The Design and Testing of a Thrust Vector Control System for Model Rockets

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

Surya Balahanumappa Shivakumar

December 2025

approved by

Dr. Periklis E. Papadopoulos
Faculty Advisor



© 2025
Surya B. Shivakumar
ALL RIGHTS RESERVED

ABSTRACT

The Design and Testing of a Thrust Vector Control System for Model Rockets

Surya B. Shivakumar

This project presents the design, simulation, and flight testing of a thrust vector controlled (TVC) model rocket capable of maintaining a stabilized vertical ascent without the use of aerodynamic fins. A dual axis gimbal integrated inertial sensors, and a PID base guidance, navigation, and control (GNC) algorithm were all implemented on the STARLIGHT flight computer. MATLAB/Simulink was used to model and compare PID, LQR, and LQG control responses, while iterative hardware testing refined the control architecture and validated its ability to maintain vertical flight under realistic conditions. Successful launches demonstrated the feasibility of small scale TVC systems, while also revealing opportunities for improvement in areas such as sensor noise mitigation and recovery system reliability.

Acknowledgments

I would like to thank Professor Periklis Papadopoulos in his help guiding me throughout this project. At the beginning of this project my advisor left the administration and Professor Papadopoulos step in to fill that gap. Throughout this project he has been nothing but supportive and given me crucial advice in helping me attain my goals. One of the biggest ones was using Black Rock, Nevada as a launch site. I was looking near SJSU and the Bay Area to launch but through countless rejections from local and county Parks and Recreation Departments I got anxious if I was able to launch. Professor Papadopoulos's eased this anxiety and suggested Black Rock, Nevada as a launch site; a place that slipped my memory. He also helped in a technical capacity when it came to debugging hardware issues that I faced as this was my first time working with Python and Raspberry Pi microcontrollers. All in all, Professor Papadopoulos and the SJSU Aerospace faculty provided a great avenue to showcase what I had learned under there tutelage.

Table of Contents

1. Introduction 1

1.1 Motivation 1

1.2 Literature Review 1

1.2.1 Guidance, Navigation, and Control Algorithms for Vertical Landing Rockets 1

1.2.2 Bridging the Gap Between Standard VTVL Rockets to TVC Rockets 6

1.2.3 Control Systems 7

1.2.3.1 PID Controller 8

1.2.3.2 Linear Quadratic Regulator Control System 9

1.2.3.3 Linear Quadratic Gaussian Control System 10

1.3 Proposal 12

1.4 Methodology 12

1.4.1 Simulations 12

1.4.2 Rocket Construction and Test Stands 13

1.4.3 Flight Testing and Evaluation 13

2. Rocket Body 14

2.1 General Rocket Design 14

2.2 Solid Rocket Motors and Altitude Simulations 15

2.3 Flight Computer Housing 16

2.4 Gimbal Design 17

3. Flight Computer Avionics 18

3.1 Central Processing Unit 19

3.2 Inertial Measurement Unit (IMU) 20

3.3 Barometer 21

3.4 Logging and Receiving Flight Data 21

3.5 PCB Layout and Design 22

3.6 Flight Cycle 25

4. Rocket Dynamics 27

4.1 Reference Frames 27

4.2 Euler's Rigid Body Equations and Dynamics 29
4.3 Pitch Axis State Representation....31
5. MATLAB System Simulations 34
5.1 System Variables and Physical Parameters....34
5.2 Open Loop Response 35
5.3 MATLAB 3DOF Rigid Body Model 36
5.4 PID Transfer Function Response 41
5.5 LQR Response 43
5.6 LQG Response 45
6. Python Embedded Flight Control Architecture....47
6.1 System Variables and Physical Parameters....47
6.1.1 Guidance, Navigation, and Control Algorithms for Vertical Landing Rockets 48
6.2 Open Loop Response 49
6.3 MATLAB 3DOF Rigid Body Model 49
6.4 PID Transfer Function Response 50
6.4.1 IMU Warm Up and Gyroscope Bias Estimation....50
6.4.2 Parachute and Motor Ignition Pin Initialization....51
6.4.3 Servo Initialization and Gimbal Configuration....51
6.4.4 Logging System Setup 51
6.4.5 Static Pitch and Roll Calibration....51
6.4.6 Launch and Timing Structure....52
6.4.7 Real Time Attitude and Pressure Measurements 52
6.4.8 Low Pass Filtering of Attitude Estimates 52
6.4.9 PID Computation Using Gyro Derivatives and Integral Terms....53
6.4.10 Mapping PID Outputs to Servo PWM Commands....53
6.4.11 Logging in Real Time and Periodic Flush 53
6.4.12 Parachute Deployment Logic....53
6.4.13 Post Flight Data Collection 54
7. Launch Results and Discussion 55
7.1 Launch 1 – October 24, 2025....55
7.2 Launch 2 – November 2, 2025....57

7.3 Launch 3 – November 9, 2025 63
7.4 Launch 4 – November 22, 2025 69
7.5 Launch 5 – November 22, 2025 71
7.6 Summary of Launch Results 79
8. Conclusion 81
References 83
Appendix A - MATLAB Simulation Code for TVC Controllers 85
Appendix B - Python 3 TVC PID Simulation 91
Appendix C - MicroPython Main Rocket Launch Code 94

List of Figures

Figure 1.1 - Vertical landing rocket method [3] 2

Figure 1.2 - Optimal control problem diagram [3] 2

Figure 1.3 Baseline attitude LGQ control and LQ control with reinforcement learning plots in tracking the reference yaw angle [2] 3

Figure 1.4 Trajectory control architecture that includes the attitude control architecture [2] 3

Figure 1.5 XOY and XOZ planes of rocket through vertical landing [5] 4

Figure 1.6 - First stage guidance cycle for vertical landing rocket flowchart [5] 5

Figure 1.7 - Trajectory optimization algorithm based on MPC framework [4] 5

Figure 1.8 - Force breakdown of a water rocket [6] 6

Figure 1.9 - Finless rocket model [7] 7

Figure 1.10 - PID control system model [9] 8

Figure 1.11 - LQR control system model [14] 10

Figure 1.12 - LQG control system model [15] 11

Figure 2.1 - General rocket design 14

Figure 2.2 - Thrust curve for Estes F15 motor [16] 15

Figure 2.3 - Rocket altitude simulation 16

Figure 2.4 - Flight computer housing for STARLIGHT flight computer 16

Figure 2.5 - Gimbal design with servo motors [17] 17

Figure 3.1 - STARLIGHT model rocket flight computer 18

Figure 3.2 - RP2040 dual core MCU and port schematic [20] 19

Figure 3.3 - ICM-42605 pin out diagram [21] 20

Figure 3.4 - BMP388 block diagram [20] 21

Figure 3.5 - PCB layout of W25Q128 QSPI flash device [18] 22

Figure 3.6 - PCB Layout of TLV76733 linear regulator [18] 22

Figure 3.7 - PCB layout for TXS0108E bidirectional level shifter [18] 23

Figure 3.8 - PCB layout for RP2040 microcontroller [18] 24

Figure 3.9 - Rocket flight cycle 25
Figure 4.1 - Inertial reference frame 26
Figure 4.2 - Body reference frame 27
Figure 5.1 - Simulink open loop model 35
Figure 5.2 – Open loop response 35
Figure 5.3 - MATLAB 3DOF Rigid Body block 36
Figure 5.4 - Total Simulink control system for pitch control 37
Figure 5.5 - Force and angle decomposition for 3DOF block 38
Figure 5.6 - Pitch attitude feedback path with servo reaction time model and PID(z) controller .39
Figure 5.7 - 3DOF closed loop response for pitch control to target angle 40
Figure 5.8 - PID Simulink model with transfer function 41
Figure 5.9 - PID controller with transfer function response to target angle 41
Figure 5.10 - Comparing MATLAB PID response to target angle 42
Figure 5.11 - LQR Simulink model for pitch control 43
Figure 5.12 - LQR response for pitch control to target angle 44
Figure 5.13 - Comparing LQR and PID pitch controller response to target angle 44
Figure 5.14 - LQG Simulink model for pitch control 45
Figure 6.1 - Closed loop pitch attitude tracking under Python 3 PID simulation 48
Figure 6.2 - Comparing MATLAB and Python PID responses to pitch for a target angle 49
Figure 7.1 - Launch 1 Early oscillation after liftoff 55
Figure 7.2 - Launch 1 Mid ascent with growing instability 56
Figure 7.3 - Launch 1 Loss of vertical alignment 57
Figure 7.4 - Launch 2 ignition and upright liftoff 58
Figure 7.5 - Launch 2 initial lateral skew 58
Figure 7.6 - Launch 2 translational instability leads to spiraling 59
Figure 7.7 - Launch 2 rocket spiraling before hitting the ground 59
Figure 7.8 - Launch 2 pitch angle and pitch PID setpoint vs time 60
Figure 7.9 - Launch 2 roll angle and roll PID setpoint vs time 60

Figure 7.10 - Launch 2 gyroscope angular rates vs time 61
Figure 7.11 - Launch 2 acceleration rates vs time 61
Figure 7.12 - Launch 2 altitude vs time 62
Figure 7.13 - Launch 3 initial ignition 63
Figure 7.14 - Launch 3 bounded lateral motion 64
Figure 7.15 - Launch 3 loss of vertical alignment 64
Figure 7.16 - Launch 3 pitch angle and pitch PID setpoints vs time 65
Figure 7.17 - Launch 3 roll angle and roll PID setpoint vs time 66
Figure 7.18 - Launch 3 gyroscope angular rates vs time 66
Figure 7.19 - Launch 3 acceleration rates vs time 67
Figure 7.20 - Launch 3 altitude vs time 68
Figure 7.21 - Launch 4 ignition 69
Figure 7.22 - Launch 4 early divergence 70
Figure 7.23 - Launch 4 oscillation and spiraling 70
Figure 7.24 - Launch 4 crash landing and smoke trail spirals 71
Figure 7.25 - Launch 5 ignition and upright liftoff 72
Figure 7.26 - Launch 5 mid ascent attitude correction 72
Figure 7.27 - Launch 5 stable mid ascent attitude 73
Figure 7.28 - Launch 5 tumbling near apogee 73
Figure 7.29 - Launch 5 pitch angel and pitch PID setpoint vs time 74
Figure 7.30 - Launch 5 roll angle and roll PID setpoint vs time 75
Figure 7.31 - Launch 5 gyroscope angular rates vs time 75
Figure 7.32 - Launch 5 acceleration rates vs time 76
Figure 7.33 - Launch 5 altitude vs time 77

Symbols

Symbol	Definition	Units (SI)
Symbols:		
C_p	Center of pressure	m
C_g	Center of gravity	m
K_p	Proportional gain	-
K_i	Integral gain	-
K_d	Derivative gain	-
K	Gain matrix	-
A	State matrix	-
B	Input matrix	-
C	Output matrix	-
D	Feedforward/through matrix	-
I	Moment of inertia tensor	$kg * m/s^2$
F	Force	N
F_T	Thrust force vector	N
δ_x	Pitch gimbal deflection	$degrees$
δ_z	Roll gimbal deflection	$degrees$
d	Distance from gimbal to C_g	m
Ob	Observation matrix	-
Co	Controllability matrix	-
g	Gravity	m/s^2
U	Longitudinal velocity	m/s
W	Vertical velocity	m/s
$\dot{U}$	Forward acceleration	m/s^2
$\dot{W}$	Vertical acceleration	m/s^2
M_y	Moment about Y	$N * m$
θ_{cmd}	Command pitch angle	$degrees$
Q_{lqr}	Q matrix for LQR	-
R_{lqr}	R matrix for LQR	-
K_{lqr}	K gain matrix for LQR	-
Q_{lqg}	Q matrix for LQG	-
R_{lqg}	R matrix for LQG	-
L	L matrix for LQG	-
b_{gx}	g_x bias	m/s^2
b_{gy}	g_y bias	m/s^2
b_{gz}	g_z bias	m/s^2
g_x	x-axis gyroscope measurements	m/s^2
g_y	y-axis gyroscope measurements	m/s^2
g_z	z-axis gyroscope measurements	m/s^2
a_x	x-axis accelerometer measurements	g

a_y	y-axis accelerometer measurements	g
a_z	z-axis accelerometer measurements	g
P	Pressure	hPa
h	Altitude	m
Greek Symbols:		
ϕ	Roll angle	$degrees$
θ	Pitch angle	$degrees$
ψ	Roll angle	$degrees$
τ	Torque	$N * m$
ω	Angular velocity vector	$rad * s^{-1}$
$\dot{\omega}$	Angular acceleration vector	$rad * s^{-2}$
$\dot{\phi}$	Roll rate	deg/s
$\dot{\theta}$	Pitch rate	deg/s
$\ddot{\theta}$	Pitch acceleration	deg/s^2
θ_{cal}	Theta calibration bias	$degrees$
ϕ_{cal}	Phi calibration bias	$degrees$
θ_f	Theta filter	$degrees$
ϕ_f	Phi filter	$degrees$
Acronym:		
VTVL	Vertical takeoff vertical landing	
TVC	Thrust vector control	
RTLS	Return to launch site	
DRL	Downrange sanding	
ACS	Aerodynamic control surfaces	
GNC	Guidance, navigation and control	
OCP	Optimal control problem	
PID	Proportional, Integral, and Derivative	
LQR	Linear Quadratic Regulator	
LQG	Linear Quadratic Gaussian	
BPNG	Biased Proportional Navigation Guidance	
MPC	Model predictive control	
BPNG	Biased navigation guidance	
3DOF	3 Degree of freedom	
PCB	Printed circuit board	
IMU	Inertial measurement unit	
CPU	Central processing unit	
GPIO	General purpose input/output	
PIO	Programmed input/output	
SWD	Serial wire debug	

1. Introduction

1.1 Motivation

In the past model rocketry has largely focused on achieving high speeds or altitudes, often with minimal emphasis on the control system aspect of aerospace engineering. This historical approach has created a disparity between traditional model rocketry and the groundbreaking innovations seen in the space launch vehicle industry. In particular, the vertical takeoff vertical landing (VTVL) and reusable rockets of companies like SpaceX and Blue Origin have revived interest in space due to the cost cutting measures that these rockets provide [1, 2]. Due to this gap, there is compelling motivation to create a thrust vector controlled (TVC) rocket that is capable of vertical takeoff and landing without relying on traditional stabilization fins.

By integrating TVC capabilities into model rocketry, it becomes possible to explore the engineering challenges and principles behind cutting edge aerospace developments while also leveraging decades of testing and infrastructure from the model rocket community. This project offers a unique opportunity to simulate and experiment with scaled down versions of industry practices while also gaining intuition around general rocketry and gaining a more concrete understanding of control system algorithms.

1.2 Literature Review

1.2.1 Guidance, Navigation, and Control Algorithms for Vertical Landing Rockets

Historically, space launch vehicles were mainly single use, with the exception being NASA's Space Shuttle program. However, the high costs of this approach led to a shift towards reusable rockets in recent years. This transition marks a pivotal moment in space exploration. To enable rocket reusability, they are designed to land vertically. This not only reduces turnaround times between launches but also minimizes wear and tear on the rocket boosters themselves.

A paper written in 2022 and published in CEAS Space Journal looks at designing landing guidance systems for vertical landing rockets [3]. Figure 1.1 shows a diagram on how these vertical landings are achieved. There are two main types of vertical landing: Return to Launch Site (RTLS) and Downrange Landing (DRL). With the RTLS system landing near the initial launch site and the DRL system landing further away but having to do less complicated maneuvers. Both systems use aerodynamic control surfaces (ACS) to manipulate the rocket into landing vertically [3].

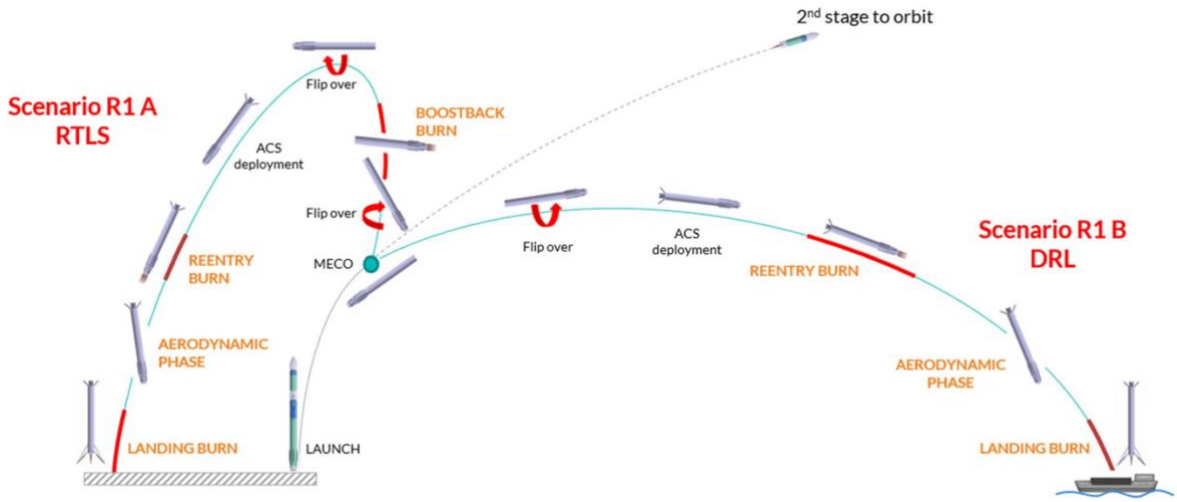


Figure 1.1 - Vertical landing rocket method [3]

This paper further examines the guidance, navigation, and control (GNC) architecture used to address this problem. The guidance algorithm generates desired trajectories using state estimates such as the velocity, altitude, and attitude which is provided by the navigation system, while the control system commands actuators based on these guidance outputs [3]. The guidance strategy is formulated as an optimal control problem (OCP), shown in Figure 1.2, which incorporates the system dynamics, objective function, and constraints. Solving the OCP produces a fuel optimal trajectory and thrust profile.

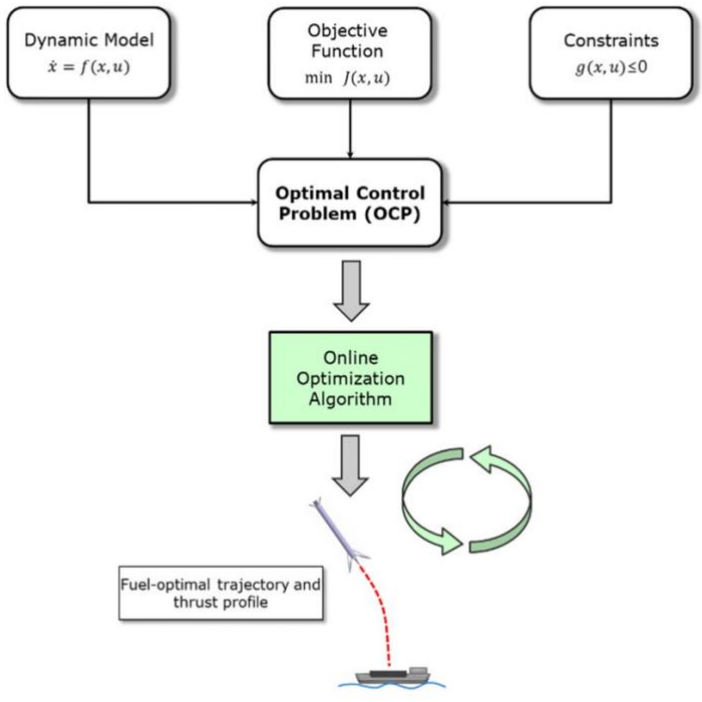


Figure 1.2 - Optimal control problem diagram [3]

To add on to this a paper written in 2023 looks to use reinforcement learning in or to model and land a reusable rocket [2]. Reinforcement learning for algorithms learns through guided trial and error, as time goes on the mistakes become smaller leading to ideal answers. They formulate their control problem with the initial state of the vehicle and formulate two problems, one for the attitude and another one for the trajectory optimization. For the attitude control problem, they created a LQ attitude control architecture and compared how a baseline attitude linear quadratic gaussian (LGQ) control kept up with a reference yaw angle compared to the reinforcement learning algorithm. The plot is shown in Figure 1.3 and shows the LQG control system with wind and disturbance present [2]. The reinforcement learning algorithm tracked the yaw marginally better but for the scope of this project the LQG control system would be easier to implement.

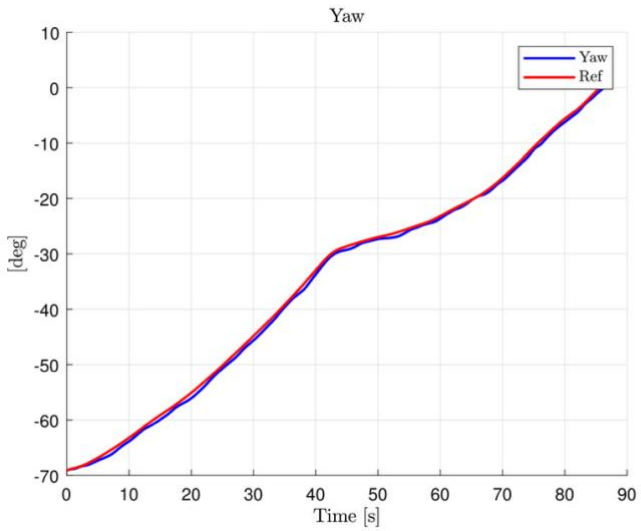


Figure 1.3 - Baseline attitude LGQ control in tracking the reference yaw angle [2]

The trajectory control architecture expands upon the attitude control architecture and uses both LQ controller and a PID controller. The LQ controller takes in altitude error and adjusts the thrust according to the algorithm. The PID controller takes in the longitude error and makes Yaw reference increments, both working together to vertically position the rocket. The system diagram is shown in Figure 1.4 and shows the two systems mesh.

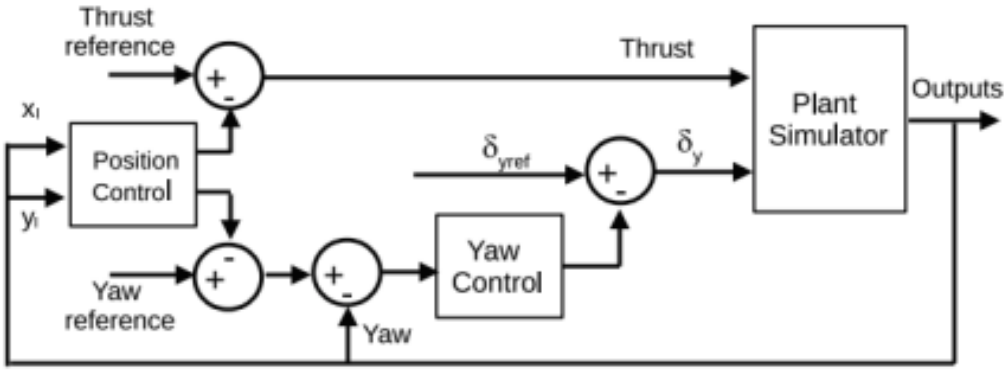


Figure 1.4 - Trajectory control architecture that includes the attitude control architecture [2]

The next two papers explore a different approach from reinforcement learning, using a modified convex optimization approach to model a vertical landing rocket [4, 5]. Both papers use model predictive control (MPC) frameworks to solve the problem through convex optimization. The first paper does this by having the guidance split along the normal and tangential planes allowing the convex optimization to occur [5]. The other uses a piecewise trajectory optimized algorithm based on convex optimization to predictively land rockets [4].

The proportional navigation model breaks up the problem into the normal and tangential planes. Figure 1.5 shows the projections in the XOY and XOZ plane that form the reference for the dynamic models [5]. With this reference a 3 degree of freedom dynamic model can be expressed by the show's variables over time. The drag and lift forces in all directions are calculated and added to the model before constraints are added.

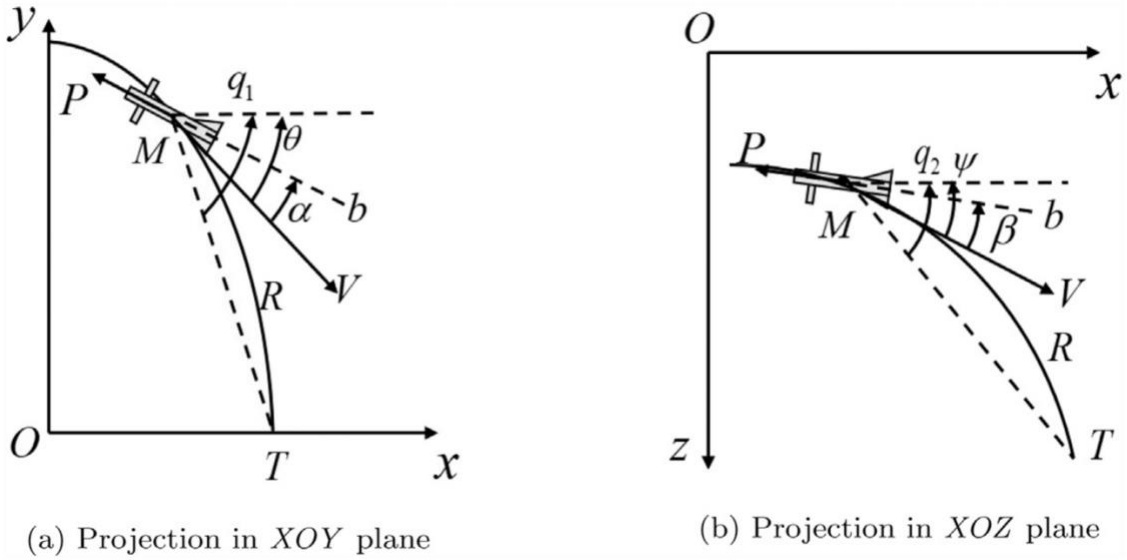


Figure 1.5 – XOY and XOZ planes of rocket through vertical landing [5]

Now, the system is split into 2 stage guidance strategies and each stage having a normal and tangential plane. Figure 1.6 shows the first stage of the two stage system. For the tangential plane a trajectory algorithm is developed through an MPC framework with the main framework based around the terminal constraint around speed. For the normal frame a biased proportional navigation guidance (BPNG) is used with the constraint being the landing angle while also trying to be within the constraints of the landing attitude angle. The second stage looks very similar to stage one with slight differences to the tangential plane due to engine and environmental circumstances.

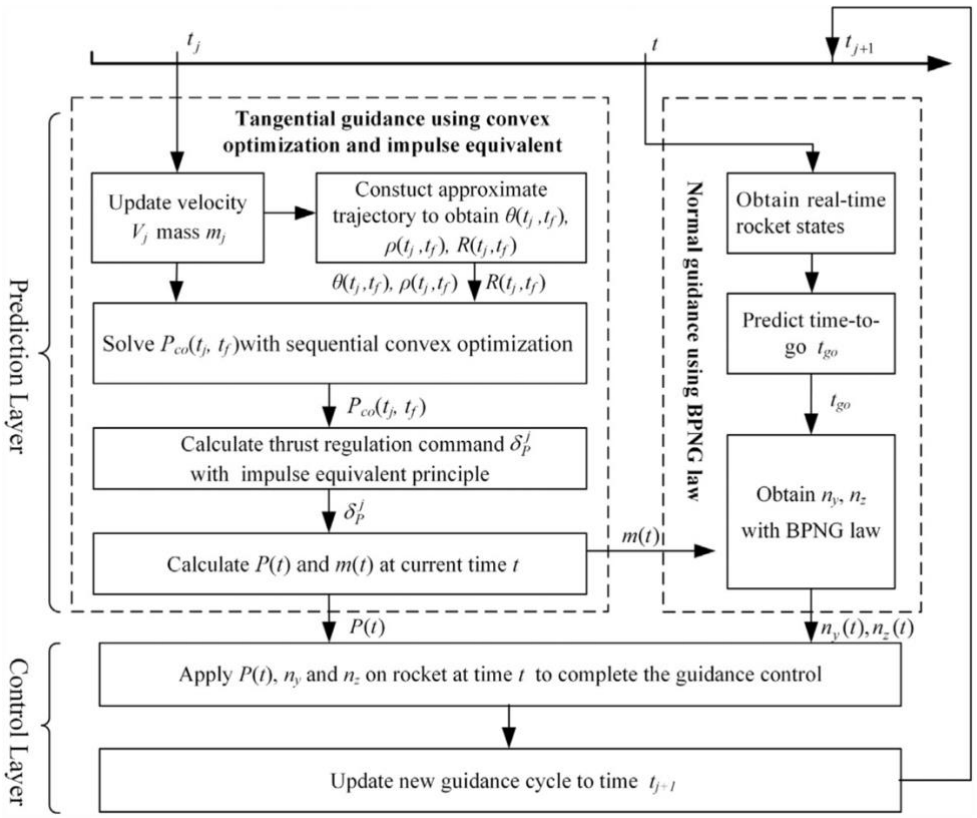


Figure 1.6 – First stage guidance cycle for vertical landing rocket flowchart [5]

The later paper presents a piecewise guidance strategy combined with convex optimization for vertical rocket landing. As shown in Figure 1.7, the algorithm uses repeated short guidance cycles that adapt near switching points, enabling finer trajectory optimization. Both this approach and the earlier framework produce reliable landing performance and are based on model predictive control structures.

Piecewise guidance algorithm based on MPC framework

Initialization: Save the initial trajectory to the database, set relevant parameters required by the trajectory optimization algorithm, and set the update index $i = 1$.

while$(\mathcal{T}_{i-1} - t_{i-1}) > 0$
  generate guidance commands according to the guidance cycle clock;
  sample the current state $\boldsymbol{x}(t_i)$(i.e. $\boldsymbol{r}(t_i)$, $\boldsymbol{v}(t_i)$, $m(t_i)$) of the rocket;
  employ the optimal trajectory $\boldsymbol{J}_{i-1}$ which was obtained in the previous guidance cycle as the initial guess. Start the trajectory optimization using the current state $\boldsymbol{x}(t_i)$. Obtain a new optimal trajectory $\boldsymbol{J}_i$ and an optimal thrust series $\boldsymbol{T}$. Save $\boldsymbol{J}_i$ to the database, implement $\boldsymbol{T}_i$, and set $i = i + 1$;
  if a fault occurs and t_i approaches the switching point
    shorten the duration of the guidance cycle: $\Delta t_i = \alpha \cdot \Delta t \ (0 < \alpha < 1)$;
  else
    $\Delta t_i = \Delta t$;
  end if
end while

Figure 1.7 - Trajectory optimization algorithm based on MPC framework [4]

1.2.2 Bridging the Gap Between Standard VTVL Rockets to TVC Rockets

All the examples above are creating algorithms for rockets that can send objects into orbit and need to be tuned down for model rockets. This is because model rockets have different properties when compared to their upscaled counterparts. Thrust, mass, center of gravity, gravity, pressures, and forces are all vastly different from real rockets. This paper looks at the dynamic model of a water rocket, a different propulsion system than a model rocket but has very similar forces acting on the rocket, as shown in Figure 1.8 [6].

