AstroJays Technical Report

Team 124 Project Technical Report for the 2018 IREC/SAC

Stéphane Testé [1], Annika Torp [2], Dan Zanko [3], Luis Miguel Rodriguez [4], Andrew Colombo [5], Benjamin Balfanz [6], Brian Prats [7], Christopher Williams [8]

The Johns Hopkins University, Baltimore, Maryland 21218, USA

[1] President, Recovery Subsystem Lead, Mechanical Engineering
[2] Secretary, Avionics Subsystem member, Mechanical Engineering
[3] Propulsion Subsystem Lead, Mechanical Engineering
[4] Payload and Communications Subsystems Lead, Chemical and Biomolecular Engineering
[5] Propulsion Subsystem member, Mechanical Engineering
[6] Structure Subsystem member, Materials Science and Engineering
[7] Recovery Subsystem member, Mechanical Engineering
[8] Simulation Subsystem Lead, Mechanical Engineering

I. Table of Contents

II. Nomenclature

AGL - Above Ground Level
CFD - Computational Fluid Dynamics
CG - Center of Gravity
COTS - Commercial, Off-The-Shelf
CP - Center of Pressure
DTF - Down To Fly, AstroJays' 2018 SAC rocket
FITC - Fluorescein Isothiocyanate
IMU - Inertial Measurement Unit
JHU - Johns Hopkins University
LiPo - Lithium Polymer
MOSFET - Metal-oxide-semiconductor field-effect transistor
PCB - Printed Circuit Board
PCR - Photochemical Reaction
PDMS - Polydimethylsiloxane
PVC - Polyvinyl Chloride
SAC - Spaceport America Cup

III. Abstract

The AstroJays are a competitive student rocketry team and a subset of the Johns Hopkins University in Baltimore, Maryland. Comprised of an interdisciplinary group of 53 students, the team is participating in the 2018 Spaceport America Cup in the 30,000ft apogee AGL, COTS solid motor category. Over the 2017-2018 school year, the AstroJays team has built and tested multiple rockets, subsystem configurations, and designs in preparation for the SAC. These prototypes primarily focused on improving rocket structure, layout, avionics reliability, reusable stage separation, and parachute reeving mechanisms. These designs and configurations were typically tested individually, validating (or in case of failure, invalidating) the varying system modifications. All tests were conducted at either Central Sod Farm or Higgs Farm in Maryland, complying with Tripoli oversight. Together, these systems are combined into a single rocket. DTF uses dual deploy event recovery with the drogue parachute deploying at apogee and the main parachute deploying at 1,000ft AGL. In addition, the payload – a microfluidics experiment – requires microgravity which is attained during the coasting ascent phase of the rocket. The primary avionics run the experiment when the end of powered flight is detected by an accelerometer onboard the avionics payload. The experiment powers an electric field which separates proteins from plasma and is only operational during the microgravity experienced during the coasting ascent. The experiment ends or the method becomes ineffective when not weightless -- this occurs at apogee when the drogue parachute deploys. However, the primary innovation in DTF is the student-developed reusable stage separation mechanism that mitigates expendables, cleaning, and launch preparations. This recovery mechanism is activated by the avionics through a relay. Furthermore, there is secondary, redundant avionic package; both with altimeter and accelerometer-triggered timed events.

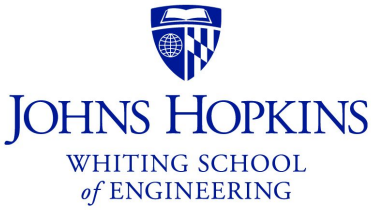
III. Introduction

A. Academic Background

The AstroJays were founded in the summer of 2017 when 4th year undergraduate Mechanical Engineering student Stéphane Testé worked in collaboration with the JHU Mechanical Engineering department to establish the team. Its primary goal is to establish and provide a clear, long-standing, and comprehensive program for students to explore and apply their knowledge to the field of aerospace engineering and rocketry. Our yearly projects open doors for students as they develop entire launch systems from scratch. In the future, the program shall integrate with the university curriculum, provide a cost-effective platform for short duration experiments in microgravity, and enhance collaborations between departments and industrial partners. All experiences gained by the 2018 team will pass onto subsequent years' teams, enriching the experience and fortifying the team's pioneering spirit.

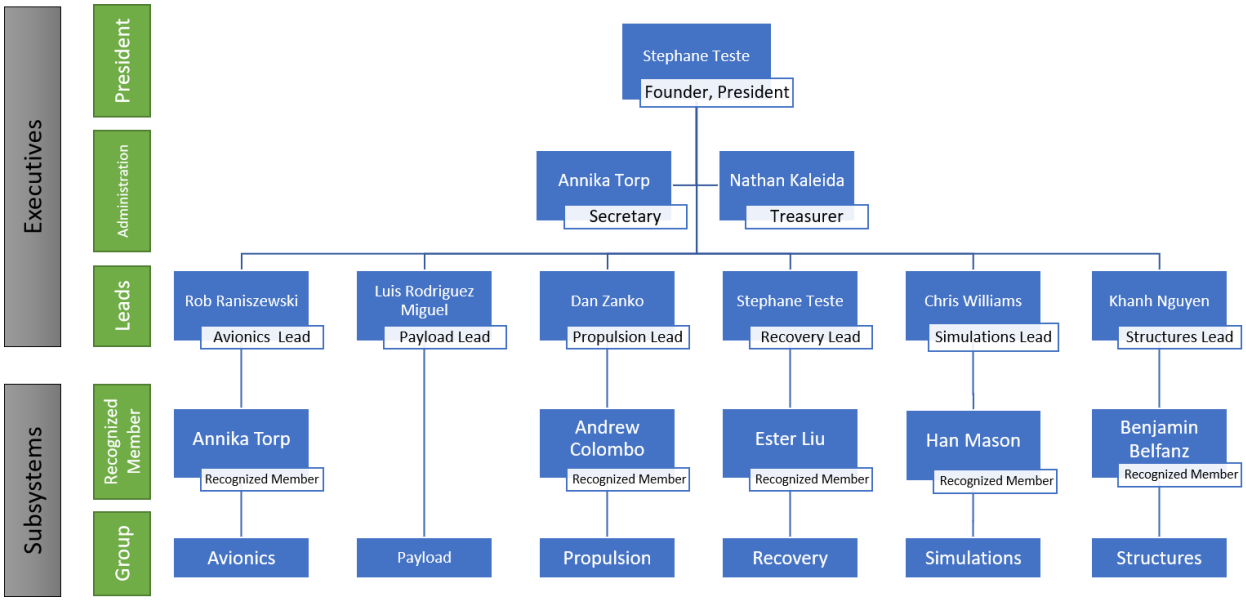
B. Stakeholders

Our sponsors are: Sigma Resources LLC, PPG Industries, Inc., Ansys, Inc., The Johns Hopkins University Applied Physics Laboratory LLC, the Johns Hopkins University Department of Mechanical Engineering and the Whiting School of Engineering.



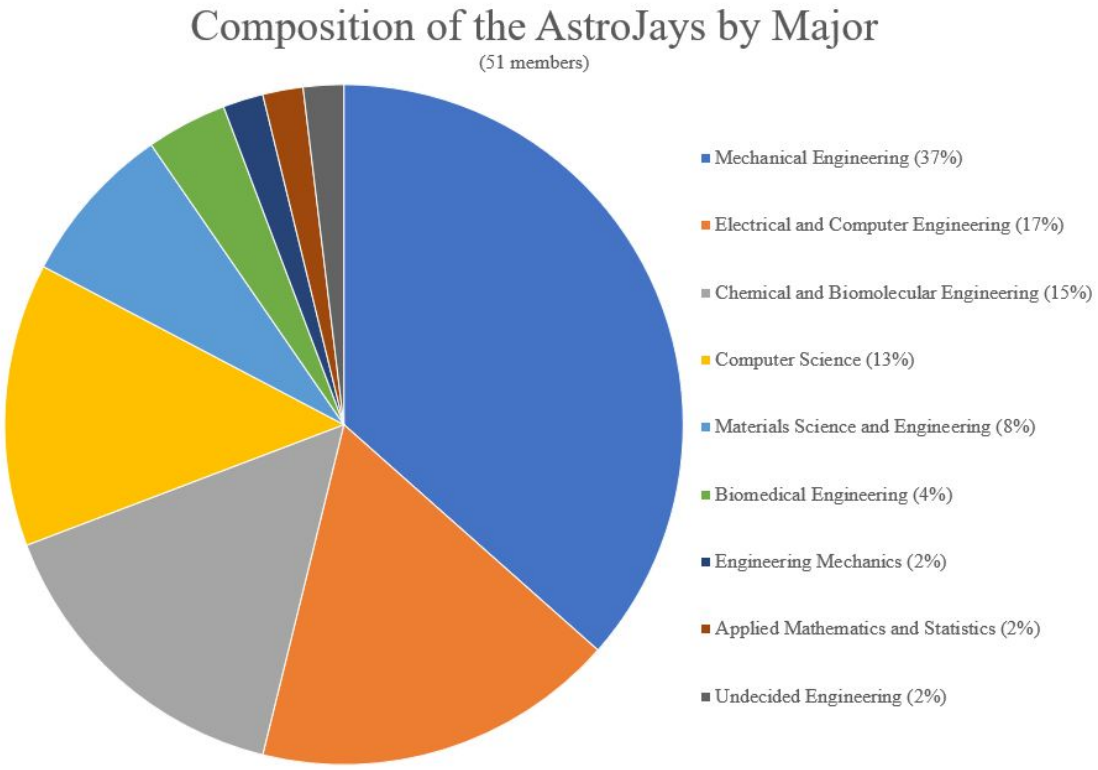
C. Team Organization and Schedule

Rocketry encompasses many disciplines and can engage individuals that are either experienced in rocketry or novices, so the work we do allows students from all years and departments to be able to be involved in the team, even if they haven't had any previous experience working on aerospace-related projects. The AstroJays is separated into six primary subsystems: avionics, payload, propulsion, recovery, simulations, and structure. Each have their own concentrations, and work in concert to make the rockets fly.



ASTROJAYS ORG. CHART

The AstroJays are composed of students from these majors, in order of percentage: Mechanical Engineering, Electrical and Computer Engineering, Chemical and Biomolecular Engineering, Computer Science, Materials Science and Engineering, Biomedical Engineering, Applied Mathematics and Statistics, Engineering Mechanics, and Undecided Engineering.



ASTROJAYS MAJOR DISTRIBUTION

From this comprehensive team makeup, a wholistic schedule was adopted -- pushing students from all disciplines to establish, pursue, and test innovative designs throughout the year leading up to the SAC.

July	• The AstroJays is born • Preliminary design process
August	• Purchase and assembly of Level 1 rockets • Academic year begins • Design process for Level 2 rocket begins
September	• Level 1 Certification launches • Organizing and building team • Define more clearly roles of subsystems
October	• Design revision, construction of Level 2 rocket in full swing • Testing of individual subsystems and team procedures • Begin hybrid engine development (not for SAC 2018)
November	• Theoretically fully-functional rocket assembled • Team procedures reevaluated • Design review
December	• Semester finals and winter break • Begin planning for improved integration
January	• Serious systems integration begins • Design review for Structure, Avionics, Recovery • Learn new techniques and skills to be utilized in final design
February	• Spring semester begins, recruit more team members • Begin their assimilation into the team • Launch with minor issues – rocket recovered with little damage • Critical design review begins
March	• Field set-up procedure finalized • Launch with issues – rocket destroyed • Carbon fiber research • Recovery re-design • Critical design review • Competition motor purchased
April	• Team elections, management turnover begins • Recovery system finalized • Carbon fiber layups begin • Relevant final simulations begin • Testing
May	• Semester finals and end of academic year • Documentation • Testing
June	• Final testing of individual subsystems • Final purchases of aesthetic parts • Final construction and integration of all subsystems • Compete in the Spaceport America Cup

ASTROJAYS 2017-2018 SCHEDULE

For the 2017-2018 academic year, team leadership consisted of an executive board of a President, Treasurer, and Secretary, and a set of subsystem leads, who headed each of the subsystems. The subsystem leads delegate team members in their work on each system of the rocket and coordinate the integration of these systems, and are ultimately more or less responsible for the success of their subsystem. The executive board oversees the logistics of the team's function and ensure that what needs to be done to keep everything running smoothly is done.

Each subsystem meets once or twice during the week as scheduled by its lead, or as needed when work is to be done (though significant individual work is done by leads and other dedicated members in some cases), and there is a weekly whole-team meeting to share the progress that has been made among the entire group.

IV. System Architecture Overview

A. Avionics

We will be employing an Arduino-based electronics system to control most of the functions of the rocket, due the ease of use and familiarity of our team with similar systems. A PCB (designed by us, printed by a commercial manufacturer) will contain the sensors needed to turn the board into an altimeter, accelerometer, and IMU, with an ATmega16u2 microcontroller (see *Avionics* in *Appendix* . In the shape of an Arduino MEGA board (correlating pin placement) to allow the mounting of a Dragino LoRa shield (used for data transmission between the rocket and the ground) directly onto the board.

The boards and their batteries will be placed in a robust electronics bay that will protect them as well as provide an easy way to insert and remove the electronics from the rocket. An umbilical power connection will supply the system with power while on the launchpad to prevent the drainage of the flight batteries.

B. Payload

Our payload consists of two major components, a visual tracking system and an experimental microfluidic serum separation payload. The visual tracking system aims to give us a visual understanding of our rocket's performance, while the experimental payload functions to assist the Johns Hopkins Microfluidics Laboratory in assessing the functionality of their experiments in a microgravity setting.

1. Microfluidic Serum Preparation Experimentation Payload:

The microfluidics laboratory at the Johns Hopkins University has recently developed a micro-total analysis platform in the form of a continuous isotachophoresis (ITP) device capable of extracting nucleic acids and proteins from known mixtures. The lab is currently researching the utilization of this device in space to aid in maintaining accurate medical records of astronauts in deep space missions. During deep space missions, astronauts are exposed to high level of radiation and unknown conditions which may cause shifts in their biological behaviors. For this reason, it is vital to be able to easily assess the medical status of a crew member. The ITP device we have decided to include in our rocket would aid in the separation and purification of nucleic acids from blood in a microgravity environment and would be used with *in situ* instrument systems such as PCR and mass spectroscopy to perform blood analyses in space.

We have decided to make this ITP device our rocket's payload to determine the device's functionality in microgravity. We also considered design revisions to make the device more robust, and capable of withstanding high level of vibrations during the rocket's ascension state. The microfluidics laboratory hopes to use this testing as a preliminary proof of concept.

This component of the payload will consist of:

- Chip (PDMS, Glass, Carbon Black)
- 4 - 9V Battery
- Tubing
- Samples
- 12 cryotubes
- Payload Bay

An example of the ITP device is shown in *Figure 1.* The experimental payload will be mounted on the rocket, on the payload bay, which is specifically designed to carry our payload and contains not only all the materials specified above for the successful operation of the experiment, but also controllers to autonomously activate the experiment at the desired altitude. The mount and all other components amount to a total weight of 8.10 lbs. This amounts to a total of 1.5kg for the experiment's necessary equipment.

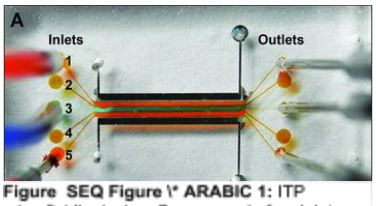


Figure SEQ Figure * ARABIC 1: ITP microfluidic device. Copmposed of an inlet stream on the left and two outlet streams on the right. The loaded sample prior to launch will be drawn from a cryo tube on the left of the device and will be separated into two cryo tubes on the left side

Figure 2: (A) Cryo tube used for sample delivery of 10mL total volume. (B) Cryo tube used for sample collection of 5mL total volume containg bright red dye to indicate a successful separation.

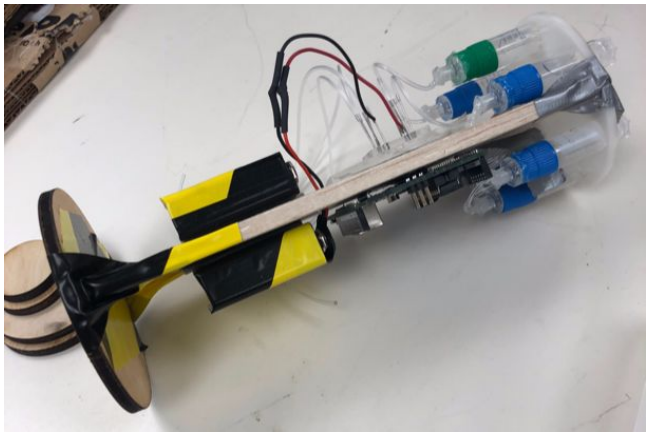
2. *Functionality:*

The experimental payload will be mounted on the rocket and will be operated from the ground. The avionics team will operate the experiment from ground control. Once the rocket is 100 meters below apoapsis the avionics team will allow for current to flow from the 9V batteries to the ITP device to allow for an electric field to form across the device. Once at apoapsis we will activate the suction pumps which will work to draw the sample fluid from the sample cryo tube containing mixture of Fluorescein isothiocyanate (FITC) and AVID through the electric field in the device and to the two sample collection cryotubes at the other end of the device (refer to the diagram for orientation). The FITC mixture will be negatively charged while the AVID solution will be positively charged representing the corresponding charges of DNA and proteins respectively. We have chosen not to use blood serum and instead to use this widely accepted mimic solution to prevent any hazards that may come from utilizing human serum, and to manipulate the solutions to survive for long periods of time in an in vitro environment exposed to such extreme temperatures. The separation works very quickly at such a micro-scale and should only take about 5 seconds to occur.

We expect to be in apoapsis for enough time for the experiment to occur without problems. We plan to include 10 mL of inlet sample in a 10 mL cryotube and use two 5 mL cryotubes to collect the separated solution. If the separation is successful our team will observe the initial sample solution (yellow) has been divided into the two 5 mL cryotubes now containing green- and red-dyed liquids. The observation will be made after recovery.

3. *Survivability:*

We have re-designed the ITP microfluidic device and all other components to withstand the extreme conditions of lift-off and landing, which should make it easy to recover. A scaled down version of the actual payload and its bay is shown below.



In order to assess the success of the experiment in microgravity we will check the two-sample acquisition cryotubes shown in *Figure 2A;* if the experiment is successful, there should be a clear separation of the green and red dye, meaning one tube should contain red liquid while the other should contain green liquid, as shown in *Figure 2B.* To quantify the success of our experiment, we will use a portable conductivity meter. The concentrated red dye, mostly composed of AVID, a highly electrophoretic component, will show a higher conductivity that the initial sample. A change in conductivity of >0.5uss should be enough to indicate a successful separation.

4. *Visual Tracking System:*

Our payload's second component will consist of a GoPro camera attached to the nose of the rocket (shown in *Figure 4*). The exact weight of the mount used to place the camera is still unknown and we are currently working on this part of the payload package. We are confident the weight will not significantly increase.

Figure 4: GoPro Camera used in our rocket.

