DAYANANDA SAGAR COLLEGE OF ENGINEERING
DEPARTMENT OF MECHANICAL ENGINEERING
(Accredited by NBA)
Accredited by National Assessment & Accreditation Council (NAAC) with ‘A’ Grade
AICTE Approved, an Autonomous Institute Affiliated to VTU, Belagavi
Shavige Malleshwara Hills, Kumaraswamy Layout, Bengaluru-560078

A Project Report on

AUTONOMOUS PRECISION LANDING OF MODEL ROCKETS

Submitted in partial fulfillment for the award of degree of

BACHELOR OF ENGINEERING

IN

MECHANICAL ENGINEERING

Submitted by

1DS17ME093	**Raghavendra N S**
1DS17ME124	**Sohail Nadaf**
1DS17ME131	**Tushar V Kulkarni**
1DS17ME145	**Vivek S Srinivas**

Under the guidance of

Sanketh S
Assistant Professor
Department of Mechanical Engineering
Dayananda Sagar College of Engineering, Bengaluru
S M Hills, Kumaraswamy Layout, Bengaluru - 560078

2020-21

DAYANANDA SAGAR COLLEGE OF ENGINEERING

DEPARTMENT OF MECHANICAL ENGINEERING

(Accredited by NBA)

Accredited by National Assessment & Accreditation Council (NAAC) with 'A' Grade
AICTE Approved, an Autonomous Institute Affiliated to VTU, Belagavi)
Shavige Malleshwara Hills, Kumaraswamy Layout, Bengaluru-560078

Certificate

Certified that the project report entitled '**Autonomous Precision Landing of Model Rockets**' is a bonafide work carried out by **Raghavendra N S, Sohail Nadaf, Tushar V Kulkarni, Vivek S Srinivas**, bearing USN: 1DS17ME093, 1DS17ME124, 1DS17ME131, 1DS17ME145, under the guidance of **Prof. Sanketh S**, Assistant Professor, Department of Mechanical Engineering, Dayananda Sagar College of Engineering, Bengaluru, in partial fulfilment for the award of **Bachelor of Engineering** in Mechanical Engineering of the Visvesvaraya Technological University, Belagavi.

Sanketh S
Assistant Professor
Dayananda Sagar College of Engineering
Bengaluru

Dr. R. Keshavamurthy
Professor & Head, Dept. of Mech. Engg
Dayananda Sagar College of Engineering
Bengaluru

Dr. C.P.S. Prakash
Principal
Dayananda Sagar College of Engineering
Bengaluru

External Viva-Voce

Sl. No.	Name of the Examiner	Signature with date
1		
2		

DECLARATION

We, **Raghavendra N S (1DS17ME093), Sohail Nadaf (1DS17ME124), Tushar V Kulkarni (1DS17ME131), Vivek S Srinivas (1DS17ME145)** hereby declare that the entire work embodied in the project report entitled '**Autonomous Precision Landing of Model Rockets**' has been independently carried out by **Batch 48** under the guidance of **Prof. Sanketh S**, Assistant Professor, Department of Mechanical Engineering, Dayananda Sagar College of Engineering, Bengaluru, in partial fulfilment of the requirements for the award of Bachelor Degree in Mechanical Engineering of Visvesvaraya Technological University, Belagavi.

I further declare that I have not submitted this report either in part or in full to any other university for the award of any degree.

1DS17ME093	Raghavendra N S	Raghav.S
1DS17ME124	Sohail Nadaf	Sohail.
1DS17ME131	Tushar V Kulkarni	Tushar
1DS17ME145	Vivek S Srinivas	VivekS

Place: Bengaluru

Date: 10-May-2021

ABSTRACT

This project aims to build a model rocket that can land vertically. By establishing an automatic control system, we contrive to pull out any perturbations and disturbances ensuring active control. Thrust Vector Control (TVC) is the technique used for attitude (pitch and yaw) control, this gimbal mount allows the rocket to be actively stabilized and is the key component to allow the rocket to land vertically.

A closed-loop feedback control system consisting of a Proportional Integral Derivative (PID) controller is employed to maneuver the rocket, additional components such as the Kalman Filter will be used to eliminate noise and disturbances from sensory feedback. A flight computer is utilized to monitor the rocket, this includes data logging, stabilization, prediction, and state indication. The rocket is powered by a solid propellant motor serving as the propulsion system.

The rocket body and its components are designed considering the principles of Design for Assembly (DFA) and Design for Manufacturing (DFM), using Computer-Aided Design (CAD) software and analyzed using Finite Element Method (FEM) solvers. Computational Fluid Dynamics (CFD) is applied to visualize the flow around the rocket airframe and to evaluate the design of passive fins.

ACKNOWLEDGEMENT

We have taken efforts in this work. However, it would not have been possible without the kind support and help of many individuals and organizations. We would like to extend my sincere thanks to all of them.

We express our deep sense of gratitude and indebtedness to **Prof. Sanketh S**, Assistant Professor, Department of Mechanical Engineering, for his valuable guidance, continuous assistance in the critical appraisal of the thesis.

We express our sincere thanks to **Dr. R. Keshavamurthy**, Professor and HOD, Department of Mechanical Engineering, for providing the facilities required for the completion of this project work.

It is with great pleasure, we extend our gratitude and thanks to **Dr. C P S Prakash**, Principal, Dayananda Sagar College of Engineering, for his encouragement throughout the project.

Contents

List of Tables

List of Figures

List of Acronyms

α	Angle of Attack
ρ	Density
c	Chord
$\mathbf{C_l}$	Sectional Lift Coefficient
$\mathbf{C_D}$	Coefficient of Drag
CFD	Computational Fluid Dynamics
$\mathbf{M}_\infty$	Freestream Mach Number
D	Drag
L	Lift
$\mathbf{r_{le}}$	Leading Edge Radius
DOF	Degrees of Freedom
CG	Center of Gravity
TVC	Thrust Vector Control
NED	North-East-Down
FEM	Finite Element Method
FVM	Finite Volume Method
RANS	Reynolds Averages Navier Stokes
LES	Large Eddy Simulation
DNS	Direct Numerical Simulation
FFF	Fused Filament Fabrication
FDM	Fused Deposition Modelling
PLA	Polylactic Acid
ABS	Acrylonitrile Butadiene Styrene
PETG	Polyethylene Terephthalate Glycol
TPU	Thermoplastic Polyurethane
DFA	Design for Assembly
DFM	Design for Manufacturing
CAD	Computer-Aided Design
FEM	Finite Element Method
CFD	Computational Fluid Dynamics
IMU	Inertial Measurement Unit
AHRS	Attitude Heading Reference System

1 Executive Summary

Traditionally, payloads have been delivered into space using rockets, upon running out of fuel these rocket boosters are expended and either burn up in the atmosphere or crash into the ocean. This is extremely wasteful and is economically unviable. Reusing launch vehicles just like airplanes will lead to a reduction of the cost of access to space by as much as a factor of a hundred. In the past few years, companies like SpaceX and Blue Origin have successfully reused a launch vehicle.

Our project proposes to bridge this gap and contribute to the increasing need for small scale launch vehicles. The goal of this project is to build a fully reusable and autonomously landing model rocket. This involves developing the hardware, software, simulations, and procedures necessary to propulsively land a small scale rocket with the capability of reuse.

Another hurdle we plan to tackle is to propulsively land using a solid propellant motor that doesn't possess throttle control, unlike large-scale rockets that use a liquid propellant engine that could use throttle control. Our project analysis indicates that the successful development of this system shows significant potential to save costs for smaller launch vehicles by enabling them to propulsively land and be reused. For report writing, the team used LaTeX software.

1.1 Objectives

- Building an aerodynamic rocket body and fins to achieve maximum passive stability.
- Developing a Thrust Vector Control (TVC) system for active attitude control.
- Integrating a Flight computer for Data logging and various other operations.
- Developing Landing legs to withstand the impact and ensuring the safety of components.
- Modelling and Simulation of rocket dynamics using MATLAB & Simulink.
- Testing and validation of designed components.

1.2 Literature Review

The concept of retrieving a launch vehicle/rocket after successful execution of its launching stages is an inherently challenging task owing to the large number of uncertainties in the upper atmosphere. These uncertainties coupled with the scale of costs involved with space missions necessitate the implementation of model rockets to realize the behaviour of control systems which are used in precision landing of the rocket. In this report we present an autonomous precision landing control system, along with the necessary dynamics and simulations required to support this claim. As this project involves application of different domains like control systems, spacecraft dynamics and controls, aerodynamic stability, MEMS, the literature survey conducted is modular, with an emphasis on control systems and dynamics.

- M. Haw, *A8-3 Model rocket impulse measurement*, Science One Research Projects (Undergraduate Science Program), University of British Columbia, British Columbia, Canada, May 2009.

  This paper depicts the measurement of thrust from the model rocket engine in the form of impulse. It also gives an emphasis on the measurement of impulse from the rocket engine as it will majorly affect the trajectory of the descending model rocket. The methodology includes the measurement of thrust as a function of time and extracting impulse readings using finite difference approximations.

- R. Ferrante, *A Robust Control Approach for Rocket Landing*, M.S. thesis, Artificial Intelligence, School of Informatics, University of Edinburgh, Scotland, 2017.

  This M.S. thesis provides a concise description of control techniques and simulation environments for modelling a rocket. Although the thesis involves actual scale, it does very well apply to model rockets with suitable assumptions.

- L. Blackmore *Autonomous precision landing of space rockets*, Frontiers of Engineering: Reports on Leading-Edge Engineering from the Symposium 2016.

  The paper discusses the importance of reusing space rockets and its challenges.

- F. Kehl, A. M. Mehta, K. S. J. Pister, *An Attitude Controller for Small Scale Rockets*, Springer Tracts in Advanced Robotics, Jan 2015.

  In this paper, the hardware testing and experimental tests were conducted along with a depiction of implementation of PID control techniques.

- Penn, Kim & Slaton, William, *Measuring Model Rocket Engine Thrust Curves*, The Physics Teacher, 2010.

  The paper provides a methodology for solid propellant model rocket engine thrust curves which is used to determine the engine ignition altitude.

- Aliyu Bhar Kisabo and Aliyu Funmilayo Adebimpe,*State-Space Modeling of a Rocket for Optimal Control System Design*, Centre for Space Transport and Propulsion (CSTP), 2010.

  This paper delineates the basic equations defining the airframe dynamics of a typical six degrees of freedom (6DoFs) are nonlinear and coupled. It also presents a novel-approach to linearization of 6DoF equations in a general form.

- Florian Kehl, Ankur M. Mehta and Kristofer S. J. Pister *An Attitude Controller for Small Scale Rockets*, Springer Tracts in Advanced Robotics, 2015.

  This paper comprises of mini-rocket design, adaptation for mini-rocket design, guidance control system, hardware testing and experimental results such as, sensor validation and attitude control validation.

- Mark D. Stevenson and Jeffrey V. Zweber, *Multidisciplinary Conceptual Vehicle Modelling*, American Institute for Aeronautics and Astronautics (AIAA), 2001.

  This paper narrates the discipline interaction, vehicle geometry, rocket engine design code, weight analysis, aerodynamic analysis, trajectory analysis, trade studies and thrust-to-weight optimization.

- B. Stevens and F. Lewis, *Aircraft Control and Simulation*. Wiley, 2003.

  This textbook covers flight control systems, flight dynamics, aircraft modeling, and flight simulation from both classical design and modern perspectives, as well as two new chapters on the modeling, simulation, and adaptive control of unmanned aerial vehicles.

- S. Niskanen, “Openrocket technical documentation,” *Development of an Open Source model rocket simulation software*.

  The thesis is used as the basis of this technical documentation, which is updated to account for later development in the software. Discussed the development and documentation for relatively easy, yet reasonably accurate methods for the calculation of the fundamental aerodynamic properties of model. Implementation of a cross-platform, open source model rocket design and simulation software.

- P. Kuentzmann, “Introduction to Solid Rocket Propulsion,” Office National d’Etudes et de Recherches Aérospatiales, 2000.

  This paper describes the fundamentals of solid rocket propulsion (SRM) from the elementary analysis of rocket operation. It also contains a short review of internal aerodynamics of SRM and the basic formula to predict the steady-state operation pressure.

2 Management Summary

2.1 Team Organization

The team comprised of four members and the responsibilities were equally distributed.

- Airframe & Propulsion - Research, Sizing, Design, CFD & Optimization, Selection of Propulsion System
- Modelling & Simulation - Research, Dynamics & Control, Analysis using MATLAB & Simulink
- Thrust Vector Control - Research, Fabrication, Testing & Optimization
- Flight Computer - Research, Flight Computer Design, PCB Design, Testing

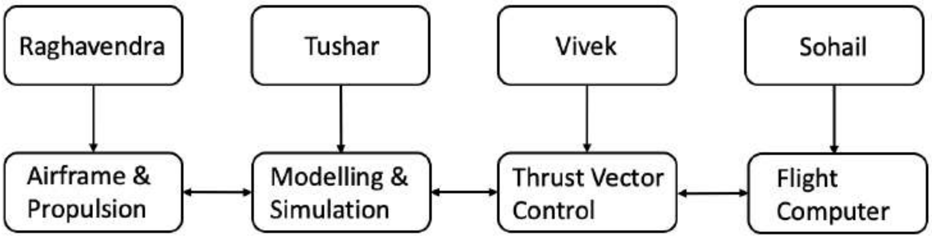


Figure 1: Team Organization

2.2 Milestone Chart

The Gantt chart was implemented to plan the workflow and to keep a track of progress. The blue bar shows the planned days and the grey bar shows the actual days. From the chart it is evident that testing and procurement of materials took longer than anticipated.

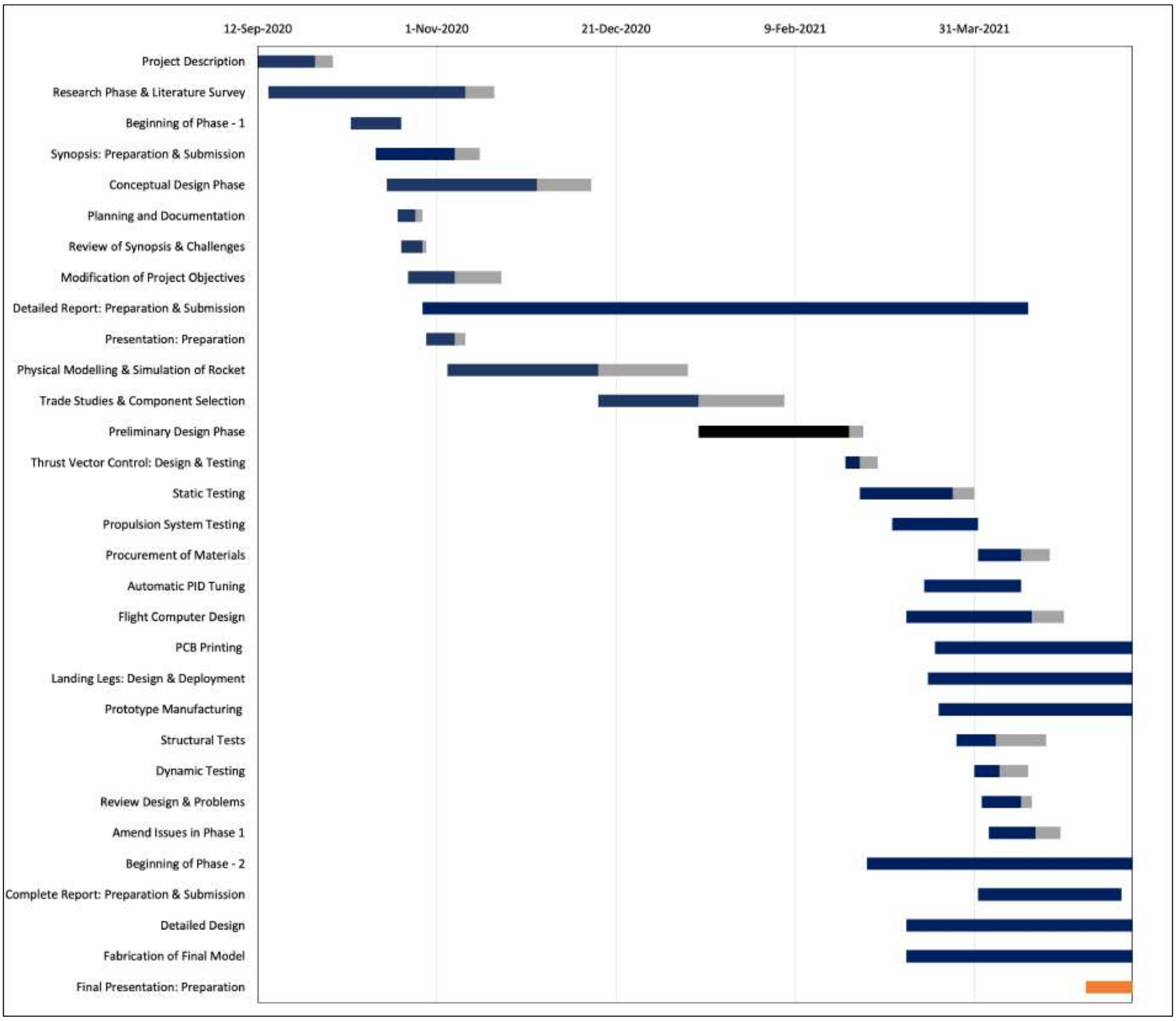
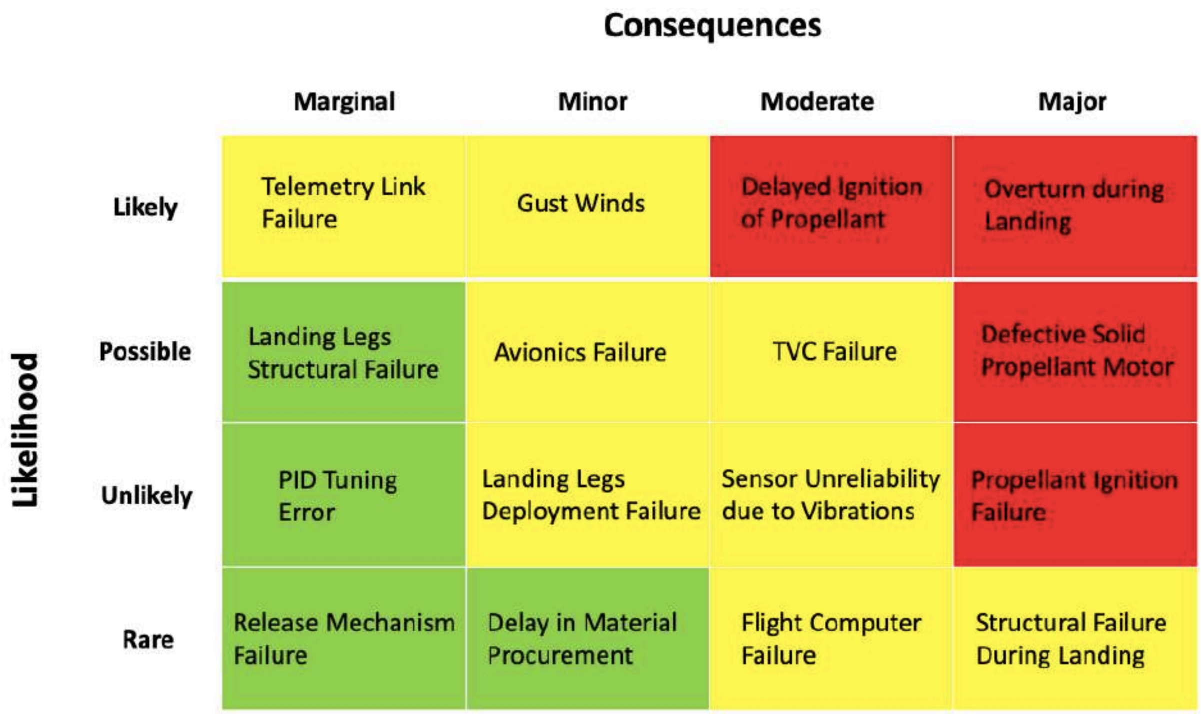


Figure 2: Gantt Chart

2.3 Risk Analysis

A risk assessment chart is depicted below and it helps in understanding and managing the risks involved. It is apparent from the chart that both the likelihood and consequences of tipping over during landing is likely and the consequences are high.



Consequences

Likelihood

	Marginal	Minor	Moderate	Major
Likely	Telemetry Link Failure	Gust Winds	Delayed Ignition of Propellant	Overturn during Landing
Possible	Landing Legs Structural Failure	Avionics Failure	TVC Failure	Defective Solid Propellant Motor
Unlikely	PID Tuning Error	Landing Legs Deployment Failure	Sensor Unreliability due to Vibrations	Propellant Ignition Failure
Rare	Release Mechanism Failure	Delay in Material Procurement	Flight Computer Failure	Structural Failure During Landing

Figure 3: Risk Identification

2.4 Budget Details

The team had a budget of Rs. 25,000 for the entire project. We managed to stay under budget, the team's total expenses came up to be Rs. 21,200. Expenditure towards avionics accounted for majority of team's total budget.

Sl. No.	Description	Cost (in Rs.)
1	Avionics	9700
2	PLA Material	1200
3	Testing Equipment	5200
4	Body	600
5	Miscellaneous	4500
6	**Total**	**21200**

Table 1: Cost break-down

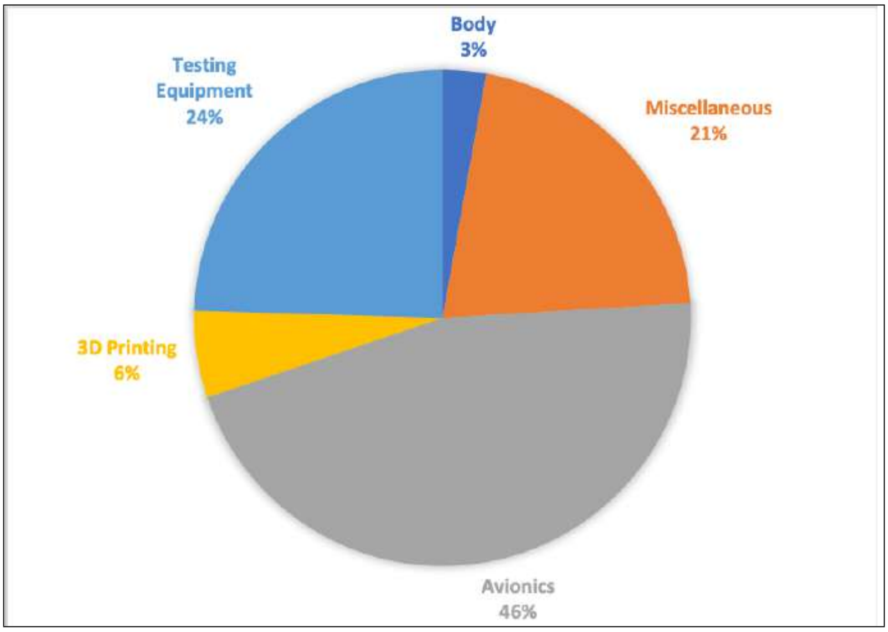


Figure 4: Cost Distribution

3 Conceptual Design

3.1 Overall Methodology

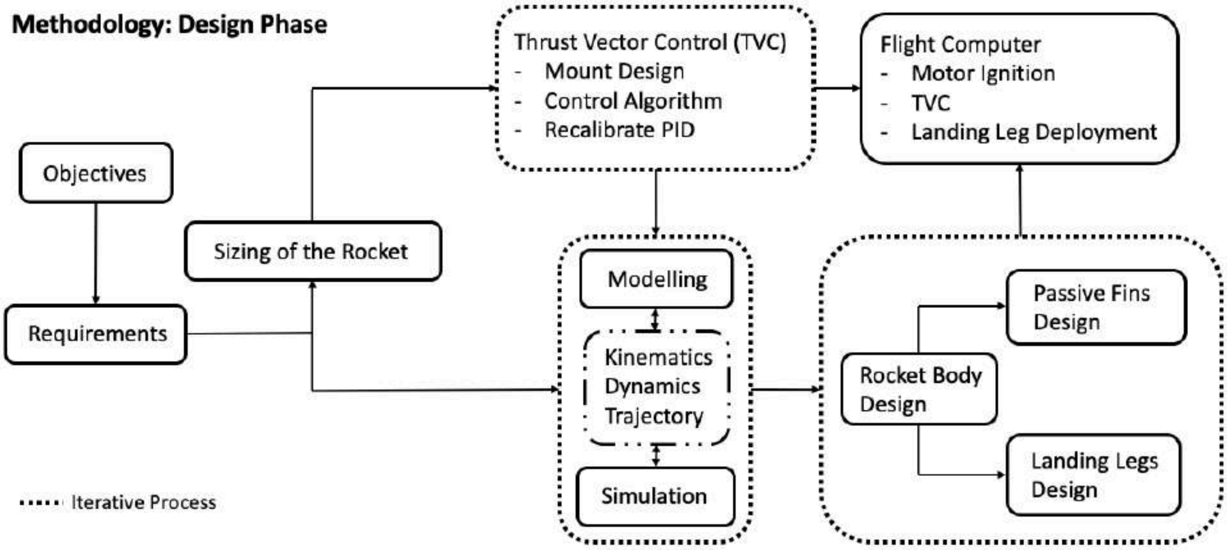


Figure 5: Methodology: Design Phase

We began by clearly defining the project's core objectives, which was followed by the description of requirements, which led to the rocket's sizing. TVC mount was created with the help of computer-aided design (CAD) software. Iterations were carried out until the desired product was achieved. Simultaneously, MATLAB and Simulink were used to model and simulate the rocket dynamics and trajectory. The rocket body, fins, and landing legs were then designed. The shape of fins was analysed using computational fluid dynamics (CFD). Various tasks, such as TVC, motor ignition, and landing leg deployment, were programmed into the flight computer. Lastly, the rocket's flight was simulated to ascertain PID gains.

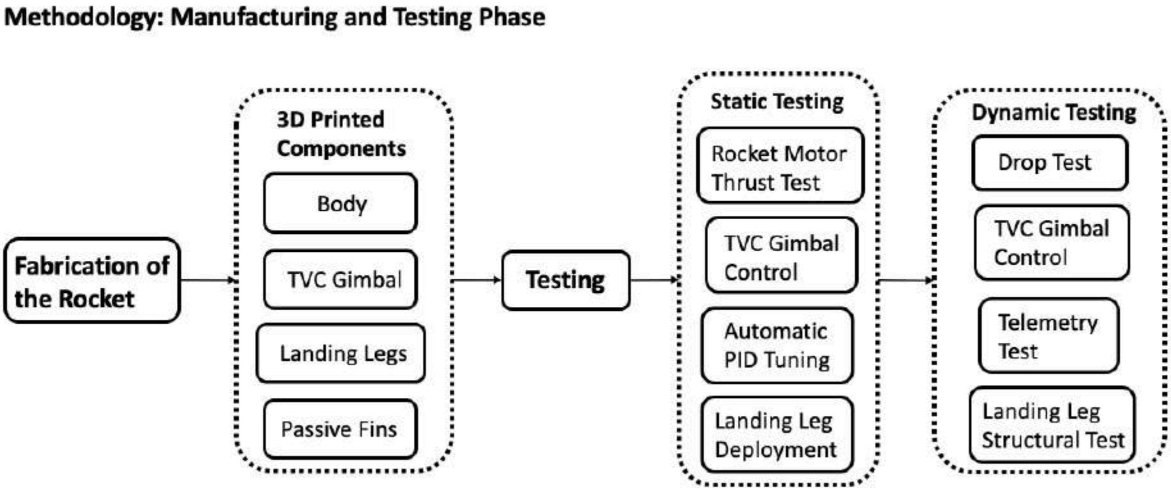


Figure 6: Methodology: Manufacturing Phase

The manufacturing and testing phase were carried out after conducting all the necessary computer simulations, design finalization and CFD analyses. 3D printing was used to fabricate the landing legs, fins and thrust vector control. The necessary components for testing as well as some internal parts for the rocket body were printed using a 3D printer. The manufacturing phase was followed by the testing phase, which began with static tests. First, a static thrust test of the rocket motor, a TVC gimbal test, and a landing leg deployment test. Then we moved on to dynamic testing, where we performed a drop test, a telemetry test, and an impact test on the landing legs.

3.1.1 Trade Studies

We used a Pugh Matrix method for selection of propulsion system, beginning with choosing the type of propulsion i.e., chemical propulsion and electric propulsion. This method of selection helped us objectively choose the kind of propulsion while having numerous subjective and arbitrary parameters. The weightage for the parameters were given as per the team's objectives and priorities, and accordingly weight and cost were given the utmost importance considering the design restriction. The reason behind considering electric propulsion in the first place was to have more control over the throttle and have capacity to conduct frequent and a number of tests without worrying about the flight time i.e., the endurance. This table below shows that the chemical propulsion is relatively more desirable for our project.

Parameters	Weight	Chemical Propulsion	Electric Propulsion
Weight	5	5	1
Complexity	3	3	3
Reusability	3	1	5
Feasibility	4	3	3
Cost	4	4	2
Total		**65**	49

Table 2: Chemical Propulsion Vs Electric Propulsion

As per the kind of propellant, we considered both solid propellant and liquid propellant. The weightage for the parameters were given as per the team's objectives and priorities, and accordingly ability to throttle and cost were given the utmost importance. From this table below we concluded to go with solid propellant.

Parameters	Weight	Solid Propellant	Liquid Propellant
Throttleable	3	1	5
Complexity	3	4	2
Reusability	3	1	4
Feasibility	4	5	3
Cost	4	5	1
Total		**58**	49

Table 3: Solid Propellant Vs Liquid Propellant

4 Airframe

4.1 Methodology

The airframe design was dictated by two parameters, namely weight and lower bending moment to reduce the chances of the rocket tipping over upon landing and this is imperative in achieving the team's objectives.

For the airframe, we considered cardboard as the choice of material since it had desirable properties such as high strength to weight ratio, low density which translates lower weight, easy to work with and economical.

The airframe diameter and length were decided upon iterative process. Parametric analysis was carried out using fineness ratio (length to diameter ratio) and this helped us in determining the optimum ratio. The range of fineness ratio is generally 10:1 to 20:1 [1] [2] [3]. The parameters that are affected by fineness ratio are:

- Skin friction drag
- Pressure drag

- Bending moment

The chosen ratio is 10:1 which is the lowest fineness ratio since this aids in maintaining a lower overall weight and also assists in keeping the bending moments to a minimum. The skin friction drag is reduced mainly due to relatively lower wetted are and the pressure drag is slightly increased this is because the pressure distribution around a body which has low fineness ratio is not ideal and this causes the increase in pressure drag. The airframe houses the propulsion system, thrust vector control system and electronics. Externally, the fins and landing legs are attached to the airframe as well.

4.1.1 Airframe Mechanical Design

The airframe design and modelling was accomplished on Autodesk Fusion 360. As for the design approach, the team was limited in terms of varying the shell thickness of the airframe due to unavailability of cardboard tubes which was the material of choice for the airframe. The airframe's diameter was chosen based on the propellent's diameter and thrust vector control system's outer gimbal diameter apart from fineness ratio. Consequently the length was determined.

Figure 7: Rocket Airframe

The material properties of rocket airframe tube i.e., cardboard is presented in the table below. A noteworthy property of cardboard is its endurance, high strength to weight ratio, low impact on sudden impulsive forces and also substantially economical.

Sl. No.	Description	Value
1	Density	195 kg/m^3
2	Poisson's Ratio	0.34
3	Elasticity Modulus	2194 MPa
4	Shear Modulus	565 MPa

Table 4: Cardboard Material Properties

Sl. No.	Parameters	Values
1	Length	0.762m
2	Outer Diameter	0.076m
3	Inner Diameter	0.073m
4	Thickness	0.003m
5	Fineness Ratio	10

Table 5: Airframe Specifications

4.2 Landing Legs

4.2.1 Design

The landing leg design was mainly inspired from SpaceX's Falcon 9 Rocket. The team used a open source design from K-9 landing leg system as a basis and we further modified the design to optimize for weight and strength. The derived design helped the team to accelerate the process and arrive at the final design with minimum iterations. The choice of material for this subsystem was Polyactic Acid (PLA) largely since the team had an easy access to a 3D printer and it was economical too. The landing leg sub-system consists of four landing legs, four struts to support the legs upon landing and a circular frame which houses all these. This circular frame is directly attached to the airframe with the help of screws. The figures below show both the open and closed form of the landing legs.

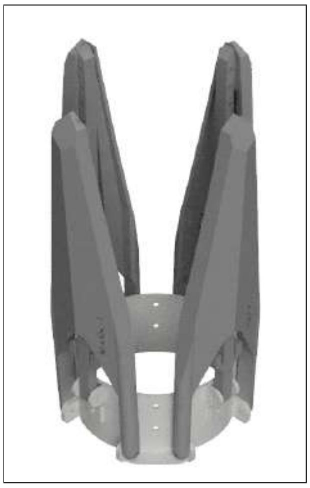


Figure 8: Landing Legs Assembly Closed Form

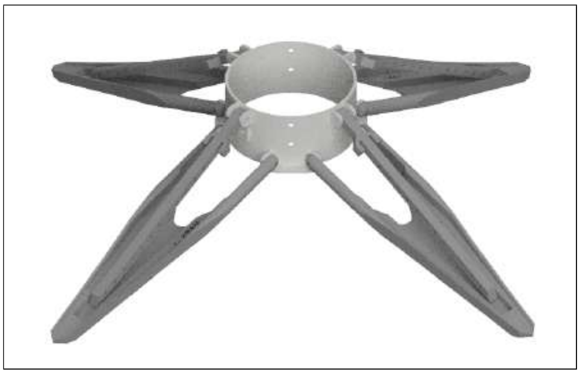


Figure 9: Landing Legs Assembly Open Form

In the figure below, the side view of the landing leg is shown, the landing legs are in open configuration and the opening angle for the leg is 109.4^{o} and for the strut it is 115.4^{o}.

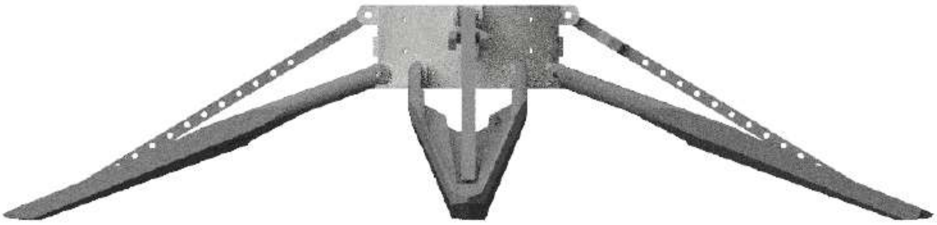


Figure 10: Landing Legs Assembly Side View

The choice of materials as previously mentioned is dictated by strength, weight and cost. Hence the team decided to move forward with PLA. PLA has significantly better elasticity and it is not as brittle as other alternatives such as ABS. Flexural strength of the material is also high which helps upon on impact on landing. The material properties of PLA are presented in the table below.

Sl. No.	Description	Value
1	Density	1250 kg/m^3
2	Poisson's Ratio	0.34
3	Elasticity Modulus	3500 MPa
4	Shear Modulus	1287 MPa
1	Melting Temperature	170^{o}C
2	Injection Molding Temperature	240^{o}C
3	Tensile Strength	66 MPa
4	Flexural Strength	110 MPa

Table 6: Polyactic Acid (PLA) Material Properties

Sl. No.	Parameters	Values
1	Length	0.195m
2	Root Width	0.45m
3	Tip Width	0.015m
4	Thickness	0.01m
5	Taper Ratio	30
6	Opening Angle	109.4^o

Table 7: Landing Leg Specifications

Sl. No.	Parameters	Values
1	Length	0.165m
2	Root Width	0.009m
3	Tip Width	0.005m
4	Thickness	0.01m
6	Opening Angle	115.4^o

Table 8: Landing Leg Strut Specifications

Sl. No.	Parameters	Values
1	Length	0.4m
2	Outer Diameter	0.079m
3	Inner Diameter	0.076m
4	Thickness	0.003m

Table 9: Landing Leg Circular Frame Specifications

4.3 Passive Fins

The stability and control of the rocket is determined by the fins's reference area, wetted area, airfoil, thickness, number of fins and position of the fins. Usually rocket's fins are usually placed in the bottom of the body around the propulsion system whereas in our project we only focused on landing the rocket and hence the fins were positioned at the top of the rocket and this helps to head the TVC and propulsion system into the wind. These fins have a passive control during the descent of the rocket and the team decided not to go with active control of the fins as this could increase the complexity of control system design as well as the dynamics of flight.

4.3.1 Design

For the design of fins the team went for a flat plate design and discarded the use of any airfoils for the sake of simplicity and also upon analysis it deemed unnecessary. The ratio of fin chord to diameter ratio and fin span to diameter ratio is 2:1 and 1:1 respectively [1]. This was considered as a design criteria and we modelled the fins using Autodesk Fusion 360 software. The team again chose to use PLA as the material for fins but the weight of the fins went beyond the restricted limit, so weight reduction was carried out and it was layered with MonoKote to cover the cut-outs. The fins were attached with a circular ring and the circular ring is attached to the airframe with screws. The range for sweep angle is between 45 to 70 degrees [1], the sweep angle the team chose was 45^o as the flight was in subsonic region.

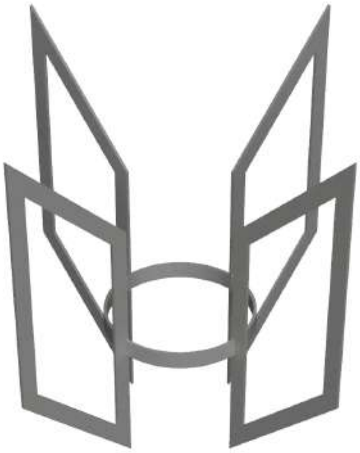


Figure 11: Fins Assembly Render

Sl. No.	Parameters	Values
1	Chord	0.152m
2	Span	0.076m
3	Thickness	0.003m
4	Sweep Angle	45^o
5	Chord to Diameter Ratio	2:1
6	Span to Diameter Ratio	1:1
6	Number of Fins	4

Table 10: Fins Specifications

4.4 Assembly

The rocket's main parts include fins, bulkheads, flight computer, airframe, landing legs, firewall, propellant and thrust vector control. The picture below depicts all the components and a render of the assembled rocket. For the rocket's assembly the team decided to use temporary joints instead of permanent joints so as to increase modularity and ease of replacement of parts in case of damage. For assembly, screws were used for attaching fins, TVC, bulkheads and landing legs.

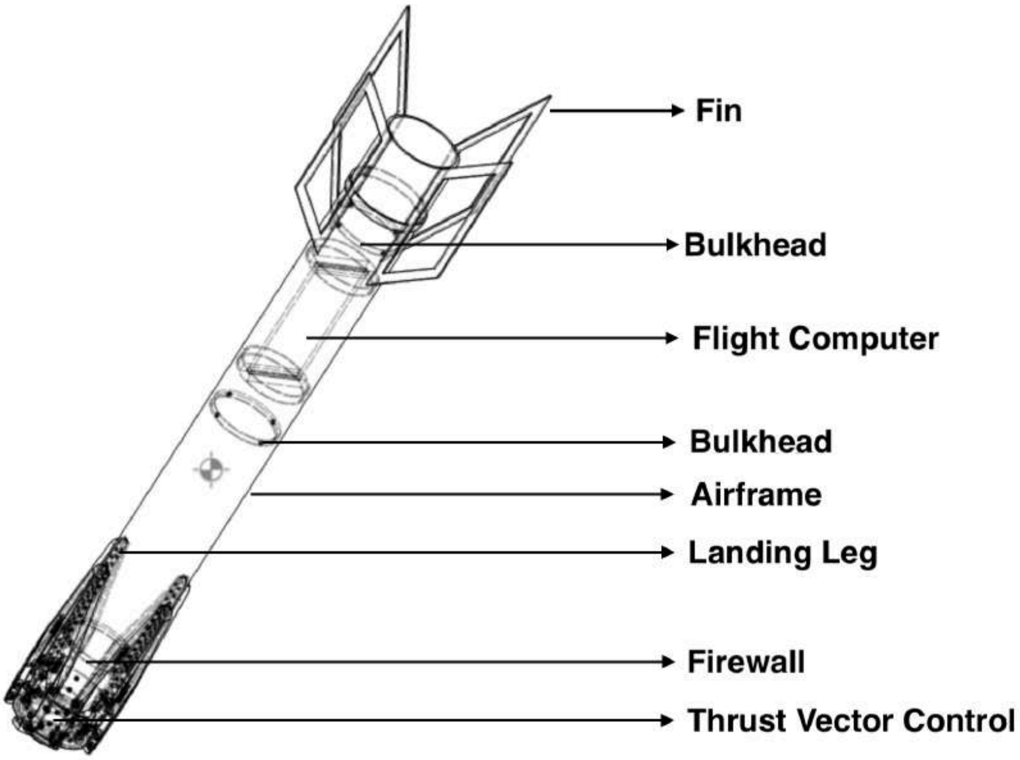


Figure 12: Rocket Part Description

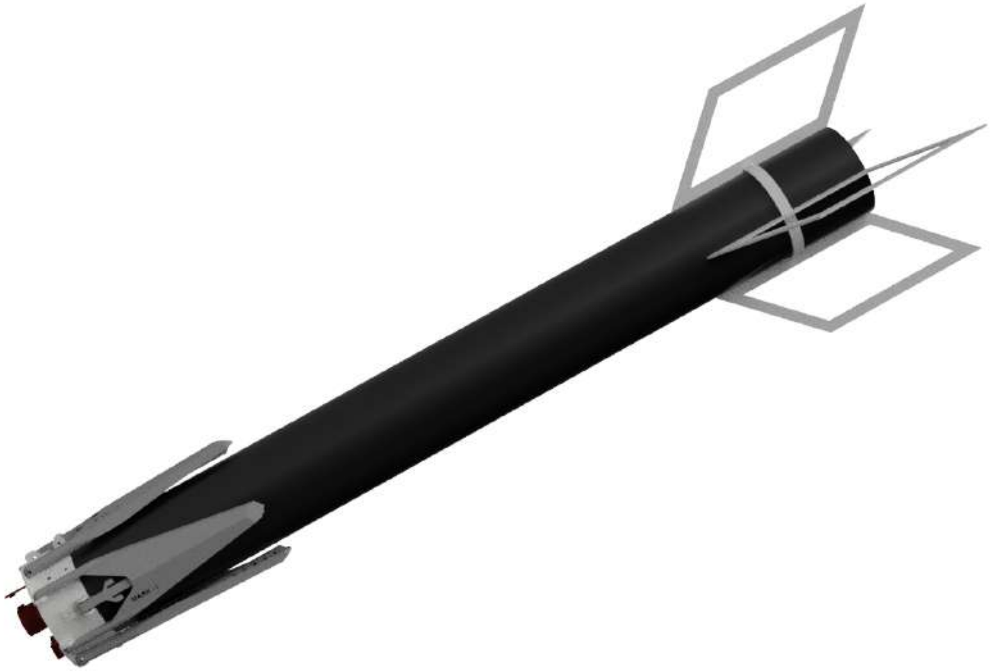


Figure 13: Rocket Assembly Render

The table below lists all components weight with airframe being the major contribution followed by landing legs and flight computer. The total weight of the rocket came up to be 842.6 grams.

