Design of a Maneuverable Rocket Using Machine Learning Tuning

A Project presented to

The Faculty of the Department of Aerospace Engineering

San José State University

in partial fulfillment of the requirements for the degree

Master of Science in Aerospace Engineering

by

William L Miller Jr

December 2021

approved by

Professor Long Lu

Faculty Advisor

ABSTRACT

Design of a Maneuverable Rocket Using Machine Learning Tuning

William L Miller Jr

Aerospace engineering has brought ingenious concepts to the forefront of science and technology. With the foray into self-landing rockets and launch vehicles come even greater challenges. This project seeks to expand upon the novel challenges of controls. Designing and implementing a PID controller that serves as a foundation for greater innovation to come. The work of this project accomplished the design of a viable control system. After which the controller was then tuned by linear regressive methods in Python using the Scikit learn package. A comparison between computer automated controller tuning, hand tuning, and the machine learning algorithmic tuning was then made. The results concluded that specialized machine learning methods for specific situations can lead to overall better performance gains when implemented. The maneuverable rocket in flight is proof of concept and design, which will lead to greater control innovation in the future.

ACKNOWLEDGEMENTS

I would like to thank my mother and father for all their support from my undergraduate career until now. The two people who believed, motivated, and stood by me no matter how arduous the road was. I give you two my heart, soul, and all the love that I can muster. Lastly, but not least, I would also like to thank my professor, and faculty advisor Long Lu. Every class and subject introduced to me through him has raised me to a higher academic level and responsibility. Thank you.

Table of Contents

LIST OF FIGURES

LIST OF TABLES

NOMENCLATURE

U	X-Direction Velocity
V	Y-Direction Velocity
W	Z-Direction Velocity
$\dot{U}$	X-Direction Acceleration
$\dot{V}$	Y- Direction Acceleration
$\dot{W}$	Z-Direction Acceleration
P	Angular Velocity in Roll
Q	Angular Velocity in Pitch
R	Angular Velocity in Yaw
$\dot{P}$	Angular acceleration in Roll
δ_{Θ}	Gimbal Angle 1
δ_{Ψ}	Gimbal Angle 2
$\dot{Q}$	Angular acceleration in Pitch
$\dot{R}$	Angular Acceleration in Yaw
F_A	Aerodynamic Force
F_P	Propulsion Force
F_G	Gravity Force
L_*	Roll Induced Moment
M_*	Pitch Induced Moment
N_*	Yaw Induced Moment
$\overrightarrow{\omega}$	Angular velocity rad/s
K_P	Proportional Gain Setting
K_D	Derivative Gain Setting
K_I	Integral Gain Setting
Y	Predicted Value

Chapter 1-Introduction

1.1 Motivation

Rockets have become increasingly advanced as the passage of time progresses. Although the purpose for a rocket aerial vehicle has steadily remained the same, technology has not. However, until as of late there has not been an effort to mobilize rocket vehicles while in flight. That is until the 21st century, with the advent of a new space age maneuverable rockets have shot to the forefront. A prime example of an advanced maneuverable rocket is the reusable rocket the Falcon 9.

With the introduction of new more maneuverable rockets unfamiliar problems, and solutions have surfaced within the aerospace industry. This thesis seeks to apply the maneuverability of a rocket in active flight not just during the landing phase. This is to combat certain problems such as collision avoidance, tracking detection, and/ or flight trajectory corrections. Extending the maneuverability of rockets into real time flight corrections could potentially solve problems that have yet to arise within the aerospace field. To achieve maneuverability of a rocket there are existing solutions that can be implemented that would work in a novel way.

The most common types of controls rocket use as stated NASA are movable fins, vernier thrusters, and thrust vectoring [1]. Using these three types of rocket controls and integrating them together could create a unique solution to this maneuverability problem. That is exactly what this project sets out to achieve. Most modern rockets of this era use primarily thrust vectoring for rocket control and stability [2]. The main type of thrust controls will also be thrust vectoring, with Vernier thrusters providing secondary support. Movable fins will be of the least importance and the last option to be explored within this project.

Exploring these options and integrating them within a control system can solve the issue of rocket maneuverability within flight. In doing so this could potentially solve problems that have not yet been recognized or presented in today's aerospace landscape. Solving tomorrow's problems now.

1.2 Literature Review

This section will introduce the governing equations of motion of an airframe, as well as rocket body aerodynamics and the types of controls used. These topics will include the following points:

- Governing Equations of Motion for an Air Frame
- Aerodynamic Considerations & Estimation
- Gravity Considerations
- Propulsion consideration
- Falcon 9 Rocket Characteristics

These points are critical to the implementation and design of the proposed rocket control system of this project. Without understanding these topics beforehand and having a firm grasp of the material proper design and mathematical modeling cannot be readily achieved.

The first and the most nontrivial step to any dynamical system modeling and design is the mathematical model. Many papers cover the dynamical system of a rocket, or airframe and its motion through the air. This model is the six degree of freedom (6dof) equations of motion. These sets of mathematical equations describe both the translational motion of the airframe, as well as the rotational motion of the airframe [3]. Each set of these equations has three variables associated with them, to obtain the coordinates desired. These variables in the translational case are U, V, W. These three variables are the velocity components specified with respect to the body coordinate axis. The next three variables needed to describe the rotational motion are R, P, Q. These are associated with roll, pitch, and yaw about the body principal axis of the airframe as well. These 6 variables come together to structure six differential equations that describe the nonlinear motion of an airframe of motion through the air. In this case these three variables will describe the motion of the simulated Falcon 9 rocket through the atmosphere. The six equations of motion that govern motion are as follows below:

$$\dot{U} = \frac{F_A+F_P+F_G}{m} - (qw - rv), \text{ m/s}^2 \quad (1.1)$$

$$\dot{V} = \frac{F_A+F_P+F_G}{m} - (ru - pw), \text{ m/s}^2 \quad (1.2)$$

$$\dot{W} = \frac{F_A+F_P+F_G}{m} - (pv - qu), \text{ m/s}^2 \quad (1.3)$$

$$\dot{P} = \frac{L_A+L_p-qr(I_z-I_y)}{I_x}, \text{ rad/s}^2 \quad (1.4)$$

$$\dot{Q} = \frac{M_A+M_p-rp(I_x-I_z)}{I_y}, \text{ rad/s}^2 \quad (1.5)$$

$$\dot{R} = \frac{N_A+N_p-pq(I_y-I_x)}{I_z}, \text{ rad/s}^2 \quad (1.6)$$

These six equations together represent the true motion through which an airframe moves through the air. Within these equations contain the force contributions from several different outside factors of which need to be either accurately estimated or tested for. Due to budget

constraints and the realistic nature of the project actual testing of for the rocket's aerodynamics were not done. Instead, software was used to estimate the aerodynamic effects of the Falcon 9 rocket. To estimate the aerodynamics of the Falcon 9 rockets multiple papers pointed to the direction of the software "Digital Datcom" [4]. This software was used by the USAF (United States Airforce) for the exact same reason, fast and accurate aerodynamic estimations. Although intended primarily for Aircraft, searching through references revealed that an accurate estimation can be obtained for a rocket. Normally one would want to use the missile related software "Missile Datcom" for such purposes but due to the military restrictions and nature of the software it is very restricted in distribution to say the least [5]. The Digital Datcom software was used to estimate the aerodynamic coefficients of the Falcon 9 rocket at angles of attack varying from -.07 rads (- 4 degrees) to .17 rads (10 degrees), and a Mach speed from one to five. This approach was taken from reference [5]. Where Digital Datcom was used to obtain the aerodynamic coefficients for a rocket, by using a technique where the rocket was broken into parts. These parts were the rocket body, and the rocket fins. After which the aerodynamic coefficients were compared to tested data to validate the results [6]. To maintain the approach and keep the accuracy obtained the same method was used for this project.

Obtaining the aerodynamic coefficients are an integral part of modeling and simulating the rocket flight through the air. As seen in the previous equations above the rocket will experience different forces through many different means. A huge contributing factor to these forces is the aerodynamic forces which will act on the rocket body [7]. These aerodynamic forces will affect the stability and control input needed to control the rocket through its long flight and will affect the attenuation needed by the rocket control system itself. All these factors will be seen in the mathematical modeling of the rocket system within the MATLAB and SIMULINK model itself as contributing factors. Next, after the aerodynamic forces are to be considered the gravity acting upon the rocket itself must also be attended to.

Gravity does not enact the same force at every height [8]. In fact, as you get further away from the center of earth gravity tends to weaken. This needs to be molded into the nonlinear model of the rocket to accurately represent the real-world physics of the problem. To do this many papers were read about the varying effects of gravity upon airframes, and the gravity model used within MATLAB itself. This ended with the following equation to model gravity being used within this project:

$$\vec{G} = G_x\hat{\imath} + G_y\hat{\jmath} + G_z\hat{k} \tag{1.7}$$

$$\overrightarrow{G_x} = \frac{\mu}{r^2}\left[1 + \frac{3}{2}J_2\left(\frac{R_o}{r}\right)^2\left[1 - 5\left(\frac{z}{r}\right)^2\right]\right]\left(\frac{x}{r}\right) \tag{1.8}$$

$$\overrightarrow{G_y} = \frac{\mu}{r^2}\left[1 + \frac{3}{2}J_2\left(\frac{R_o}{r}\right)^2\left[1 - 5\left(\frac{z}{r}\right)^2\right]\right]\left(\frac{y}{r}\right) \tag{1.9}$$

$$\overrightarrow{G_z} = \frac{\mu}{r^2}\left[1 + \frac{3}{2}J_2\left(\frac{R_o}{r}\right)^2\left[1 - 5\left(\frac{z}{r}\right)^2\right]\right]\left(\frac{z}{r}\right) \tag{1.10}$$

These equations make up the gravitational force including the affect from the Earth's center and will be used within the gravity model of this thesis.

The next consideration while researching rocket controls and their effects on rockets came to thrust vectoring, and the contributions of propulsion. While the rocket will not be vertical the while flight and the thrust will not always be in line with the axis of the body, thrust vectoring will cause a moment upon the body of the rocket. While researching about thrust vectoring it was also noted that in modern rocketry, thrust vectoring is one of the major controls of a rocket [11,12]. An example of this is the rocket used in this project, the Falcon 9. The Falcon 9 does not deploy fins in the launch stage. Only during landing does the Falcon 9 use its capable grid fins to stabilize the rocket. While in the launch phase and flight phase thrust, vectoring is primarily used to keep the rocket stable. Thrust vectoring will be the main component of control for this project as the entire flight envelope for this vehicle will be in flight, and not landing. The contribution of the propulsion to the added moments upon the rocket is easily remedied using the equation.

$$\vec{M}_p = \vec{R}\ x\ \overrightarrow{C_G} \qquad (1.11)$$

With the equation above the propulsive force contributions to the force acting upon the rocket is solved and ready to be implemented into the mathematical model of the rocket itself.

Finally, the researched topic of the Falcon 9 rocket characteristics itself was found. Although all the test and flight data are classified or at least not readily available public information, the characteristics of the actual rocket are well published. The data for the rocket was easily found in the Falcon 9 handbook that could be found for download [9,10]. Here all the available information for the rocket is published by the parent company SpaceX. This was hugely helpful as it added validity and accuracy to the mathematical model and simulation of the proposed control system for the rocket.

With the characteristics of the Falcon 9 rocket and the basic equations needed to start the modeling and exploration of a control system for the project can commence with the utmost certainty of validity and accuracy.

1.3 Project Outline

Although a prototype version of the proposed control system will not be made, the problem and simulations are still of utmost importance. An outline of the problem and solution follows below.

1. **Dynamic System Modeling:** Before a control system can be designed, the dynamics of a system must first be investigated. To do this, a mathematical model of a rocket in motion will be derived, simulated, and verified. This model will mimic the real-world system; therefore, analysis of the derived mathematical model will also pertain to its real-world counterpart. This step is the foundation of any control system.

2. **Open loop system analysis:** To begin the design of a control system the open and closed loop characteristics of a rocket in flight will be needed. This event will then be modeled in Simulink and simulated. Using this data, a control system for each of the three types of thrust control can then be implemented separately to analyze how each individual system affects rocket flight, rocket stability, and rocket maneuverability.

3. **Linearization of system:** To implement a simple controller the system must be linearized. After linearization, this new linear system should be validated and compared to the nonlinear system. This will ensure proper control design as well as system dynamics.

4. **Controller Design:** After the system has been linearized and simplified a control scheme should be implemented and tested. The system should also be checked for controllability to make sure all states, or at least the states that are in testing, can be controlled. Once verified a simple PID controller can be designed and implemented laying the foundation for future controls work to be done later with the same system.

5. **Performance & Tuning Optimization:** The final step will be to apply analysis and machine learning to tune the controller. This tuning will be benchmarked against MATLAB's performance, and hand tuning performance. This tuning process will prove or disprove whether better performance can be gained through using self-developed, specialized machine learning techniques. Versus the esoteric had tuning procedure that is usually followed when tuning common PID controllers.

Chapter 2-Nonlinear Dynamics of Rocket Airframe

Any simulation or model needs a valid mathematical model to support its validity. In this chapter the nonlinear dynamics of an airframe, specifically the Falcon 9 rocket, will be derived. These equations will be derived with respect to the following assumptions used:

- Constant mass. Although in real life the rocket will experience a center of gravity shift due to propellant (mass) being used as fuel this will not be a consideration in this project. A constant mass model will be used not a variable mass model.
- Rigid body. The rocket in this case will have non-variable inertial properties.
- The rocket has a cruciform geometry.
- Rocket is aligned among its principal axis of inertia.

These assumptions are important to note because a model is only as good as the assumptions you make. The more assumptions made the less accurate the model will be. Although these two assumptions will be made the model itself will still retain a large degree of accuracy, while also simplifying a lot of the calculations at hand.

2.1 Reference Frames

An established coordinate system and frames of reference are needed to transform forces acting in separate frames can be resolved onto the rocket body properly. Here we will define the earth centered inertial frame which is in respect of the center of the earth. This will be named the ECI frame from now on. Another reference frame that will need defining will be the Earth centered fixed frame. This can be seen clearly below. The vectors $\hat{\imath}$, $\hat{\jmath}$, and $\hat{k}$ are given in the ECI frame. While the vectors , $\hat{\imath}_e$, $\hat{\jmath}_e, and\ \hat{k}_e$ are the same vectors in the ECI frame translated into the ECF frame.

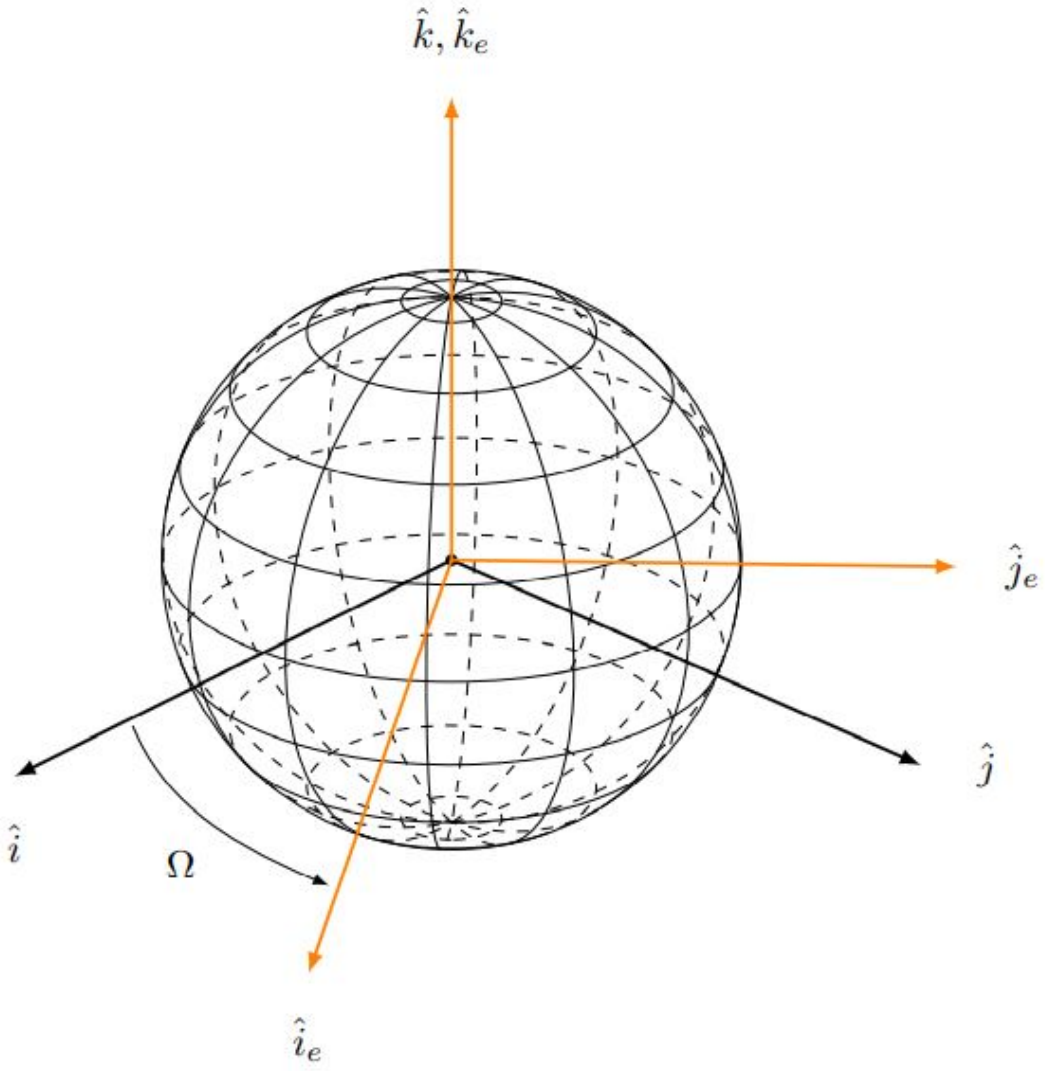


Figure 2.1: ECI to ECF reference frame through angle Ω [11]

The relationship between both reference frames can be seen in Figure 2.1 above. The ECI frame rotated about the angle (Ω) yields the ECF reference frame. To do get this transformation between reference frames we use the transformation matrix expressed as T_{ECI}^{ECF}. The matrix of transformation is given as:

$$T_{ECI}^{ECF} = \begin{bmatrix} \cos(\Omega) & \sin(\Omega) & 0 \\ -\sin(\Omega) & \cos(\Omega) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.1}$$

To describe the position of the launch vehicle or rocket anywhere on the earth's surface the Earth Geographic Reference frame is used [11]. This reference frame uses longitude and latitude to describe the position of the rocket. Using the rocket body on the launch pad as the origin for this frame the calculations are simplified. This frame is referred to as EG.[11]

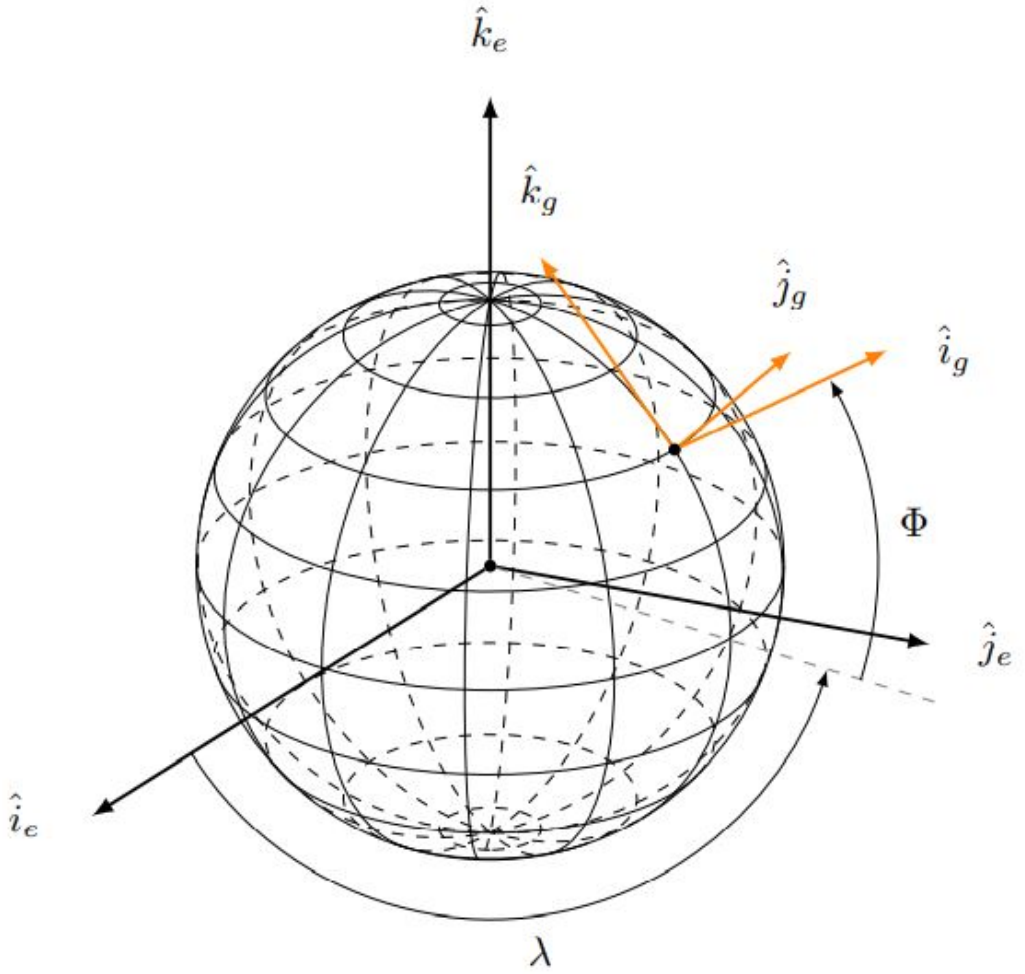


Figure 2.2: E.G Reference frame from ECF frame through angle Φ [11]

The EG reference frame is a transformation from the ECF reference frame through the angles Φ and λ. These rotations yield the vectors $\hat{\imath_g}$, $\hat{\jmath_g}, and\ \hat{k}_g$. Representing the basis vectors in the EG frame [reference bachelor thesis paper 14]. The transformation matrix is represented as T_{ECF}^{EG}. The complete transformation matrix is shown as:

$$T_{ECF}^{EG} = \begin{bmatrix} \cos(\lambda)\cos(\Phi) & \sin(\lambda)\cos(\Phi) & \sin(\Phi) \\ -\sin(\lambda) & \cos(\lambda) & 0 \\ -\cos(\lambda)\cos(\Phi) & -\sin(\lambda)\sin(\Phi) & \cos(\Phi) \end{bmatrix} \quad (2.2)$$

After now that the Earth's reference frames have been satisfied, the rocket body frame needs to be defined. Traditionally in the aerospace industry the body frame basis vectors are defined as the $\widehat{\imath_b}$, $\widehat{j_b}$, and $\widehat{k_b}$ [11]. These are rotated about the bodies yaw pitch and roll angles $\Phi, \Theta, and\ \Psi$. Using this notation, we arrive at the body basis vectors in the body reference frame transformed from the EG frame. An illustration of the body frame can be found below.

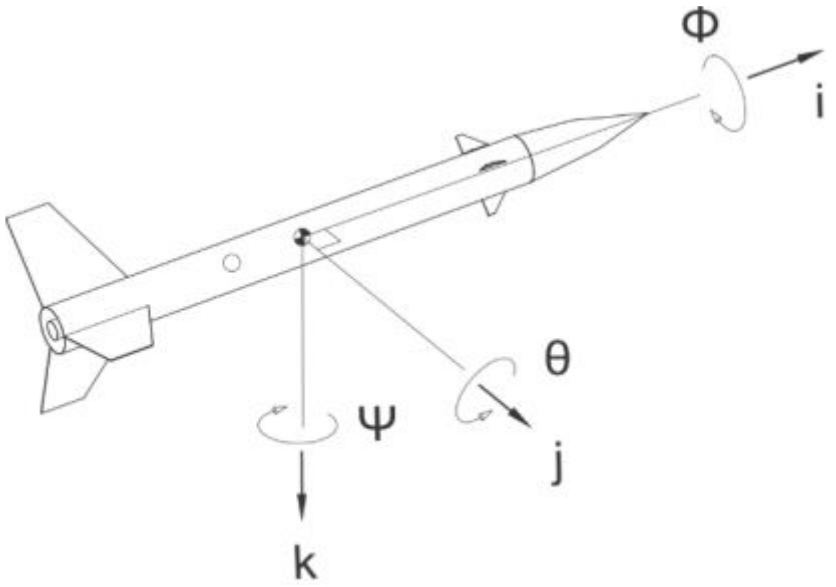


Figure 2.3: Rocket body frame [12]

With these angles of rotation, the transformation matrix is expressed as T_{EG}^{B}. The complete transformation matrix is known as:

$$T_{EG}^{B} = \begin{bmatrix} cos\ (\Theta)cos\ (\Psi) & cos\ (\Theta)sin\ (\Psi) & -sin\ (\Theta) \\ sin(\Theta)\, cos(\Psi)\, sin(\Phi) - cos(\Phi)\, sin\ (\Psi) & sin(\Phi)\, sin(\Psi)\, sin(\Theta) + cos(\Phi)\, cos\ (\Psi)) & cos\ (\Theta)sin\ (\Theta) \\ sin(\Theta)\, cos(\Psi)\, cos(\Phi) + sin(\Phi)\, sin\ (\Psi) & sin(\Theta)\, sin(\Psi)\, cos(\Phi) - cos(\Psi)\, sin\ (\Theta) & cos\ (\Theta)sin\ (\Theta) \end{bmatrix} \quad (2.3)$$

These reference frames define the rocket and its given position at any given time with respect to the earth. The forces experienced by the rocket are all resolved into the body frame then used to calculate the effects on the body. It is pertinent to note that one can transform between these reference frames at any given time given these transformation matrices. This concludes the discussion on reference frames next will be the kinematics of the rocket itself.

2.2 Rotational Kinematics

The rocket body undergoes rotation throughout the course of its flight. To resolve this rotational motion the angular rate is needed. This can be easily expressed as a vector ($\overrightarrow{\omega}$). This vector is composed of the roll, pitch, and yaw rates of the airframe expressed in the body frame. Shown as:

$$\overrightarrow{\omega} = \begin{bmatrix} p \\ q \\ r \end{bmatrix} \tag{2.4}$$

$$= \begin{bmatrix} 1 & 0 & -sin\,(\Theta) \\ 0 & cos\,(\Phi) & sin\,(\Phi)cos\,(\Theta) \\ 0 & -sin\,(\Phi) & cos(\Phi)cos\,(\Theta) \end{bmatrix} \begin{pmatrix} \dot{\Phi} \\ \dot{\theta} \\ \dot{\psi} \end{pmatrix} \tag{2.5}$$

Although the above mathematical statement is true the reverse is usually seen more often in literature. Where the angular rates are expressed in respect to the roll, pitch, and yaw. This is expressed below to maintain coherence with literature results.

$$\begin{pmatrix} \dot{\Phi} \\ \dot{\Theta} \\ \dot{\psi} \end{pmatrix} = \frac{1}{cos\Theta} \begin{bmatrix} \cos\,\theta & \sin\,\Phi sin\Theta & cos\Phi sin\Theta \\ 0 & \cos\Phi cos\Theta & -sin\Phi\cos\Theta \\ 0 & \sin\,\Phi & cos\Phi \end{bmatrix} \begin{pmatrix} p \\ q \\ r \end{pmatrix} \tag{2.6}$$

This would be the usual implemented result as seen in literature and the model simulation [11].

The above equations are using the Euler angles of rotation. Although this is what is used in the simulation of rocket flight within this project, there are some major drawbacks. At a given point there is a phenomenon named gimbal lock. This is a point of singularity which cannot be resolved, to avoid this quaternion can be used. However, quaternions were not used in this phase of the project in future worth quaternions will be implemented. The next topic to tackle will be the equations of motion in six degrees of freedom for this rocket model.

2.3 Equations of Motion

The equations of motion were derived in accordance with references [11-14] as guides.

Using Newton's second law seen below:

$$\vec{a} = \frac{1}{m}\vec{F} \tag{2.7}$$

The position of the air frame is given by the previously discussed reference frames as:

$$\vec{r} = x\hat{\imath} + y\hat{\jmath} + z\hat{k} \qquad (2.8)$$

Taking the second derivative of the position yields the equation:

$$\vec{a} = \ddot{x}\hat{\imath} + \ddot{y}\hat{\jmath} + \ddot{z}\hat{k} \qquad (2.9)$$

This satisfies the left side of Newtons second law. Now the right side must be satisfied. All forces acting upon the rocket body must be considered. These forces are broken into three subcomponents. The force from the atmosphere, force of gravity, and the propulsive force. As gravity acts within the ECI frame the other two forces must be transformed from the body frame to the Earth centered inertial frame. Once this is done simply summing the forces and substituting in for the F in equation 2.3.1 is enough to satisfy the translational kinematic equations. This is shown below:

$$\vec{a} = \frac{1}{m}[T_B^{ECI}\left(\vec{F}_{atmospheric} + \vec{F}_{Propulsive}\right) + \vec{F}_g] \qquad (2.10)$$

Where:

$$\vec{a} = \dot{\vec{v}} = \ddot{\vec{r}} \qquad (2.11)$$

The next three sets of equations that need to be resolved are the Rotational motion of the aircraft. This will be covered in the next section.

