Reaction Wheel for Active Roll Control of High Power Rockets

Nicholas Marks

December, 7, 2020

Contents

1 Introduction and Motivation

Before discussing the design of the reaction wheel in depth, a brief look at the physics behind reaction wheels will be useful in understanding the goals of this project and how they were achieved. A reaction wheel is a type of flywheel that utilizes the torque generated by a rotating mass with non-zero acceleration to control the orientation of system about the reaction wheel's axis of rotation. In this project, a reaction wheel placed within a rocket such that the rocket's central longitudinal axis (See roll axis in Figure 1) is the same as the roll axis of the rotating mass on the reaction wheel, is capable of controlling the roll orientation of the rocket. For example, if a reaction wheel is housed in a rocket as described above, if it accelerates a rotating mass clockwise, it will induce a counter clockwise torque on the rocket. However, it is important to note that since torque (τ) is defined as $\tau = I\alpha$, there will be no torque by the reaction wheel on the rocket if the rotating mass of the reaction wheel is rotating at a constant speed, or in other words $\alpha = 0$. To take advantage of these facts to facilitate control about an axis, a feedback loop will need to be developed.

In order to describe the rotations of a rocket during flight, a three dimensional coordinate system can be defined by a roll, pitch, and yaw axis, each being perpendicular to each other and passing through the rocket's center of gravity. A diagram showing these axes is presented in Figure 1. To ensure a stable and predictable flight profile, the rocket's rotations must be controlled about all three axes. Traditionally for high power sounding rockets, *passive* stability is achieved via a fixed set of fins. Fins effectively stabilize a rocket by generating a restorative lift force that keeps the rocket flying straight. However, passive fins provide little restorative force to prevent rotations about the rocket's roll axis. Assuming the fins are attached perfectly perpendicular to the airframe of the rocket, they will not induce any rolling about the roll axis, however if any rolling occurs due to other factors, such as wind, the fins will not impede this rolling very much at all. A rolling rocket will result in an increase in skin friction drag, decreasing the projected apogee, but it will also increase the stability of the flight. Depending on the mission objectives, having active control of the rocket's rotations about its roll axis can be crucial.

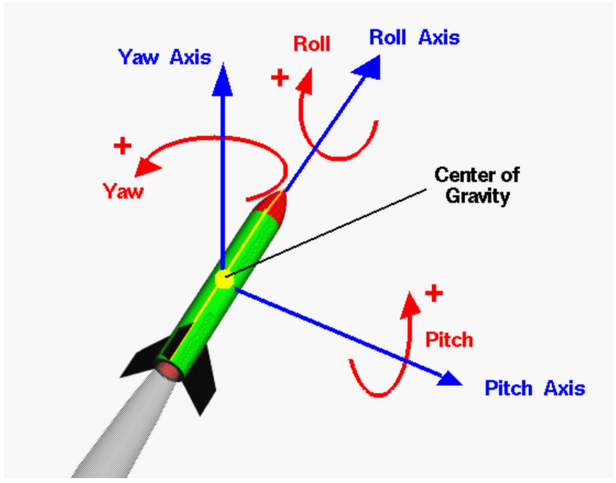

Figure 1: Rocket Coordinate System

In this report, a reaction wheel capable of facilitating active roll control for a high power sounding rocket is presented, including the design of a prototype and results obtained from ex-

perimental testing. Considering sounding rockets are typically used for research or educational purposes, roll control can be very important. For example, a mission objective may be to launch a rocket with an on-board camera to obtain stable video with minimal spinning. A reaction wheel that prevents rolling of a rocket can be utilized to achieve this goal. Another example for why active roll control is beneficial is simply to meet the requirements of a high power rocketry competition, many of which often propose mission objectives involving active roll control or other forms of stabilization.

2 Parts Selection

2.1 Electronics

Microcontroller An Arduino Uno was selected for the microcontroller for the reaction wheel. It was chosen for its flexibility, user friendly design, good documentation, and large user base and online community support.

Motor Driver A Pololu VNH5019 motor driver was used to drive the brushed DC motor. This motor driver was determined to be capable of handling the current draw of the chosen motor.

Inertial Measurement Unit The Adafruit BNO055 IMU was the sensor chosen to measure angular position, velocity, and acceleration. This sensor was chosen in particular because it has been known to be reliable and effective.

Batteries Two rechargeable 9 volt batteries were used. One to power the Arduino, and one to power the motor/motor driver. Traditional 9 volt batteries were found to be plenty capable for this motor and motor driver.

Data Acquisition To allow easier data acquisition and PID tuning, a HC-05 Bluetooth module was used to transmit data from the Arduino to a computer wirelessly. MATLAB code was used to read, save, and plot the serial data from the Bluetooth module, and can be found in Appendix B

2.2 Mechanical Components

The mechanical components were selected primarily with the goal of minimizing cost and manufacturing time required to produce a functioning prototype, and therefore 3D printing (ABS plastic) was utilized heavily. Since the majority of the difficulty of this project is the electronics and software, the mechanical design was intended to be cheap, simple, and easy modifiable, so more time and energy could be dedicated to the electronics and software.

Motor The Mabuchi RS-555SH brushed DC motor was chosen for the motor to control the reaction wheel because it was determined to be capable of delivering the torque required to rotate

the aluminum mass at a high speed.

Rotating mass A aluminum cylinder that was left over from a past project was used for the reaction wheel's rotating mass. This part can be seen in Figure 8.

Electronics tray A 3D printed electronics tray (Figures 2, 3, and 10) was used to house all the electronics. It was designed to hold just the Arduino inside, while the rest of the components were attached to the top.

DC motor mount The DC motor was mounted to the system via a friction fit to a 3D printed motor mounting disk pictured in Figure 8. Additionally, the motor shaft was friction fit to 3D printed motor shaft coupler and then attached to the rotating mass.

Threaded rods Two steel threaded rods with nuts and washers were selected to connect the electronics tray to the motor mounting plate. A third threaded rod was used to connect to the top of the electronics tray along with a ball bearing which is used for during testing.

3 Design

3.1 Mechanical Design

The main objective of the mechanical design of the prototype was to allow for easy and effective testing. The prototype, shown in Figures 2 through 10, features a 3D printed tray that houses the Arduino Uno microcontroller inside, as well as the BNO055 IMU, motor driver, and Bluetooth module on top connected together with a breadboard and jumper wires. The electronics tray features two slots on the top to allow for wires to connect to the Arduino inside. The engineering drawings for this part can be seen in Figures 2 and 3. This configuration was chosen to allow for electrical components to be easily swapped and rearranged during testing. Also connected to the 3D printed electronics compartment are three steel threaded rods secured to the top and bottom of the tray via three holes with steel nuts and washers. The threaded rod connected to the center of the top of the electronics tray is used to hold the reaction wheel during testing. Atop this rod there is a ball bearing (shown in Figure 7), which during testing is held to allow the reaction wheel system below to rotate freely about it's central axis. The two threaded rods on the bottom of the tray are used to to connect to the motor mounting disk, for which the engineering drawing is shown in Figure 4. These threaded rods are also secured to this disk through holes and with a two pairs of nuts and washers. Then, the DC motor is connected through the top of the motor mount disk by a friction fit. Finally, the rotating mass for the reaction wheel is connected by a 3D printed motor coupler, pictured in Figure 8. This motor coupler is then friction fitted onto the motor shaft.

