 showing the medians

Model	Roll error	Pitch error	Yaw error
PPO 0	4.94%	4.64%	9.73%
PPO 1	3.27%	15.00%	7.30%
EA PID	7.31%	4.76%	10.03%
ZN PID	9.22%	10.53%	14.70%

Table 5.3: Showing mean error as a % of setpoint from figure 5.5

of the same results.

The *PPO 1* model has a significantly faster response on the roll axis then all of the other models compared. It shows very quick response with close to zero overshoot and oscillations are non present. From the Rain Cloud Plot in figure 5.4, it can seen that a large portion of the datapoints lies in the sub 2 deg/s Mean Error area. However, the response for pitch, seems to be over-damped and has a very slow rise-time. The error for this has a considerably large spread as one can see in the box plot from figure 5.3, where a large amount of the datapoints are in the high-error area of +10 Mean error. Indicating that it experiences instability on a significant portion of the setpoints tested. The response for yaw, seems to be under-damped but tracks the desired velocity sufficiently after settling. This can be seen in the response graph in figure 5.5, with zoomed in versions of the same response graphs in figure 5.6 and 5.7. The zoomed in response graphs also show the mean error for that specific episode, on each axis, for each model. Table 5.3 shows the mean error from the corresponding episode, but as a percentage of the setpoints.

Both PPO models show great results, but the spread is quite high for pitch control compared to the EA PID model. *PPO 0* has datapoints with lower error than both PID controllers on all axes, shown by the lower whisker. *PPO 0* has a slightly higher median, compared to EA PID on pitch and yaw. More in depth representation of the results with rain cloud plots can be seen in the appendix.

5.2.4 Mann Whitney U test (Wilcoxon Rank Sum Test)

A statistical analysis is made on all the models with a Mann Whitney U test. Normally this test runs at a significance level of 5, meaning that if the p-value is larger than 0.05, then we do not have statistically significant evidence, that they are not sampled from the same population. Since this test is comparing multiple samples, we use the *Bonferroni* method when choosing significance level. Therefor the threshold α is lowered by α/n. With $\alpha = 5$ and $n = 6$ we get a significance level of 0.83 for our test. Table 5.4 - 5.6 shows the results from the data on roll, pitch and yaw from the models used in this thesis. The models with a p-value of >0.0083 is colored

	PPO 0	EA PID	ZN PID	PPO 1
PPO 0	1.0	7.924e-40	5.46e-29	6.936e-07
EA PID	7.924e-40	1.0	0.01	3.693e-52
ZN PID	5.46e-29	0.01	1.0	1.473e-42
PPO 1	6.936e-07	3.693e-52	1.473e-42	1.0

Table 5.4: MWU for Roll results

	PPO 0	EA PID	ZN PID	PPO 1
PPO 0	1.0	0.0284	2.917e-12	7.834e-22
EA PID	0.0284	1.0	1.283e-27	1.793e-38
ZN PID	2.917e-12	1.283e-27	1.0	4.327e-05
PPO 1	7.834e-22	1.793e-38	4.327e-05	1.0

Table 5.5: MWU for Pitch results

	PPO 0	EA PID	ZN PID	PPO 1
PPO 0	1.0	0.18	0.0223	2.248e-05
EA PID	0.18	1.0	0.0004	3.25e-05
ZN PID	0.0223	0.0004	1.0	1.716e-11
PPO 1	2.248e-05	3.25e-05	1.716e-11	1.0

Table 5.6: MWU for Yaw results

blue, meaning that the null hypotheses is kept. The data is taken from the Monte Carlo simulation.

The results show that *PPO 0* and *EA PID* are similar for the pitch-axis. Their comparison resulted in a p-value of 0.0284, which means that the null hypothesis is kept. The same models for yaw control are also similar, with a p-value of 0.18. *ZN PID* is also similar to *PPO 0* for the yaw axis, showing a p-value of 0.0223. These were the only samples which had a p-value of more than 0.0083, meaning that statistically they could've arrived from the same population.

