Comparison of Traditional Optimal Control Methodologies to Reinforcement Learning Agents in Quadcopter Attitude Control Applications

By

Stephen M. Kleppinger

Thesis Project
Submitted in partial fulfillment of the
Requirements for the degree of

MASTER OF SCIENCE IN DATA SCIENCE

December 2021
Shreenidhi Bharadwaj, First Reader
Edward Arroyo, Second Reader

Comparison of Traditional Optimal Control Methodologies to Reinforcement Learning Agents in Quadcopter Attitude Control Applications

Stephen Kleppinger

Abstract

Attitude control in quadrotor Unmanned Aerial Vehicle (UAV) systems is traditionally managed by optimal control loops tuned to minimize errors in performance. While robust, these loops perform sub-optimally in dynamic and unpredictable environments which inspire new interest in sophisticated solutions and approaches such as reinforcement learning (RL) approaches which should be able to provide control in a wider range of use cases. Proximal Policy Optimization (PPO)-trained RL agents were compared against their Proportional-Integral-Derivative (PID) controlled counterparts in order to determine why this statistically-driven approach could outperform tuned controllers in noisy flight conditions. It is shown that PID controllers perform significantly better in stable conditions, but RL agents were able to more quickly restore UAV stability in certain harsh environments. To investigate these differences a simulated environment was built for RL training and to identify if RL-based attitude control is one of the viable solutions for future quadcopter attitude control.

Index Terms: Reinforcement Learning, Intelligent Systems, Proximal Policy Optimization, Aerial Systems: Mechanics and Control, Attitude Control, Learning and Adaptive Systems

Acknowledgments

The work detailed in this paper is the culmination of almost two years of a massive amount of work, snuck in between different jobs and a variety of major life events, not the least of which being the first covid pandemic. Throughout all of this I was supported and cared for by an amazing group of family, friends, and educators that made this endeavor possible.

In no particular order I would like to thank the staff at Northwestern University, Andrew Liu, Arlo Pluberg, my brilliant and wonderful grandparents Peter and Maria Crescimanno, John Bardeen, Tim Troutman and his wife Valerie, the teams at the tourism boards of Canada and of course their Scottish counterparts, my parents, Toshi Ishihara, Philip K. Dick, Ms. Emmeline Vance of Slough, my future wife, our dog Bucephalus, Dave Brubeck, Mrs. Paulette Kutschat, and last but not least, Jeff.

Contents

List of Figures 5

List of Tables 7

1 Summary 8

2 Introduction 10

3 Simulation of the Quadrotor UAV 13

3.1 Body Summary 15

3.1.1 Systems Level Design - Hover 16

3.1.2 Rotation Matrices 17

3.1.3 Propeller Dynamics 18

3.1.4 Body Frame Dynamics 20

3.1.5 Power Train Simulation 21

3.1.6 Inertia and Rotation effects 23

3.1.7 Putting it all together 25

3.1.8 Controller Design 27

4 Reinforcement Learning 30

4.1 About Reinforcement Learning 30

4.2 Markov Decision Process 32

4.3 Gamma Discounted Returns 33

4.4 Deep Learning Methods 34

4.5 Actor-Critic Methods 37

4.6 PPO Implementation 39

5 Methods **41**
5.1 Technologies, Tools, and Libraries Used 41
5.2 Simulation Application to OpenAI Gym 41
5.3 Reward Function Engineering 43
5.4 Variable Learning Rates 45
5.5 Render and Visualization 46
5.6 Noise Addition to Step Output 46
5.7 PPO Hyperparameters 48

6 Results and Conclusions **49**
6.1 Single Flight Visualizations 50
6.2 Averaged Behavior 52
6.3 High Error Environment 54
6.4 Nonlinear Input 55
6.5 Conclusion 57
6.6 Future Work 59

A Acronyms **60**

B Model Training **61**

List of Figures

3.1 The Symv ”Sky Phantom” modeled in this paper. 14
3.2 An idealized rendering of a quadcopter, detailing the most important parts of the mechanics of its construction and the definition of its body reference frame. 16
3.3 The response curve used in this simulation to circumvent nonlinear propeller fluid dynamic modeling. 20
3.4 A simple look at a traditional PID loop. 27
3.5 An expansion of the PID loop discussed earlier, showing the interconnection of the outer global position control frame and the inner body reference control. 29

4.1 An illustration of the flow of a traditional RL learning problem. The agent suggests action a_t based off of state s_t, which causes the environment to update to new state s_{t+1} with new reward r_{t+1}. 31
4.2 An illustration of a single neuron within an ANN. Inputs are its input elements and W are the relative weights for each input. . . . 34
4.3 A high-level visualization of a densely connected neural network, showing how every node in a layer connects to every other neuron in the layers preceding and following. 36
4.4 A high level overview of actor critic training methodology 39

5.1 An example of the visualization of the drone from a takeoff. . . . 47

6.1 A comparison of AI vs Traditional PID control of the three radial degrees of freedom in the quadcopter body frame of dynamics. R (Yaw) is not included in the Reward function and is technically uncontrolled. 50
6.2 A comparison of AI vs Traditional PID control of the three linear velocity degrees of freedom in the quadcopter body frame of dynamics. These were not directly controlled by either method, so the drift shown is a consequence of the method's ability to quickly restore the quadcopter. Proportional to angle error. 51
6.3 A comparison of AI vs Traditional PID control of the three radial degrees of freedom in the quadcopter body frame of dynamics, time averaged over 2000 runs with different initial conditions. . . 53
6.4 A comparison of AI vs Traditional PID control, looking specifically at the signal output of both systems. Time averaged over 2000 runs with different initial conditions. 54
6.5 A comparison of AI vs Traditional PID control with rotational error injected regularly. Time averaged over 200 runs with different initial conditions. 56
6.6 A comparison of AI vs Traditional PID control in cases of nonlinear, rapidly changing P and Q setpoints. While neither controller performs well, the AI look clearly tracks the changes more rapidly than the PID loop. 57
6.7 A comparison of 100 episodes of AI vs Traditional PID control in cases of nonlinear, rapidly changing P and Q setpoints. Over this averaged time window the AI controlled agent outperforms the PID loops. 58

B.1 Per-Episode reward plot against training time 62

List of Tables

3.1 Simulated Battery Properties. 23

5.1 PPO2 Hyperparameters. 48

6.1 The distribution of times it took for a RL trained agent to reach a point of minimum error for quadcopter attitude over 2000 random samples 52

Chapter 1

Summary

Quadrotor UAVs are wildly popular small flight craft that are known to be exceptionally maneuverable and robust, however their underlying dynamics are inherently nonlinear and requires advanced control systems to safely operate. While these systems are well understood and widely implemented, they have inherent trade offs as a result of linearization of the plant (the combined behavior of the quadcopter and the actuator). By comparing traditional control schemes with reinforcement learning-trained agents, it can be shown that unique improvements can be made to otherwise inefficient route planning created by exclusively traditional white-box models. A comparison of PID control behavior to PPO trained RL agents will be performed to highlight these different behaviors.

The traditional method of control will be represented by cascading PID loops, tuned to exhibit ideal performance in standard flight conditions. The reinforcement learning trained agent will see the same state available to the PID loops - the instantaneous error in the six degrees of freedom - and will train its model off of millions of six second snippets of simulation. The initial conditions of this simulation act as a randomly generated 'toss', as the drone enters the environment with random bounded linear and angular velocity. It is shown that the RL agent is capable of controlling the quadcopter with less overall error in a number of environments where the perceived error is highly variable, although generally the PID controlled quadcopter outperforms its RL counterpart in more consistent cases. While not immediately viable for actual efficient quadcopter control, RL agents have a promising future in nonlinear and erratic motion control where the

external force contribution to attitude acceleration exceeds what a controller can be designed for. Future research is warranted to minimize the error exhibited by the quadcopter during steady state operation.

Chapter 2

Introduction

Multi-rotor UAVs have been shown to have a great range of possible cross-industry applications, from surveying to wildfire fighting to defense. Autonomous control is fundamental to any application of UAVs as all state-to-state transitions are managed by on-board computers which are necessary to control the drone state for all perturbations.[1] This attitude control is traditionally accomplished with ideal linear control systems such as PID controllers, although recent attempts to accomplish this by reinforcement learning agents have shown early success at mimicking their stability.[2] However, around this attitude control loop the pathfinding and piloting of the UAV must be constantly operating to update internal state targets for on target maneuvers[3].[4] These constant target updates serve as a kind of autopilot for the drone, updating attitude controller behaviors.[5] The selection of these targets have traditionally also been achieved by a variety of optimal control systems, however reinforcement learning techniques can be used to generate interesting and nonlinear equivalents to these targets. The comparison of these techniques are the focus

1. Azin Shamshirgaran, Hamed Javidi, and Dan Simon, *Evolutionary Algorithms for Multi-Objective Optimization of Drone Controller Parameters*, 2021, arXiv: 2105.08650 `[eess.SY]`.
2. William Koch et al., "Reinforcement Learning for UAV Attitude Control," *ACM Transactions on Cyber-Physical Systems*, no. 22 (Feburary 2019).
3. Amir Hussein and Rayyan Abdallah, "Autopilot Design for a Quadcopter," *self-published*, February 2018,
4. Axel Reizenstein, "Position and Trajectory Control of a Quadcopter Using PID and LQ Controllers" (master's thesis, Linköping University, 2017).
5. Lukas Bjarre, "Robust Reinforcement Learning for Quadcopter Control" (master's thesis, KTH Royal Institute of Technology, 2019).

of this paper.

PPO is a on-policy learning system that uses an Actor-Critic structure, which tends to help create an objective function during training that reliably approaches the global maximum for a wide range of RL applications. This is especially true in cases such as this where the actuators can output any value between two bounds, referred to as a continuous action spaces . PPO algorithms guarantee convergence to a stationary point on the objective function, assuming adequate smoothness and that the gradient can always be calculated.[6] They are an extension of other varieties of policy gradient algorithms (such as Trust Region Policy Optimization - TRPO) that have constrained the exploration space during each policy update and encourages rapid convergence in non-smooth objective topologies.[7] This update to traditional actor-critic approaches is part of a larger movement in the research to constrain trust regions considered by the policy in each step. As all this research is ongoing, new approaches are constantly being developed that could apply to this problem perfectly. Over the course of the experiments detailed here, an exciting new take on PPO called Phasic Policy Gradient (PPG) was developed[8] that could potentially phase out standalone PPO algorithms. However due to how new this algorithm is, PPG will not be considered in this paper. A number of methods associated with PPO models will however be discussed in the paper, ranging from reward function design, reward discounting methods, and variable learning rates.

The simulations will be discussed in detail as easy replication of results is a focus of this paper, but a number of simplifications from the full-state system will be taken. Small angle approximations will be used to reduce the complexity of coordinate frame transforms, and significant liberties will be taken to simulate thrust/motor inertia. Time delays and logarithmic fits will be used to convert control signal voltages to "thrust", skipping the need to consider motor inertia and fluid dynamics around the rotor. This simulation will run both in and out of the openAI Gym[9] environment for easy RL training.

6. Alekh Agarwal et al., "Optimality and Approximation with Policy Gradient Methods in Markov Decision Processes," *CoRR* abs/1908.00261 (2019), arXiv: 1908.00261, http://arxiv.org/abs/1908.00261.

7. Lilian Weng, "Policy Gradient Algorithms," *lilianweng.github.io/lil-log*, 2018, https://lilianweng.github.io/lil-log/2018/04/08/policy-gradient-algorithms.html.

8. Karl Cobbe et al., "Phasic Policy Gradient," *CoRR* abs/2009.04416 (2020), arXiv: 2009.04416, https://arxiv.org/abs/2009.04416.

9. Greg Brockman et al., "OpenAI Gym," *CoRR* abs/1606.01540 (2016), arXiv: 1606.01540,

The main results presented in this thesis will compare the step and frequency response behaviors of RL-trained agents to PID controllers. The RL agent will be trained exclusively on step impulse error injected at the start of each episode, to reduce external noise's deleterious effects on agent performance. As with all control problems, there is no such thing as a ”best” system, just systems that optimize against different criteria. The advantages and disadvantages discovered by this quadcopter PPO agent will then be compared against a PID counterpart in order to showcase the strengths of this new approach for application in other robotic operations.

http://arxiv.org/abs/1606.01540.

Chapter 3

Simulation of the Quadrotor UAV

The modeling behind the dynamic motion of a Quadcopter has been thoroughly researched and documented.[1] It is precisely because of these well understood behaviors that this model has been chosen as the focus of this paper. The RL and control theory topics discussed are widely applicable and the solutions presented here are considered to be representatives of more general behaviors. For the purposes of completeness, as well as to make this document serve as a complete guide for the reader, this chapter will serve as a thorough detailing of the kinematics behind quadrotor simulation through the derivation of the fundamental equations of motion of a quadcopter, as well as some of the electrical and mechanical considerations.

The drone simulated will mimic a “small” style drone, only weighing about 1kg and with a total area of around $.2m^2$. This closely resembles a Symv “Sky Phantom” drone, an inexpensive smartphone-controlled drone that is already available for inertia measurements. This style of quadcopter is popular among casual hobbyists and doesn’t have much application beyond the enjoyment of flying, and potentially some short range surveying.

