Student Rocketry: Out-of-Class Learning Experiences from a Year-Long Capstone Project at University

Mr. Tim Drake, Saint Louis University

Tim Drake is a senior undergraduate aerospace student at Saint Louis University. He is the president of the Rocket Propulsion Lab at SLU and is leading his senior design capstone project.

Dr. Srikanth Gururajan, Saint Louis University

Dr. Srikanth Gururajan is an Associate Professor of Aerospace Engineering at the Parks College of Engineering, Aviation and Technology at Saint Louis University. Dr. Gururajan's teaching interests are in the areas of Flight Dynamics and Controls and believes that student aerospace design competitions are ideal avenues for students to express their creativity while complementing the knowledge gained in the classroom with hands-on experience as well as promoting greater collaboration and learning across disciplines. Dr. Gururajan's research interests are interdisciplinary and in the fields of fault tolerant flight control, real time systems, experimental flight testing using small UAS, and the design/development of natural language interaction with drones.

©American Society for Engineering Education, 2024

Work-In-Progress: Student Rocketry – Out of Class Learning Experiences from a Year-Long Capstone Project at Saint Louis University

Every year, teams nationwide participate in rocket competitions such as the Spaceport America Cup [1] or NASA Student Launch [2]. These competitions have various altitude requirements that student-designed and built rockets must reach to qualify. Although most rockets meet the altitude requirement to qualify, they typically overachieve and fly beyond the threshold. Our senior design project aims to design, build, and test a Rocket Altitude Determination and Response System (RADARS) to reach within $\pm$ 50 ft of a given target altitude. To achieve this, my team and I will design, integrate, test, and validate an airbrake control system to decelerate the rocket during ascent using real-time data from onboard sensors.

Saint Louis University (SLU) does not have any courses focused on rocket design; consequently, to complete this project, our team must seek information and learn outside the classroom. This paper will describe the steps we have taken to accomplish this (including where to seek the information, struggles, successes, and lessons learned) and document the process to serve as a roadmap for other student teams in similar situations at their universities.

Predicting how rockets behave in flight can be challenging; however, it can be done using wind tunnel tests, Computational Fluid Dynamics (CFD) simulations, and physics-based dynamics simulations. Extensive topical knowledge is required to perform each of the tasks successfully. Moreover, rocket launches and flight tests require additional knowledge of procedures, operations, and safety considerations. This is readily available at many academic institutions with established experimental rocketry programs through coursework and institutional knowledge but is lacking at our institution due to many factors, COVID being a prime contributor to the loss of institutional knowledge.

The student-run rocket lab at SLU used to be highly active, with students at all levels participating in rocket design, building, and tests; unfortunately, most of that information and experience was left with the graduating class in 2020. When I arrived the following year, the lab could not meet due to COVID restrictions. The lab remained inactive until this past summer when I contacted some alumni previously in the rocket lab at its prime about wanting to restart the lab. Our early meetings consisted of them teaching me the basics of rocketry. I quickly learned the essential components of a rocket by building a LOC-IV cardboard kit [3] ordered online. With guidance from the alums, I picked up old tips and tricks they used and insights into constructing much larger competition rockets. Parts such as bulkheads, couplers, motors, fins, and a nosecone are in all rockets, no matter the complexity, and are made with different materials and precision – depending on the mission. Larger rockets also require a specific certification level to launch them due to the larger motors.

High Power Rocketry (HPR) certifications are granted by the National Association of Rocketry (NAR) [11] and the Tripoli Rocketry Association [12]. Three levels of certification correspond to the motor class. Rocket motors are designated on an alphabetic scale: A-G motors do not require any certification for their operation; H and I need a Level 1 HPR certification; J, K, and L require a Level 2, and M, N, O require a Level 3. A typical progression is from no certification through Levels 1, 2, and 3. To receive a Level 1 HPR certification, one must launch a rocket built from a kit or custom-built using an H or I motor. Once I received my motor, I was introduced to a program

called OpenRocket [4]. OpenRocket is an open-source software program designed to simulate the flight and dynamics of custom-designed rockets. The program creates a 2D rocket model based on components added by the user. For instance, users can incorporate motor configurations from a commercial off-the-shelf (COTS) motors list. From this, the software simulates the flight of the rocket with seemingly reasonable accuracy – our firsthand experience in validating this claim of reasonable accuracy is still under development and was one of the skills we lost during the COVID era.

