A Reaction Control System of Thrusters for Model Rocketry

Kevin Dee

May 6, 2022

Swarthmore College
Department of Engineering

A Reaction Control System of Thrusters for Model Rocketry

Kevin Dee

Abstract

In rocketry, one of the most basic design requirements is that the rocket flies with stability. In this project, I sought to bring a new method of achieving stable flight to hobby rocketry by designing a reaction control system of thrusters to work in a model rocket. I satrted by building a normal model rocket, then I designed an electronics system capable of controlling four valves. Next, I used this electornics system to control the flow of air from a source to four nozzles. I integrated this system into my rocket, and used an inertial measurement unit and Arudino microprocessor to control the system. In tests, I found some issues that are yet to be resolved, but ultimately I found that a compressed air reaction control system would work for model rocketry purposes, although such a solution is unnecessarily overcomplicated for achieving stability in flight.

Introduction

At the end of 2015, SpaceX performed the first vertical landing of an orbital rocket by returning their Falcon 9 to the launch site on the Florida coast. This achievement was made possible by the use of complex control systems that make use of inertial reaction wheels, thrust vectoring, control surfaces, and compressed gas thrusters.

Figure 1: Falcon 9's cold gas thrusters firing to reorient the first stage

These control methods have developed a lot in the past few decades as the aerospace industry has grown, but they are only used in orbital-class rockets and large satellites, meaning they are unproven on small scales. That's why I decided to design a compressed gas thruster system for model rocketry.

Background

Cold Gas Thrusters

Cold gas thrusters are a staple of aerospace control systems. They produce thrust by releasing compressed gas, typically nitrogen, through a nozzle. Nozzles are often positioned in clusters and aimed at different directions. These clusters are usually placed at the ends of the vehicle to maximize their distance from the center of mass.

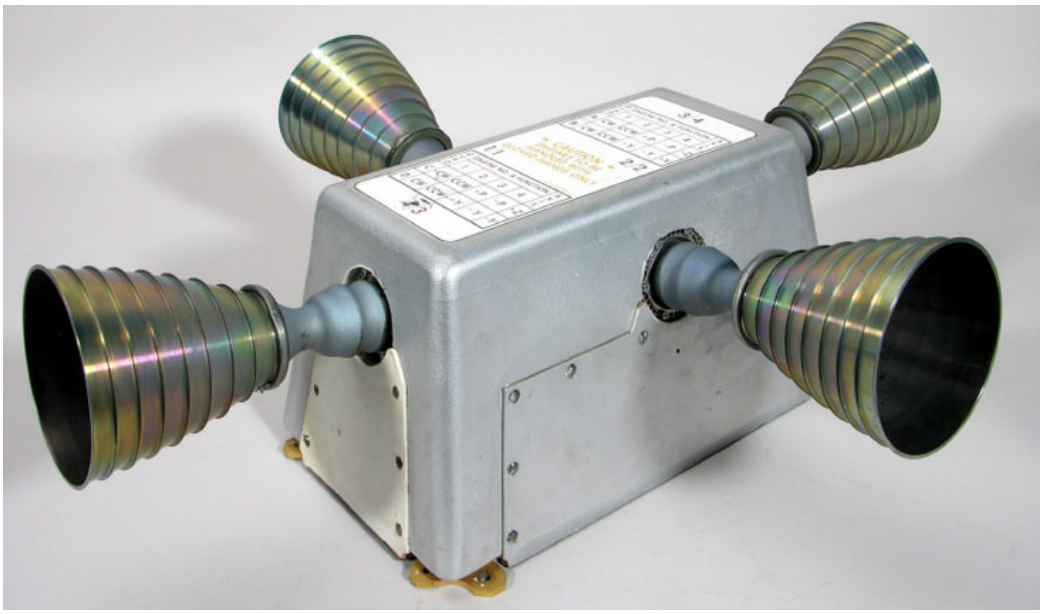

Figure 2: An Apollo service module RCS thruster block

Newton's third law ($\Sigma F = ma = \dot{m}v$) tells us that as the compressed gas exits the nozzle at very high speeds, it imparts a substantial force on the rocket. For thrusters placed far from the center of mass, these forces have a long lever arm and they produce a torque, rotating the rocket. A sophisticated digital system is required to operate the thrusters with great precision and accuracy.

Model Rocketry

My interest in this project also stems from my past experience in model rocketry, and this project coincides with my effort to establish a rocketry club at Swarthmore. For the unaware, model rocketry hobbyists build their own rockets and launch them at launch events which are often hosted in large empty spaces by local rocketry clubs. I flew my rocket in a simple configuration, without the control system project in it, at the Maryland-Delaware Rocketry Association's (MDRA) 278th Eastern Seaboard Launch (ESL #278) on March 13th, 2022. I used a Cesaroni-brand H-class motor with an average thrust of 125 Newtons to fly to an altitude of approximately 900 feet.

Figure 3: Swarthmore Rocketry Club at MDRA ESL #278

The rocket motors are commercially available solid fuel single-use motors. Ammonium perchlorate composite propellant (APCP) is the fuel of choice, and it's the same propellant as what's used in the Space Shuttle and Space Launch System (SLS) solid rocket boosters. Most use a relatively small amount of propellant compared to the volume of the rocket's airframe, and they tend to burn to completion in 2 seconds or less.

After the motor burn ends, the rocket coasts upwards, and shortly after reaching apogee, the motor will detonate a small gunpower pellet which pushes a parachute out of the rocket, enabling safe recovery.

Figure 4: A particularly violent ejection deploying parachutes

Theory

Aerodynamic Stability

Stable flight is a desirable characteristic because it reduces drag and minimizes stress on the airframe and payload. For a rocket, the condition for stability is that the center of mass be significantly upstream of the center of pressure.

For our purposes, the center of mass is the center of gravity, which is the place in the rocket about which it can be balanced in a uniform gravitational field. The center of pressure is harder to define, it is typical to use empirical methods or simulations to find it. We can think of it as the average point through which aerodynamic forces such as drag and lift act.

Having the center of mass above the center of pressure creates a stabilizing effect by the following mechanism: Consider a rocket with fins at the rear ascending vertically. The aerodynamic forces are equal on each fin, and the rocket continues to move vertically. Now consider a gust of wind which blows against the rocket from the side. Since the fins have a larger cross-sectional area, the wind will produce a greater torque where it presses against the fins. The rocket will begin to rotate and turn into the wind. This appears to be an undesirable effect, but since the rocket is moving vertically at great speed, as the rocket begins to rotate into the wind, the fins will stick out on one side and be shielded by the rocket's body on the other. The fins that have been pushed out into the airstream generate a large correcting torque with a greater lever arm which uses the rocket's forward velocity to counteract the destabilizing push from the wind. This type of aerodynamic stability can be passively achieved by using fins, which pull the center of pressure backwards, behind the center of mass.

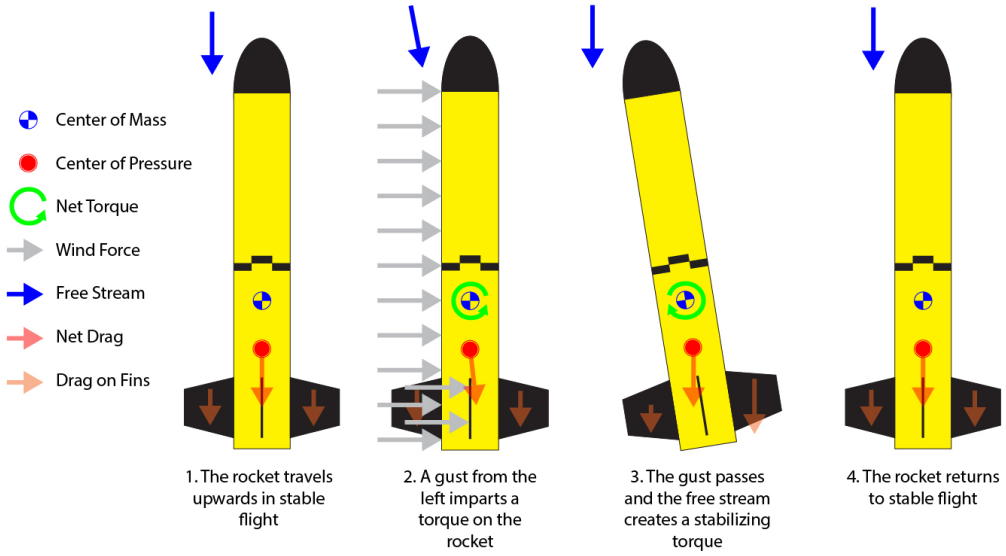

Figure 5: Passive Aerodynamic Stability

In the aerospace industry, rockets are often not passively stable but instead rely on complex digital control systems and active stabilizing methods like thrust vectoring or reaction wheels. This eliminates the need for fins, which reduces the mass of rockets greatly and minimizes drag. This means that the center of mass is really close to the center of pressure, and without the control systems working, the rocket would not be stable in flight! The purpose of the control systems are to make the margin of stability as wide as possible. These are the benefits that I aim for this project to bring to model rocketry.

