5-2023

Using Actor-Critic Reinforcement Learning for Control of a Quadrotor Dynamics

Edgar Adrian Torres
The University of Texas Rio Grande Valley

USING ACTOR-CRITIC REINFORCEMENT LEARNING FOR CONTROL OF A QUADROTOR DYNAMICS

A Thesis

by

EDGAR ADRIAN TORRES

Submitted in partial fulfillment of the

Requirements for the degree of

MASTER OF SCIENCE IN ENGINEERING

Major Subject: Mechanical Engineering

The University of Texas Rio Grande Valley

May 2023

USING ACTOR-CRITIC REINFORCEMENT LEARNING

FOR CONTROL OF A QUADROTOR

DYNAMICS

A Thesis
by
EDGAR ADRIAN TORRES

COMMITTEE MEMBERS

Dr. Tohid Sardarmehni
Co-Chair of Committee

Dr. Constantine Tarawneh
Co-Chair of Committee

Dr. Horacio Vasquez
Committee Member

Dr. Lei Xu
Committee Member

Dr. Qi Lu
Committee Member

May 2023

ABSTRACT

Torres, Edgar A., Using Actor-Critic Reinforcement Learning For Control Of A Quadrotor Dynamics. Master of Science in Engineering (MSE), May, 2022, 59 pp., 6 tables, 46 figures, references, 21 titles.

This paper presents a quadrotor controller using reinforcement learning to generate near-optimal control signals. Two actor-critic algorithms are trained to control quadrotor dynamics. The dynamics are further simplified using small angle approximation. The actor-critic algorithm's control policy is derived from Bellman's equation providing a sufficient condition to optimality. Additionally, a smoother converter is implemented into the trajectory providing more reliable results. This paper provides derivations to the quadrotor's dynamics and explains the control using the actor-critic algorithm. The results and simulations are compared to solutions from a commercial, optimal control solver, called DIDO.

DEDICATION

I dedicate this thesis to those who supported me in my undergraduate and graduate studies. To my grandparents, who taught me how to read, count, and everything else. To my aunt, for helping me focus on my studies. To my friends and coworkers Rodrigo and Alberto, that have listened to my rambles even when they would prefer not to. And to Jancarlo for being with me in a new stage of my life.

ACKNOWLEDGMENTS

This work was supported by the U.S. Department of Homeland Security under award number 21STSLA00009-01-00.

I would also like to acknowledge partial funding provided by the CREST Center for Multidisciplinary Research Excellence in Cyber-Physical Infrastructure Systems (MECIS) under NSF Award No. 2112650.

I would like to express my sincere gratitude to my research advisor, Dr. Tohid Sardarmehni, for his support and guidance throughout my academic journey. His mentorship has been instrumental in shaping my research, and I am grateful for the opportunities he has provided me. Additionally, I extend my thanks to Dr. Constantine Tarawneh and Dr. Emmett Tomai for their contributions to my academic pursuits at the University of Texas Rio Grande Valley. Their dedication to academic excellence and support for their students has been invaluable. I appreciate the efforts of everyone who has supported me in this journey and made this achievement possible.

TABLE OF CONTENTS

Page

LIST OF TABLES

Page

LIST OF FIGURES

Page

CHAPTER I

INTRODUCTION AND BACKGROUND

1.1 Optimal Control Engineering

Optimal control is a branch of control engineering focused on solving for a system's optimal control signals. Optimal control problems focus on finding the control signals to minimize a cost function. This paper uses an actor-critic algorithm to present an optimal control solution to the quadrotor's model. This introduction is split into the following sections: first, explaining the early history of optimal control with the calculus of variation method and then discuss Markov Decision Process (MDP) and its relation to optimal control and then explain Dynamic Programming (DP), a milestone algorithm proposed by Richard. Then discussing Adaptive Dynamic Programming (ADP), a method that uses Reinforcement Learning (RL) to overcome the Curse of Dimensionality (CoD). Then discussing the Hamilton-Jacobi-Bellman (HJB) equation, a Partial Differential Equation (PDE) equation that provides the necessary and sufficient condition for an optimal solution. Then introduce the Actor-Critic algorithm, an RL algorithm that uses Bellman's equation, a discrete form of the HJB equation used in this paper to solve for a near-optimal solution. Then an overview of current and upcoming quadrotor applications in the current industry. Finally, listing the contributions presented in this paper.

1.2 Other Methods

An early example of optimal control is the foundation of the calculus of variations. It stems from Newton's minimal resistance problem and has had many contributions from

mathematicians such as Joseph Lagrange, Leonhard Euler, and Carl Jacobi. A method known as the Euler-Lagrange equation can solve an optimization problem by solving a function that minimizes or maximizes the problem. This method is developed to solve mechanical systems with some degree of freedom. The method, however, may get mathematically complicated in the presence of nonlinearities in the dynamics of the problem.

Another method for solving optimal controls of problems is Markov Decision Process. Unlike calculus of variation, MDP is a discrete and stochastic method for solving chains of probabilistic decisions. It was developed in the 1950s with manufacturing, control, and economic applications. As stated, the models are probabilistic models defined by an agent having different states and choosing between different actions. A reward function quantizes a value from the different options that can be chosen. An MDP aims to create a policy function (π) that finds the best action to update the current state while maximizing the reward. A method to solve these types of problems is dynamic programming. Dynamic programming was developed by Richard Bellman using his principle of optimality. In Bellman. (1957), the principle of optimality defines an optimal policy as having the property that from any initial state, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision. Therefore, an optimal solution can be found regardless of previous decisions. Additionally, an optimal policy can be broken down into optimal subproblems. DP uses this property to solve discrete optimization problems by dividing a significant problem into smaller, simpler ones. The recursive method splits the problem into each action and finds the optimal path.

Furthermore, by working this method backward in time, computational power is saved by creating a look-up table to prevent redundant calculations. This technique is known as memorization, where the computational cost of the value function is reduced. Despite this, DP is

subject to the Curse of Dimensionality. The increase in states and complexity of the problem increases the number of subproblems needed to solve exponentially and leads to some problems being unfeasible to solve using DP. Next, we will discuss techniques to solve optimality problems while avoiding the CoD.

1.3 Reinforcement Learning

Reinforcement learning is an area of machine learning where an agent is taught how to take optimal actions in an environment. Similar to an MDP, the agent has different states, and the machines find the best action to form an optimal path. Unlike other forms of machine learning, training occurs by feeding the machine a random policy and random states and iteratively improving the policy until it merges with an optimal policy. Since the process is an approximation and the policy function is iterated to converge to the optimal solution, the output actions are now considered a "near-optimal" solution. One advantage of this trade-off is the computational cost reduction. Using neural networks to solve for the controls is much faster, and when combined with DP, it can solve solutions in problems considered unfeasible before. The application of RL with DP is known as adaptive dynamic programming. ADP is a powerful algorithm for solving discrete problems resembling MDP problems. ADP can solve complex and nonlinear problems with smaller computational costs and a reasonably accurate solution. Several algorithms use ADP, such as actor-critic, Single Network Adaptive Critic (SNAC), and Deep Q-networks. As mentioned in Frank L. Lewis and Draguna Vrabie (2009), ADP algorithms have four basic methods: heuristic, dual heuristic, action-dependent, and Q learning. The heuristic dynamic programming trains two networks as actor and critic to approximate the optimal policy and optimal value function. Meanwhile, the dual heuristic approximates the costate.

Kirk, D. E. (2004) describes the Hamilton-Jacobi-Bellman equation as a nonlinear partial differential equation. The solution of this equation is the optimal solution and the partial derivative of the value function, the costate. The HJB provides the necessary and sufficient condition for optimality of the model. Due to the complexity of the HJB, the PDE cannot be solved with analytical or numerical methods when applied to high-complexity models. However, a discrete version known as Bellman's equation can provide a solution through iterations over a discrete problem. This paper uses this equation for policy iteration in RL, proving that the policy converges to the optimal policy.

1.4 Actor-Critic

As mentioned, this paper uses ADP to solve for the optimal control of a system. Actor-critic comprises two neural networks: a neural network (actor) that learns the optimal policy and a second neural network (critic) that approximates the minimum cost-to-go. The critic utilizes a cost function to measure the value function with respect to the actor. The actor-critic is considered a heuristic method, meaning the algorithm requires a system model, and uses an approximator of the value function. Other methods exist, such as deep Q-learning, which requires no knowledge of the dynamics for the machine to learn. The actor-critic is a stochastic algorithm, meaning all weights begin as randomized data and are fed random sample data, which causes the weights to converge to the optimal weights. There are different methods of training an actor-critic algorithm.

An example is finite-horizon and infinite-horizon techniques of training. Training occurs during a set final time in the finite horizon and usually learns backward in time. The trained weights are a function of time, meaning the weights can only be used for the trained time. In infinite horizon, training occurs by iterations only. The training can be forward-in-time and relies

on updating the same weights continuously until they converge. In infinite-horizon, the weights are not a function of time, allowing them to be used in more general cases in simulations. An actor-critic can be made as an offline or online method of training. In offline, all the weights are trained beforehand and are then used for simulation. In online training, the training simulations co-occur, meaning every simulation may further develop the weights and keep training the system.

1.5 Contributions

This paper will apply the quadrotor's dynamics to solve for the rotational speed of the rotors. This method is used as the dynamics contain twelve states and nonlinearities, making it difficult for previous methods to solve. The ADP algorithm used is known as actor-critic. This algorithm combines policy iteration and value, which trains two networks as actor and critic. In this paper, the dynamics are split into two sections known as the position and attitude of the quadrotor. This simplification helps the actor-critic neural networks learn more efficiently. The actor-critic method is chosen as it is flexible enough to allow for nonlinearities. In this paper, the actor-critic is an offline, infinite-horizon, forward-in-time algorithm, and it aims to solve for the necessary control signals to move the quadrotor across a trajectory. Its applications will now be discussed.

1.6 Applications

The quadrotor has a compact design and a versatile number of real-world capabilities and is rising in popularity among many industries, as described by UAV Commercial (2016). Quadrotor's costs have been decreasing, making it an attractive choice in implementation and usage. These applications include but are not limited to surveillance, patrolling in Burggräf, P., Pérez Martínez, A. R., Roth, H., & Wagner, J. (2019), ground surveying in Schmid, K.,

Hirschmüller, H., Dömel, A., Grixa, I., Suppa, M., & Hirzinger, G. (2012), and agricultural crop maintenance in Van der Merwe, D., Burchfield, D. R., Witt, T. D., Price, K. P., & Sharda, A. (2020). Quadrotors can be equipped with various sensors, including cameras, infrared, LiDARs, and microphones, combined with onboard computational processing. Quadrotors have potential in more applications such as traffic flow analysis in De Bruin, A., & Booysen, T. (2015), structural maintenance in Flammini, F., Pragliola, C., & Smarra, G. (2016), and indoor applications in Burggräf, P., Pérez Martínez, A. R., Roth, H., & Wagner, J. (2019).

1.7 Related Work

Similar work has been done by Emmanuel Stingu and Frank L. Lewis (2011). In their paper, the dynamics are split into three separate actors and a single global critic to learn the dynamics' translation, attitude, and rotors. The system has four control inputs and seventeen states split into position, attitude, velocities, angular velocities, and rotor speeds. The results show it can track a quadrotor. However, weight imbalances and wind may cause significant disturbances. Zahra Marvi and Bahare Kiumarsi (2021) paper designed a reinforcement learning controller utilizing barrier functions. The paper proves a safe set boundary exists, making control barrier functions a viable value function in actor-critic algorithms. In work by Abhijit Das, Frank Lewis, and Makesh Subbarao (2009), backstepping was used to control a quadcopter.

CHAPTER II

QUADROTOR'S DYNAMICS

2.1 Model Definition

The dynamics and derivations presented in this chapter are from the paper by Sabatino (2015). The following two coordinates space are introduced to establish the dynamics of a quadrotor model, as shown in Figure 1. The first coordinate space is a global coordinate space known as the earth frame defined by the cardinal directions north, south, east, west, up, and down directions. The quadrotor's position is described by $[x\ y\ z]$ space, and the roll, pitch, and yaw are described as $[\phi\ \theta\ \psi]$ as shown in Figures 2 and 3. Its velocities and angular velocities are then described as $\left[\dot{x}\ \dot{y}\ \dot{z}\ \dot{\phi}\ \dot{\theta}\ \dot{\psi}\right]$. The quadrotor uses four motors with variable angular speeds to control its movement along the global coordinate system. These angular speeds are described as $[\Omega_1\ \Omega_2\ \Omega_3\ \Omega_4]$. To define the model, first, its states must be considered. In the global coordinate space, the position and velocity are defined as $[x\ y\ \ z\ \dot{x}\ \dot{y}\ \dot{z}]$ and the angular position and angular velocity are defined as $\left[\phi\ \theta\ \psi\ \dot{\phi}\ \dot{\theta}\ \dot{\psi}\right]$. These states are beneficial when defining a wanted trajectory based on the global coordinate space. However, the body coordinates space models the forces acting on the quadrotor. In this coordinate system, there is no position and angle as the coordinate origin is attached to the quadrotor, meaning the body coordinate space moves and rotates alongside the quadrotor. Instead, local velocities $[u\ v\ w]$ and local angular velocities $[p\ q\ r]$ of the quadrotor are measured. These states are helpful as the forces acting on the quadrotor, the thrust, and the torques $\left[f_t\ \tau_x\ \tau_y\ \tau_z\right]$, directly affect this coordinate space. A

method for transforming the body coordinate space into the global coordinate space is created based on rotational transformation matrices to create a complete set of dynamics.

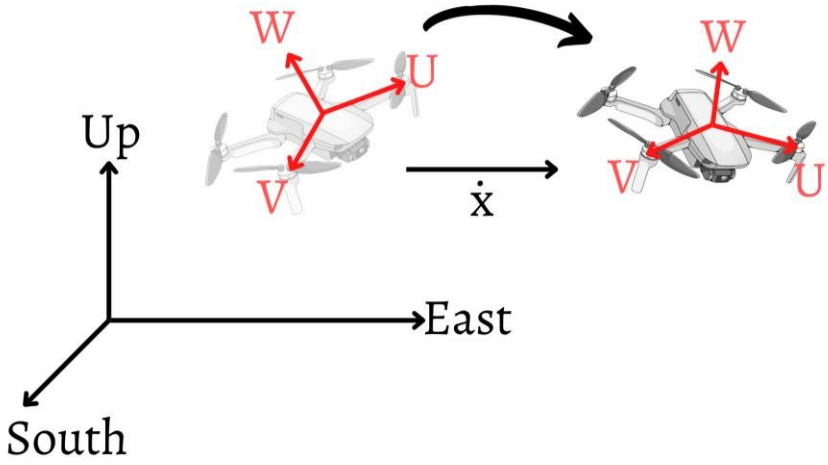

Figure 1: Comparison between the global (earth) and local (body) coordinate space.

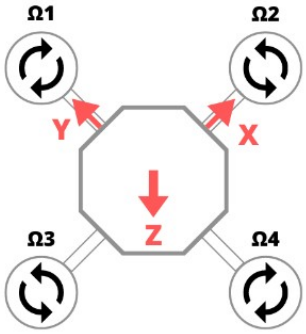

Figure 2: Graphic showing the angular speed of the rotors Ω and XYZ space.

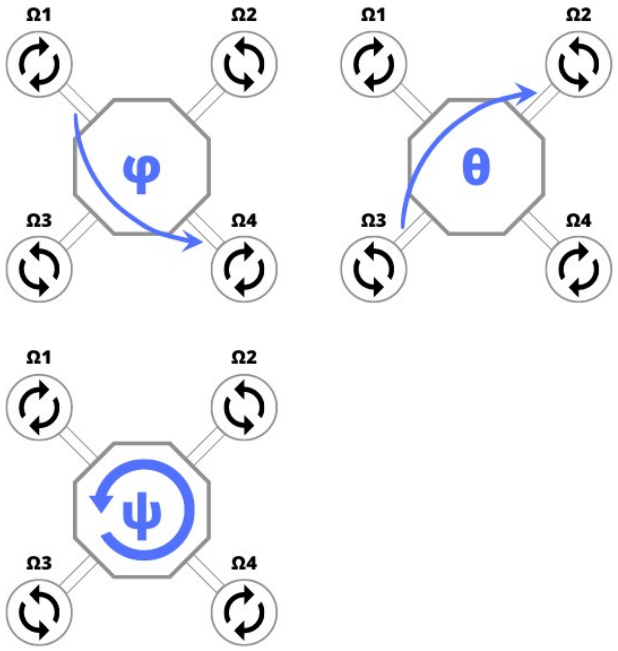

Figure 3: Graphic showing the quadrotor's roll as ϕ, pitch as θ, and yaw as ψ.

2.2 Model Formulation

For this paper, the position, angles, velocities, and angular velocities are chosen as the system states $x = [x\ y\ z\ \dot{x}\ \dot{y}\ \dot{z}\ \phi\ \theta\ \psi\ \dot{\phi}\ \dot{\theta}\ \dot{\psi}]$. The thrust and torques are chosen as the controls $u = [f_t\ \tau_x\ \tau_y\ \tau_z]$ as they represent forces actuating on the body. The reason for choosing these forces/torques as the control is their convenience in describing acceleration in the body-space coordinate system. To further derive the model, the following two derivations are needed. First, a set of equations for converting the local forces and velocities of the quadrotor into the global coordinate space. Second, a set of equations for converting the rotor speeds $[\Omega_1\ \Omega_2\ \Omega_3\ \Omega_4]$ to the input forces $[f_t\ \tau_x\ \tau_y\ \tau_z]$. These two formulations allow for the simulation of the quadrotor dynamics under different controls and conversion into real-world testing and implementation. The following section derives the equations that relate the body and global coordinate space.

