

Developing a Guidance and Control system for an F-class Model Rocket.

*Rohan Bandaru, Grade 10,
Lexington High School, Lexington MA 02421.*

ABSTRACT:

Guidance systems are becoming ubiquitous in our daily lives, from self-driving cars, robotic automations to autonomous drones. This project explored the use of quaternion based orientation paired with feedback loops as a guidance control system for an F-class model rocket to hit a target altitude. Several factors had to be optimized in the mechanical, electronic and software design of the system accounting for constraints in size, strength, weight, response times, and accuracy. After several iterations of the design with significant reductions in weight (50% reduction from Version 1 to version 3) and improvement in speed of data processing, a responsive system (~2 millisecond loop time) was developed. Gyroscopic, accelerometer and altimeter data gathered from onground simulation tests and a partially successful launch (achieved 87% of target altitude) showed the guidance mechanism performed within the tolerances (max deviation in trajectory < 20 m over 212 m). Improving the precision of data registration on the processor and enhancing the feedback controls with probabilistic models will likely improve guidance control and performance. All the designs and software developed for this project is shared publicly via github.org for future model rocketry enthusiasts to build upon.

1.0 INTRODUCTION:

Ever since the first black powder rockets were unleashed on the battlefields of ancient China¹, the quest for guidance and control (GNC) systems has been a relentless mission for aerospace engineers. From the primitive analog systems of the V2 ballistic missile², to the attempts at using homing pigeons to guide bombs towards targets³, humans have been pushing the boundaries of guidance and control systems. Billions of airline passengers travel around the world each year thanks to complex GNC systems⁴. Advances in guidance technology have unlocked many new frontiers for humanity, including space exploration, air travel, and most recently, the race for self-driving cars and autonomous drones. Guidance engineers at SpaceX have achieved the remarkable feat of propulsive landing of reusable orbital boosters on a small barge⁵. This, and the recent attempts at landing the Starship⁶ have brought us closer to becoming a space-faring civilization. A deep understanding of the algorithms and principles behind these systems is crucial to the economy and technological landscape of the future. The next generation of scientists and engineers will need to be versatile in skills from mechanical and aeronautical engineering, electrical engineering, data science, and computer science⁷.

Guidance systems involve translating sensor input into actuator movements that correct the course of the vehicle. In the case of rockets, gyroscopes are used to calculate orientation, and accordingly control fins or engines to steer the rocket. The analog systems used in early space flight were mostly mechanical systems⁸ that directly translated gyroscopic readings to either fin actuations or thrust vectoring. As faster digital data processors became possible and digital sensor technology advanced, guidance systems started to use digital computing⁹.



Fig 1. Guidance systems in a rocket.

Model rockets are an excellent application of GNC theory¹⁰. There is significant variation in the peak altitude (apogee) of unguided model rockets. These rockets can be influenced by varying weather conditions such as air pressure, temperature and wind gusts¹³ which greatly affect flight dynamics (Fig. 2). Thus, an active guidance system can reduce the effect of environmental factors and general stochasticity of the rocket. This also makes the rocket much safer and predictable. My goal was to use an active guidance system to consistently steer an F-class model rocket to an apogee of 800ft (240.24m).

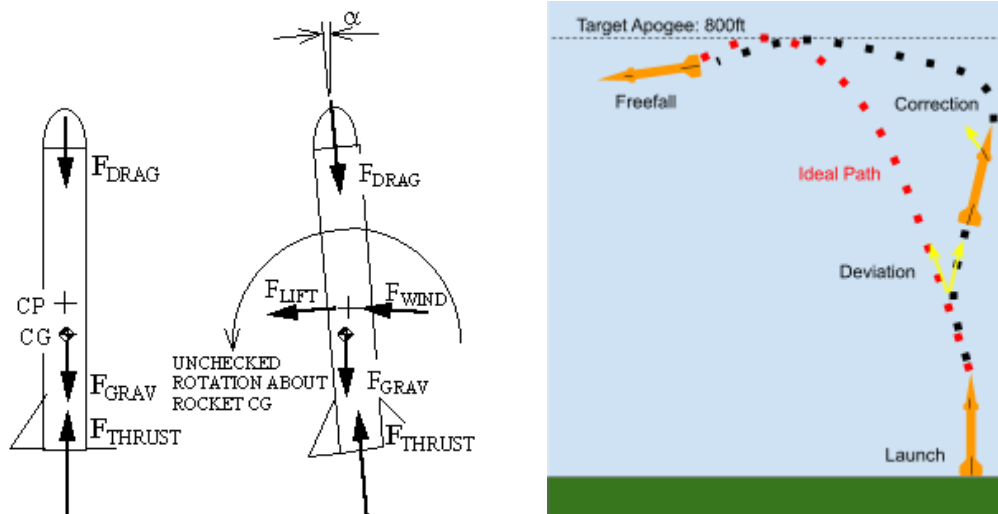


Fig 2a: The effect of aerodynamics on rocket stability.¹⁹ Fig 2b: Corrections to flight path

I implemented the system in a 34-inch, 610g, F-class model rocket as part of The American Rocketry Competition (TARC). Part of this competition was to achieve an apogee as close to 800ft as possible. Normally, teams have approached this by tuning the weight, drag, and thrust of the rocket, but these are inconsistent and prone to environmental variations. To control the trajectory of the rocket, I used movable canards at the top of the rocket to impart a rotational force. I used a 3-axis gyroscope and a 32-bit microcontroller to run the control loops to stabilize the rocket.

2.0 APPROACH AND MATERIALS:

The system works by actively steering the rocket on ascent, such that as it approaches 800ft in altitude, the rocket's trajectory levels off. The guidance system has three major components; mechanical, electrical, and software. All three needed to work together and be optimized to fit certain constraints.

2.1 Constraints and Assumptions:

2.1.1 Response time:

F-class model rockets have impulse rates of up to 80 newton seconds (Ns) and reach speeds upwards of 155mph¹¹. They require extremely responsive and robust control systems, as within fractions of a second, the rocket can completely change orientation, rendering the current output of the guidance system useless. Thus, the response time of the system has to be minimized, to about below 5 milliseconds.

2.1.2 Weight and Structural integrity:

The guidance hardware must be able to handle the hard acceleration and stress exerted on the rocket at these high speeds¹². At 155mph, a canard with a surface area of 20cm² at 45 degrees deflection will experience a torque of about 0.1 newton meters. Thus, the mechanism moving the canard must be able to withstand this force. However, the entire rocket weight was constrained to no more than 650g, meaning that the guidance system by itself cannot exceed 200g. This presents the challenge of making a strong and reliable mechanism, while minimizing weight.

Additionally the positioning and distribution of the guidance mechanism in the body of the rocket needed to be optimized such that the center of mass of the rocket was closer to the nose¹³ of the rocket.

2.1.3 Size:

The entire guidance system must fit inside a 6.5cm diameter tube. It must also not take up more than 20 cm of vertical space. Thus the parts have to be compact and designed to minimize size. The size of the canard fins needed to be calculated based on simulations to ensure they would be the right size to be able to influence the trajectory at the velocities¹³. Too large a canard would increase drag and destabilize the rocket.

2.2 MECHANICAL DESIGN:

The two main configurations of a fin guided rocket are a single set of articulating rear fins (Rear Fin Control), and a two-fin configuration¹⁴, with static rear fins and articulating front canards (Canard Control) as shown in Fig. 3.

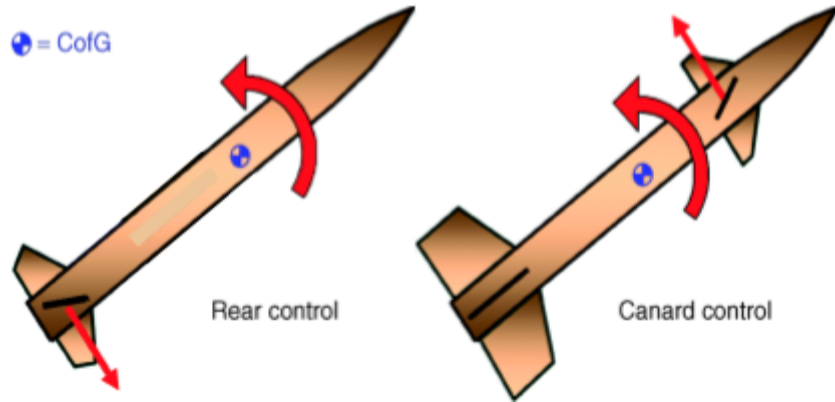


Fig 3: The two main possible configurations of fins for attitude control.

The Canard Control design has a number of advantages¹⁰. Firstly, it results in a more modular design that can be easily removed and serviced. Also, the rear fins are placed around the rocket engine, which means attempting to articulate them would expose the guidance to high temperatures, as well as the violent ejection explosion that deploys the parachute. The overall canard design is shown in Fig. 4. It consists of a small module that can be placed into the top of the rocket.

