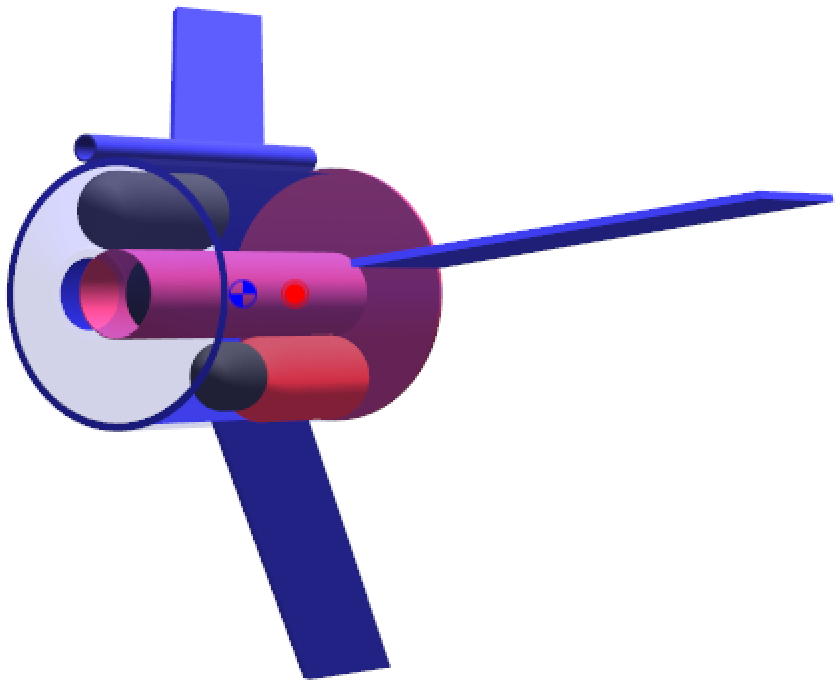
Flying Ostrich Stage 1 Design Proposal



Reinier Johnson

Stage 1 Design Proposal 3/19/2021-3/22/2021,
Reinier Johnson

Introduction:
This is a proposal for the final design of the Flying Ostrich first stage. The stage aims to propel a model rocket upper stage to an overall apogee of 500ft or approximately 151m in altitude. This will require Sufficient stability, aero, recovery and reuse capability, as other rockets incorporate.

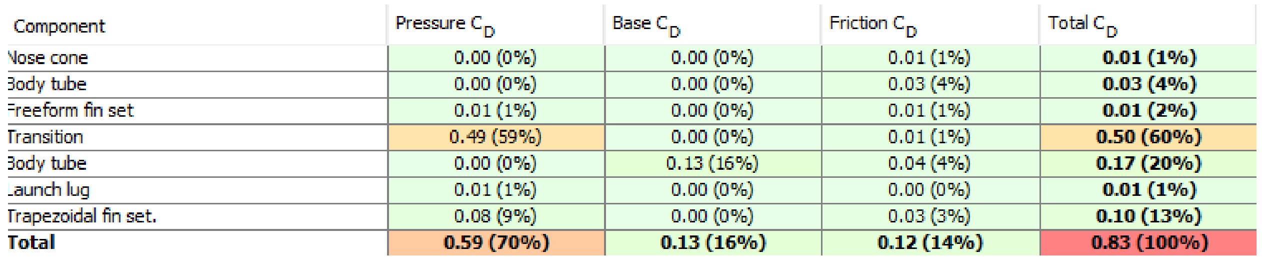
□□□

Aerodynamics Introduction:
Aerodynamics are a key part of achieving our goal. Having a simple, safe, lightweight and stable rocket stage is of the utmost priority. This first section dives into some of the challenges of this and the ways we can counter the issues with aerodynamics for The Flying Ostrich's first stage.

Part 1: Overall Design Review for First Stage Aero

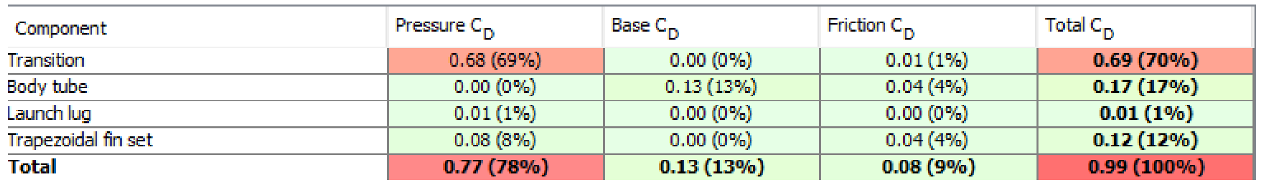
The first stage of the rocket will be a challenging part to engineer aerodynamically. With a stability coefficient that aims to be in the range of 1.4-1.5 cal., the rocket requires large fins to be connected to the aft section.
The three aft fins (simulated) have a root and tip chord of 6.1cm, or nearing half of the rocket body length (not including the transition section). The fins have a total height of 11cm with an equal sweep length. The angle of the fin sweep is 45°, providing the most stability possible, without creating too much dirty air (turbulent flow) as can be seen in part 3 (CFD).
The transition section is one of the more challenging parts of the rocket, with it creating around 70% of the total drag on the rocket (before attaching an upper stage) and after attaching a second stage the transition section creates 60% of the total drag. The estimated drag coefficient for the bottom stage on its own is 0.69 and with an upper stage it is estimated at 0.50. These are shown in the two images below:

