Developing a Control Algorithm and Simulation for Thrust Vector Controlled Rockets

New Mexico Supercomputing Challenge
Final Report
April 6, 2022

Team 47
Los Alamos High School

Team Members:
- Daniel Kim
- Andres Iturregui

Project Mentor:
- Chris Karr

Executive Summary

In the past five years, SpaceX has revolutionized the aerospace industry by introducing a new rocket that can propulsively land. At the heart of these rockets, and virtually all orbital and suborbital spacecraft is thrust vector control (TVC). TVC is where the engine of the rocket is moved using a gimbal to vector its direction and create torque forces to keep a rocket on its desired trajectory. The purpose of this project was to create a model rocket with a flight computer, develop a control algorithm for the TVC system, and program a simulation to model the flight of the thrust vector controlled rocket.

The model rocket was designed using CAD software and made using a 3D printer, and is capable of vectoring the rocket motor 5 degrees in any direction. The control system we decided to use is a PID, or proportional-integral-derivative controller. We chose this due to its simplicity, effectiveness, and ability to tune to any rocket. The flight computer we designed is able to run this PID algorithm, along with logging flight data and controlling the engine mount.

However, we needed a method to tune this PID controller. In order to do this, we developed a simulation in MATLAB Simulink so that we could find the optimal PID values for our real flight. The simulation allowed us to plug in variables such as the mass of the rocket and the amount of wind and receive a flight trajectory of the rocket.

Once both the real-life model and the simulation were finished, we did an actual test flight of the rocket. Unfortunately, the rocket quickly flipped out of control, so the flight was unsuccessful. This unsuccessful flight was caused due to a misalignment in the TVC mount that caused the rocket to immediately go off of its straight trajectory. We hope to fly again soon with an improved real-world model and simulation to back it up.

Introduction

Currently, there are over 10,000 companies in space technology development with a combined value of over $4 trillion [5] dedicated to advancing technologies in navigation, tourism, national security, communication, and outer space research. This expansive growth prompts the need for new suborbital and orbital-class vehicles to transport these technologies into outer space. Although these vehicles, the majority being rockets, present multiple challenges, one of the biggest obstacles in developing a navigation and control system to guide them.

A majority of these spacecraft use a technique called thrust vector control (Figure 1). By angling, or vectoring, the direction of the thrusting component, the rocket has control over position and orientation, even in a non-atmospheric environment. Engineers have designed and implemented gimbals into the rocket engines to allow for this technique across a variety of different rockets, but the software to control these gimbals and the rockets become very complicated because of the high speeds, large masses, and precision required from spacecraft.

Furthermore, methods to tune these control algorithms also become very complex due to the high cost and a large number of variables involved in a rocket flight. Although a majority of rockets use the same technique as TVC for active control, the control system design and tuning present many challenges.

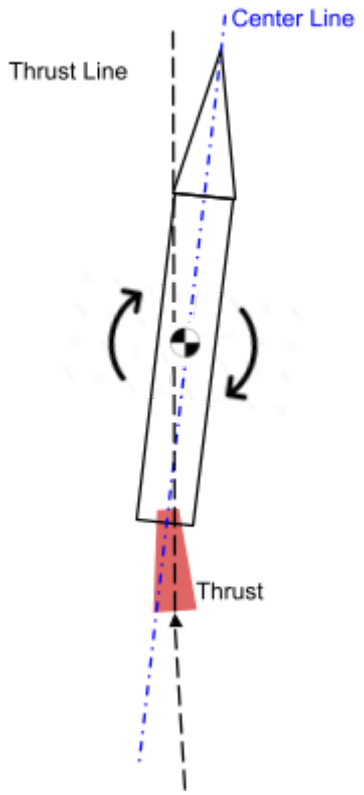

Figure 1. By vectoring the thrust, the rocket's orientation is rotated about the vehicle's center of mass. This method can be used to control the orientation and position of a model rocket.

The purpose of this research and engineering project was to design a versatile control system that is compatible throughout different dynamics of different rockets and to create a simulation to tune this control system and predict flight dynamics of different rockets. Implementation of this robust and dynamic control system and simulation offers a simpler way for the growing space industry to solve one of the most challenging problems of guidance, navigation, and control. In this project, a control algorithm, orientation scheme, and simulation were developed and tested in the form of a model rocket with a TVC system and control system implemented.

Control Theory

The control system that we decided to use and tailor to rocket guidance, navigation, and control, or GNC, was the very widely used Proportional-Integral-Derivative (PID) controller. This is a feedback controller that regulated steam engines during the industrial revolution, improved the yield from windmills and industrial processes worldwide, and lies at the heart of autopilots used in commercial airplanes. This control algorithm allows for a simple integration and tuning process for all rockets of various sizes and purposes by adjusting three values. The PID controller is defined as

$$u(t) = K_p e(t) + K_i \int e(t)\Delta t + K_d \frac{de}{dt} \quad (1)$$

Where u(t) is the output, K_p, K_i, and K_d are the three gains, e(t) is the error value and the input to the controller, and Δt is the change in time. There are three terms to this equation: the proportional, integral, and derivative of the error value $e(t)$, which are scaled using the adjustable gains K_p, K_i, and K_d. The proportional, integral, and derivative components allow for control over oscillations, overshoot, and small errors. This simplicity allows for use on lower-powered flight controllers, but the three components work together to produce a very robust, efficient, and versatile control system.

On a rocket, this method can be easily integrated. We can have three PID controllers, one for each axis to control orientation. The error, which is inputted into the controller, is equal to the setpoint subtracted from the current orientation. The setpoint is adjustable based on the desired angle. After performing the proportional, integral, and derivative components, we can then command these angles to the gimbal, or thrust vector control mount (Figure 2).

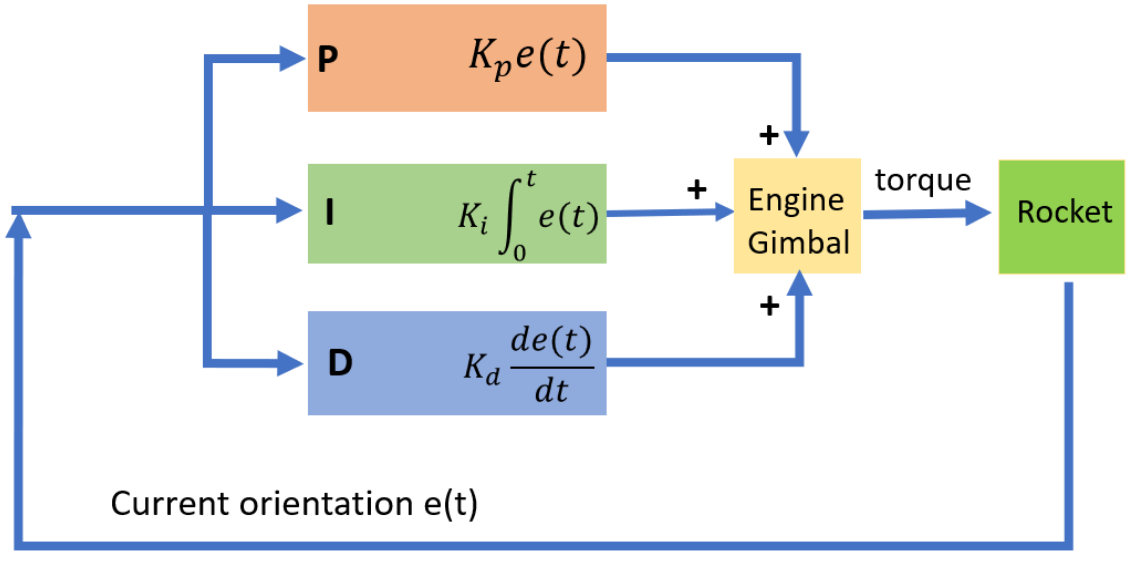

Figure 2. PID Controller for a rocket.