As the size of an UAV increases its potential functionality skyrockets, as the available thrust output for heavier payloads increases. Search and rescue, surveillance, and goods delivery have all been shown to drastically benefit from the inclusion of drones to the point where the commercial drone industry has become

1. Wolfgang Ertel Haitham Bou-Ammar Holger Voos, “Controller Design for Quadrotor UAVs using Reinforcement Learning,” *University of Applied Sciences Ravensburg-Weingarten*, 2010,

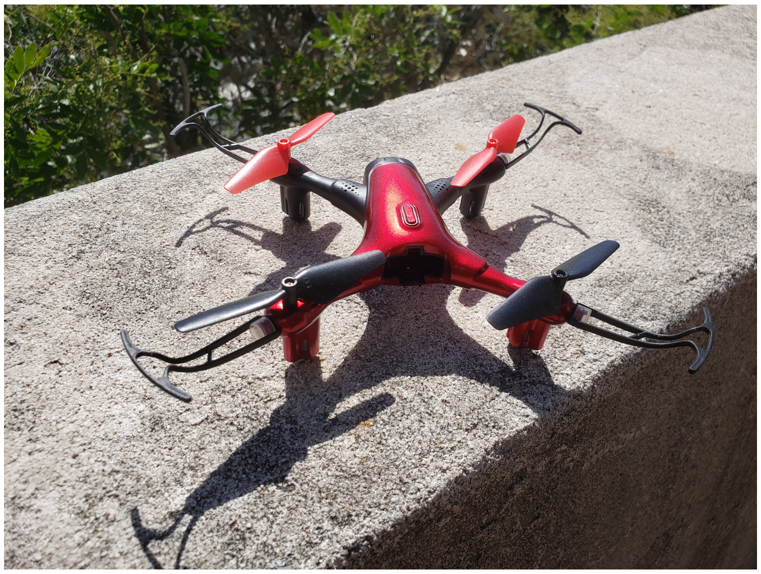

Figure 3.1: The Symv "Sky Phantom" modeled in this paper.

extremely established.[2] Other applications are constantly being explored in spaces where it is frequently dangerous for humans to go, such as in fire and natural disaster areas.[3] As the conditions the drones are expected to fly in get more hazardous, challenges in the control of these drones are introduced. For example, in direct firefighting applications the quadcopters have to navigate turbulent and hot wind, as well as the variable mass of the quadcopter before and after dropping water.[4]

No control system is designed to totally accommodate all of these eventualities, and in current tests unique logic can be added to traditional optimal control

2. Hazim Shakhatreh et al., "Unmanned Aerial Vehicles: A Survey on Civil Applications and Key Research Challenges," *CoRR* abs/1805.00881 (2018), arXiv: 1805.00881, http://arxiv.org/abs/1805.00881.

3. Hajer Ben Mnaouer et al., "FireFly Autonomous Drone Project," *CoRR* abs/2104.07758 (2021), arXiv: 2104.07758, https://arxiv.org/abs/2104.07758.

4. Elena Ausonio, Patrizia Bagnerini, and Marco Ghio, "A New Direction in Firefighting Systems," *CoRR* abs/2007.00883 (2020), arXiv: 2007.00883, https://arxiv.org/abs/2007.00883.

systems in order to make these UAVs do what they need to. RL-based agents could more easily transition between different operating regimes, depending on the state variables included during training. For these applications and others it is exciting to continue to explore RL training on quadcopters, as well as to pave the way forward to more complicated nonlinear motion models including anything from dual rotary cranes[5] to steering satellite coil motors[6] to rapeseed cleaning.[7]

3.1 Body Summary

At the face of it quadcopters seem to feature considerable complexity, between the various electrical/mechanical components and the always complicated fluid dynamics that occur around the propellers. These non-ideal components directly impact the efficiency and linearity of the quadcopter, making design and control that much more difficult. However, a quadcopter is reducible into discrete modules cleanly and its components are all effectively independent and identical. By breaking the problem down to the component level and summing up the resultant forces, there is a straightforward method to understanding the simulation and control of a quadcopter.

A quadcopter uses its 4 motors to push back against the force of gravity, but it is also carefully set up to control the apparatus against instabilities that arise during operation of the device. Each motor (labeled 1-4 in Fig. 3.1[8]) are paired, with opposite motors rotating in the same direction as each other (with matching propeller tilts). This results in the resultant force from each propeller applying a force in the same direction (-z) and torques generated by each propeller opposing the opposite rotation of the quadrotor body. The opposite rotation of the pro-

5. Ning Sun et al., "Nonlinear Motion Control of Complicated Dual Rotary Crane Systems Without Velocity Feedback: Design, Analysis, and Hardware Experiments," *IEEE Transactions on Automation Science and Engineering* 17, no. 2 (2020): 1017–1029.

6. Tadahiko Shinshi et al., "A Fast Steering Mirror Using a Compact Magnetic Suspension and Voice Coil Motors for Observation Satellites," *Electronics* 9, no. 12 (2020), https://www.mdpi.com/2079-9292/9/12/1997.

7. Xiaoyu Chai et al., "Development and Experimental Analysis of a Fuzzy Grey Control System on Rapeseed Cleaning Loss," *Electronics* 9, no. 11 (2020), https://www.mdpi.com/2079-9292/9/11/1764.

8. Yingfu Zeng et al., "Modeling Basic Aspects of Cyber-Physical Systems, Part II" (January 2013).

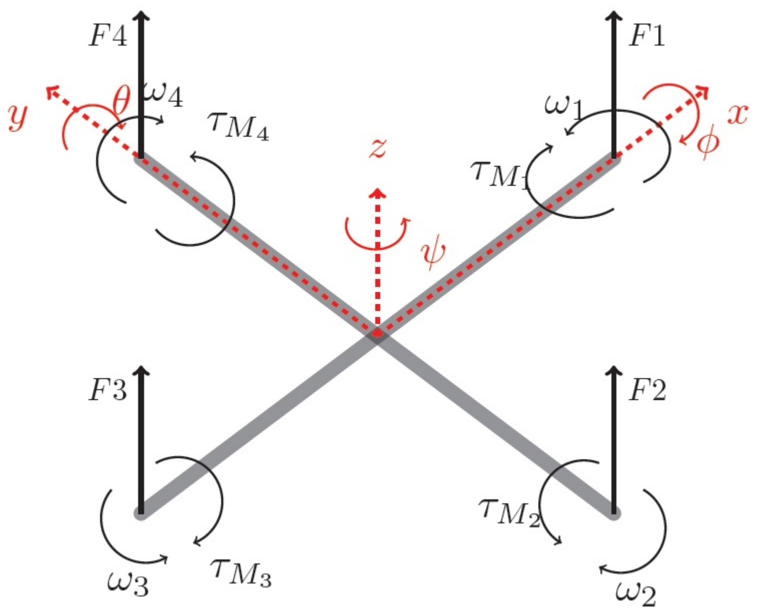

Figure 3.2: An idealized rendering of a quadcopter, detailing the most important parts of the mechanics of its construction and the definition of its body reference frame.

pellers cause the net angular momentum generated by the UAV to remain at a net zero when hovering. This will be discussed in greater depth when discussing the calculation and control of the Yaw of the UAV, but for now the discussion of the geometry is all that matters.

3.1.1 Systems Level Design - Hover

When discussing the generalities of the system, the focus will be on the behavior at hover - when the force generated by the sum total of the propellers is equal to the force of gravity pulling the UAV back to the ground and when the torques of the system sum to 0 resulting in no tilt or roll. Mathematically, this state of the system is described by the following equations:

$$\sum F = 0$$
$$\sum \tau = 0$$

From a global perspective, it's easy to observe that the force of gravity will always be pulling the quadcopter "down" towards the earth, and the vector of gravity's force can be defined as a unit vector for the simulation.

$$F_g = \begin{bmatrix} 0 \\ 0 \\ mg \end{bmatrix}$$

Performing all of the dynamics calculations from the global frame quickly becomes difficult due to the many degrees of freedom the quadcopter has as it rotates through all 3 Euler angles. Incorporating these rotating forces directly in the global reference frame quickly devolves into a mess of trigonometric functions. Therefore the modeling of these UAVs is typically managed in a local reference frame, locked to the centroid of the quadcopter where all forces generated by the propellers are inherently normal to the plane, with only the angle of the force of gravity needed to be tracked at each step and decomposed appropriately.[9] The change at each step of the simulation in the body frame then needs to be mapped to the global frame through a series of rotation matrices that join the two relative frames.

3.1.2 Rotation Matrices

The frame locked to the orientation of the drone (here on out called the *body frame*) can be linked to the global reference frame of a non-moving observer by way of a string of rotational matrices. These matrices can be used to map everything calculated in the body frame (most critically the current forces generated by the quadcopter) and update the measured positions and rotations observed in the global frame. The mapping is applied like this:

9. Tomas Jiinec, "Stabilization and Control of Unmanned Quadcopter" (master's thesis).

$$R(\phi, \theta, \psi) = R_z(\psi)R_y(\theta)R_x(\phi)$$

with each coordinate transformation looking like

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

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

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

These rotation matrices are an incredibly powerful tool in translating information between the body and global reference frames freely, as the matrix is invertible. A vector *i* in the body frame can be calculated in the global frame through **R***i*. Inversely, a vector *x* in the global frame can be transformed into the body frame by $\mathbf{R}^{-1}x$.

Moving forward in this paper, during simple high-level summaries of transformations between frames the notation R_b^G will be used to denote the combined rotation matrix that transforms one vector from the body frame to the global frame, and R_G^b will be used for the inverse.

Armed with this ability to jump between frames, now is the time to develop the equations to describe thrust generated by the propellers.

3.1.3 Propeller Dynamics

The simulation of lift in general is an incredibly complex problem that has been discussed at length in other papers, and will not be discussed or implemented here.[10] The first focus is to develop a model that will mimic the behavior of propellers without the added complexity of fluid simulations. The key feature is the time lag between the arrival of the control signal and the actual force generated by the propeller, primarily due to the inertia of the motors and the permissible load inertia caused by the propellers. Experimentation with the model drone used for this research indicates a rough motor time lag of roughly

$$\tau = 5.4$$

which is what will be used in the simulation described here. This is unique to the drone modeled here and is only a rough estimate - if the RL models generated by this simulation were pushed to this drone in real life additional estimation would need to be done. A rough estimate is all that's needed here since the main feature of this addition is to add another element of difficulty for the RL agent to overcome when learning the system - a time lag between between the action requested and its action on the system adds a non-markovian feature to the plant of the system that the model must navigate, although no features in the network are set up to expressly understand and interpret that.

The simplified model of a propeller used in this simulation must also feature the non-linearity of the control signal and an upper threshold of thrust generated by a single motor. These are both achieved by the same tool - a scaled sigmoid function. By design this function is nonlinear over the range of possible inputs (modeled by the voltage coming into the system, described below) and naturally reaches an upper threshold at its horizontal asymptote. This gives an excellent continuous action space for the RL agent to work with without having to worry about it exploring regions where unphysical behavior can occur.

A typical sigmoidal distribution takes the following form:

$$S(x) = \frac{1}{1 + e^{-x}}$$

Knowing:

- the total weight of the example drone is 1.07 kg, which requires a total thrust

10. P A Polivanov and A A Sidorenko, "Quadcopter propeller characteristics in the oblique flow," *Journal of Physics: Conference Series* 1404 (2019): 012095, https://doi.org/10.1088/1742-6596/1404/1/012095.

of 10.49 N to stay aloft at hover

- quadcopters usually are designed to operate at 60% of their max thrust at hover

the following equation can be used to simulate the lift generated by a single motor/propeller:

$$F_t = 10000 * \frac{-2}{1 + e^{(\frac{n}{10} - 5)}} - 1338.57 \tag{3.1}$$

yielding the following response curve:

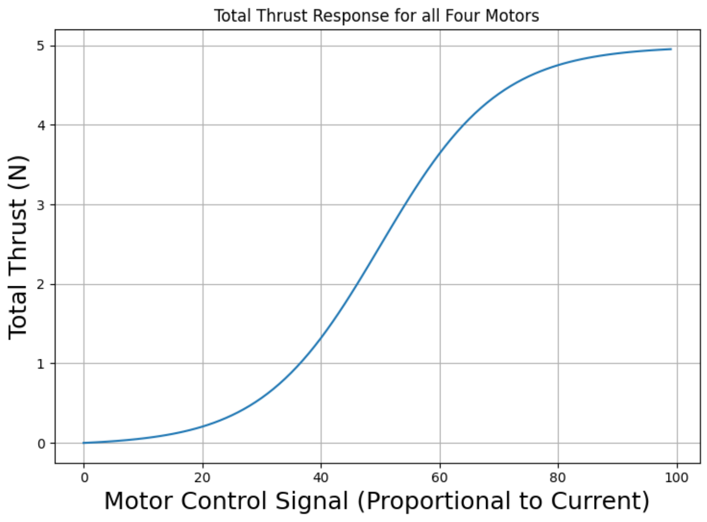

Figure 3.3: The response curve used in this simulation to circumvent nonlinear propeller fluid dynamic modeling.