The Go-Pro will be turned on at the launch pad and will record the flight from take-off to recovery. This payload component will only require operation before and after the launch and flight are complete.

The camera's will be loaded on the rocket with enough protection to withstand the rocket's maximum acceleration and all data will be recovered upon recovery of the entire rocket. The video footage will then be uploaded to computers onsite.

C. Propulsion

Being the first year of the AstroJays Rocketry Team's existence, as well as the first year of participation in the Spaceport America Cup, we opted for a COTS solid motor due to the relative reliability and integration feasibility of the technology while still meeting our mission's flight requirements.

The motor chosen for the competition launch is a N4100 from Cesaroni, with a total impulse of 17,790.3 N-s. This engine was chosen due to its thrust-curve meeting the flight requirements of our rocket, supplying the impulse and thrust levels required to reach the 30,000 feet competition apogee target. The launch rail departure velocity, calculated used the OpenRocket model, was determined to be 40.0 m/s. The launch rail length used in this calculation is the 17 ft long rail supplied at the competition.

D. Recovery

1. Recovery History

Our team has utilized several recovery systems during our inaugural year. Originally, a pyrotechnic event caused by gunpowder being ignited by the motor was employed to initiate separation and deploy a single parachute. Testing proved this method to be inconsistent and rudimentary. A compressed CO2 canister system was then created to remove the uncertainty associated with the combustion-based separation. Wanting to save resources, money, and preparation time, AstroJays began work on a reusable stage separation system.

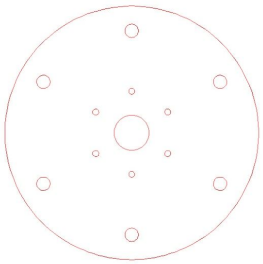
Our first iteration utilized linear actuators triggered by an altimeter in our avionics system and powered by a pair of 9 volt batteries. The actuators consisted of a rectangular body capable of moving cylindrical rods with attached triangular points. Their resting position was extended due to springs between the points and body. An upper coupler contained these actuators which depressed when the lower coupler was slid on (facilitated by the triangular point) and then extended into holes in the lower coupler, locking the two stages together. This connection provided the translational and rotational holding force needed to prevent premature separations from occurring. When the to couplers were joined, springs on the upper stage depressed, providing the separation force needed when the actuators were retracted at apogee. The upper coupler is shown in the figure below.

Upon separation, a drogue was released with pyrotechnic bolt cutters (triggered by the avionics system) a set time before the main parachute was deployed.

This system proved to be flawed in several ways. The two actuators were oriented vertically due to the limiting dimension of the outer body diameter. This was observed to cause an unwanted moment and misalignment when the two stages were attached. From this, we learned that the actuators were not an ideal size and thus not a viable component of our system. Additionally, the force caused by the springs caused issues with the attachment of the couplers to their respective rocket body halves. After several tests, the epoxy attaching the lower coupler to the outer body failed in shear. This particular issue inspired the next iteration of our recovery system. To avoid a constant large axial force normal to the coupler stages, we introduced a screw and nut system.

2. Our Solution

A 24 V 60 RPM DC motor was mounted on the upper stage of the rocket using six M3 screws and circular aluminum base plates epoxied to the rocket outer body. This mounting plate configuration is displayed below.



DC motor mounting plate configuration

It is powered by two 2200 mAh LiPo battery packs. An input signal from the altimeter turns on the motor through our circuit which contains a MOSFET transistor. The circuit is shown in *Recovery Circuitry* in *Appendix F*.

A ½”-8 ACME lead screw made of 1018 carbon steel was welded to a steel coupler which is attached to the motor pinion with a set screw. This interfaces with a precision ½”-8 ACME round nut attached to a 1” inner diameter PVC tube. This tube was mounted similarly to the motor with two base plates in the lower stage of the rocket. The components are shown in the figure below.

The rocket is assembled by screwing the upper stage onto the lower one. Separation is initiated at apogee by powering on the motor and allowing the screw to unscrew from the nut. The rotation of the lower half, caused by the torque of the motor, is prevented by using an interference fit between the upper and lower stage. The precision of the nut and screw also prevent motor torque from being transferred to the lower stage. The same drogue and main parachute setup is utilized with bolt cutters. This system proved a viable, reusable recovery method which is shown in full in Appendix F.

3. Analysis

With an upper body mass of 3.41 kg, the force exerted on the screw by the nut is 33.49 kN. Having a mean diameter of ½ " we find the Tensile-Stress Area is 0.1419 in^2 from Table 8-2 in . Using the equation below we obtain the stress in the screw.

$\sigma = F/A = 0.236$ MPa

This value is less than the yield strength of 1018 carbon steel so the screw will withstand this force.

The torque required to lift the upper stage of the rocket is given by the following equation from Shigley's.

$$T = F * d_m/2 * (l + \pi f d_m)/(\pi d_m - f l)$$

Where T is torque, F is force, d_m is the mean diameter of the screw, l is the thickness of one thread, and f is the coefficient of friction between lubricated steel and brass.

Using the values dictated by the geometry of the screw, we get a value of T = 12 lb-in which our motor is capable of providing.

With the rocket at an angle, the force on the screw becomes a bending moment. Assuming the screw acts as a cantilever beam and the perpendicular force is 2 kg, we know

$$\sigma = Fl/I = 64 * Fl/\pi d^4$$

Where F is the force, l is the length of the screw, I is the second moment of area of the screw (approximated as a circular cross-section), and d is the mean diameter.

We get $\sigma = 318\,MPa$ which is still less than the yield strength of 1018 carbon steel, so the screw can withstand this bending.

4. Tests and Results

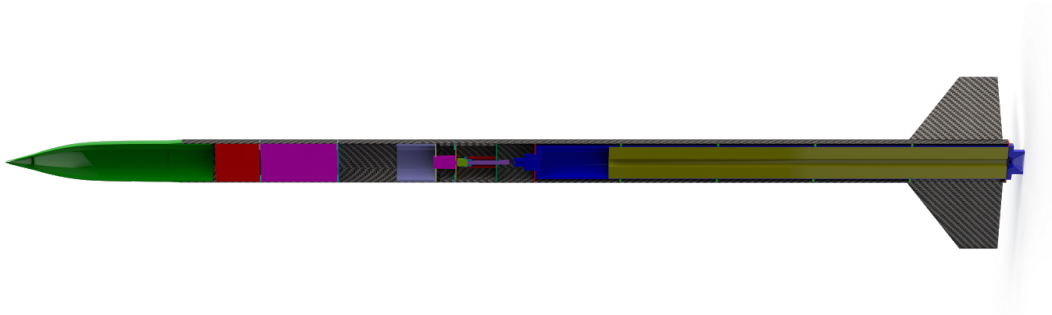
The recovery system is tested using a step-by-step process, to most effectively discover and eliminate problems that arise in the development of a new recovery system. These tests are summarized in *Recovery System Testing* under *Appendix B*.

The first test was a proof-of-concept which was done under no constraints. The free device only rotated initially but when rotationally constrained, functioned as intended. Next the recovery system was axially loaded with up to 16kg - nearly two times the weight of the rocket (*Rocket Parameters - Appendix A*). This validated the linear relationship seen between axial load, rotational speed, and power drawn. Next, the recovery device was loaded with 10kg and oriented in a non-vertical position such that the motor felt 8kg of axial force and 2kg of bending (cantilevered). This configuration severely strained the motor. It was only able to rotate upon vibrating or decreasing the bending moment. From this test configuration, it becomes apparent that the system is highly sensitive to bending moments. Several solutions are implemented to counteract bending moments on the lead screw/dc motor. A thick, rigid coupler composed of PVC tubing (see next section *Structure*) drastically reduces the bending moment on the dc motor. Additionally, the lead screw has been welded vertically by first making two tack welds, on opposite sides to reduce thermal bending. This is then followed up by a crescent weld - going two

Arrows describe welding motion when affixing the lead screw to the adaptor

half-circumferences about the screw from the same starting location. This symmetric approach decreases the thermal deformations. Combined, the two of these provide buffers between bending moments on the rocket and the recovery module. Next, the recovery system was controlled via a MOSFET with a variable power supply. The results matched the theoretical linear correlation between voltage, speed, and power drawn and proceeded as expected. However during this test, the insulation of the connecting wires pulled back far enough that the wires began to arc over the metal casing of the dc motor. From then on, all motor casings are electrically insulated. The last test completed as of yet is the parachute deployment. As the traditional method is too cumbersome for a compact recovery system and the parachute had to be wrapped about a central shaft without catching, other folding techniques were attempted. After a multitude of iterations, a reliable method was found, tested and certified. A picture-set demonstrates the successful unraveling in *Ground Testing* of *Appendix F*

E. Structure



1. Body Tube

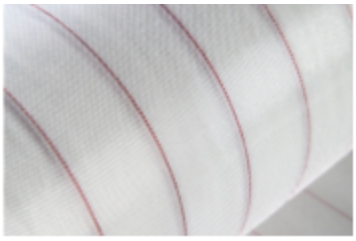
The overall rocket is constructed from three body tubes with an inner diameter of 4.5” and an outer diameter of 4.72”, each constructed via a carbon fiber layup. The carbon fiber used was plain weave with no tracers, and the epoxy used was made of Fibre Glast 2000 resin and Fibre Glast 2020/2060 cure- the 2020 was used for the payload tube and the 2060 for the upper tube and the motor tube. The tubes were wrapped around a mandrel constructed of Schedule 40 4” PVC, which was then wrapped with .005” mylar to ease the removal of the carbon

fiber tube after hardening. After the initial layup, the tubes were polished down to a smoother texture using

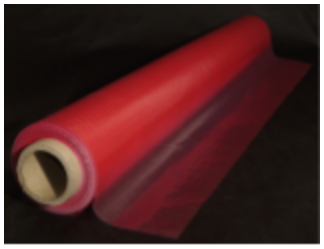
sandpaper of grits increasing from 100-1500, reapplying and hardening epoxy as needed to prevent sanding into the actual carbon fiber. Once each tube was smooth, it was finished with clear gloss spray paint to improve its appearance.

2. Vacuum Bagging Process

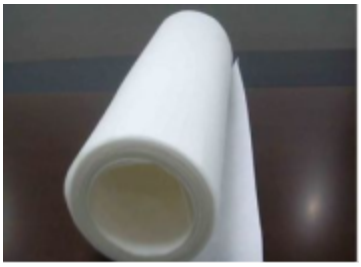
Once the carbon fiber has been completely wrapped around the mandrel and saturated with 1.2 times the weight of the carbon fiber in epoxy (see above picture), the piece is prepared for the vacuum bagging to produce an even finish. The piece is wrapped first in peel ply, then with parting cloth, and then with breather cloth, as seen in the pictures below.



1) Peel Ply

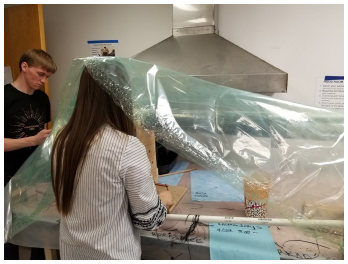


2) Parting Cloth



Breather Cloth

Then, the piece (along with the mandrel) is placed in a vacuum bag constructed of two sheets of plastic melted together with a heat gun, and the proper attachments to the vacuum pump are attached to the bag. The



vacuum is then turned on and left to run for several hours before finally removing the carbon fiber piece.

Piece with peel ply still attached

Piece with Peel Ply Removed

3. Couplers

The body tubes are connected by couplers constructed of Schedule 40 4" PVC, the same material used as the mandrel (This insures that the couplers outer diameter near perfectly matches the inner diameter of the body tube). The PVC is simply cut down to 3" and then a series of holes are drilled into it circling the tube. A matching set of holes are drilled into the body tube that the coupler is to be inserted into- this ensures that the body tubes can then simply be bolted together for the competition and then easily separated for transportation, construction and repairs. The coupler is then epoxied approximately halfway into the body tube with no holes drilled into it. Furthermore, these couplers are very rigid and absorb bending moments that could influence the rocket's internal structure or mechanisms.

4. Nose Cone

The COTS nose cone has a 4:1 ogive shape, and is made of fiberglass with a metal tip to better handle the heating caused by supersonic travel. It has an inner diameter of 4.37", an outer diameter of 4.48", and a length of 21.5".

5. Fins

The three fins are also made of fiberglass, specifically .125" thick G-10. The chosen shape is clipped delta, and the fins have a wedge shaped profile on both the leading and trailing edge for best performance and least drag while supersonic. These were ordered custom to have the following dimensions: Fin Span: 7.87", Height: 3.46", Sweep Angle: 45°.

6. Nose Cone Transition

As the closest COTS nose cone had an outer diameter of 4.5" but the body tubes have this as a inner diameter, a minor modification was needed to ensure the nose cone did not simply slide into the body tube. The 4" fiberglass coupler that came with the nose cone was thickened with carbon fiber until it reached an outer diameter of 4.5" to snugly fit within the body tube. This coupler was then stuck one inch within the upper stage body tube, and screwed in place. Finally, the body tube was sanded down into a gradient to meet the coupler to reduce drag.

7. Motor Retention

The motor retention system is simply a .125" thick aluminum thrust plate attaching the base of the rocket to the base of the motor, alongside a series of 4 evenly distributed wooden centering rings epoxied to the motor body tube to mainly provide support as opposed to strength.

8. Subassembly Interfacing

The interfacing between the structure and the sub-assemblies is provided by a series of .125" thick aluminum retention rings epoxied to the body tubes, with each pair of two rings surrounding each of the subassemblies (payload, avionics, recovery). These have a series of holes water-jetted into them in a circle around the ring to provide locations for the subassemblies to bolt into. This allows these subassemblies to be removed from the rocket when needed, but ensures that they are tightly attached to the rocket during the launch.

9. Finishing

Two launch lugs are then added for the attaching of the rocket to the launch stand, each with a height of .67" and a diameter of 1.18". Penultimately, the rocket is painted white to protect the structure, decrease drag, and help prevent the electronics from overheating. Finally, the logos of the sponsors are attached to the rocket.

The final overall specifications and dimensions of the rocket can be seen under *Rocket Information* in *Appendix A*.

F. Static and Dynamic Stability

1. Static Stability

Static flight stability occurs when the corrective torques induced by the rocket's body and fins are sufficient enough to counter deviations in the rocket's angle of attack. A good yardstick to adhere by is the Barrowman[1] rule of thumb which states that the rocket's CG is greater than one body diameter in front of the rocket's CP. To achieve static stability, the AstroJays have assumed the following approach:

- Determine rocket body length for DTF by defining a package size for every subassembly
- Implement detailed CAD model to extract weight and CG
- Choose arbitrary fin size and determine the CP computationally
- Iteratively adapt fin size to ensure a static stability margin of 1 to 2 calibers
- Perform flight simulations in OpenRocket and Matlab script/simulation to validate the stability margin

A virtual model of the rocket was created in OpenRocket - an off-the-shelf simulation for model rocketry. The static margin plot in *Stability Plots* in *Appendix A* shows the minimum stability margin for the flight obtained in OpenRocket when using the input dimensions and weights(see Open*Rocket Parameters* in *Appendix A*). However, OpenRocket is a black box -meaning it does not model the rocket's Inertia accurately and does not allow us to observe and adapt how aerodynamic coefficients and forces are calculated.

Due to this lack of transparency, flexibility and to account for more aspects such as fluid compressibility, a full flight simulator was drafted, edited, and implemented in MATLAB. This simulator is greatly influenced by Box et Hunt[8]. All atmospheric parameters as well as masses and inertias are updated throughout the flight. Aerodynamic coefficients for drag, lift and normal force are computed based on the current rocket and environmental properties according to Mandell[2] and the OpenRocket[3]. Furthermore, these coefficients are verified through ANSYS simulations for converging CFD drag coefficient solutions (see *Computational Fluid Dynamic Results* in *Appendix A*). These solutions are of extremely high fidelity and serve as a check for the MATLAB code. Through this, it was

discovered that about the transonic region, the MATLAB code becomes unreliable and its assumptions break down (see *ANSYS Drag Plot* in *Appendix A* and *MATLAB Drag Coefficients* in *Appendix A*). The resulting forces in axial and normal directions are then used to compute the rocket's state at each time interval. The rocket's state comprised of linear/angular velocities and linear/angular momentums. See for the calculation of all aerodynamic coefficients relevant for the simulation. (see *MATLAB Code* in *Appendix G*)

2. Dynamic Stability

Small disturbances cause the rocket to leave its theoretical trajectory, changing the angle of attack. Fins then induce a corrective moment to restore the original attitude. During this the rocket may overshoot and therefore tends to oscillate in the form of a nutation. In order to minimize oscillations and to ensure dynamic stability Mandell[2] proposes a damping ratio between 0.05 and 0.30. DTF shows a damping ratio of at least 0.025. throughout the flight (See *Stability Plots* in *Appendix A*). This indicates that the rocket will likely undergo some nutations upon leaving the launch rail which will rapidly decrease during the flight. This is reinforced as the static stability margin comes into play as the rocket reaches the critical velocity for stable flight.(see *Stability Plots* in *Appendix A)* Larger fins may be suggested to counteract these effects.

V. Mission Concept of Operations Overview

0. PRELAUNCH:

Assembly and Shipping:	The rocket (DTF) is transported on the ground to the competition site at which it will be fully assembled and given a final inspection.
Pre-launch:	This starts upon arriving at the competition site. DTF is thus prepared according our checklists. The phase ends when DTF is placed on the launchpad/launch rail.
Avionics Armed:	Students within 3 meters of the rocket shall wear safety glasses as well as the appropriate safety equipment. The avionics switched on and ready for flight. The phase green status LED will indicate that the power is connected, while a red LED indicates that the avionics are prepared for flight.
Recovery System Armed:	After arming the avionics, the designated responsible shall flipped the recovery switch on. A red(?) LED indicates that the system is live and ready for launch.

1. LAUNCH:

Ignition:	Starts when the propellant is ignited from an ematch and ends upon leaving the launch rail.
Lift-off Detection:	Avionics detect launch conditions and shifts from pad mode to flight mode.

2. ASCENT:

Powered ascent:	Rocket flies under the power of its motor for a duration of nearly 7 seconds.
Burnout:	Occurs when the motor has completely burned through its fuel mixture.
Coasting ascent:	After completing powered ascent, the rocket's forward moment continues its upwards trajectory until its vertical kinetic energy is spent.

3. APOGEE:

Apogee:	After spending its vertical kinetic energy, the rocket achieves its highest altitude.
Payload experiment runs:	Avionics detects apogee, running the payload experiment.

4. SEPARATION:

Stage separation:	Avionics waits for payload experiment to run, then initiates stage separation.
Drogue chute deploys:	Immediately after stage separation, the drogue chute is pulled out of its housing, and unfolds.

5. DESCENT:

Controlled high-speed descent:	After deploying the drogue chute, the rocket will undergo a controlled

high-speed descent of approximately 30 m/s.