Methods

Real-World Model

To validate that the control system and simulation worked properly, we designed a model rocket and a TVC gimbal in computer-aided-design, CAD, and 3D printed components to replicate a real-life rocket.

The rocket itself was printed in PLA and supported by four carbon fiber rods. Parachutes that were stored in the upper body were deployed using a small pyrotechnic charge ignited by a load driver aboard the flight computer.

The thrust vector control mount was a two-axis (Figure 4) gimbal that was designed to vector the rocket motor using two servo motors that were controlled by the flight computer. The gear ratio of the motors and the mount was 2:1, which allowed for more accuracy and less error within the mount, and the mount had a range of +/- 5 degrees in both axes.

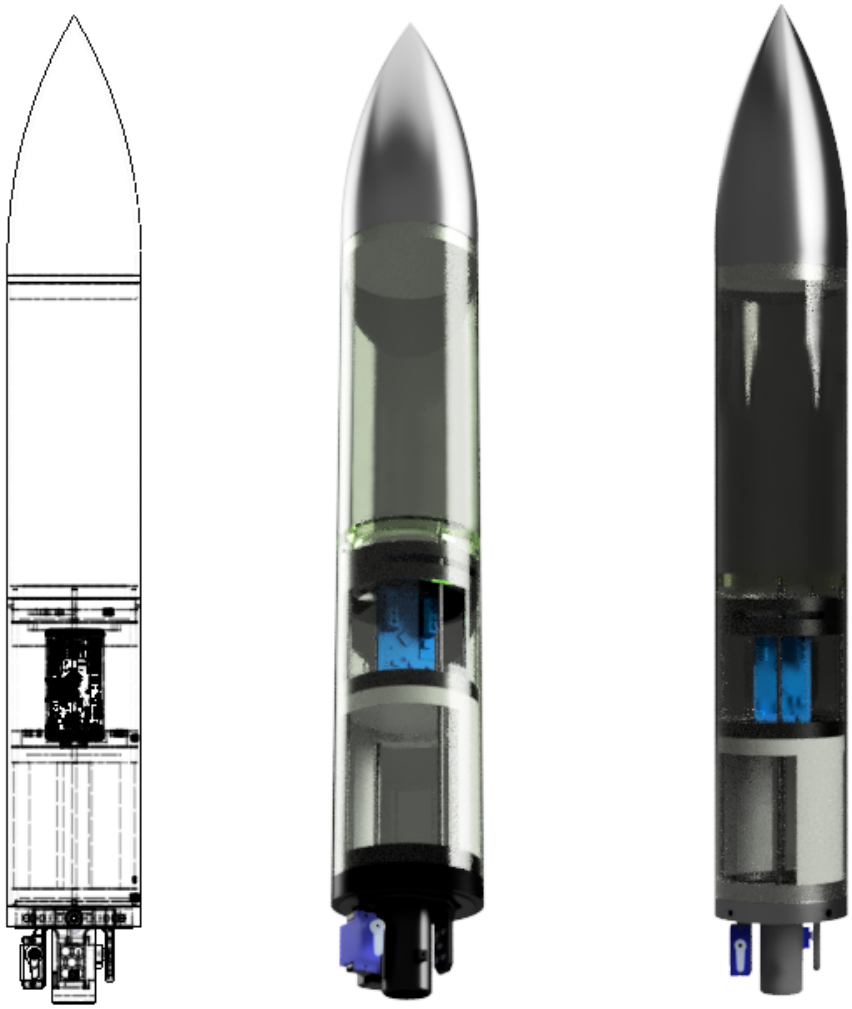

Figure 3. The model rocket design in CAD

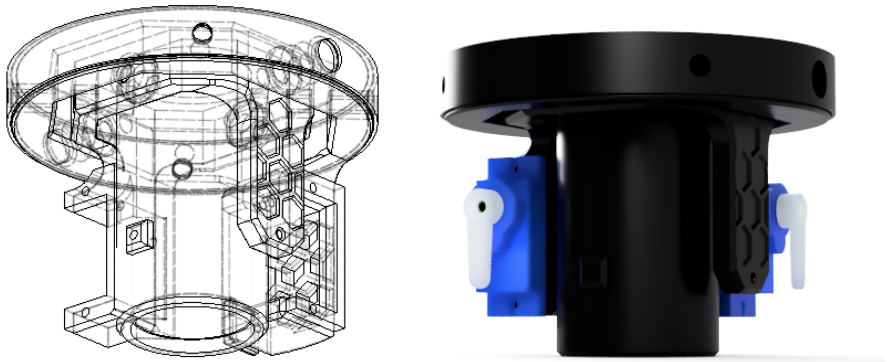

Figure 4. The thrust vector control gimbal

Flight Computer

To control the servo motors, calculate orientation, deploy recovery parachutes, log data, and execute the control algorithm, we designed and fabricated a custom flight computer (Figure 5). By collecting and analyzing sensor data, the flight computer was designed to control all aspects of our model.

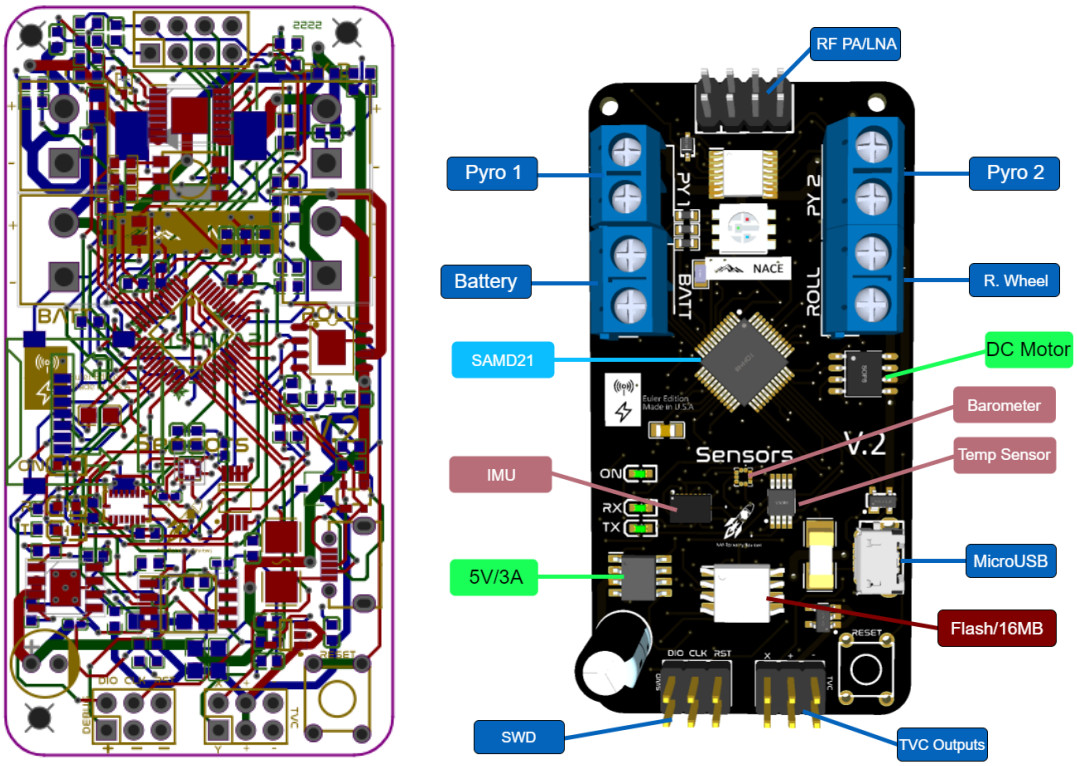

Figure 5. The flight computer in the design phase

Flight Software Design

The flight software was written in C++ and controlled all the components of the flight computer and control system. The flight computer ran a finite-state automation function to control parachute deployment and data logging tasks. To calculate the orientation of the craft using the angular rate measurements from the inertial-measurement-unit, or IMU, the flight computer used quaternions and a linear/1D Kalman Filter to estimate the orientation of the rocket.