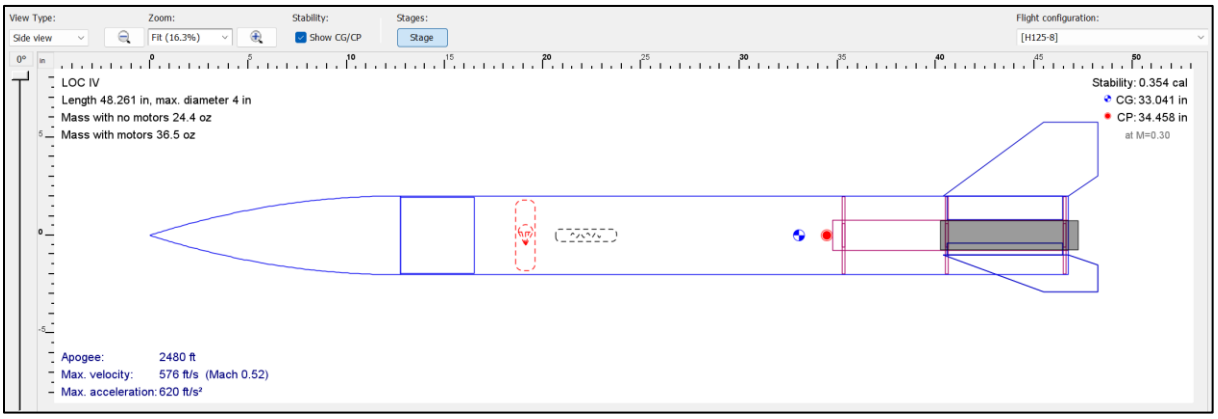

Figure 1: LOC-IV OpenRocket Model

The program takes each component's weight and aerodynamic properties and calculates the center of gravity (CG) and center of pressure (CP). These two values can then be used to calculate the rocket's Stability Margin (SM). This unitless number is an indicator of the stability of the rocket's flight - the higher the number, the less likely the rocket will weathervane or be susceptible to wind gusts. It is calculated by subtracting the distance of the CG (blue dot in the sketch in Figure 1 above) and CP (red dot in the sketch in Figure 1 above) to the nosecone and dividing by the diameter of the body tube. By measuring the components from the LOC-IV kit, I modeled the rocket in OpenRocket. I assembled the rocket as if it were ready to launch and then weighed and measured the CG location. The option to override the default values in OpenRocket allows for a more accurate simulation of how the real rocket will fly.

I did not retain much of this information the first time I heard about it. However, the consistency of working in the rocket lab and meeting with alumni strengthened my knowledge. By the beginning of the fall 2023 semester, I had gathered enough knowledge (and confidence to run the student club) to recruit other students to get the lab on the right track. The same approach from the summer was used when teaching new members of the rocket lab. More LOC-IV kits were purchased and used to explain the basics of rocketry. We set up meetings with the alums to ensure I was teaching the correct information correctly, but mostly to pass down the alums' experience to a larger population. New members asked the alum questions I never thought to ask, enabling me to learn even more. We started attending launch events in the local community once enough members built their Cert 1 kits. These events were day-long trips to remote farm fields with temporary no-fly zones and taught us the practical aspects of flight preparation, operations, and safety.

Here, we launched and received our Level 1 HPR certifications. Attending these events enabled our lab to experience firsthand what the simulations of stability margin, impulse, recovery, and other factors meant. Not only did we see our work pay off, but we realized what we could build

next. We experienced small A-level rockets made by kids and O-level rockets breaking the sound barrier built by experienced university clubs. These new opportunities also enabled my senior design group to understand what was required to design, construct our rocket, and implement an altitude control system.

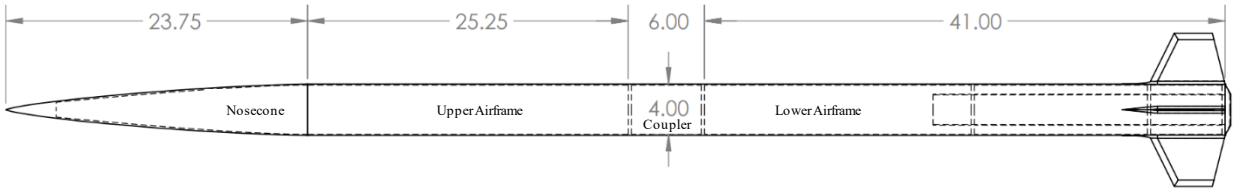

Figure 2: Rocket Component Design (all dimensions are in inches)

Figure 2 displays the final rocket dimensions used within each section. Additional details on the design choices that led to this configuration can be found in our recent AIAA paper, *Project RADARS Rocket Altitude Determination and Response System*. [13]

Our first step towards this goal was to explore the use of the open-source software program OpenRocket. Among other results, OpenRocket can estimate the stability margin of a rocket and how that affects its flight; this information is used to evaluate the placement of the airbrakes in the rocket as it changes the stability margin. As mentioned, the software is mainly developed for a niche segment and has advantages and shortcomings. The significant advantage is that it is open source – meaning it is free and provides access to its source code. Resource-strapped student teams, like ours, can leverage this to analyze preliminary designs. It also has a vibrant community of users and developers and can be a knowledge resource. On the other hand, a potential downside is that a volunteer user community develops this, and updates to the software and documentation happen on their schedule. For instance, as of this writing, the user manual was last updated in February 2022. We have not yet encountered issues with this, but we anticipate doing so!