2.4 Rotational Equations of Motion

Just as Newton's Second Law was used for the derivation of the first the sets of equations of motion, Euler's Second Law. These three new degrees of freedom will be derived in the EG frame to keep the yaw, pitch, and roll angles well defined. This is different from the first three degrees of freedom for translation which were derived with respect to the ECI frame.

Euler's Second Law states that the sum of external moments or torques acting on a body is equal to the angular momentum rate. This can be shown as:

$$\vec{M} = \dot{\vec{L}} \qquad (2.12)$$

The rocket body is assumed to be aligned to its principal axis meaning the simplest case. This leads to:

$$\dot{\vec{L}} = \hat{J} \cdot \vec{\omega} \qquad (2.13)$$

With $\hat{J}$ representing the moment of inertia of the rocket body where in a symmetric body aligned with its principal axis is simply:

$$\hat{J} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{Vmatrix} J_{11} \\ J_{22} \\ J_{33} \end{Vmatrix} \qquad (2.14)$$

Where $\vec{\omega}$, represents the total angular velocity of the body as well. This leads to the rotational motion of equations assuming a rigid body can be written as:

$$\vec{M} = \hat{J} \cdot \vec{\omega} + \vec{\omega} \times [\vec{J} \cdot \vec{\omega}] \quad (2.15)$$

Just as the forces acting on the rocket were broken into subcomponents the same can be done for the moments. As there are several factors contributing to the external forces on a rocket these moments will be broken into two parts. $\vec{M}$ will be broken into $\vec{M}_{propulsive}, and\ \vec{M}_{atmospheric}$. Using this equation 2.4.4 can be solved for the angular velocity $\vec{\omega}$.

$$\vec{\omega} = \hat{J}^{-1}[\vec{M}_{atmospheric} + \vec{M}_{propulsive} - \vec{\omega} \times [\hat{J} \cdot \vec{\omega}]] \quad (2.16)$$

Combining the above equation with the translational equations of motion yield six equations which when solved give the position and angular rates of the airframe. These equations are highly nonlinear. The next step is to simulate and verify these equations implemented into the model of Simulink are correct and in line with benchmark data.

2.5 Simulation of Nonlinear Equations of Motion

Next a couple of simulations were running to verify that the implemented mathematical equations were done properly and accurately. Benchmarked data was of immense importance to this step, references [11,15] had clear and accurate graphs of a 6dof rocket simulation. These were used as a benchmark to verify the given mathematical relations and equations. The downside to this approach was the input data was not specified therefore complete matching of the benchmark data would be impossible. The benchmark data in this case was meant to show the behavior of a rocket system and how it should vary overtime. Although the input data was not explicitly stated within the paper, the rocket was said to be cruciform and have many similar characteristics of the Falcon 9 rocket. Below is the benchmark data taken from reference [15].

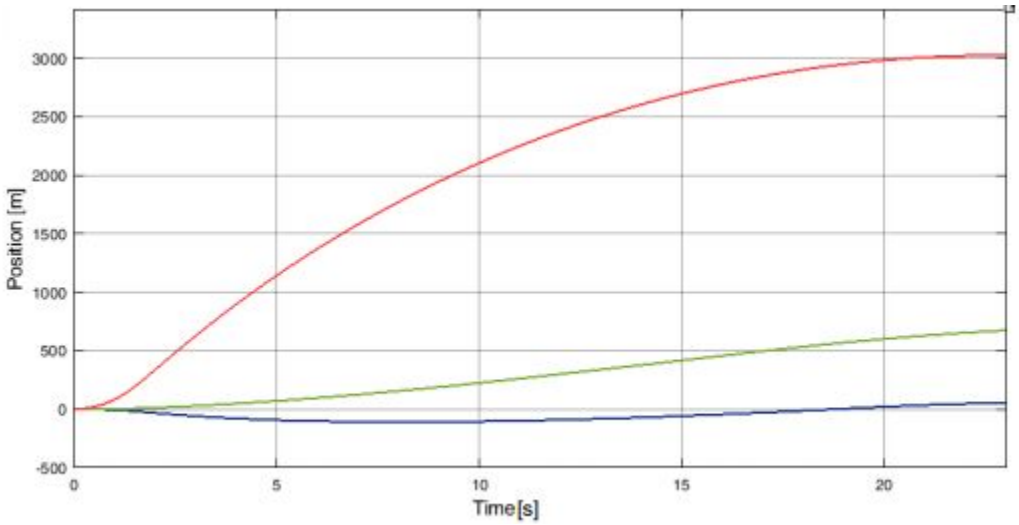


Figure 2.4: Position data of rocket reference data [15]

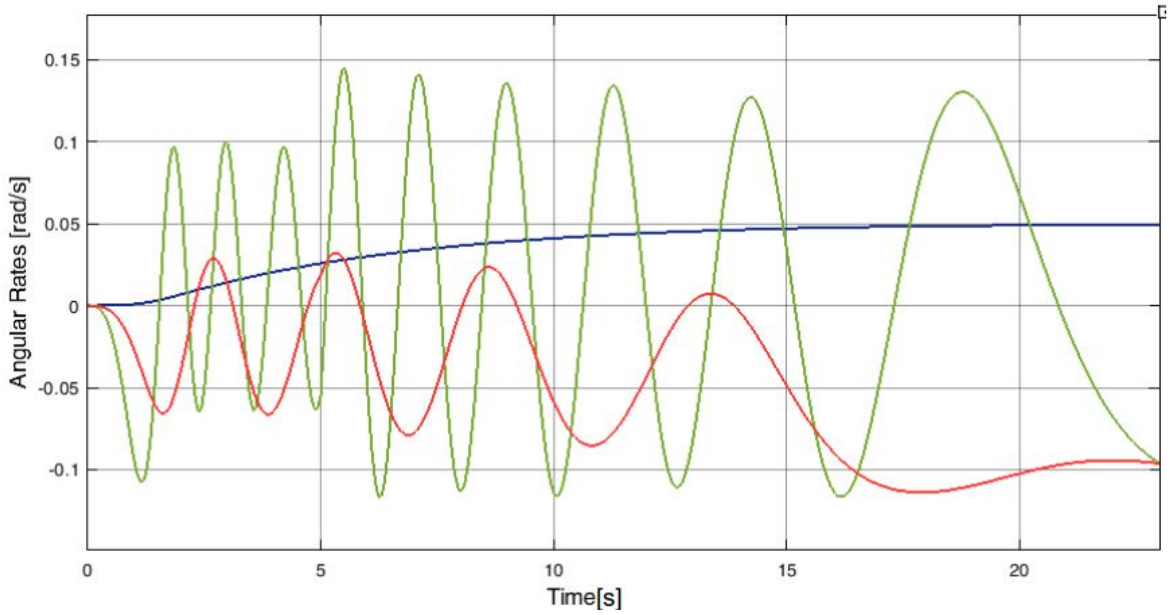


Figure 2.5: Angular rates of rocket from benchmarked data [15]

In figure 2.5.1 the main variable of interest is the red line representing the height in meters (feet). This simulation initial conditions were not known however the paper did state the rocket was at rest on the ground leading to zero height in the z-direction or height. This should be reproduceable within the simulation at hand. In figure 2.5.2 the main interest is to show the angular rates are divergent leading to an unstable system. This too should also be reproduced in the results obtained from the simulation and model. Before presenting the results, the Simulink model can be shown below. Using reference [11] as a guide this Simulink model was made to simulate the same rocket in reference [11] as simulated in this paper.

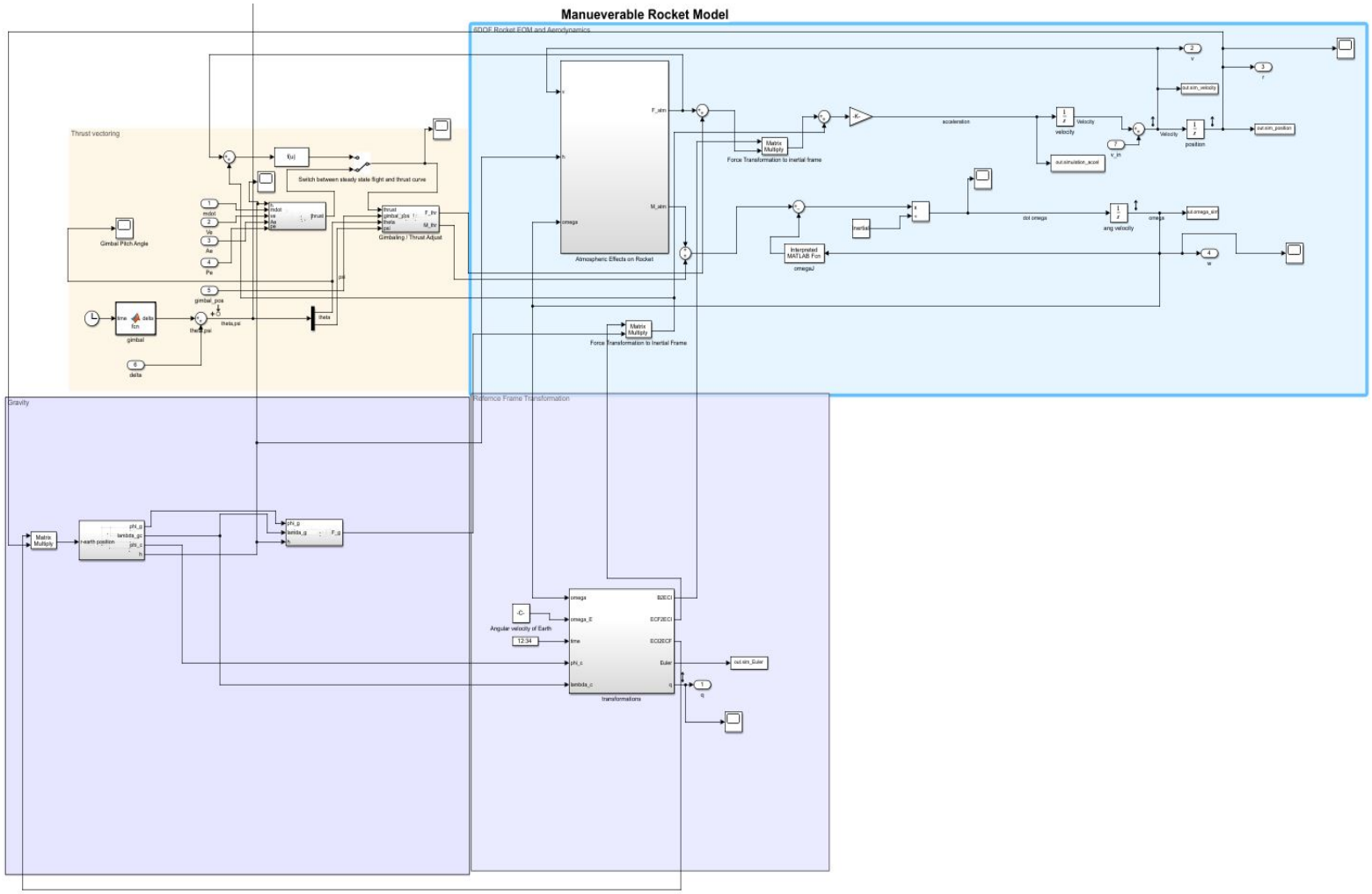


Figure 2.6: Maneuverable rocket (6-Dof)

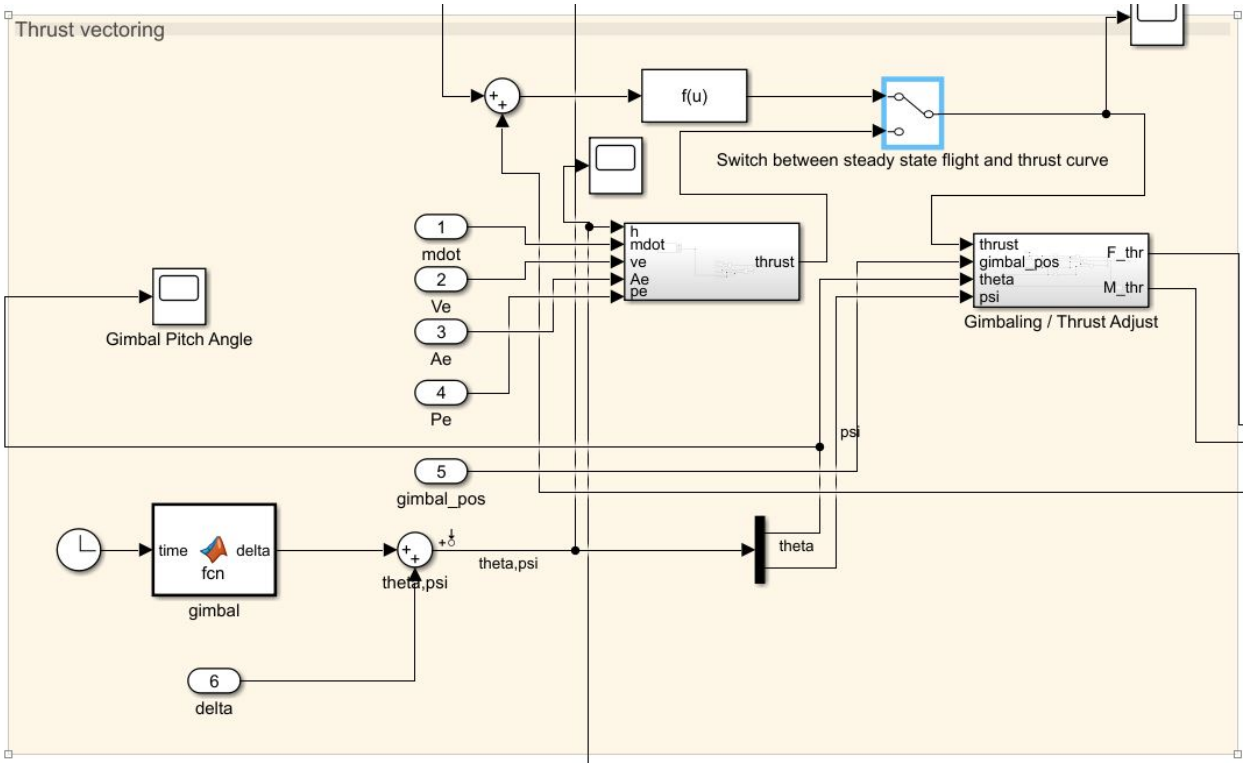


Figure 2.7: Thrust vectoring control

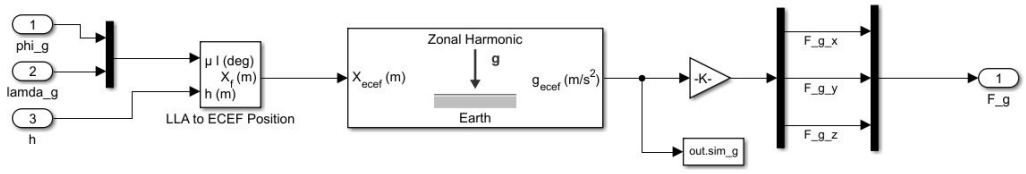


Figure 2.8: Gravity force subsystem

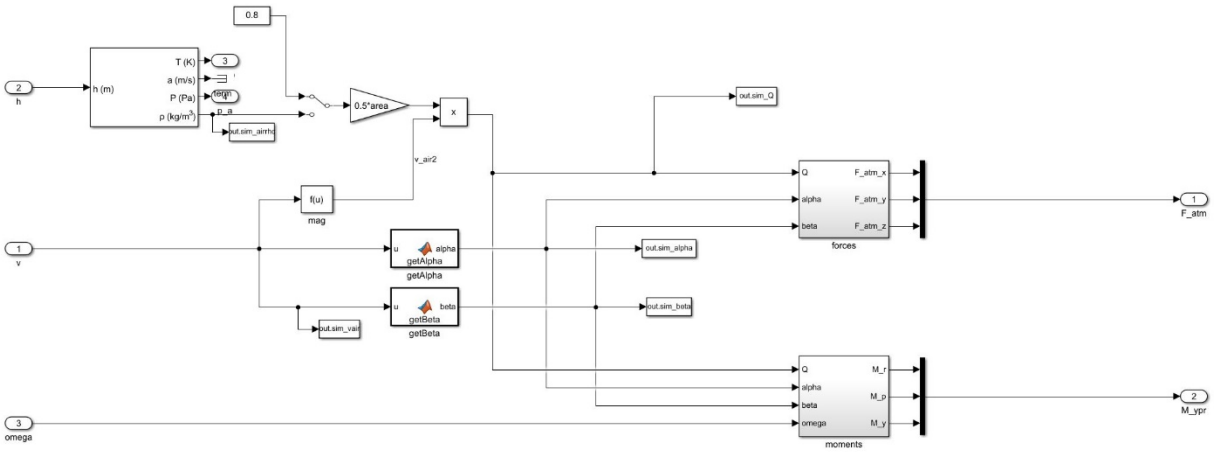


Figure 2.9: Atmospheric forces subsystem

The above systems make up the entirety of the six degree of freedom system needing to be simulated. With this system and a similar simulation time the results came to be:

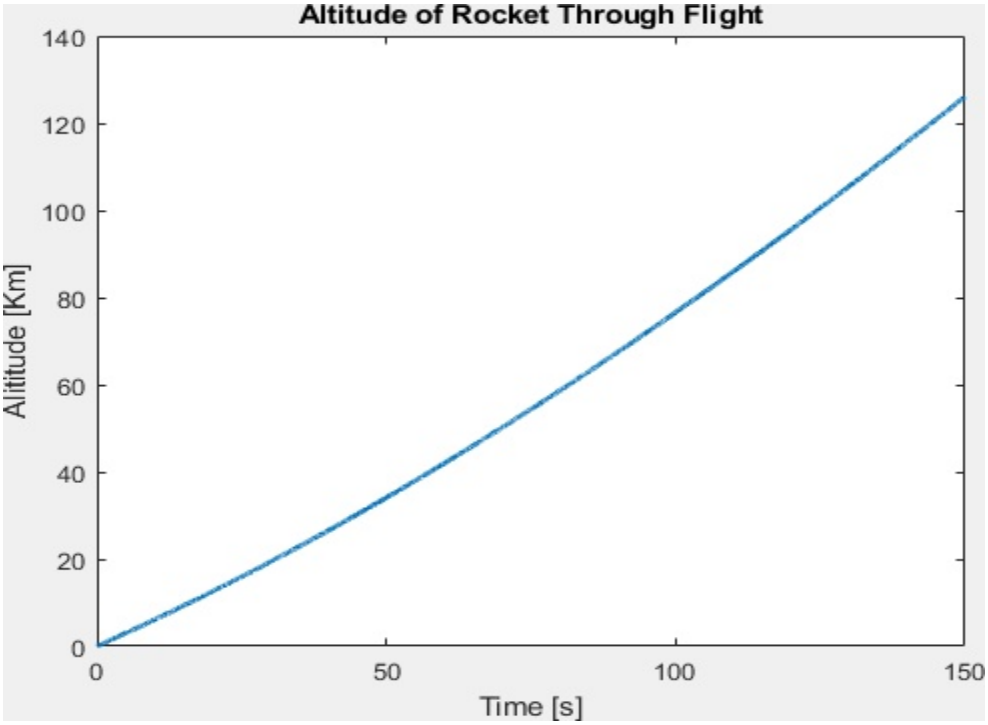


Figure 2.10: Altitude of rocket over time of flight

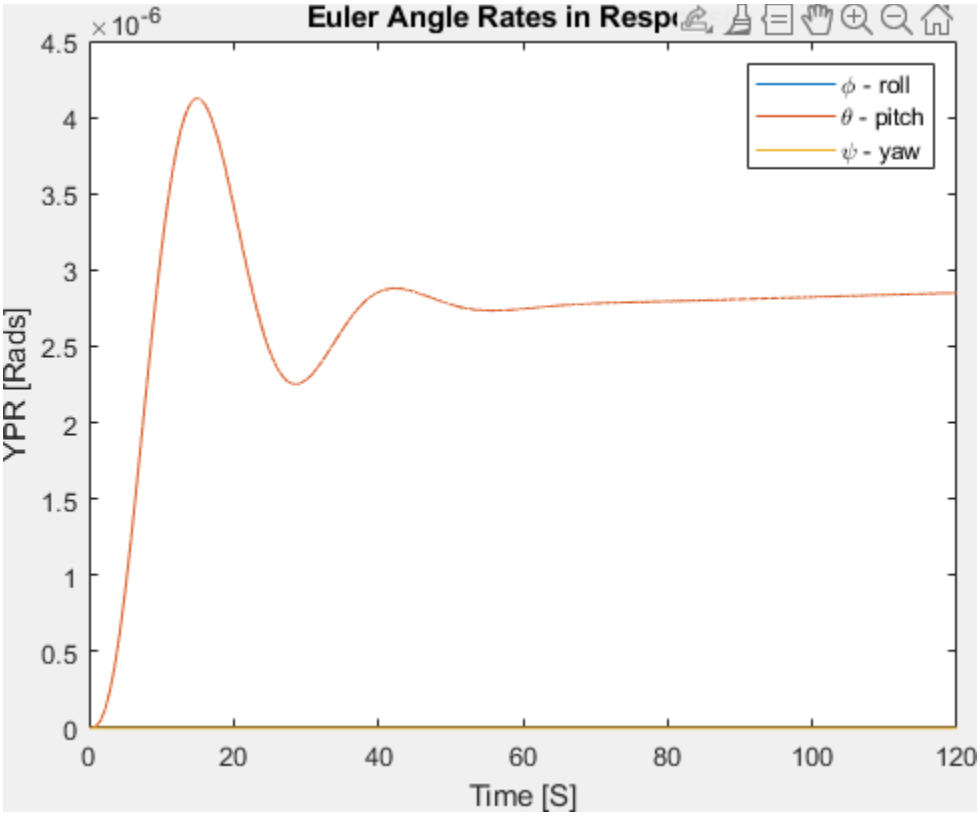


Figure 2.11: Euler angular rates

Comparing the benchmark figure 2.5.1 “Position Data of Rocket Simulation,” with that of figure 2.5.7 Altitude of Rocket. The behavior of the two graphs is in line. Although it takes longer for the Falcon 9 to start to reach its peak at around one minute and thirty seconds versus the benchmark data of 15 seconds. This can be attributed to different propulsion systems, mass, or other outside factors. The response and altitude of the Falcon 9 rocket is in line with expectations. This verifies the translational motion of the rocket.

Next in comparing the figure 2.5.2 “Angular Rates of Rocket” benchmark data to figure 2.5.8 “Euler Angle Rates.” Here the two graphs are significantly different, however this can be due to initial conditions and the response to a given stimulus. Here the Falcon 9 rocket is not reacting to any stimulus to the rocket. This is just a simple test to determine whether the Falcon 9 is stable or unstable. At first the angular rate seeks stability by oscillating, however as time continues to pass there is exponential growth of the angular rate. This leads to the airframe being highly unstable. This is in line with what is expected of the rocket airframe. All rockets have an onboard computer-controlled control system, this means that without an implemented control system the general response of the airframe is unstable. The Falcon 9 simulation done here is in accordance with this statement. Without a control system, the Falcon 9 is an unstable system. For further confirmation of this point a control input was used to excite the Falcon 9 system. The goal being to confirm whether the system is stable, unstable, oscillatory, or exponentially unstable. The thrust vectoring input can be seen below. This is a deflection of .00054 radians

from time 20 seconds to time 30 seconds and a negative .00054 radians from 30 seconds to 40 seconds.

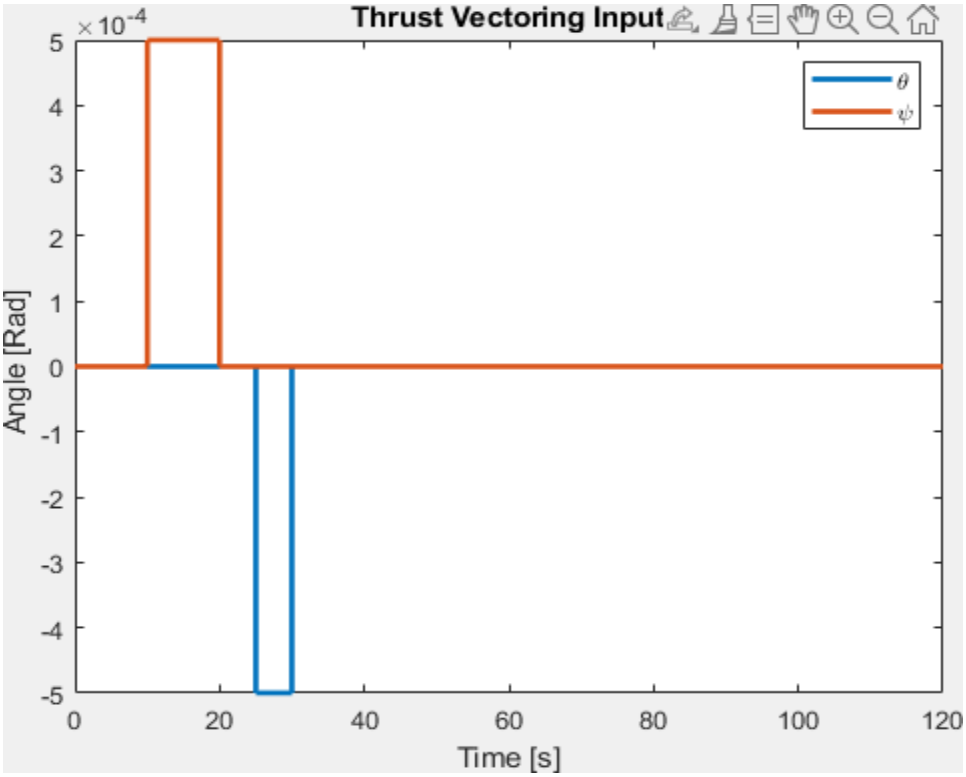


Figure 2.12: Thrust vectoring input for Falcon 9 rocket

With the control input the responses of the Falcon 9 rocket are as follows:

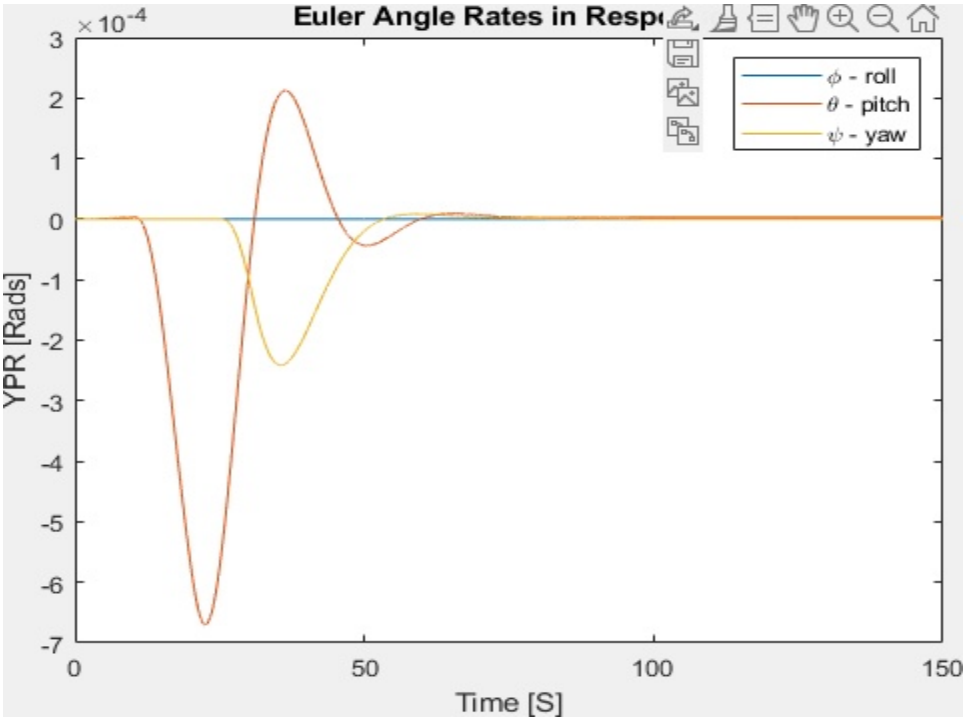


Figure 2.13: Euler angle rates response to thrust vector input

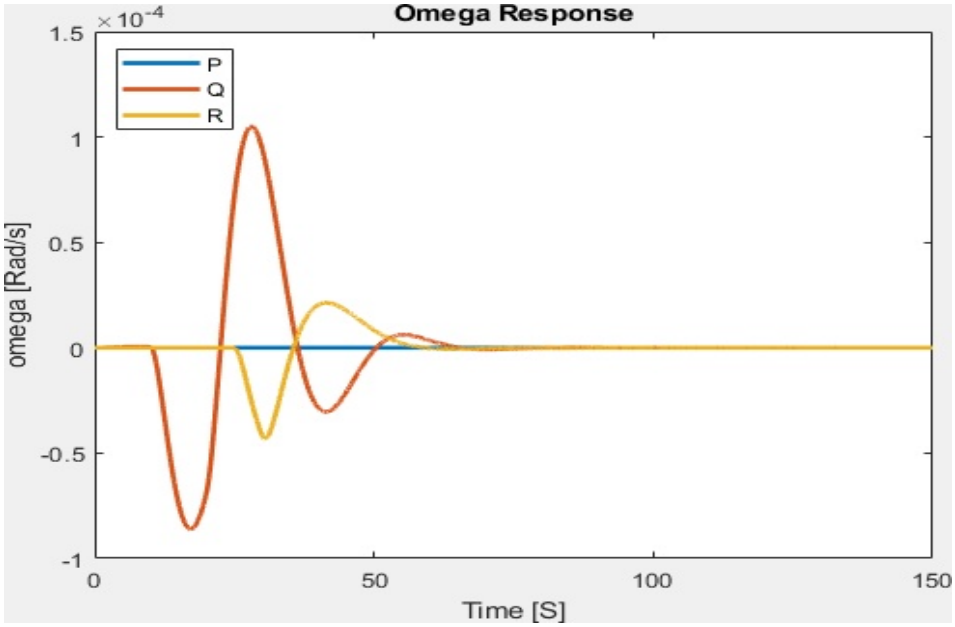


Figure 2.14: Angular velocity response to thrust vector input

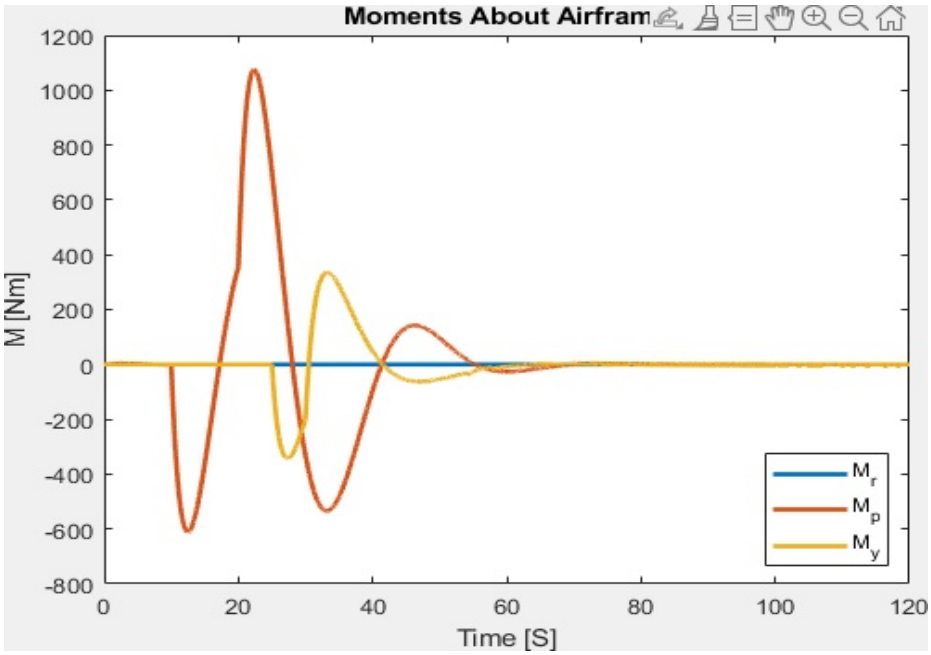


Figure 2.15: Moments of airframe in response to thrust vectoring input

From the above figures the initial analysis stating the Falcon 9 airframe is unstable holds when a control input of thrust vectoring is introduced. Above in figure 2.5.10 all angular rates are exponentially divergent as time increases; this leads to exponentially unstable behavior. Next in figure 2.5.11 the angular velocities grow in magnitude for however long the input is introduced into the system. This is another divergent behavior of the system that leads to instability, a stable system would seek back to zero after an input is introduced. Here the system grows in response for as long as the response is present, this is another unstable behavior. Lastly, analyzing the moment forced produced by the thrust vectoring also indicates the structure itself would not be able to physically maintain structural integrity if a control input of thrust vectoring is introduced due to the growth of the magnitude of the moments produced during thrust vectoring. The longer thrust vectoring is introduced the greater the moment generated, eventually this growth would reach the theoretical limiting load of the system, leading to catastrophic failure. This is the final indication that an uncontrolled Falcon 9 is indeed unstable.