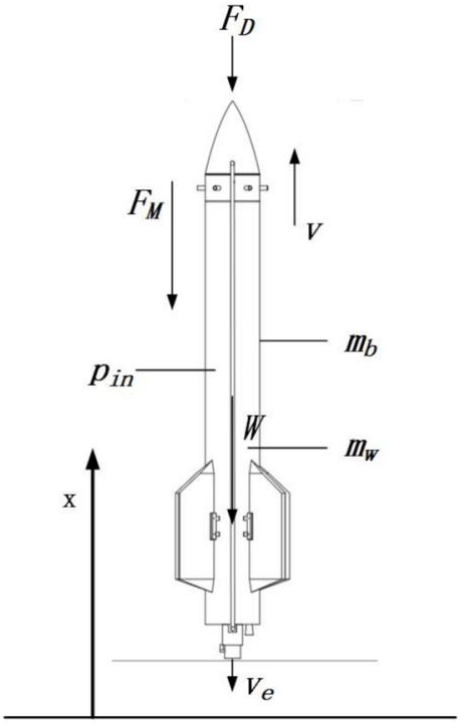


Figure 1.8 - Force breakdown of a water rocket [6]

By deriving equations for the forces, the authors were able to obtain position and velocity equations, which they then used with a list of parameters to generate position and velocity plots. Although the values are greatly higher on model rockets, these dynamic models can be adjusted to create an accurate analysis of model rocketry. To add on to that the dynamic models can help once again by adjusting the forces that the rocket would face on the descent and seeing how the dynamic model changes.

Another paper looks at and compares TVC for a finless rocket [7]. The paper shows how a finless rocket has a higher center of pressure (Cp) and how that leads to instability, in which the TVC algorithm is trying to consider, Figure 1.9 shows this in detail as well as other forces acting on the body of the rocket.

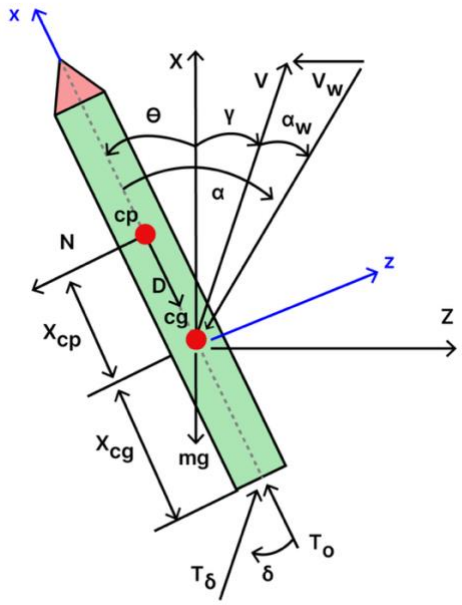


Figure 1.9 - Finless rocket model [7]

The paper goes through gaining the quotations of motion and doing a stability analysis of the entire system. After designing the controller, it does show that the system is shown to be controllable as the controllability matrix, found through control theory, was shown to be full rank. The paper then goes through tuning an LQR, LQG, and a PID controller to compare their performance; it was found that the LQ models were slightly better than the PID controller [7].

Another paper backs this up by creating a multitude of controllers for finless rockets. The controllers that are compared are continuous LQR, discrete LQR, discrete gain scheduled LQR as well as a PID controller. They found that a time delayed LQR controller performs the best overall [8].

1.2.3 Control Systems

A control system regulates the behavior of other systems using a control loop. It takes an input and compares it with the current state of the system and makes corrections to minimize the difference between the output of the system and the given input. Control systems are used in a wide variety of applications and industries; they can help regulate the temperature in a room to be able to control the flight path of rockets and satellites.

In aerospace engineering, control systems play a vital role in maintaining stability, guidance and performance for airplanes, rockets, and drones. Control systems help regulate flight surfaces (ailerons, elevators, and rudders) to maintain various aspects of flight. In drones control systems are often used to stabilize the drone by adjusting the rotors based on the orientation of the drone. Modern aerospace vehicles, including thrust vectoring rockets, use advanced control systems to enable precise vertical takeoff and self landing by adjusting engine direction through gimbal mechanisms allowing for real time orientation control. In gimbal controlled rockets, control systems are crucial for enabling vertical takeoff and self landing. A gimbal mechanism allows the

rocket engine to pivot, changing the direction of thrust and thus controlling the rocket's orientation and position during flight. This project uses a control system to perform thrust vector control on model rockets. There are several types of control systems can be used such as a PID (Proportional, Integral and Derivative) control system, an LQR (Linear Quadratic Regulator) controller as well as an LQG (Linear Quadratic Gaussian) control system all which will be explained and talked about in this section.

1.2.3.1 PID Controller

A PID control system is one of the most common feedback control systems used in engineering and general automation. A PID control system works by measuring the difference between the desired setpoint and the actual output of the system. This difference is known as the error. The PID controller takes this error and adjusts the system's input to minimize the error. The PID controller adjusts and controls the output using three different components: the proportional, integral and derivative values. The proportional term applies a correction that is proportional based on the current error meaning that the higher the error value the higher the correction action the controller takes. The integral value sums up the past errors over time and uses those values to correct for any offset between the setpoint and the actual value. The derivative value predicts future error trends by looking at the rate of change in the system, allowing the system to anticipate and counteract any over or undershooting. By using all three values the controller can provide a balanced and timely response to the system's output.

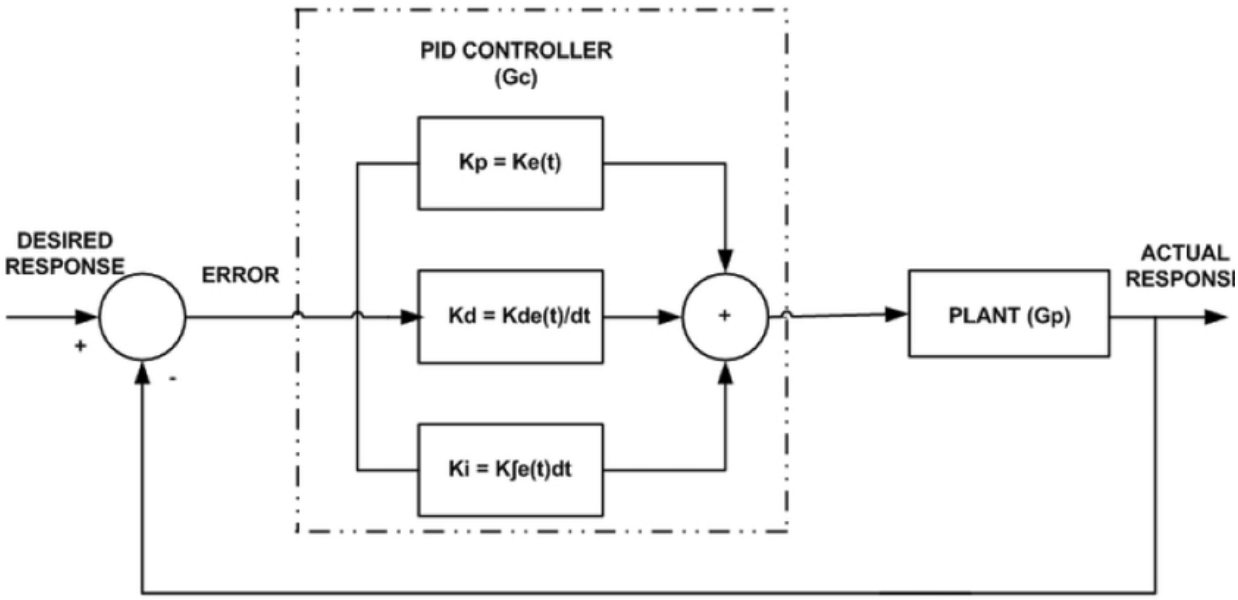


Figure 1.10 – PID control system model [9]

Figure 1.10 shows a basic PID model, where the desired response is compared to the actual response to calculate the error. This error is processed through the proportional (K_p), integral (K_i), and derivative (K_d) terms, producing a control signal that is sent to the plant. The plant is the system being controlled. For example, a rocket's thrust vector gimbal mechanism. The PID output is applied to the plant's actuators, changing the system's behavior. The plant responds, new sensor

readings are taken, and the PID loop repeats, often many times per second. This feedback cycle allows the controller to continuously correct the system's state, keeping it as close as possible to the desired setpoint. PID controllers are relatively easy to design and tune when compared to the control systems later in this section and are widely used in industry [10].

To tune a PID controller the P, I, and D values need to be changed and tweaked to gain the right exposure to each component to track the reference signal [11]. These values are symbolized as K_p, K_i, and K_d and for each distinctive part. One of the main ways of tuning is to start with tweaking just one value. The most ideal option would be the proportional component, K_p, as it causes the biggest change. After setting the K_i and K_d values to zero to the K_p value should be selected such that the system begins to oscillate. This value should ensure that the system reduces a significant portion of its error but avoids instability. The K_i value has a similar philosophy. After getting a nice oscillating system response the K_i should be gradually increased such that the system corrects error effectively without compromising recovery time or stability of the system. The last value, K_d, is used to smooth out the curve and dampen any overshoot and can be found once again by starting out with a small K_d value and gradually increasing it to not affect vital aspects of the system [12]. Now once these initial values are found the system can be further fine tuned based on whether the system is too slow to respond, it oscillates too much, or if there are still errors and stability issues. Table 1.1 shows how each value of the PID controller can affect the overall system.

Table 1.1 - PID tuning table [13]

Parameter	**Rise Time**	**Overshoot**	**Settling Time**	**Steady State Error**	**Stability**
K_p	Decrease	Increase	Small change	Decrease	Degrade
K_i	Decrease	Increase	Increase	Eliminate	Degrade
K_d	Little change	Decrease	Decrease	Theoretically no change	Improve if K_d is low

1.2.3.2 Linear Quadratic Regulator Control System

A LQR (Linear Quadratic Regulator) control system is an optimal control method used to regulate dynamic systems by minimizing a specific cost function. It works with a mathematical model of the system, typically expressed in a state space form, where the system's behavior is described by its state variables and their evolution over time. The LQR controller determines the control inputs works by keeping the system's states close to their desired values and minimizing the effort or energy used by the actuators. This balance is achieved by weighting the importance of state deviations versus control effort through two matrices, Q and R, which are chosen based on performance priorities. In operation, the LQR measures or estimates the system's current state and calculates the control input using the equation u = −Kx, where u is the control signal, x is the state

vector, and K is a gain matrix derived from the system model and cost function. This gain matrix is precomputed using optimization methods, allowing the controller to be stable and efficient.

$$\dot{x} = Ax + Bu \quad (1.1)$$
$$y = Cx + Du \quad (1.2)$$

The state vector x represents variables such as position, velocity, or angle, while u is the control input. The matrices A and B define the system dynamics, with A describing the natural state evolution and B capturing the influence of control inputs. Matrices C and D map states and inputs to outputs but are not always directly used in the control law.

$$u = -Kx \quad (1.3)$$

The controller calculates an optimal feedback law, shown in Equation 1.3, where K is a gain matrix designed to balance keeping the system stable and minimizing control effort. This balance is achieved by minimizing a cost function that weighs both state errors and control input use. During operation, the controller continuously measures the system's state, computes the control input using this feedback law, and applies it to the system to keep it stable and efficient. Combining these we get a control system that looks like Figure 1.11. An LQR system is effective because it provides optimal, balanced control that minimizes both error and control effort based on a mathematical model of the system [10].

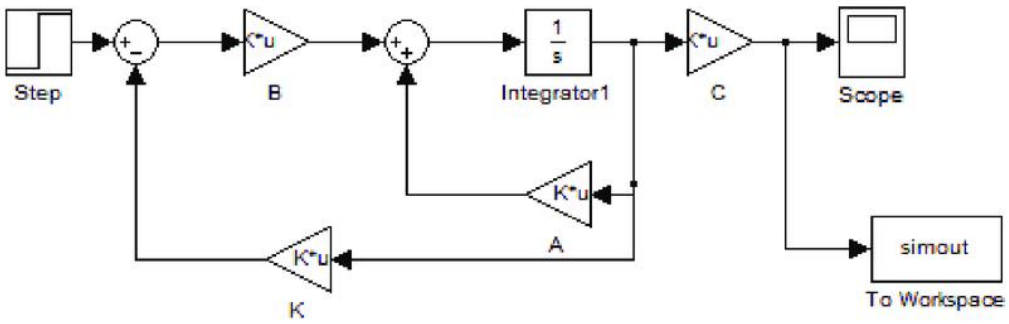


Figure 1.11 – LQR control system model [14]

1.2.3.3 Linear Quadratic Gaussian Control System

The LQG (Linear Quadratic Gaussian) control system is an extension of the LQR controller that handles situations where the system's states cannot be measured directly or are affected by noise. Like the LQR, the LQG aims to minimize a quadratic cost function balancing state errors and control effort, using a gain matrix to generate optimal control inputs. However, unlike LQR, which assumes full knowledge of the system's states, LQG incorporates a Kalman filter to estimate the true states from noisy or incomplete sensor measurements.

A Kalman filter is an algorithm that estimates the true state of a system by combining predictions from a mathematical model with noisy sensor measurements. It operates in two steps: prediction and update. First, it predicts the next state and its uncertainty based on the previous estimate and control inputs [10]. Then, it updates this prediction by comparing it to the actual measurement,

weighting each according to their uncertainties to produce a more accurate estimate. By continuously repeating these steps, the Kalman filter effectively reduces noise and errors, providing smooth and reliable state estimates even when data is noisy or incomplete, making it widely used in control systems, navigation, and robotics.

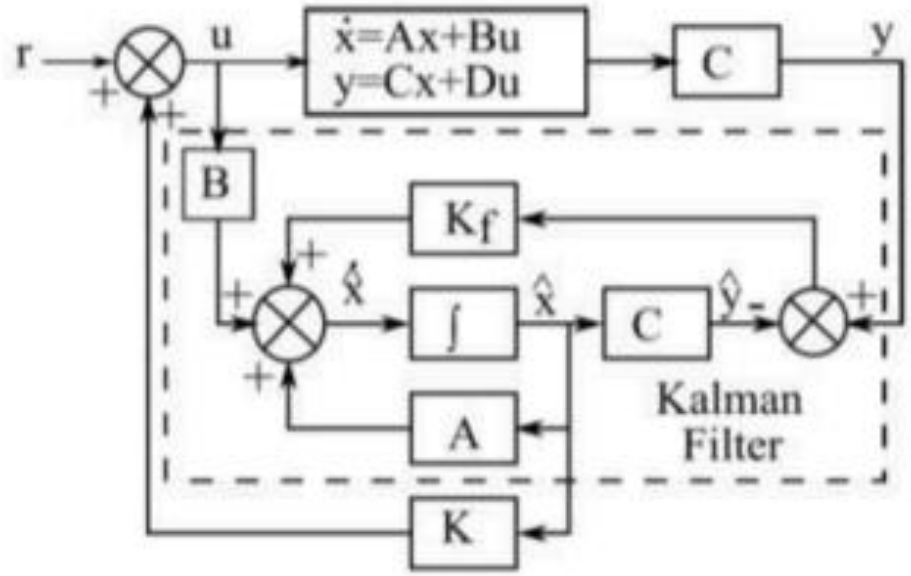


Figure 1.12 – LQG control system model [15]

Figure 1.12 shows the full LQG model with the Kalman filter outlined. The main difference between a standard LQR and an LQG controller is that LQG includes this state estimation step via the Kalman filter, making it robust to measurement noise and partial observability. While a normal LQR requires exact state information, LQG works effectively even when sensors are noisy or some states are unmeasurable. The benefit of this approach is improved reliability and accuracy in realistic conditions where noise and uncertainty are unavoidable. However, the added complexity of the Kalman filter increases computational demands and requires accurate modeling of noise characteristics.

The PID controller was chosen for this project due to its reliability and lower complexity compared to more advanced control methods such as LQR and LQG. Its straightforward error based approach simplifies implementation and tuning, especially in cases where an exact mathematical model is unavailable. Furthermore, the 3DOF block in MATLAB models the rocket using an error based system that tracks deviations from desired states rather than providing a full state space representation. This modeling approach is well suited for PID control but incompatible with LQR or LQG controllers, which require an accurate state space model of the system dynamics to compute optimal control inputs [10].

1.3 Proposal

The objective of this project is to design and build a thrust vector controlled (TVC) rocket capable of reaching a minimum of 30 meters and thereafter deploy a parachute safely landing to the ground. A crucial aspect of this project revolves around implementing an effective guidance, navigation, & control (GNC) algorithm to accurately thrust vector control the rocket. Iterative testing and refinement of both the software and hardware components of the rocket will be employed to ensure the successful vertical landing of the model rocket.

1.4 Methodology

Concurrently, an extensive background research phase is conducted to inform the design and development process. This involves reviewing the principles of model rocketry, including airframe materials, motor mounts, recovery systems, and the use of sensors and flight computers. Attention is given to prior attempts at vertical landing in amateur rocketry to identify effective strategies and common pitfalls. The research also covers the evaluation of solid rocket motors, comparing thrust curves, burn times, peak and average thrust, total impulse, and cost to determine the optimal choice for the project. Additionally, design considerations for the TVC gimbal are explored, including the selection of servos, mounting strategies, mechanical range, materials, and control software. This comprehensive background research establishes a foundation for the overall rocket design and informs critical decisions throughout the project.

1.4.1 Simulations

The simulation phase is divided into two main categories: rocket flight simulations and gimbal/control system simulations. Rocket flight simulations aim to predict the vehicle's performance and identify potential issues before any physical components are built. These simulations include aerodynamic modeling, stability analysis, and predictions of altitude, velocity, and acceleration profiles. They also guide the selection of motors and recovery systems and help determine appropriate sensor configurations for measuring flight parameters. By this stage, the rocket design is also translated into CAD models to verify component fit, assess the location of the center of mass, and evaluate structural integrity.

Gimbal and control system simulations focus on testing the thrust vector control mechanism in a virtual environment. These simulations evaluate the mechanical range and torque limits of the gimbal servos, model how thrust deflection affects rocket orientation, and test control algorithms under simulated flight conditions. Safety features such as servo limits, fault detection, and fallback modes are incorporated to prevent catastrophic failure. The simulations also aim to optimize the control algorithm to minimize unnecessary gimbal movement, ensuring the rocket remains as vertically aligned as possible with minimal actuation. By completing this phase, the project team can refine both the flight control software and the mechanical design before moving to physical testing.

1.4.2 Rocket Construction and Test Stands

Once the simulations validate the design, the physical construction of the rocket begins. Materials such as airframe tubes, structural reinforcements, the flight computer, sensors, power systems, and gimbal components are procured. The gimbal is either machined or 3D printed using heat resistant materials, and the rocket is assembled to ensure proper fit, wiring, and structural alignment. A decision is made regarding whether to use a commercially available flight computer or a custom Raspberry Pi based design, considering tradeoffs such as reliability, flexibility, and integration complexity.

Test stands are built to evaluate both the gimbal and the complete rocket assembly. The gimbal test stand allows static motor tests to validate servo performance under thrust, assess the response of the control system to disturbances, and measure durability under mechanical stress. Any necessary design adjustments are made before full assembly. A rocket launch or test stand is also constructed to verify mass distribution, assess stability, and examine component interactions under small, applied disturbances. This setup ensures that the rocket can safely undergo flight testing while providing early insight into potential mechanical or control system issues.

1.4.3 Flight Testing and Evaluation

The final phase of the project involves systematic flight testing and evaluation. A comprehensive safety plan is developed, including parachute deployment, fail safe mechanisms, servo deflection limits, safe distances, and detailed launch protocols. A manual documenting checklists, procedures, and contingency plans is prepared to ensure consistent and safe operation.

Initial test flights focus on validating basic rocket performance, including stable ascent, data logging, and control system operation. In the event of failure, relaunch plans are implemented after identifying minor issues, such as motor anomalies, launch pad inconsistencies, or sensor misalignment. When significant problems occur, detailed analysis is conducted using both recorded flight data and visual observations. The root cause of failures that be software, hardware, or operational is determined and modifications are implemented accordingly. Iterative testing, data analysis, and documentation continue until the rocket consistently demonstrates controlled, vertical flight and the desired performance objectives are met.

2. Rocket Body

2.1 General Rocket Design

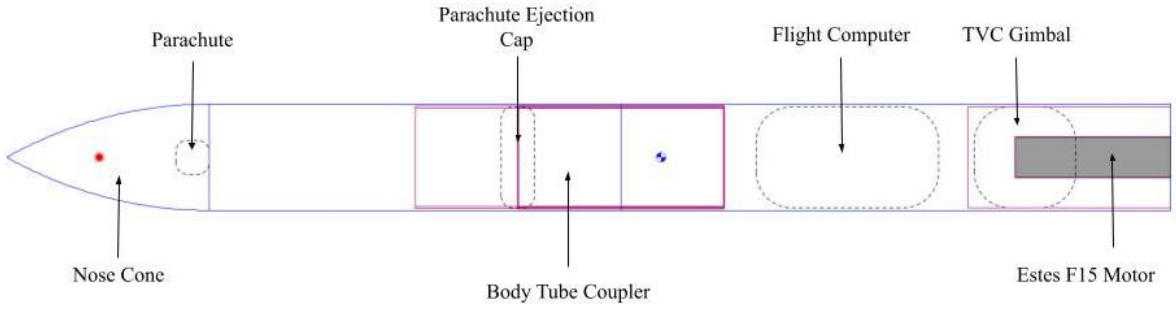


Figure 2.1 – General rocket design

Figure 2.1 illustrates the internal layout and structural configuration of the thrust vector controlled model rocket. The vehicle has an outer diameter of 7.87 cm (3.1 in) and an overall length of approximately 1.0 m, providing sufficient internal volume for the parachute bay, avionics module, gimbal assembly, and 29 mm Estes F15 motor. These dimensions yield a slenderness ratio near 13:1, which supports good aerodynamic characteristics while also providing the longitudinal space required for the flight computer and thrust vectoring hardware.

The internal components are arranged sequentially from nose to tail: the parachute and shock cord hardware are stored beneath the nose cone, followed by the parachute ejection cap and coupler, the avionics bay containing the flight computer, and finally the TVC gimbal and motor assembly at the bottom of the rocket. This layout ensures that the parachute is housed safely away from the high temperature motor compartment while positioning the flight electronics near the rocket's center for improved measurement fidelity and protection from excessive vibration.

Figure 2.1 also shows the positions of the center of gravity in blue (C_g) and the center of pressure in red (C_p). Maintaining adequate separation between these two points is essential to preventing aerodynamic instability during ascent. In a typical fin stabilized rocket, the C_p naturally sits behind the C_g due to the presence of tail fins, ensuring passive stability. However, because this rocket deliberately omits fins to enable thrust vector maneuvering, it lacks this passive stability mechanism. As a result, the vehicle behaves aerodynamically like an inverted pendulum during powered flight, requiring continuous closed loop control from the gimbal and flight computer to maintain orientation. The C_g is intentionally placed forward of the gimbal by design, ensuring that thrust vector corrections produce meaningful moments about the pitch and roll axes.

The fully assembled rocket, including avionics, wiring, parachute hardware, and the Estes F15 motor, has a dry mass of approximately 950g without the Estes F15 motor and a total mass 1.1 kg with the motor included. This mass allocation ensures compatibility with the thrust capabilities of the F15 motor while providing sufficient inertia to resist oscillations induced by gimbal actuation. The overall configuration is optimized for a single powered ascent followed by parachute based recovery.

2.2 Solid Rocket Motors and Altitude Simulations

The rocket is powered by a single Estes F15 solid rocket motor, selected for its long burn duration, reliability, and suitability for thrust vector controlled ascent. The F15 provides an average thrust of approximately 14.5 N over a burn time of roughly 3.5 seconds, producing a sustained and controllable acceleration phase that allows the TVC system to actively regulate the rocket's orientation throughout powered flight. The extended burn duration is particularly advantageous for attitude control, as it provides a long window during which the flight computer can measure pitch and roll deviations and apply corrective gimbal commands. The thrust curve is shown below in Figure 2.2.

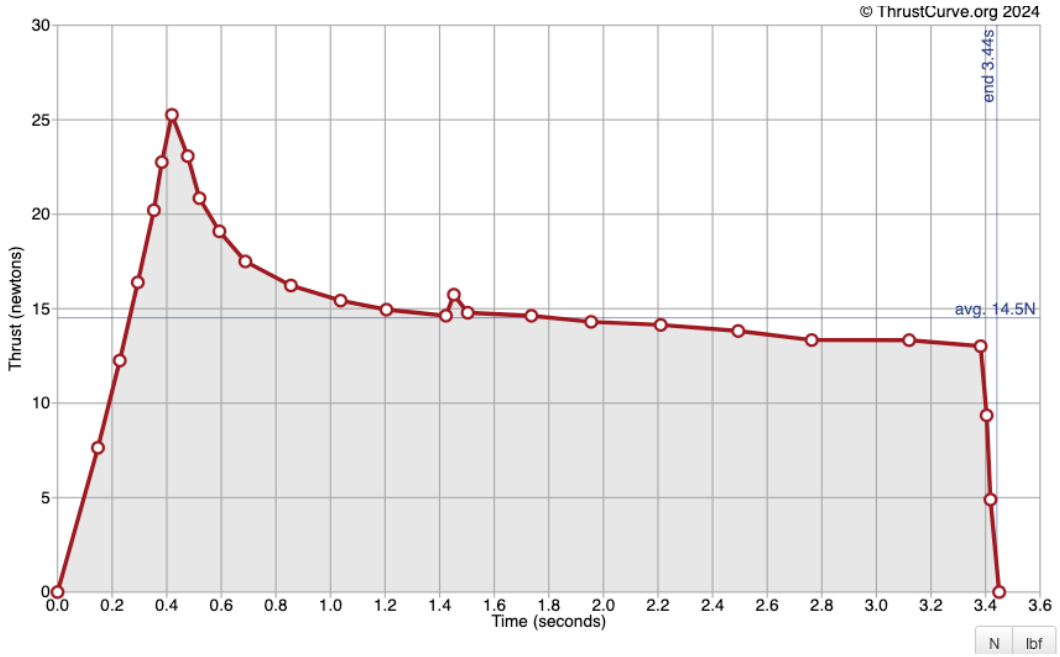


Figure 2.2 - Thrust curve for Estes F15 motor [16]

A notable characteristic of the F15 motor is its gently decreasing thrust curve. The motor initially produces a brief thrust rise before settling into a smooth, negatively sloped profile. For a thrust vector controlled rocket, this is advantageous because the initial higher thrust rapidly establishes upward acceleration and helps counteract any launch disturbances. As the burn progresses and vehicle velocity increases, the stability of the thrust reduces the likelihood of excessive acceleration while still providing sufficient force to maintain vertical orientation.

The performance of the rocket using the F15 motor was evaluated using an OpenRocket simulation. Figure 2.3 shows the altitude, vertical velocity, and acceleration (blue, red, and yellow)

of the vehicle throughout powered ascent and coasting flight. With a total liftoff mass of approximately 1.1 kg, the rocket reaches an apogee of roughly 35 to 40 meters, with motor burnout occurring shortly near 25 meters. This close timing reflects the relatively low average thrust of the motor and the mass of the payload and gimbal system yet still demonstrates a stable and well behaved ascent profile suitable for evaluating the TVC system. Because this vehicle uses parachute recovery rather than powered landing, the ascent phase becomes the sole period requiring active control input; following burnout, the vehicle coasts passively to apogee before deploying its parachute for descent.

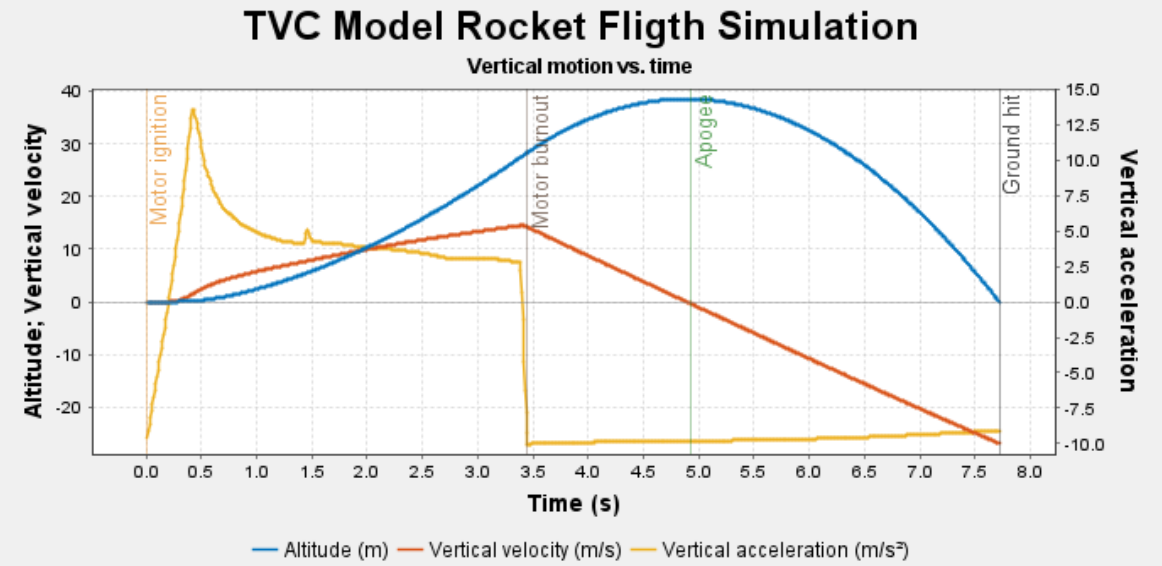


Figure 2.3 – TVC model rocket flight simulation

2.3 Flight Computer Housing

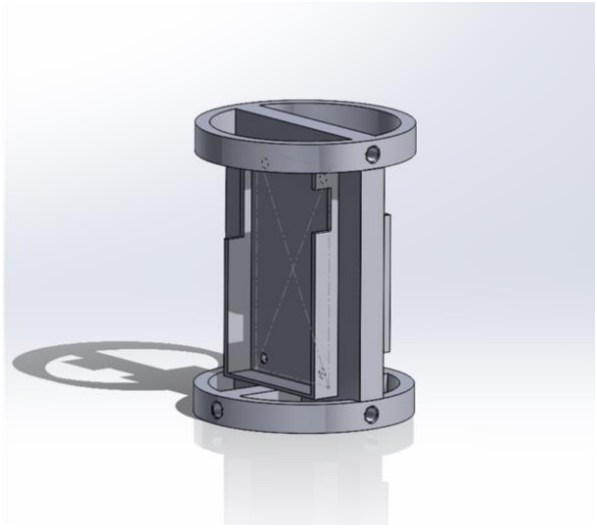


Figure 2.4 – Flight computer housing for STARLIGHT flight computer

The flight computer housing shown in Figure 2.4 is designed to fit snugly within the rocket's body tube, providing a rigid internal structure that securely supports the STARLIGHT flight controller. The cylindrical frame ensures precise alignment inside the airframe, while the central mounting plates allow the flight computer to be firmly screwed into place, minimizing wobble, flexing, and vibration during ascent. The open rear section provides adequate clearance for the battery connection, enabling stable electrical interfacing without obstructing the main electronics. Overall, the housing functions as a durable protective shell that isolates the avionics from structural loads and high frequency vibrations, ensuring reliable sensor readings and uninterrupted control operation throughout flight.

2.4 Gimbal Design

The thrust vector control system uses a two axis gimbal mounted at the base of the rocket to steer the motor and generate corrective moments during ascent. The gimbal enables independent pitch (x-axis) and yaw (z-axis) motion, allowing real time redirection of the thrust vector for vehicle stabilization. Two micro servos actuate the system through control arms that independently tilt the gimbal along each axis, keeping pitch and yaw mechanically decoupled and reducing both structural complexity and onboard computational load. This functionality is achieved through a dual ring gimbal architecture, with an outer frame and an inner motor mount providing smooth, orthogonal two degree of freedom rotation.