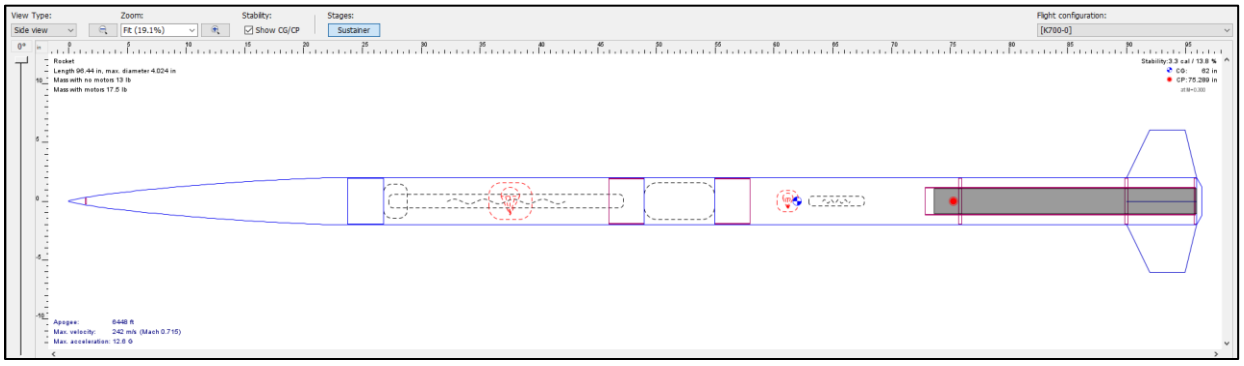

Figure 3: R.A.D.A.R.S OpenRocket Model

OpenRocket does not allow a wide range of structures, such as airbrakes. Our team explored the use of CFD simulations to determine the position of the airbrakes. We explored another open-source software for CFD simulations, namely OpenFOAM [5]. While OpenFOAM simulations are useful, they are computationally intensive to be executed on standard desktop/laptop computers. To address this issue, I worked over the past summer to assemble a custom computing cluster using available hardware at SLU. This task brought with it its own set of challenges. While SLU offers high-performance computing (HPC) classes, they are taught from the perspective of developing and executing algorithms, and there are no classes to teach the actual building of a computing

cluster. This is another prime example of our need to learn outside the classroom to accomplish our goals. As most students do, we turned to Artificial Intelligence (AI)/ChatGPT [6] for help – to be used as an interpreter or editor rather than a coder.

Following a tutorial from a website [10], I learned the basics of setting up a multiple-node cluster on Ubuntu. I checked that each PC had adequate RAM, a hard drive, and a simple graphics card. The cluster hardware consists of 20 desktops, each with an Intel i7-4790 quad-core 3.60 GHz processor and 16GB of RAM, running Ubuntu 22.04.3 LTS connected to a 24-port gigabit switch. Once all PCs were in good condition, I connected each to the ethernet switch. I then began setting up the cluster by wiping all 20 desktops and installing the Ubuntu Linux distribution from a USB drive. With my previous experience and knowledge of building desktop PCs, everything up to this point was very familiar. However, the actual configuration of the cluster took many weeks since I had no experience with Linux.

While I knew how to read and write code, I was not fluent in bash or C++. I wanted to understand what I typed into the command terminal, not copy and paste from a website. Using ChatGPT was vital in learning terminal workflows. For example, the line: *$ mpirun -np 5 -hosts worker,localhost ./mpi_hello* was gibberish to me when I first read it. When I asked GPT, "*What does this line do in an Ubuntu terminal?*" it gave an overview of the Message Passing Interface (MPI) [7], as well as the structure of the command. Using ChatGPT was integral to my learning since I did not have to know what MPI was to understand the command now (in hindsight, it does appear that my learning would be more complete with an in-depth knowledge of MPI, but it does come at the cost of time and potentially missed deadlines). One of the computers on the cluster was designated as the main computer, which communicates with each node using Open MPI and shares directories using Network File System (NFS). The rest were given names such as *acl_n1* and *acl_n2* so they could be referenced later quickly. The IP address of each computer was also noted to configure the host file located on the main computer and the node. The user can access each node from the main computer using the Secure Shell Protocol (SSH). A script that calls each node to return the standard "*Hello World!*" greeting was used to test if the cluster works in parallel. Once all computers were able to run in parallel, I was able to begin working on OpenFOAM.