Sl. No.	**Components**	**Values (in grams)**
1	Airframe	370
2	Fins	55.6
3	Landing Legs	214.5
4	Propulsion	50
5	Thrust Vector Control	52.5
6	Flight Computer	100
7	**Total Weight**	**842.**6

Table 11: Weight Distribution

Sl. No.	**Parameters**	**Values**
1	Length	0.835m
2	Width	0.381m
3	Weight	842.6g

Table 12: Overall Specifications

4.5 Analysis

4.5.1 Airframe - Structural Analysis

The airframe takes in notable amount of stress during descent phase, apart from the force due to its weight a landing impact load of five times its weight is applied on the airframe. The stresses on the airframe also includes upward thrust force from the propulsion system which is transferred through the thrust vector control system. Autodesk Fusion 360 software was used for both modelling and analysis using finite element method. In the Figure 14 below von Mises stress analysis is carried out and the results are tabulated. The applied force is 50N on the rim of the airframe which is a point force, the stresses and the displacement due to stress is minimal and within the factor of safety and this does not lead to bending or yielding of airframe.

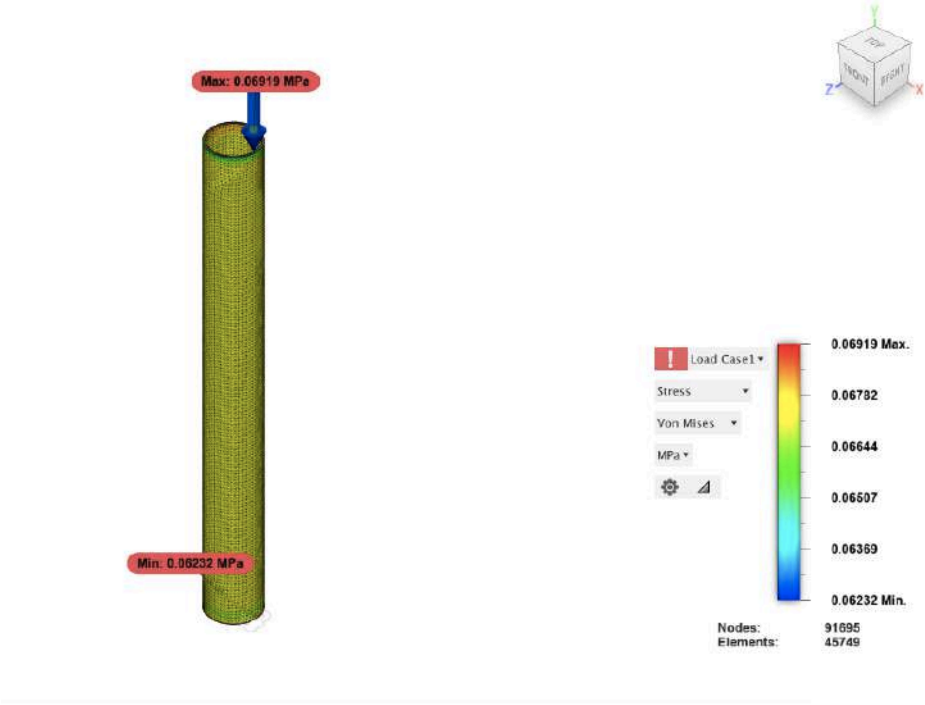


Figure 14: Airframe von Mises Stress

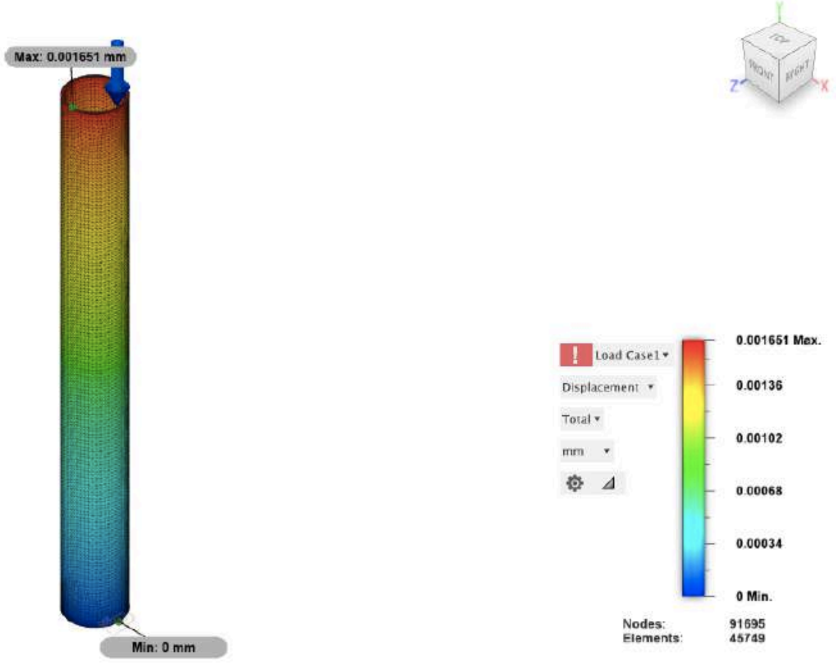


Figure 15: Airframe Displacement

Sl. No.	Parameters	Values
1	Applied Force	50N
2	Maximum Stress	0.069MPa
3	Minimum Stress	0.063MPa
4	Maximum Displacement	0.002mm
5	Minimum Displacement	0mm
6	Mesh Size	1mm
7	Nodes	91695
8	Elements	45749

Table 13: Airframe Structural Analysis Results

4.5.2 Airframe - Modal Frequency Analysis

Modal frequency analysis was conducted to determine the natural frequency of the airframe. This was considered paramount since we aimed to increase the natural frequency of the airframe to minimize flutter by preventing the occurrence of resonance. The potential of resonance can lead to colossal damage to the airframe, hence this analysis was helpful in determining the eigenfrequencies and its respective eigenmodes. The natural frequencies and the mode shapes are tabulated.

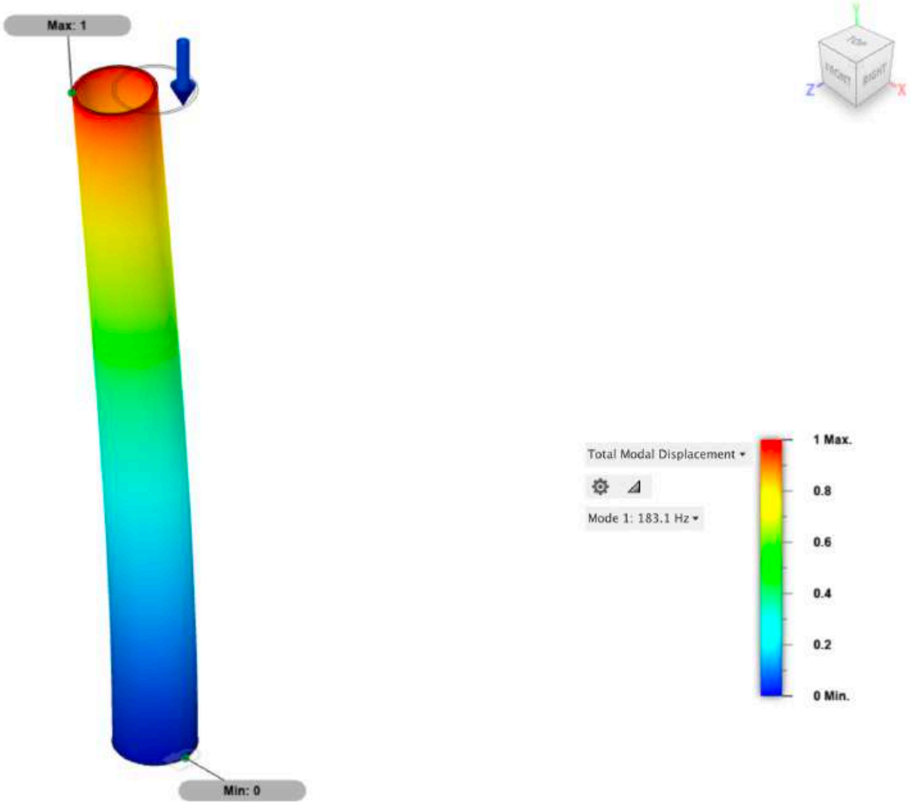


Figure 16: Airframe Mode 1 Shape

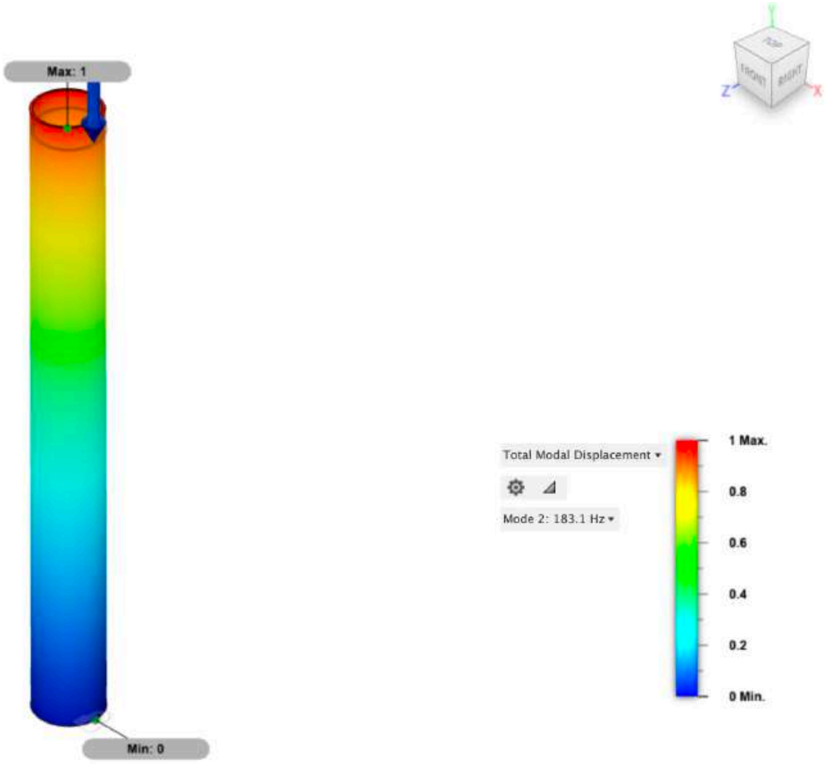


Figure 17: Airframe Mode 2 Shape

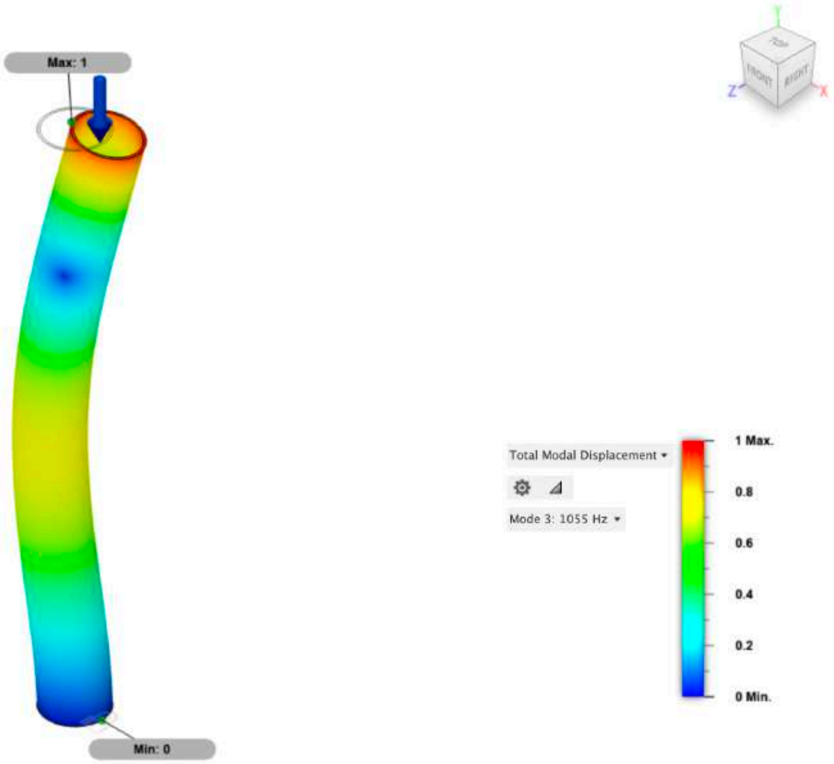


Figure 18: Airframe Mode 3 Shape

Sl. No.	Mode Shape	Natural Frequency (in Hz)
1	Mode 1	183.1
2	Mode 2	645.7
3	Mode 3	1055

Table 14: Modal Frequencies of Airframe

4.5.3 Fins - Structural Analysis

The fins undergo immense flutter during descent phase and since they play a crucial role in enabling passive stability so as to maintain upright position, following through with static structural analysis was integral. The joints strength, flutter analysis, deformation, maximum and minimum stresses and strain, and the displacement due to stress were the parameters that were evaluated. A uniformly applied force of 10N was applied on the each of the four fins and from the von Misses stress analysis it is evident that the stress was mostly concentrated at the joint of fin ring and the fins while the displacement of the fins was at the far end of the fin. All the parameters and their are tabulated below.

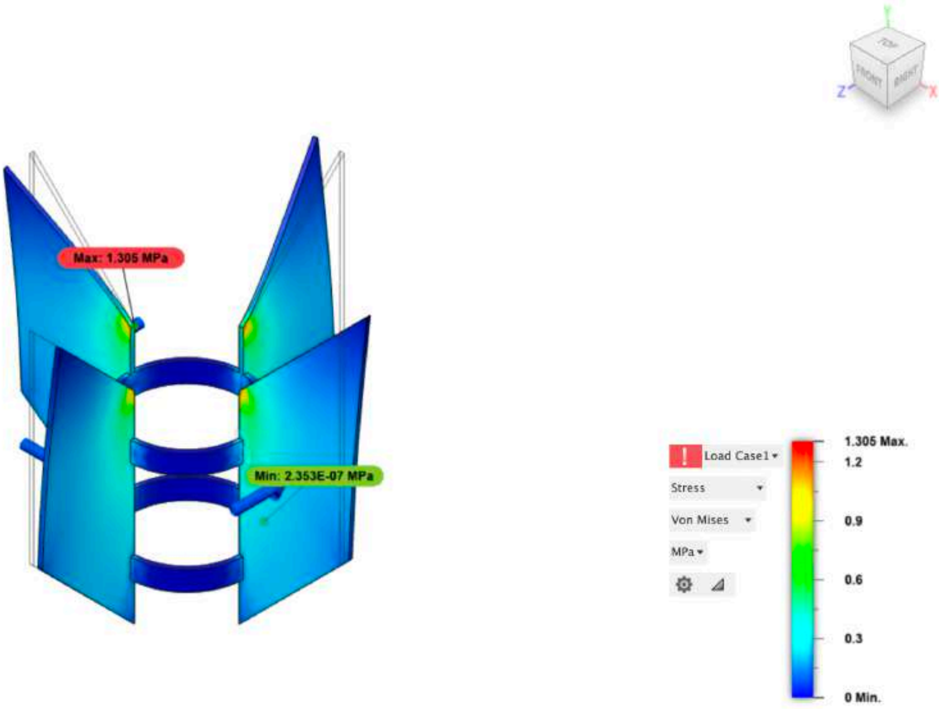


Figure 19: Fins von Mises Stress

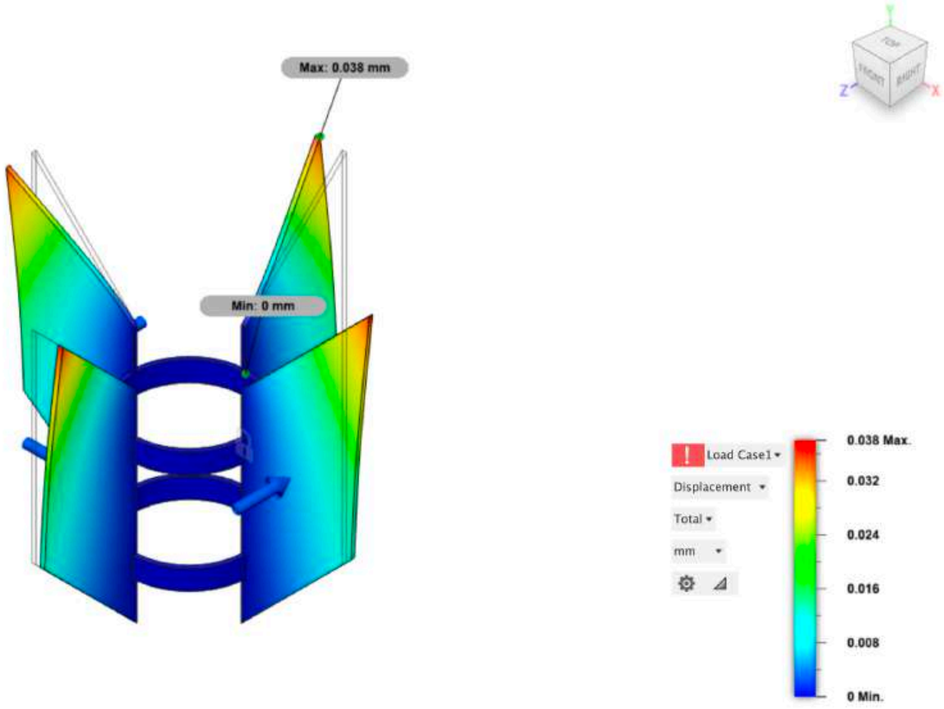


Figure 20: Fins Displacement

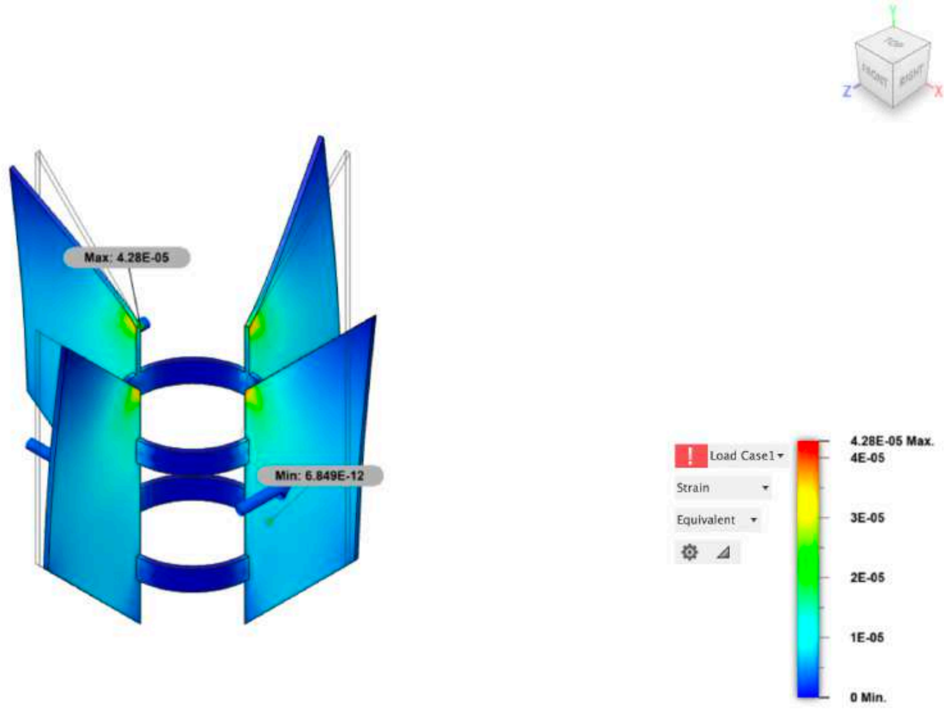


Figure 21: Fins Strain

Sl. No.	Parameters	Values
1	Applied Force	10N
2	Maximum Stress	1.305MPa
3	Minimum Stress	2.35e-07MPa
4	Maximum Displacement	0.038mm
5	Minimum Displacement	0mm
6	Maximum Strain	4.28e-05
7	Minimum Strain	6.84e-12
8	Mesh Size	1mm

Table 15: Fins Structural Analysis Results

4.5.4 Fins - Modal Frequency Analysis

Modal frequency analysis was conducted to determine the natural frequency of the fins. This was considered paramount since we aimed to increase the natural frequency of the fins to minimize flutter by preventing the occurrence of resonance. The potential of resonance can lead to colossal damage to the fins and fins play a huge role in maintaining the stability of the flight during descent, hence this analysis was helpful in determining the eigenfrequencies and its respective eigenmodes. The natural frequency of fins in mode 8 is 1216Hz.

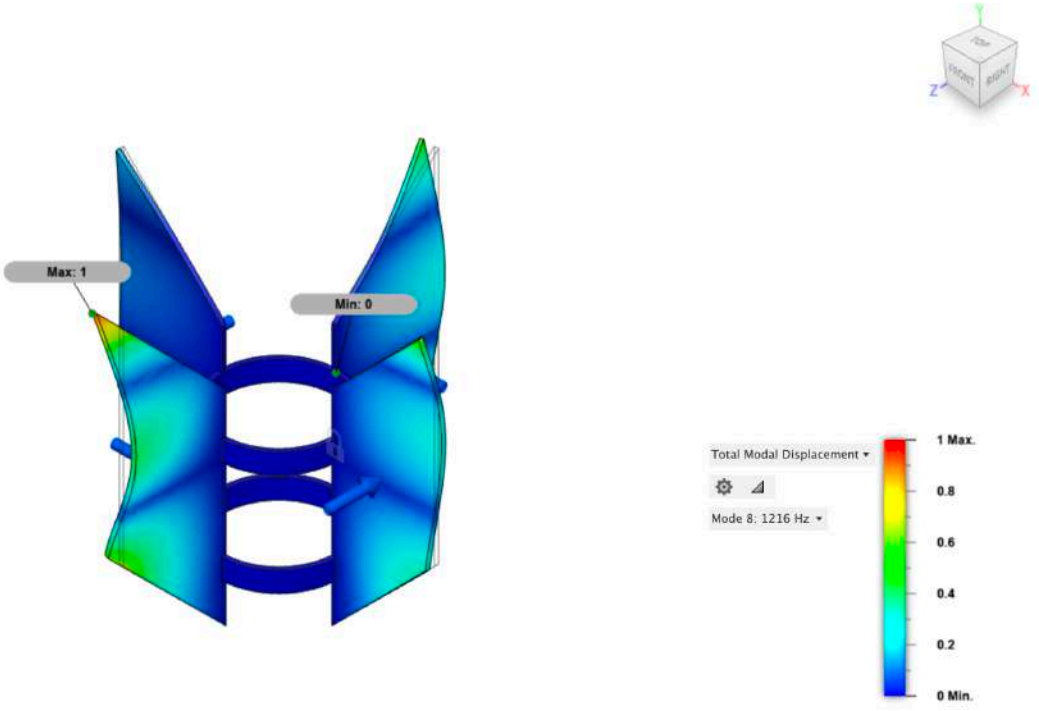


Figure 22: Fins Mode 8 Shape

4.5.5 Landing Legs - Structural Analysis

The landing legs is the only component that is going to experience imminent stress from the impact due to landing. The stresses from landing legs is transferred to the struts which help them stay in that position. The stresses from the struts are transmitted to the circular frame and then to the airframe. The analysis of von Mises stress on landing legs showed significant stress at the joint of landing leg to the circular frame and minimum stress at the point of contact with the ground. For this analysis an impact force of 50N was applied which is five times the weight of the entire rocket and upon analysis, the landing legs can sustain the stresses without leading to breaking or yielding of the material. The results from the analysis are tabulated below.

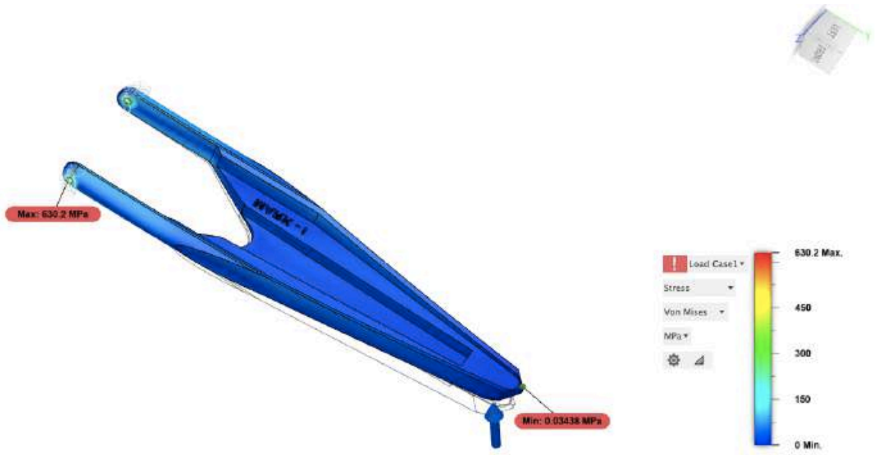


Figure 23: Landing Leg von Mises Stress

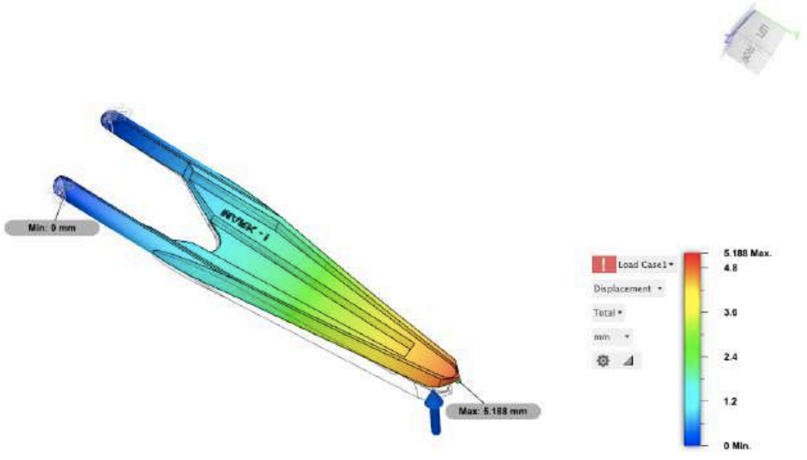


Figure 24: Landing Leg Displacement

Sl. No.	Parameters	Values
1	Applied Force	50N
2	Maximum Stress	630.2MPa
3	Minimum Stress	0.034MPa
4	Maximum Displacement	5.188mm
5	Minimum Displacement	0mm
6	Mesh Size	1mm

Table 16: Landing Leg Structural Analysis Results

Similar to landing legs static structural analysis, we carried out a structural analysis on the landing leg strut and the results too were quite similar. The applied force was 50N at the contact point and maximum stress was experienced at the joint of the strut with the circular frame. The results from the analysis are tabulated below.

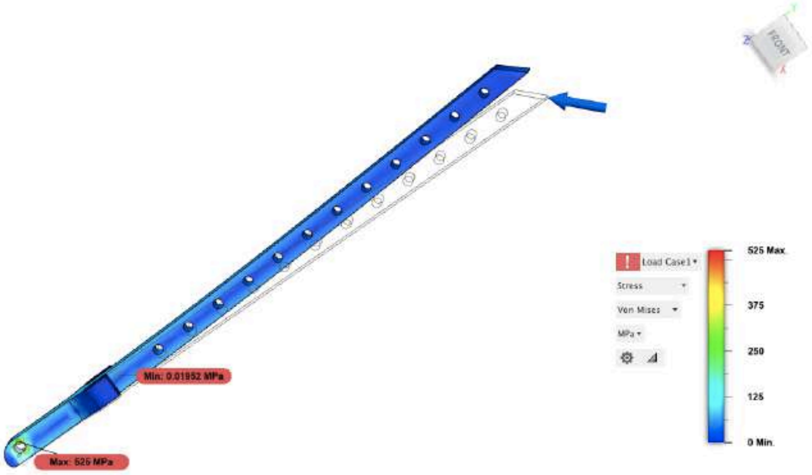


Figure 25: Landing Leg Strut von Mises Stress

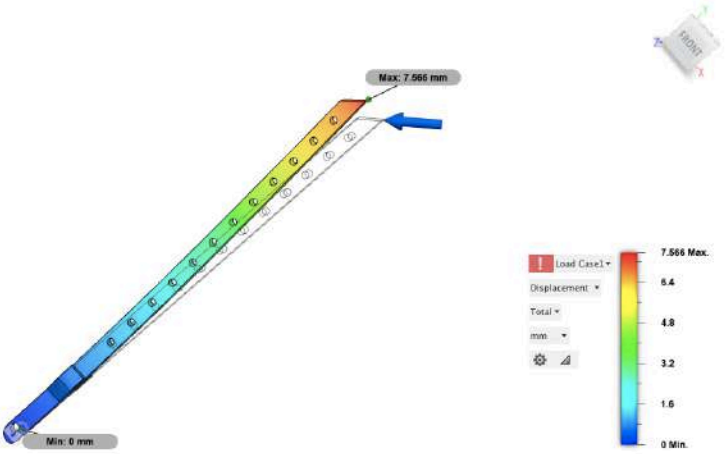


Figure 26: Landing Leg Strut Displacement

Sl. No.	Parameters	Values
1	Applied Force	50N
2	Maximum Stress	525MPa
3	Minimum Stress	0.019MPa
4	Maximum Displacement	7.566mm
5	Minimum Displacement	0mm
6	Mesh Size	1mm

Table 17: Landing Leg Strut Structural Analysis Results

4.5.6 Bulkhead - Structural Analysis

The bulkhead analysis was conducted to make sure that PLA material could withstand the forces due to impact from landing. The bulkhead is also used a firewall as it is placed right above the propulsion and the TVC system. An uniformly distributed force of 10N is applied to the circumference of the bulkhead. The results from the analysis are tabulated below.

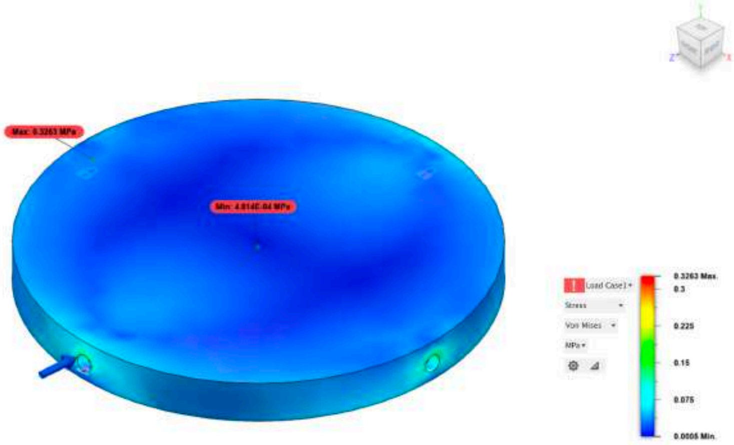


Figure 27: Bulkhead von Mises Stress

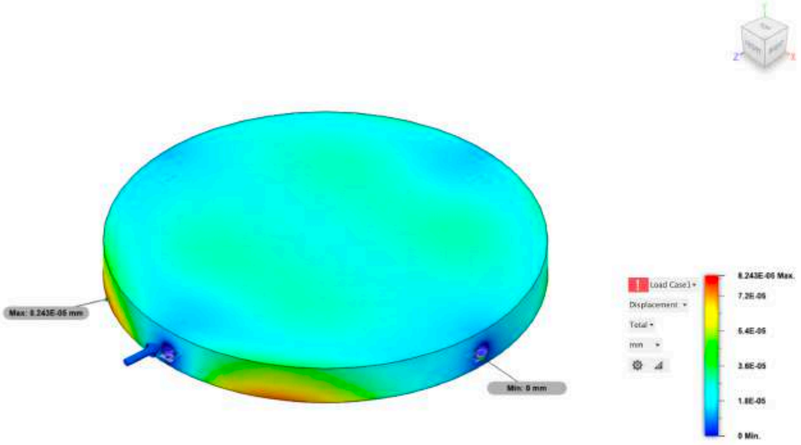


Figure 28: Bulkhead Displacement

Sl. No.	Parameters	Values
1	Applied Force	10N
2	Maximum Stress	0.326MPa
3	Minimum Stress	4.81e-07MPa
4	Maximum Displacement	8.24e-05mm
5	Minimum Displacement	0mm
6	Mesh Size	1mm

Table 18: Bulkhead Structural Analysis Results

5 Propulsion

5.1 Methodology

The propulsion system was decided upon conducting trade study analysis and solid propellant was chosen. The primary purpose of propulsion system with respect to our project objectives is to reduce the velocity and acceleration during descent phase of the rocket, especially the velocity of the rocket right before touch down should be decreased to almost zero. To achieve this, the team decided to look through numerous classes of solid propellent and we chose class F motor i.e., F-15. This was decided after running various simulations on OpenRocket and Simulink to test if the average thrust, maximum thrust and the burn time is sufficient to reach the almost zero velocity at touch down. This is required for a successful and smooth landing without tipping over. Further, the propellant's thrust curve is studied and analyzed to understand its characteristics, and this is imperative to figure out the altitude for reigniting the motor.

5.1.1 Solid Rocket Motor Specifications

The solid rocket motors were compared on Estes and on National Rocketry Association website and the specification are tabulated below [4].

Sl. No.	Parameters	Values
1	Length	114mm
2	Diameter	29mm
3	Estimated Weight	101.8g
4	Total Impulse	46.61N
5	Time Delay	4 seconds
6	Maximum Thrust	25.26N
7	Average Thrust	14.5N
8	Initial Thrust	14.3N
9	Thrust Duration	3.45 seconds
10	Initial Weight	101.5g
11	Propellant Weight	60g
12	Burn Time	3.5 seconds
13	Propellant	Black Powder
14	Manufacturer	Estes Industries

Table 19: F-15 Motor Specications

5.1.2 Motor Performance Analysis

The solid rocket motors were compared on Estes and on National Rocketry Association website and the thrust curve of the chosen motor and its simulation on OpenRocket is shown below [5].

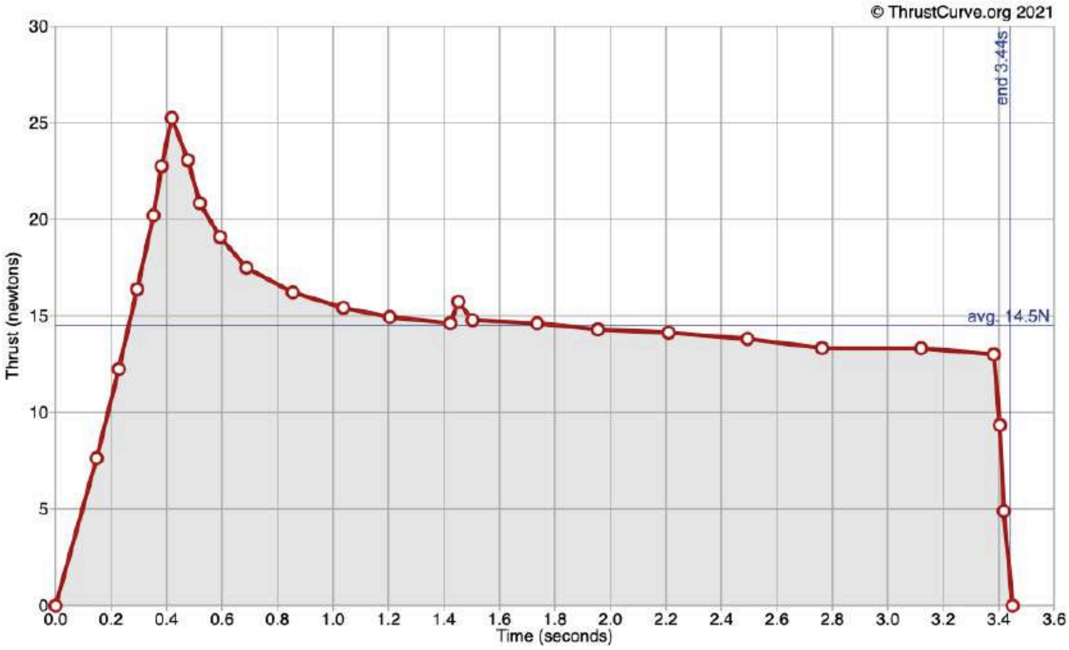


Figure 29: F-15 Motor Thrust Curve

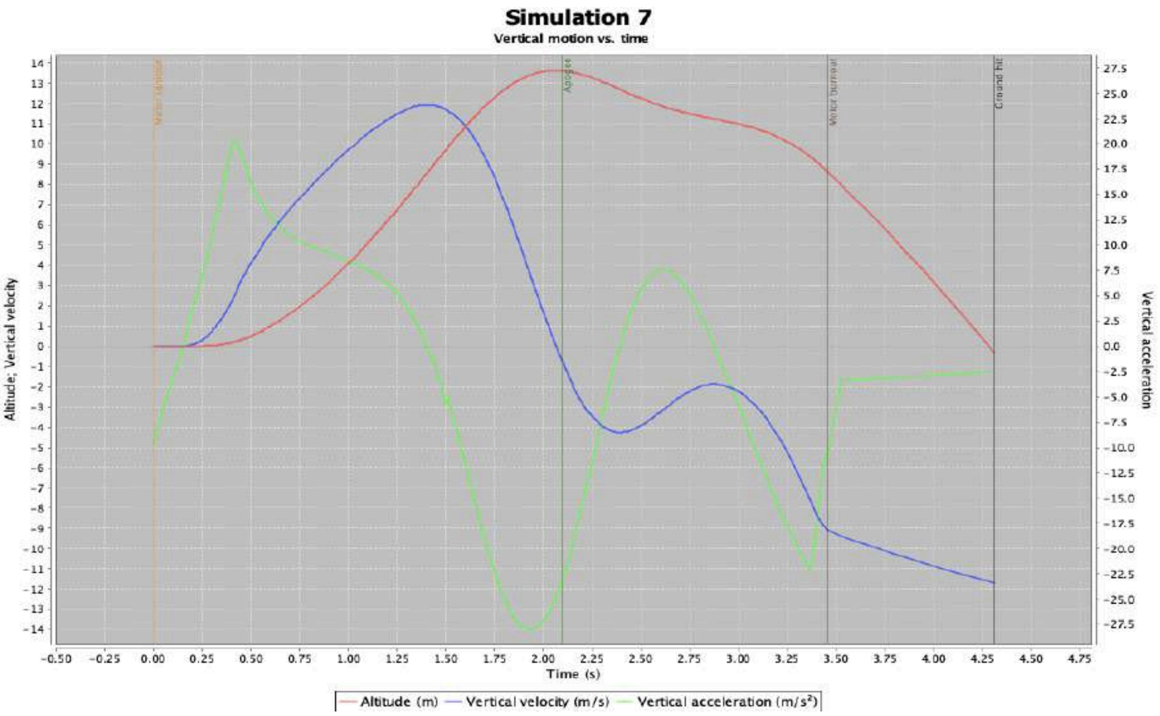


Figure 30: OpenRocket: Simulation

6 Dynamic Flight Analysis

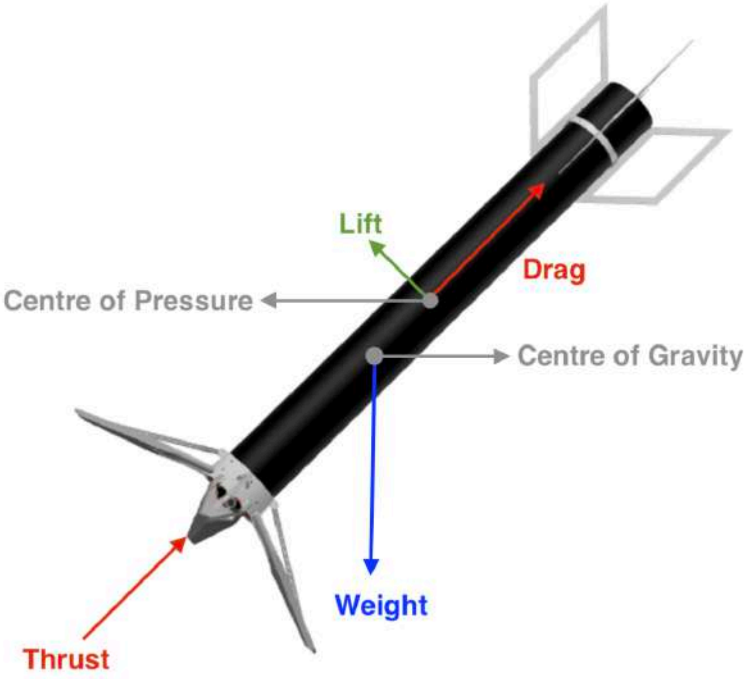


Figure 31: Forces on Rocket

6.1 Stability Characteristics

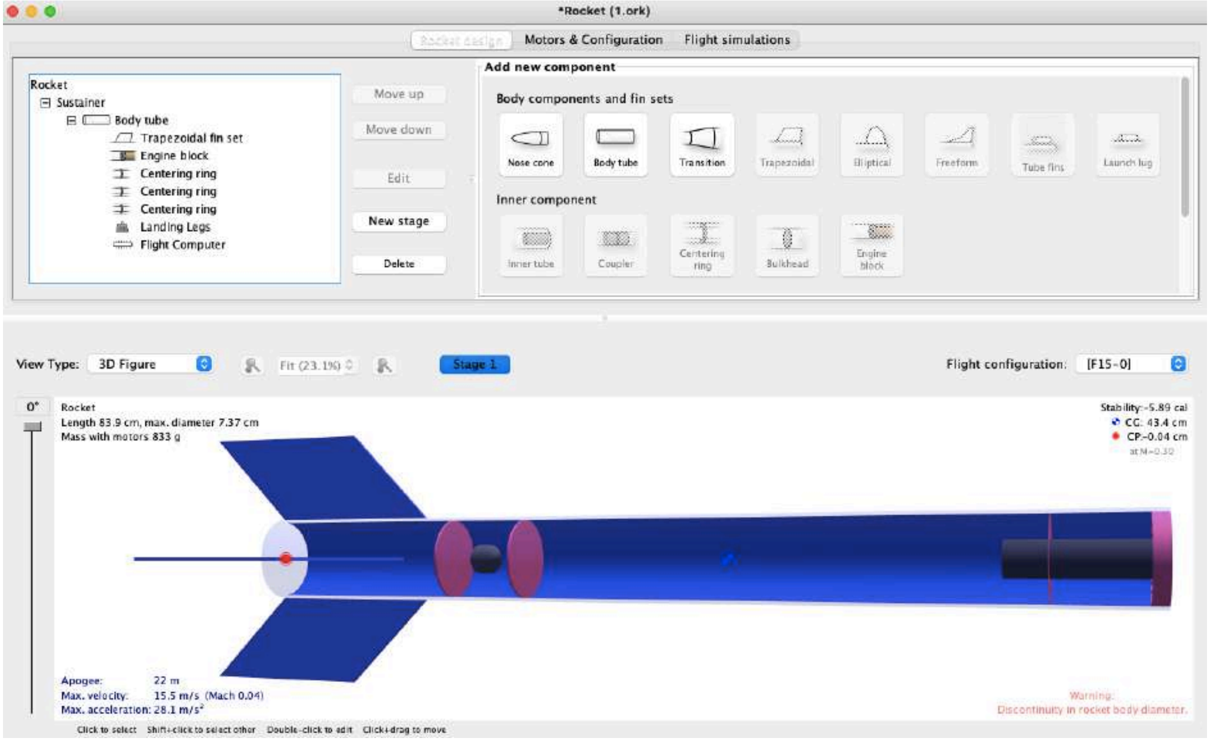
The stability characteristics of the rocket is determined mainly by the location of centre of gravity and centre of pressure. The location of the centre of gravity of the rocket is governed by the weight and the arm length of each and every component that the rocket houses. The location of the centre of pressure of the rocket is dictated by the each component's reference area and the location the component from a reference line, here the reference line is assumed to be the end of thrust vector control system. The location of the centre of gravity of the rocket is evaluated by both analytical and through the help of CAD modelling using Autodesk Fusion 360 software. The location of the centre of pressure of the rocket is estimated by both analytical and through the help of an open source software known an OpenRocket. Since we are dealing with model rockets, the team decided to use approximated equation for determining the centre of pressure for analytical calculation which considered landing legs whilst with the software the landing legs area could not accurately represented due limitations of the software. The table below provides a detailed breakdown of individual component weight and its respective arm length. The equations used for determining the centre of gravity and centre of pressure are mentioned below [6] [7].

$$cgW = d_n w_n + d_r w_r + d_b w_b + d_e w_e + d_f w_f \tag{1}$$

$$cpA = d_n a_n + d_b a_b + d_f a_f \tag{2}$$

Component	Weight (in grams)	Arm Length (in metres)
Airframe	370	0.381
Fins	55.6	0.735
Landing Legs	214.5	0.115
Propulsion	50	0.038
Thrust Vector Control	52.5	0.038
Flight Computer	100	0.723
X_CG	**842.6**	**0.297**

Table 20: Moment Table



*Rocket (1.ork)

Rocket design | Motors & Configuration | Flight simulations

Rocket
- Sustainer
  - Body tube
    - Trapezoidal fin set
    - Engine block
    - Centering ring
    - Centering ring
    - Centering ring
    - Landing Legs
    - Flight Computer

Move up | Move down | Edit | New stage | Delete

Add new component

Body components and fin sets: Nose cone | Body tube | Transition | Trapezoidal | Elliptical | Freeform | Tube fins | Launch lug

Inner component: Inner tube | Coupler | Centering ring | Bulkhead | Engine block

View Type: 3D Figure | Fit (23.1%) | Stage 1 | Flight configuration: [F15-0]

0°

Rocket
Length 83.9 cm, max. diameter 7.37 cm
Mass with motors 833 g

Stability: -5.89 cal
CG: 43.4 cm
CP: -0.04 cm
at M=0.30

Apogee: 22 m
Max. velocity: 15.5 m/s (Mach 0.04)
Max. acceleration: 28.1 m/s²

Warning:
Discontinuity in rocket body diameter.