Nozzle Design

One of my original interests in this project was the design of the nozzles. In a rocket engine, the nozzle converges, forming a throat of small cross-sectional area through which flowing high-energy gasses speed up to become supersonic. After the converging section, there is a diverging section through which the fast, high-pressure gasses gain speed and lower their pressure.

To find the theoretical speed of air out of the pressurized air tank, we can use Bernoulli's Equation:

$$p + \frac{1}{2}\rho v^2 + \rho g h = \text{constant}$$

There is a caveat. Air is compressible and Bernoulli's equation is for incompressible fluids, but since we are dealing with small pressures, this may serve as a good approximation. The experiment takes place at a constant altitude, so h remains constant throughout the experiment. $p_1 = 621$ kPa, $p_2 = 101$ kPa, $v_1 = 0$ m/s, $\rho_1 = 7.4$ kg/m^3, $\rho_2 = 1.2$ kg/m^3 This allows us to solve for the velocity of air exiting the tank:

$$v_2 = 29.4 \text{ m/s}$$

This ends up being about Mach 0.1 when no nozzle is used and the high-pressure air is simply released into the atmosphere. When introducing a converging-diverging nozzle, this method will no longer be useful. Wolfram Alpha has a useful calculator[1] for converging-diverging (de Laval) nozzles. Assuming $T = 273$ K and the molecular mass of air equal to approximately 29 g/mol and a heat capacity ratio is $\gamma = 1.4$ and an exhaust gas pressure of 1 atm and inlet gas pressure of 100 psi, the calculator claims a air speed of 480.7m/s, which is Mach 1.4!

In reality, I expect the true airspeed out of my nozzle to be somewhere between these two values. In my experimentation, I believe anything above Mach 1 to be unreasonable, but I was unable to make any direct measurements of the airspeed.

[1] de Laval Nozzle Calculator

Design

Early on in my design process, I made some big decisions about the scope of the project. I decided to use compressed air as a propellant so I could do a lot of testing very easily by hooking my system up to an air compressor. Also, compressed air is a lot less dangerous to work with than fuels and oxidizers. With combustion, there are problems of acquiring, storing, cooling, mixing, and igniting. The system is necessarily more complex with separate systems for fuel and oxidizer needed. Working with compressed air brought up the least associated problems. I used the Swarthmore engineering department machine shop's air compressor, which is capable of achieving pressures of up to 100 psi, to do all of my ground testing. For a flight test, I would have to get a large tank capable of storing air at 100 psi, or a smaller tank which is able to store air at higher pressures, with a regulator to bring the output pressure down to 100 psi.

I quickly chose to use 4 nozzles placed at right angles because I was constrained by the volume of my rocket's airframe and I thoguht fitting more than 4 in would be unreasonably difficult. 6+ nozzles could theoretically give me better control, and also allow for me to correct roll with the correct nozzle placement, but yaw and pitch are the main concerns for determining the course of the rocket, and since the nozzles fire so briefly, roll can be considered insignificant for reasonably small angular velocities of roll.

The Rocket

In many ways, the design of my rocket is very standard for a model rocket, which was intentional. I wanted to have a base upon which to integrate my system that was simple and familiar. For those unfamiliar with model rocketry, I will discuss my design choices here in full.

The design process began with simulation. I opted to use OpenRocket[2], which is free and open-source software for model rocket design and flight simulation. I started by choosing an airframe for the rocket, and I chose 5.5 inch diameter Blue Tube[3]. The material is a vulcanized fiber tube, similar to cardboard tubes smaller rockets use, but much stronger and able to withstand greater forces without crumpling. The diameter I chose was rather large, but I wanted to pick something with plenty of space for the system hardware.

[2] https://openrocket.info/

[3] Blue Tube body tube 5.5 inch diameter

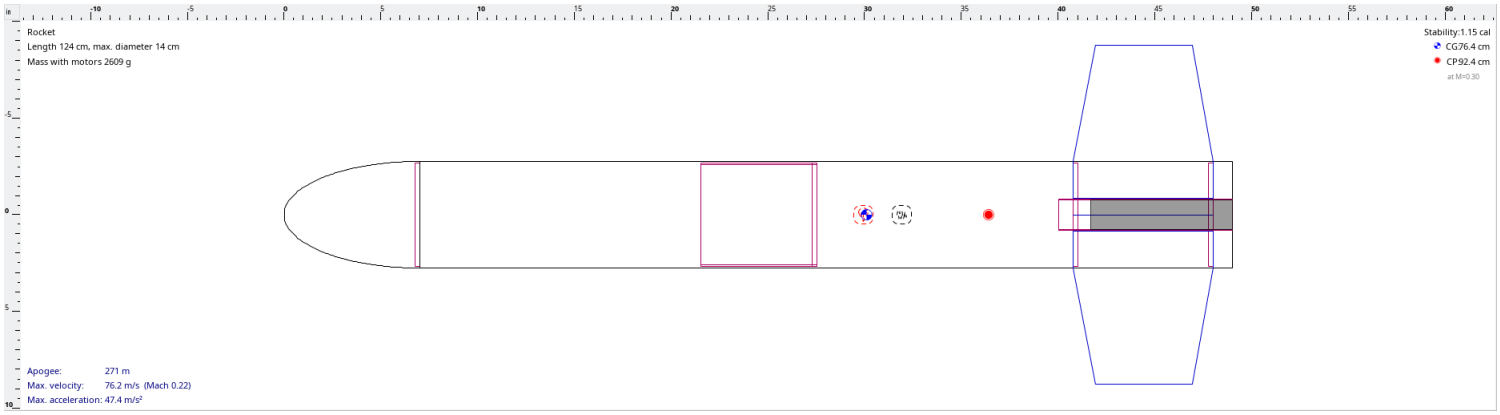

Figure 6: OpenRocket Schematic

Next, I chose a motor tube, and I again went with Blue Tube, but in a 38 millimeter diameter[4]. A smaller diameter motor mount would not support powerful enough motors to lift such a large rocket, but motors at the next size up would be too powerful, and I was looking for a slower, steadier ascent to allow for more time for my control system to react.

To give the rocket passive stability, I added 4 trapezoidal fins. I chose the trapezoidal shape to make the rocket more likely to land on the tail when it is recovered. Landing on the corner or edge of the fin can often break the fins if they protrude too much. The size of the fins was chosen using OpenRocket's center of pressure calculations to achieve a stability factor of 1.15 caliber, where the acceptable region is between 1 and 3 caliber. I chose this nearly-marginal stability factor because I knew that adding my control system near the nose of the rocket would move the center of mass towards the top by a significant amount, and as designed, the rocket remains stable with up to 2 kilograms of extra mass at the base of the nose cone, which is the ideal location for my thruster block to be placed.