Understanding the workflow for OpenFOAM consisted of reading the user guide and prompting ChatGPT to explain more topical questions (another learning experience in how to make the most of ChatGPT by giving it appropriate prompts – a brief foray into the field of "prompt engineering"). OpenFOAM has many tutorial cases that help understand how the software works and can be used to write custom simulations. I started running the airFoil2D tutorial case on the main computer, simulating airflow over a 2D airfoil in incompressible initial conditions. I asked ChatGPT to "*Explain each file in a typical OpenFOAM case*." It would respond with the following response (shown in *Figure 4* below) that I followed along with the airFoil2D case.

In a typical OpenFOAM case directory, you'll find several files and directories that are essential for running and managing simulations. OpenFOAM follows a specific directory structure and naming convention for its files. Here's an explanation of some common files you might find in an OpenFOAM case directory for version 9:

1. 0: This directory contains initial conditions for all fields in your simulation. Within this directory, you might find files like:
- U: Initial conditions for velocity.
- p: Initial conditions for pressure.
- T: Initial conditions for temperature (if applicable).
- alpha: Initial conditions for phase fractions (for multiphase flows).

2. constant: This directory contains files that are constant throughout the simulation. Some important files in this directory include:
- polyMesh: Contains information about the mesh used in the simulation.
- transportProperties: Defines transport properties like viscosity and diffusivity.
- turbulenceProperties: Defines turbulence models and related parameters.

3. system: This directory contains various files that control the numerical aspects of the simulation. Key files include:
- controlDict: Controls the overall simulation time and other time-related parameters.
- fvSchemes: Defines discretization schemes for different equations.
- fvSolution: Specifies solution algorithms and tolerances.
- decomposeParDict: Configuration file for domain decomposition in parallel simulations.

4. constant/polyMesh: This directory contains files related to the mesh, including point and face information.

*5. 0.orig: Sometimes, you might encounter this directory alongside **0**. It contains the initial condition files as they were initially set up before any modifications or boundary conditions are applied.*

6. Allrun: A shell script that can be executed to run the simulation in one go. It typically includes commands to mesh the geometry, set up boundary conditions, and run the solver.

7. Allclean: A shell script used to clean up simulation results and intermediate files. It's often used before rerunning simulations or archiving results.

8. log: This file records the log messages generated during the simulation run. It's useful for debugging and monitoring the simulation progress.

9. postProcessing: This directory contains scripts and files for post-processing simulation results, such as visualizations and data analysis.

10. README: A text file providing information about the case, including its purpose, setup instructions, and any other relevant details.

These are some of the common files and directories you might encounter in a typical OpenFOAM case directory. The specific files and their contents can vary depending on the nature of the simulation and the solver being used.

Figure 4: Prompts and results from interactions with ChatGPT

I could paste a single file from these initial definitions and get a line-by-line explanation of the code. Doing this taught me how meshing works for a 2D case and the solver, simpleFoam. Learning a new coding language and how to use command terminal-based software is an iterative process. Knowing if the simulation ran correctly took minutes to converge and return any values, which adds up. ChatGPT was used to read dense errors, as I could paste it in, ask where the problem is, and extrapolate from there. These problems were the extent ChatGPT was able to solve. ChatGPT only knew the scope of the problem I gave it, as it could not read my files and explain where I went wrong. Even though this sounds intrusive, it would be interesting to point ChatGPT to a specific directory and ask where the problem lies.

The most challenging thing to figure out was where variables and functions were being called from. I had to navigate multiple files and lines of code to change initial conditions, mesh settings, 3D models, post-processing settings, and others for a single simulation. Learning from the available information, I was able to write a script to simplify the process of testing cases and record the time the simulation takes to run. I relied heavily on ChatGPT for translations to bash script to write the script. However, the more I worked on the script, the less I relied on ChatGPT for basic syntax help. I decreased the time to set up a simulation by compiling all the essential information I would need to change into one file. This also made it less likely to make a mistake inputting new initial conditions, leading to less debugging.

Our team used the cluster to simulate how our rocket would fly. Knowing how OpenFOAM operates, we utilized the open-source code to run highly accurate cases in less time than other standard CFD programs such as ANSYS [15].To calculate the correct airbrake deflection angle that will apply adequate drag force on the rocket, we simulated the change in the drag coefficient ($\boldsymbol{C_D}$) for each deflection angle at different airspeeds. We can calculate the drag produced using the basic drag equation:

$$\boldsymbol{D = \frac{1}{2}\rho V^2 S_{ref} C_D} \qquad \textbf{Eq. 1}$$

The reference area ($\boldsymbol{S_{ref}}$) of a rocket is the cross-sectional area of the body tube. Using simple trigonometry, we can calculate the added reference area for different deflection angles:

$$\boldsymbol{S_{ref} = \pi r_{BT}^2 + w_{AB} l_{AB} \sin(\theta°)} \qquad \textbf{Eq. 2}$$

Using OpenFOAM, we wrote a script that iterates through each rocket model and simulates the drag coefficients and forces at varying airbrake deflection angles. These values will then be stored in a lookup table, which the algorithm will reference during flight. The cluster drastically decreased the time since multiple simulations were run simultaneously on different nodes. Figure 5 illustrates the velocity values on a 60° airbrake deflection model at 205 m/sec (671 ft/s or 457.5 mph).