6. DEPLOYMENT:

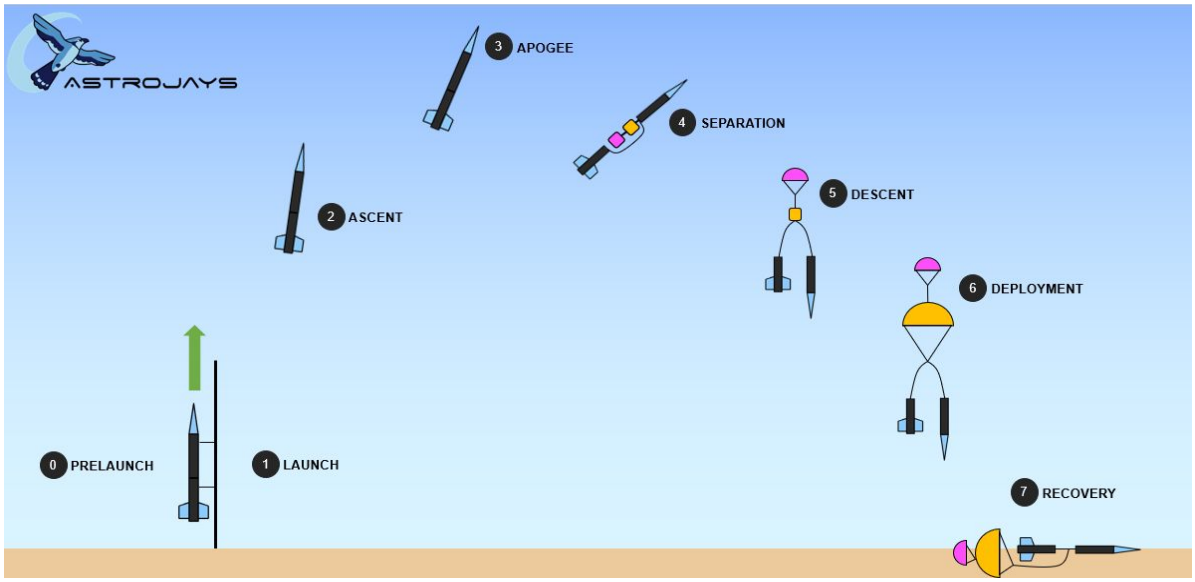
Main chute deployment: The phase is trigger at an altitude of 300 meters (1.000 ft) and cuts the reeving on the main parachute, deploying it.

Low-speed stabilized descent: Upon deploying the main parachute, the rocket's descent slows to 5 m/s. Ends upon landing back on earth.

7. RECOVERY:

Touchdown: The rocket settles on the ground, completing its flight.

Ground recovery: A ground crew follows a pre-defined search operations checklist and retrieves the rocket.



VI. Conclusions and Lessons Learned

The AstroJays seek to represent the pioneering and hard-working spirit of JHU. Within six intensive months DTF was built, a 2.8m long rocket with a COTS solid motor delivering a 4kg payload to 30,000ft at the 2018 Spaceport America Cup in New Mexico, USA. DTF carries custom-developed avionics with pressure sensors, IMU and telemetry in its bow section. During unpowered ascent, DTF runs a microfluidics experiment until apogee, which signals the recovery phase - complete with a dual-event recovery. DTF's subsystems have undergone significant testing in the past year and seeks to validate this in June. After the competition, data acquired shall be evaluated and used to improve the performance of future rocket endeavors for the AstroJays.

As the first JHU student team, AstroJays encountered many challenges, ranging from unknown technical and recovery problems, to organization, logistics, under estimating the required effort, and up to internal group dynamics. The biggest challenge -the way to the competition and the competition itself- is still ahead. Nevertheless, AstroJays has already arrived at a place we previously thought unimaginable. An industrial network adjoining academic and industrial partners has been established to foster this vision of annual student projects encompassing hands-on project experience and passion for engineering excellency.

We are most thankful to all our partners which share our passion and support this student initiative!

VII. Appendices

A. System Weights, Measures, and Performance Appendix

1. ROCKET INFORMATION:

Overall rocket parameters:

Parameter	Measurement (metric)	Measurement (imperial)
Airframe Length	2.79 m	107 in
Airframe Diameter	0.12 m	4.7 in
Nose Cone Length	0.61 m	24 in
Number of Fins	3	-
Fin-span:	0.2 m	7.87 in
Fin Height	0.088 m	3.46 in
Fin Sweep Length	0.088 m	3.46 in
Fin Sweep Angle (degrees)	45	-
Fin Thickness	0.003175 m	.125 in
Inertia: Ixx (kg*m^2)	8.081081	-
Inertia Iyy (kg*m^2)	0.307156	-
Inertia: Izz (kg*m^2)	8.081081	-
Launchpin Length	0.017 m	-
Launchpin Diameter	0.03 m	-
Vehicle Mass	22.155 kg	-
Center Mass	1.433 kg	-
Vehicle Weight	9.38 kg	20.68 lbs
Propellant Weight	9.099 kg	20.06 lbs
Payload Weight	3.69 kg	8.14 lbs
Liftoff Weight	22.166 kg	48.88 lbs
Number of Stages	1	-
Strap-on Booster Cluster	No	-

Propulsion Type	Solid	-
Propulsion Manufacturer	Commercial, Cesaroni	-
Kinetic Energy Dart	No	-

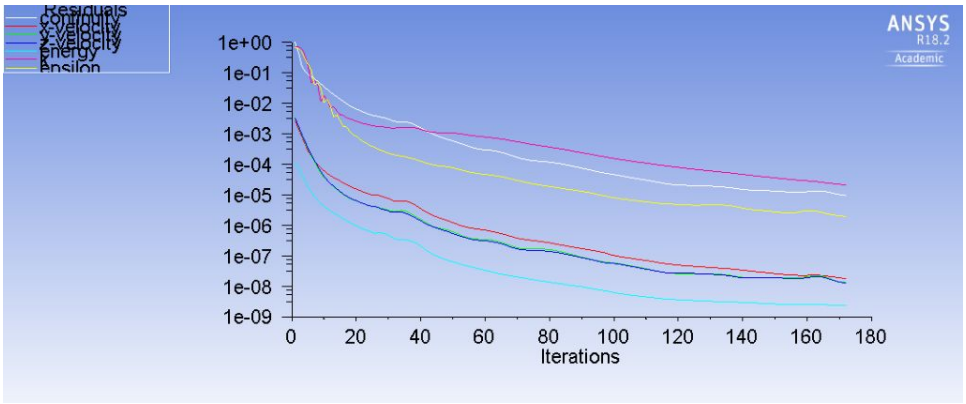
2. PROPULSION SYSTEMS:

Stage 1: Cesaroni, Pro98 17790N4100-P, N Class, 17790.3 Ns

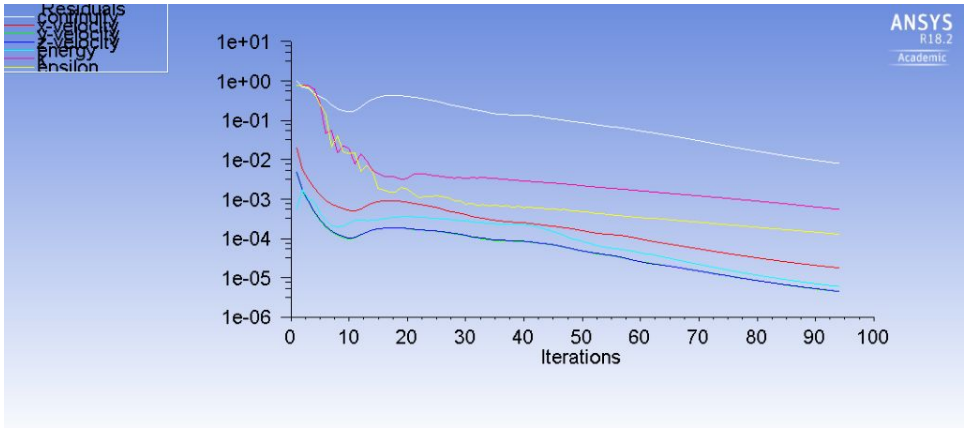
3. COMPUTATIONAL FLUID DYNAMICS RESULTS:

The following results were obtained using a k-epsilon model in the ANSYS 18.2 CFD solver. All calculations were performed by bisecting the rocket along a plane of symmetry, and running a flow simulation with around the bifurcated geometry. A y+ value of 1 mm was used for the turbulent flow, and a total of fifteen inflation layers were inserted into the mesh in order to capture boundary layers.

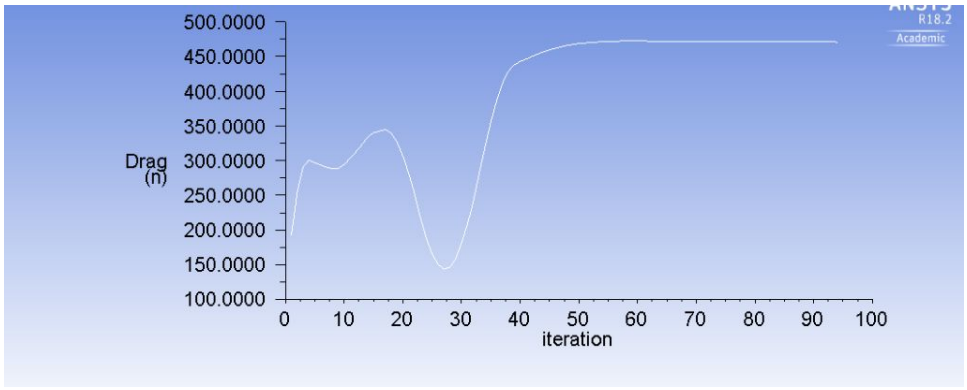
Reference Cross-Sectional Area	0.01380 m^2
Drag Force for Freestream Velocity of 165 m/s	86.6 N
Drag Force for Freestream Velocity of 446 m/s	941.7 N
Drag Force for Freestream Velocity of 274 m/s	247.6 N
Subsonic (Mach 0.5) Drag Coefficient	0.376
Supersonic (Mach 1.3) Drag Coefficient	0.560
Transonic (Mach 0.8) Drag Coefficient	0.390



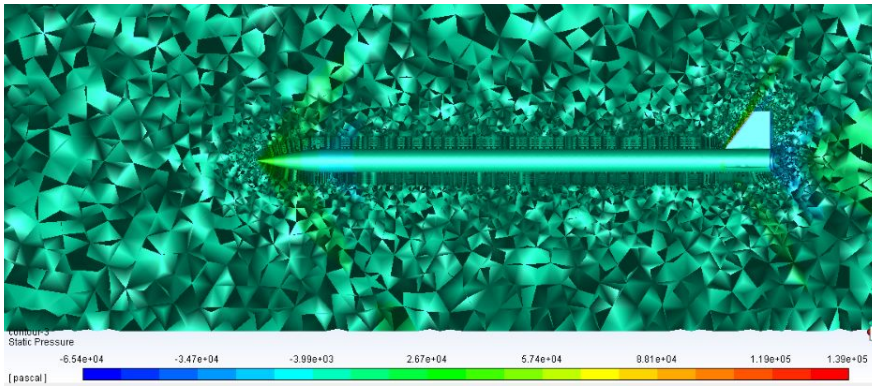
RESIDUALS FOR MODELING OF EXTERNAL AERODYNAMICS AT MACH 0.5



RESIDUALS FOR MODELING OF EXTERNAL AERODYNAMICS AT MACH 1.3



CONVERGENCE OF DRAG FORCE ON BISECTED GEOMETRY FOR FREESTREAM VELOCITY OF MACH 1.3

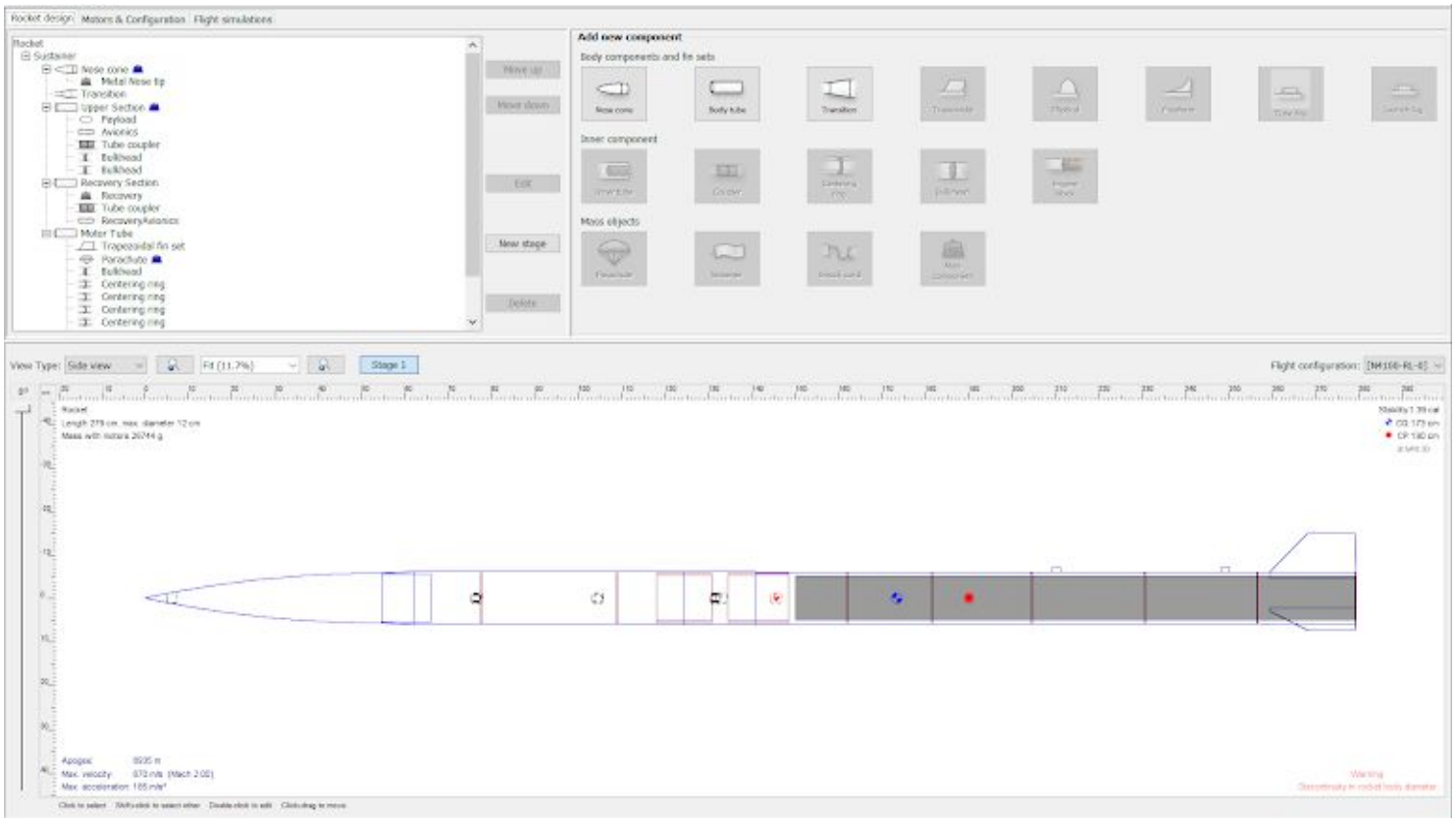


STATIC PRESSURE CONTOURS FOR FREESTREAM VELOCITY OF MACH 1.3

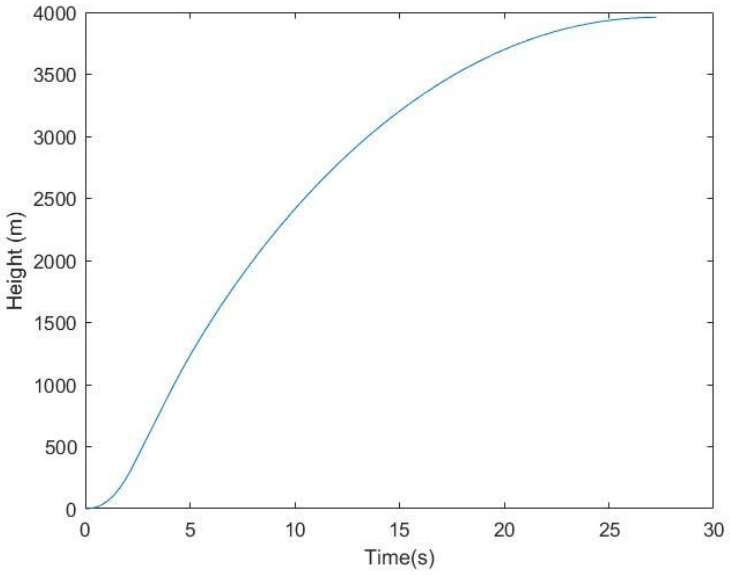
4. PREDICTED FLIGHT DATA AND ANALYSIS:

	Measurement

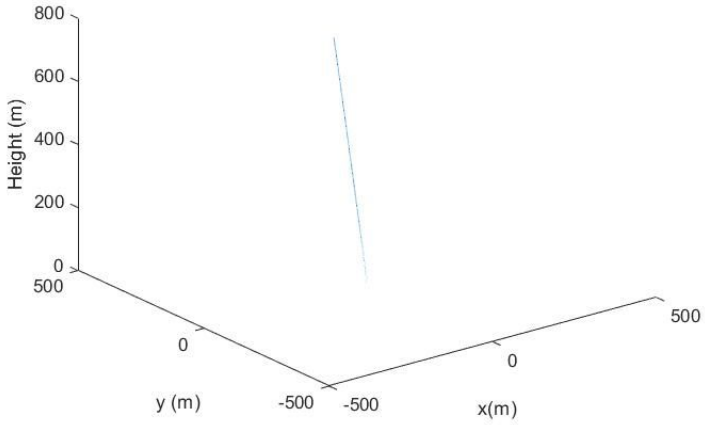
Launch Rail:	ESRA Provide Rail
Rail Length (feet):	17
Liftoff Thrust-Weight Ratio:	24.255
Launch Rail Departure Velocity (feet/second):	131
Minimum Static Margin during Boost:	1.39
Maximum Acceleration (G):	19.17
Maximum Velocity (feet/second):	2,220
Target Apogee (feet AGL):	30K
Predicted Apogee (feet AGL):	29,500