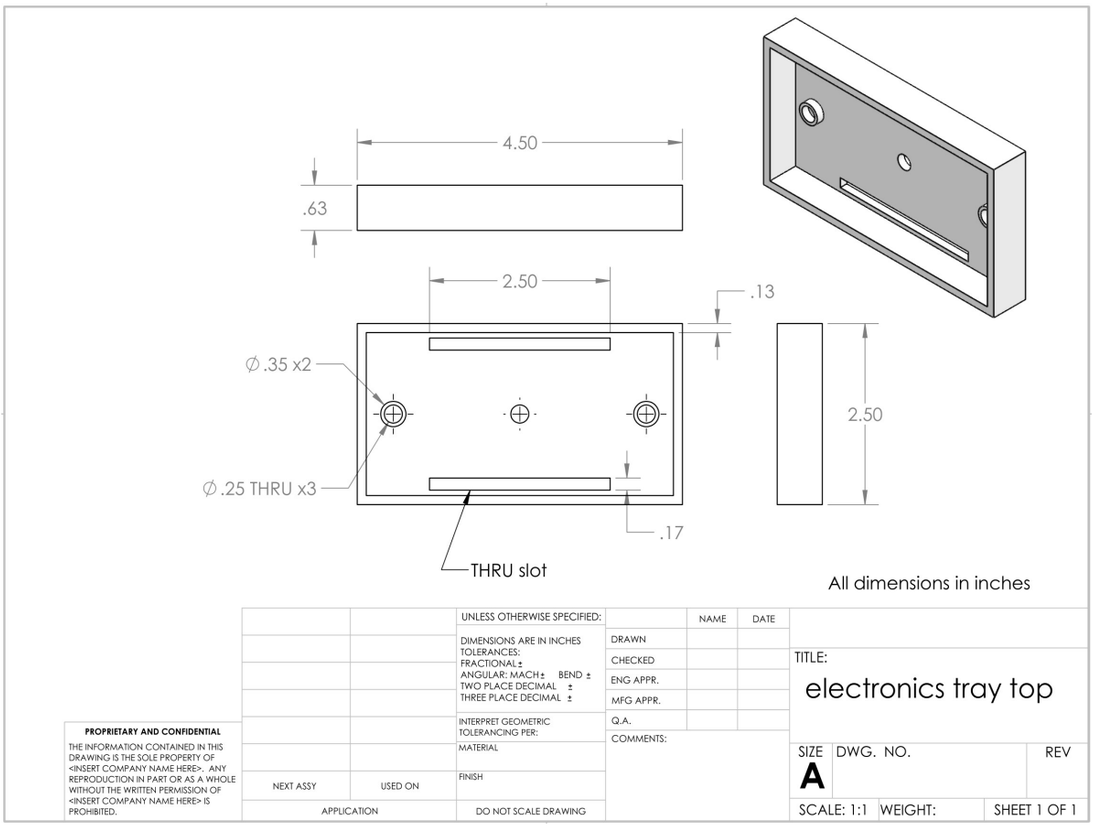

Figure 2: Top half of electronics tray

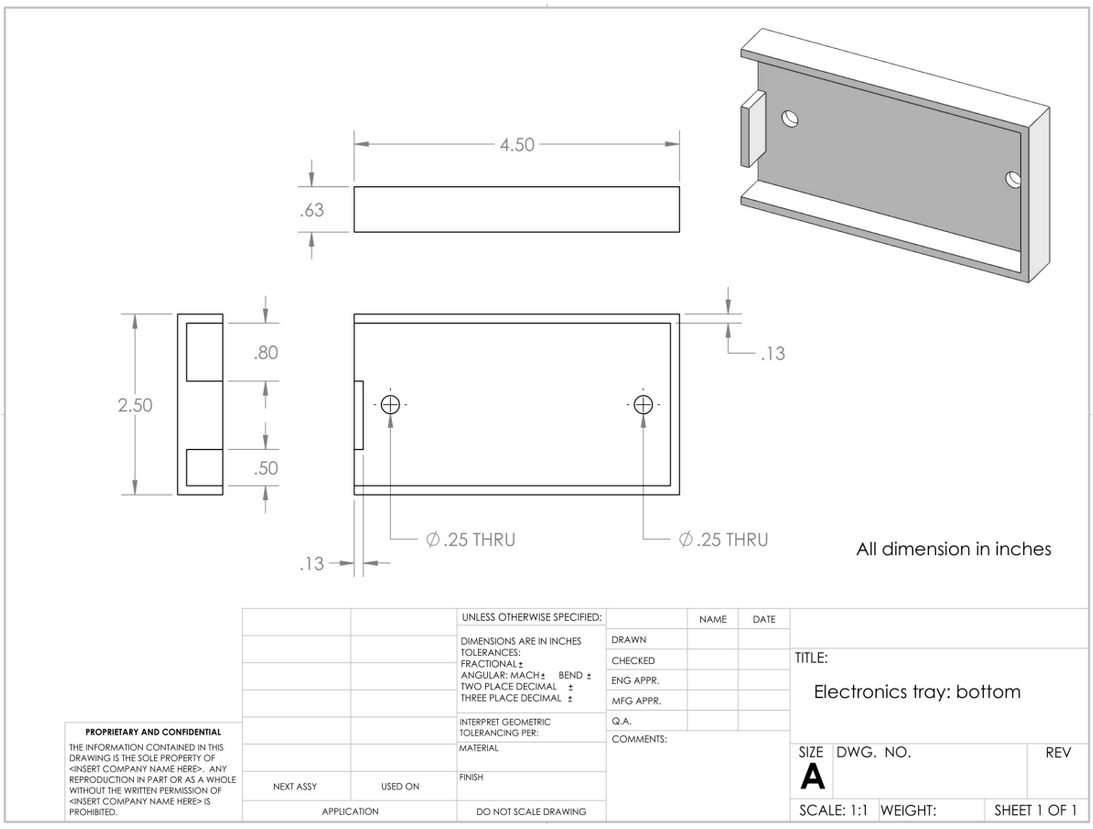

Figure 3: Bottom half of electronics tray

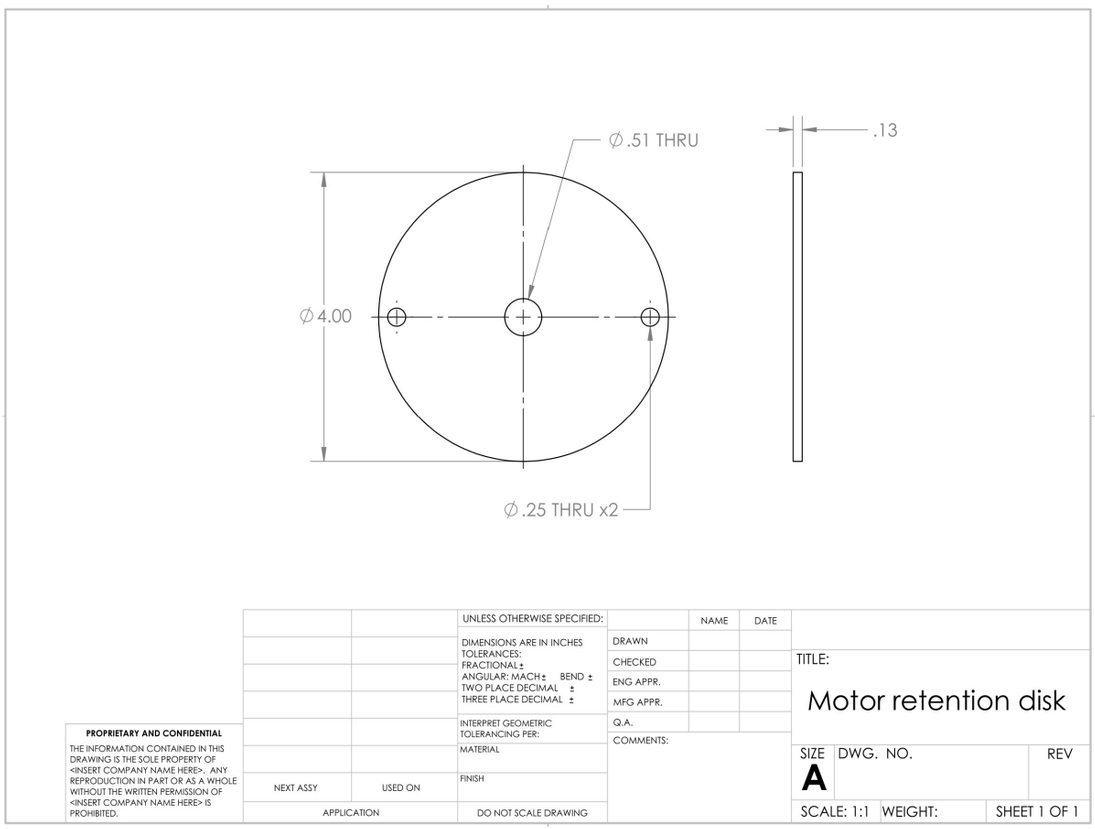

Figure 4: Motor retention disk

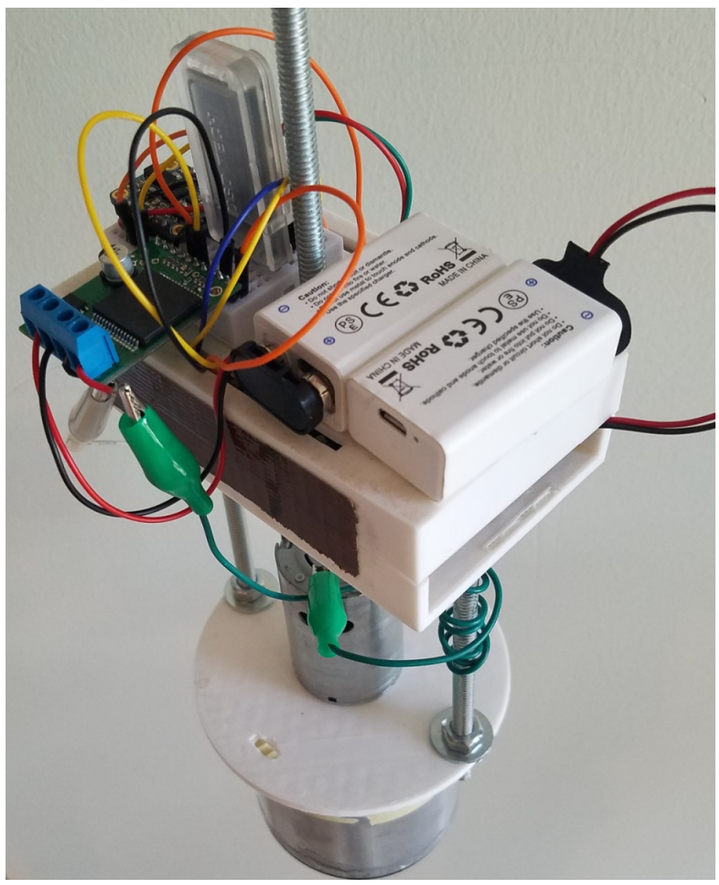

Figure 5: Full view of the reaction wheel

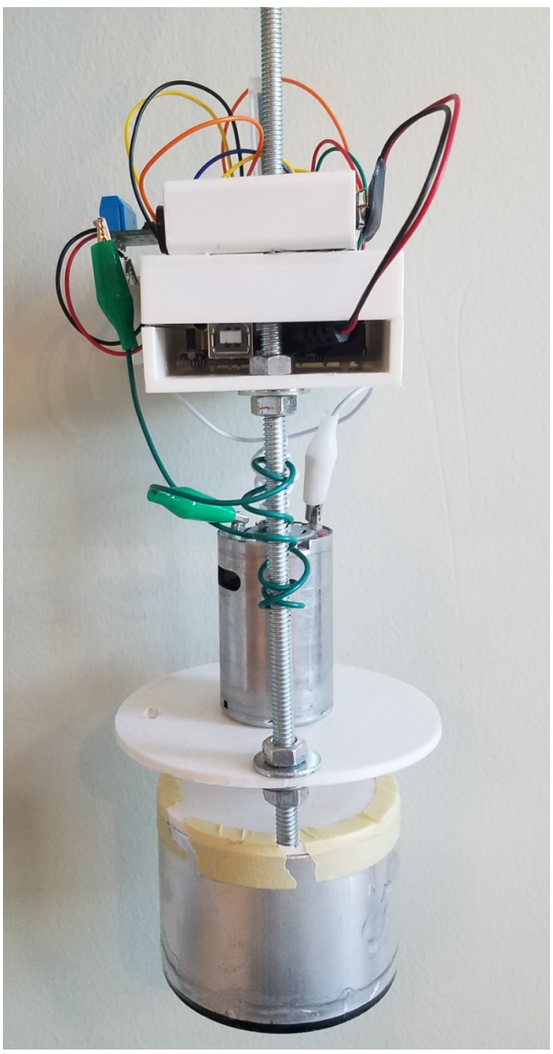

Figure 6: Side view of reaction wheel

Figure 7: Ball bearing on top of threaded rod for testing

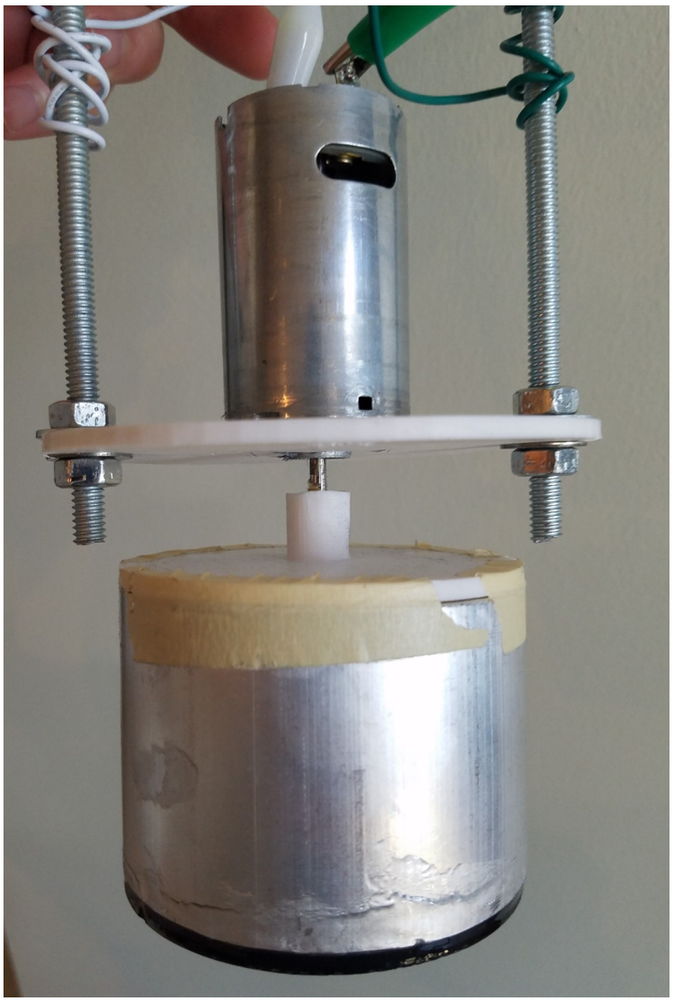

Figure 8: Rotating mass and motor

3.2 Electronics Design

A conceptual diagram of the electronics connections can be seen in Figure 9. All the components are either wired directly to the Arduino via jumper wires, or through a small breadboard which is adhered to the top of the electronics tray. Two 9-volt batteries are used to power the system, one for the Arduino, and one for the motor/motor driver. Figure 10 shows a top view of the reaction wheel showcasing the electronics.