With upper stage attached to the first stage:



Component	Pressure C_D	Base C_D	Friction C_D	Total C_D
Nose cone	0.00 (0%)	0.00 (0%)	0.01 (1%)	**0.01 (1%)**
Body tube	0.00 (0%)	0.00 (0%)	0.03 (4%)	**0.03 (4%)**
Freeform fin set	0.01 (1%)	0.00 (0%)	0.01 (1%)	**0.01 (2%)**
Transition	0.49 (59%)	0.00 (0%)	0.01 (1%)	**0.50 (60%)**
Body tube	0.00 (0%)	0.13 (16%)	0.04 (4%)	**0.17 (20%)**
Launch lug	0.01 (1%)	0.00 (0%)	0.00 (0%)	**0.01 (1%)**
Trapezoidal fin set.	0.08 (9%)	0.00 (0%)	0.03 (3%)	**0.10 (13%)**
Total	**0.59 (70%)**	**0.13 (16%)**	**0.12 (14%)**	**0.83 (100%)**

Without upper stage attached to the first stage of the rocket:



Component	Pressure C_D	Base C_D	Friction C_D	Total C_D
Transition	0.68 (69%)	0.00 (0%)	0.01 (1%)	**0.69 (70%)**
Body tube	0.00 (0%)	0.13 (13%)	0.04 (4%)	**0.17 (17%)**
Launch lug	0.01 (1%)	0.00 (0%)	0.00 (0%)	**0.01 (1%)**
Trapezoidal fin set	0.08 (8%)	0.00 (0%)	0.04 (4%)	**0.12 (12%)**
Total	**0.77 (78%)**	**0.13 (13%)**	**0.08 (9%)**	**0.99 (100%)**

These simulations aim for a stability caliper of 0.364 for the first stage alone and 1.44 for the two stages of the rocket attached. To get these stages to an optimal stability caliper, 1.5cal., we can make the root chord of the find 6.5, and make the fins 12cm in height to get close (1.49cal.) although i=this may sacrifice structure within the rocket itself.. As the materials of the fins are most likely going to be PLA, the rocket will have less stability as if they were in cardboard or in wood. The stability table is found below, the stability is measured with both rocket stages connected:

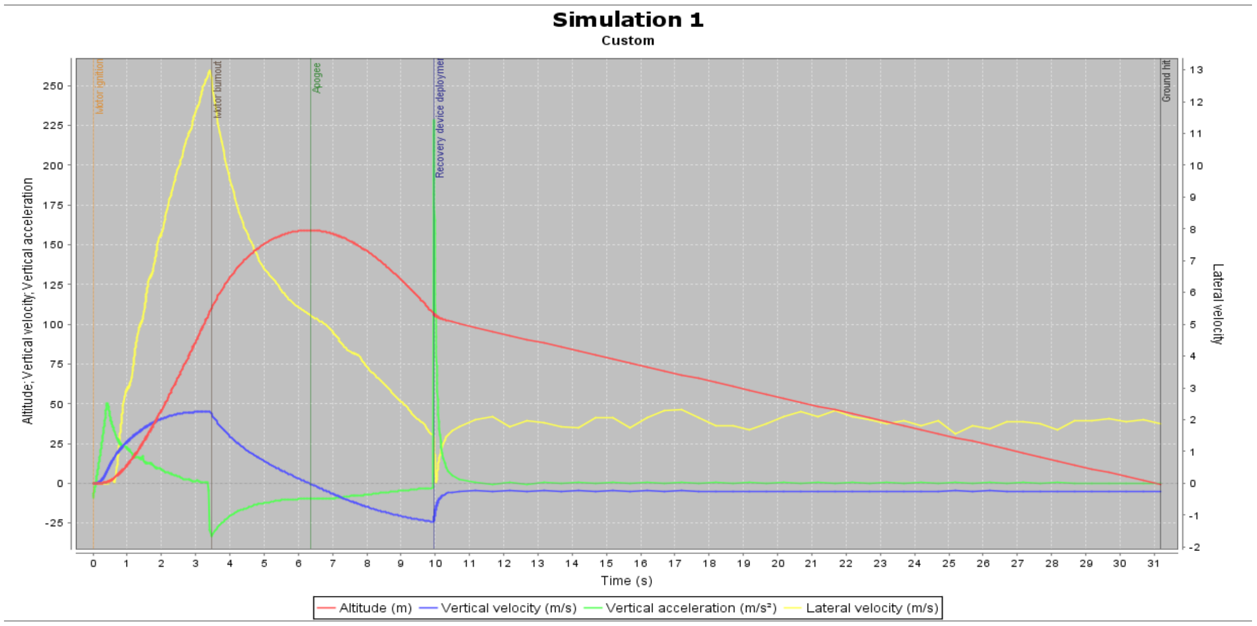
PLA (1.24g/cm^3)	1.44cal.
Cardboard (0.68g/cm^3)	1.51cal.
Wood (0.17g/cm^3)	1.57cal.