Click to select | Shift+click to select other | Double-click to edit | Click+drag to move

Figure 32: OpenRocket: Centre of Pressure

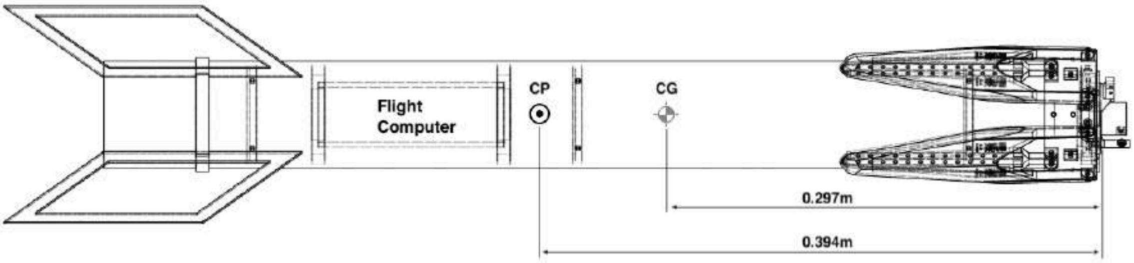


Figure 33: C.G and C.P Mark

6.2 Dynamic Analysis

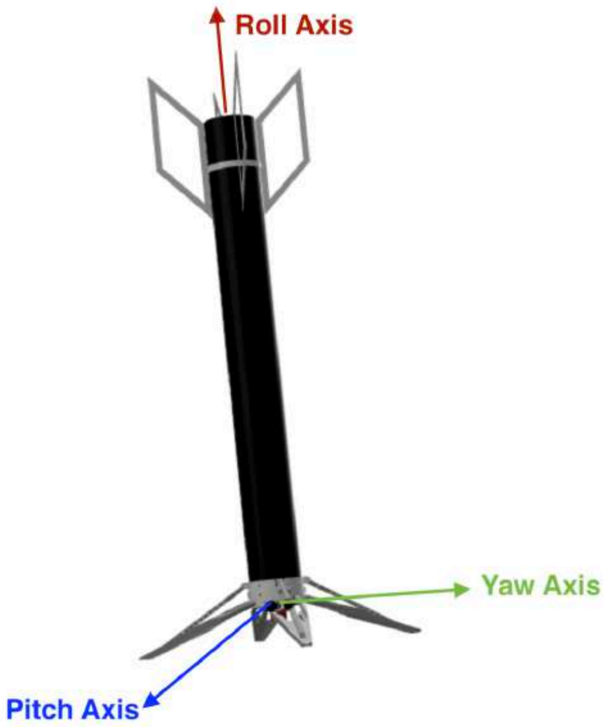


Figure 34: Rocket Axis Orientation

6.2.1 Corrective Moment Coefficient

Corrective moment coefficient is used to determine the corrective force that the rocket creates in response to a disturbance in flight. The larger the value, the more force the rocket creates and the quicker it will react to a disturbance. The corrective moment coefficient is used to determine how quickly and with how much force the rocket reacts to a disturbance in flight. A high corrective moment coefficient value means that the rocket will produce forces to counter the disturbing force, such as a gust of wind [1] [8]. The team's main objective being to land the rocket vertically, making sure that the rocket maintains vertical orientation at all times is salient. The value of corrective moment coefficient depends on the area of fin, stability margin and velocity. Hence, we decided to maximize the fin area and stability margin. The value for normal force coefficient is obtained from the open source software called OpenRocket. The equation used to determine the corrective moment coefficient is mentioned below [9].

$$C_1 = \frac{1}{2}\rho v^2 A_r C_{N\alpha}[CG - CP] \tag{3}$$

where,

$\mathrm{C_1}$ - Corrective Moment Coefficient

$\mathrm{A_r}$ - Reference area (maximum frontal area)

$\mathrm{C_{N\alpha}}$ - Normal Force Coefficient

6.2.2 Damping Moment Coefficient

The damping moment coefficient is what determines how far the rocket will be deflected, and how fast the oscillations die down to zero. The damping moment coefficient is made up of two components, one of which is aerodynamic in origin, and the other is propulsive. They are simply added together to get the total damping moment coefficient. The propulsive damping moment coefficient is the gases coming out of the nozzle seem to make the whole rocket harder to turn. This is similar to increasing the moment of inertia of the rocket. Each external component of the rocket such as fins, landing legs and airframe add to the overall damping of the rocket. And like the moment of inertia and corrective moment coefficient, the location of each of these parts in relation to the CG affects their values. But when compared to the aerodynamic damping moment coefficient the propulsive damping moment coefficient is relatively insignificant especially when it comes to scaled-down rockets. The equation used to determine the damping moment coefficient is mentioned below [1] [10].

$$\mathrm{C_2 = C_{2A} + C_{2R}} \tag{4}$$

where,

C_2 = Damping Moment Coefficient

A_{2A} = Aerodynamic Damping Moment Coefficient

C_{2R} = Propulsive Damping Moment Coefficient

$$\mathrm{C_{2R}} = \dot{\mathrm{m}}\,[\mathrm{L_{ne}} - \mathrm{CG}]^2 \tag{5}$$

where,

C_2 = Propulsive Damping Moment Coefficient

$\dot{m}$ = rate of mass expulsion from the nozzle, grams/second

L_{ne} = Distance the nozzle is from the tip of the nose cone

$$\mathrm{C_{2\ A}} = \frac{\rho \mathrm{VA_r}}{2} \sum \left(\mathrm{C_{N\alpha_{component}}}\ [\mathrm{CP_{component}}\ - \mathrm{CG}]^2 \right) \tag{6}$$

where,

C_{2A} = Aerodynamic Damping Moment Coefficient

A_r = Reference Area

$C_{N\alpha(component)}$ = Normal Force coefficient of the individual component

$CP_{component}$ = Distance from the nose tip to the CP of the component

6.2.3 Damping Ratio

The damping ratio is a dimensionless number, and it is independent of the speed of the rocket. The damping ratio is computed after we have found the longitudinal moment of inertia, corrective moment coefficient, and the damping moment coefficient. The system is found to be underdamped as desired. Since the damping ratio is less than i.e., underdamped the rocket oscillates back and forth and the frequency of the vibration dies out slowly. The parameters that affects the damping ratio are fin area, moving the fin away from centre of gravity, sweep aft fins, the rocket's weight and length. The equation used to determine the damping ratio is mentioned below [1] [11].

$$\zeta = \frac{C_2}{2\sqrt{C_1 I_L}} \tag{7}$$

where,

C_1 = Corrective Moment Coefficient

C_2 = Damping Moment Coefficient

I_L = Longitudinal Moment of Inertia

I_R = Radial Moment of Inertia

ω = Natural Frequency

ζ = Damping Ratio

Sl. No.	Parameters	Values
1	Centre of Gravity	0.297m
2	Centre of Pressure	0.394m
3	Static Margin	1.27
4	Normal Force Coefficient	10.3
5	Corrective Moment Coefficient	0.145
6	Damping Moment Coefficient	9.87 x 10^4
6	Damping Ratio	0.004
7	Natural Frequency	1.288 Hz
8	Longitudinal Moment of Inertia	0.09 kgm^2

Table 21: Flight Dynamics Parameters

6.3 Aerodynamic Drag Analysis

Aerodynamic modelling and analysis is imperative to understand the effects on the dynamics of the rocket. The team approached the aerodynamics analysis in an unconventional manner, instead of optimizing the design for reducing the drag force, we focused on increasing the drag force whilst keeping weight and other parameters in check. The figure below represents the different types of drag on the rocket including body drag, tail drag, fin drag and base drag. The pressure drag generated by the body is almost nonexistent, nevertheless the team evaluated its value. The total drag coefficient is the sum of base drag coefficient and pressure drag coefficient [12].

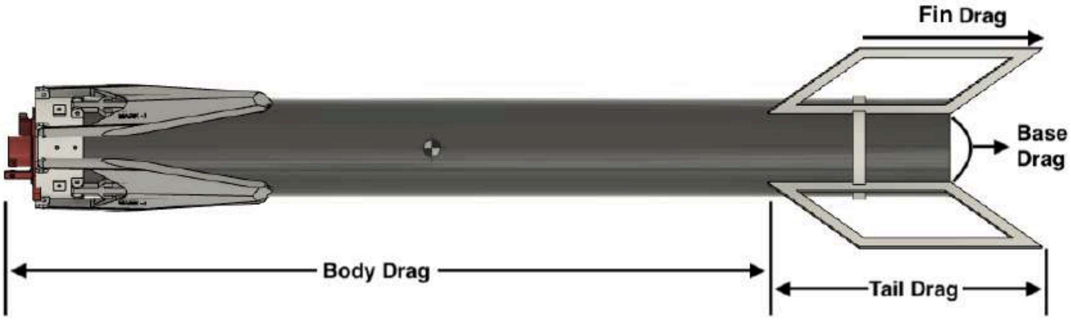


Figure 35: Aerodynamic Drag Representation

Figure 36 shows the variation of incompressible skin friction drag coefficient with respect to velocity. The value of incompressible skin friction drag coefficient is primarily dependent on the Reynolds number. The maximum value of incompressible skin friction drag coefficient is when velocity is zero, as shown in the graph. Figure 37 shows the variation in compressible skin friction drag coefficient with respect to velocity. The value of compressible skin friction drag coefficient is contingent on both incompressible skin friction drag coefficient and mach number and its value is maximum when velocity is zero, as shown in the graph.

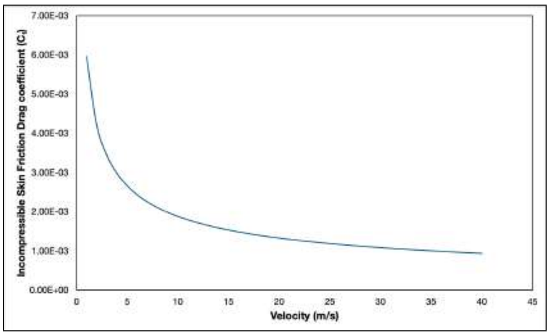


Figure 36: C_f Vs Velocity

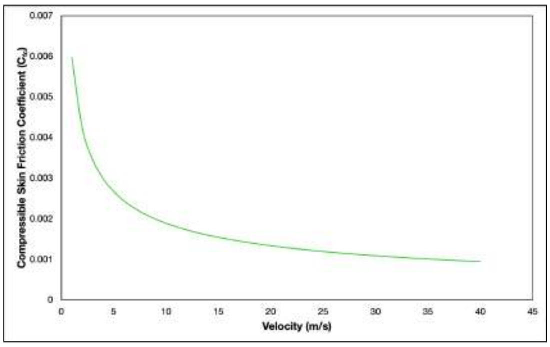


Figure 37: C_{f_c} Vs Velocity

Figure 38 shows the variation of tail leading edge drag coefficient with respect to velocity. The value of tail leading edge drag coefficient is primarily dependent on the reference area of the fin and the number of fins, and its maximum value is when velocity is maximum, as shown in the graph. Figure 39 shows the variation of tail trailing edge drag coefficient with respect to velocity. The value of tail trailing edge drag coefficient is primarily dependent on the reference area of the fin and the number of fins, and its maximum value is when velocity is maximum, as shown in the graph.

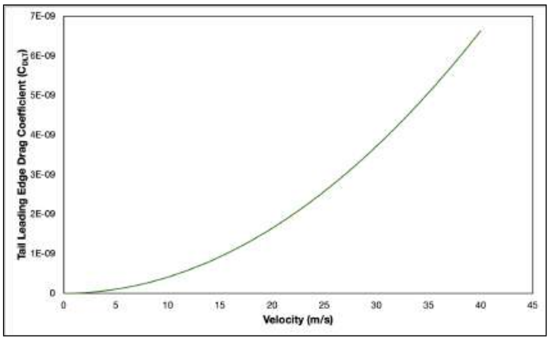


Figure 38: $C_{D_{L_T}}$ Vs Velocity

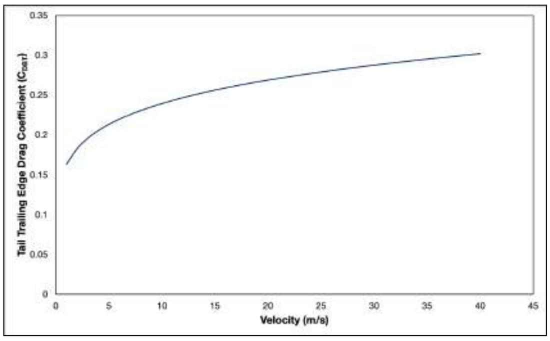


Figure 39: $C_{D_{B_T}}$ Vs Velocity

Figure 40 shows the variation of tail skin friction drag coefficient with respect to velocity. The value of tail skin friction drag coefficient is primarily dependent on the reference area of the fin and the number of fins, and its maximum value is when velocity is zero, as shown in the graph. Figure 41 shows the variation in tail thickness drag coefficient with respect to velocity. The value of tail thickness drag coefficient is primarily dependent on the reference area of the fin and the number of fins. In our passive fins design, the thickness is considerably low and the its contribution to overall drag is relatively less.

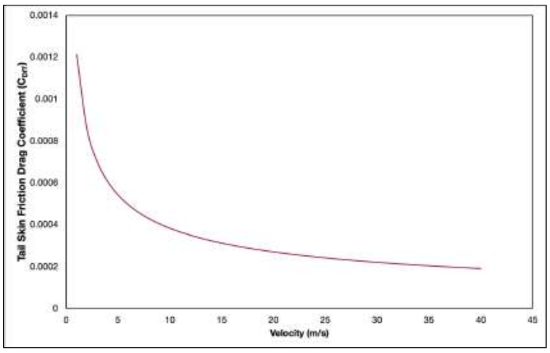


Figure 40: $C_{D_{f_T}}$ Vs Velocity

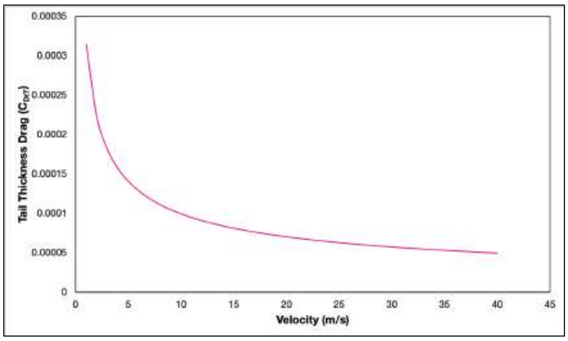


Figure 41: $C_{D_{t_T}}$ Vs Velocity

In Figure 42 the variation in tail drag coefficient with respect to velocity is observed. The value of tail drag coefficient is contingent on the above component drag i.e., leading edge, trailing edge and thickness of tail, and its value is maximum when velocity is maximum, as shown in the graph.

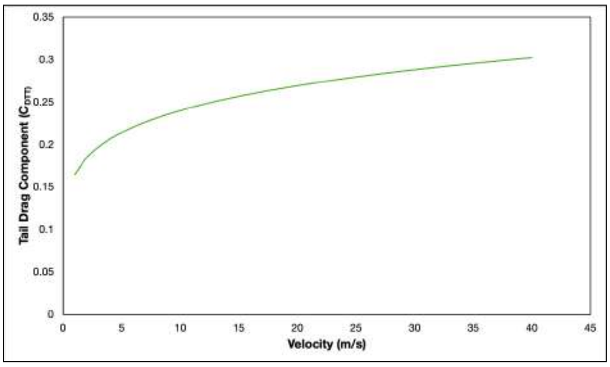


Figure 42: $C_{D_{T_T}}$ Vs Velocity

Figure 43 shows the variation of body pressure drag coefficient with respect to velocity. The value of body pressure drag coefficient is primarily dependent on the total wetted area and its maximum value is when velocity is zero, as shown in the graph. Figure 44 shows the variation in body base drag coefficient with respect to velocity. The value of body base drag coefficient is contingent on both incompressible skin friction drag coefficient and mach number and its value is maximum when velocity is maximum, as shown in the graph. The base drag component is significantly more than pressure drag component of the body.

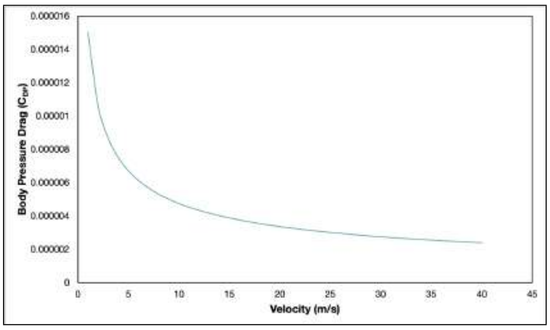


Figure 43: $C_{D_{B_P}}$ Vs Velocity

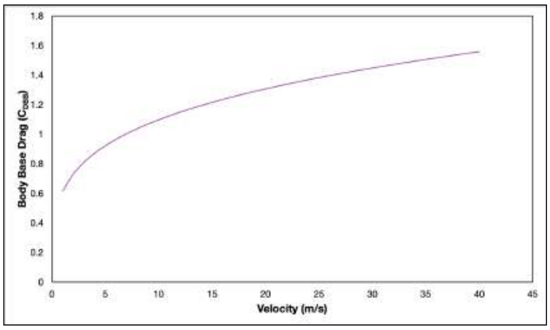


Figure 44: $C_{D_{B_B}}$ Vs Velocity

Figure 45 shows the variation of body incompressible skin friction drag coefficient with respect to velocity. The value of body incompressible skin friction drag coefficient is primarily dependent on the ratio of fin root chord to root fin trailing edge thickness and its maximum value is when velocity is zero, as shown in the graph. Figure 46 shows the variation in body component drag coefficient with respect to velocity. The value of body component drag coefficient is contingent on both the pressure drag and base drag of the body and its value is maximum when velocity is maximum, as shown in the graph.

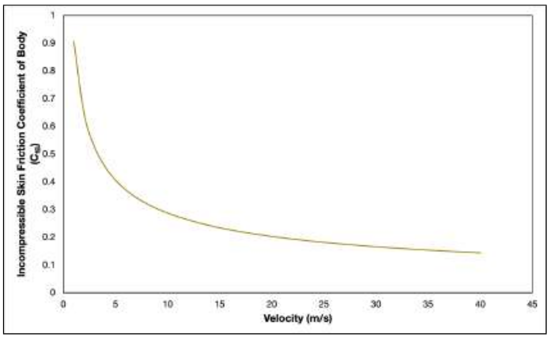


Figure 45: C_{f_B} Vs Velocity

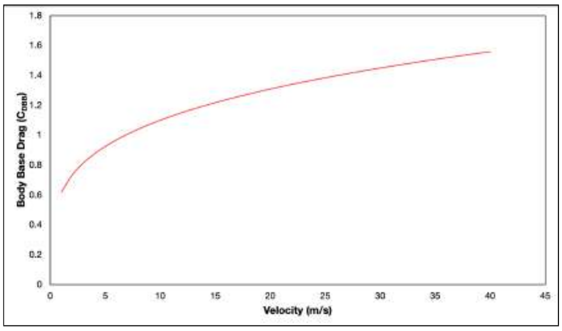


Figure 46: $C_{D_{T_B}}$ Vs Velocity

Figure 47 shows the variation of drag coefficient with respect to velocity. The value of drag coefficient is the sum of tail drag component and body drag component and its maximum value is when velocity is maximum i.e., 1.86, as shown in the graph. Figure 48 shows the variation in drag coefficient with respect to Reynolds number. The range of Reynolds number is 0 to 1.98 million during the descent phase of the rocket, the Reynolds number progressively increases and then starts to decrease once the propellant is ignited.

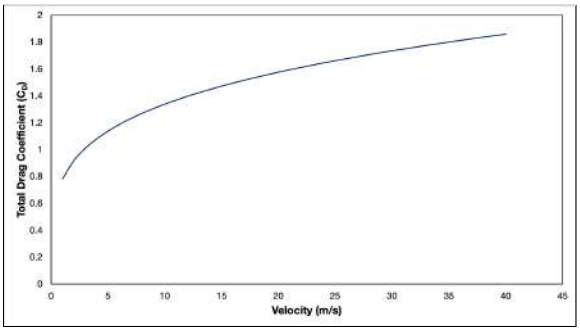


Figure 47: C_D Vs Velocity

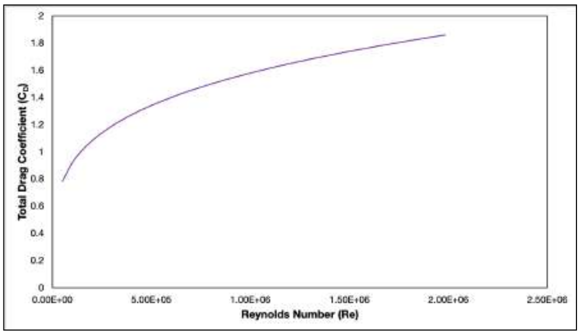


Figure 48: C_D Vs Reynolds Number

Figure 49 shows the variation of component drag coefficient with respect to velocity. It is evident from the graph below that the majority of drag is contributed from the body component of the rocket and followed by the tail component. Figure 50 shows a pie chart of the distribution of the component drag.

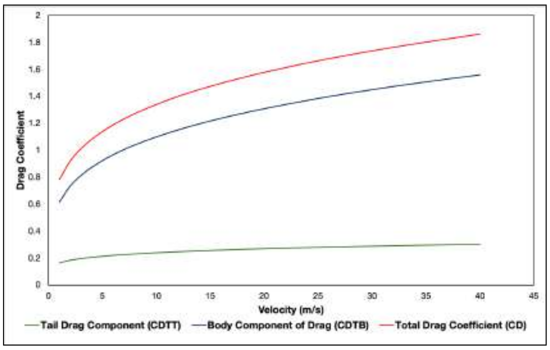


Figure 49: Component Drag Vs Velocity

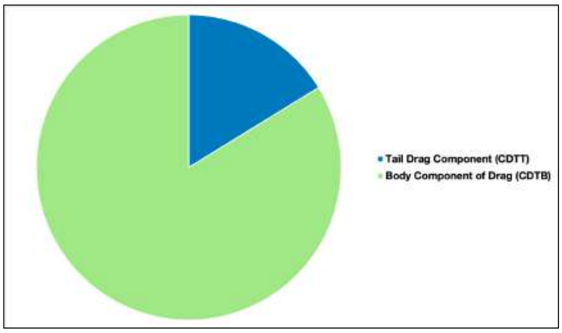


Figure 50: Component Drag Distribution

The table below is a list of all the user inputs that is used for analytical evaluation of the aerodynamics of the rocket body and the results from this drag analysis.

Sl. No.	Parameters	Values
1	Length	0.762m
2	Diameter	0.076m
3	Fin Span	0.076m
4	Fin Chord	0.152m
5	A_T	$0.011m^2$
6	A_r	$0.455m^2$
6	A_B	$0.011m^2$
7	A_{WB}	$0.191m^2$
8	A_W	$0.284m^2$
10	A_f	$0.023m^2$
11	f_B	10
12	c	$343ms^{-1}$
13	M	0.116
14	N	4
15	AR	0.011
16	ρ	$1.204kgm^3$
17	μ	$1.85e\text{-}05kgm^{-1}s^{-1}$
18	CG	0.297m
19	CP	0.394m
20	B	0.997
21	Γ_L	0.785
22	c_r	0.152
23	t_r	0.002
24	h_r	0.002
25	r_L	2.89e-05
26	K_1	11.69
27	K_p	1.001
28	K_2	4.391

Table 22: Input Parameters for Drag Analysis

Sl. No.	Parameters	Values at 40ms^{-1}
1	C_f	9.43e-04
2	C_{f_c}	9.41e-04
3	$C_{D_{L_T}}$	6.62e-09
4	$C_{D_{B_T}}$	0.301
5	$C_{D_{f_T}}$	1.91e-04
6	$C_{D_{t_T}}$	4.95e-05
6	$C_{D_{T_T}}$	0.302
7	$C_{D_{B_P}}$	2.39e-06
8	$C_{D_{B_B}}$	1.558
9	C_{f_B}	0.143
10	$C_{D_{T_B}}$	1.559
11	$\mathbf{C_D}$	**1.86**
12	Reynolds Number	1.98e+6

Table 23: Drag Analysis Results

6.4 Computational Fluid Dynamic Analysis

Computational fluid dynamic analysis was executed on a open source software, OpenFOAM (Open Field Operation and Manipulation). The team chose to use this software because of high degree of freedom, versatility and complete control over the source code including but not limited to governing equations, solver and several utilities. This software also uses finite volume method (FVM) instead of finite element method (FEM). A motorBike tutorial from the OpenFOAM website was used as reference [13]. BlockMesh is used for defining the domain around the airframe model. Since this is a 3D model, snappyHexMesh was used for meshing. The preprocessing part of this includes:

- blockMesh
- snappyHexMesh
- OpenFOAM Utilities

Some of the solvers are:

- Incompressible
- Compressible
- Multiphase
- laplacianFoam

- potentialFoam

Solvers for incompressible flow are:

- icoFOAM
- simpleFoam
- pisoFoam

The team performed a turbulent flow analysis to obtain the pressure field and velocity field of the rocket. To run this analysis, we used Reynolds-Averaged Navier Stokes (RANS) turbulent model instead of Large Eddy Simulation (LES) or Direct Numerical Simulation (DNS) as both of these were computationally expensive. We used the simpleFoam solver as our analysis includes turbulent flow study. Apart from this potentialFoam solver was used to approximate for irrotational flows. Finally the results are viewed on paraFoam.

The steps for analysis that are entered in the terminal are:

- cd $FOAM_RUN
- cd AirFrame
- blockMesh
- surfaceFeatures
- snappyHexMesh -overwrite
- potentialFoam
- simpleFoam
- paraFoam

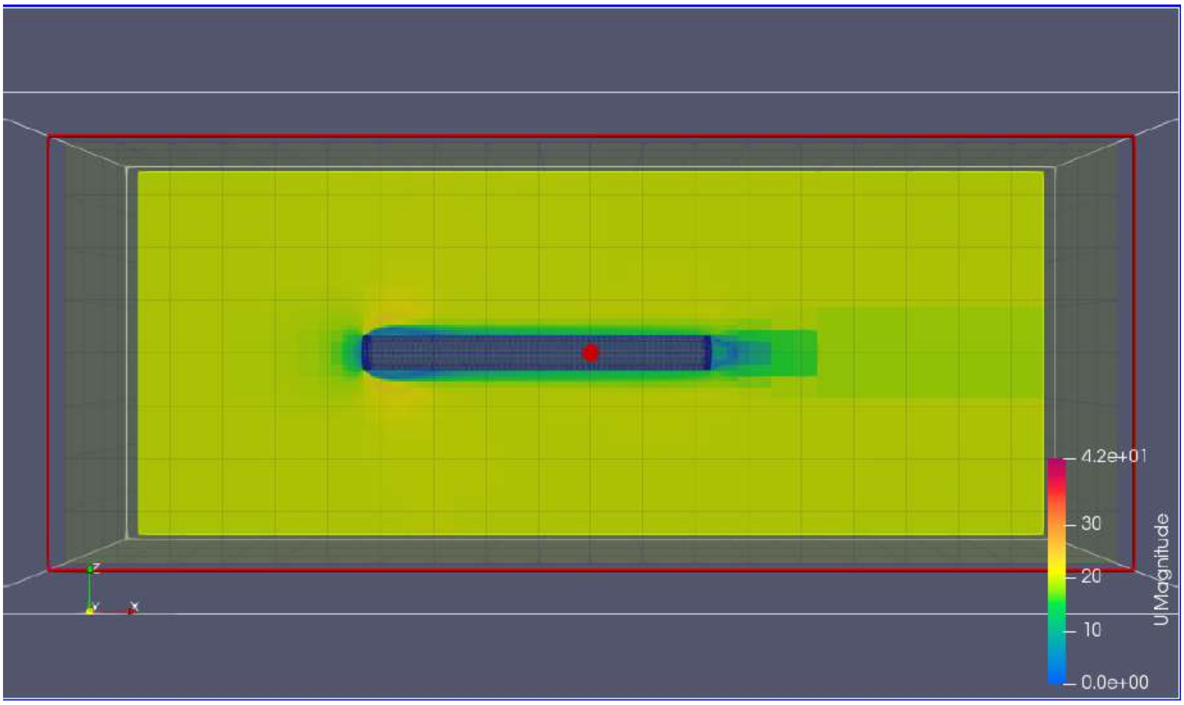


Figure 51: Velocity Field

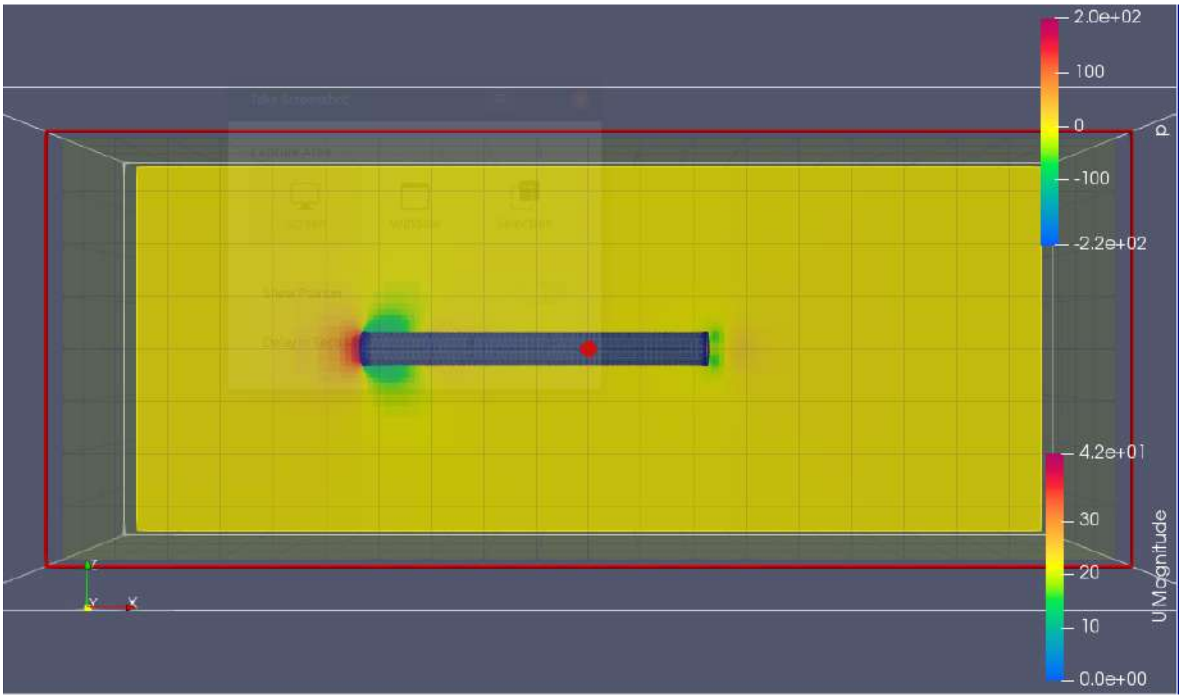


Figure 52: Pressure Field

7 Thrust Vector Control

7.1 Thrust Vectoring

Thrust Vectoring or Thrust Vector Control (TVC) can be defined as the ability of vehicles such as rockets and aircrafts to manipulate the direction of thrust from its motor or engine to control attitude or angular velocity. This can be seen in rockets and ballistic missiles which achieve thrust vectoring through four basic means:

- Gimbaled Engine(s) or Nozzle(s)
- Propellant Injection
- Vernier Thrusters
- Exhaust Vanes

7.1.1 Gimbaled Thrust Vectoring

Gimbaled Thrust Vectoring allows for movement in two directions or axes, namely the pitch and yaw axes, thus ensuring adequate attitude and angular velocity control. Figure 53 shows the working of this concept. The angle between the rocket center line and the thrust line is called gimbal angle. In the first case, the thrust line and center of gravity (CG) are aligned and we arrive at a straight line or normal configuration, with the gimbal angle being zero. In the second case, the nozzle is titled towards the left, making a gimbal angle *a* with respect to the rocket center line. As a result of this, a torque is generated about the CG which causes the rocket nose to tilt towards the left. Lastly, when the rocket nozzle moves right, the rocket nose tilts right due to the generation of torque about the CG.

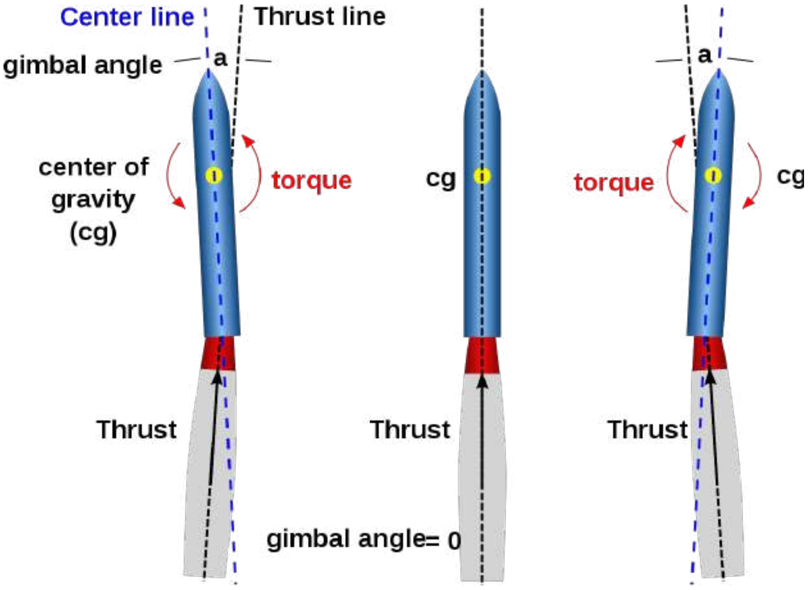


Figure 53: Gimbaled TVC
[14]

7.2 Thrust Vector Control Design

The main objective of the TVC for the model rocket is to achieve accurate movement in the pitch and yaw axes. After selecting the motor and airframe diameters which are 22mm and 76.2mm respectively, two TVC designs were developed. These designs consist of three main parts namely:

- Motor Mount
- Inner Gimbal
- Outer Gimbal

To enable movement in the pitch and yaw directions, the designs incorporate the usage of servos. Both the designs are connected to SG92r servos to check for accuracy and response in movement.

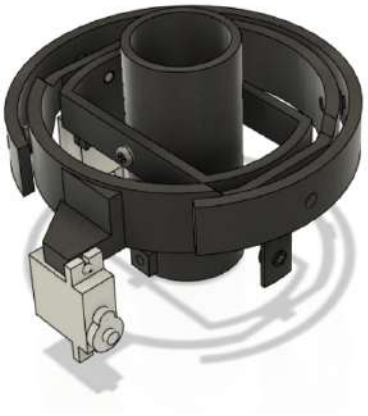


Figure 54: Design I

Design I [Figure 54] is a four part assembly as opposed to Design II [Figure 55] which is a three part assembly. The only difference between the two designs is that Design I has an outer holder which serves as an attachment to the rocket airframe.

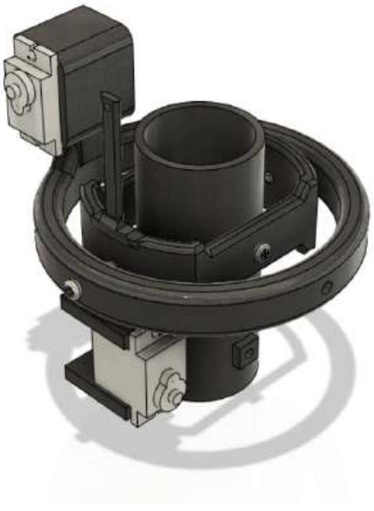


Figure 55: Design II

7.3 TVC Dimensions

7.3.1 Design I:

Sl. No.	Part	Inner Diameter (mm)	Outer Diameter (mm)
1	Motor Mount	22.5	28
2	Inner Gimbal	45	50
3	Outer Gimbal	54	60
4	Holder	67	73

Table 24: Design I Specifications

7.3.2 Design II:

Sl. No.	Part	Inner Diameter (mm)	Outer Diameter (mm)
1	Motor Mount	22.5	28
2	Inner Gimbal	45	52
3	Outer Gimbal	61	73

Table 25: Design II Specifications

Both designs make use of M3 screws. Platforms to accommodate the servos were designed after thorough measurement of the SG92r servos. The servo arms with the push rods are to be aligned with the linkage stoppers present in the motor mount and inner gimbal. This will allow smooth gimbal movement and the design is made accordingly.

8 Dynamics

Dynamics is a subdivision of mechanics dealing with the motion of materials or objects in time, in relation to the physical factors that affect this motion like force, momentum, mass, and energy. Dynamics is further subdivided into *kinematics*, where motion is considered (with respect to position, velocity and acceleration) without regard to its causes, and *kinetics* which deals with the effects of forces and torques on the motion of a moving body.

Before we delve into the dynamics behind the working of a model rocket, we will first introduce the concepts of scalar and vector products.

8.1 Scalar Products

Also known as dot product, it gives the length of projection of vector $\vec{a}$, onto another vector $\vec{b}$, multiplied by the length of $\vec{b}$, as shown in Figure 56.

$$\vec{a} \cdot \vec{b} = |\vec{a}| \cdot |\vec{b}| \cos\theta \tag{8}$$

Where, θ is the angle between the two vectors $\vec{a}$ and $\vec{b}$.

8.1.1 Inner Products Properties

$$\vec{a} \cdot \vec{b} = \vec{b} \cdot \vec{a} \tag{9}$$

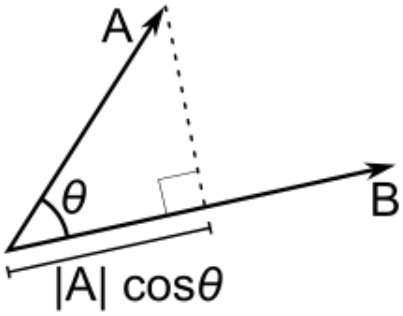


Figure 56: Inner Product

$$\vec{c}(\vec{a} + \vec{b}) = (\vec{c} \cdot \vec{a}) + (\vec{c} \cdot \vec{b}) \tag{10}$$

From equations 9 and 10, we can conclude that scalar products and both commutative and distributive respectively.

8.2 Vector Products

A cross product or vector product is a mathematical operation in which two vectors belonging to a three dimensional space yields a new vector in a direction perpendicular to both the vectors. This is illustrated in Figure 57.

$$\vec{a} \times \vec{b} = |\vec{a}||\vec{b}| \sin\theta \cdot \hat{n} \tag{11}$$

Here, the new vector formed as a result of the cross product between $\vec{a}$ and $\vec{b}$ is along the direction of $\hat{n}$.

$$\vec{a} = a_1\hat{i} + a_2\hat{j} + a_3\hat{k}$$
$$\vec{b} = b_1\hat{i} + b_2\hat{j} + b_3\hat{k}$$

$\hat{i}$, $\hat{j}$, $\hat{k}$, are unit vectors in the x, y,z directions respectively.

8.2.1 Cross Product Properties

Vector products are anti-commutative:

$$\vec{b} \times \vec{a} = -\vec{a} \times \vec{b} \tag{12}$$

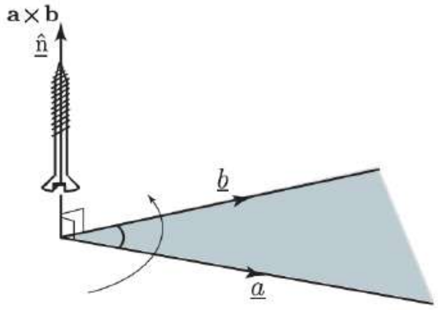


Figure 57: Vector Product
[15]

They are distributive over addition:

$$\vec{a} \times (\vec{b} + \vec{c}) = \vec{a} \times \vec{b} + \vec{a} \times \vec{c} \tag{13}$$

Two vectors in the same direction give a cross product of zero:

$$\vec{a} \times \vec{b} = \vec{0} \tag{14}$$

Additional properties:

$$\vec{a} \cdot (\vec{b} \times \vec{c}) = \vec{b}(\vec{c} \times \vec{a}) = \vec{c}(\vec{a} \times \vec{b}) \tag{15}$$

$$\vec{a} \times (\vec{b} \times \vec{c}) = \vec{b}(\vec{c} \cdot \vec{a}) - \vec{c}(\vec{a} \cdot \vec{b}) \tag{16}$$

8.3 Expressing Vectors In Different Frames

Vectors can be oriented differently when seen from different perspectives. Let us consider a reference frame (F_r), which consists of $\vec{P}$. This $\vec{P}$ can be expressed in terms of F_r by using $\vec{a}$, $\vec{b}$ and $\vec{c}$ along the $\hat{x}$, $\hat{y}$, $\hat{z}$ directions respectively. $\vec{a}$, $\vec{b}$ and $\vec{c}$ have magnitudes of α, β and γ respectively.