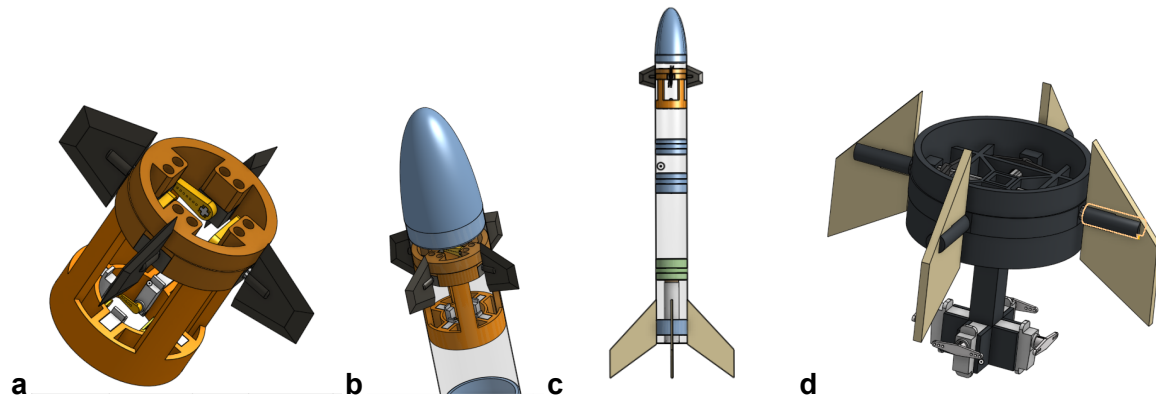


Fig 4a: Version 2 of Canard mount module design. 4b: Design close-up of the canard mount in the rocket. 4c: Overall design of the rocket, with the guidance system in the nose. 4d: Version 3 of design reduced the weight by over 50% by moving the servos to a single center shaft.

I created all of the 3D models using Onshape, a 3D modeling CAD(Computer aided design) software. It allowed me to create complex parts with high precision so they could later be manufactured.

The guidance system needs actuators to move the canards. To choose actuators, I looked at a variety of factors. Most microcontrollers operate on 3.3V logic, so the actuator must provide enough torque (0.1 newton meters or 14.4 oz inches) to articulate the canards at 3.3V. In addition, the actuators should be small, light, and very responsive. This rules out heavy brushed dc motors or large gearboxes. RC model aircraft servos are perfect for this application, which is why I chose to use HK-5320 servos¹⁵ Fig 5. They are extremely small, lightweight, and were purposely designed to actuate aerodynamic control surfaces.

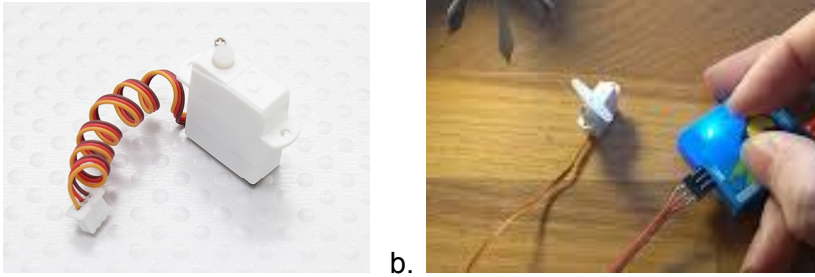


Fig. 5a: HK-5320 servos, 5b: Testing the torque.

Using the following formula to estimate torque needed for the canards helped select the appropriate servo.

$$\text{Torque} = c^2 * v^2 * l * \sin(f) * \tan(f) / \tan(s) * 1.9 * 10^{-5}$$

Where:

c is the average cord of the fin (cm)

l is the length of the fin (cm)

v is velocity (m/s)

f is the fin deflection (rads)

s is the servo deflection (on the linkage mechanism) (rads)

Torque vs Fin Cord and Length

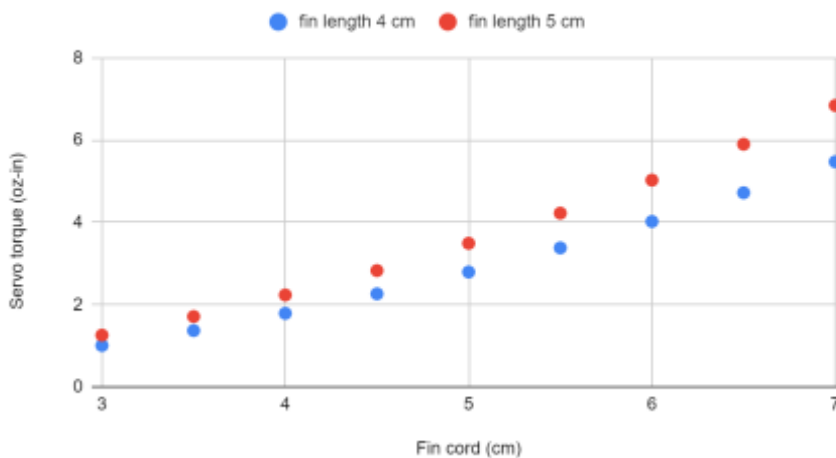


Fig. 6: Plot of the torque that is required of the servo based on varying fin cord and fin length dimensions.

However, these servos alone do not have enough torque to turn the canards. Thus, I used linkage arms to increase the torque output of the mechanism. Torque can be increased by utilizing mechanical advantage. By decreasing the ratio of the output shaft rotation to the input shaft rotation, a linkage can do just that¹⁵. This mechanism also has the advantage of protecting the servo in the event of a crash. I arranged the 4 linkage mechanisms vertically in a cross pattern to make the mechanism compact yet easy to assemble and service. Linkages are similar

to gears, in that they provide mechanical advantage, however they can be much lighter due to the amount of extra mass of solid gear wheels.

One additional difference is that the rotation of a linkage mechanism is not linear. The ratio between the input shaft and output shaft isn't constant. This can be mitigated by increasing the length of the linkage arm (Fig. 7):

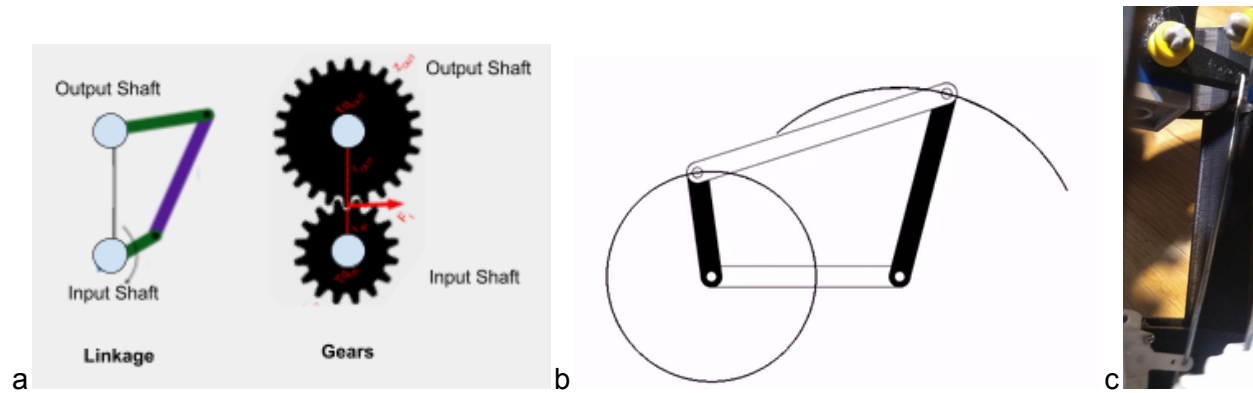


Fig. 7a: Linkage mechanism compared to gears. 7b: Visualizing the motion of a linkage mechanism. 7c: The actual linkage mechanism as constructed.

Another method in which I compensated for the torque deficiency was by using 4 canards rather than 3. Although complete rotational holonomy can be achieved with only 3 canards, using 4 allowed me to reduce the size of the canards without sacrificing control authority. I stopped at 4 rather than adding more, as the weight of a canard and servo is roughly 10g, and I had already reached an adequate torque.

Because the canards are placed at the top of the rocket, they should not cause turbulence or create unwanted airflow separation that will harm the performance of the rear fins. Thus, I used a trapezoidal design with tapered edges to ensure that the airflow around the canard is smooth¹⁸.

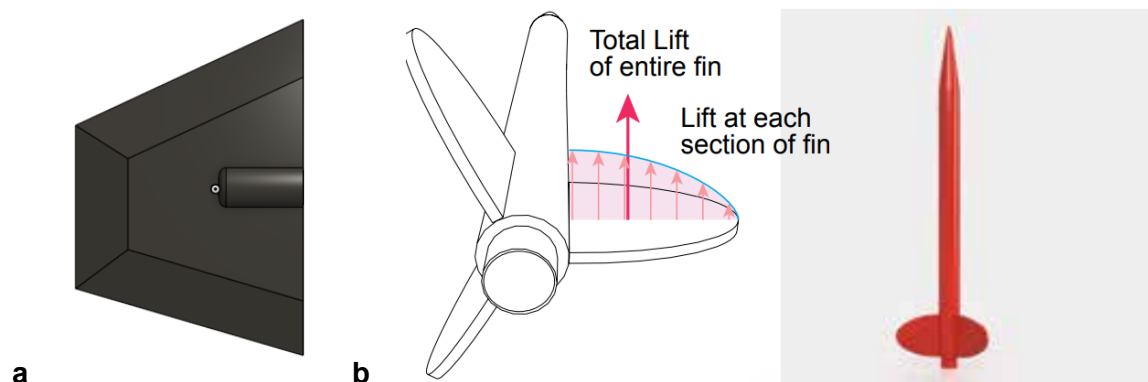


Fig. 8a: Design of the canards. 8b: The ideal rocket fin shape to minimize turbulence and drag¹⁸

The entire mechanism is 3D printed using PLA plastic. 3D printing is an additive manufacturing process that allows for the creation of complex custom parts without the need for expensive machinery. I printed the guidance mechanism on a Creality Ender 3 V2 FDM printer, using CURA to prepare the gcode files.