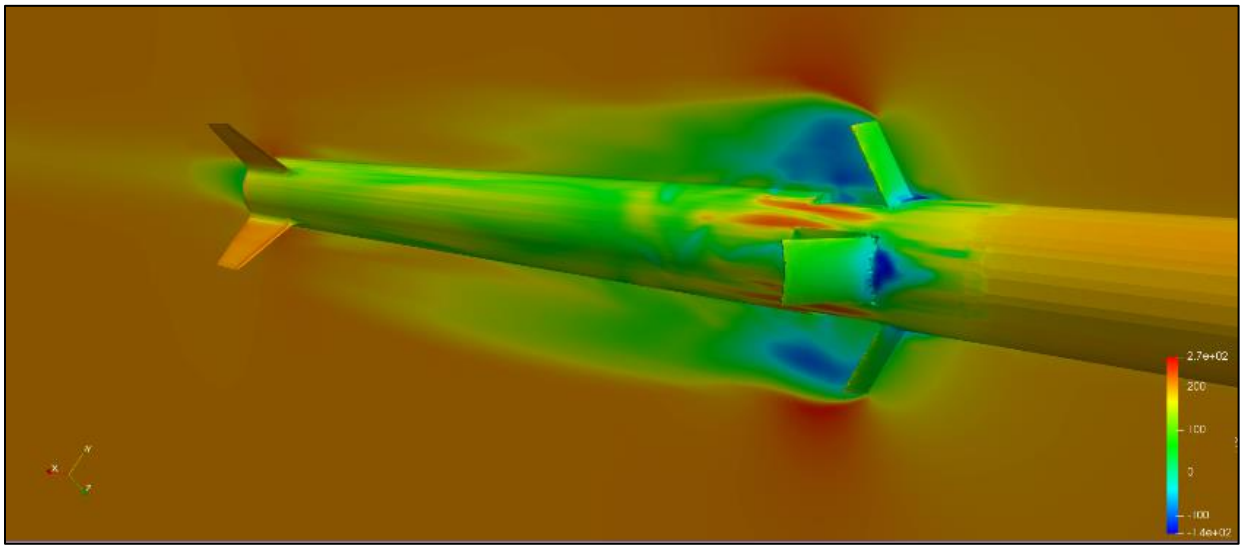

Figure 5: Paraview Post-Processing of Model with 60° Deflection of the Airbrakes (m/s)

My team manufactured most of the rocket's structures, such as the lower airframe, fins, centering rings, motor mount, and airbrake mechanism. SLU offers introductory CAD and basic machining courses on lathes and mills. Taking these in previous years helped us with the initial design of our rocket. However, we were never taught the multiple manufacturing methods needed to produce parts.

Our rocket lab uses an X-Winder [14] carbon fiber winding machine to wind the lower airframe and motor mount. The tubes are wound with carbon fiber filament around multiple foam circles stacked on a steel bar. We worked with our lead technician in our department, who runs the department's water jet, to cut these circles from 2" thick pink insulation foam. From him, we learned what file type the water jet needed and how to format the file. The alums showed us how to use the X-Winder to wind our motor mount and we later wound our body tube. We designed our centering rings and retaining tail cone to be machined on a lathe and mill. With the SolidWorks drawings, the operator advised us on what tooltips to use and how to set up the lathe and mill. Figures 6 and 7 below show some of the manufacturing processes mentioned here.

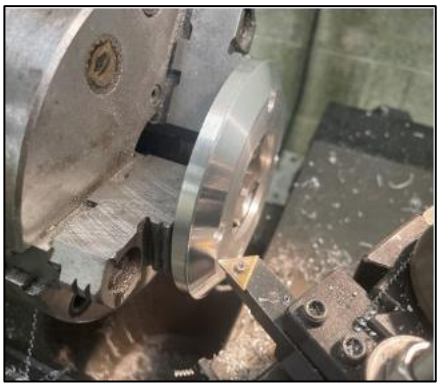

Figure 6: Tail Cone Manufacturing

Figure 7: X-Winder Lower Airframe Tube Winding

From what I learned over the summer, I instructed my team on how the airframe should be constructed. The fin can was constructed by epoxying the motor mount, centering rings, and fins inside a jig, ensuring the fins were perpendicular. While the fin can was curing, the lower body tube was prepped by cutting slits into the bottom for the fins to slide into. All internal parts in the avionics sled were designed and 3D printed using the rocket lab's Bambu Lab X1 Carbon 3D printer [16]. The sled was limited to a 3.91" diameter and 12" long section of the rocket since it is housed inside the coupler. With such a small space, 3D printing parts enabled my team to quickly design, print, and test fit a part in the sled.