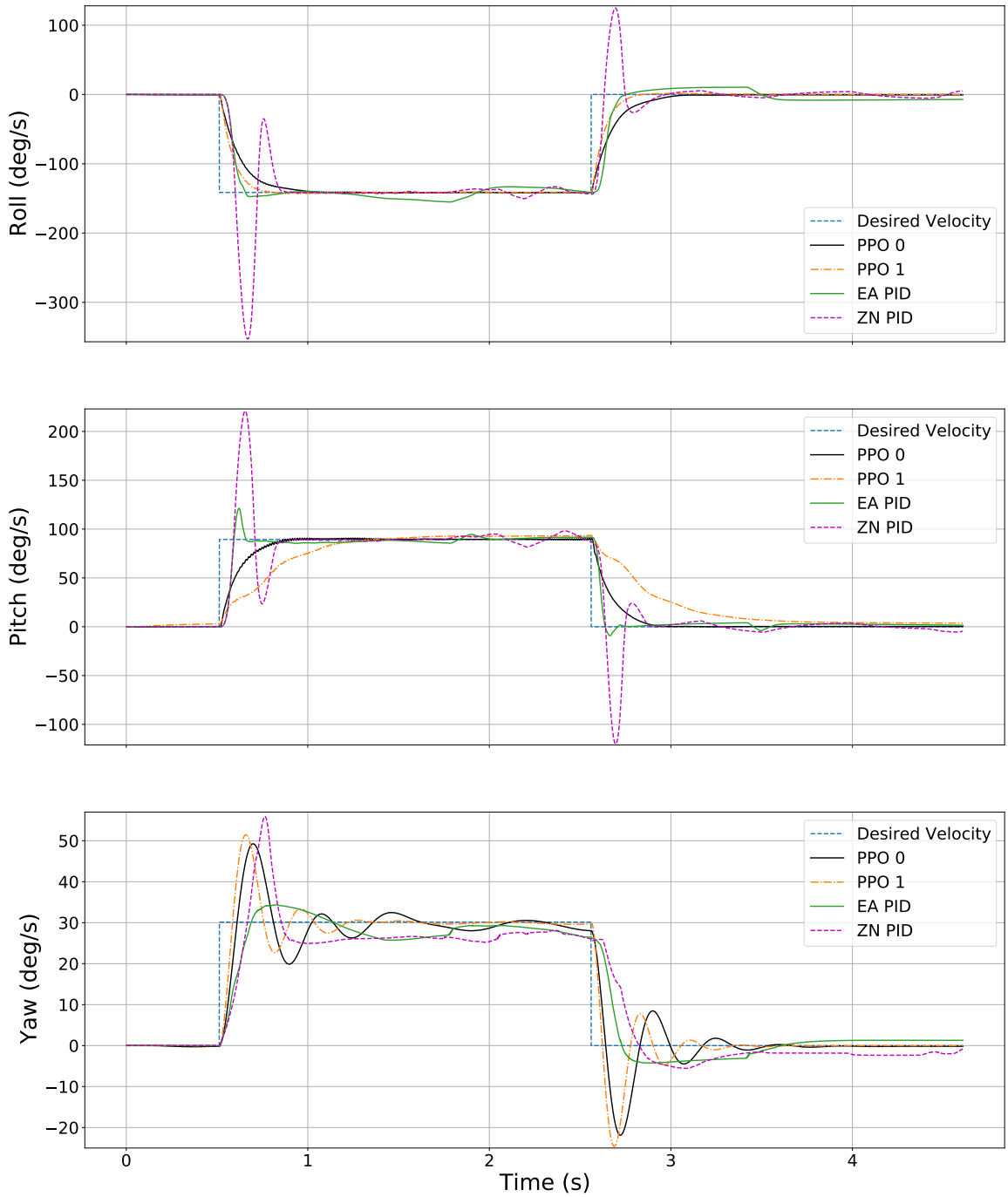

Figure 5.5: Response comparison of PPO 0, PPO 1, EA- and ZN-optimized PID

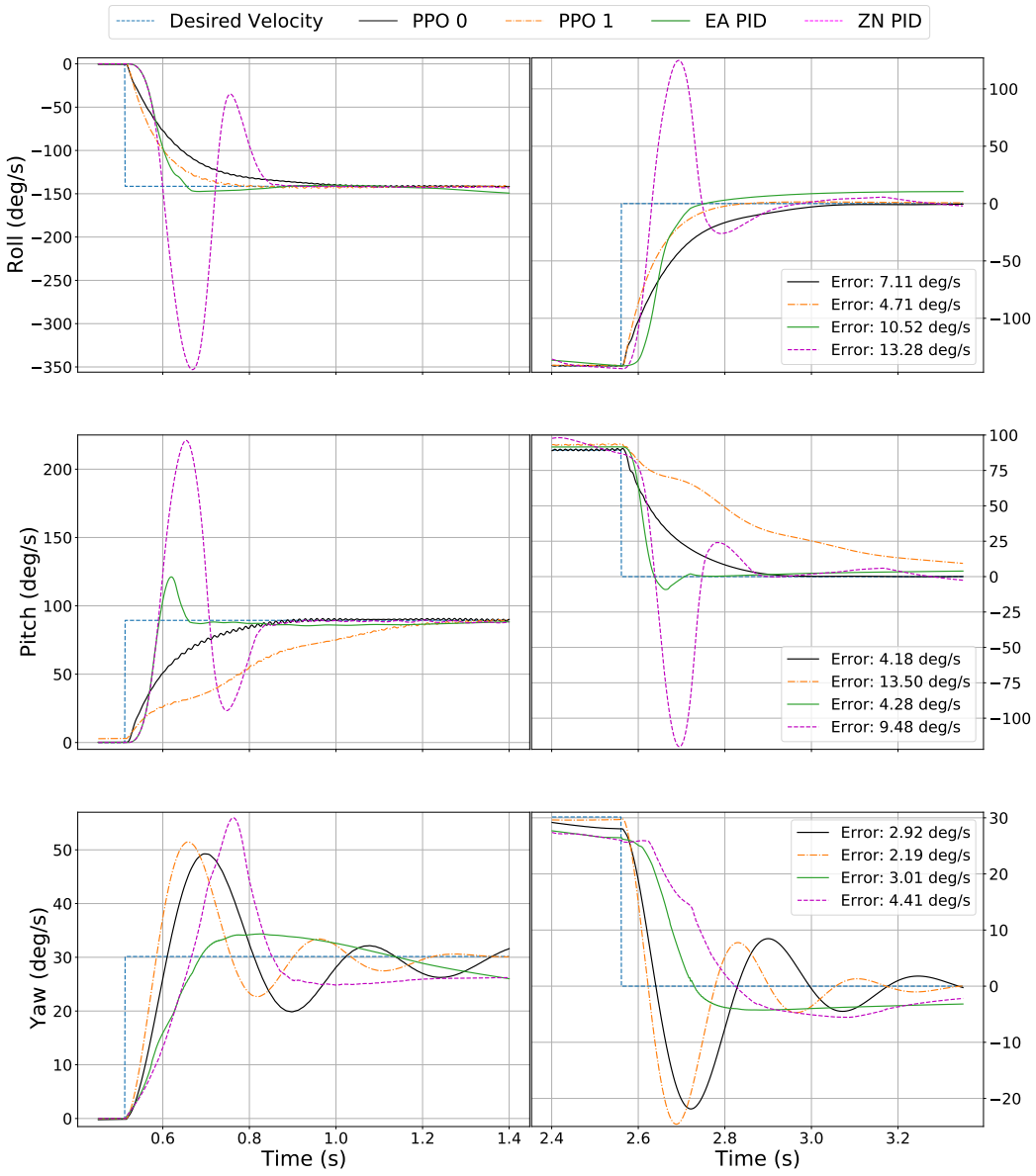

Figure 5.6: zoomed 1. part

Figure 5.7: zoomed 2. part, w. error

6 Discussion

The Proximal Policy Optimization algorithm did not have a problem learning how to control the quadrotor. Reasonable results could be seen after training for only $1.5 \cdot 10^6$ timesteps. This converts to approximately 1 hour on the hardware used, which was released in 2013. This training period could be reduced substantially on more modern hardware. This shows that PPO and the reward function used, was very effective in the learning process. The algorithm usually took $2.2 - 3.0 \cdot 10^6$ timesteps to train the best RL controllers.

The Monte Carlo tests show that using Machine Learning algorithms for optimizing PID controllers have great potential. The Evolutionary Algorithm could probably give even better results if the fitness-function was further optimized, but we wanted the PPO and EA models to have the same fitness function. This can explain why the control for PPO and EA PID behaved similar on some axes, especially pitch. The most notable difference is how PPO performs a little slower but experiences no overshoot (on roll and pitch) and keeps the velocity closer and more steady near the setpoint. While EA is faster but is not following the setpoint perfectly, having some corrections, over the duration of the given input. However, the PID controller tuned with EA heavily outperforms the controller tuned after Ziegler Nichols' method.

There seems to be a trade-off between reasonable response-time on all axis, or fast response on two of the axes, while the response on one axis is compromised. This may be because the vehicle is given the task of changing velocity on all axis at the same time, resulting in that some of the axes has to be prioritized over the other. The tests that the vehicle is given is a worst case scenario. A quadrotor is underactuated, meaning it has a lower number of actuators than Degrees Of Freedom. It changes orientation by adjusting the speed of several motors. Combining

the difference in speed on the four motors, resulting in the change of orientation.

To roll, the vehicle increases the speed on motors 1 and 3 (motors on the same side of the x-axis), while reducing or holding the speed on the opposite side. To pitch the vehicle, the motors on one side of the y-axis is increased or reduced, relatively to the opposite side. When both of these actions is executed simultaneously, the roll and pitch will interfere with each other, since the motors in use for roll and pitch overlap. Same with rotation on the yaw axis. As a result, some of the torques about each axis may cancel each other out. This could be why *PPO 1* experiences instability on one axis, while models like *EA PID* show acceptable performance on all axes. The reason to why the *PPO 1* was kept in the comparison, was to show the potential that the RL control system have. None of the EA models tested were able mirror the performance on any of the axes.