2.2.1 Housing of the guidance system:

The guidance mechanism was positioned as far towards the nose of the rocket as possible. This increases the distance of the mechanism from the center of mass, increasing its ability to steer the rocket. The main mechanical parts with the canard fins and servos were designed to fit neatly into the body tube on top of the rocket, just below the nose cone.

The Nose cone housed the electronics and battery which allowed me to move the center of mass of the rocket towards the nose, increasing stability²⁰.

The Nose cone was molded out of expanding marine urethane foam with a central chamber to house the electronics and battery Fig. 9. This design allowed me to easily access the electronics for data downloads from the microcontroller post flight.

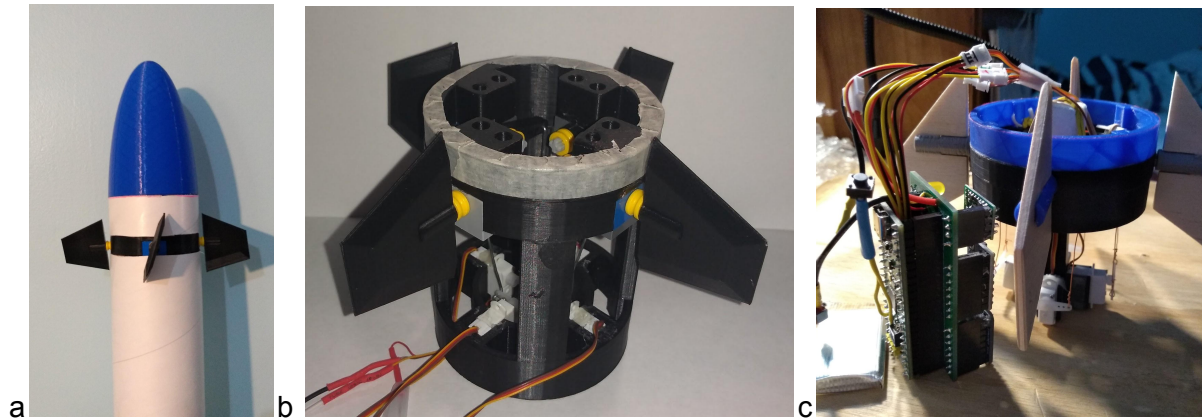


Fig. 9: Fully assembled guidance controls along with the electronics and power supply, b. Version 1 and c. is the fully assembled version 3 connected to the controller unit.

2.2.2 Tools used for structural design:

1. Onshape - Software for 3D modeling (CAD)
2. CURA - Software for 3D printer
3. Creality Ender 3 V2 - 3D printer
4. Openrocket - Simulation software for model rockets
5. General machining and fabrication tools.
6. Weighing scale with +/- 5 gm limit of sensitivity.

2.3 ELECTRICAL DESIGN:

The microcontroller is the core of the flight computer. It does all the computations and input/output in the guidance system. As mentioned, the system needs to be very responsive, at least a 200hz update rate²¹. The STM32F401 microcontroller, commonly known as the BlackPill, was used for its high clock speed (84Mhz), as well as large amount of RAM (256kb). I flashed it with the Arduino core, as it's a versatile ecosystem with many built in libraries and support¹⁹. For

orientation measurements, I used an MPU6050 3 axis gyroscope and accelerometer inertial measurement unit (IMU). I chose this because it has an extremely fast polling rate ($>400\text{kHz}$) and high dynamic range ($\pm 200\text{ deg/s}$). In addition, a BMP388 altimeter was included for its high accuracy and polling rate²⁰.

The entire system runs on 3.3v as that is the logic level for all the components. The IMU and altimeter both communicate with the microcontroller over a single I2C bus. The I2C bus is a common communication protocol used between embedded electronic devices²³. It works over two open drain lines - one for clock (SCL) and one for data (SDA) (Fig. 10). The clock line pulses at a constant frequency and allows the two devices to coordinate transmissions over the data line. To help the clock and data lines pulse more accurately, I attached two 4.7k ohm pull-up resistors²⁴.

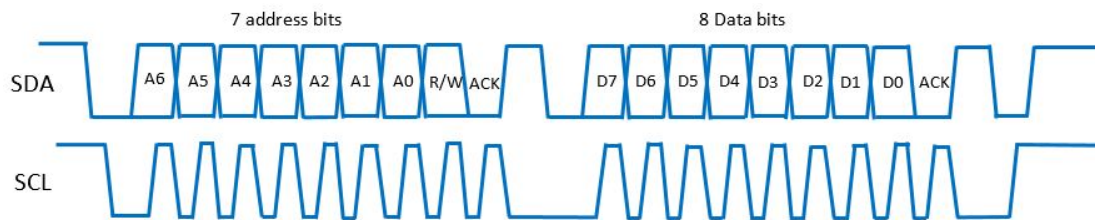


Fig. 10 I2C communication protocol

In addition to the clock and data lines, all devices need power and ground connections making for a total of four connections from each sensor to the microcontroller. In addition to the sensors, the microcontroller also needs to connect to the actuators which are the four servos (Fig. 11). Each signal wire of the servos connects to an analog pin on the microcontroller that is PWM (Pulse Width Modulation) capable. For debugging, I included an indicator LED, as well as a pushbutton (Fig. 11). The LED flashes in different patterns depending on the state of the flight computer. The pushbutton is used to reset the microcontroller, as well as for data reporting functions.

The entire system needs to be portable and battery powered. Also, the battery should be light and able to supply enough current to all four servos and the microcontroller. A single cell LiPo (lithium polymer) battery is perfect for this application, due to being lightweight (15g) and energy dense (3.7v at 600mAh). I used a buck-switching voltage regulator to regulate the voltage down to 3.3v.

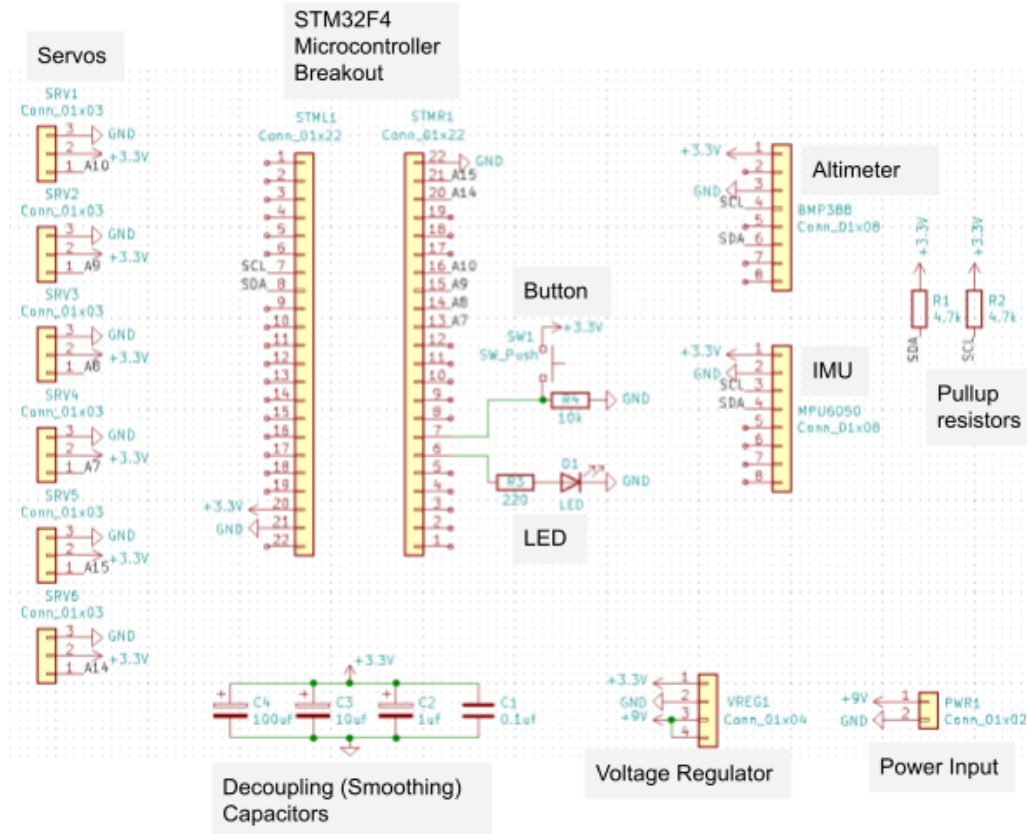


Fig. 11: Schematic of the flight computer circuit in KiCAD.

With so many connections, using wires would be heavy and disorganized, so I decided to connect the components via a PCB (Fig. 12). This made the electronics very compact and robust. I went through two design iterations of the PCB, each time trying to make it lighter, more compact and feature rich. The PCB design process involves placing components and then routing traces according to the schematic²⁶. I designed the PCBs using the Computer Aided Design (CAD) software KiCAD and it was manufactured by JLCPCB Inc. China.

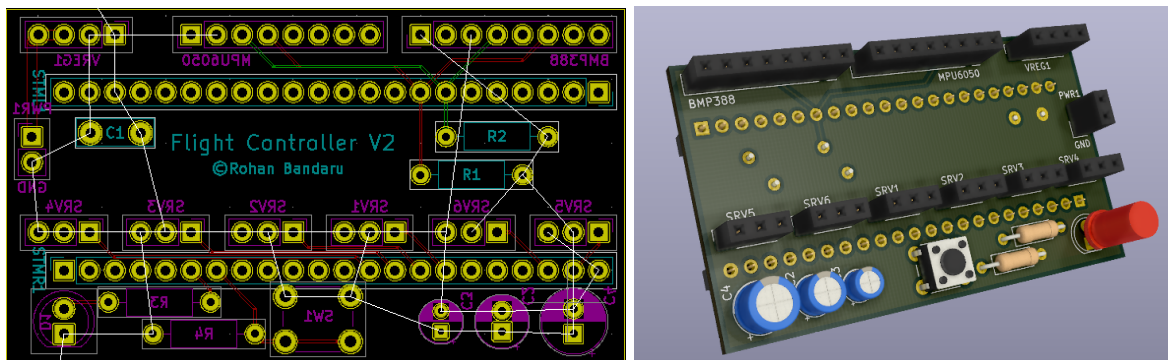


Fig. 12: Rendering of the PCB in KiCAD.