I made the fins myself at the campus makerspace out of 1/4 inch plywood on the laser cutter. The center of the fins is made to be cut out, leaving a gaping hole in the fin that I then covered with shrink-wrap. This resulted in a mass savings of 60 grams per fin, and nearly halved the mass of the fin assembly as a whole, which did a lot to keep the center of mass higher and increase stability. The tab on the inside edge of the fin is where the fin

[4]Blue Tube motor mount tube 38 millimeter diameter

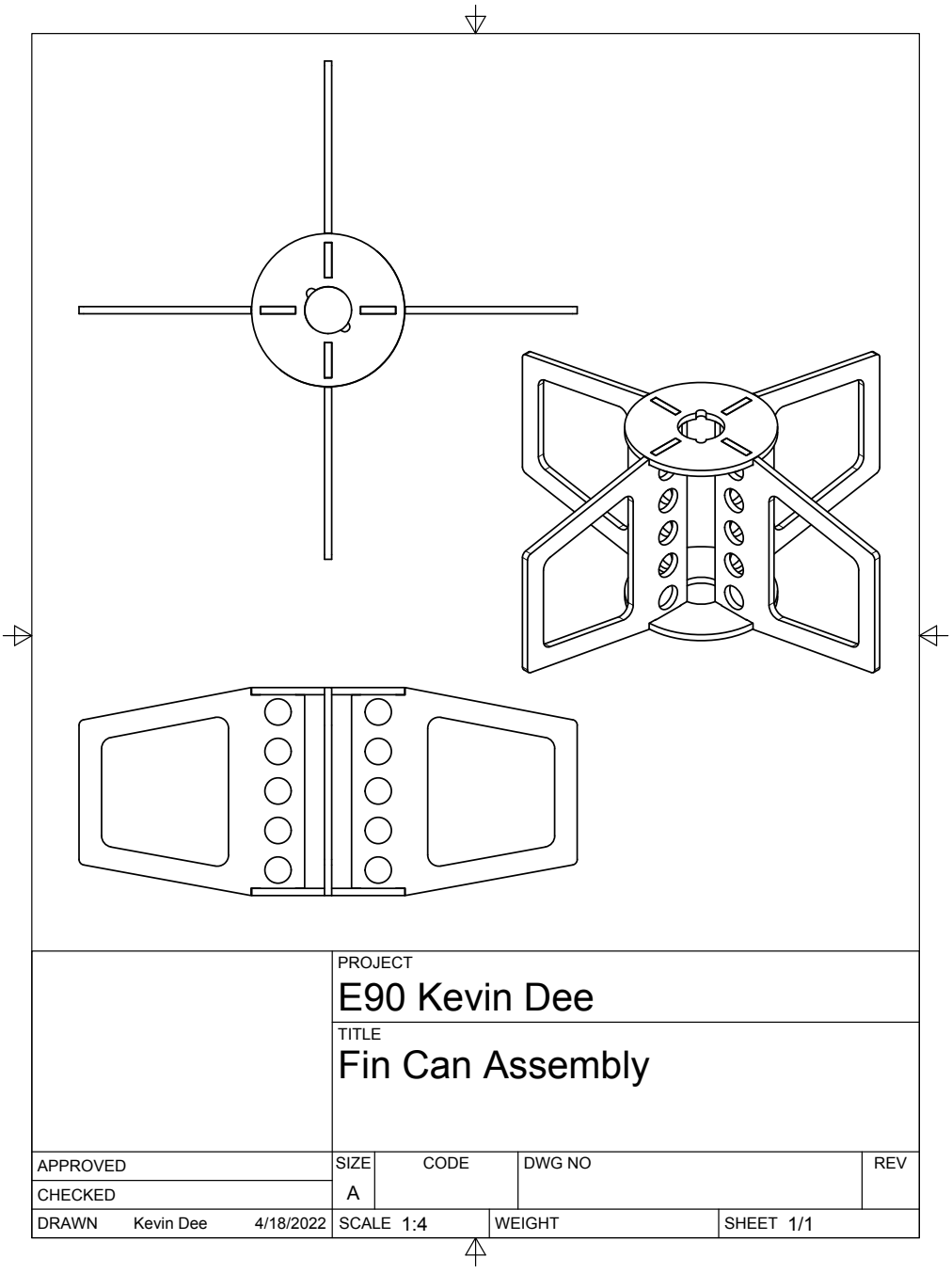

Figure 7: Fin Can Assembly

gets attached to the motor mount and centering rings. The centering rings are also of my own design, they are laser-cut from the same plywood as the fins, and they center the motor mount in the airframe and provide lots of surfaces on which to glue the fins into the rocket. A strong kevlar cord is also glued into the centering rings to serve as an anchoring point for the recovery parachute.

Halfway up the rocket, I split the airframe in two. The bottom section, referred to as the booster, has empty space between the fins and the center which reserved for the parachute and other important recovery hardware. The top half is designated the payload section, and it has lots of empty space for the thruster system. The two halves are separated by a solid laser-cut plywood bulkhead, but they are tied together through the bulkhead so that they come down on the same parachute. The payload is fitted into the booster by use of Blue Tube couplers, which fit snugly into the airframe tube. At the end of the motor's burning, the ejection charge separates the rocket by pushing the payload section out of the booster, and this also deploys the parachute.

The nose cone is of my own design. I made it on a campus makerspace 3D printer out of PLA plastic. It is elliptical in form, and is about 6 inches long. I drilled two holes into the coupling section of the nose cone and epoxied in two nuts, then drilled two corresponding holes into the airframe. This allows me to use bolts to secure the nose cone onto the top of the rocket, which is important because it keeps the payload hardware contained during flight, but it's easy to remove on the ground if work needs to be done inside.

For the recovery system, I chose a FruityChutes 48 inch diameter ellipitical parachute[5] and I used a FruityChutes 18 inch Nomex parachute protector[6] to insulate it from the flames of the ejection charge. I chose a parachute of this diameter based on simulation predictions and suggested mass and parachute diameter pairings from the manufacturer.

Lastly, I had to choose which motors I should fly my rocket on. I chose to use Cesaroni Technology Incorporated[7] motors since they have a lot of options, they are generally easy to find in-stock at multiple online vendors, and they have a very simple motor design, which makes for easy assembly and preparation. Since my motor mount was 38 millimeters, I was restricted

[5] FruityChutes 48 inch diameter elliptical parachute

[6] FruityChutes *X* Nomex parachute protector

[7] `http://cesaroni.net/`

to motors of that diameter, and due to the length of the motor tube, I was restricted to motors that are no longer than 3 propellant grains. For the inaugural flight, I chose a H125[8], which is a 38 millimeter 2-grain motor that uses Cesaroni's "Classic" propellant.