2.3 Model Derivation

The derivation begins with Newton's second law. This derivation aims to create a set of equations that transform the body-coordinate system into the global-coordinate system. Equations (1) and (2) establish this relationship through the R and T transformation matrix as defined in (3) and (4). In these equations, $v = [\dot{x}\ \dot{y}\ \dot{z}]$ represents the velocities in the global coordinates, $v_B = [u\ v\ w]$ represents the velocities in the body coordinates, $\omega = [\dot{\phi}\ \dot{\theta}\ \dot{\psi}]$ represent the angular velocities in the global coordinates, and $\omega_B = [p\ q\ r]$ represent the angular velocities in the body coordinate system. Equation (5) completes the derivation of the kinematics of the quadrotor. The kinematics define the movement and speeds of the quadrotor as a function of the body velocities and angular velocities. To complete the dynamics, the equations simulating $[u\ v\ w\ p\ q\ r]$ are derived using the kinetics of the system.

$$v = R \cdot v_B \tag{1}$$

$$\omega = T \cdot \omega_B \tag{2}$$

$$R = \begin{bmatrix} \mathrm{c}(\theta)\mathrm{c}(\psi) & \mathrm{s}(\varphi)\mathrm{s}(\theta)\mathrm{c}(\psi) - \mathrm{c}(\varphi)\mathrm{s}(\psi) & \mathrm{c}(\varphi)\mathrm{s}(\theta)\mathrm{c}(\psi) + \mathrm{s}(\varphi)\mathrm{s}(\psi) \\ \mathrm{c}(\theta)\mathrm{s}(\psi) & \mathrm{s}(\varphi)\mathrm{s}(\theta)\mathrm{s}(\psi) + \mathrm{c}(\varphi)\mathrm{c}(\psi) & \mathrm{c}(\varphi)\mathrm{s}(\theta)\mathrm{s}(\psi) - \mathrm{s}(\varphi)\mathrm{c}(\psi) \\ -\mathrm{s}(\theta) & \mathrm{s}(\varphi)\mathrm{c}(\theta) & \mathrm{c}(\varphi)\mathrm{c}(\theta) \end{bmatrix} \tag{3}$$

$$T = \begin{bmatrix} 1 & \mathrm{s}(\varphi)\mathrm{t}(\theta) & \mathrm{c}(\varphi)\mathrm{t}(\theta) \\ 0 & \mathrm{c}(\varphi) & -\mathrm{s}(\varphi) \\ 0 & \dfrac{\mathrm{s}(\varphi)}{\mathrm{c}(\theta)} & \dfrac{\mathrm{c}(\varphi)}{\mathrm{c}(\theta)} \end{bmatrix} \tag{4}$$

$$
\begin{aligned}
\dot{x} &= u[c(\psi)c(\theta)] - v[c(\phi)s(\psi) - c(\psi)s(\phi)s(\theta)] + w[s(\phi)s(\psi) + c(\phi)c(\psi)s(\theta)] \\
\dot{y} &= u[c(\theta)c(\psi)] + v[c(\phi)c(\psi) + s(\phi)s(\psi)s(\theta)] - w[c(\psi)s(\phi) - c(\phi)s(\psi)s(\theta)] \\
\dot{z} &= -u[s(\theta)] + v[c(\theta)s(\phi)] + w[c(\phi)c(\theta)] \\
\dot{\phi} &= p + q[s(\varphi)t(\theta)] + r[c(\varphi)t(\theta)] \\
\dot{\theta} &= q[c(\varphi)] - r[s(\varphi)] \\
\dot{\psi} &= q\left[\frac{s(\varphi)}{c(\theta)}\right] + r\left[\frac{c(\varphi)}{c(\theta)}\right]
\end{aligned}
\tag{5}
$$

The kinetics are derived from Newton's law. First, the total forces, $f_B = [f_x\ f_y\ f_z]$ are defined in equation (6). This equation represents that the total forces in each axis on the local body are equal to the mass times the total acceleration in the axis. Repeating a similar process to Euler's equation results in equation (7). In this case, the total torques $m_B = [m_x\ m_y\ m_z]$, are equal to the summation of rotational acceleration and angular momentum, where I_{xyz} is the diagonal inertia matrix, as shown in equation (8). Evaluating the forces and torques results in equation (9). This equation describes the kinetics of the quadrotor as a function of generic forces, local velocities, and angular velocities.

$$f_B = m(\omega_B \times v_B + \dot{v}_B) \tag{6}$$

$$m_B = I_{xyz} \cdot \dot{\omega_B} + \omega_B \times (I_{xyz} \cdot \omega_B) \tag{7}$$

$$I_{xyz} = \begin{bmatrix} I_x & 0 & 0 \\ 0 & I_y & 0 \\ 0 & 0 & I_z \end{bmatrix} \tag{8}$$

$$\begin{cases} f_x = m(\dot{u} + qw - rv) \\ f_y = m(\dot{v} - pw + ru) \\ f_z = m(\dot{w} + pv - qu) \\ m_x = \dot{p}I_x - qrI_y + qrI_z \\ m_y = \dot{q}I_y + prI_x - prI_z \\ m_z = \dot{r}I_z - pqI_x + pqI_y \end{cases} \tag{9}$$

The external forces of the body must be defined to complete the derivation of the kinetics of the body. In this paper, the linear external forces considered are the forces of gravity, thrust force, and external wind force. In equation (10), $\hat{e}_z$ is a unit vector pointing to the inertial z-axis, $\hat{e}_3$ is the unit vector pointing in the body coordinate z-axis, $f_w = \begin{bmatrix} f_{wx} & f_{wy} & f_{wz} \end{bmatrix}$ are the wind forces, and f_t is the control input thrust force. The moments acting on the quadrotor in equation (11) are $\tau_\omega = \begin{bmatrix} \tau_{wx} & \tau_{wy} & \tau_{wz} \end{bmatrix}$ the torques caused by external wind, g_a, the gyroscopic movement, and τ_B are the control input torques. Each rotor's inertia is considered small enough to be assumed as zero in this derivation. Therefore, the effect of gyroscopic movement is ignored. Evaluating these equations results in equation (12).

$$f_B = mgR^T \cdot \hat{e}_z + f_w - f_t\hat{e}_3 \tag{10}$$

$$m_B = \tau_\omega - g_a + \tau_B \tag{11}$$

$$\begin{cases} -mg[s(\theta)] + f_{wx} = m(\dot{u} + qw - rv) \\ mg[c(\theta)s(\phi)] + f_{wy} = m(\dot{v} - pw + ru) \\ mg[c(\theta)c(\phi)] + f_{wz} - f_t = m(\dot{w} + pv - qu) \\ \tau_x + \tau_{wx} = \dot{p}I_x - qrI_y + qrI_z \\ \tau_y + \tau_{wy} = \dot{q}I_x + prI_x - prI_z \\ \tau_z + \tau_{wz} = \dot{r}Iz - pqI_x + pqI_y \end{cases} \tag{12}$$

Up to this point, the kinetics and kinematics have been fully defined for the system. The derivation will now define the inputs as the force thrust and control torques $u = \begin{bmatrix} f_t & \tau_x & \tau_y & \tau_z \end{bmatrix}$. Recall that the original inputs defined for the system are the angular speed of the motors Ω_i. Using equation (13), the angular speeds and control forces can be algebraically solved. For the

remainder of this paper, the forces are treated as the control signals of the system. However, the speeds of the rotors must be algebraically solved if any quadrotor flight simulation or real-world testing is performed, as shown in equation (14). The next step in deriving the equations is formatting them into a state space format.

$$\begin{cases} f_t = b(\Omega_1^2 + \Omega_2^2 + \Omega_3^2 + \Omega_4^2) \\ \tau_x = bl(\Omega_3^2 - \Omega_1^2) \\ \tau_y = bl(\Omega_4^2 - \Omega_2^2) \\ \tau_z = d(-\Omega_1^2 + \Omega_2^2 - \Omega_3^2 + \Omega_4^2) \end{cases} \tag{13}$$

$$\begin{cases} O_1 = \Omega_1^2 = -\dfrac{2\tau_x d - f_t dl + \tau_z bl}{4bdl} \\ O_2 = \Omega_2^2 = -\dfrac{2\tau_y d - f_t dl - \tau_z bl}{4bdl} \\ O_3 = \Omega_3^2 = \dfrac{2\tau_x d + f_t dl - \tau_z bl}{4bdl} \\ O_4 = \Omega_4^2 = \dfrac{2\tau_y d + f_t dl + \tau_z bl}{4bdl} \end{cases} \tag{14}$$

2.4 Model's Dynamic Equations

To complete the derivation of the dynamic, equations (5) and (12) are reorganized into equations (15) and (16). Equation (15) describes the position of the quadrotor where the states are a function of the attitudes and thrust force. Equation (16) describes the attitude where the states are a function of the torques. Additionally, both equations take into account the external wind forces and torques. In this paper, the wind is assumed to be zero.

$$
\begin{aligned}
\dot{x} &= u[c(\psi)c(\theta)] - v[c(\varphi)s(\psi) - c(\psi)s(\varphi)s(\theta)] + w[s(\varphi)s(\psi) + c(\varphi)c(\psi)s(\theta)] \\
\dot{y} &= u[c(\theta)s(\psi)] + v[c(\varphi)c(\psi) + s(\varphi)s(\psi)s(\theta)] - w[c(\psi)s(\varphi) - c(\varphi)s(\psi)s(\theta)] \\
\dot{z} &= -u[s(\theta)] + v[c(\theta)s(\varphi)] + w[c(\varphi)c(\theta)] \\
\dot{u} &= rv - qw - g[s(\theta)] + \frac{f_{wx}}{m} \\
\dot{v} &= pw - ru + g[s(\phi)c(\theta)] + \frac{f_{wy}}{m} \\
\dot{w} &= qu - pv + g[c(\theta)c(\varphi)] - \frac{f_{wz} - f_t}{m}
\end{aligned}
\tag{15}
$$

$$
\begin{aligned}
\dot{\psi} &= q\left[\frac{s(\varphi)}{c(\theta)}\right] + r\left[\frac{c(\varphi)}{c(\theta)}\right] \\
\dot{\theta} &= q[c(\phi)] - r[s(\phi)] \\
\dot{\varphi} &= p + q[s(\varphi)t(\theta)] + r[c(\varphi)t(\theta)] \\
\dot{p} &= rq\frac{I_y - I_z}{I_x} + \frac{\tau_x + \tau_{\omega x}}{I_x} \\
\dot{q} &= pr\frac{I_z - I_x}{I_y} + \frac{\tau_y + \tau_{\omega_y}}{I_y} \\
\dot{r} &= pq\frac{I_x - I_y}{I_z} + \frac{\tau_z + \tau_{\omega z}}{I_z}
\end{aligned}
\tag{16}
$$

The dynamics are split into two sections, position and attitude. As a whole, the quadrotor is a twelve-state and four-control system. The system is underactuated and, combined with the complex nonlinearities, increases the complexity of the problem. By splitting the equations into these two sections, two separate neural networks can be trained to solve each individually, converting the problem into two simpler subproblems.

Additionally, a small angle approximation is implemented to simplify the dynamics further. With the assumption that the angles are small enough, the system is further simplified, as shown in equations (17) and (18). The accelerations can also be estimated, as shown in equation (19). These accelerations provide an algebraic way to convert accelerations from a trajectory into the necessary angles and thrust force. In this paper, ψ is assumed to be small enough to be

approximated as zero. Therefore, the algebraic solution is shown in equations (20). This system is used as an intermediate step between position and attitude.

$$\begin{aligned} \dot{x} &= u \\ \dot{y} &= v \\ \dot{z} &= w \\ \dot{u} &= \ddot{x} \\ \dot{v} &= \ddot{y} \\ \dot{w} &= \ddot{z} + g \end{aligned} \tag{17}$$

$$\begin{aligned} \dot{\phi} &= p + q[s(\varphi)t(\theta)] + r[c(\varphi)t(\theta)] \\ \dot{\theta} &= q[c(\phi)] - r[s(\phi)] \\ \dot{\psi} &= q\left[\frac{s(\varphi)}{c(\theta)}\right] + r\left[\frac{c(\varphi)}{c(\theta)}\right] \\ \dot{p} &= rq\frac{I_y - I_z}{I_x} + \frac{\tau_x}{I_x} \\ \dot{q} &= pr\frac{I_z - I_x}{I_y} + \frac{\tau_y}{I_y} \\ \dot{r} &= pq\frac{I_x - I_y}{I_z} + \frac{\tau_z}{I_z} \end{aligned} \tag{18}$$

$$\begin{aligned} \ddot{x} &= -\frac{f_t}{m}[s(\phi)s(\psi) + c(\phi)c(\psi)s(\theta)] \\ \ddot{y} &= -\frac{f_t}{m}[c(\phi)s(\psi)s(\theta) - c(\psi)s(\phi)] \\ \ddot{z} &= g - \frac{f_t}{m}[c(\phi)c(\theta)] \end{aligned} \tag{19}$$

$$\begin{aligned} f_t &= m * \left(g^2 - 2g\ddot{z} + \ddot{x}^2 + \ddot{y}^2 + \ddot{z}^2\right) \\ \phi &= \arcsin\left(\frac{m\ddot{y}}{f_t}\right) \\ \theta &= \arcsin\left(\frac{-m\ddot{x}}{f_t c(\phi)}\right) \\ \psi &= 0 \end{aligned} \tag{20}$$

To transform the system of equations into a state space model, the dynamics are defined as a system of $\dot{x} = f(x) + g(x)u$, where $f(x)$ and $g(x)$ represents the state and control. A discretized version is defined as $x_{k+1} = \bar{f}(x_k) + \bar{g}(x_k)u_k$ where k represents the current discrete time, $\bar{f}(x_k) = x_k + \Delta t f(x_k)$, and $\bar{g}(x_k) = \Delta t g(x_k)$. Tables 1 and 2 summarize the states, controls, and dynamics for the position and attitude model of the quadrotor.

Table 1: State-space definition for the position

Variables:	$x = [x\ y\ z\ u\ v\ w], \qquad u = [\ddot{x}\ \ddot{y}\ \ddot{z}]$
State Dynamics:	$f(x) = \begin{matrix} u \\ v \\ w \\ 0 \\ 0 \\ g \end{matrix}$
Control Dynamics:	$g(x) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$

Table 2: State-space definition for the attitude

Variables:	$x = [\phi\ \theta\ \psi\ p\ q\ r], \qquad u = [\tau_x\ \tau_y\ \tau_z]$
State Dynamics:	$f(x) = \begin{matrix} p + q[s(\varphi)t(\theta)] + r[c(\varphi)t(\theta)] \\ q[c(\phi)] - r[s(\phi)] \\ q\left[\frac{s(\varphi)}{c(\theta)}\right] + r\left[\frac{c(\varphi)}{c(\theta)}\right] \\ rq\frac{I_y - I_z}{I_x} \\ pr\frac{I_z - I_x}{I_y} \\ pq\frac{I_x - I_y}{I_z} \end{matrix}$
Control Dynamics:	$g(x) = \begin{bmatrix} \frac{1}{I_x} & 0 & 0 \\ 0 & \frac{1}{I_y} & 0 \\ 0 & 0 & \frac{1}{I_z} \end{bmatrix}$

CHAPTER III

ACTOR-CRITIC ALGORITHM

3.1 Background

An actor-critic algorithm is used to learn and solve for the quadrotor's control signals for the near-optimal solution. The actor-critic algorithm is an ADP solution that approximates the optimal control solution using two separate neural networks. These two networks are named the actor and critic. The actor and critic are trained using reinforcement learning methods. There is a large variety of actor-critic algorithms available, each with different traits and advantages.

There are different ways to set up the actor-critic algorithm. For example, the algorithm can have offline or online training. In offline training, all the training occurs beforehand, leaving the weights static during use. The weights update after every simulation in online training, meaning the system is always learning. Online training is useful in applications where learning new samples may be necessary, especially in systems with unstable or unpredictable states. Offline training performs its training before use, which is helpful for applications in which the system is more consistent and is not likely to face new scenarios. Additionally, offline training requires fewer computation resources as no training is involved in running the simulation.

A different style for actor-critic algorithms is in finite-horizon or infinite-horizon training. In finite-horizon, the training happens under a set training time with a predefined trajectory and simulation time. Additionally, a set of weights can be saved for each discrete time

step, and training occurs backward in time. Otherwise, training may happen as an infinite horizon in which training is iterative forward-in-time. Finite-horizon is ideal for high complexity systems in which a different set of weights are necessary for every time step.

Finally, the actor-critic algorithm can be implemented in two methods, hard-terminal or soft-terminal constraints. The terminal constraint refers to the endpoint the system reaches. In a regulation simulation, the controller's objective is to move the states and controls of the system to zero. In hard-terminal constraints, the system is trained to guarantee the final state reaches the exact desired final state, in the case of a regulator, reaches zero. In soft-terminal constraints, the system is trained to approach the desired final state without necessarily reaching the exact desired final state. In this paper, the soft-terminal constraints method is used. The remainder of this chapter explains a basic formulation for the actor and critic neural networks and the algorithm.