2.3.1 Tools used:

1. KiCAD - PCB CAD software
2. Soldering Iron, heat guns, helping hands, wire cutters, and other tools for electronics assembly.

2.4 SOFTWARE DESIGN:

The flow control diagram of the flight controller is shown in Fig. 13. Sensor inputs are on the left and actuator outputs are on the right.

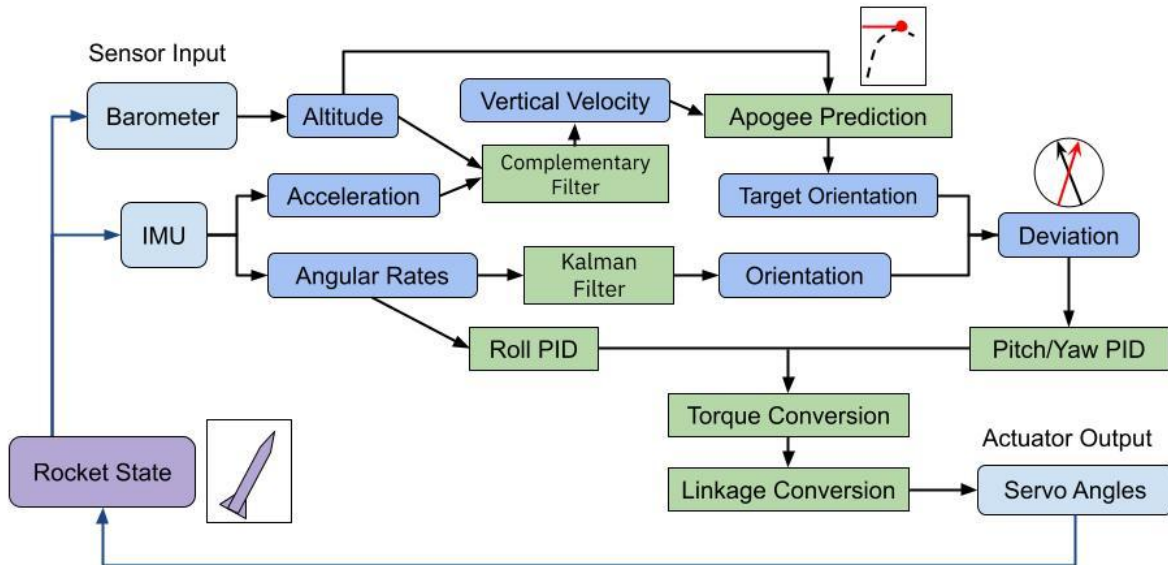


Fig. 13: Flow control outline of the flight computer.

The IMU outputs the rate of angular rotation on three axes: X, Y and Z, or Pitch, Roll and Yaw respectively (Fig. 14). The values are sampled at 500hz, and ran through a Kalman filter²⁸ to eliminate gaussian noise. I implemented this using an Arduino library by Kristian Lauszus, with some modifications.

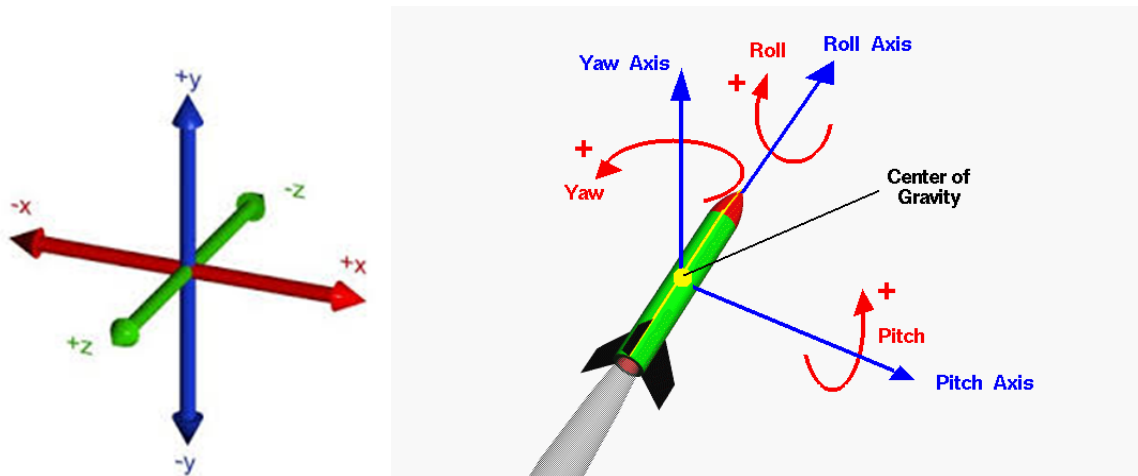


Fig. 14: Angular velocities in the 3 axes measured by the gyroscopic sensor.

Since these Angular velocities are in a local reference frame (the axes move with the rocket)²⁹, they need to be converted to a global reference frame where the axes are stationary. So, the filtered data is used to update a quaternion in the following manner:

Quaternions are 4 dimensional numbers that represent orientation^{30,31,32}. Just as vectors can be used to represent directional forces, Quaternions quantify orientation, allowing you to add, multiply and divide them. For example, the product of two quaternions, A and B, results in a quaternion that represents A rotated by B.

The delta-quaternion is defined as:

$$Q = a + bi + cj + dk$$

where i, j, k are imaginary

And a, b, c, d are:

$$a = \cos(gX / 2) * \cos(gY / 2) * \cos(gZ / 2) + \sin(gX / 2) * \sin(gY / 2) * \sin(gZ / 2)$$

$$b = \sin(gX / 2) * \cos(gY / 2) * \cos(gZ / 2) - \cos(gX / 2) * \sin(gY / 2) * \sin(gZ / 2)$$

$$c = \cos(gX / 2) * \sin(gY / 2) * \cos(gZ / 2) + \sin(gX / 2) * \cos(gY / 2) * \sin(gZ / 2)$$

$$d = \cos(gX / 2) * \cos(gY / 2) * \sin(gZ / 2) - \sin(gX / 2) * \sin(gY / 2) * \cos(gZ / 2)$$

Where gX, gY, gZ are the gyro Angular velocities in degrees/second in the three axes respectively.

This quaternion Q represents the change in orientation as measured by the gyro rates. So, it is integrated to provide the orientation quaternion P .

P is initialized as $(1, 0, 0, 0)$ and is updated with $P_i = Q_i * P_{i-1}$

Since Quaternions are 4-dimensional, which makes them hard to depict visually. Fig. 15, shows one possible representation of the Quaternion framework.

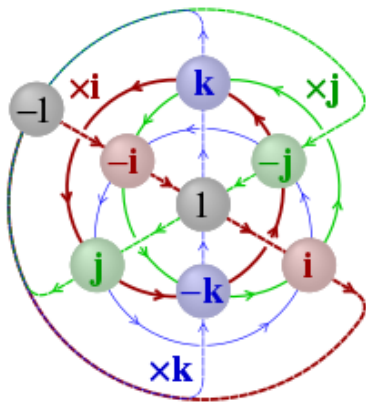


Fig. 15: Visual representation of Quaternions with the imaginary components (i,j,k) and their respective axis.

The orientation quaternion is then used to find the deviation quaternion, which represents the deviation of the rocket from it's target path. This is done by dividing the current orientation by the target orientation.

The deviation quaternion $D = a, b, c, d$, is converted back into angles using the following formula:

$$Z_d = \text{atan}(2 * (b * c + a * d) / (a^2 + b^2 - c^2 - d^2))$$

$$X_d = \sin(2 * (b * d - a * c))$$

$$Y_d = \text{atan}(2 * (a * b + c * d) / (a^2 - b^2 - c^2 + d^2))$$

These angles are back in the local reference frame of the rocket, meaning they can be transferred into the canard movements.

To calculate the canard movements, PID controllers are used. A PID is a control loop with 3 components, the Proportional component, Integral component, and Differential component^{33,34,35} (Fig. 16).

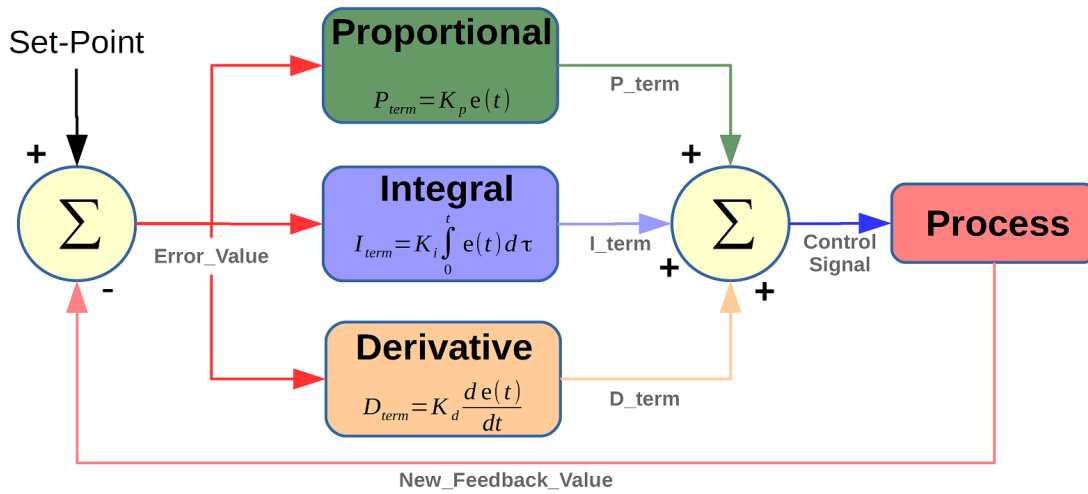


Fig. 16: Components of a PID controller decision flow.