Part 2: OpenRocket Aerodynamics Simulations

After modifying a new OpenRocket file to simulate only the aft end with our selected solid fuel motor, we will have a good reference for aft end stability. The simulations expect an apogee of 119 meters without the top stage, although stability margins are minimal when the first stage is on its own.
Predictions from OpenRocket simulated (Values are without a top stage attached, top stage adds stability and weight so these calculations are solely for comparison later):

Apogee	147m
Max Vel.	42.4m/2 (Approx. Mach 0.13)
Max Accel.	39.7m/s^2
Wet/Launch Mass	507g
Est. Ground Hit Vel.	4.95m/s
Vel. at Parachute Deploy	24.5m/s (Lateral) 0 (Vertical)
Optimum Delay for Deploy	2.97s

These simulations can be corroborated with this graph:



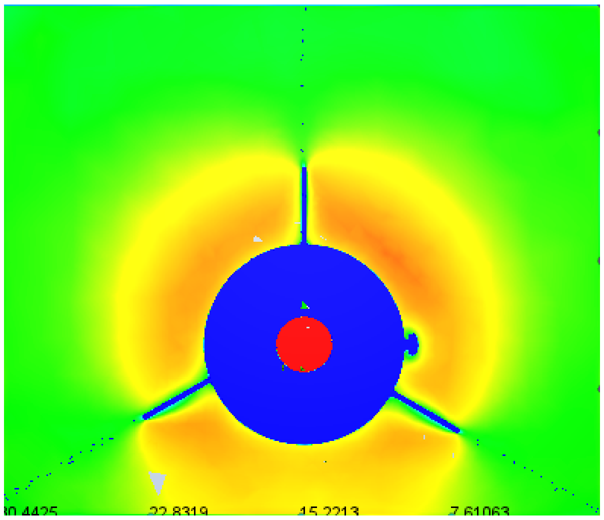
Note the lateral velocity is equivalent to -24.5m/s and the velocity laterally is leveled off after parachute deployment, this is due to the recovery device being deployed after apogee with an effective 0m/s relative velocity (vertically) at deploy. Something that can be considered with the MPU6050 data unit (Part 5) is the measure for acceleration across axes at this point, the MPU6050 will measure the acceleration across the X, Y, and Z axes using G units.
Part 3: Converted OpenRocket Model CFD (Computational Fluid Dynamics) and Comprehensive Analysis of CFD Model

The OpenRocket model for the rocket stage is very in depth, the first stage (which is the focus of this CFD analysis) has highly optimized measurements that can be visualized using CFD software provided through an educational license by Autodesk. This CFD can give us insights on the rocket, and help us further optimize the rocket for the incoming final iteration.
To start off, the model was converted into a 3d figure using Blender, which was then imported into Fusion 360 for optimization and file conversion for other Autodesk Suite applications. The rocket body that was posted in the google classroom was used as a base model for the rocket body, and fins of the proper size were added to the model, which later provided good visualizations in Autodesk CFD 2021.

Part 3: More Stats for Simulations and CFD Analysis
Simulated with airflow coming from the top, or the direction the rocket would be travelling in. The airflow rate is 110m/s with steady velocity, to simulate an anominal launch profile, and gives data at the normal, and worst case scenarios, simultaneously.

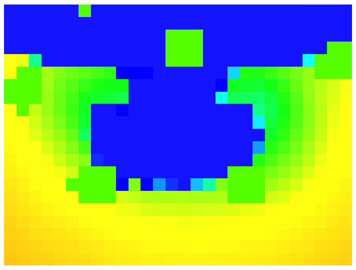
Here is the comprehensive analysis of the CFD simulations:

Figure 1: Top-down view



This top-down view of the rocket's first stage visualizes air speed across the rocket body, with red being higher speed (max 200m/s for true red color) and blue being low speed (minimum 0m/s for true blue color. This top-down picture of the simulation provides a good visual of turbulence for a straight flight. There are three clear sections in the yellow-orange flow (about 60-80% maximum turbulence) which are divided by the three fins. One thing you may notice from this picture is the launch rail attachment block, this is providing low turbulence for reasons explained under Figure 2. Otherwise, there is the true red part of the model, which is the estimated turbulence of a straight through hole in the rocket, or a simulation of a fully burnt motor. This is providing the max turbulence for two reasons, curling airflow and straight through flow, ultimately the turbulence is created at the intersection of the air waves and causes a highly unstable area within the rocket body.