3.2 Actor-Critic Derivation

The derivation is presented in this chapter is by Heydari, A., & Balakrishnan, S. N. (2013). The paper presents the following equations for a discrete-time HJB equation as equations (21) and (22).

$$J_k^*(x_k) = \frac{1}{2}(\bar{Q}(x_k) + u(x_k)^T \bar{R} u(x_k) + J_{k+1}^*(x_{k+1}^*) \tag{21}$$

$$u_k^*(x_k) = -\bar{R}\bar{g}(x_k)\frac{\partial J_{k+1}^*}{\partial x_{k+1}}|_{x_{k+1}^*} \tag{22}$$

Where J_k^* is the optimal cost, u_k^* is the optimal control, $\bar{Q}(x_k) = \Delta t x_k^T Q x_k$, $Q \in \mathbb{R}^{N_s} \rightarrow \mathbb{R}$ is the penalizing state matrix, $\bar{R} = \Delta t R$, $R \in \mathbb{R}^{N_i} \rightarrow \mathbb{R}$ is the penalizing control matrix, $x_{k+1}^* = \bar{f}(x_k) + \bar{g}(x_k)u_k^*(x_k)$, $\bar{f}(x_k) = \Delta t f(x_k)$, and $\bar{g}(x_k) = \Delta t g(x_k)$. Then the actor and

critic are defined in equations (23) and (24). Where $\lambda(x_k)$ is the actor's basis functions, $\kappa(x_k)$ is the critic's basis functions, $x_{k+1} = \bar{f}(x_k) + \bar{g}(x_k)V_k^{i^T}\lambda(x_k)$, and *i* refers to an iteration counter for the actor.

$$W_k^T\kappa(x_k) = J_k(x_k) = \frac{1}{2}(\bar{Q}(x_k) + u(x_k)^T\bar{R}u(x_k)) + W_{k+1}^T\kappa(x_{k+1}) \quad (23)$$

$$V_k^{i+1^T}\lambda(x_k) = u_k^{i+1} \cong -\bar{R}^{-1}\bar{g}(x_k)^T\nabla\kappa(x_n)^T W_{k+1} \quad (24)$$

Equations (23) and (24) represent a finite horizon, backward-in-time actor-critic algorithm. These equations are now modified to be an infinite-horizon algorithm in equation (25) and (26). The weights no longer require to be time-dependent and instead update the same set of weights.

$$W^T\kappa(x_k) = J(x_k) = \frac{1}{2}(\bar{Q}(x_k) + u(x_k)^T\bar{R}u(x_k)) + W^T\kappa(x_{k+1}) \quad (25)$$

$$V^{i+1^T}\lambda(x_k) = u^{i+1} \cong -\bar{R}^{-1}\bar{g}(x_k)^T\nabla\kappa(x_n)^T W \quad (26)$$

3.3 Algorithm

To summarize the actor-critic algorithm, four figures are given to clarify how the actor-critic algorithm functions. First, Table 3 lists the main parameters and values of the algorithm. These include names for parameters such as N_s and N_i count the total number of states and inputs. Table 4 summarizes the differences between the actor and the critic. Each neural network has an independent set of weights and basis functions. For the actor, they are referred to as V and $\lambda(x)$, and for the critic, they are referred to as W and $\kappa(x)$.

Table 3: List of different properties and variables in the actor-critic algorithm

Property	Description
N	Total training iterations
N_s	System number of states
N_i	System number of inputs
N_λ	Actor's number of neurons
N_κ	Critic's number of neurons
N_p	Total number of patterns in training
Ω	Domain of training
δ	Actor's convergence threshold
R, Q, H	State penalizing matrices
$\bar{R}, \bar{Q}, \bar{H}$	Discrete penalizing matrices e.g. $\bar{Q} = \Delta t Q$
$f(x), g(x)$	State-space form dynamics
$\bar{f}(x), \bar{g}(x)$	Discrete form of dynamics e.g. $\bar{f}(x_k) = x_k + \Delta t f(x_k), \quad \bar{g}(x) = \Delta t g(x_k)$
$\lambda(x), \kappa(x)$	Actor and critic's set of basis functions

Table 4: Summary of differences between the actor and critic

	Actor	Critic
Weights:	$V: \rightarrow \mathbb{R}^{N_\lambda \times N_i}$	$W: \rightarrow \mathbb{R}^{N_\kappa}$
Basis:	$\lambda(x)$	$\kappa(x)$

The final actor-critic algorithm is now outlined below. The result is a soft-terminal infinite horizon algorithm. The algorithm is defined across eight steps. Additionally, Figure 4 outlines the algorithm as a flowchart. The flowchart highlights the difference between what is referred to as the inner loop and the outer loop. To train the actor, the updated policy is iterated

by the inner loop to ensure the actor converges to its optimal value. The outer loop runs for N number of iterations. N should be chosen to be a high enough number to train the system fully. The number depends on the complexity of the system.

Actor-Critic Training Algorithm

Step 1: Set V to a randomly selected matrix of $\mathbb{R}_\sigma \times \mathbb{R}_i$ and set $W = LSR(\kappa(x_k),\ J)$ for randomly selected $x_k \in \Omega$

Step 2: Set $k = 0$

Step 3: Set $i = 0$

Step 4: Train V^{i+1} such that

$$V^{i+1}\sigma(x_k) = u \cong -\bar{R}^{-1}\bar{g}(x_k)^T\nabla\rho\left(\bar{f}(x_k) + \bar{g}(x_k)u_k\right)^T W$$

where $u_k = V^T\sigma(x_k)$, for randomly selected $x_k \in \Omega$

Step 5: Set $i = i + 1$. Repeat until $\left|V^{i+1} - V^i\right|$ converges to some small value δ

Step 6: Update V to V^i

Step 7: Train W such that

$$W\kappa(x_k) = J = 0.5x_k^T\bar{Q}x_k + 0.5u_k^T\bar{R}u_k + W^T\kappa\left(\bar{f}(x_k) + \bar{g}(x_k)u_k\right),$$

where $u_k = V^T\lambda(x_k)$, for randomly selected $x_k \in \Omega$

Step 8: Set k = k + 1. Repeat until k = N

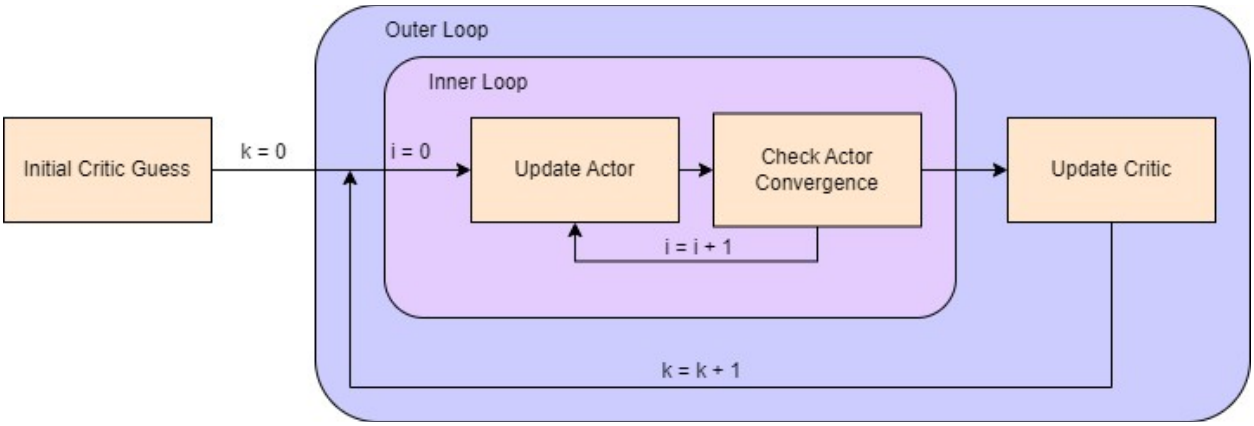

Figure 4: Generic actor-critic algorithm

CHAPTER IV

RESULTS

4.1 Actor-Critic Implementation

To implement the algorithm in chapter three, a MATLAB program is created to train two separate neural networks and run a controller. The program is meant to complete two separate objectives. First, it implements an actor-critic algorithm to train two separate neural networks. Second, a controller must be designed and uses the trained neural networks. The implemented neural networks use the actor-critic algorithm as specified in Figure 5. Since the quadrotor dynamics are split into two sections, the position, and attitude, each portion has a separate neural network created. These are the alpha neural network (position) and beta neural network (attitude). Each network is trained separately with its portion of the quadrotor dynamics. The alpha network is trained to learn the position dynamics of the quadrotor shown in Table 1, while the beta network is trained to learn the attitude dynamics in Table 2. Each network has similar structures in its MATLAB implementation. In this section, a detailed explanation of the implementation of the actor-critic algorithm is given.

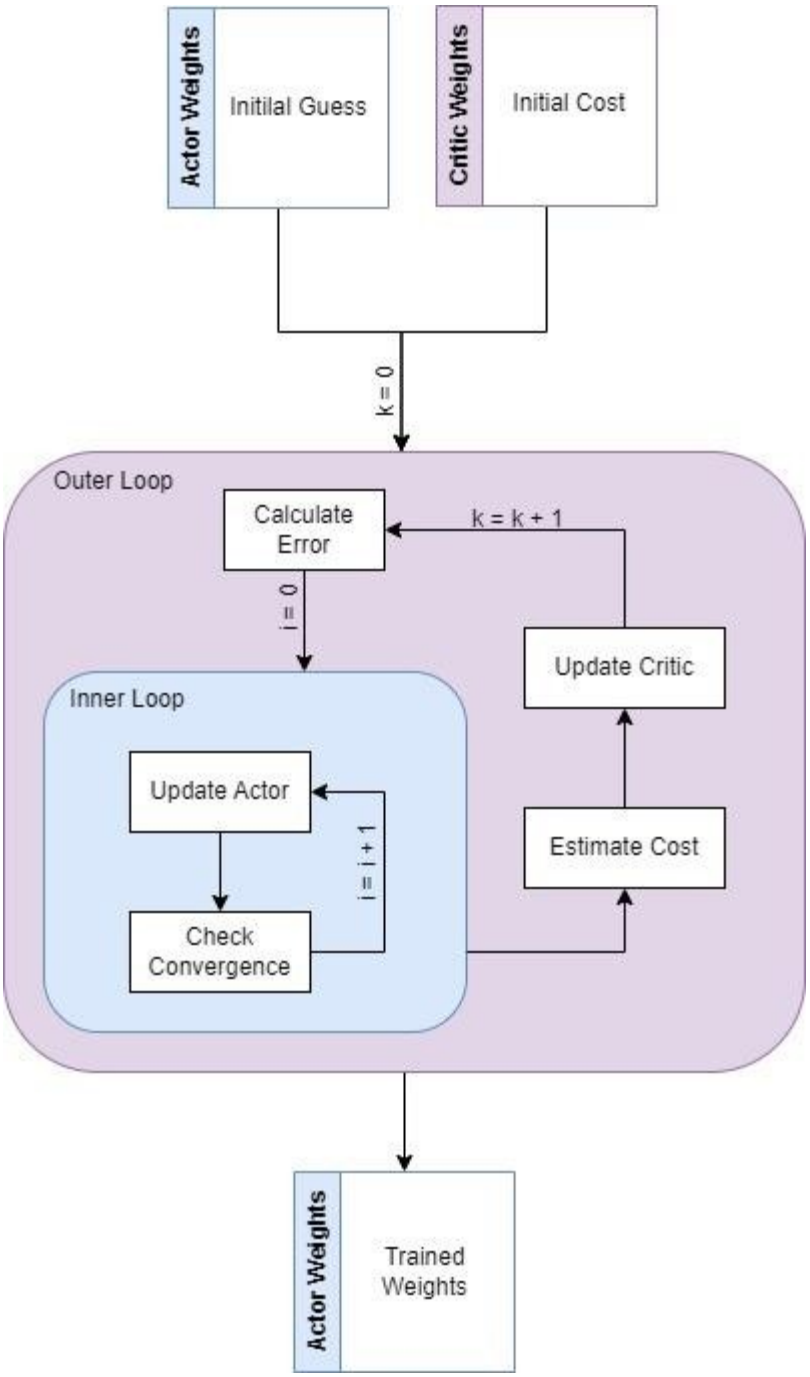

Figure 5: Actor-critic training MATLAB implementation

First, each network is set up with the following programs as specified in Table 5. Each file does a separate part of the program. The alpha and beta neural networks are designed in the same format, with the main significant difference being in the dynamic's programs. Each neural

network uses its appropriate set of dynamics. The trainer file runs the actor-critic learning algorithm. After finishing training, this program loads the networks' dynamics and outputs the resultant weights. The performer can load the weights anytime, which runs an offline control. The file needs a reference path and initial conditions and outputs the states and generated control signals for the simulation.

Table 5: Program list for the actor-critic algorithm

Program Name	Description
Dynamics	Contains the position or attitude dynamics as specified in Tables 1 and Table 2.
Trainer	Runs an actor-critic algorithm with the specified dynamics. Outputs the trained actor weights.
Weights	Trained matrix set data $N_\lambda \times N_i$
Performer	Loads the trained weights to estimate the near-optimal control signals in some given initial conditions and trajectory path.

The actor-critic algorithm requires tuned hyperparameters for each trainer program. These parameters are specified in Table 6. The No. Iterations is the number of outer loop iterations performed to train the system. This number is manually chosen and should be high enough for the critic to converge. R is the control penalizing matrix, Q is the state penalizing diagonal matrix, and H is the initial-state penalizing matrix. These three matrices affect each

control or state's priority when performing regression. Therefore, a carefully chosen balance should be searched for and experimented with. δ, represent the convergence check for the inner loop. This value should be the smallest without interfering with the actor's weight convergence. A similar note is for No. of Patterns, increasing this value helps in learning more complex dynamics with the downside of increasing computational cost. Ω is the domain of training for each state. This domain needs to be set up in the range the states may reach.

Table 6: Hyper-parameters for alpha neural network

Hyper-parameter	**Value**
No. Iterations	5000
R	$10^5 * [0.5\ 0.5\ 0.5]$
Q	$10^5[1\ 1\ 1\ 1\ 1\ 1]$
H	$10^5[1\ 1\ 1\ 1\ 1\ 1]$
δ	0.001
Max Actor Iterations	10
No. of Patterns	100
Ω	[1 1 1 1 1 1]

The MATLAB implementation script ran for both the alpha and beta networks. Below is an overview of the convergence of weights across training. It is expected to see the actor's weights converge to some final value. Figures 6 and Figure 8 show the actor's weight, The error during training is also shown in Figure 7.

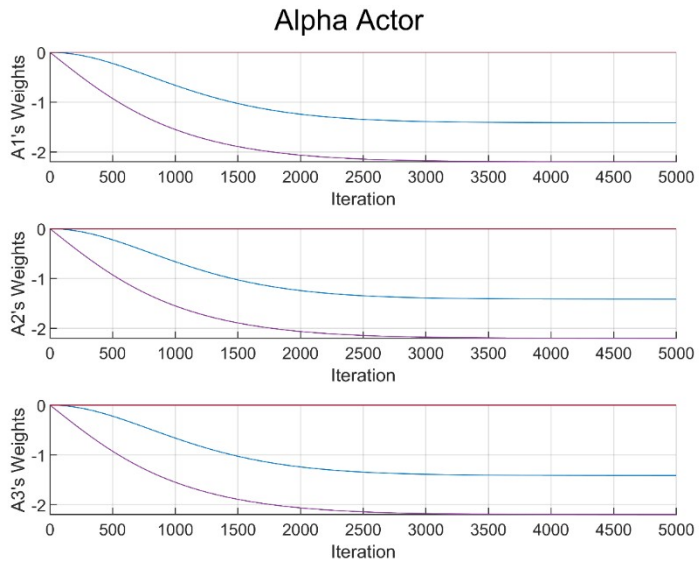

Figure 6: Alpha's NN Actor Weight Convergence

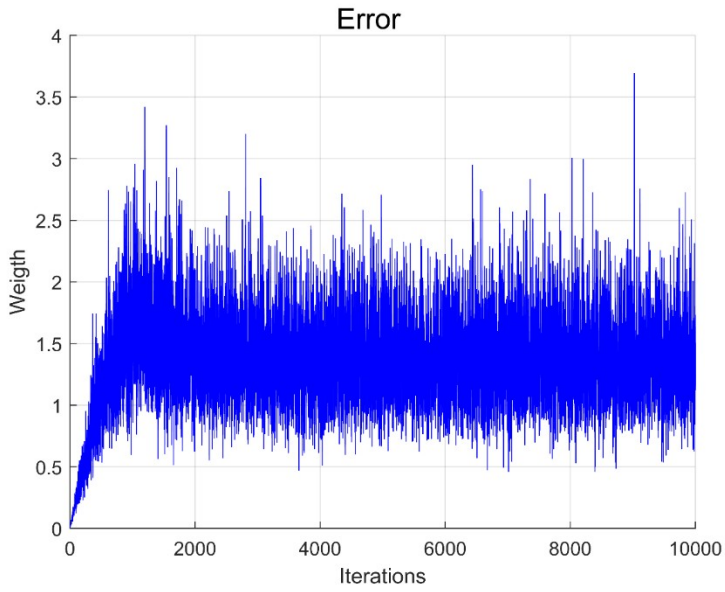

Figure 7: Training runtime error for beta NN

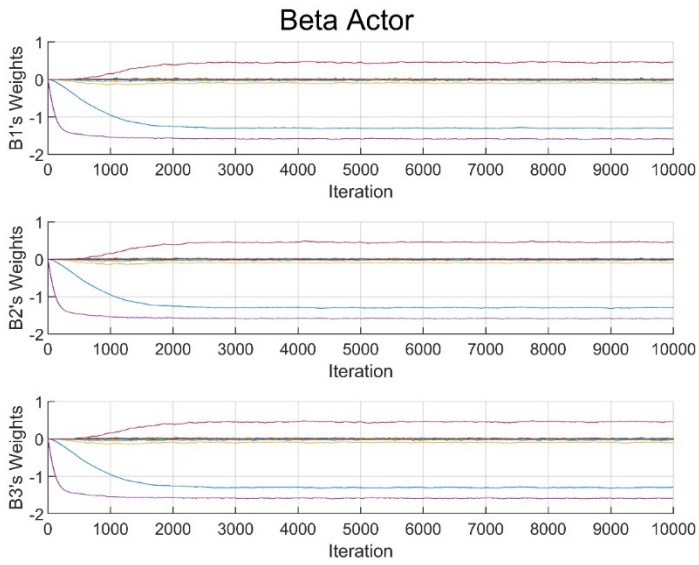

Figure 8: Beta's NN actor weight convergance

After training the alpha and beta neural network weights, the offline control is run through the performer file. The performer loads in the dynamics and trained weights and simulates the system under a given trajectory and initial conditions. Figure 9 shows a diagram of the structure of the file. Figures 10 through 19 show performer tests in the cases for regression and following a trajectory for both alpha and beta neural networks.