The set-point(target) for each controller is 0, as the rocket wants to have 0 deviation from its target orientation. Thus, the error for the Pitch(X), Roll(Y), and Yaw(Z) axis ends up being $-X_d$, $-Y_d$, and $-Z_d$ respectively.

The goal of a PID controller is to account for systems that have momentum. For example, when a force is imparted on the rocket, due to Newton's first law, the rocket will keep turning until acted on by an external force. In reality, aerodynamics will eventually slow the rocket down, however factors like momentum and dampening still need to be accounted for. This is the purpose of a PID, and why complex control systems exist.

The proportional component, P, is proportional to the error, so:

$$P = \text{error} * p_gain$$

The integral component, I, is equal to the sum of the error, so:

$$I = \text{sum of error} * i_gain$$

The differential component, D, is equal to the rate of change, or derivative of the error, so:

$$D = (\text{error} - \text{previous error}) * d_gain$$

The final output of the PID controller is the sum of all the components, so:

$$\text{Correction} = P + I + D$$

The p_gain, i_gain, and d_gain are constants that influence the behavior of the PID control loop. For example, a high p_gain will increase the effects of the proportional term, making the control loop more responsive yet prone to oscillation. Increasing the i_gain increases the effects of the integral term, making the control loop more responsive to accumulated error and improving accuracy. Increasing the d_gain dampens the system, making corrections slower but reducing oscillation (Fig. 17). These values have to be tuned to result in the most effective PID controller.

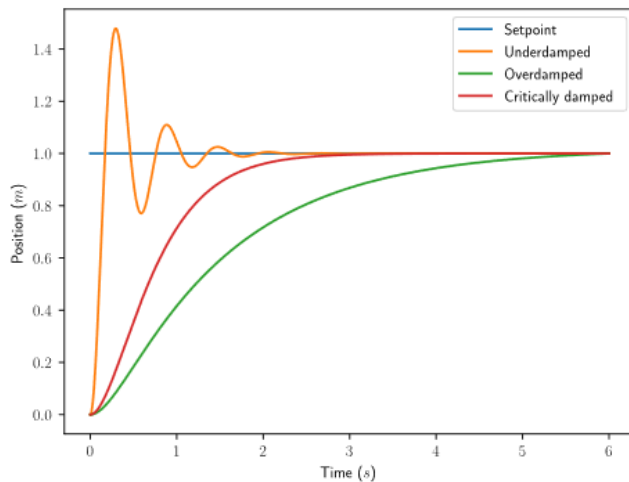


Fig. 17: The effect of damping on a PID feedback system.

The PID corrections are then combined in the following manner and sent to the servos:

```
Servo_Pitch_1 = forceToServoAngle(midpoint + Pitch_correction + Roll_correction / 2);
Servo_Pitch_2 = forceToServoAngle(midpoint - Pitch_correction + Roll_correction / 2);
Servo_Yaw_1 = forceToServoAngle(midpoint + Yaw_correction + Roll_correction / 2);
Servo_Yaw_2 = forceToServoAngle( midpoint - Yaw_correction + Roll_correction / 2);
```

Where Servo_Pitch_1, Servo_Pitch_2, Servo_Yaw_1, Servo_Yaw_2, are positioned with their corresponding canards as shown in figure 18.

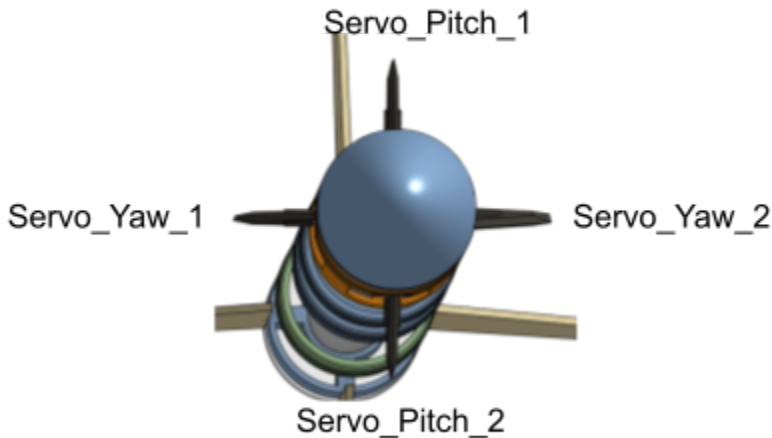


Fig. 18: The labeling of the servos to their corresponding canards

midpoint is a predetermined constant that positions the canard to the neutral position.

Roll_correction is divided by 2 because it affects all 4 fins, meaning that it's effect would be double that of the *Yaw_correction* and *Pitch_correction*.

Data Collection

Capturing data is critical to improving this system. To avoid any external storage devices such as SD card readers and flash chips, I decided to store the data in the RAM of the microcontroller. The main control loop still needs RAM to run and store variables so I partitioned the RAM into equal halves, with 32kb for data storage and 32kb for the main control loop. I can store 2000 points of 3x 16-bit gyro readings, 3x 16-bit accelerometer readings, and 1 32-bit reading for altitude. I decided to record the data at 100Hz, as I didn't want to perform too many interrupts to the main control loop.

The rocket sits on the pad for a long time before flying, so it will have to automatically trigger data collection. Data collection is triggered when the accelerometer in the IMU detects a large upwards acceleration which would indicate engine ignition.

Then once the rocket is recovered, I can plug it into my laptop and press the button on the PCB. It then starts streaming all of the data it captured to my laptop over USB serial. I then use a python script to interpret this and write it to a file.

2.4.1 Tools Used:

Arduino IDE - Development environment for programming the microcontroller

1. STM32 Cubeprogrammer - Software tool for flashing the bootloader
2. Wire - Arduino library for I2C communication
3. Servo - Arduino library for servos
4. [BMP388_Dev](#) - Arduino library for interfacing with altimeter
5. [KalmanFilter](#) - Arduino library for kalman filters
6. [Quaternion](#) - Arduino library for quaternion
7. Pycharm/Python - Python IDE
8. OpenGL - Package for rendering graphics

9. Pygame - Python package for graphics
10. Numpy - Python package for matrix operations
11. PySerial - Python package for reading serial bus
12. For a full list of software and code, please visit my github: <https://github.com/rohan2017>

2.5 Cost and Effort

All the electronics, 3D printed parts and tools did not cost more than \$500 in total. A full list of parts and the costs and the source is detailed in the appendix as well as my engineering notebook.

This project was started in Oct. 2020 as part of a larger team effort to build a guided model rocket and the first functional system was tested in Jan. 2021. The guidance control system alone took approximately about 150 hours of effort from start to finish. My team collaborated closely on designing and building the remainder of the rocket. A detailed project plan is available in my engineering notebook.

3.0 RESULTS:

The design of this system went through multiple evolutions. A lot of the changes centered around weight reduction, improved tolerances, and feature additions. One of the critical changes I did was to reduce the total weight of the system by reducing the thickness of many parts.

3.1 Unit Testing:

Prior to assembly all mechanical components were put through a series of rigorous stress tests.

3.1.1 Canard Axle:

The tolerances of a 3D-printed axle for the canards were not tight enough for smooth rotation and strength. These were replaced with lego axles, which have much better tolerances of only 0.1mm.

3.1.2 Linkages:

The original linkage design used paper clips, which easily flexed and caused the fins to wobble during testing. I replaced them with linkage rods that were more rigid and made the canards more responsive.

3.1.3 Nose Cone:

The first attempt at 3D printing a nose cone was not successful as the PLA material was too dense and heavy. In simulated drops of the rockets with the parachute deployed I realized it was not necessary for the nose cone to be as strong. In the final version of the nose cone design I used expanding urethane foam. This significantly reduced the weight of the nose cone from 40g to under 10g.

3.1.4 Servo tests:

I tested the response of the servos to the microcontroller outputs by using a test program I wrote in python. Slow motion video capture of the servos was used to calculate the response times. The performance of the servos was within limits of tolerance with the goal of test to track the time intervals to fin extreme positions. To test the shear force on the fins due to wind resistance

I built a small rig attached to a scale to test the maximum torque force that the servos translated on to the fins shown in Fig.19.