Now that the Falcon 9 system has been verified as unstable, simplification, and linearization about an operating point is next. This will be explored in the next chapter.

Chapter 3-Linearization and Simulation

3.1 Linearization

As the Falcon 9 rocket has been identified as unstable without a control system simplifying and linearizing the system to evaluate for controllability is next. Linearizing the system will decouple the lateral and longitudinal planes of the rocket [15]. Linearizing will also "…allow for the implementation of simple and systemic linear control design techniques" [13]. This is especially useful when first designing a control system for any vehicle, in this case the Falcon 9 rocket. The methodology of linearization used here can be seen in references [12, 13,14]. This is also known as the perturbation method in the aerospace industry. Along with the perturbation method Simulink was used to yield the state space linear formulation for the, A, B, C, D matrices. Next the derivation of the linearized equations will be walked through as derived from the perturbation method.

The linearization point at which the perturbation will be done will be in steady state flight. This is at a velocity of 1,200 m/s which was picked as the average speed of a rocket within the atmosphere. During steady state flight the following assumptions can be made which have been summarized in a table below.

Table 3.1: Zero and non-zero quantities at steady flight conditions

Zero Quantities	Non-Zero Quantities
Y_0	X_0=Operating Longitudinal Position in Meters
L_0	$\dot{x}_0$ = 1,200 m/s (3,937 ft/s)
M_0	$Z_0 = Operating\ Altitude\ in\ Meters\ (Feet)$
N_0	
v_0	
w_0	

From the above information we take and add a small disturbance of $\Delta(*)$, which is a slight perturbation in each of the axis forces, moment, velocity, and rates with respect to time. This will look like the table of information below, with guidance from reference [12-14].

Table 3.2: Perturbation variables and effects

Variable	Perturbed Variable
u	$u(t) = u_0 + \Delta\mathrm{u(t)}$
v	$v(t) = v_0 + \Delta\mathrm{v(t)}$
w	$w(t) = w_0 + \Delta\mathrm{w(t)}$
p	$p(t) = p_0 + \Delta\mathrm{p(t)}$
q	$q(t) = q_0 + \Delta\mathrm{q(t)}$
r	$r(t) = r_0 + \Delta\mathrm{r(t)}$
X	$X(t) = X_0 + \Delta\mathrm{X(t)}$
Y	$Y(t) = Y_0 + \Delta\mathrm{Y(t)}$
Z	$Z(t) = Z_0 + \Delta\mathrm{Z(t)}$
L	$L(t) = L_0 + \Delta\mathrm{L(t)}$
M	$M(t) = M_0 + \Delta\mathrm{M(t)}$
N	$N(t) = N_0 + \Delta\mathrm{N(t)}$

With these new perturbed variables, the six degrees of freedom motions of equations are substituted with the new perturbed terms. Using small angle approximations and eliminating small terms such as two derivatives multiplied by each other. Doing so for such a huge system such as a Falcon 9 rocket is not handedly done by analytical methods. This is where the computational tools take over and derive the linearized model in Simulink. Using reference [17] the model inputs are designated as the thrust vectoring inputs both $\delta_\theta, and\ \delta_\psi$. While the model outputs are All of the above variables included in the table, and in the future the quaternions of the system. This leads to measuring in total 13 states of the system, with 2 inputs to the system. As one could see, doing this by hand would be an analytical nightmare.

With the help of the model linearizer in MATLAB the following sate space matrices are generated and can be tested against the nonlinear system to verify accuracy.

Usually, the expected C matrix would be an identity matrix of the number of states being measured. However, due to the model linearizer in MATLAB there is an option to order states. The states were ordered in a way not normally seen, this was done for ease signal tracking and state recognition. Another important note, the model was linearized with alpha as an output state as well. This leads to one state being dependent on another as alpha can be derived from the position states. This was done again for ease of state space variable measurement and tracking. The consequence of this is that the state space system is not fully controllable when using all 14 states, due to one state not being linearly independent which is alpha in this case. To remedy this issue alpha must be removed as a state then the controllability and observability must be observed for the system. The complete linear state space formulation for this model will be shown in the next sub section along with an analysis and plots.

3.2 State Space Representation

After linearizing the six degree of freedom system into a linear state space system using MATLAB, analysis of the linearized system can be done. The state space matrices for the linear system are shown below:

$$A = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 3.4305e-04 & 1.0459e-9 & 2.9176e-11 & -.0054 & 0 & 0 & 0 & 0 \\ -9.1995e-08 & 2.9837e-07 & 0 & 0 & 0 & 0 & 0 & 0 \\ 2.9175e-11 & 0 & 2.9336e-07 & 0 & 0 & -.6982 & 23.4346 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & .500 \\ 0 & 0 & 0 & 0 & 0 & -.0040 & 0 & -.3751 \end{bmatrix} \quad (3.1)$$

$$B = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ .0067 & 0 \\ 13.3420 & 0 \\ 0 & 13.3420 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \quad (3.2)$$

The C and D matrix is a 14x8 identity matrix and a 14x2 zero matrix. These will not be displayed here. The state variables, inputs, and outputs are designated as:

$$x = [\dot{\omega} \quad x \quad y \quad z \quad \dot{x} \quad \dot{y} \quad \dot{z} \quad \alpha] \quad (3.3)$$

$$u = [\delta_\Theta \quad \delta_\Psi] \quad (3.4)$$

$$y = x \quad (3.5)$$

It should be noted as stated earlier this is a unique state space formulation designed for the ease of measurement and handling for this specific case. This formulation and state space variables are different from what may be seen in literature and other sources. The complete state space system is to be represented as the following system:

$$\dot{x} = Ax + Bu \quad (3.6)$$

$$y = Cx + Du \quad (3.7)$$

With this complete state space formulation and linearization of the nonlinear rocket model validation, confirmation, and analysis of the system is needed now. This will be shown in the next subsection to follow.

3.3 State Space Analysis of Linear System

3.3.1 Controllability

Now that a linear state space model has been obtained the controllability and observability must be checked. This will either confirm or deny that the system can be controlled. Ignoring state alpha because it is dependent upon the position states of the system. After which MATLAB is used to obtain the controllability matrix for the system. From there it can be seen the controllability matrix is full rank, meaning the system is fully controllable. The same can be done for observability of the system leading to the system being fully observable as well.

These are expected results as most rockets, and airframes of the modern era employ some type of control system. If these states were not controllable, then that would present a problem with modern aerospace practices.

3.3.2 Stability of State Space System

From earlier analysis of the nonlinear system, it was seen that the system was not stable, having an exponentially divergent response after a disturbance was introduced. To confirm this the poles and zeros of the system must be obtained. These values are simply the eigenvalues of the state space A matrix. Computing the eigenvalues of a 14x8 matrix analytically is very tedious and should not be attempted. Since the model is already present in MATLAB the computational approach was used. The poles of the system as well as a poles-zero map was generated as seen below.

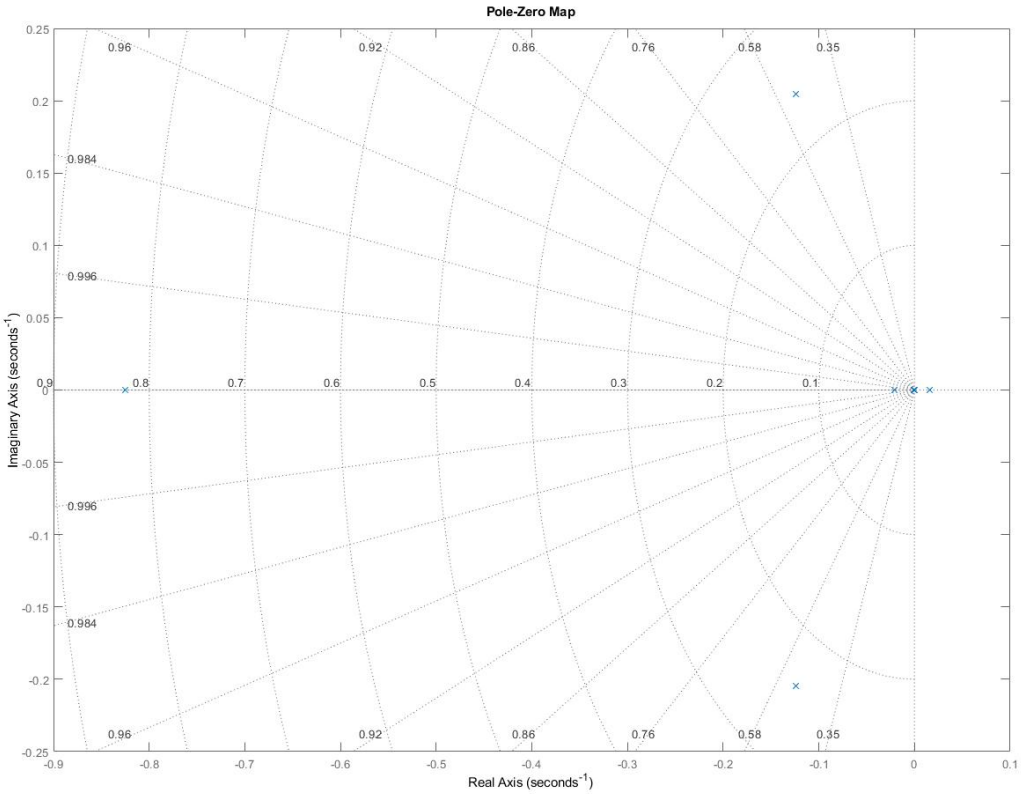


Figure 3.1: Poles-Zero Map of Linear System

```
poles =

  -0.8253 + 0.0000i
  -0.1240 + 0.2045i
  -0.1240 - 0.2045i
  -0.0214 + 0.0000i
   0.0160 + 0.0000i
   0.0005 + 0.0000i
  -0.0005 + 0.0000i
  -0.0000 + 0.0000i
```

Figure 3.2 Poles values of linear system

Although the system has many poles that are negative there are two that are slightly in the positive, this immediately leads to the system being characterized as unstable. The system is fully controllable as well as unstable, a controller is needed for this system. Before moving into controller design. Verification of the linear model is needed which will be done in the next section.

3.3.3 Linear Versus Non-Linear Model Response

This system has been identified as needing a controller. With a linearized model generated, and stability analysis performed controller design can begin immediately. Before this validation of the linear model compared to the non-linear model should be done. This is done by using reference [11] as benchmark data to be reproduced here. The benchmark data needing to be reproduced is shown below from reference [11].

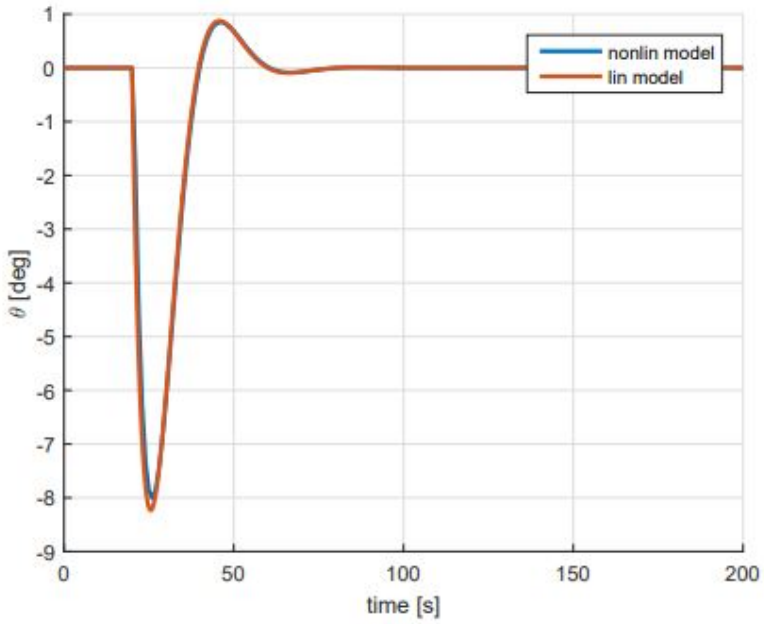


Figure 3.3: Benchmark data of pitch angle response over time [11]

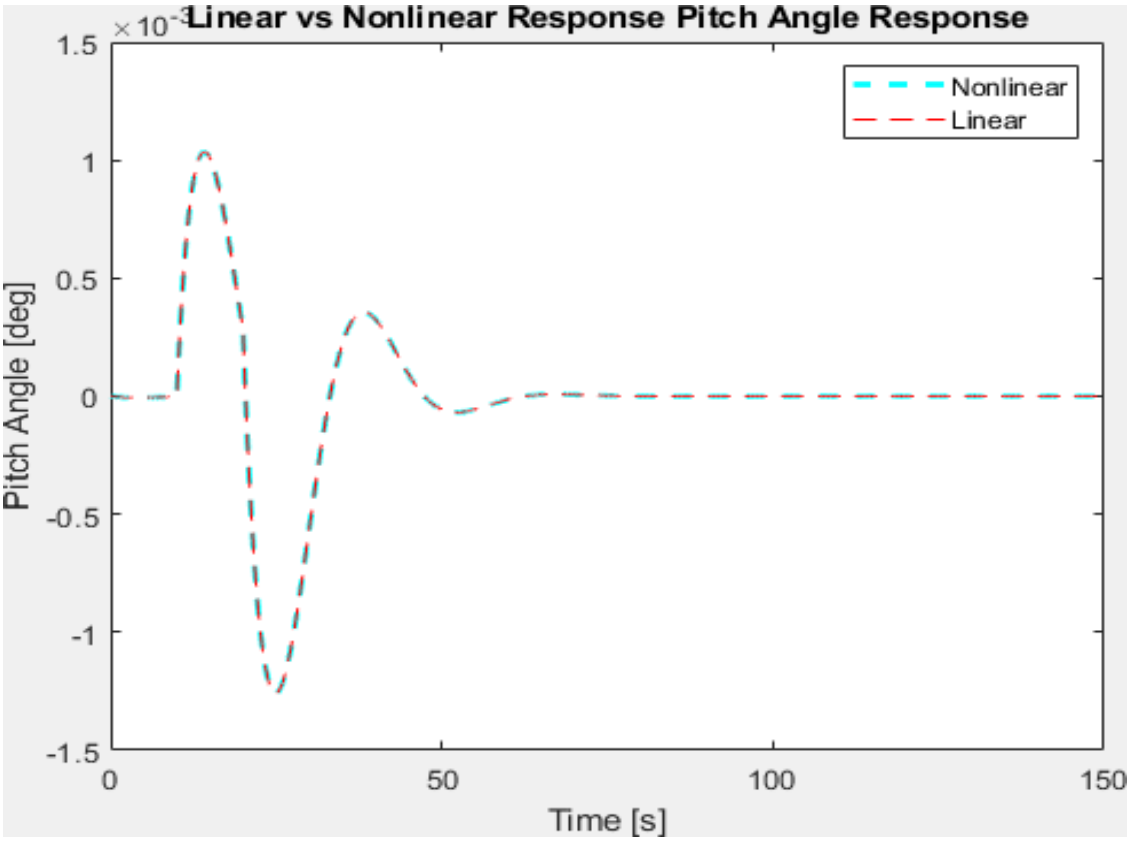


Figure 3.4: Linear vs non-linear pitch angle response over time

The two sets of data mimic each other via response and peak amplitudes. Although there is no oscillation after the peak amplitude is achieved in this model. This can be attributed to different linear model linearization parameters, and operating points. As well as using state ordering as well. Although there seems to be a slight error of .025 degrees difference between the linear and non-linear response this is only a percentage difference error of approximately .5%. This linear model accurately reproduces the non-linear dynamics of the system analyzed. Next is designing and implementing a simple PID controller. This will build the basis of this system's control law and allow further advanced control design in the future.

Chapter 4-Basic PID Controller Design

4.1 PID Controller Design

The first proposed controller to design was a simple PID controller. This controller is most used in different industries and can serve as a basis for more advanced controller design in the future.

Designing a PID controller first began in MATLAB by introducing state feedback, and the PID controller block. Once introduced, tuning the gains of the PID controller was next. Reference [18] was used as a guide on how to tune the PID controller as well as what each part of the controller is best at. From reference [18]. The general approach to tuning a PID controller is as followed from the excerpt below:

- Obtain an open-loop response and determine what needs to be improved.
- Add proportional control to improve the rise time.
- Add a derivative control to reduce the overshoot.
- Add an integral control to reduce the steady state error.
- Adjust each of the gains $K_P, K_i,$ and K_D until you obtain a desired overall response.

Using this approach tuning the PID controller to a superior performance was obtained. The closed loop system used in designing the PID controller can be seen below as a state space representation.

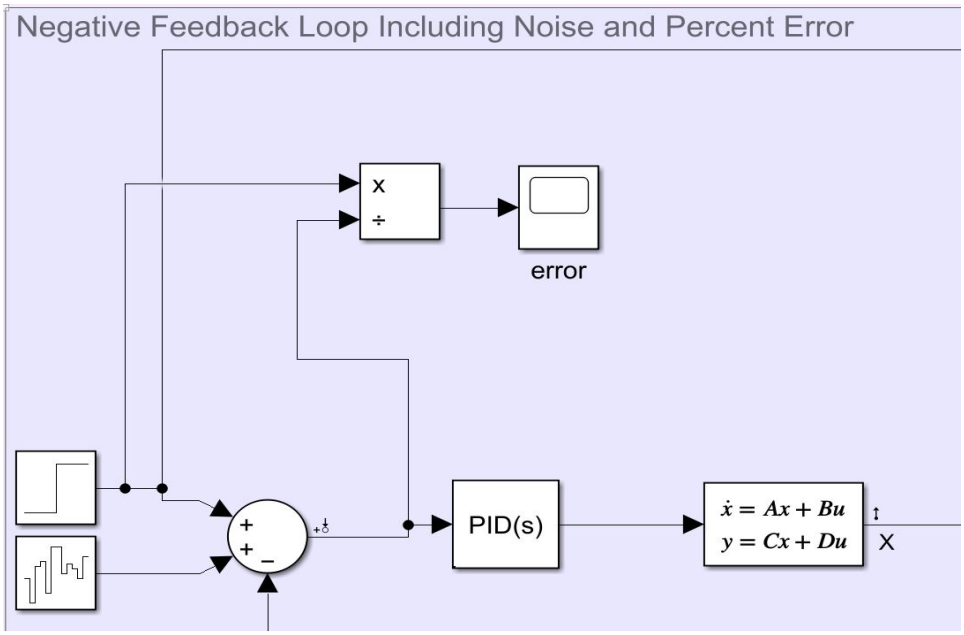


Figure 4.1: Closed loop PID controller system

Using the system pictured above the controller was tuned. This would result in controller inputs of the gimbal angles $\delta_\theta, and\ \delta_\psi$. These two inputs which control the pitch angle and heading of the rocket would then in turn direct the propulsive forces of the rocket in the desired direction thus "maneuvering" in midair. This would accomplish a rudimentary maneuverability model that was set out to be achieved. The inputs were replaced with a general step function to model the response of the system. Later after a satisfactory response was obtained from the just a step function, noise was introduced into the signal. This was done to test the robustness of the simple PID controller.

The gains of the PID controller were set using the methods stated above, which were simply trial and error. In the future when designing a more advanced controller this technique would not suffice, however that is one of the strengths of the simple PID controller. This type of controller is simple, easy to use, and effective. The control settings of the PID block can be seen below.

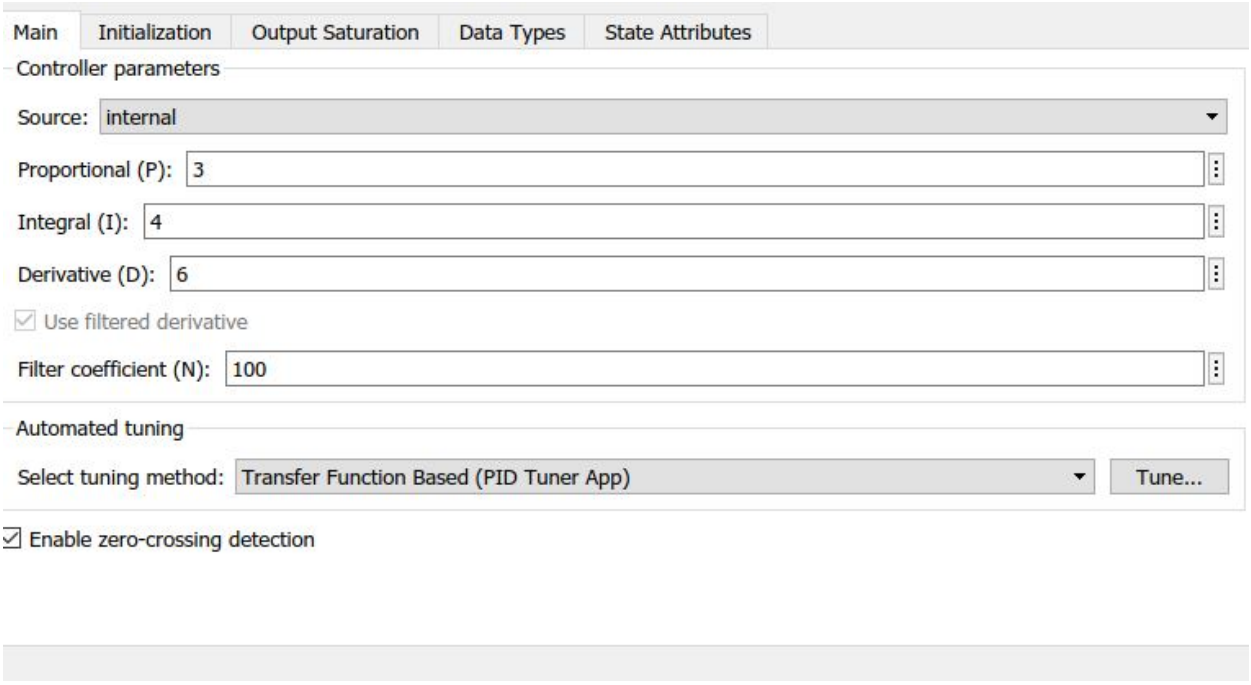


Figure 4.2: PID controller gains setting

Through trial and error these settings give the best performance, as well as error reduction. This can be seen in the next section when discussing stability.

4.2 PID Controller Performance

Although the PID controller is simple the performance is undeniable. The PID controller quickly matches the step input functions with a minimum overshoot value [17-19]. The controller and negative feedback loop as well keep the percentage error to a minimum as well. This can be seen in the following two response plots below:

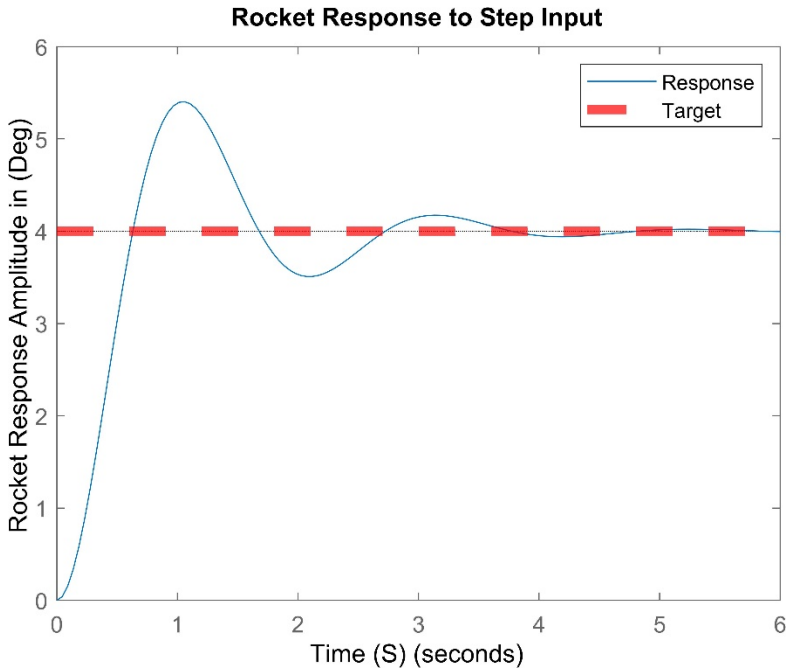


Figure 4.3: Desired vs achieved response of the system, α

Shown here is the step response or the desired heading the system needs to achieve. This is represented as a step function as the rocket system would need to match this gimbal angle and apply thrust to change its heading. The yellow line is the achieved gimbal angle. The rise is very quick with minimal amount of overshoot that is quickly reduced to the desired value. Although not very visible at first the error is also within acceptable range as well. To check this the percentage error was plotted. This can be seen below.