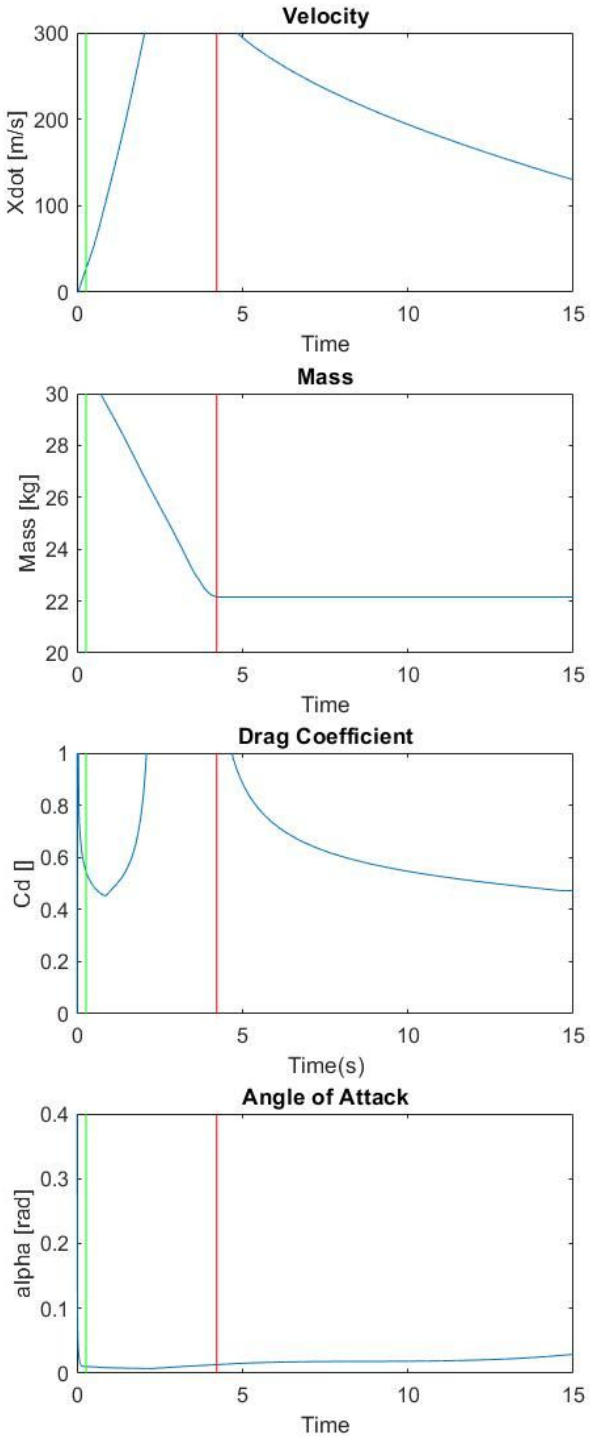
OpenRocket Parameters



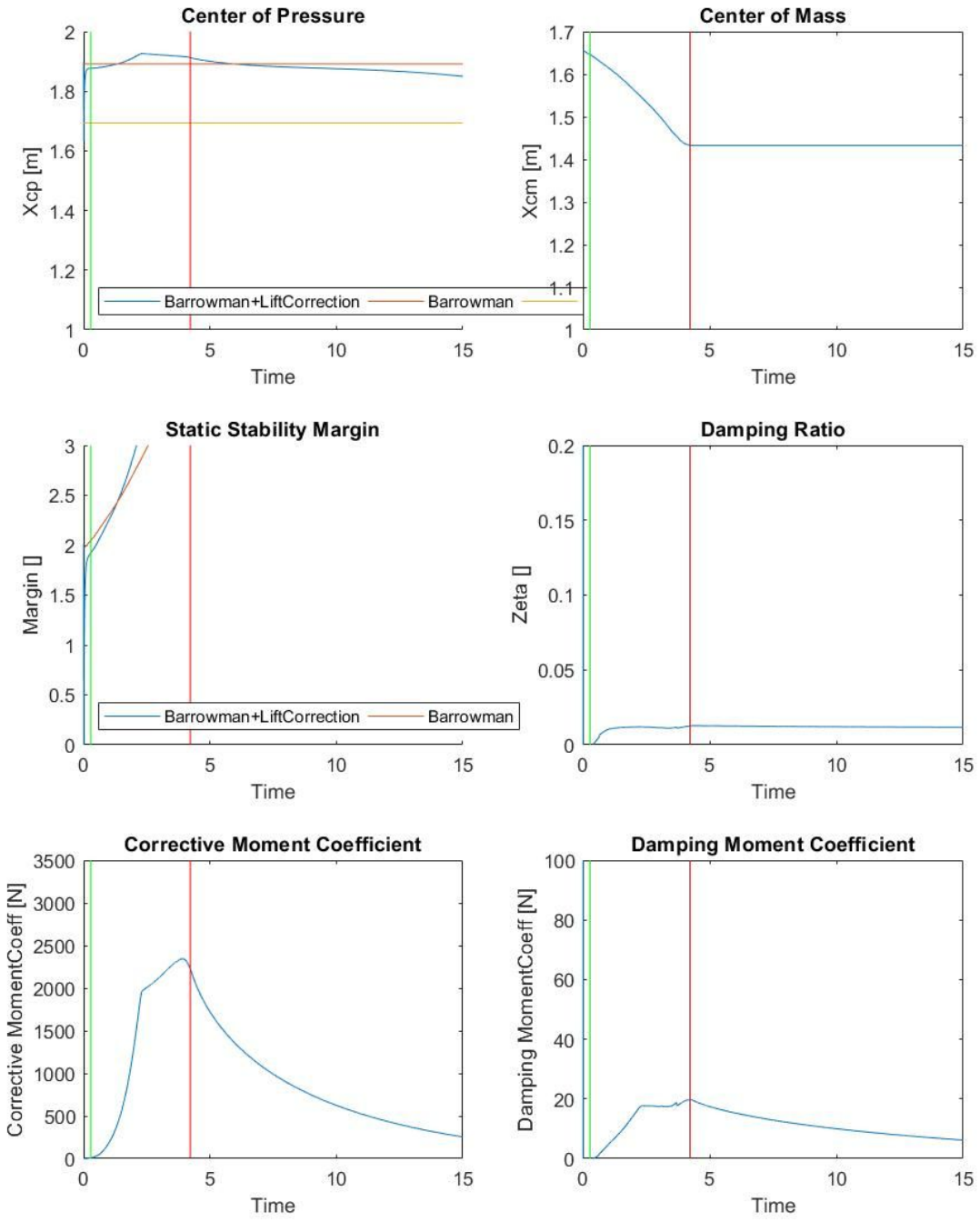
DTF altitude



Initial Flight Trajectory



Predicted flight Parameters of DTF in initial flight parameters



Stability Plots for DTF in the first portion of flight.

5. PLANNED TESTS:

Date	Type	Description	Status	Comments
11/17/17	Ground	Avionic Test of Reeving Mechanism (cable cutters)	Successful	~
11/18/17	In-Flight	~	~	Scrubbed
1/14/18	In-Flight	~	~	Scrubbed
2/17/18	In-Flight	Testing structure, new flight computer	Minor Issues	Nosecone with electronics separated from body, preventing main parachute deployment
3/17/18	In-Flight	Testing avionics, Level 2 Certification Attempt	Major Issues	Avionics or motor preparation issue -- did not separate
5/14/18	Ground	Overhauled avionics test of reeving mechanism (cable cutters)	Successful	~
5/19/18	In-Flight	~	~	Scrubbed
6/4/18	Ground	Static stage separation	TBD	~.
6/9/18	Ground	Avionics-Recovery stage separation	TBD	~

B. Project Test Reports Appendix

1. Recovery System Testing

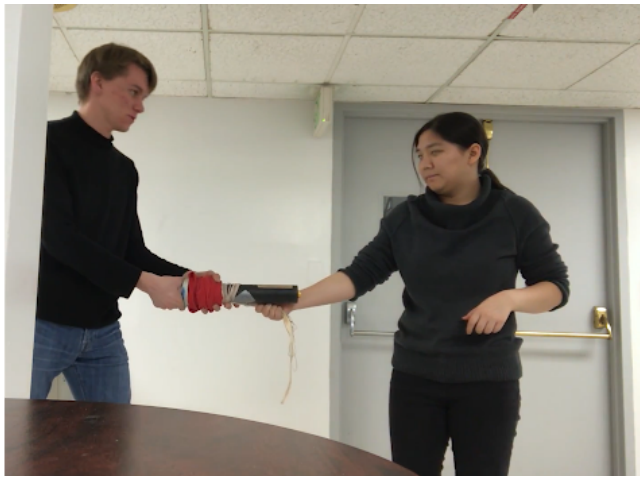
a. Tests Run:

Test Format:	Result:	Comments:
Free	Success	Requires a rotation lock to properly separate stages
Loaded, Axial (16kg)	Success	DC motor stayed within expected range of conditions
Loaded, Axial (10kg)+Cantilever Bending (2kg)	Limited Success	> Bending more concerning of two loading modes > Nutations induce bending Avoid bending as much as possible
Electronically Controlled, via MOSFET and variable power source	Success	> DC motor must be completely insulated or else it may arc across metal casing > Expected electro-mechanical

		loads matched measurements (i.e. Voltage vs. Speed, force output)
Parachute Deployment	Success	> No spring needed as mechanism ensures complete separation > Developed, tested new parachute folding method due to traditional technique too cumbersome
Avionics Controlled	TBD 06/02/18	N/A
Integrated, Static	TBD 06/06/18	N/A
Integrated, Simulated	TBD 06/07/18	N/A

b. Ground Test Demonstration

Successful Parachute Wrapping Test:

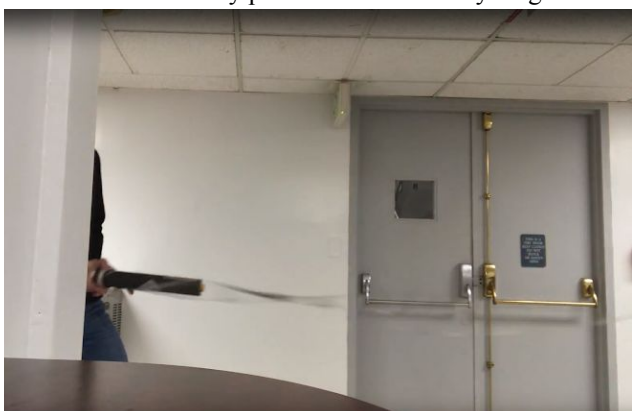


Initial Position of wrapped parachute

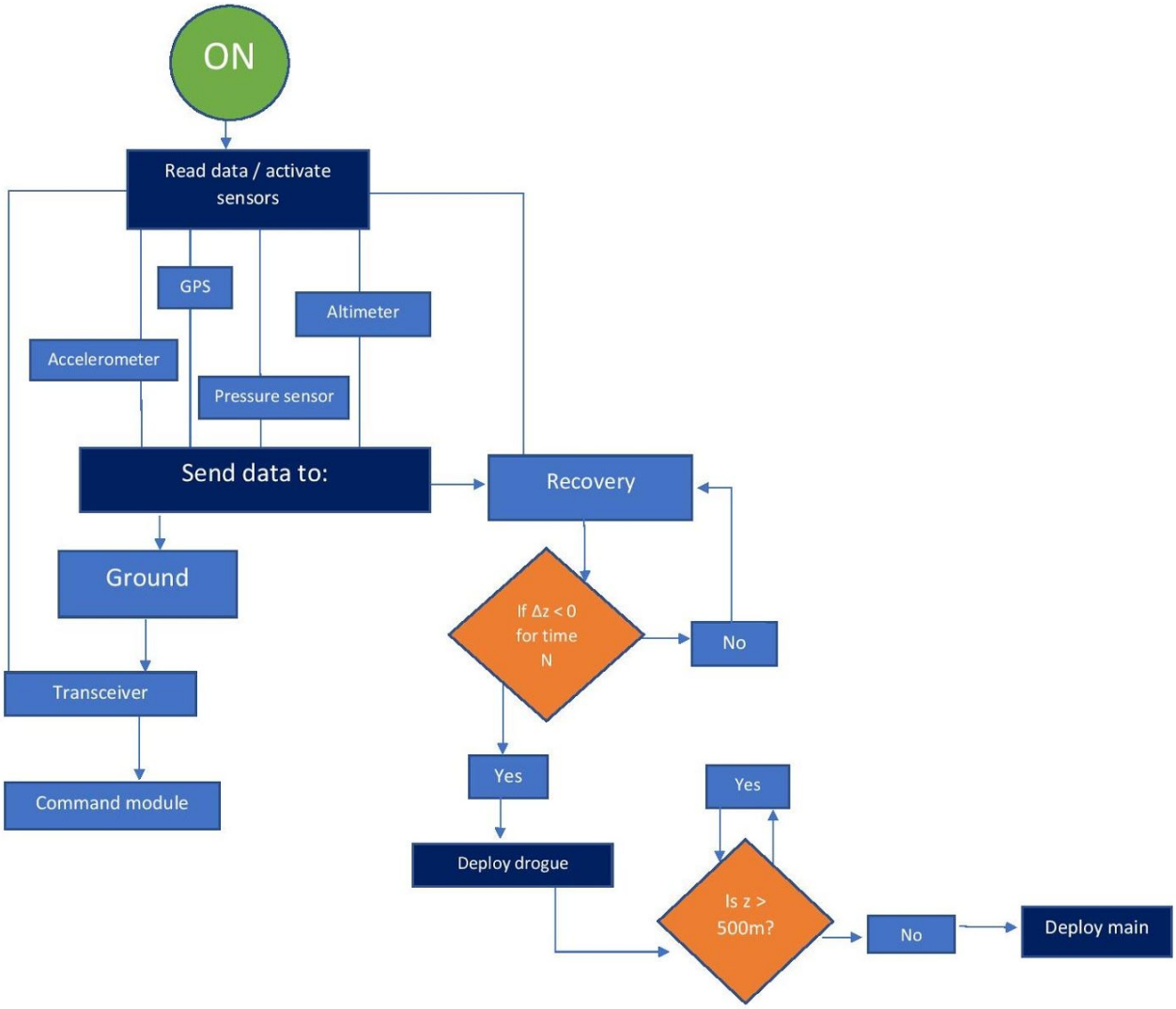
Unraveling begins

Parachute parallel to unraveling shock cord

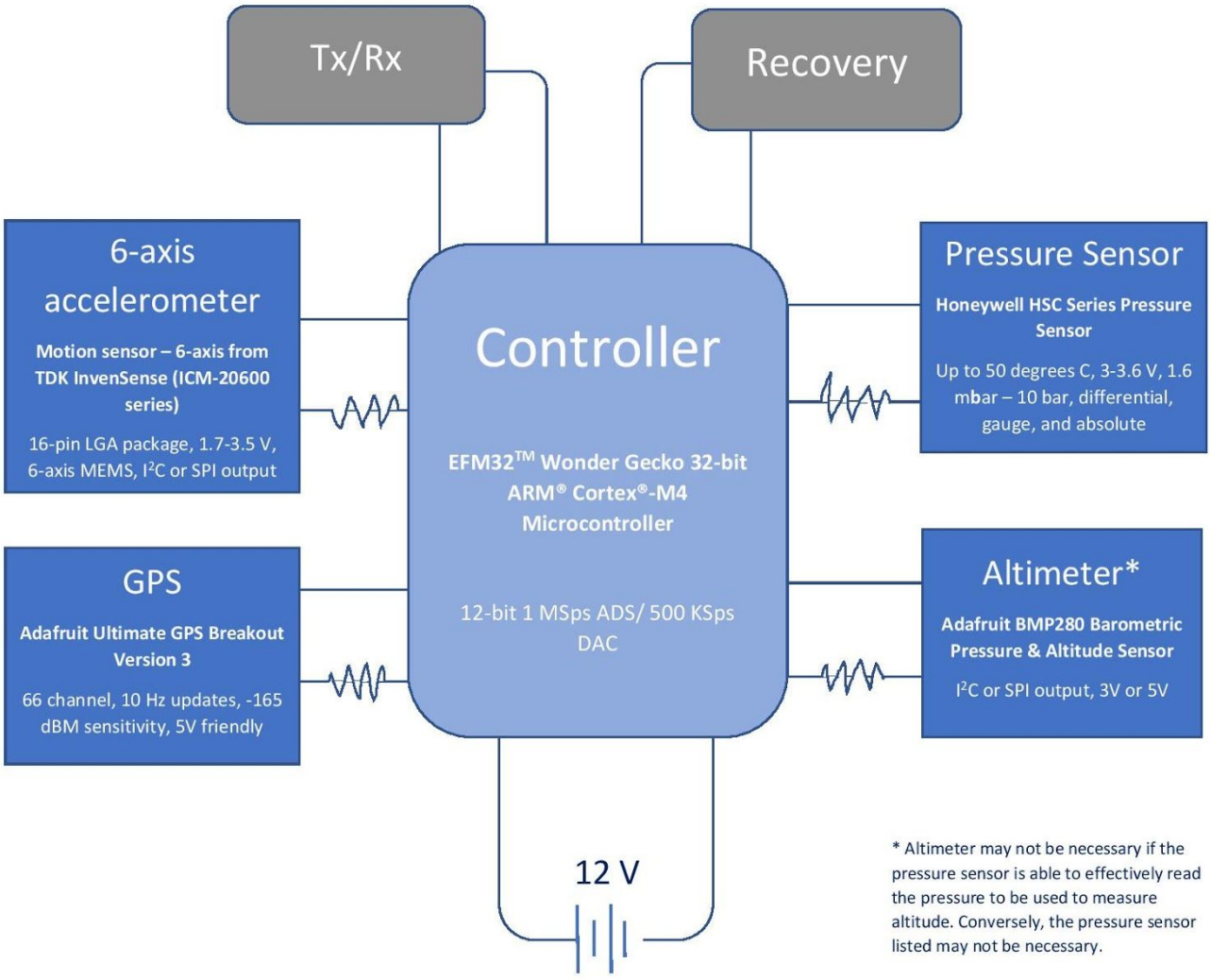
Parachute successfully pulled out without any tangles and begins to deploy



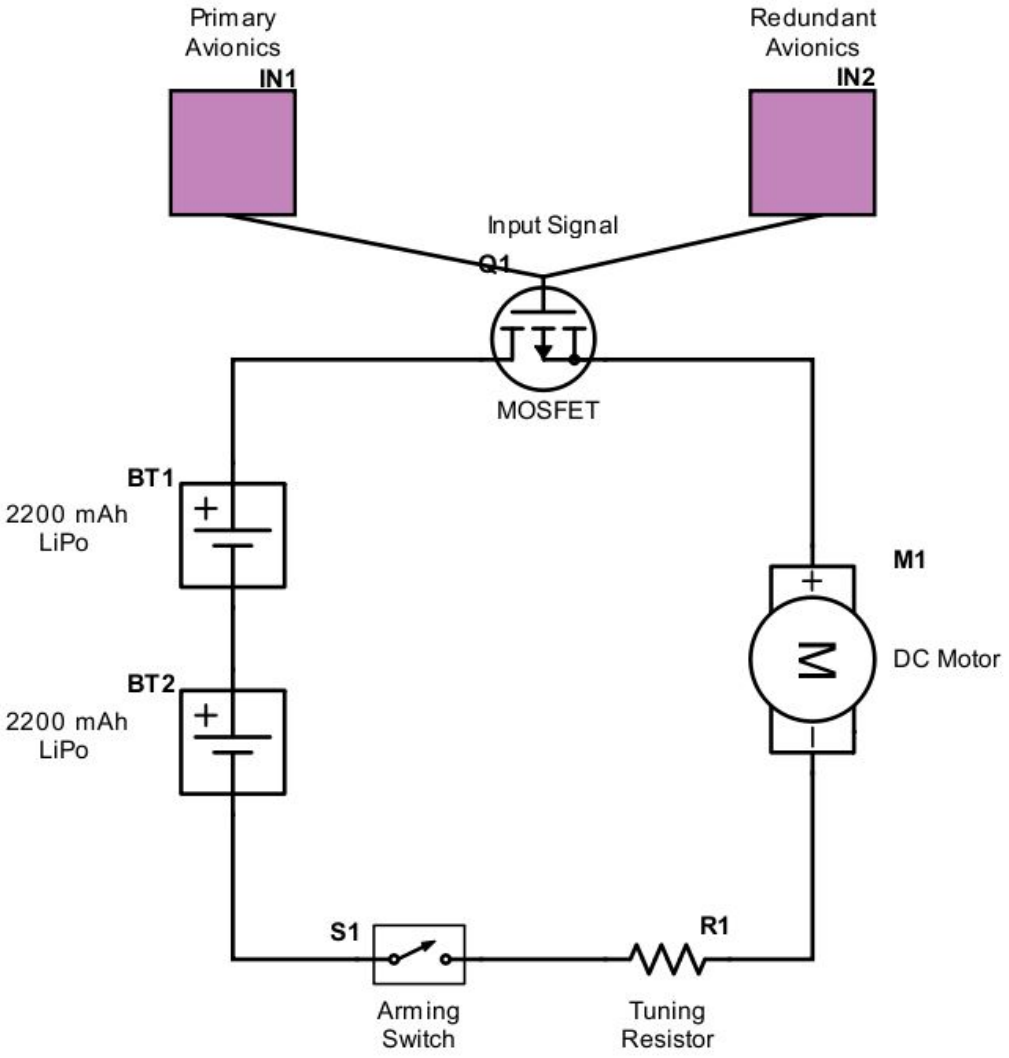
Shock cord completes its unravel and parachute remains free to deploy