A 3D model of the gimbal, shown in Figure 2.5, was adapted from an established design created by Joe Bernard of BPS Space [17]. This design was selected due to its proven reliability, mechanical simplicity, and compatibility with 29 mm motor hardware, including the Estes F15 used in this project. Its modular structure also accommodates standard body tube couplers, enabling straightforward integration into the rocket's airframe. By leveraging an existing and well tested gimbal platform, development time was reduced while ensuring that the TVC mechanism met the precision and responsiveness required for stable ascent



Figure 2.5 - Gimbal design with servo motors [17]

3. Flight Computer Avionics

The flight computer serves as the rocket's central processor, integrating sensor data, controlling actuators, logging critical information, and executing GNC algorithms. It continuously measures orientation and angular velocities using the IMU, while determining altitude, vertical velocity, and ascent/descent rates with the barometer and auxiliary sensors. This data allows the flight computer to actively control the dual axis gimbal, correcting deviations in orientation to maintain vertical alignment during both ascent and descent. Throughout the mission, the flight computer logs flight data for post flight analysis and improvement, while providing real time diagnostic feedback through onboard LEDs and telemetry interfaces. By seamlessly combining real time sensor input with control logic, it ensures stable flight, precise maneuvering, and safe recovery of the rocket.

To streamline development and minimize risk, this project utilizes the commercially available STARLIGHT Model Rocket Flight Computer, produced by Circuit Wizardry. STARLIGHT provides a fully integrated solution, featuring a dual core RP2040 CPU, a 6-axis IMU, barometer, temperature sensors, servo outputs, 16 MB of flash storage, and LED indicators. By leveraging a prebuilt board, the project avoids the common challenges associated with custom PCB design, including soldering errors, component compatibility issues, and extended development time. The STARLIGHT flight computer includes all the features required for this project, as illustrated in Figure 3.1 [18].



Figure 3.1 - STARLIGHT model rocket flight computer [18]

3.1 Central Processing Unit

The STARLIGHT Flight Computer is powered by the RP2040 dual core microcontroller from Raspberry Pi, providing a versatile and reliable platform for rocket control. The RP2040 operates at up to 133 MHz and includes 264 kB of on chip SRAM, with support for up to 16 MB of off chip flash memory [19]. It offers over 30 (GPIO) pins for connecting sensors and actuators, 8 PIO ports for custom peripheral control, and a USB interface for programming, data transfer, and debugging.

The dual core architecture of the RP2040 enables one core to handle real time flight control computations, while the other manages logging, communication, and peripheral operations. This separation ensures that critical control loops remain uninterrupted by data transfers or logging tasks, enhancing system reliability. The Raspberry Pi ecosystem further supports development with extensive documentation, example code, and community resources, providing guidance on microcontroller configuration, pin assignments, and programming. These features simplify integration of the flight computer with the rocket's hardware and software systems, facilitating rapid development and robust performance. Figure 3.2 illustrates the RP2040 dual core MCU and its port layout [20].

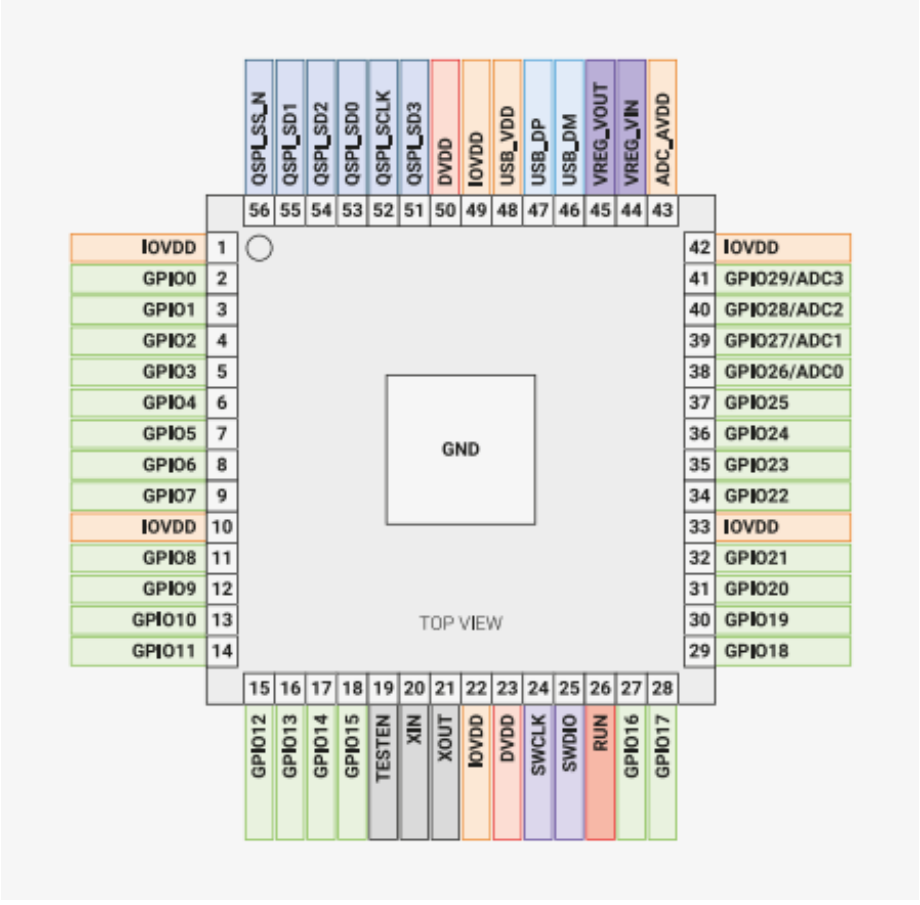


Figure 3.2 - RP2040 dual core MCU and port schematic [20]

3.2 Inertial Measurement Unit (IMU)

IMU's are critical for a TVC rocket, as they provide real time measurements of the rocket's orientation relative to an inertial reference frame. The STARLIGHT flight computer integrates the ICM-42605 IMU from TDK InvenSense, which combines a 3 axis accelerometer and a 3 axis gyroscope to provide six degrees of freedom [21]. The accelerometers measure linear acceleration along the X, Y, and Z axes, including both gravitational and linear motion. The gyroscopes detect angular velocity using MEMS based Coriolis effect sensing. This works when a tiny vibrating element inside the sensor rotates and deflects due to the Coriolis force; this deflection is measured to calculate angular velocity.

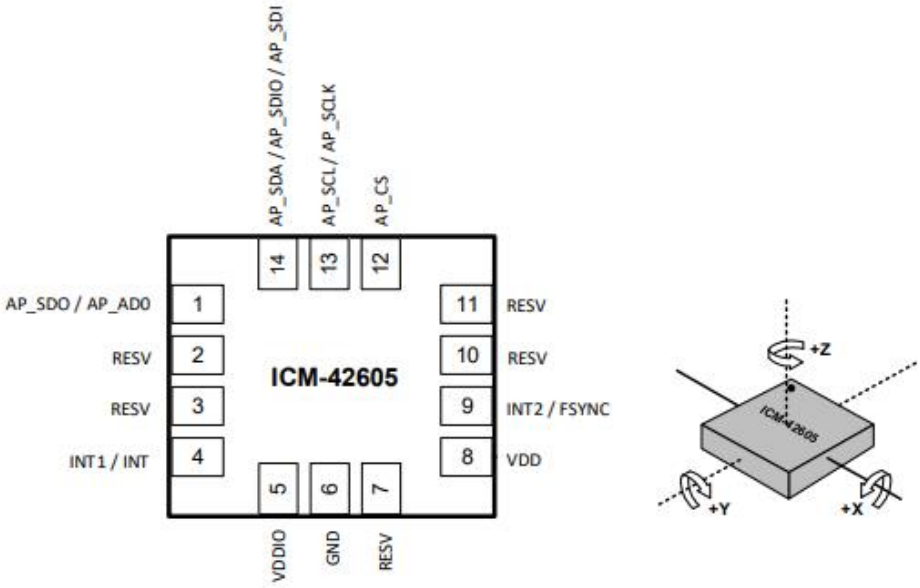


Figure 3.3 – ICM-42605 pin out diagram [21]

The raw data from the IMU is processed using integration algorithms to calculate angular displacement, allowing the flight computer to continuously update the rocket's orientation from its initial position. The MEMS technology in both the accelerometers and gyroscopes provides high accuracy, durability, and resilience against the vibration and acceleration stresses encountered during launch. The ICM-42605 is compatible with the RP2040 microcontroller used in the STARLIGHT flight computer, enabling seamless real time processing, reliable attitude determination, and robust thrust vector stabilization. Figure 3.3 illustrates the IMU and its pin layout [21].

3.3 Barometer

Altitude measurement in the STARLIGHT flight computer is accomplished using the BMP388 barometric pressure sensor from Bosch. This sensor determines altitude by detecting changes in atmospheric pressure, which decreases predictably as altitude increases. Calibration is performed to account for local weather conditions, temperature variations, and pressure drift, ensuring accurate measurements. The BMP388 features high resolution (0.01 hPa) and accuracy (±0.08 hPa), with integrated temperature compensation, providing reliable altitude data for flight stabilization, apogee determination, and deployment of recovery systems such as parachutes [22].

In addition to providing primary altitude information, the barometer supplements the IMU by helping correct for minor drift or sensor noise, improving overall navigation and guidance accuracy. Its compatibility with platforms like Arduino and Raspberry Pi, along with available libraries, example code, and Bosch datasheets, simplifies integration and enables rapid development. Figure 3.4 illustrates the BMP388 and its pin layout [20].

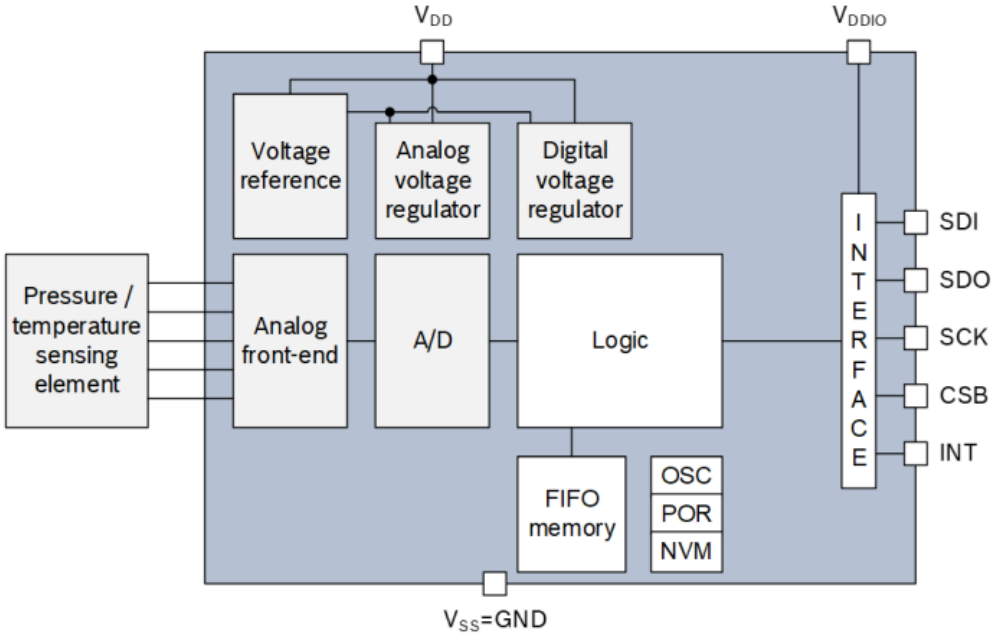


Figure 3.4 – BMP388 block diagram [20]

3.4 Logging and Receiving Flight Data

Recording flight data is essential for evaluating the rocket's performance and refining the control algorithms, as real world dynamics often deviate from simulation based predictions. As shown in Figure 3.5, the STARLIGHT flight computer incorporates a 16 MB W25Q128 QSPI flash device mounted directly on the PCB, enabling continuous data logging at 10 Hz for up to 15 minutes [23]. This provides ample margin for capturing auxiliary diagnostics or extended ground testing.

Using onboard flash memory eliminates the mechanical vulnerabilities associated with removable storage, such as SD cards, which are susceptible to vibration induced intermittent contact or data corruption during launch. The nonvolatile flash supports reliable recording of key flight variables,

including orientation, acceleration, altitude, servo commands, and event timestamps, throughout the entire ascent and descent.

After landing, the recorded data can be retrieved through the STARLIGHT micro USB interface for offline analysis. These logs enable reconstruction of the vehicle's flight trajectory, assessment of controller performance, and identification of anomalies such as drift, oscillation, or sensor bias. This post flight insight allows iterative improvements to the vehicle's hardware and control software, ultimately enhancing stability and accuracy in future launches.

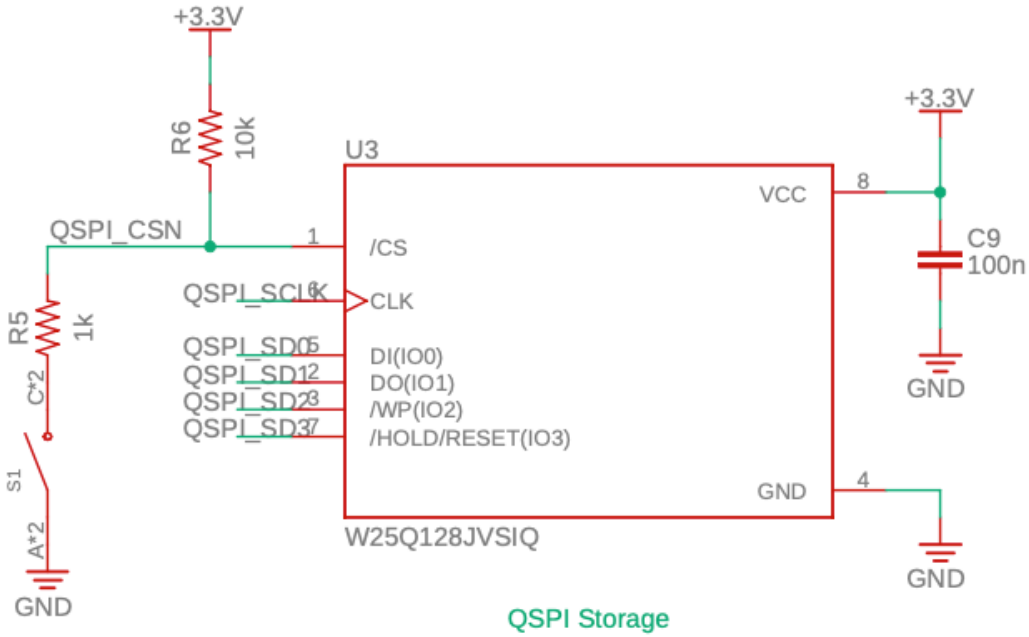


Figure 3.5 - PCB layout of W25Q128 QSPI flash device [18]

3.5 PCB Layout and Design

Power distribution on the PCB is organized around a wide input battery front end, which accepts approximately 2.7–20 V and feeds an LM7805 based 5 V rail. From this intermediate rail, a TLV76733 linear regulator generates the 3.3 V logic supply used by the RP2040, sensors, and level shifting circuitry. Local 100 nF and 1 µF decoupling capacitors are distributed near the RP2040 power pins and regulators to suppress high frequency noise and transient voltage dips [24]. This staged regulation scheme, illustrated in Figure 3.6, isolates the sensitive digital electronics from servo and igniter transients while allowing the same battery source to simultaneously support high current loads and low power logic.

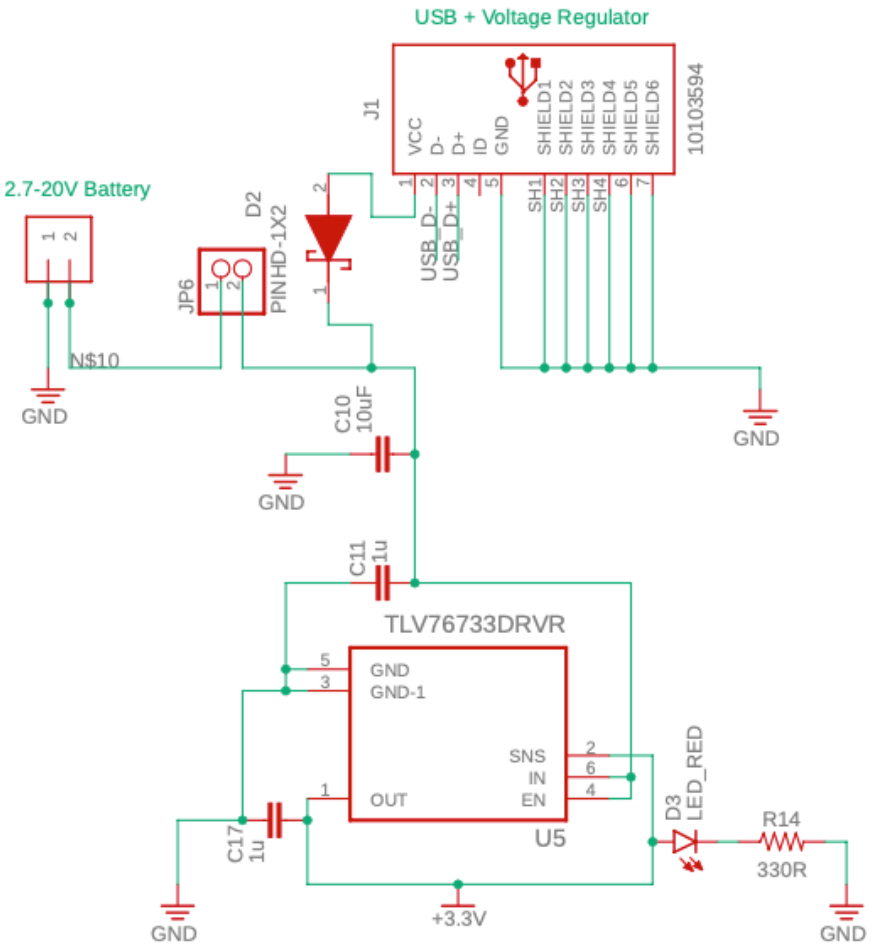


Figure 3.6 - PCB Layout of TLV76733 linear regulator [18]

Actuation and pyrotechnic control are handled by dedicated sections of the PCB. Two three pin headers provide servo outputs for the thrust vectoring gimbal, with signal lines routed through a TXS0108E bidirectional level shifter [25]. As shown in Figure 3.7, this device translates the 3.3V logic from the RP2040 to 5V servo compatible control signals while keeping the microcontroller electrically isolated from the higher voltage servo domain. Separate 5V pins on the headers power the servos directly from the regulated rail, allowing the TVC actuators to draw substantially more current than the logic without disturbing microcontroller operation. For ignition and recovery, the board incorporates two AO3400 N-channel MOSFET outputs labeled IGNITER and EJECTION, which switch battery voltage into the motor igniter and ejection charge under software control.

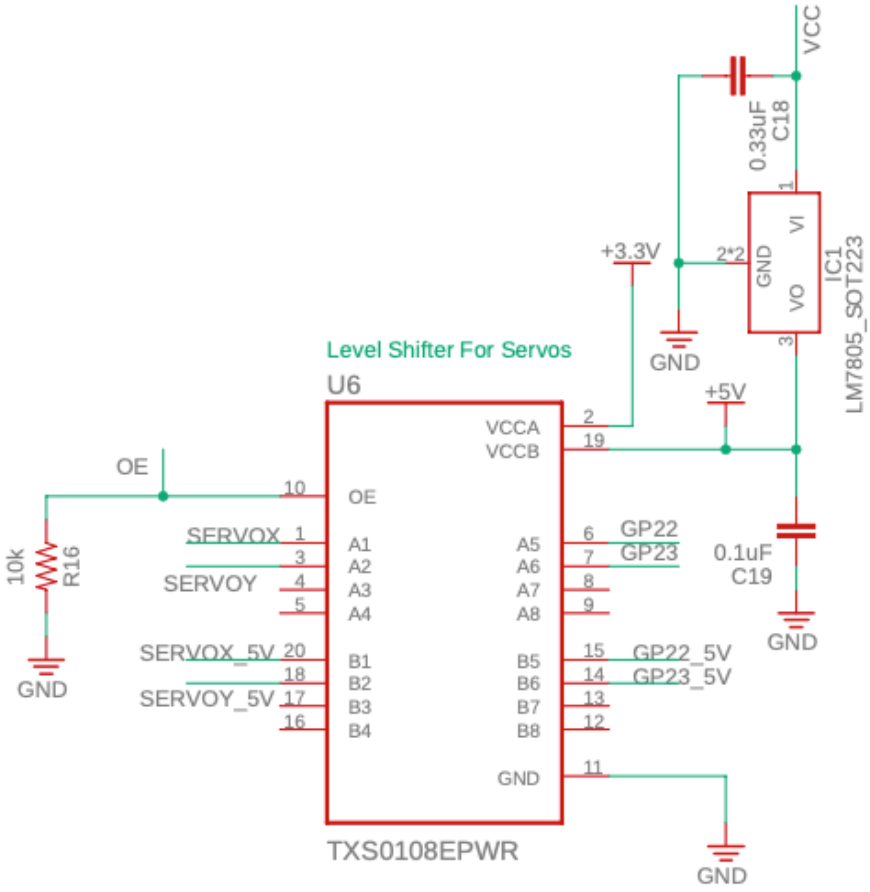


Figure 3.7 - PCB layout for TXS0108E bidirectional level shifter [18]

Finally, the PCB exposes several interfaces that facilitate development, testing, and integration into the rocket. As shown in Figure 3.8, a micro USB connector ties directly into the RP2040's USB D+ and D– pins, providing a convenient pathway for programming the flight software and downloading logged flight data after recovery. A small SWD header breaks out the RP2040's debug pins alongside ground, enabling in circuit debugging and firmware reprogramming when necessary. Additional GPIO pins are routed to a 1×7 expansion header for future payloads or auxiliary sensors. Collectively, these layout choices yield a flight computer that is compact yet fully featured, with clearly separated power domains and short, direct signal paths between the RP2040, sensors, and actuators, thereby improving robustness in the harsh environment of a model rocket launch.

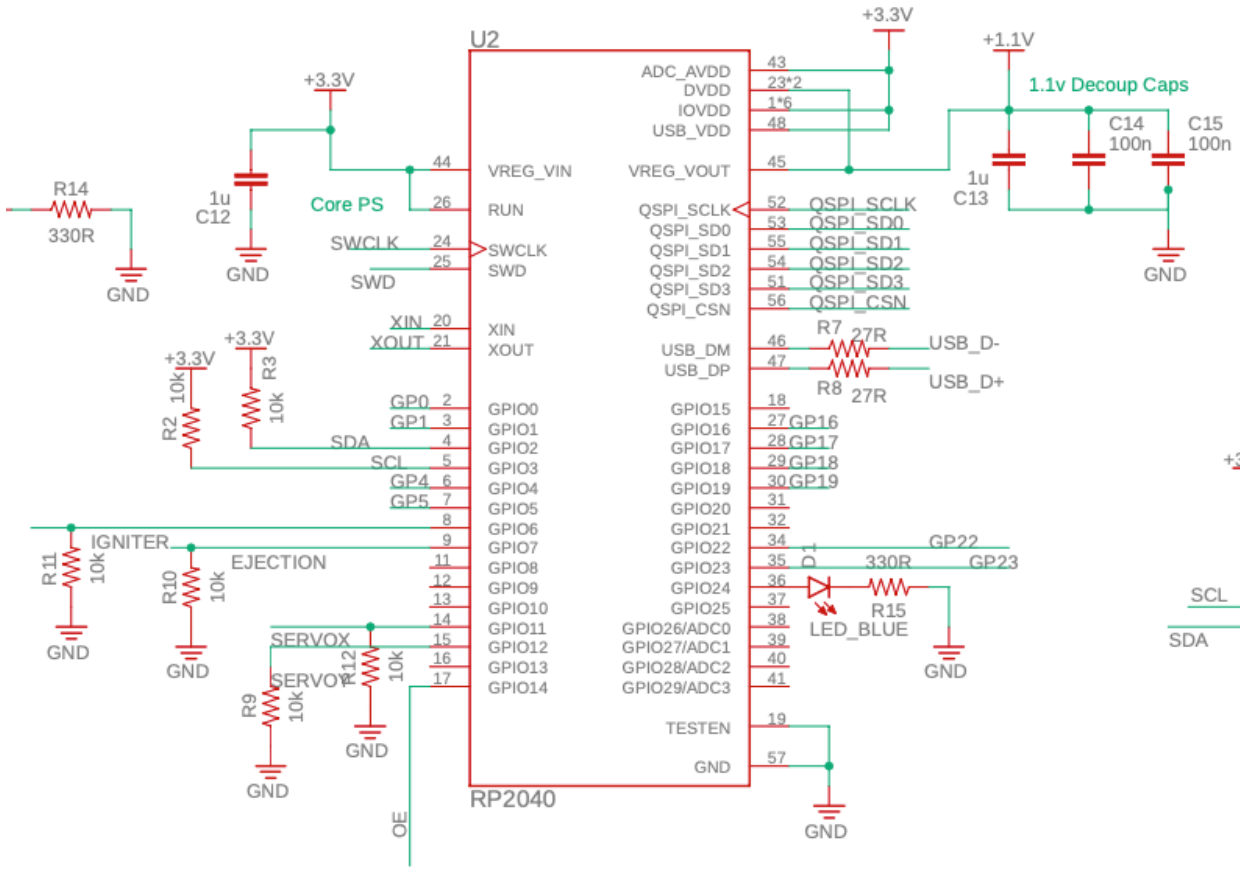


Figure 3.8 - PCB layout for RP2040 microcontroller [18]

3.6 Flight Cycle

The flight of a TVC rocket is governed by a sequence of software states, each corresponding to a distinct phase of the mission. The sequence begins with the start and initialization state, during which the flight computer powers on and initializes all onboard systems, including the IMU, barometer, servos, and other core peripherals. Following initialization, the servo and sensor check state performs diagnostics to ensure that the gimbal servos are responsive and that all critical sensors are communicating properly. Once system integrity is confirmed, the rocket transitions into the flight mode arming state, preparing the flight computer to monitor for launch conditions.

Immediately thereafter, the rocket transitions into the ascent stabilization state, during which the TVC system employs a PID controller to continuously adjust the gimballed motor, countering disturbances and maintaining a vertical ascent profile. Throughout powered flight, the control loop reads sensor data, computes attitude error, processes these errors through the PID controller, and outputs corresponding servo commands at a rate limited by servo response time and the flight computer's processing cycle. This continuous feedback action is essential for minimizing drift, preserving stability, and preventing attitude loss during the ascent.

After motor burnout and a little bit more time for the rocket to coast based on its own momentum the parachute ejection cap is ignited deploying the parachute. Following parachute deployment, the rocket enters the descent state, during which it freely descends under canopy. No thrust or TVC control is applied, and the rocket relies entirely on aerodynamic stability for a safe return. Finally, the sequence concludes with the landing state, where the rocket makes ground contact and the flight computer ceases active operations. Figure 3.9 illustrates the complete flight cycle, providing a visual overview of each software state and their progression.

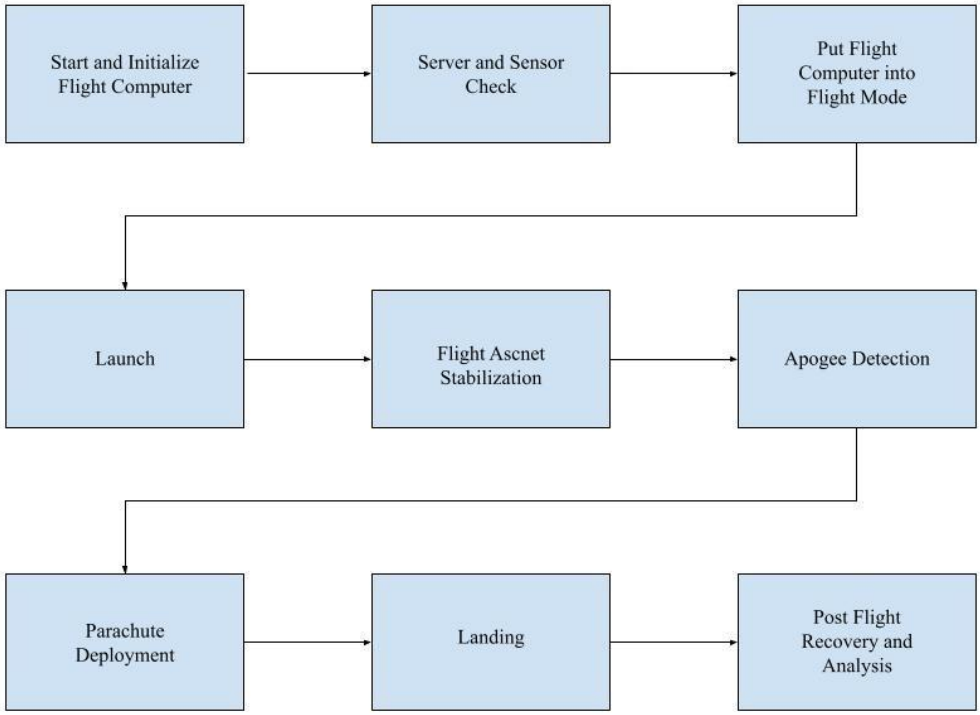


Figure 3.9 - Rocket flight cycle

4. Rocket Dynamics

Accurate characterization of a rocket's body dynamics is essential for predicting its motion and developing an effective control architecture. Rocket dynamics describe the way external forces and moments influence both translational and rotational behavior. The dominant forces acting on the vehicle consist of motor thrust, aerodynamic loading, and gravity. These effects become especially significant in TVC systems, where small angular deviations in nozzle orientation can produce noticeable changes in the rocket's response and overall stability.

A rigorous dynamic model of a TVC rocket requires two key elements. The first is a clearly defined set of reference frames that allows consistent representation of positions, velocities, orientations, and forces relative to the Earth and the rocket body. The second element if the application of Euler's rigid body rotational equations which relate applied torque to angular motion through the rocket's mass moment of inertia properties. Together, these components establish the theoretical framework necessary for analyzing the rotational behavior of a TVC model rocket.

4.1 Reference Frames

References frames provide the foundational structure for describing the motion of a rocket, as they establish consistent coordinate system for defining positions, velocities, and accelerations. A clear distinction between reference frames is essential because external forces and moments must be expressed in a form that aligns with the vehicle's translational and rotational equations of motion. Without well defined frames, it becomes difficult to quantify the effects of thrust, gravity, and aerodynamic loading. Establishing appropriate reference frames ensures that these forces can be systematically projected onto the rocket's axes. This enables accurate calculation of its dynamic behavior.

The inertial reference frame, commonly referred to as the Earth fixed frame, remains stationary relative to the Earth and does not translate or rotate with the rocket. This frame provides a global coordinate system for defining absolute position, velocity, and trajectory. In the present model, the inertial frame is defined with the y-axis aligned upward, the x-axis pointing horizontally to the right, and the z-axis directed out of the page towards the observer. Within this frame, gravitational acceleration acts along a fixed direction, and environmental effects such as aerodynamic forces can be consistently evaluated. A visual representation of this inertial frame is provided in Figure 4.1.

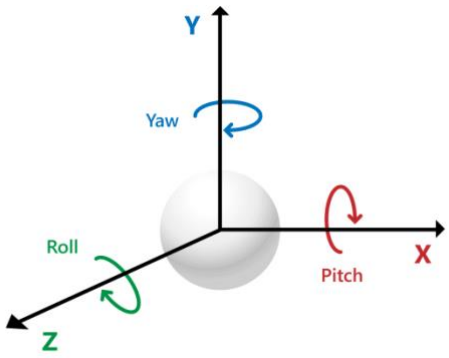


Figure 4.1 – Inertial reference frame

To characterize the rocket's orientation relative to the inertial frame, Euler angles roll (ϕ), pitch (θ), and yaw (ψ) are used. These angles describe sequential rotations about the rocket's body fixed axes: roll about the body z-axis, pitch about the body x-axis, and yaw about the body y-axis. Although the axes themselves move with the rocket, the Euler angles define its orientation relative to the Earth. This establishes a connection between the vehicle's body centric behavior and its global trajectory. The associated angular velocity components denoted p, q, r for roll, pitch, and yaw rates respectively. These body frame angular velocities are essential for modeling rotational dynamics, applying aerodynamic moments and implementing control strategies.