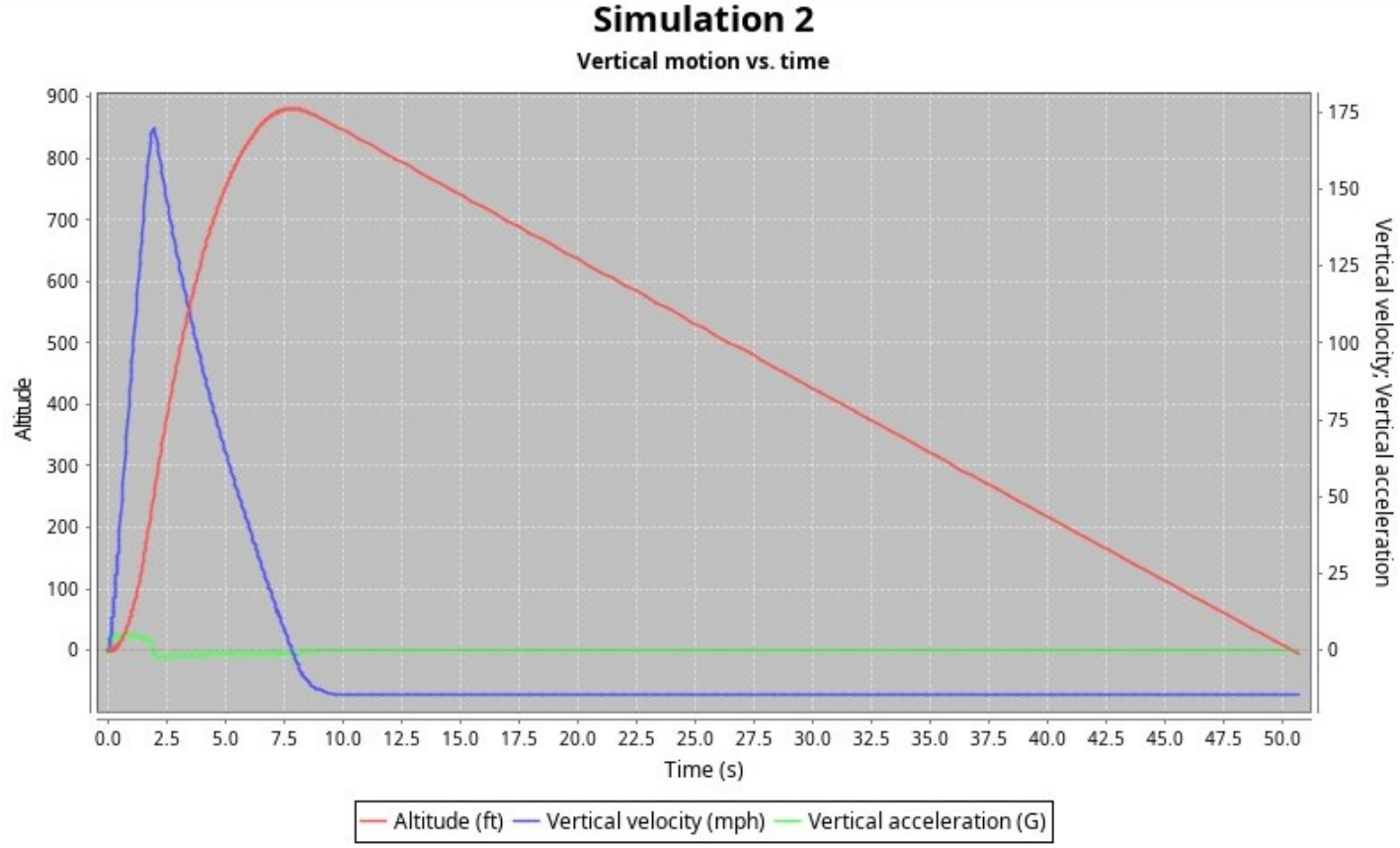

Figure 8: Flight simulated in OpenRocket with an H125 motor

For this test flight, my rocket was very conventional. The control system was still in development, so I flew it like any other model rocket. The flight went well, no damage occurred. I did not have an altimeter on board, but the simulation in OpenRocket suggested an apogee around 875 feet. The test flight itself went well, the rocket flew with stability on a nearly-vertical path, and I observed a very slow roll rate during the ascent.

[8] H125 motor data and thrust curve

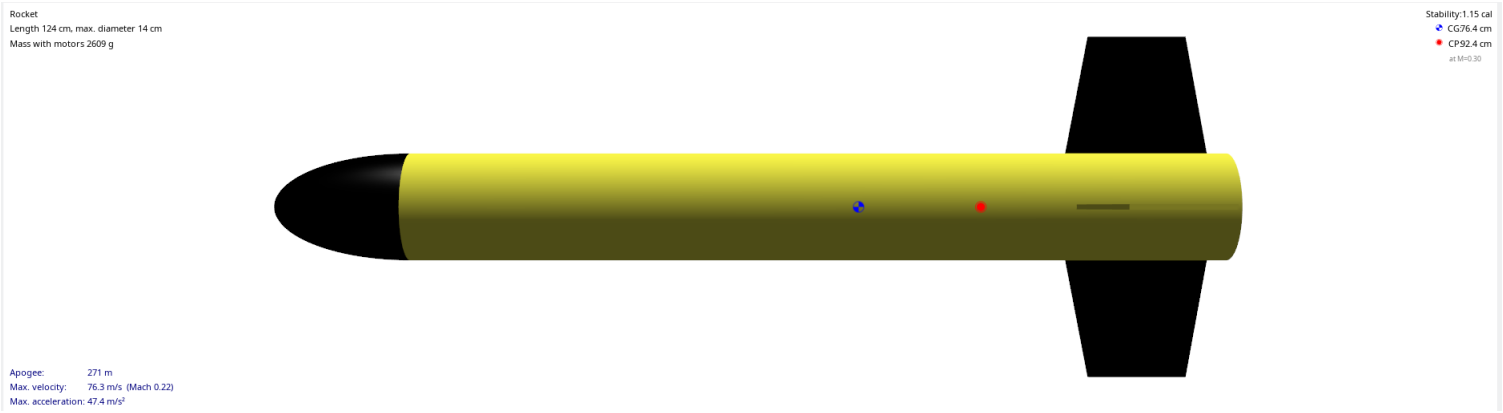

Figure 9: OpenRocket Finished Model

The Electronics

I designed the electronics system with a few simple goals in mind:

- Operable using on-board power
- Programmable in Arduino
- Able to control 4 valves
- Able to determine vehicle orientation from sensors

For the Arduino, I chose Sparkfun's SAMD21 Mini Breakout board[9] for its 32-bit processor, fast clock speed, generous number of I/O pins, and low power consumption. To determine the rocket's orientation, I chose Sparkfun's 9 Degree of Freedom Inertail Measurement Unit (IMU) Breakout board[10]. I picked it for its ability to measure rotational speed and acceleration in three axes, as well as measure magnetic fields in three axes, giving it the ability to use the Earth's magnetic field as a reference frame.

I started by using a breadboard to test the parts in isolation, powering the Arduino and actuating the valves. I made note of every piece's voltage and current limits, and I found by experimentation that the 12V valves were actually operable down to 6V with minimal changes in their response time. With this information in mind, I purchased a few 3.7V 2000mAh batteries to

[9] Sparkfun SAMD21 Mini Breakout
[10] Sparkfun 9DoF IMU Breakout (ICM-20948)

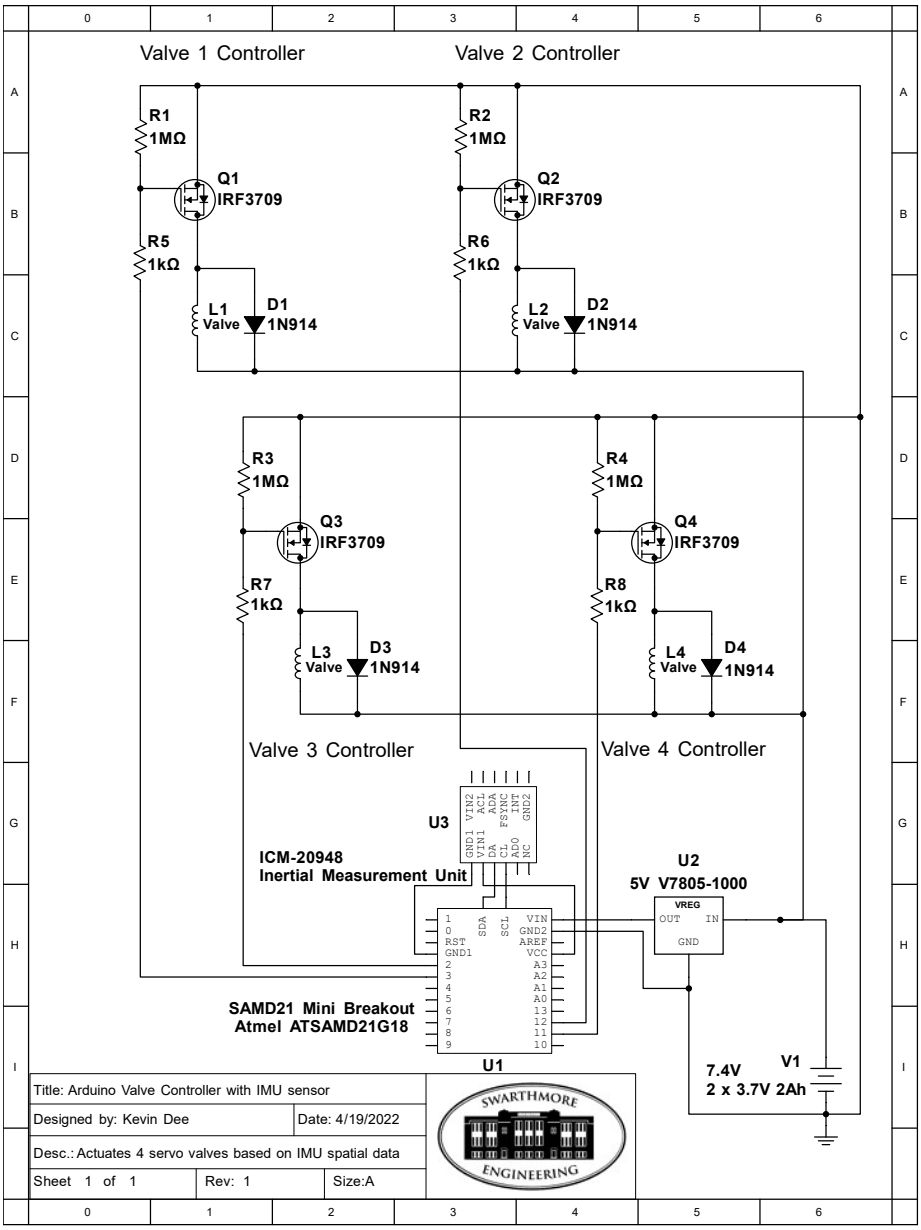

Figure 10: Electronics Schematic

use to power the system. With 2 in series, I was able to make a 7.4V source, enough to operate the valves even with some depletion.