Notation: $\vec{P}^r$, when $\vec{P}$ is expressed in F_r.

$$\vec{P}^r = \vec{a}^r + \vec{b}^r + \vec{c}^r \tag{17}$$

$$\vec{a}^r = \begin{bmatrix} \alpha \\ 0 \\ 0 \end{bmatrix}$$

$$\vec{b}^r = \begin{bmatrix} 0 \\ \beta \\ 0 \end{bmatrix}$$

$$\vec{c}^r = \begin{bmatrix} 0 \\ 0 \\ \gamma \end{bmatrix}$$

Therefore,

$$\vec{P}^r = \begin{bmatrix} \alpha \\ \beta \\ \gamma \end{bmatrix} \tag{18}$$

Let F_r be rotated through an angle of ψ along the z axis, such that a new frame (F_1) is produced as shown in the figure below. The objective now is to express $\vec{P}$ in F_1.

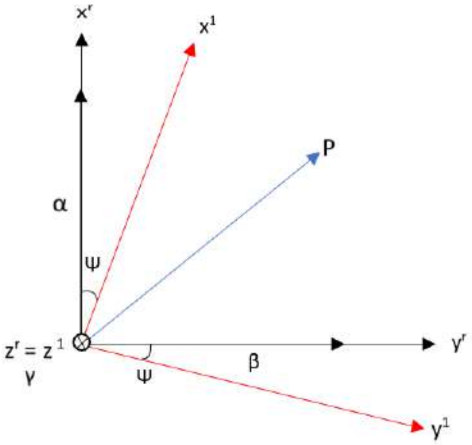


Figure 58: $\vec{P}$ being expressed in F_r and F_1

Notation: $\vec{P}^1$, where 1 is used to denote F_1.

$$\vec{P}^1 = \vec{a}^1 + \vec{b}^1 + \vec{c}^1 \tag{19}$$

$$\vec{a}_1 = \begin{bmatrix} \alpha \cos\psi \\ -\alpha \sin\psi \\ 0 \end{bmatrix}$$

$$\vec{b}^1 = \begin{bmatrix} \beta \sin\psi \\ \beta \cos\psi \\ 0 \end{bmatrix}$$

$$\vec{c}^r = \begin{bmatrix} 0 \\ 0 \\ \gamma \end{bmatrix}$$

Therefore,

$$\vec{P}^1 = \begin{bmatrix} \cos\psi & \sin\psi & 0 \\ -\sin\psi & \cos\psi & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \alpha \\ \beta \\ \gamma \end{bmatrix}$$

From equation 18,

$$\vec{P}^1 = \begin{bmatrix} \cos\psi & \sin\psi & 0 \\ -\sin\psi & \cos\psi & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \vec{P}^r \end{bmatrix} \tag{20}$$

$$\vec{P}^1 = C_{1/r}.\vec{P}^r \tag{21}$$

$C_{1/r}$ is the rotation/representation of $\vec{P}$ from F_r to F_1, which is called Direction Cosine Matrix (DCM).

8.3.1 Properties of Direction Cosine Matrix

- Orthogonal:

$$C^T_{1/r} = C^{-1}_{1/r} = C_{r/1} \tag{22}$$

- Eigenvalues of a DCM is always 1 (λ = 1). This is because there is only rotation of the vector and no scaling (stretching or shrinking).

8.4 Rodrigues' Rotation Formula

Let us consider vector v which is rotated about a unit vector r by some angle α. Here, α is a left handed rotation. The new vector formed is $\bar{v}$. This can be seen in the figure below. The main goal is to relate $\bar{v}$ as a function of v, r, α.

$$\bar{v} = (1 - \cos\alpha)\langle v \times r\rangle r + \cos\alpha(v) - \sin\alpha(r \times v) \tag{23}$$

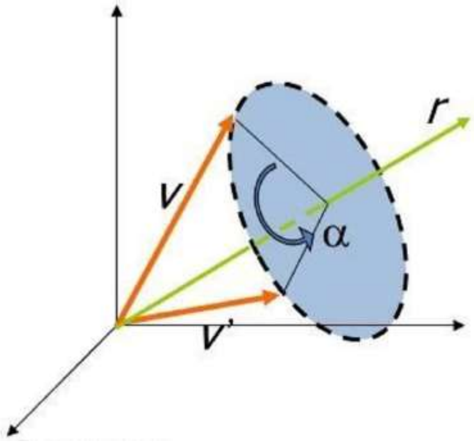


Figure 59: Rodrigues' Rotation

8.5 Euler Angles

Euler angles were developed first by the Swiss mathematician ***Leaonhard Euler***, which describes the orientation of a particular rigid body with that of a fixed coordinate system, which is also commonly known as the earth or inertial reference frame. Euler angles are used intensively in the domains of aeronautics and engineering.

Euler angles make use of three angles which can be used to completely define the orientation of a body frame with respect to an inertial frame. Euler angles are not unique i.e. there can be more than one sequence of Euler angles that can be used to define an orientation. The objective is to arrive from a vehicle carried inertial frame also called North-East-Down (NED) frame to the vehicle carried body frame.

F_{NED}: North East Down frame or Inertial frame.

F_b: Vehicle carried body frame.

To achieve this, an Euler rotational sequence is carried out in the following specific order:

- Right hand rotation about the z_v axis (also the z_1 axis) through an angle ψ, Yaw Angle. (Rotation from F_{NED} or F_v to F_1 shown in Figure 60).

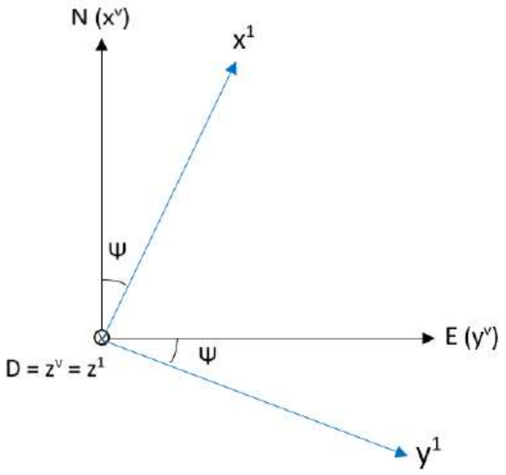


Figure 60: F_v to F_1

- Right hand rotation about the y_1 axis (also the y_2 axis) through an angle θ, Pitch Angle. (Rotation from F_1 to F_2 shown in Figure 61)
- Right hand rotation about the x_2 axis (also the x_b axis) through an angle ϕ, Roll Angle. (Rotation from F_2 to F_b shown in Figure 62)

In terms of rotational matrices:

Let us consider two frames namely, F_{NED} / F_v and F_b. Assume that there exists a vector $\vec{u}$ in F_v (which will be denoted as $\vec{u}^v$). We will now try to express this vector in the NED frame to the body frame.

This can be performed in the following three steps:

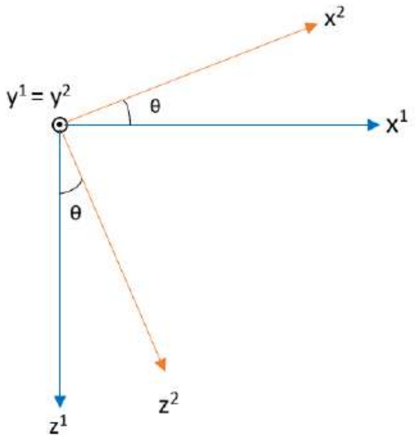


Figure 61: F_1 to F_2

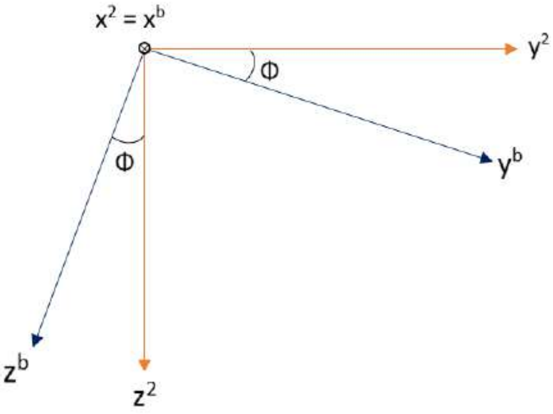


Figure 62: F_2 to F_b

- Step 1

$$\vec{u}^1 = c_{1/v}(\psi)\vec{u}^v \tag{24}$$

$$C_{1/v} = \begin{bmatrix} \cos\psi & \sin\psi & 0 \\ -\sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

This shows the rotation of F_{NED} or F_v about the Z-axis through an angle of ψ, resulting in a new frame F_1.

- Step 2

$$\vec{u}^2 = c_{2/1}(\theta)\vec{u}^1 \tag{25}$$

$$C_{2/1} = \begin{bmatrix} \cos\theta & 0 & -\sin\theta \\ 0 & 1 & 0 \\ \sin\theta & 0 & \cos\theta \end{bmatrix}$$

This allows the rotation of F_1 about the Y-axis through an angle of θ, resulting in a new frame F_2.

- Step 3

$$\vec{u}^b = c_{b/2}(\phi)\vec{u}^2 \tag{26}$$

$$C_{1/v} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & \sin\phi \\ 0 & -\sin\phi & \cos\phi \end{bmatrix}$$

This allows the rotation of F_2 about the X-axis through an angle of ϕ, hence arriving at the desired frame F_b.

Adding equations (24 + 25 + 26), we arrive at:

$$\vec{u}^b = C_{b/2}(\phi) \cdot c_{2/1}(\theta) \cdot c_{1/v}(\psi) \cdot \vec{u}^v$$

$$\vec{u}^b = C_{b/v}(\phi, \theta, \psi) \cdot \vec{u}^v \tag{27}$$

Where,

$$C_{b/v}(\phi, \theta, \psi) = DCM = \begin{bmatrix} c\theta \cdot c\psi & c\theta \cdot s\psi & -s\theta \\ c\psi \cdot s\theta - c\phi \cdot s\psi & c\phi \cdot c\psi + s\theta \cdot s\phi \cdot s\psi & c\theta \cdot s\phi \\ c\phi \cdot c\psi \cdot s\theta + s\phi \cdot s\psi & -c\psi \cdot s\phi + c\phi \cdot s\theta \cdot s\psi & c\theta \cdot c\phi \end{bmatrix} \tag{28}$$

where c - Cosine, s - Sine.

NOTE:

- Order of this rotation is important, starting with the rotation about Z axis then Y axis and then the X axis.
- The typical reference order or data order while mentioning Euler angles is along the lines of ϕ, θ, ψ (X - Y - Z).
- $C_{b/v}(\phi, \theta, \psi)$ is a Direction Cosine Matrix (DCM), and is not unique.
- $C_{b/v}(\phi, \theta, \psi)$ when analysed for Eigenvalues, always has one row having unity as its element, implying that only rotation is taking place on the considered vector $\vec{u}$ (No stretching or shrinking).

Important: Recalling the concepts of Rodrigues' Rotation and equation 23 and assuming that $v = r$, we shall arrive at $\bar{v} = v$. This implies that any vector which is aligned with its axis of rotation does not change and will possess an eigenvalue of unity.

From the eigenvector equation,

$$A \cdot \vec{x} = \lambda \cdot \vec{x} \tag{29}$$

Where, $\vec{x}$ and λ is the eigenvector and eigenvalue respectively.

This has an implication on Euler angles. When the DCM is multiplied with its eigenvector, it equals the same eigenvector (as $\lambda = 1$). This goes to say that when the eigenvector is rotated through the DCM, it remains the same, making it the axis of rotation.

Therefore, $\vec{x}$ is the axis of rotation for the entire Euler sequence. So, instead of using three separate rotations to arrive at $\vec{u}^b$, we can arrive at $\vec{u}^b$ by a single rotation about $\vec{x}$. Here, $\vec{x}$ is the eigenvector (of $C_{b/v}(\phi, \theta, \psi)$) corresponding to unity eigenvalue.

8.6 Euler Kinematical Equations

A *rate gyro* is used to measure the rate of angular movement which indicates the change of an angle with time, instead of indicating direction. They are used in Inertial Measurement Units (IMUs) to measure angular change.

Let us consider the body frame of the rocket to rotate about an axis which can be called the angular velocity vector (ω). This can be expressed as $\vec{\omega_{b/v}}$.

$$\vec{\omega}_{b/v} = \begin{bmatrix} p \\ q \\ r \end{bmatrix} = \begin{bmatrix} \text{Roll rate} \\ \text{Pitch rate} \\ \text{Yaw rate} \end{bmatrix} \tag{30}$$

These inputs from the rate gyros are required to determine the Euler angles. A specific function which relates the data from the IMU has to be converted into the three Euler angles.

NOTE:

- The rates of the Euler angles $\begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix}$ are not equal to $\begin{bmatrix} p \\ q \\ r \end{bmatrix}$

$$\begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} \neq \begin{bmatrix} p \\ q \\ v \end{bmatrix} \tag{31}$$

Therefore, a function which maps the inputs from the IMU to the three Euler angles has to be specified. This entire system that takes in inputs from the IMU and maps it to the three Euler angles (this sytem also includes an integrator) is called an *Attitude Heading Reference System (AHRS)*.

8.6.1 Properties of Angular Velocity Vector ($\vec{\omega_{b/v}}$):

- $\vec{\omega_{b/v}}$ is s unique vector that relates the derivatives of a vector taken in two frames.

- Additive in nature:

$$\vec{\omega}_{c/a} = \vec{\omega}_{a/b} + \vec{\omega}_{b/c} \tag{32}$$

From equation 32, we can write $\vec{\omega}_{b/v}$ as:

$$\vec{\omega}^b_{b/v} = \vec{\omega}^b_{b/2} + \vec{\omega}^2_{2/1} + \vec{\omega}^1_{1/v} \tag{33}$$

Expressing the above equation with respect to the body frame F_b:

$$\vec{\omega}^b_{b/v} = \vec{\omega}^b_{b/2} + c_{b/2}(\phi) \cdot \vec{\omega}^2_{2/1} + C_{b/2}(\phi) \cdot C_{2/1}(\theta)\vec{\omega}^1_{1/v} \tag{34}$$

Where,

$$\vec{\omega}^1_{1/v} = \begin{bmatrix} 0 \\ 0 \\ \dot{\psi} \end{bmatrix}$$

$$\vec{\omega}^2_{2/1} = \begin{bmatrix} 0 \\ \dot{\theta} \\ 0 \end{bmatrix}$$

$$\vec{\omega}^b_{b/2} = \begin{bmatrix} \dot{\phi} \\ 0 \\ 0 \end{bmatrix}$$

$$\vec{\omega}^b_{b/v} = \begin{bmatrix} 1 & 0 & -\sin\theta \\ 0 & \cos\phi & \sin\phi \cdot \cos\theta \\ 0 & -\sin\phi & \cos\theta \cdot \cos\theta \end{bmatrix} \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} \tag{35}$$

Let,

$$\begin{bmatrix} 1 & 0 & -\sin\theta \\ 0 & \cos\phi & \sin\phi \cdot \cos\theta \\ 0 & -\sin\phi & \cos\phi \cdot \cos\theta \end{bmatrix} = P$$

Therefore, $\begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} = P^{-1} \cdot \begin{bmatrix} p \\ q \\ r \end{bmatrix}$

Let $P^{-1} = H(\theta, \phi)$

Hence,

$$\dot{\Phi} = H(\theta, \phi) \cdot \vec{\omega}_{b/2} \tag{36}$$

$$\begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} 1 & s\phi t\theta & c\phi t\theta \\ 0 & c\phi & -s\phi \\ 0 & \sec\theta \cdot s\phi & \sec\theta \cdot c\phi \end{bmatrix} \begin{bmatrix} p \\ q \\ r \end{bmatrix} \tag{37}$$

where c - Cosine, s - Sine, t - Tangent.

From equation 36, we can now take in the outputs from the IMU and map it to Euler angles through the function $H(\theta, \phi)$.

Equation 37, which transforms the output of the rate gyros to the desired Euler angles is called the ***Euler Kinematical Equations***.

8.7 Quaternions

Quaternions were first introduced by an Irish mathematician ***William Rowan Hamilton***. It is a more robust form of attitude representation, which eliminates the singularity or gimbal lock posed by Euler angles. A quaternion is a (4x1) vector, which can describe complex rotations.

This is shown in Figure 63.

$$q = \begin{bmatrix} Scalar \\ \bar{v}(Vector) \end{bmatrix} = \begin{bmatrix} q_s \\ q_x \\ q_y \\ q_z \end{bmatrix} = \begin{bmatrix} q_1 \\ q_2 \\ q_3 \\ q_4 \end{bmatrix} \tag{38}$$

$$q = \begin{bmatrix} \cos\theta/2 \\ \bar{e} \cdot \sin\theta/2 \end{bmatrix} \tag{39}$$

Where,

θ = rotation angle or transformational angle about a unique arbitrary axis.

$\bar{e}$ = normalized axis of rotation (3x1)

NOTE:

- q here denotes quaternion, not the pitch rate.
- θ here denotes the transformation angle, not the pitch angle.

8.7.1 Quaternion Mathematics:

- Norm:

$$|q| = \sqrt{q_s^2 + q_x^2 + q_y^2 + q_z^2}$$

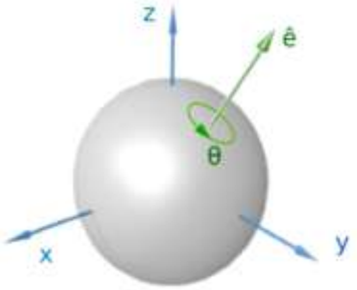


Figure 63: Quaternions

- Normalized quaternion:

$$\text{Normalization } = \frac{q}{|q|}$$

- Conjugate:

$$q* = \begin{bmatrix} q_s \\ -q_x \\ -q_y \\ -q_z \end{bmatrix}$$

- Inverse:

$$q^{-1} = \frac{q*}{|q|}$$

8.8 Attitude Representation using Quaternions

Objective:

It is important to be able to convert Euler angles to quaternions and vice versa. The reason is because it is arduous to visualize quaternions as opposed to Euler angles.

$$\boxed{\text{Euler Angles - DCM}(\phi,\theta,\psi)} \rightleftharpoons \boxed{\text{Quaternions}}$$

8.8.1 DCM to Quaternions

- $\xrightarrow[C_{b/v}\text{ (3x3)}]{\text{DCM}} \boxed{\text{MATLAB - dcm2quat}} \xrightarrow[\text{q (4x1)}]{\text{Quaternion}}$

8.8.2 Quaternions to DCM

- $\xrightarrow[\text{q (4x1)}]{\text{Quaternion}} \boxed{\text{MATLAB - quat2dcm}} \xrightarrow[C_{b/v}\text{ (3x3)}]{\text{DCM}}$

It is possible to obtain Euler angles from the DCM after conversion from quaternions.

8.9 Transport Theorem

In 2D, if frame P is rotating with respect to frame O at a rate $\dot{\theta}$, then we say that the angular velocity of P with respect to O is $\omega_{P/O} = \dot{\theta}k$. An illustration is shown in figure below. The following equation, known as the 'Transport Theorem', is a formula for the derivative of any vector r with respect to frame of reference O.

$$\left(\frac{d\mathbf{r}}{dt}\right)_{/O} = \left(\frac{d\mathbf{r}}{dt}\right)_{/P} + \boldsymbol{\omega}_{P/O} \times r \tag{40}$$

Transport theorem is used to switch between frames, for instance switch between inertial and body frames. Transport theorem derivation is shown in appendix 14.7.

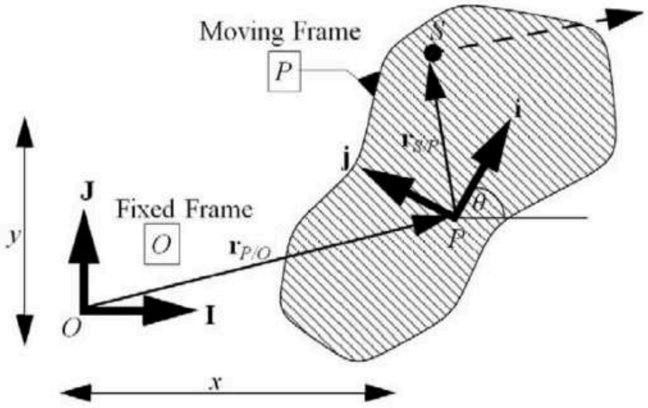


Figure 64: Rigid body hurtling through space
[16]

8.10 Euler Equations

This consists of first order ODE's describing the rotation of a rigid body using a non-inertial/rotating frame of reference with axes fixed to the body and parallel to body's principle axes of inertia. This results in a total of 6DOF (3 translational and 3 rotational). The linear velocity, angular velocity, force and torque about the body frame are shown in equations 41, 42, 43, 44 respectively.

$$\text{Linear Velocity } v_b = \begin{bmatrix} u \\ v \\ w \end{bmatrix} \tag{41}$$

Where,
u, v, w are the components of the linear velocity vector in the x, y, z directions respectively.

$$\text{Angular Velocity } \omega = \begin{bmatrix} p \\ q \\ r \end{bmatrix} \tag{42}$$

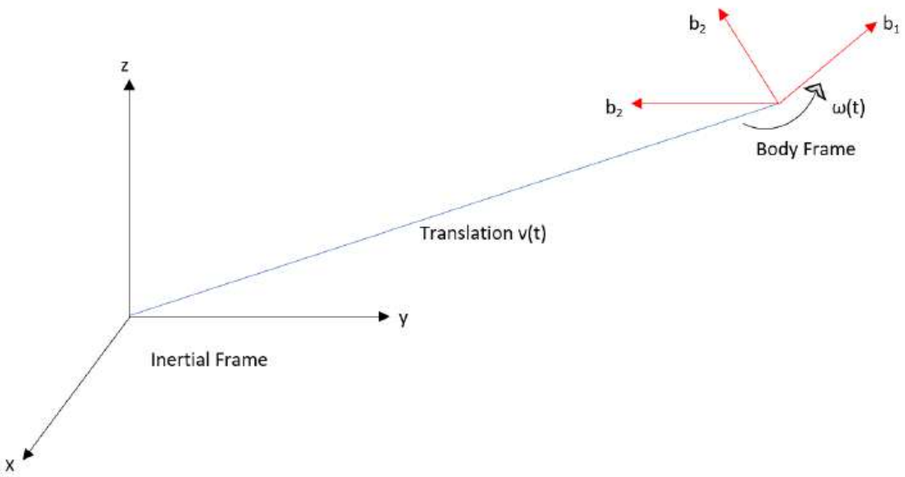


Figure 65: Rotation and Translation

$$\text{Force } F = \begin{bmatrix} F_x \\ F_y \\ F_z \end{bmatrix} \tag{43}$$

$$\text{Torque } T = \begin{bmatrix} L \\ M \\ N \end{bmatrix} \tag{44}$$

Where,

L, M, N are the components of torque in the x, y, z directions respectively.

$$\text{Navigation Coordinates } = \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} \text{Longitudinal Position} \\ \text{Lateral Position} \\ \text{Height} \end{bmatrix} \tag{45}$$

$$v^b = \begin{bmatrix} u \\ v \\ w \end{bmatrix}^b = u\hat{b}_1 + v\hat{b}_2 + w\hat{b}_3 \tag{46}$$

$$\left(\frac{dv}{dt}\right)_{inert} = \frac{du}{dt} \cdot \hat{b}_1 + \frac{dv}{dt}\hat{b}_2 + \frac{dw}{dt} \cdot \hat{b}_3 + u \cdot \frac{d\hat{b}_1}{dt} + v \cdot \frac{d\hat{b}_2}{dt} + w \cdot \frac{d\hat{b}_3}{dt} \tag{47}$$

First three terms in the RHS of equation 47 deal with the change in velocity with respect to body frame and the next three terms deal with rotation of body coordinate frame with respect to inertial frame.

We arrive at,

$$\dot{v}_{inert} = \dot{v}_b + (\omega_b \times v_b) \tag{48}$$

$$\dot{v}_{inert} = \begin{bmatrix} \dot{u} \\ \dot{v} \\ \dot{w} \end{bmatrix} + \begin{bmatrix} 0 & -r & q \\ r & 0 & -p \\ -q & p & 0 \end{bmatrix} \begin{bmatrix} u \\ v \\ w \end{bmatrix} \tag{49}$$

$$\dot{v}_{inert} = \begin{bmatrix} \dot{u} + qw - rv \\ \dot{v} + ru - pw \\ \ddot{w} + pv - qu \end{bmatrix} \tag{50}$$

From Newton's second law, $F = \frac{dp}{dt} = \frac{d(m.v)}{dt}$

Mass (m) is constant, $F = m \cdot \dot{v}_{inert}$

$$F = m(\dot{v}_b + (\omega_b \times v_b) \tag{51}$$

$$\begin{bmatrix} F_x \\ F_y \\ F_z \end{bmatrix} = m \cdot \begin{bmatrix} \dot{u} + qw - rv \\ \dot{v} + ru - pw \\ \ddot{w} + pv - qu \end{bmatrix} \tag{52}$$

Gravity from the inertial frame needs to be converted to the body frame using DCM,

$$F^b_{gravity} = C^b_{inert} \cdot F^{inert}_{gravity} \tag{53}$$

$$F^b_{gravity} = \begin{bmatrix} c\theta \cdot c\psi & c\theta \cdot s\psi & -s\theta \\ c\psi \cdot s\theta - c\phi \cdot s\psi & c\phi \cdot c\psi + s\theta \cdot s\phi \cdot s\psi & c\theta \cdot s\phi \\ c\phi \cdot c\psi \cdot s\theta + s\phi \cdot s\psi & -c\psi \cdot s\phi + c\phi \cdot s\theta \cdot s\psi & c\theta \cdot c\phi \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ mg \end{bmatrix} \tag{54}$$

$$F^b_{gravity} = \begin{bmatrix} -mg \sin\theta \\ mg \sin\phi \cos\theta \\ mg \cos\phi \cos\theta \end{bmatrix} \tag{55}$$

Substituting equation 55 in equation 52

$$\begin{bmatrix} F_x - mg \sin\theta \\ F_y + mg \sin\phi \cos\theta \\ F_z + mg \cos\phi \cos\theta \end{bmatrix} = m \begin{bmatrix} \dot{u} + qw - rv \\ \dot{v} + ru - pw \\ \ddot{w} + pv - qu \end{bmatrix} \tag{56}$$

Similarly, $T = \frac{dH}{dt}$, where H = Angular Momentum (H=I$\times\omega$)

$$T = I^b \dot{\omega}_b + \omega_b \times I^b w_b \tag{57}$$

Moment of inertia is given by,

$$I = \int_b r \times r \times dm \tag{58}$$

$$I = \begin{bmatrix} I_{xx} & -I_{xy} & -I_{xz} \\ -I_{xy} & I_{yy} & -I_{yz} \\ -I_{xz} & -I_{yz} & I_{zz} \end{bmatrix} \tag{59}$$

Where, I_{xx}, I_{yy}, I_{zz} are the Moment of Inertia along principal axes and I_{xy}, I_{zx}, I_{yz} are the Products of Inertia due to coupling between axes.

Symmetry about the X and Y axis makes the above equation equal to

$$I = \begin{bmatrix} I_{xx} & 0 & 0 \\ 0 & I_{yy} & 0 \\ 0 & 0 & I_{zz} \end{bmatrix} \tag{60}$$

Cross-product matrix [r ×] derivation is shown in appendix 14.8.

Hence Equation 57 equals,

$$\begin{bmatrix} L \\ M \\ N \end{bmatrix} = \begin{bmatrix} I_{xx} & 0 & 0 \\ 0 & I_{yy} & 0 \\ 0 & 0 & I_{zz} \end{bmatrix} \begin{bmatrix} p \\ q \\ r \end{bmatrix} + \begin{bmatrix} 0 & -r & q \\ r & 0 & -p \\ -q & p & 0 \end{bmatrix} \begin{bmatrix} 0 & -r & q \\ r & 0 & -p \\ -q & p & 0 \end{bmatrix} \begin{bmatrix} I_{xx} & 0 & 0 \\ 0 & I_{yy} & 0 \\ 0 & 0 & I_{zz} \end{bmatrix} \tag{61}$$

$$\begin{bmatrix} L \\ M \\ N \end{bmatrix} = \begin{bmatrix} I_{xx}\dot{p} - I_{yy}qr + I_{zz}qr \\ I_{yy}\dot{q} + I_{xx}pr - I_{zz}pr \\ I_{zz}\dot{r} - I_{xx}pq + I_{yy}pq \end{bmatrix} \tag{62}$$

Navigation Equations,

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ -\dot{z} \end{bmatrix} = \begin{bmatrix} C_b^{inert} \end{bmatrix} \begin{bmatrix} u \\ v \\ w \end{bmatrix} \tag{63}$$

C_b^{inert} is the Direction Cosine Matrix (DCM).

In total there are 12 equations:

- 3 DOF Translation (56)
- 3 DOF Rotation (62)
- Euler Angles - Angular velocity relations (37)
- Navigation/Position Equations (63)

8.11 6 DOF and 3 DOF Algorithms

Algorithms used in Simulink 6 Degree of Freedom and 3 Degree of Freedom blocks.

8.11.1 6 Degree of Freedom Algorithms

Translational motion of the body-fixed coordinate frame, where the applied forces [Fx Fy Fz]^{T} are in the body-fixed frame, and the mass of the body m is constant.

$$\begin{gathered}
\bar{F}_b = \begin{bmatrix} F_x \\ F_y \\ F_z \end{bmatrix} = m\left(\bar{V}_b + \bar{\omega} \times \bar{V}_b\right) \\
A_{bb} = \begin{bmatrix} \dot{u}_b \\ \dot{v}_b \\ \dot{w}_b \end{bmatrix} = \tfrac{1}{m}\bar{F}_b - \bar{\omega} \times \bar{V}_b \\
A_{be} = \tfrac{1}{m}F_b \\
\bar{V}_b = \begin{bmatrix} u_b \\ v_b \\ w_b \end{bmatrix}, \bar{\omega} = \begin{bmatrix} p \\ q \\ r \end{bmatrix}
\end{gathered} \tag{64}$$

The rotational dynamics of the body-fixed frame, where the applied moments are [L M N]^{T}, and the inertia tensor I is with respect to the origin.

$$\begin{gathered}
\bar{M}_B = \begin{bmatrix} L \\ M \\ N \end{bmatrix} = I\bar{\omega} + \bar{\omega} \times (I\bar{\omega}) \\
I = \begin{bmatrix} I_{xx} & -I_{xy} & -I_{xz} \\ -I_{yx} & I_{yy} & -I_{yz} \\ -I_{zx} & -I_{zy} & I_{zz} \end{bmatrix}
\end{gathered} \tag{65}$$

The relationship between the body-fixed angular velocity vector, [p q r]^{T}, and the rate of change of the Euler angles, $\begin{bmatrix} \dot{\phi} \; \dot{\theta} \; \dot{\psi} \end{bmatrix}^{T}$, are determined by resolving the Euler rates into the body-fixed coordinate frame. [17]

$$\begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} = J \begin{bmatrix} p \\ q \\ r \end{bmatrix} = \begin{bmatrix} 1 & (\sin\phi\tan\theta) & (\cos\phi\tan\theta) \\ 0 & \cos\phi & -\sin\phi \\ 0 & \frac{\sin\phi}{\cos\theta} & \frac{\cos\phi}{\cos\theta} \end{bmatrix} \begin{bmatrix} p \\ q \\ r \end{bmatrix} \tag{66}$$

8.11.2 3 Degree of Freedom Algorithms

We obtain 3 DOF algorithms from 6 DOF algorithms following these assumptions:

1. No Roll i.e. $\phi = 0$. As we do not have roll control, it is a reasonable assumption.

2. No motion in Y axis i.e. v = 0, zero lateral velocity.

Zero roll angle ($\phi = 0$) leads to decoupling of flat earth equations. Decoupling means that the equations of motion separate into two independent sets. One set is longitudinal motion (pitching and translating in X-Z plane) and the other set describes lateral-directional motion (rolling, yawing and sideslipping). Decoupled equations are much easier to handle in analytical studies. [18]

Referring to equation 56,

$$\begin{bmatrix} F_x - mg\sin\theta \\ F_y + mg\sin\phi\cos\theta \\ F_z + mg\cos\phi\cos\theta \end{bmatrix} = m \begin{bmatrix} \dot{u} + qw - rv \\ \dot{v} + ru - pw \\ \ddot{w} + pv - qu \end{bmatrix} \tag{67}$$

Substituting $\phi = 0$ and v = 0 in equation 67 and simplifying, we get: [19]

$$A_{xb} = \dot{u} = \tfrac{F_x}{m} - qw - g\sin\theta, A_{xe} = \tfrac{F_x}{m} - \sin\theta$$

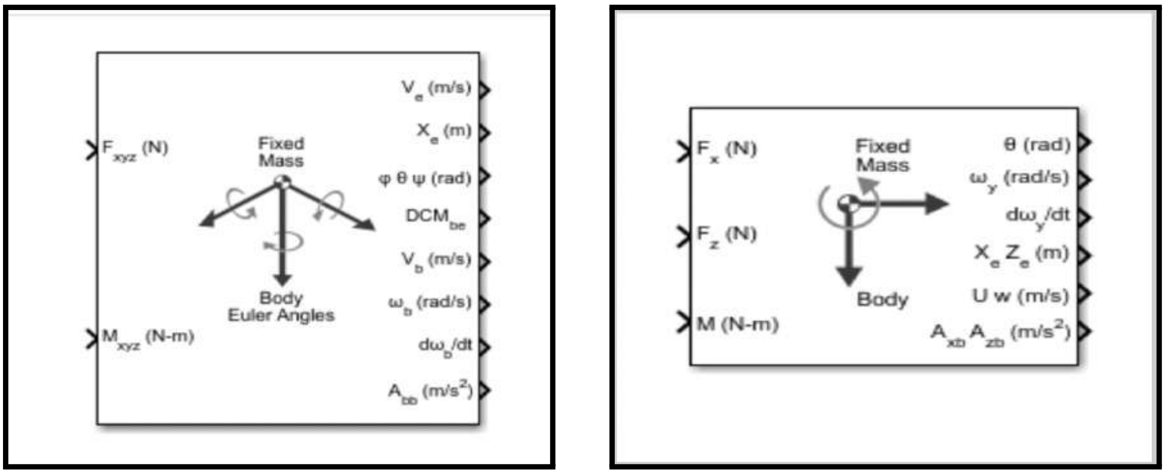
$$A_{zb} = \dot{w} = \tfrac{F_z}{m} + qu + g\cos\theta, A_{ze} = \tfrac{F_z}{m} + \cos\theta$$

where b - Body frame, e - Earth frame. (68)

$$\dot{q} = \tfrac{M}{I_{yy}}$$

$$\dot{\theta} = q$$

8.11.3 Simulink 6 DOF and 3 DOF Blocks



(a) 6 DOF Block

(b) 3 DOF Block

Figure 66: Simulink Blocks

9 Modelling and Simulation

Modelling is a technique for creating a virtual model of a physical system that involves both software and hardware. If the model's software components are driven by mathematical relationships, users can simulate this virtual representation under a variety of conditions to see how it performs.

Modelling and simulation are particularly useful for testing conditions that are difficult to replicate with hardware prototypes alone, especially early in the design process when hardware is not available. Iterating between modelling and simulation can improve the standard of the system design early on, reducing the number of errors detected later on. MATLAB Code - Data for Simulation is shown in appendix 14.9.

Properties	Value	Units
Mass	0.8246	Kilogram
Moment of Inertia (YY)	0.048	Kilogram metre^2
Moment Arm	0.29	Metre
Diameter of Rocket body	0.076	Metre
Drag Coefficient	0.33	-
Delay	2.5	Seconds
Drop Height	62	Metre
TVC Misalignment Angle	0.4	Degree

Table 26: Properties used in simulation

9.1 Basic Flight Simulation

In this simulation, the rocket is dropped from a particular height to obtain position, velocity and acceleration graphs. This simulation is performed to ensure the rocket reaches the ground at near-zero velocity further ensuring that there's minimal damage to the rocket body and landing legs. The simulation is divided into three sections namely, Input, Calculation and Output.

Input Section: Input section consists of the thrust subsystem and rocket mass. The Lookup Table block in Simulink allows for thrust variation with respect to time. A Transport Delay block is used to make provisions for time delay ensuring the solid motor to ignite just before landing. Dry mass and solid propellant mass are both included in mass block.

Calculation Section: This section consists of drag subsystem, integration, summation and division blocks. Net force on the rocket body is calculated and acceleration is obtained. Acceleration is integrated to obtain velocity which is further integrated to obtain the position of the rocket. Velocity obtained is used to calculate the drag force.

Output Section: A Scope block is used to visualise the position, velocity, acceleration and thrust curves.

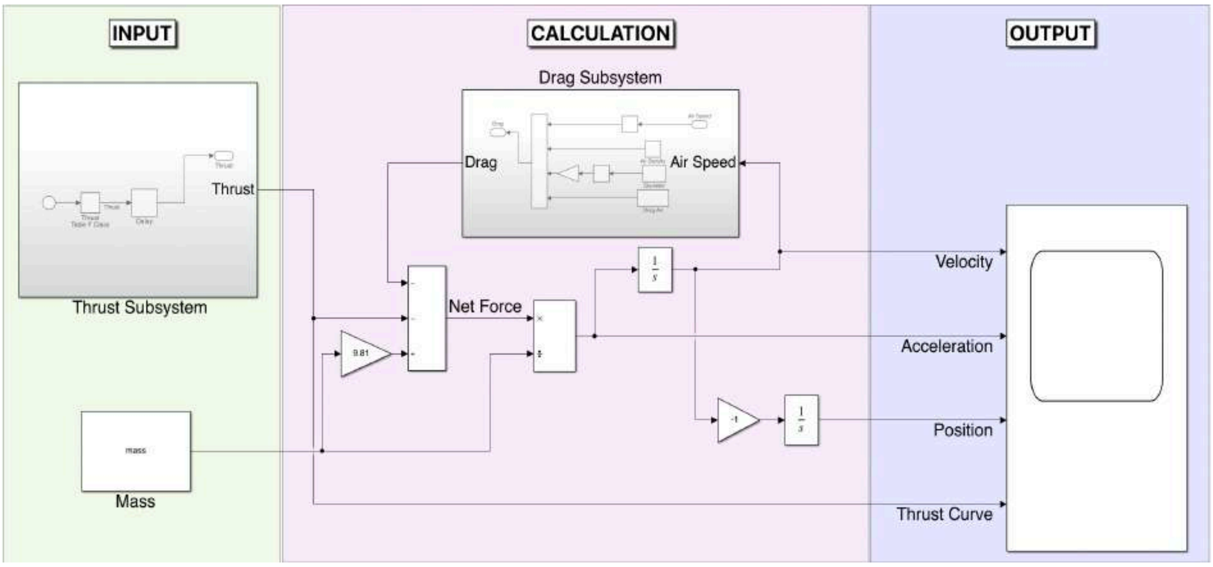


Figure 67: Basic Flight Simulation

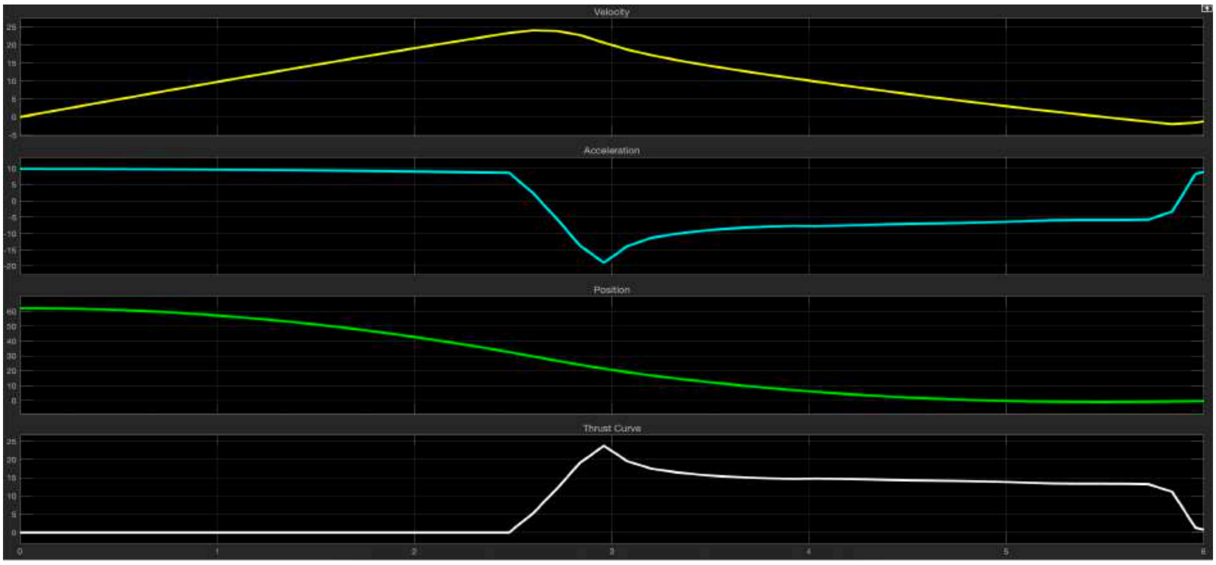


Figure 68: Results

Analysis and Interpretation of Results: The rocket experiences free-fall as it is dropped from the drop height; its velocity increases and reaches its maximum value until the solid motor is ignited. Upon ignition, acceleration drops and so does the velocity until it reaches near-zero velocity at 6 seconds. Thus zero velocity is achieved close to the ground. Thrust curve of an F-15 solid motor is also shown.

9.2 3 Degree of Freedom Flight Simulation

In this simulation, a closed-loop feedback system is designed to control the rocket's attitude. The simulation is divided into three sections namely, Input, Dynamics and Control.

Input Section: Input section consists of the F-15 thrust curve which is represented by From Workspace block and the TVC misalignment angle block (Imperfections in TVC mounting).

Dynamics Section: Dynamics section consists of the actuator subsystem and 3 DOF block 8.11.2.