One aspect of control that the PPO model does well, is consistent and smooth tracking of the desired velocity. It is able to keep a very steady change of velocity, followed by perfectly holding that amount, even when given setpoints for high velocities. PID-control on the other hand, is not able to keep the same velocity as steady for the duration of the input. At least from how they are tuned in this thesis. Rather the error increases slowly until the integral-part of the controller accumulates enough error, before it correct itself. A high integral-value for the PID would make it hold the desired velocity closer, but this increases the amount of corrections and would result in many small oscillations. Resulting in a less stable controller overall. The EA-PID controllers would get this kind of behaviour when removing the oscillation-penalty from the reward-function. This is one of the disadvantages with PID control. It has to obey to a mathematical formula, where there might not exist a perfect solution to the gain parameters. In this sense, RL control is more "free". The only weak point here is the reward-function, which is clearly not perfected.

From figure 8.3, one can see that the *PPO 0* model had the overall best results, if we consider the average between all axes. Even though the median error was slightly higher than *EA PID* for pitch and yaw, it still experienced notably better results for a portion of the tests on pitch and yaw. Shown by the lower whisker and the density of the rain cloud plots

in figure 8.1 and 8.2.

Another thing to point out is that every datapoint from the Monte Carlo simulation, is the mean error from an entire episode on each axis, including the start and end where the vehicle is just idling. This means that rapid overshoots and small corrections will not greatly affect the mean error, calculated from the episode. Those models that had the greatest results, were the ones that stayed the closest within the desired velocity, on average through the whole episode. If we were looking for the fastest models with no overshoot, we would have to use a different measurement of the performance. One could also choose to only measure error when the desired velocity is not 0. The disadvantage to this is that the models which are unable to hold a steady idle, would perform better on this metric. Another thing to point out is the episodes where the setpoints had a high angular velocity, the mean error is also usually higher, than those with lower velocities. This can be seen in the results chapter from figure 5.6 and 5.7, where the corresponding error to each model is shown. All models show larger error on roll axis, compared to yaw axis, even though most models show visibly better performace for roll control. This is because the setpoint for roll was -144 deg/s and 30 deg/s on yaw. A possibility could be to measure the error as a percentage of the setpoint. The results for each model on that specific episode, by measuring error as a percentage is shown in table 5.3 in results. Here it can be seen that the error for roll control is lower than for yaw, by this metric.