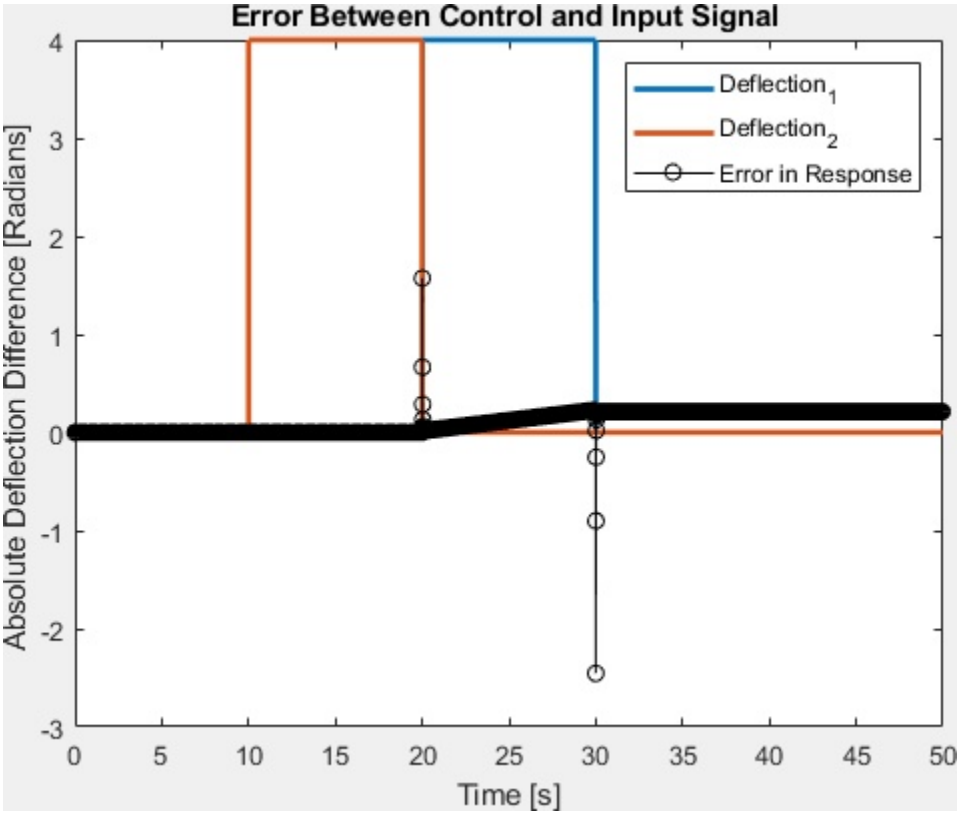


Figure 4.4: Absolute error between thrust vector input and desired

The percentage error spikes correlate to exactly when the step response was introduced. The asymptotic spike is when the step response was introduced, analyzing this in comparison to the response plot above. It can be seen this is where the gimbal angle was trying to correct for the desired alpha. In doing so the applied gimbal angle overshot the desired angle needed, but then quickly declined to match the desired results. The absolute error was only about 3.5% this much is acceptable.

Next, the robustness of the controller was tested by introducing some noise to the system [20-22]. The response of noise present in the system can be seen below.

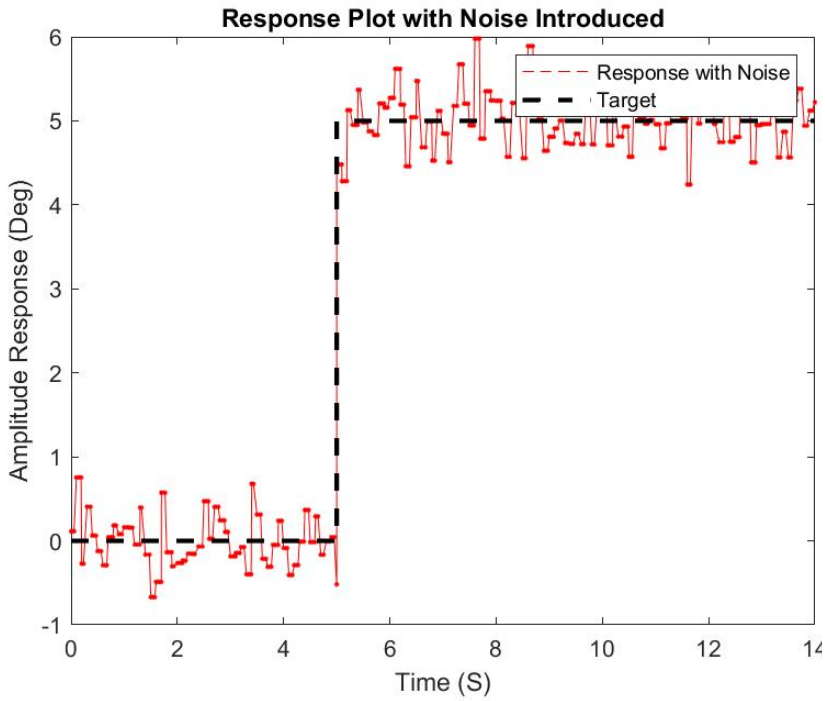


Figure 4.5: Gimbal response with noise present

As seen here the PID controller by itself cannot filter the noise and achieve a steady response as needed to match the step response [24,25]. This also shows that this controller is not robust enough to handle such complications. Although a P.I.D controller is fine for most industry practices, for an expensive technological piece of hardware a P.I.D by itself will not suffice. The designed P.I.D controller performs poorly in the presence of noise, to remedy this issue a method of filtering was introduced into the system. This filter filters out most of the noise from the feedback loop.

Chapter 5-Noise and Filtering

5.1 Noise Implementation

As seen in the previous section, introducing noise into the system reduces the performance of the controller. A poor performing controller when noise is introduced means that the controller is not robust enough. This could lead to catastrophic failure when implemented in prototyping and flight-testing phase. As noise in a system is omnipresent no matter where implemented. Whether that nose be an ambient signal, motor spool time, or even just noise of the measuring instruments such as sensors. A controlled system needs to be able to not only respond to a noisy signal but respond well.

White noise was introduced into the feedback loop of the control loop to simulate a noisy signal that must be dealt with, while also having the system still track accurately to the desired flight path angle. As this project is purely computational a Simulink "white noise" block was the only way to model in the noise accurately. The white noise signal itself was set to a value of .001. The white noise block was set to this value as to not overpower the signal, but also mimic real life situations. By doing this real-life noise signals and situations can be simulated computationally. The sample of the white noise block implemented within the linear system can be seen below.

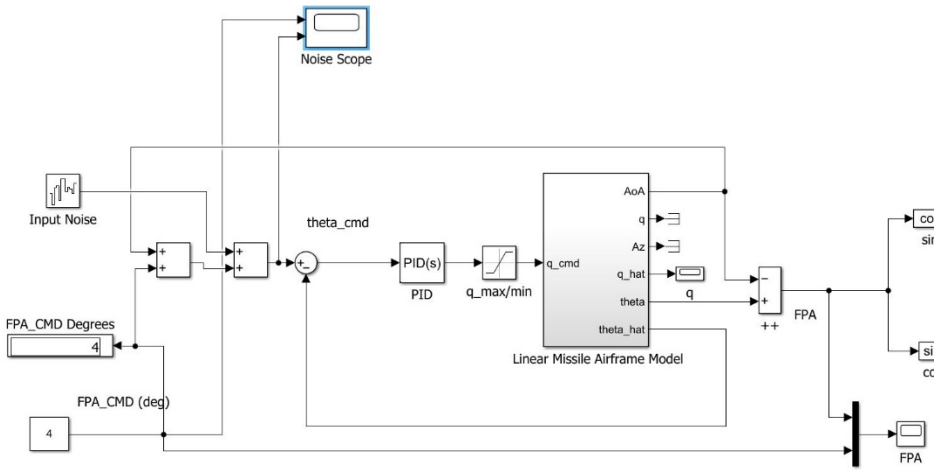


Figure 5.1: Simplified linear plant

To compare the reference flight path angle with noise, the two variables were plotted with one another. This is to give the scale of the noise introduced in the system while giving some expectation of how this noise will affect the system once implemented.

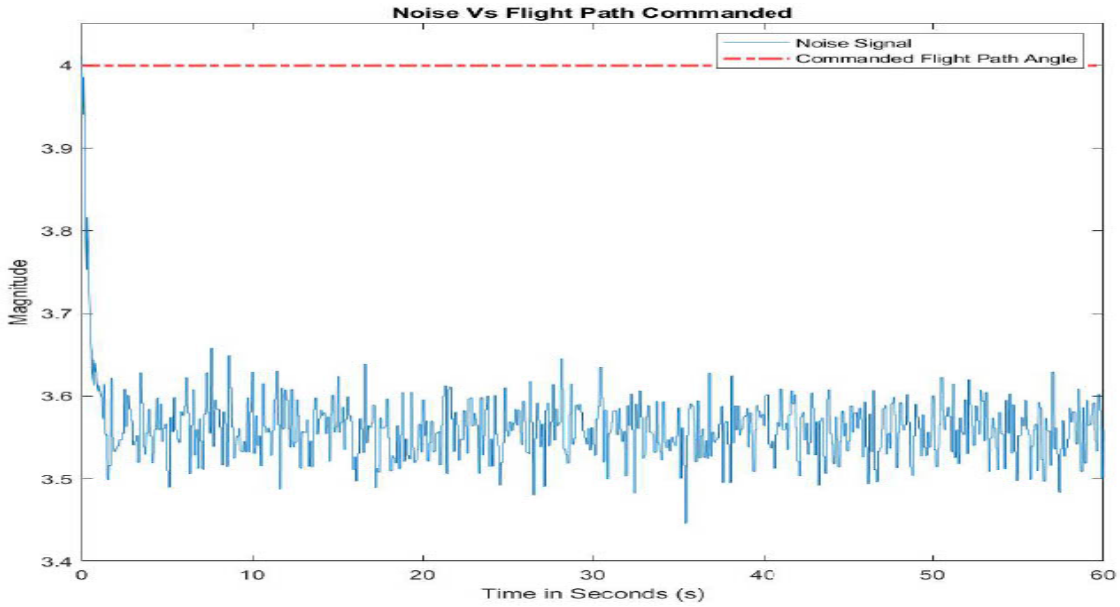


Figure 5.2: Plot of noise added to the reference signal

The noise and the commanded flight path angle signals are both mixed when returning in the feedback loop of a control system. Therefore, the P.I.D controller from before had a challenging time tracking towards the commanded flight path angle. Instead of the intended commanded flight path angle the controller was correcting for every bit of noise introduced into the system giving inferior performance.

To remedy the issue of noise introduced into the system, extensive use of Kalman filters is used in the aerospace industry. However, this system is a P.I.D controller and not an LQR controller. To implement a Kalman filter the controller must be an LQR, and once the Kalman filter is applied this transforms the LQR controller into a LQG controller. The solution used here was just a simple noise filter, much simpler than the common Kalman filter commonly used.

5.2 Filter Implementation

Implementing the filter into the Simulink model is a simple task. There were two options in the implementation, either implement the filter block or adjust the noise coefficient for the P.I.D block. The chosen approach was to autotune the noise coefficient using Simulink. By doing this, the noise would be effectively filtered, and the performance of the control system should increase significantly. To understand the noise coefficient and where the filter is used inside the P.I.D block refer to the figure below of the internal workings of the P.I.D controller.

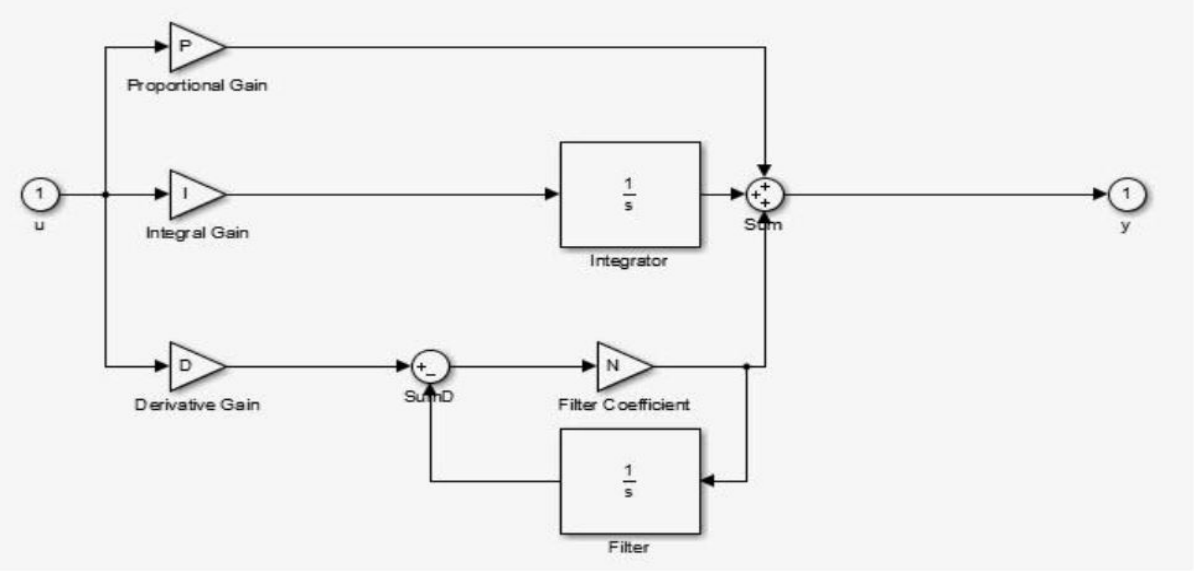


Figure 5.3: PID inner workings

From above when tunning the Filter Coefficient N, is then integrated over by the filter and fed back as the error state to subtract from the derivative gain. This is effectively filtering on the D of the P.I.D controller and the results were improved. The P.I.D parameters can be seen below, along with the autotuned parameters of the noise filter.

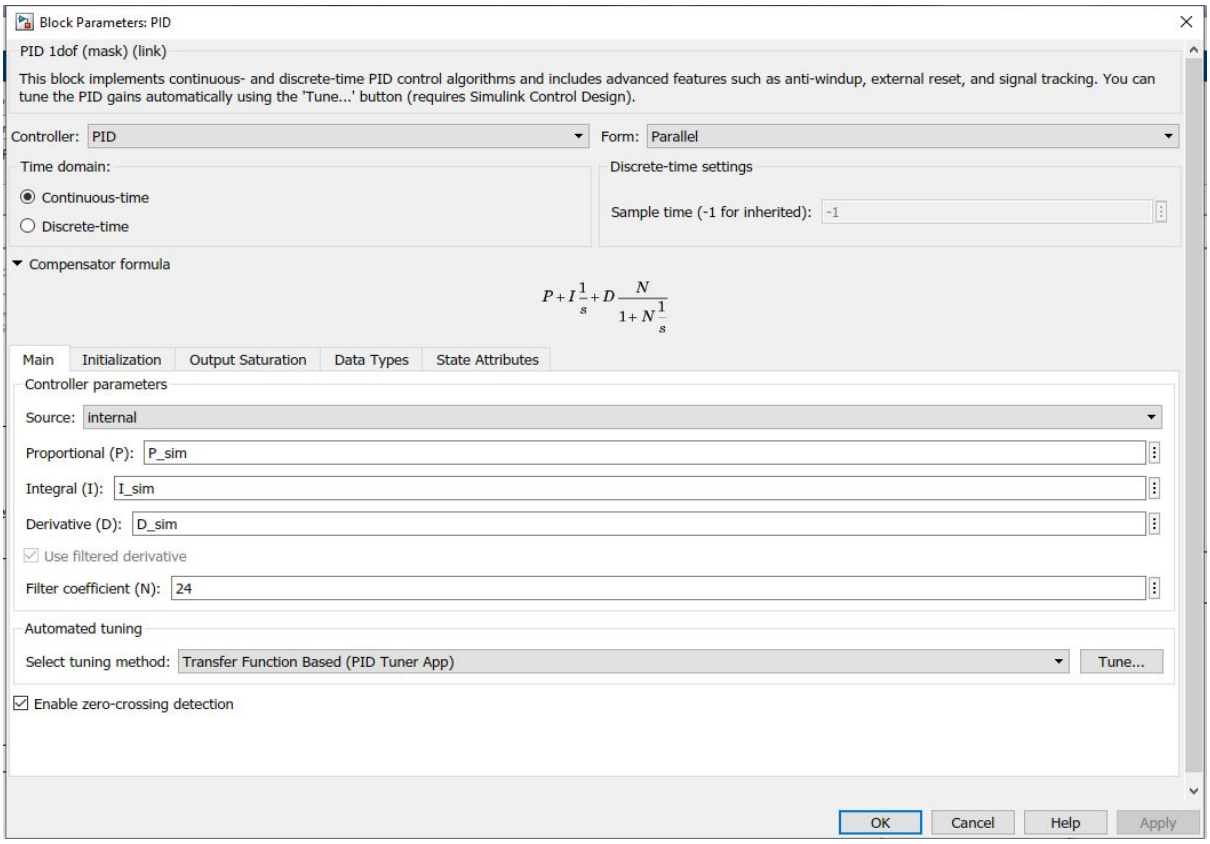


Figure 5.4: PID block properties

The filter coefficient was automatically tuned to a value of 24. This variable will stay the same for future tuning of the P.I.D controller using machine learning algorithms. With the filter implemented the previous problem of a non-robust P.I.D controller was solved. Now this system can be realistically and adapt to poor signal read performance. This is a crucial step and part of any control system. The next step will be to tune the P.I.D controller to optimal performance.

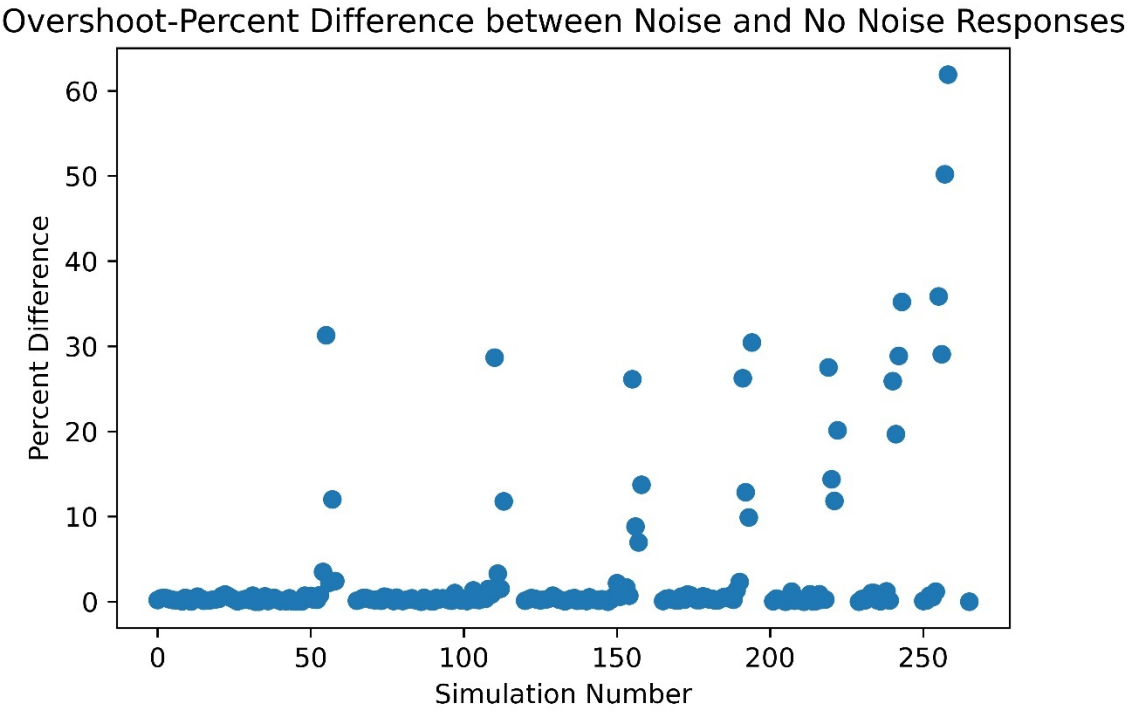


Figure 5.5: Responses with and without noise for overshoot

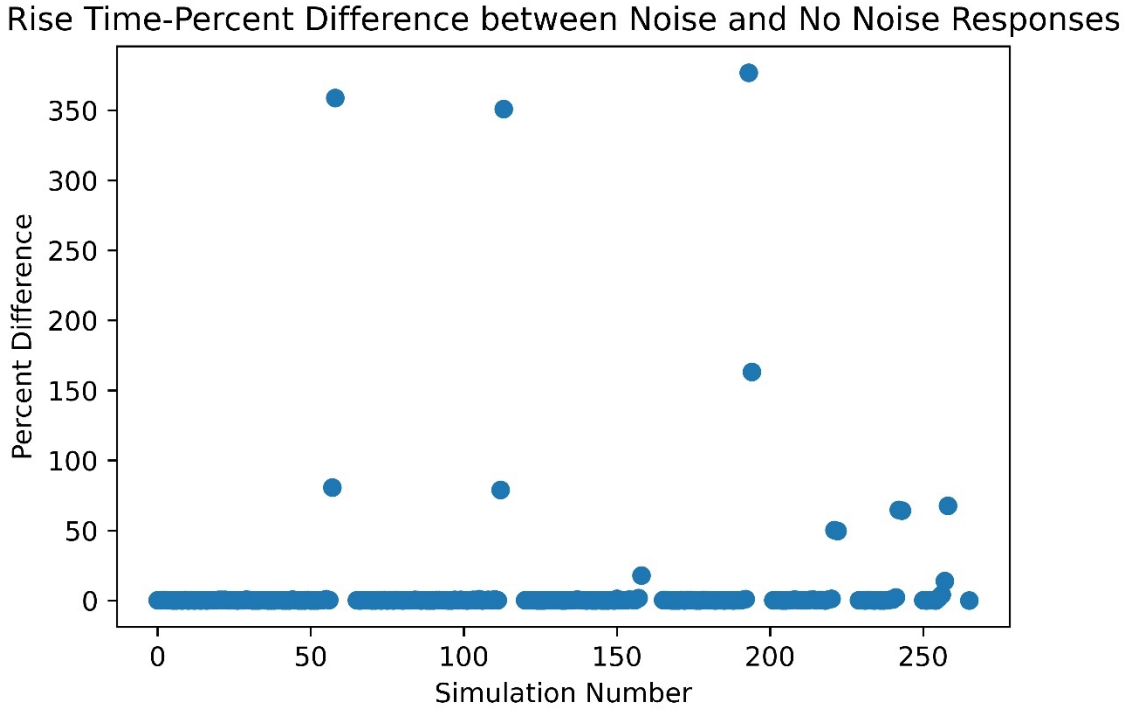


Figure 5.6: Responses with and without noise for rise time

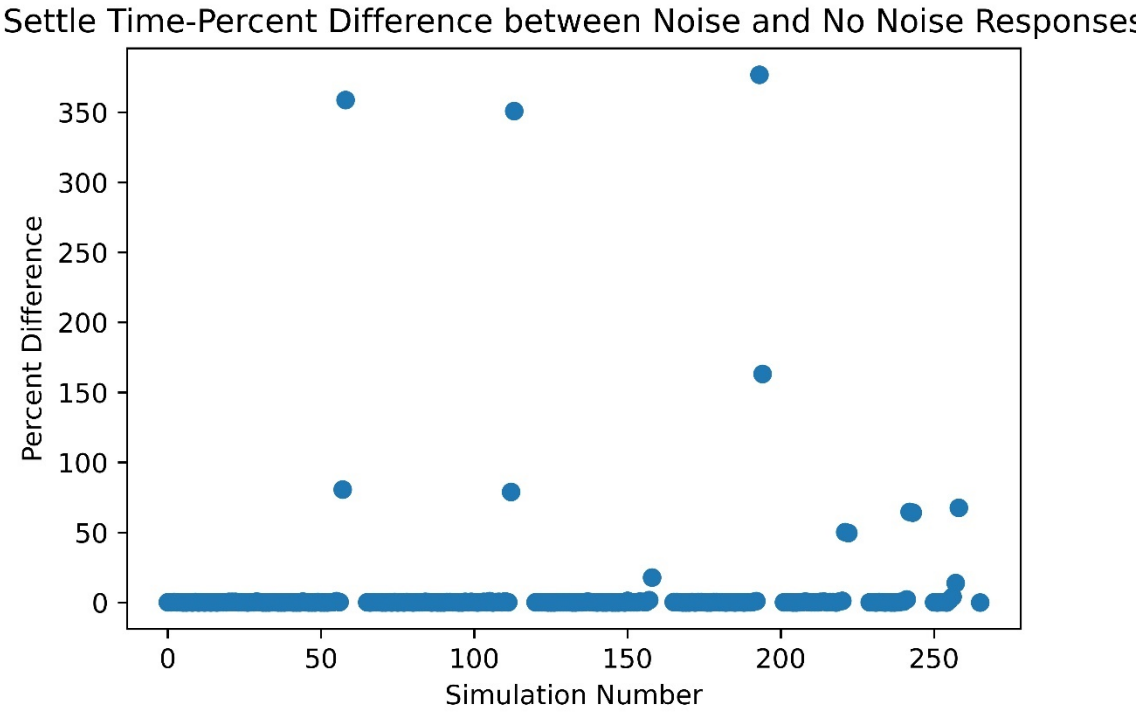


Figure 5.7: Responses with and without noise for settling time

The above plots are the difference between the simulations when noise is introduced versus no noise. For the vast majority there is almost no noticeable difference when the filter is introduced, however for some runs there is a very noticeable difference. These are only a handful of runs therefore they will be discarded from the simulation design space as not to introduce more errors into the machine learning model when trained. From here onwards only the noise data set is used to train the machine learning model. This is because in practical applications there will always be noise present in the system that has to be filtered out. As seen above in the plots the difference between noise introduced and no noise is insignificant. This also means the filter is working properly and the controller can respond well to noisy input.

Chapter 6-Machine Learning Tuning of P.I.D Controller

Manual tuning of a P.I.D controller can be considered just as much as an art, as it is a science. There are two common ways to tune a P.I.D controller. The first is to" ... first set integral constant K_I and derivative constant K_D values to zero. Increase the proportional constant K_P until the output of the loop oscillates, then the K_P should be set to half of that value for a "quarter amplitude decay" type response. Then increase K_I until any offset is corrected in sufficient time for the process. However, too much K_I will cause instability. Finally, increase K_D, if required, until the loop is quick to reach its reference after a load disturbance. However, too much K_D will cause excessive response and overshoot" [25]. This manual method can be seen as a try and fail method until you succeed, of course with such a complicated and expensive system such as a rocket this cannot be done.

The next method introduced is the Ziegler-Nichols tuning method. This method can be summarized in the table below with instructions to follow.

Table 6.1: Relationship table between gains and performance [25]

Response	Rise Time	Overshoot	Settling Time	Steady State Error
K_P	Decrease	Increase	No Trend	Decrease
K_I	Decrease	Increase	Increase	Eliminate
K_D	No Trend	Decrease	Decrease	No Trend

The steps for tuning a P.I.D Controller are as follows. [25]

- Determine what characteristics of the system need to be improved
- Use K_P to decrease the rise time
- Use K_D to reduce the overshoot and settling time
- Use K_I to eliminate steady state error.

These two methods are the most common approach to tuning P.I.D controllers. Both are either a heuristic model such as the Ziegler – Nichol's method or an iterative process as the manual tuning implies. Although these are the usual methods, by no means are these methods an exhaustive approach.

Although these methods are the tried and tested approaches used in the industry. With the emergence of Machine Learning and Artificial Intelligence unique and novel opportunities for innovation have formed within the aerospace industry. With the developed P.I.D controller system, the tuning needs to be completed to have the optimal performance. Instead of using the already established methods exclusively, developing a machine learning model to tune the P.I.D

controller will be done. This model will use the same performance metrics such as the rise time, overshoot, and settling time. Therefore, when comparing the machine learning model and hand tuned model it will be a fair comparison.

The proposed method of prediction for tuning the P.I.D controller will be a model based on the multilinear regression algorithm. The dependent variables are the controller parameters the proportional, integral, and derivative control gains. The independent variables will be the performance parameters which are the rise time, settling time, and overshoot percentage. A flow chart of how this subsequently works is shown on the next page.

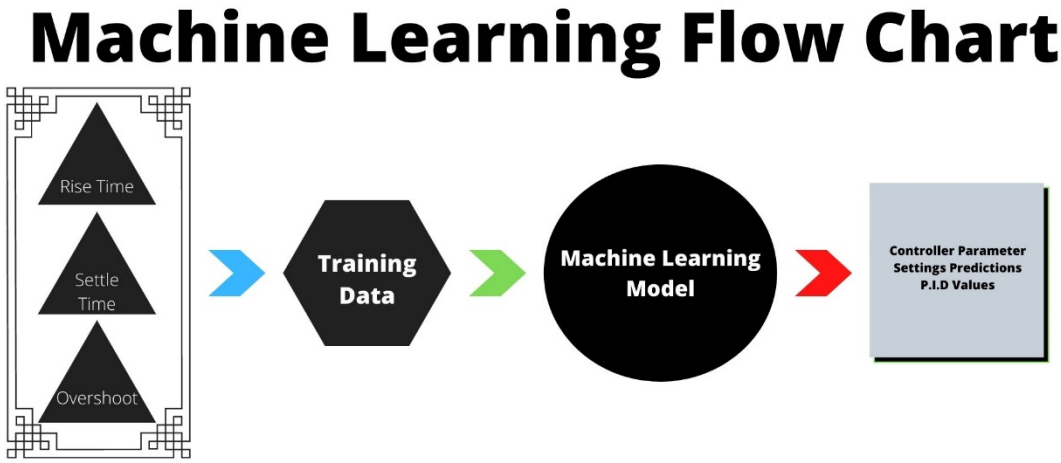


Figure 6.1: Machine learning flow chart

Using this process, the predicted P, I, and D parameters should be the optimal settings when trained in the design space. By simulating over a large design space in Simulink, a design space is created. Over this design space, the subsequent model is trained using these independent variables. With these independent variables, assumptions about the data are made through statistical testing and analysis of this dataset. Once the dataset satisfies the primary assumptions of the linear regression framework, a model is developed within python. This regressive model is then trained on the independent variables that were declared at the beginning. After the model is trained predictions can be made about the controller parameters. These predicted parameters and the performance will then be compared to the hand tuned values to evaluate if the machine learning model provides another way to tune the P.I.D control system.