The body reference frame is fixed to the rocket and translates and rotates with the vehicle. In this frame forces such as thrust, aerodynamic loads, and gimbal induced torques align naturally with the vehicle's geometry. This enables straightforward computation of linear and angular accelerations along its principal axes. For the present configuration the body frame uses the same axis orientation as the inertial frame: the y-axis along the rocket's longitudinal direction, the x-axis across its width, and the z-axis perpendicular to the page. The body reference frame can be seen in Figure 4.2.

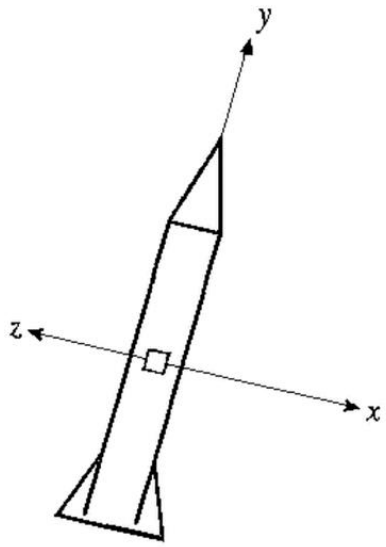


Figure 4.2 – Body reference frame

Transforming forces and moments between the inertial and body frames is essential for predicting the rocket's response to control inputs such as thrust vector gimbal actuation and for accurately modeling both translational and rotational motion. By defining both inertial and body reference frames, the rocket's motion can be described consistently in global and vehicle centric terms. This dual frame representation ensures that external forces and moments are interpreted correctly, and that the resulting translational and rotational responses of the rocket remain physically consistent throughout the analysis.

4.2 Euler's Rigid Body Equation and Dynamics

The rotational motion of a rocket can be described using Euler's rigid body equation, which establishes the relationship between applied external torques and the resulting angular accelerations. In the body fixed reference frame, the governing equation is expressed in vector form as shown in Equation 4.1. This form captures both the inertial resistance to changes in angular velocity and the gyroscopic coupling that arises when a rigid body rotates with nonuniform mass distribution.

$$I\,\dot{\omega} + \omega \times (I\,\omega) = \tau \tag{4.1}$$

In this expression, I denotes the inertia tensor, ω represents the body frame angular velocity vector, and τ is the total external torque applied about the rocket's center of mass. The term Iω̇ characterizes the angular acceleration response due to inertia, while the cross product term ω × (Iω) accounts for gyroscopic effects generated by rotational motion. When the inertia tensor is diagonal, an assumption appropriate for rockets with approximate body symmetry, Euler's vector equation can be expanded into three coupled scalar differential equations, shown in Equations 4.2 and 4.3.

$$I = \begin{bmatrix} I_x & 0 & 0 \\ 0 & I_y & 0 \\ 0 & 0 & I_z \end{bmatrix} \tag{4.2}$$

$$\begin{aligned} I_x\dot{\omega}_x &= \tau_x + (I_y - I_z)\,\omega_y\omega_z \\ I_y\dot{\omega}_y &= \tau_y + (I_z - I_x)\,\omega_x\omega_z \\ I_z\dot{\omega}_z &= \tau_z + (I_x - I_y)\,\omega_x\omega_y \end{aligned} \tag{4.3}$$

For rockets with nearly axisymmetric geometry, such as slender cylindrical airframes with mass distributed uniformly around the longitudinal axis, the lateral moments of inertia satisfy $I_x \approx I_z$**.** This symmetry arises because rotations about the x-axis and z-axis sweep through identical circular cross sections, each having the same radius and mass distribution. This symmetry arises because rotations about the x-axis and z-axis sweep through identical circular cross sections, each having the same radius and mass distribution. As a result, the rocket exhibits nearly identical resistance to angular acceleration in the pitch and roll directions**,** while the yaw axis inertia I_y differs due to the rocket's length being much greater than its diameter. When internal components and structural features are arranged symmetrically around the central axis, the approximation $I_x = I_z$ becomes appropriate**,** simplifying the dynamic model and aligning with standard assumptions for axisymmetric rigid bodies.

$$\tau = r \times F \tag{4.4}$$

The motor is positioned a distance L below the center of mass along the body y-axis, thus yielding a position vector of $r = [0, d, 0]^T$ in the body frame. Small gimbal deflections δ_x and δ_z introduce

lateral components into the primarily axial thrust. Under the small angle approximation, the thrust vector may be modeled as a dominant component along the negative body y-axis, with small lateral components proportional to the gimbal deflections. This leads to the thrust vector form in Equation 4.5.

$$F_T = \begin{bmatrix} T\,\delta_x \\ -T \\ T\,\delta_z \end{bmatrix} \tag{4.5}$$

By substituting r and F into Equation 4.4, the resulting torque components about the body axes are obtained. These torques directly govern roll and pitch motions and are given in Equation 4.6.

$$\begin{aligned} \tau_x &= dT\,\delta_z \\ \tau_y &= 0 \\ \tau_z &= dT\,\delta_x \end{aligned} \tag{4.6}$$

These torque expressions highlight the inherent cross axis coupling produced by thrust vector control. Because torque is defined by the cross product $\tau = r \times F$, a lateral deflection of the thrust vector in one direction produces a rotational moment about the perpendicular body axis. Thus, a gimbal deflection δ_z generates a roll moment about the x-axis, while δ_x produces a pitch moment about the z-axis. This perpendicular relationship is a direct consequence of the vector cross product and forms the basis for attitude control via gimbal actuation. Substituting the derived torque expressions into Euler's scalar equations yields the nonlinear rotational equations of motion for the rocket, represented in Equation 4.7.

$$\begin{aligned} \dot{\omega_x} &= \frac{dT}{I_x}\,\delta_z \\ \dot{\omega_y} &= 0 \\ \dot{\omega_z} &= \frac{dT}{I_z}\,\delta_x \end{aligned} \tag{4.7}$$

To express the linearized rotational dynamics of the gimballed rocket in state space form, we begin by selecting a state vector that captures both the roll and pitch orientations of the vehicle together with their corresponding angular rates. For small angular rates and small deviations from the upright equilibrium, the gyroscopic coupling terms in Equation 4.7 become negligible. Under this assumption, the nonlinear rotational equations reduce to linear relations in which the angular accelerations depend directly on the applied gimbal deflections. To express these dynamics in a form appropriate for control design, the equations must be rewritten in first order form. To express these dynamics in a form appropriate for control design, the equations must be rewritten in first order form. The rotational dynamics can be recast into a four state representation describing the evolution of roll angle, pitch angle, and their corresponding angular rates.

The resulting state vector is defined as $x = \left[\, \phi,\ \dot{\phi},\ \theta,\ \dot{\theta} \,\right]^T$, where ϕ and θ denote roll and pitch angles and $\dot{\phi}$ and $\dot{\theta}$ correspond to the body axis angular velocities. The control inputs are the commanded gimbal deflections, written as $u = [\, \delta_x,\ \delta_z \,]^T$, which correspond to the nozzle deflections about the body x-axis and z-axis respectively. Substituting these definitions into the

linearized dynamics yields the standard control form, $\dot{x} = Ax + Bu$, where matrices A and B are shown in Equation 4.8 and 4.9.

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{4.8}$$

$$B = \begin{bmatrix} 0 & 0 \\ 0 & \frac{dT}{I_x} \\ 0 & 0 \\ \frac{dT}{I_z} & 0 \end{bmatrix} \tag{4.9}$$

To complete the representation, an output equation is required to identify the states measured or fed back for attitude stabilization. Since roll and pitch angles are the primary quantities used for feedback, the output vector is defined as $y = [\phi, \theta]^T$, obtained from the general relation $y = Cx + Du$, where C selects the first and third elements of the state vector and D is a zero matrix because the control inputs do not appear directly in the measured outputs. The corresponding C and D matrices are also included in Equation 4.10.

$$C = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{4.10}$$

The rotational dynamics of the rocket were formulated by beginning with Euler's rigid body equation and incorporating the torques produced by the gimballed thrust vector. By expressing the governing relation in vector form, expanding it into the corresponding scalar equations, and then applying the small angle assumptions the resulting nonlinear dynamics were reduced to a simplified set of linear expressions suitable for analysis. This linear approximation was subsequently rewritten in state space form, using roll and pitch angles together with their angular rates. With the dynamics now expressed in this standardized form, the system is directly compatible with modern control design methods.

4.3 Pitch Axis State Space Representation

Although the rocket's thrust vector controlled rotational dynamics are inherently coupled in roll and pitch, controller development is simplified by isolating a single rotational axis. In this work, the system is reduced to pitch only motion so that the dominant attitude instability can be analyzed with a plant model. This reduction preserves the essential thrust vectoring physics while producing a low order representation that is easier to evaluate, simulate, and use for PID, LQR, and LQG design. Starting from Euler's rigid body equation, shown in Equation 4.1, the rotational dynamics are considered about the pitch axis. Under the small angle assumption and near vertical operating conditions, gyroscopic coupling terms are negligible, and the thrust vector provides the dominant pitching moment. Using the torque relationship developed in Section 4.2, the pitch equation of motion reduces to the scalar form shown in Equation 1 below. where I is the pitch axis moment of

inertia, d is the distance from the center of gravity to the motor, T is the thrust magnitude, and $\delta(t)$ is the commanded gimbal deflection. Equation 4.11 shows that the pitch dynamics behave as a double integrator with no inherent damping or restoring force. To express the pitch dynamics in a form suitable for control analysis, Equation 4.11 is rewritten in state space from.

$$I\,\ddot{\theta}(t) = d\,T\,\delta(t) \tag{4.11}$$

The pitch only state vector is defined as $x_p(t) = \left[\theta(t)\ \dot{\theta}(t)\right]^T$ and the control input as $u(t) = \delta(t)$. Due to this the entire system can be written as $\dot{x_p}(t) = A_p\,x_p(t) + B_p\,u(t)$. For attitude stabilization, the measured output of interest is the pitch angle. The output equation is therefore defined as $y(t) = C_p x_p(t)$. The corresponding A, B, and C matrices can be seen below in Equation 4.12, 4.13, and 4.14 respectively.

$$A_p = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \tag{4.12}$$

$$B_p = \begin{bmatrix} 0 \\ \dfrac{d\,T}{I} \end{bmatrix} \tag{4.13}$$

$$C_p = [1 \quad 0] \tag{4.14}$$

With the reduced order pitch model being defined classical controllability and observability analysis can be applied. Controllability is assessed by forming the controllability matrix as shown in Equation 4.15. By subsisting the appropriate matrices, we can see that the system as full rank for nonzero thrust and moment arm, confirming that the gimbal input can influence both pitch angle and pitch rate.

$$Co = [B_p \quad A_pB_p] = \begin{bmatrix} 0 & \dfrac{d\,T}{I} \\ \dfrac{d\,T}{I} & 0 \end{bmatrix} \tag{4.15}$$

Observability is evaluated by forming the observability matrix shown in Equation 4.16. By substituting the appropriate matrices into the observability matrix, we can see that the system is also full rank, demonstrating that the pitch states can be reconstructed from pitch angle measurements alone.

$$Ob = \begin{bmatrix} C_p \\ C_pA_p \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \tag{4.16}$$

Since the system is both fully controllable and full observable, we can take the Laplace transform of the pitching equations of motion to find the pitch axis transfer function. Equation 4.17 explicitly highlights the unstable double integrator structure of the thrust vector controlled pitch dynamics.

While this reduced order model neglects thrust variation, translational motion, and aerodynamic effects, it retains the controllability and observability properties of the full rotational system while significantly simplifying analysis. This pitch axis representation therefore serves as the foundation for the open loop analysis and the PID, LQR, and LQG controller designs presented in the following chapter.

$$\frac{\Theta(s)}{\Delta(s)} = \frac{d\,T/I}{s^2} \tag{4.17}$$

5. MATLAB System Simulation

MATLAB and Simulink provide a flexible environment for modeling, analyzing, and validating the rotational dynamics of the TVC rocket. Simulation tools are essential in the development of the control system because they enable rapid evaluation of controller performance under a wide range of operating conditions without requiring physical flight tests**.** By constructing mathematical models directly from the linearized equations of motion derived in Chapter 4, the simulation framework serves as a bridge between theoretical analysis and practical controller implementation.

The simulation workflow begins with an examination of the rocket's open loop behavior**,** which characterizes the natural response of the system to thrust vector inputs without any feedback control applied. This provides insight into the system's stability properties, dominant modes, and sensitivity to gimbal actuation. Following the open loop analysis, a 3DOF body axes model is implemented in Simulink to capture the coupled translational and rotational effects experienced during flight. This model enables realistic evaluation of controller performance in an environment that incorporates thrust variation, gravitational effects, and aerodynamic loading. Subsequent sections apply this modeling environment to the design and evaluation of several control strategies. A conventional PID controller is first developed to provide baseline stabilization performance. Building on this foundation, a LQR controller is designed to exploit state space optimal control techniques, followed by the LQG controller**,** which combines LQR with a Kalman filter for state estimation under measurement noise. Together, these simulations demonstrate how feedback control laws can be systematically developed and validated prior to integration into the embedded flight computer.

5.1 System Variables and Physical Parameters

The system simulation relies on a well defined set of physical parameters and actuator limits that determine the rocket's dynamic response under thrust vector controlled flight. These parameters are based on vehicle properties and servo actuator performance and are used consistently across all simulation environments. To ensure accuracy and repeatability, the key variables used by the dynamic model, control laws and force moment computations are summarized in Table 5.1.

Table 5.1 – Model rocket system variable

Name	Symbol	Value	Unit
Rocket Mass	m	1	kg
Pitch Axis Moment of Inertia	I	0.05	$kg \cdot m^2$
Moment Arm	d	0.485	m
Simulation Time Step	dt	0.01	s
Servo Reaction Time	t_{servo}	0.05	s
Maximum Gimbal Angle	δ_{max}	± 5	$degrees$
Estes F15 Average Thrust	T_{avg}	15	N

The rocket's mass, principal moments of inertia, and thrust vector moment arm define its fundamental resistance to rotational acceleration and strongly influence the plant dynamics. In the 3DOF Simulink model, these parameters govern the rigid body translational and rotational equations, which are integrated at a fixed sample time of 0.01 seconds. The pitch attitude computed by this model serves as the primary feedback signal for evaluating controller performance, while the commanded pitch setpoint defines the desired attitude for stabilization.

A key distinction between the simulation environments lies in how thrust is represented. The 3DOF model incorporates a time varying thrust curve corresponding to the Estes F15 motor profile shown previously in Figure 2.2, ensuring that the simulation reflects realistic motor behavior throughout the burn. In contrast, the transfer function based controllers (PID, LQR, and LQG) uses an average constant thrust level of 15 N. This simplification is required because linear state space models depend on fixed system matrices and therefore the thrust term must remain constant. Using an average thrust value ensures that these controllers remain effective over the majority of the flight. As shown in Figure 2.2, following the initial thrust spike, the motor output stabilizes for the remainder of the burn duration. While the initial peak thrust may introduce control challenges due to increased control authority, the average thrust provides a representative approximation of mid burn operating conditions.

Section 5.2 Open Loop Response

The pitch subsystem behaves mathematically as a double integrator, meaning that any nonzero input produces an angular acceleration that continuously integrates into pitch rate and ultimately into pitch angle. Due to no natural restoring forces or aerodynamic damping act on the rocket in this configuration, the motion is inherently unstable and grows without bound. To illustrate this behavior, an open loop Simulink model was constructed using the linearized state space representation derived in Chapter 4. The model layout shown in Figure 5.1 applies a constant gimbal deflection rather than a step command allowing the uncontrolled plant dynamics to be observed directly. This constant input is used because the 3DOF block introduced in the following section does not accept step inputs, ensuring consistency and enabling direct comparison across subsequent simulations.

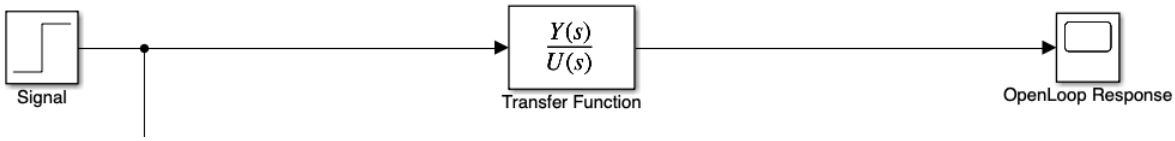


Figure 5.1 – Simulink open loop model

The resulting time response, presented in Figure 5.7, exhibits the characteristic parabolic growth expected from a double integrator system. The thrust vectoring torque generates a constant angular acceleration, which causes the pitch rate to increase linearly and the pitch angle to diverge quadratically. In the absence of feedback, nothing counteracts this acceleration, so the rocket continually rotates faster while drifting farther from its nominal vertical orientation. This behavior reflects a fundamental characteristic of TVC rockets, where the absence of an inherent restoring

moment about the pitch axis causes even small disturbances to result in unbounded attitude divergence.

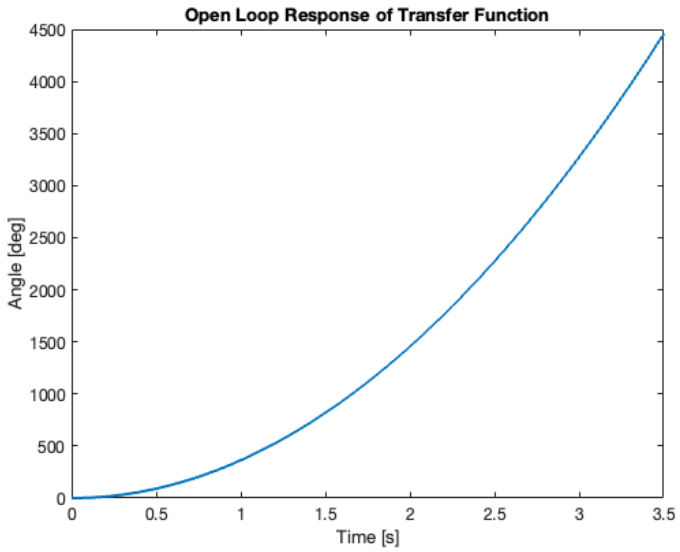


Figure 5.2 – Open loop response

The open loop response therefore emphasizes the necessity of closed loop control. Without an active feedback controller to regulate the thrust vector and oppose unwanted angular motion, the rocket cannot maintain a stable ascent and will continue rotating indefinitely. This motivates the subsequent development of PID, LQR, and LQG controllers, each designed to introduce the stabilizing dynamics that the open-loop system lacks.

5.3 MATLAB 3DOF Rigid Body Model

The MATLAB 3DOF rigid body block provides a simplified, computationally efficient representation of the rocket's planar motion while still capturing the essential translational and rotational dynamics needed for control analysis. As shown in Figure 5.3, the model constrains motion to the X–Z plane and allows rotation about the body y-axis, corresponding to pitch. The block accepts three inputs: the lateral body axis force F_x, the axial force F_z, and the pitching moment M_y. These quantities are derived from thrust vector deflection and are sufficient to propagate the vehicle's motion over time.

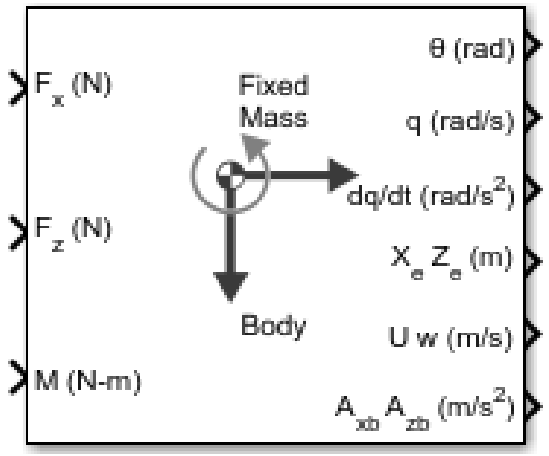


Figure 5.3 – MATLAB 3DOF Rigid Body block

The translational dynamics are governed by Newton's second law expressed in the body frame. Let U and W denote the forward and vertical velocity components respectively. The force components F_x and F_z, along with the gravitational acceleration resolved in the body frame, determine the accelerations $\dot{U}$ and $\dot{W}$. These fundamental equations can be seen in Equation 5.1 below.

$$m\dot{U} = F_x + mgsin(\theta)$$

$$m\dot{W} = F_z - mgcos(\theta) \tag{5.1}$$

$$I_y\dot{q} = M_y$$

Here, θ is the pitch angle and q is the pitch rate. The first two equations describe translation in the X–Z plane, while the third represents the rotational dynamics about the body y-axis. The pitching moment M_y arises from the thrust vector being applied at a distance from the center of mass, and it dictates the evolution of the pitch rate. Pitch control is achieved by commanding a desired pitch orientation $\theta_{\text{cmd}}(t)$, which is compared to the measured pitch angle $\theta(t)$ to form the tracking error $e(t) = \theta_{\text{cmd}}(t) - \theta(t)$. This error drives a PID controller, which computes the required pitching moment $M_y(t)$ shown in Equation 5.2.

$$M_y(t) = K_p e(t) + K_{i\int e(t)dt} + K_d \frac{de(t)}{dt} \tag{5.2}$$

The complete control architecture used to regulate the rocket's pitch dynamics is shown in Figure 5.4, which presents the fully integrated Simulink model incorporating the 3DOF Rigid body dynamics block and the closed loop PID controller. The total system will be broken down in later paragraphs to thoroughly analyze what each component does. This diagram represents the full feedback loop implemented for thrust vector control and serves as the foundation for all subsequent simulations and controller evaluations. At a high level, the model captures the typical workflow of an attitude control system in which a target pitch orientation is specified, the actual pitch angle is

continuously measured by the dynamics block, and the resulting error between these signals is used by the PID controller to generate corrective gimbal commands.

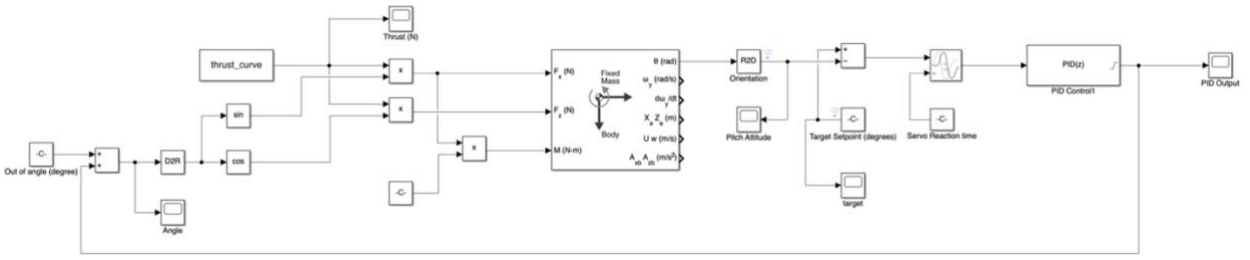


Figure 5.4 - Total Simulink control system for pitch control

Figure 5.5 illustrates the thrust angle decomposition subsystem, which serves as the first stage of the control architecture by converting the commanded gimbal angle into the forces and pitching moment acting on the rocket. The PID gimbal command is summed with any offset, converted from degrees to radians, and then routed to both a monitoring scope and sine and cosine blocks for computing the thrust vector components. These angular components are multiplied by the instantaneous thrust obtained from the thrust curve, producing the lateral and axial force components in the body frame, while a separate multiplication using the same thrust value, the sine of the deflection angle, and the moment arm generates the pitching moment applied to the rigid body dynamics block. Through this sequence, the subsystem transforms a scalar thrust magnitude and a commanded nozzle angle into the complete set of forces and moments needed for the 3DOF simulation.

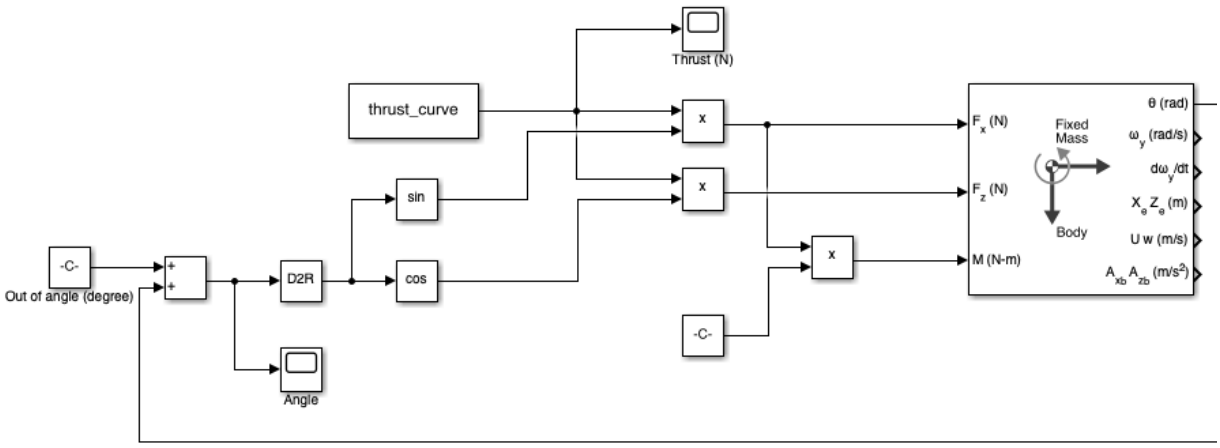


Figure 5.5 - Force and angle decomposition for 3DOF block

The 3DOF rigid body dynamics block receives the body frame forces F_x and F_z, along with the pitching moment M_y, generated by the thrust angle decomposition subsystem and uses these quantities to propagate the rocket's translational and rotational motion. As described in Figure 5.3,

the applied forces determine the horizontal and vertical accelerations, while the pitching moment governs the angular acceleration about the rocket's center of mass. These signals are numerically integrated to form the complete state vector, which includes the inertial positions (X_e, Z_e), body frame velocities (U,W), pitch rate q, and pitch angle θ. Although the block computes all of these states, the pitch angle serves as the primary feedback variable for attitude control, while the remaining states are used for diagnostic evaluation and verification of simulation behavior.

The pitch angle produced by the 3DOF dynamics block is converted from radians to degrees and routed into the control loop, where it is compared directly with the commanded setpoint to compute the pitch tracking error. This process is illustrated in Figure 5.6. The resulting error signal is first passed through a servo reaction time model, which accounts for actuator latency and limited bandwidth, ensuring that the simulated gimbal response more closely reflects the behavior of the physical thrust vectoring system. The output of this servo model is then processed by the PID(z) controller, which computes the gimbal deflection required to minimize the attitude error. This commanded deflection is simultaneously recorded for analysis and fed back into the thrust angle decomposition subsystem, thereby completing the closed loop thrust vector control architecture.

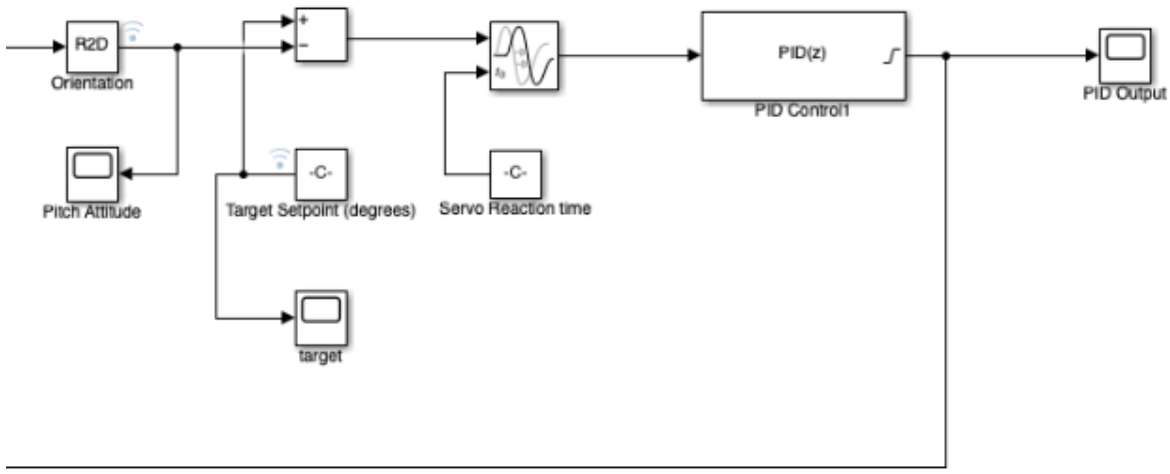


Figure 5.6 – Pitch attitude feedback path with servo reaction time model and PID(z) controller

The PID controller was tuned using the Simulink model by evaluating the system at thrust levels representative of the rocket's operation. The initial tuning was performed near 15 N, which corresponds to the mid burn region of the motor and provides a stable basis for identifying gains that balance responsiveness and damping. However, when these gains were tested at higher thrust levels, particularly near 20 N, the closed loop response became unstable due to the increased control authority associated with higher thrust. To resolve this issue, a second tuning was conducted at the peak thrust level of 25 N, representing the most demanding operating condition. The resulting gains from the mid burn and peak thrust tuning sessions were interpolated across the full thrust curve to form the PID values seen in Table 5.2. This approach ensured that the controller

remained responsive during low thrust periods while maintaining stability during high thrust transients.

Table 5.2 – MATLAB PID values for pitch control

K_p	K_i	K_d
0.35	1.10	0.075

Figure 5.7 illustrates the closed loop pitch response obtained using this gain set. As shown, the system experiences an initial overshoot before converging to the 5° target angle. This overshoot is caused by the characteristic thrust spike of the Estes F15 motor immediately after ignition, which temporarily increases the effective loop gain and leads to a more aggressive pitch correction. As the thrust decays toward its mid burn value, the control authority diminishes, producing a well damped response. The PID output curve reflects this behavior, showing large corrective actions during the high thrust transient that gradually shrink as the thrust level decreases and the pitch error approaches zero.

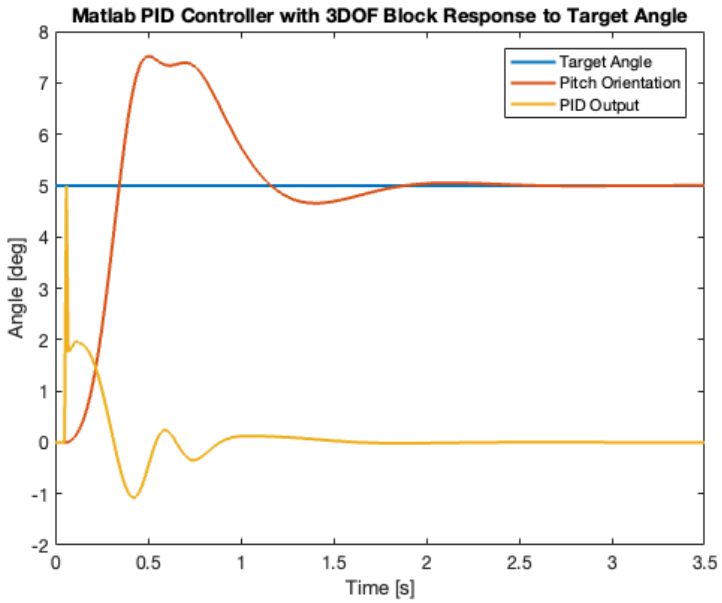


Figure 5.7 – 3DOF closed loop response for pitch control to target angle

A MATLAB and Simulink dynamic model was created to evaluate the thrust vector control system and tune the PID controller prior to flight. The simulation focuses on pitch stabilization while still propagating full 3DOF translational and rotational motion for diagnostic purposes. The Simulink architecture integrates three core components: a thrust angle decomposition subsystem that converts gimbal commands into body frame forces and pitching moment; a 3DOF rigid body block that computes the rocket's motion; and a PID controller with a servo reaction time model that generates corrective thrust vectoring inputs. The resulting closed loop response shows an expected initial overshoot due to the ignition thrust spike followed by smooth convergence to the commanded pitch angle.