The Monte Carlo simulation tells us which models were the most predictable and stable. The models with the smallest variance in error, could be the controller of choice when looking for reliability. The reason for using Ziegler Nichols and Evolutionary Algorithm when tuning the PID controller, was to get an objective comparison with the new neural network controller. Other comparisons often use a "trial and error" method when tuning PID gains [38]. While this method can provide good results and being the most common method for tuning. The result will depend on the tuners intuition, unless they use a strict systematic approach. A problem that may arise from this, is that one could skew the results by tuning it badly, or having a bias towards a behaviour that may be seen as sub optimal to some users. In this thesis we partly removed this human factor by using mathematical methods and ML algorithms to tune

the controllers. The comparison also got an interesting view of two different controllers tuned after the same behaviour, since they share the same reward-function.

Looking back on the process of modeling the vehicle and developing the controllers. It appears like an easy and straightforward process, but the procedure did not seem that certain after trying to understand the available documentation. Going back to the problem statement and goals of comparing the resulting controller to existing solutions, with development time and cost in mind. When one has spent the time going through the process of modeling and optimizing such a controller once, it can easily be done the second time. The time it took to train a RL model with great results, took substantially longer than optimizing PID gains with EA. Concluding that RL and other ML algorithms *can* be a viable tool for new attitude controllers for aerial vehicles.

Future Work

There are lots of things that could be tested further between the PPO and PID controllers. To make an even better comparison, one could use RL to tune the PID gains, by using the same training environment as the PPO model.

An obvious point would be to compare the best performing models from this thesis, on the real drone. Secondly and maybe within the same scope of the previous point, would be to see which controller that adapts the best to changes in the environment. This could be done by changing the physical components of the vehicle after its deployment in the real world, or fly in environments where external disturbances like wind is present.

A digital twin is a model that has measurements from the real world. One would always want to increase the fidelity of the digital version, to make the reality gap closer. It could be interesting to collect data from a real flight test and use this data for the RL model. To further improve the algorithms for learning how to control the vehicles.

7 | Conclusion

The UAV industry is always evolving into new areas and industries, and the vehicles are getting more accessible to the average consumer. This has left a wide variety of aerial vehicles available with varying degree of quality. This thesis looked into the challenging part of developing new controllers for UAVs, and compared the results.

GymFC showed to be a great framework that simplified the modeling aspect of aerial vehicles, while also providing a great platform for developing and testing controllers. The resulting controllers showed that their was still room for improvement, as the reward function did mostly produce adequate results, but the behaviour for the RL model was sub optimal on some occasions. The optimization of PID gains with Evolutionary Algorithm showed stable results, and was on par with the best RL model in some areas. Both RL and EA optimized PID controller showed to be reliable for attitude control. By eliminating the human factor in the tuning process, and using the same training environment for the Reinforcement Learning and Evolutionary Algorithm, we got an interesting comparison between the two control systems.

Some classes of UAV could benefit from this more than others, like the Ducted Fan Micro Air Vehicle (DFMAV), which is actively unstable and has more complex dynamics, but has great potential because of its high payload-carrying efficiency. However, this thesis focused on the quadrotor, since it's a much more researched platform, with more information and tools available. Developing a model for a DFMAV from the ground up, would be beyond the scope of this thesis, and one would risk not getting any reasonable results within the timeframe. GymFC is designed to work with any type of airframe, so one can use the tuning algorithms seen in this thesis on the more challenging aerial vehicles.

One of the goals for this thesis were to deploy the flight controller onto

the real life quadrotor, that we modeled after. Getting good results in the simulator took longer time than expected and we realized we were short on time. The focus was shifted on trying to get the best result in simulator, rather than spending a lot of time with the expected troubleshooting, from putting the project into the real world, and extracting data from real life tests.

Bibliography

[1] R.M. Howard and I. Kaminer. 'Survey of unmanned air vehicles'. In: *Proceedings of 1995 American Control Conference - ACC'95*. Vol. 5. 1995, 2950–2953 vol.5. DOI: 10.1109/ACC.1995.532054.

[2] Guowei Cai, Jorge Dias and Lakmal Seneviratne. 'A Survey of Small-Scale Unmanned Aerial Vehicles: Recent Advances and Future Development Trends'. In: *Unmanned Systems* 02 (Apr. 2014), pp. 175–199. DOI: 10.1142/S2301385014300017.

[3] ITF Research Reports. 'Ready for Take Off? Integrating Drones into the Transport System'. In: (2021), p. 98.

[4] AiRMOUR. *Opening up the skies for medical emergency drones*. URL: https://airmour.eu/ (visited on 13/05/2022).

[5] Sergey Levine et al. 'Learning Hand-Eye Coordination for Robotic Grasping with Deep Learning and Large-Scale Data Collection'. In: *The International Journal of Robotics Research* 37 (Mar. 2016). DOI: 10.1177/0278364917710318.

[6] David Silver et al. 'Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm'. In: (Dec. 2017).

[7] David Silver et al. 'Mastering the game of Go without human knowledge'. In: *Nature* 550 (Oct. 2017), pp. 354–359. DOI: 10.1038/nature24270.

[8] Oriol Vinyals et al. 'StarCraft II: A New Challenge for Reinforcement Learning'. In: (Aug. 2017).

[9] Jemin Hwangbo et al. 'Control of a Quadrotor With Reinforcement Learning'. In: *IEEE Robotics and Automation Letters* 2.4 (2017), pp. 2096–2103. DOI: 10.1109/LRA.2017.2720851.

[10] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2015.

[11] U. Ansari, A. Bajodah and M. T. Hamayun. 'Quadrotor Control Via Robust Generalized Dynamic Inversion and Adaptive Non-Singular Terminal Sliding Mode'. In: *Asian Journal of Control* 21 (2019), pp. 1237–1249.

[12] The Linux Foundation. *The Dronecode Project*. URL: https://www.dronecode.org/ (visited on 13/05/2022).