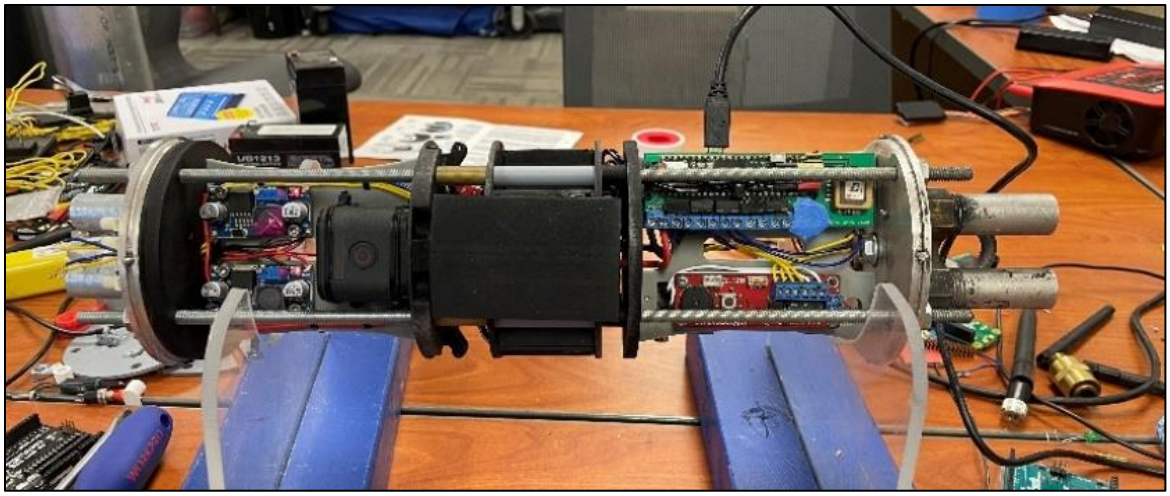

Figure 8: Fully Assembled Avionics Sled

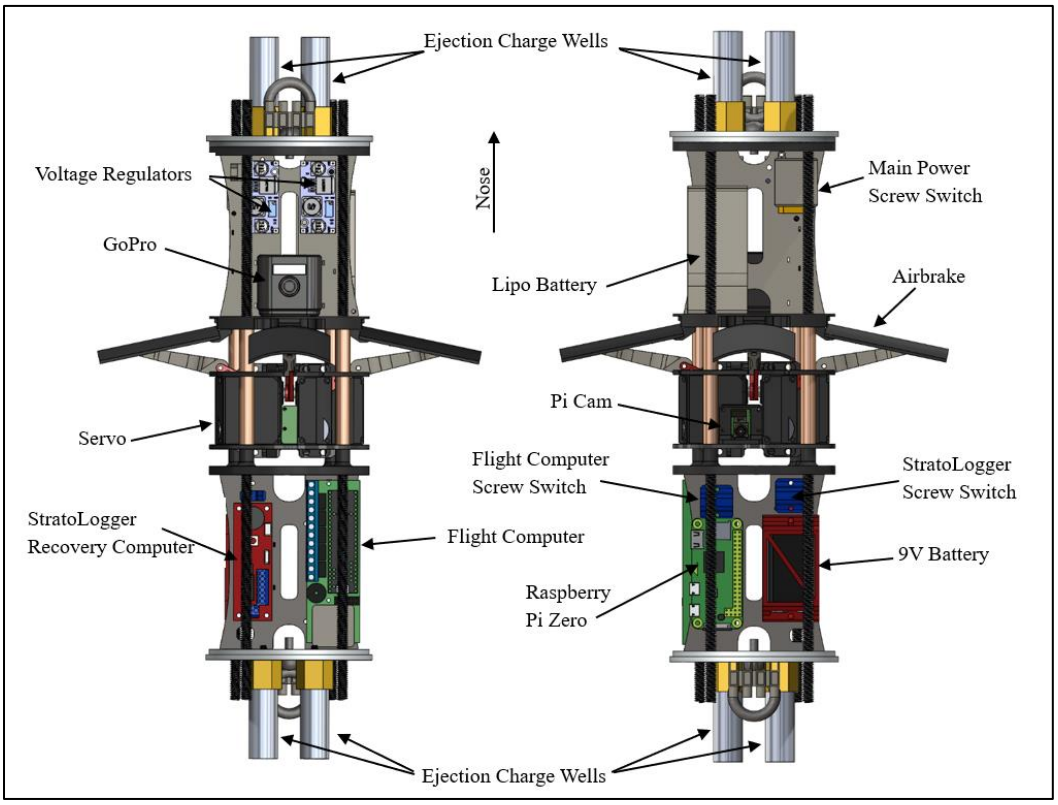

Figure 9: Avionics Sled Cross-Section

The sled is comprised of the airbrake mechanism, flight computer, StratoLogger [8], two voltage regulators, 9v battery, 2000mAh 3S LiPo battery, Raspberry Pi Zero [9], GoPro Session video camera, and screw switches for each computer and main power. The LiPo battery was selected for its long battery life since it needs to power the servos, flight computer, and Raspberry Pi Zero. Learning from experienced practitioners about operational safety also taught us to have separate power sources on the rocket. Consequently, we used a separate 9V battery to power the StratoLogger, the primary recovery computer.