3.1.4 Body Frame Dynamics

Now with the mathematical language to describe what is happening at the individual propeller level, it's now possible to quantify the motion of the drone in the body frame of reference. At any time *t* the force felt by the drone will be that of the thrust of the propellers and the force of gravity:

$$F^b_{thrust} = \sum_{i=1}^{4} F_{t_i}$$

exclusively in the +z frame of the body

$$F_g = \begin{bmatrix} -g * sin(\theta) \\ g * sin(\phi) \\ g * cos(\phi) * cos(\theta) \end{bmatrix}$$

At hover of course, the angles in the equation for the force of gravity are 0 and $F_g - F_t = 0$. Meaning there's no need for any further work. Until of course some small angle creeps into the system, which requires a long series of intricate math to correct.

3.1.5 Power Train Simulation

Having discussed the simplifications of the dynamics of the quadcopter in detail in the above sections, here briefly the estimation of the power train of the quadcopter simulation which acts as an input to the propeller thrust calculation will be discussed. While batteries are more readily understandable than the fluid dynamics around propellers, they still require a fair amount of elaborate modeling to accurately mimic the real-life counterpart in the simulated world. These relationships are mostly linearlizable as well, however oversimplifying the power behavior of the quadcopter can result in non-physical erratic behavior. Electrical loads on power subsystems can fluctuate wildly during operation, and the required instantaneous power draw of flight demands a host of specific requirements to be met to make quadcopter flight possible. On physical drones, onboard Electronic Speed Controllers (ESCs) receive desired control information from the computer and convert that into requested voltage from the battery in order to provide the needed power to its motor. These components all have response times and design constraints as with all mechatronic components, and without including some of the latency from these parts instantaneous power draw and rapidly variable power to motors would be possible by the AI. On top of looking ”unreal” when reviewing results, this could also generate policies operating on local maxima that are not comparable to anything meaningful. By modeling some linearized constraints of the power train, these corner cases can be limited.

EMF noise is generated to rapid changes in power draws in the UAV, and this noise frequently causes erroneous signals to in and downstream of the on board computer, requiring careful design of the internal layout of the quadcopter. For purposes of this simulation's simplicity, these concerns won't be directly addressed.

Working backward from the power needed at hover and thrust:

$$T = 2\rho A V_h * 2$$

$$P_h = T * V_h = \frac{T^{3/2}}{\sqrt[2\rho A]{2}}$$

where V_h is the induced velocity at hover, there is now a relationship to calculate for the power needed at hover P_h.

The power needed by the motor is more dependent on engineering design specs which will ultimately need to be estimated as direct characterization of these parameters (Torque proportionality coefficient, back emf coefficient, and the Thrust proportionality coefficient) is out of the scope of this paper. The power required by the motor can be described by:

$$V = IR_m + K_v\omega$$

$$\tau = K_t(I - I_o)$$

$$P_m = IV = fracK_vK_\tau K_t * T\omega$$

$$P_m = \frac{K_vK_\tau}{K_t} * T\omega$$

Where ω is the angular velocity of the propeller at hover. This means at hover:

$$\frac{K_vK_\tau}{K_t} * T\omega = \frac{K_vK_\tau}{K_t} * T\omega$$

$$T = K_t\omega^2$$

Knowing the required voltage for quadcopter operation, and knowing that the real-life drone counterpart was designed for a 3S 11.1V battery, a Tattu-brand equivalent model had following specs 3.1.5:

Table 3.1: Simulated Battery Properties.

	Units	Value
Capacity	mAh	450
Voltage	V	11.1
Discharge Rate	C	75
Max Burst discharge Rate	C	150

While these specs are primarily for estimating flight time, they also show how the battery will perform under load during normal operation. Controlling for unnatural current draw in the final simulation constrains the RL agent to a more natural appearing solution space.

3.1.6 Inertia and Rotation effects

So far the quadcopter has been treated as if it were a flying powered point source, but in this section the torque and rotational effects of propellers will be considered.

Calling the rotational moment of inertia J_b and looking at the free body diagram in section 3.1

$$J_b * \dot{\omega} = \mathcal{T}_m - \mathcal{T}_{gyro} - (\omega \times J_b \omega)$$

with the last term with a cross product being the Coriolis ”force” seen by the drone from its perspective.

$$J_b\dot{\omega} = \begin{bmatrix} J_x & 0 & 0 \\ 0 & J_y & 0 \\ 0 & J_z & 0 \end{bmatrix}$$

$$= \begin{bmatrix} \ddot{\phi} \\ \ddot{\theta} \\ \ddot{\psi} \end{bmatrix} (\omega \times J_b \omega)$$

$$= \begin{bmatrix} \dot{\theta}\dot{\psi}(J_z - J_y) \\ \dot{\psi}\dot{\phi}(J_x - J_z) \\ \dot{\theta}\dot{\phi}(J_y - J_x) \end{bmatrix}$$

As an undercontrolled system (6 degrees of freedom without 4 motors) it can be confusing looking at a free body diagram of a quadcopter and trying to figure out how ψ, the rotation around the Z-axis of the quadcopter, could possibly be controlled with the standard geometry of the propellers. Cleverly, original designers realized this angle could be controlled through the use of the change in the angular momentum of paired motors, cause a knock on effect in the body of the quadcopter to cause it to spin around its vertical axis.[11] This does result in slower and less reliable control than in other directions, but as ψ is often seen as less critical this is the commonly accepted method. The motor moment of inertia J_m can be defined by:

$$J_m\dot{\omega} = \mathcal{T}_m - \mathcal{T}_\psi$$

From a Newtonian approximation of air resistance

$$Drag = \frac{1}{2}\rho C_D A * (\omega R)^2$$

it's shown that the torque due to drag around ψ is:

$$\mathcal{T}_\psi = 1/2\rho C_D A(\omega_T R)^2$$
$$== \quad K_D({\omega_1}^2 - {\omega_2}^2 + {\omega_3}^2 - {\omega_4}^2)$$

which means rotational torques in other euler angles can be expressed as:

$$\mathcal{T}_\phi = lK_T({\omega_4}^2 - {\omega_2}^2)$$
$$\mathcal{T}_\theta = lK_T({\omega_1}^2 - {\omega_3}^2)$$

In aggregate the torque experienced on the quadcopter body by the motors can

11. Robert Niemiec and Farhan Gandhi, "Multirotor Controls, Trim, and Autonomous Flight Dynamics of Plus- and Cross-Quadcopters," *Journal of Aircraft* 54, no. 5 (2017): 1910–1920, https://doi.org/10.2514/1.C034165.

be described as

$$\mathcal{T}_m = \begin{bmatrix} lK_T({\omega_1}^2 - {\omega_3}^2) \\ lK_T({\omega_4}^2 - {\omega_2}^2) \\ K_D({\omega_1}^2 - {\omega_2}^2 + {\omega_3}^2 - {\omega_4}^2) \end{bmatrix}$$

The Gyroscopic torque from the first equation in this section isn't present in ψ and can be described as:

$$\mathcal{T}_{gyro} = \begin{bmatrix} \dot{\theta} \\ -\dot{\phi} \\ 0 \end{bmatrix} J_r \sum_{i=0}^{4} (-1)^{i+1} \omega_i$$

$$= \begin{bmatrix} J_r\dot{\theta}(\omega_1 - \omega_2 + \omega_3 - \omega_4) \\ -J_r\dot{\phi}(\omega_1 - \omega_2 + \omega_3 - \omega_4) \\ 0 \end{bmatrix}$$

3.1.7 Putting it all together

Having now defined all of the forces at play on the quadcopter, all that's left is to define the discrete update steps that occur at each time step in the simulation. The tracked state (x) of the simulation features twelve elements:

$u, v, w \rightarrow$ the x,y,z velocities of the body frame

$p, q, r \rightarrow$ the angular velocities of the body frame

$X^G, Y^G, Z^G \rightarrow$ the x, y, z coordinates in the global frame

$\phi, \theta, \psi \rightarrow$ the Euler angles of the quadcopter in the global frame

Part in parcel, there are derivative quantities of all the elements in the state calculated during each update step in order to update the current positions and angles that are needed to visualize. The update state ($\dot{x}$) contains:

$\dot{u}, \dot{v}, \dot{w} \rightarrow$ the linear accelerations of the body frame

$\dot{p}, \dot{q}, \dot{r} \rightarrow$ the angular accelerations of the body frame

$$\dot{X^G}, \dot{Y^G}, \dot{Z^G} \rightarrow \text{the x, y, z velocities in the global frame}$$

$$\dot{\phi}, \dot{\theta}, \dot{\psi} \rightarrow \text{the Euler angles of the quadcopter in the global frame}$$

The starting conditions of these vectors will be discussed in more detail in section 5.2, but for now know the update equations can be described by:

$$\begin{bmatrix} \dot{X^G} \\ \dot{Y^G} \\ \dot{Z^G} \end{bmatrix} = R_b^G \begin{bmatrix} u \\ v \\ w \end{bmatrix}$$

where:

$$R_b^G = \begin{bmatrix} cos(\psi)cos(\theta) & cos(\psi)sin(\theta)sin(\phi) - cos(\phi)sin(\psi) & cos(\phi)cos(\psi)sin(\theta) + sin(\phi)sin(\psi) \\ sin(\psi)cos(\theta) & sin(\psi)sin(\theta)sin(\phi) + cos(\phi)cos(\psi) & cos(\phi)sin(\psi)sin(\theta) - sin(\phi)cos(\psi) \\ -sin(\theta) & cos(\theta)sin(\phi) & cos(\phi)cos(\theta) \end{bmatrix}$$

$$\begin{bmatrix} \ddot{X^G} \\ \ddot{Y^G} \\ \ddot{Z^G} \end{bmatrix} = \begin{bmatrix} 1/m[-(cos(\phi)cos(\psi)sin(\theta) + sin(\theta)sin(\psi))F_T^b - K_d x\dot{X^G}] \\ 1/m[-(cos(\phi)sin(\psi)sin(\theta) - sin(\theta)cos(\psi))F_T^b - K_d y\dot{Y^G}] \\ 1/m[-(cos(\phi)cos(\theta))F_T^b - K_d z\dot{Z^G} + g] \end{bmatrix}$$

$$\begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} 1 & sin(\phi)tan(\theta) & cos(\phi)tan(\theta) \\ 0 & cos(\phi) & -sin(\phi) \\ 0 & \frac{sin(\phi)}{cos(\theta)} & \frac{cos(\phi}{\cos(\theta)} \end{bmatrix} \begin{bmatrix} p \\ q \\ r \end{bmatrix}$$

$$\begin{bmatrix} \ddot{\phi} \\ \ddot{\theta} \\ \ddot{\psi} \end{bmatrix} = \begin{bmatrix} \frac{1}{J_x}[(J_y - J_z)qr - J_r q(\omega_1 - \omega_2 + \omega_3 - \omega_4 + lK_T(\omega_4^2 - \omega_2^2)] \\ \frac{1}{J_y}[(J_z - J_x)pr - J_r p(\omega_1 - \omega_2 + \omega_3 - \omega_4 + lK_T(\omega_1^2 - \omega_3^2)] \\ \frac{1}{J_z}[(J_x - J_y)pq - K_d(\omega_1 - \omega_2 + \omega_3 - \omega_4)] \end{bmatrix}$$

With all of the above equations,forces generated by the propellers at any timestep can be quickly propagated to both updated reference frames. This completes the complex but computationally efficient implementation of the quadcopter simulator used by this RL training algorithm.

3.1.8 Controller Design

The control system used by the quadcopter traditionally features cascading PID loops designed to quickly reach target setpoints of the attitude of the quadcopter, either set by an operator using a controller or an autopilot. This approach has changed over the years and can be approached in a number of different ways, but for the purposes of this paper one of the more simple and straightforward approaches was explored rather than the state of the art. As will be shown later in the paper, this provides rapid and accurate control while remaining entirely linear, which makes for a more illuminating contrast to the methods used by the RL-trained agent.

At a high level, a PID control loop is a remarkably simple and computationally lightweight method to transform the error of a variable into constructive corrections to the control signal.[12]

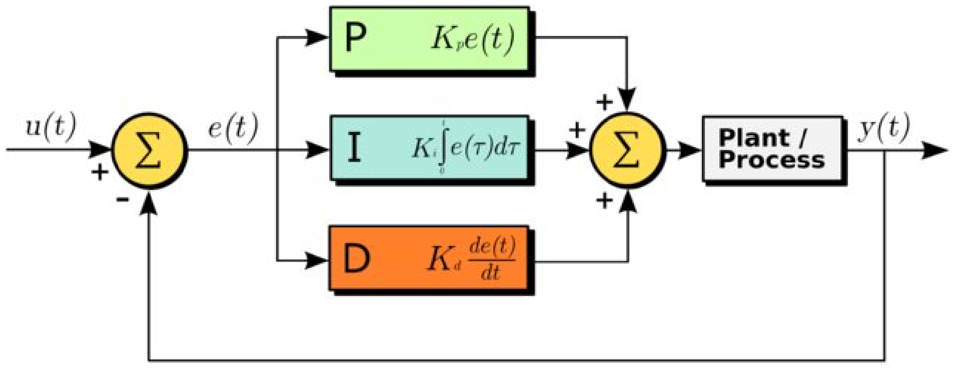

Figure 3.4: A simple look at a traditional PID loop.