The Arduino operated on 3.3V logic levels, and the IMU used 1.8V logic, but thankfully the Arduino was able to accept up to 5V inputs and the IMU was able to accept up to 3.3V inputs, so I used a simple 5V DC-DC buck converter to bring the 7.4V battery supply down to 5V to power the Arduino, then I used the Arduino to power the IMU at 3.3V. With this system, I was able to power all of the boards at the appropriate level and still have high enough voltages to operate the valves while using just one power source.

Knowing that I had all of the power electronics figured out, I started designing a circuit capable of controlling a single valve. I put the battery voltage on one side of the solenoid valve and a IRF2709 Power Transistor[11] drain on the other. The source of the transistor was wired to ground, and the gate was connected to ground by a 1MΩ resistor and to an I/O pin on the Arduino by a 1kΩ resistor. These resistors were added to prevent an output from the Arduino at 3.3V from being directly connected to ground. Putting a significantly larger resistance between the gate and ground than between the Arduino and the gate kept the voltage at the gate higher, if the voltage was too low there, the transistor would not turn on. I also added an 1N194 diode[12] going from the unpowered side of the solenoid valve back to the battery, knowing that if the valve were on for some time, magnetic fields in the solenoid would continue to induce currents through the solenoid briefly after the transistor turns off. The diode provides a way for the current to flow, which prevents nasty voltage spikes. After completing this design on the breadboard for a single valve, I scaled it up to control 4 valves and I started soldering everything onto a perfboard small enough to fit inside the rocket.

[11] IRF2709 Power Transistor
[12] 1N194 Diode

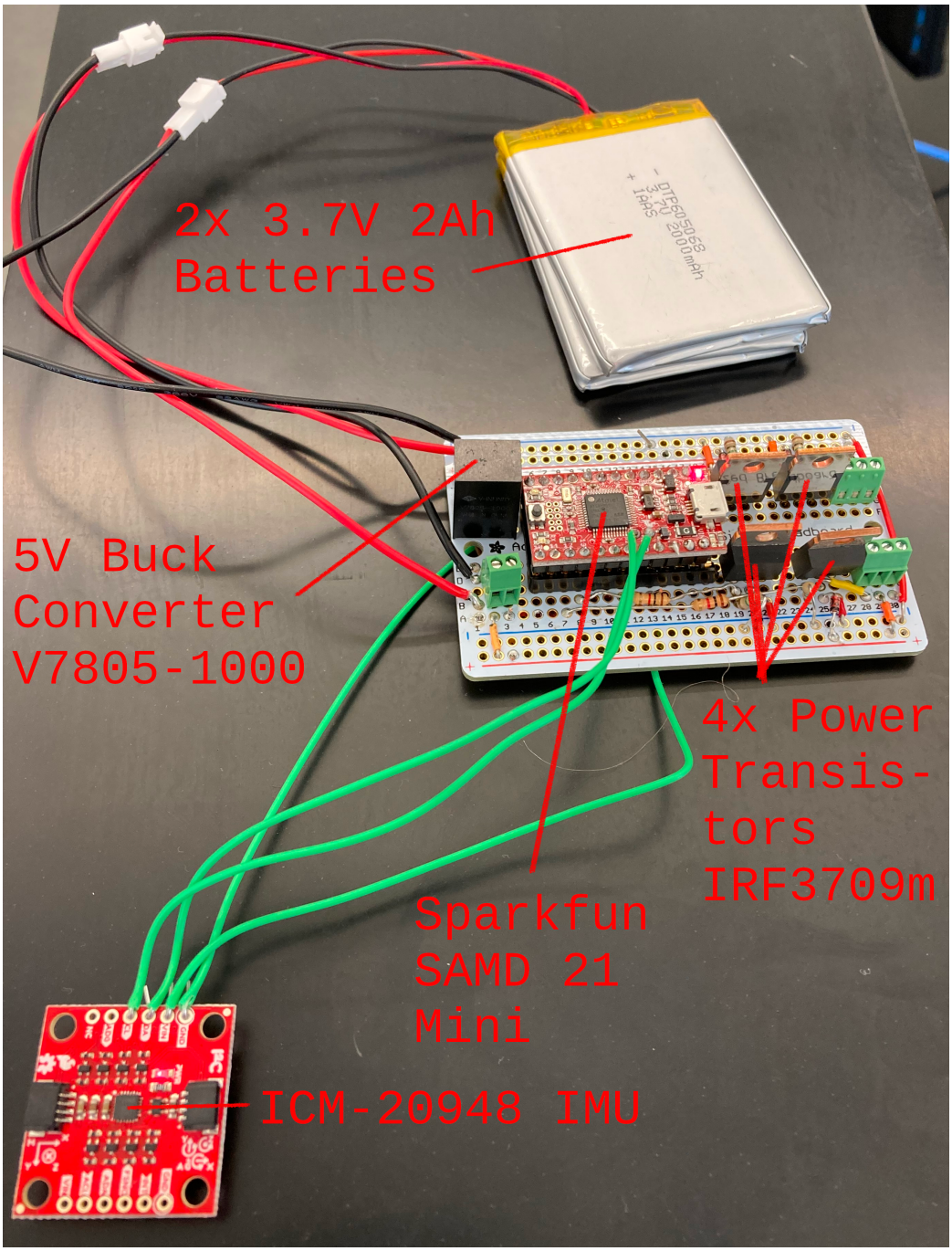

Figure 11: The complete electronics package

The Pressure System

The physical system that I made had to be able to deliver compressed air through valves into nozzles, be lightweight, and fit inside my rocket.

I chose plastic water solenoid valves from Adafruit[13] to control the flow of compressed air to the nozzles because they are rated for up to 116 PSI and they are super cheap and rather small. Most of the rest of the design in this part revolves around these valves and the planned integration of the whole system into the rocket.

I had to make pipes which split from the compressed air source, so I designed a part to be 3D printed which would take a $^3/_4$ inch brass pipe fitting and split it into four $^1/_2$ inch PVC pipe fittings, the correct size for the valves. The piece was made to fit into the cylindrical 5.5 inch airframe of the rocket. I used epoxy to glue the valves into their connections here, then moved on to the part on the outlet end of the valves.

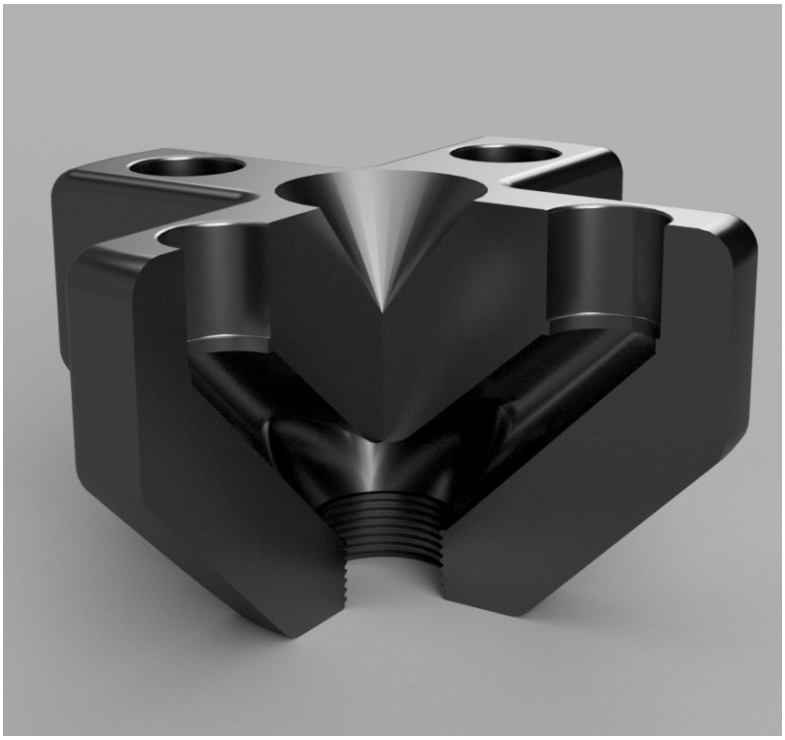

Figure 12: Quarter-sliced view of the gas manifold

[13] Adafruit Plastic Water Solenoid Valves

The final piece in this assembly had to be able to take all 4 separate valve outputs and deliver the propellant through them into 4 nozzles pointed radially out of the rocket at right angles to each other. I could have made 4 separate pieces, but to save space, I 3D printed a single part with common walls of the pipes wherever they met. Each pipe snakes around from the valves towards the central axis of the rocket and then back outwards to a small chamber immediately before the converging-diverging nozzle.