To prepare for the first test flight, my team wrote a checklist of essential items to do up to launch. The list was detailed; however, we found it must be detailed enough so we do not have to think about anything and check it off. We also did not account for any setbacks when setting up. The main flight computer shorted before it could fly, with the recovery system being the most likely source of the issue, as when in test mode, the flight computer fires the pyros from any slight movement. Future flights will use two StratoLogger computers for dual deployment to prevent any shorting of the main flight computer.

We were able to fly to 7314 ft. without the airbrakes deployed and used the StratoLogger to collect altitude and velocity data. From this data, my team found that our MATLAB flight simulation algorithm was more accurate (predicted altitude of 7361 ft) than OpenRocket (predicted altitude of 6448 ft). We could attribute this to the fact that the values of the drag coefficient $\boldsymbol{C_D}$ used in our algorithms were derived from OpenFOAM simulations, and OpenRocket assumes a $\boldsymbol{C_D}$ of 0.6. This leads us to believe that the $\boldsymbol{C_D}$ values derived from OpenFOAM simulations will represent the drag values on the rocket when flying with airbrakes. My team expected four launches to test and modify the airbrake algorithm. Figure 10 plots the simulated flight and actual altitude with no deflected airbrakes.

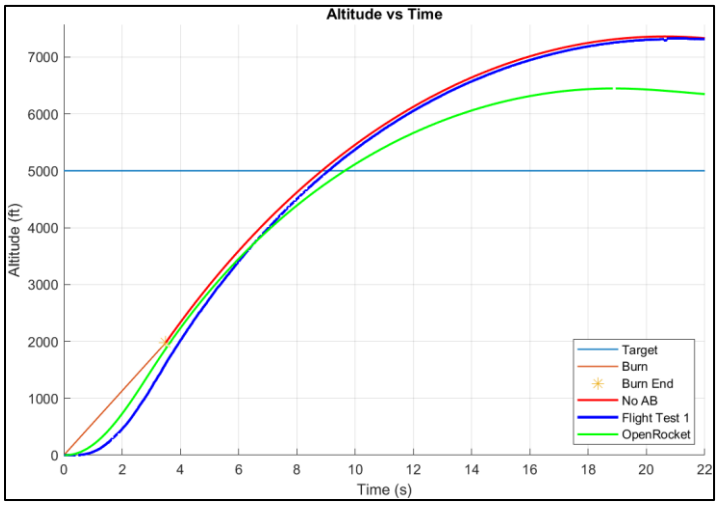

Figure 10: Flight Data vs. OpenRocket and Custom Simulations

Our second flight test revealed some flaws in our initial design. The initial acceleration from the motor caused the flight computer's screw switch wires to come loose from the terminal blocks, disabling the airbrakes. Both pyros were blown at apogee, causing a large enough pressure difference in the avionics bay to turn off both Stratologgers temporarily. This power loss led to both Stratologgers not blowing the pyros at 1000 feet, which would allow the main parachute to deploy. Luckily, our airframe only suffered a minor crack on a fillet between the fin and body from the high impact. Once recovered, the avionics bay had minor damage to some printed parts, and one servo was broken from the impact. The Stratologgers both blew at apogee because I forgot to change the delay in the PerfectFlite software [17]. They are programmed by changing the apogee delay and the main deployment altitude. I changed the redundant computer to have a 5s delay and deploy at 800 feet, while the main computer had a 0-second delay and deployed at 1000 feet. The crack was sanded down and epoxied over, and the servo was replaced.

Figure 11: Flight 2 Liftoff

Figure 12: Flight 2 Burn

Unfortunately, our luck ran out on our third flight test. 1.5s after liftoff, the forward closure in the motor was blown upwards out of the casing and into the avionics bay. This had enough force to shear the lower airframe pins to the coupler and eject three propellant grains out of the motor. The lower airframe was left in a free fall since the shock chord attaches to the forward closure. The Stratologgers detected the sideways acceleration and blew all four charges to deploy the parachutes, but the main charges did not shear the upper airframe from the coupler. The upper airframe and coupler were too close to the ground for the drogue to fully deploy, causing it to hit the ground at around 100 ft/s (~70 mph). This led to the coupler cracking in half, the nose tip snapping off, and the avionics in pieces.