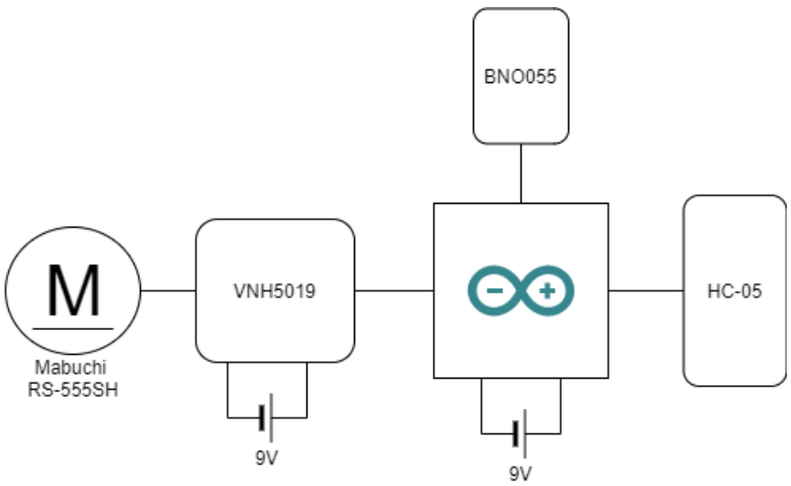

Figure 9: Electronics connections diagram

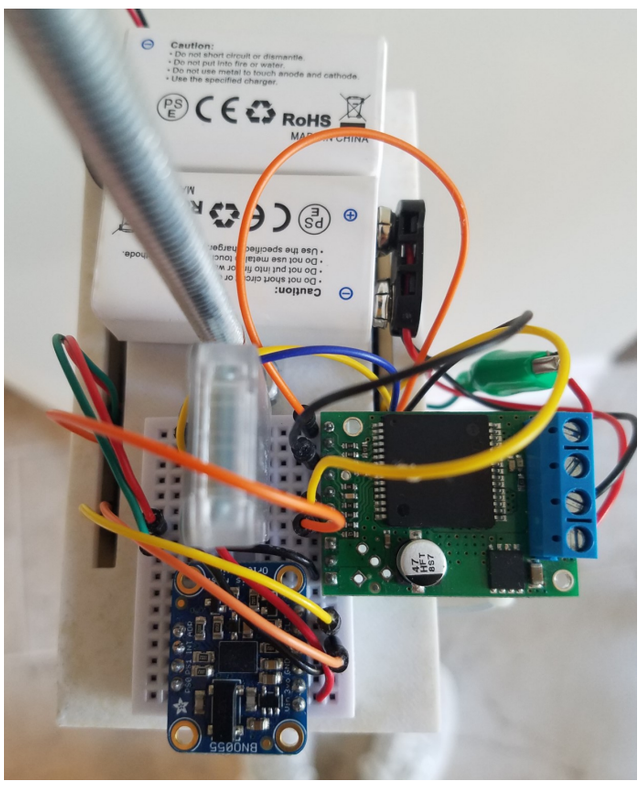

Figure 10: Top view of reaction wheel electronics

4 Software and Control Loop

After designing a mechanical prototype and selecting electronic components, the rest of the project is a software problem. More specifically the challenge is to write a feedback controller capable of controlling the roll of a rocket. The goal of this stage of the project is to *counteract* the rolling of a rocket, and to do this a target orientation of the system was defined, and the control scheme seeks to keep the rocket in this target range. If for example an external force perturbs the rocket, causing it to roll outside of the target region, the reaction wheel will automatically apply the control scheme to roll the rocket back to the target position.

PID Controller The feedback control scheme chosen for this project was a PID (proportional-integral-derivative) controller. The power of a PID controller is in its ability to control a system without "knowledge" of its dynamics. This control loop must consider the current error and target, and compute an output in such a way so as to decrease the error for each iteration of the loop, thus achieving negative feedback. The general form of the PID controller equation can be seen in Equation 1, where u is the output, and K_p, K_i, and K_d, are the proportional gain, integral gain, and derivative gain respectively.

$$u(t) = K_p e(t) + K_i \int_0^t e(t')dt' + K_d \frac{de(t)}{dt} \tag{1}$$

The PID gains will need to be "tuned" to allow for the controller to work properly for this specific mechanical system. There are several methods for gain tuning, and the method used here will be discussed later. Moreover, the control scheme used for this project will not just be a PID controller, but it will also consider a few aspects of the dynamics of the system, and therefore one could describe it as a *hybrid* controller. Using this control scheme was found to be very effective, while also allowing for easier understanding of how the reaction wheel works fundamentally.

4.1 Control Scheme

Determining Motor Acceleration First, recall that an accelerating reaction wheel will induce a torque on the rocket in a direction opposite the direction the reaction wheel is spinning. Keeping this in mind, the required sign of the acceleration vector (note that a reaction wheel with a clockwise positive acceleration will be equivalent to one with a counterclockwise negative acceleration) will need to be defined for every angle the rocket may roll through. The Adafruit BNO055 inertial measurement unit was used to measure the current x-axis Euler angle (roll angle) of the rocket at any time. Since the position of the rocket about its roll axis is an Euler angle, it ranges from 0^{circ} to 360°, and therefore a direction for the acceleration vector of the reaction wheel must be defined for each "hemisphere", depending on the target position. For example, if the target position is 0^{circ}, and if the current position, the Euler angle measured by the BNO055, is between 0^{circ} and 180^{circ}, then clearly the shortest distance along the circular arc which the rocket must roll through to get to 0^{circ} is in the counterclockwise direction, meaning the acceleration vector of the reaction wheel must be in the clockwise direction. Conversely, if the current position of the rocket is between 0^{circ} and -180^{circ}, (note the domain restriction of the Euler angle to be [-180, 180] instead of [0,360])

then the acceleration vector of the reaction wheel must be in the counterclockwise direction. This concept is then applied to arbitrary target positions, and a dedicated function was written in Arduino (see Appendix A, function motorSign()) to compute the required direction of the reaction wheel's acceleration.

Calculating the Error The error is defined to be the angular distance between the current position and the target position. The target position is a variable that the user is free to change, so therefore the error must be calculated depending on the arbitrary user-defined target position and the current position. The function computeError(), which can be see in Appendix A, shows how the error is calculated, provided that the target is in the first quadrant. Due to time restrictions, this function has been left unfinished, and target angles outside of the first quadrant will not be defined on the controller, however this is not of paramount importance and can be updated relatively easily. This function also takes advantage of the motorSign() function since many of the same relationships between the current angle and target angle were derived for it. The error will be used as input to the PID controller, along with the target position, which will be discussed in depth in the next paragraph.

Implementation of PID Control For this prototype reaction wheel, it was decided that the motor that spins the reaction wheel should only be permitted to operate in one direction, and the direction of the acceleration of the rocket will be controlled by either speeding up or slowing down the rotating mass. This decision was made because it is somewhat difficult, and perhaps unnecessary, to implement bi-direction motor control in this hybrid PID controller, although a method for doing this to improve the robustness of the reaction wheel will be discussed in the Further Development section of this paper.

Next, the PID controller must be set up in the Arduino code. To do this, a function called computePID() was created that will compute an output given proportional, integral, and derivative error terms. The output for the reaction wheel/rocket system is the PWM (pulse width modulation) value sent to the motor that rotates the reaction wheel. Essentially this is just the voltage sent to the motor, which corresponds to the speed the motor will spin. Since this function, computePID() will be executed at each iteration of the loop, the actual proportional, integral, and derivative terms will not look like Equation 1, but instead just a piece of it. The proportional term will simply be the current error, multiplied by the proportional gain, the integral term will be the current error summed with the previous error multiplied by the integral gain, and the derivative term will be the difference between the current error and last error multiplied by the derivative gain. One can see how these terms will yield the terms shown in Equation 1 after many iterations of the control loop. Since the PWM value controls the motor's speed, it may seem that this control scheme will not properly control the motor's acceleration, but this is fortunately not true. Since a change in voltage to the motor results in a change in motor's speed, and this change occurs in a finite, non-zero amount of time, there is indeed an acceleration, and since the PID controller only cares about how an output changes the error, the acceleration can be controlled. Furthermore, it is important to note that the integral term can easily lose control. This is because

it will also be summing the previous error to the current one and after a while it can become large enough to interfere with normal operation of the feedback controller. Let it be noted that typically the purpose of the integral term is to pull the system through the last few degrees where the proportional and derivative terms do not give enough acceleration to rotate the rocket anymore. In other words, the integral term is a destabilizing term that can be utilized to eliminate steady state error from the system. To fix the problem of the integral term losing control without rendering it useless, a conditional statement is used to only allow the integral term to accumulate only if the error is less than 25°, and it will only accumulate for ten iterations of the loop, after which it will reset to zero. These exact cutoff values for were chosen based on experimental testing. Once the proportional, integral, and derivative terms are defined, the output PWM value is defined as $PWMval = kp * error + ki * intError + kd * drvError$.

Execution of Feedback Loop Before entering the feedback loop, the reaction wheel is set to an idle speed. The purpose of this is so that the reaction wheel has room to accelerate in both directions (spin up or spin down) since it is only permitted to spin in one direction. The sequence of the control loop is as follows:

1. Compute current error and required reaction wheel acceleration
2. If the error is outside the target range, enter the inner while loop and continue to Step 3, otherwise repeat the first step.
3. Compute PWM value from PID controller and add or subtract it from the idle PWM value to spin up or down according to the required reaction wheel acceleration.
4. Compute error and reaction wheel acceleration again, and break out of inner while loop if the error is now within the target range and continue to Step 5, otherwise repeat step 3
5. Start over from the first step

The code for this feedback loop can be seen explicitly in Appendix A.