5.4 PID Transfer Function Response

To evaluate the closed loop behavior of the thrust vectoring controller, the reduced pitch transfer function derived earlier was paired with a PID controller implemented in Simulink. This configuration provides a simplified yet dynamically representative model for controller assessment. This formulation allows the controller's performance to be examined in isolation from the full 3DOF dynamics. The corresponding block diagram, shown in Figure 5.8, processes the orientation error through proportional, integral, and derivative paths, whose outputs are summed and applied to the reduced plant model. In this configuration, the PID controller provides the necessary damping and stiffness that the open loop system inherently lacks, enabling stable regulation of pitch motion.

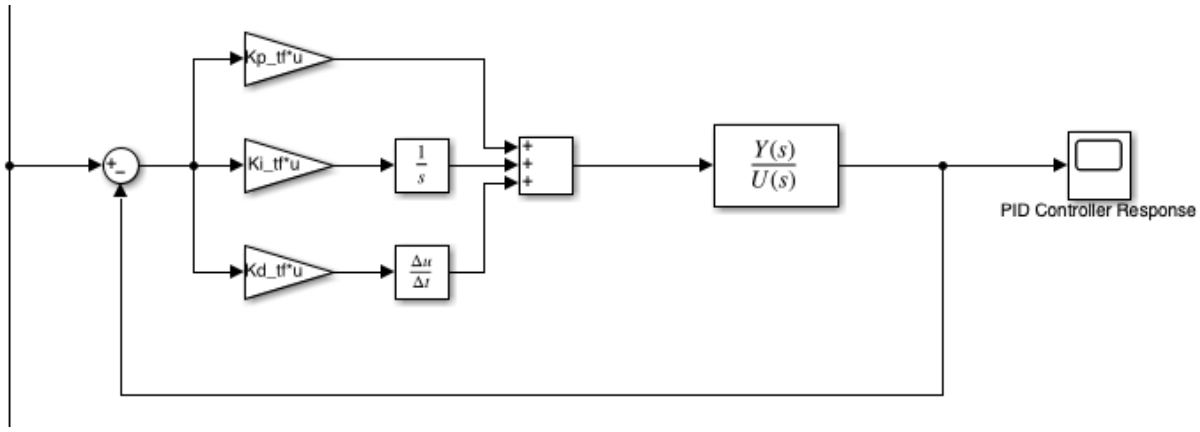


Figure 5.8 – PID Simulink model with transfer function

The PID controller was implemented using the gains shown in Table 5.2. Although the 3DOF and transfer function models were tuned independently during preliminary testing, the resulting values were found to be highly consistent, and thus a common set of gains was adopted for both simulations. Using identical controller parameters ensures that differences in behavior between the models arise from the underlying system dynamics rather than from variations in controller tuning, enabling a clean and direct comparison.

The closed loop response of the transfer function model is presented in Figure 5.9. The PID controller effectively regulates the commanded pitch angle, producing a modest overshoot followed by a well damped convergence to the desired steady state value. This behavior stands in clear contrast to the unstable open loop response discussed in Section 5.3, demonstrating that proper PID action provides the stabilizing authority required to regulate pitch motion in thrust vector controlled rockets.

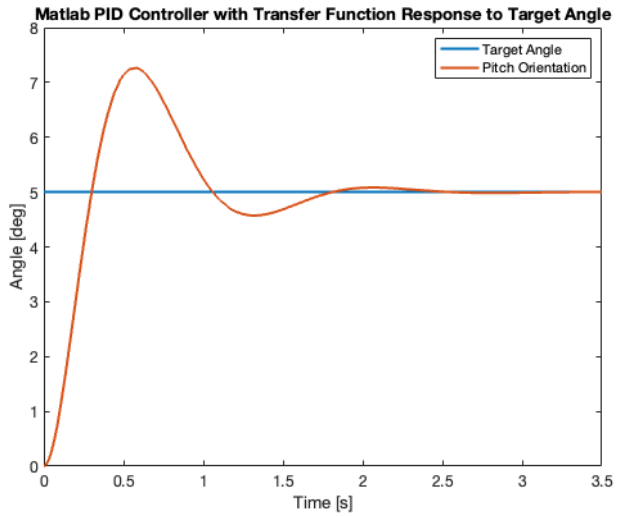


Figure 5.9 – PID controller with transfer function response to target angle

To assess how the simplified transfer function model compares with a higher fidelity representation, the same PID gains were applied to the full 3DOF Simulink model. The resulting comparison is shown in Figure 5.10. Both models exhibit similar rise times, overshoot characteristics, and settling behavior, indicating that the reduced model captures the essential pitch dynamics relevant to controller performance. However, small differences between the two responses arise due to the additional physical effects present in the 3DOF system. In the full model, the thrust entering the rotational equations is taken directly from the Estes F15 thrust curve, which features a pronounced initial spike followed by a lower but more stable thrust for the remainder of the burn. In contrast, the transfer function model uses a constant representative thrust value, which simplifies the dynamics but eliminates these variations. Despite this difference, the overall behavior remains consistent across the two models, confirming that the reduced transfer function captures the dominant structure of the pitch dynamics and provides a reliable foundation for controller design.

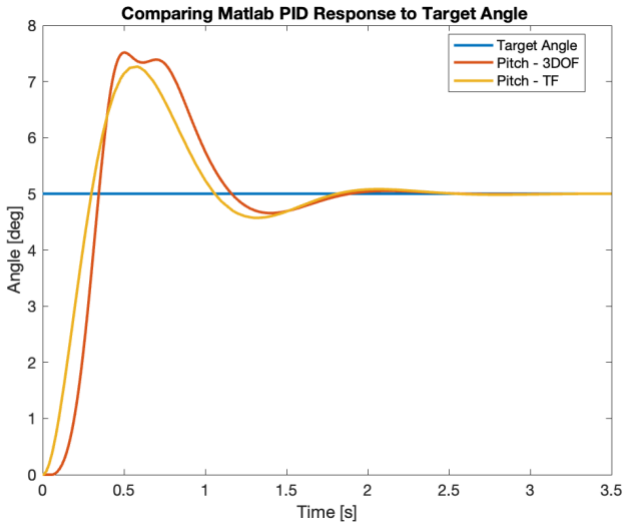


Figure 5.10 – Comparing MATLAB PID response to target angle

5.5 LQR Response

In addition to the PID controller evaluated in the previous section, a Linear Quadratic Regulator (LQR) was implemented to investigate the performance of an optimal state feedback approach for stabilizing the pitch dynamics. Because the pitch subsystem is fully controllable, as established in Section 5.2, it is well suited for LQR design. The corresponding Simulink implementation is shown in Figure 5.11.

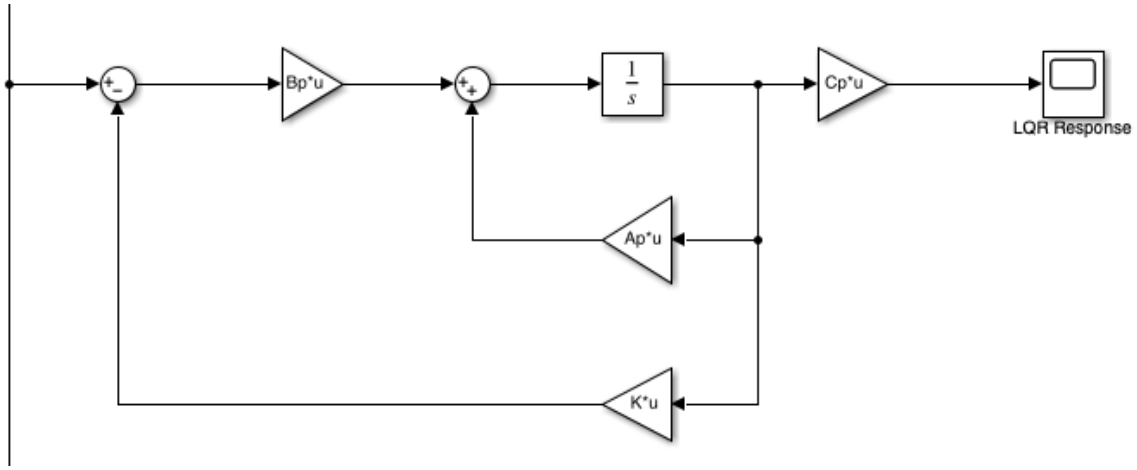


Figure 5.11 – LQR Simulink model for pitch control

To compute the optimal gains, the LQR algorithm was applied using the weighting matrices shown in Equation 5.6 and 5.7. The Q_{lqr} and R_{lqr} matrices define the relative penalties to pitch angle deviation, pitch rate deviation, and control effort. The larger weight on pitch angle reflects the priority placed on maintaining accurate orientation, while the more modest weighting on pitch rate encourages smooth dynamic behavior. The scalar R_{lqr} penalizes large gimbal deflections, ensuring that the control effort remains efficient and avoiding unnecessarily aggressive actuation. Solving the algebraic Riccati equation with these selections yields the optimal state feedback gain matrix K_{lqr}, which is shown in Equation 5.8, forming an optimal state feedback controller that minimizes state deviation and control effort.

$$Q_{lqr} = \begin{bmatrix} 10 & 0 \\ 0 & 1 \end{bmatrix} \tag{5.6}$$

$$R_{lqr} = 10 \tag{5.7}$$

$$K_{lqr} = [1.000 \quad 0.3373] \tag{5.8}$$

The resulting closed loop LQR response is shown in Figure 5.12. Compared to the PID controller, which exhibited an underdamped transient with modest overshoot, the LQR response is monotonic and overdamped. The pitch angle converges smoothly toward the commanded value without oscillation, reflecting the influence of the quadratic cost function, which explicitly discourages aggressive or high frequency control actions. This behavior demonstrates the stabilizing capability of LQR when applied to the inherently unstable double integrator pitch dynamics.

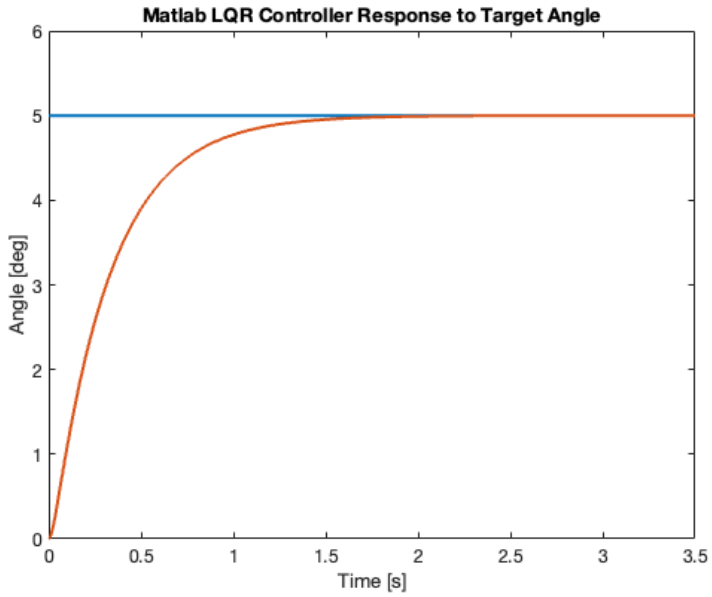


Figure 5.12 – LQR response for pitch control to target angle

To directly compare LQR performance with the PID controller, the LQR response was plotted alongside both the transfer function PID response and the 3DOF simulation, as shown in Figure 5.13. The PID based simulations exhibit a faster rise time and a small overshoot due to their more reactive corrective structure, whereas the LQR trajectory prioritizes smoothness and efficiency, settling without oscillation. Despite these differences, all controllers ultimately track the desired pitch command with similar steady state accuracy

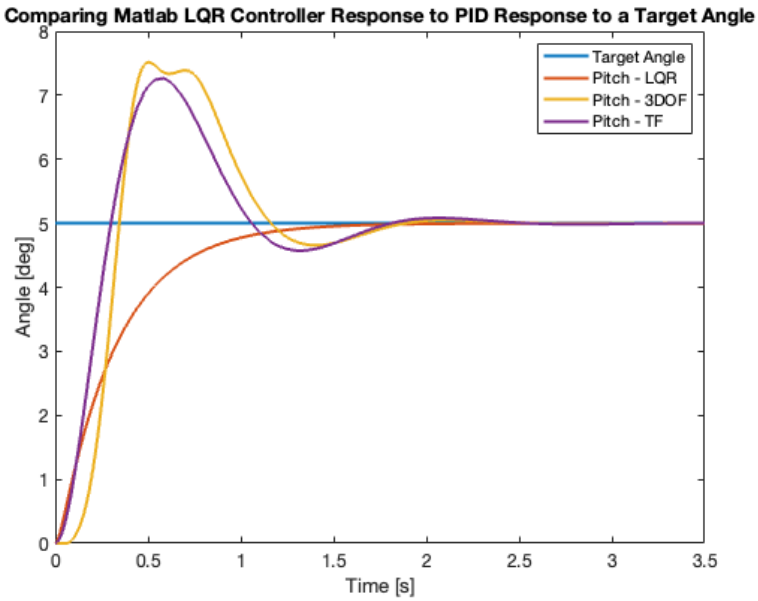


Figure 5.13 – Comparing LQR and PID pitch controller response to target angle

Taken together, these results demonstrate that the LQR formulation provides a robust and effective method for stabilizing the rocket's pitch axis under the reduced order model. Although it produces a slower initial response than PID control, LQR yields a smooth, non oscillatory trajectory with minimal control effort.

5.6 LQG Response

While the LQR controller presented in the previous section provides an optimal state feedback law under the assumption of full state availability, real flight systems must operate in the presence of sensor noise, modeling uncertainty, and imperfect state measurements. To evaluate controller performance under these more realistic conditions, a LQG controller was implemented. The LQG framework augments the LQR controller with a Kalman filter, which estimates the system states from noisy measurements. The full Simulink implementation is shown in Figure 5.14, where the estimator and state feedback components form the closed loop LQG architecture.

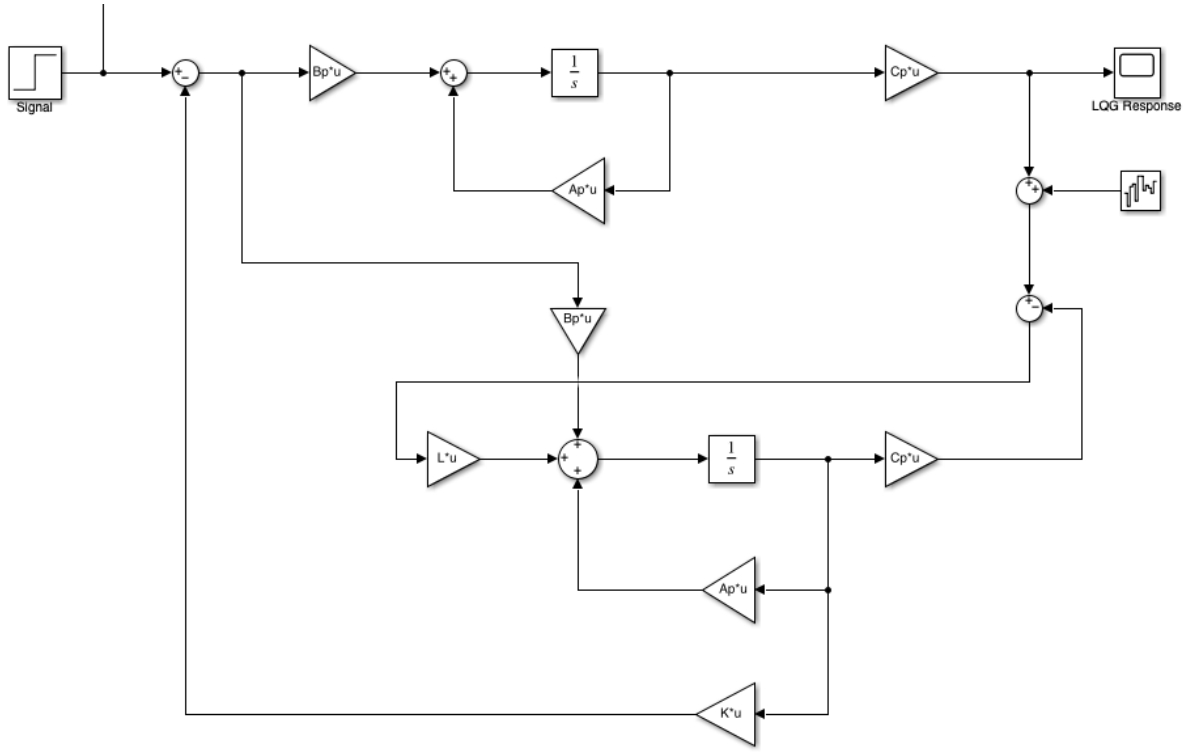


Figure 5.14 – LQG Simulink model for pitch control

The Kalman filter gain was computed using the process noise input matrix together with the process noise covariance Q_{lqg} and measurement noise covariance R_{lqg} shown in Equation 5.8 and 5.9. This determines how strongly the estimator accounts for uncertainty in the system dynamics versus uncertainty in the sensor measurements. The selected values of Q_{lqg} penalize deviations in the internal state, while R_{lqg} reflects the expected level of noise in the pitch angle measurement, modeled in simulation as an additive noise of amplitude 0.5.

$$Q_{lqg} = \begin{bmatrix} 0.0125 & 0 \\ 0 & 0.0250 \end{bmatrix} \tag{5.8}$$

$$R_{lqg} = 1 \tag{5.9}$$

Solving the Kalman filter Riccati equation using these weighting matrices yields the optimal estimator gain L, presented in Equation 5.10, whose entries weigh the pitch and pitch rate measurement residuals to reconstruct an accurate estimate of the system state. When this estimator is combined with the same LQR gain matrix used in the previous section, the resulting LQG controller forms a complete optimal output feedback architecture capable of stabilizing the pitch dynamics in the presence of realistic measurement noise.

$$L = \begin{bmatrix} 0.5733 \\ 0.1581 \end{bmatrix} \tag{5.10}$$

The resulting LQG closed loop response is shown in Figure 5.15, plotted alongside the corresponding LQR response, and both controllers achieve stable convergence to the commanded pitch angle. However, several important distinctions arise between them. The LQR response, computed under ideal and noise free conditions, produces a smooth, monotonic rise toward the target orientation because it has direct access to the full state vector. The LQG response, by contrast, exhibits slightly slower convergence and mild fluctuations throughout the transient period. These small oscillations stem from the Kalman filter's continuous correction process, as noisy measurements introduce adjustments in the estimated state. Despite these disturbances, the LQG trajectory remains well damped, stable, and capable of accurately tracking the desired pitch angle.

This comparison highlights the advantages of incorporating a state estimator when full state feedback is unavailable. The LQG controller stabilizes the thrust vectoring pitch dynamics even under significant measurement noise, providing robustness that pure LQR does not explicitly address. Although its response is slightly more conservative and shows small fluctuations, the LQG controller effectively filters noisy measurements and maintains accurate tracking performance Together with the PID and LQR results, these findings show that estimator based optimal control extends the idealized LQR approach to realistic operating conditions and that the reduced order pitch model can support multiple viable control strategies depending on mission requirements and hardware constraints.

6. Python Embedded Flight Control Architecture

The active stabilization of the rocket is achieved through a real time attitude control system that combines inertial sensing, sensor fusion, and a thrust vectoring PID controller. Raw measurements from the ICM-42605 gyroscope and accelerometer are processed through a quaternion based fusion algorithm to obtain pitch and roll estimates of the rocket. These attitude estimates serve as the feedback input into the PID controller which then computes the gimbal deflections required to vertically orient the rocket throughout its ascent. All flight logic is implemented in Python on the STARLIGHT embedded flight computer which operates the control loop at 50-100 Hz. This ensures the timely actuator response and consistent state estimation for the rocket.

The following subsections describe the Python based simulation environment used for controller design, the Starlight hardware library for sensor data collection, the Fusion attitude estimation library, and the structure of the main flight loop executed during launch.

6.1 Python 3 PID Simulation

Prior to implementing the controller on the embedded hardware, a Python 3 simulation was developed to evaluate the performance of the thrust vectoring PID controller in a simplified rotational model. This standalone script models the rocket as a rigid body with a single rotational degree of freedom and a constant moment of inertia. The Estes F15 thrust curve is represented as a discrete time series and interpolated as a function of time to obtain a continuous thrust profile. The gimballed motor produces a control moment about the rotation axis, modeled as where d is the moment arm between the nozzle and the center of mass, $T(t)$ is the thrust, and $\delta(t)$ is the gimbal angle. Equation (6.1) provides the input torque for the rotational dynamics.

$$\tau(t) = dT(t)sin\big(\delta(t)\big) \tag{6.1}$$

A PID controller is applied to regulate this angle toward an upright reference. The discrete PID law computes a desired gimbal setpoint based on the attitude error, the accumulated integral of the error, and the rate of change of the angle. The controller gains were tuned iteratively within this Python environment to achieve a fast, well damped response to disturbances. The final gain values used in the simulation are summarized in Table 6.1, and the resulting attitude response for an initial 5° perturbation is shown in Figure 6.1.

Table 6.1 – PID gains used in Python 3 attitude control simulation

$\boldsymbol{K_p}$	K_i	K_d
35	0.25	5

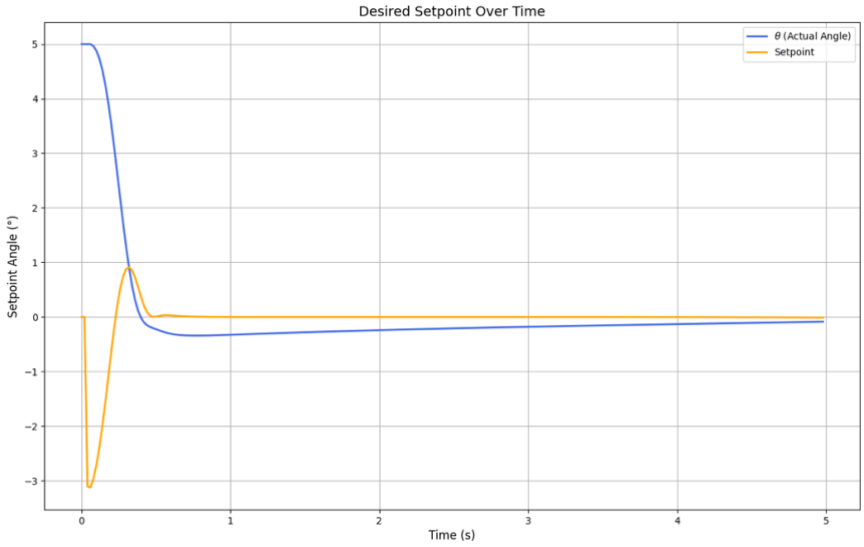


Figure 6.1 – Closed loop pitch attitude tracking under Python 3 PID simulation

The simulator provides a controlled environment to visualize the closed loop response under realistic motor thrust and disturbance conditions. By plotting the actual angle and the commanded setpoint over time, it is possible to tune K_p, K_i, and K_d, and to evaluate the impact of rate and amplitude limits before deploying the controller on the flight computer. This offline design step reduces the risk of unstable or overly aggressive behavior during actual launches.

It is important to note that the gains identified in this Python model differ from those obtained in the MATLAB/Simulink simulation. The Python simulation assumes a single degree of freedom rigid body with constant inertia and simplified torque generation, whereas the MATLAB model incorporates a more detailed plant representation and a different numerical integration and discretization structure. These variations in modeling assumptions, numerical schemes, and actuator representations naturally lead to different optimal gain sets, even though both controllers are designed to achieve similar attitude stabilization objectives.

6.1.1 MATLAB and Python 3 PID Comparison

To evaluate the consistency of control behavior across modeling environments, the Python PID simulation was compared against both the MATLAB transfer function model and the MATLAB 3DOF simulation. As shown in Figure 6.2, all three environments produce a fast rise time followed by a damped convergence toward the commanded orientation, validating the general consistency of the PID structure. The MATLAB transfer function model and the 3DOF model display similar overshoot levels due to their shared continuous time structure and identical gain sets. On the other hand, the Python simulation exhibits a smoother and more damped response. This difference arises from the Python model's simplified dynamics, discrete time integration scheme, and the different gain set used for the Python controller, which were tuned specifically for its numerical behavior and rigid body assumptions. Despite these differences in overshoot and damping, all implementations achieve stable regulation toward their respective setpoints, demonstrating that the PID framework behaves consistently across both simplified and higher fidelity environments.

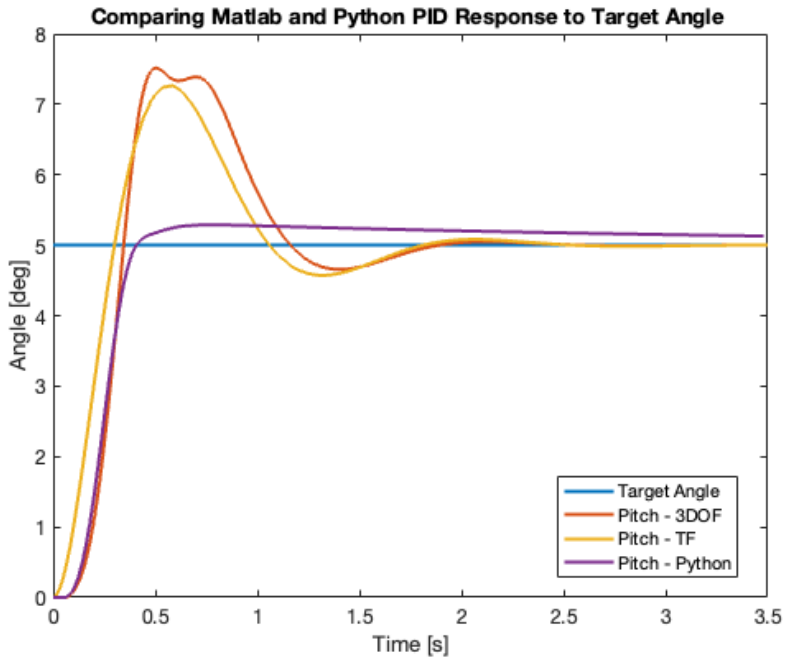


Figure 6.2 – Comparing MATLAB and Python PID responses to pitch for a target angle

6.2 Sensor Data Collection and Starlight Library

On the embedded system, low level access to the inertial and barometric sensors is provided by the custom Starlight library. This library wraps the hardware interfaces for the ICM 42605 IMU and the BMP388 barometric pressure sensor into Python.

The ICM 42605 class configures the gyroscope and accelerometer via I2C register writes. Raw sensor readings are converted using the device resolution factors and stationary bias terms are estimated through dedicated calibration routines. These calibration procedures average multiple samples while the board is held still, producing bias offsets for each axis of both the gyroscope and accelerometer. Subsequent calls to get_accel_and_gyro_data() return bias corrected accelerations and angular rates, which significantly reduces drift in the downstream fusion algorithm.

The BMP388 class performs a similar role for the barometer. It reads factory calibration coefficients from the device, applies the manufacturer's compensation formulas to raw pressure and temperature data, and provides pressure in hPa. Static helper methods convert pressure to altitude and compute rocket altitude relative to a baseline established at startup. This enables real time estimation of vertical motion for logging and parachute deployment logic

6.3 Fusion Library for Pitch and Roll Estimation

Pitch and Roll attitude estimation is performed using the Fusion library, which implements the Madgwick AHRS algorithm in Python. This algorithm fuses accelerometer and gyroscope measurements to produce a quaternion representation of the rocket's orientation, from which pitch

and roll are derived. The fusion algorithm operates in a complementary manner: gyroscope measurements provide high frequency information through integration of angular rates, while accelerometer measurements provide a long term reference to gravity, correcting gyroscope drift.

The Fusion class maintains the current quaternion state and uses a DeltaT helper to compute the elapsed time between updates. At each iteration, update_nomag() is called with the current 3-axis acceleration and angular rate vectors. The method normalizes the acceleration, computes a gradient descent correction term that aligns the estimated gravity direction with the measured one, and integrates the quaternion differential equation using the measured angular rates and the computed correction. The quaternion is then renormalized to mitigate numerical drift.

Pitch and roll are obtained by converting the updated quaternion to Euler angles using standard trigonometric relations. In this implementation, heading (yaw) is not required for thrust vector control, so the magnetometer is omitted and only the accelerometer gyroscope variant of the algorithm is used. The resulting pitch and roll angles form the primary feedback signals for the attitude controllers after additional calibration and filtering.

6.4 Flight Computer Main Loop Breakdown

The main launch script integrates all sensing, estimation, control, and safety functions required for autonomous flight. The execution proceeds through a series of distinct phases that prepare the inertial sensors, initialize the actuators, establish baseline attitude references, and finally conduct the real time stabilization and recovery operations.

6.4.1 IMU Warm Up and Gyroscope Bias Estimation

Upon powering the flight computer, the embedded Python code begins by warming up the ICM-42605 IMU for approximately 20 seconds. This warmup period is necessary because MEMS gyroscopes and accelerometers exhibit temperature dependent drift during their first several seconds of operation. As the sensor package reaches its steady thermal state, the zero rate output of the gyroscope becomes stable, allowing accurate determination of its inherent bias.

Following the warmup interval, a dedicated gyroscope bias calibration procedure is executed. The IMU is held stationary while hundreds of samples are averaged to compute the biases b_{gx}, b_{gy}, and b_{gz}. These values represent the constant offsets in the gyroscope output when the rocket is motionless. Removing these biases is essential because even a small uncompensated offset on the order of $0.1 – 0.5°/s$ will accumulate into the several degrees when integrated over time, making the Fusion attitude estimates unusable.

6.4.2 Parachute and Motor Ignition Pin Initialization

After completing the IMU warmup and gyroscope bias calibration, the flight computer initializes the digital output pins required for motor ignition and parachute deployment. These pins interface directly with the pyrotechnic firing circuits and therefore must be placed in a known safe LOW state prior to arming the rocket. Initializing these pins early in the launch sequence ensures that the ignition and ejection charges cannot be triggered inadvertently during sensor calibration or servo setup. Establishing these pins before entering the main control loop also guarantees that the flight computer has full command authority over all recovery and propulsion events from the moment the countdown begins.

6.4.3 Servo Initialization and Gimbal Configuration

Following the configuration of critical event pins, the software initializes the two thrust vectoring servos that control the pitch and roll axes of the gimbal. Each servo is assigned a dedicated PWM channel operating at 50 Hz, with empirically determined center, minimum, and maximum duty cycles that correspond to the mechanical limits of the gimbal mechanism. Immediately after initialization, both servos are driven to their center values so that the nozzle is aligned with the longitudinal axis of the rocket at launch. This ensures a nominal thrust direction and avoids imparting unintended torques at liftoff. The scale factors used later in the control loop are denoted as deg_per_x and deg_per_y which define the number of servo duty cycle units corresponding to one degree of commanded gimbal motion.