At any time in the operation of a controlled machine like a quadcopter, a de-

12. Chinari Subhechha Subudhi and D. Ezhilarasi, "Modeling and Trajectory Tracking with Cascaded PD Controller for Quadrotor," International Conference on Robotics and Smart Manufacturing (RoSMa2018), *Procedia Computer Science* 133 (2018): 952–959, https://www.sciencedirect.com/science/article/pii/S1877050918310275.

fined error function can be calculated to quantify how far away the current state of the machine is from ideal (the error (e(t)) - for example a drone with a non-zero value of ϕ when it is supposed to be hovering, causing the quadcopter to drift. The value of this angle ϕ can be compared against the setpoint (u(t))to obtain this error, which then can be put into Proportional, Integrative, and Derivative filters (hence PID) that have relative "strengths" or gains (K_P, K_I, K_D) that output a new control signal. This value is entered into the plant of the system, which in this application is a simulation of the ESCs, Battery, Motor, and Propeller dynamics until a new angle ϕ can be calculated. This value is fed back into the PID controller (hence the term *feedback loop*) where the cycle continues again. It's through careful tuning of the gains of this loop that the process yields optimal update rules to the propellers at each time step, creating a quadcopter that is capable of achieving any combination of positions and angles within spec in relatively quick and stable order.

This feedback loop is unique to each control variable present in the quadcopter, meaning that the gains K_P, K_I, K_D need to be tuned and used each time step for all of the 6 degrees of freedom present in a quadcopter to achieve stable operation. The design of these interacting loops needs to be thought through as a result, as quadcopters are undercontrolled and multiple loops will require updates to multiple propellers every update. Additionally, the relative speed of the loops needs to be considered, as some loops require the output of other loops to perform optimally (i.e. if the drone has an non-zero ϕ causing it to drift off the X target, the ideal restorative angle $\tilde{\phi}$ will be different than if the drone just needs to restore its angle to 0). As a result, the following cascading PID loop design was used to control the drone:

The cascading model uses the output of the x and y position control loops as setpoints for the ϕ, θ loops, which allows the whole quadcopter to target its ultimate location. To make this happen, fine tuning of the gains for each coordinate was carried out in the simulated environment under a few conditions, looking to achieve what appeared to be stable and fast control.

As PID gain tuning is an optimization problem, there's no inherently correct solution for best control. For real life applications, the tuning is usually performed iteratively around the initial settings. Experienced drone operators have developed a feeling for the effects of tuning, looking at overshoot and response time to opti-

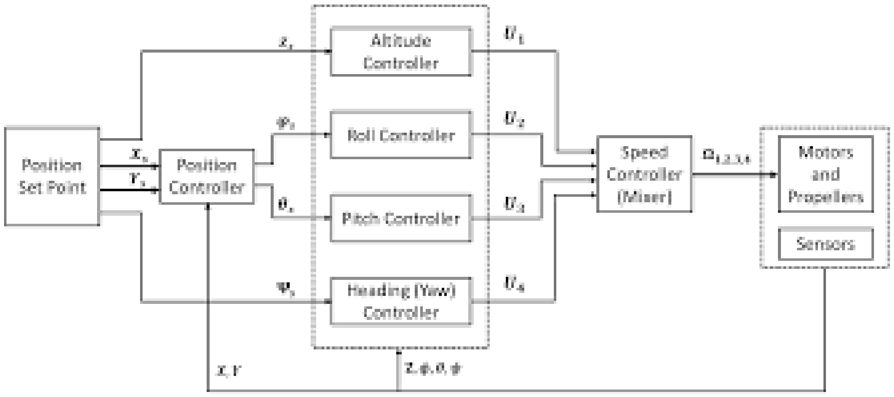

Figure 3.5: An expansion of the PID loop discussed earlier, showing the interconnection of the outer global position control frame and the inner body reference control.

mize each of the gains in all the axes.

Chapter 4

Reinforcement Learning

The focus of this paper is of course to design and build a complete and robust agent capable of attitude control of the described UAV in any condition, and then comparing the fine points of RL-trained movement to traditional optimal controls. To that end this section discusses the theory behind Reinforcement Learning (RL) which has been applied to this goal.

4.1 About Reinforcement Learning

Reinforcement Learning (RL) is a topic in Machine Learning that is rapidly gaining prominence in the field, frequently at the center of attention for new and exciting developments.[1] At its core RL applies the techniques and tools developed in earlier disciplines of machine learning to problems requiring constant decision making.[2] RL excels in Markovian environments where the only needed information is what is immediately quantifiable in what is called the *state* of the system. As a result, these tools are frequently used in real-time game environments. There are many examples of researchers and hobbyists using RL to have computers learn

1. Amit Kumar Mondal, “A Survey of Reinforcement Learning Techniques: Strategies, Recent Development, and Future Directions,” *CoRR* abs/2001.06921 (2020), arXiv: 2001.06921, https://arxiv.org/abs/2001.06921.

2. Arthur Delarue, Christian Tjandraatmadja, and Ross Michael Anderson, “Reinforcement Learning with Combinatorial Actions: An Application to the Capacitated Vehicle Routing Problem,” in *Advances in Neural Information Processing Systems* (2020), https://proceedings.neurips.cc/paper/2020/file/06a9d51e04213572ef0720dd27a84792-Paper.pdf.

optimal strategies in a number of games, from simple classics like pong to more complex modern examples like Starcraft.[3] Since the problem at the heart of this paper can be phrased as if it were a video game(i.e. what inputs on a controller are needed to get this quadcopter to fly ideally), RL was an obvious choice in tackling this challenge.

RL is fundamentally concerned about the creation of an autonomous *agent* that is capable of receiving information about its environment, performing the action it believes to be appropriate, and updating its behavior based on the results.[4]

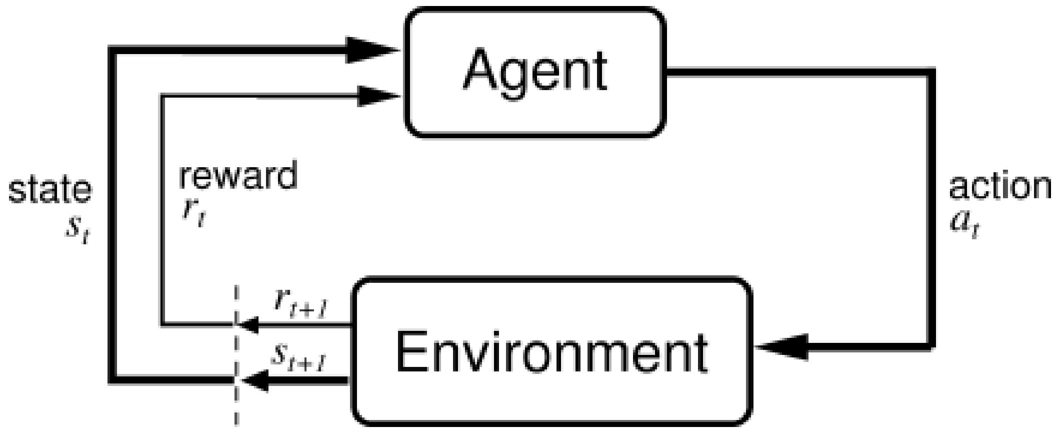

Figure 4.1: An illustration of the flow of a traditional RL learning problem. The agent suggests action a_t based off of state s_t, which causes the environment to update to new state s_{t+1} with new reward r_{t+1}.

The agent's perception of the environment comes in the form of a state vector $\mathcal{S}$, which is an array of values that can map to any data present in the environment at time *t*. In a pong-style environment, it would contain the image data of the frame at time *t*, leaving it up to the agent to develop an understanding of what the value actually means. The action the agent is able to perform comes in a vector

3. Oriol Vinyals et al., "Grandmaster level in StarCraft II using multi-agent reinforcement learning," *Nature* 575 (2019): 350–354.

4. Maxim Lapan, *Deep Reinforcement Learning Hands-On* (Packt Publishing, 2018).

defined in the creation of the agent that correlates to the input the environment is expecting. In a pong-like example, it could be as simple as a single number that reads

$$Actions = \begin{bmatrix} 1 \\ 0 \\ -1 \end{bmatrix} \begin{array}{c} \text{for move up} \\ \text{for don't move} \\ \text{for move down} \end{array}$$

After the agent performs its action, it's up to the environment to calculate the effects caused by this action and update the new state $\mathcal{S}^{'}$. As part of this update step, the environment must also "grade" the agent's choice of action by means of a reward function, which must be defined uniquely for the challenges the agent is trying to overcome. In this pong environment, the reward function could be along the lines of a positive value if the paddle moves towards the ball's path, or a negative value if the paddle moves in the wrong direction.

4.2 Markov Decision Process

As described above, RL performs best when certain properties are consistent with the framework of a Markov Process. Critically, it is needed to describe the current state of the system with all necessary information needed to make the best decision, rather than assume something will be remembered or propagated forward from a previous state. In this way the problems best solved by MPs are called *memoryless*.

An extension of the MP is called the *Markov Decision Process*. The critical features of a MP are:

- State s_t for all possible states within State Space $\mathcal{S}^*$. The State Space is a set that contains all states possible for the environment in question. For state spaces representing the physical world, assumptions must be made to constrain this space to a possible digital representation. This means that the *observation space* of continuous environments are limited to a certain resolution and size for the issue at hand.

- Action a_t in Action Space $\mathcal{A}$ represents the action taken by an agent at any given time. Like the observation space, the action space can also be limited to a discrete space with only ”valid” inputs being used. Typically out-of-bounds choices in action are clipped to the maxima or minima.

- r_t is the reward estimated by the reward function $\mathcal{R}$ at some instantaneous time *t*. Rewards are issued after every action during an episode, and then are typically summarized separately to describe the quality of the episode/batch. The instantaneous reward is typically a single value that depends on the designed reward function for the environment in question.

The interplay between the agent and the environment form the backbone of RL, and create the framework to which machine learning is applied. As the agent performs thousands of actions within the environment, the history of its behaviors and rewards are fed into the optimization algorithms at the heart of the RL setup. This iterative process is responsible for the training that develops the agents capable at navigating the environments as efficiently as possible.

The transitions between all of these key variables within the Markov Process make up the Markov Chain - a map of all the decisions and results of those decisions over the course of the process at hand. It’s this chain that is of interest to the RL algorithms responsible for developing the agent - it’s not just a matter of did the agent do the right thing, but how, how quickly, and why.

4.3 Gamma Discounted Returns

Part of the above Markov Chain is the array of reward values that the agent generated over the course of the episode. The definition of the reward function will be discussed later in section 5.3, but for now understand that it is a single value that describes how good an action the agent took at time *t*. The reward array can be turned into a **return** by

$$G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

with γ being the **discount factor**. The result of this function is inherently less intuitive than the rewards array, but provides a useful knob for controlling the perception of the agent as it learns its activity. A return calculated when $\gamma = 0$

is just the instantaneous reward at time *t*, prioritizing immediate rewards. When $\gamma = 1$ the return is instead a sum of all rewards generated that episode, effectively favoring the long history of rewards instead of anything in the short term. By using a $\gamma \epsilon [.9, .99]$ an agent designer can usually find a happy medium - an agent that wants to be successful immediately, but can sacrifice short-term success for long-term goals.

4.4 Deep Learning Methods

Deep learning is a field of study in data science that focuses on building high-level features up from raw inputs. The ”deep” in deep learning describes the large number of in-between ”hidden” layers in these networks that allow them to serve as a universal classifier, over a single layer perceptron that can only produce linear variants. These models are based on the concept of Artificial Neural Networks (ANNs), which are really a buzzword for an interesting branch of linear algebra that yields powerful results.

The basis of an ANN is called a *neuron* - a single mathematical unit that receives some input, transforms it based off of some internal rules, and passes it forward to the next neurons in its network.

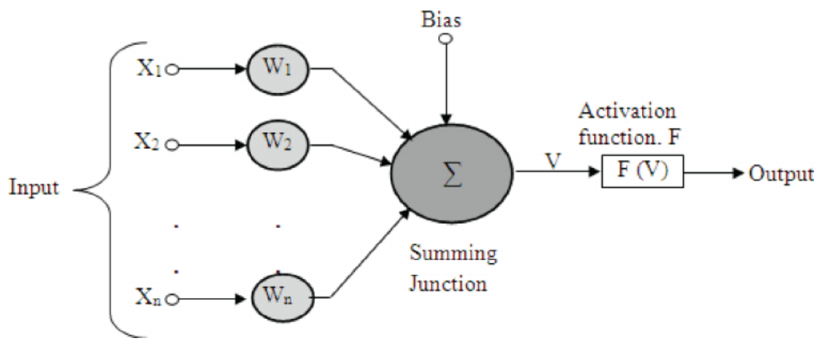

Figure 4.2: An illustration of a single neuron within an ANN. Inputs are its input elements and W are the relative weights for each input.