Linear Kalman Filter

In our model, a linear Kalman filter is used to filter noise from sensor data [7]. A Kalman filter calculates an estimate of unknown variables within measurements caused by external factors. Since velocity and position are calculated by integrating and double-integrating acceleration respectively, even a small amount of noise will cause significant drift in the measurements.

We define the Kalman Gain, which is

$$G_n = \frac{p_n}{p_n + r_n} \tag{2}$$

Where:

- G_n is the Kalman Gain where $0 \leq G_n \leq 1$
- p_n is the estimated uncertainty
- r_n is the measurement uncertainty

We then calculate the current estimate X_t by

$$X_t = X_{t-1} + G_n \bullet (Z_n - X_{t-1}) \tag{3}$$

Where:

- X_t is the current estimate at time step t
- X_{t-1} is the previous estimate
- Z_n is the measurement or the sensor data in our case

And the estimation uncertainty can be calculated using the Covariance Update Equation

$$p_n = (1 - G_n) \bullet p_n + \left|X_{t-1} - X_t\right| \bullet q \tag{4}$$

Where $q$is the process variance, which increases the accuracy based on how much the measurement moves. We can then set the previous estimate, X_{t-1}, to the current estimate

$$X_{t-1} = X_t \tag{5}$$

Quaternions

The simple integration of the angular rates provided by the IMU to Euler angles results in gimbal lock, where two axes align and differentiation between the two axes is not possible. Quaternions and quaternion algebra, which were invented by Sir William Rowan Hamilton, were used in this project to solve this issue and to allow for 3D rotations. Quaternions represent orientation in four dimensions in 3D space. Since the IMU outputs the angular rate of *x, y,* and *z* in rad/s so we can refer to those as

$$\omega_t = [0, x, y, z] \tag{6}$$

We can also refer to the base world reference frame, and since the rocket points upwards at launch, we can introduce Q_{init}

$$Q_{init} = [1,\ 0,\ 0,\ 0] \tag{7}$$

Next, we can calculate the quaternion derivative that describes the rate of change of orientation relative to the earth.

$$\frac{dQ_t}{dt} = \frac{1}{2} \bullet Q_{init} \otimes \omega_t \tag{8}$$

Where:

- $\frac{dQ_t}{dt}$is the quaternion derivative at time step t
- $Q_t$is the orientation at time step t
- $\otimes$is the Hamilton Product operator (Appendix A)

We can integrate the quaternion derivative to determine the orientation at t with

$$Q = Q_{init} + \Delta t \bullet \frac{dQ_t}{dt} \tag{9}$$

And $\Delta t$is the time step. We now normalize the quaternion by first calculating the norm, and then dividing the orientation by the norm

$$Q = \frac{Q}{\sqrt{Q_1^{\,2} + Q_2^{\,2} + Q_3^{\,2} + Q_4^{\,2}}} \tag{10}$$

Where Q_1,Q_2,Q_3,$Q_4$are the individual elements of quaternion Q. This quaternion can be converted back to Euler angles without the risk of gimbal lock by equations 11-13.

$$\alpha = tan^{-1}\left(\frac{2 \bullet Q_3 Q_1 - 2 \bullet Q_2 Q_4}{1 - 2 \bullet {Q_3}^2 - 2 \bullet {Q_4}^2}\right) \quad (11)$$

$$\beta = sin^{-1}(2(Q_2 Q_3 + 2 \bullet Q_4 Q_1)) \quad (12)$$

$$\theta = tan^{-1}\left(\frac{2 \bullet Q_2 Q_1 - 2 \bullet Q_3 Q_4}{1 - 2 \bullet {Q_2}^2 - 2 \bullet {Q_4}^2}\right) \quad (13)$$

Where α is the roll of the craft, β is the yaw of the craft, and θ is the pitch of the craft expressed in radians (Figure 6).

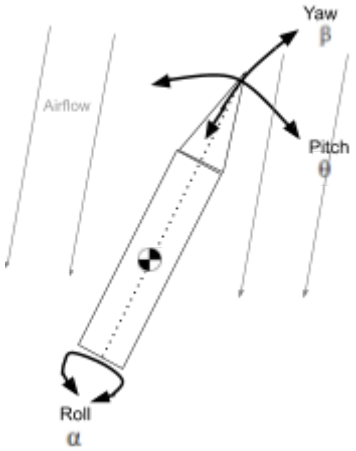

Figure 6. Rocket Rotations

To reduce unnecessary complexity, the real-world model did not include roll control, which is not controllable by the TVC mount. To account for any roll changes during the flight, θ, and β must be decoupled from α using

$$\theta = cos(\alpha) \bullet \theta - sin(\alpha) \bullet \beta \quad (14)$$

$$\beta = cos(\alpha) \bullet \beta + sin(\alpha) \bullet \theta \quad (15)$$

θ andβ can then be inputted into the two PID controllers for both the pitch and yaw axis.

Quaternions are also extremely useful in rotating different orientations. The accelerometer unit on the IMU provides linear acceleration in the vehicle reference frame. This reference frame is not compatible with velocity and position measurements, as well as accelerometer measurements used to detect launch, burnout, and apogee for the finite state

automation function. By finding the Hamilton product of the orientation quaternions and the acceleration readings, we can convert the measurements to a world reference frame. We can refer to the accelerometer readings as

$$a_t = [0, a_x, a_y, a_z] \tag{16}$$

Then, we can find the world reference frame orientation by finding the first Hamilton product of the orientation quaternion and a_t.

$$A_{first} = Q \otimes a_t \tag{17}$$

Next, we find the Hamilton product of A_{first} and $-Q$ to flip the reference frame to the correct orientation

$$A = A_{first} \bullet - Q \tag{18}$$

Where A_2,A_3, and A_4 the 3 of the 4 components of quaternion A, represent the x, y, and z linear accelerations in the world frame respectively.

In conclusion, the flight computer:

1. Reads raw data from sensors.
2. Filters the data using the linear Kalman filter.
3. Converts the raw gyro and time parameters into Euler angles using quaternions.
4. Inputs the Euler angles into the PID controller and commands the output of the PID controller to the servo motors of the TVC mount.
5. Rotates the filtered acceleration readings to the world frame using quaternion operators and uses these values for the state automation function.

Simulation

To tune the PID controller and verify that our real-life system would work properly, we have created a simulation of the rocket in MATLAB Simulink. We have worked to account for many real-life variables, including:

- Noise from sensor data
- Latency from flight computer and servo motor
- Wind disturbances

The most fundamental part of the simulation is the 3DOF block, as it utilizes all of the equations in three degrees of freedom motion to simulate an object in 2-dimensional space. This allows us to input x and z direction vector forces and torque, then receive x and z position and velocity, along with an angular position and velocity. real-life this to work properly, we needed to know the mass moment of inertia (MMOI) and mass of our actual rocket. To find the MMOI, we used the bifilar pendulum method, which involves hanging the rocket by two parallel strings equidistant from the center of mass and measuring the period of the swing rotating around the center of mass. The equation to find the MMOI is:

$$MMOI = \frac{mgp^2r^2}{4\pi^2L} \quad (19)$$

Where $MMOI$ is the mass moment of inertia, m is mass, g is the gravitational constant, p is the period for one complete swing of the rocket, r is the distance from the center of mass and where the string is attached, and L is the length of the strings.