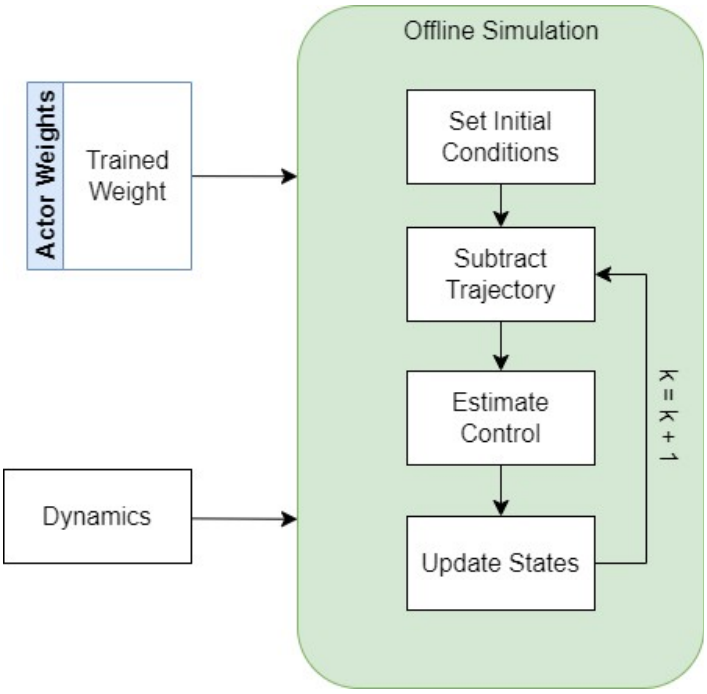

Figure 9: Actor-critic offline control MATLAB implementation

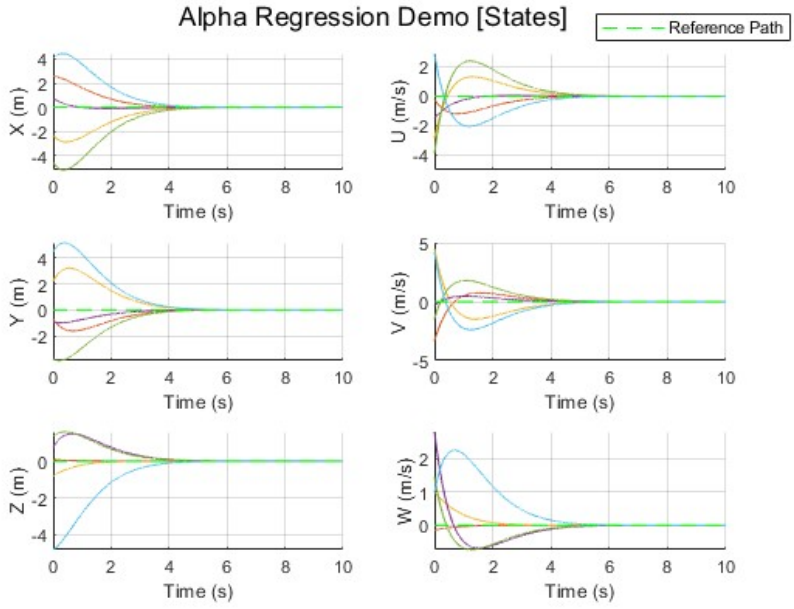

Figure 10: State performance demonstration on regression (alpha)

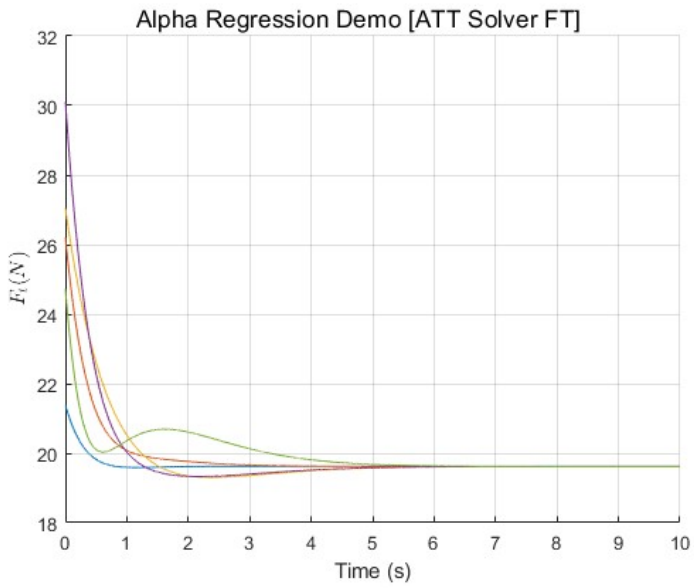

Figure 11: Solved thrust force from demonstration on regression (alpha)

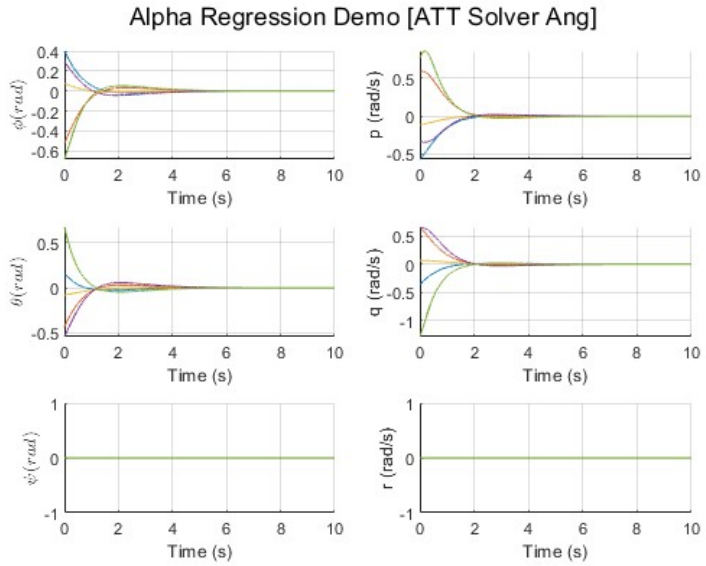

Figure 12: Solved attitude from demonstration on regression (alpha)

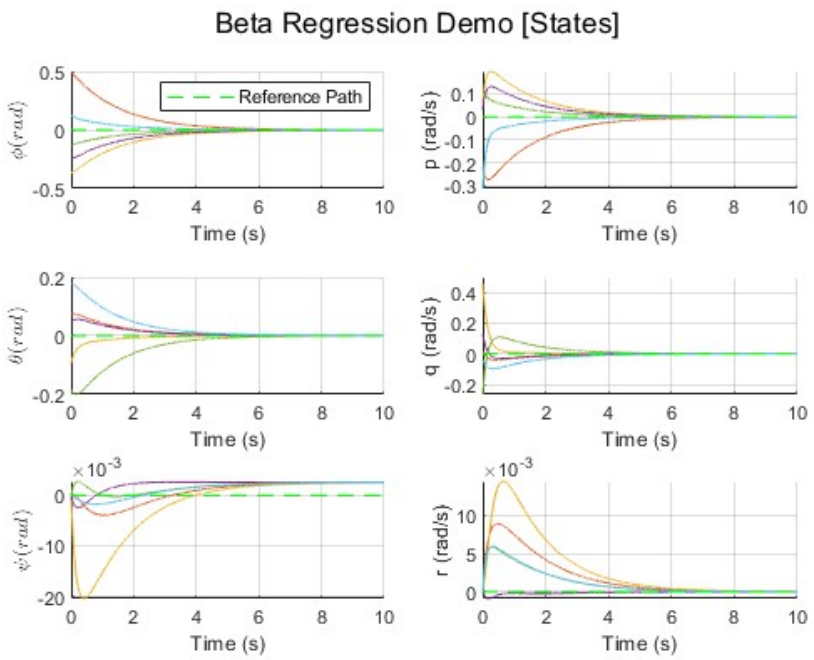

Figure 13: State performance demonstration on regression (beta)

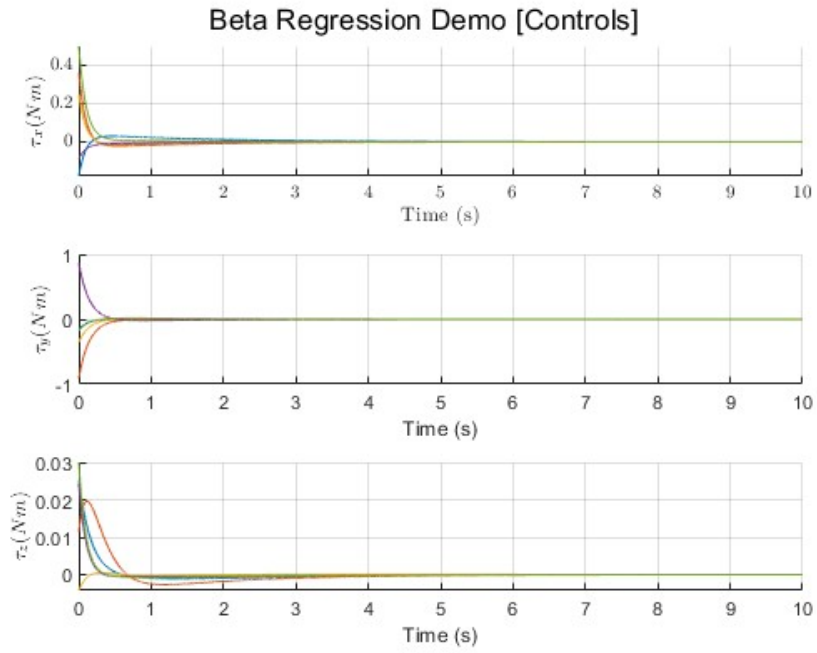

Figure 14: State performance demonstration on regression (beta)

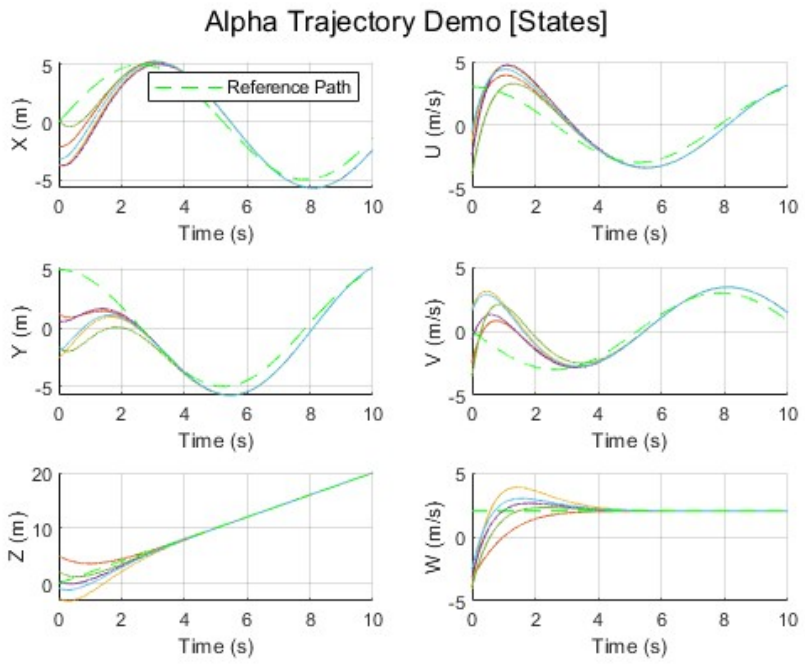

Figure 15: State performance demonstration on tracking (alpha)

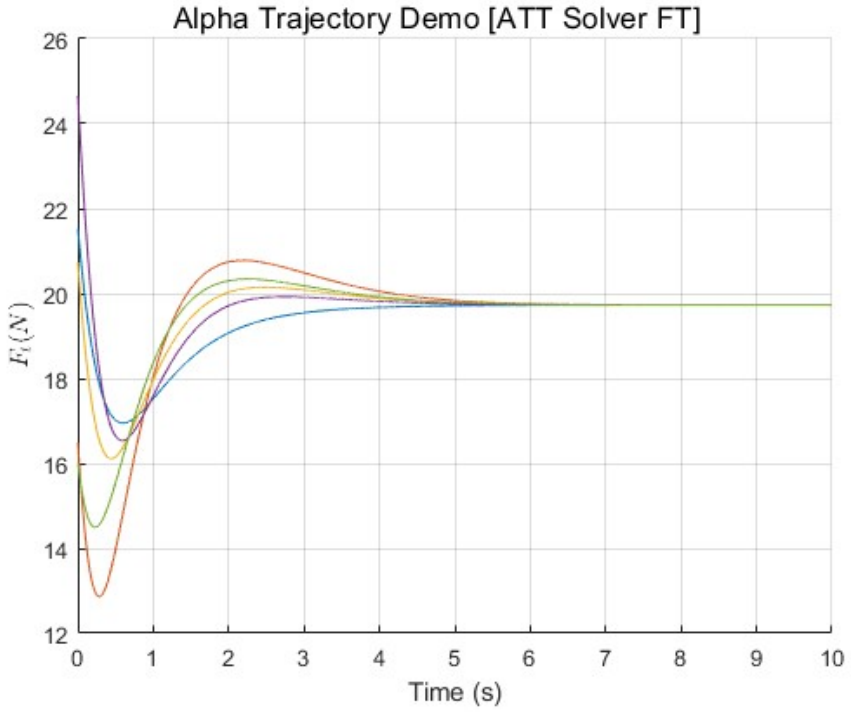

Figure 16: Solved thrust force from demonstration on tracking (alpha)

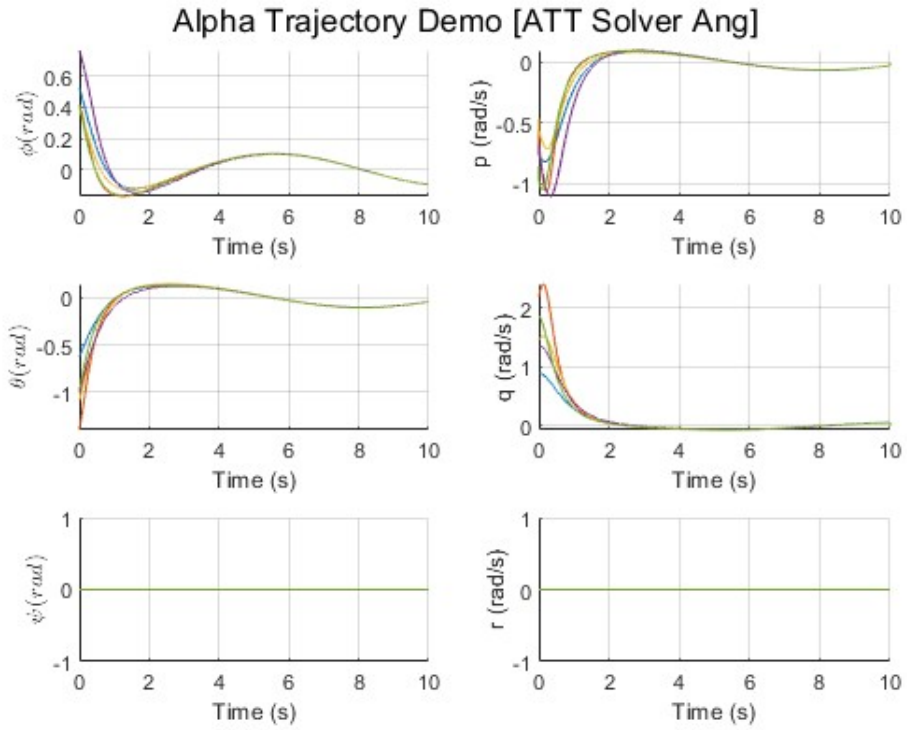

Figure 17: Solved attitude from demonstration on tracking (alpha)