6.4.4 Logging System Setup

To support post flight analysis of the ascent, a comprehensive data logging system is initialized prior to the countdown. The flight computer generates a unique filename based on the mission date and existing log files, then writes a header row defining each recorded variable. Throughout the flight, the logger captures time, pitch, filtered pitch, roll, filtered roll, absolute altitude, rocket relative altitude, pitch and roll PID setpoints, gyroscope measurements (g_x, g_y, g_z), accelerometer measurements (a_x, a_y, a_z), and whether the parachute has been triggered. This data set provides the basis for quantitative assessment of controller performance and anomaly detection.

6.4.5 Static Pitch and Roll Calibration

Although the rocket is mechanically level on the launch rail, small mounting tolerances between the PCB and the airframe introduce fixed offsets into the IMU's interpretation of pitch and roll. To address this, the flight computer performs a static calibration just before launch. Several hundred samples of Fusion estimated pitch and roll are averaged to determine the static offsets. This can be shown in Equation 6.2 and 6.3. These offsets are subtracted from all subsequent attitude estimates to recover the true body frame angles. This process ensures that the control loop regulates the rocket's actual orientation rather than the IMU's mechanically biased reference frame.

$$\theta_{\text{cal}} = \frac{1}{N}\sum_{i=1}^{N}\theta_i \tag{6.2}$$

$$\phi_{\text{cal}} = \frac{1}{N}\sum_{i=1}^{N}\phi_i \tag{6.3}$$

6.4.6 Launch and Timing Structure

The flight computer operates its main control loop for a total of 10 seconds, even though the rocket's motor burns for approximately 3.5 seconds. This extended duration provides sufficient time to capture critical ascent dynamics during both powered flight and the coast phase. Motor ignition is commanded at the first iteration of the control loop, after which the stabilization logic remains active for the full data collection window. This timing structure ensures that all relevant control behavior, thrust vector response, and altitude profiles are captured without interruption.

6.4.7 Real Time Attitude and Pressure Measurements

During each cycle of the main loop, the flight computer first acquires new accelerometer and gyroscope measurements from the ICM-42605. These are passed to the Madgwick Fusion algorithm, which outputs updated pitch and roll estimates after compensating for gyroscope bias and integrating angular rates. In parallel, pressure measurements from the BMP388 are used to compute altitude using the standard barometric formula shown in Equation 6.4. Rocket relative altitude is computed by subtracting the prelaunch baseline pressure.

$$h = 44330\left(1 - \left(\frac{P}{1013.25}\right)^{0.1903}\right) \tag{6.4}$$

6.4.8 Low Pass Filtering of Attitude Estimates

Attitude estimates during ascent are affected by motor vibration, aerodynamic buffeting, and accelerometer noise, each pitch and roll measurement is passed through a first order low pass filter to attenuate high frequency disturbances while preserving the underlying attitude trends. The filtered values θ_f and ϕ_f are stored and reused at each iteration to maintain continuity of the filter state. This provides a balanced compromise between noise suppression and controller responsiveness, ensuring stable yet timely attitude estimates. The full equation of the low pass filter is shown in Equation 6.5.

$$x_f[k] = ax[k] + (1-\alpha)x_f[k-1] \tag{6.5}$$

6.4.9 PID Computation Using Gyro Derivatives and Integral Terms

The flight computer then updates the pitch and roll PID controllers using both the filtered attitude angles and the gyroscope derived angular rates. The gyroscope's g_x rate is used as the derivative term for pitch, while g_z provides the derivative term for roll, consistent with the IMU's axis orientation. The integral terms accumulate the filtered angle error over time, allowing the controller to correct steady state biases. Using these components, the PID laws generate continuous control outputs for both pitch and roll by combining proportional action on the filtered angle, derivative action on the corresponding gyro rate, and integral action on the accumulated error. These outputs represent the desired corrective moments and are subsequently mapped to PWM servo commands to drive the thrust vectoring gimbal.

6.4.10 Mapping PID Outputs to Servo PWM Commands

Because the servos respond to PWM duty cycle values rather than angular commands, the PID outputs are converted into servo positions using linear calibration factors. The mapping for the pitch and roll servos are given by Equation 6.6 and 6.7 respectively. The negative sign in the roll mapping arises from the physical orientation of the roll servo, which moves opposite to the pitch servo. These commands are saturated to remain within calibrated limits to prevent mechanical overtravel. This mapping represents the primary difference from the Python 3 simulation, where the controller directly outputs gimbal angles rather than servo duty cycles.

$$servo_x = center_x + \frac{\theta_f}{deg\ per\ x} \tag{6.6}$$

$$servo_y = center_y + \frac{\phi_f}{deg\ per\ y} \tag{6.7}$$

6.4.11 Logging in Real Time and Periodic Flush

At each iteration of the flight loop, the computer appends the full state vector and control outputs to the onboard log file. To mitigate the risk of data loss from battery disconnects or mechanical impacts during recovery, the system flushes the log to flash memory every fifteen iterations. This periodic flush guarantees that at least 90–95% of the dataset is preserved even in the event of unexpected power interruption.

6.4.12 Parachute Deployment Logic

Recovery logic is embedded directly within the main control loop. The parachute is deployed under either of two conditions. The first is a time based trigger in which the parachute pin is activated after approximately four seconds of flight, corresponding to the 3.5 second motor burn plus a brief coast period. The second is an attitude based emergency trigger in which the parachute is deployed if the magnitude of either filtered pitch or roll exceeds 60°, indicating a potential tumbling event.

These thresholds are monitored each iteration, and when met, the parachute ejection charge is fired by pulsing the deployment pin HIGH.

6.4.13 Post Flight Data Collection

At the end of the 10 second data collection window, the flight computer concludes the control sequence by returning both gimbal servos to their calibrated center positions. This prevents unnecessary mechanical loading on the gimbal assembly during descent and retrieval. As a final safeguard, the parachute ejection pin is activated once more to ensure that the recovery system has fired. The log file is then closed properly, preserving all recorded data, and a completion message is printed to indicate that the flight computer has safely exited the operational state.

7. Launch Results and Discussion

All flight tests were conducted in the Black Rock Desert region of Nevada, a historically significant site known for its flat topology, sparse vegetation, and suitability for experimental aeronautics. Due to the vehicle's limited maximum altitude (<200 ft), the series of flights qualified as casual launches under local fire season and permitting regulations. Environmental conditions were typical of Black Rock: low humidity, relatively stable winds, and unobstructed visibility, making the site well suited for iterative testing of the TVC model rocket. The following sections summarize the results from five flight tests, each building upon the modifications and lessons of its predecessors. Launches 1 and 4 produced no flight data due to logging or initialization failures. However, their qualitative behavior remains relevant to the system's developmental trajectory.

7.1 Launch 1 – October 24, 2025

Launch 1 served as the baseline attempt to verify the functionality of the STARLIGHT flight computer, inertial measurement unit (IMU), and TVC actuation system. During this test, the flight computer did not contain a functioning data flush routine, resulting in a complete absence of recorded telemetry. As a result, all interpretation of the rocket's motion relied entirely on qualitative observation and the available launch video footage.

Despite the lack of sensor data, visual observation indicated that the rocket exhibited immediate and uncontrollable oscillations. The vehicle's attitude appeared to be dominated by sensor noise, overwhelming the controller and preventing effective stabilization. The rocket's responses resembled a system where the noise exceeded the meaningful IMU signal, causing the control loop to actuate randomly. The video of Launch 1 provides a clear illustration of the rapid instability that developed immediately after ignition.



Figure 7.1 – Launch 1 Early oscillation after liftoff

Prior to launch, the rocket remains upright and stable on the lakebed, with no environmental disturbance. As shown in Figure 7.1, the moment the motor ignites the vehicle begins exhibiting rapid lateral oscillations, initiated almost instantaneously after liftoff. These motions grow progressively larger, as the controller commands abrupt, high frequency thrust corrections that do not correspond to actual attitude changes. By the time shown in Figure 7.2, the rocket is visibly tilting off axis, with the thrust vectoring system amplifying the behavior instead of mitigating it.



Figure 7.2 - Launch 1 Mid ascent with growing instability

Ultimately, as captured in Figure 7.3 below, the rocket departs significantly from vertical flight, indicating that the stabilization loop has been overwhelmed by noise dominated IMU data. This sequence visually confirms that the TVC system was unable to maintain control due to unfiltered sensor noise driving the feedback loop.



Figure 7.3 - Launch 1 Loss of vertical alignment

The footage confirms that the rocket never achieves a stable, vertical ascent. Within the first second, the instability becomes unrecoverable, and the vehicle deviates away from the intended trajectory. This test demonstrated that reliable data logging, proper IMU calibration, and low pass filtering are mandatory prerequisites for meaningful closed loop control. Launch 1 established that unfiltered and unvalidated IMU measurements would render the thrust vectoring control system ineffective. The rocket's instability was not a result of aerodynamics, mechanical failure, or environment but rather, it was directly tied to sensor noise overwhelming the controller.

7.2 Launch 2 – November 2, 2025

Launch 2 was conducted to investigate the severe instability observed during Launch 1 by examining raw IMU behavior, control loop execution timing, and the effect of the derivative term within the PID controller. No low pass filtering was implemented due to loop rate limitations, and a critical configuration error was later identified. The PID derivative term incorrectly used the g_y axis instead of g_z, introducing erroneous rotational information into the controller and significantly degrading stability. This launch provided the first complete dataset combining video evidence and sensor telemetry, making it possible to correlate visual behavior with onboard measurements.



Figure 7.4 - Launch 2 ignition and upright liftoff

Figure 7.4 shows the rocket at the moment of ignition. The vehicle lifts off cleanly, maintaining an upright posture with no apparent aerodynamic disturbances. Shortly after liftoff, as shown in Figure 7.5, the rocket begins to exhibit the first signs of lateral skew. This off axis motion corresponds to the controller applying noisy and derivative generated torques. Due to this the deviation increases rapidly.



Figure 7.5 - Launch 2 initial lateral skew

In Figure 7.6, the rocket departs more aggressively from vertical as the PID controller commands thrust vector deflections based on flawed gyroscope data. By the time captured in Figure 7.7, the rocket has entered a clearly observable spiraling motion, rotating uncontrollably as the control system becomes fully saturated. This visual sequence confirms that the instability originated from incorrect derivative inputs and unfiltered IMU noise rather than from environmental disturbances or propulsion anomalies.



Figure 7.6 - Launch 2 translational instability leads to spiraling



Figure 7.7 - Launch 2 rocket spiraling before hitting the ground

The IMU and barometer data recorded during Launch 2 provide quantitative confirmation of the instability observed in the video frames. Figure 7.8 presents the measured pitch and pitch setpoint. The pitch angle exhibits large excursions exceeding ±60° throughout flight. The significant

divergence between the measured and commanded pitch indicates that the controller was unable to compensate for noise amplified disturbances, consistent with the corrupted derivative term.

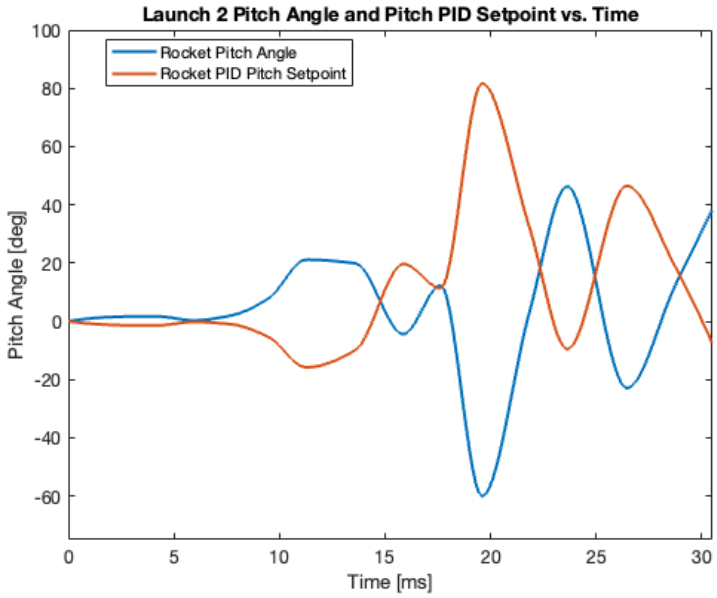


Figure 7.8 – Launch 2 pitch angle and pitch PID setpoint vs time

Roll behavior, shown in Figure 7.9, demonstrates even more severe instability. Modified roll diverges sharply, reaching values near –230°, indicating runaway rotational growth. The roll setpoint oscillates aggressively in response to corrupted gyroscope data, confirming derivative induced controller saturation.

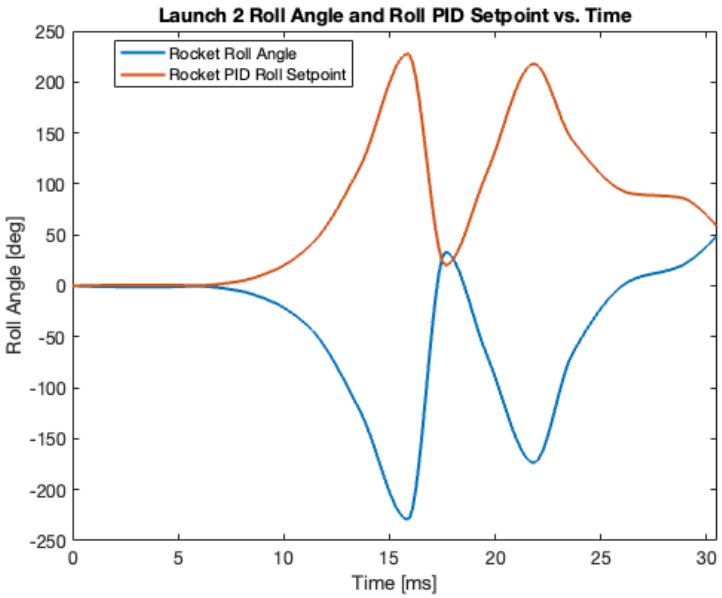


Figure 7.9 – Launch 2 roll angle and roll PID setpoint vs time

The gyroscope data in Figure 7.10 show rapid drift and high magnitude spikes, particularly on the g_x and g_y axes, where values approach –200 deg/s. This behavior matches expectations for a derivative term based on the wrong rotational axis and explains the uncontrolled torque commands observed in flight.

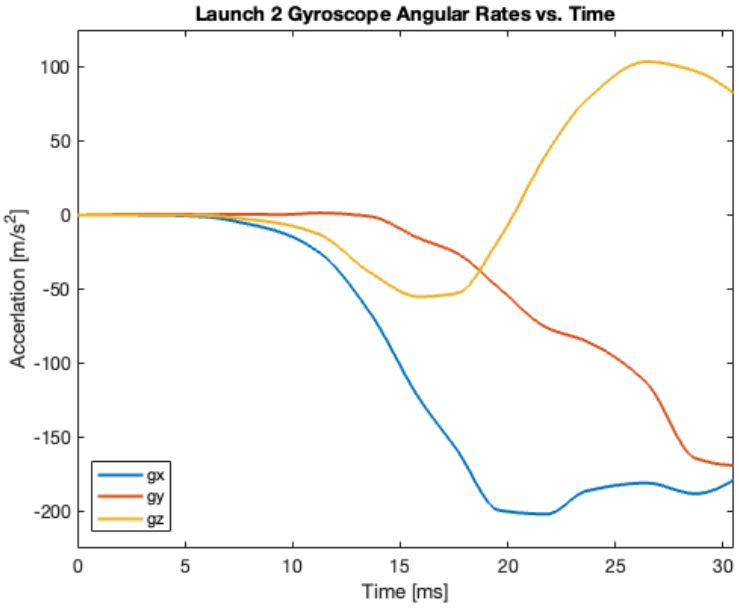


Figure 7.10 – Launch 2 gyroscope angular rates vs time

As shown in Figure 7.11, the a_y accelerometer channel exhibits peaks exceeding 3g, while a_x and a_z remain near zero. This asymmetry reflects unbalanced loads and high frequency vibration coupling into the IMU due to unstable control activity.

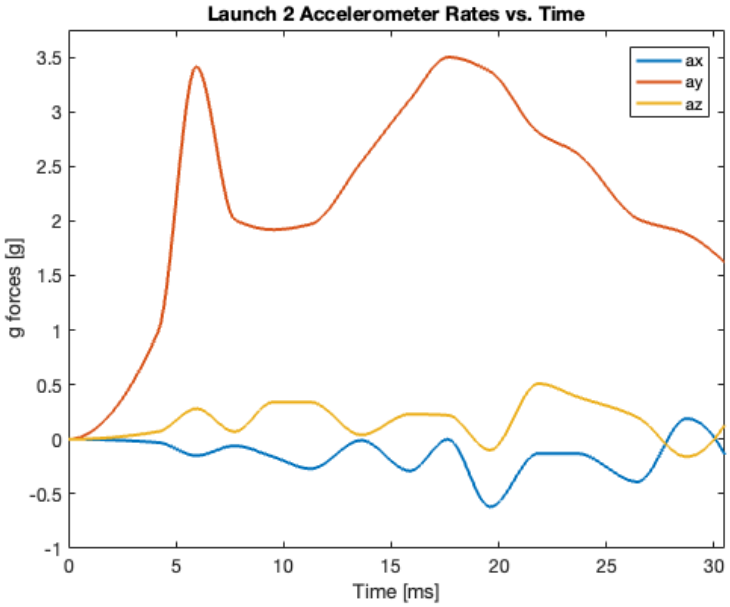


Figure 7.11 – Launch 2 acceleration rates vs time

The altitude trace in Figure 7.12 varies between approximately 1090–1108 m, reflecting brief vertical perturbations rather than sustained upward flight. The sharp peak near 15 ms corresponds to a transient acceleration pulse, after which the altitude quickly returns toward baseline. This behavior indicates that the rocket did not achieve meaningful ascent during Launch 2

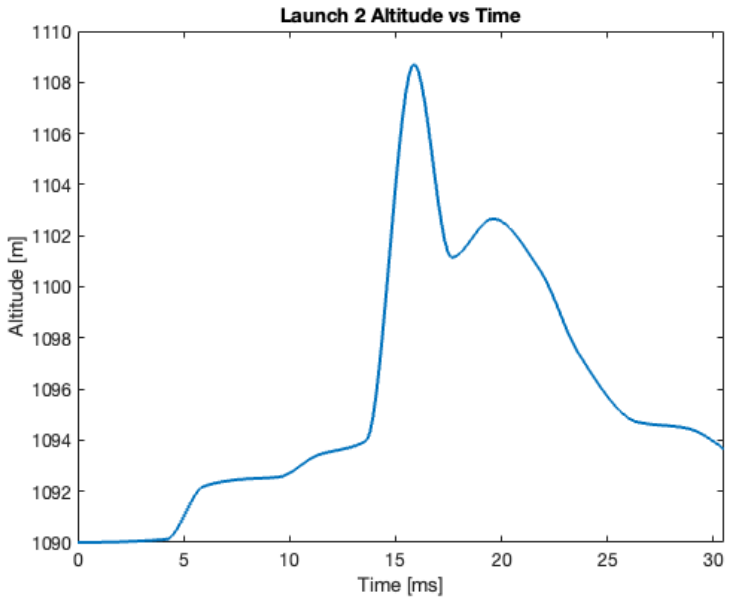


Figure 7.12 – Launch 2 altitude vs time

Launch 2 provided critical insight into the root causes of the instability observed during the initial test flight. The combined analysis of flight video and onboard telemetry demonstrated that the vehicle's loss of control originated from two primary deficiencies within the attitude control system: severe IMU noise and a misconfigured derivative term within the PID controller. The absence of low pass filtering allowed high frequency disturbances, structural vibrations, and sensor quantization noise to propagate unchecked into the control loop. This noise corrupted both the measured pitch and roll signals, producing large deviations in the modified attitude estimates and rendering the controller incapable of tracking its commanded setpoints. As a result, the control authority was effectively saturated during the earliest moments of ascent.

The PID derivative term furthered this behavior. Due to a configuration error, the derivative computation utilized the gy axis rather than the correct gz axis for yaw related rotational feedback. This axis mismatch caused the controller to interpret rotation about the wrong body axis, introducing destabilizing torques rather than corrective ones. The gyroscope data confirmed this error, showing rapid drift and excursions along the wrong axes. These derivative signals drove aggressive, noise dominated adjustments, which were clearly observable in both the telemetry and the video based still frames.

The accelerometer and altitude data reinforced these findings. Large lateral acceleration spikes indicated significant coupling of unfiltered vibration into the IMU. Instead, the vehicle exhibited rapid thrashing and rotational divergence consistent with complete loss of attitude control. The video frames captured this progression, transitioning from a nominal liftoff to significant off axis deviation and ultimately to fully developed spiraling motion.

Collectively, these results established that Launch 2 instability was not the result of environmental disturbances, propulsion irregularities, or structural anomalies. Rather, the instability stemmed from deficiencies in sensor processing and control loop configuration. The findings underscored the necessity of implementing appropriate IMU filtering, validating gyroscope axis mappings, and ensuring correct derivative computation before attempting further closed loop thrust vector control flights. These lessons directly informed the modifications applied in subsequent launches.

7.3 Launch 3 – November 9, 2025

Launch 3 was executed under a rapid turnaround schedule following the instability observed in Launch 2. By the time of this test, only limited modifications had been implemented, consisting primarily of minor software filtering and refinement of the derivative computation in an effort to reduce high frequency noise entering the PID controller. Although the fusion algorithm was initially believed to be functioning correctly, post flight analysis revealed that the IMU fusion system still contained a Y–Z axis swap identical to the configuration error present in Launch 2. As a result, the purpose of Launch 3 shifted in retrospect: while originally intended to validate early filtering improvements, it ultimately diagnosed the axis mapping error.

The Launch 3 video provides valuable insight into the system's behavior prior to discovering the persistent axis error. In Figure 7.13, the rocket begins ignition in a stable, upright posture with no immediate sign of the violent off axis thrusting observed in Launch 2. This stability was initially attributed to the minor filtering improvements.



Figure 7.13 - Launch 3 initial ignition

By Figure 7.14, the rocket begins exhibiting slight lateral motion. Unlike Launch 2, the motion remains relatively bounded. With the axis error still present, this behavior reflects the partial benefit of added filtering causing reduced noise correction but not eliminating it entirely.



Figure 7.14 - Launch 3 bounded lateral motion

In Figure 7.15, the rocket maintains a comparatively stable posture without the early rotational divergence seen in earlier launches. Although the underlying fusion axis mapping was incorrect, the reduced noise levels masked the full destabilizing effect that appeared in Launch 2. This partial improvement initially suggested that the fusion algorithm was operating correctly; however, later data analysis refuted this interpretation. These frames demonstrate that Launch 3 exhibited noticeably improved early stability compared to Launch 2. However, because the underlying axis error remained, this stability was misleading and attributable primarily to reduced sensor noise rather than correct fusion outputs.



Figure 7.15 – Launch 3 loss of vertical alignment

The pitch dynamics for Launch 3 are shown in Figure 7.16, which plots the measured pitch angle and the commanded pitch setpoint. The data reveal that pitch remained tightly bound within approximately ±5° throughout the duration of the test. This represents a substantial improvement compared to Launch 2, where pitch values diverged beyond ±60°, indicating complete loss of attitude control.

Although the fusion axis error remained uncorrected during this test, the minor filtering implemented prior to Launch 3 significantly reduced the high frequency noise that previously destabilized the controller. As a result, the pitch channel no longer exhibited the abrupt spikes or large excursions that characterized earlier launches. The relatively smooth pitch setpoint further confirms that the controller was not overwhelmed by corrupted derivative feedback. However, because the fusion axis swap was still present, the pitch channel's apparent stability must be interpreted cautiously; the observed bounds reflect reduced noise, not fully accurate pitch estimation.

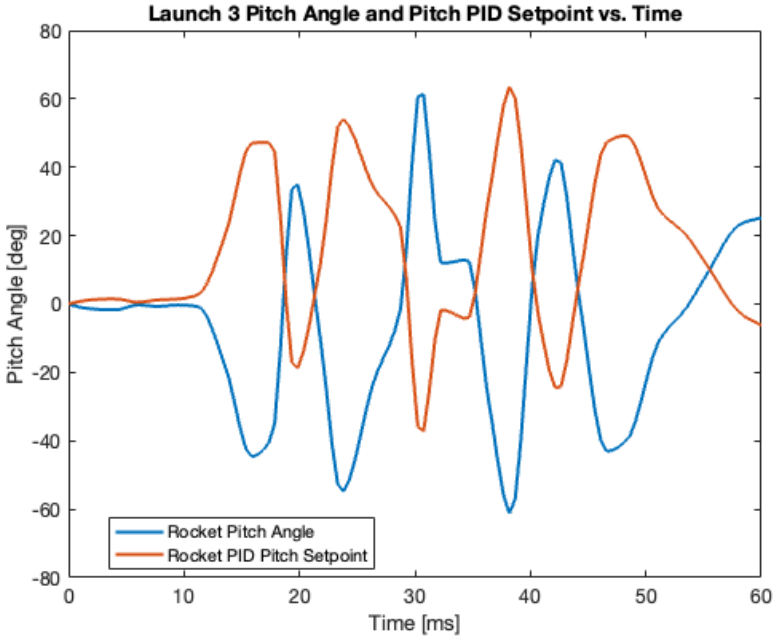


Figure 7.16 – Launch 3 pitch angle and pitch PID setpoints vs time

Roll behavior during Launch 3 is depicted in Figure 7.17, which shows the measured roll, modified roll estimate, and roll command. Compared to the extreme ±200° roll divergence seen in Launch 2, Launch 3 exhibited a far more controlled response. The roll signal remains more oscillatory than pitch. The remaining roll noise likely stems from local IMU vibration, unmodeled structural resonances, and the persistent Y–Z fusion axis mapping error, which introduced cross coupling between pitch and roll. Even with these deficiencies, the filtering applied prior to Launch 3 successfully damped the sharp high frequency disturbances that previously caused runaway roll divergence. This behavior demonstrates partial but not complete improvement in roll axis stability.

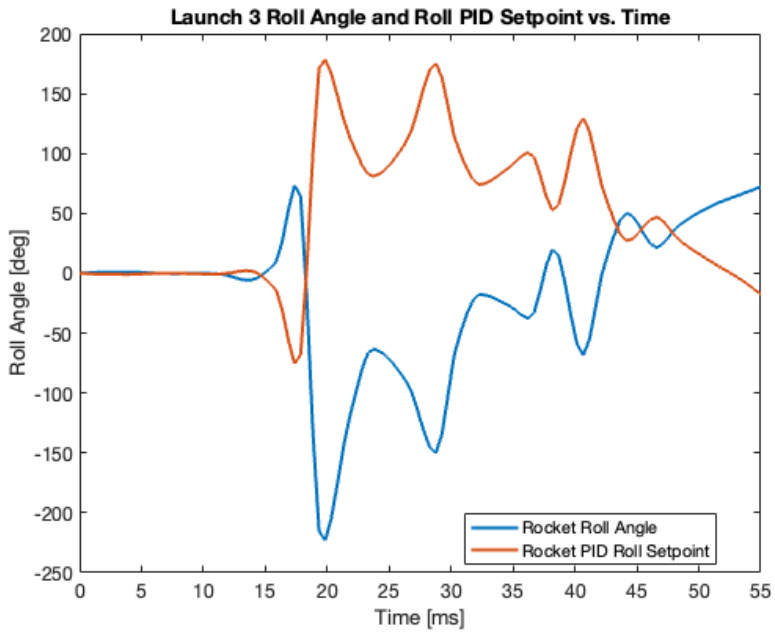


Figure 7.17 – Launch 3 roll angle and roll PID setpoint vs time

The gyroscope data shown in Figure 7.18 provide important diagnostic insight into the underlying estimation issues during Launch 3. All three axes remain relatively well bounded, with g_x and g_z staying close to zero throughout the test. The g_y channel, however, exhibits a subtle but persistent drift that became a crucial indicator of the still unresolved Y–Z axis swap in the fusion algorithm. Because the minor filtering applied prior to Launch 3 reduced the high frequency noise that previously masked these trends, the drift in gy became more observable and ultimately pointed to the incorrect axis mapping.

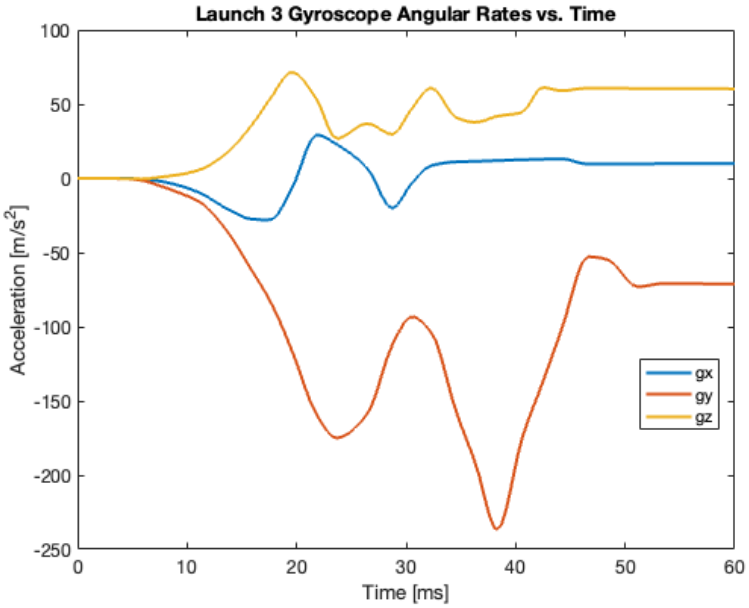


Figure 7.18 – Launch 3 gyroscope angular rates vs time

As shown in Figure 7.19, the accelerometer signals exhibit a notable improvement in stability compared to earlier launches. All three axes remain smooth and low in amplitude, with no sharp spikes, abrupt discontinuities, or vibration induced disturbances. This behavior reflects the effectiveness of the minor filtering that was introduced prior to Launch 3, which successfully reduced the noise that previously destabilized the controller. Although the underlying fusion axis error persisted, the cleaner accelerometer data prevented large noise driven control inputs and supported the bounded pitch and roll responses observed during the test. Overall, the accelerometer behavior indicates that sensor level improvements significantly reduced the likelihood of noise induced instability.

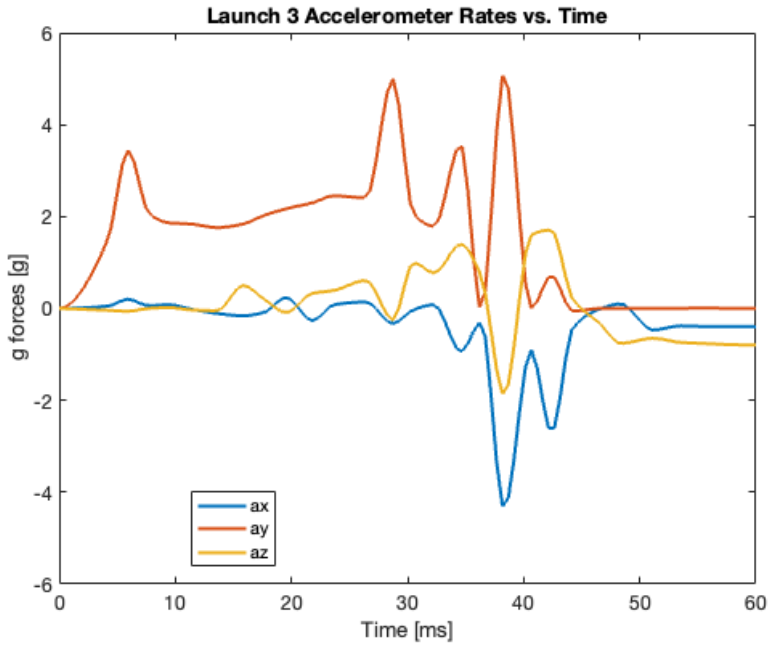


Figure 7.19 – Launch 3 acceleration rates vs time

The altitude profile for Launch 3, shown in Figure 7.20. At approximately 4.17 seconds, the rocket reaches an altitude of 1042.53 m, climbing to 1049.17 m by 5.94 seconds, a net increase of roughly 6.6 meters. This initial upward motion is followed by consecutive spirals where it descends downwards and then corrects its behavior before stalling out and hitting the ground. These rises and drops in altitude correspond directly to the instability captured in the pitch, roll, and gyroscope data. The oscillatory control loop behavior caused the rocket to alternately lift and settle instead of executing a smooth ascent. The altitude trace therefore provides clear evidence that Launch 3 not only achieved liftoff but also exhibited the same loop driven vertical modulation seen in its lateral dynamics, reinforcing that the remaining fusion axis mapping error continued to disrupt stable, controlled flight.