[13] Xiaodong Zhang et al. 'A Survey of Modelling and Identification of Quadrotor Robot'. In: *Abstract and Applied Analysis* 2014 (Oct. 2014). DOI: 10.1155/2014/320526.

[14] Prafulla Dhariwal et al. *OpenAI Baselines*. https://github.com/openai/baselines. 2017.

[15] Olfa Boubaker. 'The Inverted Pendulum Benchmark in Nonlinear Control Theory: A Survey'. In: *International Journal of Advanced Robotic Systems* 10 (Sept. 2013), pp. 233–242. DOI: 10.5772/55058.

[16] Keith D. Foote. *A Brief History of Machine Learning*. URL: https://www.dataversity.net/a-brief-history-of-machine-learning/.

[17] J.M. Tao and J.Y.S. Luh. 'Application of neural network with real-time training to robust position/force control of multiple robots'. In: *[1993] Proceedings IEEE International Conference on Robotics and Automation*. 1993, 142–148 vol.1. DOI: 10.1109/ROBOT.1993.291974.

[18] Sun ChangHao Duan HaiBin. 'Pendulum-like oscillation controller for micro aerial vehicle with ducted fan based on LQR and PSO'. In: (2013). DOI: 10.1007/s11431-012-5065-5.

[19] Rong-Jong Wai, Jeng-Dao Lee and Kun-Lun Chuang. 'Real-Time PID Control Strategy for Maglev Transportation System via Particle Swarm Optimization'. In: *Industrial Electronics, IEEE Transactions on* 58 (Mar. 2011), pp. 629–646. DOI: 10.1109/TIE.2010.2046004.

[20] Ali Emami and Amin Rezaeizadeh. 'Adaptive model predictive control-based Attitude and Trajectory Tracking of a VTOL Aircraft'. In: *IET Control Theory and Applications* 12 (May 2018). DOI: 10.1049/iet-cta.2017.1048.

[21] Bora Erginer and Erdinc Altug. 'Modeling and PD Control of a Quadrotor VTOL Vehicle'. In: *2007 IEEE Intelligent Vehicles Symposium*. 2007, pp. 894–899. DOI: 10.1109/IVS.2007.4290230.

[22] Ning Cao and Alan Lynch. 'Inner-Outer Loop Control for Quadrotor UAVs With Input and State Constraints'. In: *IEEE Transactions on Control Systems Technology* 24 (Dec. 2015), pp. 1–8. DOI: 10.1109/TCST.2015.2505642.

[23] Xiangdong Meng, Yu Qing He and Jianda Han. 'Survey on Aerial Manipulator: System, Modeling, and Control'. In: *Robotica* 38 (Oct. 2019), pp. 1–30. DOI: 10.1017/S0263574719001450.

[24] Taeyoung Lee, Melvin Leok and N.H. Mcclamroch. 'Control of Complex Maneuvers for a Quadrotor UAV using Geometric Methods on SE(3)'. In: (Mar. 2010).

[25] Shuanghou Deng, Siwei Wang and Zheng Zhang. 'Aerodynamic performance assessment of a ducted fan UAV for VTOL applications'. In: *Aerospace Science and Technology* 103 (June 2020), p. 105895. DOI: 10.1016/j.ast.2020.105895.

[26] Aruneshwaran Rajashekaran et al. 'Neural adaptive flight controller for ducted-fan UAV performing nonlinear maneuver'. In: Apr. 2013, pp. 51–56. DOI: 10.1109/CISDA.2013.6595427.

[27] Chao Zhang, Huosheng Hu and Wang Jing. 'An adaptive neural network approach to the tracking control of micro aerial vehicles in constrained space'. In: *International Journal of Systems Science* 48 (Mar. 2016), pp. 1–11. DOI: 10.1080/00207721.2016.1157223.

[28] Zihuan Cheng, Hailong Pei and Shuai Li. 'Neural-Networks Control for Hover to High-Speed-Level-Flight Transition of Ducted Fan UAV With Provable Stability'. In: *IEEE Access* 8 (2020), pp. 100135–100151. DOI: 10.1109/ACCESS.2020.2997877.

[29] Dag Sonntag. 'A Study of Quadrotor Modeling'. MA thesis. Linköpings Universitet, 2011.

[30] Jemin Hwangbo et al. 'Learning agile and dynamic motor skills for legged robots'. In: *Science Robotics* 4 (Jan. 2019), eaau5872. DOI: 10.1126/scirobotics.aau5872.

[31] Paul Fahlstrom and Thomas Gleason. 'Introduction to UAV Systems: Fourth Edition'. In: *Introduction to UAV Systems: Fourth Edition* (Aug. 2012). DOI: 10.1002/9781118396780.

[32] C Balan. 'Modelling and Linear Control of a Quadrotor'. MA thesis. Cranfield University, 2007.

[33] Robert Mahony, Vijay Kumar and Peter Corke. 'Multirotor Aerial Vehicles: Modeling, Estimation, and Control of Quadrotor'. In: *Robotics Automation Magazine, IEEE* 19 (Sept. 2012), pp. 20–32. DOI: 10.1109/MRA.2012.2206474.

[34] J. Gordon Leishman. *Principles of Helicopter Aerodynamics*. Cambridge University Press, 2016. ISBN: 9781107013353.

[35] E. M. Greitzer Z. S. Spakovszky and I. A. Waitz. 'Performance of propellers'. In: (). URL: https://web.mit.edu/16.unified/www/FALL/thermodynamics/notes/node86.html.

[36] *PX4 User Guide*. URL: https://docs.px4.io/ (visited on 13/05/2022).

[37] M.A. Johnson et al. *PID control: New identification and design methods*. Jan. 2005, pp. 1–543. DOI: 10.1007/1-84628-148-2.