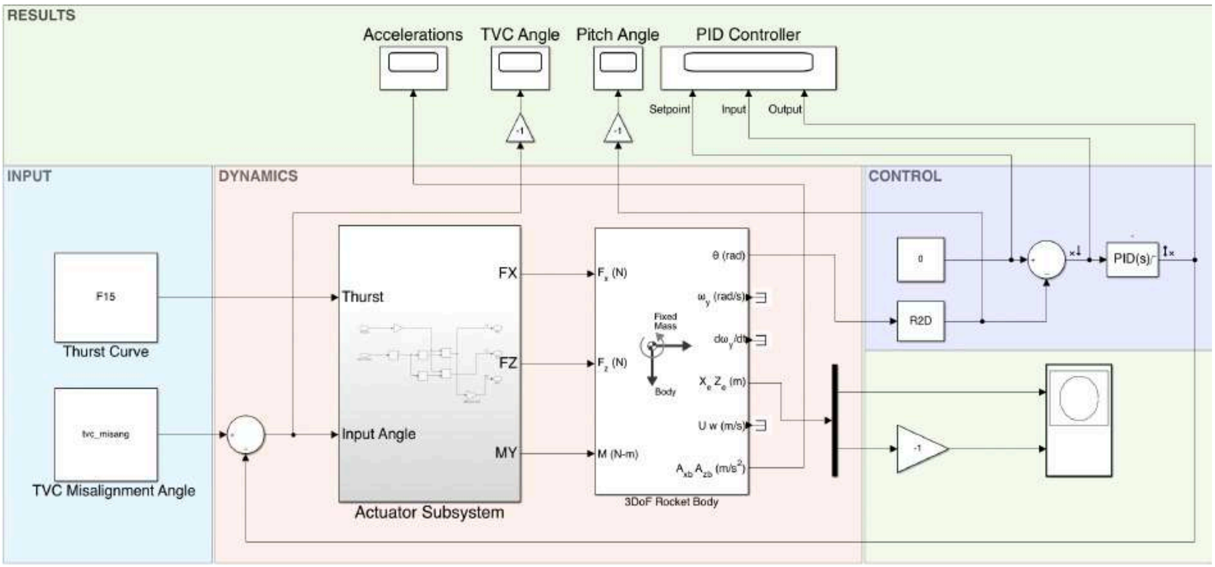


Figure 69: 3 DOF Flight Simulation

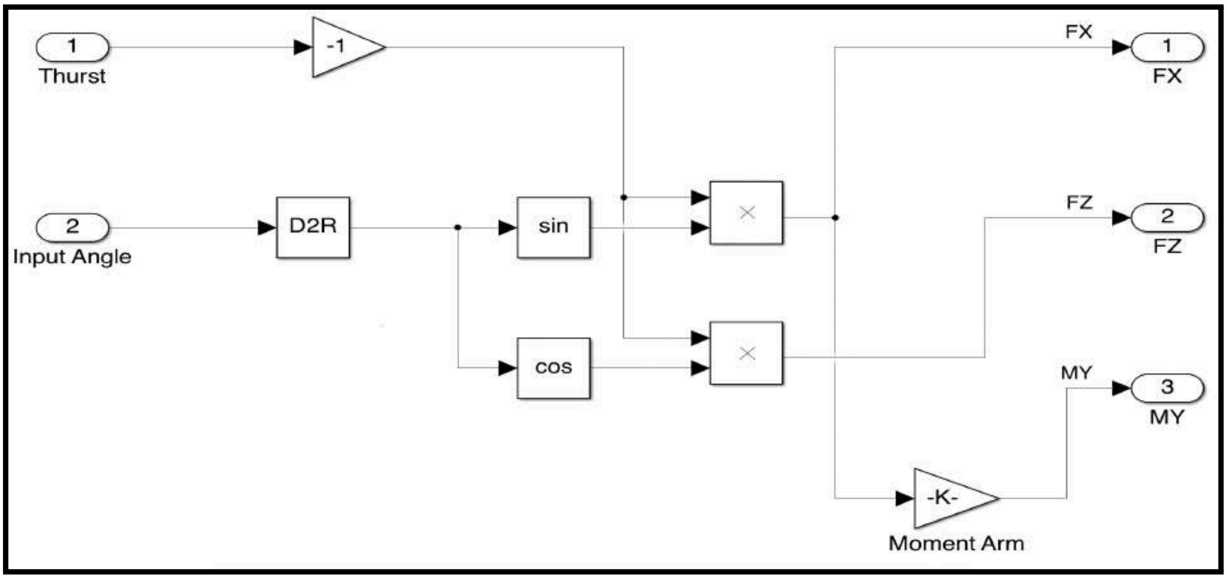


Figure 70: Actuator Subsystem

Actuator Subsystem:

$$\begin{aligned} F_x &= \text{Thrust} \times \sin(\theta) \\ F_z &= \text{Thrust} \times \cos(\theta) \\ M_y &= \text{Thrust} \times \sin(\theta) \times \text{Moment Arm} \end{aligned} \tag{69}$$

where $\theta =$ Input Angle, Moment Arm = Distance from CG to TVC mount location.

Control Section: The rocket's attitude is controlled by a Proportional Integral Derivative (PID) controller. The desired orientation of the rocket is vertical, i.e. zero degree deviation. Hence, a set-point of zero is used, and the error is the difference between the set-point and the rocket angle. The PID controller constantly monitors the error value and uses it to calculate the proportional, integral, and derivative gains. The controller then combines these three values to produce the output. The output of the PID block is subtracted from the TVC misalignment angle and fed to the actuator subsystem as an input. As a result, a closed-loop feedback system is established.

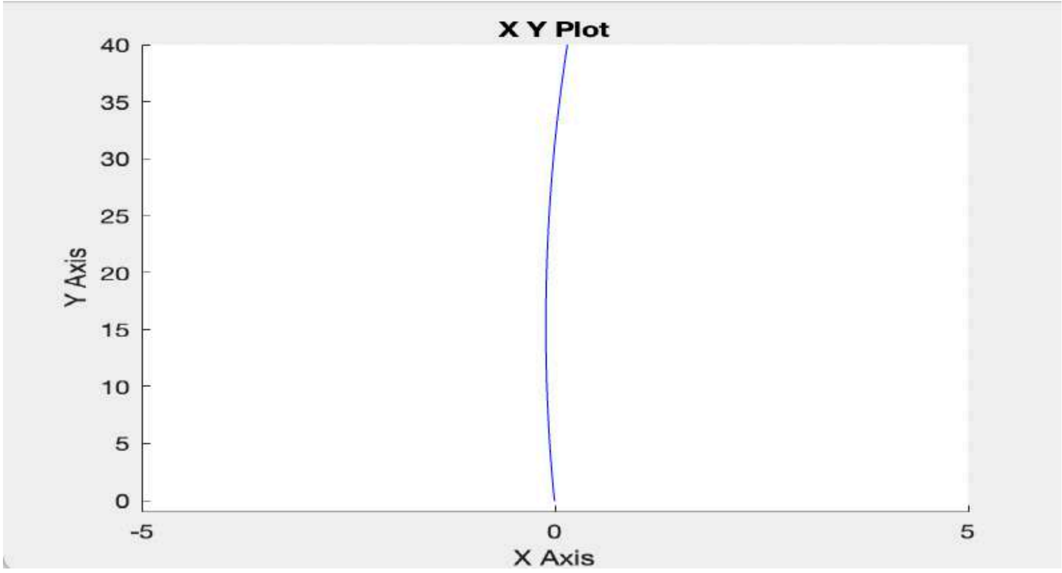


Figure 71: Trajectory

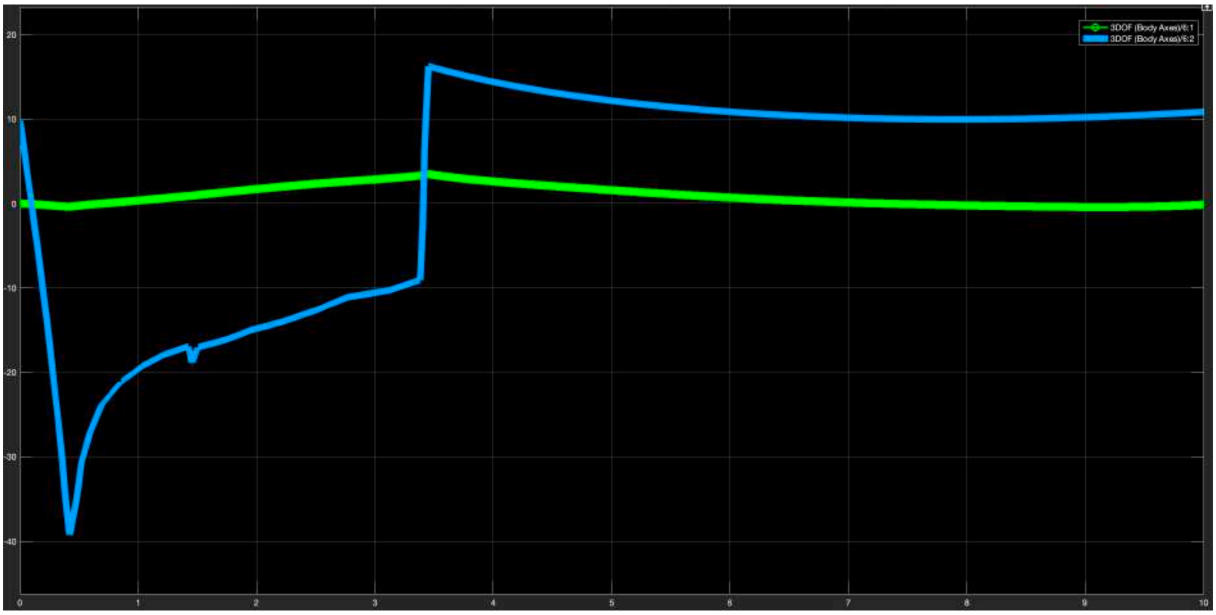


Figure 72: Acceleration

Analysis and Interpretation of Results: The rocket's trajectory during the first 40 metres, when the thrust vector control is active, that is, when the solid motor is ignited, is shown. The rocket attempts to maintain a straight trajectory, however the deviation is caused due to the lift force acting perpendicular to the direction of travel. The green line in the acceleration graph shows the acceleration in the lateral direction (a_X). The PID controller effectively reduces the deviation. The blue line represents the F-15 thrust curve.

Errors in the trajectory and acceleration graph can be corrected using advanced controllers such as Full State Feedback Control. Unlike traditional controllers such as PID, Linear Quadratic Regulator (LQR) can handle cases with parameter uncertainties and disturbances. LQR produces a good approximation for nonlinear systems that occur in real world.

10 Flight Computer

10.1 Introduction

The Flight computer is a microcontroller unit coupled with peripheral components consisting of input devices such as sensors, potentiometer, switch etc and output devices such as relays, servos, and motors. The microcontroller unit interfaces with the sensors and sends signals as output to control external processes such as the triggering of the engine, the actuation of servos and the transmission of data. The interface between all devices connected to the MicroController Unit (MCU) takes place with the help of certain communication protocols. To name some of the protocols, we have, the Integrated Circuit (I2C) Protocol, the Serial Peripheral Interface(SPI) Protocol etc.

The need for a Flight Computer arises from the fact that inspite of having an adequate amount of passive stability rooting form the rocket design, the uncertainties due to numerous variables affecting the attitude of the model rocket during the landing phase warrants the need of a controller which corrects the attitude of the system. The project uses PID controller to provide output to the TVC which actuates the rocket motor.

For simplicity the microcontroller along with its peripherals will be referred to as Flight Computer from here on.

10.2 Components

10.2.1 Microcontroller

Figure 73: Microcontroller: Teensy 3.2

The microcontroller acts as the brain of the control system. We used a Teensy 3.2 microcontroller board owing to its availability and versatility. With a 32 bit ARM Cortex-M4 at 72 MHz(96MHz overclocked), 256Kb Flash, 64Kb RAM and 34 pin input/output ports, Teensy 3.2 by far outperforms its rivals in the same footprint range.

10.2.2 Inertial Measurement Unit (IMU)

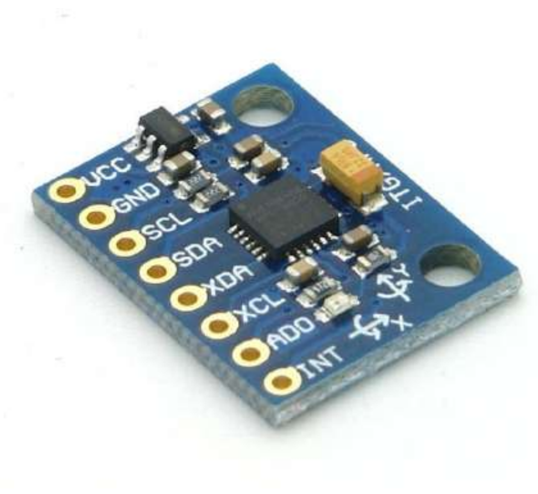


Figure 74: IMU: MPU6050 Module

The IMU is responsible for determining the orientation of the rocket during it time in descent. The MPU6050 sensor by Invensense does the job needed and is abundant in the market, thus we found it easy to incline towards this particular sensor. In addition the MPU6050 has enough documentation and open source libraries to make it easy for us to access the orientation parameters from its registers without worrying much about the inner workings of the sensor. It interfaces with the Teensy 3.2 MCU using the I2C protocol.

10.2.3 Barometric Pressure Sensor (Altimeter)

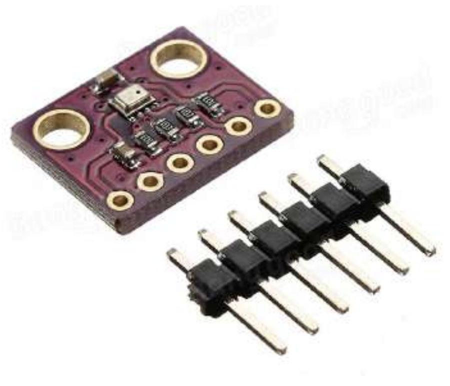


Figure 75: Altimeter: BMP180 Module

As our project's performance depends on accurate measurements of altitude above the ground in order to trigger the model rocket engine, it is crucial to select a sensor which measures altitude with appropriate accuracy. The BMP280 by BOSCH was considered suitable as it is a barometric pressure sensor and is sensitive to slight changes in the atmospheric pressure (accurate upto 1meter) this pressure can be used to calculate the altitude above the ground level. The manufacturer projected sensitivity and range is +- 1 hPa and -500m to +9000m.

10.2.4 Transceivers

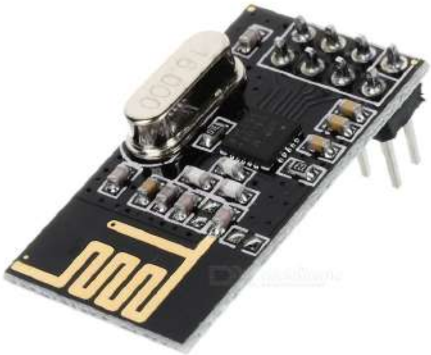


Figure 76: nRF24l01+ Transceiver module

Transceivers are devices which can act as a transmitter and a receiver based on the command input provided by the microcontroller. For our project we went with a pair of nRF24L01+ modules to transmit data from the model rocket to the ground station. The device consumes a maximum of 13.5mA at a data rate of 2Mbps. The module is widely popular for its compatibility with microcontrollers and its ultra low power consumption.

10.2.5 Voltage Regulator

Figure 77: Voltage Regulator

In order to function accurately the flight computer needs a stable power supply, failing which the sensors can malfunction and provide garbage values. This problem is eliminated by the use of a voltage regulator. We chose the LM2596 S DC-DC Buck Convertor as it was readily available and some LM7805 linear regulators as backups, however it is highly unlikely that the buck would fail as we would be operating well within the operating voltage and current parameters.

10.2.6 Trigger Channels

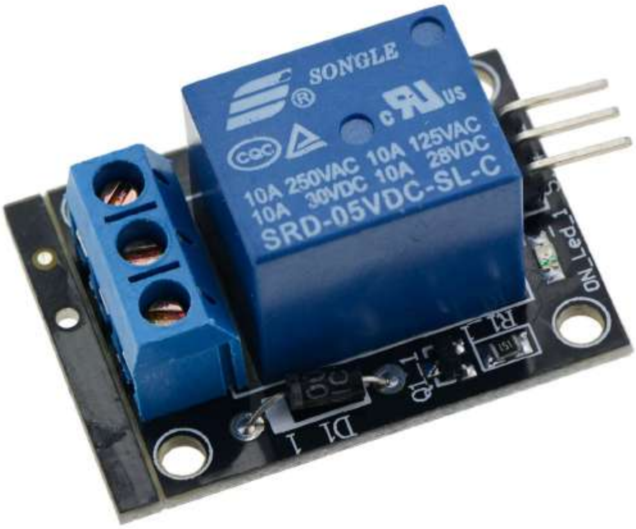


Figure 78: Trigger Channels

The model rocket engine is required to be ignited at the calculated altitude so as to have its burn time adjusted in accordance with the landing. To fulfil this requirement, we used a single-channel relay to bring about the triggering of the model rocket engine.

10.2.7 Servo Motors

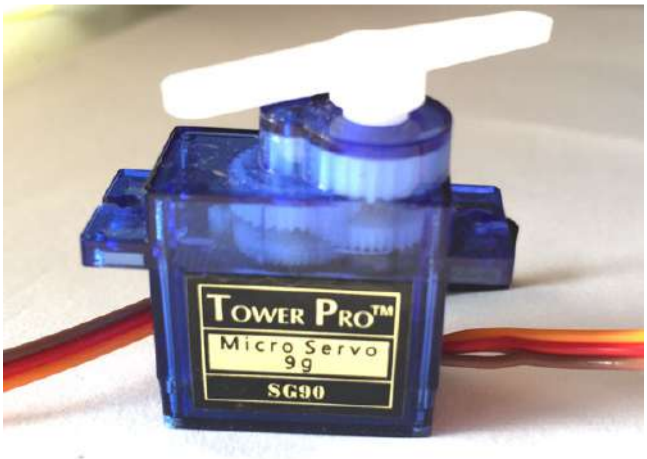


Figure 79: Metal Gear Servo Motor

After the sensing phase the controller gives a controlled output which in turn actuates the servo motors. These servo motors move the TVC and as a result the vectoring of the thrust provided by the rocket engine produces an appropriate amount of torque to restore the rocket body back to vertical. For the project we use SG90 metal gear servos.

10.2.8 Power Source

Figure 80: LiPO Battery

To power the onboard flight computer a Lithium Polymer (LiPo) Battery is used. The battery employed has a capacity of 850mAh, a discharge rate of 30C, a nominal voltage of 7.4v and weighs around 53g.

10.3 Configuration Selection

The selection of these components was primarily based on availability. However, in terms of performance our concern revolved around the weight of the components and in particular, the current draw. This is because the electronic components such as sensors and the microcontroller are quite sensitive to current as they get fried in an episode where the current overshoots the maximum value allowed by their inner circuitry. That said, these episodes are rare and hardly any cause of concern. The following table lists the current drawn by each component in their maximum power setting. The total current drawn is then compared with the maximum discharge capabilities of the battery to avoid over-discharging the battery [21] [22] [23] [24] [25].

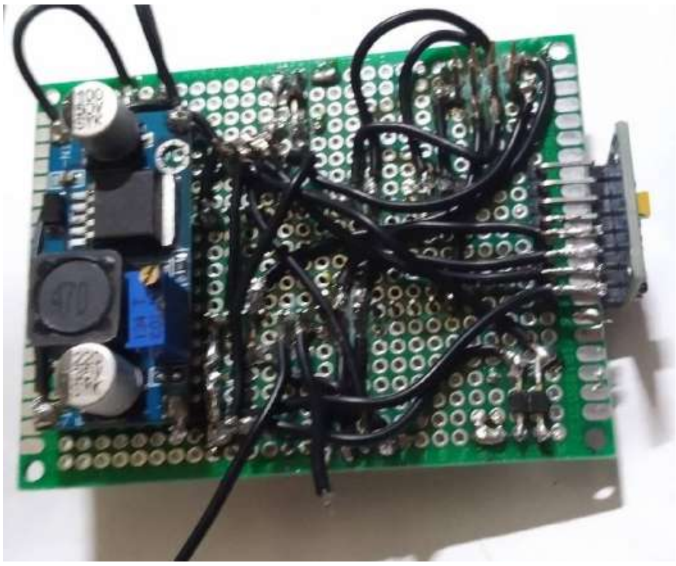


Figure 81: Flight Computer - 1

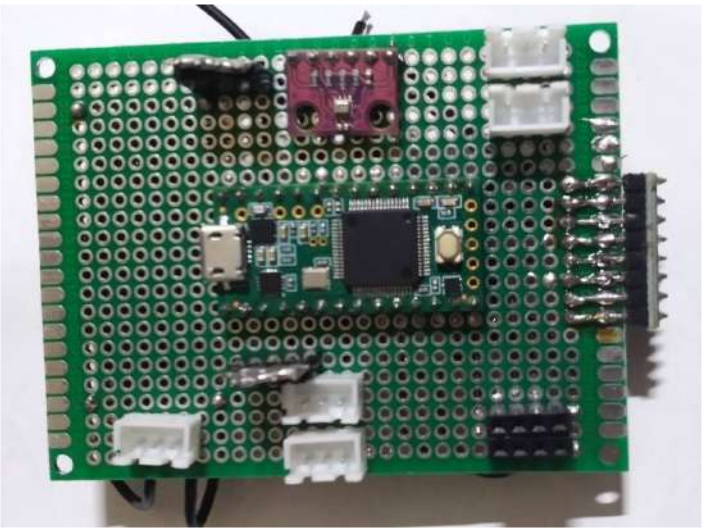


Figure 82: Flight Computer - 2

Component	Module Name	Operating Voltage (V)	Max. Current Draw (mA)
Microcontroller	Teensy 3.2	3.6V - 6.0V	1100 mA
Pressure Sensor	GY-BMP280	3.0V - 3.3V	1.12 mA
Inertial Measurement Unit (IMU)	GY-521 MPU60050	3.0V - 5.0V	3.9 mA
Transceiver	nRF41|01+	1.9V - 3.6V	13.5 mA
Servo Motor x3	SG90	3.0V - 6.0V	750 mA x3 = 2250 mA
Relay	1 channel 5V Relay	5.0V	20 mA
			3388.52 mA = 3.39 A

Table 27: Components Specifications

10.4 Schematic Diagram

The pin-out diagram of each component and the interconnections between them is shown in the following schematic.

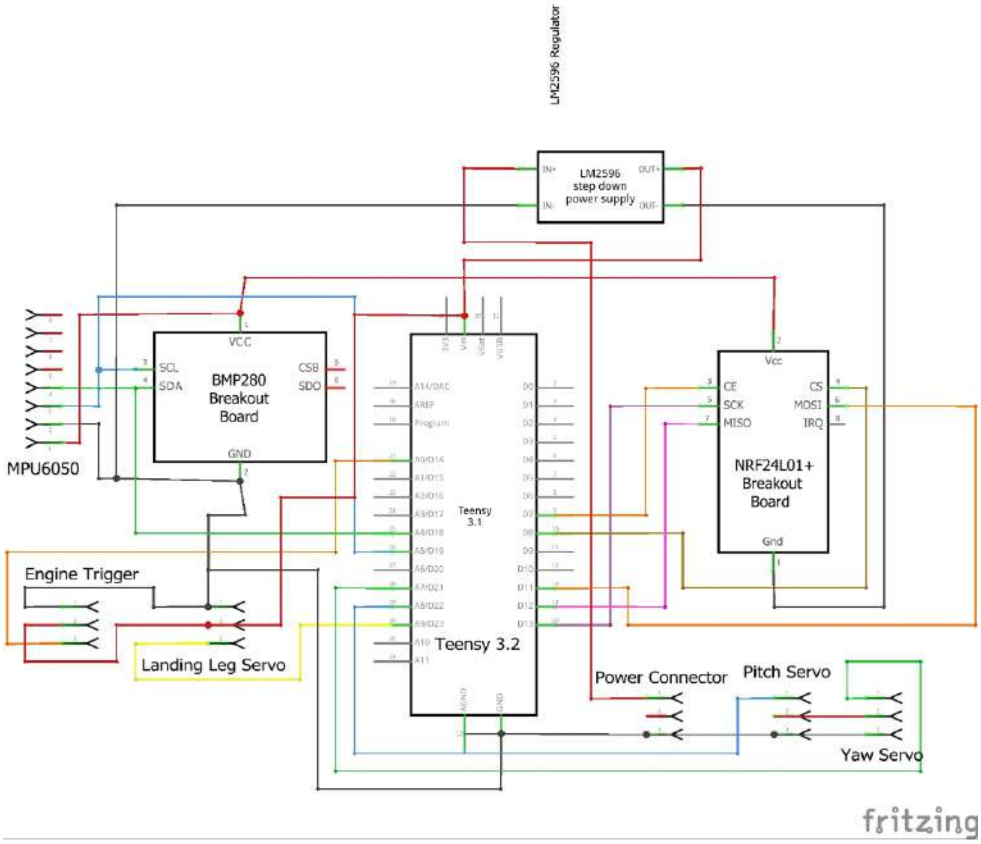


Figure 83: Schematics

The schematic diagram is one of the starting points for making a flight computer. It becomes easy to understand the logic levels at which each component operates to avoid overloading the components with excess power. The above schematic helps us understand the connections in a much simpler way. However, this information only acts as an initial step in making a prototype of the flight computer board. This was done in an open source PCB design software called Fritzing.

10.5 Printed Circuited Board Diagram

After the initial prototyping of the flight computer, we came to realize that printing a PCB is a much simpler way of making multiple flight computers ready to being tested. This saves lot of time in testing and is much quicker to prototype. This was also created through Fritzing version 0.9.3b.

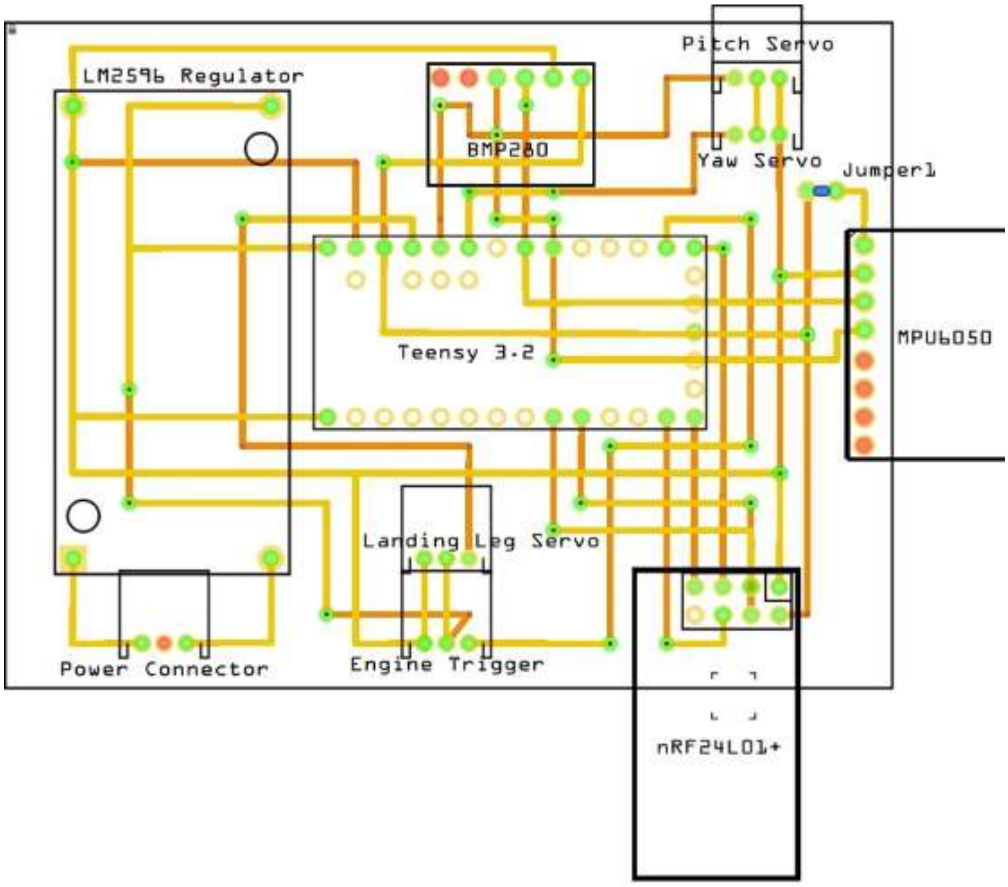


Figure 84: Printed Circuit Board

10.6 Block Diagram

The Block Diagram conveys all the superficial information about the flight computer hardware such as the protocols followed to communicate between the component and the microcontroller (i.e. I2C, SPI, CAN, UART etc). Furthermore, it also contains arrows representing the flow of signal to or from the microcontroller.

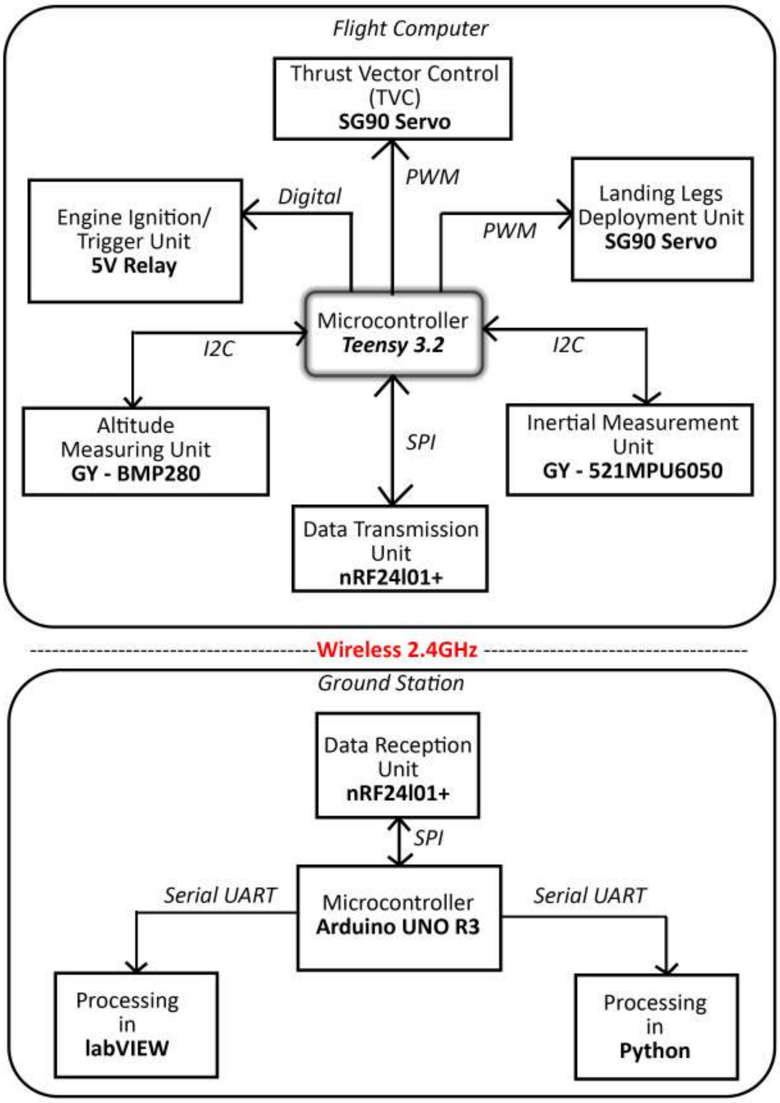

Flight Computer
Thrust Vector Control (TVC) SG90 Servo
PWM
Digital
Engine Ignition/ Trigger Unit 5V Relay
PWM
Landing Legs Deployment Unit SG90 Servo
Microcontroller Teensy 3.2
I2C
I2C
Altitude Measuring Unit GY - BMP280
SPI
Inertial Measurement Unit GY - 521MPU6050
Data Transmission Unit nRF24l01+
Wireless 2.4GHz
Ground Station
Data Reception Unit nRF24l01+
SPI
Serial UART
Microcontroller Arduino UNO R3
Serial UART
Processing in labVIEW
Processing in Python

Figure 85: Block Diagram

10.7 Flow Chart

The flowcharts very well represent the control flow and the algorithm followed by the program. The following flowchart depicts underlying logic and algorithm behind the program.

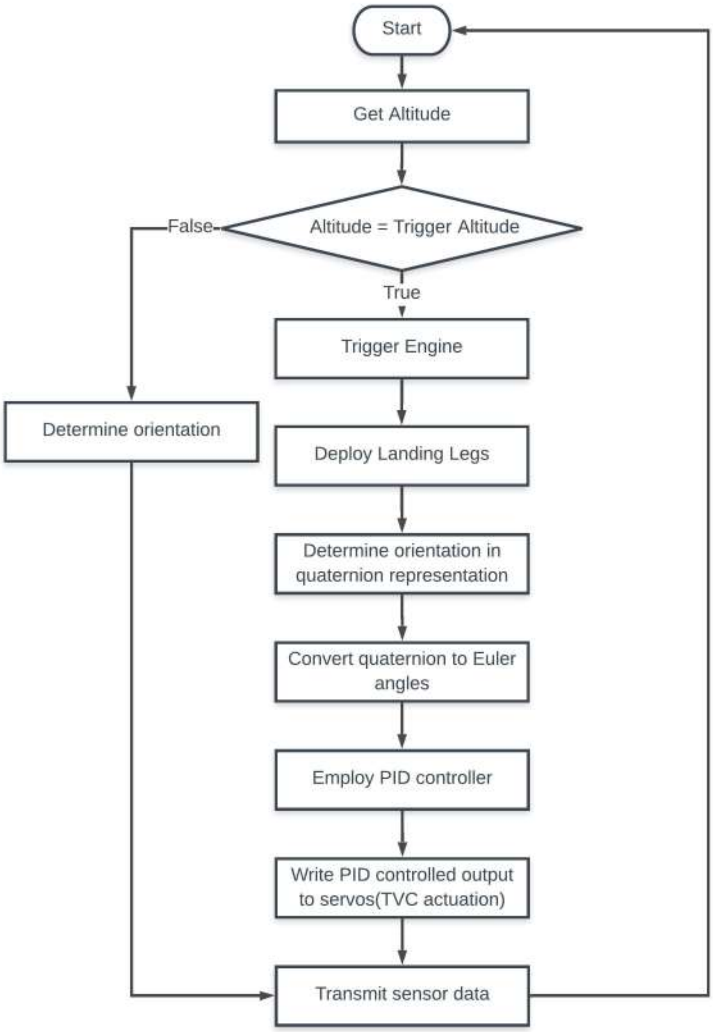

Start
Get Altitude
Altitude = Trigger Altitude
False
True
Trigger Engine
Determine orientation
Deploy Landing Legs
Determine orientation in quaternion representation
Convert quaternion to Euler angles
Employ PID controller
Write PID controlled output to servos(TVC actuation)
Transmit sensor data

Figure 86: Flow Chart

10.8 Processing in Python

The flight computer transmits data to the ground station, where the transmitter data gets picked up by another nRF24l01+ module and transferred to a processing environment such as python or labVIEW. For this purpose we chose python environment and programmed it to process the orientation and altitude data sent by the flight computer inside the model rocket, and provide a visual understanding of the real-time orientation of the rocket during its descent. The following snippet shows a 3D representation of the model rocket and its orientation with respect to the vertical.

The IMU sensor data transmitted by the flight computer in the form of a quaternion needs to be converted into the parameters understood by the python environment. Thus, in order to convert a quaternion representation of orientation to the Euler angle representation, the following operations are performed.

$$\begin{bmatrix} \phi \\ \theta \\ \psi \end{bmatrix} = \begin{bmatrix} \mathrm{atan}\,2\left(2\left(q_0q_1 + q_2q_3\right), 1 - 2\left(q_1^2 + q_2^2\right)\right) \\ \mathrm{asin}\left(2\left(q_0q_2 - q_3q_1\right)\right) \\ \mathrm{atan}\,2\left(2\left(q_0q_3 + q_1q_2\right), 1 - 2\left(q_2^2 + q_3^2\right)\right) \end{bmatrix} \tag{70}$$

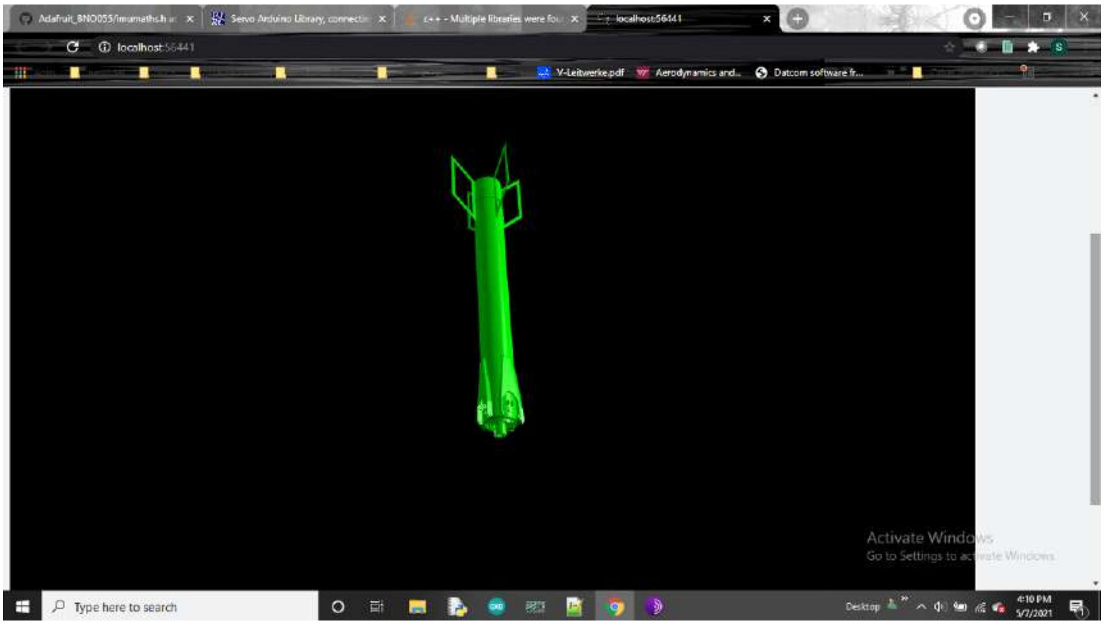


Figure 87: Processing via Python

11 Manufacturing

To develop some of the components required for the model rocket, we employed a 3D-printing process. Fused Deposition Modelling (FDM) also commonly known as Fused Filament Fabrication (FFF), is an additive manufacturing process wherein a 3D-printing material (mostly thermoplastics) are melted by the means of an extrusion process. The melted printing material is then moved in a predetermined direction as obtained from sliced CAD data. FDM is the most common type of 3D printing technology. An FDM printer is shown in Figure 88.

Figure 88: FDM Printer: Ender 3

Process of working:

- Once the nozzle of the 3D printer has reached its desired temperature, the printing filament is fed to the extrusion head.
- A 3-directional system which is attached to the extrusion head allows for movement in the X,Y,Z directions. To fill an area, multiple passes are required.
- Cooling of the melted 3D printing material is can be accelerated by the usage of a fan.
- To fill an area, multiple passes are required and the process is repeated until the part is complete.

We used this process for the fabrication of the Thrust Vector Control (TVC), passive fins, landing legs and bulkheads. The 3D printing material chosen was Polylactic Acid (PLA). PLA was a good choice as opposed to other materials such as Acrylonitrile Butadiene Styrene (ABS), Polyethylene Terephthalate (PET), Polyethylene Terephthalate Glycol (PETG), Thermoplastic Polyurethane (TPU), mainly due to the reason that PLA is made of lactic acid and is biodegradable.

11.1 Manufactured Parts

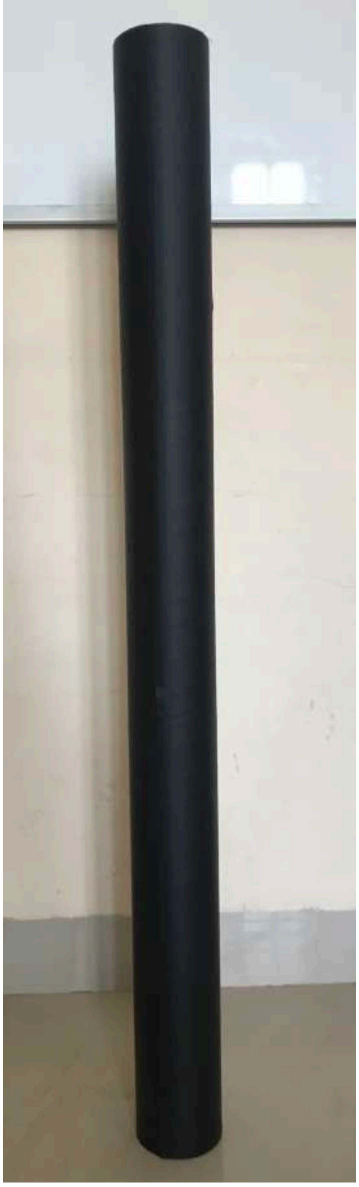


Figure 89: Airframe

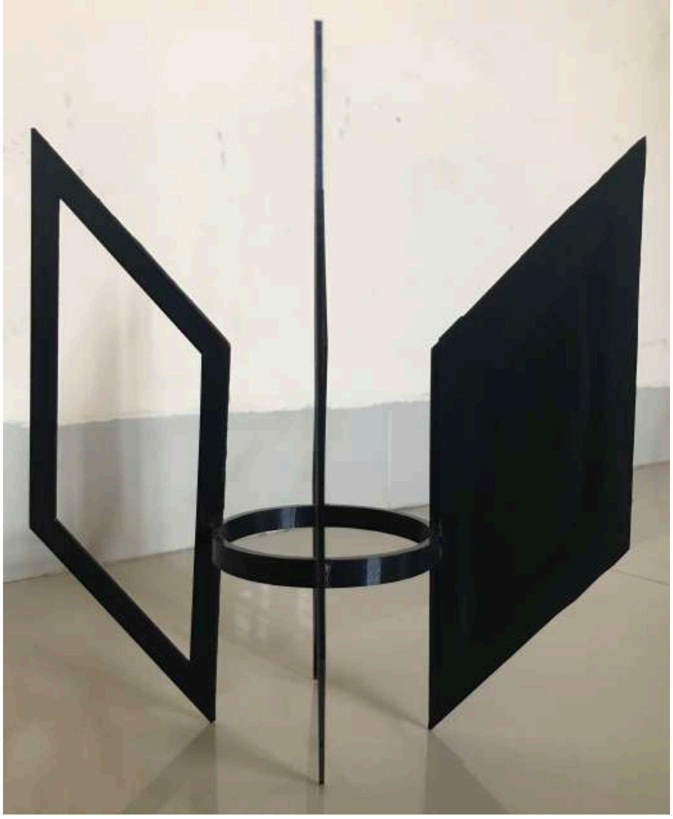


Figure 90: Fins

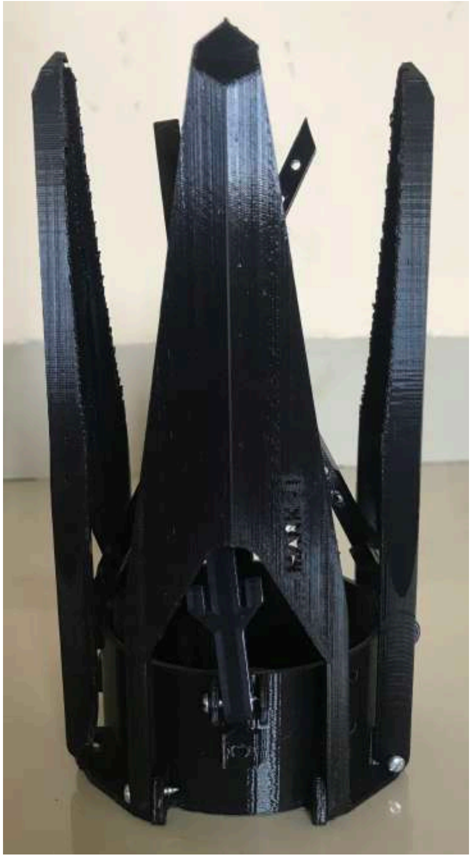


Figure 91: Landing Leg Assembly - Closed Form

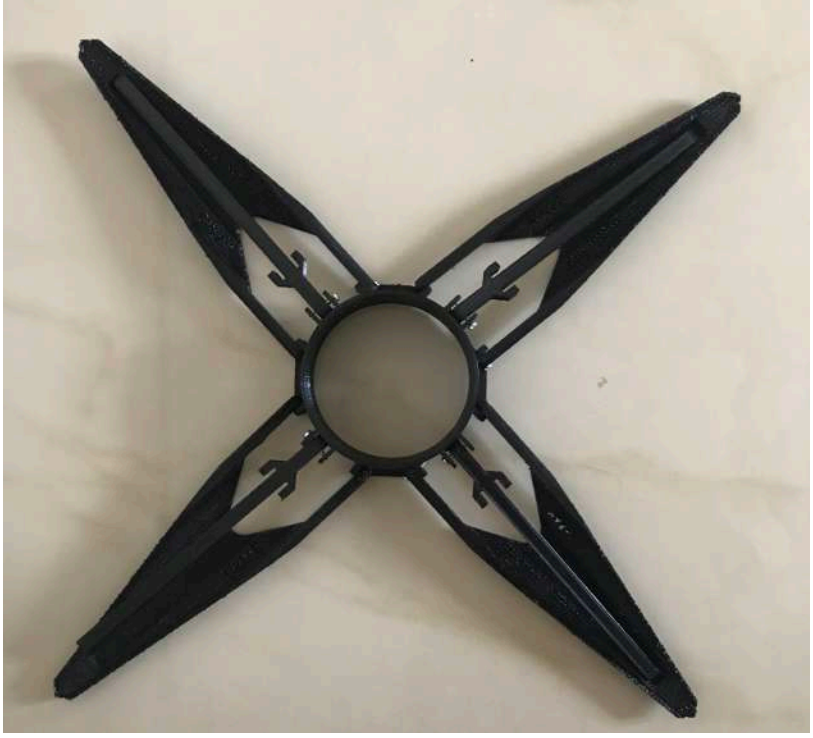


Figure 92: Landing Leg Assembly - Open Form

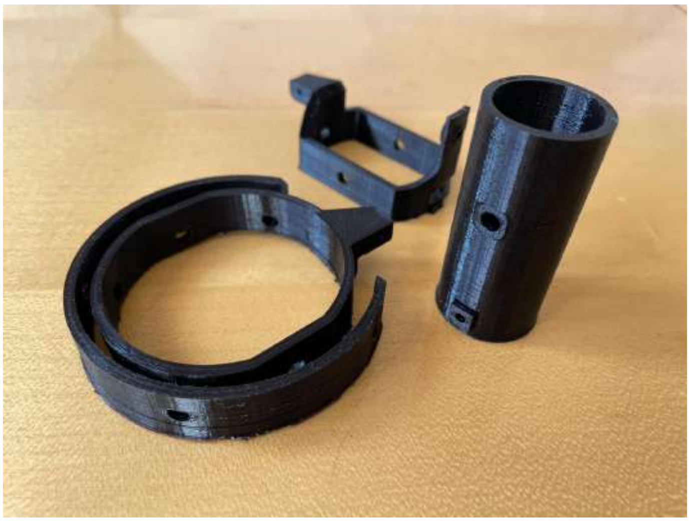


Figure 93: Thrust Vector Control

12 Conclusions

The design and sizing of the rocket was mainly dictated by the solid rocket motor's average thrust. Since the average thrust was just adequate and just sufficed the requirements set by the objectives, so the team predominantly focused on optimizing the weight and hence optimization of design was prioritized. Design streamlining was assisted by conducting an array of analysis including static structural analysis, modal frequency analysis and Computational Fluid Dynamics (CFD).