Gain Tuning and Data Transmission To relate the controller to the dynamics of this particular reaction wheel, the PID gain constants K_p, K_i, and K_d must be determined experimentally. There are other ways to determine these parameters, however for this case, empirical gain tuning is most likely the best method. First however, a method for data collection must be devised. As hinted at previously, an HC-05 Bluetooth module was used to transmit data wirelessly from the Arduino in the reaction wheel to a computer. From there, the data was read live into MATLAB (see Appendix B), and then saved and plotted. The data that was collected was the angle of the system, and the time that angle was measured. From these data one can generate plots of the roll angle versus time to determine the time it takes for the reaction wheel to correct for rolling of the rocket. Now that a data collection method has been established, the empirical gain tuning is possible. To tune the gains, the reaction wheel is powered on and the MATLAB script is run. First K_p is increased while leaving K_i and K_d zero. After each run, the plot of the roll angle versus time is observed. Once K_p is high enough to were it can bring the system close to the target, perhaps

with some oscillations, then K_d can begin to be increased. K_d is a damping term, and therefore it will help to eliminate some oscillations and overshoot. Once K_d is high enough to allow the reaction wheel to bring the system to near the target angle with minimal overshoot and oscillation, K_i can begin to be increased. This term will help get rid of steady state error and bring the reaction wheel through the last few degrees to the target. A more in-depth tutorial on how to do this can be found in this YouTube video from Northwestern Robotics. Plots of the roll angle of the system versus time during tuning will be discussed in the Results section, along with plots of the final results.

5 Results

This section will discuss the results of the reaction wheel active roll control system for rockets. Although there are various areas that need improvement, the principle goals of the project have been met. It has been shown that this reaction wheel design is capable of bringing the system from an error $\theta \approx 180°$ to and error of less than $10°$ in under 2 seconds.

Figure 11 shows a case where the system was perturbed by $-152.31°$. The reaction wheel actively responded to this perturbation and brought the system to within $10°$ of its target angle, $-7.69°$, within just 1.12 seconds, and to within $5°$ of its target angle in 2.44 seconds. Another

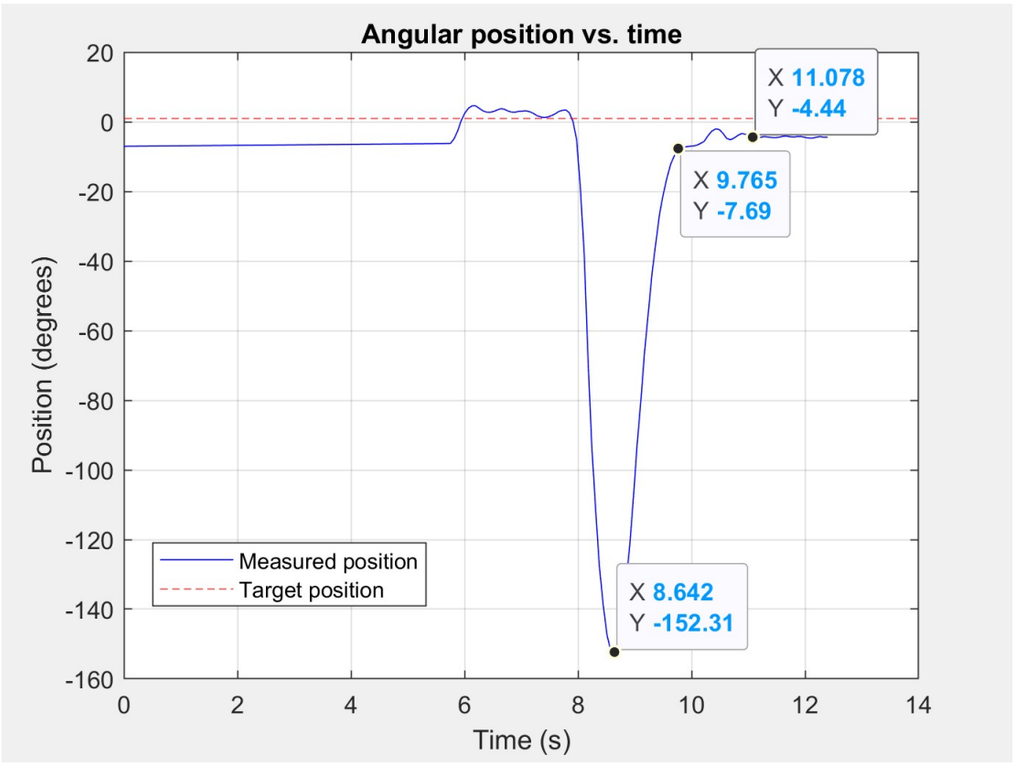

Figure 11: Error angle vs. time of system; demonstration of correction time.

example of a correction from a single perturbation is shown in Figure 12. This time the reaction wheel corrected the system to be within $10°$ of its target in 1.47 seconds.

Arguably more impressive results have been shown in cases where successive perturbations of the system are produced in both directions. Correction times remain similar to those shown in Figures 11 and 12, and the error after correction is shown to be even smaller. An example of this can be see in Figure 13 and Figure 14.