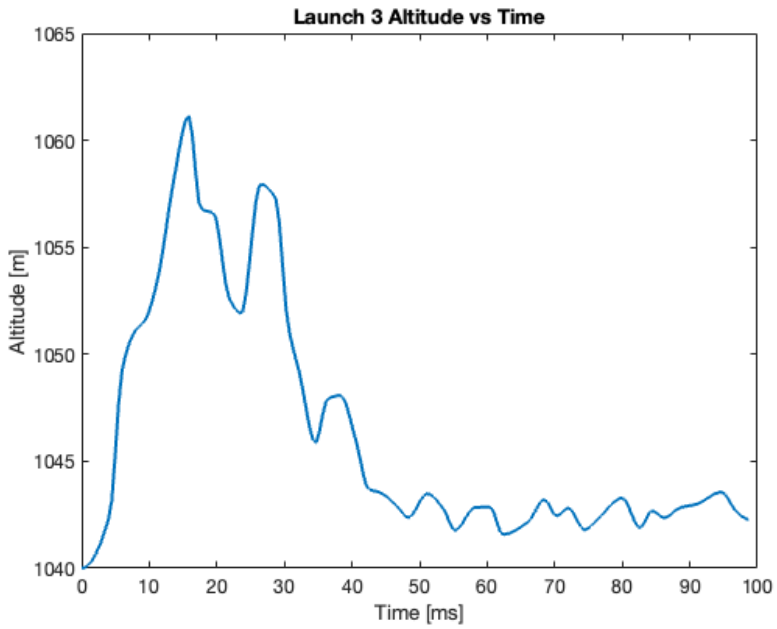


Figure 7.20 – Launch 3 altitude vs time

Launch 3 demonstrated meaningful but incomplete improvements in system behavior relative to the preceding attempts. Minor filtering and code refinements noticeably reduced the high frequency noise that had overwhelmed the controller during Launch 2, allowing pitch and roll responses to remain significantly more controlled. However, although the rocket appeared more stable both visually and through the recorded telemetry, the underlying Y–Z fusion axis mapping error had not yet been corrected during this test. As a result, the attitude estimates used by the controller remained misaligned with the physical IMU axes, and the perceived stability reflected reduced noise rather than accurate state estimation.

Analysis of the gyroscope data revealed a characteristic drift on the g_y channel that did not correspond to the rocket's actual motion, an inconsistency that ultimately indicated the persistent axis mapping fault. Accelerometer measurements further supported this conclusion, showing smoother, noise reduced behavior without the large vibration induced spikes observed in earlier launches. Together, these measurements confirmed that filtering mitigated sensor noise but did not resolve the deeper estimation issue.

Altitude data from the early portion of the flight further reinforce this interpretation. The rocket clearly lifted off, rising approximately 6.6 meters in the first two seconds before entering a sequence of vertical oscillations. These rises and drops correspond directly to the unstable feedback patterns evident in the pitch, roll, and gyroscope channel. This demonstrates that the same control loop oscillations driving lateral deviations were also the cause of the rocket's vertical acceleration and deceleration.

A notable distinction of Launch 3 is that the flight cycle completed fully, allowing the flight computer to record and flush the entire dataset before touching down. In contrast, launch 2 ended with the flight computer being disconnected upon impact, resulting in only the minimal emergency flush values being saved. The complete dataset from Launch 3 therefore provided substantially

more telemetry, enabling clearer identification of the fusion axis mapping error and a more comprehensive analysis of the controller's behavior.

Overall, launch 3 showed that noise reduction alone was insufficient for stable thrust vector controlled flight. While filtering contributed to smoother short term behavior, the persistent fusion axis error remained the dominant source of instability. The more extensive dataset obtained during Launch 3, combined with IMU and altitude evidence, made it clear that correcting the fusion algorithm's axis alignment was essential before stable, sustained ascent could be achieved. This realization directly shaped the corrective actions implemented prior to Launch 4.

7.4 Launch 4 – November 22, 2025

Launch 4 represented the most problematic test in the early flight sequence. Prior to ignition, the STARLIGHT flight computer powered on normally, and all indications suggested that the flight program loaded correctly. However, upon impact, the board suffered a critical hardware fault; the BOOTSEL interface became nonfunctional, preventing any post flight data extraction. As a result, no telemetry, IMU data, or onboard logs were recoverable, despite the system operating at liftoff. This distinguishes Launch 4 from Launch 2, where the flight computer disconnected midflight and only emergency flush values were saved. In Launch 4, the data loss occurred after the flight when the device was physically damaged, not due to initialization failure.

The progression of Launch 4's instability is clearly illustrated by the four extracted frames, which capture the vehicle's motion at key moments during the ascent. In Figure 7.21, the rocket has just begun its upward acceleration following ignition. Thrust appears to develop symmetrically, and the rocket remains relatively upright, consistent with the earliest stage of ascent where aerodynamic forces and attitude disturbances are still small.



Figure 7.21 - Launch 4 ignition

The first signs of instability emerge in Figure 7.22. Here, the rocket departs noticeably from vertical, beginning to yaw and roll as it climbs. The lateral motion is slight but distinctly visible, signifying either a delayed or ineffective control response. This early divergence is characteristic of a system flying without stabilizing corrective input behavior consistent with prior uncontrolled launches.



Figure 7.22 - Launch 4 early divergence

By Figure 7.23, the rocket's attitude has deteriorated substantially. The vehicle exhibits both rotational acceleration and lateral drift, with the axis of rotation becoming clearly offset from the thrust vector. The oscillation amplitude has increased, and the rocket appears to be entering a positive feedback condition where each deviation induces a larger subsequent correction or aerodynamic imbalance. Without telemetry, it is impossible to determine the exact control loop contribution to this motion, but the dynamics match the response of a rocket receiving no stabilizing feedback.



Figure 7.23 - Launch 4 oscillation and spiraling

Finally, in Figure 7.24 this shows the result of the flight. The rocket displays pronounced angular displacement and significant translation away from its ascent trajectory. Together, these images provide a clear qualitative record of Launch 4 trajectory. The sequence captures a transition from an initially upright ascent to rapid divergence and full rotational instability.



Figure 7.24 - Launch 4 crash landing and smoke trail spirals

With no onboard data available, the video record became the primary diagnostic tool for understanding the behavior of Launch 4, and the progressive instability visible in the extracted frames closely resembled the uncontrolled flight dynamics observed in Launch 1. However, the complete loss of telemetry following impact highlights a deeper issue. Without reliable data recovery pathways and hardware survivability, diagnosing thrust vector control performance becomes nearly impossible. In this launch, the inability to extract post flight data was as detrimental to analysis as the inflight instability itself.

7.5 Launch 5 – November 22, 2025

Launch 5 was flown after the failure of Launch 4 and represented the next major attempt at restoring stable thrust vector controlled flight. For this test, a backup flight computer was used. In contrast to earlier flights where the control loop frequency was effectively limited through loop count gating (i.e., skipping control updates to avoid noisy commands), Launch 5 relied on a software low pass filter in the attitude estimation chain. This configuration allowed the controller to run continuously at its full rate, with the low pass filter reducing high frequency noise but, by design, permitting more of the low frequency noise and real dynamics to be passed downstream into the control loop. The goal of this approach was to preserve responsiveness while still preventing the worst sensor noise from destabilizing the system.

The qualitative behavior of Launch 5 is best understood through the extracted video frames, which capture the rocket at four key moments during the ascent. These images illustrate a substantially more stable trajectory than earlier launches and provide visual confirmation of active thrust vector control throughout the early flight. In Figure 7.25, the rocket has cleared the pad cleanly. The thrust column is symmetric, and the vehicle shows none of the immediate oscillatory divergence seen in Launches 1 and 4. This image confirms that the backup flight computer initialized correctly at ignition and that the TVC servos were operational at lift off.



Figure 7.25 - Launch 5 ignition and upright liftoff

In Figure 7.26, the rocket continues upward with only slight angular deviations. These small corrections demonstrate that the control loop is engaged and maintaining stability, despite the increased noise passed down through the low pass filtered backup system.



Figure 7.26 - Launch 5 mid ascent attitude correction

In Figure 7.27, the rocket is captured during a stable portion of the ascent. The attitude offset is minor and well bounded, demonstrating that the controller is actively compensating for disturbances. This frame highlights the improved stability afforded by the backup flight computer and low pass filtered control inputs, even though more low frequency noise is present compared to previous loop suppressed configurations. Figure 7.28 captures the rocket at a critical transition point in Launch 5. Although the ascent up to this moment remained largely stable this frame reveals a noticeable increase in attitude deviation. The rocket appears to be tumbling and rotating away from its controlled ascent axis. This frame therefore represents the onset of uncontrolled motion, signaling the end of effective powered ascent control and the transition into ballistic and tumbling flight.



Figure 7.27 - Launch 5 stable mid ascent attitude



Figure 7.28 - Launch 5 tumbling near apogee

The pitch response during Launch 5 demonstrates a substantially more stable behavior than earlier flights. As shown in Figure 7.29 the raw pitch angle oscillates within a bounded range of approximately –40° to +50°, indicating that the rocket experienced noticeable but manageable attitude motion. The filtered pitch signal, produced through the low pass filter implemented in the backup flight computer, remains significantly smoother and confined to roughly ±30°, demonstrating that the filter effectively attenuated high frequency noise while preserving meaningful attitude trends. Importantly, the pitch setpoint stays near zero for most of the ascent, reflecting that the controller did not need to issue large corrective commands. Pitch deviations, while present, were not large enough to push the system into saturation. This response contrasts sharply with Launch 2, where incorrect derivative signals and unfiltered noise drove pitch angles into large and uncontrolled excursions.

Overall, Figure 7.29 shows that pitch remained consistently bounded throughout the powered ascent, and the controller never lost authority in this channel. Occasional increases in pitch error and corrective commands correspond to natural variations in the aerodynamic environment or slight gimbal induced disturbances. This behavior reflects a well balanced control loop where low pass filtering provided adequate noise suppression without compromising responsiveness.

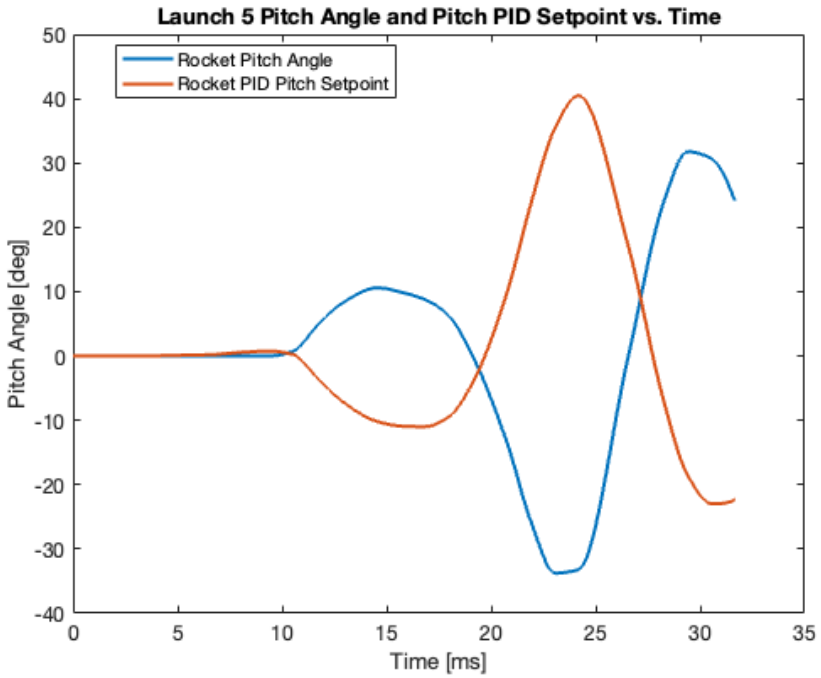


Figure 7.29 – Launch 5 pitch angel and pitch PID setpoint vs time

Roll behavior was more challenging than pitch, but still reflects significant improvement compared to earlier launches. As seen in Figure 7.30 the filtered roll signal captures this trend while reducing high frequency jitter, revealing a persistent lean that the controller continuously worked to counteract. The roll setpoint oscillates frequently and reaches values above +50°, showing that the controller was actively issuing strong corrective commands to manage this bias. Despite these continuous demands on the controller, roll deviations remain bounded and never exhibit the runaway divergence seen in Launch 2.

The stability of roll in Figure 7.30 demonstrates that although the rocket faced a persistent rotational disturbance. The filtered roll signal provides a cleaner trajectory for the control loop, while the increased low frequency noise allowed by the low pass filter contributes to the broader variations visible in the data. Nevertheless, the overall behavior underscores that the backup system's filtering strategy enabled effective stabilization, even under a unideal roll environment.

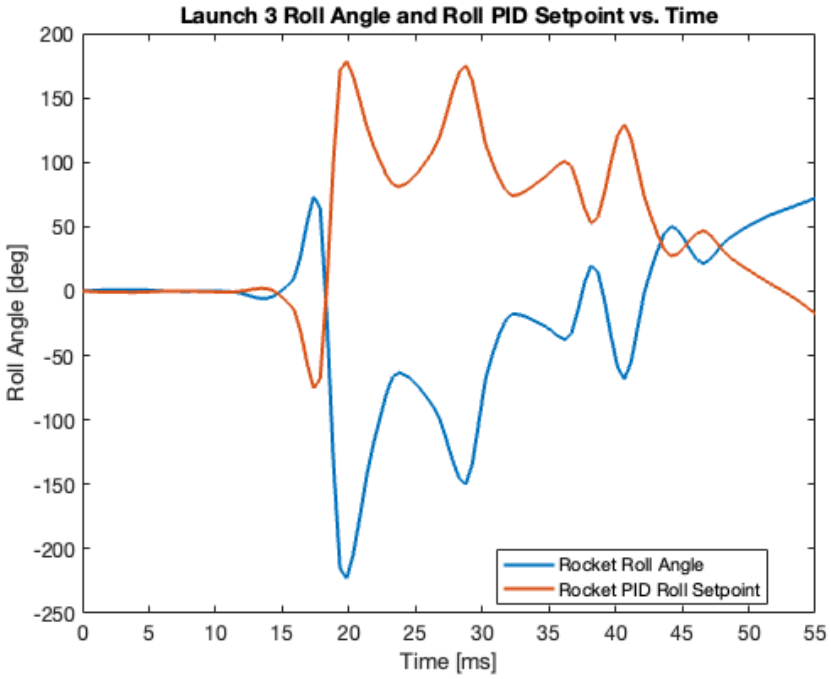


Figure 7.30 – Launch 5 roll angle and roll PID setpoint vs time

The gyroscope data provide insight into both the vehicle's actual rotational motion and the characteristics of the noise propagated through the low pass filter. In Figure 7.31, the gyroscope channels show distinct patterns: g_x ranges from approximately –62 to 0 deg/s, g_y displays consistently large negative values reaching nearly –213 deg/s, and g_z exhibits positive spikes up to 130 deg/s. These signatures indicate that the rocket experienced meaningful rotational loads, particularly around the axis measured by g_y, which aligns with the persistent roll bias observed in the filtered roll data. The moderate drift in both g_x and g_z is consistent with natural disturbances during ascent and the increased low frequency information allowed through the filter. Thus, the gyro plot supports the interpretation that Launch 5 maintained stable, controlled flight until aerodynamic stabilization became insufficient near the top of the trajectory.

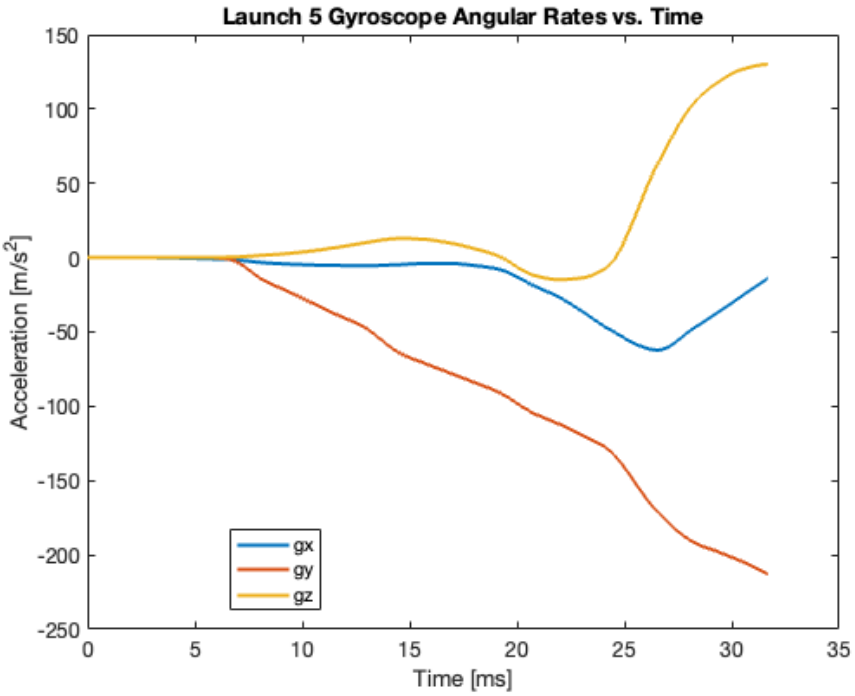


Figure 7.31 – Launch 5 gyroscope angular rates vs time

Accelerometer signals provide a complementary view of the dynamics and noise environment. As shown in Figure 7.32, the ay axis, which is aligned with the primary thrust direction, ranges from about 1.0 g to 3.4 g during powered ascent, reflecting thrust loading and the influence of gravity. Meanwhile, the lateral axes (a_x and a_z) remain close to zero, with only small excursions throughout the flight. This indicates that cross axis vibration and structural coupling into the sensor were limited, and that the low pass filter preserved meaningful acceleration information while suppressing noise.

The clean accelerometer profile in Figure 7.32 confirms that Launch 5 did not suffer from the severe vibration induced noise seen in earlier tests, especially Launch 2. The absence of extreme lateral spikes suggests that the rocket experienced relatively smooth thrust application and that the TVC system did not introduce large oscillations. The stability of the accelerometer signals also supports the pitch and roll findings, reinforcing the conclusion that the vehicle remained well controlled during the primary ascent.

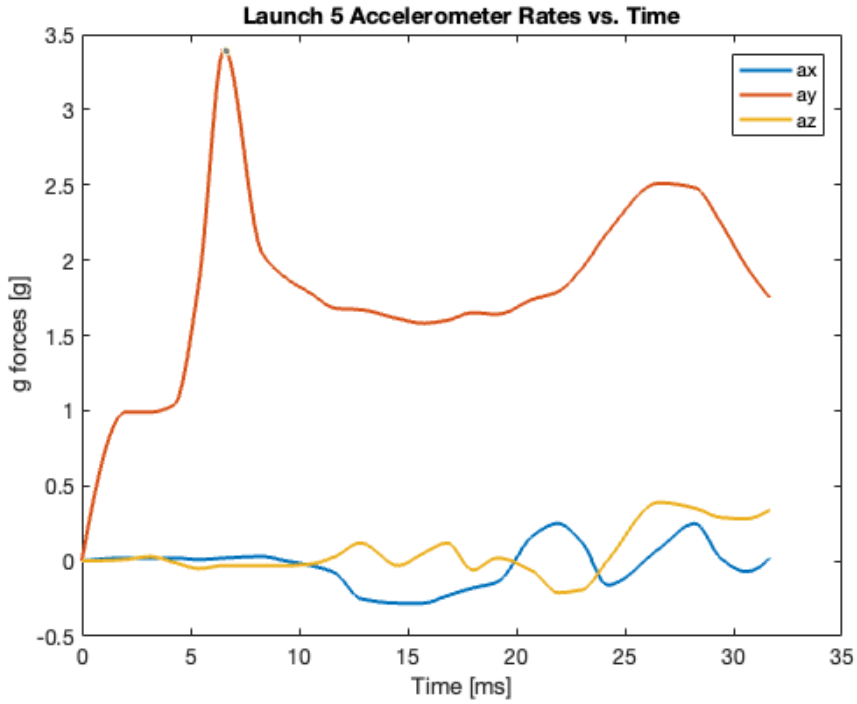


Figure 7.32 – Launch 5 acceleration rates vs time

The altitude data in Figure 7.33 show a smooth and monotonic climb from roughly 1094.6 m to 1117.7 m over the recorded interval, corresponding to a net gain of about 23 meters. This steady ascent indicates that the thrust vector was effectively aligned with the vertical direction for the majority of powered flight. The absence of altitude reversals, dips, or oscillations distinguishes Launch 5 from earlier unstable attempts particularly Launch 3, where repeated small altitude cycles corresponded to control loop over corrections.

As Figure 7.33 illustrates, the vehicle continued climbing until it approached apogee, at which point the vertical acceleration began to diminish. This behavior directly supports the visual observation of attitude drift near apogee and aligns with the expected reduction in aerodynamic damping at higher altitude. The smooth altitude profile is one of the clearest indicators that Launch 5 achieved a controlled and meaningful ascent, validating the backup flight computer configuration and continuous low pass filter approach.

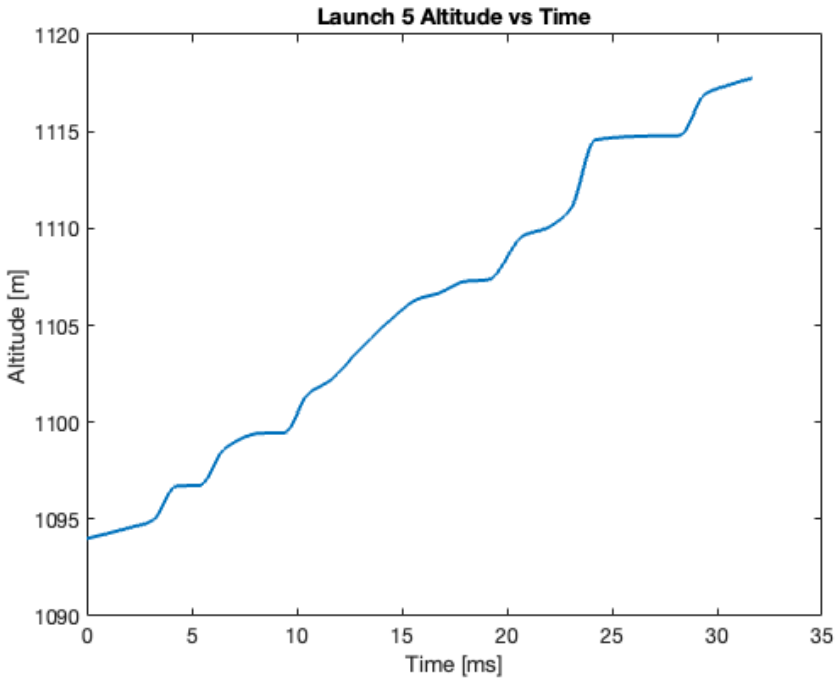


Figure 7.33 – Launch 5 altitude vs time

Launch 5 demonstrated the strongest and most coherent TVC ascent of the entire test series. The integration of the backup STARLIGHT flight computer, combined with a low pass filter based fusion and control strategy, proved to be a meaningful and effective improvement over previous configurations. Unlike earlier flights, particularly launches 1, 2, and 4, this system delivered a sustained period of stable operation where the controller consistently retained authority and prevented early divergence in pitch or roll. The flight video and telemetry together provide clear evidence that the rocket maintained a generally upright trajectory throughout the powered segment, with bounded attitude excursions that did not grow uncontrollably.

The results confirm that the continuous updated control loop allowed the thrust vectoring servos to respond more rapidly to real time disturbances. Although this approach introduced more low frequency noise into the system, the low pass filter ensured that high frequency sensor jitter was effectively suppressed. The quantitative telemetry data reinforce this conclusion. The pitch and roll both remained within operationally manageable bounds, angular rates did not exhibit runaway growth, and accelerometer readings showed a clean thrust profile with minimal lateral vibration. The altitude trace provided unambiguous confirmation of a steadily increasing climb, validating that thrust was consistently and effectively directed upward throughout the burn.

Importantly, the loss of stability observed near apogee occurred only once aerodynamic damping naturally diminished, and not because of controller instability. This marks a meaningful milestone, demonstrating that the TVC system could maintain control through the entire powered ascent phase. In contrast to earlier flights where instability originated immediately or within the first few tenths of a second, launch 5 showed that the revised architecture was robust enough to regulate

disturbances even under imperfect sensor alignment, asymmetric roll tendencies, and the noisier signal environment inherent in the backup configuration.

Taken together, launch 5 provides the clearest demonstration that the TVC system, when paired with appropriate filtering and continuous control updates, can achieve the fundamental objective of stable powered flight. This launch establishes a strong baseline for future refinement, and the insights gained regarding roll bias, gyro behavior, and filter tuning offer clear guidance for subsequent iterations of both hardware and software.

7.6 Summary of Launch Results

Across the five launches conducted in this test series the TVC system, inertial sensing pipeline, and flight software underwent a clear progression from initial instability and instrument failures toward sustained controlled ascent. Each launch contributed key insights into the performance limitations of the STARLIGHT flight computer and the sensitivity of the controller to IMU configuration, filtering, and hardware reliability. Together, these tests outline the path by which the system evolved from noise dominated instability to the first successful demonstration of vertically stabilized flight.

Launch 1 served as the baseline system test but produced no telemetry due to the absence of a functioning data flush routine. The rocket exhibited immediate, uncontrolled oscillations driven by unfiltered sensor noise, highlighting early the necessity of both robust logging procedures and meaningful filtering of inertial measurements.

Launch 2 provided the first complete dataset and revealed extreme instability. A misconfigured derivative term using the g_y axis instead of g_z as well as the absence of low pass filtering resulted in violent oscillations and rapid controller saturation. This launch confirmed that the control architecture was highly sensitive to axis verification and noise management, and that even subtle configuration errors could render the TVC system ineffective.

Launch 3 introduced corrected axis mapping and modest filtering, resulting in greatly improved behavior. Pitch and roll remained within a narrow band, gyro drift decreased substantially, and accelerometer data became more interpretable. Although the vehicle did not execute a full ascent, launch 3 conclusively demonstrated that proper sensor alignment and even minimal filtering dramatically enhance stability.

Launch 4 again produced no telemetry due to a software initialization failure, and post impact inspection revealed damage to the flight computer's BOOTSEL interface, preventing data recovery. The behavior captured in video closely resembled Launch 1, with immediate instability and no evidence of closed loop control. This launch underscored the importance of robust software deployment, initialization checks, and the need for more survivable electronics mounting strategies.

Launch 5 marked the first successful demonstration of sustained thrust vector controlled ascent. Using a backup STARLIGHT board and a low pass filtered, continuously updating control loop, the rocket maintained bounded pitch and roll throughout the powered flight. Sensor data showed

moderate but manageable rotation rates, clean vertical acceleration, and a monotonic altitude increase of approximately 23 meters. Launch 5 therefore established the clearest evidence of a functioning, noise tolerant, closed loop TVC system.

Despite these advances, launch 5 also exposed a major limitation in the recovery system, as the parachute failed to deploy properly. The incomplete deployment resulted in a high speed impact that caused significant structural damage to the rocket body and compromised the flight computer housing. This failure highlights a critical area for future iteration, ensuring reliable parachute ejection, correct packing procedures, robust retention mechanisms, and sufficient deployment force. Stabilizing ascent is necessary for successful missions but without a dependable recovery sequence, vehicle reusability, post flight data extraction, and long term testing cannot be sustained.

In summary, the progression from Launch 1 to 5 demonstrates the maturation of the TVC system from highly unstable initial behavior to stable ascent under the backup configuration. The series identifies key areas of focus for future work: improved filtering strategies, better roll bias management, enhanced initialization reliability, and a redesigned recovery system capable of surviving nominal and off nominal flights. Launch 5 provides a strong foundation on which to build the next iteration of both flight control and recovery subsystems.

8. Conclusion

This project set out to design, model, implement, and flight test a thrust vector controlled (TVC) finless model rocket capable of demonstrating stable vertical ascent using real time guidance, navigation, and control (GNC). The work bridges a fundamental gap between traditional model rocketry, which relies primarily on passive aerodynamic stability, and the actively controlled vertical takeoff and landing (VTVL) rockets used by modern reusable launch vehicles. By integrating a dual axis gimbal, an embedded flight computer, and a mathematically grounded control algorithm, the project provides a complete end to end investigation of TVC at small scale and adds to the growing body of research on controllable amateur rockets.

From a hardware standpoint, the project produced a fully integrated flight system built around the STARLIGHT flight computer. With its RP2040 microcontroller, onboard IMU, barometer, servo outputs, and 16 MB of nonvolatile flash memory, STARLIGHT eliminated the need for a custom PCB while still providing the computational and sensing capabilities needed for closed loop control. Likewise, the dual ring gimbal optimized from the BPS Space design offered mechanically decoupled pitch and yaw control, smooth articulation, and compatibility with the Estes F15 motor. This combination of electronics tuned simulation parameters, and a proven TVC gimbal significantly reduced development risk and allowed greater emphasis on control system performance.

A key contribution of the project is the development of a dynamic model that supports multiple control analysis approaches. Starting from Euler's rigid body equations, both nonlinear and small angle linearized forms of the rocket's rotational dynamics were derived, capturing how gimbal deflections generate corrective torques. These equations were expressed in state space form to create a direct relationship between control inputs and attitude response, enabling analysis through PID transfer function modeling, MATLAB's 3DOF body axes simulation environment, and optimal control techniques such as LQR and LQG. Together, these methods formed an integrated simulation and control framework that incorporated thrust curve interpolation, servo reaction dynamics, and closed loop feedback, providing high fidelity predictions of the rocket's performance across the F15 motor burn.

The transition from simulation to flight testing formed the most significant experimental component of the work. Across five launches at the Black Rock Desert, the rocket progressed from early instability and missing telemetry to a stable, thrust vector controlled ascent in the final flight. Launches 1 and 2 revealed critical software and sensor fusion issues, including axis misalignment, insufficient filtering, and missing flash logging routines. These failures demonstrated the importance of reliable data acquisition and clean IMU preprocessing in TVC applications. Launch 3 marked a turning point, showing that even modest filtering and correct axis mapping substantially improved stability. Launch 4 highlighted the need for robust initialization and avionics mounting after a boot failure prevented logging. Finally, launch 5 achieved the central goal of this project: a sustained TVC stabilized ascent with bounded pitch and roll, clean acceleration profiles, and monotonic altitude increase. Clear evidence that the control system, gimbal, and avionics were functioning as intended. The only major failure in Launch 5 occurred during recovery, when a partial parachute deployment resulted in significant airframe damage,

underscoring the importance of reliable recovery mechanisms for iterative experimentation. These results demonstrate that a finless model rocket can achieve stable vertical ascent using thrust vector control alone, even when subject to the inertial, aerodynamic, and vibrational constraints of amateur rocketry.