6.1 Design Space

Before a machine learning model can be built, modeled, and implemented. The data needs to exist first, here the data is nonexistent specifically for this rocket model. Therefore, the data must then be generated, cleaned, and then implemented. The bulk of the work and development process of a machine learning process is the data collection, and cleaning, as this was the case here as well.

The first proposed design space was a three-column vector of the Proportional, Integral, and Derivative control gains. This vector spanned a linear space from [0,15] with 100 points in between. This vector was constructed for each controller parameter setting. These three column vectors were then permutated to obtain all the possible combinations with no repetitions allowed for the first design iteration. This initial design space totaled over 100,000 different combinations and simulations possibilities to obtain information on. However, due to a lack of computational resources and the intensity of such a task this design space was quickly abandoned.

The second iteration to improve the design space and simulation data gathering was to limit the design space in the amount of design and simulation points. Shortening the 100 linearly spaced points to 30 spaced points between the values of [0,10]. This drastically reduced the number of simulation and design points, however the problem here was deficient performance for most of the recorded datapoints. This would develop a poorly predictive model, as the model will be only as accurate and useful as the data that trained it. This data was thrown out and another iteration was needed once again.

On the third iteration negative gains were introduced in the range from [-7,3]. The positive gains were limiting to be from zero to three because after this range no useful responses were generated. Even within the range between zero to three a lot of the performance charts, and variables were of no use. However, there were a few responses that performed outstandingly this led to this range being kept as the data could always be cleaned. The purpose of sweeping in this large variable space was to catch all the performance characteristics from the permutations of these combinations. Of course, this would generate unfavorable data as well. Generating a small number of unusable data points in favor of generating a vastly greater number of viable design points, was an acceptable action. This iteration of design space iteration also lowered the number of linearly spaced vectors from 100 to 11. An enormous drop in the amount of design points generated was experienced. Starting with over 100,000 data points to just under 300 for the final design space. A sample of parameters can be seen on the next page.

Table 6.2: Partial representation of the permutations of p, i, d values

P	I	D
-7	-7	-7
-7	-7	-6
-7	-7	-5
-7	-7	-4
-7	-7	-3
-7	-7	-2
-7	-7	-1
-7	-7	0
-6	-7	-7
-6	-7	-6
-6	-7	-5
-6	-7	-4
-6	-7	-3

6.2 Data Collection & Analysis

As seen in the data table the combinations are taken from the interval [-7,3] with every combination possible of the three variables. These values were then automatically simulated in Simulink to retrieve and store the performance variables. By collecting all the responses to these combinations of P, I and D parameters the same sized matrix of results were gathered. The gathered data is the independent variables needed to train a sufficient model. This matrix of gathered data will be referred to as the Simulation Design Space or (Sim Space) for short. The complete plot of Sim Space can be seen below as a contour plot.

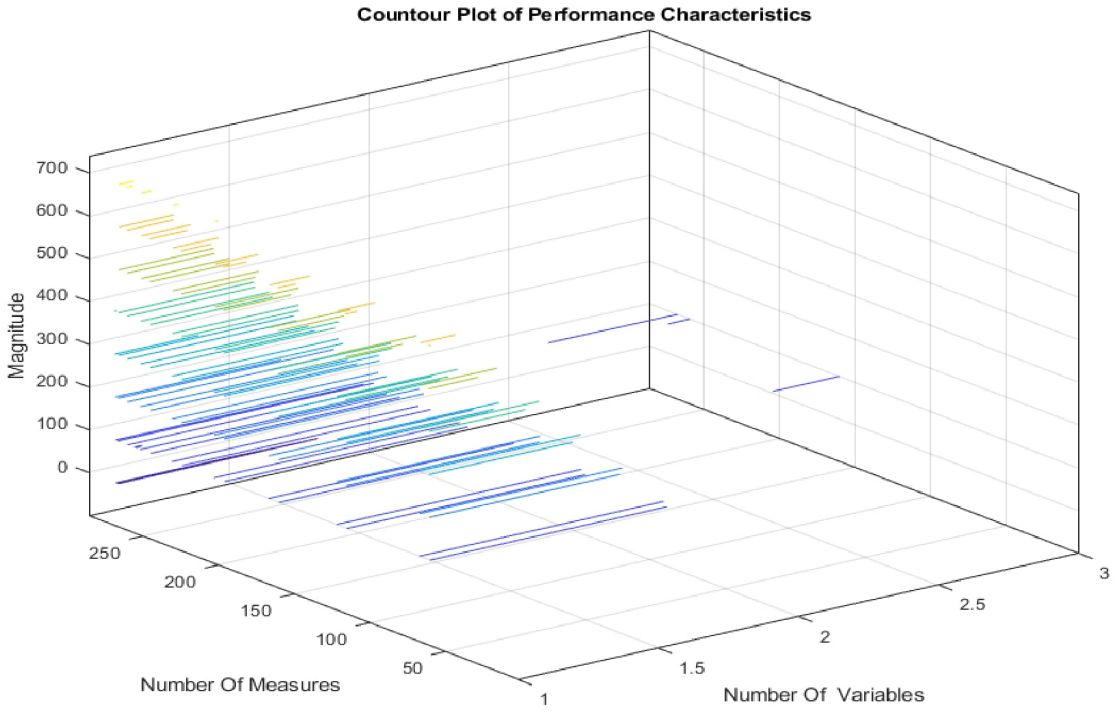


Figure 6.2: Contour of responses by magnitude

The Z-axis is the norm of the responses aka the magnitude, the X- axis measures the number of measures or simulations, and the Y-axis is the number of dependent variables. From initial inspection this contour plot suggests a linear relation between the variables. Further analysis is needed to validate this relationship.

Once the entire data needed was collected, the rows with a Nan (not a number) were dropped from the data set. This was to exclude nonrealistic values that either grew the response to infinity or were not captured within the period of the simulation. Both scenarios are not physically possible for the system or performance. A type of such response can be seen on the next page.

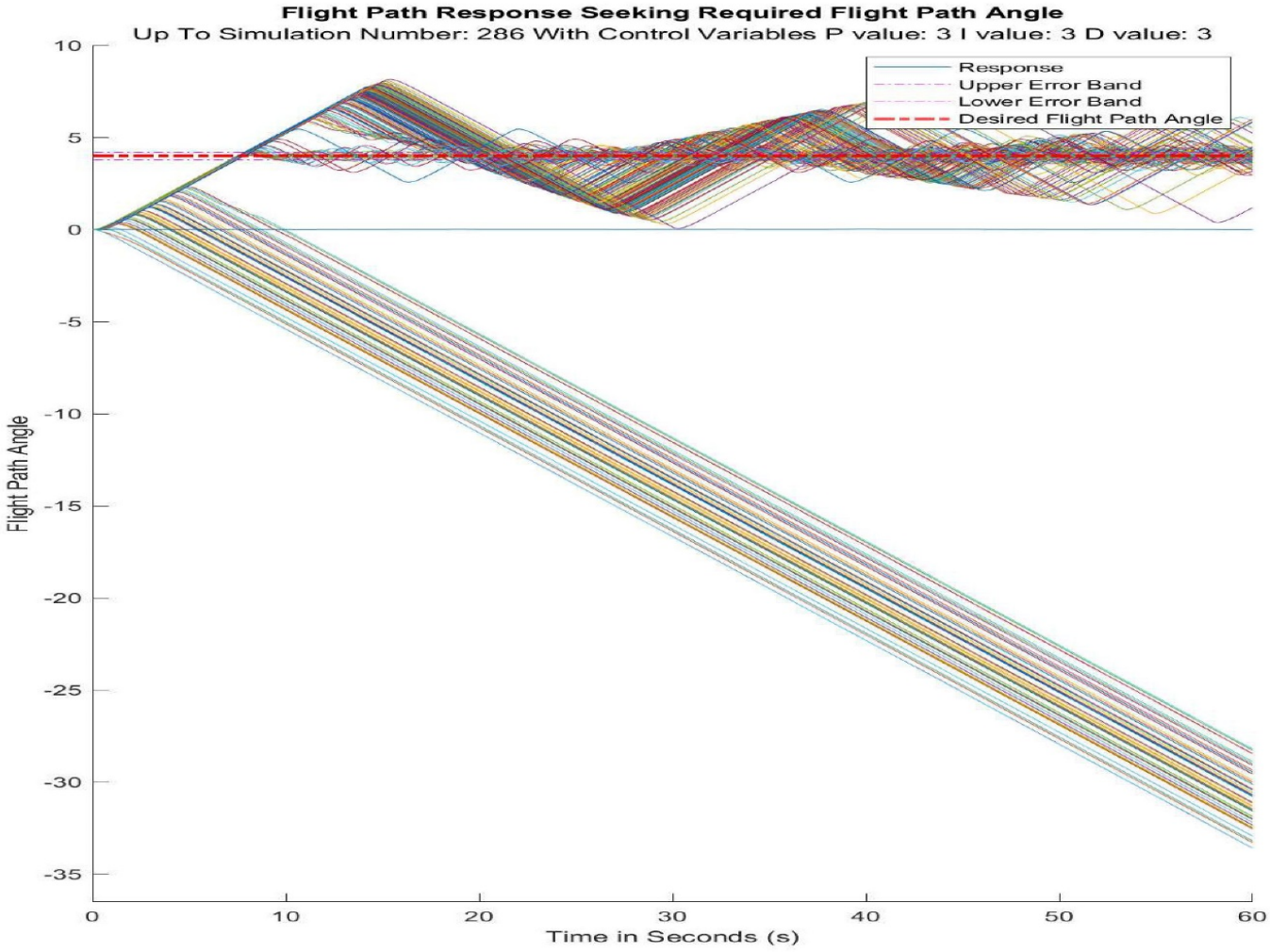


Figure 6.3: Full evolution of all recorded responses

The responses that grow to infinity and never seek back to the reference signal are the values that were removed from the dataset. This reduced the Sim space from 283 simulations to just under 270 simulations. Most of the simulations performed were successful and provided valuable responses.

The figure above represents all the simulations performed and logged. Taking a deeper look into these responses graphed there seems to be groups of responses that are remarkably similar. To visualize the relationship between these responses the evolution from simulation to simulation was summarized every 30 simulations. This can be seen below.

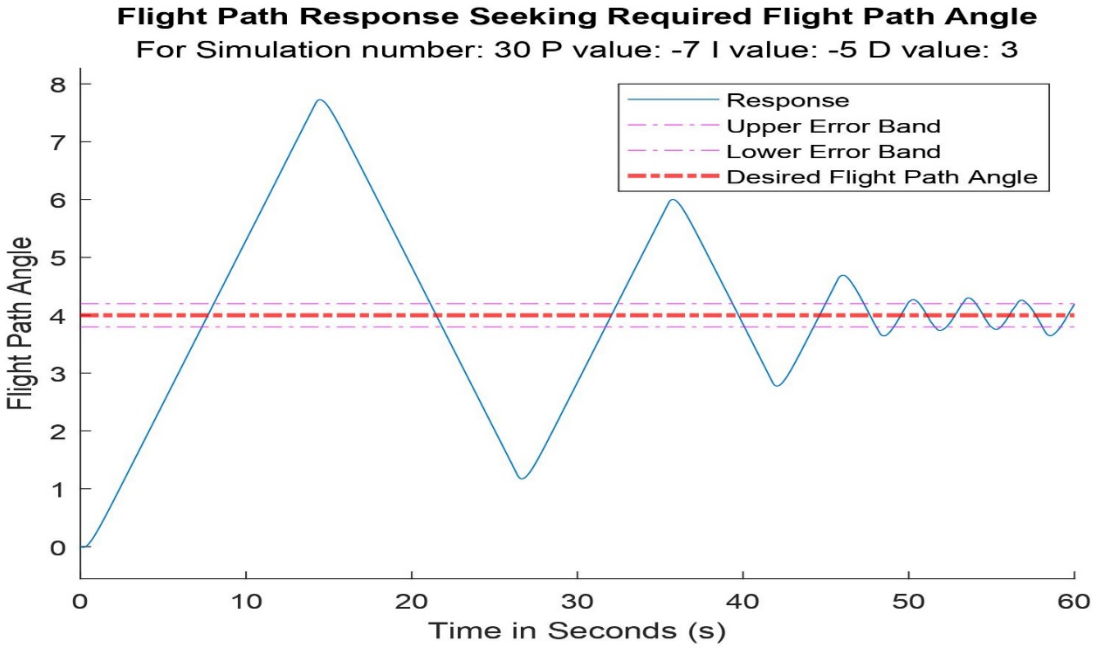


Figure 6.4: Family of run #30 simulation graphs

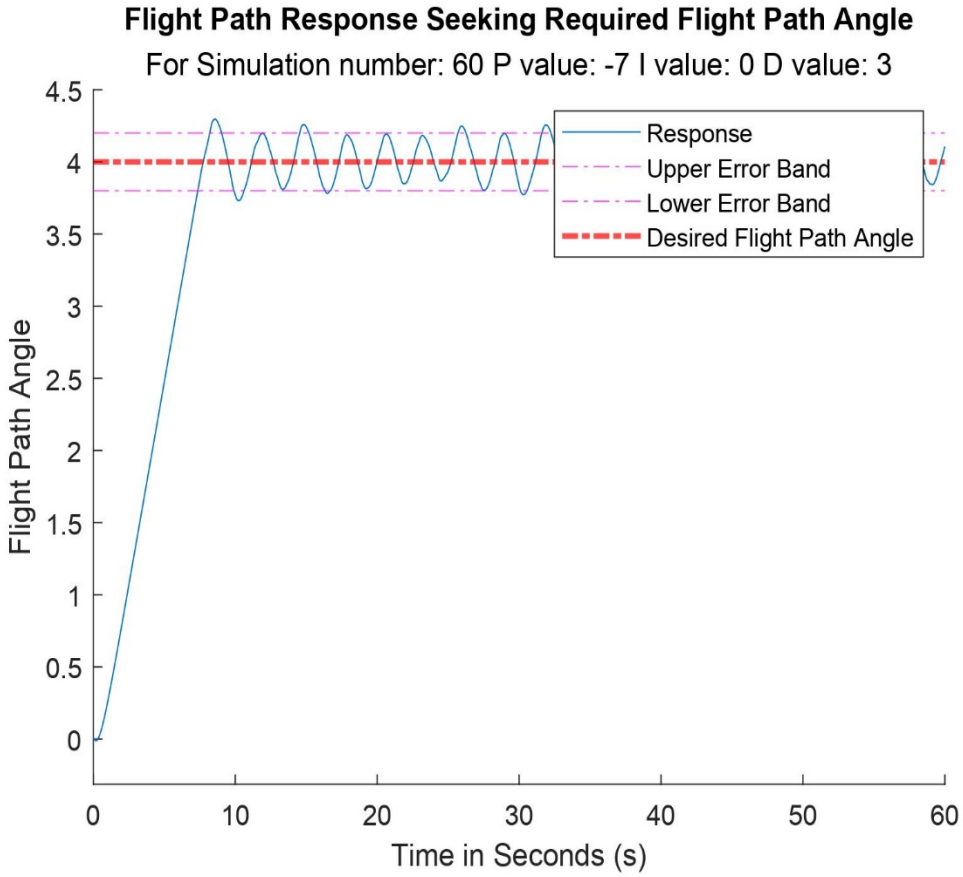


Figure 6.5: Family of run #60 simulation graphs

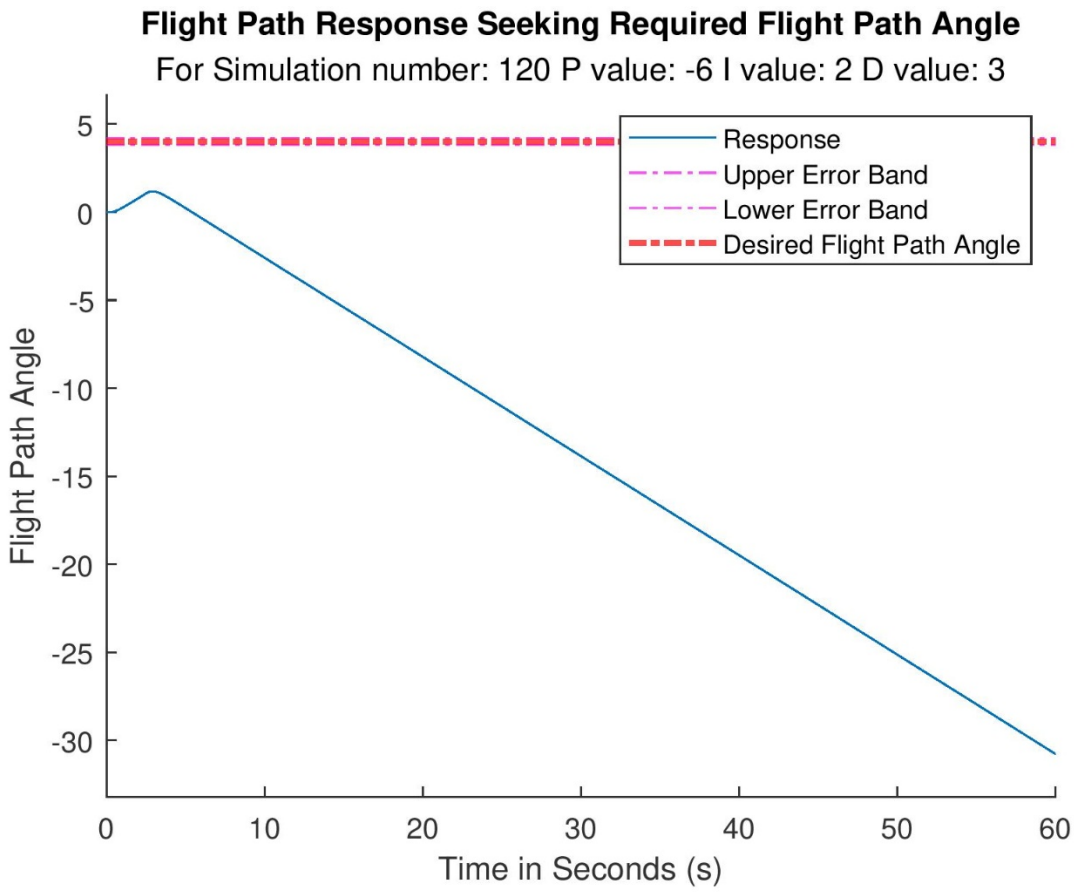


Figure 6.6: Family of run #120 simulation graphs

These three figures succinctly summarize all the responses seen in accordance with the Sim space parameters. The differences between the graphs of the responses can be broken into optimization of one of the performance parameters. A full graph of responses with some of the best graphical responses can be seen below. These were selected plots to visualize how some of the better values look once plotted against each other.

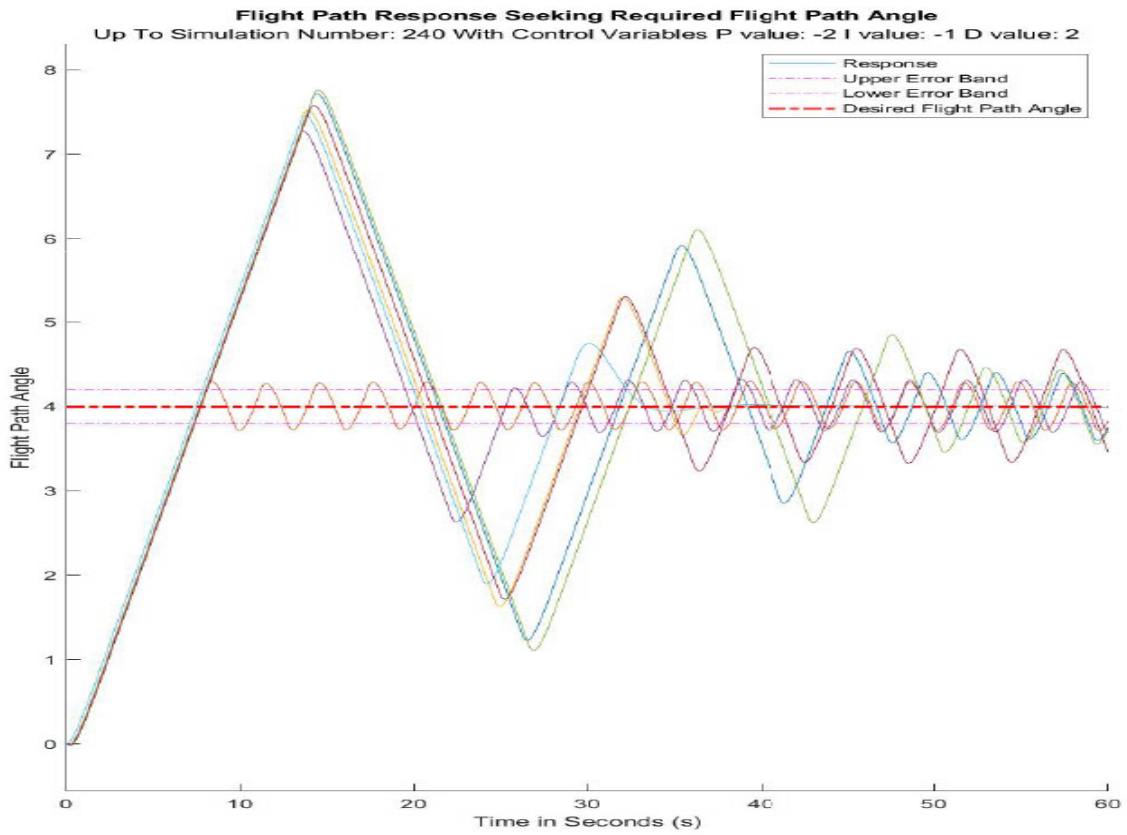


Figure 6.7: Evolutions of response with varying p.i.d parameters

The data collected in these simulations were also done in two distinct categories, one category for the simulations ran without noise the other category for simulations ran with noise. It is important to note while there were differences in the end when predicting values for the P.I.D values. Graphically the responses did not differentiate between the two categories of simulations used when collecting data. However, for practicality purposes and real-life applications the simulation data generated with noise would be the most accurate, and the set generated without noise would be the most idealistic cases.

After visualizing specific runs to get a general sense of the data, and the performance that could be expected from the Sim space. General data analysis was run to visualize and describe the data that was collected. This step was necessary to visualize any discrepancies that might have occurred that were not easily detectable within the simulation runs or by inspection. One crucial factor of the data is the distribution of the results that we are receiving from the simulations. The distribution of the overshoot, rise time, and settling time can be seen in the next three figures.

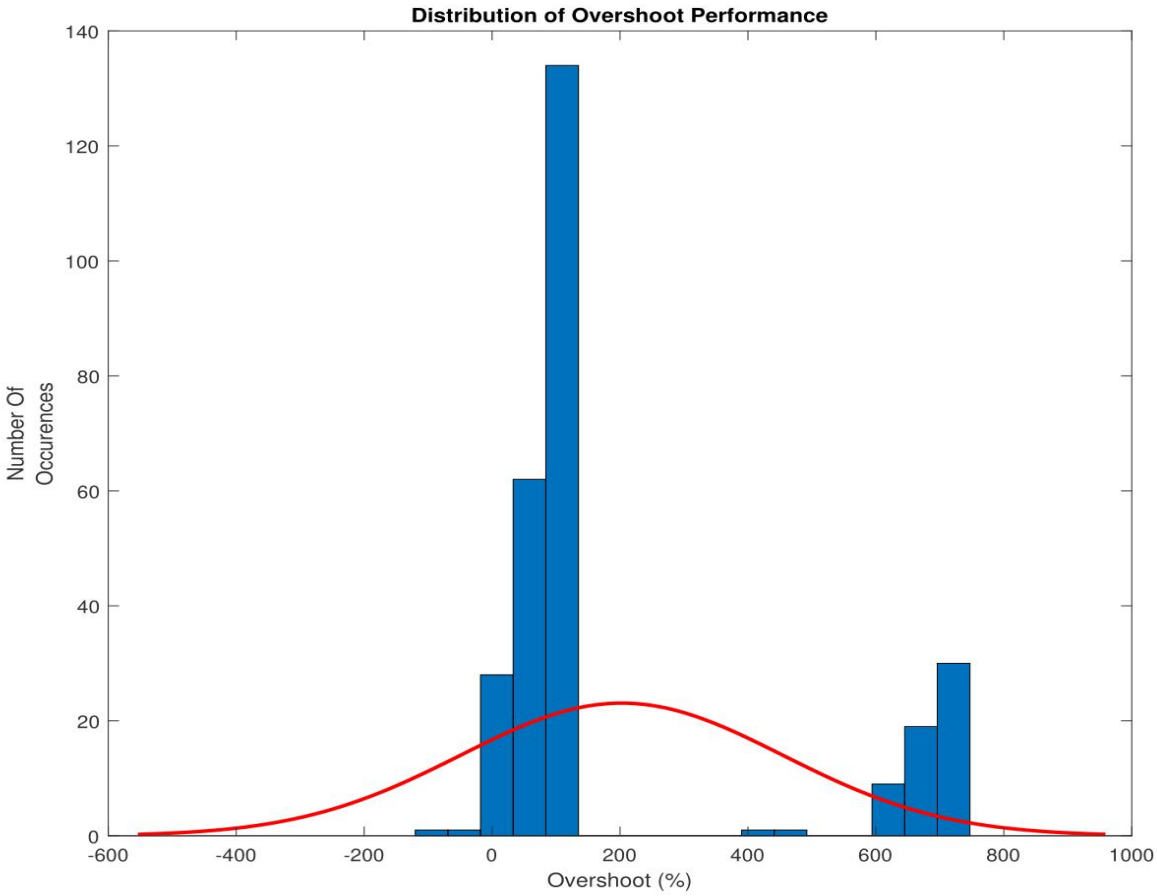


Figure 6.8: Distribution of measured overshoot percentage

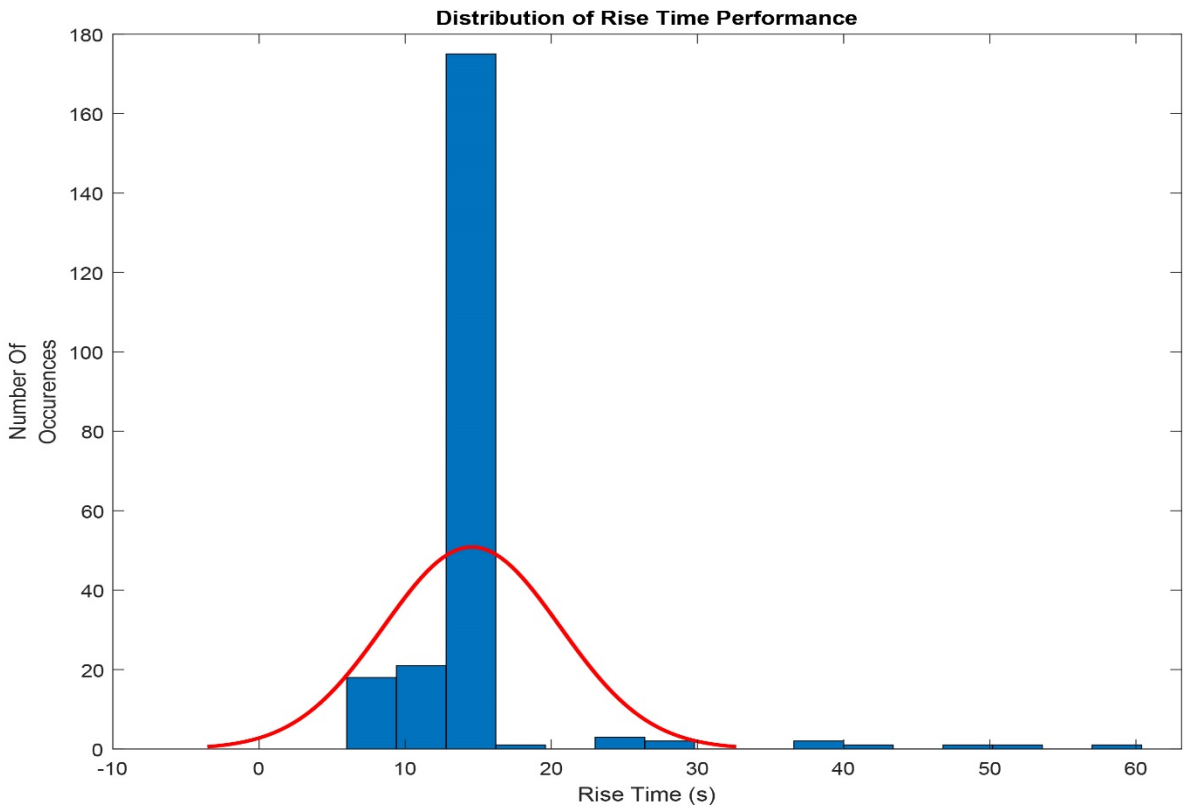


Figure 6.9: Distribution of rise time measurements

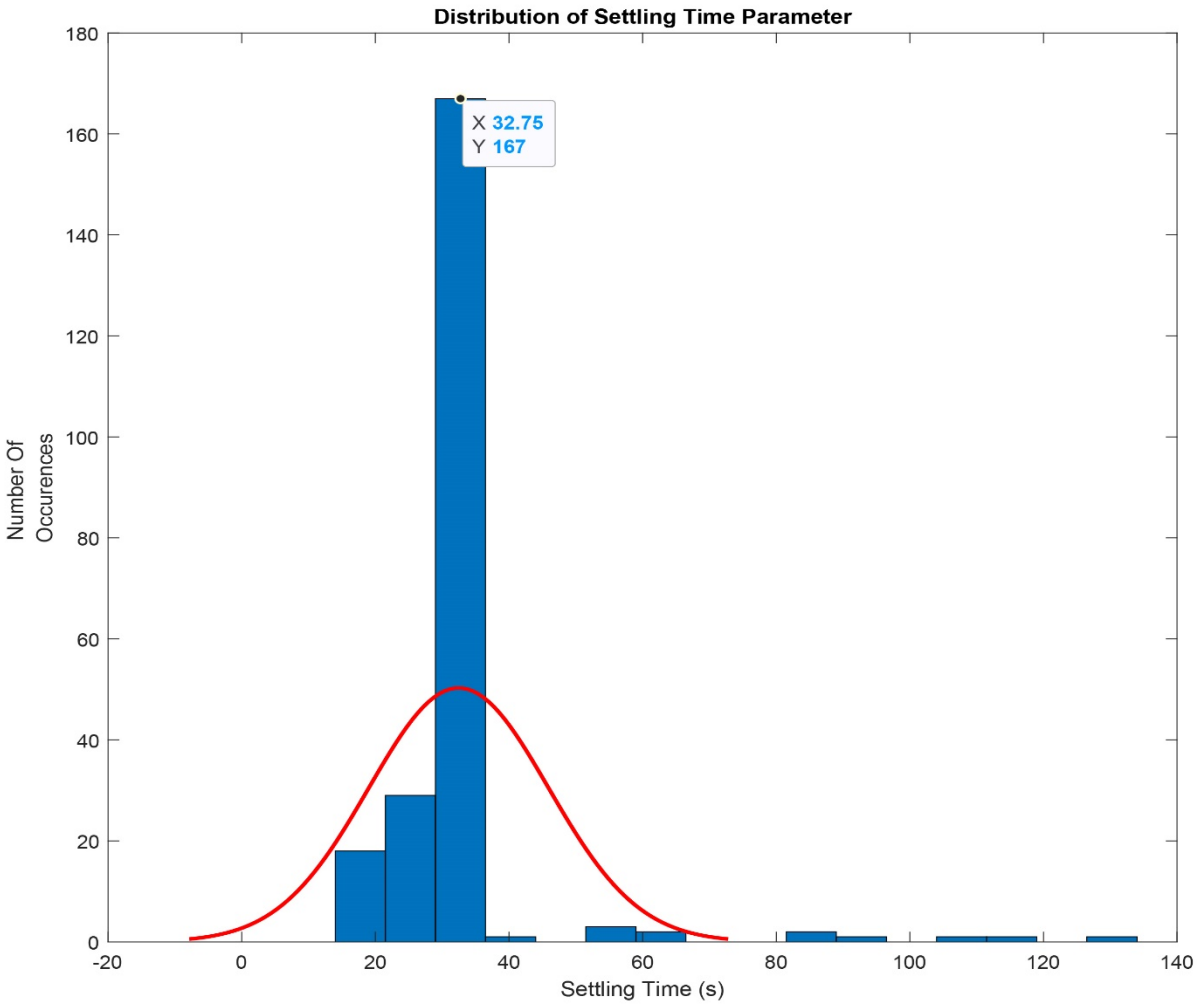


Figure 6.10: Distribution of settling time measurements

From the presented plots the performance parameters can be seen to have a few outliers in the data set, but these outliers are low in frequency compared to the data collected. Another insight into the data is that the performance parameters are close to being normally distributed, with a long left or right tail. This can be important because some probabilistic distributions must be transformed depending on the machine learning algorithm applied. Now that the general distribution of the data can be seen, a descriptive box plot would be nice to visualize as well.