Engine Force and Gravity

The two most critical forces being applied to a rocket are gravity and the engine force. The 3DOF block already has a feature for applying gravity, which made the implementation extremely simple. As for the engine force, the data for the D12-0 engine we are using is already provided by Estes (Appendix C), so we were able to implement that data into the simulation. However, since we are using a TVC mount, we didn't just want the engine force to be applied in the z-direction, as it needed to be vectored based on the TVC mount's position. To solve this, we split the force into x and z vectors using the following two equations:

$$F_{motor(z)} = F_{motor} \times cos(\sigma + e) \quad (20)$$

$$F_{motor(x)} = F_{motor} \times sin(\sigma + e) \quad (21)$$

In these equations, $F_{motor(x)}$ and $F_{motor(z)}$ are the x and z direction vector forces respectively, F_{motor} is the total engine force, and σ is the angle the TVC mount. The motor mount misalignment, represented by e is to 1 degree to account for any real-life misalignments that may occur. We also use this equation

$$T_{motor} = F_{motor(x)} \times a \quad (22)$$

To calculate the torque produced by the TVC mount, where T_{motor} is the torque produced by the engine and a is the distance from the center of mass to the end of the rocket motor. The way the TVC mount position is calculated will be explained later in the simulation overview.

Noise Addition and PID Calculations

Since the accelerometer used on the real rocket does not yield perfect results, we wanted to simulate some of that noise. A random number generator generating values of +/- 0.1 degrees was added to the actual angular position of the rocket in degrees.

To calculate the PID values, we use the PID equation. To calculate the output, this equation must be the same in both the real-world model and the simulation because we want the K_p, K_i, and K_d gains found in our simulation to be used in the real-life model with optimal results.

Real-Life Limitations

Next, we need to account for some of the real-life limitations of the rocket, such as flight computer delay, servo speed limitations, and servo position limitations. The TVC control mount on the real rocket can only move the engine ±5 degrees, so this is accounted for by saturating the output $u(t)$ to have a maximum of 5 degrees and a minimum of 5 degrees. In addition, the TVC servos have a limited angular rate. To measure the speed of the servo, we recorded a slow-motion video of the servo moving as fast as it can and calculated the maximum rotational speed to be 258 degrees per second. We used this value as the maximum rate at which the PID output could change. Finally, flight computer latency needs to be accounted for. Although the flight computer latency is estimated to be only 7.5 ms, we decided to increase it to 50 ms to have the simulation account for more imperfection. After all three of these limitations are applied to $u(t)$, the final resulting angle will be the position of the TVC mount, σ.

Wind Forces

To account for wind, we applied another torque in the form of a sine wave. Since we don't have any data on how much torque is applied for a given velocity of wind, we experimented with the amount of wind torque and set it to be as high as the rocket could handle. After some experimentation, we found that the most wind the rocket could handle was a sine wave going between 0 Nm and 0.1 Nm.

Aerodynamic Forces

The two most important aerodynamic forces that act on the rocket are the drag and lift force [4]. The drag force acts opposite to the direction the rocket is pointing towards, and the lift force acts perpendicular to the rocket but is applied from the center of pressure [4]. We were able to implement these forces with the help of a software called OpenRocket. OpenRocket is a software that allows for the simulation of model rocket launches and lets the user customize the rocket. We used this software and created our real-life rocket, ensuring that parameters such as the mass, center of mass, MMOI, and shape of the rocket were all the same. Then, we exported a large amount of data from OpenRocket for various amounts of wind, including the angle of attack, total velocity, drag force, lift force coefficient, and center of pressure position. The angle of attack is the angle at which the rocket is relative to the air flowing past it as opposed to the ground [10]. All of this data was then implemented into the simulation. However, we only received the lift force coefficient, but not the lift force. The equation to find the lift force is

$$F_l = \frac{1}{2} \times c_l \times d_{air} \times a_{ref} \quad (23)$$

Where F_l is the lift force, c_l is the lift force coefficient, d_{air} is the density of air, and a_{ref} is the reference area of the rocket [6]. The density of air is 1.225 kg/m^3 at sea level, and the reference area is $5.6745 \bullet 10^{-3}$ m^2 according to OpenRocket. To find the torque produced by the lift force, we multiply the lift force by the distance between the center of pressure and center of gravity, which varies based on the angle of attack of the rocket.

Tuning for Optimal Results

After each component of the simulation was completed, the next step was to tune the PID values K_p, K_i, and K_d. To begin tuning, we set K_i to 0 since its effect was not determined to be significant, and began experimenting with K_p and K_d values. We knew that low K_p gains would result in insufficient correction, and high K_p values would result in significant overshooting. K_d gains that were too low would result in overshooting even with low K_p values, and K_d values too high would result in extremely rapid oscillations. When we eventually found a K_p and K_d value combination that resulted in a successful flight, we began fine-tuning these values by moving each of them up and down by small increments individually and seeing if the flight had less or more error. Once we found the K_p and K_d values with the smallest amount of error, we added back the K_i term and increased it gradually until it hindered the results compared to lower values.

Results and Discussion

Simulation

Once tuning was complete, we got PID values of K_p=0.5, K_i=1.4, and K_d=0.08 inside of the simulation. As seen in Figure 7, the rocket coasts up to an altitude of about 18 meters before falling back down to the ground. Figure 8 shows the angle of the rocket as a function of time, which is much more important than the altitude. From 0-1.6 seconds, the TVC mount can handle the wind and prevents the rocket from exceeding an angle of 10 degrees, after this, the rocket motor burns out and is uncontrolled.

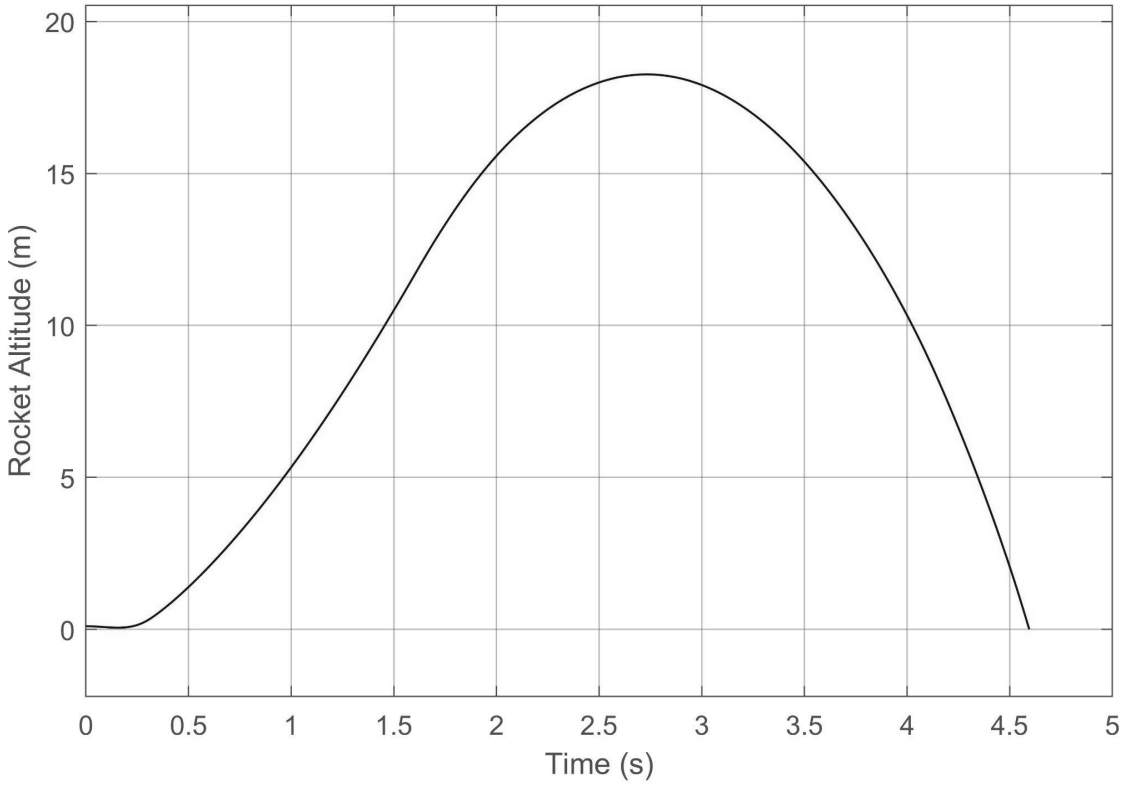

Figure 7. Shows the altitude of the simulated rocket as a function of time.