Figure 2: Zoom-In on Launch Rail Connection Block



This view of the launch lug/connection block, provides an interesting insight on the aerodynamics of the first stage. Although the resolution is not the greatest, there is a clear phenomenon taking place. The green region (around 20-30% maximum turbulence) is clearly differing from the yellow area, along with a clear cutoff. The estimation as to why this happens is that attack angle of the rocket as it is slicing the air. Because the rocket is going straight up, the air is flowing directly through the lug, creating a low pressure area within the launch lug slot. This low pressure causes the airflow to be laminar to an extent, which is effectively causing the flow to be ejected at high velocity, after being accelerated in a straight line. This is shown in the next figure (Figure 3).

Figure 3: Side View of Stage 1

This side view of stage one provides some peculiar information around the bottom section as well as near the end of the fin roots. Looking into the information provided by Figure 2 analysis, we can see that near the bottom of the plume, there is a slight bump out and some excess green colored flow, which has been ejected from the launch lug. (Looking to splatter paint on a test print to see this in action soon!) Moving back to the fin roots, we can see from the right side of the model that there is a clear fin outline, which is the aero-profile of our fin. As expected, there is discontinuity in the aerodynamics from this particular view of the fin.We can look to the bottom right first, for some great laminar flow being produced. This is due to the lack of turbulence as the tips of the fins are out of the impact of the rocket body, and it's features. This is why taking the fins from 11cm in height to 12cm can impact the stability so much, the more of this flow that the fins catch, the more stable the rocket becomes. Moving onto the inside of the fins, near the roots, we can see a faint cutaway in the airflow. Right under the fin the color fades from red, to yellow, to green and then back in a palindromic pattern. This could likely be due to the fin aero intersecting with the curling flow from the body (which is explained later with the consideration of the boat-tail for future iterations. Moving back to the roots of the fins, we can notice that the flow expansion does not apex at the cross section of the pressure peak within the fin (red color on the fin), but rather stops near the apex of the yellow and green flow, although all of the colors do have some correlation.

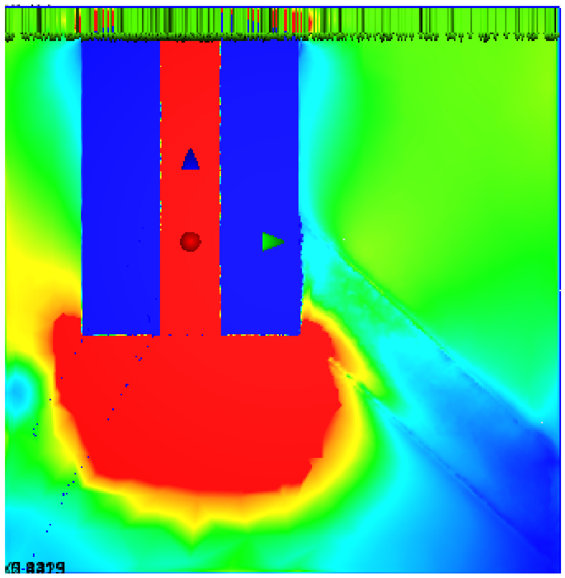


Figure 4 and 5: More Analysis on Figure 3's Second Half

(This image has been converted to grayscale for simplicity so the lighter parts of the image are the more turbulent areas.) This picture shows the flow separation on the bottom of a single fin, although the flow separation occurs across all fins on the rocket. This picture accurately shows the amount that the flow separates, although not much (approx. 1-2mm) the flow effect has a large impact on the overall stability, as it expands the turbulent flow that would not occur without the fins. Another place to note is near the bottom of the image, towards the middle, where flow expansion is seriously prevalent. This shows that the flow may be expanding up to 8mm which is detrimental to the overall stability, as the 8mm is translated radially around the entire rocket, and not just under the fins (see Figure 3 above).

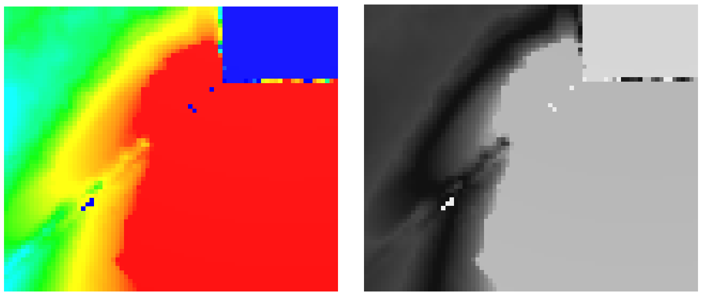
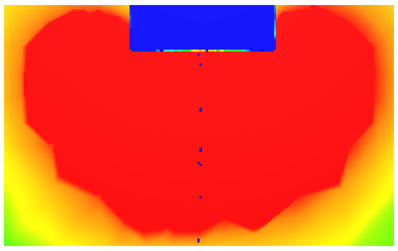


Figure 6: Flow Expansion Further Visualized



Here the flow expansion near the end of the body can be seen very clearly. The red in the image is the turbulent flow, which makes turbulent flow very prominent in this picture. This shows that the turbulence actually curls back up onto the rocket body due to the Coandă Effect (more information here!) This principle states that the pressure flow will move as follows: High pressure will attract to low pressure. This shows that there is a virtual 0 pressure area where the turbulent flow is curling over. This is a peculiar type of flow that is not easily controlled without the use of a boat-tail or another similar design, which is explored later. One other thing to note before the CFD section comes to a close is that the flow is fairly symmetrical, and this obviously follows the symmetrical rocket body design as most rockets do.