Looking forward, several improvements can be made to advance this project. The recovery mechanism requires substantial redesign to ensure reliable parachute deployment and to prevent structural damage during landing. Sensor performance also emerged as a limiting factor during flight testing. The IMU exhibited significant noise and gyro drift, which introduced fluctuations in the estimated attitude and reduced the controller's ability to maintain stable flight. Further progress may be achieved through the development of a complete 6 DOF model incorporating aerodynamic forces and moments, enabling more advanced gain scheduled or optimal controllers suitable for aggressive maneuvers, guided descent, and, ultimately, powered landing. Additional refinements to the hardware, including better vibration isolation, strengthened avionics mounting, and upgraded servos or gimbal geometries, may also broaden the attainable control envelope in future iterations.

In conclusion, this project successfully demonstrated the feasibility of thrust vector control in a model rocket and established a complete workflow that spans literature review, dynamic modeling, hardware integration, simulation, and field testing. The stable ascent achieved in the final launch validates the theoretical and computational foundations laid throughout the report and provides a strong basis for future students, researchers, and hobbyists seeking to develop fully reusable, vertically landing small scale rockets. The work illustrates not only the challenges of implementing advanced guidance and control techniques on constrained hardware but also the significant potential of TVC when supported by rigorous engineering and iterative experimentation.

References

[1] Drake, N. (2015, December 22). Touchdown! spacex rocket makes a perfect upright landing. National Geoographic. https://www.nationalgeographic.com/science/article/151222-spacex-landing-rocket-video-space

[2] Costa, B. A., Parente, F. L., Belfo, J., Somma, N., Rosa, P., Igreja, J. M., Belhadj, J., & Lemos, J. M. (2024). A reinforcement learning approach for adaptive tracking control of a reusable rocket model in a landing scenario. Neurocomputing, 577, 127377. https://doi.org/10.1016/j.neucom.2024.127377

[3] Botelho, A., Martinez, M., Recupero, C., Fabrizi, A., & De Zaiacomo, G. (2022). Design of the landing guidance for the retro-propulsive vertical landing of a reusable rocket stage. CEAS Space Journal, 14(3), 551–564. https://doi.org/10.1007/s12567-022-00423-6

[4] Li, J., Long, Y., Su, M., Liu, L., Wang, B., & Cheng, Z. (2022). Fault-tolerant guidance of Rocket Vertical Landing Phase based on MPC Framework. International Journal of Aerospace Engineering, 2022, 1–11. https://doi.org/10.1155/2022/9104823

[5] Xiong, F., Li, C., Zhao, Y., & Hu, X. (2022). Rocket landing guidance using convex optimization and proportional navigation considering performance-limited engine. Acta Astronautica, 201, 209–223. https://doi.org/10.1016/j.actaastro.2022.07.033

[6] Ding, S., Liu, L., & Zhang, S. (2023). Dynamics model and numerical simulation of Single Stage Water Rocket. Journal of Physics: Conference Series, 2508(1), 012053. https://doi.org/10.1088/1742-6596/2508/1/012053

[7] Sopegno, L., Livreri, P., Stefanovic, M., & Valavanis, K. P. (2023). Thrust vector controller comparison for a Finless Rocket. Machines, 11(3), 394. https://doi.org/10.3390/machines11030394

[8] M, M., T, L., & G, M. K. (2022). A review of various control algorithms used for a solid rocket motor to achieve vertical takeoff and vertical landing. International Journal of Recent Technology and Engineering (IJRTE), 11(4), 68–73.https://doi.org/10.35940/ijrte.d7337.1111422

[9] Kishore, S., & Laxmi, V. (2019, May 6). Closed-loop control system with Cascade PID controller. | download scientific diagram. Research Gate. https://www.researchgate.net/figure/A-PID-controller-in-cascade-with-the-plant_fig1_339640693

[10] Ogata, K. (2021). Modern Control Engineering (5th ed.). Pearson.

[11] Anderson, M. (n.d.). PID tuning: How to tune a PID controller - realpars. RSS. https://www.realpars.com/blog/pid-tuning

[12] How to tune a PID controller • PID explained. PID Explained. (2024, July 16). https://pidexplained.com/how-to-tune-a-pid-controller/

[13] Tuning methods of a PID controller. CTRL ALT FTC. (n.d.). https://www.ctrlaltftc.com/the-pid-controller/tuning-methods-of-a-pid-controller

[14] Rajendran, P. (2015, July). Figure 2: Simulink model of LQR controller. Research Gate. https://www.researchgate.net/figure/Simulink-model-of-LQR-controller_fig1_281255620

[15] Gogoi, D., Buragohain, M., & Patowary, M. (1970, January 1). Comparative study of load frequency control using LQG and mras controllers. SpringerLink. https://link.springer.com/chapter/10.1007/978-981-99-4175-9_29

[16] AeroTech. (2007, January 9). Thrustcurve Hobby Rocket Motor Data for Estes F15. • ThrustCurve. https://www.thrustcurve.org/motors/Estes/F15/

[17] Bernard, J. (n.d.). Thrust Vector Control 3D files. BPS.Space. https://bps.space/products/thrust-vector-control

[18] Wizardy, C. (2023, August 9). Circuitwizardry. https://circuitwizardry.com/wp-content/uploads/2024/04/STARLIGHT-MINI-User-Manual.pdf

[19] Raspberry Pi Foundation. (2025, February 20). Raspberrypi. raspberrypi.com. https://pip.raspberrypi.com/documents/RP-008371-DS-1-rp2040-datasheet.pdf

[20] BMP388 digital pressure sensor. BMP388 Digital Pressure Sensor Data Sheet. (n.d.). https://www.bosch-sensortec.com/media/boschsensortec/downloads/datasheets/bst-bmp388-ds001.pdf

[21] ICM-42605 datasheet - TDK Invensense. (n.d.). https://invensense.tdk.com/wp-content/uploads/2022/09/DS-000292-ICM-42605-v1.7.pdf

[22] Bosch Sensortec. (n.d.). Bosch Sensortec | BMP388 Datasheet 1 | 57. https://www.mouser.com/pdfdocs/BST-BMP388-DS001-01.pdf

[23] Winbond Electronics Corporation. (2013, April 29). 3V 128M-BIT Serial Flash Memory with Dual/Quad SPI & QPI. PJRC. https://www.pjrc.com/teensy/W25Q128FV.pdf

[24] Texas Instruments. (2021, July). TLV767 1-A, 16-V Precision Linear Voltage Regulator. ti.com.https://www.ti.com/lit/ds/symlink/tlv767-q1.pdf?ts=1696809772851&ref_url=https%253A%252F%252Fwww.ti.com%252Fpower-management%252Flinear-regulators-ldo%252Fproducts.html

[25] Texas Instruments. (2007, December). TXS0108E 8-bit bi-directional, level-shifting, voltage translator. ti.com. https://www.ti.com/lit/ds/symlink/txs0108e.pdf

Appendix A – MATLAB Simulation Code for TVC Controllers

```
%% Surya Shivakumar Block PID, Transfer PID, LQR, & LQG Controllers for a Thrust Vector Rocket
clc, clear all, close all

%% Constants and Rocket Characteristics
q
T = 15; % [N] Average Thrust
d = 0.485; % [m] distance between cg and gimbal
I = 0.05; % [kg*m^2] inertia of model rocket

mass = 1; % [kg] mass of model rocket w/ motors
moment_arm = 0.295; % [N*m] moment arm of gimbal

sample_t = 0.01;
servo_reaction_time = 0.05; % seconds [s]

initial_angle = 0; % starting angle
target_angle = 5; % target angle

%% Thurst Curve of Rocket

burn_time = 3.5; % [s] burn time of estes F15
f15_time = 0:0.1:burn_time; % Estes F15 flight time
f15_curve = [0, 5, 10, 15.5, 22.75, 25, 21, 19, 16, 15.5, 15, 15, 15,...
        15, 14.5, 14.5, 14.5, 14.5, 14.5, 14.5, 14.5, 14.5, 14,...
        14, 14, 14, 13.75, 13.5, 13, 13, 13, 13, 13, 12.75, 12.5, 0];

% thrust curve of rocket
thrust_curve = [f15_time', f15_curve'];

%% Fixed Body Mass PID for Body-Based PID

P_body = 0.35; % Porportional
I_body = 1.1; % Integral
D_body = 0.075; % Derivative

%% Total State Space Transfer Function

% Total State Space Blocks
A = [0 0 1 0; 0 0 0 1; 0 0 0 0; 0 0 0 0];
B = [0 0; 0 0; T*d/I 0; 0 T*d/I];
C_tot = eye(4);
D = zeros(4,2);

% Configurating State Space and TF
```

```
sys_tot = ss(A,B,C_tot,D);
tf_tot = tf(sys_tot);

co_tot = ctrb(A,B) % controllability matrix for total
co_tot_rank = rank(co_tot) % controllability rank for total
ob_tot = obsv(A,C_tot) % observable matrix for total
ob_tot_rank = rank(ob_tot) % observable rank for total

%% Simplified Pitch Only State Space Transfer Function

% Pitch Only State Space Blocks
Ap = [0 1; 0 0];
Bp = [0; T*d/I];
Cp = [1 0];
Dp = [0];

% Configurating State Space and TF
sys_pitch = ss(Ap,Bp,Cp,Dp)
tf_pitch = tf(sys_pitch)

tf_p_num = [145.5] % numerator of TF
tf_p_den = [1 0 0] % denomenator of TF

% Testing Pitch Controllability and Observability
co_pitch = ctrb(Ap,Bp) % controllability matrix for pitch
co_pitch_rank = rank(co_pitch) % controllability rank for pitch
ob_pitch = obsv(Ap,Cp) % observable matrix for pitch
ob_pitch_rank = rank(ob_pitch) % observable rank for pitch

%% PID, LQR, LQG Values for Transfer Functions

% PID Values for Transfer Functions
Kp_tf = 0.35;
Ki_tf = 1.10;
Kd_tf = 0.075;

% LQR Values
Q_lqr = [10 0; 0 1]; % Q_lqr matrix
R_lqr = 10; % R_lqr matrix
[K,S,e] = lqr(Ap, Bp, Q_lqr, R_lqr); % lqr block

% LQG Values
G = eye(2);
Q_lqg = diag([0.0125 0.0250]); % Q_lqg matrix
R_lqg = 1; % R_lqg matrix
```

```
[L,P,e_kf] = lqe(Ap,G,Cp,Q_lqg,R_lqg)
sensor_noise = 0.5;

%% Sims and Plotting

open("final_report_sim.slx");
sim("final_report_sim.slx");

%
target = ans.target();
pitch_att = ans.pitch_att();
PID_out = ans.PID_out();

open("final_controller_response.slx");
sim("final_controller_response.slx");

%% Extracting Python PID Data for Comparisons

[t, theta_actual, theta_target, pid_output] = python_pid_export();

%% Plotting

% Thrust Curve
figure,
plot(thrust_curve(:,1), thrust_curve(:,2))
title("Thrust Curve of Estes F15")
xlabel("Time [s]")
ylabel("Thrust [N]")
set(findall(gcf, 'Type', 'Line'), 'LineWidth', 2);
set(gca, 'FontSize', 12);

% Open-Loop Response
figure,
plot(ans.open_loop(:,1), ans.open_loop(:,2))
title("Open Loop Response of Transfer Function")
xlabel("Time [s]")
ylabel("Angle [deg]")
set(findall(gcf, 'Type', 'Line'), 'LineWidth', 2);
set(gca, 'FontSize', 12);

% Plotting Body PID Response
figure,
plot(target(:,1), target(:,2))
hold on
plot(pitch_att(:,1), pitch_att(:,2))
plot(PID_out(:,1), PID_out(:,2))
```

```
legend('Target Angle', 'Pitch Orientation', 'PID Output', 'Location','best')
title("Matlab PID Controller with 3DOF Block Response to Target Angle")
xlabel("Time [s]")
ylabel("Angle [deg]")
set(findall(gcf, 'Type', 'Line'), 'LineWidth', 2);
set(gca, 'FontSize', 12);

% Plotting Matlab PID TF Response
figure,
plot(ans.input(:,1), ans.input(:,2))
hold on
plot(ans.pid_tf_out(:,1), ans.pid_tf_out(:,2))
legend('Target Angle', 'Pitch Orientation')
title("Matlab PID Controller with Transfer Function Response to Target Angle")
xlabel("Time [s]")
ylabel("Angle [deg]")
set(findall(gcf, 'Type', 'Line'), 'LineWidth', 2);
set(gca, 'FontSize', 12);

% Plotting Python PID Response w/ Data
figure,
plot(t, theta_target)
hold on
plot(t, theta_actual)
plot(t, pid_output)
legend('Target Angle', 'Pitch Orientation', 'PID Output', 'Location', 'best')
title("Python PID Response to Target Angle")
xlabel("Time [s]")
ylabel("Angle [deg]")
set(findall(gcf, 'Type', 'Line'), 'LineWidth', 2);
set(gca, 'FontSize', 12);

% Comparing Matlab PID Outputs
figure,
plot(ans.input(:,1), ans.input(:,2))
hold on
plot(pitch_att(:,1), pitch_att(:,2))
plot(ans.pid_tf_out(:,1), ans.pid_tf_out(:,2))
legend('Target Angle', 'Pitch - 3DOF', 'Pitch - TF', 'location', 'best')
title("Comparing Matlab PID Response to Target Angle ")
xlabel("Time [s]")
ylabel("Angle [deg]")
set(findall(gcf, 'Type', 'Line'), 'LineWidth', 2);
set(gca, 'FontSize', 12);

% Comparing PID Outputs
```

```
figure,
plot(ans.input(:,1), ans.input(:,2))
hold on
plot(pitch_att(:,1), pitch_att(:,2))
plot(ans.pid_tf_out(:,1), ans.pid_tf_out(:,2))
plot(t, theta_actual)
legend('Target Angle', 'Pitch - 3DOF', 'Pitch - TF', 'Pitch - Python','location', 'best')
title("Comparing Matlab and Python PID Response to Target Angle ")
xlabel("Time [s]")
ylabel("Angle [deg]")
set(findall(gcf, 'Type', 'Line'), 'LineWidth', 2);
set(gca, 'FontSize', 12);

% LQR Response
figure,
plot(ans.input(:,1), ans.input(:,2))
hold on
plot(ans.lqr_out(:,1), ans.lqr_out(:,2))
title("Matlab LQR Controller Response to Target Angle")
xlabel("Time [s]")
ylabel("Angle [deg]")
set(findall(gcf, 'Type', 'Line'), 'LineWidth', 2);
set(gca, 'FontSize', 12);
axis([0 3.5 0 6])

% Comparing LQR and PID TF Response
figure,
plot(ans.input(:,1), ans.input(:,2))
hold on
plot(ans.lqr_out(:,1), ans.lqr_out(:,2))
plot(pitch_att(:,1), pitch_att(:,2))
plot(ans.pid_tf_out(:,1), ans.pid_tf_out(:,2))
legend('Target Angle', 'Pitch - LQR', 'Pitch - 3DOF', 'Pitch - TF')
title("Comparing Matlab LQR Controller Response to PID Response to a Target Angle")
xlabel("Time [s]")
ylabel("Angle [deg]")
set(findall(gcf, 'Type', 'Line'), 'LineWidth', 2);
set(gca, 'FontSize', 12);

% LQG Response
figure,
plot(ans.lqg_out(:,1), ans.lqg_out(:,2))
title("Matlab LQG Controller Response to Target Angle")
xlabel("Time [s]")
ylabel("Angle [deg]")
set(findall(gcf, 'Type', 'Line'), 'LineWidth', 2);
```

```
set(gca, 'FontSize', 12);

% Comparing LQR and LQG Response
figure,
plot(ans.lqr_out(:,1), ans.lqr_out(:,2))
hold on
plot(ans.lqg_out(:,1), ans.lqg_out(:,2))
legend('LQR Pitch Response', 'LQG Pitch Response')
title("LQR and LQG Response to Target Angle")
xlabel("Time [s]")
ylabel("Angle [deg]")
set(findall(gcf, 'Type', 'Line'), 'LineWidth', 2);
set(gca, 'FontSize', 12);
```

Appendix B – Python 3 TVC PID Simulation

```
# Surya Shivakumar, Python PID Simulation for a Thrust Vector Rocket
# Imports and Libraries
import matplotlib.pyplot as plt
import math
import random

# Simulation Constants
pi = 3.14;
dt = 0.02; # time step [s]
I = 0.05; # [kg*m^2] moment of inertia
runtime = 3.5; # [s] burn time of Estes F15 Motor
samples = int(runtime/dt) + 1; #
d = 0.485; # [m] cg to bottom of rocket
thrust = 15; # [N] approximate average thrust

# Target orientation angle
theta_goal_deg = 5; # [deg] target angle in degrees
theta_goal_rad= theta_goal_deg * pi/180; # [rad] converts to radians

# PID Controller Gains
Kp = 35
Kd = 5
Ki = 0.2

running_sum = 0; # integral accumalator
max_angle = 5; # max angle of gimbal

# Array Initialization Initial Conditions
theta = [0 for t in range(samples)]
t = [x * dt for x in range(samples)]
torque = [0 for q in range(samples)]
setpoint = [0 for ang in range(samples)]

# Estes F15 Thrust Curve
f15_time = [i * 0.1 for i in range(36)]
f15_curve = [0, 5, 10, 15.5, 22.75, 25,
        21, 19, 16, 15.5, 15, 15,
        15, 15, 14.5, 14.5, 14.5, 14.5,
        14.5, 14.5, 14.5, 14.5, 14, 14,
        14, 14, 13.75, 13.5, 13, 13,
        13, 13, 13, 12.75, 12.5, 0]

# Function to get Thrust [N] at Specific Time [s]
# uses linear interpolation
def getForce(t):
```

```
    if t <= f15_time[0] or t >= f15_time[-1]:
        return 0.0

    # Find the interval [t_i, t_{i+1}] where t lies
    for i in range(1, len(f15_time)):
        if t < f15_time[i]:
            t1, t2 = f15_time[i - 1], f15_time[i]
            f1, f2 = f15_curve[i - 1], f15_curve[i]

            # Linear interpolation
            slope = (f2 - f1) / (t2 - t1)
            return f1 + slope * (t - t1)

    return 0.0

# Initial Conditions
initial_theta = 0;
theta[0] = initial_theta*pi/180
theta[1] = initial_theta*pi/180
running_sum = theta[0] + theta[1]

# PID Simulation Loop
for n in range(2, samples):
    # torque equation
    torque[n-1] = d * getForce(n*dt)*math.sin(pi*setpoint[n-1]/180.)

    # double integrator for rotational dynamics
    theta[n] = 2*theta[n-1] - theta[n-2] + torque[n-2]*dt*dt/I

    # total error over time
    running_sum = running_sum + (theta[n]-theta_goal_rad)

    # PID Controller for Target and Saves Setpoint
    target = - Kp*(theta[n]-theta_goal_rad) - Kd*(theta[n]-theta[n-1])/dt - Ki*running_sum
    setpoint[n] = target

# Sets Up Arrays for Plotting
theta_deg = [math.degrees(th) for th in theta]
theta_goal_array = [theta_goal_deg for _ in range(samples)]


# Plotting
plt.figure(figsize=(10, 5))
plt.plot(t, theta_deg,  color='royalblue', label='PID Output',)
plt.plot(t, setpoint, color='orange', label='Setpoint')
plt.plot(t, theta_goal_array, color='red', label='Target Angle')
plt.xlabel("Time (s)", fontsize=10)
```

```
plt.ylabel("Setpoint Angle (°)", fontsize=10)
plt.title("Desired Setpoint Over Time", fontsize=12)
plt.grid(True)
plt.legend()
plt.tight_layout()
plt.show()
```

Appendix C – MicroPython Main Rocket Launch Code

```
# Surya Shivakumar, Main Rocket Launch Code
# Imports and Libraries

import starlight
from starlight import Servo
import fusion
import os
from machine import Pin, PWM, I2C
import time
import math

## Flight Board LED Light Initiation
# shows that the flight computer is running code

led = Pin(24, Pin.OUT)
led.value(1)

## i2c and sensor Initialization
i2c = machine.I2C(1, scl=machine.Pin(3), sda=machine.Pin(2), freq = 9600)


# creates bmp388 Object
bmp = starlight.BMP388(i2c, 0x76)
bmp.enable_temp_and_pressure()
bmp.calibrate()

# fusion object for pitch and roll
fus = fusion.Fusion()

# ICM42605 intialization
icm = starlight.ICM42605(i2c, 0x68)
icm.config_gyro
icm.enable()

## Warmup and Calibration
warmup_time_ms = 20000; # 20 second warmup
stabilty_interval = 500; # checks every 0.5s
stability_threshold = 0.05; # deg/s, consider stable if bias changes less than this

print("Warming up IMU: ICM42605...")
warmup_start = time.ticks_ms()

# previous values
pgx = 0;
pgy = 0;
```

```
pgz = 0;

# loop warming up ICM42605
while time.ticks_diff(time.ticks_ms(), warmup_start) < warmup_time_ms:
    icm.updateData(0)
    gx, gy, gz = icm.gx, icm.gy, icm.gz
    print("gx: " + str(gx) + " gy: " + str(gy) + " gz: " + str(gz))
    time.sleep_ms(stabilty_interval)


# ICM42605 Gyro Bias
icm.calibrate_gyro_bias(samples=500, delay=0.01) # gathers gx, gy, gz bias

# motor and parachute pins
pin1 = 6; # parachute ejection pin
pin2 = 7; # motor ignition pin

parachute_pin = machine.Pin(pin1, machine.Pin.OUT)
motor_ignition_pin = machine.Pin(pin2, machine.Pin.OUT)

## Servo Initiation

servo_enable = Pin(14, Pin.OUT)
servo_enable.value(1)

servoX = PWM(Pin(11))
servoY = PWM(Pin(12))

# Servo Positions

# Servo Y
servoYcenter = 6500;
servoYback = 5500;
servoYfront = 7500;

# Servo X
servoXcenter = 5500;
servoXright = 6500;
servoXleft = 4500;

servoX.freq(50)
servoY.freq(50)

servoX.duty_u16(servoXcenter)
servoY.duty_u16(servoYcenter)
```

```
## Logging Setup

date = "112125"  # <-- manually set date here (MMDDYY format)
n = 1

# Find next available flight log filename
while "flight_log_rk3_test_{}_{}.csv".format(date, n) in os.listdir("/"):
    n += 1

log_filename = "/flight_log_rk3_test_{}_{}.csv".format(date, n)

try:
    log_file = open(log_filename, "w")
    log_file.write("time_S, pitch_deg, mod_pitch, roll_deg, mod_roll, altitude_m, P_set, R_set,
gx, gy, gz, ax, ay, az, velo, prcht_e\n")
    log_file.flush()
    print("Logging to: ", log_filename)
except Exception as e:
    print("Logging setup failed:", e)
    log_file = None

## PID Setup

pi = 3.14;
desiredRoll = 0;
desiredPitch = 0;

Kp = 45 * pi/180;
Kd = 10 * pi/180;
Ki = 0.2 * pi/180;

roll_integral = 0;
pitch_integral = 0;

roll_old = 0;
pitch_old = 0;

roll_setpoint = 0;
pitch_setpoint = 0;


# pause before ignition and calibration section
ignition_timer = time.ticks_ms()
ignition_runtime_ms = 10000; # countdown in ms

start_interval = 0.25;
end_interval = 1;
```

```
while time.ticks_diff(time.ticks_ms(),ignition_timer) < ignition_runtime_ms:
    elapsed_var_s = time.ticks_diff(time.ticks_ms(), ignition_timer)/1000;
    ignition_cdw = ignition_runtime_ms/1000 - elapsed_var_s;
    led_interval = start_interval + (end_interval -
start_interval)*(elapsed_var_s/(ignition_runtime_ms/1000))

    led.value(1)
    time.sleep(led_interval)
    led.value(0)
    time.sleep(led_interval)


# Roll and Pitch Calibration
current_pitch = 0;
current_roll = 0;
# pitch_array = 0
# roll_array = 0;
# flight_time_array = 0;

## Calibration
# number of samples
calibration_samples = 350;
calibration_time = time.ticks_ms()

pitch_sum = 0;
roll_sum = 0;

for i in range (calibration_samples):
    data = icm.get_accel_and_gyro_data()

    # getting in accel and gyro
    ax, ay, az = data[0], data[1], data[2]
    gx, gy, gz = data[3], data[4], data[5]

    fus.update_nomag((ax, az, ay), (gx, gz, gy)) # the y and z axis where switched

    current_pitch = fus.pitch;
    current_roll = fus.roll;

    pitch_sum += current_pitch
    roll_sum += current_roll

    # print("Pitch: " + str("%.2f" % current_pitch) + " Roll: " + str("%.2f" % current_roll) + "
CALIBRATION ")
    led.value(1)
```

```
pitch_calibration = pitch_sum / calibration_samples
roll_calibration  = roll_sum / calibration_samples
print("Pitch Calibration: " + str(pitch_calibration) + " Roll Calibration: " + str(roll_calibration))
led.value(0)

## PID Loop and Gimbal Characteristics

maxDegreesX = 5;
maxDegreesY = 5;

deg_per_x = 0.005;
deg_per_y = 0.005;

init_pressure = bmp.getPressure()
init_alt = bmp.getAltitude(init_pressure)

rocket_alt = bmp.getRocketAlt(init_alt, bmp.getPressure()) # different so check
vertical_velocity = 0;

previous_pitch = 0;
previous_roll = 0;
previous_alt = 0;

flight_time_ms = 4500; # approximate time to reach apogee
data_collection_time = 10000; # how long the flight computer stays on for the PID loop
launch_time = time.ticks_ms()
start = time.ticks_ms()
previous_time = time.ticks_ms()

modifier = 1;
alpha = 0.5; # low pass filter alpha
parachute_ejection = 0;
count = 0;

while time.ticks_diff(time.ticks_ms(), launch_time) < data_collection_time:
# while True:
    icm.updateData(time.ticks_diff(time.ticks_ms(), start))
    start = time.ticks_ms()

    current_time = time.ticks_ms()
    deltaT = time.ticks_diff(current_time, previous_time) # real loop time
    previous_time = current_time

    data = icm.get_accel_and_gyro_data()
```

```
# getting in accel and gyro
ax, ay, az = data[0], data[1], data[2]
gx, gy, gz = data[3], data[4], data[5]

fus.update_nomag((ax, az, ay), (gx, gz, gy)) # the y and z axis where switched
launch_time_elapsed = time.ticks_diff(time.ticks_ms(), launch_time)/100
count += 1

if count == 1:
    motor_ignition_pin.value(1)
    time.sleep(0.01) # delay so that it can fly straight
    motor_ignition_pin.value(0)
    print("motor ignition")


if count > 5:
    if count % 1 == 0:

        # roll and pitch data collection
        current_pitch = fus.pitch - pitch_calibration;
        current_roll = fus.roll - roll_calibration;

        pressure = bmp.getPressure()
        altitude = bmp.getAltitude(pressure)
        rocket_altitude = bmp.getRocketAlt(init_alt, pressure)

        # Low Pass Filter

        filtered_pitch = alpha * current_pitch + (1 - alpha) * previous_pitch
        filtered_roll = alpha * current_roll + (1 - alpha) * previous_roll

        previous_pitch = filtered_pitch
        previous_roll = filtered_roll

        # Modifier, knocks out very small pitch/roll and sets it to 0

        angle_mag = abs(filtered_pitch) + abs(filtered_roll)

        if angle_mag < 2:
            modifier = 0;
        elif angle_mag < 5:
            modifier = 0.25;
        else:
            modifier = 1;

        filtered_pitch = filtered_pitch * modifier
```

```
        filtered_roll = filtered_roll * modifier

        # PID Loop for Pitch and Roll

        roll_derivative = icm.gz
        roll_integral = roll_integral + (filtered_roll-desiredRoll)*(deltaT)
        rollSetpoint = -Kp*(filtered_roll-desiredRoll) - Kd*(roll_derivative) - Ki*(roll_integral)

        pitch_derivative = icm.gx
        pitch_integral = pitch_integral + (filtered_pitch - desiredPitch)*(deltaT)
        pitchSetpoint = -Kp*(filtered_pitch-desiredPitch) - Kd*(pitch_derivative) -
Ki*(pitch_integral)

        servoYval = servoYcenter - filtered_roll/deg_per_y # center

        if( servoYval > servoYfront ):
           servoYval = servoYfront
           # print(servoYval)
        if( servoYval < servoYback ):
           servoYval = servoYback

        servoXval = servoXcenter + filtered_pitch/deg_per_x # plus or minus idk

        if( servoXval > servoXright ):
           servoXval = servoXright
           # print(servoXval)
        if( servoXval < servoXleft ):
           servoXval = servoXleft

        # Moving Servos

        servoY.duty_u16(int(servoYval))
        servoX.duty_u16(int(servoXval))

        # Time and Logging/Printing

        current_time = time.ticks_ms()
        delta_t = time.ticks_diff(current_time, start) / 1000.0

        # not most accurate readings
        vertical_velocity = (rocket_altitude - previous_alt) / delta_t
        previous_alt = rocket_altitude
        start = current_time

        if log_file:
           try:
```

```
                log_file.write(f"{launch_time_elapsed:.2f},{current_pitch:.2f},{filtered_pitch:.2f},{
current_roll:.2f},{filtered_roll:.2f},{altitude:.2f},{pitchSetpoint:.2f},{rollSetpoint:.2f},{icm.gx:.
2f},{icm.gy:.2f},
{icm.gz:.2f},{icm.ax:.2f},{icm.ay:.2f},{icm.az:.2f},{vertical_velocity:.2f},{parachute_ejection:.
2f}\n")
            except:
                pass

        print("Time: %.2f | Pitch: %.2f | Mod Pitch: %.2f | Roll: %.2f | Mod Roll: %.2f | Alt:
%.2f | P_set: %.2f |  R_set %.2f | gx %.2f | gy %.2f | gz %.2f | ax %.2f | ay %.2f | az %.2f |
velocity %.2f | pcht_e %.2f" %
              (launch_time_elapsed, current_pitch, filtered_pitch, current_roll, filtered_roll,
rocket_altitude, pitchSetpoint, rollSetpoint, icm.gx, icm.gy, icm.gz, icm.ax, icm.ay, icm.az,
vertical_velocity, parachute_ejection))

        if count % 15 == 0:
            print("flushing")
            log_file.flush()

    # look into putting this as one, but will that improve things?
    if count % 15 == 0:
        # print("hit at time: " + str(launch_time_elapsed))
        if abs(current_pitch) > 60 or abs(current_roll) > 60:
            print("Motor Burnout Deploying parachute!")
            parachute_pin.value(1)
            time.sleep(0.01)
            parachute_pin.value(0)
            parachute_deployed = True
            parachute_ejection = 1;

        if time.ticks_diff(time.ticks_ms(), launch_time) > flight_time_ms: # after
            print("Motor Burnout Deploying parachute!")
            parachute_pin.value(1)
            time.sleep(0.01)
            parachute_pin.value(0)
            parachute_deployed = True
            parachute_ejection = 1;
            # break

        # combine if
        # if abs(current_pitch) > 60 or abs(current_roll) > 60 or time.ticks_diff(time.ticks_ms(),
launch_time) > flight_time_ms:
            # add in parachute_pin ejection


# resetting servos
```

```
servoX.duty_u16(servoXcenter)
servoY.duty_u16(servoYcenter)

print("Succesful Flight Parachute Deployed!")
parachute_pin.value(1)
time.sleep(0.25)
parachute_pin.value(0)
parachute_deployed = True
parachute_ejection = 1;


# logging cleanup
if log_file:
    try:
        log_file.close()
        print("Flight log saved as flight_log.csv")
    except:
        print("Error closing log file")

print("Flight Complete.")
```