Thrust Vector Control was an integral part of the model rocket, which allowed effective attitude control. We used a gimballed thrust vectoring mechanism and to do so, two designs were developed through optimised iterations. The three parts of the TVC namely motor mount, inner gimbal and outer gimbal were used to house the motor and provide movement in the pitch and yaw axes.

We first understood the dynamics behind the 6 DOF and 3 DOF blocks to ensure that the simulations were not a blackbox.The use of Euler angles and Quaternions to represent attitudes was also discussed. The equations for the flat earth six degrees of freedom were derived and reduced to three degrees of freedom. MATLAB and Simulink were used to create the basic flight simulation that ensured the rocket touched the ground at a near-zero velocity. The mass of the solid rocket motor was also determined using this simulation, which took into account the thrust generated by the solid rocket motor. We were able to fine-tune the PID gains which ensured the rocket remained vertical using the 3 DOF flight simulation.

As with all the parts of the rocket, the flight computer auto corrects the error in the inclinations with respect to the vertical caused due to uncertain parameters such as wind and manufacturing error leading to CG imbalances. The correction brought about by the flight computer was essentially a controlled output produced by the PID controller in response to the error between actual and the set point readings of inclinations. However, due to it being dependent on a PID controller, the tuning of the gains was almost always an experimental problem. The simulations performed in MATLAB served as a sufficient approximation to obtain the gains needed by the flight computer. The realtime knowledge of orientation gives an insight of how well the whole system responds in an experimental environment. Therefore, the model rocket comprising of all the subsystems discussed, does hold true promise in the understanding of autonomous landings for similar vehicles in its class.

This report comprises an overall structure and work-flow to understand the fundamentals and working behind low and medium powered model rockets. Due to the COVID-19 crisis, the testing phase could not be completed.

12.1 Scope for future work

If there was no time barrier present, the team wanted to indulge in other methods for achieving these objectives. To begin with, PID tuning using neural networks (NN) and using reinforcement learning (RL) to land a model rocket. Apart from this, using computer vision for orientation data, the control system can be designed such that precise landing can be obtained with high accuracy too and for this method we would use convolutional neural networks (CNN). But this approach would require expensive electric ducted fans so as to repeat the tests frequently and for larger duration. Advanced control algorithm such as Linear Quadratic Regulator (LQR) or Model Predictive Control (MPC) can be implemented. LQR provides the control engineer with two knobs to directly penalize states and controls respectively and tries to synthesize a controller which respects the cost function.

13 References

[1] S. Niskanen, "Openrocket technical documentation," *Development of an Open Source model rocket simulation software*, pp. 11–13, 2013.

[2] G. K. Mandell, G. J. Caporaso, and W. P. Bengen, *Topics in advanced model rocketry*. MIT Press, 1973.

[3] M. Canepa, *Modern high-power rocketry*. Trafford Publishing, 2005, vol. 2.

[4] "F-15 motor specifications," https://www.grc.nasa.gov/www/k-12/rocket/rktcp.html, accessed:2021/05/1.

[5] "F-15 motor thrust curve," https://www.thrustcurve.org/motors/Estes/F15/, accessed:2021/05/1.

[6] "Determining centre of gravity," https://www.grc.nasa.gov/www/k-12/rocket/rktcg.html, accessed:2021/05/1.

[7] "Determining centre of pressure," https://www.grc.nasa.gov/www/k-12/rocket/rktcp.html, accessed:2021/05/1.

[8] N. Tun, M. Thuri, Z. Li *et al.*, "Aerodynamic coefficients prediction for 122 mm rocket by using computational fluid dynamics," in *IOP Conference Series: Materials Science and Engineering*, vol. 816, no. 1. IOP Publishing, 2020, p. 012010.

[9] "Basics of dynamic flight analysis, part 2, the corrective moment coefficient," https://www.apogeerockets.com/education/downloads/Newsletter193.pdf, accessed:2021/05/1.

[10] "Basics of dynamic flight analysis, part 3, the damping moment coefficient," https://www.apogeerockets.com/education/downloads/Newsletter195.pdf, accessed:2021/05/1.

[11] "Basics of dynamic flight analysis, part 5, the damping ratio," https://apogeerockets.com/education/downloads/Newsletter197.pdf, accessed:2021/05/1.

[12] J. S. Barrowman, "The practical calculation of the aerodynamic characteristics of slender finned vehicles," 1967.

[13] "Openfoam tutorial guide," https://www.openfoam.com/documentation/tutorial-guide.

[14] "Gimbaled thrust," https://www.grc.nasa.gov/WWW/K-12/rocket/gimbaled.html.

[15] "The vector product," https://www.mathcentre.ac.uk/resources/uploaded/mc-ty-vectorprod-2009-1.pdf.

[16] *J. Vandiver, and David Gossard. 2.003SC Engineering Dynamics. Fall 2011. Massachusetts Institute of Technology: MIT OpenCourseWare, https://ocw.mit.edu. License: Creative Commons BY-NC-SA.*

[17] "6 dof (euler angles)," https://in.mathworks.com/help/aeroblks/6dofeulerangles.html, accessed:2021/05/1.

[18] B. Stevens and F. Lewis, *Aircraft Control and Simulation*. Wiley, 2003.

[19] "3 dof (body axes)," https://in.mathworks.com/help/aeroblks/3dofbodyaxes.html, accessed:2021/05/1.

[20] “Flight mechanics,” https://faculty.washington.edu/lum/EducationalVideos.htm#FlightMechanics, accessed:2021/05/1.

[21] “Mpu6050 module - data sheet,” https://invensense.tdk.com/wp-content/uploads/2015/02/MPU-6000-Datasheet1.pdf, accessed:2021/05/1.

[22] “Bmp280 module - data sheet,” https://www.bosch-sensortec.com/media/boschsensortec/downloads/datasheets/bst-bmp280-ds001.pdf, accessed:2021/05/1.

[23] “nrf24l01+ transceiver - data sheet,” https://www.sparkfun.com/datasheets/Components/SMD/nRF24L01Pluss_Preliminary_Product_Specification_v1_0.pdf, accessed:2021/05/1.

[24] “Teensy 3.2 - data sheet and schematics,” https://www.pjrc.com/store/teensy32.html, accessed:2021/05/1.

[25] “Sg90 - servo motor - specifications,” https://www.towerpro.com.tw/product/sg90-7/, accessed:2021/05/1.

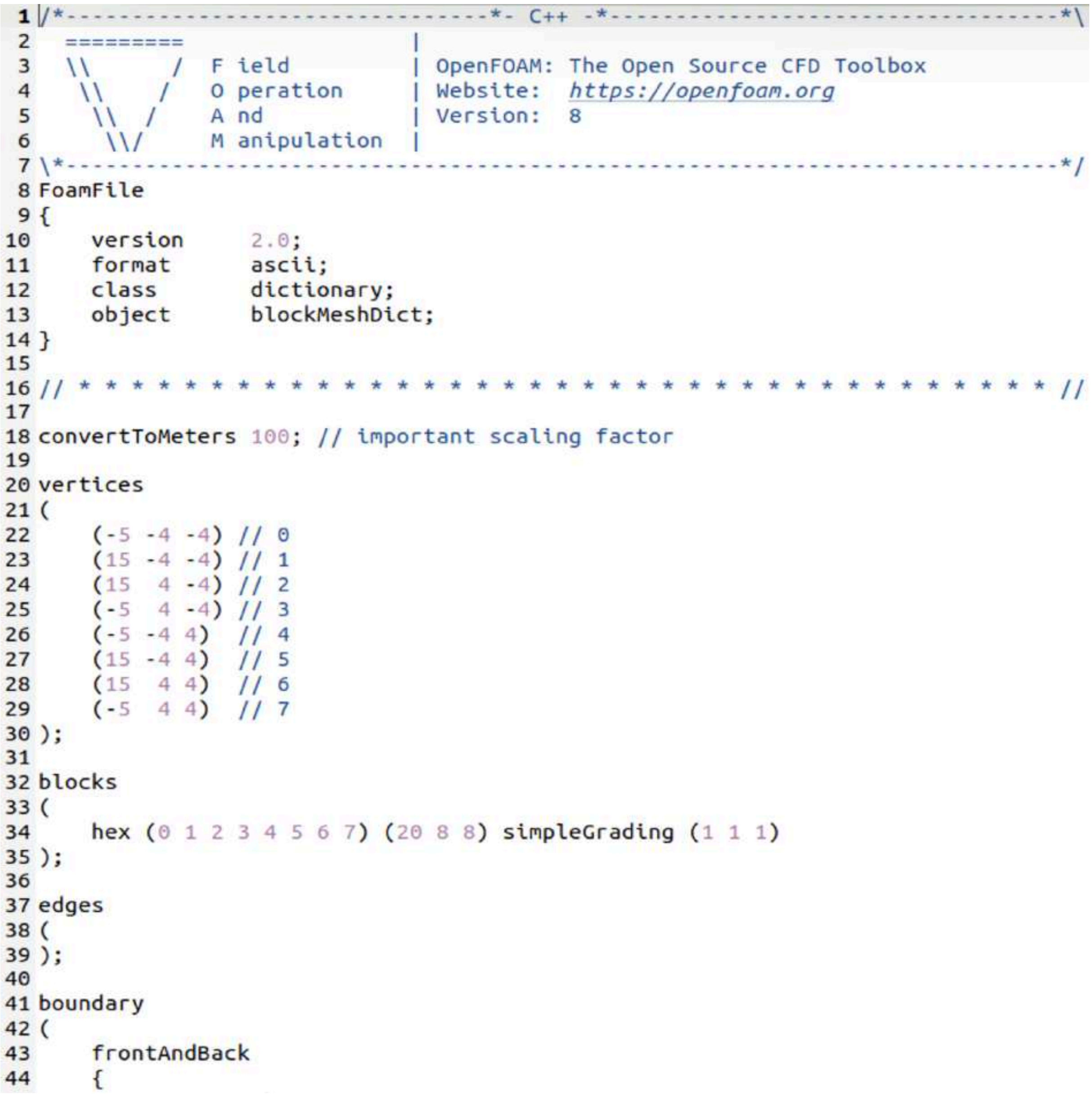
14 Appendix

14.1 Appendix A: Aerodynamic Modelling Functions and Description

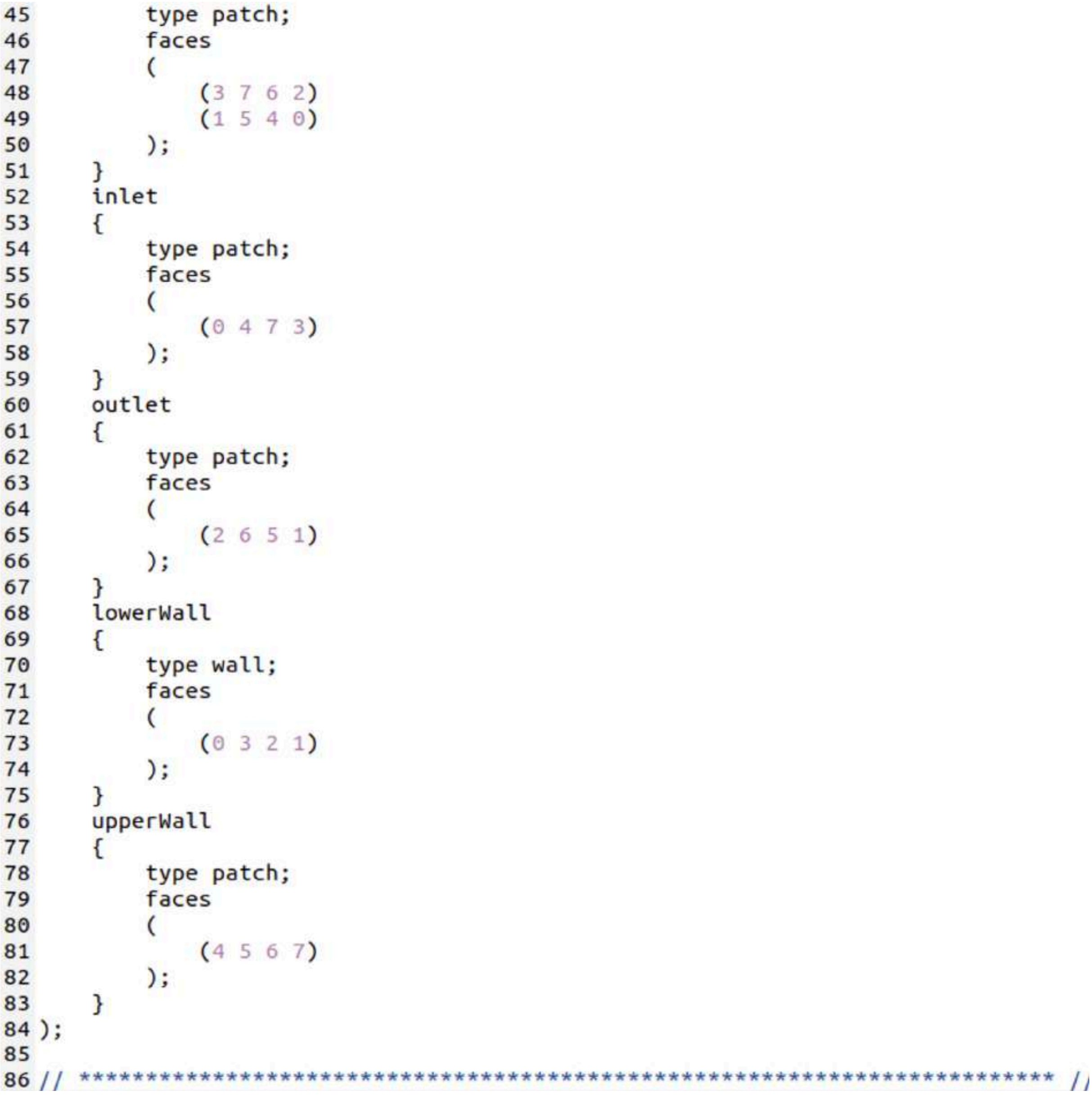
Symbol	Description	Function
Constants		
L	Reference length, Diameter of body tube	-
A_T	Planform area of one fin; area of fin when viewed from above	-
A_r	Reference area; cross-sectional area of body tube	-
S	Exposed fin semi-span; measured from root chord	-
r_L	Leading edge sweep angle	$r_L = 1.1019\frac{t_r^2}{c_r}$
A_{B_f}	Base area of one fin at root	-
c_r	Fin root chord	-
h_r	Root fin trailing edge thickness	-
t_r	Root fin thickness	-
f_b	Body fineness ratio	$f_b = \frac{Length\ of\ rocket}{Width\ of\ rocket}$
A_{W_B}	Total body wetted area	-
A_B	Base area at tail	-
A_{B_N}	Nose base area	-
A_{W_N}	Nose wetted area	-
A_W	Total wetted area	-
AR	Aspect Ratio of exposed fin	$AR = \frac{S^2}{A_r}$
A_f	Total area of one fin	-
User inputs		
u	Velocity	-
M	Mach Number	$M = \frac{V}{u_{sound}}$
ρ	Air density	-
μ	Air Viscosity	-
α	Angle of Attack	-
N	Number of fins; 3 or 4	-
R_e	Reynolds Number	$\frac{\rho u L}{\mu}$

C_D	Total Drag Coefficient	$C_D = C_{D_{T_T}} + C_{D_{B_T}}$
$C_{D_{T_T}}$	Tail Drag Coefficient	$C_{D_{T_T}} = C_{D_{f_T}} + C_{D_{B_T}} + C_{D_{L_T}} + C_{D_{T_T}}$
$C_{D_{f_T}}$	Tail skin friction drag coefficient	$2NC_{f_C}\frac{A_T}{A_r}$
$C_{D_{L_T}}$	Tail leading edge drag coefficient	$C_{D_{L_T}} = 2N\left(\frac{S_{r_L}}{A_r}\right)\cos^2\Gamma_L\left(\Delta C_D\right)$
$C_{D_{B_T}}$	Tail trailing edge drag coefficient	$C_{D_{B_T}} = \frac{0.135N\left(\frac{A_{B_f}}{A_r}\right)}{\sqrt[3]{C_{f_B}}\sqrt{K-M^2\cos^2\Gamma_c}}$
C_{DTT}	Tail thickness drag	$C_{D_{TT}} = 4NC_{f_c}\left(\frac{A_T}{A_T}\right)\left[\left(\frac{t_r}{c_r}\right)\cos^2\Gamma_c + \frac{30\left(\frac{t_r}{c_r}\right)^4\cos^2\Gamma_c}{\left(\sqrt{K-M^2\cos^2\Gamma_c}\right)^3}\right]$
$C_{D_{T_B}}$	Body component of drag	$C_{D_{T_B}} = C_{D_P} + C_{D_{BB}}$
C_{D_P}	Body pressure drag	$C_{D_P} = \frac{6A_{W_B}C_{f_c}}{f_R^3A_r(K-M^2)^{0.6}}$
$C_{D_{BB}}$	Body base drag	$C_{D_{BB}} = \frac{0.29\left(\frac{A_B}{A_r}\right)}{\sqrt{C_{f_c}\left(\frac{A_W}{A_B}\right)(K-M^2)}}$
Other Calculations		
C_f	Incompressible Skin coefficient drag	$C_f = \frac{1.328}{\sqrt{R_e}}$
C_{f_C}	Compressible Skin coefficient drag	$C_{f_C} = C_f(1-0.12M^2)$
ΔC_D	Drag coefficient of leading edge	$\Delta C_D = \left(1-M^2\right)^{-0.417} - 1$
C_{f_b}	Incompressible Skin friction coefficient drag	$C_{f_b} = 2C_f\frac{c_r}{h_r}$
K_1	Trailing edge drag correction constant	$K_1 = \left(1+\frac{18C_{fc}\left(\frac{t_r}{h_r}\right)^2}{N(1-0.52)\left(\frac{A_{Ef}}{A_r}\right)}\right)\left(\frac{0.135N\left(\frac{A_{E_f}}{A_r}\right)}{C_{fe}^{\frac{1}{3}}}\right) + \cos^2\Gamma_c$
K_{cone}	Body pressure drag correction constant	$K_{\text{cone}} = 1+\left[\frac{6.82A_{W_B}C_{f_c}(f_N+0.7)^{1.29}}{\left(f_B^3A_r\right)}\right]^{\frac{5}{3}}$
K_2	Body base drag correction constant	$K_2 = 1+\frac{1}{\left[6.38+39.7\left(\frac{h_r}{c_r}\right)\right]C_{f_c}\left(\frac{A_W}{A_B}\right)}$
B	Compressibility Factor	$B = \sqrt{1-M^2}$
X_{TB}	Longitudinal Tail center of pressure	-
X_{CG}	Location of CG from tip	-
Δx	Distance between aerodynamic center of tail and center of gravity	$\Delta x = X_{TB}$ - X_{CG}

14.2 Appendix B: OpenFOAM blockMeshDict Code



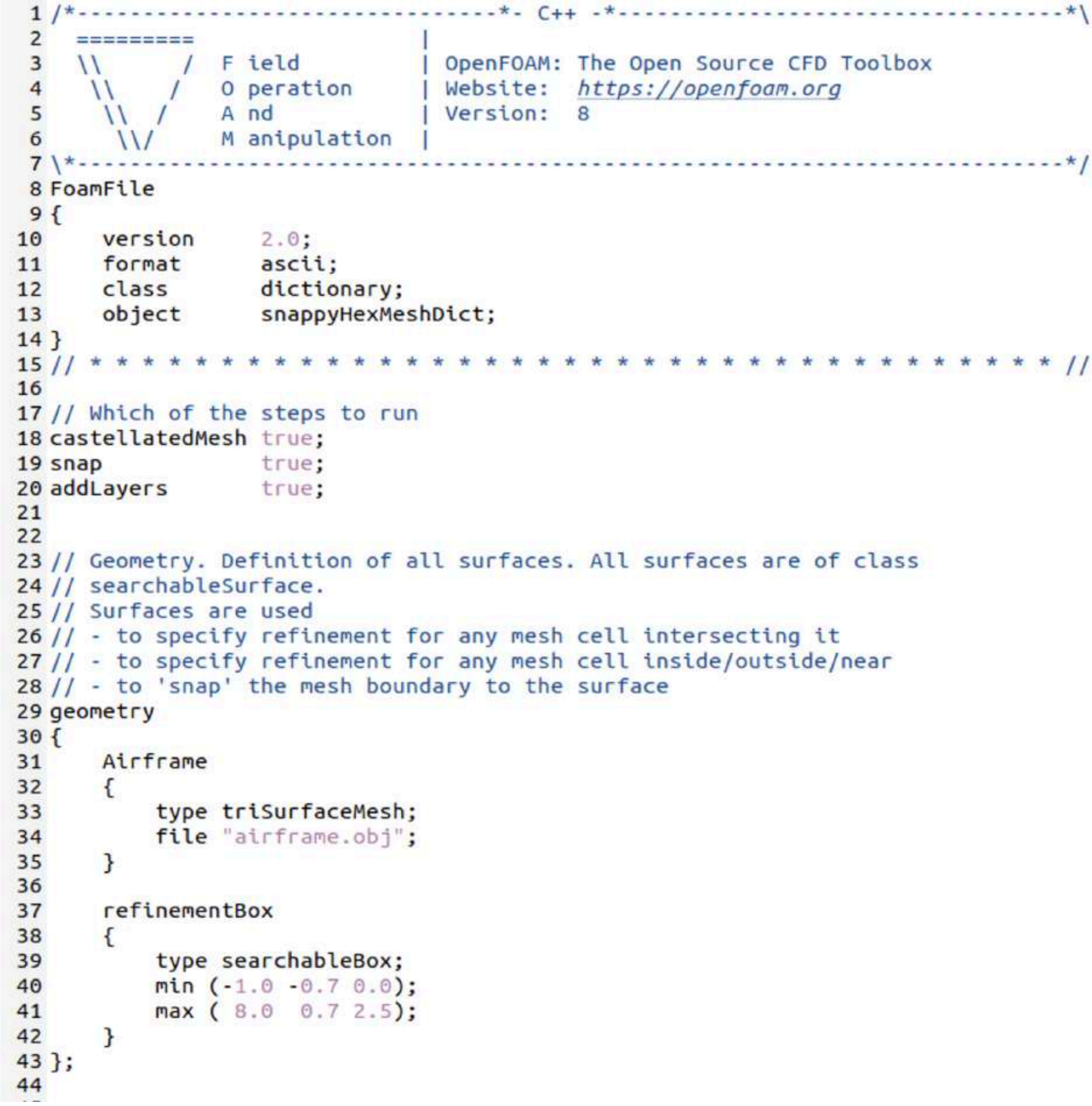
```
/*--------------------------------*- C++ -*----------------------------------*\
  =========                 |
  \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
   \\    /   O peration     | Website:  https://openfoam.org
    \\  /    A nd           | Version:  8
     \\/     M anipulation  |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      blockMeshDict;
}

// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

convertToMeters 100; // important scaling factor

vertices
(
    (-5 -4 -4) // 0
    (15 -4 -4) // 1
    (15  4 -4) // 2
    (-5  4 -4) // 3
    (-5 -4 4)  // 4
    (15 -4 4)  // 5
    (15  4 4)  // 6
    (-5  4 4)  // 7
);

blocks
(
    hex (0 1 2 3 4 5 6 7) (20 8 8) simpleGrading (1 1 1)
);

edges
(
);

boundary
(
    frontAndBack
    {
```



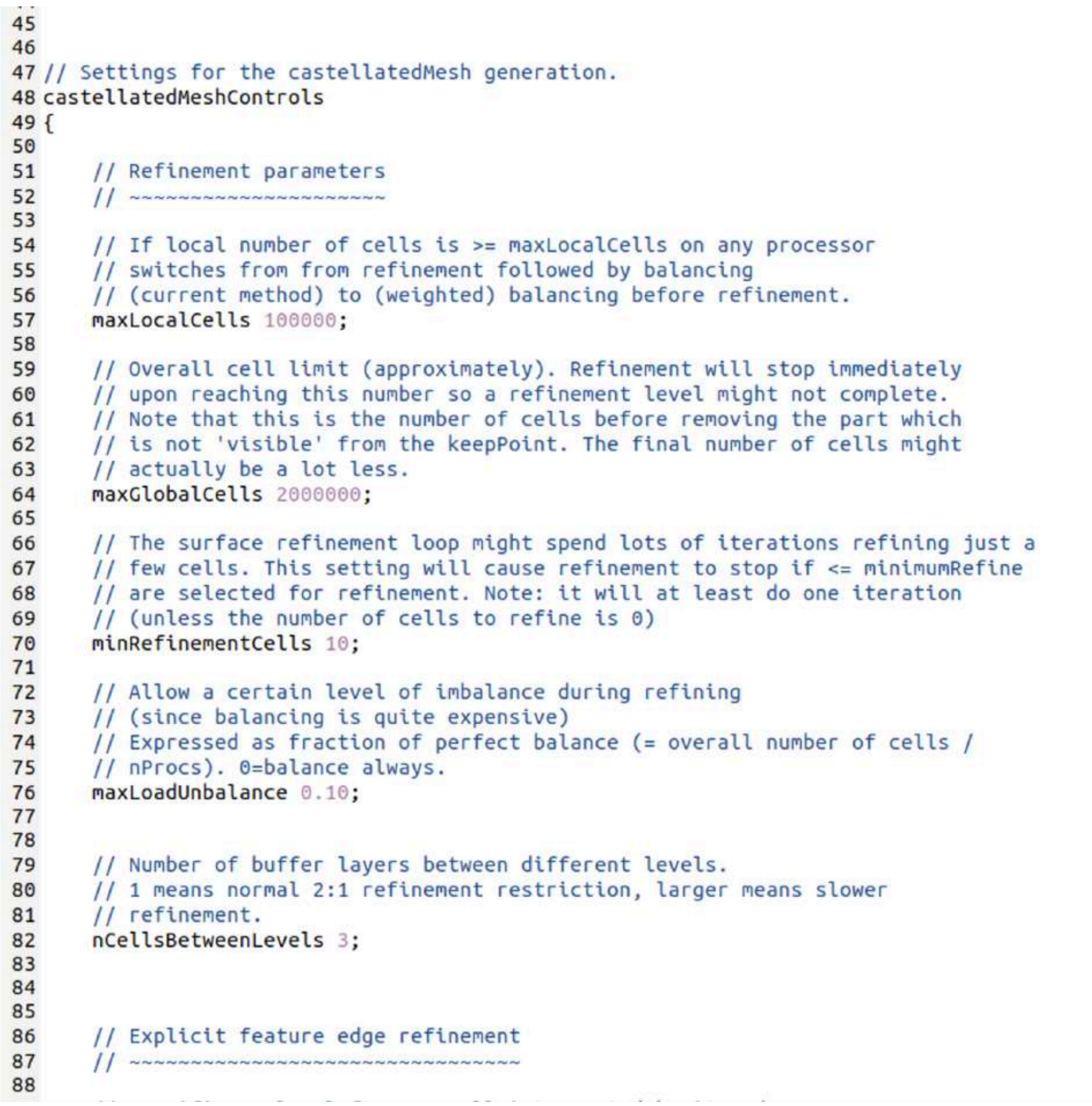
```
        type patch;
        faces
        (
            (3 7 6 2)
            (1 5 4 0)
        );
    }
    inlet
    {
        type patch;
        faces
        (
            (0 4 7 3)
        );
    }
    outlet
    {
        type patch;
        faces
        (
            (2 6 5 1)
        );
    }
    lowerWall
    {
        type wall;
        faces
        (
            (0 3 2 1)
        );
    }
    upperWall
    {
        type patch;
        faces
        (
            (4 5 6 7)
        );
    }
);

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

s

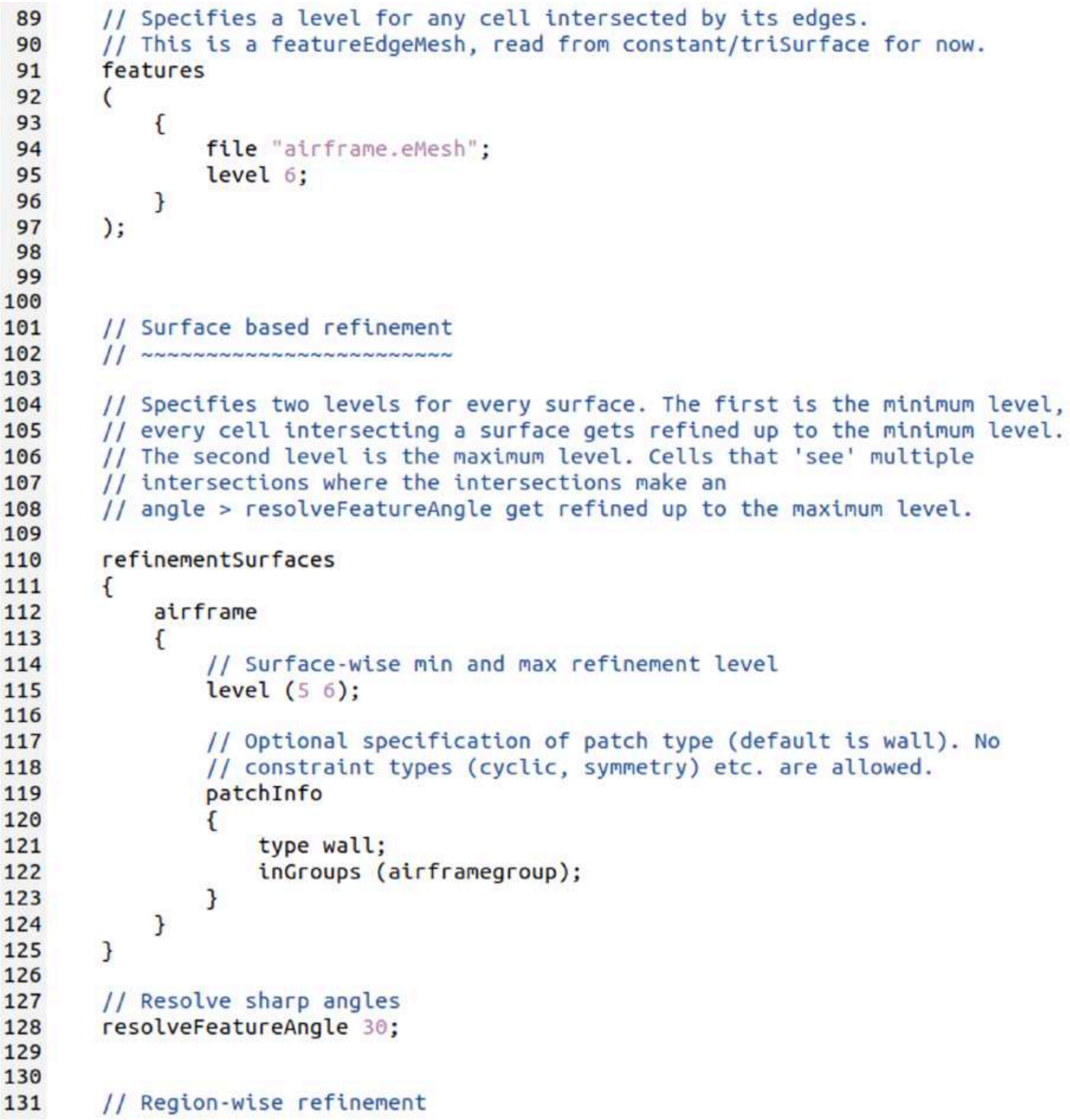
14.3 Appendix C: OpenFOAM snappyHexMesh Code



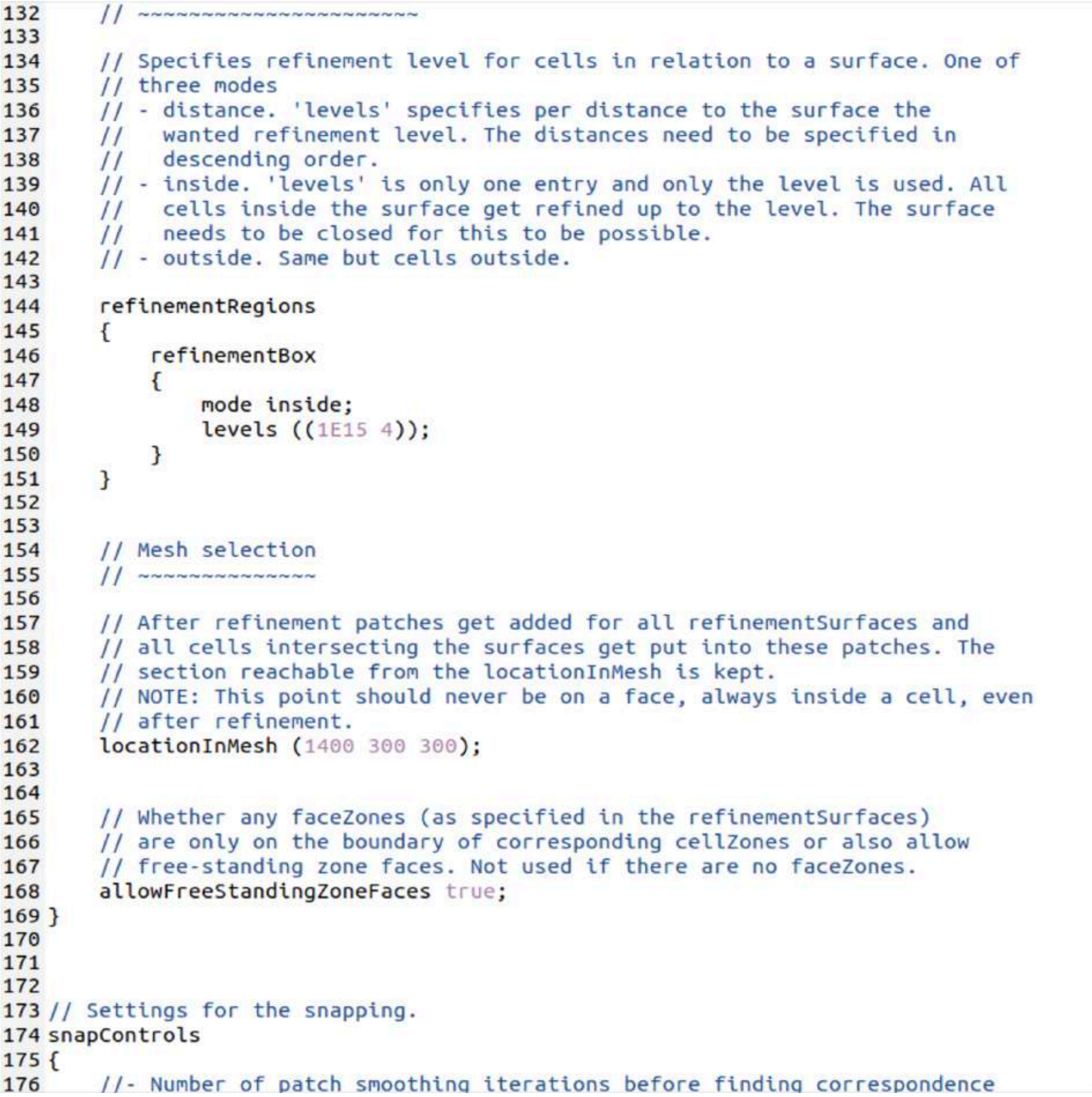
```
/*--------------------------------*- C++ -*----------------------------------*\
  =========                 |
  \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
   \\    /   O peration     | Website:  https://openfoam.org
    \\  /    A nd           | Version:  8
     \\/     M anipulation  |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      snappyHexMeshDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

// Which of the steps to run
castellatedMesh true;
snap            true;
addLayers       true;


// Geometry. Definition of all surfaces. All surfaces are of class
// searchableSurface.
// Surfaces are used
// - to specify refinement for any mesh cell intersecting it
// - to specify refinement for any mesh cell inside/outside/near
// - to 'snap' the mesh boundary to the surface
geometry
{
    Airframe
    {
        type triSurfaceMesh;
        file "airframe.obj";
    }

    refinementBox
    {
        type searchableBox;
        min (-1.0 -0.7 0.0);
        max ( 8.0  0.7 2.5);
    }
};

```



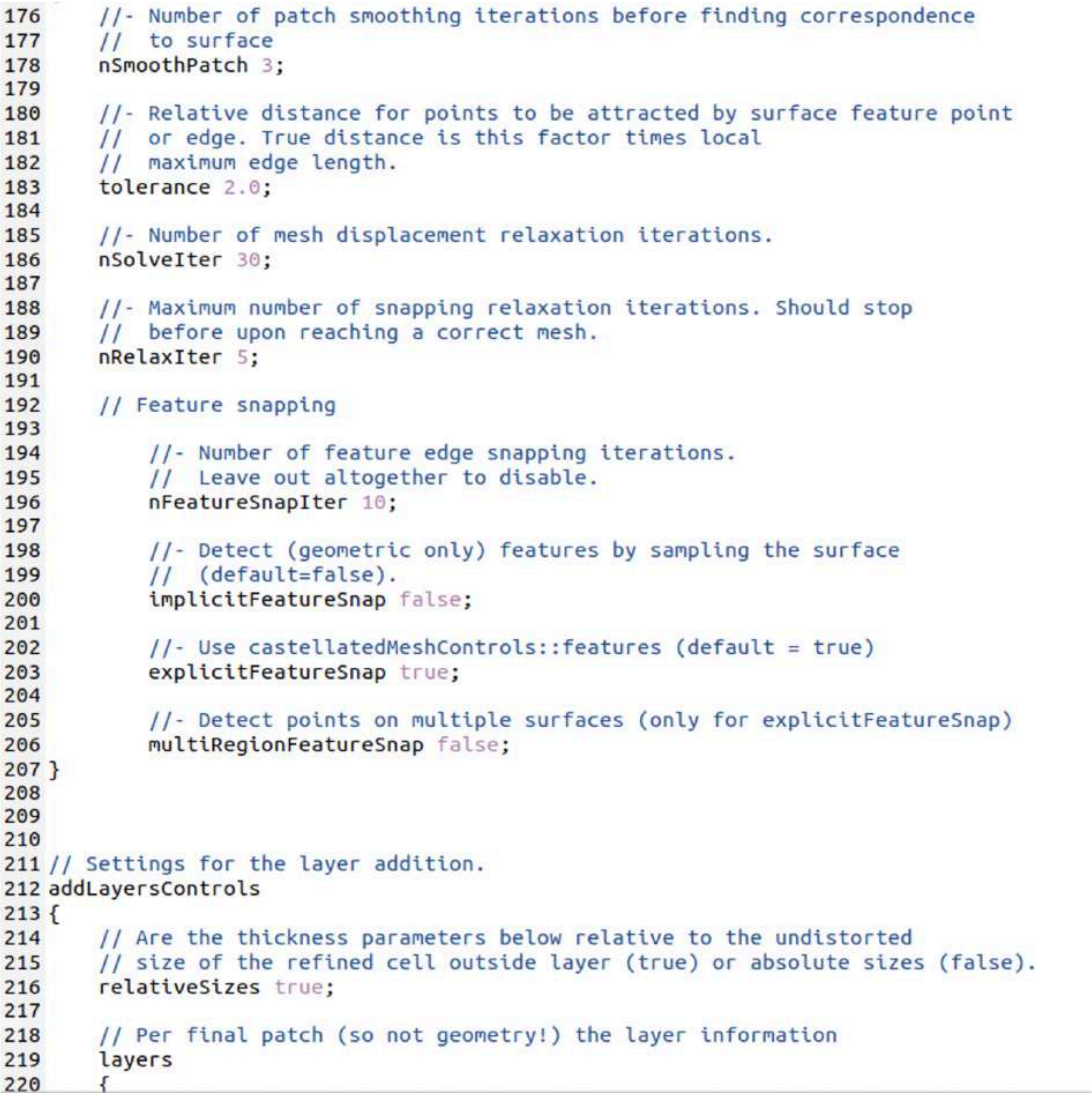
```


// Settings for the castellatedMesh generation.
castellatedMeshControls
{

    // Refinement parameters
    // ~~~~~~~~~~~~~~~~~~~~~

    // If local number of cells is >= maxLocalCells on any processor
    // switches from from refinement followed by balancing
    // (current method) to (weighted) balancing before refinement.
    maxLocalCells 100000;

    // Overall cell limit (approximately). Refinement will stop immediately
    // upon reaching this number so a refinement level might not complete.
    // Note that this is the number of cells before removing the part which
    // is not 'visible' from the keepPoint. The final number of cells might
    // actually be a lot less.
    maxGlobalCells 2000000;

    // The surface refinement loop might spend lots of iterations refining just a
    // few cells. This setting will cause refinement to stop if <= minimumRefine
    // are selected for refinement. Note: it will at least do one iteration
    // (unless the number of cells to refine is 0)
    minRefinementCells 10;