Aerodynamics Conclusion:
This section overviewed some of the issues the rocket currently has with aero, what can be improved and some potential features for future rocket prototypes, such as a second serial number for the Flying Ostrich. The section also went through data that OpenRocket provided, Non-visual simulations provided by OpenRocket and in depth analysis of Autodesk CFD 2021 3d simulations for the rocket. The Final note for this section is that all of the information was provided for a separated first stage except when otherwise noted, therefore some of the information is subject to change under different simulations, which may include updates to parts or inner mass components.

□□□

Electronics Introduction:
The electronics for the first stage of the Flying Ostrich consist of three main parts, the main computer (Part 1), the data unit and servo motor (Part 2), and lastly, the framework of the operation, the code (Part 3). These areas work in unison to form the deployment mechanism for the rocket, as well as provide some data to review the flight. The electronics proposal highlights these areas individually, in the respective parts mentioned above.

Part 1: The Main Computer, Raspberry Pi 0 W
The Raspberry Pi 0 W is the latest computer from Raspberry Pi that incorporates support for a multitude of devices, with constant speed. This computer differs from the Raspberry Pi Pico, the industry's smallest board that has similar functionalities although lacks in data rate and online usability/SSH connection capability. The Pico can only carry a single program, pre-loaded onto the computer and will start running that program as soon as the computer is started, much like other microcontrollers on the market (for more speculation on the Pico for future rockets consult

the Future Considerations section below). The Pi 0 W however, has capability to be connected to with ease and has ability to edit, compile, run and open data, locally on the Raspberry Pi. We can use these capabilities to their greatest extent as the need for us to be able to connect to the computer remotely at the launch site over WiFi (WiFi hotspot will be utilized to connect the rocket to the laptop and vice versa) and to edit the timing right before launch. The Pi can also connect to different devices (the ones we will be flying are covered in Part 2), and get data with sufficient speed. Stats on the computer can be found below.

Raspberry Pi 0 W Stats:

CPU/Processor	ARM11
CPU Clock Rate	200Mhz (Normal), 300Mhz (Overclocked)
RAM	512MB
Storage	32GB (Provided by SD card)
Power	1200MaH (Supplied by external UPS/battery)
Data Rate (Rate is not final, information on possible change is provided in next column)	10Hz (Looking to improve by removing overclock)

This information may not seem like much but some of these stats show liabilities with the system, as well as some areas we could improve. One liability is the clock rate. Even if the clock rate might seem to be low, the computer is only running a single program. The program does not use much of the CPU capability, although the data rates are slower due to the easy overclock, provided by the onboard configuration file, found within the SD card, once plugged into the computer. The clock rate is currently 300Mhz, with the overclock running, although due to thermal capability, and potential liabilities, the overclock might be removed providing that the data rate is lower than expected. The improvement aims to reach 15Hz to 20Hz in a good scenario. Some areas we could improve are the SD card, 32GB is a lot of room for the card, which may seem like a good idea, although with more storage comes the opening for more unneeded software. All of this software can be removed, however, some of the software does not fully remove itself, and can still be found running in the background.

Part 2: Flight Hardware

The flight hardware used is fairly simple. An MPU6050 data unit, providing X, Y and Z axis data to the computer, as well as a 9g servo motor from Tower Mini that provides a deployment mechanism for the parachute. These units will be covered in separate paragraphs below.

The MPU6050 is an extremely popular data unit, that has capability to provide accelerometer and gyroscope data to the computer. The MPU6050 will not be providing gyroscope data to the computer in our case, due to data speed and the lack of need for this data. The data unit will however be providing accelerometer data at higher speeds. The accelerometer data will be routed into the program, which will process the data into a .csv file. The .csv, named MPUData.csv will hold three columns of data: X, Y, and Z respectively. The data will later be processed using a premade python file, located on my laptop. The file will be processed and run in Jupyter Notebook, the most trusted, and capable online notebook for data scientists. The program reads the .csv and converts it into an array. That array contains three labeled columns that contain the data for each axis of the accelerometer. However, this program is not fundamental to the flight. The program that does communicate primarily with the MPU6050 is the one spoken about before. THe program that converts the data to a .csv file will also process the data points live. However the data coming from the MPU6050 is not in the proper format. We have to divide each datapoint by a certain amount to get the G scale value of the data point. After dividing this data, we can output it to the terminal using the sys library. This allows us to provide usable data for the user to see in real time. This data is also piped into the fault sensing section of the program. This section takes the G values and runs them through an algorithm that decides whether the seen data is safe for the computer or not. If the data is seen as unsafe within a certain period of time the parachute will be deployed. This period is in between the end of stage separation/second engine start and nominal parachute deploy timing. This is the last stop for the data before the servo program iterates. For more information on the program see Part 3.