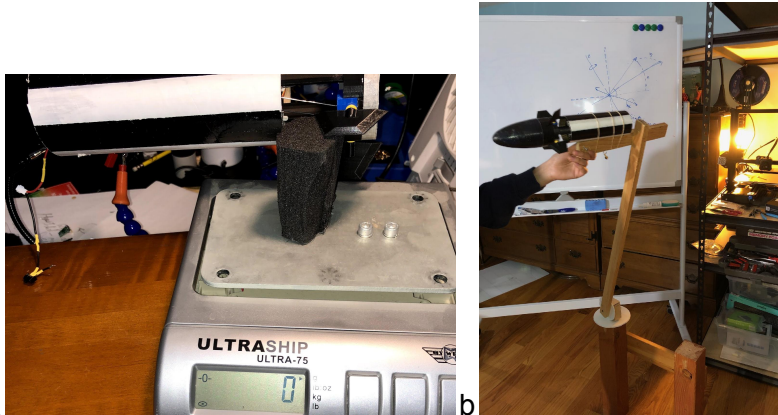


Fig 19a: Estimating the torque force of the servos using a precision weighing scale. Trying different gear ratios/ linkage length to test the configuration that allowed for most drag force on the fins. 19b. 3 Axis Gimbal to test the gyro sensor orientation calculations.

3.1.5 Gyroscope test:

Gyroscope sensor readings were collected and visually compared to the system orientation. A 3 axis gimbal was constructed on which I mounted the guidance control mechanism and collected data at various orientations and compared with the sensor values (Fig. 20).

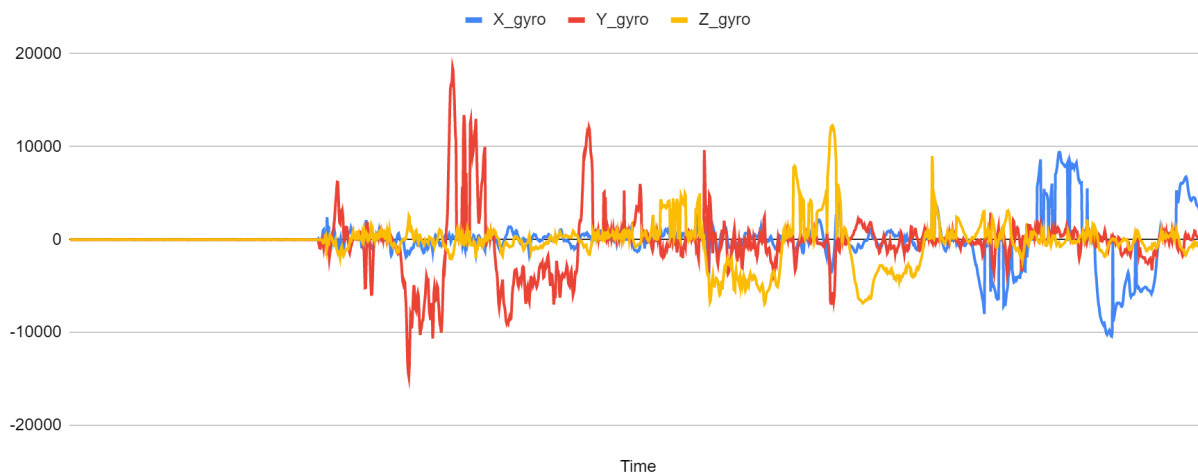


Fig. 20: Gyro angular velocities in the 3 axes from simulated orientation tests.

3.2 Electronics Configuration Redesign:

I went through a couple of iterations of the PCB design. The first design was developed to be mounted vertically and did not include the altimeter on the same board. In the second version, I added a bus for the altimeter to be included and designed it for being able to mount in either configuration.

3.3 Integration testing:

The fully assembled guidance system was put through various levels of tests and has proved to be functional within the limits of tolerance.

Since I didn't have access to a wind-tunnel, I rode in a car with the guidance mechanism held out on the side of the car and tested the performance at speeds up to 60 mph. In the actual flight the system will have to work at much greater velocities up to 160 mph. Data was stored on board for a duration of 10 seconds and processed. Real time orientation (integrated values) are shown in Fig. 21.

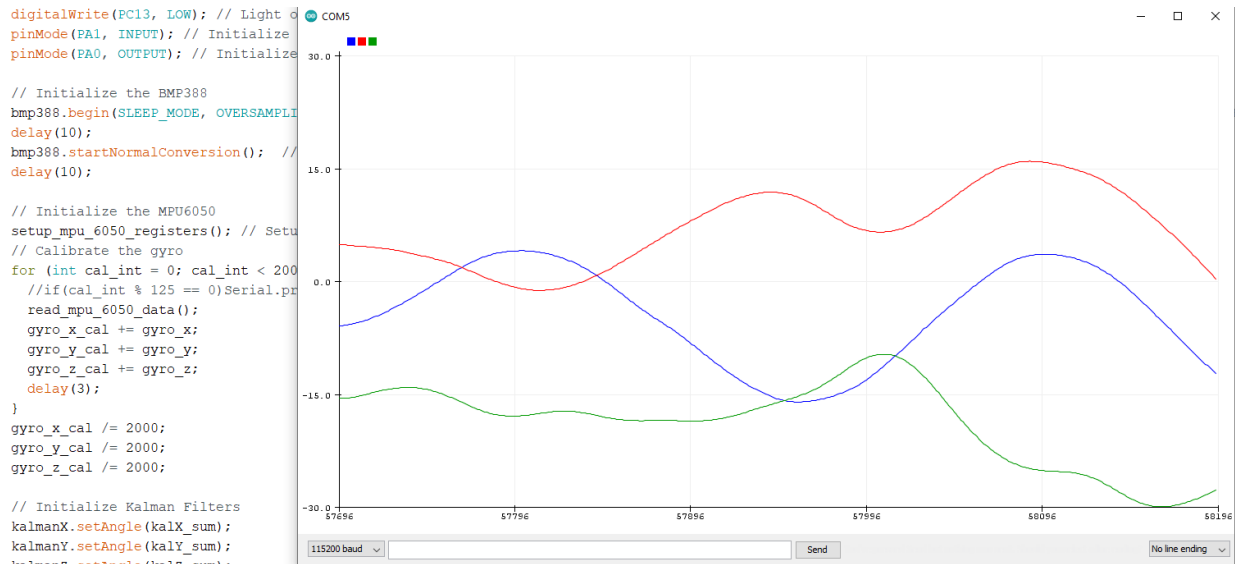


Fig 21: The plot shows the orientation data for pitch, roll and yaw as the guidance system housed in the aft portion of the rocket was held against the wind at 60 mph.

Firstly, to test the orientation measurements, I created a python script^{36,37} to communicate with the system over serial-bus and render the orientation in real time. This validated that the orientation tracking does indeed work, and showed that it is responsive and not prone to drift. The python script used OpenGL and pygame to render a cube (Fig. 22) which rotated based on the orientation quaternion outputted by the microcontroller.

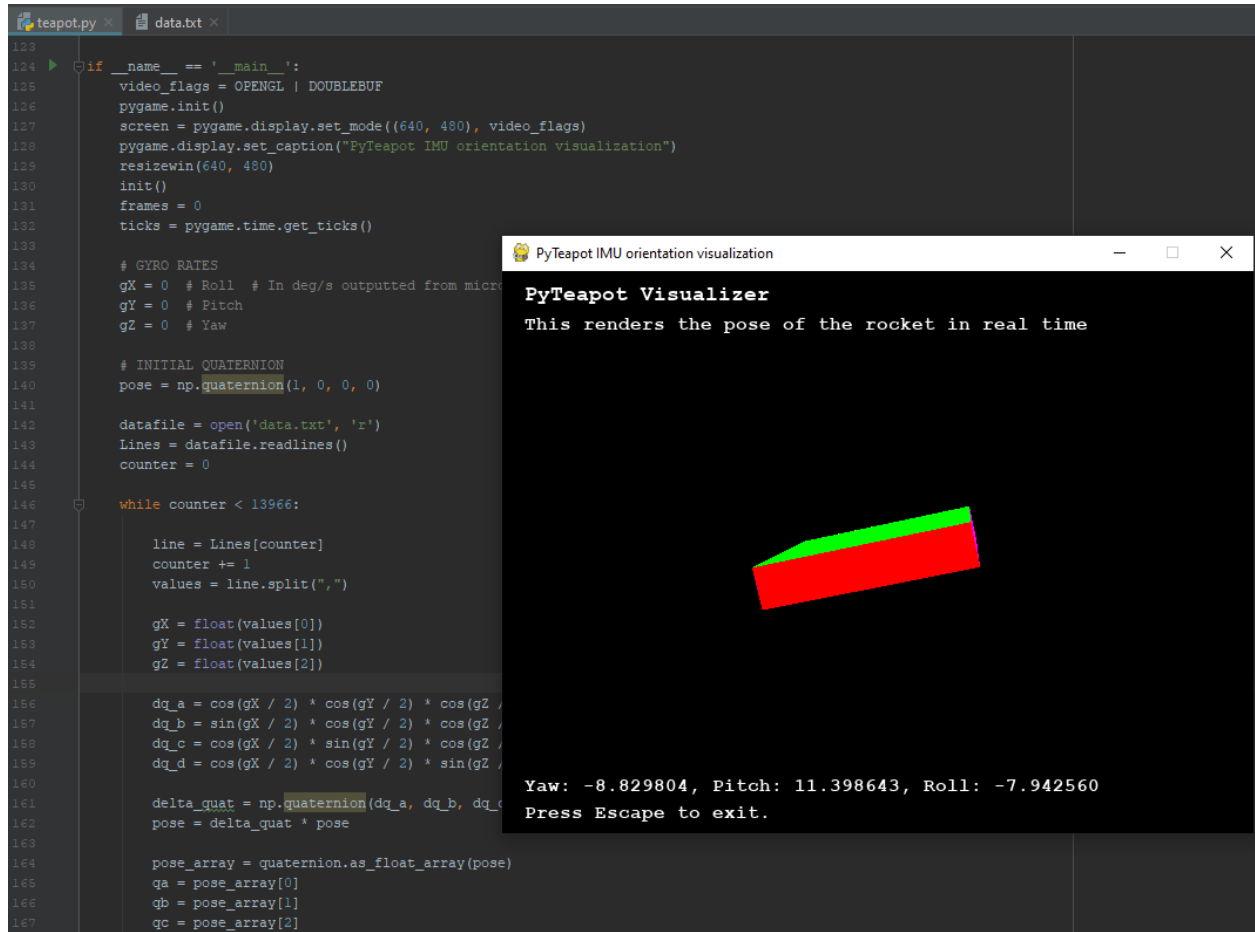


Fig. 22: Real time visualization of the quaternion 4D values to the rocket's orientation.