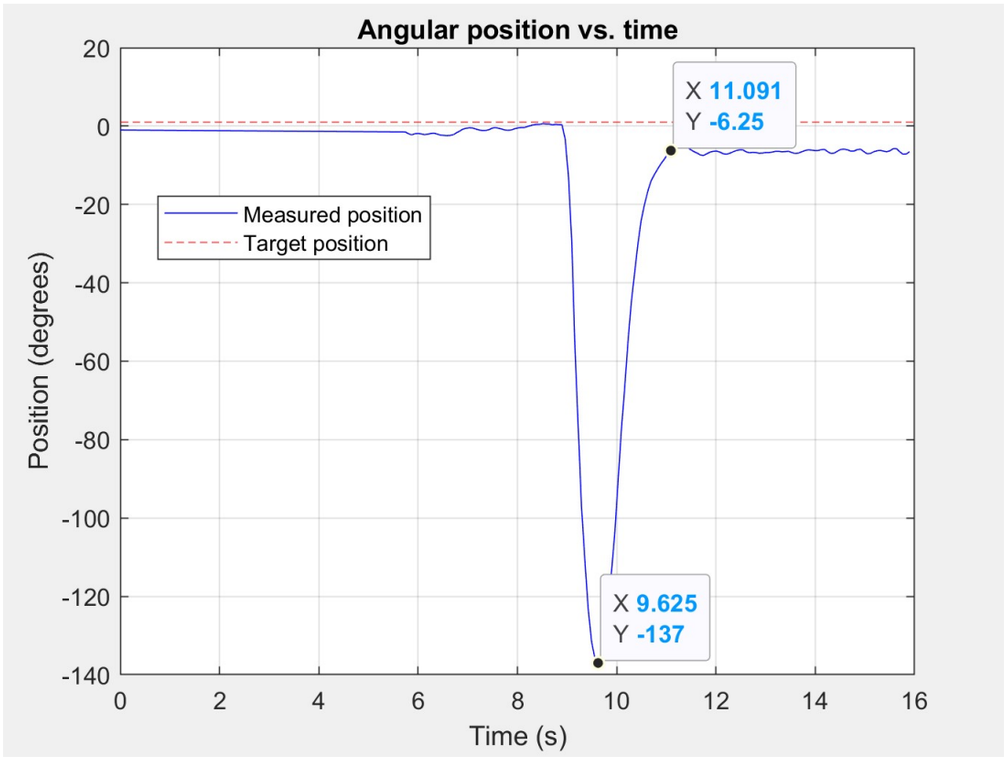

Figure 12: Error angle vs. time of system; demonstration of correction time

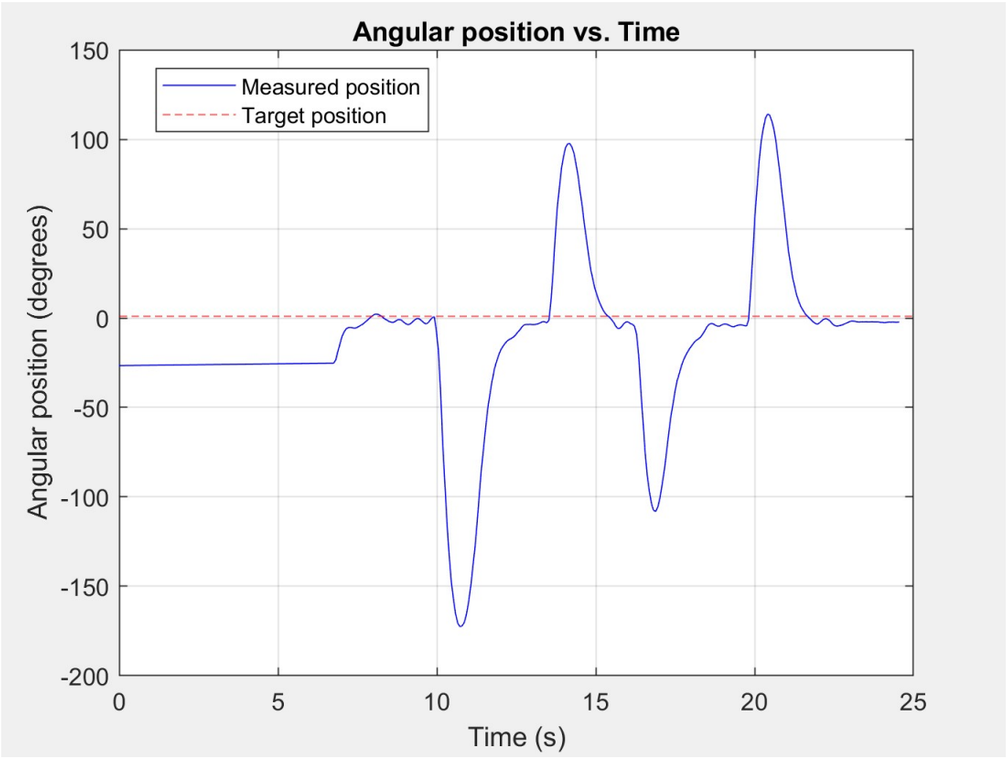

Figure 13: Error angle versus time with successive perturbations in both directions

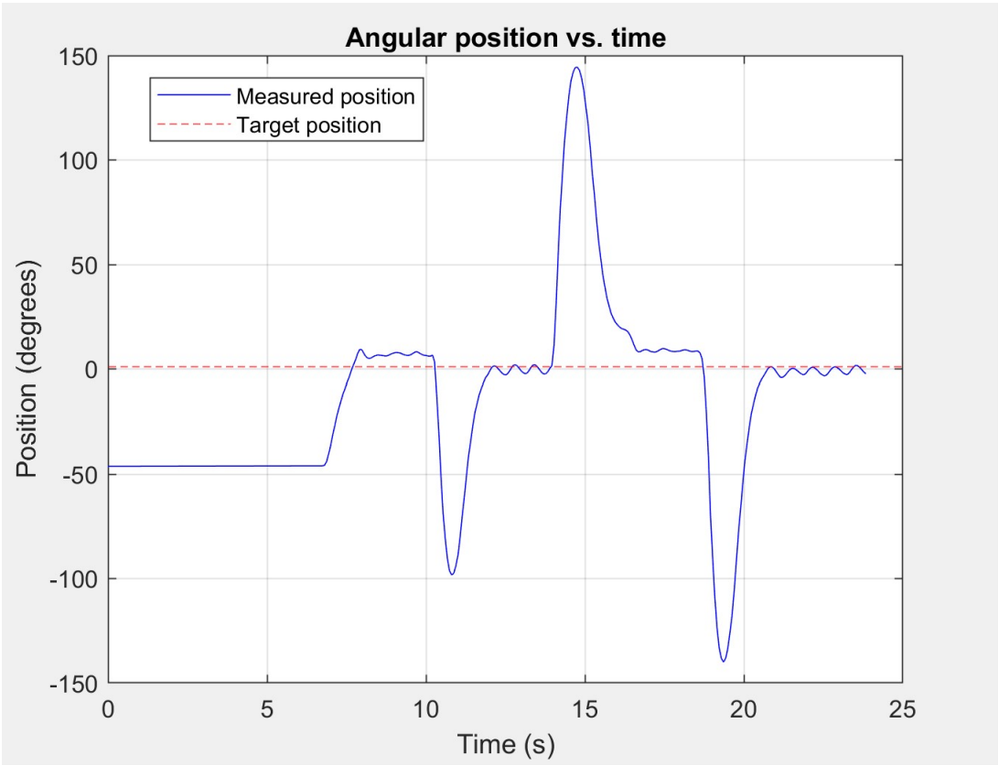

Figure 14: Error angle versus time with successive perturbations in both directions

Tuning Additionally, it is perhaps useful to see a few examples of data before the gain tuning process was complete. In Figure 15, a plot of angle versus time is shown at the start of tuning. Here only a proportional term is used. Note that eventually the systems nears the target angle, however it takes a very long time. In Figure 16, damping is introduced by adding a small derivative term by making K_d nonzero. Note how the oscillations and overshoot are shortened compared Figure 15, however there is still plenty of room for improvement. Increasing K_d even more can give a result like Figure 17. This is an example of a good response curve. The last thing to do is to increase K_i to eliminate the steady state error found in Figure 17. Once the integral term is added in, and it is controlled to prevent it from growing too large as discussed previously, the response is very good, as shown in Figures 11, 12, 13, and 14.