    // Allow a certain level of imbalance during refining
    // (since balancing is quite expensive)
    // Expressed as fraction of perfect balance (= overall number of cells /
    // nProcs). 0=balance always.
    maxLoadUnbalance 0.10;


    // Number of buffer layers between different levels.
    // 1 means normal 2:1 refinement restriction, larger means slower
    // refinement.
    nCellsBetweenLevels 3;



    // Explicit feature edge refinement
    // ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

```



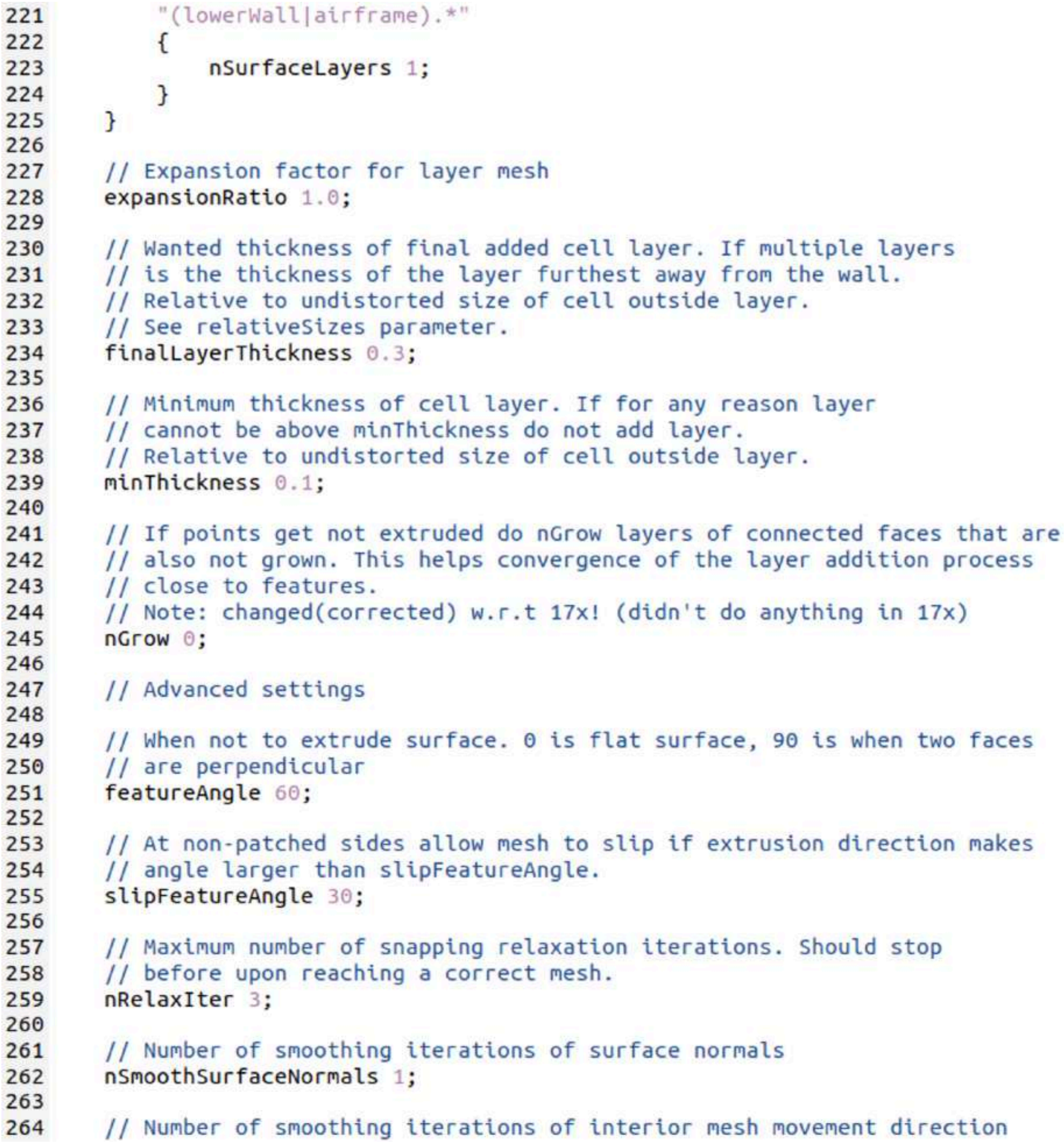
```
// Specifies a level for any cell intersected by its edges.
// This is a featureEdgeMesh, read from constant/triSurface for now.
features
(
    {
        file "airframe.eMesh";
        level 6;
    }
);



// Surface based refinement
// ~~~~~~~~~~~~~~~~~~~~~~~~

// Specifies two levels for every surface. The first is the minimum level,
// every cell intersecting a surface gets refined up to the minimum level.
// The second level is the maximum level. Cells that 'see' multiple
// intersections where the intersections make an
// angle > resolveFeatureAngle get refined up to the maximum level.

refinementSurfaces
{
    airframe
    {
        // Surface-wise min and max refinement level
        level (5 6);

        // Optional specification of patch type (default is wall). No
        // constraint types (cyclic, symmetry) etc. are allowed.
        patchInfo
        {
            type wall;
            inGroups (airframegroup);
        }
    }
}

// Resolve sharp angles
resolveFeatureAngle 30;


// Region-wise refinement
```



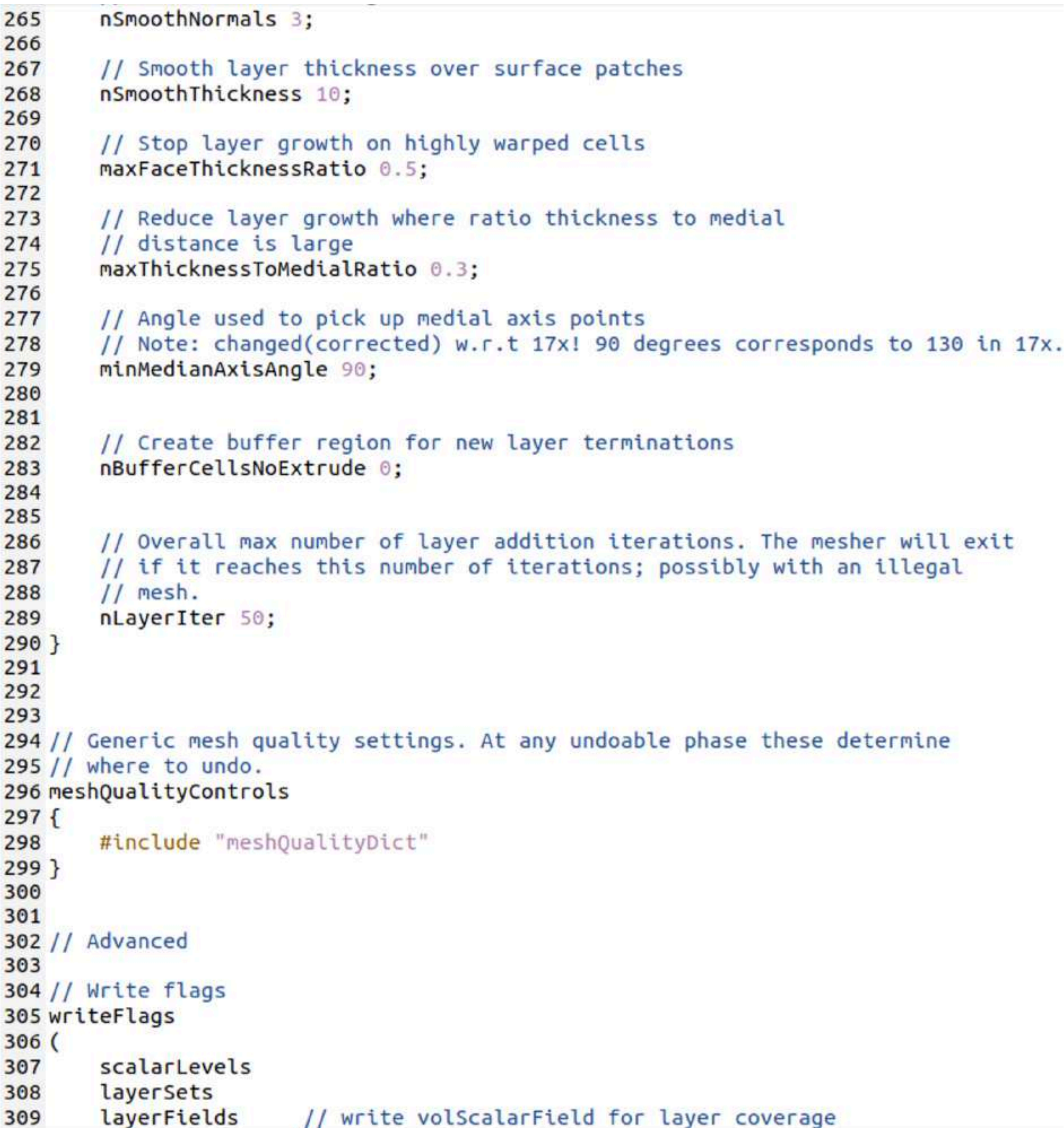
```
    // ~~~~~~~~~~~~~~~~~~~~~~

    // Specifies refinement level for cells in relation to a surface. One of
    // three modes
    // - distance. 'levels' specifies per distance to the surface the
    //   wanted refinement level. The distances need to be specified in
    //   descending order.
    // - inside. 'levels' is only one entry and only the level is used. All
    //   cells inside the surface get refined up to the level. The surface
    //   needs to be closed for this to be possible.
    // - outside. Same but cells outside.

    refinementRegions
    {
        refinementBox
        {
            mode inside;
            levels ((1E15 4));
        }
    }


    // Mesh selection
    // ~~~~~~~~~~~~~~

    // After refinement patches get added for all refinementSurfaces and
    // all cells intersecting the surfaces get put into these patches. The
    // section reachable from the locationInMesh is kept.
    // NOTE: This point should never be on a face, always inside a cell, even
    // after refinement.
    locationInMesh (1400 300 300);


    // Whether any faceZones (as specified in the refinementSurfaces)
    // are only on the boundary of corresponding cellZones or also allow
    // free-standing zone faces. Not used if there are no faceZones.
    allowFreeStandingZoneFaces true;
}



// Settings for the snapping.
snapControls
{
    //- Number of patch smoothing iterations before finding correspondence
```



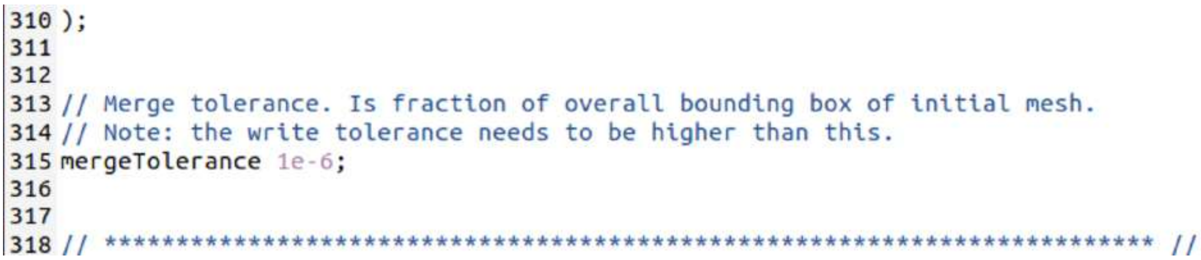
```
    //- Number of patch smoothing iterations before finding correspondence
    //  to surface
    nSmoothPatch 3;

    //- Relative distance for points to be attracted by surface feature point
    //  or edge. True distance is this factor times local
    //  maximum edge length.
    tolerance 2.0;

    //- Number of mesh displacement relaxation iterations.
    nSolveIter 30;

    //- Maximum number of snapping relaxation iterations. Should stop
    //  before upon reaching a correct mesh.
    nRelaxIter 5;

    // Feature snapping

        //- Number of feature edge snapping iterations.
        //  Leave out altogether to disable.
        nFeatureSnapIter 10;

        //- Detect (geometric only) features by sampling the surface
        //  (default=false).
        implicitFeatureSnap false;

        //- Use castellatedMeshControls::features (default = true)
        explicitFeatureSnap true;

        //- Detect points on multiple surfaces (only for explicitFeatureSnap)
        multiRegionFeatureSnap false;
}



// Settings for the layer addition.
addLayersControls
{
    // Are the thickness parameters below relative to the undistorted
    // size of the refined cell outside layer (true) or absolute sizes (false).
    relativeSizes true;

    // Per final patch (so not geometry!) the layer information
    layers
    {
```



```
        "(lowerWall|airframe).*"
        {
            nSurfaceLayers 1;
        }
    }

    // Expansion factor for layer mesh
    expansionRatio 1.0;

    // Wanted thickness of final added cell layer. If multiple layers
    // is the thickness of the layer furthest away from the wall.
    // Relative to undistorted size of cell outside layer.
    // See relativeSizes parameter.
    finalLayerThickness 0.3;

    // Minimum thickness of cell layer. If for any reason layer
    // cannot be above minThickness do not add layer.
    // Relative to undistorted size of cell outside layer.
    minThickness 0.1;

    // If points get not extruded do nGrow layers of connected faces that are
    // also not grown. This helps convergence of the layer addition process
    // close to features.
    // Note: changed(corrected) w.r.t 17x! (didn't do anything in 17x)
    nGrow 0;

    // Advanced settings

    // When not to extrude surface. 0 is flat surface, 90 is when two faces
    // are perpendicular
    featureAngle 60;

    // At non-patched sides allow mesh to slip if extrusion direction makes
    // angle larger than slipFeatureAngle.
    slipFeatureAngle 30;

    // Maximum number of snapping relaxation iterations. Should stop
    // before upon reaching a correct mesh.
    nRelaxIter 3;

    // Number of smoothing iterations of surface normals
    nSmoothSurfaceNormals 1;

    // Number of smoothing iterations of interior mesh movement direction
```

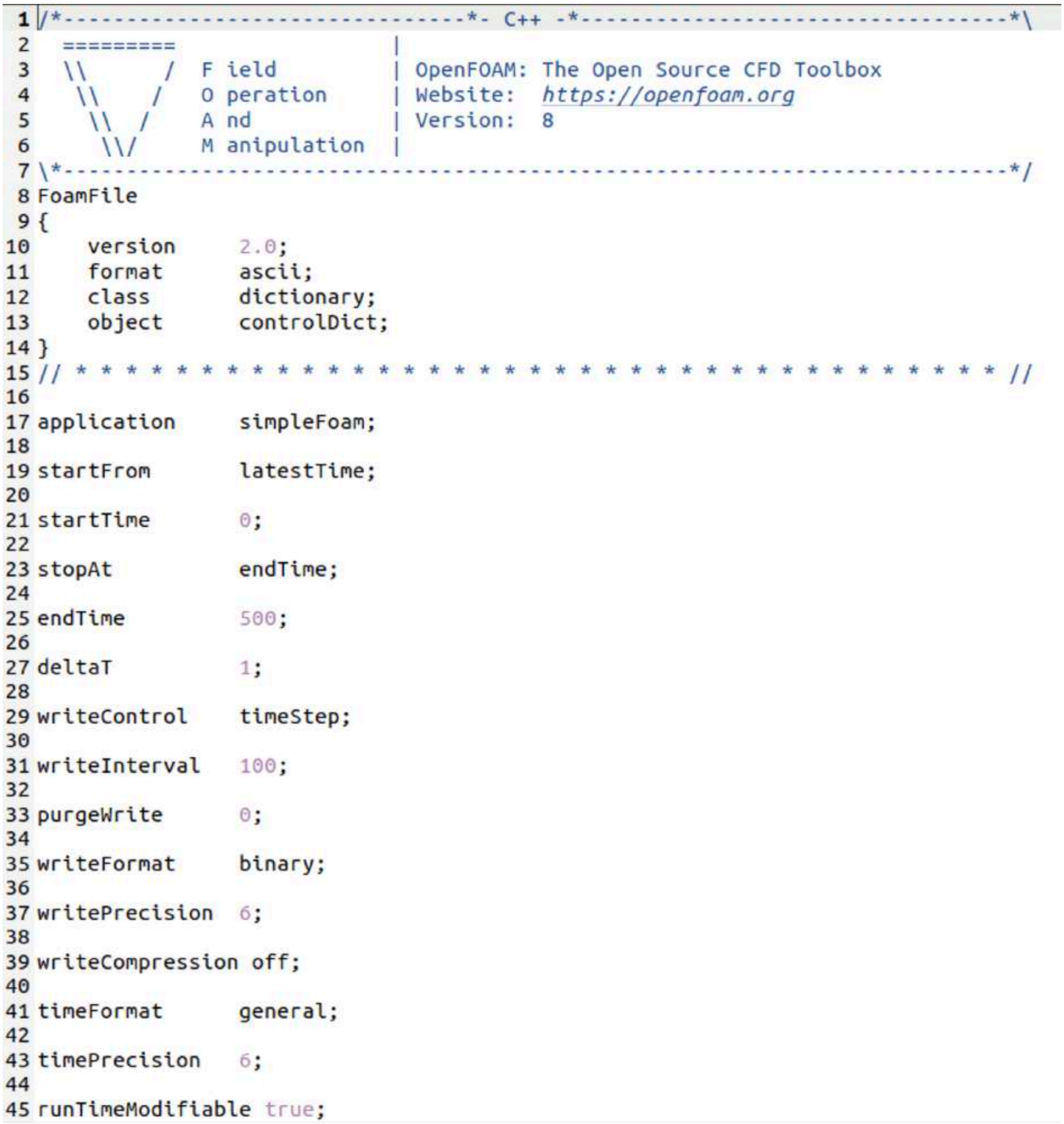


```
    nSmoothNormals 3;

    // Smooth layer thickness over surface patches
    nSmoothThickness 10;

    // Stop layer growth on highly warped cells
    maxFaceThicknessRatio 0.5;

    // Reduce layer growth where ratio thickness to medial
    // distance is large
    maxThicknessToMedialRatio 0.3;

    // Angle used to pick up medial axis points
    // Note: changed(corrected) w.r.t 17x! 90 degrees corresponds to 130 in 17x.
    minMedianAxisAngle 90;


    // Create buffer region for new layer terminations
    nBufferCellsNoExtrude 0;


    // Overall max number of layer addition iterations. The mesher will exit
    // if it reaches this number of iterations; possibly with an illegal
    // mesh.
    nLayerIter 50;
}



// Generic mesh quality settings. At any undoable phase these determine
// where to undo.
meshQualityControls
{
    #include "meshQualityDict"
}


// Advanced

// Write flags
writeFlags
(
    scalarLevels
    layerSets
    layerFields     // write volScalarField for layer coverage
```

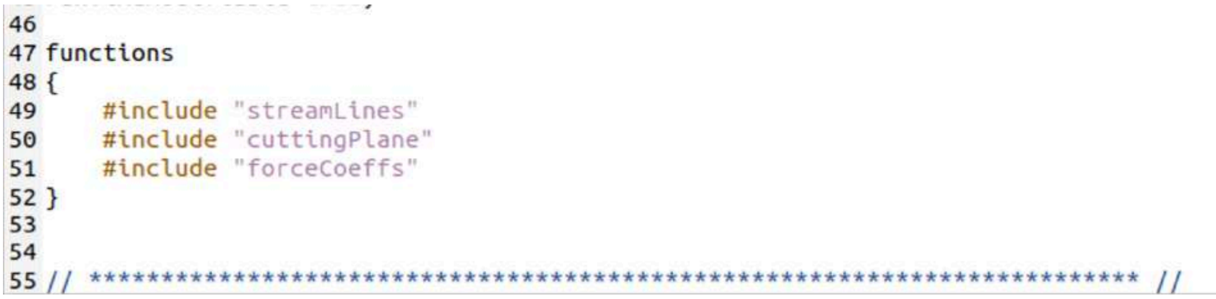


```
);


// Merge tolerance. Is fraction of overall bounding box of initial mesh.
// Note: the write tolerance needs to be higher than this.
mergeTolerance 1e-6;


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

14.4 Appendix D: OpenFOAM controlDict Code

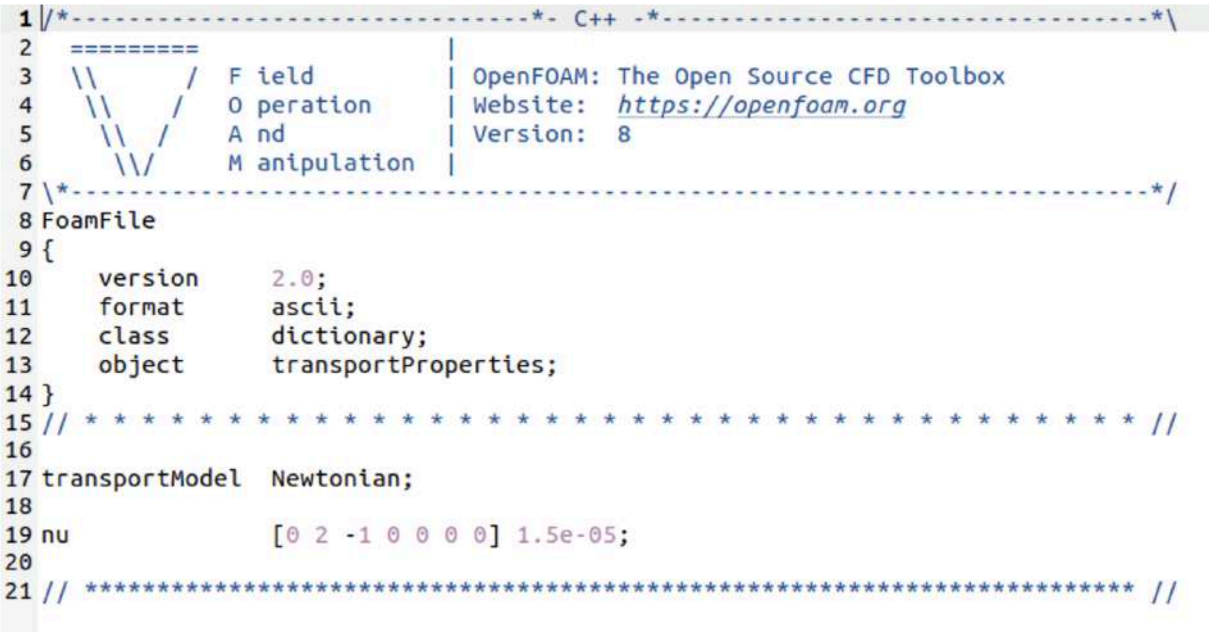


```
/*--------------------------------*- C++ -*----------------------------------*\
  =========                 |
  \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
   \\    /   O peration     | Website:  https://openfoam.org
    \\  /    A nd           | Version:  8
     \\/     M anipulation  |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      controlDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

application     simpleFoam;

startFrom       latestTime;

startTime       0;

stopAt          endTime;

endTime         500;

deltaT          1;

writeControl    timeStep;

writeInterval   100;

purgeWrite      0;

writeFormat     binary;

writePrecision  6;

writeCompression off;

timeFormat      general;

timePrecision   6;

runTimeModifiable true;
```



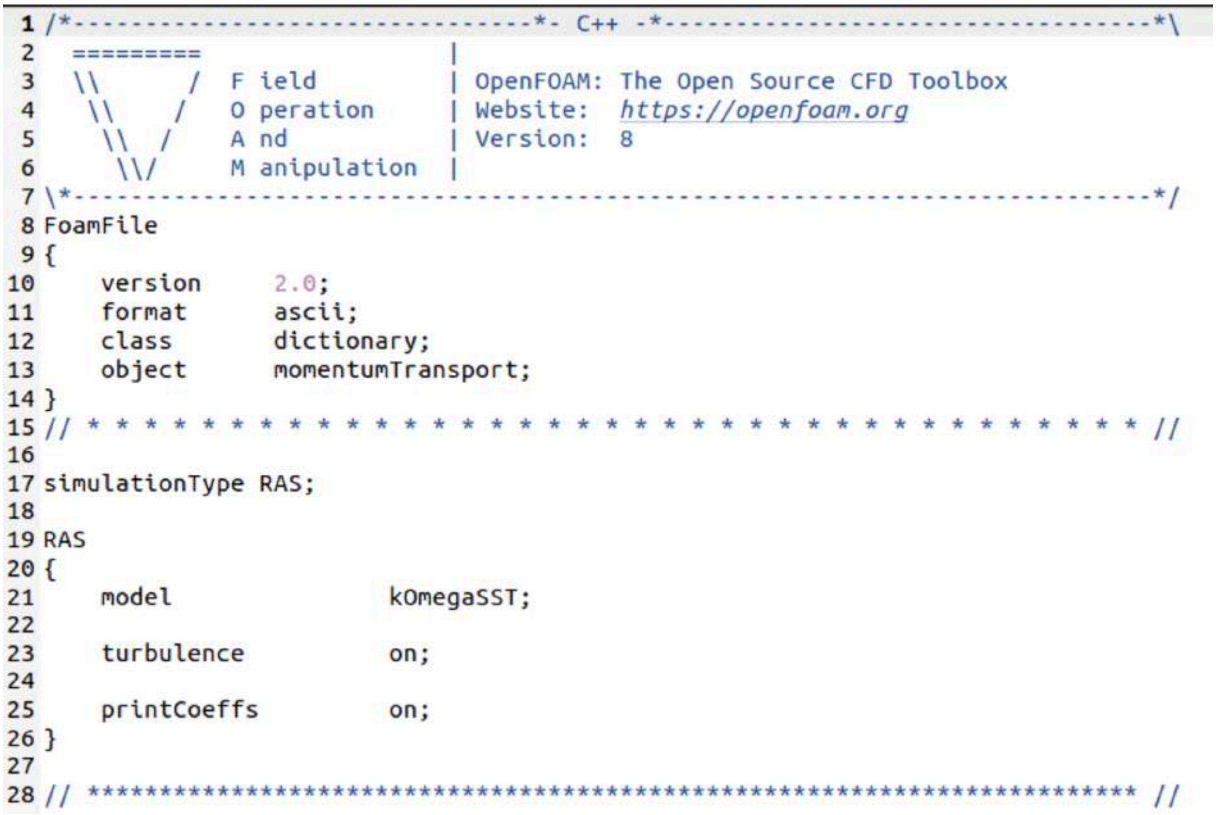
```

functions
{
    #include "streamLines"
    #include "cuttingPlane"
    #include "forceCoeffs"
}


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

14.5 Appendix E: OpenFOAM Transport Properties Code



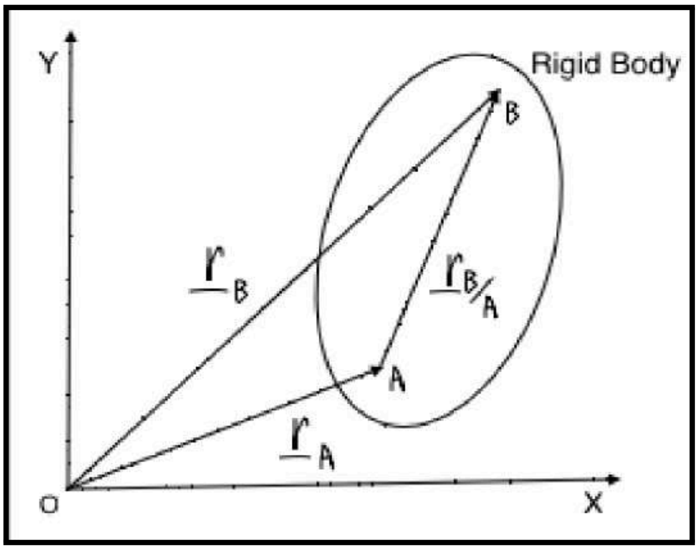
```
/*--------------------------------*- C++ -*----------------------------------*\
  =========                 |
  \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
   \\    /   O peration     | Website:  https://openfoam.org
    \\  /    A nd           | Version:  8
     \\/     M anipulation  |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      transportProperties;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

transportModel  Newtonian;

nu              [0 2 -1 0 0 0 0] 1.5e-05;

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

14.6 Appendix F: OpenFOAM Momentum Properties Code



```
/*--------------------------------*- C++ -*----------------------------------*\
  =========                 |
  \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
   \\    /   O peration     | Website:  https://openfoam.org
    \\  /    A nd           | Version:  8
     \\/     M anipulation  |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      momentumTransport;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

simulationType RAS;

RAS
{
    model               kOmegaSST;

    turbulence          on;

    printCoeffs         on;
}

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

14.7 Appendix G: Transport Theorem / Equation of Coriolis Derivation



$$r_B = r_A + r_{B/A} \tag{71}$$

Differentiating r with respect to time,

$$\dot{r}_B = \dot{r}_A + \dot{r}_{B/A} \tag{72}$$

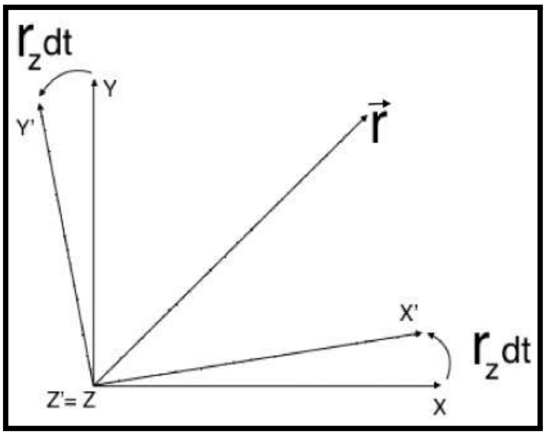
We know that,

$$\dot{r}_{B/A} = \omega \times r_{B/A} \tag{73}$$

Substituting equation 73 in equation 72,

$$\begin{aligned} \dot{r}_B &= \dot{r}_A + \omega \times r_{B/A} \\ V_B &= V_A + \omega \times r_{B/A} \end{aligned} \tag{74}$$

14.8 Appendix H: Cross-Product Matrix Derivation $[r\times]$



$$\text{Let } \vec{r} = r_x\hat{\imath} + r_y\hat{\jmath} + r_z\hat{k} \tag{75}$$

Rotating $\vec{r}$ about Z axis:

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} \cos(r_z dt) & \sin(r_z dt) & 0 \\ -\sin(r_z dt) & \cos(r_z dt) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

Inverting the above matrix,

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} \cos(r_z dt) & -\sin(r_z dt) & 0 \\ \sin(r_z dt) & \cos(r_z dt) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x' \\ y' \\ z' \end{bmatrix}$$

Small angle approximation:

$$\cos(\theta) \approx 1, \sin(\theta) \approx \theta$$

$$\cos(r_z dt) \approx 1, \sin(r_z dt) \approx r_z dt$$

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 1 & -r_z dt & 0 \\ r_z dt & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x' \\ y' \\ z' \end{bmatrix}$$

Similarly, rotating about Y axis:

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 1 & 0 & r_y dt \\ 0 & 1 & 0 \\ -r_y dt & 0 & 1 \end{bmatrix} \begin{bmatrix} x' \\ y' \\ z' \end{bmatrix}$$

Rotating about X axis:

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & -r_x dt \\ 0 & r_x dt & 1 \end{bmatrix} \begin{bmatrix} x' \\ y' \\ z' \end{bmatrix}$$

Differentiating the above matrices with respect to time,

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \end{bmatrix} = \begin{bmatrix} 0 & -r_z & 0 \\ r_z & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \end{bmatrix} = \begin{bmatrix} 0 & 0 & r_y \\ 0 & 0 & 0 \\ -r_y & 0 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & -r_x \\ 0 & r_x & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

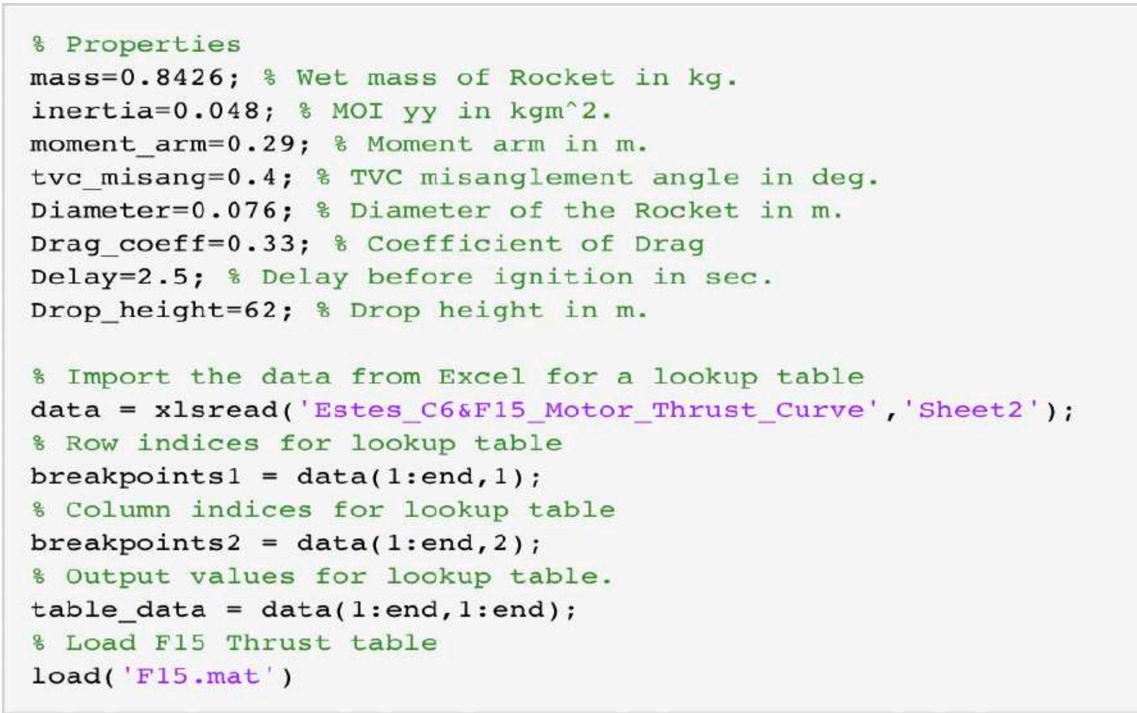
Combining all 3 equations,

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \end{bmatrix} = \begin{bmatrix} 0 & -r_z & r_y \\ r_z & 0 & -r_x \\ -r_y & r_x & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

Therefore, for any vector $\vec{r}$, Cross-Product matrix:

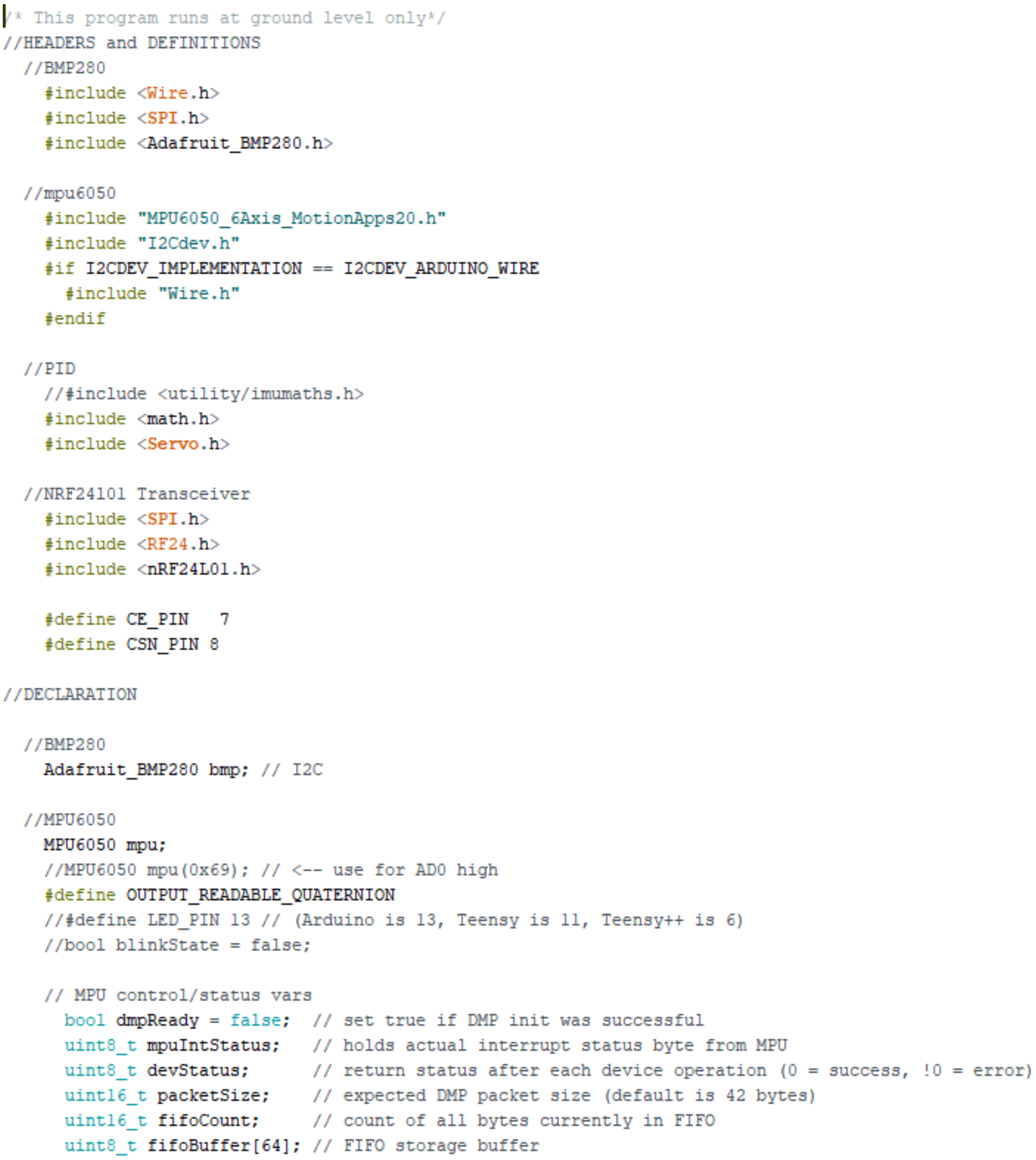
$$[r\times] = \begin{bmatrix} 0 & -r_z & r_y \\ r_z & 0 & -r_x \\ -r_y & r_x & 0 \end{bmatrix}$$

14.9 Appendix I: MATLAB Code - Data for Simulation

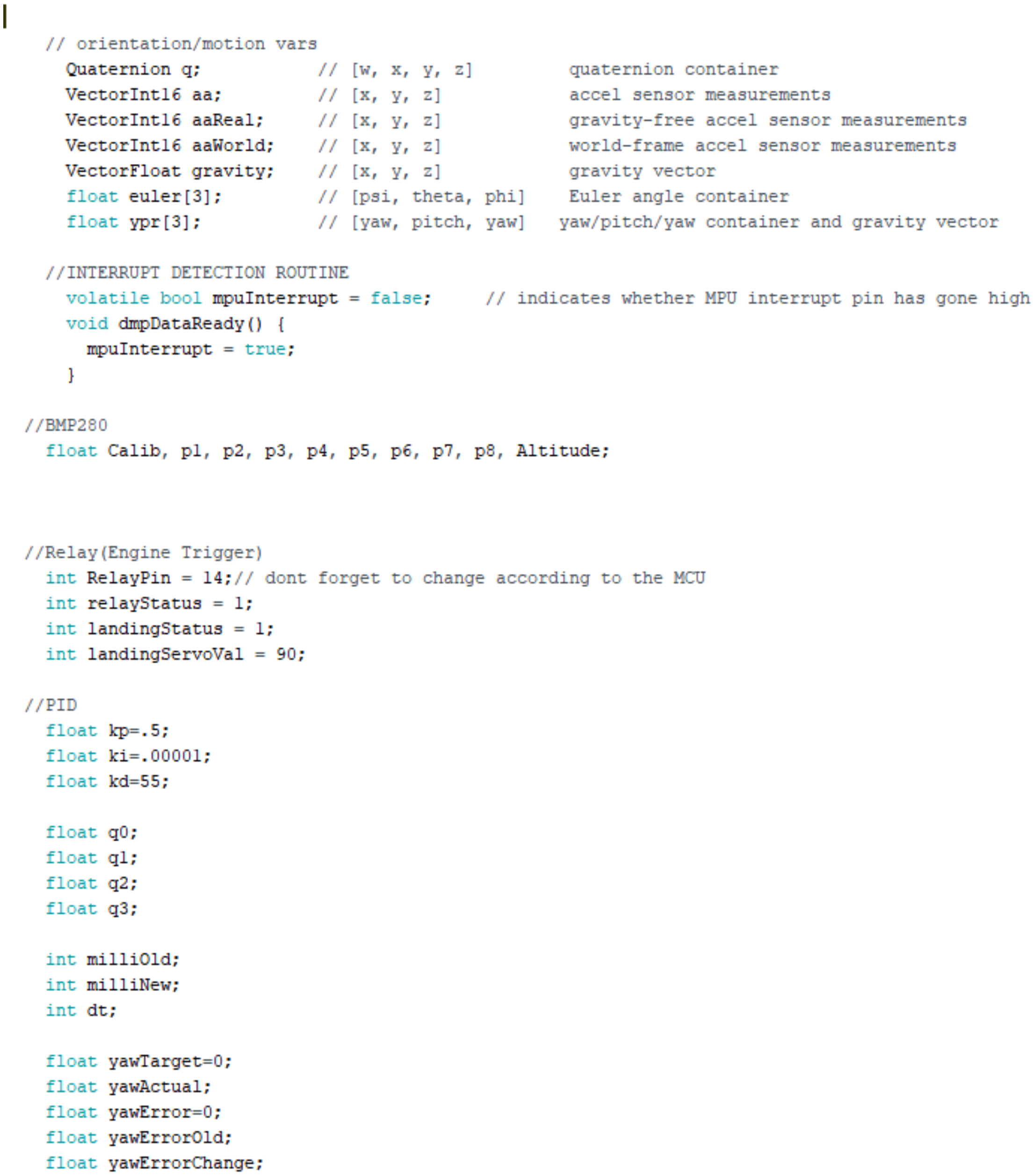


```
% Properties
mass=0.8426; % Wet mass of Rocket in kg.
inertia=0.048; % MOI yy in kgm^2.
moment_arm=0.29; % Moment arm in m.
tvc_misang=0.4; % TVC misanglement angle in deg.
Diameter=0.076; % Diameter of the Rocket in m.
Drag_coeff=0.33; % Coefficient of Drag
Delay=2.5; % Delay before ignition in sec.
Drop_height=62; % Drop height in m.