3.4 Results from the test launch:

The first test launch successfully reached an apogee of ~700ft and the data collection system worked perfectly. However a failure in the bulkhead during parachute deployment caused the top half of the rocket to plummet to the ground without a parachute. This was captured by the sensors onboard the guidance system. The goal of this flight was to capture data that could be used to improve and test the guidance system before attempting to control the rocket. So the canards were set to the neutral position and no active corrections were employed. In that manner the flight was successful.

The microcontroller recorded the following readings at a rate of 100hz:

- Gyro rates (Angular velocities) in X, Y and Z axis
- Acceleration in X, Y and Z axis
- Altitude (in meters)

The rocket reached an apogee of 700ft, and a maximum ascent rate of 57m/s.

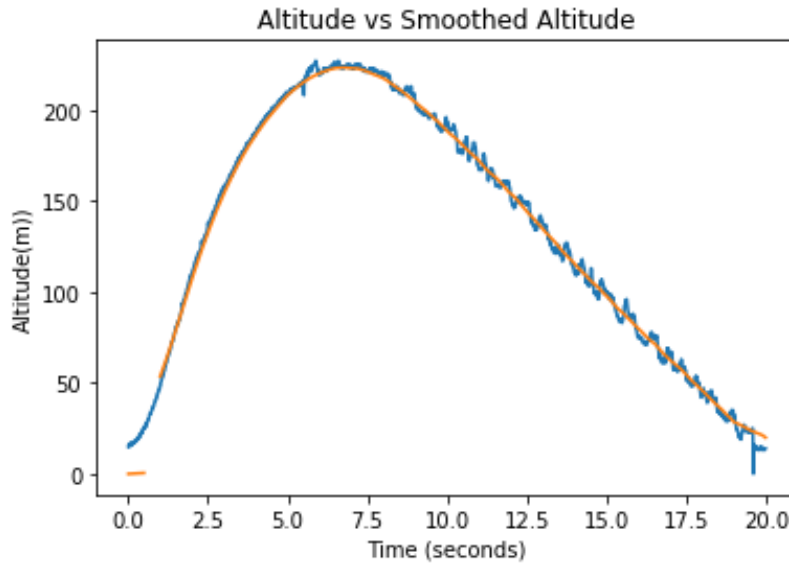


Fig. 23: Plot of altitude over time shows an apogee of 211meters (~700ft) achieved at 5.73 seconds

Data from the accelerometer showed peak thrust at T+2.5 seconds and a peak acceleration of 6 Gs. The oscillations seen in the data in the second half of the plots (post 7 seconds) is due to the violent spinning of the nose cone section as it fell to the ground.

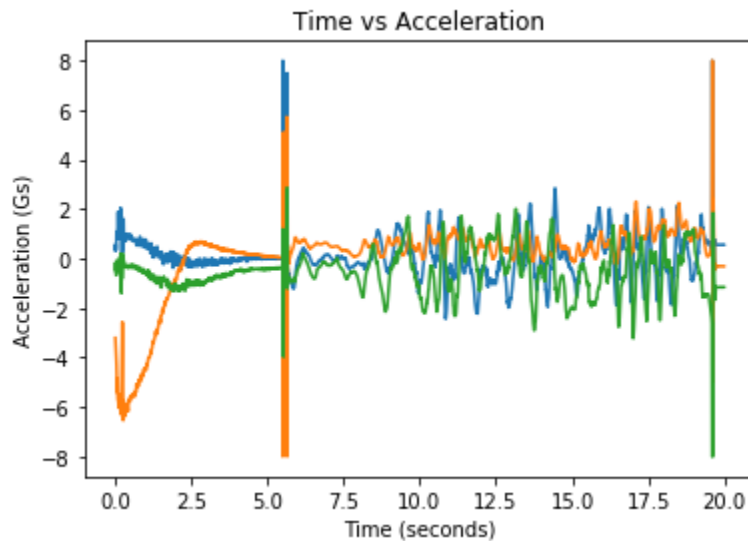


Fig. 24: Plot of acceleration in all three X, Y and Z axes. A spike seen at 5.1 seconds corresponds to the ejection charge firing and the final spike near 20 seconds corresponds to the hard-landing of the guidance module.

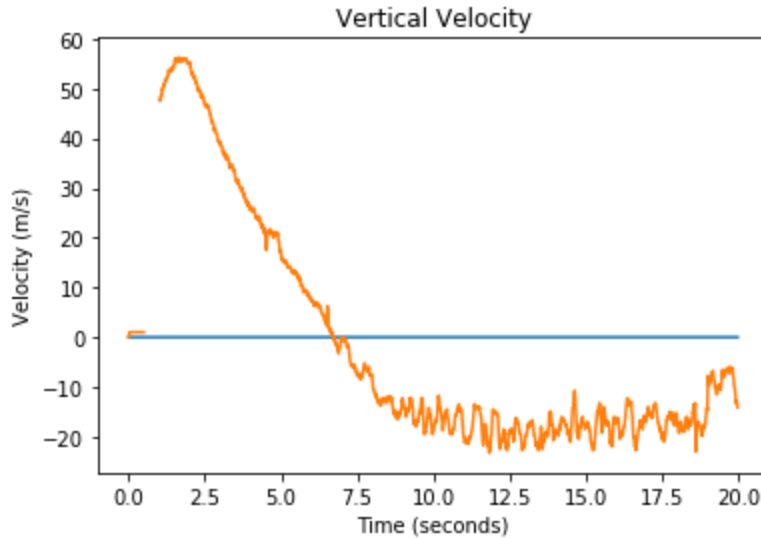


Fig. 25: Plot of instantaneous vertical velocity. It shows engine burnout at approximately 2.3 seconds and velocities during the descent (negative values).

All the data was downloaded from the microcontroller post launch and analyzed in Python in a jupyter notebook.

4.0 DISCUSSION:

Use of quaternions in orientation calculations can be very efficient, especially in small scale guidance systems that cannot afford large computational hardware. As seen with the real-time 3d render, this method of calculating orientation is responsive, accurate and free of drift. Simulation based approaches³⁹ to test the performance of the guidance system had limitations to tune the PIDs. Access to a wind tunnel could potentially help with this. We tried to drive in a car at about 40 mph holding the rocket out the window to test the response of the guidance system. This approach was only partially successful due various challenges of keeping the rocket steady and friction issues with the makeshift gimbal.

The first partially successful launch of the rocket with the guidance control system on board provided very valuable data and insights to the flight characteristics of our rocket and performance tolerances that would be expected of the guidance control. This data also was matched to the apogee prediction algorithm which will be key for the guidance control to know at which point to start leveling of the rocket so that it achieves the target apogee (in this case 800 ft). As seen below in Fig. 26, by using the instantaneous velocity of the rocket and the current orientation of the rocket, I am able to predict quite accurately the maximum altitude of the rocket a whole 3 seconds prior to the rocket reaching apogee. This should allow sufficient time for the guidance control system to relay appropriate corrections bringing the rocket to level off at the desired altitude. I am currently improving this algorithm to also consider drag forces on the rocket and effect of drag variation.

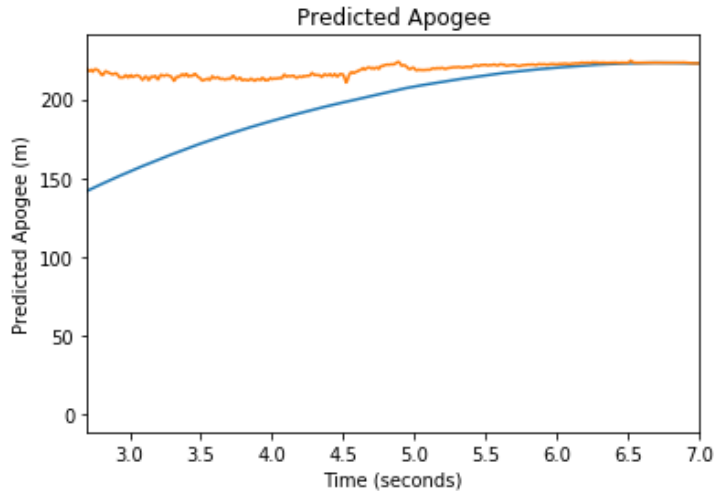


Fig 26. Predicted apogee of the rocket (orange line) vs the actual apogee achieved, showed very close predictions as early as 3 seconds into the flight.

Some of the other challenges that remain to be resolved are around the numerical accuracy of the sensor readings. In the current version the sensors output only upto two decimal places. I believe that with increased accuracy of the sensor outputs, the guidance performance can be greatly improved. However, compared to using Euler geometry⁴⁰, this approach is much more mathematically simple and effective. The data from the on ground testing of the system shows that this system can be very responsive and function within the limits of tolerance.