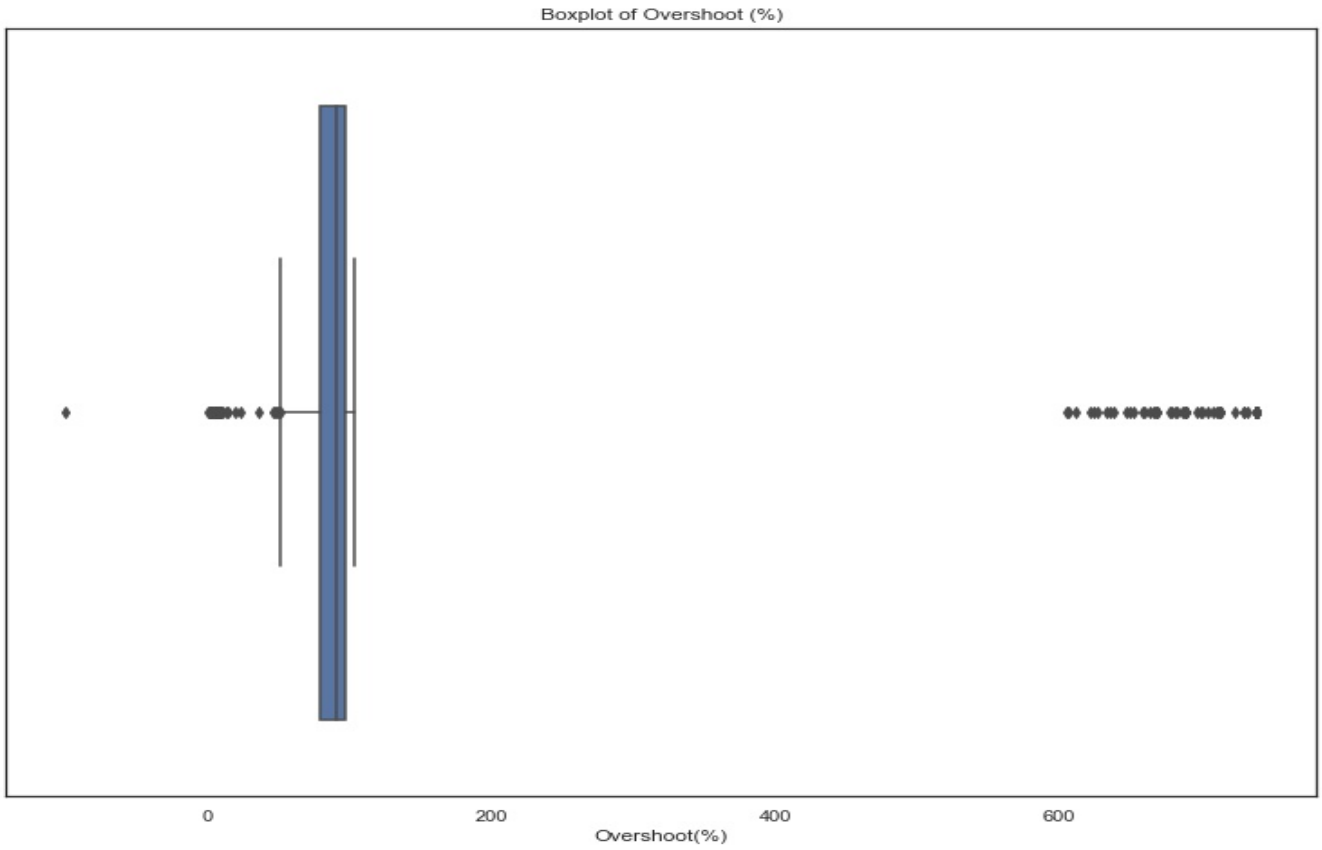


Figure 6.11: Boxplot of overshoot performance

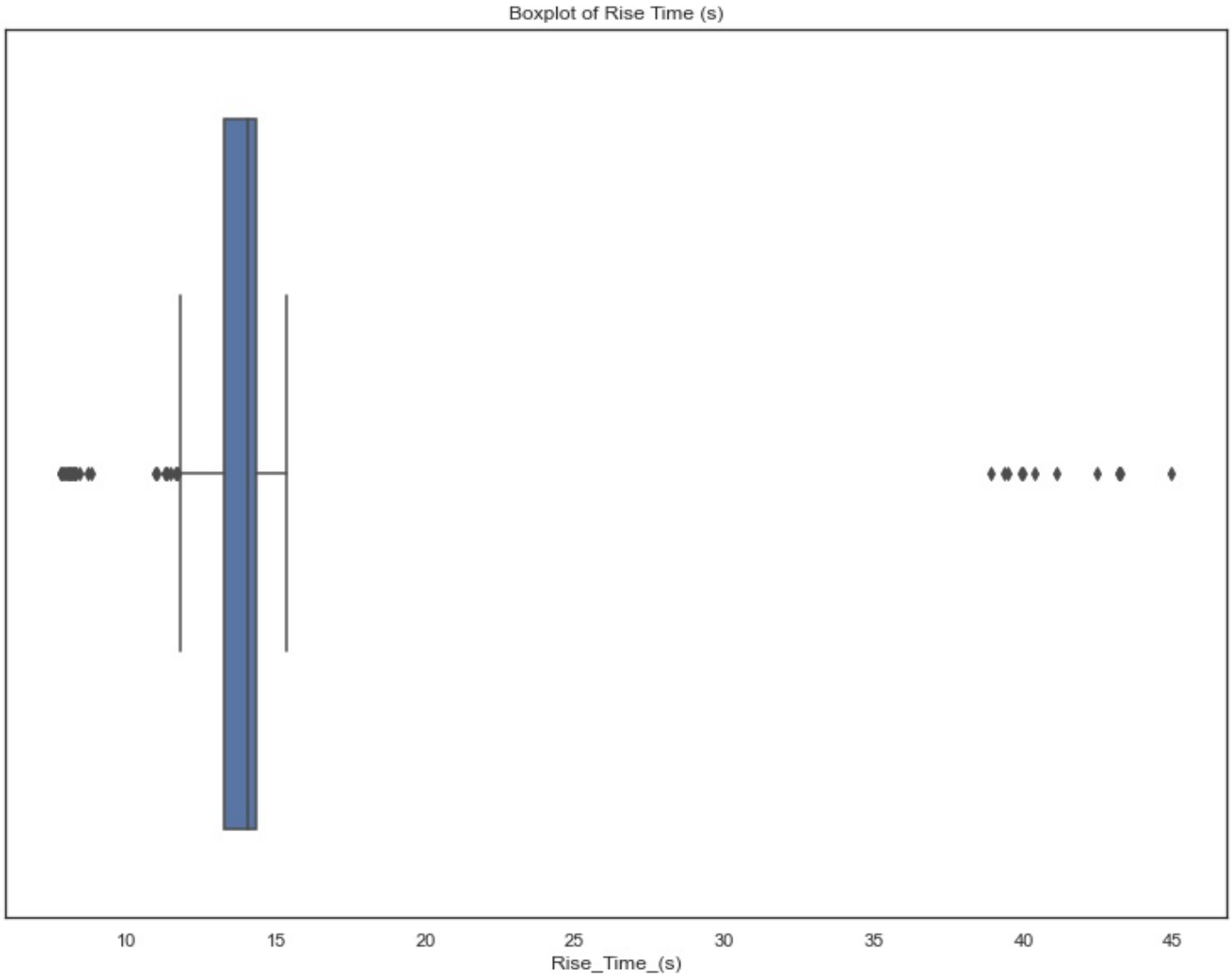


Figure 6.12: Boxplot of rise time measurements

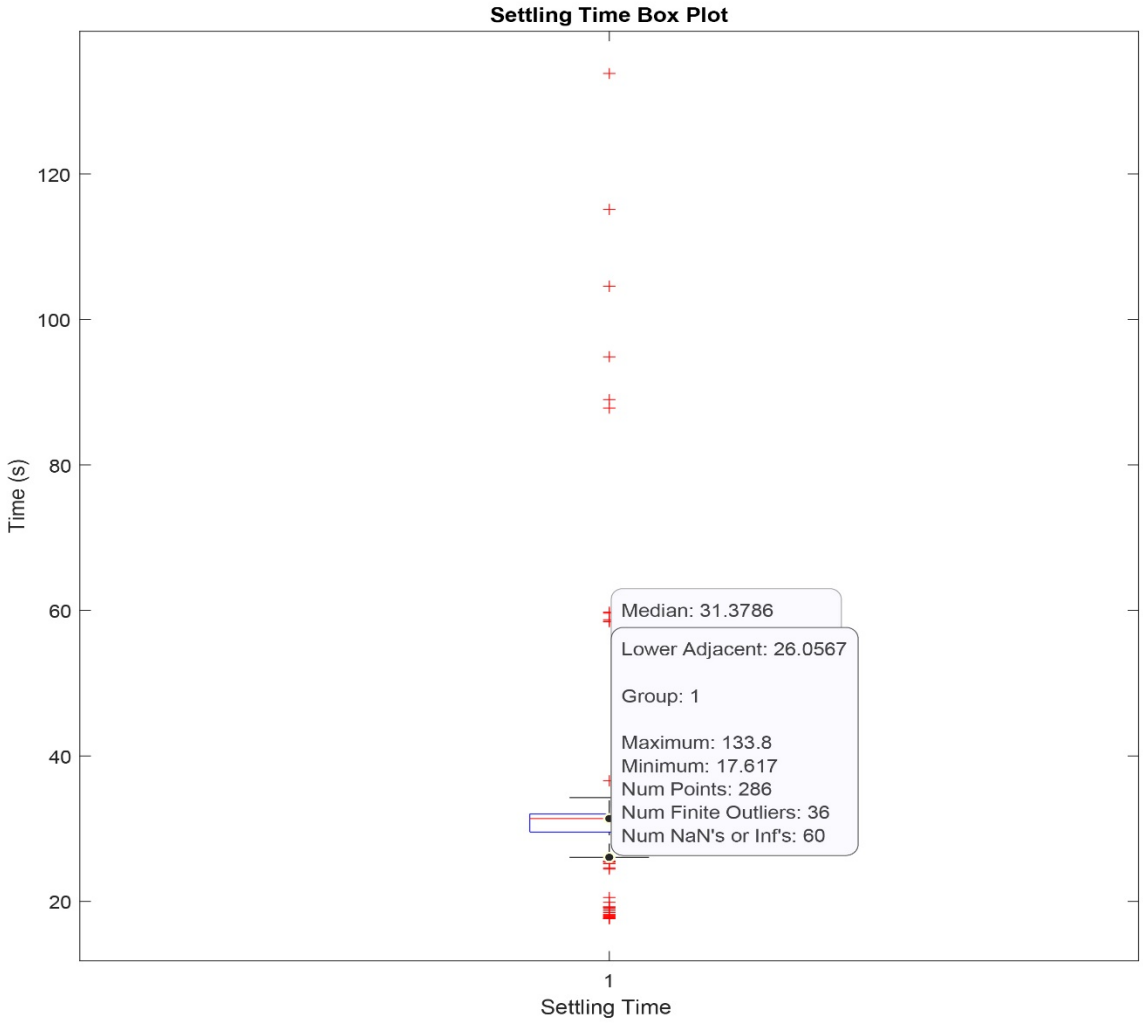


Figure 6.13: Boxplot of settling time measurements

The shown boxplots can help identify where the median lies within the values, and even more so how this dataset is spread in comparison. Through this visualization the outliers are definite standouts, however the outliers are evenly distributed on both sides of the spectrum. There are noticeably high outliers in each case and noticeably low outliers as well. As the balance in the data is kept an assumption cannot be made one way or the other. This leads to a correlation study needing to be done between not only the P, I, D values, but the performance parameters as well. The following are the results of those correlations.

	p	i	d
p	1.000000	0.574923	0.327693
i	0.574923	1.000000	0.569976
d	0.327693	0.569976	1.000000

Figure 6.14: Correlation study between p, i, d settings

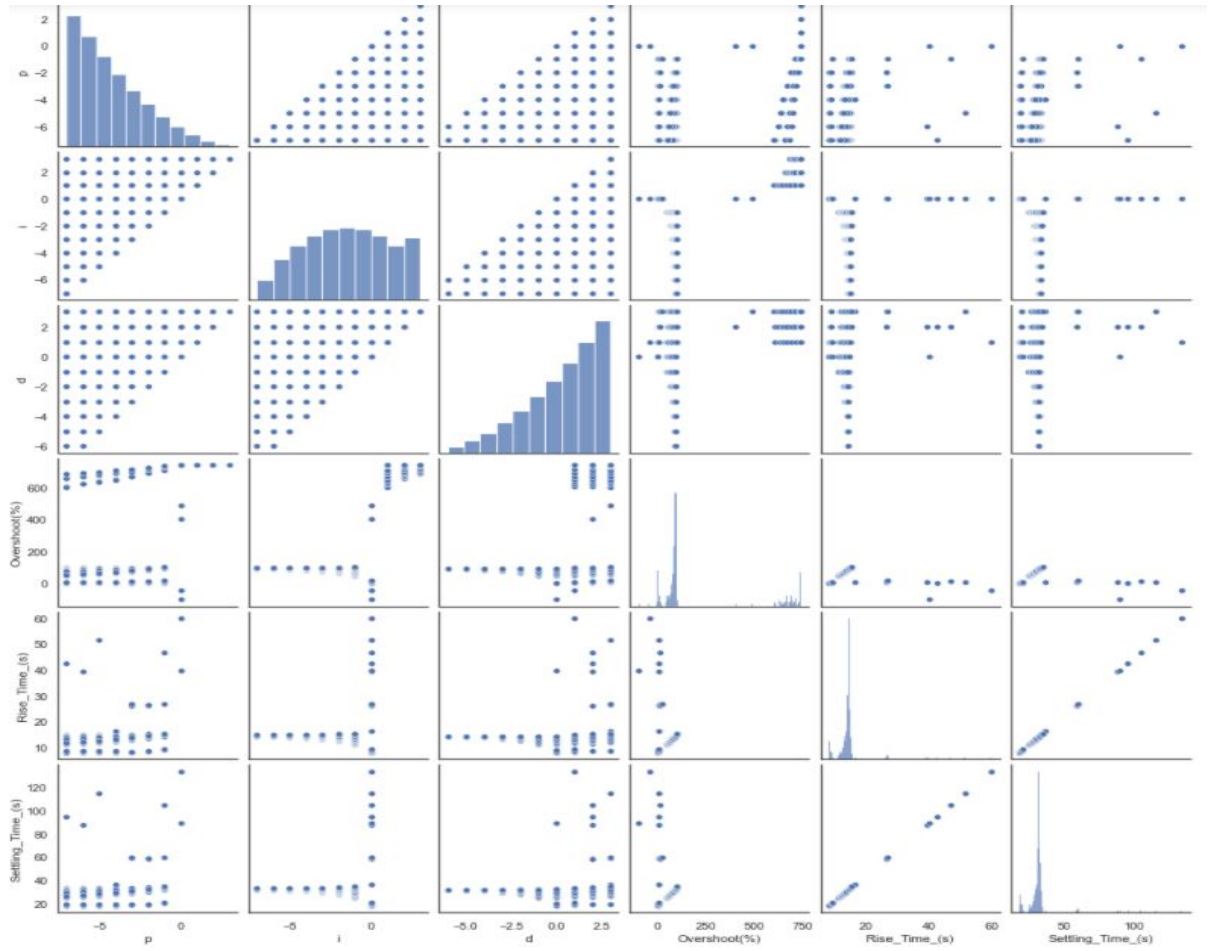


Figure 6.15: Correlation pair plot between all variables

From these correlation studies the P, I, and D values are all positively correlated to one another. The main thing to note is even though they are correlated to one another there still is enough difference to develop a model. Although these variables are correlated towards one another they are not correlated enough to introduce multicollinearity which is what is needed to avoid when using linear regression models. Moving onto the complete correlation pair plot between all variables, the performance parameters and Sim space, there are no variables that are too correlated with each other that could introduce multicollinearity besides the rise and settling time. Those two variables have an extraordinarily strong relationship towards one another, however because of the space of the simulation and the system at hand this is to be expected. This is a steady state system, that is time invariant and only changing due to the commanded flight path angle. With so many things held constant a relationship between some variables is unavoidable.

With the correlation studies completed the bare minimum has been satisfied to develop a linear regression model. The assumptions made about the data have been confirmed to hold true allowing for a model to be used to predict the independent variables of P, I, and D. The next step is to introduce linear regression, explain the theory and assumptions behind it, build the model, obtain prediction, then compare the results achieved.

6.3 Linear Regression Modeling

Linear regression "... can be used to forecast effects or impact of changes. That is, the regression analysis helps us to understand how much the dependent variable changes with a change in one or more independent variables" [26]. In this case there will be three different models for each dependent variable, based on the independent variables of the performance parameters.

There are multiple types of linear regression. This model will take advantage of multilinear regression forecasting to hopefully obtain the best solution possible for tuning the P.I.D controller. Before diving into the model, the underlying mathematical implications, assumptions, and loss functions must be introduced in explored in context of this model.

6.3.1 Linear Regression Equation

The simplest form of the linear regression equation can be explained from the algebraic expression for a line.

$$Y = c*x + b \tag{6.3}$$

Where Y is the prediction, b is a constant, x is the score of the independent variable, and c is the regression coefficient [26]. This is no different from the linear regression equations used to fit the data. Linear regression is just fitting a line through the data in the best way possible to minimize a certain type of error. This model used the commonly used Ordinary Least Squares or (OLS) approach.

The OLS approach minimizes the Mean Squared Error from each observations distance from the line fitted to the data. The Mean Squared Error is just the difference between the predicted value and the actual value squared, the summed over the design space. An example and visual representation can be seen below.

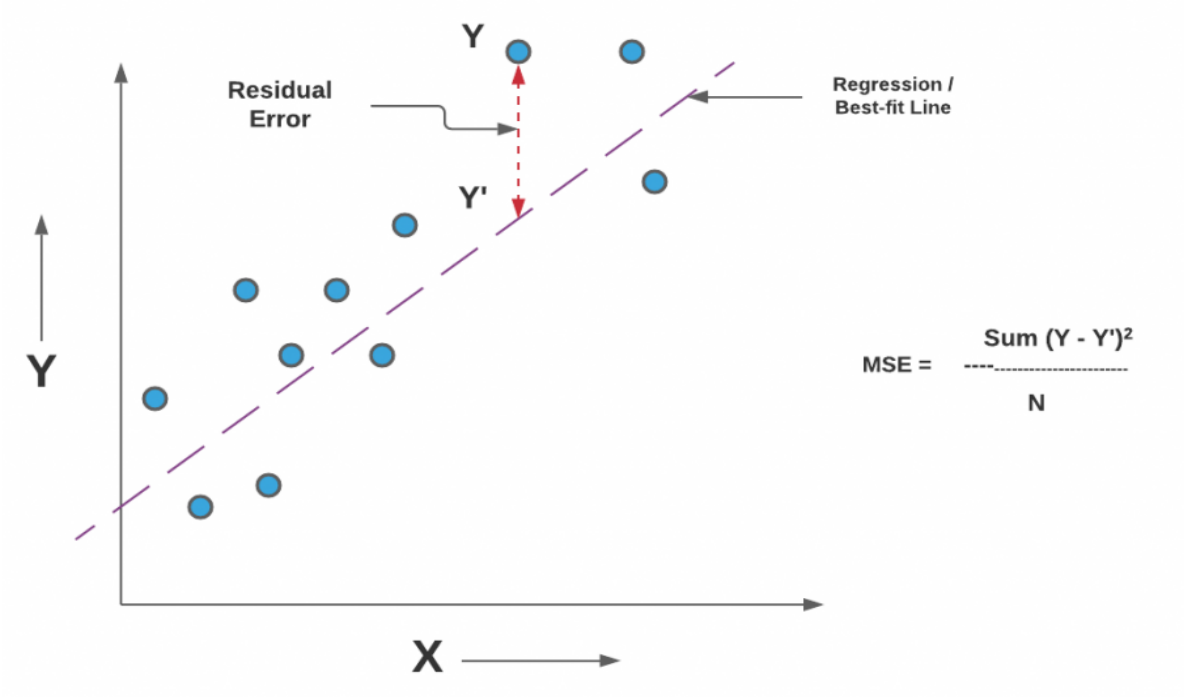


Figure 6.16: Mean squared error example [27]

The MSE for above would be the sum of all the distances of each blue dot from the red dotted line, then divided by the number of observations. This is how the MSE is calculated for models that take this approach. The MSE will be the parameter that will be minimized for when training the model later and optimizing the predictions. The way this model will optimize the cost function will be by the technique named gradient descent which will be covered in a later section. Now that a general understanding of the error has been introduced the next step is introducing the cost function for the linear regression model.

Luckily, the cost function here is just the MSE equation applied to three different variables at one time. Since there will be three independent variables the MSE has to consider each of the observations. The equation now accounts for the error in all three measured instances. Another name for the MSE using the OLS approach is the cost function. Therefore, the cost function for this model is as follows:

$$MSE = \frac{1}{2N}\sum_{i=1}^{n}\left(y_i - (W_1x_1 + W_2x_2 + W_3x_3)\right)^2 \quad (6.4)$$

Where the variables Y_I is the actual value, and $(W_1X_1+W_2X_2+W_3X_3)^2$ is the predicted value. This is the same as the MSE introduced earlier just extended to three variables. It should be noted the entire expression is only divided by two to make taking derivates easier, which is the next step.

After defining the cost function of the multilinear regression model, the next step would be of course to minimize it. By minimizing the cost function, the error would be minimized as well, giving the best possible prediction for that step. Applying this principle at every step until a global minimum for the cost function is reached would lead to the lowest possible error for the entire model. This technique is called gradient descent, and this is also the technique used to optimize the model developed here.

6.3.2 Cost Function Optimization (Gradient Descent)

The cost function for this model was introduced in the section before. The MSE will be the cost function to optimize for in this example. The way to go about this is to find the gradient of the cost function, when using multilinear regression which is the case here the partial derivatives need to be taken. Once the gradient of the cost function is found an iterative approach for the next prediction that will optimize the cost function can be written as follows:

$$\theta^{next\ step} = \theta - \eta \nabla_{\theta} MSE(\theta) \tag{6.5}$$

Where the Greek letter Etta is the learning rate, a hyperparameter to be tuned in the model, and theta is the cost of the next step while using the optimization method. Using gradient descent can optimize the cost function and improve overall performance of the machine learning algorithm. The pitfalls to this approach are that the algorithm itself may get stuck into a local minimum instead of reaching the global minimum, our desired target. However, from the data displayed earlier the design space contains no other minimums, there are outliers on the contour plot but there is only one minimum seen. On the contrary there are many local maxima contained in the dataset.

After introducing the model, the cost function, the last bit to become familiar with is the performance of the model. Luckily enough the cost function parameter here can also tell the story of the performance of the developed model. Because this is a prediction model based on regression, the best method to evaluate the performance of this mode is none other than the MSE or the Cost function mentioned above. By optimizing this cost function, the performance metric is also maximized.

The evaluation of this model was chosen to be based on the MSE because of this model needed to predict very closely the needed settings of the controller. This is well suited for MSE because this metric sums all the distances of the predictions and true values, then the gradient descent method minimizes this distance. Taking all these things into consideration the distance of the predictions from the actual value is minimized leading to the closest possible prediction that can be acquired using this type of modeling.

6.3.3 Model Building

The actual linear regression model was built in python using 3rd party packages, and modules. These packages and modules included Pandas, Sci-Kit learn (SK learn), NumPy, Statsmodel, Seaborne, and Matplotlib. SK learn has all the parts needed to do the modeling, and cost optimization. After importing all the needed modules and packages, the generated simulation data that has been cleaned and prepped was split into training and testing data. The training and testing split were 80/20. 80% of the data was used for training, and 20 % was used for testing. A random seed of 42 was declared to keep the runs across sessions consistent. A small excerpt of the python code using the SK learn packages in appendix C-3 shown.

6.3.4 Model Performance

The initial model developed, without using Gradient Descent, performance was evaluated to establish a baseline. The model as expected performed poorly when not optimized correctly. The baseline performance of this model when unoptimized was calculated to have an MSE of 1.162. This was not as bad of an error at all; however, this is just the baseline for comparison. Substantial improvement can be made regarding improving the MSE score. Moving forward to optimizing the model and its performance by adding different techniques such as gradient descent, random sampling and bagging known as bootstrapping, then tuning model specific hyperparameters. A much more acceptable MSE was achieved. The improved and optimized model average MSE was calculated to be close to .3. This was an amazing improvement and was expected when optimizing for this type of measure. After optimizing the machine learning model, the desired performance for the control system was entered and the model prediction was made.

The main attribution to making such an improvement in the MSE was SK learns Stochastic Gradient Descent package combined with the Grid Search package. Gradient Descent has been covered previously, the grid search package does exactly what its name implies. Grid Search allows you to automatically iterate over a list of parameters while recording the model's performance, then at the end the model with the best score can be retrieved. This allows you to optimize the algorithm as much as possible to get the best performance possible. A snippet of the code implemented can in appendix C-4.