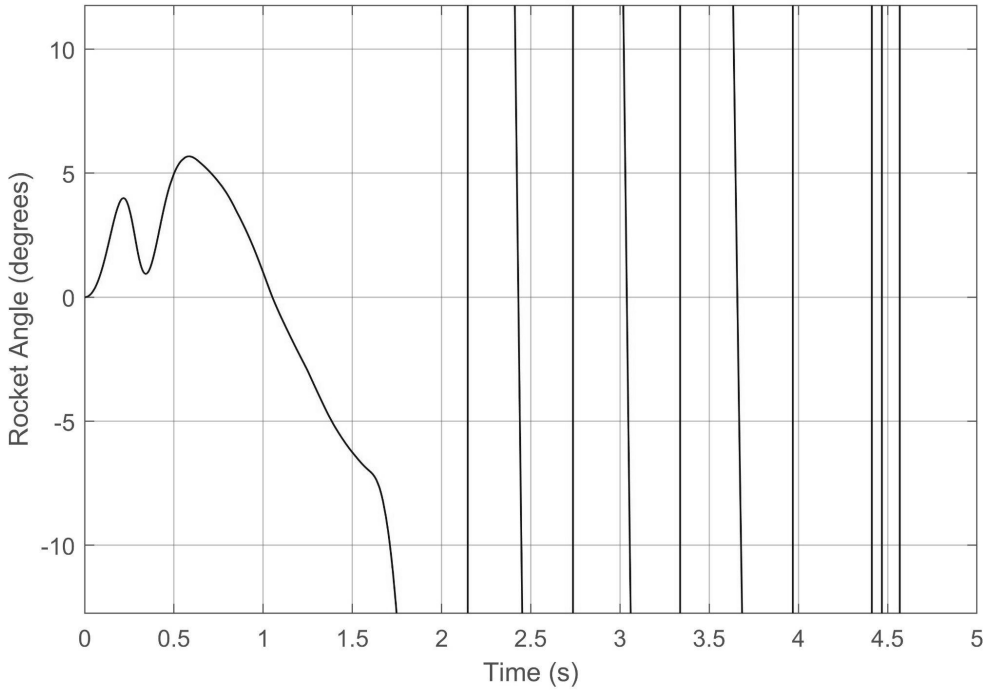

Figure 8. Shows the rocket's angle relative vertical in degrees as a function of time.

Unfortunately, the simulation was less advanced at the time that we flew the real rocket, so the PID values were instead K_p=0.3, K_i=0.3, and K_d=0.05. These PID gains were then inputted into the real-world model's flight software and flown on the Estes D12-0 rocket motor (Appendix C). The rocket had a mass of 0.605kg and a mass-moment-of-inertia of 0.01428 kgm^2. For simplicity purposes, the software had a constant setpoint of zero on both axes, meaning that the rocket would try to stay fully upright.

Real-World Model

The real-world model as discussed on page 5 was flown once using the parameters calculated from the simulation. The rocket was flown on an Estes D12-0 (Appendix C).

Figure 9 shows the estimated orientation of the rocket after liftoff was detected. The rocket almost immediately pitched over after liftoff. At around 0.372 seconds, data logging stopped and the PID loop was terminated because the flight software had an integrated abort system that triggered when the pitch or yaw surpassed +/- 30 degrees. This abort sequence saved all the data into the SD card, detached electrical connections to the servo motors, terminated all functions and deployed the parachutes. This sequence ensured that all relevant data would be saved and that damage to the rocket and avionics would be minimized when the rocket's orientation surpassed a point of no return.

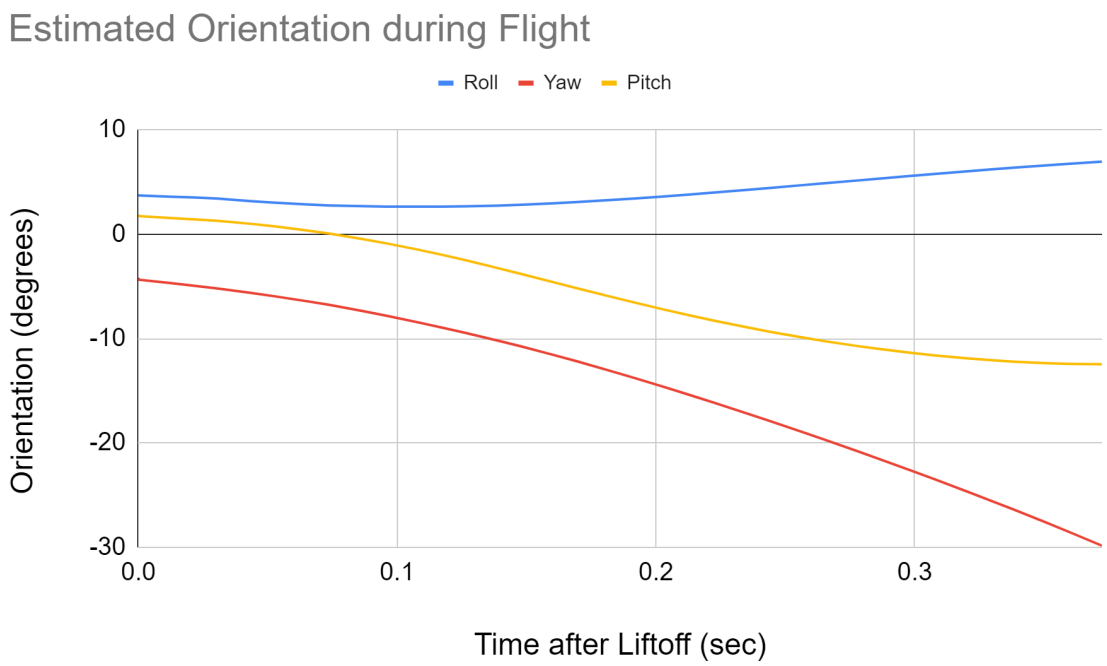

Figure 9. Orientation of the rocket in degrees as a function of time after liftoff was detected by the finite state automation function. Data was cut off at about 0.372 seconds by the abort sequence.

The large undesired orientation change was caused by several issues within the physical model. When assembling the thrust vector control gimbal, there were misalignments within the

two axes. So at liftoff, the rocket immediately pitched over, as seen in Figure 9. As the PID controller tried to correct this error, the misalignment caused larger errors to occur. As can be seen in Appendix B and Figure 10, the PID controller tried to output a value much greater than 5 degrees, and subsequently kept outputting values greater than 5 degrees. The TVC mount had an actuator limit of +/- 5 degrees, which means it was not accurately reflecting the command output of the PID loop.