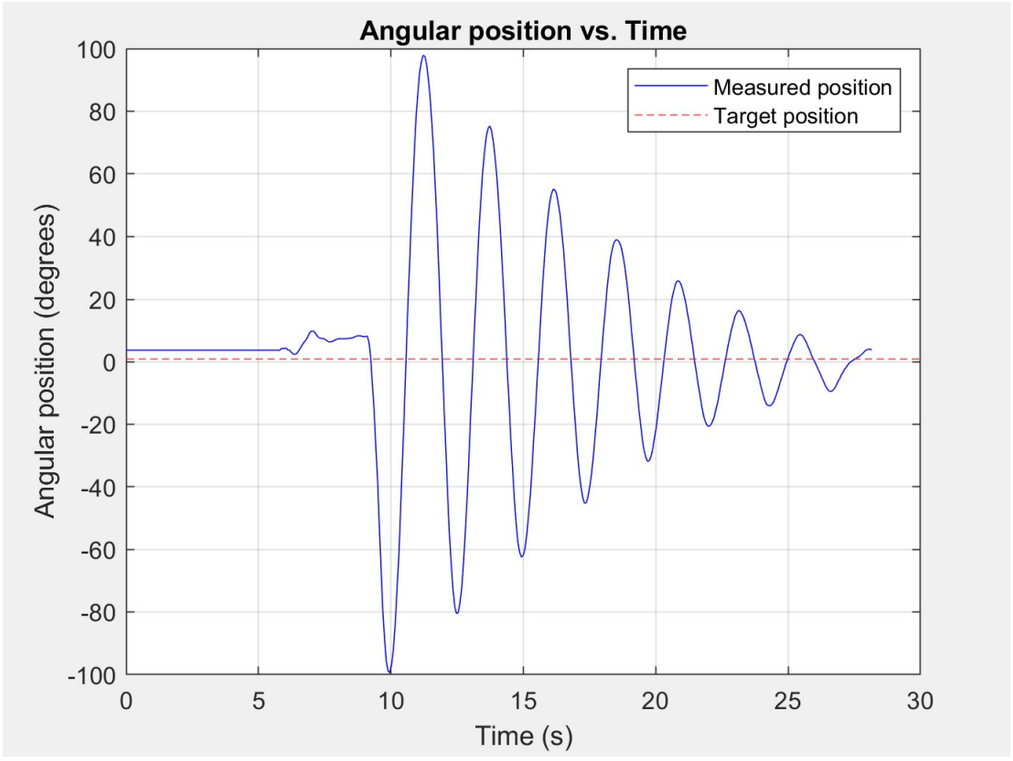

Figure 15: Response with only the proportional term

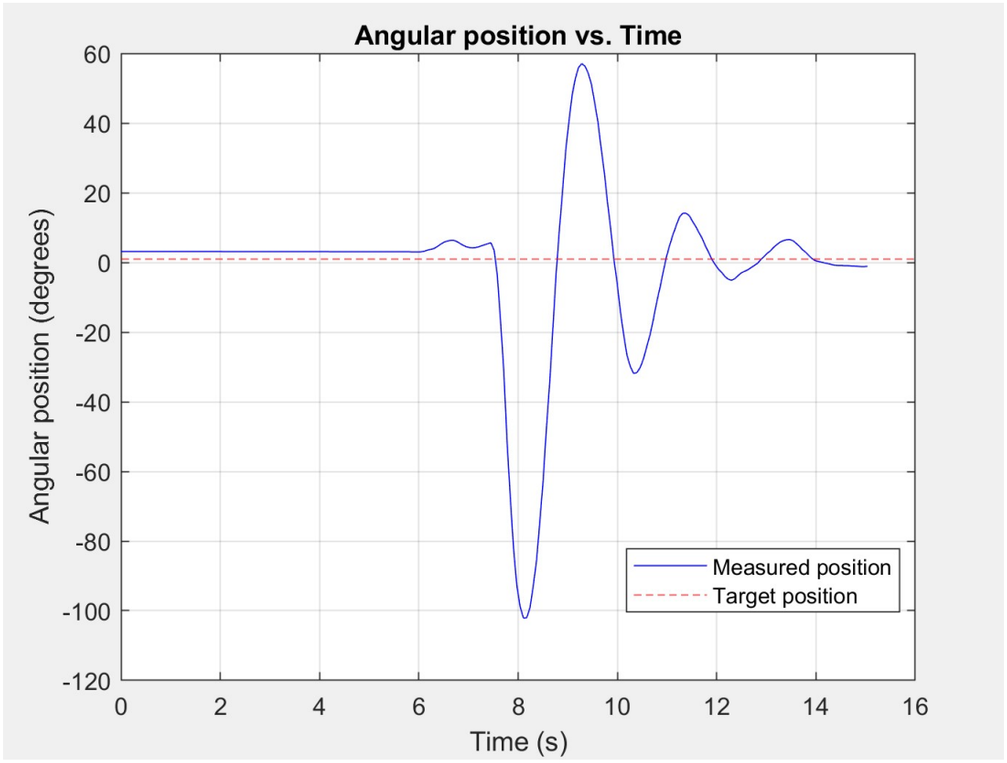

Figure 16: Response with proportional term and small derivative term

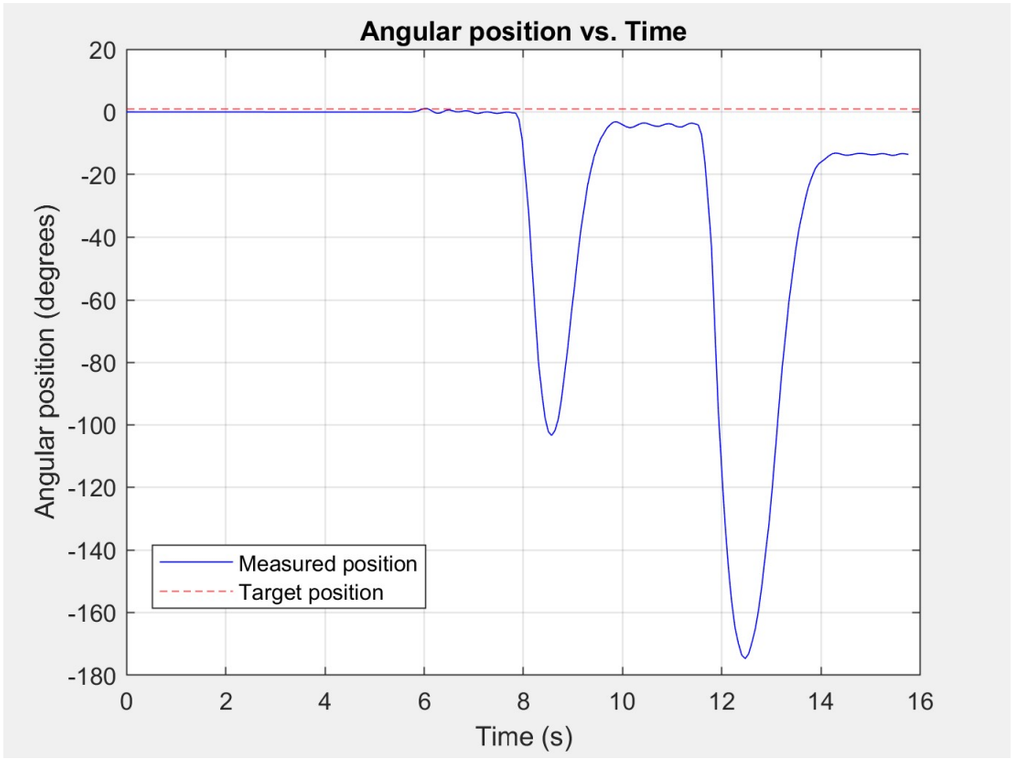

Figure 17: Response with proportional and derivative term

6 Conclusion and Further Development

In conclusion, this report presented a proof-of-concept design for a reaction wheel to facilitate active roll control for a high power rocket. Experimental tests, the results of which were presented, show the reaction wheel to be very effective at correcting for a angular displacement in a short amount of time. In general, it has been shown that the reaction wheel is capable of bringing the system from an error $\theta \approx 180°$ to and error of less than $10°$ in under 2 seconds. However, there are several areas in the design that need improvement before this reaction wheel can be implemented effectively in a rocket.

Firstly, the reaction wheel must be tested to see how well it performs at controlling the roll of a much more massive object. In these preliminary tests presented in this report, the mass of the system is nothing more that the weight of the threaded rods, electronics, electronics housing, and batteries. In theory, up to a certain point, through simply re-tuning the PID gains, the reaction wheel can operate effectively for a wide range of masses, although this needs to be tested more in depth. Furthermore, with the way the reaction wheel control scheme is setup, the motor and rotating mass could easily be upgraded to allow for it to facilitate control of heavier objects, so long as the PID gains are tuned again.

Next, there are several upgrades to the physical design that should be upgraded. Some form of latch should be designed for the electronics housing so it can be secured closed and opened when needed, since currently it is secured closed with tape. Additionally, to allow for a smaller footprint and less loose wires, a printed circuit board (PCB) could be designed and implemented in the reaction wheel. This upgrade may also required a redesign of the electronics housing. A commercially available motor shaft coupler should be purchased to properly connect the DC motor to the motor mounting plate. This will take the place of the 3D printed friction fit motor mounting coupler that is currently used in the design. Screws should also be used to secure the motor to the mounting plate instead of a friction fit. Lastly for the physical design, a method of securing the reaction wheel within a rocket should be devised.

As for the software, there are a few improvements that should be made. First the computeError() function should be updated to allow for target angles to be defined outside of the first quadrant. Furthermore, as discussed previously, the current design only spins the reaction wheel in one direction, and controls the roll of the system by either accelerating or decelerating. A potential problem with this is that the system can saturate rather quickly. This is because once the PWM value hits its maximum or minimum, it will remain there even if the controller wants to push it further. This is because the motor is restricted to just rotating in one direction. To make the reaction wheel more robust and less prone to saturation, software could be added that permits the motor to change direction when the motor saturates. For example, if the reaction wheel needs to spin down clockwise to control a certain roll of the rocket, but it hits zero and stops spinning, and therefore stops accelerating, the software could trigger it to then begin accelerating positively in the counterclockwise direction. This will yield the same torque as negatively accelerating in the clockwise direction and could decrease the frequency which the reaction wheel saturates. Lastly, more data could be read from the BNO055 such as the acceleration of the rocket/system. This

would allow for other conditions to be defined in cases where perhaps the angular acceleration of the rocket is very high or very low when the control loop begins. This also will create a more robust reaction wheel.