Culminating all these different techniques and practices the best model was selected. As such the model building for the predictive model was completed. The next step was to use the trained model to predict the optimal settings based on the performance that could be achieved with the controller.

6.4 Results & Conclusion

Before diving into the results of the model prediction a few things need to be reviewed. It should be noted that some performance must be sacrificed to achieve the best overall model. As seen in previous sections there are tradeoffs for each performance parameter, minimizing overshoot may increase the rise time, optimizing the rise time may increase the rise and settling time. From previous data analysis of this system in its performance the rise and settle time are very correlated so increasing one will increase the other in this specific model. These tradeoffs cannot be accounted for in this linear model, as all this model performs is a linear regressive prediction based on the performance parameters entered. This model cannot differentiate between parameters that cannot be made physically possible. Therefore, it is up to the engineer to use sound engineering judgement and principles to discern when specific performance points are not tangible.

Keeping performance restraints in mind the desired performance of the tuner was decided to prioritize rise time, overshoot, while settling time was of the least importance. Looking back to figures 40 there is no shortage of optimal rise and overshoot performance parameters within the acceptable range that is needed for this controller system. The desired point of operation for this rocket to obtain its commanded flight path angle as dictated by the inflight controller was set to be as close to the following, 10 seconds or less rise time, and less than 10% overshoot. This would lead to quick maneuvers as intended by the controller design in earlier sections. With the operation point specifics identified the prediction and comparison of the machine learning tuned controller and hand tuned controller could be compared.

6.4.1 Results & Comparisons

The hand tuned controller settings from earlier barely met the requirements needed for the performance of the system. Tuning the controller came with more complications than initially thought, lowering rise and overshoot time to a range within tolerance was soon found to be extremely hard to achieve. The repetitive process of simulating and changing the setting was difficult, especially not having an idea of the design space available. The settled upon hand tuned parameters and response graph can be seen below.

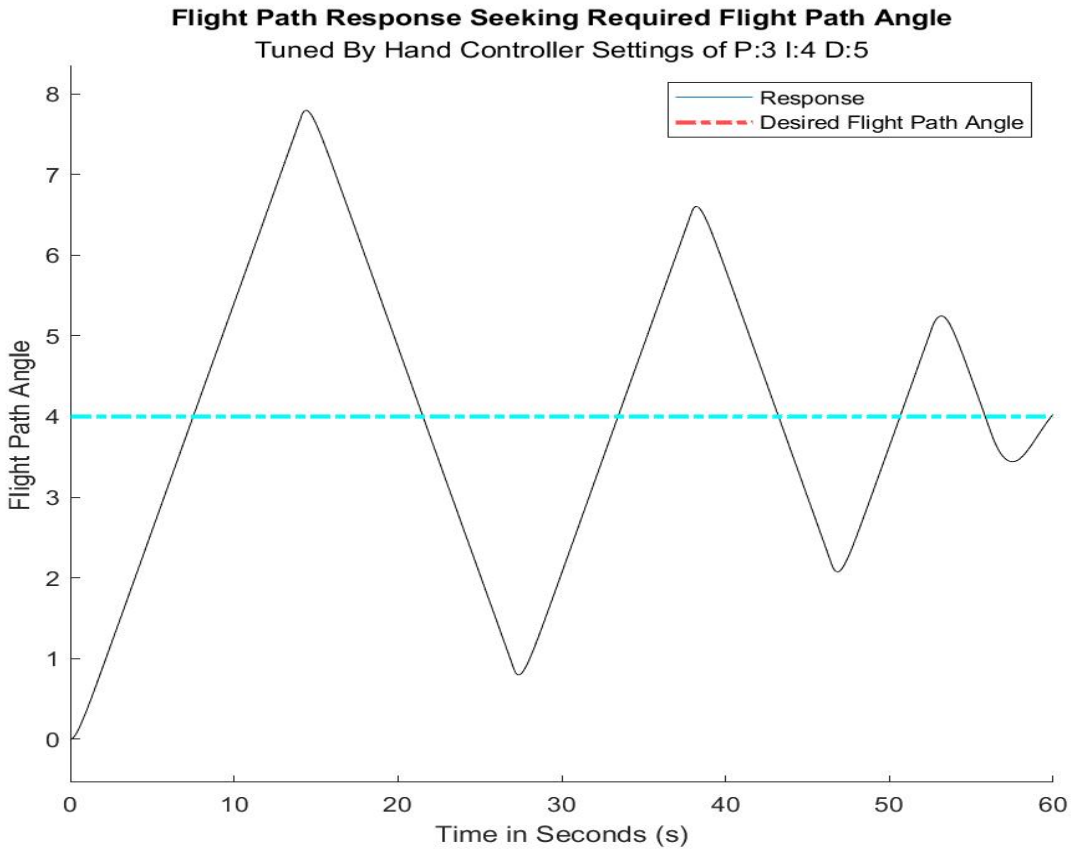


Figure 6.17: Hand tuned controller response

Although in theory this response is stable as it seeks back to the commanded point. The viability of this in a physical space is not the best performance that we can achieve. Simply by inspecting the design space and the response graphs shown there are far better performance points than this graph but due to hand tuning and the repetitiveness of it this point was settled upon. This point gave a satisfactory performance at the time of conception.

All of this was remedied with the machine learning model built, that then covered an expansive design space to encompass most viable solutions. Using the regressive model

developed and entering the wanted performance parameters the closest possible response that satisfies the performance requirements can be seen below.

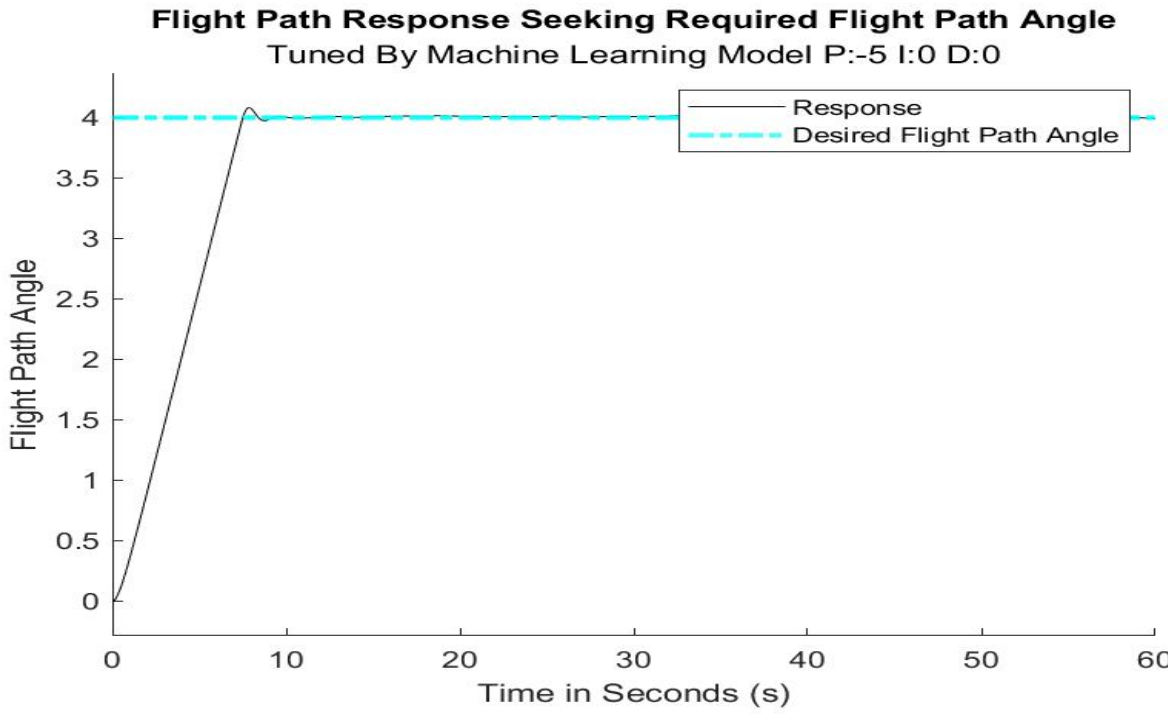


Figure 6.18: Machine learning tuned response

The machine learning tuned response has an obvious advantage over the hand tuned response. The machine learning model meets all the performance requirements and performs remarkably better than the hand tuned response. Not only did the overshoot keep minimized and the rise time meet requirements, but the settling time exceeded expectations. This was explored earlier the rise and settling time are highly correlated so minimizing the rise time was expected to have this effect. Next let us go even further and compare this to a general automatically tuned block using the built-in features of MATLAB.

Another viable option was to auto tune the controller settings in MATLAB. Using the tune option in the PID block itself, MATLAB will tune the controller to your liking. However, this is remarkably like using machine learning and other A.I models to do the tuning for you just as was implemented here. The automatically tuned response for the best optimized performance of overshoot, rise time, and settling time can be seen below.

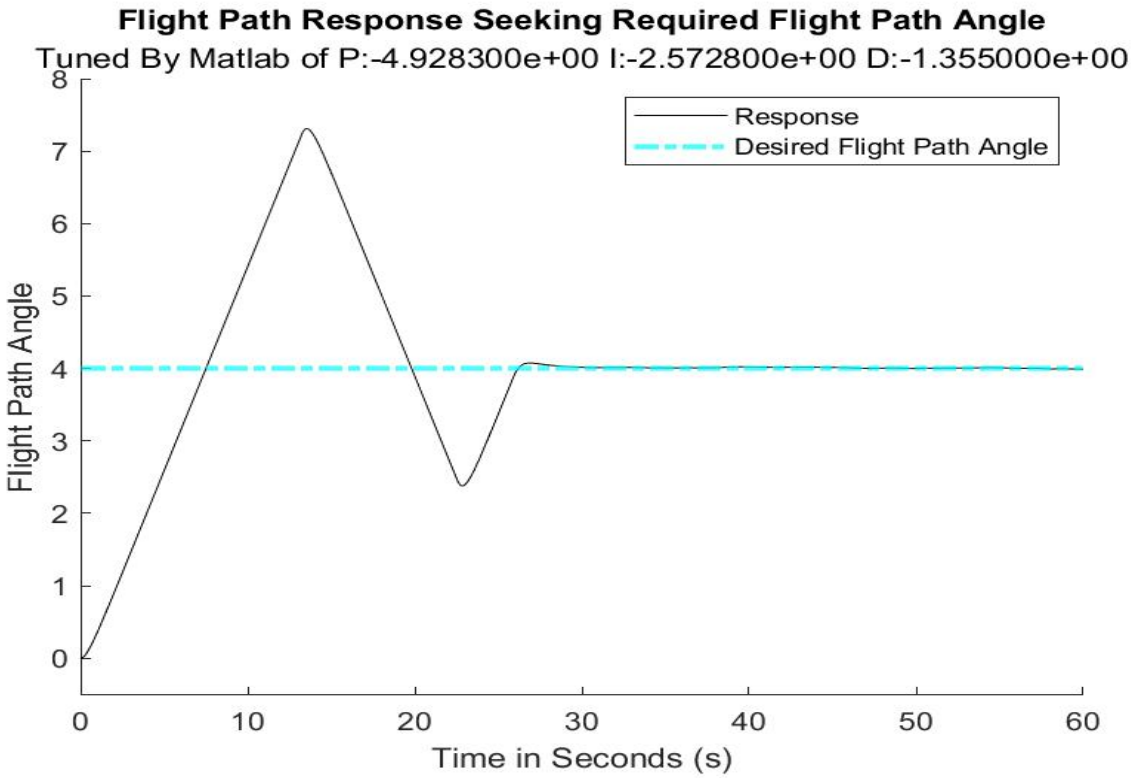


Figure 6.19: MATLAB tuned response of pid controller

Although the MATLAB tuned response is not optimal, the derived gains were found in under five minutes. These results are also only the preliminary response. An engineer can tune with respect to the optimization point even further to obtain a better response. This has its advantages over both hand tuning and machine learning tuning as well.

The best performance for this time seems to come from the made machine learning model. Although this is the best method for this specific situation, the difficulties, assumptions, and methods employed should also be considered. There may be cases where spanning an entire design space is not computationally efficient. This design concept was simplified into its bare components to prove a design solution exists and can be implemented into a physical system. In other situations, these simplifications may not exist, or a linear relationship may not exist at all.

Another consideration to keep in mind is the time spent developing the machine learning model itself. The data collection took the better half of three months while the model building took less than two weeks itself. Scaling this solution to bigger systems could exponentially lengthen the development and data collection time. When compared to both the hand and MATLAB tuning this is the obvious disadvantage to the machine learning method.

The last reason to consider when deciding to apply which method would be better for your proposed situation is the relationships between variables. Here the variables were very linear, the independent variables were not very strongly correlated, and the cost function was able to reach a global minimum easily. This may not be the case for every situation, physical

systems have large numbers of dimensions and complex relationships. These relationships may be much more complex than linear, or even polynomial, in these cases using a linear model will not suffice. As the complexity of the relationships of variables graduate so does the model complexity and run time. At some point using a supervised model may not even be viable, an unsupervised neural network may need to be used to evaluate the relationships between variables. As with all engineering practices the tradeoffs need to be well documented and considered before traversing a path to a solution.

Chapter 7-Conclusion and Future Work

In conclusion the proposed maneuverable control system was successfully designed, implemented, and met performance benchmarks. The Falcon 9 rocket control system designed here achieved a rise time of less than ten seconds, an overshoot of less than ten percent, and a comparably fast settle time. This will lead to a highly maneuverable rocket.

After the design of the maneuverable rocket with a PID control system. The established PID tuning status quo was challenged. While tuning the designed PID controller by hand, MATLAB's automatic controller, and linear regression model. The clear winner was the machine learning model tuned parameters. Although tuning by hand and MATLAB did give quicker results that were satisfiable, the machine learning tuner gave the best possible performance. This can be due to the specific nature of the tuned model versus the general algorithm used in MATLAB, and the not so accurate iterative methods used by hand tuning.

The drawbacks to the machine learning tuner however are; long development time, assumptions about data must be correct, a known design space must be known, and you need to have knowledge of software development with advanced programming skills. All these drawbacks can be quite costly when implemented via industry practices. Such as an engineer with advanced programming skills and knowledge of machine learning techniques coupled with an advanced aerospace engineering background would need to be compensated much more than a regular aerospace engineer. Therefore, the machine learning tuning of controllers would only be possibly advantageous depending upon the situation.

Overall, this project set out to propose a solution to make a maneuverable rocket in steady state flight. After this stage was completed, optimizing performance was the main goal, along the way different methods were compared. This eventually led to the conclusion that if applicable to the engineering application, a custom designed machine learning algorithm can provide the best tuning results.

7.1 Improvements

There are a few improvements that could be made to make a better performing controller, and machine learning tuner. These improvements are:

1. **More advanced control method:** A more advanced controller such as an LQG controller could in fact perform better than the implemented PID with filter. Although the PID controller meets the design parameters, and performance metrics. Setting cost functions for fuel, thrust, and other parameters in an LQG controller may obtain better function and optimization. This could be a point of improvement and design further in the future.

2. **Neural-Networks:** Along the same lines of thinking a Neural Network machine learning model is much more robust than the linear regressor used. Although the linear regressor

achieved satisfactory optimization, there were assumptions that had to be made and confirmed. If the data collected did not have the qualities that were shown, then a linear regressive model could not be implemented. Conversely a Neural Network exceeds looking into data that may not have any correlations or assumptions and finding those patterns. By implementing a Neural Network, the machine learning tuner could be more robust and apply to more situations than the flight conditions specified here. This could lead to faster tuning, over wider areas of the flight envelope, and better performance as seen in previous sections.

This project's work met its goal, and performance metrics. However, there can always be improvements made, with knowledge to gain and use further in the future. Here it is no different.

7.2 Possible Extension & Future Work

The next step after the completion of the computational modeling would be to prototype. A full-scale prototype of the Falcon 9 rocket would of course be cost inefficient, a scale model would be best. If the aerodynamic properties are similarly scaled and reproduced. Once the rocket body is prototype, the designed controller here could be implemented. Given the computational models are a success, flight testing in a real environment should also be a success.

However, if the prototyped rocket and control system behave unexpectedly, failure analysis would need to be done. After which it would be back to the computational model to pinpoint what went wrong, how to change it, and where fixes need to be implemented. Design is an iterative process that must be repeated. There is no difference with this design.

This design has moved from the conceptual phase, such as asking, defining, designing, and implementing a solution to the problem. To the prototyping, testing, and improving phase. There is much more work in the future that will need to be done to optimize this system. With the help of the machine learning algorithm designed, tuning should go much faster as shown in this work.

References

[1] Benson, T., “Examples of Controls,” Rocket Index Database, retrieved 12, February 2021. https://www.grc.nasa.gov/www/k-12/rocket/shortr.html

[2] Tewari, A., “*Advanced Control of Aircraft, Spacecraft, and Rockets*,” Vol.1, Wiley, New York, 2011.

[3] T. Chelaru and V. Pana, "Six Degree of Freedom Model for Testing Vehicle with Liquid Rocket Engine,” 9*th International Conference on Recent Advances in Space Technologies* RAST, July 2019, doi: 10.1109/RAST.2019.8767859.

[4] Bruns, K., Moore, M., Sto, S., "*Missile Datcom User's Manual*," McDonnel Douglas Missile Systems Company, Wright Patterson Airforce Base, 1991

[5] Williams, J. E., Vukelich, S. R., "*The USAF Stability and Control Datcom User's Manual*," The United States Air Force, 1979

[6] Haque A, Asrar, W, Omar A.A, Sulaeman E, and Ali J.S.M,“Comparison of Digital DATCOM and Wind Tunnel Data of a Winged Hybrid Airship’s Generic Model.” *Applied Mechanics and Materials*, 1st ed, Vol. 1, pp.33 -50, 2014

[7] Marco B, Jonathan M, and Bob B, "Modeling and Simulation of Flight Dynamics of Supersonic Inflatable Advanced Decelerator," *Proceedings, AIAA/AAS Astrodynamics Specialist Conference*, AIAA, August 2014, doi: 10.2514/6.2014-4454

[8] Juneja P.K., Sunori, S.K., Sharma, A., Joshi, V., and Bhasin, P., “A Review on Control System Applications in Industrial Processes,” *Proceedings, Institute of Physics Conference Series*, PIPCS, September 2021, doi: 10.1088/1757-899/1022/1/012010

[9] SpaceX, "*Falcon 9 Launch Vehicle Payload User's Guide,"* 1st ed., Space Exploration Technologies Corporation, 2008

[10] SpaceX, "*Falcon 9 User's Guide*," 2nd ed., Space Exploration Technologies Corporation, 2020

[11] Lustig M., "Modeling of Launch Vehicle during the Lift-off Phase in Atmosphere," Bachelors Thesis, Aeronautics Dept., Czech Technical University, Prague, Czechia, 2017

[12] Preet M., “Spacecraft and Aircraft Dynamics” *Lecture 9: 6-DOF Equations of Motion*, Tempe Arizona, 2021

[13] Preet M., “Spacecraft and Aircraft Dynamics” *Lecture 10: Linearized Equations of Motion*, Tempe Arizona, 2021

[14] Preet M, “Spacecraft and Aircraft Dynamics” *Lecture 11: Longitudinal Dynamics*, Tempe Arizona, 2021

[15] Guerrero A, Barranco A, and Conde D, "Scholar Commons Active Control Stabilization of High-Power Rocket,” Master’s Thesis, Mechanical Engineering Dept., Santa Clara University, Santa Clara, CA, 2018

[16] Caughey, D.A., "Introduction to Aircraft Stability and Control Course Notes for M&AE 5070," Ithaca, NY, 2011

[17] Giodano, A., and Levesque, A., "*Simulink Getting Started*,” 1st ed., MathWorks, 2021

[18] Gantmacher F.R., and Levin, L.M., "Equations of Motion of a Rocket," National Advisory Committee for Aeronautics, Fort Belvoir, Virginia, April 1950

[19] Chusilp, P., Charubhun, W., and Nutkumhang, N., “A Comparative Study on 6-DOF Trajectory Simulation of a Short-Range Rocket using Aerodynamic Coefficients from Experiments and Missile DATCOM,” International Conference on Mechanical Engineering, ICM, October 2011, doi: 10.5772/cpwm.82292

[20] Kisabo, A.B, Adebimpe A.F, and Okwo, O.C., "State-Space Modelling of a Rocket for Optimal Control System Design," *Journal of Aircraft and Spacecraft Technology*, Vol. 3, January 2019, pp. 128-137, doi: 10.5772/intechopen.82292

[21] Bellis, M., “Rocket Stability and Flight Control,” ThoughtCo [online article], URL: https://www.thoughtco.com/rocket-stability-and-flight-control-systems-4070617

[22] Kovalenko, N. D., Sheptun, U. D., Kovalenko, T. A., and Strelnikov, G. A., “The new concept of thrust vector control for rocket engine,” Technical Mechanics, October 2016

[23] Choi H., and Bang, H., "*An adaptive control approach to the attitude control of a flexible rocket*," Control Engineering Practice, Vol. 8, No. 9, 2000, pp. 1003-1010

[24] Nanjangud A, and Eke F.O., "Lagrange’s Equations for Rocket-Type Variable Mass Systems," *Proceedings*, *International Review of Aerospace Engineering (IREASE)*, Vol. 5, October 2012, pp. 256, doi: 10.15866/irease.v5i5.15613

[25] Odwyer A., “PI and PID controller tuning rules: an overview and personal perspective,” *Proceedings*, *IET Irish Signals and Systems Conference)*, Dublin, Ireland, ISSC, June 2006, doi: 10.21427/ekry-ap03

[26] Schneider A., Hommel G., and Blettner, M. “*Linear regression analysis: part 14 of a series on evaluation of scientific publications,”* Deutsches Arzteblatt international, June 2005, pp. 776–782.

[27] Rougier J. “Ensemble Averaging and Mean Squared Error,” *Journal of Climate Science*, Nov 2021, pp. 8865–8870 doi: 10.1175/JCLI-D-16-0012.1f

[28] Chen, S. “Introduction to Machine Learning,” [online PowerPoint slides], San Jose, Ca, 2021

Appendix

Appendix A - MATLAB Scripts

A.1 Importing Digital Datcom Aerodynamic Coefficients

```
aero = datcomimport('referencemach.dat');

aero1 = datcomimport('for006mach1.dat');

aero2 = datcomimport('mach2.dat');

aero3 = datcomimport('mach3.5.dat');

aero4 = datcomimport('mach3.dat');

aero5 = datcomimport('mach5.dat');
```

A.2 Initial Conditions for Earth Frame Data

```
%%% Earth Data

omega_Earth = 7.2921e-5;      % angular velocity of Earth [rad/s]

phi_c_init = 0;           % initial geocentric latitude [deg]
lambda_c_init = 0;        % initial geocentric longitude [deg]
r_pos_intial = [6378137; 0; 0]; % initial LV position vector r [m]
```

A.3 Falcon 9 Rocket Properties

```
%Rocket Properties
area=21; % m^2

Cg=26.52; % Center of Pressure Calculated

Cp=Cg; % Cp set to Cg for early preliminary results not used in the final model

Mass=570000; %Kg

Mass_inverse= 1/Mass; %Kg

Length=70; % Meters

Ixx=1.4*10^5; %Principal axis of inertia

Iyy=3.2*10^7; %Principal axis of inertia

Izz=3.2*10^7; %Principal axis of inertia

Inertial=[Ixx;Iyy;Izz]; %Inertial Matrix

J=Inertial; % Moment of Inertial Matrix Used in Simulink

Sref=21; %Meters^2 Aerodynamic Ref Area

Lref=3.24; %Meters

burn=162; %seconds

gimbal_pos=[-21 0 0]; %meters
```

A.4 Simulink initial Conditions for 6dof Flight Block

```
initial_pos=[6378137; 0; 0];   % initial LV position vector r [m]


intial_omega=[.0000727;0;0]; % initial angular velocity


initial_euler=[0;0;0]; %rocket at rest
```

initial_velocity=[0;0;0]; %Rocket at rest

A.5 Frame Transformations

% Initialization of transformation matrices.

EGtoB = eye(3);

ECFtoEG = ECF2EG(phi_c_init, lambda_c_init);

ECItoECF = eye (3);

A.6 Falcon 9 Engine Data

```
% Merlin 1D engine (9x) from Falcon 9 used as a reference
engines = 9;              % 9 Engines are used
Isp = 282;                % specific impulse at sea level [s]
ve = 251;                 % exit velocity [m/s]
mdot = 280;               % mass flow rate [kg/s]
pe = 9e5;                 % exit pressure [Pa]
Ae = 0.97;                % nozzle exit area [m^2]
```

A.7 Aerodynamic Coefficients - All estimated data due to lack of published data used for Falcon 9 were used in steady state flight conditions for ease of state estimation.

```
%Steady state flight at Mach 3.5, 1200 m/s

C_A=.2; %Axial Coefficients

C_A=.05; %Axial Coefficients of alpha

C_A=.05; %Axial Coefficients of beta

CS=0; % Side force Coefficient

CS_b=.9; %Side force coefficient of beta

CN=0; %Normal force coefficient

CN_a=.9; %Normal force coefficient of alpha

CM_rp=.8; %Roll moment for p

CM_Pa=.05; %Pitch moment coefficient for

CM_Pq=.8;%Pitch moment coefficient for q

CM_YB=.05;   %yaw moment coefficient for beta

CM_Yr=.8;   %yaw moment coefficient for r

C_A_0 = 0.2;

C_A_alpha = 0.05;

C_A_beta = 0.05;

C_N_0 = 0;

C_N_alpha = 0.9;

C_S_0 = 0;

C_S_beta = 0.9;

C_Mr_0 = 0;

C_Mp_alpha = 0.09;

C_My_beta = 0.05;

C_Mr_p = 0.8;

C_Mp_q = 0.8;

C_My_r = 0.8;
```

A.8 State Space from Simulink model to calculate Poles.

```
[A, B, C, D] = linmod("manueverablerocket");
sys=ss(A,B,C,D);
eig(A);
%% Design Space For PID Controller, and Initializations
```

A.9 Simulation Space Initialization

```
v=linspace(-7,3,11);
Sim_space=nmultichoosek(v,3);
PID=array2table (Sim_space,'VariableNames',{'P', 'I', 'D'});
P_controller=Sim_space(:,1);
I_controller=Sim_space(:,2);
D_controller=Sim_space(:,3);
Overshoot=zeros(size(Sim_space));
rise_time=zeros (size (Sim_space,1));
settling_time=zeros (size (Sim_space,1));
Overshoot=Overshoot (:1);
rise_time=rise_time(:,1);
settling_time=settling_time(:,1);
peak_mat=zeros(size(rise_time));
```

A.10 Simulink model Automation Simulation

```
%model=('missle_modle_with_noise.slx');
%open(model)
tolerance = .05;
%% Looping Simulations and Logging Data
for repeat = 1: len(Sim_space)
```

```
 %if repeat==30 || repeat ==60 || repeat == 90|| repeat==150 || repeat== 180 || repeat == 210|| repeat == 240 % Used For evolutions of plots at every 30 intervals

P_sim=P_controller(repeat);

I_sim=I_controller(repeat);

D_sim=D_controller(repeat);

model=('missle_modle_with_noise.slx');

sim(model)

Flight_path=Flight_Path_angle.signals.values(:,1);

FPA_cmmnd=Flight_Path_angle.signals.values(1,2);

upper_band=FPA_cmmnd+(FPA_cmmnd*tolerance);

lower_band=FPA_cmmnd - (FPA_cmmnd*tolerance);


Time=Flight_Path_angle.time;

peak=max(abs(Flight_path));

peak_mat(repeat)=peak;

over= ((peak - FPA_cmmnd) / FPA_cmmnd) *100;

Overshoot(repeat)= over;
```

A.11 Logging the Rise and Settle Time for Every Simulation

```
%%% Rise Time

rise_time_index=find(Flight_path==peak);

if isempty(rise_time_index)

   rise_time(repeat)= NaN;

else

rise_time(repeat)=Time(rise_time_index);

end

%% Settling Time
```

```
%for I=1: length(Flight_path)
%    If Flight_path(i) > 0
%       If (Flight_path(i) >= lower_band & Flight_path(i) <= upper_band)
%          settle_point_data(i)=Flight_path(i);
%       Else
%          settle_point_data(i)=NaN;
%       End
%Else
%   Continue
%End
```