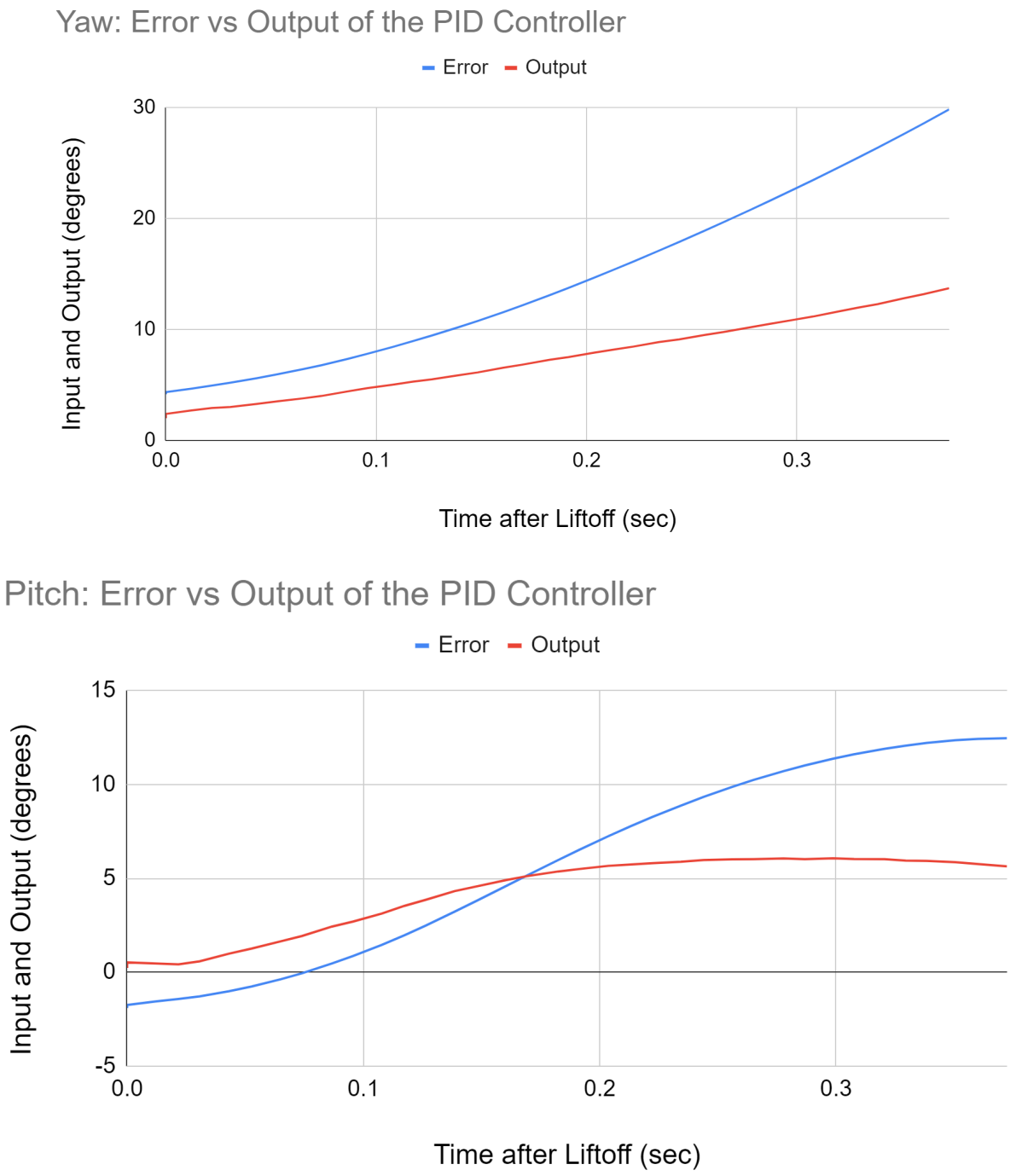

Figure 10. Error inputted to the PID controller versus output as a function of time detected after liftoff.

Figure 11. Different phases of the rocket flight are depicted in one photo. By the second and third freeze frames, the abort sequence had already fired and the rocket was uncontrolled.

After the abort was called, the rocket flew completely out of control and collided with the ground. Although the rocket's outer body frame was shattered and the TVC mount was broken, the flight avionics, data, and servo motors were still intact. Figure 12 depicts other parameters measured during the flight, where X, Y, and Z are the roll, yaw, and pitch axes respectively. All measurements were plotted as a function of time detected after liftoff.

Figure 12 also depicts the raw versus filtered linear acceleration readings in the vehicle reference frame. The raw readings are passed into separate Kalman filters, which smooth the data.

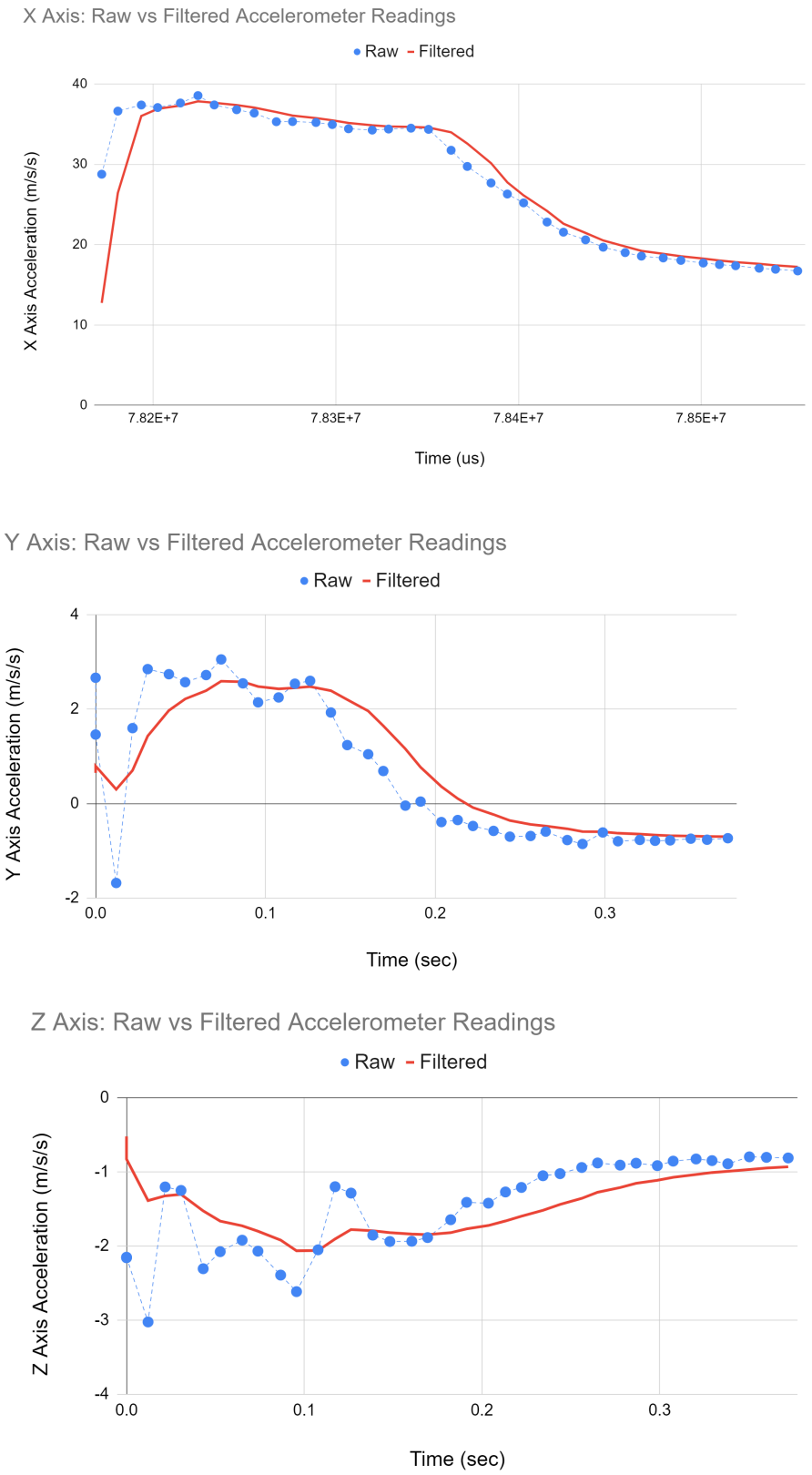

Figure 12. Raw and filtered accelerometer readings. Once again, the process variance in the Z-axis needs to increase since the filtered values are not accurately accounted for the large changes in acceleration.