[38] Jemie Muliadi and Benyamin Kusumoputro. 'Neural Network Control System of UAV Altitude Dynamics and Its Comparison with the PID Control System'. In: *Journal of Advanced Transportation* 2018 (Jan. 2018), pp. 1–18. DOI: 10.1155/2018/3823201.

[39] J. G. Ziegler and N. B. Nichols. 'Optimum settings for automatic controllers.' In: *ASME* (1942).

[40] S. Marsland. *Machine Learning: An Algorithmic Perspective, Second Edition*. Chapman & Hall. CRC Press, 2014. ISBN: 9781466583337. URL: https://books.google.no/books?id=6GvSBQAAQBAJ.

[41] *Example of a deep neural network*. URL: https://commons.wikimedia.org/wiki/File:Example_of_a_deep_neural_network.png.

[42] John Schulman et al. *Proximal Policy Optimization Algorithms*. 2017. arXiv: 1707.06347 [cs.LG].

[43] Asteris Apostolidis and Konstantinos P. Stamoulis. 'An AI-based Digital Twin Case Study in the MRO Sector'. In: *Transportation Research Procedia* 56 (2021). 1st International Conference on Aviation Future: Challenge and Solution (AFCS 2020), pp. 55–62. ISSN: 2352-1465. DOI: https://doi.org/10.1016/j.trpro.2021.09.007. URL: https://www.sciencedirect.com/science/article/pii/S2352146521006347.

[44] Edward H. Glaessgen, D. S. Stargel and D. S. Stargel. 'The Digital Twin Paradigm for Future NASA and U.S. Air Force Vehicles'. In: (2012).

[45] Adrian Barbu and Song-Chun Zhu. 'Introduction to Monte Carlo Methods'. In: *Monte Carlo Methods*. Singapore: Springer Singapore, 2020, pp. 1–17. ISBN: 978-981-13-2971-5. DOI: 10.1007/978-981-13-2971-5_1. URL: https://doi.org/10.1007/978-981-13-2971-5_1.

[46] William Koch. *Flight controller synthesis via deep reinforcement learning*. 2019. DOI: 10.48550/ARXIV.1909.06493. URL: https://arxiv.org/abs/1909.06493.

[47] Fadri Furrer et al. 'Robot Operating System (ROS): The Complete Reference (Volume 1)'. In: ed. by Anis Koubaa. Cham: Springer International Publishing, 2016. Chap. RotorS—A Modular Gazebo MAV Simulator Framework, pp. 595–625. ISBN: 978-3-319-26054-9. DOI: 10.1007/978-3-319-26054-9_23. URL: http://dx.doi.org/10.1007/978-3-319-26054-9_23.

[48] *Fusion 360: Integrated CAD, CAM, CAE, and PCB software*. URL: https://www.autodesk.com/products/fusion-360 (visited on 13/05/2022).

[49] Michael L. Waskom. 'seaborn: statistical data visualization'. In: *Journal of Open Source Software* 6.60 (2021), p. 3021. DOI: 10.21105/joss.03021. URL: https://doi.org/10.21105/joss.03021.

[50] Kirstie Whitaker et al. *RainCloudPlots/RainCloudPlots: WellcomeOpenResearch*. Version v1.1. Aug. 2019. DOI: 10.5281/zenodo.3368186. URL: https://doi.org/10.5281/zenodo.3368186.

[51] Patrick E. McKnight and Julius Najab. 'Mann-Whitney U Test'. In: *The Corsini Encyclopedia of Psychology*. John Wiley Sons, Ltd, 2010, pp. 1–1. ISBN: 9780470479216. DOI: https://doi.org/10.1002/9780470479216.corpsy0524. eprint: https://onlinelibrary.wiley.com/doi/

pdf/10.1002/9780470479216.corpsy0524. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/9780470479216.corpsy0524.

[52] Winston Haynes. 'Bonferroni Correction'. In: *Encyclopedia of Systems Biology*. Ed. by Werner Dubitzky et al. New York, NY: Springer New York, 2013, pp. 154–154. ISBN: 978-1-4419-9863-7. DOI: 10.1007/978-1-4419-9863-7_1213. URL: https://doi.org/10.1007/978-1-4419-9863-7_1213.

[53] G.K. Ananda et al J.B. Brandt R.W. Deters. *UIUC Propeller Database, Vols 1-3, University of Illinois at Urbana-Champaign*. URL: https://m-selig.ae.illinois.edu/props/propDB.html.

[54] *GrabCad: 3D CAD Model Library*. URL: https://grabcad.com/library/s500-frame-1 (visited on 13/05/2022).

[55] *S500 V2 Kit*. URL: http://www.holybro.com/product/pixhawk4-s500-v2-kit/ (visited on 13/05/2022).

[56] *BetaFlight*. URL: https://github.com/betaflight/betaflight (visited on 13/05/2022).

[57] *Custom Motor Mixing Multirotor*. URL: https://oscarliang.com/custom-motor-output-mix-quadcopter/ (visited on 15/05/2022).

8 Appendix

Rain cloud plot of the roll control can be seen in the results chapter 5.4.