Avionics Status Flow Chart



Avioncis System Module



Recovery Circuitry Schematic

c. Flight Test Demonstration

Avionics were tested on multiple occasions, and which led to valuable insights in the avionic code. During the first launch attempt, wires were accidentally crossed leading to an accidental discharge of the capacitors - scrubbing the launch. After having insulated the capacitor wires, there occured a second attempt. However during the loading process of the rocket, the capacitors discharged once more. This has led the AstroJays to come to the conclusion that breadboards with openly accessible wires must equally be insultated - as well as all electricalconnections within the rocket. Having taken care of this issue, the following rocket launched successfully. However, due to an incorrect locking of the structure, the nosecone (and avionics housed inside) were ejected. Thankfully, the rocket itself sustained only minor damage and all subsystems were recovered. Lastly, during the final launch, a black-powder stage separation was adopted with the remaining systems updated. After a favorable

launch, the rocket returned through cloud cover -without any stage separation, resulting in a litho-braking event. Upon forensic inspection of the remains, it was determined that the black powder (Pyrodex) utilized was a variant with a slower burn. The Pyrodex burned it's first layer, beneath the powder and before the rest of the Pyrodex could burn, the resulting gas blew this powder up and out of its container. This incomplete burn thus failed to produced enough internal pressure to separate the two rocket stages.

d. SRAD Propulsion Testing / SRAD Pressure Vessel Testing

NOT APPLICABLE: THIS SECTION IS INTENTIONALLY LEFT BLANK

C. Hazard Analysis Appendix

Team 124	**Down To Fly**	**19 May 2018**		
Hazard	**Possible Causes**	**Risk of Mishap and Rationale**	**Mitigation Approach**	**Risk of Injury after Mitigation**
Unintentional ignition of Pyrodex/black powder	Inappropriate storage, unsafe transport, unknown spills	Low; we understand the dangers of working with black powder -- however, accidents can happen	Stored in a fire cabinet in the lab, in a tightly sealed plastic jar. Closed immediately after getting required powder to prevent spills	Low
Unintentional motor ignition while not attached to a rocket	Manufacturer error, unsafe storage or transport	Low; a COTS motor from a reputable manufacturer is unlikely to have a malfunction that would make it susceptible to such an accident	Motors are kept in a fire safe, packaged in foam so that they can't roll around or bump into anything. When being transported, they are kept in this case. Motors are kept in the case except when being brought out for integration onto the rocket	Low

D. Risk Assessment Appendix

Team 124	**Down To Fly**	**19 May 2018**		
Hazard	**Possible Causes**	**Risk of Mishap**	**Mitigation Approach**	**Risk of Injury**

		and Rationale		**after Mitigation**
Explosion of solid-propellant rocket motor during launch with blast or flying debris causing injury	Manufacturer error, personnel too close	Low; using COTS motor with testing and regulations	Purchase motors from reputable manufacturers, personnel stand at least 200 ft from rocket at launch	Low
Rocket deviates from nominal flight path, comes in contact with personnel at high speed	Human error in manufacturing, rocket is unable to stabilize quick enough	Low-Moderate; it is entirely possible that deviation occurs, but that it would be so severe is unlikely	Decreased rocket mass, larger motor, longer launch rail means that the rocket will stabilize quicker, preventing a large deviation from the nominal flight path	Low
Recovery system fails to deploy, rocket or payload comes in contact with personnel	Sensor failure, poor electrical connections, out of battery, code issues	Moderate; student-built and assembled system and electronics could have failures, have had issues in past with electronics	Double redundancy for recovery system, proper soldering technique, battery redundancy, lots of testing	Low-Moderate
Recovery system partially deploys, rocket or payload comes in contact with personnel	Out of battery, unforeseen issue with recovery system	Moderate; recovery system is the most experimental system, has yet to have a proper in-flight test	Lots of ground testing, battery redundancy,	Low-Moderate
Recovery system deploys during assembly or prelaunch, causing injury	Poor electrical connections, avionics system malfunction, crossed wires	Low; primary separation system is not explosive, and deploys relatively slowly, cable cutters are mostly contained and will be assembled and set up with caution	Recovery system cannot deploy via correct electronics function due to the nature of the code; using proper technique when soldering, utilizing repetitive testing. Set-up will be careful	Low
Main parachute deploys at or near apogee, rocket or payload drifts to highway(s)	Poor electrical connections, sensor malfunction, code issues	Low; if parachute deploys properly, most aspects of the avionics are functioning as they should	Recovery system cannot deploy via correct electronics function due to the nature of the code, sensors made by reputable manufacturers	Low

Rocket does not ignite when command is given ("hang fire"), but does ignite when team approaches to troubleshoot	Motor malfunction, incorrect motor setup	Moderate; while motor manufacturer is reputable, incorrect setup can ruin a launch	Our team is diligent in its motor preparation due to past errors and failed flights	Low
Rocket falls from launch rail during prelaunch preparations, causing injury	Strong wind, incorrect mounting of launch lugs onto rocket and rail	Low; launch lugs have been secure enough for our tall rockets in the past even in strong wind	Take time to accurately and securely place launch lugs on rocket	Low

E. Assembly, Preflight, and Launch Checklists Appendix

1. Rocket Field Setup and Checklist

1) DISCHARGE CAPACITORS with spare wire, voltmeter
2) Run the recovery wires through body tube if they are not already there
3) Set up two cable cutters (with e-matches) attached to one zip-tie, WITHOUT tying the zip-tie
4) Coat the parachute in chalk powder and then fold it according to parachute folding instructions (when wrapping line, wrap first pass UNDER tip, and make sure the line does not cross)
5) Tie the zip-tie with the cord cutters onto parachute, then snip the end of the zip-tie
6) Tie the end of white rope to the eyehole screwed into the upper segment of the rocket
7) Tie the parachutes to the white rope
8) Attach the e-matches to recovery wires, using the clamp
9) Loosely zip-tie the recovery wires to the white rope, tugging excess wire towards the nosecone where there is more room to store it, and snip the ends of the zip-ties
10) Attach the drogue parachute with chalk
11) Plug in the 9V batteries (cord cutters now LIVE) (green "on" LED on)
12) Arm the avionics by switch (red "armed" LED on), check that the LEDs on the boards are lit (press reset for good measure)
13) Mount the electronics bay into the body tube, coiling excess wire underneath carefully, leaving the nosecone off for now
14) Coil the white rope and the recovery wires around the recovery mechanism and stuff the parachute and drogue into the gap in between the mechanism and the walls of the rocket
15) Mount the nosecone onto the rocket
16) Slide the upper section of the rocket onto the center recovery section and bolt in place
17) Rotate the joined recovery & upper section onto the lower section and lock in place.
18) Attach the launch lugs
19) Slide the rocket onto the launch rail
20) Attach umbilical power
21) Install ematches/igniters onto the motor
22) Go for launch

2. How to Fold the Parachute

1. Tie end to base stage
2. Make sure, there are no protrusions from the base
    a. If so, add padding foam to level the protrusion
3. Tightly wrap shock cord clockwise - no higher than 1cm

    a. If must continue, add new winding location above
4. Once near end of cord + reached parachute
    a. Bring free end up (higher) on tube
    b. Wrap parachute (and strings) **downwards** and **counter clockwise** around tube
    c. Make sure both sides of shock cord connected to parachute are **above** parachute windings
5. Compress wraps towards one another
    a. Ex. 3-4 inches tall around 54mm tube
6. Tie free end to opposite stage

3. Equipment

Rocket:

A. Structure
    a. Rocket
        i. Nosecone
        ii. Adaptor
        iii. Upper stage
        iv. Recovery Module
        v. Lower stage
    b. Launch Lugs
B. Propulsion
    a. N4100 Engine
    b. 98mm 6XL Hardware/Casing
C. Recovery
    a. Parachutes
        i. Main
        ii. Drogue
    b. Cable Cutters
        i. Hardware
        ii. Black Powder
    c. Recovery Module
        i. Lead screw
        ii. Lead nut
        iii. Attachments
        iv. DC Motor
        v. 2 LiPo batteries
            1. 2200mAh
D. Avionics
    a. Flight computers
        i. Avionics module
        ii. Redundant module
    b. Sensors
        i. Accelerometers (x2)
        ii. Altimeters (x2)
        iii. GPS
        iv. IMU
        v. SD reader
        vi. Antennae
    c. Batteries
        i. 9V batteries
        ii. 1 2200mAh LiPo
E. Other
    a. Binoculars

Tools:

- Screwdrivers (multiple sizes each)
    - Cross
    - Flathead
    - Hex
        - Allen wrenches acceptable
- Wire Strippers
- Wire cutters
- Scissors
- Tape Measures
- Cordless Drill
    - Drill bits
    - Screw bits
- Sandpaper (grit)
    - 220
    - 400
    - 800
    - 1200

Duplicate equipment:

- Transistors (NPN)
- MOSFETs
- Wires
- XT60 Connections
- Twist-ons
- Nuts and Screws
    - M3
    - M6
    - 5/16”
- SD cards and Readers
-

Dispensibles

- 5 Minute Epoxy
- Glue
- Zip-ties
- Heat Shrink Tube
- Electrical Tape
- Duct Tape

Setup:

- Printed Master Checklist
    - 3 printed
        - 1 posted for all to see
        - 2 extras/replacements
- Tarp
    - Must fit under Canopy
- Chairs
    - 3-4 foldable ones
- Canopy
    - Must be able to fit 2 persons beneath, lying down
    - No tears, preferably
- Tables

    - Workbenches long enough for assembled rocket
    - Foldable
- Cooler with (any):
    - Ice
    - Dry ice
    - Ice Packs

4. Personal Items Checklist

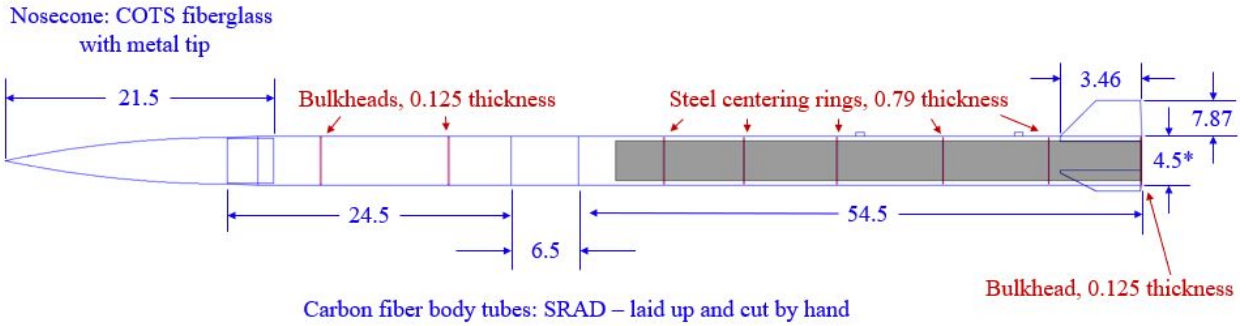
A. TO WEAR:

- Sunglasses
    - Polarized
- Hat
    - Must cover head
- Shirts
    - 1 Professional Polo or Shirtwear
    - Extra shirt
- Boots
    - Open footwear will not be tolerated
- Socks
    - Socks are required
- Bandana - for dust
    - Recommend 2-3 bandanas/person

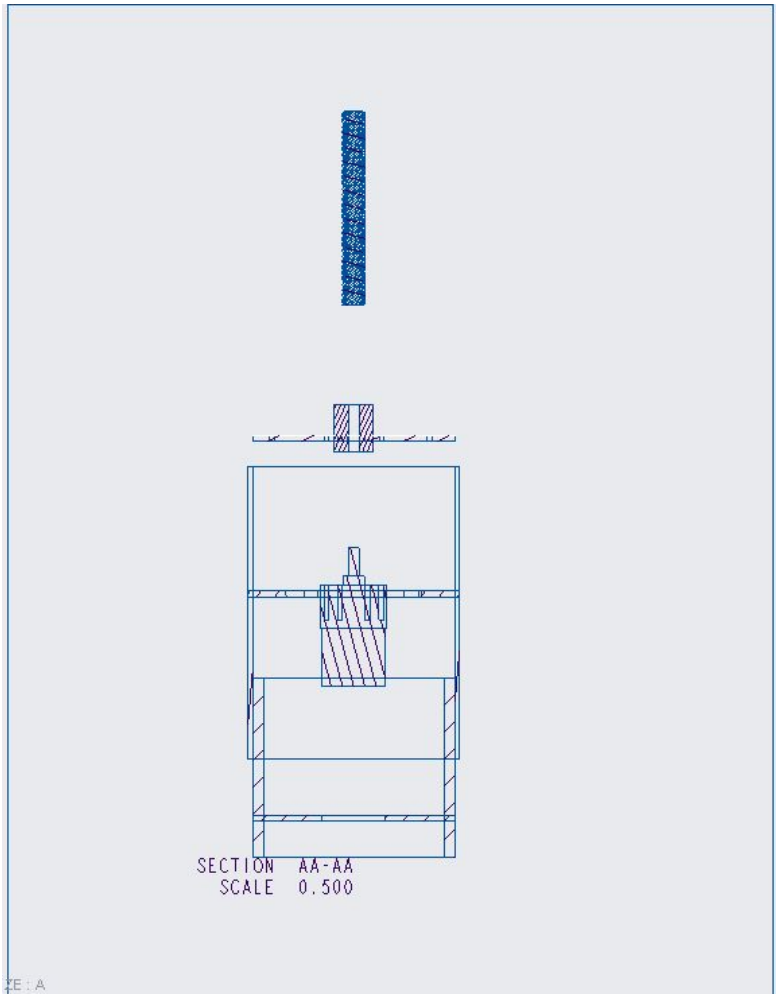
B. TO BRING:

- FIRST AID KIT
    - 1 minimum per team
    - Just in case
- WATER
    - 2 gal/day/person
    - Emergency Water
- Food/snacks
    - 1 Lunch per person
    - 2 Snacks
- Walkie-Talkies
    - Phones with connection may replace these
- Notebook
    - To jot down any anomalies
- Pencil/Pen
    - Multiple ones
    - Sharpies as well
- Sunscreen
- A good attitude

F. Engineering Drawings Appendix



Structure Diagram (measurements in inches)
*Inner diameter, outer diameter = 4.72 in
Fins are made-to-order fiberglass with a wedge shaped profile to reduce issues when going supersonic



Cross Section of Recovery Module

G. MATLAB Code Appendix

Created by Hassan Arif, Team RORO. [7]
Revised and Adapted by: Stephane Teste, AstroJays

Data Files:

Rocket_prop_DTF2.txt

```
Lenght,            2.79
Cone Lenght,    0.61
Diameter,         0.12
Number of Fins, 3
Fin height,        8.8e-2
Fin base length,20e-2
Fin top length, 8.8e-2
Fin Sweep angle,45
Fin thickness,  3.175e-2
Mass dry,       22.155
Ixx,             0.1049605
Iyy,             2.761e+02
Izz,             2.761e+02
Xcm dry,           1.433
Launchpin height, 17e-3
Launchpin Dia,    30e-3
Trim mass,        0.0
Cesaroni_N4100.eng, 0
```

Cesaroni_N4100.eng (motor data from ThrustCurves [3])

```
; Cesaroni N4100RL
; converted from TMT test stand data
; Pro98-6GXL 17790N4100-RL P
N4100-RL 98 1293 P 9.38 14.748000000000001 CTI
  0.0030 203.877
  0.05 2362.879
  0.078 3946.845
  0.121 4281.412
  0.652 4370.281
  1.123 4453.923
  1.655 4772.807
  2.353 4621.206
  3.035 4511.427
  3.7 4375.509
  3.733 4182.087
  3.887 2969.282
  4.036 1589.193
```