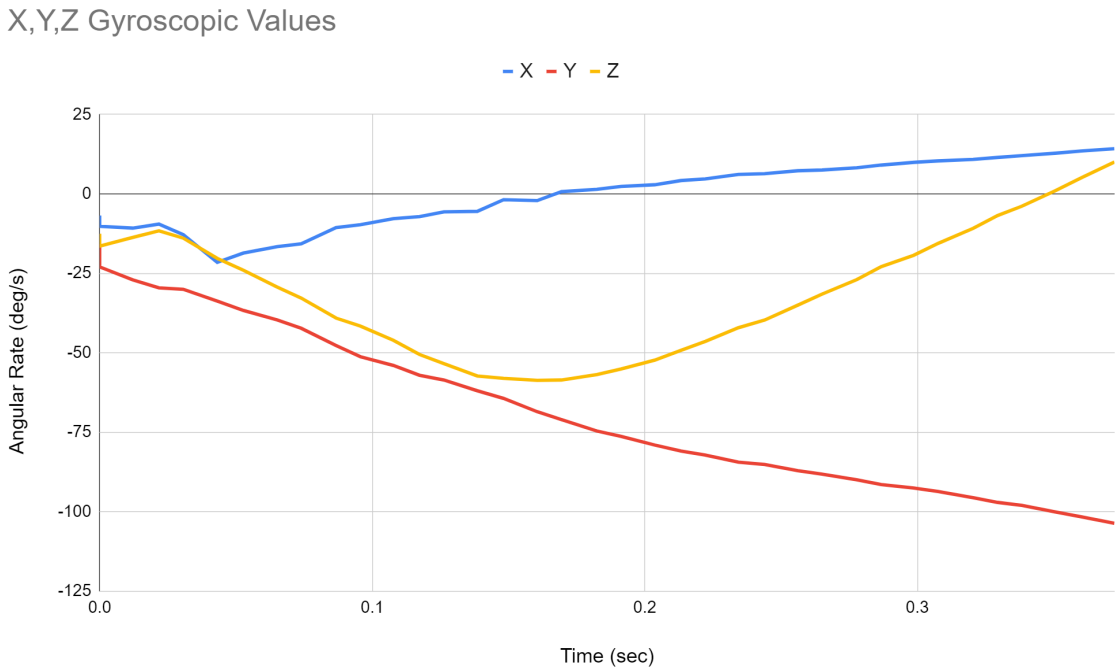

Figure 13. X, Y, Z (roll, yaw, pitch) readings from the gyroscope. These values along with time parameters were converted to Euler angles using quaternions.

Model and Simulation Verification

Although the flight did not take the desirable flight path due to errors in the physical model, we cross-referenced data from the real-world model's flight to the simulation that was run of that model. Figure 15 depicts the simulated flight orientation versus the actual flights in the Y and X axes (yaw and pitch respectively).

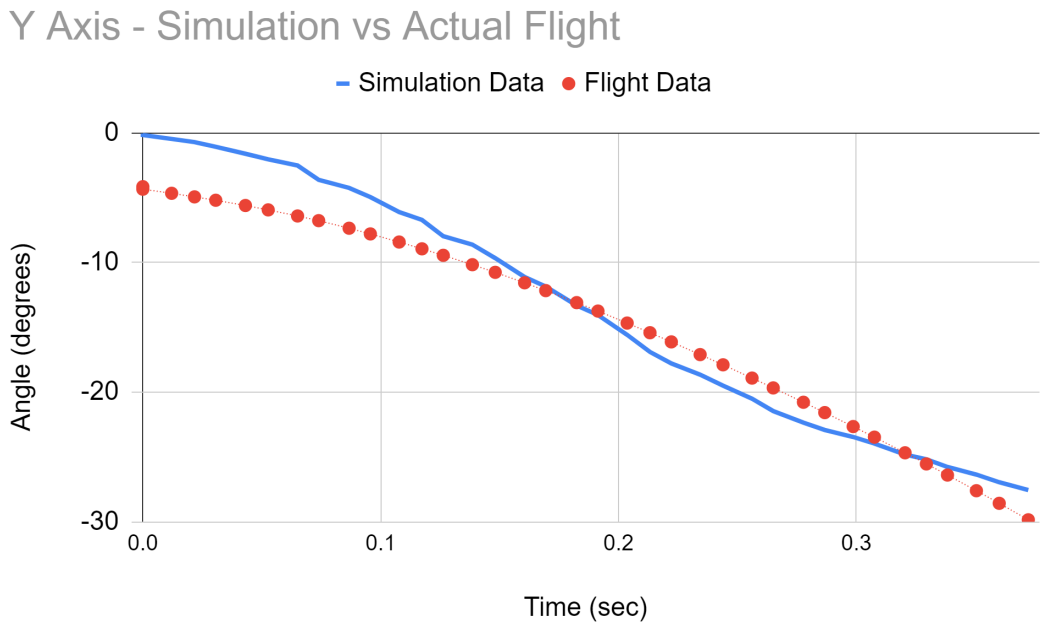

Figure 14. Y-axis (yaw) orientation of the simulated flight of the real-world model and the actual flight.

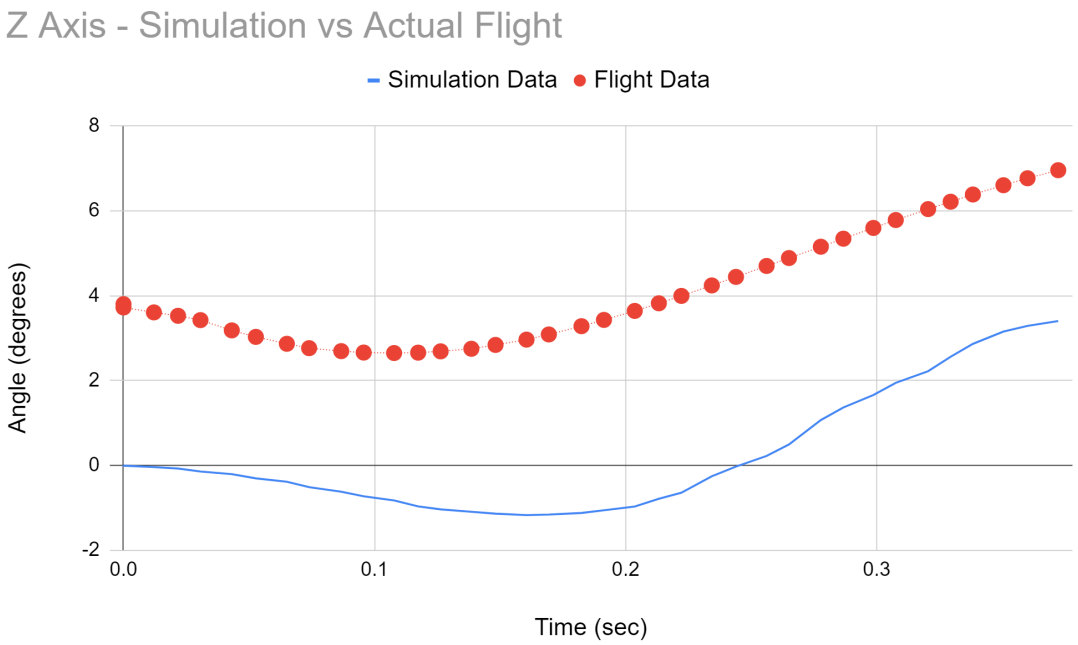

Figure 15. Z-axis (pitch) orientation of the simulation versus the actual flight. The real-world model's orientation values have drifted up about 4 degrees due to the integration of small noise values, causing an offset.

Conclusion

The results of the simulation and model indicate that a PID controller-based navigation system would be a viable option for rockets to use as a control system. Using an accurate simulation that matched real-world data (Figure 15), the three tuning parameters of the PID system could be calculated for an optimal flight. Furthermore, a quaternion-based orientation system was developed to solve gimbal lock and 3D space orientation problems, along with an implementation of a Kalman filter to reduce the effects of noise and increase the robustness of the overall system.

Although the real-world model did not fly as expected, the analysis showed that this was caused by misalignment of the parts and by design errors. We hope to fly another version of the real-world model soon after conducting a few improvements to the model and further improving the simulation for more accuracy.