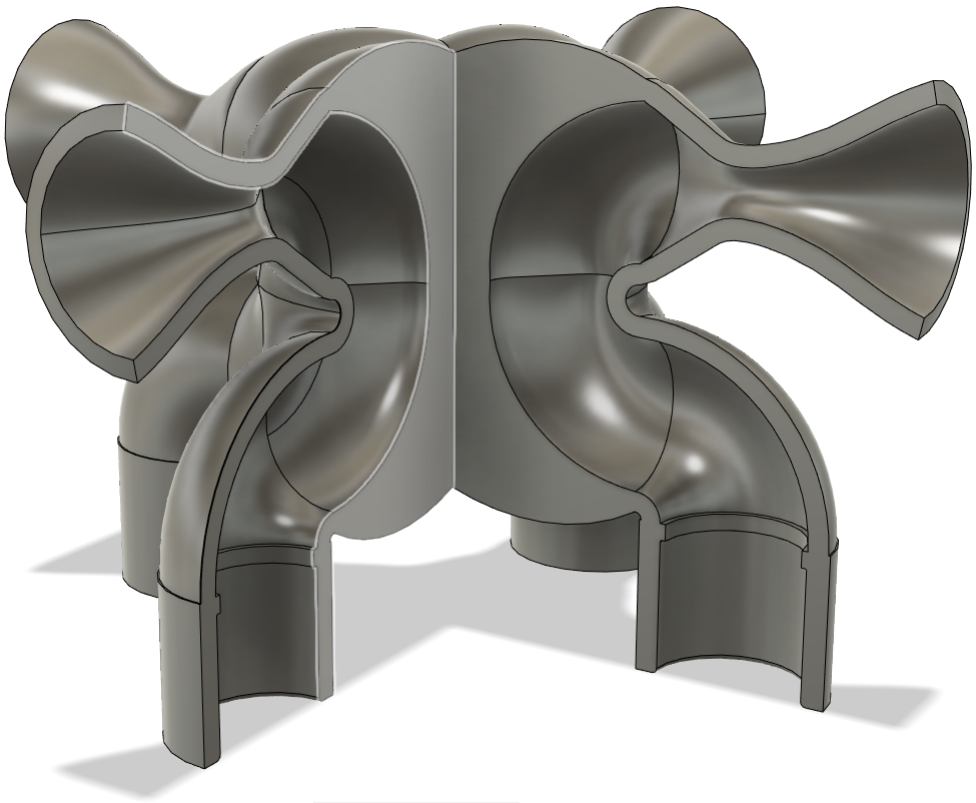

Figure 13: Quarter-sliced view of the nozzle assembly

The first assembly of this system was done entirely in PLA and the gas manifold failed under pressure at about 30 PSI, so I remade it using Onyx[14], a type of carbon-fiber composite with fiberglass reinforcement.

With the manifold printed in Onyx, it held under pressure when it was tested at up to 100 PSI, so I assembled the whole system and test-fired the valves. At this point, I made measurements of the force a single engine could impart at various chamber pressure levels.

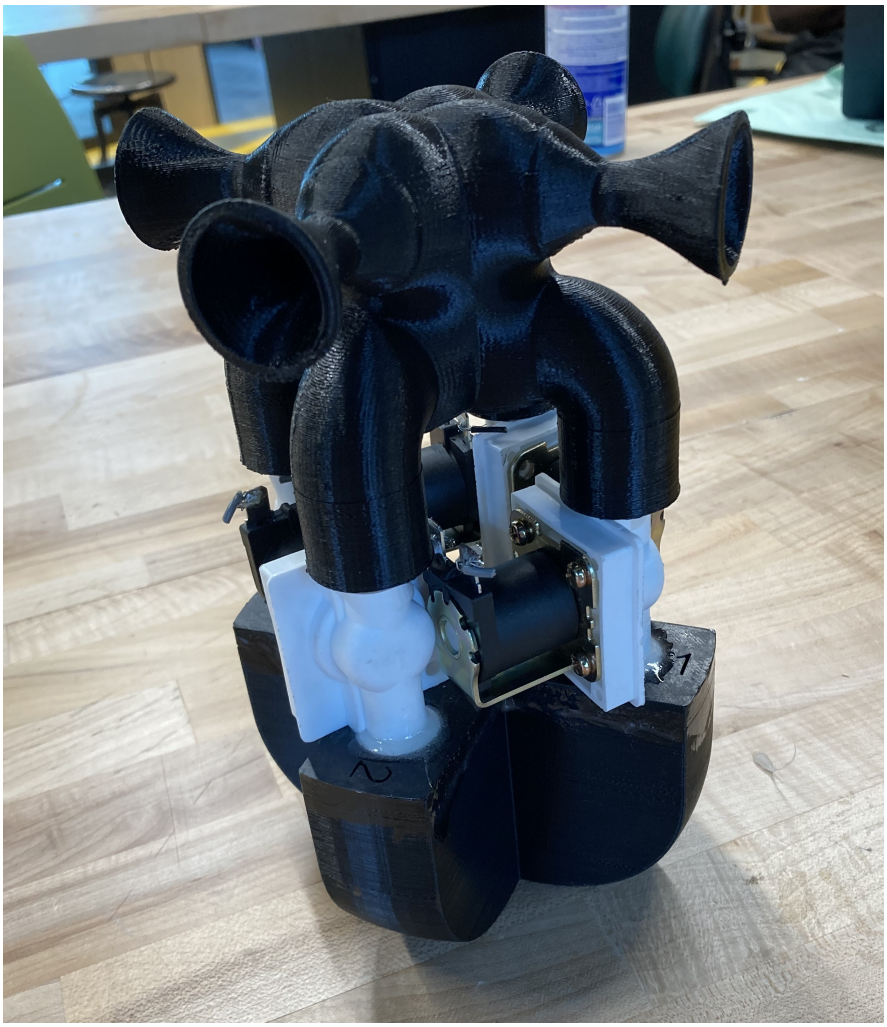

Figure 14: Propellant System Assembly

[14] https://markforged.com/materials/plastics/onyx

Integration

Placing the system into the rocket had some challenges. As discussed previously, my simulation suggested a mass limit of 2 kilograms for my entire system. In the ground testing configuration, the system only massed 1600 grams, but this likely means that the system would be slightly over the mass budget with an appropriately sized compressed air tank attached.

I inserted the system into the rocket upside-down at first, intentionally, so I could use a hose to supply the system with compressed air directly for testing in the lab. If I were to fly the system, it would be a simple matter of unbolting it from the airframe and reinserting it in the correct orientation so it could be fueled from an internal tank.

Control System

The coding is all done in Arduino, and it can be read in full in Appendix A. The code that I started with comes from the example code bundled in the library for the ICM-20948 IMU. The example which was most helpful to me gave the orientation of the IMU in Euler angles at a high refresh rate. Then, all that remained for me to do is devise a control system to use yaw, pitch, and roll data to effectively arrest divergent motion.