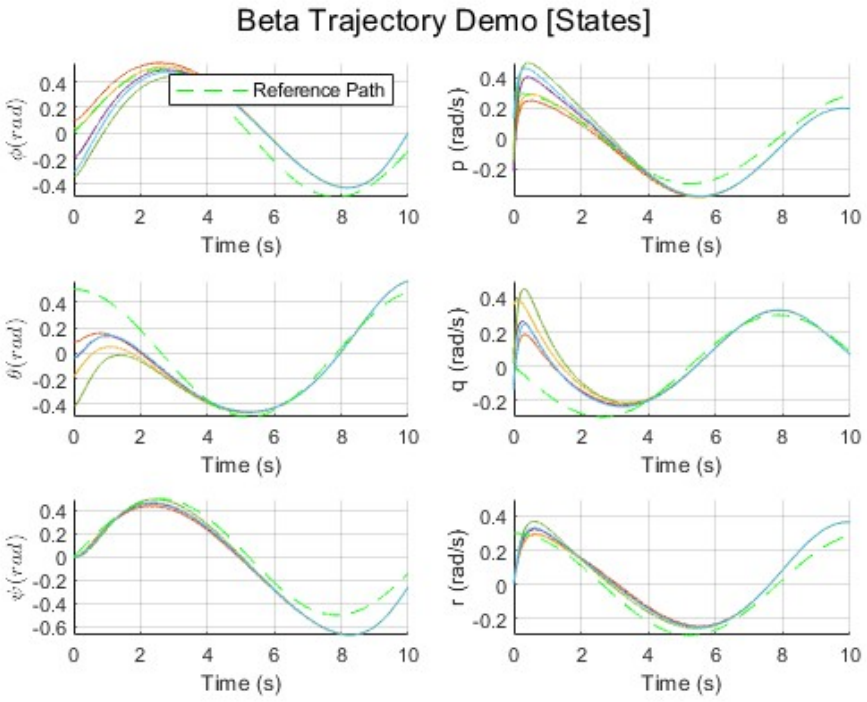

Figure 18: State performance demonstration on tracking (beta)

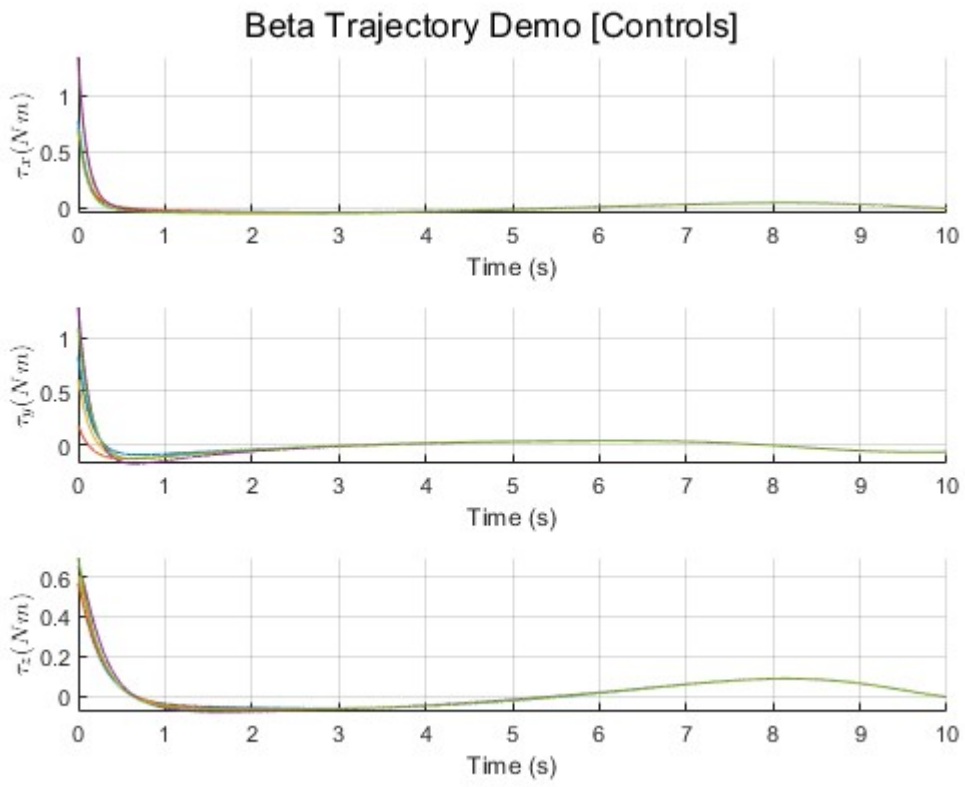

Figure 19: Control performance demonstration on tracking (beta)

4.2 Controller Implementation

The proposed controller aims to combine the alpha and beta neural networks into one cohesive program. The program must find the optimal control signals given some initial position, initial velocity, and trajectory. The proposed controller is outlined in Figure 20.

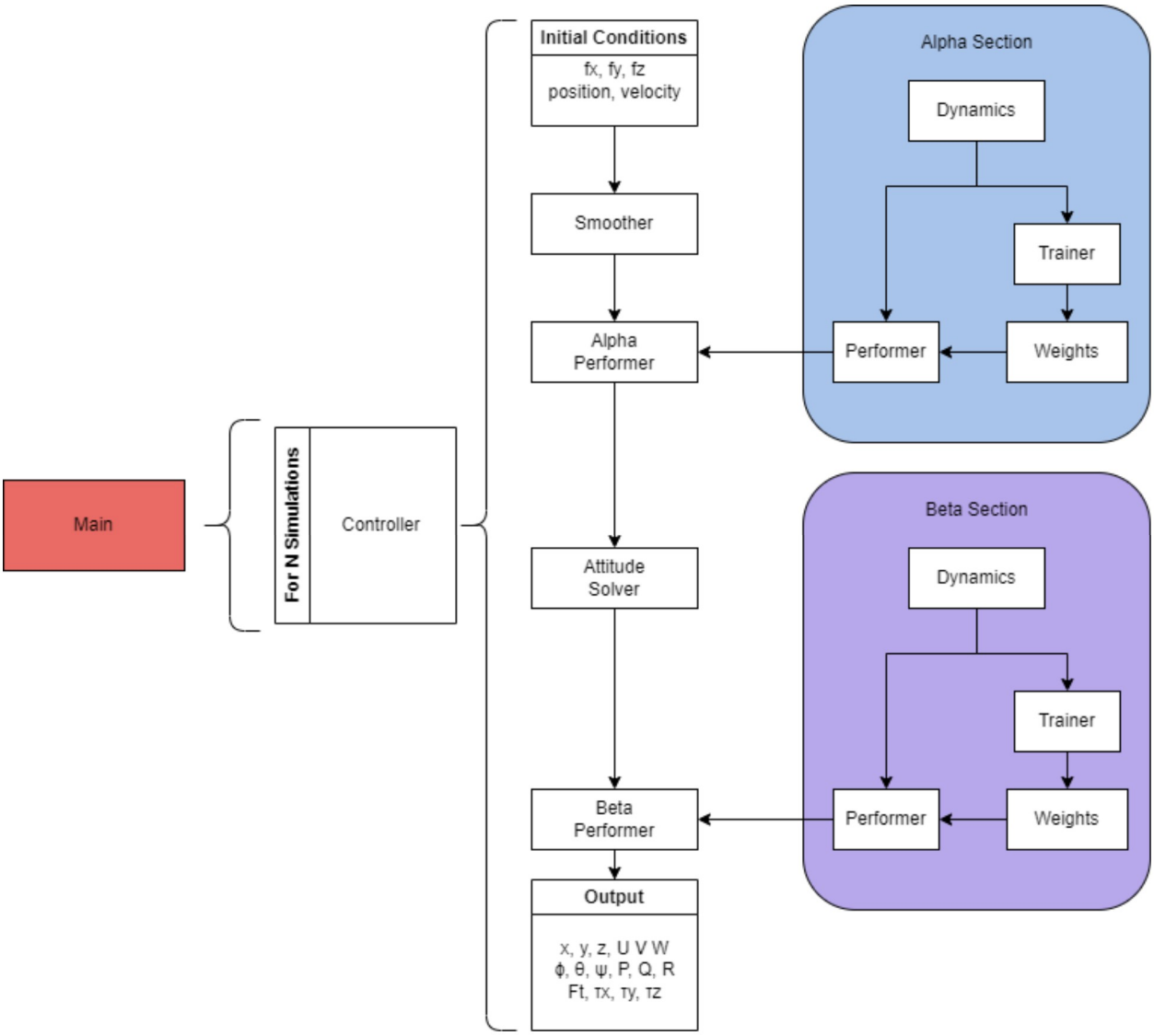

Figure 20: Flowchart of proposed diagram controller

The controller contains the two previously mentioned alpha and beta neural networks. Additionally contains a smoother and attitude solver function. These files generate the entire states and control signal during the complete simulation. The only necessary inputs are the trajectory functions $[f_x(t)\ f_y(t)\ f_z(t)]$, the initial position, and initial velocities. In this section, a detailed explanation of the implementation of the controller is given.

The first step is to smooth out the reference trajectory. The smoother function does two critical steps. First, it considers the case in which the initial position does not match the starting point of the trajectory. Second, it considers the initial velocity and modifies the beginning of the trajectory to accommodate. These two changes create a more realistic path for the quadrotor, resulting in smoother and more stable results. The equations used to parse the trajectory guarantee mathematical smoothness, and all deviations from the original trajectory are guaranteed to converge back to the original reference path. An example of this smoothness effect is shown in Figure 21, and the equations are shown in (27).

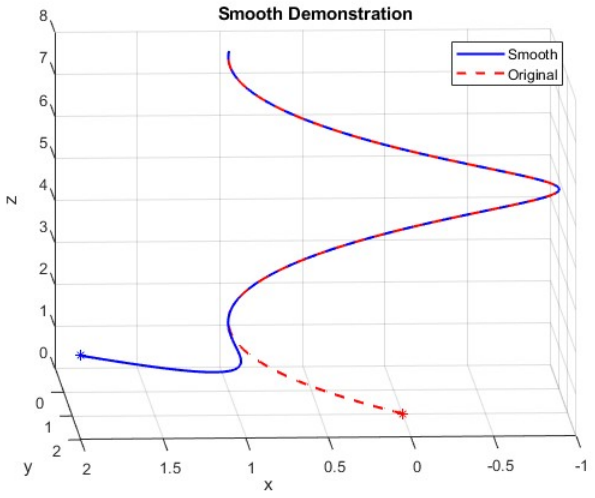

Figure 21: Demonstration of the smoother function

$$
\begin{aligned}
R'_{xyz}(t) &= [1-\phi(t^*)]V'_0(t) + \phi(t^*)R_{xyz}(t) \\
\bar{t} &= \frac{t}{T},\ \ \overline{V_0} = \frac{V_0}{|V_0|}\ for\ V_0 \neq 0 \\
t^* &= \frac{e^{-a\bar{t}} - 1}{e^{-a} - 1} \\
V'_0(t) &= \overline{V}\bar{t} + x_0 \\
\phi(t) &= \frac{\psi(t)}{\psi(t) + \psi(1-t)} \\
\psi(t) &= e^{-\frac{1}{t}}
\end{aligned}
\tag{27}
$$

After smoothing the reference trajectory, it is considered that the positional reference path is given to the alpha performer. Additionally, the path is discretely derived to find the velocity reference path. These six references provide the complete state reference for the alpha performer. The reference alongside the initial conditions is given to the alpha neural network to approximate the near-optimal accelerations $[\ddot{x}\ \ddot{y}\ \ddot{z}]$.

With the accelerations approximated, equation (20) is used to solve for the thrust force and the attitude. To simplify the model, the assumption $\psi = 0$ is made. With the angles solved, the angular velocities are calculated by discretely deriving. The states are sent to the beta performer with the angle and angular velocities solved.

The control signals generated from the beta neural network are the torque control signals required for the simulation. The required rotor speeds are solved with the four control signals solved. Figures 22 through 26 show example simulations of the controller. Each simulation has an identical trajectory with different initial positions. All simulations converge to the trajectory displaying the system's effectiveness in accurately learning the system's dynamics.