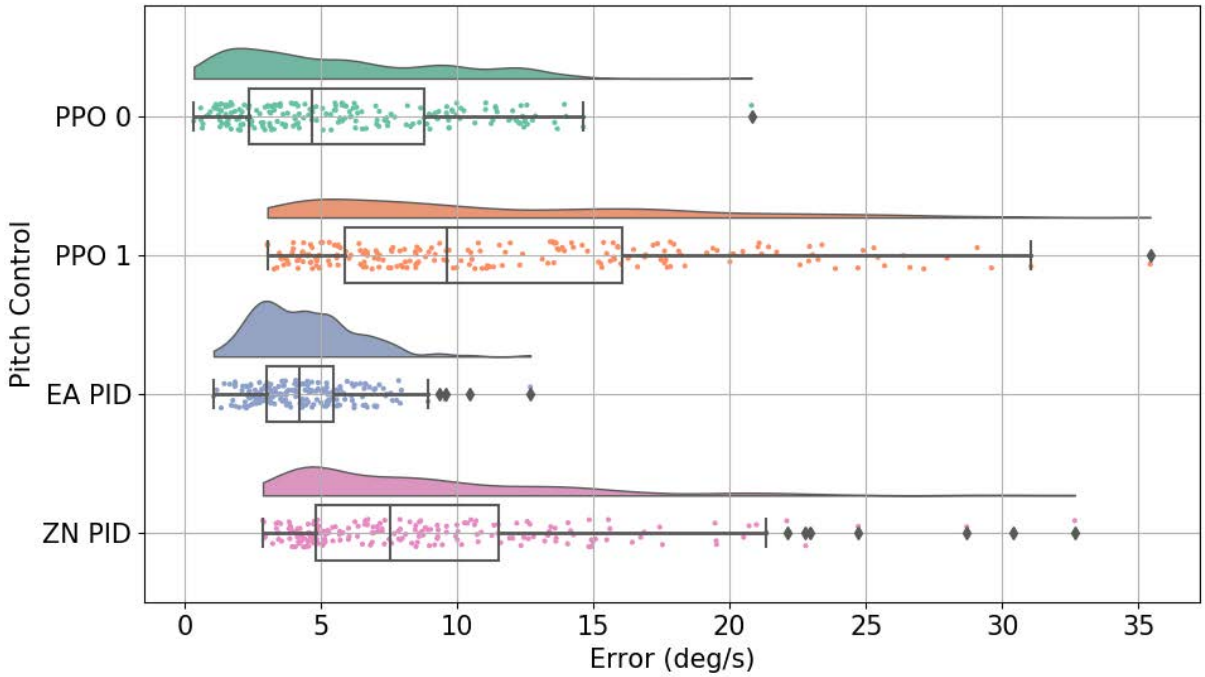

Figure 8.1: Pitch error comparison from Monte Carlo sim. in a rain cloud plot

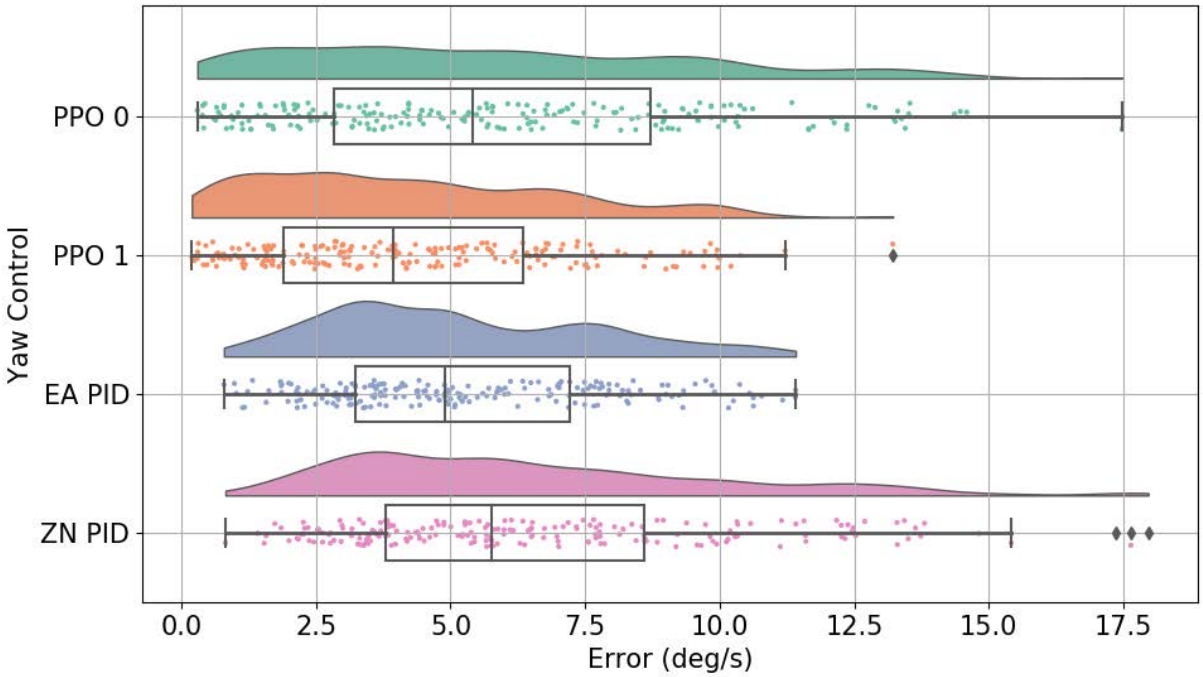

Figure 8.2: Yaw error comparison from Monte Carlo sim. in a rain cloud plot

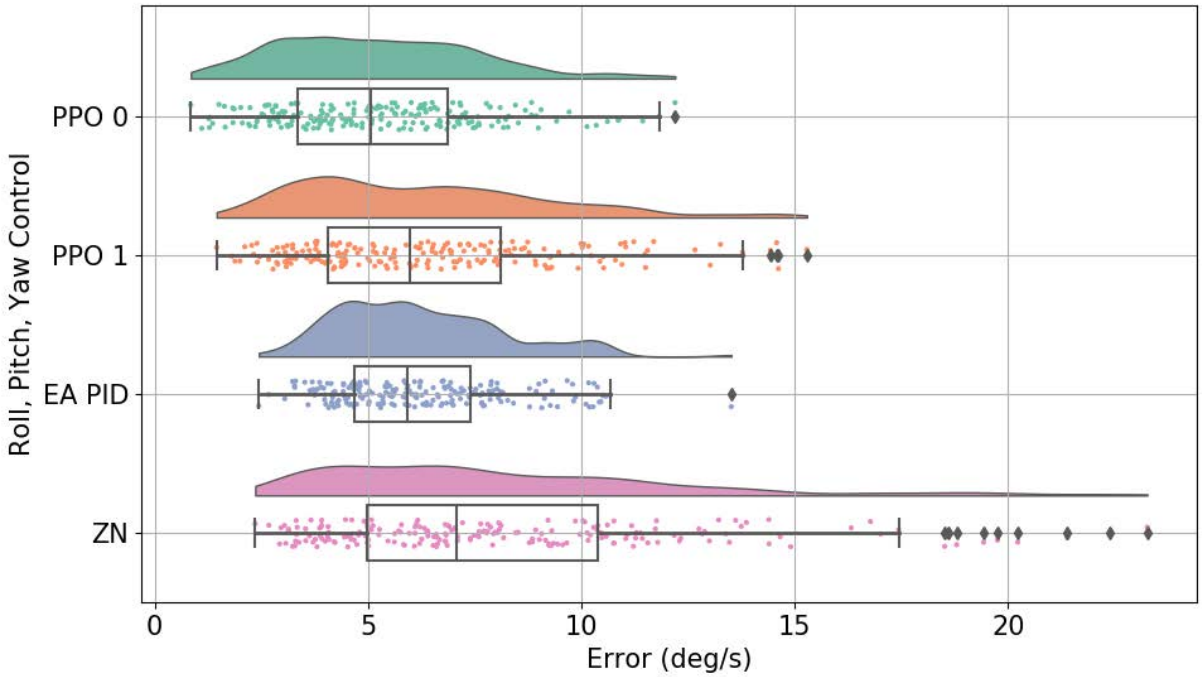

Figure 8.3: Average error from all axes from Monte Carlo sim. in a rain cloud plot

```
<plugin name="prop_1_motor_model" filename='libgazebo_motor_model.so'>
      <robotNamespace></robotNamespace>
      <jointName> prop_1_joint </jointName>
      <linkName> prop_1 </linkName>

      <turningDirection> cw </turningDirection>
      <timeConstantUp> 0.0125 </timeConstantUp>
      <timeConstantDown> 0.025 </timeConstantDown>
      <maxRotVelocity> 1096 </maxRotVelocity>
      <motorConstant> 8.365e-6 </motorConstant>
      <momentConstant> 0.0140 </momentConstant>

      <motorNumber> 0 </motorNumber>
      <rotorDragCoefficient> 0.000175 </rotorDragCoefficient>
      <rollingMomentCoefficient> 1e-06 </rollingMomentCoefficient>
      <rotorVelocitySlowdownSim> 10 </rotorVelocitySlowdownSim>
      <rotorVelocityUnits> rpm </rotorVelocityUnits>

      <escTransferFunction>
          <term1Coefficient> -3316.98 </term1Coefficient>
          <term2Coefficient> 13849.13 </term2Coefficient>
          <term3Coefficient> -11.41 </term3Coefficient>
      </escTransferFunction>
    </plugin>


<link name="motor_1">
        <pose>-0.165 -0.165 0.12465 0 -0 0</pose>
      <inertial>
        <pose>0.000000 -0.000000 0.016000</pose>
        <mass>0.066</mass>
        <inertia>
          <ixx>8.8677e-06</ixx>