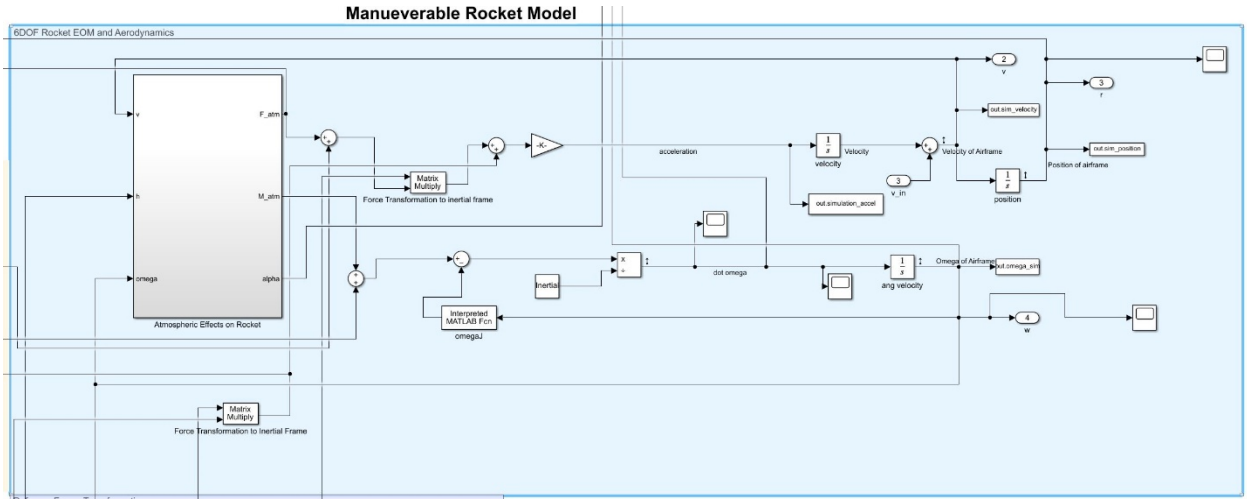
A.12 Saving and Graphing of all Simulation Variables

```
%% Simulation Information and Plots
    fprintf("Run number %d:'\n' ", repeat)
    fprintf("P_sim for this run is %d: '\n' ", P_sim)
    fprintf("I_sim for this run is %d:'\n' ",I_sim)
    fprintf("D_sim for this run is %d:'\n' ",D_sim)
   filename=sprintf('Run_number%d',repeat);
  %% Evolution Plots
    figure (1);
    hold on
    Y= Flight_path;
    X=Time;
    plot (X, Y)
    yl1=yline(upper_band,'-.m','LineWidth', .5);
    yl2=yline(lower_band,'-.m','LineWidth', .5);
    yl=yline(4,'-. r','LineWidth',2);
    title ('Flight Path Response Seeking Required Flight Path Angle')
```

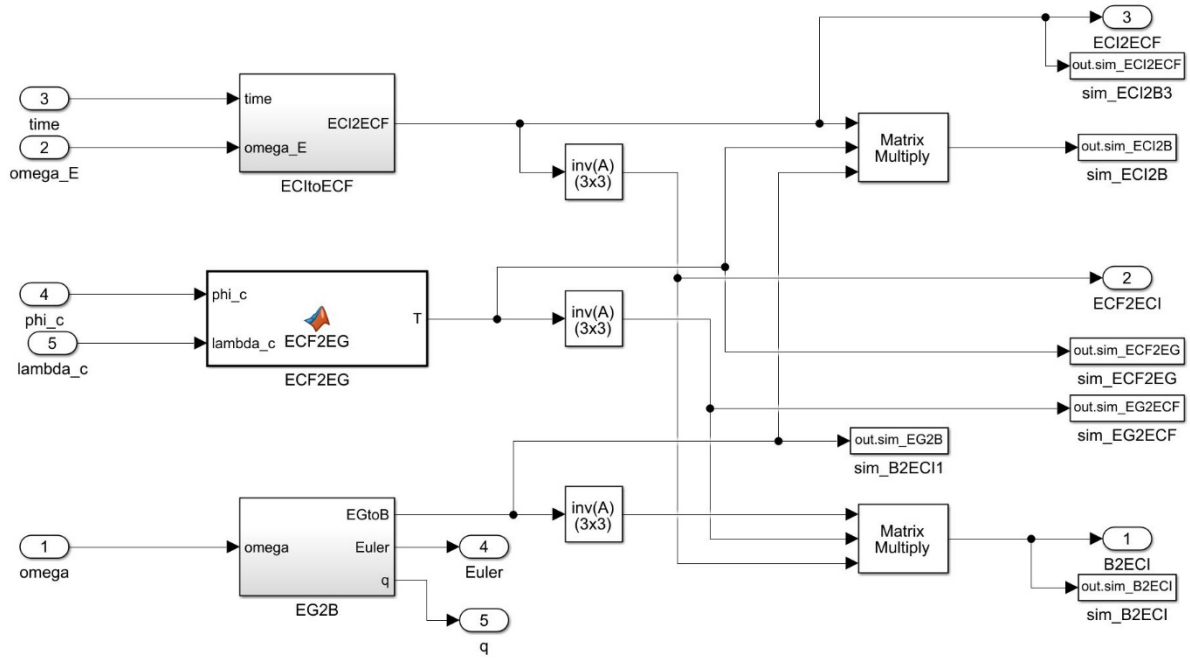
```
        subtitle(sprintf('Up to Simulation Number: %d With Control Variables P value: %d I value: %d D value: %d',repeat, P_sim,I_sim,D_sim))
        ylabel('Flight Path Angle')
        xlabel('Time in Seconds (s)')
        legend ('Response', 'Upper Error Band', 'Lower Error Band', 'Desired Flight Path Angle')
        ylim padded
        %saveas(gcf,filename,'pdf');
        hold off
    %Else
    %    Continue
     %End
end
%Training_param=[Overshoot,rise_time,settling_time];
%% Write Design Space For Machine Learning Use
%writematrix(Sim_space,'Design_Space_with_noise.xls')
%writematrix(Training_param,'Regular_data.xls').
```

Appendix B - Simulink Diagrams

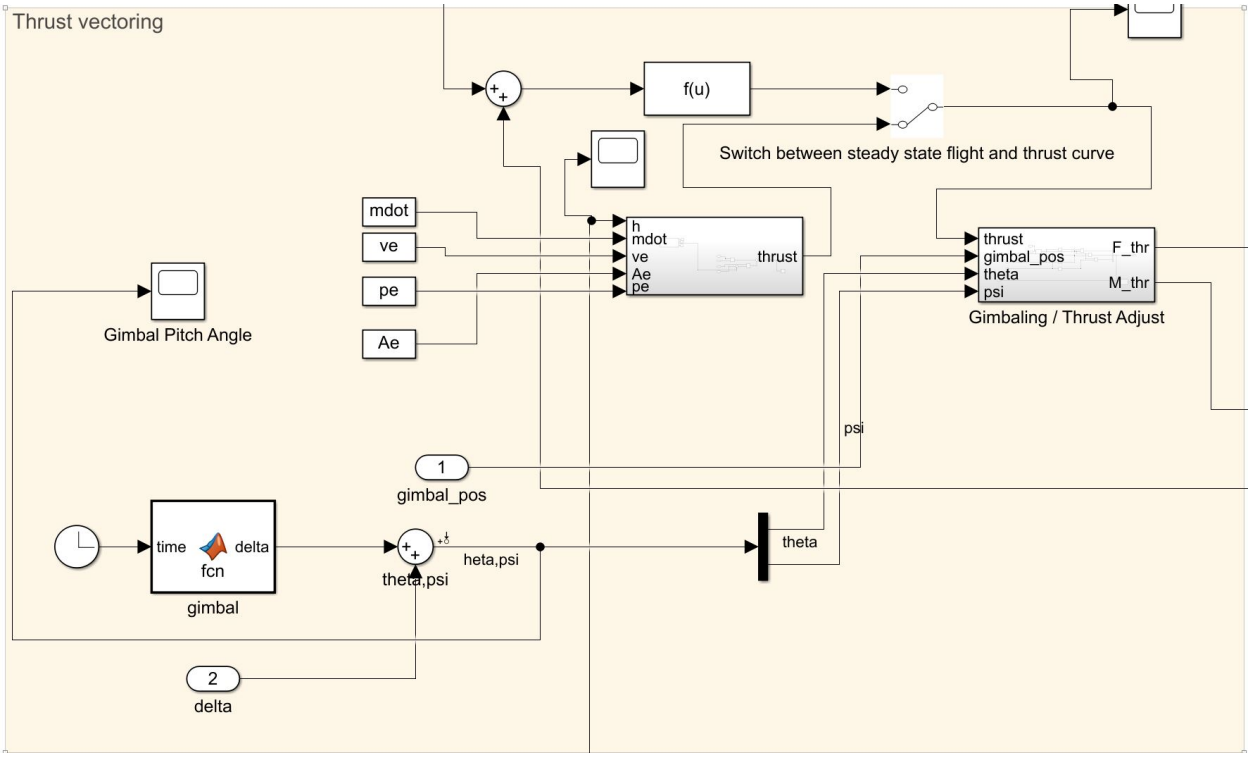
B.1 Equations of Motion Subsystem



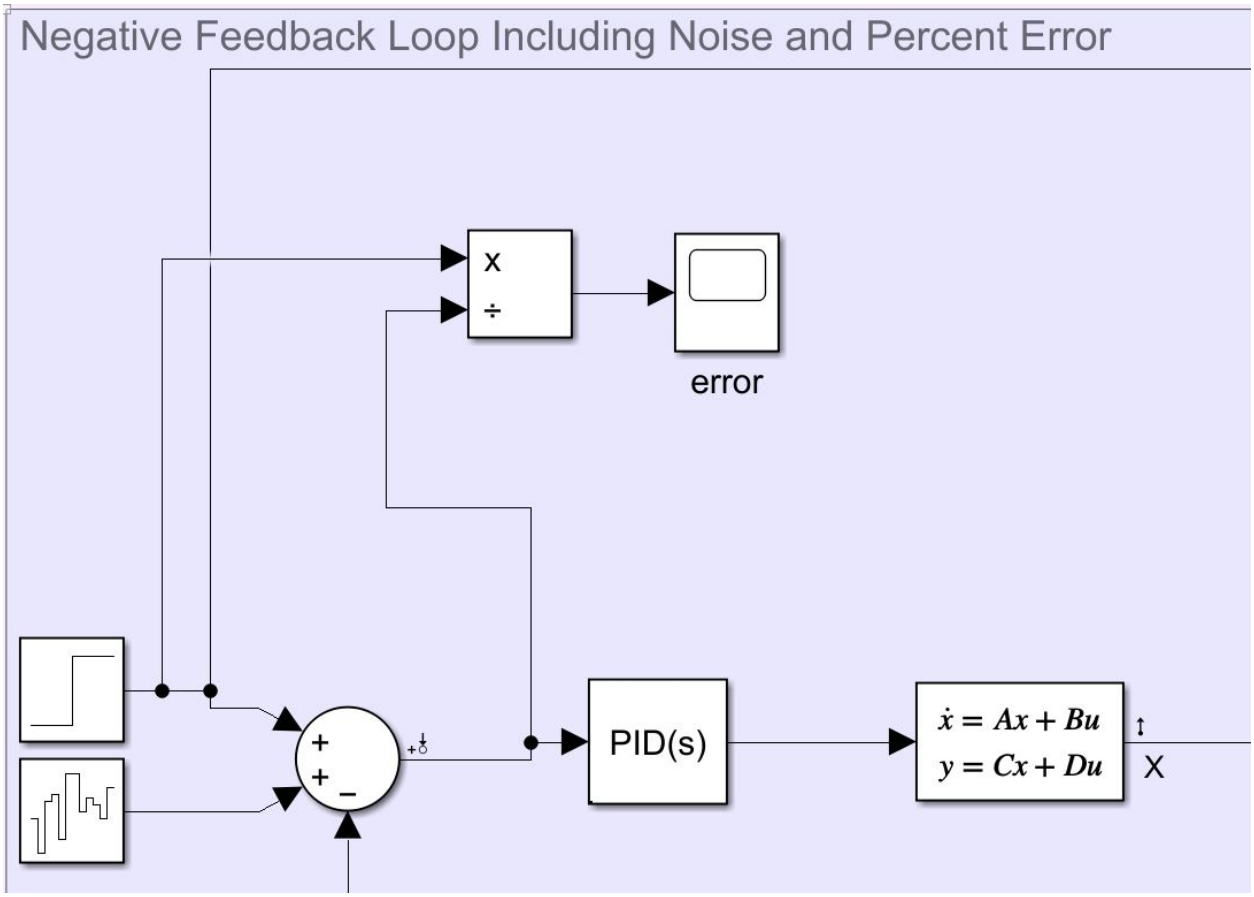
B.2 Transformation Between Frames



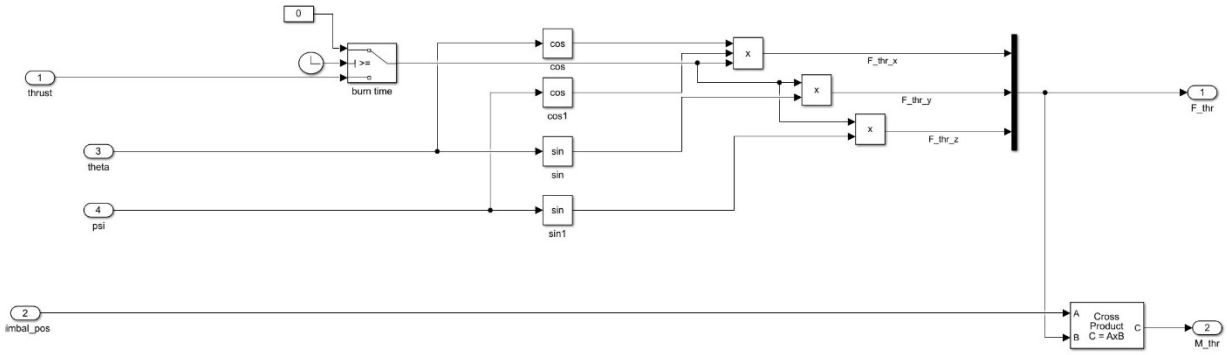
B.3 Thrust Vectoring Subsystem



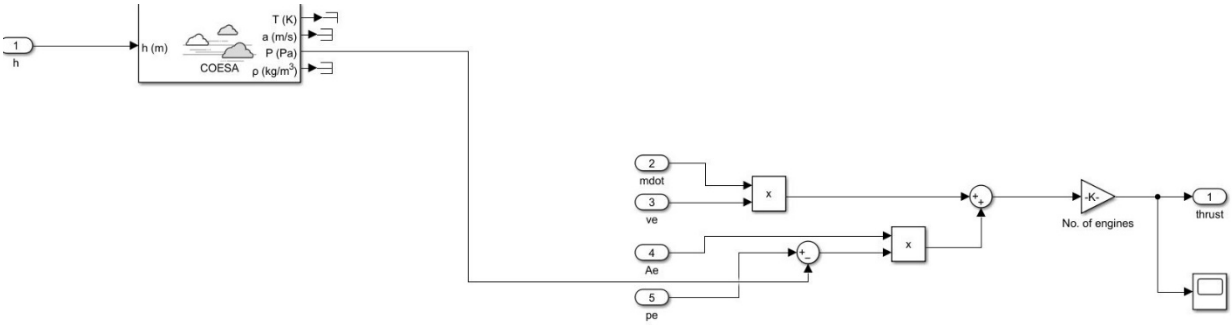
B.4 PID Control System Feedback Loop



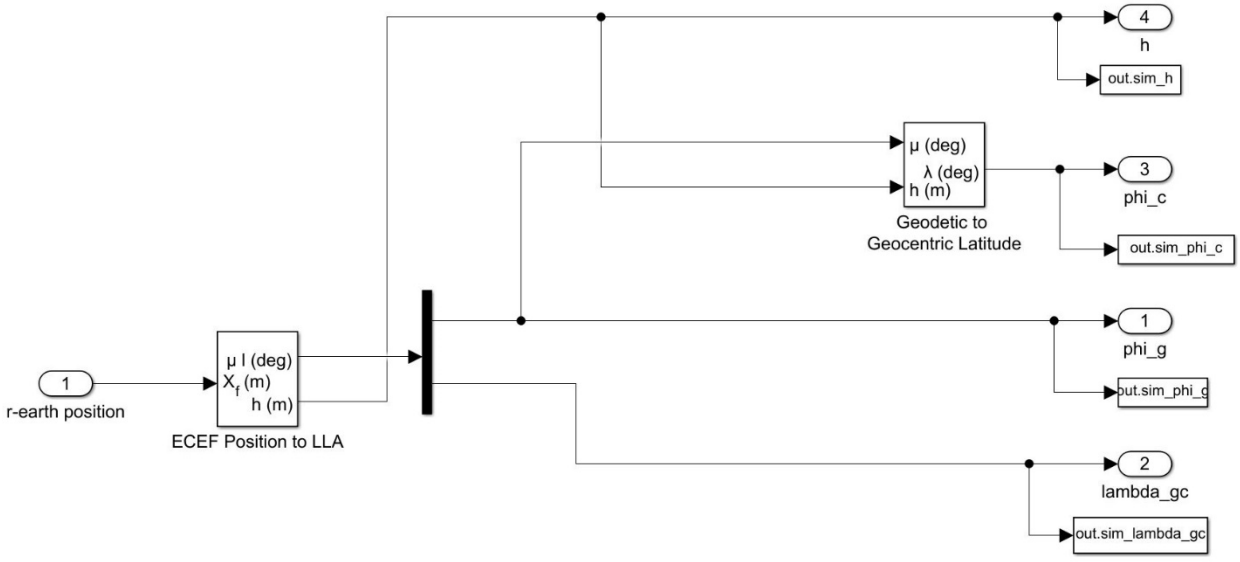
B.5 Thrust Vectoring Moment Calculations



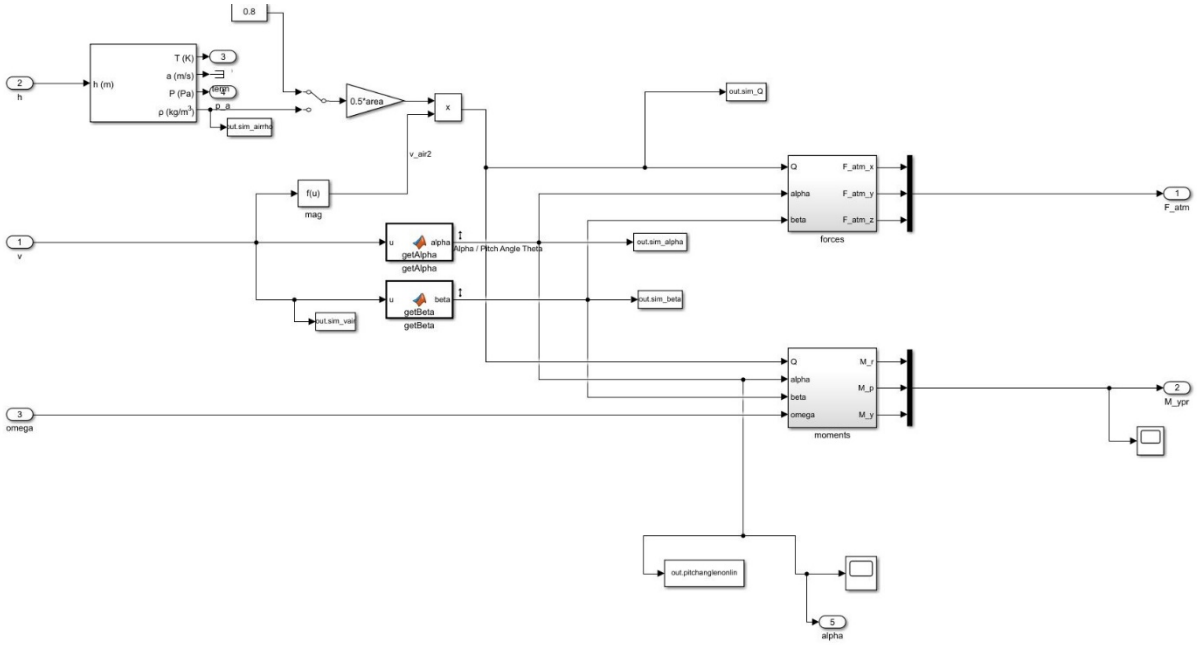
B.6 Atmospheric Subsystem Model



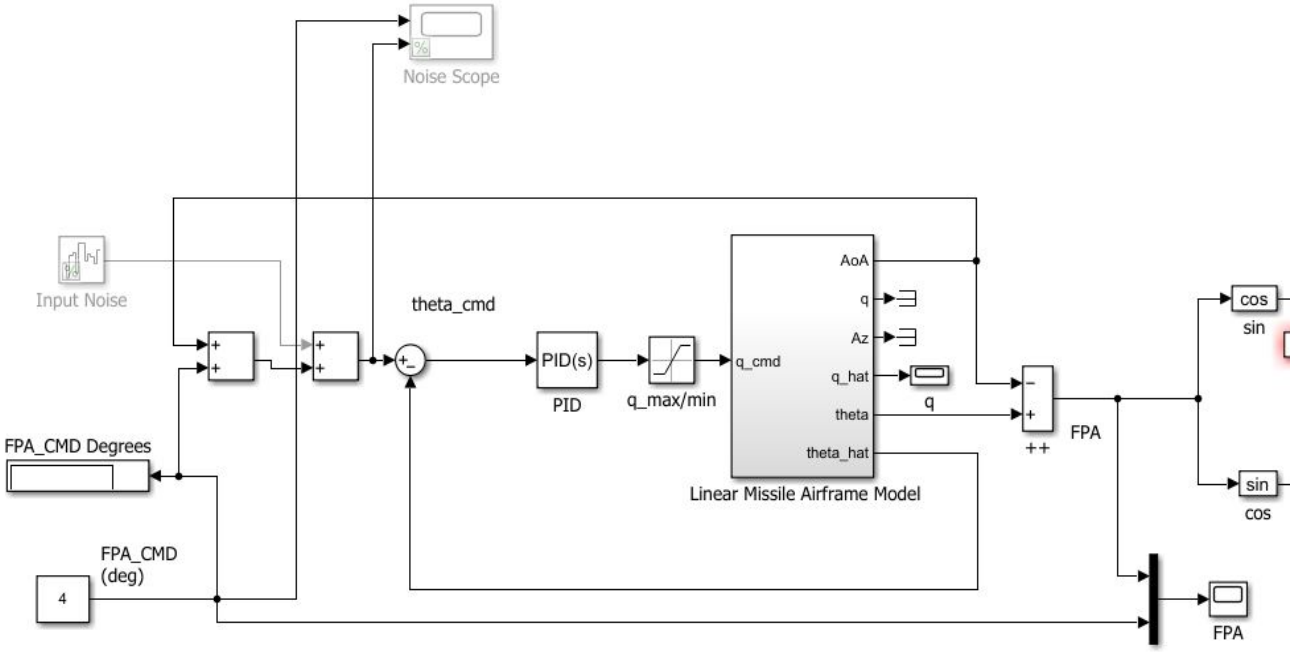
B.7 Earth Position Calculation Subsystem



B.8 Aerodynamic Forces Calculations Subsystem



B.9 Simplified Linear Diagram



Appendix C – Python Scripts

C.1 Differences Between Noise and No Noise Plots

```
import seaborn as sns

import matplotlib.pyplot as plt

import pandas as pd

noise_df=pd.read_excel(r'C:\Users\broth\Documents\Master's Degree\Finished Results pics\noise_data.xls',names=['overshoot','rise_time','settling_time'])

no_noise_df=pd.read_excel(r'C:\Users\broth\Documents\Master's Degree\Finished Results pics\regular_data.xls',names=['overshoot','rise_time','settling_time'])

noise_df.dropna(inplace=True)

no_noise_df.dropna(inplace=True)

overshoot_diff=((abs(noise_df.overshoot - no_noise_df.overshoot)) / noise_df.overshoot) *100

rise_diff=((abs(noise_df.rise_time - no_noise_df.rise_time)) / noise_df.rise_time) *100

settle_diff=((abs(noise_df.settling_time - no_noise_df.settling_time)) / noise_df.settling_time) *100

fig=plt.gcf()

plt.scatter(overshoot_diff.index,overshoot_diff)

plt.xlabel('Simulation Number')

plt.ylabel('Percent Difference')

plt.title('Overshoot-Percent Difference between Noise and No Noise Responses')

fig.savefig('Overshoot Difference.jpg',dpi=800)

plt.show()

fig=plt.gcf()

plt.scatter(rise_diff.index,rise_diff)

plt.xlabel('Simulation Number')

plt.ylabel('Percent Difference')

plt.title('Rise Time-Percent Difference between Noise and No Noise Responses')

fig.savefig('Rise Time Difference.jpg',dpi=800)
```

```
plt.show()
fig=plt.gcf()
plt.scatter(settle_diff.index,settle_diff)
plt.xlabel('Simulation Number')
plt.ylabel('Percent Difference')
plt.title('Settle Time-Percent Difference between Noise and No Noise Responses')
fig.savefig('Settle Time Difference.jpg',dpi=800)
plt.show()
```

C.2 Design Space

```
""" Import all the modules needed to make a model, import data, visualize, and plot data points. This also import a lot of the statistics that
will be needed to analyze our model and see how well it performs in a machine learning sense.
"""
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn import linear_model as lm
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error as mse
import statsmodels.api as sm
from statsmodels.stats.anova import anova_lm
from statsmodels.formula.api import ols
df_pid=pd.read_excel("design_space.xls")
print(f' Number of missing or null values are: {df_pid.isnull().sum()}')
print(f'The information about this data is: {df_pid.info()}')
# Formatting Column Names
```

```
df_pid.columns=['p','i','d']

print(f'Now the columns look like this: \n{df_pid.columns} ')

print(f'Check the first 5 rows of the data set {df_pid.head(5)}')

print(f'Description of data is: {df_pid.describe()}')

print(f'The shape of the Data set is: {df_pid.shape}')

# Casting Data frame as float

df_pid=df_pid.astype(float)

print(f'Now the info about the data is: \n {df_pid.info()}')
```

C.3 Correlation Studies

```
""" Correlation study between variables. Using The values for P I D as dependent variables and the other three variables as independent

variables

"""

sns.pairplot(df_pid)

plt.show()

correlation=df_pid.corr()

print(correlation)

correlation.style.background_gradient(cmap='coolwarm')
```

C.3 Visualization of PID Variables

```
""" Next is a visualition to describe the dataset and how it is about."""

visual=df_pid.copy(deep=True)


# We are assigning labels either 1 or 0 for 1 = high, and 0 = low. This corresponds to values of the

#P, I, or D that are greater than the average measurement

visual['p'] = [ 1 if p >= -4.4 else 0 for p in list(visual['p'].values)]

visual['i'] = [ 1 if i >= -2 else 0 for i in list(visual['i']. values)]

visual['d'] = [ 1 if d >= .5 else 0 for d in list(visual['d']. values)]
```

```
# Now that these values have been assigned labels, we can create a box plot to see how often a high or

# Low value occurs within the data set

sns.countplot(x="p", data= visual)

plt.xlabel("High = 1, Low = 0")

plt.ylabel("Count")

plt.title("Count plot of High and Low Value Occurrences for the P parameter")

sns.countplot(x="i", data= visual)

plt.xlabel("High = 1, Low = 0")

plt.ylabel("Count")

plt.title("Count plot of High and Low Value Occurrences for the I parameter")

sns.countplot(x="d", data= visual)

plt.xlabel("High = 1, Low = 0")

plt.ylabel("Count")

plt.title("Count plot of High and Low Value Occurrences for the D parameter")
```

C.3 Model Generation

Model Generation

```
P_multiple_regression = lm.LinearRegression()
I_multiple_regression = lm.LinearRegression()
D_multiple_regression = lm.LinearRegression()

learning_data=df_regular.copy(deep=True)

#independat variables
x=learning_data[["Overshoot(%)","Rise_Time_(s)","Settling_Time_(s)"]]

#Dependant Variable 1 - P
y_p=learning_data[['p']]

#Dependant Variable 1 - I
y_i=learning_data[['i']]

#Dependant Variable 1 - D
y_d=learning_data[['d']]

# Model for P tuner
x_train_p, x_test_p, y_train_p, y_test_p = train_test_split(x,y_p, test_size=.2, random_state=42)
P_multiple_regression.fit(x_train_p,y_train_p)
P_pred=P_multiple_regression.predict(x_test_p)

#Model for I tuner
x_train_i, x_test_i, y_train_i, y_test_i = train_test_split(x,y_i, test_size=.2, random_state=42)
I_multiple_regression.fit(x_train_i,y_train_i)
I_pred=I_multiple_regression.predict(x_test_i)

#Model for D tuner
x_train_d, x_test_d, y_train_d, y_test_d = train_test_split(x,y_d, test_size=.2, random_state=42)
D_multiple_regression.fit(x_train_d,y_train_d)
D_pred=D_multiple_regression.predict(x_test_d)
```

C.4 Parameter Generation

```
params={'alpha': 10.0 ** -np.arange(1, 7),
    'penalty': ['l2', 'l1', 'elasticnet'],
    'learning_rate': ['constant', 'optimal', 'invscaling'],}
#Grid Search for the best parameters
clf_p = GridSearchCV(model, params)
clf_i= GridSearchCV(model, params)
clf_d=GridSearchCV(model, params)
#Fit for the best parameters
clf_p.fit(X_train_p, y_train_p)
clf_i.fit(X_train_i, y_train_i)
clf_d.fit(X_train_d, y_train_d)

#Print out the best score
print(f'Best score for P model:  + {str(clf_p.best_score_)}')

print(f'Best score for the I model: {str(clf_i.best_score_)}')

print(f'Best score for the D model: {str(clf_d.best_score_)}')

# Best Models

print(f'Best model for P is {str(clf_p.best_estimator_)}')

print(f'Best model for I is {str(clf_i.best_estimator_)}')

print(f'Best model for D is {str(clf_d.best_estimator_)}')
```