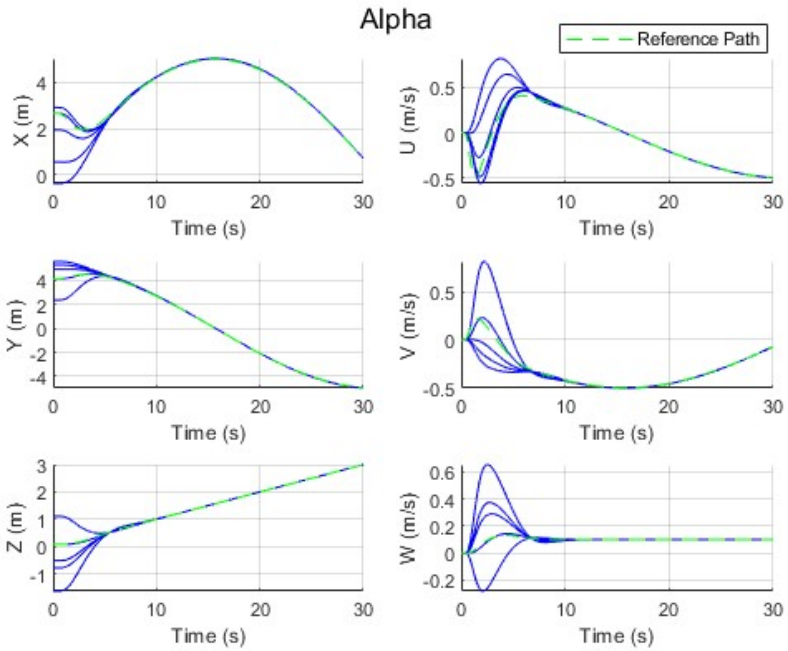

Figure 22: Example of positional controller states

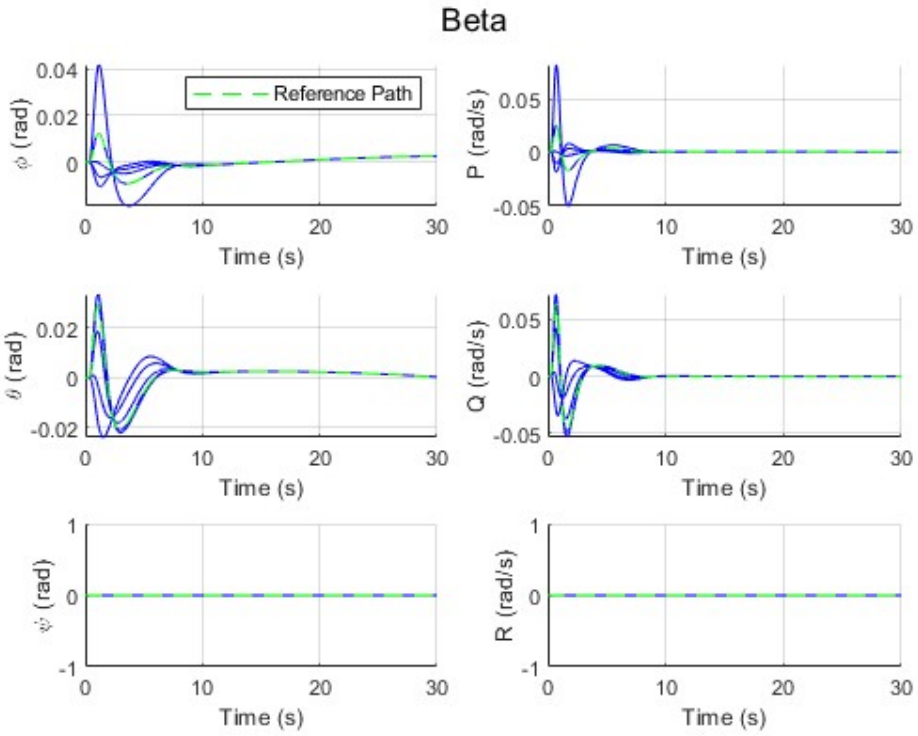

Figure 23: Example of attitude controller states

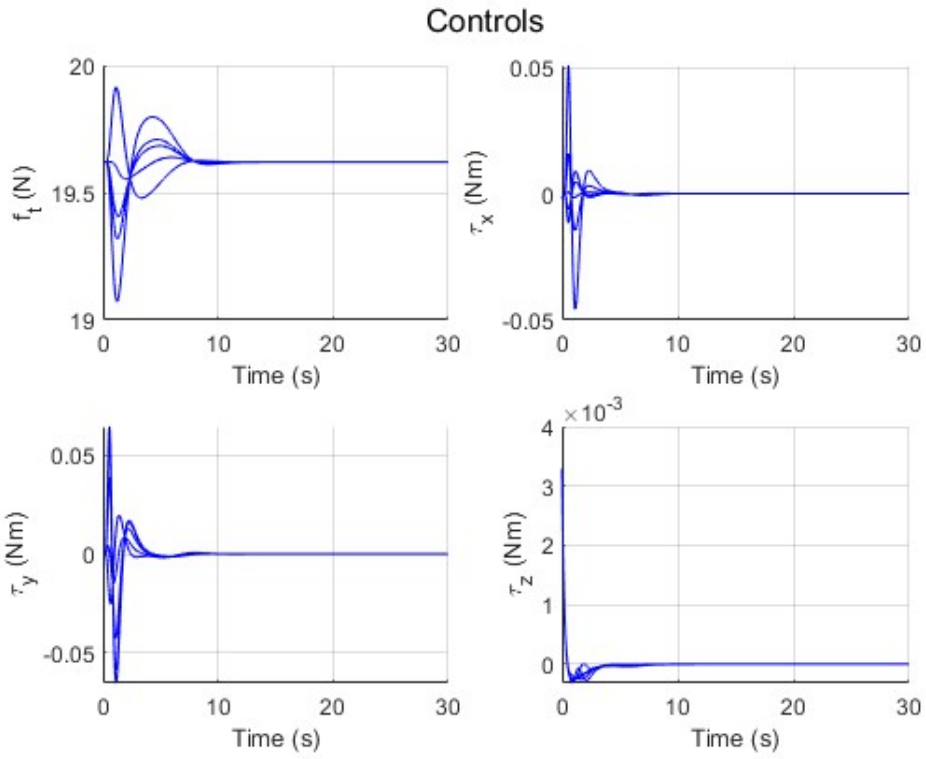

Figure 24: Example of generated control signals

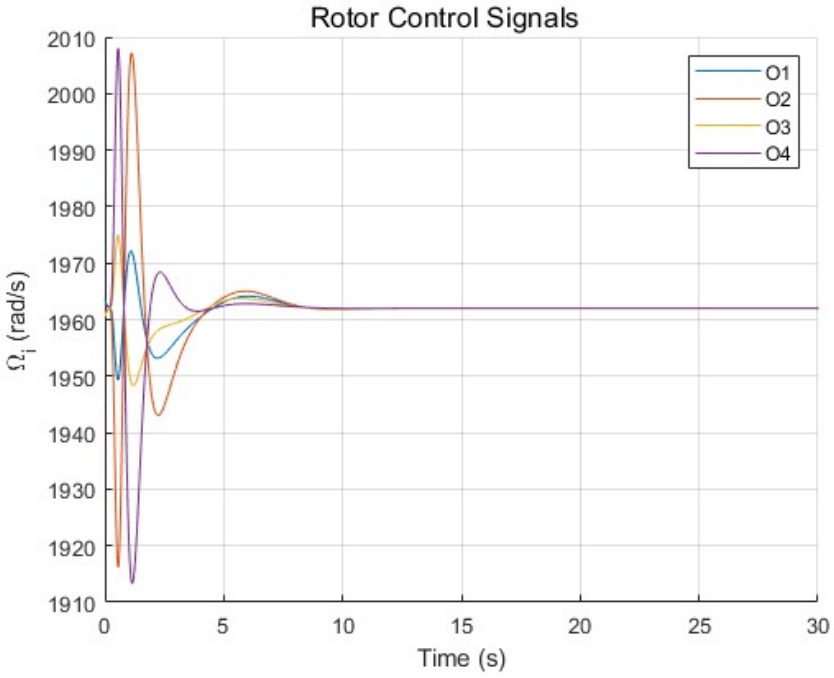

Figure 25: Example of generated rotor speeds

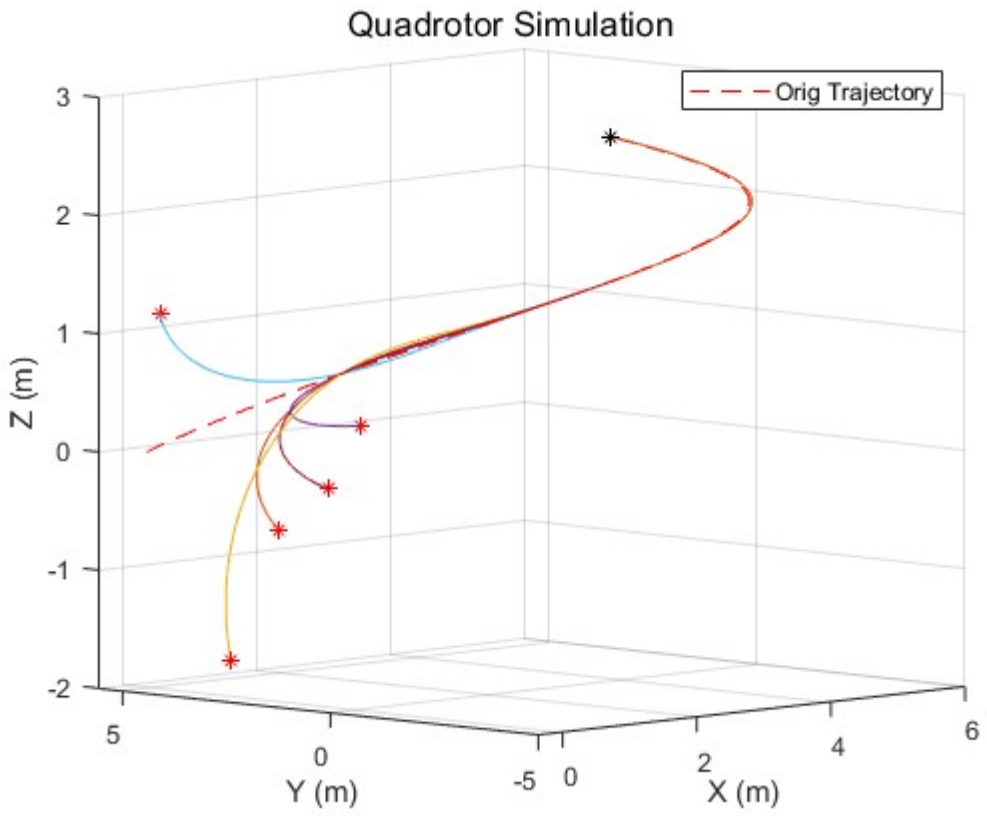

Figure 26: Example of generated simulation paths

CHAPTER V

CONCLUSION

5.1 Controller Results and Analysis

Figures 27 through 29 show the entire simulation data performed under the proposed quadrotor controller. The quadrotor was simulated for 50 seconds to follow a helix-shaped trajectory. The entire dataset is composed of the states: $[x\ y\ z\ \phi\ \theta\ \psi\ u\ v\ w\ p\ q\ r]$, controls: $\left[f_t\ \tau_x\ \tau_y\ \tau_z\right]$, the rotor speeds $[\Omega_1\ \Omega_2\ \Omega_3\ \Omega_4]$, and the references for alpha and beta performers. Figure 32 shows a 3D plot of the simulated trajectory. It also shows a comparison of the smooth trajectory versus the raw trajectory. The red dashed line represents the input trajectory, and the red star represents the initial starting position. The green dashed line is the modified trajectory applied with the smoother function. The green line smooths out the starting position with the given trajectory and is guaranteed to converge with the red dashed line. The yellow line is the simulated trajectory aiming to follow the modified trajectory. Figure 33 shows the results from the alpha neural network. The green dashed line is the reference path generated from the modified trajectory and by numerically deriving the speeds. The alpha neural network then outputs the optimal control solution shown in Figure 29.

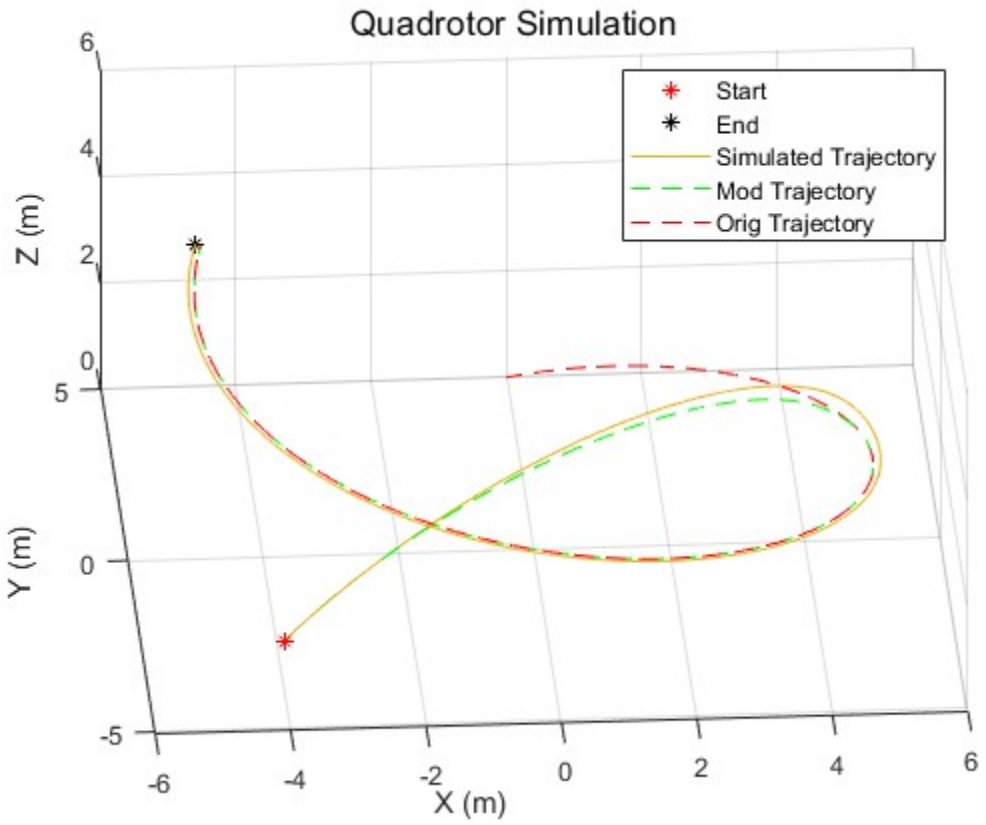

Figure 27: Simulated 3D plot comparing the original and modified trajectories

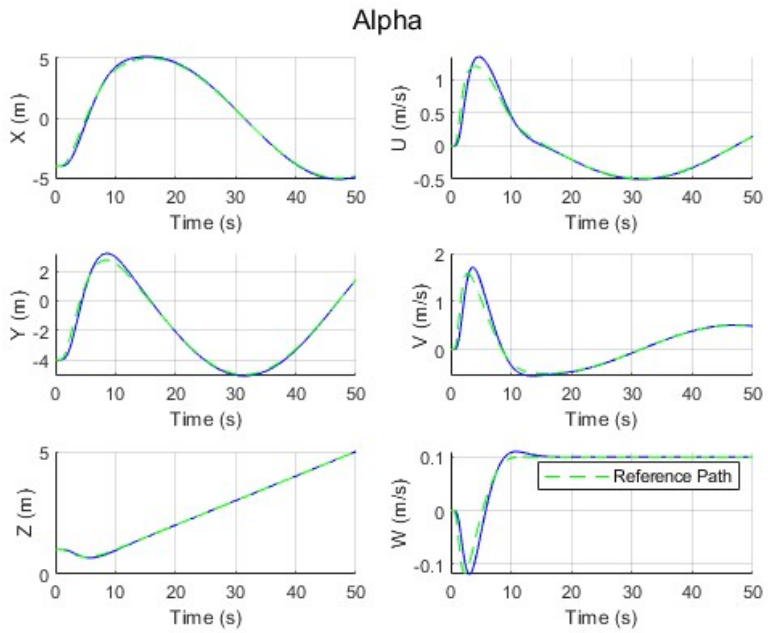

Figure 28: State simulation done by the alpha neural network

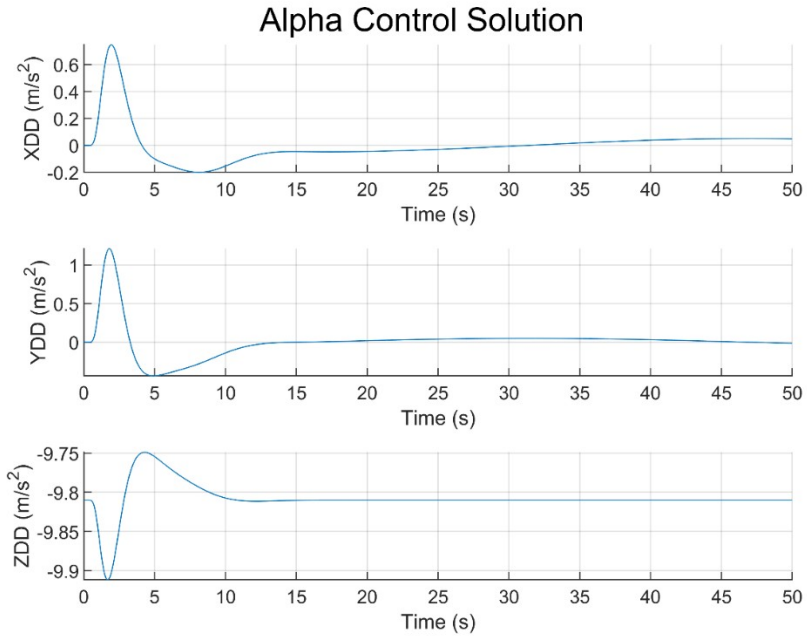

Figure 29: Optimal control solution found by the alpha neural network

The solution is passed to an attitude solver, which uses equation (20) to solve for the angles, angular speeds, and required thrust of the system. This data is then used as a reference for the beta neural network. In Figure 30, the green dashed line represents the calculated data from the attitude solver. The beta neural network then tracks the necessary attitude and solves the necessary torques for the system. Figure 30 and Figure 31 shows the simulated states and controls from the neural network. Combining the calculated thrust from the alpha neural network and the torques from the beta neural network, the required motor speeds can be found and are shown in Figure 32.

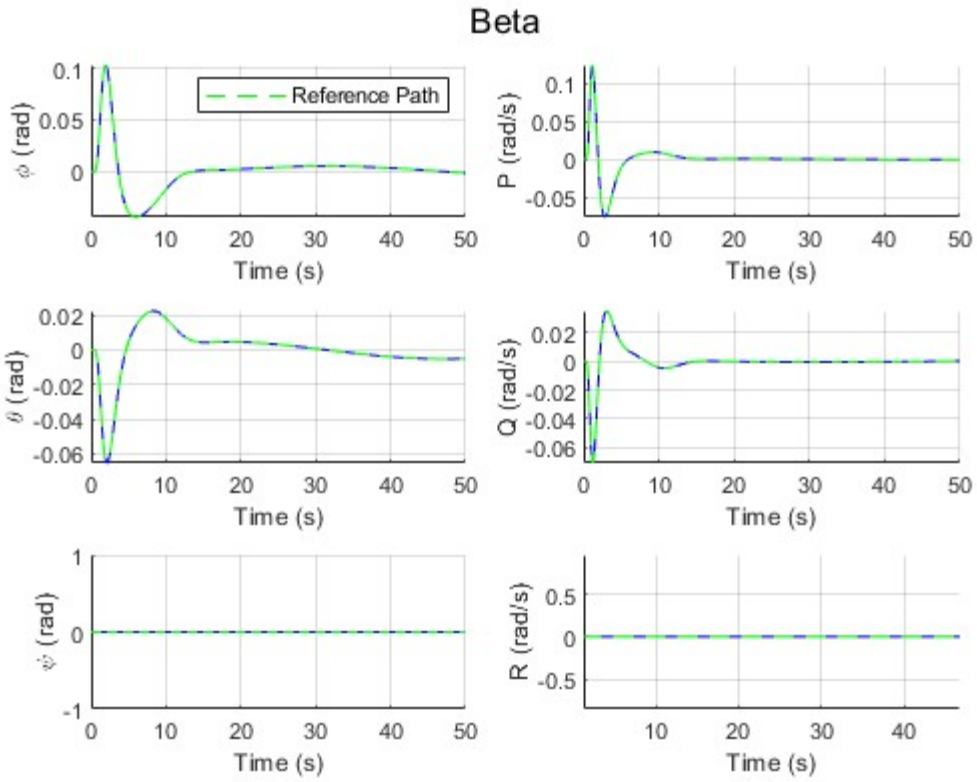

Figure 30: Simulation results from the beta neural network

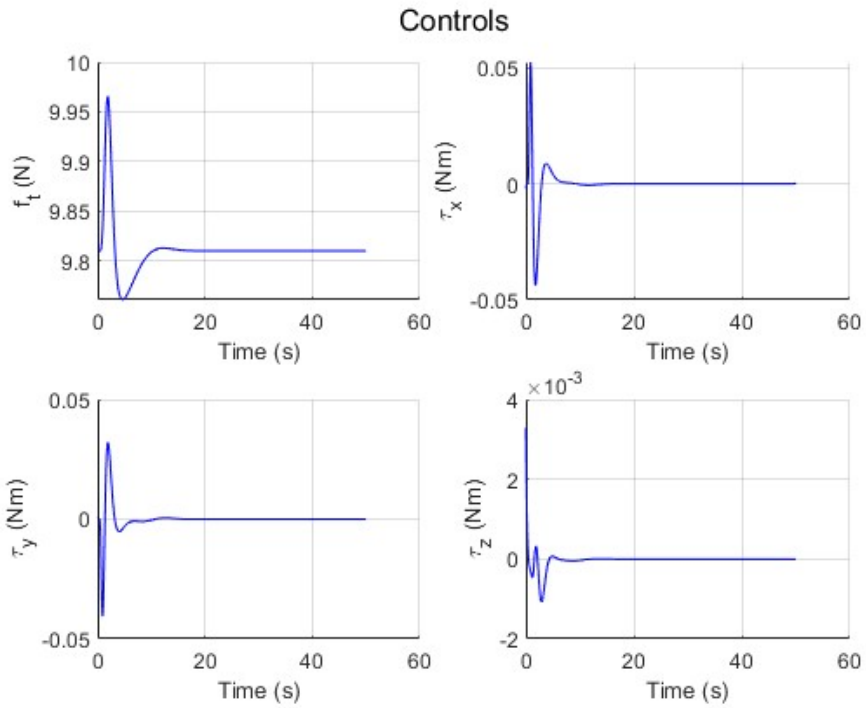

Figure 31: Optimal control solution found by the beta neural network

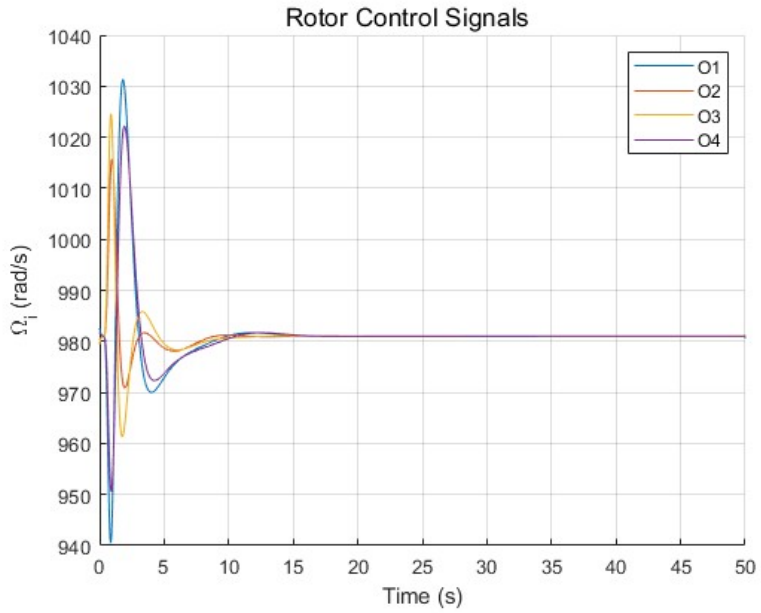

Figure 32: The angular speeds of the motor solved by the controller

From the total 50 seconds of the quadrotor, as simulated, it took the system 20 seconds to stabilize and converge to the given reference path. This can be best noticed in the plot for the simulated W speeds. The velocity overshoots to a max of 0.11 [m/s] and then to the optimal 0.1 [m/s].