As one could imagine, the linear combination of a large number of elements as shown in figure 4.2 could create a new unique value that could help identify interesting features in the data, however that new value isn't necessarily understandable to a human reviewer, as the output may still have to go through hundreds of other neurons before it finally is aggregated to a human-readable number. This is why these techniques are frequently considered to be *black box* tools. Unlike some other models that can be tweaked and fine-tuned at any point with human review, the individual weights in a model can be so obscure in their effect on the output that tuning outside of the typical update step is effectively not possible. As a result, the tuning of the weights and the bias of the neuron occurs iteratively by an update function over many millions of calculations, as the desired behavior of this specific neuron gets tuned to produce ”high” outputs only when specific features in the input data occur.

The activation function shown on the right side of the figure above is another critical design choice when setting up NN models. These activation functions also have a biological corollary in the *Action potential* of a cell - the neuron's ability to start a chain reaction in its axon based on its input. In this ANN system, the activation function acts similarly, to describe the range of values needed by the neuron in order to actually contribute a non-zero value to the other neurons in deeper layers it's connected to. Interestingly, for as essential as these functions are (and for how complex it was to determine potentially valid types of activation functions) the most common functions used today are both few in number and fairly simple. The two used in the model described in this paper are the **ReLU** - Rectified Linear Unit - and **sigmoid** activation functions. ReLU functions transform the neuron input as follows:

$$\phi(v) = max(0, a + v^{'}b)$$

where the sigmoid activation function follows the same relationship used in the propeller modeling described earlier:

$$\phi(v) = \frac{1}{1 + e^{-v^{'}}}$$

The ins and outs of these functions won't be described in detail here, but suffice it to say that they are frequently experimentally shown to perform ideally for

ANN training for a wide variety of problem classes and a safe bet for this application.

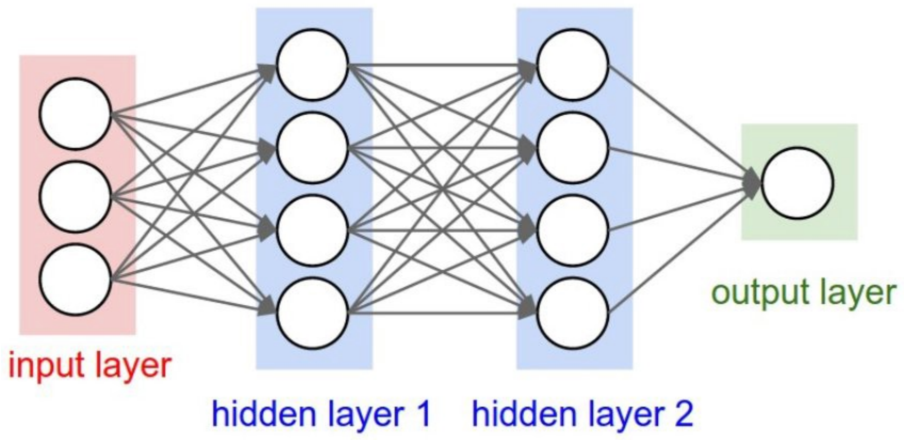

Figure 4.3: A high-level visualization of a densely connected neural network, showing how every node in a layer connects to every other neuron in the layers preceding and following.

While there are many ways to connect the neurons in between layers, the illustration above shows the main connection type used in this paper - *Dense*. Using this method every neuron from one layer connects to every other neuron in the layers on either side of it, allowing for complex feature identification to be possible.

Having established the basics of a neural network architecture, now is a good time to discuss how all these weights and biases are found for a specific application, through a process called training. First off, let's define a loss function:

$$\mathcal{L} = -\hat{A}(s,a) log \pi(s|a)$$

A loss function is used to quantify how well a model has just performed, looking at the result. A simpler and more easily understood version of a loss function is that of a least-squared function, which just compares the predicted value to the actual data, which is then minimized by updating whatever regression model by updating the predicted value. The more complex loss function shown above is needed in this application, as the goal is to update a fairly complex policy agent π which decides on action *a* given state *s*. The Advantage Function $\hat{A}$ represents the quality of the action and will be discussed further in the next section. Ultimately, the training optimization seeks to minimize this loss while maximizing the reward found at each timestep.

The gradient of this loss function is possible to be solved for with respect to each of the individual layers of the network, unlocking the main power of ANN approaches, commonly referred to as *backpropagation*. By estimating the change in the loss with regard to the weights of each of the layers of the network, the small and targeted modifications to these weights can be applied, causing the whole network to stochastically descend to a region where the loss is lowest. This comes with additional hyperparameters of the system that are commonly discussed with ANN, such as learning rate, which needs to be selected at the start of the training process. The details of this optimization won't be provided here as it's fairly dense, but it yields the policy update step:

$$\theta_{t+1} = \theta_t + \alpha \hat{A}(s, a) \div_\theta log \pi_\theta(s|a)$$

Providing a slow and targeted update of policy parameters θ. This drives the network towards a configuration best suited to producing the minimum loss, scaled by the designed rate of convergence α.

4.5 Actor-Critic Methods

The RL focus of this paper will be on a specific subset of RL algorithms called Proximal Policy Optimization (PPO), a subset of a class of methods known as **Policy Gradients**. These tools focus on directly updating the policy of the agent, that is - directly updating how the agent makes its decisions at any given state. In comparison to the other main branch of RL called Q-learning, these approaches do not directly evaluate a (value, action) pair, but instead output a statistical probability of all actions. These methods are excellent in cases where the action space is continuous, making simpler but less adaptive solutions less viable.

The goal of all policy gradient approaches is to maximize the expected reward through small updates of K policy parameters $\Delta\theta$ for $\theta \in \mathbb{R}^K$.

$$J(\theta) = \frac{\mathbb{E}}{r(\tau)}$$

These policy gradient methods are built off the comparison of the discounted rewards (section 4.3) across many different action sets. By subtracting the mean total reward over the entire training session of the agent from the current discounted reward, there is a naturally updating baseline level of expected reward given the current state of the agent, meaning that small relevant changes in behavior, even if only returning a slightly less negative number, encourages the same policy update.

A stronger approach to the above methods is to instead use the value $V(s)$ of the state as a baseline, rather than the mean reward. This would provide a more nuanced and flexible approach to maintaining policy gradient variance, however - using strict PG approaches sidesteps the estimation of the value of the state in favor of better understanding the ideal policy gradient. The trick is to utilize another deep learning network in conjunction with the policy network in order to estimate the value of the system at each observation.

These side-by-side networks receive the same input and can even share a few layers, but ultimately split off to estimate $\pi(a|s)$ and $V(s)$ (Actor and Critic) respectively, for use in the gradient update step. Other considerations also crop up with this change that won't be discussed in detail here, such as changes to the loss function, entropy, and multiple environments (for Asynchronous Actor-Critic).

These Actor-Critic (A2C) networks provide a number of benefits over standalone policy and value-gradient networks, including better action selection, convergence speed, and stability. To continue improving on these outcomes though, the modeling done in this paper will be performed with more recent developments to the A2C algorithm, known as Proximal Policy Optimization (PPO). The main feature included in this updated algorithm is a built-in surrogate clipping function that stops the policy gradient update from ever changing too quickly, preventing erroneous exploration into obviously worse action spaces.

In addition to the policy update discussed earlier, the value network also needs

```
Algorithm 1 Q Actor Critic
Initialize parameters $s, \theta, w$ and learning rates $\alpha_\theta, \alpha_w$; sample $a \sim \pi_\theta(a|s)$.
for $t = 1 \dots T$: do
    Sample reward $r_t \sim R(s, a)$ and next state $s' \sim P(s'|s, a)$
    Then sample the next action $a' \sim \pi_\theta(a'|s')$
    Update the policy parameters: $\theta \leftarrow \theta + \alpha_\theta Q_w(s, a)\nabla_\theta \log \pi_\theta(a|s)$; Compute
    the correction (TD error) for action-value at time t:
        $\delta_t = r_t + \gamma Q_w(s', a') - Q_w(s, a)$
    and use it to update the parameters of Q function:
        $w \leftarrow w + \alpha_w \delta_t \nabla_w Q_w(s, a)$
    Move to a $\leftarrow a'$ and s $\leftarrow s'$
end for
```

Figure 4.4: A high level overview of actor critic training methodology

to be updated at each timestep, in order to continue producing the best estimate of policy action selection quality. Traditional update steps for the value function are described by the Bellman Equation:

$$V_\pi(s) = \mathcal{E}_\pi[G_t|s = s_t] = \mathcal{E}_\pi[\sum_{j=0}^{T} \gamma^j r_{t+j+1}|s = s_t]$$

Which estimates the expected total return G at each time step, starting from state s and following the current policy π. This is a robust method for estimating the quality of a policy repeatedly throughout the training iterations, and feeds in nicely to the Actor-Critic flow discussed earlier. By having the Critic network calculating the value and feeding that information to the Actor network, more rapid convergence were found across a variety of different problem types.

4.6 PPO Implementation

Finally, there's one more extension to policy gradients that needs to be discussed here to fully describe the approach used in this paper. Proximal Policy Optimization (PPO) is a simplification of an earlier algorithm called Trust Region Policy Optimization (TRPO), focusing on clipping the surrogate objective in a efficient way to restrict the update of policies during training to prevent runaway conditions.[5] Defining the ratio of new to old policies as:

$$r(\theta) = \frac{\pi_\theta(a|s)}{\pi_{\theta_{old}}(a|s)}$$

The PPO algorithm limits r to a range around 1; $[1-\epsilon, 1+\epsilon]$, resulting in a new objective function:

$$J^{CLIP}(\theta) = \hat{\mathcal{E}}_t[min(r(\theta)\hat{A}_{\theta_{old}}(s|a), clip(r(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_{\theta_{old}}(s|a))]$$

This shows how the choice of limiting the rate of change of the objective function is achieved, but doesn't explain why. In practice, whether or not the advantage measured from a single action is estimated to be good or bad, rapidly moving away from current setpoints introduces variance in the optimization of the agent which can cause the network to converge slowly, erratically, or not at all. By disadvantaging highly variant actions from being selected, this smooths the descent of the optimization function.

This includes updates to the loss function, detailed here:

$$L_t^{CLIP+VF+S}(\theta) = \hat{\mathcal{E}}[L_t^{CLIP}(\theta) - c_1 L_t^{VF}(\theta) + c_2 S[\pi_\theta](s_t)]$$

Where S is an added entropy bonus, experimentally noted to help encourage exploration in cases where clipping can discourage it too much.

The implementation of this algorithm was done in the *stable-baselines-3* repository, available on github.[6]

5. Perttu Hämäläinen et al., "PPO-CMA: Proximal Policy Optimization with Covariance Matrix Adaptation," *CoRR* abs/1810.02541 (2018), arXiv: 1810.02541, http://arxiv.org/abs/1810.02541.

6. Antonin Raffin et al., *Stable Baselines3*, https://github.com/DLR-RM/stable-baselines3, 2019.

Chapter 5

Methods

With the description of the algorithms and models used in this paper out of the way, it's now time to discuss the specific implementations used for the purposes of getting a quadcopter to teach itself how to fly.

5.1 Technologies, Tools, and Libraries Used

The bulk of the simulation and RL work described in this paper was carried out using custom or open source Python modules, all hosted either on Github or Pypi. Python 3.7 was used as an engine for this code, edited with PyCharm 2017. No custom C backend files were created to optimize any of the simulation, although that would have dramatically sped up the execution of the training steps. As described in section 5.5, the visualization was executed by a javascript program in browser.

5.2 Simulation Application to OpenAI Gym

As discussed in section 3, the first fundamental step of this project is to develop a robust simulation of the model in order to give the RL algorithms a space to work in during training. A number of publicly available simulations exist already online and are free to download,including an excellent one by.[1] However after experi-

1. Koch et al., "Reinforcement Learning for UAV Attitude Control."

menting with a number of different ready-made simulations, it was determined to be necessary to create a separate model for this application. There was a number of criteria that informed this decision:

- **No intention to port to hardware** - a number of papers in this space have approached this problem with the intent to replicate their result on actual hardware, porting the RL agent over to the on-board computer. This was out of the scope of what was going to be attempted here, and added too much complexity to the development of the agent.
- **Reduce Fluid Dynamics Complexity** - Hand in hand with the point above, without the need to port the RL agent to the quadcopter there is no need to calculate absolutely accurate thrust caused by the propellers. Reducing this relationship to a single control variable related to real voltage massively simplifies the calculations needed at each step, while still considering the key elements of complexity introduced to quadcopters by the use of brushless motors. Discussion on how this simplification was made can be found in section 3.1.3.
- **Optimization for Iteration** - As this simulation can be run through many millions of times, it was critical that the calculations were set up ideally for rapid GPU computation. While this isn't unique to this model,creating a custom simulation allowed for greater control in the approach to training

OpenAI is a tech nonprofit started in 2015 that works as an AI research lab, responsible for a number of great achievements in machine learning. One of these developments is the publicly available tool called OpenAI Gym - a RL framework that was developed in-house and released to the public in 2016. This established their approach as the de-facto standard to all RL problems - an interactive environment space that can accept rapid updates from a training RL agent and provide environmental feedback back to the agent. The key phases of a Gym environment are:

- **reset** - [*input: None, returns: None*] This function is called at the start of each episode, resetting all state variables back to their initial state. Any randomness needed to help generalize the RL agent is usually added in here.