Most model rockets that deploy guidance systems have previously taken the conventional approach of corrections based on Euler geometry, but these systems are prone to gimbal lock and slow in responsiveness. Use of a Quaternion framework to determine orientation the necessary correction avoids these problems.

5.0 CONCLUSIONS AND FUTURE PLANS:

Guidance systems are very complex electrical, mechanical and analytical platforms. I was able to show successfully that a functioning guidance mechanism can be developed for model rocketry. This project taught me many different skills, from designing electronic circuits and PCBs to solving complex quaternion based mathematical functions and writing highly efficient and responsive algorithms. Further, this is a very deep and extensively researched topic with a lot of information published. I have only touched the tip of the iceberg and plan to continue to explore and learn more including advanced probabilistic modeling approaches to control²⁹. After successful testing of the guidance mechanism in a real flight my goal is to continue to improve the algorithms and gather additional data through both simulations and additional launches that I plan to use for training a machine learning^{41,42,43} model that could do the same or faster guidance control as the quaternion based preprogrammed calculations.

6.0 ACKNOWLEDGEMENTS:

This project would not have been possible without the support and guidance of a number of people. First thanks to my high school teacher Mrs. Legge for her encouragement and sponsoring me to participate in the science fair, and Dr. Legge (MIT Lincoln labs) for valuable discussions on guidance control. Dr. Allen (LHS) for the mentorship and Mr. Hille (Raytheon Inc.) for directing me to various education materials and links. Mr. Robinson of JHU CTY for teaching me concepts of statistics and data analysis. Mr. Nitin Shyamkumar (PhD Student, NYU Courant institute of mathematical sciences) for helping me understand complex mathematical concepts. Ms. Ali Hamlin (Electrical Engineering Graduate Student, Dartmouth College, NH) for advice on microcontrollers. Anhad Sawhney for teaching me design of electronic circuits and PCBs. Miles Houston (LHS sophomore) for helping with fabrication of Rocket fins. Thanks to Vidushi Meel (Junior LHS, TARC club captain) for inviting me to join the rocketry club and for helping review and formatting this manuscript. Thanks also to the rest of the LHS rocketry club for the fun times and help with 3D printing of certain components. And finally my parents for the financial support and encouraging me to pursue my passion.

7.0 REFERENCES

1. Schombert (2021), Early Rocketry Lecture notes. Uoregon.edu.
<http://abyss.uoregon.edu/~js/space/lectures/lec01.html>
2. Dick, J.S and Garber S. (2006). Sputnik Biographies--Wernher von Braun (1912-1977). Nasa.gov.
<https://history.nasa.gov/sputnik/braun.html>
3. Couch, C. (2019, October 24). Pigeon pilots. MIT Technology Review; MIT Technology Review.
<https://www.technologyreview.com/2019/10/24/238466/pigeon-pilots/>
4. The World of Air Transport in 2018. (2018). Icao.int.
<https://www.icao.int/annual-report-2018/Pages/the-world-of-air-transport-in-2018.aspx#:~:text=A%20according%20to%20ICAO%27s%20preliminary%20compilation,a%203.5%20per%20cent%20increase>
5. Blackmore, L. (2018). Autonomous Precision landing of space rockets. SpaceX.
<https://www.naefrontiers.org/55374/Paper>
6. SpaceX Starship. (2021) <https://www.spacex.com/vehicles/starship/>
7. Segnale, V. (2019). Working with autonomous vehicles-the skills you need.
<https://blog.v-hr.com/blog/working-with-autonomous-vehicles-the-skills-you-need>
8. Benson, T. (2014). Guidance System. Nasa.gov.
<https://www.grc.nasa.gov/www/k-12/rocket/guidance.html#:~:text=The%20guidance%20system%20has%20two,control%20the%20rocket%20during%20maneuvers.&text=Many%20different%20methods%20have%20been%20developed%20to%20control%20rockets%20in%20flight>
9. Amato, N et. al. (2001). Design, Analysis and Test of a high powered model Rocket. Aerospace Engineering Qualifying Project. Worcester Polytechnic Institute.
10. Krivanosov, A. (2002). Khartron: Computers for Rocket Guidance Systems
<http://web.mit.edu/slava/space/essays/essay-krivanosov.htm>
11. Rocket Motors - Motors by Impulse Class - Page 1 - Rocketry Works. (2021). Rocketryworks.com. <https://www.rocketryworks.com/rocket-motors/motors-by-impulse-class/>
12. Cipolla, J. (1999). Aerofin Sim 5.5. Fin Sim Description page
13. Basic Rocket stability. <http://www.rockets4schools.org/images/Basic.Rocket.Stability.pdf>

14. Siltan, S., Fresconi, F., Celmins, I. (2014). Investigation of fin and canard sizing for optimum maneuverability. Army Research Labs. <https://apps.dtic.mil/dtic/tr/fulltext/u2/a611082.pdf>
15. Roos, F., Wikander, J. (2006). The influence of gear ratio on electromechanical servo systems. IFAC Proc. Vol 39. 16. <https://www.sciencedirect.com/science/article/pii/S1474667015342804>
16. Gloss, B. (1974). Effect of canard size and location on canard interference and maneuverability. NASA technical notes. <https://ntrs.nasa.gov/api/citations/19740020361/downloads/19740020361.pdf>
17. Box, S, Bishop, C. and Hunt, H, (2009). Estimating the dynamic and aerodynamic parameters of passively controlled high power rockets for flight simulation
18. Milligan, T. (2017). Issue 442. What is the best fin shape for a rocket. ApogeeRockets.com <https://apogeerockets.com/education/downloads/Newsletter442.pdf>
19. Nakka, R. (2001). Nakkas website for model rocketry. <https://www.nakka-rocketry.net/fins.html>
20. Benson, T. (2014). Conditions for Rocket Stability. <https://www.grc.nasa.gov/www/k-12/rocket/rktstabc.html>
21. Beningo, J. (2014). 10 Steps to selecting the ideal microcontroller. Arm Community. <https://community.arm.com/developer/ip-products/system/b/embedded-blog/posts/10-steps-to-selecting-a-microcontroller>
22. Arduino Reference guide. (2021). <https://www.arduino.cc/reference/en/>
23. I2C bus: What's that. <https://www.i2c-bus.org>
24. AspenCore Inc. (2021). Pull up resistors. <https://www.electronics-tutorials.ws/logic/pull-up-resistor.html>
25. Kok, M., Hol, J. Schon, T. (2018). Using inertial sensors for position and orientation estimation. Foundations and Trends in Signal Processing: Vol. 11: No. 1-2, pp 1-153. <https://arxiv.org/pdf/1704.06053.pdf>
26. Sparkfun Electronics Inc. (2012). PCB basics of design. <https://learn.sparkfun.com/tutorials/pcb-basics/all>
27. Campbell, S. (2016). How to design a custom PCB. CircuitBasics. <https://www.circuitbasics.com/make-custom-pcb/>
28. Kim, Y. and Bang, H. (2018). Introduction to Kalman Filters and its applications. Intech.com. <https://www.intechopen.com/books/introduction-and-implementations-of-the-kalman-filter/introduction-to-kalman-filter-and-its-applications>
29. Corke, P. (2018). Roll Pitch and Yaw Angles: PeterCorke.com. <https://petercorke.com/robotics/roll-pitch-yaw-angles/>
30. Anderson, T., Christianson, R. (2017). Quaternion Guidance and control of a quadrotor <https://ieeexplore.ieee.org/document/7991509>
31. Sanderson, G. and Eater, B. (2019). Quaternions and Spatial orientation. <https://eater.net/quaternions>
32. CHRobotics (2021). Understanding Quaternions. <http://www.chrobotics.com/library/understanding-quaternions>
33. Tehrani, K., Mpanda, A. (2012) PID control theory. IntechOpen.com https://cdn.intechopen.com/pdfs/29826/intech-pid_control_theory.pdf
34. Hogenson, J. (2020) PID for dummies. Control Solutions Minnesota. https://www.csimn.com/CSI_pages/PIDforDummies.html.
35. Zhuge, S. (2020) PID control theory. Crystal Instruments Inc. <https://www.crystalinstruments.com/blog/2020/8/23/pid-control-theory>
36. Programming Arduino using Python: <https://create.arduino.cc/projecthub/smart-tech/programming-arduino-using-python-f3d2c0>
37. Eubank, N. (2015). Data analysis with Python. <http://www.data-analysis-in-python.org/>
38. Selvaraj, N (2020). A beginners guide to data analysis using python. <https://towardsdatascience.com/a-beginners-guide-to-data-analysis-in-python-188706df5447>
39. openRocket: Technical documentation. <http://openrocket.info/documentation.html>

40. Slabaugh, G. Computing Euler Angles from a Rotation Matrix. Queen Mary University, London.
<http://www.gregslabaugh.net/publications/euler.pdf>
41. Khargonekar, P. (2018). Future directions in control in the era of machine learning and artificial intelligence.
https://cpb-us-e2.wpmucdn.com/faculty.sites.uci.edu/dist/8/644/files/2019/04/Khargonekar_Tel_Aviv_2018_Final.pdf
42. Gaudet, B., Linares, R. and Furfaro, R. (2018). Integrated guidance and control for pinpoint mars landing using reinforcement learning. MIT Arclab
http://arclab.mit.edu/wp-content/uploads/2018/12/2018_astro04.pdf
43. Koch, W. et. al. (2019) Reinforcement learning for UAV attitude control. ACM Transactions.
<https://dl.acm.org/doi/fullHtml/10.1145/3301273>
44. Valledor, C., Wimmer, A., DeHate, R., (2012). MIT Rocket team Critical Design Review.