Figure 13: Upper (left) & Lower (right) airframe damage

Figure 14: Avionics Bay Damage

While we quickly blamed the manufacturer, Aerotech, we found that we put the motor together incorrectly. The forward and aft closures are screwed in to hold the grains and nozzle in the motor casing. When we put it together previously, there was a small gap between the aft closure and the casing and no gap between the forward closure and casing. Although not specified in the instructions, this is how it should be put together. In preparation for our third flight, we left a gap between the forward closure and casing, which led to its failure.

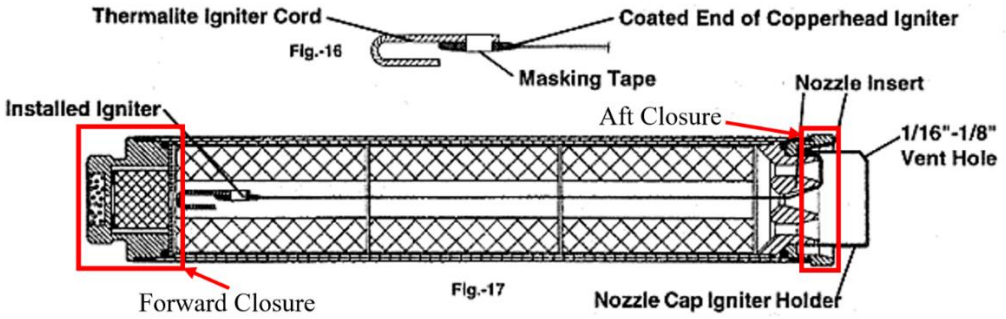

Figure 15: Example of an Aerotech High Power Rocket Motor

Conclusion

A university rocket lab allows students to understand the challenges of designing, manufacturing, and flying something they created. My professor likes to say, "Designing a plane is more of an art than a science." The same can be said for designing rockets. The initial airframe design affects the center of pressure. Later, adding avionics such as batteries, flight computers, recovery devices, and epoxy shifts the center of gravity, affecting the rocket's stability. Balancing these variables takes time, effort, and creativity. While it is optimistic to expect a project to work perfectly the first time, mistakes happen. Making mistakes allows students to learn and grow, even if it causes total failure. I can accept that this 8-month project is in pieces, and I am more grateful for everything I have learned because of it. Experiencing the entire design process as an undergrad will enable me to make more educated decisions when I enter the workforce. I plan to present the learning experiences between my Level 1 LOC-IV kit and Project RADARS rocket at the conference this summer.

References:

[1] "Spaceport America-The World's First Purpose-Built Commercial Spaceport," *Spaceport America Cup*. https://spaceportamericacup.com/

[2] "NASA Student Launch Challenge - NASA." https://www.nasa.gov/learning-resources/nasa-student-launch/

[3] "LOC-IV 4"," *LOC Precision / Public Missiles Ltd.* https://locprecision.com/products/loc-iv (accessed May 09, 2024).

[4] "OpenRocket Simulator," *openrocket.info*. https://openrocket.info/

[5] OpenCFD, "OpenFOAM® - Official home of The Open Source Computational Fluid Dynamics (CFD) Toolbox," *www.openfoam.com*. https://www.openfoam.com/

[6] OpenAI, "ChatGPT," *chat.openai.com*, 2024. https://chat.openai.com/

[7] "Open MPI: Open Source High Performance Computing," *Open-mpi.org*, 2019. https://www.open-mpi.org/

[8] "SLCF," *www.perfectflite.com*. http://www.perfectflite.com/SLCF.html

[9] R. P. (Trading) Ltd, "Raspberry Pi," *Raspberry Pi*. https://www.raspberrypi.com/products/raspberry-pi-zero/

[10] "How to setup a basic LXD cluster | Ubuntu," *Ubuntu*, 2024. https://ubuntu.com/tutorials/how-to-setup-a-basic-lxd-cluster#1-overview (accessed May 09, 2024).

[11] "National Association of Rocketry," *Nar.org*, 2019. https://www.nar.org/

[12] "Home - Tripoli Rocketry Association," *www.tripoli.org*. https://www.tripoli.org/

[13] T. Drake, M. Muetzel, C. Cummins, and M. Otten, "Project RADARS Rocket Altitude Determination and Response System." Accessed: May 09, 2024. [Online]. Available: https://www.timdrake.org/wp-content/uploads/2024/03/AIAA_Paper.pdf

[14] "X-Winder Filament Winding Machines for sale," *xwinder.com*. https://xwinder.com/

[15] ANSYS, "Engineering Simulation & 3D Design Software | ANSYS," *Ansys.com*, 2017. https://www.ansys.com/

[16] "Bambu Lab X1-Carbon Combo 3D Printer," *Bambu Lab US*. https://us.store.bambulab.com/products/x1-carbon-combo

[17] "Perfect Flite Software," *www.perfectflite.com*. http://www.perfectflite.com/Download.html