4.197 533.216
4.262 240.47
4.3 0.0

Program Files:

main.m

```
%% Main
clear all; clc; close all

% Global varialbes


global env
global log

% Create rocket class

DTF = rocket(init_rocket());% creates class with the initial values
%%
motor_init( DTF ); %loads rocket motor
%%
% Initilize Environmental variables
% optional argument: Elevation(m) Temperature(C)and Pressure(Pa)
env = environement(350, 15, 86000, DTF );

%% Phase: Accent
tend=30;
[t, state] = accent_calc(DTF,tend);
%%
% figure(1);
plot(t,state(:,3))
xlabel('Time(s)')
ylabel('Height (m)')
%
figure(2);
plot3(state(:,1),state(:,2),state(:,3))
xlabel('x(m)')
ylabel('y (m)')
zlabel('Height (m)')
axis([-500 500 -500 500 0 800])
h_max=max(state(:,3))
```

```
%%
extract_data ( state,t);

%% Debugging plots
% figure(3)
%
% plot(log(:,10),log(:,2))
%
% xlabel('Time')
% ylabel('Value1, Value2')
% %axis([0 20 0.0 1])
% %
% figure(4)
% plot(log(:,10),log(:,1))
% xlabel('Time')
% ylabel('Value1, Value2')

%% Plot flight and stability data
```

ascent_calc.m

```
function  [t, state] = accent_calc( DTF,tend )
%Function calculates the assent phase of the rocket
        global env;
        global log;
        state_0 = [DTF.X; DTF.Q; DTF.P; DTF.L];
        tspan = [0:0.005:tend];

        % Event function to stop at max height
        options = odeset('Events',@event_function);

        % Solve flight using ODE45
        [t, state]= ode45(@flight,tspan,state_0,options);

        % ------------------------------------------------------------------------
        %% Equations of motion discribed to be sloved in ode45
        function state_dot = flight(t,state)
        %TODO: put condition on burn data so it does not excecute after
        %bunout

        if (t>0)
        DTF.deltat = t - DTF.time;
        DTF.time = t;
        burn_data(DTF); % runs each cycle to update motor stats

        end
        X= state(1:3);
        Q= state(4:7);
```

```
P= state(8:10);
L= state(11:13);

DTF.X= state(1:3);
DTF.Q= state(4:7);
DTF.P= state(8:10);
DTF.L= state(11:13);
% Rotation matrix for transforming body coord to ground coord
Rmatrix= quat2rotm(DTF.Q');

% Axis wrt earth coord
YA = Rmatrix*env.YA0';
PA = Rmatrix*env.PA0';
RA = Rmatrix*env.RA0';
CnXcp = DTF.CnXcp;
Cn= CnXcp(1);
Xcp= CnXcp(2);
Cda = CnXcp(3); % Damping coefficient
zeta = CnXcp(4); % Damping ratio
Ssm = CnXcp(5); % Static stability margin
%% ------- X Velocity-------
Xdot=P./DTF.Mass;

%% ------- Q Angular velocity--------- in quarternians
invIbody = DTF.Ibody\eye(3); %inv(DTF.Ibody); inverting matrix
omega = Rmatrix*invIbody*Rmatrix'*L;
s = Q(1);
v =[Q(1); Q(2); Q(3)];
sdot = -0.5*(dot(omega,v));
vdot = 0.5*(s*omega + cross(omega,v));
Qdot = [sdot; vdot];

%% -------Angle of attack-------
% Angle between velocity vector of the CoP to the roll axis, given in the ground coord
% To Do : windmodel in env, Model gives errors
if(norm(X) < DTF.Rail)
W = [0, 0, 0]';
else
W = env.W;
end

Vcm = Xdot  + W;
Xstab = Xcp- DTF.Xcm;

omega_norm = normalize(omega); %normalized
Xperp =Xstab*sin(acos(dot(RA,omega_norm))); % Prependicular distance between omaga and
RA
```

```
Vomega = Xperp *cross(RA,omega);

V = Vcm + Vomega; % approxamating the velocity of the cop

Vmag = norm(V);
Vnorm = normalize(V);
alpha = acos(dot(Vnorm,RA));
DTF.alpha = alpha;
%% ------- P Forces = rate of change of Momentums-------

Fthrust = DTF.T*RA;

mg = DTF.Mass*env.g;
Fg = [0, 0, -mg]';

% Axial Forces
Famag = 0.5*env.rho*Vmag^2*DTF.A_ref*DTF.Cd; % To DO: make axial

Fa = -Famag*RA;

% Normal Forces
Fnmag = 0.5*env.rho*Vmag^2*DTF.A_ref*Cn;

RA_Vplane = cross(RA,Vnorm);
Fn = Fnmag*(cross(RA,RA_Vplane));

if (DTF.T< mg && X(3)< 0.1)
Ftot = [0, 0, 0]';
else
Ftot = Fthrust + Fg + Fa + Fn;
end
%% ------- L Torque-------
Trqn = Fnmag*Xstab*(RA_Vplane);

m=diag([1, 1, 0]);
invR = Rmatrix';
Trq_da = -Cda*Rmatrix*m*invR*omega;
%Tqm=(Cda1*omega)*omegaax2; rotational torque by motor
%       r_f = %TODO roll damping
%       Trmag = 0.5*env.rho*V^2*DTF.A_ref*DTF.Cld*r_f;
%       Tr = Trmag*RA;
if(norm(X) < DTF.Rail)
Trq = [0, 0, 0]';
DTF.departureState(1) = norm(Xdot); % Get rail departure vel from here wrt earth
DTF.departureState(2) = t;
else
Trq = Trqn+Trq_da;
end
```

```
        %% -------Update rocket state derivatives-------
        DTF.Xdot= Xdot;
        DTF.Qdot= Qdot;
        DTF.Pdot= Ftot;
        DTF.Ldot= Trq;

        state_dot =[Xdot; Qdot; Ftot;Trq];

        %% -------Burnout time-------
        if(DTF.propM_current<0.01 && DTF.t_Burnout == 0 )
        DTF.t_Burnout = t;
        end

        %% Log Data

        %logData(DTF.alpha, DTF.Cd, Cda, DTF.Xcm, DTF.Mass, Vmag, Xcp, zeta, Ssm, t);

        end

        function [value,isterminal,direction] = event_function(t,state)
        %% stops ode integration when the max height is reached
        if (t > 1 && state(10) <= 0) % Linear momentum in z direction is zero
        value = 0; % when value = 0, an event is triggered
        else
        value =1;
        end
        isterminal = 1; % terminate after the first event
        direction = 0;  % get all the zeros
        end
end
```

burn_data.m

```
function burn_data( DTF )
        % Burn data function is executed at the start of each iteration in the
        % solver. This updates the motor mass and inertias by assuming the
        % impulse generated is proportional to the mass consumed
        t= DTF.time;
        T=DTF.T; % Thrust
        tt=DTF.motordata(:,1);
        TT=DTF.motordata(:,2);
        ind= find(tt<t); % Finds index less then curretn time
        ind=ind(end);
        tt =[tt(1:ind); t]; % Burn uptill
```

```
        TT= [TT(1:ind); T];
        DTF.impulseGen =  trapz(tt,TT); % impulse generated upto that point.

        propM_used = DTF.propM_tot/DTF.Motor_impulse*DTF.impulseGen;
        DTF.propM_current = DTF.propM_tot  - propM_used; % Remaining prop mass
        if(DTF.deltat == 0)
        DTF.deltaMass =0 ;
        else
        % taking average for numerical stability
        DTF.deltaMass = (DTF.propM_current - DTF.propM_prev)/ DTF.deltat; %1.2902/1.8;%
        end
        DTF.propM_prev = DTF.propM_current;

        % New Inertias at time step
        % IMP:The mass of the prop should already be updated in DTF to get correct Xcm

        d = DTF.Xcm_prop - DTF.Xcm; % distance from Cm of propellent and???
        prop_OD = DTF.prop_OD;
        prop_h = DTF.prop_h;
        Mass_prop = DTF.propM_current;
        prop_density = DTF.prop_density;
        prop_ID = sqrt(prop_OD^2 -4*Mass_prop/(pi*prop_h*prop_density ));

        propIx = 0.5*Mass_prop*(prop_OD^2+prop_ID^2)/4;
        propIy = Mass_prop/12*(3*(prop_ID^2+prop_ID^2)/4 + prop_h^2) + DTF.propM_current*(d);
        DTF.Iprop = [propIx, 0, 0; 0, propIy, 0; 0, 0, propIy]; % propIy = propIz

end
```

Cd_mandell.m

```
function [Cd]=Cd_mandell(DTF)
        % Equations from Mandell
        global env
        rho =  env.rho; % density
        mu =  env.mu;  % dynamic viscosity
        C = env.C; %speed of sound dry air 15C sea level
        V = norm(DTF.Xdot,2);   %ms-1 Mag of  characteristic velocity at center of pressure location
        M = V/C;
        Re  = DTF.Re;

        % Approx laminar flow over rocket

        %% Rocket dimentions
        L = DTF.Length;
        L_cone = DTF.Cone_L;
```

```
L_cyl = L - L_cone;
D_cyl = DTF.D;
R_cyl = D_cyl/2;

%Fin Geometry
fin.n=DTF.fin_n;
fin.sweep = DTF.fin_sweep;
fin.h = DTF.fin_h;
fin.topchord = DTF.fin_top;
fin.basechord = DTF.fin_base;
fin.t = DTF.fin_t;
fin.a_ref = fin.n*fin.h*fin.t;
fin.area = (fin.topchord+fin.basechord)/2*fin.h;
fin.a_wet = fin.n*2*fin.area;
fin.c = (fin.topchord+fin.basechord)/2;
fin.X_b =  L - fin.basechord; % fin location

%A_ref
A_ref = pi*R_cyl^2;

F_ratio = L / D_cyl;


%%
R_ogive = (L_cone^2+ D_cyl^2/4)/D_cyl;
%A_wet

fun = @(x) 2*pi*(sqrt(R_ogive^2 - power((L_cone - x),2))+R_cyl-R_ogive);

A_wet =  integral(fun, 0, L_cone);

A_wet = A_wet + 2*pi*R_cyl*L_cyl;



%% Assigning to equations as discribes in mandell
l_n = L_cone;

d_n = D_cyl;
d_b = D_cyl;
d_f = D_cyl;
d_d = D_cyl;
l_c  = 0; % no btail
l_b = L_cyl;
Re_c = 5e5;
T_f = fin.t;
l_TR = L;
n=fin.n;
```

```
% mid chord sweep

temp.x1 = fin.h*tan(fin.sweep);

temp.x2 = temp.x1 + fin.topchord - fin.basechord;

fin.sweepc = atan2((fin.basechord/2 + (temp.x2-fin.topchord/2)),fin.h);

clear temp fun fun2
l_m = fin.h/acos(fin.sweepc); % length midchord

A_fe= (fin.topchord+fin.basechord)/2*fin.h;

A_fp = A_fe + 0.5*d_f*fin.basechord;

%% ------Viscous Friction------
% Viscous friction ROCKET FORBODY Cf
B = Re_c*(0.074/Re^(0.2) - 1.328/sqrt(Re));

if (Re < Re_c)
Cf =  1.328/sqrt(Re);
else
Cf=0.074/Re^(0.2)-B/Re;
end

%  Viscous friction ROCKET FINS Cf_f
Re_f  = env.rho*V*l_m/env.mu;  %Note the V is at the cop not the finneed to recalculate for
better results

B_f = Re_c*(0.074/Re_f^(0.2) - 1.328/sqrt(Re_f));

if (Re_f < Re_c)
Cf_f =  1.328/sqrt(Re_f);
else
Cf_f=0.074/Re_f^(0.2)-B_f/Re_f;
end

%% -------Drag at zero AoA-------
% Body drag, Box Eq41
Cd_fb = (1 + 60/(l_TR/d_b)^3+0.0025*l_b/d_b)*(2.7*l_n/d_b +4*l_b/d_b +
2*(1-d_d/d_b)*l_c/d_b)*Cf;
% Base drag, Box Eq42
Cd_b = 0.029*(d_d/d_b)^3/sqrt(Cd_fb);
% Fin drag, Box Eq44
Cd_f = 2*Cf_f *(1+2*T_f/l_m)*4*n*A_fp/(pi*d_f^2);
% Interference drag, Box Eq44
Cd_i = 2*Cf_f*(1+2*T_f/l_m)*4*n*(A_fp-A_fe)/(pi*d_f^2);
% Total drag coefficient at zero angle of attack
```

```
        Cd0 = Cd_fb + Cd_b + Cd_f + Cd_i;

        % Launch pin drag % estimated from Mandell
        A_pin = DTF.L_pinDia*DTF.L_pinH;
        Cd_pin = 2*0.8*A_pin/A_ref;
        Cd0 = Cd0 + Cd_pin;
        % compressibility correction
        %% -------Additional drag at AoA-------
        % Alpha
        alpha = DTF.alpha;
        % Coefficients delta dn eta from windtunnel experiments, See Box p13
    deltaktab=[4 6 8 10 12 14 16 18 20;0.78 0.86 0.92 0.94 0.96 0.97 0.975 0.98 0.982];
        etatab=[4 6 8 10 12 14 16 18 20 22 24;0.6 0.63 0.66 0.68 0.71 0.725 0.74 0.75 0.758 0.77 0.775];
        % error in paper
    etak=interp1(etatab(1,:),etatab(2,:),F_ratio,'linear','extrap');
    deltak=interp1(deltaktab(1,:),deltaktab(2,:),F_ratio,'linear','extrap');
        if etak>1;
        etak=1;
        end
        if deltak>1;
        deltak=1;
        end
        % Body drag at angle alpha
        Cd_b_alpha = 2*deltak*alpha^2 + 3.6*etak*(1.36*l_TR - 0.55*l_n)*alpha^3/(pi*d_b);
        % Fin body interference coefficients
        Rs = R_cyl/(R_cyl+fin.h);
        Kbf=0.8065*Rs^2+1.1553*Rs;
        Kfb=0.1935*Rs^2+0.8174*Rs+1;
        % Fin drag at angle alpha
        Cd_f_alpha = (1.2*A_fp*4/(pi*d_f^2) +3.12*(Kfb +Kbf-1)*A_fe*4/(pi*d_f^2))*alpha^2;

        %% -------Totol Drag Coefficient-------
        Cd = Cd0 + Cd_b_alpha + Cd_f_alpha;
        Cd = Cd/sqrt(1-M^2);
        CnXcp = DTF.CnXcp;
        Cn= CnXcp(1);
        Cd = (Cd*cos(alpha) -0.5*Cn*sin(2*alpha))/(1-sin(alpha)^2);

end
```

clean_log.m

```
function  log_cleaned = clean_log( t )
        global log
```

```
log_cleaned = [];
l1 =length(log(:,10));
l =length(t);
for i=1:l
for j=1:l1
flag =0;
if (t(i) == log(j,10))
log_cleaned = [log_cleaned; log(j,:)];
flag =1;
end
if (flag ==1)
break;
end
end
end

end
```

Cn_alphaXcp.m

```
function [Cn_alpha, Xcp, Cda, zeta, Ssm, Ssm_B, Ccm]=Cn_alphaXcp(DTF)
        % Takes rocket handle and environment  to calculate Cn, location of cop

        % using barrowman implemented in OpenRocket
        global env

        rho =  env.rho; % density
        mu =  env.mu;  % dynamic viscosity
        C = env.C; %speed of sound dry air 15C sea level
        V = norm(DTF.Xdot,2);   %ms-1 Mag of characteristic velocity at center of pressure location
        M = V/C;
        Re  = DTF.Re;

        % correction for compressible flow
        beta = sqrt( 1 - M^2); % M <1

        % Rocket dimentions
        L = DTF.Length;
        L_cone = DTF.Cone_L;
        L_cyl = L - L_cone;
        D_cyl = DTF.D;
        R_cyl = D_cyl/2;
```

```
%A_ref
A_ref = pi*R_cyl^2;

%F_ratio = L / D_cyl; <-clean


%%
R_ogive = (L_cone^2+ D_cyl^2/4)/D_cyl;
%A_wet <-clean
%
% fun = @(x) 2*pi*(sqrt(R_ogive^2 - power((L_cone - x),2))+R_cyl-R_ogive);
%
% A_wet =  integral(fun, 0, L_cone);
%
% A_wet = A_wet + 2*pi*R_cyl*L_cyl;

% Cone Vol

fun2 = @(x) pi*(power((sqrt(R_ogive^2 - power((L_cone - x),2))+R_cyl-R_ogive),2));
cone.vol =  integral(fun2, 0, L_cone);

%A_plan
temp_theta = atan(L_cone/(R_ogive-R_cyl));

cone.A_plan = 2*(R_ogive^2*temp_theta/2-((R_ogive-R_cyl)*L_cone)/2);

cyl.A_plan = D_cyl* L_cyl;
%Fin Geometry

fin.n=DTF.fin_n;
fin.sweep = DTF.fin_sweep;
fin.h = DTF.fin_h;
fin.topchord = DTF.fin_top;
fin.basechord = DTF.fin_base;
fin.t = DTF.fin_t;
fin.a_ref = fin.n*fin.h*fin.t;
fin.area = (fin.topchord+fin.basechord)/2*fin.h;
fin.a_wet = fin.n*2*fin.area;
fin.c = (fin.topchord+fin.basechord)/2;
fin.X_b =  L - fin.basechord; % fin location

% mid chord sweep

x1 = fin.h*tan(fin.sweep);

x2 = x1 + fin.topchord - fin.basechord;
```

```
fin.sweepc = atan2((fin.basechord/2 + (x2-fin.topchord/2)),fin.h);

%clear temp fun fun2

%% Center of presure
alpha = DTF.alpha;

% Cone

if (alpha ==0)
alpha = 0.00001;
end
K=1.1;
cone.Cn_correction = K * cone.A_plan/A_ref*sin(alpha)^2;
cone.Cn_alpha = 2* (A_ref/A_ref)*sin(alpha)/ alpha + cone.Cn_correction/alpha;

% cylinder
% Cn_alpha

cyl.Cn_correction = K * cyl.A_plan/A_ref*sin(alpha)^2;

cyl.Cn_alpha =  cyl.Cn_correction/alpha;
% Fins


fin.Cn1_alpha = ((2*pi*fin.h^2)/ A_ref)/...
(1 + sqrt(1 + (beta*fin.h^2/(fin.area*cos(fin.sweepc)))^2));

% N fins corrected for body interference n >= 3
fin.Cn_alpha = (1 + (R_cyl)/(R_cyl+ fin.h))*(fin.Cn1_alpha * fin.n/2*1);

%% CoP location

% Cone

cone.Xcp = (L_cone*(A_ref)-cone.vol)/A_ref;

% cyl

cyl.Xcp = L_cone + L_cyl/2;

%  Fins at 25% mac
Xt = fin.h/tan(fin.sweep);
fin.Xcp = fin.X_b + (Xt/3*(fin.basechord +  2*fin.topchord) +
1/6*(fin.basechord+fin.topchord)^2)/(fin.basechord+fin.topchord);
```

```
        Xcp = (fin.Cn_alpha*fin.Xcp + cone.Cn_alpha*cone.Xcp +
cyl.Xcp*cyl.Cn_alpha)/(fin.Cn_alpha+cone.Cn_alpha+cyl.Cn_alpha);
        %% Xcp without Body lift for documentation
        [Xcp_Barrowman, Xcp_Planform, Ssm_Barrowman] = Xcp_Barrowman_f(DTF);
        Xcp_B = Xcp_Barrowman;
        %% Roll damping
        % % omega = deg2rad(140);
        % % Cn_alpha0 = 2*pi/beta; % from potential flow over a thin foil.
        % %
        % % temp = (fin.basechord+fin.topchord)*R_cyl^2*fin.h/2 +
(fin.basechord+2*fin.topchord)*R_cyl*fin.h^2/3  + (fin.basechord+3*fin.topchord)*fin.h^3/12;
        % % Cld = fin.n*Cn_alpha0/(A_ref*v0*D_cyl) * omega * temp;

        %% Pitch Damping  [Nm/s]
        % Jet damping
        lcn = DTF.Length - DTF.Xcm;
        lcc = DTF.Xcm_prop -DTF.Xcm;
        Cda_jet = DTF.deltaMass *(lcn^2 - lcc^2);

        % Aerodynamic damping
        Cda_l = 0.5*rho* V * A_ref *( fin.Cn_alpha*(fin.Xcp-DTF.Xcm)^2+...
        cone.Cn_alpha*(cone.Xcp - DTF.Xcm)^2 + cyl.Cn_alpha*(cyl.Xcp - DTF.Xcm)^2);

        % Damping Moment Coefficient
        Cda  =  Cda_l+ Cda_jet;

        %% Cn_alpha
        Cn_alpha = fin.Cn_alpha + cyl.Cn_alpha + cone.Cn_alpha;

        %% Static Stability Margin
        Ssm = (Xcp-DTF.Xcm)/DTF.D;
        Ssm_B = (Xcp_B-DTF.Xcm)/DTF.D;
        %% Damping Ratio
        % Corrective Moment Coefficient
        Ccm = (rho/2 * V^2 * DTF.A_ref * Cn_alpha) * (Xcp-DTF.Xcm);
        Ibody = DTF.Ibody;
        zeta = Cda/(2*sqrt(Ccm*Ibody(2,2)));

    end
```