The Tower Mini servo is a simple 9g servo motor that will be used to discharge a door located on the bottom of the aft section. The servo motor weighs, of course, 9g meaning that the ballast in the aft end will have to comply accordingly, depending on the placement of the motor. The servo aims to pull out a half door on the bottom of the rocket. Although no visuals for this are yet to be included, there is a design in the modeling phase. The servo is one of the most needed parts of the rocket. With the aim to successfully recover the rocket using electronics, a servo is going to be the key part. The servo will aim to provide decent pressure for the door to actually disconnect from the rocket body. This could be performed a number of different ways, although this proposal opts for a two door design. One door will overlap the second door so when it is released, the two doors will clamshell out making an opening for the air to enter the cavity and deploy the parachute. The servo is connected to the program at all times, although it is only used once in the entire operation. The servo rotates the most it can which provides the maximum torque to deploy the door. The servo as mentioned will always be connected to the program so that it is ready to be given a command with minimal delay. The command gives the motor in the servo an amount to move based on a calculation for angle and the servo rotates the motor a certain amount and with some gear reduction the servo rotates 180 degrees. Next in the review is the entire program overview.

Part 3: First Stage Ascent and Landing Program Review

First off, the entire program can be found in the Google Classroom for the project, feel free to follow along with the review by having the program open.

The first part of the program is importing the proper libraries. We have to import 7 total libraries for the program to run properly. These include: datetime to get the time, smbus to connect to the

MPU6050, math to do calculations, time to sleep the program at certain intervals, sys to output ot the terminal, pandas to get the dataset, and RPi.GPIO to connect to the servo. These libraries are REQUIRED parts of the program that essentially act as a backbone for many of the parts.

The second part of the program is setting up the servo PWM signals and the MPU6050 signals. First we set up the PWM (Pulse Width Modulation) signal boundaries for the servo. This helps the program interact with the servo on a level it can understand.
The second part skips over a part of the program we come back to later. Now we have to set up the MPU6050, a lot of this part is challenging to understand so to keep it simple, this part of the program defines values, operations, and establishes connections with the MPU6050. The program in this section consists of six functions (preceded by 'def' or definitions): read_byte reads data from the MPU, read _word reads the byte as a word, read_word_2c reads the read_word data as something the MPU can understand, dist returns the scaled values, then get_x(or y)_rotation is used for gyroscope data if needed. The last part is outside of function parameters and defines variables for the MPU6050 SMBus connection
After this we get to the third part, the iterating loop. This loop gives us the data we need to make the decisions. The loop starts by defining four key variables, the deploy, spike, spike_t and dep_time variables. These variables get true or false values for the deploy, and any given dataspike. The second two get the time the deploy occurred at and the time the spike occurred. Then the first if statement of many occurs. This first statement can be adjusted depending on the current data rate, and decides the time the parachute deploy triggers. After that there is the terminal output code, this reads all of the data from the MPU6050 and outputs it to the terminal. The program outputs: 'Accel (new line) ------' to give a heading, then gives all the measurements for the axes. Each of these chunks looks like this:

```
print ("\n")
 sys.stdout.write(f"Scaled Z: {a_z / 16348}")
 if a_z /16348 > 1.2:
     spike = True
     spike_t = str(datetime.now())
 sys.stdout.flush()
```

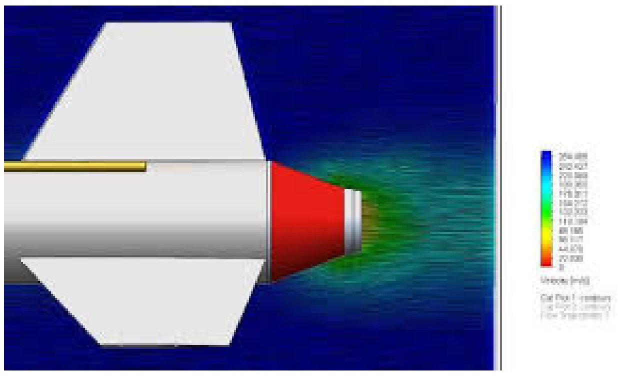
This snippet is for the z axis, as prompted by the use of a_z as a measurement. You can also see that the program takes a certain limit for Gs (highest G from OpenRocket data) and passes it through a loop. If the data given is larger than the limit, the data will log a spike using the yellow highlighted code.
The final part of the code cleans up the loop and exits it based on a landing dataspike after a certain amount of time. THis time limit, and all other limits, are defined by the OpenRocket simulation data, which may not be completely accurate, but is the most accurate you can get. The last part also confirms touchdown with a lot of repeating write and flush commands, which write to the terminal and clean the write command up respectively. Once this is finished it sets the status variable which defines the ability for the main loop to run as false, exiting the loop and moving to the cleanup loop.
The cleanup loop completes a base level data review and exits the code on its own, not much special code is used, because this part is purely cosmetic, as the data can be reviewed by the Jupyter Notebook program later.