I started with a simple idea: if yaw or pitch exceed a threshhold, pulse a valve open for a preset duration to correct it. This is a good starting point, but it is rather primitive. It is clear to me that a more effective, elegant system would rely on other criteria in addition to angular displacement, such as angular velocity and acceleration, and it would fire the thrusters for variable durations based on how significant the rocket's divergence from the vertical trajectory is. The Arduino might struggle with performing more complex calculations on the fly, but by doing calculations beforehand and using lookup tables, computing needs could be significantly reduced.

Results

To get force measurements, I placed the system on a digital scale before integrating it into the rocket, and I fired the nozzle continuously for 3 seconds, and I read out the final value on the scale. I did this for pressures from 20 PSI to 90 PSI in intervals of 5 PSI.

There was significantly more thrust immediately after the valve opened than there was a half-second or so after the valve opened. More thorough testing with faster equipment would be able to establish a thrust profile for a nozzle firing at various pressures.

The results are as follows:

P (psi)	T_{Peak} (lbf)	T_{Steady} (lbf)
20	0.0625	0.03125
25	0.1125	0.05625
30	0.1625	0.075
35	0.2063	0.08125
40	0.2563	0.1
45	0.3438	0.125
50	0.425	0.1875
55	0.5063	0.225
60	0.55	0.2375
65	0.5813	0.2
70	0.8125	0.2563
75	0.6188	0.2063
80	0.7688	0.2438
85	0.8125	0.225
90	0.9375	0.3813

Table 1: Pressure and Thrust Data

As expected, there is a positive correlation between propellant pressure and thrust. The steady state thrust trails the peak thrust by a factor of about 1/3. Since the rocket only weights about 6 lbs without the system installed, the thrusters are more than capable of of generating enough torque to right the rocket to vertical from significant angular displacements.

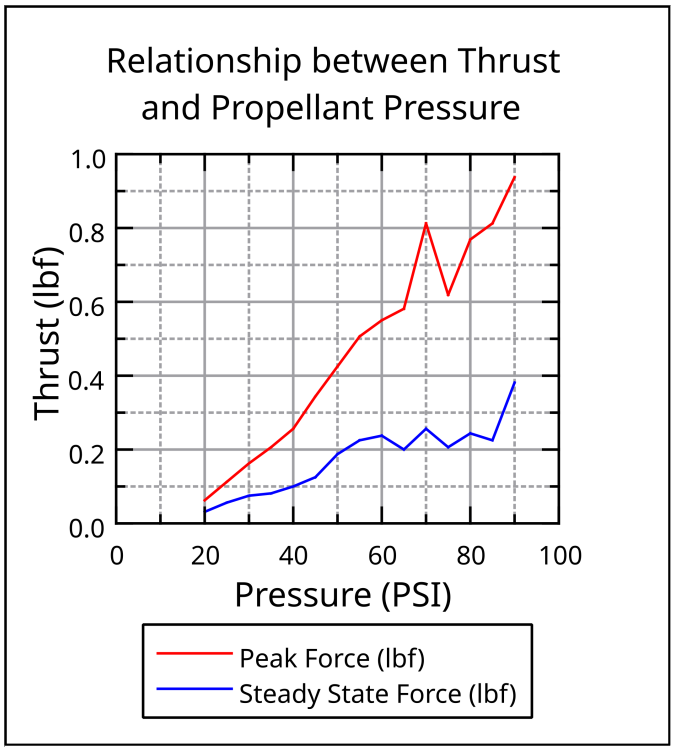

Figure 15: Pressure-Thrust relationship

After integrating the system into the rocket, I conducted tests by holding the rocket vertically and leaning it from side to side, noting how the nozzles responded. In the future, I wish to make a gimbal mount which holds the rocket at the center of mass and allows for free rotation in pitch and yaw (and optionally, but not critically, roll). Then I would let the rocket swing freely on the mount after an initial perturbance, and see if the system is capable of correcting it or not.

I imitated this type of test by hanging the rocket from the air hose that supplied the system and tilting it from side to side, observing how the nozzles would release pressurized air to correct the rocket's orientation, in many

cases, overcorrecting it. This overcorrection could be solved by reducing the working pressure or by implementing a more accurate control system

One of the biggest problems I uncovered in testing is that the sensor would operate significantly more slowly when the valves were open. My program would hold the valves open for a set duration, but this programmed delay was warped because the sensor had a lag in delivering data to the Arduino while valves were open. This resulted in a programmatically-induced stickyness of the valves. To solve this, I would have to redesign the electornics to move the valves to a separate power supply.

If these problems could be fixed, the system would be ready for a flight test. On the first test flight, where I just flew the rocket, I observed pretty stable flight and a low roll rate, which indicates a very good use case of my system. For a second flight test, I would have chosen to fly my rocket on a slightly undersized motor, so the rocket would be slow and therefore more susceptible to instability since the passive stability features are less useful at slow speeds.

Discussion

My goal in starting this project was to develop a reaction control system of thrusters for use in a model rocket. The motivation was to find a method of achieving aerodynamic stability besides building fins on the rocket. As I expected from the start, overwhelmingly from the perspective of complexity and cost, fins remain a much better choice for hobbyists who want to fly straight. Implementing a reaction control system is overengineering the problem, since model rockets generally only care about flying straight, but reaction control systems were developed in rockets for precise orbital insertions and maneuvers. With these facts in mind, let's consider some of the successes of this project.

The fact that I was able to succeed in building a fuctional RCS thruster block at low cost (relative to the prices in the aerospace industry) is significant. I successfully proved that such a system can be developed and deployed at such unusually small scales, with minimal computing power and extremely safe propellant, which means such a system could be used affordably in sounding rockets or small satellites.

What this means for model rocketry is that solutions other than fins could be acceptable for designing stable rockets. Hobbyists will know that fins are

often the weak points of a rocket's design since they protrude out from the base of the airframe and are often thin and therefore susceptible to breaking. Fin assembly is generally the most complex and time-consuming part of a rocket assembly process, and fins also add a lot of drag, reducing apogees, so there is certainly a desire for an alternative stability solution in the hobbyist community.

This project proves the feasibility of implementing professional aerospace solutions at the hobbyist level, and I would be interested in seeing if other methods such as thrust vectoring and reaction wheels could be used in model rocketry with success.

Successful implementation of guidance and control hardware in model rocketry could also have a benefit for safety where one could programmatically command a rocket to fly away from the area where spectators are located, or where there are an abundance of trees or power lines. Since one of the biggest challenges in high-power model rocketry is finding an appropriately sized launch site, being able to control where the rocket goes could make smaller launch sites safe options for high-power flights.

Another possibile future for the methods demonstrated in this project is in a propulsive landing system for a model rocket. This idea was actually my original inspiration for this project, but I ruled it out as too risky. I think it could be done with a sufficiently lightweight rocket and much higher propellant pressures. If the relationship I found between pressure and thrust is linear, about 1600 psi could be used to slow down a 4 lb rocket in a well-timed "suicide burn" and bring it to a gentle impact. This would be achievable, using paintball hardware for instance, which stores compressed air at pressures of 4500 psi.

Conclusion

In this project, I identified a rapidly-improving technology in the aerospace industry and implemented it myself on a small scale. I wanted to bring the sophisticated controls used in orbital rockets to a hobbyist level, and I succeeded in some regards, but more refinement is necessary before this can be considered a flight-ready system. Importantly, I learned that there are not physical limitations that prevent this project from succeeding, only that it is very, very hard and continued work to develop the controls fully and conduct multiple flight tests would require lots of patience and perseverence.

Acknowledgements

I want to give credit to Owen Lyke of SparkFun Electronics for the example code for the IMU's digital motion processor, which forms a significant basis upon which my code has been built.

Thank you to my advisor, Professor Carr Everbach, for his help in getting this project started and keeping me on the right track throughout the semester.

I would also like to thank Ed Joudi and J. Johnson for helping me source materials and tools and enabling me to work on this project to the fullest extent. Also, thank you to Jacquelline Tull of the makerspace for her help on this project and in everything the rocketry club at Swarthmore College has done.