In this application, the starting body-frame angular velocities *p, q,r* can start the episode with any value from -.9457718 to .9457718 rad/sec.

- **step** - [*input: action, returns: state, reward, done*] The key update step of the simulation, the step function gets called at each time step of the simulation. The estimated action of the agent gets applied and translated to machine-units by equations in section 3.1.3. This action is propagated throughout the system, and the updated position and velocities of the quadcopter are calculated. If certain conditions have failed , or if the simulation time hits a timeout condition (in this application 6 seconds), a *done* signal is returned to the agent. This ends the episode, causing the reset function to be called and the neural network to update its weights. The reward for the current time step is also calculated and returned to the agent - discussed in the next section.
- **render** - [*input: None, returns: None*] During development of things as opaque as RL agents, it is important to be able to convert as many things as possible to some kind of human-readable format to inform future decisions and just to explain what is going on. Rendering full X11 graphics every timestep can be computationally expensive and slow down training, but being able to occasionally see what's going on within the training loop is extremely helpful. In this application, the render function dumps the full 6-second state of the quadcopter into a csv file that can be visualized by a javascript app.

5.3 Reward Function Engineering

The development of a good reward function can frequently be the standalone topic in itself, as these complex and multivariate functions are responsible for creating the topology of the cost function that the PPO algorithm is working to maximize. This function needs to be able to assess the quadcopter's action at each time step and determine if its behavior is improving or not. For this project specifically, it is not generally straightforward to figure out what is ”good” behavior. Should the quadcopter's ability to stay near a specific point in the global coordinate frame be the goal? Should it receive an award if it reaches the target in a quick amount of time? Should it be penalized for exceeding a critical angle, or rewarded if it's able to recover?

A number of considerations regarding good reward function design were considered to answer these questions.

The domain of a robust reward function:

- should have values between -1 and 1
- should be relatively smooth and continuous
- shouldn't be based off of generic references

As the focus of the agent is to learn how to control the quadcopter's flight and stability, it makes sense to move away from position-centric reward and focus on angular velocity. While RL applications applied towards the study of quadcopter piloting are not uncommon, the focus here is on the attitude control. While this results in the quadcopter not necessarily finding a stable position and as a result drifts across the render, it will still have stabilized the initial thrown velocity of the quadcopter.

As a result, the main feedback the agent will receive is the current deviation from the target angular velocity in the body-frame, which in the case of stable flight is 0 (although this value could be dynamically updated to allow for flight control). However, directly feeding back this signal as error was shown not to be effective, as the angular velocity can vary wildly during early training cycles. What was shown to be more directly helpful was the improvement of the error of the angular velocity, found by

$$reward = -\sum_{i \in \{p_t, q_t, r_t\}} (0 - i)^2 + \sum_{i \in \{p_{t-1}, q_{t-1}, r_{t-1}\}} (0 - i)^2$$

Focusing on the change of angular velocity error provides a nice target for an attitude control agent to narrow in on - as long as the agent is improving its rotation it's receiving positive rewards. This also provides a possible success criteria in the initial moments of the episode, when other reward metrics would be negatively penalizing the drone due to the random offset given to it at the start.

Unfortunately, this smooth function isn't enough to get the desired behavior for the quadcopter, since it has got a few too many loopholes that allow for sub-optimal descent. Steady state oscillation error get's washed out in this function, allowing for constant oscillation around the target. Other target behaviors of the quadcopter are also not considered by their function - while the agent may find semi-stable solutions that involve over and under-saturated control signals, it's obvious to a human agent designer that these sort of features promote instability in the system. As a result, a number of additional criteria are added as penalties to the reward to suppress these sorts of actions. Specifically, these actions are:

- **Oversaturation** For each control signal set above its clip value, a negative award is added.
- **Do Nothing Penalty** For some seeds of the environment, there exists a trivial minimal solution that is achieved when the agent does nothing. To suppress this local minima a penalty is added when the agent doesn't try to do anything
- **Oscillation** When the agent starts rapidly oscillating between two values which is an odd and non-obvious state the algorithm seems to like to settle in, a penalty is applied.

5.4 Variable Learning Rates

The learning rate of the network is a commonly adjusted hyperparameter for all neural networks in use today, as the optimization that occurs at the heart of training is variable depending on the model. Based on empirical evidence, Adam optimizer was used for this application. The main input an engineer can have on the performance of an Adam optimizer is the learning rate, which is the proportion that weights are updated during training.

An ongoing issue discovered during training was the wide exploration space available to the drone at the beginning of training. Without any indication of the right direction to move to, the network would regularly spend hours or days exploring configurations that would never result in flight. This incentivized some research into variable learning rates, with the thinking that a larger rate in the beginning of training would allow for the network to more quickly find potential sub-optimal configurations that could be explored further during training with a smaller learning rate. To that end, the following generator was used to create a sinusoidal learning rate that synced with the total number of iterations, allowing 1/2 of a period to elapse during training.

$$lr = .00025 * cos(\frac{t}{2000} * \pi) + .00025$$

going from the range 0.0005 to just a float away from 0. Over the course of training this variable rate seemed instrumental in encouraging the network to find viable branching paths early, which greatly accelerated other investigations in this paper.

5.5 Render and Visualization

As noted by many other researchers in this space, the ability to render the output of the RL agent to a human-viewable GUI is not technically necessary, but immensely helpful in the troubleshooting and tuning of the many interconnecting pieces that make up this RL project. Even professional quadcopter design fundamentally comes down to the look and feel of the quadcopter in the hands of an expert. As it was, being able to plot the trajectory of the quadcopter in 3d space highlighted a number of interesting features in the behavior of the quadcopter that just weren't visible in the number of 2d plots being generated constantly.

The state of the quadcopter was already calculated as part of the simulation discussed here, and as such saving the state for a single episode was simple. In order to visualize this data, javascript code was modified to accommodate the output of the simulator.[2] This allowed for the real-time viewing of the quadcopter, which is incredibly helpful for troubleshooting and understanding the state of the agent.

5.6 Noise Addition to Step Output

During repeated training sessions, a number of false plateaus were found by the optimizer that appeared to be stuck at the boundaries of one or all of the control

2. Charles Tytler, *QuadcopterSim*, https://github.com/charlestytler/QuadcopterSim, 2018.

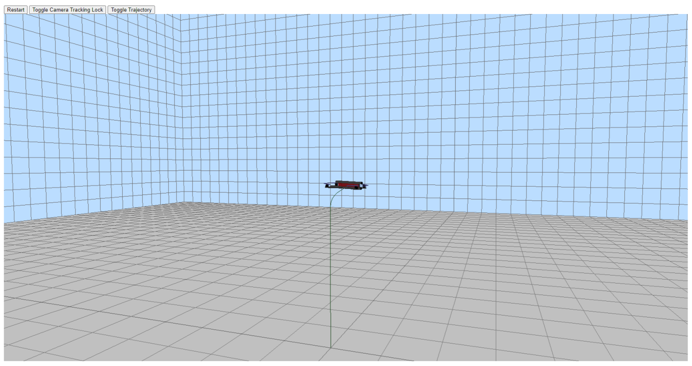

Figure 5.1: An example of the visualization of the drone from a takeoff.

inputs and no amount of time seemed to change the network's descent. There are a number of reasons why this sort of error could creep into the training cycle, mostly concerning how the boundary in allowable actions was handled in this implementation. The space allows for continuous action selection between 0 and 1, but clipping needed to occur in the environment side out of the view of the network to ensure that no non-valid inputs were allowed. Instead of troubleshooting this problem directly, an additional step was added to the step function that removed this error mode.

By injecting noise into each of the state measurements output from the environment this boundary effect caused by a network clipping at a threshold disappeared since, from the networks perspective, it wasn't hitting a hard boundary but rather a soft wall. Additionally, this incorporates another feature of testing on hardware quadcopters that was to this point ignored for this model - the inherent noise of sensors and electronics. Incorporating some noise generation to the quadcopter both improved the quality of the simulation and removed some of the sticking points.

While the amount of noise injected could have been variable, some sweeps were performed to determine a single acceptable value of no more than 5% the current variable. Gaussian white noise capped by this amount was generated and added to each of the state values reported by the environment.

5.7 PPO Hyperparameters

For the ease of replication, here are the training parameters that were found to be effective over the course of the model training attempted for this project. The starting point of these parameters were found using DOE exploration generated by open-source software.[3] These optimization algorithms work by running a limited series of episodes with 1000+ different sets of hyperparameter settings, mapping out the rough space available for viable hyperparameters. The following were the parameters used during the successful training sessions.

Table 5.1: PPO2 Hyperparameters.

Parameter	Value
Batch Size	4
Cliprange	0.2
Entropy Coefficient	0.00000444
Gamma (Γ)	.999
Lambda (λ)	0.95
Learning Rate	Function
Number of Steps	512
Noptepochs	20

3. Antonin Raffin, *RL Baselines3 Zoo*, https://github.com/DLR-RM/rl-baselines3-zoo, 2020.

Chapter 6

Results and Conclusions

After an extended iteration of RL training and hyperparameter tuning, finally functioning agents started to present themselves. During training all agents spend millions of iterations struggling to rise up out of the pit of first flight, independent of any set of initial conditions. However, after a lot of study of the potential reward functions possible for this application, the final iterations of the rewards discussed in section 5.3 started to yield agents that could transition from random exploration to actual organized flight. Once this transition was finally reached, another problem started to present itself.

As discussed earlier, PPO algorithms produce PDF distributions of the output variables, which are then sampled in order to decide on an action at any given time. This is generally a strength in model training, as it builds in an intuitive and adaptable exploration rate, which also removes arbitrary hyperparameters that are present in Q-networks like ϵ. However, this output remains a PDF even when training is complete - potentially not a problem in applications when the AI is suitably confident in its choice of action each timestep. However even after many days of training with the hardware available, the average standard deviation of each action recommended by the agent was creeping up towards .3, in effect forcing the quadcopter to fly with control signals that have signal-to-noise ratio of 30%.

The result of this behavior is apparent in the following plots - the steady state error of the quadcopter remains high, especially when compared to the near-0 steady state error of the tuned PID quadcopter equivalent. While the focus of the rest of this section will be on both control systems performance with regard to step responses, this steady state behavior does appear to temper expectations that RL-trained agents could likely outperform traditional optimal control systems in real life applications.

6.1 Single Flight Visualizations

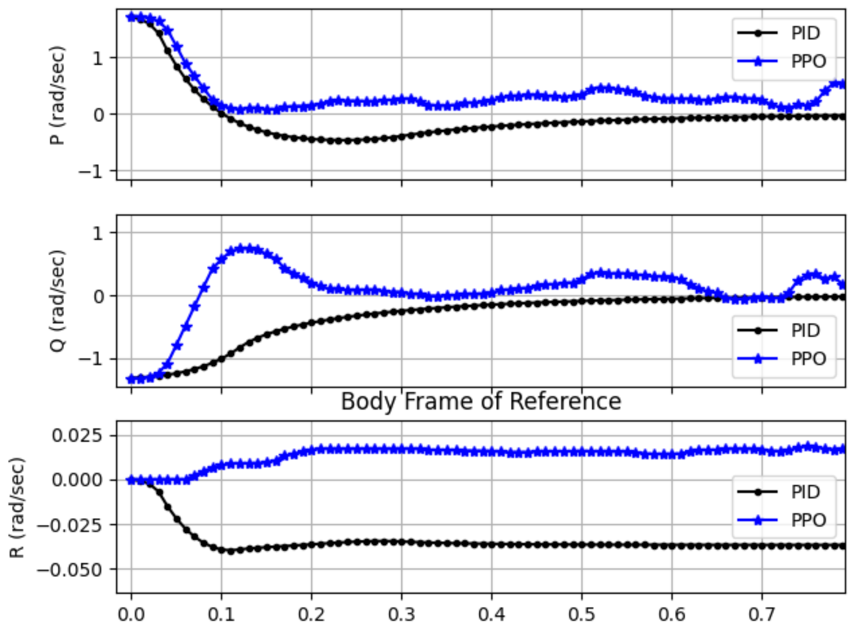

Figure 6.1: A comparison of AI vs Traditional PID control of the three radial degrees of freedom in the quadcopter body frame of dynamics. R (Yaw) is not included in the Reward function and is technically uncontrolled.