Simulation Improvements and Future Implementations

The simulation worked very well at predicting the flight path and properties of the real-world model but also has some future implementations and improvements such as:

- Implementing the actual formulas for the fluid dynamic calculations
- Adding a visualization of the flight using animations.

Real-World Model Improvements and Future Implementations

Software

- Faster loop speeds by optimizing the available SRAM onboard the flight computer and creating more efficient functions.
- Add anti-windup to the integral term by clamping the output to the actuator limit of the TVC mount.
- Experiment with different methods of control such as model-predictive control or a linear quadratic regulator.
- Potentially implementing roll control with a reaction wheel, or adding position control.
- A dynamic setpoint that pitches the rocket in a certain direction.

Hardware

- Adding a self-alignment system in the form of a hex driver that inserts into the side.
- Moving avionics upward will lift the center of mass, giving the TVC more torque authority by increasing the lever arm.

Finally, the fact that this project is not the end cannot be overemphasized. The real-world model, control system, simulation, and flight software can have many new improvements, and with more research and development, we hope to see a larger growth within the space industry.

Appendix A

List of symbols, operators, and abbreviations

Symbol	**Description**
TVC	Thrust-vector-control
PID-Controller	Proportional-Integral-Derivative Controller
GNC	Guidance, navigation, and control.
IMU	Inertial-measurement-unit
3DOF	Three degrees-of-freedom
MMOI	Mass moment of inertia
CAD	Computer-Aided Design
PLA	Polylactic Acid
a	Distance from end of engine and center of mass
A	World reference frame linear acceleration
a_t	Linear acceleration
c_l	Coefficient of lift force
a_{ref}	Reference area of the rocket
d_{air}	Density of air
$e(t)$	Error
e	Motor mount misalignment
F_l	Lift force
F_{motor}	The total force from the rocket motor
$F_{motor(x)}$	x-direction vector of the engine force
$F_{motor(z)}$	z-direction vector of the engine force
G_n	Kalman gain
g	Gravitational constant in m/s^2

K_d	Derivative gain
K_p	Proportional gain
K_i	Integral gain
L	Length of string
m	Rocket mass
p	Period of the rocket swing in the two string pendulum
p_n	Estimate uncertainty
Q	Quaternion orientation
Q_{init}	Base world reference frame quaternion
Q_t	Quaternion orientation at time step t
q	Process Variance
r	Distance from center of mass and string
r_n	Measurement uncertainty
$u(t)$	PID controller output
X_t	Kalman filter estimate
Z_n	Measurement input
α	Roll
β	Yaw
θ	Pitch
σ	The angular position of the TVC gimbal
Δt	Change in time
ω_t	Angular rate

$\otimes$ The Hamilton product operator is the noncommutative multiplication of two quaternions. For quaternions $a$and b, $a \otimes b = i$is defined as

$$i_1 = a_1 b_1 - a_2 b_2 - a_3 b_3 - a_4 b_4$$

$$i_2 = a_1 b_2 + a_2 b_1 + a_3 b_4 - a_4 b_3$$

$$i_3 = a_1 b_3 - a_2 b_4 + a_3 b_1 + a_4 b_2$$

$$i_4 = a_1 b_4 + a_2 b_3 - a_3 b_2 + a_4 b_1$$

Appendix B

Flight 1 PID controller outputs

Time (sec)	Yaw Axis Output	Pitch Axis Output
0	2.001	0.232
0	2.386	0.523
0.012112	2.704	0.468
0.021728	2.922	0.419
0.030656	3.009	0.579
0.043136	3.286	0.99
0.05272	3.514	1.26
0.065064	3.785	1.649
0.073936	4.004	1.929
0.086728	4.427	2.434
0.095664	4.713	2.699
0.107768	5.018	3.127
0.11736	5.298	3.536
0.12632	5.497	3.856
0.138552	5.859	4.319
0.148176	6.127	4.579
0.16052	6.557	4.911
0.169408	6.826	5.123
0.182416	7.267	5.367
0.191368	7.514	5.498
0.203632	7.911	5.666
0.213232	8.203	5.742
0.222208	8.457	5.811
0.23432	8.861	5.884
0.243952	9.104	5.976
0.256144	9.509	6.013
0.265056	9.769	6.022
0.277752	10.199	6.059
0.286728	10.496	6.023
0.298672	10.879	6.068
0.3076	11.168	6.033

0.32048	11.658	6.019
0.329456	11.983	5.948
0.33832	12.285	5.929
0.35048	12.799	5.861
0.36008	13.181	5.761
0.372304	13.723	5.642

Appendix C

Estes D12-0 Specifications

Flight 1 used the Estes D12-0 rocket motor.

Length	7cm
Diameter	24mm
Estimated Weight	40.5g
Total Impulse	20.00 N-sec
Estimated Max Lift Weight	32.90 N
Burn Duration	1.60 sec
Initial mass	40.4 g
Propellant Mass	23.8 g

Table C. Technical Specifications of the D12-0 provided by the manufacturer

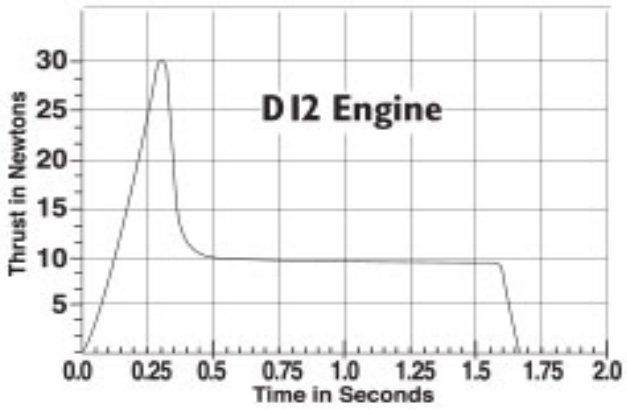

Figure C. Thrust curve of the D12-0 rocket motor provided by the manufacturer

Acknowledgments

We would like to express my gratitude to our mentor Dr. Robert Hermes for his invaluable advice and teachings in model rocketry. We thank our parents for their unending support and our sponsor Ms. Ombelli.

References

[1] Becker, A. Online Kalman Filter Tutorial. https://www.kalmanfilter.net/kalman1d.html (accessed Feb 22, 2022).

[2] Center of pressure. https://www.grc.nasa.gov/www/k-12/airplane/cp.html (accessed Feb 22, 2022).

[3] D12-0 engines. https://estesrockets.com/product/001565-d12-0-engines/ (accessed Feb 22, 2022).

[4] Four forces on a model rocket. https://www.grc.nasa.gov/www/k-12/VirtualAero/BottleRocket/airplane/rktfor.html (accessed Feb 22, 2022).

[5] Koetsier, J. Space inc: 10,000 companies, $4T value ... and 52% American. https://www.forbes.com/sites/johnkoetsier/2021/05/22/space-inc-10000-companies-4t-value--and-52-american/?sh=4861da5955ac (accessed Feb 22, 2022).

[6] The lift equation. https://www.grc.nasa.gov/www/k-12/VirtualAero/BottleRocket/airplane/lifteq.html (accessed Feb 22, 2022).

[7] Looney, M. Anticipating and managing critical noise sources in MEMS gyroscopes. https://www.analog.com/en/technical-articles/critical-noise-sources-mems-gyroscopes.html (accessed Feb 22, 2022).

[8] Maths - conversion quaternion to Euler. https://www.euclideanspace.com/maths/geometry/rotations/conversions/quaternionToEuler/ (accessed Feb 22, 2022).

[9] Maths - quaternions. https://www.euclideanspace.com/maths/algebra/realNormedAlgebra/quaternions/ (accessed Feb 22, 2022).

[10] Niskanen, S. OpenRocket technical documentation. http://openrocket.sourceforge.net/techdoc.pdf (accessed Feb 22, 2022).

[11] Vectored thrust. https://www.grc.nasa.gov/www/k-12/airplane/vecthrst.html (accessed Feb 22, 2022).