Appendix A: Code

```
// E90 Project: RCS system for model rocketry
// Swarthmore College
// Kevin Dee
// 4/18/2022
// Lots of code setting up I2C, serial, ports and debugging comes from Sparkfun's IMUExample1_Basics sketch

// Libraries
#include "ICM_20948.h" // IMU chip library
#include "HeartBeat.h" // Heartbeat LED library
#include "math.h"      // Math functions

// Serial setup
#define SERIAL_PORT SerialUSB

// I2C setup
#define WIRE_PORT Wire // Desired wire port
#define AD0_VAL 1      // The value of the last bit of the I2C address.
ICM_20948_I2C myICM;   // Creates ICM_20948 object

// Heartbeat setup
HeartBeat heart;

// Pins
const int BLUE_LED = 13; // Blue "stat" LED on pin 13 (same as valve 4, do not use!)
const int RX_LED = PIN_LED_RXL; // RX LED on pin 25, we use the predefined PIN_LED_RXL to make sure
const int TX_LED = PIN_LED_TXL; // TX LED on pin 26, we use the predefined PIN_LED_TXL to make sure
const int V1 = 2;
const int V2 = 3;
const int V3 = 12;
const int V4 = 11;

// Variables and constants
const int threshhold = 10; // Angular threshhold at which valves should fire (degrees)
const int firingtime = 50; // Time to open the valves for (ms)
volatile bool inFlight = false;
```

```
void setup() {
  // Start Heartbeat
  heart.begin(RX_LED, 1);

  // Blocking code, waiting for serial to open
  SERIAL_PORT.begin(115200);
  //while (!SERIAL_PORT){};
  //SERIAL_PORT.println(F("DMP enabled!"));

  // Start I2C wire port
  WIRE_PORT.begin();
  WIRE_PORT.setClock(400000);

  bool initialized = false;
  while (!initialized){
    // Connect ICM to I2C
    myICM.begin(WIRE_PORT, AD0_VAL);

    // Print ICM initialization status to serial
    //SERIAL_PORT.print(F("Initialization of the sensor returned: "));
    //SERIAL_PORT.println(myICM.statusString());

    // If ICM isn't ready, try again
    if (myICM.status != ICM_20948_Stat_Ok){
      //SERIAL_PORT.println("Trying again...");
      delay(500);
    }
    else{
      initialized = true;
    }
  }
  //SERIAL_PORT.println(F("Device connected!"));

  bool success = true; // Use success to show if the DMP configuration was successful

  // Initialize the DMP. initializeDMP is a weak function. You can overwrite it if you want to e.g. to change the sample rate
  success &= (myICM.initializeDMP() == ICM_20948_Stat_Ok);

  // Enable the DMP Game Rotation Vector sensor
  success &= (myICM.enableDMPSensor(INV_ICM20948_SENSOR_GAME_ROTATION_VECTOR) ==
```

```
    ICM_20948_Stat_Ok);
77
  // Configuring DMP to output data at multiple ODRs:
79  // DMP is capable of outputting multiple sensor data at
    different rates to FIFO.
  // Setting value can be calculated as follows:
81  // Value = (DMP running rate / ODR ) - 1
  // E.g. For a 5Hz ODR rate when DMP is running at 55Hz,
    value = (55/5) - 1 = 10.
83  success &= (myICM.setDMPODRrate(DMP_ODR_Reg_Quat6, 0) ==
    ICM_20948_Stat_Ok); // Set to the maximum

85  // Enable the FIFO
  success &= (myICM.enableFIFO() == ICM_20948_Stat_Ok);
87
  // Enable the DMP
89  success &= (myICM.enableDMP() == ICM_20948_Stat_Ok);

91  // Reset DMP
  success &= (myICM.resetDMP() == ICM_20948_Stat_Ok);
93
  // Reset FIFO
95  success &= (myICM.resetFIFO() == ICM_20948_Stat_Ok);

97  if (success)
  {
99    //SERIAL_PORT.println(F("DMP enabled!"));
  }
101 else{
    //SERIAL_PORT.println(F("Enable DMP failed!"));
103   //SERIAL_PORT.println(F("Please check that you have
    uncommented line 29 (#define ICM_20948_USE_DMP) in
    ICM_20948_C.h..."));
    while (1){}; // Do nothing more
105 }
}
107
void loop() {
109 // Features (in order of priority):
  // X 1. Heartbeat LED(s)
111 // X 2. Get angular displacement measurements
  // 3. Should wrap values from -180 to 180 smoothly and
    continuously (but hopefully this will never be necessary)
113 // X 4. Implement a basic pulsing correction system based
    on this alone (fire for X seconds after Y angular
```

```
    displacement)
  // 5. Use angular velocity measurements to know how much
    correction is necessary
  // 6. Determine how to consider roll
  // 7. Detect apogee and close all valves permanently
  // 8. Detect launch (use an interrupt from a rise in
    rolling average of vertical acceleration to set a flag)

  // Heartbeat LEDs
  heart.beat(); // Flash LED (not blocking)

  // Read the next frame of DMP data from the FIFO
  icm_20948_DMP_data_t data;
  myICM.readDMPdataFromFIFO(&data);

  if ((myICM.status == ICM_20948_Stat_Ok) || (myICM.status ==
    ICM_20948_Stat_FIFOMoreDataAvail))
  {
    // A valid frame was read, more may be available
    if ((data.header & DMP_header_bitmap_Quat6) > 0) // We
    have asked for GRV data so we should receive Quat6
    {
      // Q0 value is computed from this equation: Q0^2 + Q1^2
    + Q2^2 + Q3^2 = 1.
      // In case of drift, the sum will not add to 1,
    therefore, quaternion data need to be corrected with right
    bias values.
      // The quaternion data is scaled by 2^30.

      // Scale to +/- 1
      double q1 = ((double)data.Quat6.Data.Q1) /
    1073741824.0; // Convert to double. Divide by 2^30
      double q2 = ((double)data.Quat6.Data.Q2) /
    1073741824.0; // Convert to double. Divide by 2^30
      double q3 = ((double)data.Quat6.Data.Q3) /
    1073741824.0; // Convert to double. Divide by 2^30

      // Convert the quaternions to Euler angles (roll, pitch
    , yaw)
      // https://en.wikipedia.org/w/index.php?title=
    Conversion_between_quaternions_and_Euler_angles&section=8#
    Source_code_2

      double q0 = sqrt(1.0 - ((q1 * q1) + (q2 * q2) + (q3 *
    q3)));
```

```
145         double q2sqr = q2 * q2;

147         // roll (x-axis rotation)
            double t0 = +2.0 * (q0 * q1 + q2 * q3);
149         double t1 = +1.0 - 2.0 * (q1 * q1 + q2sqr);
            double roll = atan2(t0, t1) * 180.0 / PI;
151
            // pitch (y-axis rotation)
153         double t2 = +2.0 * (q0 * q2 - q3 * q1);
            t2 = t2 > 1.0 ? 1.0 : t2;
155         t2 = t2 < -1.0 ? -1.0 : t2;
            double pitch = asin(t2) * 180.0 / PI;
157
            SERIAL_PORT.print(F("Roll:"));
159         SERIAL_PORT.print(roll, 1);
            SERIAL_PORT.print(F(" Pitch:"));
161         SERIAL_PORT.println(pitch, 1);

163         // Rudimentary control algorithm
            // Fire the valves in a direction if the rocket tilts
        past a threshhold in that direction
165         if (pitch > threshhold){
              // fire pitchwise -
167           digitalWrite(V4, HIGH); // Valve 1 open
              delay(firingtime);
169           digitalWrite(V4, LOW); // Valve 1 closed
            }
171         else if (pitch < -threshhold){
              // fire pitchwise -
173           digitalWrite(V2, HIGH); // Valve 3 open
              delay(firingtime);
175           digitalWrite(V2, LOW); // Valve 3 closed
            }
177         else if (roll > threshhold){
              //fire yawwise -
179           digitalWrite(V1, HIGH); // Valve 2 open
              delay(firingtime);
181           digitalWrite(V1, LOW); // Valve 2 closed
            }
183         else if (roll < -threshhold){
              // fire yawwise +
185           digitalWrite(V3, HIGH); // Valve 4 open
              delay(firingtime);
187           digitalWrite(V3, LOW); // Valve 4 closed
```

```
      }
189   }
  }
191
  if (myICM.status != ICM_20948_Stat_FIFOMoreDataAvail) // If
    more data is available then we should read it right away
   - and not delay
193 {
    delay(100);
195 }
}
197
```

Listing 1: e90.ino