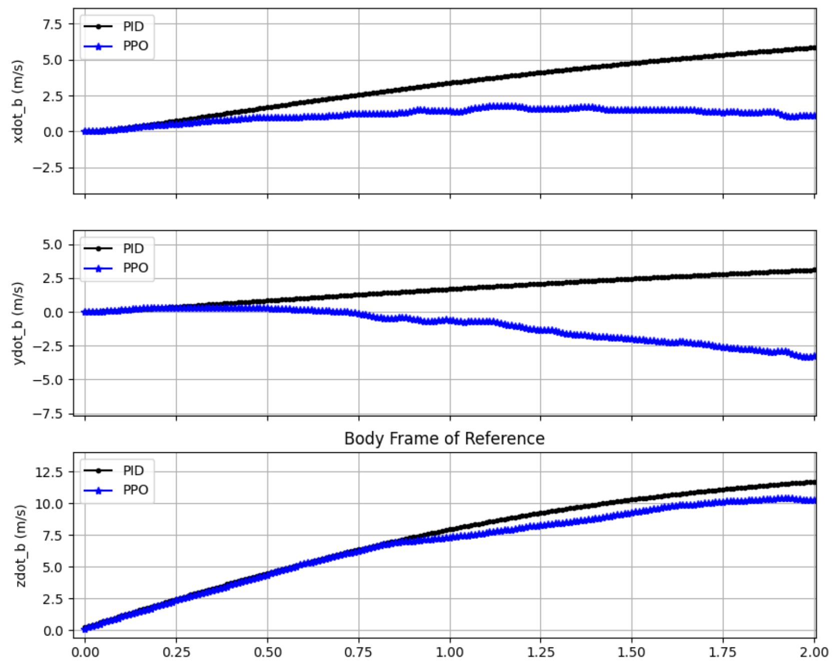

Figure 6.2: A comparison of AI vs Traditional PID control of the three linear velocity degrees of freedom in the quadcopter body frame of dynamics. These were not directly controlled by either method, so the drift shown is a consequence of the method's ability to quickly restore the quadcopter. Proportional to angle error.

The one randomly selected run of the two types of quadcopters, each starting from the same random "thrown" offset of a quadcopter with random angular and linear velocity, shows some of the interesting behavior that inspired this paper. PID tunings are good at controlling for a wide range of behavior, but are ultimately optimized for a certain range. In the case presented above, the PID is set up to quickly restore what will be called ”medium” sized error - between .75 - 1.1 rad/sec. This run the random initial errors were greater than this range, resulting in PID overshoot in P and slower than desired restoration of stability in Q.

The RL agent, on the other hand, was trained on all ”realistic” attitude step errors and as a result should be able to control quickly on a wider range of situations. As a result, the agent was able to quickly restore stability in both controlled attitude degrees of freedom, although it failed to stay stable for the reasons discussed above.

6.2 Averaged Behavior

To better see how the RL agent and the PID controller contrast, the states of hundreds of runs were combined together to produce a snapshot of each system's performance across all possible initial conditions. Immediately apparent is the consistency of the PID approach, which reliably restores the quadcopter to its target in an amount of time proportional to the size of the error. The RL agent is of course less predictable - generally able to get to the setpoint incredibly quickly, but not generally able to maintain that stability for long enough to be meaningful in its current implementation.

	P Mean (sec)	P Std	Q Mean (sec)	Q Std
Agent	0.223	0.192	0.192	0.156
PID	0.423	0.155	0.421	0.156

Table 6.1: The distribution of times it took for a RL trained agent to reach a point of minimum error for quadcopter attitude over 2000 random samples

Aggregating over enough samples, the increase in stability of traditional PID control is apparent. The RL trained agent is able to greatly reduce the initial error imparted on to the quadcopter, but can't resolve small angle errors beyond .2 rad/sec. A variety of weightings in the reward function have been tried to incentivize the RL algorithm to drive this steady state error down, but to date no system has been able to squash this small error without introducing greater issues in the quadcopter performance. This plot also questions the speed of error resolution brought up in the earlier table 6.1 - while it's true that the drone reaches a stable orientation more quickly the the PID loops on average, it clearly is accelerating too fast currently as it overshoots the target setpoint. As advances are made to improve steady state stability, this overshoot should also be reduced in a meaningful way.

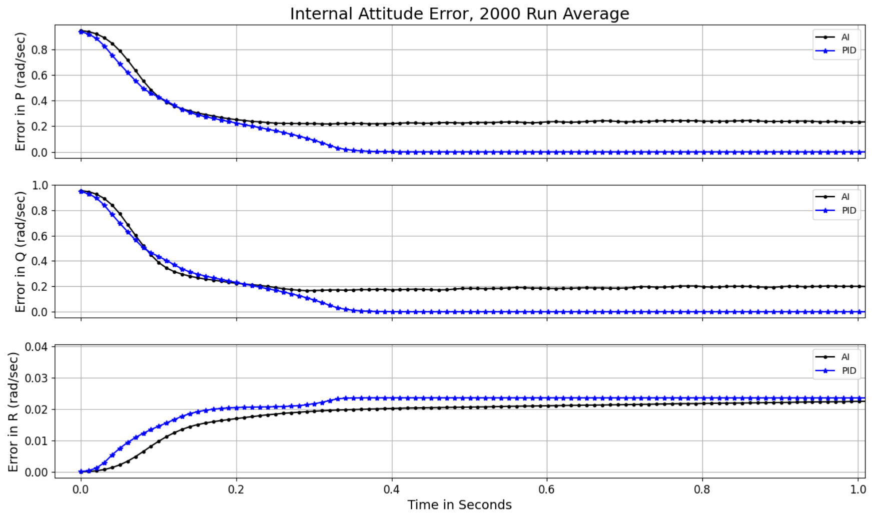

Figure 6.3: A comparison of AI vs Traditional PID control of the three radial degrees of freedom in the quadcopter body frame of dynamics, time averaged over 2000 runs with different initial conditions.

When looking at the control signal output of the two systems differences in the rate of signal change are apparent, which yields some interesting observations. As discussed in section 3.1.8, the control signal change is clipped to a max delta of 4 in order to mimic the inertia of the motor. What's interesting is that the PID controller regularly clips this value due to the large proportional gain dictating a specific setpoint to reach for the given error, whereas the AI agent better controls its output based off its learned behavior - the agent knows its inputs greater than that clipped value are likely ignored so it avoids exceeding those limits.

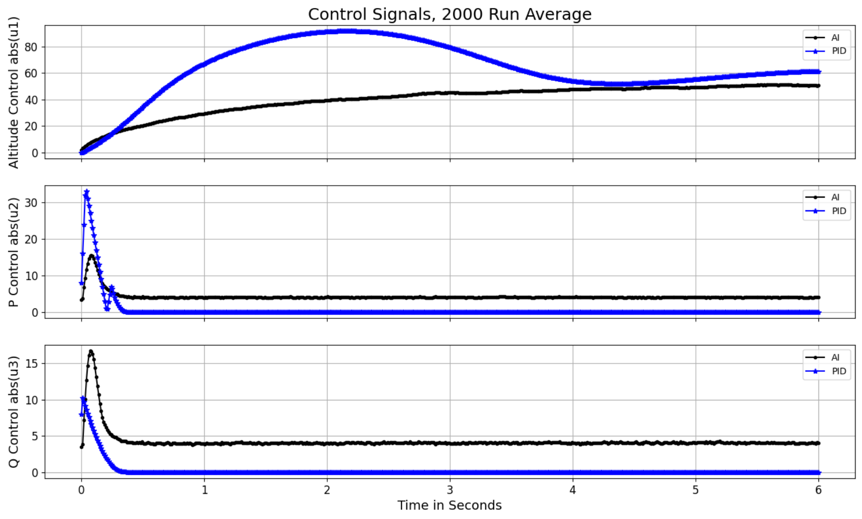

Figure 6.4: A comparison of AI vs Traditional PID control, looking specifically at the signal output of both systems. Time averaged over 2000 runs with different initial conditions.

6.3 High Error Environment

The potential versatility of RL agents is one of the main reasons for the excitement around this approach for quadcopter control. Traditional control loops are optimized for specific flying conditions, so in theory a RL model trained across a variety of flying conditions could outperform a PID loop when competing in all of these arenas. A few limitations are present in this application though:

- RL training algorithms don't respond well to arbitrary interference, when considering negotiating an agent that has to overcome constant wind forces per se. The reward loss is entangled with the agent's action, independent of there being external cause.

- Limitations in the training of the quadcopter for this paper necessitated a simpler and more direct approach, meaning that unnecessary variation of attitude error was reduced.

- Standards of this kind of testing are undeveloped, meaning there isn't an established suite of tests to put these controllers through in order to characterize their strong suits. As a result, only a subset of interesting cases are explored.

The first test cases to be explored is when constant, repeated offsets are applied to a quadcopter, in effect showcasing a windy day. A constant period was used in this case to aid in the analysis even though it clearly isn't physical. This perturbation is a normal distribution of angular rotation, of $\mu = 2.8$ and $\sigma = 1$.

6.4 Nonlinear Input

The main reason why RL approaches to traditional control problems are so attractive is due to their unintended adaptability. Traditional PID loops are famously adaptable, but feature major downsides when dealing with a nonlinear plant and rapidly varying operating points. As discussed in section 3.1, some simplifications were made in the plant simulation that ultimately reduced the complexity for both control engines. The focus of the input was also centered around initial step responses, largely to compensate for the practicalities of RL agent training - the agents are unable to differentiate between an externally changing setpoint and its own error.

These features of the models used here make the behavior during nonlinear input very interesting. The PID loops were tuned to respond to infrequent and relatively small error correction, and the RL agents were only ever taught to restore its attitude from initial offsets. Remarkably, when presented with nonlinear

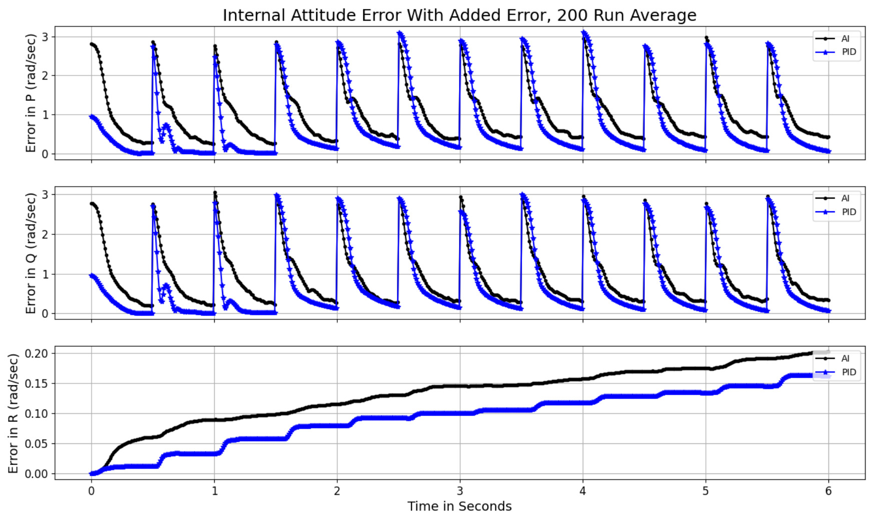

Figure 6.5: A comparison of AI vs Traditional PID control with rotational error injected regularly. Time averaged over 200 runs with different initial conditions.

chaotic motion, the RL agents are able to track in phase with the setpoint, where the PID loops end up drastically offset.

Both loops start the episode struggling to keep up with the unpredictable setpoint velocity, however in both Roll and Pitch the RL trained agent is able to get in phase with the setpoint signal and track it until the end of the episode. This run is just an exemplar run from a larger set of 100 episodes that do a better job showing the general behavior of both agents in practice.

Throughout the epoch, the RL agent is shown to perform worse in the early .5 sec of the episodes, but recovers its setpoint curve and tracks it through the end of the episode. The total factor increase in the steady state error of the RL agent is around 9.1, whereas an increase of around 17.8 was seen in the PID control. This is largely verifiable by quadcopter piloting experience, where pilots have to train themselves to ”slow down” how quickly they manipulate the setpoint controls of

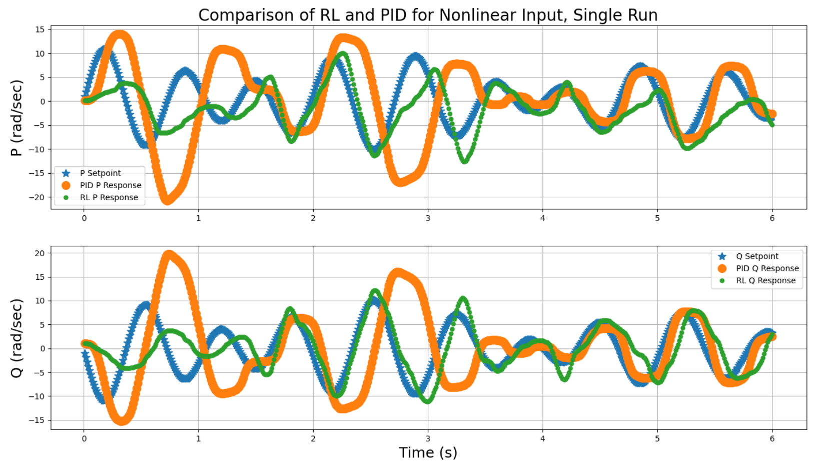

Figure 6.6: A comparison of AI vs Traditional PID control in cases of nonlinear, rapidly changing P and Q setpoints. While neither controller performs well, the AI look clearly tracks the changes more rapidly than the PID loop.

their UAVs. Rapid changes to setpoints can cause errors to grow rapidly, resulting in PID loops losing track of their targets. These issues appear to not affect RL-trained agents in the same way.