          <ixy>0</ixy>
          <ixz>0</ixz>
          <iyy>8.8677e-06</iyy>
          <iyz>0</iyz>
          <izz>6.3810e-06</izz>
        </inertia>
      </inertial>
```

Figure 8.4: Showing the parts of the vehicle configuration file. This is only showing one motor. The other 3 are identical except for their position which is mirrored since this is a symmetrical x-frame.

```
<link name="prop_1">
    <pose>-0.165 -0.165  0.1264 0 -0 0</pose>
      <inertial>
        <pose>0.000000 0.000000 0.00553</pose>
        <mass>0.012</mass>
        <inertia>
          <ixx>4.0426e-05</ixx>
          <ixy>5.1814e-07</ixy>
          <ixz>0</ixz>
          <iyy>3.3892e-07</iyy>
          <iyz>0</iyz>
          <izz>4.0665e-05</izz>
        </inertia>
      </inertial>


<link name='battery'>
        <pose>0 0 0 0 0 1.5708</pose>
      <inertial>
        <pose>0.000000 0.000000 -0.019000</pose>
        <mass>0.490</mass>
        <inertia>
          <ixx>1.501e-4</ixx>
          <ixy>0</ixy>
          <ixz>0</ixz>
          <iyy>8.416e-04</iyy>
          <iyz>0</iyz>
          <izz>8.730e-04</izz>
</inertia>
      </inertial>


<link name='frame'>
        <pose>0 0 0.039 0 0 1.5708</pose>
        <inertial>
            <pose>0.000000 0.000000 0.00000 </pose>
            <mass>0.514</mass>
            <inertia>
              <ixx>0.003260</ixx>
              <ixy>0</ixy>
              <ixz>0.000001819</ixz>
              <iyy>0.003632</iyy>
              <iyz>0.000000205</iyz>
              <izz>0.005573</izz>
            </inertia>
        </inertial>
    </link>
```