Appendices

A Arduino Code to Control the Reaction Wheel

```
#include <Wire.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_BNO055.h>
#include <utility/imumaths.h>
#include <SoftwareSerial.h>

/* Define pins */
int speedPin = 6;
int cw = 7;
int ccw = 8;
// Connect SCL from BNO055 to A5
// Connect SDA from BNO055 to A4

/* Initialize some variables */
float PWMval; // PWM value sent to motor, output from PID controller
float idlePWM;  // PWM value for idle speed
float maxPWM; // Maximum allowable PWM value
float theta;  // Angular position of system (x-axis Euler angle)
float delay_time; // Delay time in loop
int target_quad;  // Quadrant target position is located in
float targetTheta;  // Target position
float devTheta; // Allowable deviation from the target
int a_motor;  // Flag to assign motor acceleration direction. Either 0 (sys needs t
float error;  // Distance between current postion and target position

/* Define the BNO055 IMU */
Adafruit_BNO055 bno = Adafruit_BNO055(55, 0x28);

/* Setup bluetooth connection */
SoftwareSerial BT(4, 5); // RX,TX



/**************************************************************************/
/*
    Function to calculate error (distance betweeen current and target position
    TODO: Add case where target position is not in quadrant 1
*/
```

```
/*****************************************************************************/


/* Calculates the distance between the current angle and the target angle */
float computeError(){

  /* Error calculation when system needs to spin ccw (a_motor == 0) and target in q
  if (a_motor == 0){
    if (target_quad == 1){
      if (theta > 0){
        error = theta - targetTheta;
      }
      else if (theta < 0){
        error = ((180 - targetTheta) + (180 + theta));
      }
    }

  } // END case where a_motor == 0

  /* Error calculation when system needs to spin cw (a_motor == 1) and target in qu
  else if (a_motor == 1){
    if (target_quad == 1){
      if (theta > 0){
        error = targetTheta - theta;
      }
      else if (theta < 0){
        error = abs(theta - targetTheta);
      }
    }


  }//END case where a_motor == 1

  return error;
}//END FUNCTION


/*****************************************************************************/
/*
    Setup PID controller
*/
/*****************************************************************************/
```

```
/* Define PID constants */
float kp = 0.5;
float ki = 0.1;
float kd = 3.5;

float intError, drvError, lastError;
int intCount = 0;

float computePID(){

  /* Calculate errors */
  error = computeError();  // Error = distance from target position

  /* Only accumulate intergral error when the proportional error is small */
  if (error < 25){
    intError += error;  // Integral error
    intCount ++;
    if (intCount > 10){
      intError = 0;
      intCount = 0;
    }
  }
  else{
    intError = 0;
  }

  drvError = (error - lastError); // Derivative error

  /* Calculate output (PWM value) */
  PWMval = kp*error + ki*intError + kd*drvError;

  /* Store previous error */
  lastError = error;

  /* Return the PWM valute */
  return PWMval;

}


/*****************************************************************************/
```

```
/*
    Arduino setup function
*/
/*****************************************************************************/
void setup(void){

  /* Begin serial communication for bluetooth module */
  BT.begin(9600);

  /* Give some values to variables */
  idlePWM = 50.0;
  targetTheta = 1.0;
  devTheta = 2.0;
  delay_time = 50;
  maxPWM = 85;

  /* Initialize motor control pins */
  pinMode(speedPin,OUTPUT);
  pinMode(cw,OUTPUT);
  pinMode(ccw,OUTPUT);

  /* Set motor to idle */
  PWMval = idlePWM;
  analogWrite(speedPin,idlePWM);
  digitalWrite(ccw,HIGH);

  /* Initialize the IMU */
  if(!bno.begin())
  {
    /* BNO055 not detected */
    BT.print("No BNO055 detected. Check wiring");
    while(1); // Loops forever doing nothing if BNO055 is not detected
  }

  /* Wait for motor to reach idle speed */
  delay(5000);

}//END SETUP



/*****************************************************************************/
```

```
/*
    Function to determine the required direction of the motor's accerlation
*/
/*****************************************************************************/

void motorSign(){

  int opp;  // Angle opposite the target angle

  if ((targetTheta >= 0) && (targetTheta <= 90)){ // Target in quadrant 1 (includin
    opp = targetTheta - 180;
    target_quad = 1;
    if ((theta < 0) && (theta < opp)){
      a_motor = 0;   // want sys to spin ccw, 0 means negative motor accerlation
    }
    else if ((theta > targetTheta) && (theta < 180)){
      a_motor = 0;  // want sys to spin ccw, 0 means negative motor acceleration
    }
    else{
      a_motor = 1;  // want sys to spin cw, 1 means positive motor acceleration
    }
  }

  else if ((targetTheta < 0) && (targetTheta > -90)){ // Target in quadrant 2
    opp = targetTheta + 180;
    target_quad = 2;
    if ((theta > 0) && (theta < opp)){
      a_motor = 0;  // want sys to spin ccw, 1 means negative motor acceleration
    }
    else if ((theta < 0) && (theta > targetTheta)){
      a_motor = 0;  // want sys to spin ccw
    }
    else{
      a_motor = 1;
    }
  }

  else if ((targetTheta < -90) && (targetTheta > -180)){ // Target in quadrant 3
    opp = targetTheta + 180;
    target_quad = 3;
    if ((theta > 0) && (theta < opp)){
      a_motor = 0;  // want sys to spin ccw
```

```
    }
    else if ((theta < 0) && (theta > targetTheta)){
      a_motor = 0;  // want sys to spin ccw
    }
    else{
      a_motor = 1;  // want sys to spin cw
    }
  }

  else{  // Target is in quadrant 4
    opp = targetTheta - 180;
    target_quad = 4;
    if ((theta > 0) && (theta > targetTheta)){
      a_motor = 0;  // want sys to spin ccw
    }
    else if ((theta < 0) && (theta < opp)){
      a_motor = 0;  // want sys to spin ccw
    }
    else{
      a_motor = 1;  // want sys to spin cw
    }
  }
}




/**************************************************************************/
/*
    Arduino loop function
*/
/**************************************************************************/
void loop(void){

  /* Get Euler angle vectors from the IMU */
  imu::Vector<3> euler = bno.getVector(Adafruit_BNO055::VECTOR_EULER);

  /* Get current angular position (x axis Euler angle) and restrict domain between
  theta = euler.x();
  if (theta > 180){
    theta = theta - 360;
  }
```

```
/* Print some values */
BT.print(theta);
BT.print(",");
BT.print(millis());
BT.println("");

/* Run functions to compute current error and required direction of the motor's
motorSign();
error = computeError();

delay(delay_time);

/* While the current angle is outside the target range, run feedback loop to brin
while (error > devTheta){

  /* Get current position, same at before */
  imu::Vector<3> euler = bno.getVector(Adafruit_BNO055::VECTOR_EULER);
  theta = euler.x();
  if (theta > 180){
    theta = theta - 360;
  }

  /* Calculate the direction motor needs to spin, the error, then calculate PWM
  motorSign();
  error = computeError();
  PWMval = computePID();

  /* Reaction wheel will spin up or spin down accordingly */
  if (a_motor == 0){
    PWMval = idlePWM - PWMval;
  }
  else if (a_motor == 1){
    PWMval = idlePWM + PWMval;
  }

  /* Restrict the PWM value to be postive and less than the pre-defined maximum
  if (PWMval > maxPWM){
    PWMval = maxPWM;
  }
  else if (PWMval < 0){
    PWMval = 0;
  }
```

```
    /* Send new PWM value to motor */
    analogWrite(speedPin,PWMval);

    /* Print some values */
    BT.print(theta);
    BT.print(",");
    BT.print(millis());
    BT.println("");

    delay(delay_time);
}

}//END LOOP
```

B MATLAB Code for Reading Bluetooth Data

```
% This script reads, saves, and plots data from Arduino bluetooth module

delete(instrfindall);    % Close all serial ports
BT = serialport('COM5',9600);     % Define bluetooth serial port

% Make a button to stop data aquisition when desired
figure;
set(gcf,'position',[500,500,100,50])
ButtonHandle = uicontrol('Style', 'PushButton', ...
                         'String', 'Stop', ...
                         'Callback', 'delete(gcbf)');

% Aquire data until button press
data = [];
i = 1;
while 1 == 1
    if ~ishandle(ButtonHandle)
        disp('Data aquisition stopped');
        break
    end
    rawline = readline(BT);
    elem = strsplit(rawline,',');
    fprintf('theta: %f\ttime: %.2f\n',elem(1),elem(2));
    data(i,1) = elem(1);
    data(i,2) = elem(2);
    i = i + 1;
end

% Normalize time to first data point and convert to seconds
for i = 1:length(data(:,2))
    data(i,2) = (data(i,2) - data(1,2))/1000;
end
theta = data(:,1);
time = data(:,2);
target = 1; % Target is 1 degree

% Save the data as a text file to the DATA folder
name = input('Enter filename to save, or 0 to not save: ');
if name ~= 0
    folderpath = "C:\Users\Owner\OneDrive\Documents\ME399 Reaction Wheel\MATLAB Co
```

```
    filename  = fullfile(folderpath,name);
    writematrix(data,filename);
end

% Plot the data
figure;
plot(time,theta,'b');
hold on
grid on
yline(target,'--r');
legend('Measured position','Target position');
title('Angular position vs. time');
xlabel('Time (s)');
ylabel('Position (degrees)');

clearvars -except data
```

C Google Drive

For the interested reader who would like to see videos of the reaction wheel in action, all the code used in this project, and CAD files, Google Drive Folder has been made available. Below are a few quick links for easy access.

- Final prototype demonstration video
- MATLAB data acquisition demonstration