6.5 Conclusion

In this paper, quadcopters were used as a toy model to describe and compare the strengths and weaknesses of RL-trained agents and traditional PID loops as controllers for complex, nonlinear plants. In addition there was a deep dive into the methodology required to simulate a mechanism as complicated as a quadcopter, as well as the procedure required to get nascent policy-gradient RL algorithms to successfully find a useful model that results in flight. This was done by constraining the complexities of the plant down to a manageable scope that allowed for

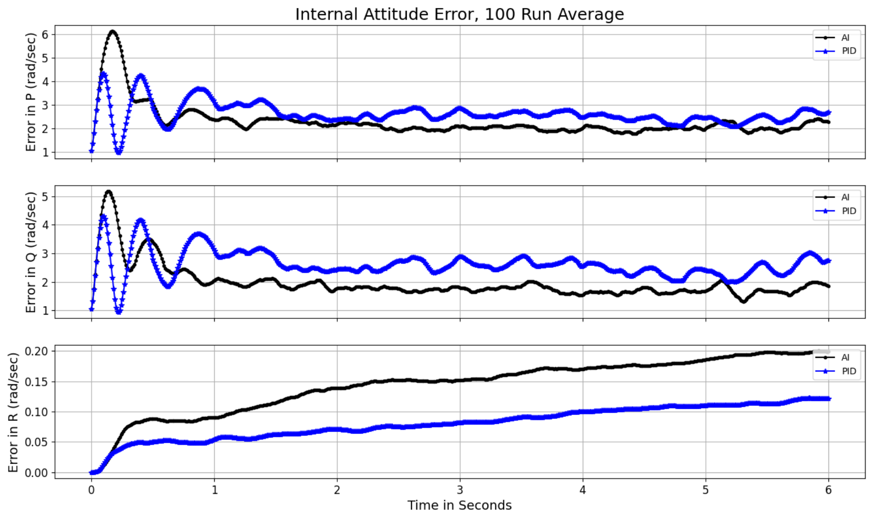

Figure 6.7: A comparison of 100 episodes of AI vs Traditional PID control in cases of nonlinear, rapidly changing P and Q setpoints. Over this averaged time window the AI controlled agent outperforms the PID loops.

rapid, repeated computation during the training of new PPO algorithms. This was largely achieved through reward function engineering, which provided a smooth and continuous curve for the training algorithm to track to a stable condition.

These control systems were focused on the problem of quadcopter attitude control, an inherently unstable and undercontrolled system. It was found that for step impulses, PID loops drastically outperform RL agents in terms of robustness and flexibility, as the plant is not so complex where PIDs are unable to perform. The RL agents were able to find a niche in controlling rapidly changing nonlinear inputs, a case in which PID loops seemed to fail.

6.6 Future Work

This work in quadcopter attitude control, which has been ongoing for the last few years, meshed nicely with previous work done in the field for managing quadcopter positions with RL-trained agents. A logical next step would be to combine the two control systems into a single state machine. This poses many interesting questions, as an RL agent wouldn't be constrained to the same cascading architecture that traditional control systems have to follow. This end-to-end control approach could yield improvements in both high-level tasks to low-level control. While by no means essential, applications in miniquad racing could be very exciting.

This paper only considered the PPO algorithm, a relatively new policy gradient approach. Many other algorithms could be used as a comparison as well, including most interestingly the very different value gradient networks, which traditionally struggle in continuous action spaces.

Lastly, the greatest downsides exhibited by the RL agents in this paper is that of steady-state error, compounding consistently over the last half of each episode. This is reduced in traditional controllers by integral terms. As a Markovian agent, the quadcopter RL pilot has no knowledge of its past error by default and as such reduced its effective integral gain. By including some integral into the system state in some capacity, this could encourage the agent to drive down steady state errors in future implementations.

Appendix A

Acronyms

(UAV)	Unmanned Aerial Vehicle
(RL)	Reinforcement Learning
(PPO)	Proximal Policy Optimization
(PID)	Proportional-Integral-Derivative
(TRPO)	Trust Region Policy Optimization
(PPG)	Phasic Policy Gradient
(MP)	Markov Process
(MDP)	Markov Decision Process
(A2C)	Actor to Critic
(PDF)	Probability Density Function

Appendix B

Model Training

The following plots detail the behavior of the PPO2 algorithm implemented in stable-baselines-3 with the hyperparameters detailed in section 5.7. In this particular example training was stopped at the highest seen performance previously however in other cases it was allowed to run for much longer, although no real improvement was seen.

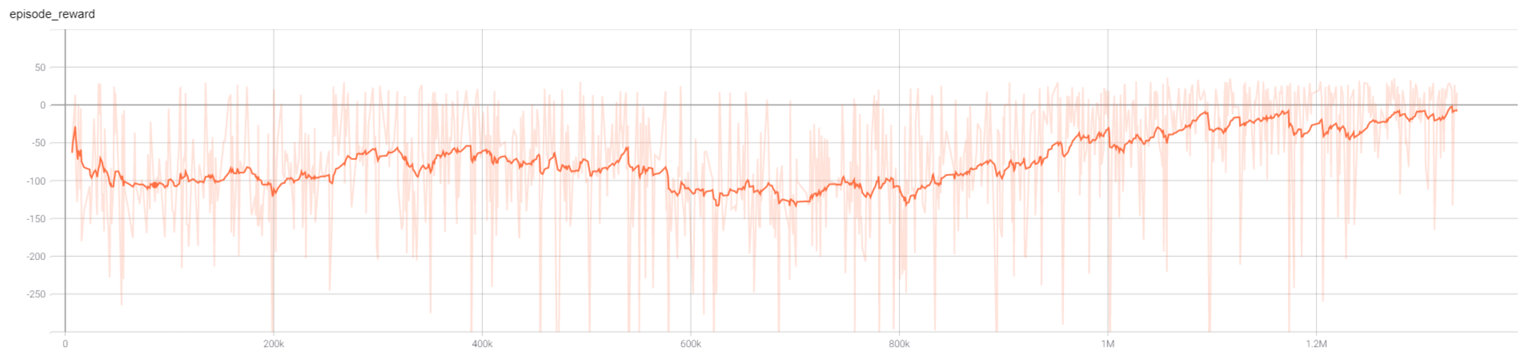

Figure B.1: Per-Episode reward plot against training time

Bibliography

Agarwal, Alekh, Sham M. Kakade, Jason D. Lee, and Gaurav Mahajan. "Optimality and Approximation with Policy Gradient Methods in Markov Decision Processes." *CoRR* abs/1908.00261 (2019). arXiv: 1908.00261. http://arxiv.org/abs/1908.00261.

Ausonio, Elena, Patrizia Bagnerini, and Marco Ghio. "A New Direction in Firefighting Systems." *CoRR* abs/2007.00883 (2020). arXiv: 2007.00883. https://arxiv.org/abs/2007.00883.

Bjarre, Lukas. "Robust Reinforcement Learning for Quadcopter Control." Master's thesis, KTH Royal Institute of Technology, 2019.

Brockman, Greg, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. "OpenAI Gym." *CoRR* abs/1606.01540 (2016). arXiv: 1606.01540. http://arxiv.org/abs/1606.01540.

Chai, Xiaoyu, Lizhang Xu, Yang Li, Jie Qiu, Yaoming Li, Liya Lv, and Yahui Zhu. "Development and Experimental Analysis of a Fuzzy Grey Control System on Rapeseed Cleaning Loss." *Electronics* 9, no. 11 (2020). https://www.mdpi.com/2079-9292/9/11/1764.

Cobbe, Karl, Jacob Hilton, Oleg Klimov, and John Schulman. "Phasic Policy Gradient." *CoRR* abs/2009.04416 (2020). arXiv: 2009.04416. https://arxiv.org/abs/2009.04416.

Delarue, Arthur, Christian Tjandraatmadja, and Ross Michael Anderson. "Reinforcement Learning with Combinatorial Actions: An Application to the Capacitated Vehicle Routing Problem." In *Advances in Neural Information Processing Systems*. 2020. https://proceedings.neurips.cc/paper/2020/file/06a9d51e04213572ef0720dd27a84792-Paper.pdf.

Haitham Bou-Ammar, Wolfgang Ertel, Holger Voos. "Controller Design for Quadrotor UAVs using Reinforcement Learning." *University of Applied Sciences Ravensburg-Weingarten*, 2010.

Hämäläinen, Perttu, Amin Babadi, Xiaoxiao Ma, and Jaakko Lehtinen. "PPO-CMA: Proximal Policy Optimization with Covariance Matrix Adaptation." *CoRR* abs/1810.02541 (2018). arXiv: 1810.02541. http://arxiv.org/abs/1810.02541.

Hussein, Amir, and Rayyan Abdallah. "Autopilot Design for a Quadcopter." *self-published*, February 2018.

Jiinec, Tomas. "Stabilization and Control of Unmanned Quadcopter." Master's thesis.

Koch, William, Renato Mancuso, Richard West, and Azer Bestavros. "Reinforcement Learning for UAV Attitude Control." *ACM Transactions on Cyber-Physical Systems*, no. 22 (Feburary 2019).

Lapan, Maxim. *Deep Reinforcement Learning Hands-On*. Packt Publishing, 2018.

Mnaouer, Hajer Ben, Mohammad Faieq, Adel Yousefi, and Sarra Ben Mnaouer. "FireFly Autonomous Drone Project." *CoRR* abs/2104.07758 (2021). arXiv: 2104.07758. https://arxiv.org/abs/2104.07758.

Mondal, Amit Kumar. "A Survey of Reinforcement Learning Techniques: Strategies, Recent Development, and Future Directions." *CoRR* abs/2001.06921 (2020). arXiv: 2001.06921. https://arxiv.org/abs/2001.06921.

Niemiec, Robert, and Farhan Gandhi. "Multirotor Controls, Trim, and Autonomous Flight Dynamics of Plus- and Cross-Quadcopters." *Journal of Aircraft* 54, no. 5 (2017): 1910–1920. https://doi.org/10.2514/1.C034165.

Polivanov, P A, and A A Sidorenko. “Quadcopter propeller characteristics in the oblique flow.” *Journal of Physics: Conference Series* 1404 (2019): 012095. https://doi.org/10.1088/1742-6596/1404/1/012095.

Raffin, Antonin. *RL Baselines3 Zoo*. https://github.com/DLR-RM/rl-baselines3-zoo, 2020.

Raffin, Antonin, Ashley Hill, Maximilian Ernestus, Adam Gleave, Anssi Kanervisto, and Noah Dormann. *Stable Baselines3*. https://github.com/DLR-RM/stable-baselines3, 2019.

Reizenstein, Axel. “Position and Trajectory Control of a Quadcopter Using PID and LQ Controllers.” Master’s thesis, Linköping University, 2017.

Shakhatreh, Hazim, Ahmad Sawalmeh, Ala I. Al-Fuqaha, Zuochao Dou, Eyad Almaita, Issa Khalil, Noor Shamsiah Othman, Abdallah Khreishah, and Mohsen Guizani. “Unmanned Aerial Vehicles: A Survey on Civil Applications and Key Research Challenges.” *CoRR* abs/1805.00881 (2018). arXiv: 1805.00881. http://arxiv.org/abs/1805.00881.

Shamshirgaran, Azin, Hamed Javidi, and Dan Simon. *Evolutionary Algorithms for Multi-Objective Optimization of Drone Controller Parameters*, 2021. arXiv: 2105.08650 [eess.SY].

Shinshi, Tadahiko, Daisuke Shimizu, Kazuhide Kodeki, and Kazuhiko Fukushima. “A Fast Steering Mirror Using a Compact Magnetic Suspension and Voice Coil Motors for Observation Satellites.” *Electronics* 9, no. 12 (2020). https://www.mdpi.com/2079-9292/9/12/1997.

Subudhi, Chinari Subhechha, and D. Ezhilarasi. “Modeling and Trajectory Tracking with Cascaded PD Controller for Quadrotor.” International Conference on Robotics and Smart Manufacturing (RoSMa2018), *Procedia Computer Science* 133 (2018): 952–959. https://www.sciencedirect.com/science/article/pii/S1877050918310275.

Sun, Ning, Yu Fu, Tong Yang, Jianyi Zhang, Yongchun Fang, and Xin Xin. “Nonlinear Motion Control of Complicated Dual Rotary Crane Systems Without Velocity Feedback: Design, Analysis, and Hardware Experiments.” *IEEE Transactions on Automation Science and Engineering* 17, no. 2 (2020): 1017–1029.

Tytler, Charles. *QuadcopterSim.* https://github.com/charlestytler/QuadcopterSim, 2018.

Vinyals, Oriol, Igor Babuschkin, Wojciech M. Czarnecki, Michaël Mathieu, and et. al. “Grandmaster level in StarCraft II using multi-agent reinforcement learning.” *Nature* 575 (2019): 350–354.

Weng, Lilian. “Policy Gradient Algorithms.” *lilianweng.github.io/lil-log*, 2018. https://lilianweng.github.io/lil-log/2018/04/08/policy-gradient-algorithms.html.

Zeng, Yingfu, Chad Rose, Paul Brauner, Walid Taha, Jawad Masood, Roland Philippsen, Marcia O’Malley, and Robert Cartwright. “Modeling Basic Aspects of Cyber-Physical Systems, Part II.” January 2013.