5.2 Results Comparison (Modified trajectory)

From the total 50 seconds of the quadrotor, as simulated, it took the system 10 seconds to stabilize and converge to the given reference path. Figures 33 and 34 show an identical path traveled as Figures 27 and 28. The difference is in the overshoot caused by the initial position mismatching the reference path. Removing the smooth function raises the overshoot to 1.65 [m/s], 2.54 [m/s], and -0.19 [m/s].

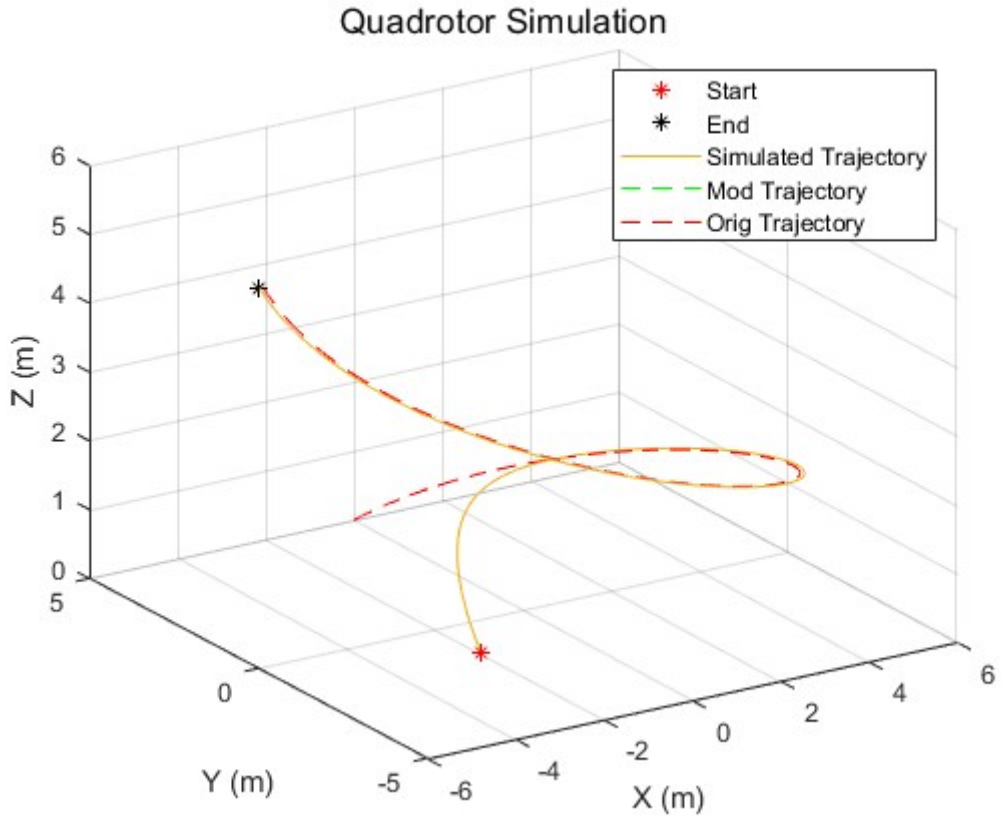

Figure 33: Control simulation with no modified trajectory

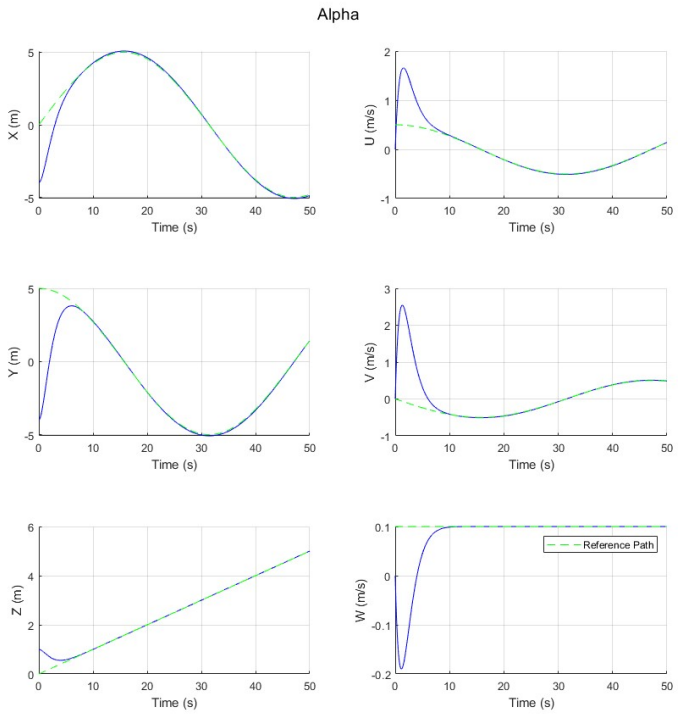

Figure 34: Alpha simulation with no smoother function

5.3 Results Comparison (Noise)

To test the robustness of the controller, noise is added to the simulation. The generated controls were multiplied by random noise with coefficients of 5%, 25%, and 100%. Figures 35 through 37 show the controller with 5% added noise, Figures 38 through 41 show 25% noise and Figures 41 through 43 show 100% noise. The critical points of these are the increase in the amplitude of the control signals generated. The increase in noise creates unstable paths through the trajectory, as seen with 100% noise. It also has a significant impact on the amplitude of the control signal reach. However, these major peaks do not create peaks in the position or attitude of the simulation. In all noise cases, the attitude never reaches an impossible or dangerous angle

and, in general, appears primarily unaffected. In practice, it appears the controller is robust against noise in its data. The controller is still able to manage to track the trajectory.