cop.m

```
    clc; clear;

    w = length(0:0.5:5);
```

```
alphatab = [0:0.5:5];

v0tab= [0:10:250];
h= length(v0tab);
Cn = zeros(h,w);
Xcppos= zeros(w,1);
%% CP  calculator from openrocket
%for i=1:h
L = 2.646; %characteristic length
rho =  1.225; % density
mu =  1.7894e-5;  % dynamic viscosity
C = 340.3; %speed of sound dry air 15C sea level
v0 = v0tab(1);   %ms-1 characteristic velocity
M = v0/C;
Re  = rho*v0*L/mu;

% correction for
beta = sqrt( 1 - M^2); % M <1

% Rocket dimentions
L_cone = 0.61;
L_cyl = L - L_cone;
D_cyl = 0.158;
R_cyl = D_cyl/2;

%A_ref
A_ref = pi*R_cyl^2;

F_ratio = L / D_cyl;


%%
R_ogive = (L_cone^2+ D_cyl^2/4)/D_cyl;
%A_wet

fun = @(x) 2*pi*(sqrt(R_ogive^2 - power((L_cone - x),2))+R_cyl-R_ogive);

A_wet =  integral(fun, 0, L_cone);

A_wet = A_wet + 2*pi*R_cyl*L_cyl;

% Cone Vol

fun2 = @(x) pi*(power((sqrt(R_ogive^2 - power((L_cone - x),2))+R_cyl-R_ogive),2));
cone.vol =  integral(fun2, 0, L_cone);

%A_plan
temp_theta = atan(L_cone/(R_ogive-R_cyl));
```

```
cone.A_plan = 2*(R_ogive^2*temp_theta/2-((R_ogive-R_cyl)*L_cone)/2);

cyl.A_plan = D_cyl* L_cyl;
%Fin Geometry

fin.n=3;
fin.sweep = deg2rad(45);
fin.h = 11.5e-2;
fin.topchord = 11.5e-2;
fin.basechord = 21e-2;
fin.t = 1e-2;
fin.a_ref = fin.n*fin.h*fin.t;
fin.area = (fin.topchord+fin.basechord)/2*fin.h;
fin.a_wet = fin.n*2*fin.area;
fin.c = (fin.topchord+fin.basechord)/2;
fin.X_b =  L - fin.basechord; % fin location

% mid chord sweep

x1 = fin.h*tan(fin.sweep);

x2 = x1 + fin.topchord - fin.basechord;

fin.sweepc = atan2((fin.basechord/2 + (x2-fin.topchord/2)),fin.h);

%clear temp fun fun2

%% Center of presure
%for j=1:w
alpha =deg2rad(alphatab(1));

%% Cone
% Cn_alpha
if (alpha ==0)
        alpha = 0.00001,
end
K=1.1;
cone.Cn_correction = K * cone.A_plan/A_ref*sin(alpha)^2;
cone.Cn_alpha = 2* (A_ref/A_ref)*sin(alpha)/ alpha + cone.Cn_correction/alpha;

%% cylinder
% Cn_alpha

cyl.Cn_correction = K * cyl.A_plan/A_ref*sin(alpha)^2;

cyl.Cn_alpha =  cyl.Cn_correction/alpha;
%% Fins
```

```
fin.Cn1_alpha = ((2*pi*fin.h^2)/ A_ref)/(1 + sqrt(1 + (beta*fin.h^2/(fin.area*cos(fin.sweepc)))^2));

% N fins corrected for body interference n >= 3
fin.Cn_alpha = (1 + (R_cyl)/(R_cyl+ fin.h))*(fin.Cn1_alpha * fin.n/2*1);

%% CoP location
% Cone

cone.Xcp = (L_cone*(A_ref)-cone.vol)/A_ref

% cyl

cyl.Xcp = L_cone + L_cyl/2

%  Fins
% at 25% mac
Xt = fin.h/tan(fin.sweep);
fin.Xcp = fin.X_b + (Xt/3*(fin.basechord +  2*fin.topchord) +
1/6*(fin.basechord+fin.topchord)^2)/(fin.basechord+fin.topchord)


Xcp = (fin.Cn_alpha*fin.Xcp + cone.Cn_alpha*cone.Xcp +
cyl.Xcp*cyl.Cn_alpha)/(fin.Cn_alpha+cone.Cn_alpha+cyl.Cn_alpha)
%% Roll damping
% omega = deg2rad(140);
% Cn_alpha0 = 2*pi/beta; % from potential flow over a thin foil.
%
% temp = (fin.basechord+fin.topchord)*R_cyl^2*fin.h/2 +
(fin.basechord+2*fin.topchord)*R_cyl*fin.h^2/3  + (fin.basechord+3*fin.topchord)*fin.h^3/12;
% Cld = fin.n*Cn_alpha0/(A_ref*v0*D_cyl) * omega * temp;


%% Cn_alpha
Cn_alpha = fin.Cn_alpha + cyl.Cn_alpha + cone.Cn_alpha;

F_n = 0.5* v0^2* rho* A_ref * Cn_alpha*alpha;

% Cn(i,j) = Cn_alpha*alpha;
% Xcppos(j) = Xcp;
% %end
% %end
%
% surf(alphatab',v0tab',Cn)
% xlabel('alpha (deg)')
% ylabel('V0 (m/s)')
% zlabel('Cn')
```

```
%
% %%
%
% plot(alphatab,Xcppos)
% xlabel('alpha (deg)');
% ylabel('Xcp (m)');
```

Drag_datcom1.m

```
clc; clear all;

%%  Drag using Mandel


w = length(0:0.5:5);
alphatab = [0:0.5:5];

v0tab= [10:10:250];
h= length(v0tab);
%Cd = zeros(h,w);
%for i=1:h

L = 2.646; %characteristic length
rho =  1.225; % density
mu =  1.7894e-5;  % dynamic viscosity
C = 340.3; %speed of sound dry air 15C sea level
v0 = 50;%v0tab(1); %150;   %ms-1 characteristic velocity
M = v0/C;
Re  = rho*v0*L/mu;

% Approx laminar flow over rocket

% Rocket dimentions
L_cone = 0.61;
L_cyl = L - L_cone;
D_cyl = 0.158;
R_cyl = D_cyl/2;

%Re  = rho*v0*D_cyl/mu;
```

```
%%
R_ogive = (L_cone^2+ D_cyl^2/4)/D_cyl;
%A_ref

fun = @(x) 2*pi*(sqrt(R_ogive^2 - power((L_cone - x),2))+R_cyl-R_ogive);

A_wet =  integral(fun, 0, L_cone);

A_wet = A_wet + 2*pi*R_cyl*L_cyl;

A_ref = pi*R_cyl^2;

F_ratio = L / D_cyl;

%Fin Geometry
fin.n=3;
fin.sweep = deg2rad(45);
fin.h = 11.5e-2;
fin.topchord = 7e-2;
fin.basechord = 21e-2;
fin.t = 1e-2;
fin.a_ref = fin.n*fin.h*fin.t;
fin.area = (fin.topchord+fin.basechord)/2*fin.h;
fin.a_wet = fin.n*2*fin.area;
fin.c = (fin.topchord+fin.basechord)/2;
fin.X_b =  L - fin.basechord; % fin location

l_n = L_cone;

d_n = D_cyl;
d_b = D_cyl;
d_f = D_cyl;
d_d = D_cyl;
l_c  = 0; % no btail
l_b = L_cyl;
Re_c = 5e5;
T_f = fin.t;
l_TR = L;
n=fin.n;
% mid chord sweep

temp.x1 = fin.h*tan(fin.sweep);

temp.x2 = temp.x1 + fin.topchord - fin.basechord;

fin.sweepc = atan2((fin.basechord/2 + (temp.x2-fin.topchord/2)),fin.h);

clear temp fun fun2
```

```
l_m = fin.h/acos(fin.sweepc);

A_fe= (fin.topchord+fin.basechord)/2*fin.h;

A_fp = A_fe + 0.5*d_f*fin.basechord;

%%
B = Re_c*(0.074/Re^(0.2) - 1.328/sqrt(Re));

if (Re < Re_c)
        Cf =  1.328/sqrt(Re);
else
        Cf=0.074/Re^(0.2)-B/Re;
end

%
Re_f  = rho*v0*l_m/mu;

B_f = Re_c*(0.074/Re_f^(0.2) - 1.328/sqrt(Re_f));

if (Re_f < Re_c)
        Cf_f =  1.328/sqrt(Re_f);
else
        Cf_f=0.074/Re_f^(0.2)-B/Re_f;
end
%

Cd_fb = (1 + 60/(l_TR/d_b)^3+0.0025*l_b/d_b)*(2.7*l_n/d_b +4*l_b/d_b + 2*(1-d_d/d_b)*l_c/d_b)*Cf;

Cd_b = 0.029*(d_d/d_b)^3/sqrt(Cd_fb);

Cd_f = 2*Cf_f *(1+2*T_f/l_m)*4*n*A_fp/(pi*d_f^2);

Cd_i = 2*Cf_f*(1+2*T_f/l_m)*4*n*(A_fp-A_fe)/(pi*d_f^2);

Cd0 = Cd_fb + Cd_b + Cd_f + Cd_i;

% compressibility correction
%for j=1:w
% Alpha
alpha =deg2rad(alphatab(1));
deltaktab=[4 6 8 10 12 14 16 18 20;0.78 0.86 0.92 0.94 0.96 0.97 0.975 0.98 0.982];
etatab=[4 6 8 10 12 14 16 18 20 22 24;0.6 0.63 0.66 0.68 0.71 0.725 0.74 0.75 0.758 0.77 0.775];
% error in paper
etak=interp1(etatab(1,:),etatab(2,:),F_ratio,'linear','extrap');
deltak=interp1(deltaktab(1,:),deltaktab(2,:),F_ratio,'linear','extrap');
if etak>1;
        etak=1;
```

```
end
if deltak>1;
        deltak=1;
end
Cd_b_alpha = 2*deltak*alpha^2 + 3.6*etak*(1.36*l_TR - 0.55*l_n)*alpha^3/(pi*d_b);

Rs = R_cyl/(R_cyl+fin.h);
Kbf=0.8065*Rs^2+1.1553*Rs;
Kfb=0.1935*Rs^2+0.8174*Rs+1;

Cd_f_alpha = (1.2*A_fp*4/(pi*d_f^2) +3.12*(Kfb +Kbf-1)*A_fe*4/(pi*d_f^2))*alpha^2;

Cd = Cd0 + Cd_b_alpha + Cd_f_alpha


% Cdsurf(i,j) = Cd/sqrt(1-M^2)
% end
% end
%
% surf(alphatab',v0tab',Cdsurf)
% xlabel('alpha (deg)')
% ylabel('V0 (m/s)')
% zlabel('Cd')
```

environement.m

```
%% Environment

classdef environement<handle
  properties
        gamma = 1.4;
        R = 287;
        Ref_Temp = 291.15;      % Ref temp for mu
        Ref_Dyn_viscosity = 1.827e-5;
        Sutherlands_c = 120;
        Earth_M = 5.97237e24;
        Earth_R = 6378000;      % Radius at equator (m)
        G = 6.67408e-11;        % Gravatational constant
        P_sea = 1.01325e5;      % Pressure at sealevel
        rho_sea = 1.225;        % Air density at sealevel
        Temp_grad = 0.0065;     % Temperature gradient troposphere(K/m)
        YA0 = [1, 0, 0];
        PA0 = [0, 1, 0];
```

```
            RA0 = [0, 0, 1];
            h_g = 0;              % Ground height above sealevel (m)
            Pressure_g = 1.01325e5; % Ground Pressure (Default:sealevel)
            Temp_g = 288.16;            % Ground temp (Default:sealevel (15C)) (K)
            rho_g = 1.225;
            rkt

        end
        methods
            function obj = environement(val1, val2, val3,val4)
            if nargin > 0
            if (isnumeric(val1) && isnumeric(val2) && isnumeric(val3))
            obj.h_g = val1;
            obj.Temp_g = val2+273.15;
            obj.Pressure_g = val3;
            %obj.rho_g =  build to update rho at h_g
            else
            error('Enter numeric elevation(m) Temperature(C)and Pressure(Pa)')
            end
            obj.rkt =val4;
            end
            end

            function g = g(obj)% Calculates g at current altitude
            %global DTF
            h = obj.rkt.X(3);
            g = obj.G*obj.Earth_M/((obj.Earth_R+ h) + obj.h_g)^2;
            end

            function Temp = Temp(obj) % Calculates temperature at current altitude
            %global DTF
            h = obj.rkt.X(3);
            Temp = -obj.Temp_grad*(h)+obj.Temp_g();
            end
            function mu = mu(obj) % Calculates mu at current altitude
            mu = obj.Ref_Dyn_viscosity*(obj.Sutherlands_c + obj.Ref_Temp)/...
            (obj.Sutherlands_c + obj.Temp)*...
            (obj.Temp/obj.Ref_Temp)^(3/2);
            end
            function Pressure = Pressure(obj) % Calculates air density at current altitude
            n = (obj.g/(obj.Temp_grad*obj.R));
            Pressure = obj.Pressure_g*(obj.Temp/obj.Temp_g)^n;  %  alternate eq
https://www.mide.com/pages/air-pressure-at-altitude-calculator

            end
            function rho = rho(obj) % Calculates air density at current altitude
            n = (obj.g/(obj.Temp_grad*obj.R))-1;
            rho = obj.rho_g*(obj.Temp/obj.Temp_g)^n;
```

```
        end
        function C = C(obj) % Calculates speed of sound at current altitude
        C = sqrt(obj.gamma*obj.R*obj.Temp);
        end
        function W = W(obj) % Wind Vecor
        global var
        W = [0, 0, 0]'; % Access wind model from here
        end

  end
end
```

ectract_data.m

```
function a = extract_data ( state,t)
% Function run the states through the model again to extract the needed
% internal parameters. This has to be done becasue ode45 does not allow for
% the extraction of internal parameters
        global env
        DTF_data =rocket(init_rocket());
        motor_init( DTF_data );
        env = environement(1400, 25, 86000, DTF_data );
        %%
        DTF_data.X = state(1,1:3)';
        DTF_data.Q = state(1,4:7)';
        DTF_data.P = state(1,8:10)';
        DTF_data.L = state(1,11:13)';
        DTF_data.time= t(1);
        %%
        ilast = length(t);
        a = 0; %zeros(ilast,3);
        %%
        for i = 2:ilast

        DTF_data.time= t(i);
        DTF_data.deltat = t(i)-t(i-1);
        DTF_data.X = state(i,1:3)';
        DTF_data.Q = state(i,4:7)';
        DTF_data.P = state(i,8:10)';
        DTF_data.L = state(i,11:13)';
        burn_data(DTF_data);
        DTF_data.Xdot = (state(i,1:3)' - state(i-1,1:3)')/DTF_data.deltat;
        DTF_data.Qdot = (state(i,4:7)' - state(i-1,4:7)')/DTF_data.deltat ;
        DTF_data.Pdot = (state(i,8:10)' - state(i-1,8:10)')/DTF_data.deltat;
        DTF_data.Ldot = (state(i,11:13)' -state(i-1,11:13)')/DTF_data.deltat;
```

```
                Rmatrix= quat2rotm(DTF_data.Q');
                RA = Rmatrix*env.RA0';
                X = DTF_data.X;
                Xdot = DTF_data.Xdot;
                L = DTF_data.L;
                if(norm(X) < DTF_data.Rail)
                W = [0, 0, 0]';
                else
                W = env.W;
                end
                CnXcp = DTF_data.CnXcp;
                Cn= CnXcp(1);
                Xcp= CnXcp(2);
                Cda = CnXcp(3); % Damping moment coefficient
                zeta = CnXcp(4); % Damping ratio
                Ssm = CnXcp(5); % Static stability margin
                Ssm_B = CnXcp(6); % Static stability margin without body lift correction
                Ccm = CnXcp(7); % Corrective moment coeff
                invIbody = DTF_data.Ibody\eye(3); %inv(DTF.Ibody); inverting matrix
                omega = Rmatrix*invIbody*Rmatrix'*L;
                Vcm = Xdot  + W;
                Xstab = Xcp- DTF_data.Xcm;

                omega_norm = normalize(omega); %normalized
                Xprep =Xstab*sin(acos(dot(RA,omega_norm))); % Prependicular distance between omaga and
RA

                Vomega = Xprep *cross(RA,omega);

                V = Vcm + Vomega; % approxamating the velocity of the cop

                Vmag = norm(V);
                Vnorm = normalize(V);
                alpha = acos(dot(Vnorm,RA));
                DTF_data.alpha = alpha;

                %% Log Data

                logData(DTF_data.alpha, DTF_data.Cd, Cda, DTF_data.Xcm, DTF_data.Mass, Vmag, Xcp, zeta,
Ssm, Ssm_B, Ccm, t(i));

                %logData(DTF_data.X(3),DTF_data.Cd,t(i)); % Eg DTF.Cd for drag norm(Xdot)/env.C
        %       log(i,1) = norm(DTF_data.Xdot);
        %       log(i,2) = DTF_data.Re;
        %       log(i,3) = t(i);
                end

        end
```

init_rocket.m

```
plotData(log, DTF);
 function prop = init_rocket()

        prop = readtable('rocket_prop_DTF2.txt','Format','%s%f');
%       motorname=char(table2array(prop(end,1)));
%       motordata = rocketmotor(motorname);

end
```

logData.m

```
function logData(alpha, Cd, Cda, Xcm, Mass, Vmag, Xcp, zeta, Ssm, Ssm_B, Ccm, t)

        global log

        %log = [log;[Xdot, Xstab, rad2deg(alpha),t]];
        log = vertcat(log,[alpha, Cd, Cda, Xcm, Mass, Vmag, Xcp, zeta, Ssm, Ssm_B, Ccm, t]);
%       if(t==0)
%       file1 = fopen('log.dat','w');
%       else
%       file1 = fopen('log.dat','a');
%       end
%       input = ('%f,%f,%f,%f,%f,%f \n');
%       fprintf(file1,input,[Xdot, Xstab, rad2deg(alpha),t]);
%       fclose(file1);

        % if(t==0)
        %       save('log',X, Xdot, accel, alpha,t);
        % else
        %       save('log',X, Xdot, accel, alpha,t,'-append');
        % end
        %
        %
        % end
end
```

motor_init.m

```
function motor_init( DTF )
        %% data from thrustcurve.org
```

```
% Reads the .eng thrust file of the motors and calculates the reletive
% the reletive properties of the motor.

% Extracting thrust curve
R1=4;
C1=0;
motorname = char(DTF.motorname);
motordata = dlmread(motorname,'',R1,C1);
DTF.motordata = [0, 0; motordata]; % at 0 0 at start of date

% Very strange way of reading mass of motor amd propellent from the
% file
fid = fopen(motorname);
tline = fgets(fid);
tline = fgets(fid);
tline = fgets(fid);
tline = fgets(fid);
C = strsplit(tline);
Motor_diameter = str2double((C(2)))*1e-3; %[m]
Motor_lenght = str2double((C(3)))*1e-3; %[m]
Mass_prop = str2double((C(5)));
Mass_motor = str2double((C(6)));
DTF.propM_tot = Mass_prop;
DTF.Mass_motor = Mass_motor - Mass_prop;
fclose(fid);

%Initally current and total prop mass are equal
DTF.propM_current = Mass_prop;
% Calculated impulse from motor data using spline
DTF.Motor_impulse = trapz(DTF.motordata(:,1),DTF.motordata(:,2));

%Initializing propellent dimentions
prop_density =   1.4905e+03;%1.5079e+03; %  calciualed by volume and mass of prop from
drawing   <-
prop_OD =  Motor_diameter  -6.5000e-3;%-12.4000e-3; %[m] %Offsets estimates from drawing
<-
prop_h = Motor_lenght - 69e-3;%-111.10003e-3;   %[m] <-
prop_ID = sqrt(prop_OD^2 -4*Mass_prop/(pi*prop_h*prop_density ));
DTF.Xcm_prop = DTF.Length - prop_h/2; % Has to be confirmed with drawing every time the
motor changes

DTF.prop_density= prop_density;
DTF.prop_OD = prop_OD;
DTF.prop_h = prop_h;

%Initializing propellent inertias w.r.t. cm
d = DTF.Xcm_prop - DTF.Xcm; % Note: The mass of the prop should already be updated in DTF
to get correct Xcm
```

```
        propIx = 0.5*Mass_prop*(prop_OD^2+prop_ID^2)/4;
        propIy = Mass_prop/12*(3*(prop_ID^2+prop_ID^2)/4 + prop_h^2) + Mass_prop*(d);
        DTF.Iprop = [propIx, 0, 0; 0, propIy, 0; 0, 0, propIy]; % propIy = propIz



end
```