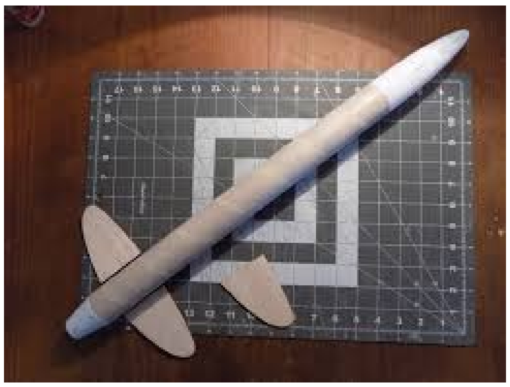
Electronics Conclusion:
Overall, the electronics in the aft section of the rocket are fairly simple and run very fluidly. The electronics have a low cost and effective use, while providing good results in initial testing, with more to come throughout the final weeks. After review of the electronic systems we can see that the electronics are in a sturdy place, with minimal to no flaws in terms of the program and hardware.

□□□

Future Designs Introduction:
In the future the design of our rocket has potential to be drastically improved. These new designs could further aerodynamically benefit our rocket, get rid of excess weight and more. Some of these designs include a more aerodynamic transition, a boat-tail design near the back, and updated electronics. We will cover the boat-tail in this section.

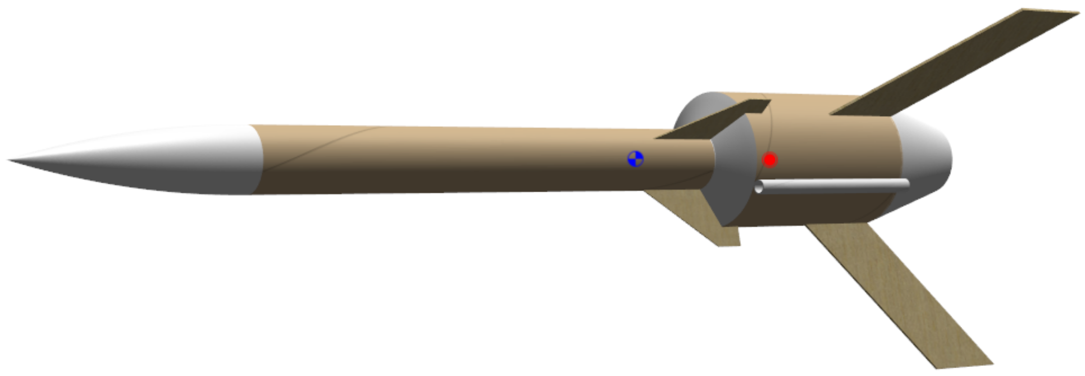
Part 1: The Boat-Tail
A boat-tail is a conical shape added to the rear of a rocket. This provides a nosecone like aero design that creates a new place for dirty or turbulent air to flow. Here are some pictures from online of boat-tails on model rockets:

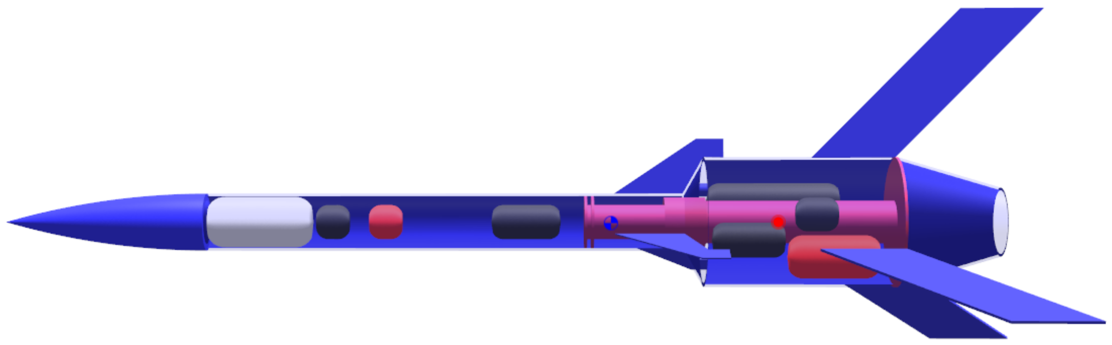


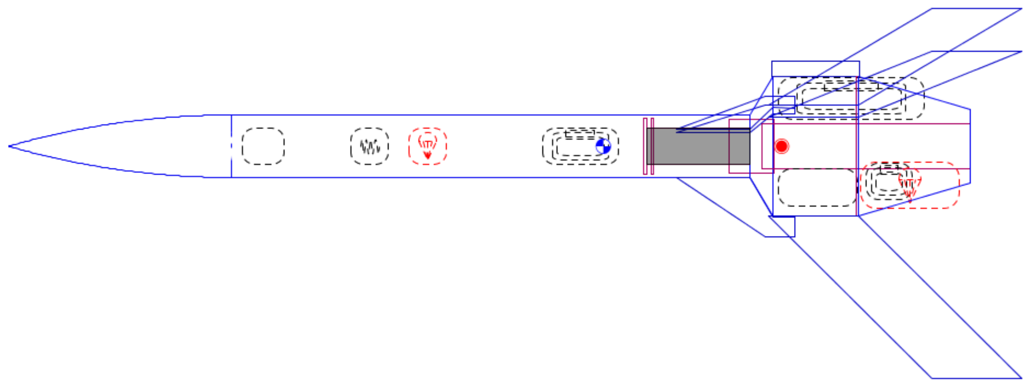


(This CFD visual is not mine but provides a good visual representation of how a boat-tail can positively affect aerodynamics.)

These pictures can give us a good visual of what a boat tail could look like on a normal rocket, although not on our rocket. We can visualize this by using OpenRocket to add a 'transition' section to the rear, this will simulate a boat-tail design. Below here are some visuals of a finished-product design with a boat-tail (possible look for SN2). Also attached is how the moto mounting would work with the new design, as the section for the motor is much larger than it theoretically should be for a staging rocket design. Here are the pictures of the rocket (first finished, second unfinished/cutaway and the last one of a potential improved/new inside):







(This last one shows a smaller section for the electronics, the next part goes over the drawbacks of this design.)

The last picture shows what a rocket that would fit the F15-0 motor would look like. The rocket would have a large boat-tail that would stretch about 4/6 the way up the existing rocket. This causes there to be less room for electronics in the upper stage. This means we would have to go with a Raspberry Pi Pico (as mentioned above and in Section 2, Part 3). Using the Pico would mean less fluidity for the rocket operations at the launch site, so the option would not necessarily be out of question, but would have to include a locked code file that would not be able to be reconfigured on launch day. Another drawback to this design would be the parachute and overall deployment mechanism. The mechanism would have to disconnect the entire boat-tail from the body and that would include separating it from holding a motor. The parachute would have to be significantly smaller to fit in the parachute bay as there would be about ½ to ¼ as much room as a design that does not incorporate a boat-tail or does not safely incorporate one (like the first two pictures above show). This unsafe design would use the normal payload and parachute bay size for the bottom stage with a conical tail added on to the back, this design would run the risk of burning the tail or even melting in away. This of course would only happen if we did not opt for thermal tape added to it. Now for the statistics of a boat-tail.

OpenRocket Simulation predicted Stats for Entire Rocket (Chute deploy is for Stage 1):

Apogee	130m (429ft)
Max Vel.	49.2m/s (approx. mach 0.14)
Max Accel.	61.5m/s^2
Wet/Launch Mass	805g
Est. Ground Hit Vel.	9.1m/s
Velocity at Deploy	41.9m/s (Lateral)
Optimal Delay	3.35s

As you can see, some of the stats were not improved from test to test. These were: apogee, launch mass, ground hit vel., and vel. at deploy. These are crucial areas that a fully 3D printed boat-tail would make worse. These stats depict the rocket as failing to pass the 500ft goal for apogee, falling 71 feet short, having greater velocity for parachute deploy (this is in the lateral direction, went up by 17.4m/s) lastly, the mass went up by 43 grams.
Overall these stats (these are predicted, although hard to go unnoticed) would harm the rocket flight too much to fulfill the goals for the rocket, and therefore should not be considered for the first serial number, as further iterations, tests and data review/implementation would have to be implemented into the process. This is, however, a very good idea for a second serial number, which could, after some further consideration for a boat-tail and new electronics.

Future Design Conclusion:
After consideration of a new design, the boat-tail and new electronics should be put off until SN2 due to the need for a finished product in little time(aiming for two weeks for empirical testing time in the final weeks), and so we can meet our goal of 500ft fully reused in that amount of time. Another consideration is the post-flight review that should be done after the rocket takes flight for the first time, and times after that.

Overall Conclusion:
After the design proposal data was finished, we looked into Aerodynamics first. This included CFD analysis, simulation analysis and looking over some data. The process was to model the first stage, run it through CFD and an open rocket model individually and get useful data that could prove the effectiveness of this rocket stage design. The CFD analysis went over areas we could further improve in later iterations and places where we are doing well.
The second set of five pages went over the flight hardware and deployment mechanisms. The first section covered the flight hardware's main part, the Raspberry Pi 0 W. The area went over the flight computer datasheet provided by Raspberry Pi and more data provided by various notable tech review magazines. The next part went over the hardware such as the servo and MPU6050, and covered some of the uses for these parts. The final part of this went over the program we will use for the main ascent, parachute deploy, and touchdown mechanism control. Finally we went over the possibility of a boat-tail for SN2 as it would not be feasible for SN1 due to constraints mentioned in the section. Overall this design proposal for SN1 has been 14 pages of data, reviews, and more that give an insight on a possible final design for SN1's first stage.