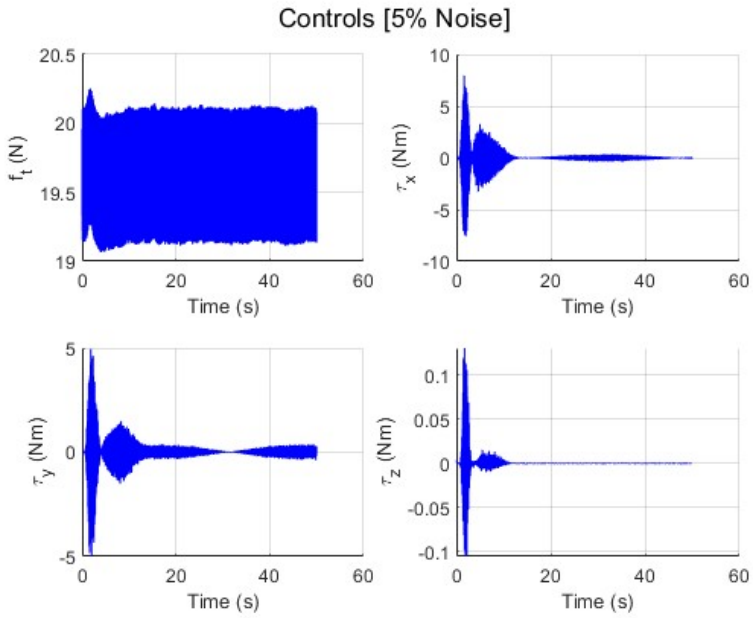

Figure 35: Control simulation with 5% Noise

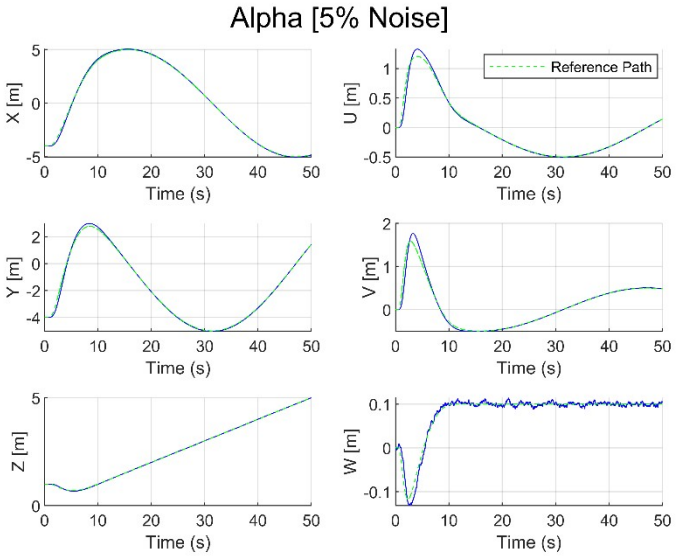

Figure 36: Alpha simulation with 5% Noise

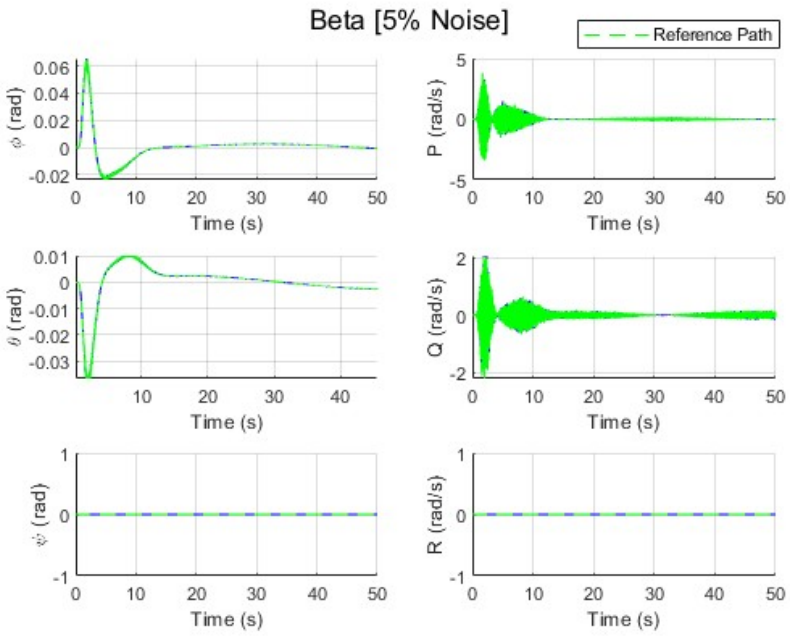

Figure 37: Beta simulation with 5% Noise

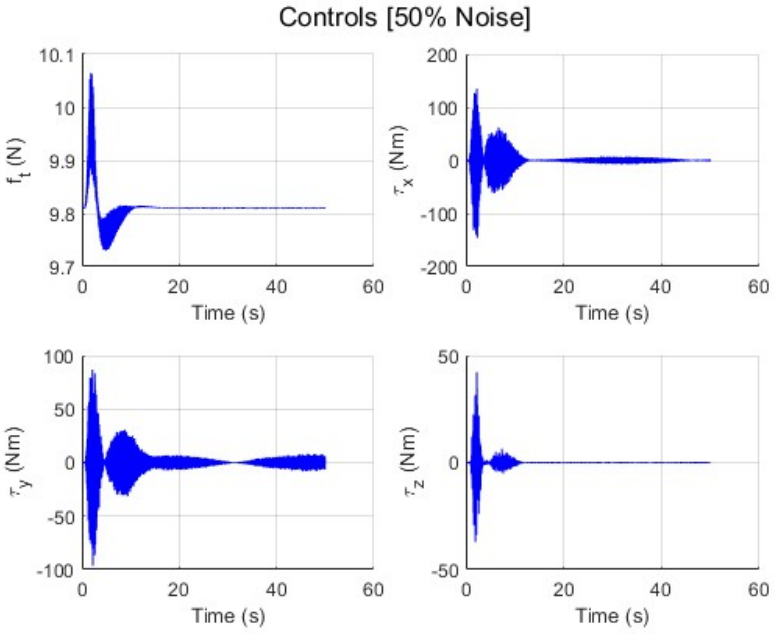

Figure 38: Control simulation with 50% Noise

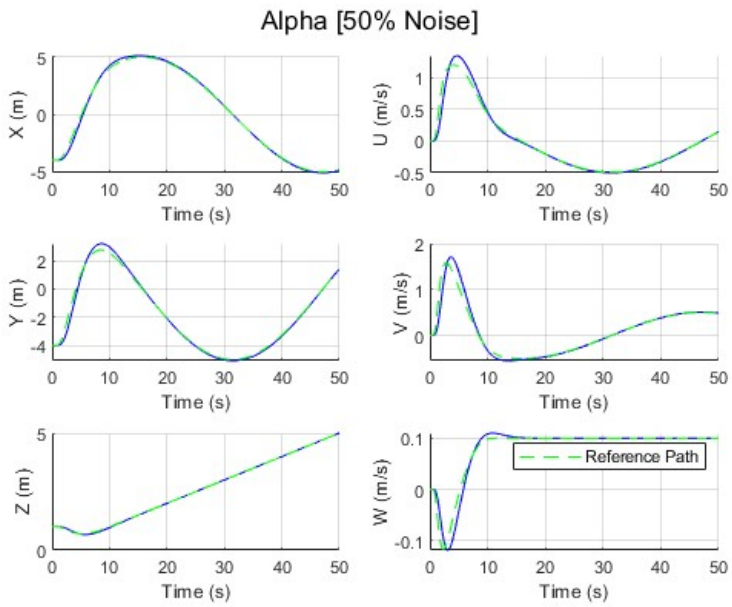

Figure 39: Alpha simulation with 50% Noise

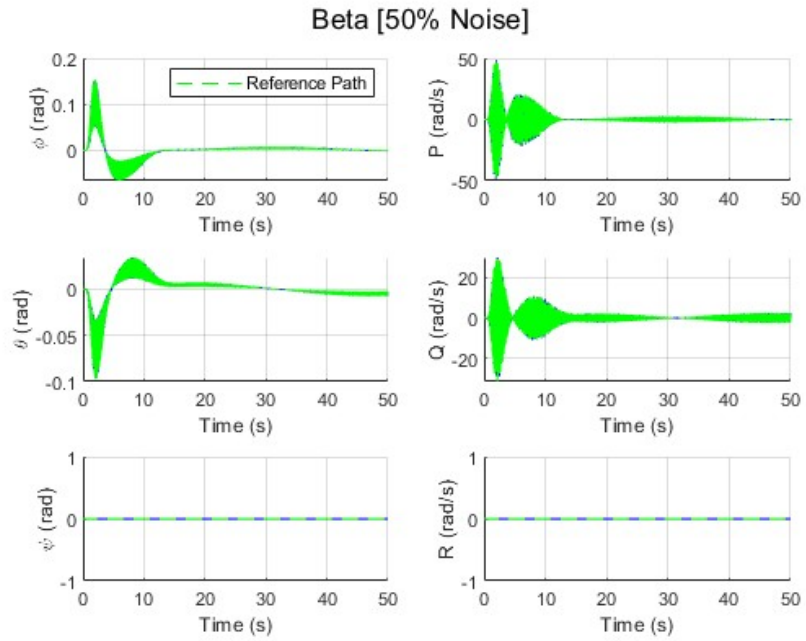

Figure 40: Beta simulation with 50% Noise

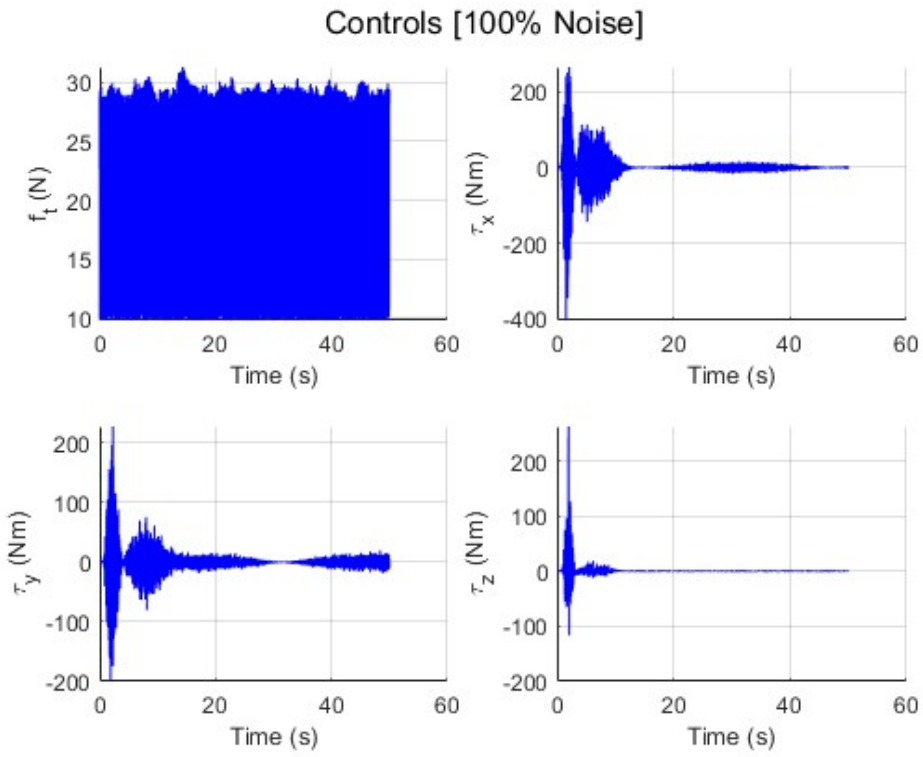

Figure 41: Contol simulation with 100% Noise

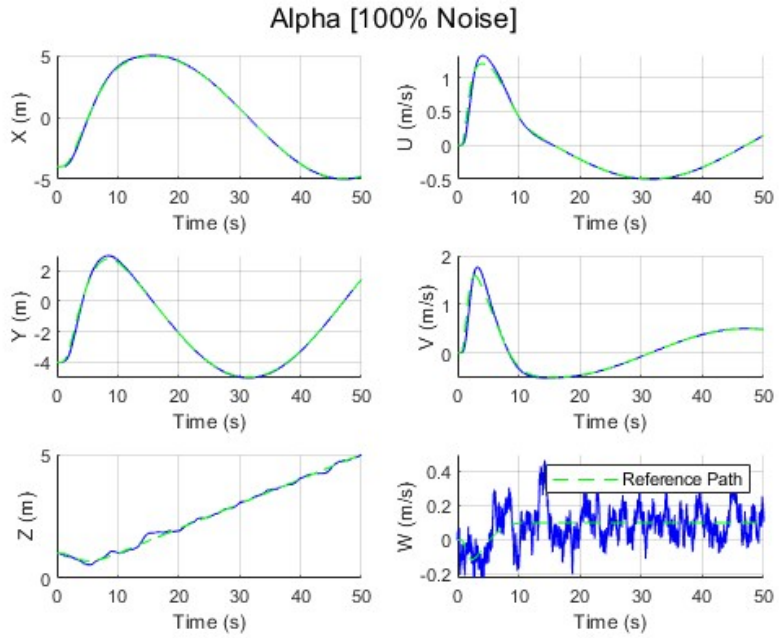

Figure 42: Alpha simulation with 100% Noise

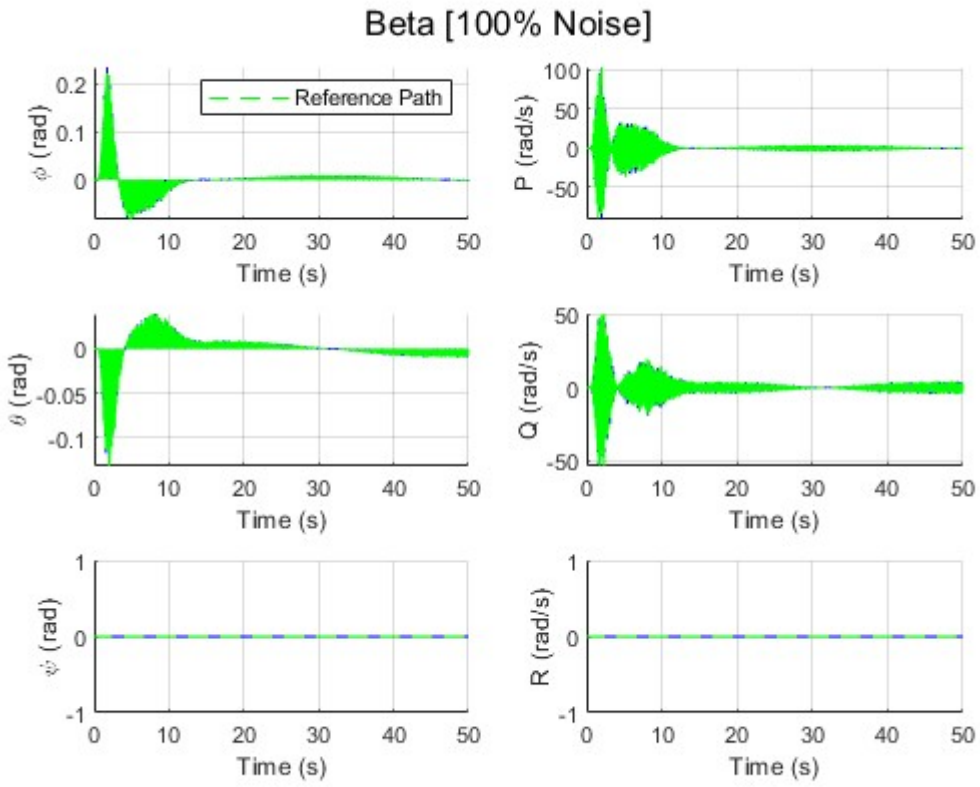

Figure 43: Beta simulation with 100% Noise

5.4 DIDO Comparison

DIDO is proprietary software for solving optimal control solutions. The system finds an open loop solution meaning the system uses no feedback when formulating its solution. DIDO finds this solution using Pontryagin's maximum principle. DIDO is a powerful program being able to find solutions along a set state boundary. In Figure 44 and Figure 45, the alpha dynamics are implemented into DIDO. As seen in Figure 45, DIDO creates large spikes in the velocity simulation, unlike the actor-critic solutions that show smoother and more realistic velocity samples.

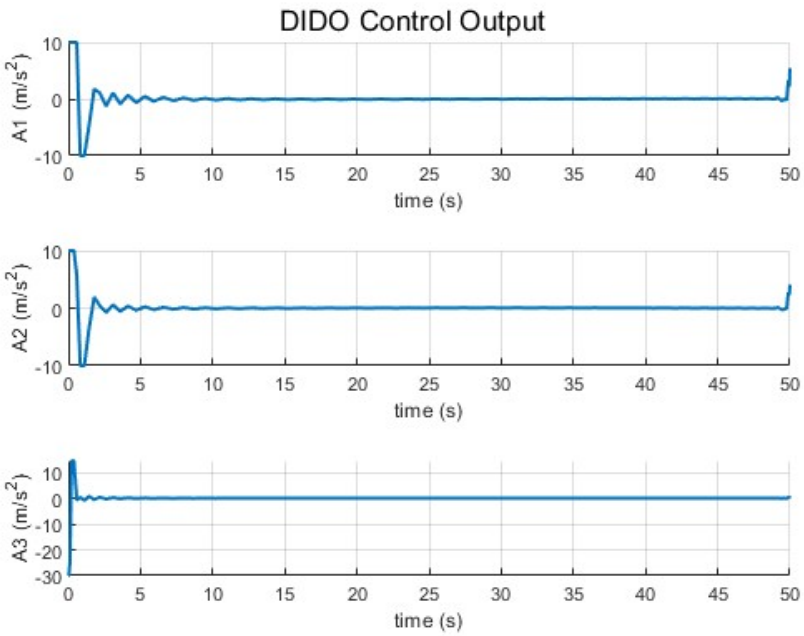

Figure 44: DIDO Alpha Control Output

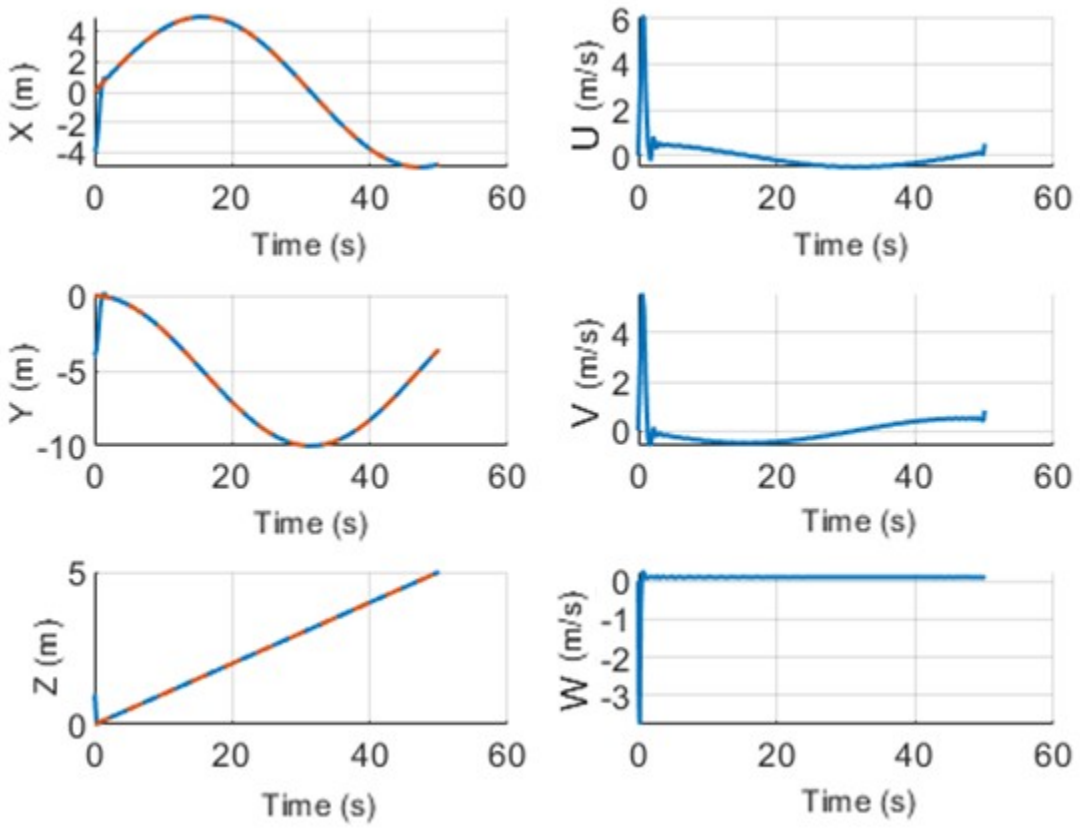

Figure 45: DIDO alpha state output

5.5 Future work

For future work, the controller needs robustness implemented at the dynamic levels. Currently, some robustness is possible due to the neural network's effectiveness in handling large noise samples. However, implementing the wind forces and torques $\left[f_{wx}\ f_{wy}\ f_{wz}\ \tau_{wx}\ \tau_{wy}\ \tau_{wz} \right]$ into the state, dynamics may yield improved results. Additionally, the controller can be improved by removing the small angle approximation implemented into the dynamics.

REFERENCES

Bellman, R. (1957). *Dynamic Programming*. Princenton University Press.

Burggräf, P., Pérez Martínez, A. R., Roth, H., & Wagner, J. (2019). Quadrotors in factory applications: design and implementation of the quadrotor's P-PID cascade control system. *SN Applied Sciences*, *1*(7), 1-17.

Chi, W., Ji, Y., & Gao, Q. (2019, June). Attitude tracking control for a quadrotor via backstepping and adaptive dynamic programming. In *2019 Chinese Control And Decision Conference (CCDC)* (pp. 3108-3113). IEEE.

Commercial, U. A. V. (2016). Commercial Drone Market Analysis By Product (Fixed Wing, Rotary Blade, Nano, Hybrid), by Application (Agriculture, Energy, Government, Media & Entertainment) and Segment Forecasts to 2022. *Grand View Research: San Francisco, CA, USA*.

Das, A., Lewis, F., & Subbarao, K. (2009). Backstepping approach for controlling a quadrotor using lagrange form dynamics. *Journal of Intelligent and Robotic Systems*, *56*, 127-151.

De Bruin, A., & Booysen, T. (2015). Drone-based traffic flow estimation and tracking using computer vision: transportation engineering. *Civil Engineering= Siviele Ingenieurswese*, *2015*(8), 48-50.

EpsilonDelta. (2022, August 16). *Smooth Transition Function in One Dimension | Smooth Transition Function Part 1* [Video]. Youtube. https://www.youtu.be/ vD5g8aVscUI

Flammini, F., Pragliola, C., & Smarra, G. (2016, November). Railway infrastructure monitoring by drones. In *2016 International Conference on Electrical Systems for Aircraft, Railway, Ship Propulsion and Road Vehicles & International Transportation Electrification Conference (ESARS-ITEC)* (pp. 1-6). IEEE.

Heydari, A., & Balakrishnan, S. N. (2013). Fixed-final-time optimal control of nonlinear systems with terminal constraints. *Neural Networks*, *48*, 61-71.

Kirk, D. E. (2004). *Optimal control theory: an introduction*. Courier Corporation.

Lewis, F. L., & Vrabie, D. (2009). Reinforcement learning and adaptive dynamic programming for feedback control. *IEEE circuits and systems magazine*, *9*(3), 32-50.

Marvi, Z., & Kiumarsi, B. (2021). Safe reinforcement learning: A control barrier function optimization approach. *International Journal of Robust and Nonlinear Control*, *31*(6), 1923-1940.

Muñoz, L. E., Santos, O., Castillo, P., & Fantoni, I. (2013, June). Energy-based nonlinear control for a quadrotor rotorcraft. In *2013 American Control Conference* (pp. 1177-1182). IEEE.

Pereira, A. A., Espada, J. P., Crespo, R. G., & Aguilar, S. R. (2019). Platform for controlling and getting data from network connected drones in indoor environments. *Future Generation Computer Systems*, *92*, 656-662.

Sabatino, F. (2015). Quadrotor control: modeling, nonlinear control design, and simulation.

Schmid, K., Hirschmüller, H., Dömel, A., Grixa, I., Suppa, M., & Hirzinger, G. (2012). View planning for multi-view stereo 3d reconstruction using an autonomous multicopter. *Journal of Intelligent & Robotic Systems*, *65*(1), 309-323

Siebert, S., & Teizer, J. (2014). Mobile 3D mapping for surveying earthwork projects using an Unmanned Aerial Vehicle (UAV) system. *Automation in construction*, *41*, 1-14.

Stingu, E., & Lewis, F. L. (2011, April). An approximate dynamic programming-based controller for an underactuated 6dof quadrotor. In *2011 IEEE*

Symposium on Adaptive Dynamic Programming and Reinforcement Learning (ADPRL) (pp. 271-278). IEEE.

Van der Merwe, D., Burchfield, D. R., Witt, T. D., Price, K. P., & Sharda, A. (2020). Drones in agriculture. *Advances in agronomy*, *162*, 1-30.

Yang, Y., Yin, Y., He, W., Vamvoudakis, K. G., Modares, H., & Wunsch, D. C. (2019, July). Safety-aware reinforcement learning framework with an actor-critic-barrier structure. In *2019 American Control Conference (ACC)* (pp. 2352-2358). IEEE.

APPENDIX

APPENDIX

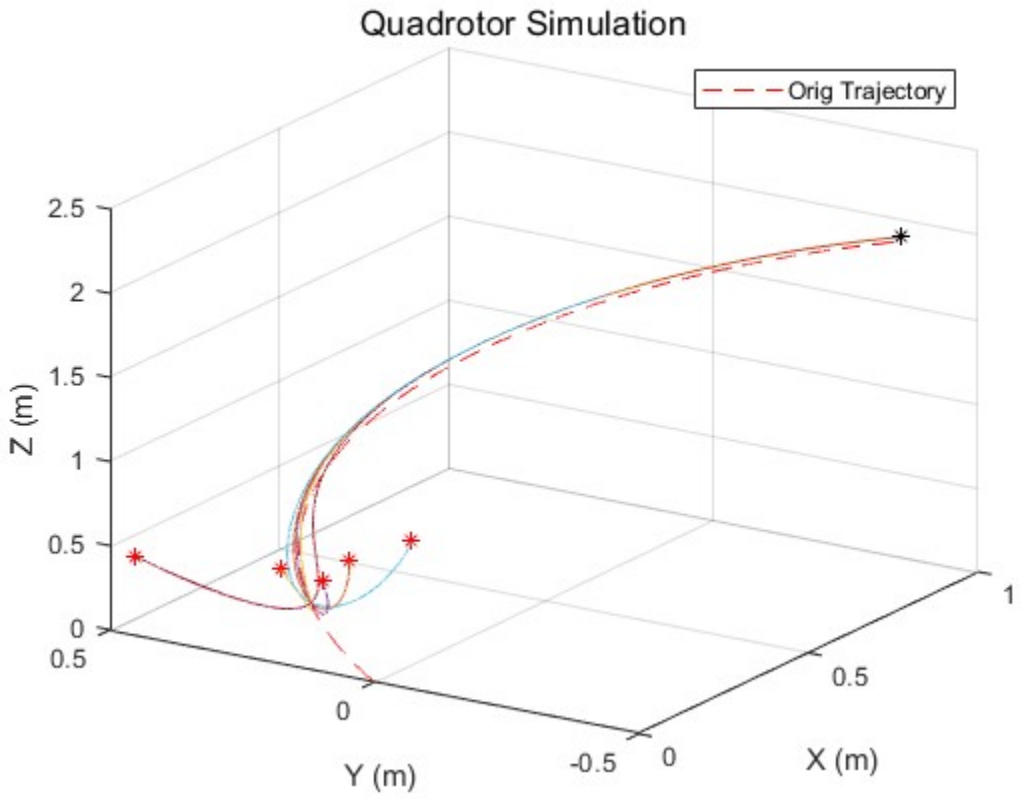

Figure 46: Sample quadrotor simulation

BIOGRAPHICAL SKETCH

Edgar Adrian Torres was born in Brownsville, Texas. He attended the Mathematics and Science Academy at UTRGV and graduated high school under the class of 2018. He started his undergraduate studies in UTRGV the following year. He graduated with Magna Cum Laude at May 2021. He started his graduate studies right after and completed his Master of Science in Mechanical Engineering on May 2023. He can be reached at adrianedgar021@gmail.com.