MyClass.mm

```
classdef MyClass
  properties
        Prop1
  end
%       methods
%       function obj = set.Prop1(obj,value)
%       if (value > 0)
%       obj.Prop1 = value;
%       else
%       error('Property value must be positive')
%       end
%       end
%       end
end
```

normalize.m

```
function [ normalizedVector ] = normalize( vector )
 % normalise vector
 if (norm(vector) == 0 )
        normalizedVector = vector;
 else
        normalizedVector = vector/norm(vector);
 end

end
```

plotData.m

```
function plotData( clog, DTF )


        tend = 15;
        t_Burnout = DTF.t_Burnout;
        t_RailExit = DTF.departureState(2);
```

```
% vel_RailExit = DTF.departureState(1);

% Calculate Xcp Barrowman for documentation
[Xcp_Barrowman, Xcp_Planform, Ssm_Barrowman] = Xcp_Barrowman_f(DTF);
Xcp_B = ones(length(clog),1) * Xcp_Barrowman;
Xcp_P= ones(length(clog),1) * Xcp_Planform;

%% --- Figure Flight Information ---

figure('Name','Flight information','Position', [800 0 400 1000]);
subplot(4,1,1)
plot(clog(:,12),clog(:,6))
xlabel('Time')
ylabel('Xdot [m/s]')
title('Velocity')
axis([0 tend 0 300])
line([t_RailExit t_RailExit],[0 300],'Color',[0 1 0])
line([t_Burnout t_Burnout],[0 300],'Color',[1 0 0])

subplot(4,1,2)
plot(clog(:,12),clog(:,5))
xlabel('Time')
ylabel('Mass [kg]')
title('Mass')
axis([0 tend 20 30])
line([t_RailExit t_RailExit],[0 300],'Color',[0 1 0])
line([t_Burnout t_Burnout],[0 300],'Color',[1 0 0])

subplot(4,1,3)
plot(clog(:,12),clog(:,2))
xlabel('Time(s)')
ylabel('Cd []')
title('Drag Coefficient')
axis([0 tend 0 1])
line([t_RailExit t_RailExit],[0 300],'Color',[0 1 0])
line([t_Burnout t_Burnout],[0 300],'Color',[1 0 0])

subplot(4,1,4)
plot(clog(:,12),clog(:,1))
xlabel('Time')
ylabel('alpha [rad]')
title('Angle of Attack')
axis([0 tend 0 0.4])
line([t_RailExit t_RailExit],[0 300],'Color',[0 1 0])
line([t_Burnout t_Burnout],[0 300],'Color',[1 0 0])

%% --- Figures Stability Analysis ---
% COP
```

```
                figure('Name','Stability Analysis', 'Position', [0 0 800 1000])
                subplot(3,2,1)
                hold on
                xcp1 = plot(clog(:,12),clog(:,7));
                xcp2 = plot(clog(:,12),Xcp_B);
                xcp3 = plot(clog(:,12),Xcp_P);
                hold off
                xlabel('Time')
                ylabel('Xcp [m]')
                title('Center of Pressure')
                axis([0 tend 1 2])
                line([t_RailExit t_RailExit],[0 300],'Color',[0 1 0])
                line([t_Burnout t_Burnout],[0 300],'Color',[1 0 0])
                legend([xcp1 xcp2
xcp3],'Barrowman+LiftCorrection','Barrowman','Planform','Location','southwest','Orientation','horizontal')
                % COG
                subplot(3,2,2)
                plot(clog(:,12),clog(:,4))
                xlabel('Time')
                ylabel('Xcm [m]')
                title('Center of Mass')
                axis([0 tend 1 1.7])
                line([t_RailExit t_RailExit],[0 300],'Color',[0 1 0])
                line([t_Burnout t_Burnout],[0 300],'Color',[1 0 0])
                % STABILITY MARGIN
                subplot(3,2,3)
                hold on
                ssm1 = plot(clog(:,12),clog(:,9));
                ssm2 = plot(clog(:,12),clog(:,10));
                hold off
                xlabel('Time')
                ylabel('Margin []')
                title('Static Stability Margin')
                axis([0 tend 0 3])
                line([t_RailExit t_RailExit],[0 300],'Color',[0 1 0])
                line([t_Burnout t_Burnout],[0 300],'Color',[1 0 0])
                legend([ssm1 ssm2],
'Barrowman+LiftCorrection','Barrowman','Location','southwest','Orientation','horizontal')
                % DAMPING RATIO
                subplot(3,2,4)
                plot(clog(:,12),clog(:,8))
                xlabel('Time')
                ylabel('Zeta []')
                title('Damping Ratio')
                axis([0 tend 0.0 0.2])
                line([t_RailExit t_RailExit],[0 300],'Color',[0 1 0])
                line([t_Burnout t_Burnout],[0 300],'Color',[1 0 0])
                % C_CORRECTIVE
```

```
        subplot(3,2,5)
        plot(clog(:,12),clog(:,11))
        xlabel('Time')
        ylabel('Corrective MomentCoeff [N]')
        title('Corrective Moment Coefficient')
        axis([0 tend 0.0 3500])
        line([t_RailExit t_RailExit],[0 5000],'Color',[0 1 0])
        line([t_Burnout t_Burnout],[0 5000],'Color',[1 0 0])
        % C_DAMPING
        subplot(3,2,6)
        plot(clog(:,12),clog(:,3))
        xlabel('Time')
        ylabel('Damping MomentCoeff [N]')
        title('Damping Moment Coefficient')
        axis([0 tend 0.0 100])
        line([t_RailExit t_RailExit],[0 5000],'Color',[0 1 0])
        line([t_Burnout t_Burnout],[0 5000],'Color',[1 0 0])



end
```

rocket.m

```
%% Rocket
% Defines all the properties of the rocket. The dynamics are continuously
% updated, any varibales need for functions should come from this class
classdef rocket <handle
  properties
        % Rocket Characteristics, These do not change during flight
        Length
        D
        Cone_L
        fin_n
        fin_h
        fin_base
        fin_top
        fin_sweep
        fin_t
        Mass_dry        %Rocket and motor housing no prop
        Ibody_dry       %Rocket and motor housing no prop
        Xcm_dry         %Rocket and motor housing no prop
```

```
Rail = 3.5;
        L_pinDia
        L_pinH
        A_ref %Clean this up?

        %Motor Characteristics: Updated in each iteration of the accent_calc
        motorname
        motordata
        Mass_motor
        Motor_impulse               % Total Impulse of motor
        propM_tot           % Total mass of prop
        Iprop               % Inertia matrix of prop wrt Cg
        Xcm_prop            % Center of mass of the prop
        prop_OD
        prop_ID
        prop_h
        prop_density

        deltat              % Size of time step calcualted in accent_calc
        deltaMass = 0;              % initally zero
        impulseGen                  % Impulse generated upto that point
        propM_current               % Remaning prop mass
        propM_prev = 0;         % Mass for previous time step to calcualte deltaMass

        departureState              %Saves the state vector at departure
        t_Burnout = 0;

        % Current State Vector with Initial values, Updated in accent_calc
        time = 0;               %time
        X = [0; 0; 0];          % Position x, y, z
        Q = [1; 0; 0; 0]; % Angle in quarternions
        P = [0; 0; 0];          % Linear Momentum
        L = [0; 0; 0];          % Angular momentum
        Xdot = [0; 0; 0];   % Velocity xdot, ydot, zdot
        Qdot = [0; 0; 0; 0];% Angular rates
        Pdot = [0; 0; 0];   % Linear Momentum rates = applied force
        Ldot = [0; 0; 0];   % Angular Momentum rates= applied torques
        alpha = 0;


end
methods
        %Constructor takes data from rocket property file
        function obj = rocket(val) %Val is the table of properties val2 motor data
        if nargin > 0
        prop=table2array(val(1:end-1,2));
        motorname=table2array(val(end,1));
        %if isnumeric(val)
```

```
obj.Length = prop(1);
obj.Cone_L = prop(2);
obj.D = prop(3);
obj.fin_n = prop(4);
obj.fin_h = prop(5);
obj.fin_base = prop(6);
obj.fin_top = prop(7);
obj.fin_sweep = deg2rad(prop(8));
obj.fin_t = prop(9);
obj.Mass_dry = prop(10);
obj.Ibody_dry = [prop(11), 0 ,0; 0 , prop(12), 0; 0, 0, prop(13)];
obj.Xcm_dry = prop(14);
obj.L_pinDia = prop(15);
obj.L_pinH = prop(16);
% Launch Rail Heading
Ra =  -0.1047; %deg2rad(90-prop(17));
Rax = [0.1684;     0.985;                  0];
obj.Q=[cos(Ra/2) sin(Ra/2)*Rax(1) sin(Ra/2)*Rax(2) sin(Ra/2)*Rax(3)]';

%motor
obj.motorname=motorname;
obj.A_ref = (pi*obj.D^2/4);
%else
%error('Value must be numeric')
%end
end
end
%Fuctions to calcualte various properties

function Cd = Cd(obj) % Drag in axial direction
Cd = Cd_mandell(obj);
if (isinf(Cd) || Cd > 10)
Cd =10;
end
end

function CnXcp = CnXcp(obj) % Normal force and Cop location
[Cn_alpha, Xcp, Cda, zeta, Ssm]=Cn_alphaXcp(obj);
CnXcp = [Cn_alpha*obj.alpha, Xcp, Cda, zeta, Ssm];
[Cn_alpha, Xcp, Cda, zeta, Ssm, Ssm_B, Ccm]=Cn_alphaXcp(obj);
CnXcp = [Cn_alpha*obj.alpha, Xcp, Cda, zeta, Ssm, Ssm_B, Ccm];
end

function T = T(obj) % Thrust curve
M = obj.motordata;
T = interp1(M(:,1),M(:,2),obj.time,'spline');
if ( obj.time > M(end,1))
T = 0;
```

```
        end
        end

        function Re = Re(obj) % Re of rocket
        global env
        Re = env.rho*norm(obj.Xdot)*obj.Length/env.mu;

        end

        function Mass = Mass(obj) % Current mass of rocket
        M = obj.motordata;
        if ( obj.time > M(end,1)); % To assure it goes to zero incase of integartion error
        obj.propM_current =0;
            end
        Mass= obj.Mass_dry + obj.propM_current;
        end

        function Xcm = Xcm(obj) % Current Center of mass of rocket
        M = obj.motordata;
        if ( obj.time > M(end,1)); % To assure it goes to zero incase of integartion error
        Xcm = obj.Xcm_dry;
        end
        Xcm= obj.Xcm_dry*obj.Mass_dry + obj.Xcm_prop*obj.propM_current;
        Xcm = Xcm/obj.Mass;
        end

        function Ibody = Ibody(obj) % Current Inertia of the rocket
        M = obj.motordata;
        if ( obj.time > M(end,1)); % To ensure Iprop goes to zero incase of integartion error
        Ibody = obj.Ibody_dry;
        end
        Ibody = obj.Ibody_dry + obj.Iprop;
        end
  end
end
```

splinemotordata.m

```
        t= DTF.motordata(:,1);
        F=DTF.motordata(:,2);
        xev = 3*linspace(0,1,300);
        spl = spline(t,F);
        plot(t,ppval(spl,t));
        % integrate the spline
        spl.coefs = spl.coefs*[diag(1./[4 3 2 1]'),zeros(4,1)];
        spl.order = 5;
```

```
%%
dx = diff(spl.breaks)';
C = spl.coefs(:,1);
for i = 1:4
C = C.*dx + spl.coefs(:,i+1);
end
spl.coefs(2:end,5) = cumsum(C(1:(end-1)));
xev = t;
splint = ppval(spl,xev);
plot(xev,splint)
```

Xcp_Barrowman_f.m

```
function [Xcp_Barrowman, Xcp_Planform, Ssm_Barrowman] = Xcp_Barrowman_f( DTF )
%Calculate Xcp according to Barrowman for documentation
        %% Rocket Dimensions

        L = DTF.Length;
        L_cone = DTF.Cone_L;
        L_cyl = L - L_cone;
        D_cyl = DTF.D;
        R_cyl = D_cyl/2;

        % Fin geometry

        fin.n=DTF.fin_n;
        fin.sweep = DTF.fin_sweep;
        fin.h = DTF.fin_h;
        fin.topchord = DTF.fin_top;
        fin.basechord = DTF.fin_base;
        fin.l_forward = tan(fin.sweep)*fin.h; % Length of forward pointing triangle
        fin.l_backward = fin.basechord-fin.topchord-fin.l_forward; % length of backward
        fin.t = DTF.fin_t;
        fin.a_ref = fin.n*fin.h*fin.t;
        fin.area = (fin.topchord+fin.basechord)/2*fin.h;
        fin.a_wet = fin.n*2*fin.area;
        fin.c = (fin.topchord+fin.basechord)/2;
        fin.X_b =  L - fin.basechord; % fin location

        % mid chord sweep
        x1 = fin.h*tan(fin.sweep);
        x2 = x1 + fin.topchord - fin.basechord;
        fin.sweepc = atan2((fin.basechord/2 + (x2-fin.topchord/2)),fin.h);
        fin.l_m = fin.h/cos(fin.sweepc); %midchord length;


        %% Cn_alpha and CoP: Classic Barrowman (According to Box S. et al, 2009)
        % Cone (See Box S., 2009, p9-10)
```

```
        cone.Cn_alpha = 2;
        cone.Xcp = 0.466 * L_cone;
        %Fins (See Box S., 2009, p10)
        K_bf = 1+(DTF.D/2)/(fin.h+DTF.D/2);
        fin.Cn_alpha = K_bf * 4 * fin.n *(fin.h/DTF.D)^2 / ...
        (1+sqrt(1+(2*fin.l_m / (fin.topchord+fin.basechord))^2));
        fin.Xcp = fin.X_b + fin.l_m * (fin.basechord + 2*fin.topchord) / (3*(fin.basechord +
fin.topchord))...
        + (fin.basechord + fin.topchord - (fin.topchord*fin.basechord)/(fin.basechord + fin.topchord))/6;

    Cn_alpha_Barrow = cone.Cn_alpha + fin.Cn_alpha;
        Xcp_Barrowman = (fin.Cn_alpha*fin.Xcp + cone.Cn_alpha*cone.Xcp) / Cn_alpha_Barrow;
        %Static stability amrgin
        Ssm_Barrowman = (Xcp_Barrowman - DTF.Xcm)/DTF.D;

        %% CoP: Center of Planform Area
        % A_planform
        cone.A_plan = 2/3 * L_cone * DTF.D;
        cyl.A_plan = D_cyl* L_cyl;
        fin.A_plan1 = 6*(0.5 * fin.l_forward * fin.h);
        fin.A_plan2 = 6*(fin.topchord^2);
        fin.A_plan3 = 6*(0.5 * fin.l_backward * fin.h);
        A_plan = cone.A_plan + cyl.A_plan + fin.A_plan1 + fin.A_plan2 + fin.A_plan3;
        % Distances center planform to nose cone tip
        cone.X_plan = 5/8 * L_cone;
        cyl.X_plan = L_cone + L_cyl/2;
        fin.X_plan1 = fin.X_b + 2/3 * fin.l_forward;
        fin.X_plan2 = fin.X_b + fin.l_forward + 0.5*fin.topchord;
        fin.X_plan3 = fin.X_b * fin.basechord - 2/3 * fin.l_backward;

        Xcp_Planform = (cone.A_plan * cone.X_plan + cyl.A_plan * cyl.X_plan...
        + fin.A_plan1 * fin.X_plan1 + fin.A_plan2 * fin.X_plan2 + fin.A_plan3 * fin.X_plan3) / A_plan;
    end
```

Acknowledgments

The AstroJays would like to thank their sponsors for their support.

References

[1] Barrowman, J.A., Theoretical Prediction of the Center of Pressure, Cambridge, 1966.
[2] Mandell, G., Caporaso, G., Bengan, W.,Topcis in Advanced Model Rocketry, NARAM-8, 1978.
[3] Niskanen, S., Development of an Open Source model rocket simulation software, M.Sc. thesis, Helsinki University of Technology, 2009. Available at http://openrocket.sourceforge.net/documentation. html Accessed: 2018-05-28
[4] Thrust Curve Hobby Rocket Motor Data http://www.thrustcurve.org// Accessed: 2018-05-28
[5] Intercollegiate Rocket Competition http://www.soundingrocket.org/2017-irec.html
[6]Xiaotong Fu, Nicholas Mavrogiannis, Markela Ibo, Francesca Crivellari, Zachary R. Gagnon, Microfluidic free‐flow zone electrophoresis and isotachophoresis using carbon black nano‐composite PDMS sidewall membranes, Electrophoresis, 31 May, 2016
[7] Simulation Script, RORO, 2017, https://github.com/irec-duster/RORO/ Accessed: 2018-02-18
[8] Box, S., Bishop, C. Hunt, H., Estimating the dynamic and aerodynamic paramters of passively controlled high power rockets for flight simulaton, 2009.