% Import the data from Excel for a lookup table
data = xlsread('Estes_C6&F15_Motor_Thrust_Curve','Sheet2');
% Row indices for lookup table
breakpoints1 = data(1:end,1);
% Column indices for lookup table
breakpoints2 = data(1:end,2);
% Output values for lookup table.
table_data = data(1:end,1:end);
% Load F15 Thrust table
load('F15.mat')
```

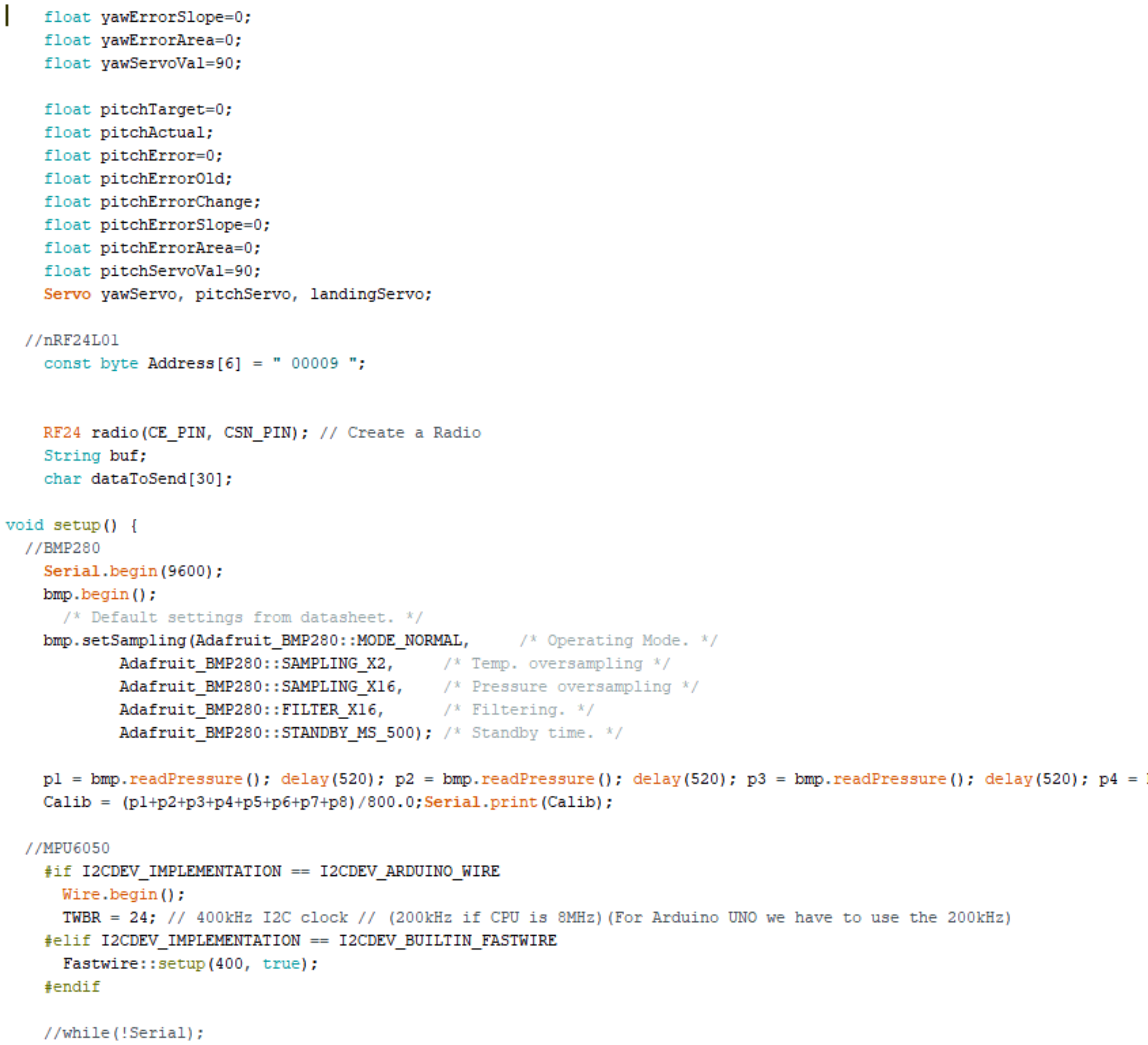
14.10 Appendix J: Arduino Flight Computer Code



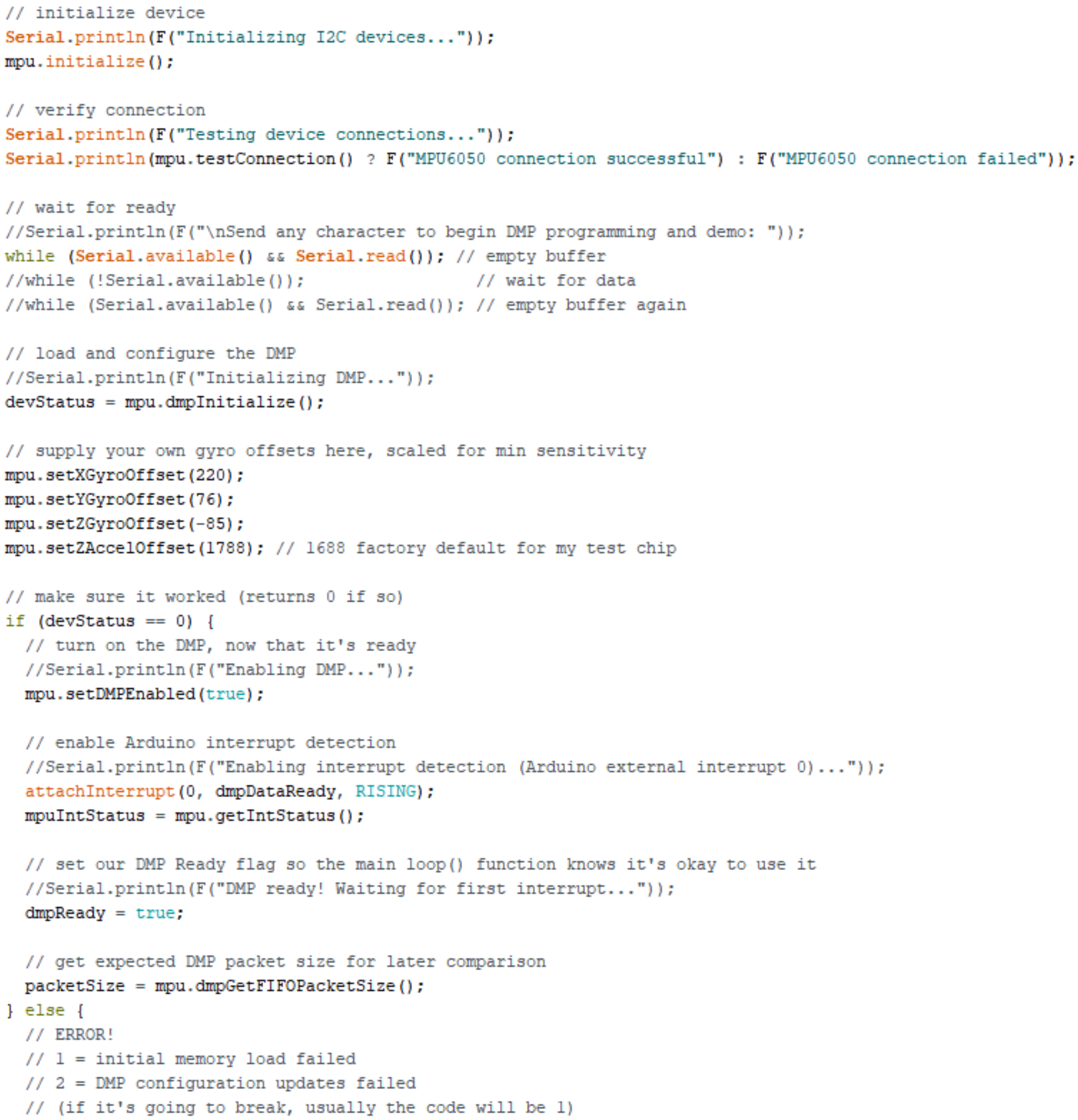
```
/* This program runs at ground level only*/
//HEADERS and DEFINITIONS
  //BMP280
    #include <Wire.h>
    #include <SPI.h>
    #include <Adafruit_BMP280.h>

  //mpu6050
    #include "MPU6050_6Axis_MotionApps20.h"
    #include "I2Cdev.h"
    #if I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE
      #include "Wire.h"
    #endif

  //PID
    //#include <utility/imumaths.h>
    #include <math.h>
    #include <Servo.h>

  //NRF24101 Transceiver
    #include <SPI.h>
    #include <RF24.h>
    #include <nRF24L01.h>

    #define CE_PIN   7
    #define CSN_PIN 8

//DECLARATION

  //BMP280
    Adafruit_BMP280 bmp; // I2C

  //MPU6050
    MPU6050 mpu;
    //MPU6050 mpu(0x69); // <-- use for AD0 high
    #define OUTPUT_READABLE_QUATERNION
    //#define LED_PIN 13 // (Arduino is 13, Teensy is 11, Teensy++ is 6)
    //bool blinkState = false;

    // MPU control/status vars
      bool dmpReady = false;  // set true if DMP init was successful
      uint8_t mpuIntStatus;   // holds actual interrupt status byte from MPU
      uint8_t devStatus;      // return status after each device operation (0 = success, !0 = error)
      uint16_t packetSize;    // expected DMP packet size (default is 42 bytes)
      uint16_t fifoCount;     // count of all bytes currently in FIFO
      uint8_t fifoBuffer[64]; // FIFO storage buffer
```



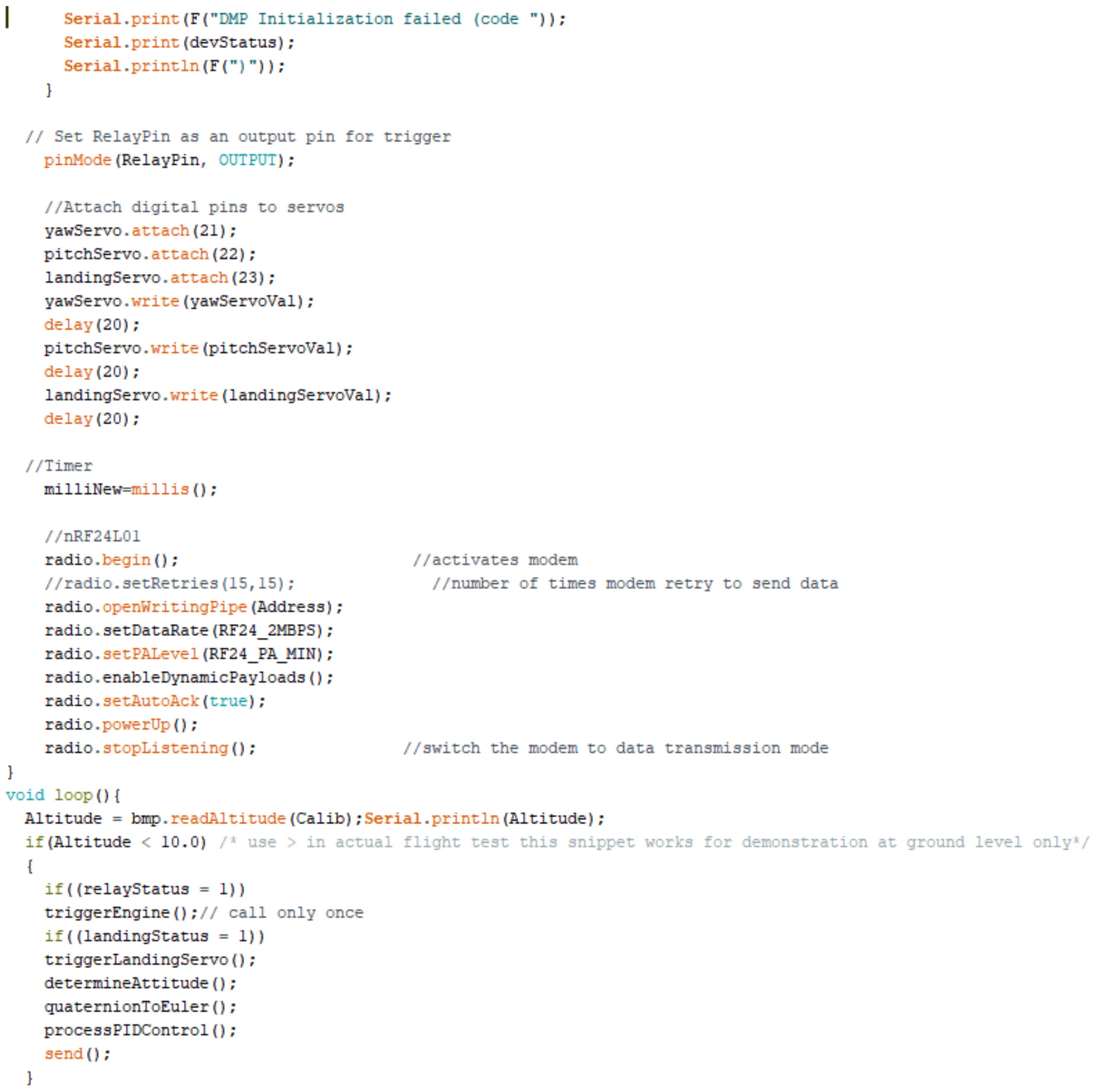
```
    // orientation/motion vars
      Quaternion q;           // [w, x, y, z]         quaternion container
      VectorInt16 aa;         // [x, y, z]            accel sensor measurements
      VectorInt16 aaReal;     // [x, y, z]            gravity-free accel sensor measurements
      VectorInt16 aaWorld;    // [x, y, z]            world-frame accel sensor measurements
      VectorFloat gravity;    // [x, y, z]            gravity vector
      float euler[3];         // [psi, theta, phi]    Euler angle container
      float ypr[3];           // [yaw, pitch, yaw]   yaw/pitch/yaw container and gravity vector

    //INTERRUPT DETECTION ROUTINE
      volatile bool mpuInterrupt = false;     // indicates whether MPU interrupt pin has gone high
      void dmpDataReady() {
        mpuInterrupt = true;
      }

  //BMP280
    float Calib, p1, p2, p3, p4, p5, p6, p7, p8, Altitude;



  //Relay(Engine Trigger)
    int RelayPin = 14;// dont forget to change according to the MCU
    int relayStatus = 1;
    int landingStatus = 1;
    int landingServoVal = 90;

  //PID
    float kp=.5;
    float ki=.00001;
    float kd=55;

    float q0;
    float q1;
    float q2;
    float q3;

    int milliOld;
    int milliNew;
    int dt;

    float yawTarget=0;
    float yawActual;
    float yawError=0;
    float yawErrorOld;
    float yawErrorChange;
```



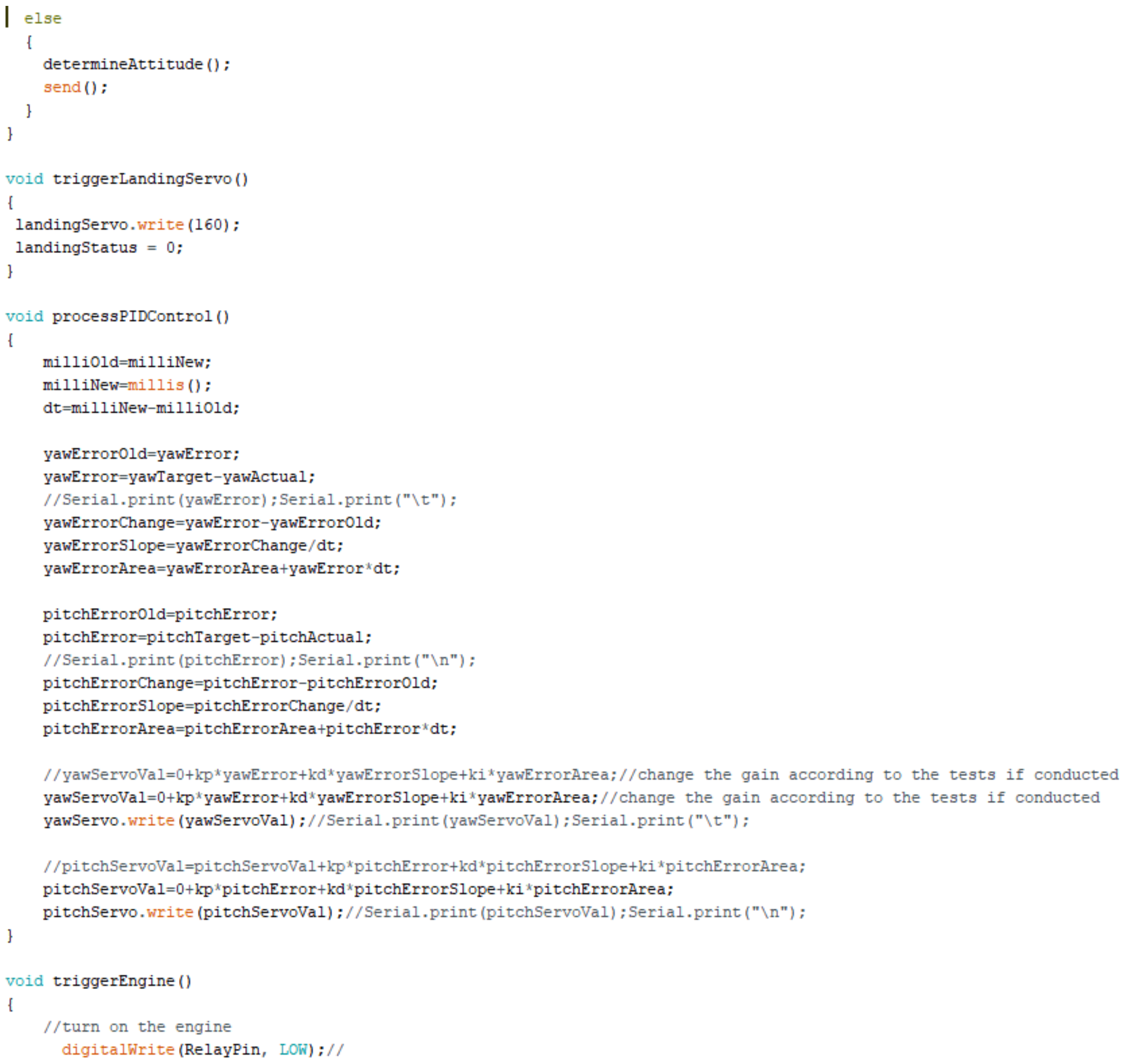
```
    float yawErrorSlope=0;
    float yawErrorArea=0;
    float yawServoVal=90;

    float pitchTarget=0;
    float pitchActual;
    float pitchError=0;
    float pitchErrorOld;
    float pitchErrorChange;
    float pitchErrorSlope=0;
    float pitchErrorArea=0;
    float pitchServoVal=90;
    Servo yawServo, pitchServo, landingServo;

  //nRF24L01
    const byte Address[6] = " 00009 ";


    RF24 radio(CE_PIN, CSN_PIN); // Create a Radio
    String buf;
    char dataToSend[30];

void setup() {
  //BMP280
    Serial.begin(9600);
    bmp.begin();
      /* Default settings from datasheet. */
    bmp.setSampling(Adafruit_BMP280::MODE_NORMAL,     /* Operating Mode. */
            Adafruit_BMP280::SAMPLING_X2,     /* Temp. oversampling */
            Adafruit_BMP280::SAMPLING_X16,    /* Pressure oversampling */
            Adafruit_BMP280::FILTER_X16,      /* Filtering. */
            Adafruit_BMP280::STANDBY_MS_500); /* Standby time. */

    p1 = bmp.readPressure(); delay(520); p2 = bmp.readPressure(); delay(520); p3 = bmp.readPressure(); delay(520); p4 = 
    Calib = (p1+p2+p3+p4+p5+p6+p7+p8)/800.0;Serial.print(Calib);

  //MPU6050
    #if I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE
      Wire.begin();
      TWBR = 24; // 400kHz I2C clock // (200kHz if CPU is 8MHz)(For Arduino UNO we have to use the 200kHz)
    #elif I2CDEV_IMPLEMENTATION == I2CDEV_BUILTIN_FASTWIRE
      Fastwire::setup(400, true);
    #endif

    //while(!Serial);
```



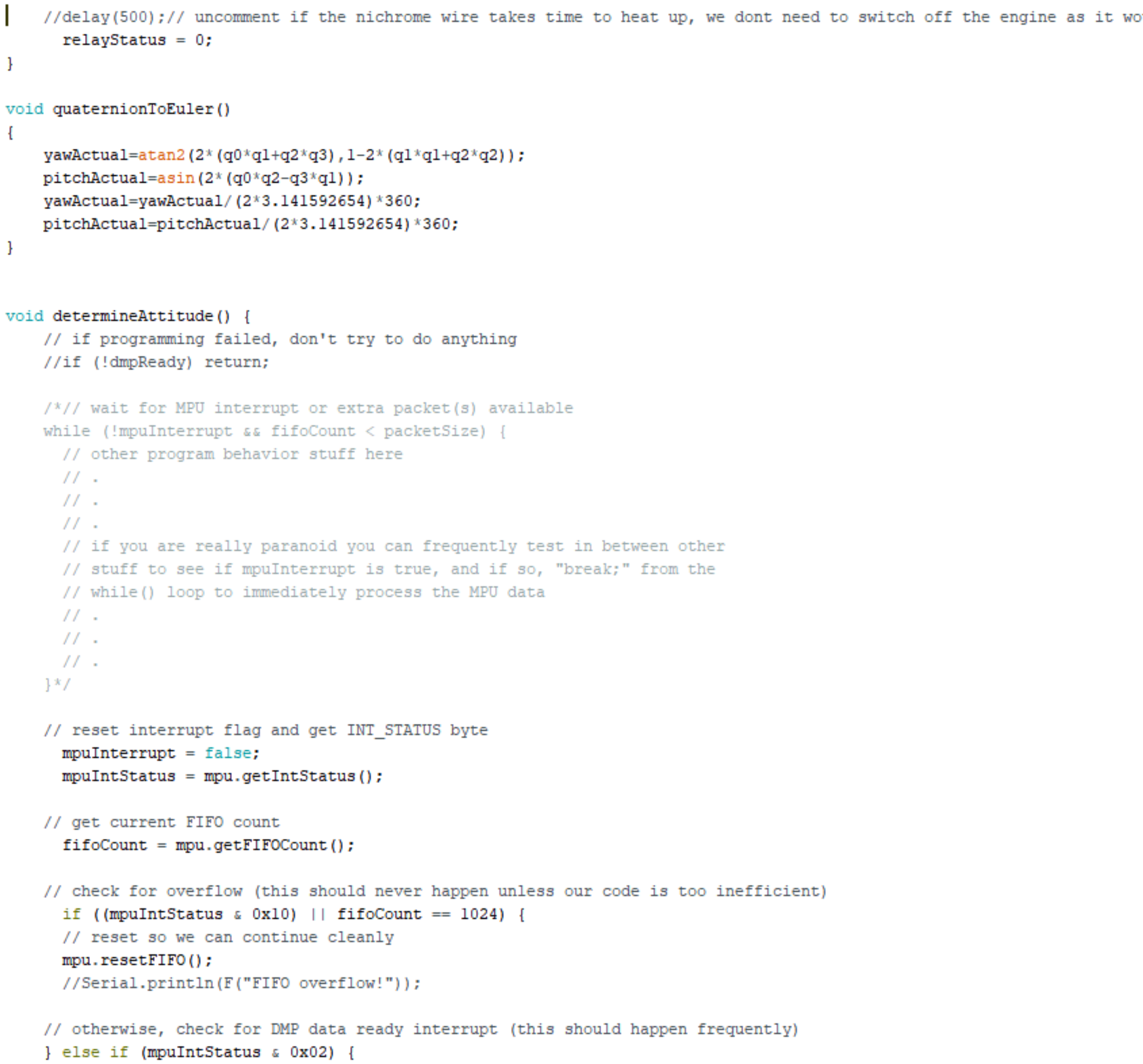
```
// initialize device
Serial.println(F("Initializing I2C devices..."));
mpu.initialize();

// verify connection
Serial.println(F("Testing device connections..."));
Serial.println(mpu.testConnection() ? F("MPU6050 connection successful") : F("MPU6050 connection failed"));

// wait for ready
//Serial.println(F("\nSend any character to begin DMP programming and demo: "));
while (Serial.available() && Serial.read()); // empty buffer
//while (!Serial.available());                 // wait for data
//while (Serial.available() && Serial.read()); // empty buffer again

// load and configure the DMP
//Serial.println(F("Initializing DMP..."));
devStatus = mpu.dmpInitialize();

// supply your own gyro offsets here, scaled for min sensitivity
mpu.setXGyroOffset(220);
mpu.setYGyroOffset(76);
mpu.setZGyroOffset(-85);
mpu.setZAccelOffset(1788); // 1688 factory default for my test chip

// make sure it worked (returns 0 if so)
if (devStatus == 0) {
  // turn on the DMP, now that it's ready
  //Serial.println(F("Enabling DMP..."));
  mpu.setDMPEnabled(true);

  // enable Arduino interrupt detection
  //Serial.println(F("Enabling interrupt detection (Arduino external interrupt 0)..."));
  attachInterrupt(0, dmpDataReady, RISING);
  mpuIntStatus = mpu.getIntStatus();

  // set our DMP Ready flag so the main loop() function knows it's okay to use it
  //Serial.println(F("DMP ready! Waiting for first interrupt..."));
  dmpReady = true;

  // get expected DMP packet size for later comparison
  packetSize = mpu.dmpGetFIFOPacketSize();
} else {
  // ERROR!
  // 1 = initial memory load failed
  // 2 = DMP configuration updates failed
  // (if it's going to break, usually the code will be 1)
```



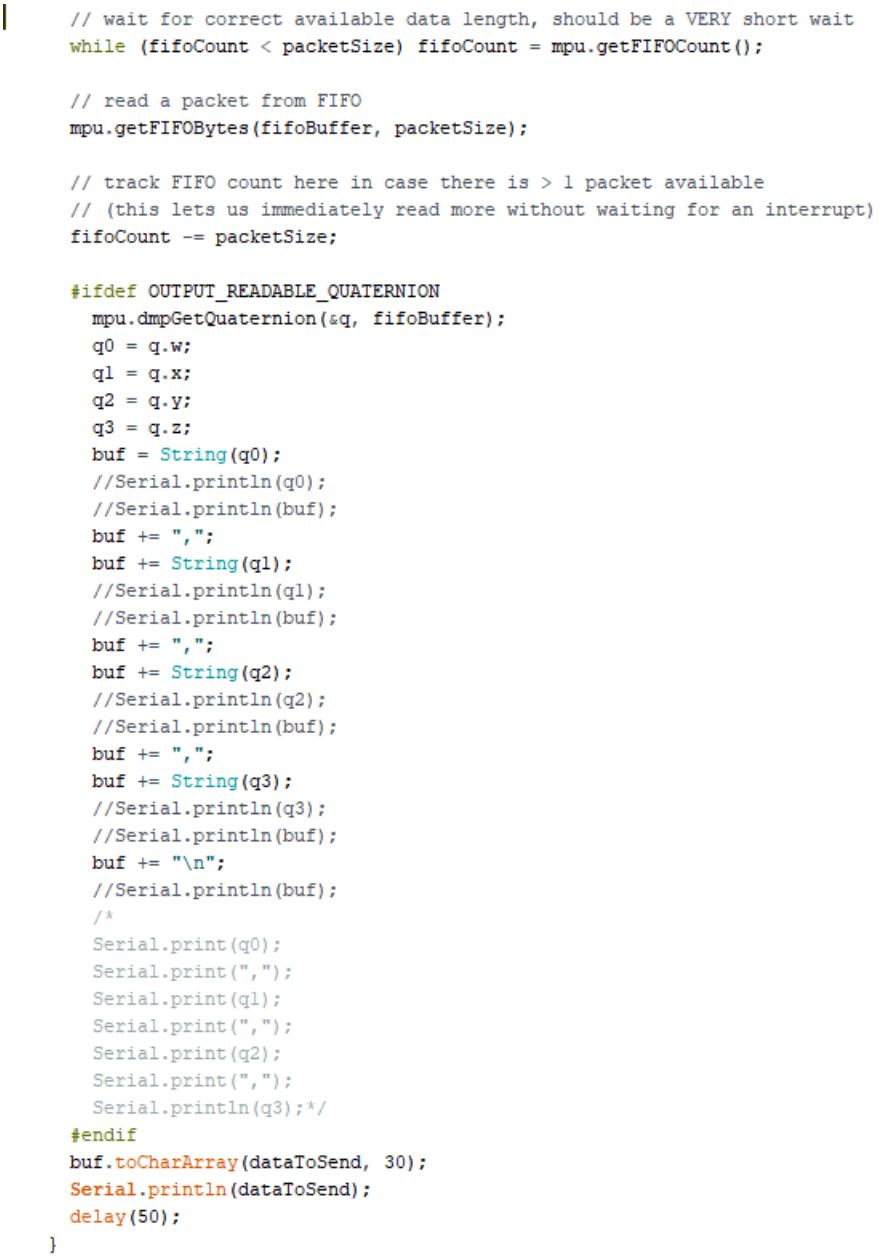
```
      Serial.print(F("DMP Initialization failed (code "));
      Serial.print(devStatus);
      Serial.println(F(")"));
    }

  // Set RelayPin as an output pin for trigger
    pinMode(RelayPin, OUTPUT);

    //Attach digital pins to servos
    yawServo.attach(21);
    pitchServo.attach(22);
    landingServo.attach(23);
    yawServo.write(yawServoVal);
    delay(20);
    pitchServo.write(pitchServoVal);
    delay(20);
    landingServo.write(landingServoVal);
    delay(20);

  //Timer
    milliNew=millis();

    //nRF24L01
    radio.begin();                        //activates modem
    //radio.setRetries(15,15);              //number of times modem retry to send data
    radio.openWritingPipe(Address);
    radio.setDataRate(RF24_2MBPS);
    radio.setPALevel(RF24_PA_MIN);
    radio.enableDynamicPayloads();
    radio.setAutoAck(true);
    radio.powerUp();
    radio.stopListening();               //switch the modem to data transmission mode
}
void loop(){
  Altitude = bmp.readAltitude(Calib);Serial.println(Altitude);
  if(Altitude < 10.0) /* use > in actual flight test this snippet works for demonstration at ground level only*/
  {
    if((relayStatus = 1))
    triggerEngine();// call only once
    if((landingStatus = 1))
    triggerLandingServo();
    determineAttitude();
    quaternionToEuler();
    processPIDControl();
    send();
  }
```



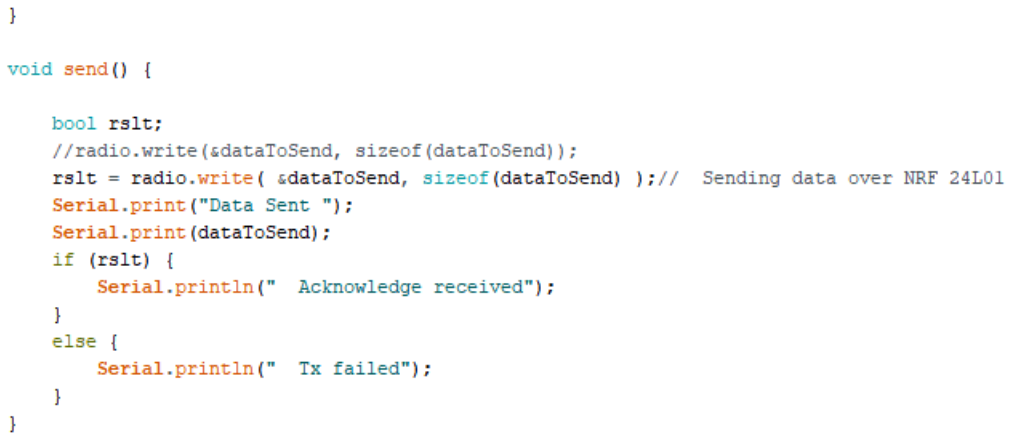
```
  else
  {
    determineAttitude();
    send();
  }
}

void triggerLandingServo()
{
 landingServo.write(160);
 landingStatus = 0;
}

void processPIDControl()
{
    milliOld=milliNew;
    milliNew=millis();
    dt=milliNew-milliOld;

    yawErrorOld=yawError;
    yawError=yawTarget-yawActual;
    //Serial.print(yawError);Serial.print("\t");
    yawErrorChange=yawError-yawErrorOld;
    yawErrorSlope=yawErrorChange/dt;
    yawErrorArea=yawErrorArea+yawError*dt;

    pitchErrorOld=pitchError;
    pitchError=pitchTarget-pitchActual;
    //Serial.print(pitchError);Serial.print("\n");
    pitchErrorChange=pitchError-pitchErrorOld;
    pitchErrorSlope=pitchErrorChange/dt;
    pitchErrorArea=pitchErrorArea+pitchError*dt;

    //yawServoVal=0+kp*yawError+kd*yawErrorSlope+ki*yawErrorArea;//change the gain according to the tests if conducted
    yawServoVal=0+kp*yawError+kd*yawErrorSlope+ki*yawErrorArea;//change the gain according to the tests if conducted
    yawServo.write(yawServoVal);//Serial.print(yawServoVal);Serial.print("\t");

    //pitchServoVal=pitchServoVal+kp*pitchError+kd*pitchErrorSlope+ki*pitchErrorArea;
    pitchServoVal=0+kp*pitchError+kd*pitchErrorSlope+ki*pitchErrorArea;
    pitchServo.write(pitchServoVal);//Serial.print(pitchServoVal);Serial.print("\n");
}

void triggerEngine()
{
    //turn on the engine
      digitalWrite(RelayPin, LOW);//
```



```
    //delay(500);// uncomment if the nichrome wire takes time to heat up, we dont need to switch off the engine as it wo
      relayStatus = 0;
}

void quaternionToEuler()
{
    yawActual=atan2(2*(q0*q1+q2*q3),1-2*(q1*q1+q2*q2));
    pitchActual=asin(2*(q0*q2-q3*q1));
    yawActual=yawActual/(2*3.141592654)*360;
    pitchActual=pitchActual/(2*3.141592654)*360;
}


void determineAttitude() {
    // if programming failed, don't try to do anything
    //if (!dmpReady) return;

    /*// wait for MPU interrupt or extra packet(s) available
    while (!mpuInterrupt && fifoCount < packetSize) {
      // other program behavior stuff here
      // .
      // .
      // .
      // if you are really paranoid you can frequently test in between other
      // stuff to see if mpuInterrupt is true, and if so, "break;" from the
      // while() loop to immediately process the MPU data
      // .
      // .
      // .
    }*/

    // reset interrupt flag and get INT_STATUS byte
      mpuInterrupt = false;
      mpuIntStatus = mpu.getIntStatus();

    // get current FIFO count
      fifoCount = mpu.getFIFOCount();

    // check for overflow (this should never happen unless our code is too inefficient)
      if ((mpuIntStatus & 0x10) || fifoCount == 1024) {
      // reset so we can continue cleanly
      mpu.resetFIFO();
      //Serial.println(F("FIFO overflow!"));

    // otherwise, check for DMP data ready interrupt (this should happen frequently)
    } else if (mpuIntStatus & 0x02) {
```

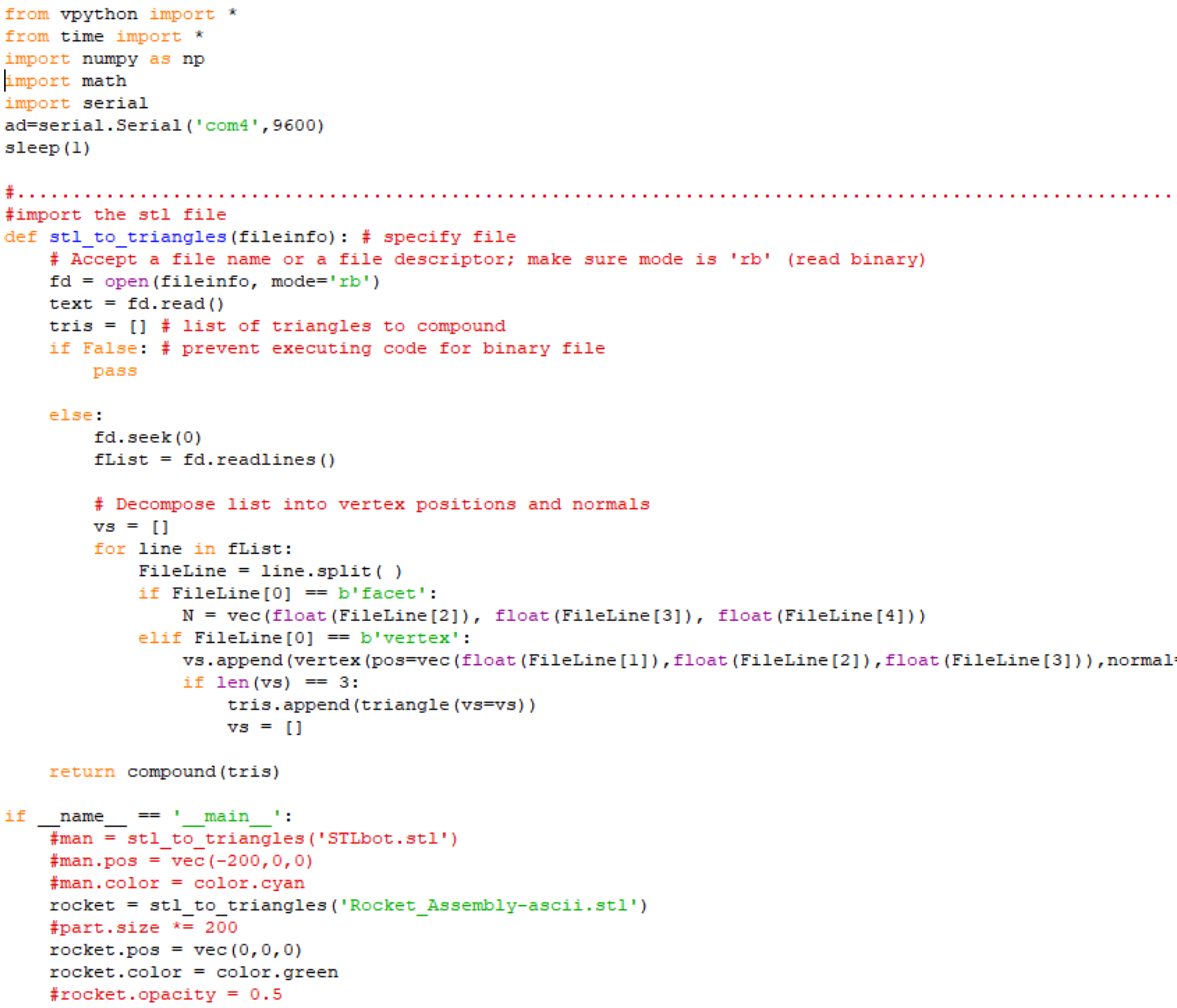


```
  // wait for correct available data length, should be a VERY short wait
  while (fifoCount < packetSize) fifoCount = mpu.getFIFOCount();

  // read a packet from FIFO
  mpu.getFIFOBytes(fifoBuffer, packetSize);

  // track FIFO count here in case there is > 1 packet available
  // (this lets us immediately read more without waiting for an interrupt)
  fifoCount -= packetSize;

  #ifdef OUTPUT_READABLE_QUATERNION
    mpu.dmpGetQuaternion(&q, fifoBuffer);
    q0 = q.w;
    q1 = q.x;
    q2 = q.y;
    q3 = q.z;
    buf = String(q0);
    //Serial.println(q0);
    //Serial.println(buf);
    buf += ",";
    buf += String(q1);
    //Serial.println(q1);
    //Serial.println(buf);
    buf += ",";
    buf += String(q2);
    //Serial.println(q2);
    //Serial.println(buf);
    buf += ",";
    buf += String(q3);
    //Serial.println(q3);
    //Serial.println(buf);
    buf += "\n";
    //Serial.println(buf);
    /*
    Serial.print(q0);
    Serial.print(",");
    Serial.print(q1);
    Serial.print(",");
    Serial.print(q2);
    Serial.print(",");
    Serial.println(q3);*/
  #endif
  buf.toCharArray(dataToSend, 30);
  Serial.println(dataToSend);
  delay(50);
}
```

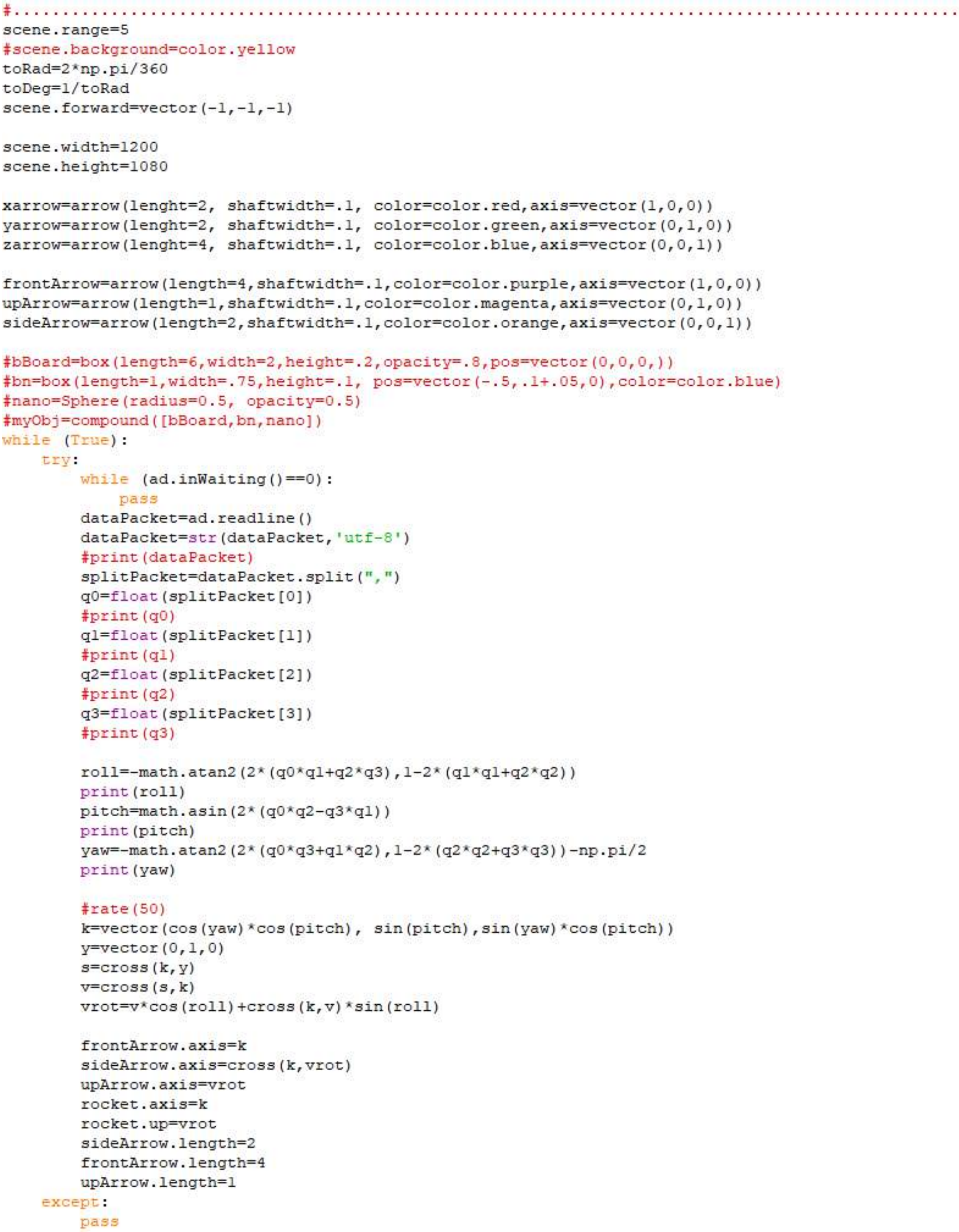


```
}

void send() {

    bool rslt;
    //radio.write(&dataToSend, sizeof(dataToSend));
    rslt = radio.write( &dataToSend, sizeof(dataToSend) );//  Sending data over NRF 24L01
    Serial.print("Data Sent ");
    Serial.print(dataToSend);
    if (rslt) {
        Serial.println("  Acknowledge received");
    }
    else {
        Serial.println("  Tx failed");
    }
}
```

14.11 Appendix K: Python Animation Processing Code



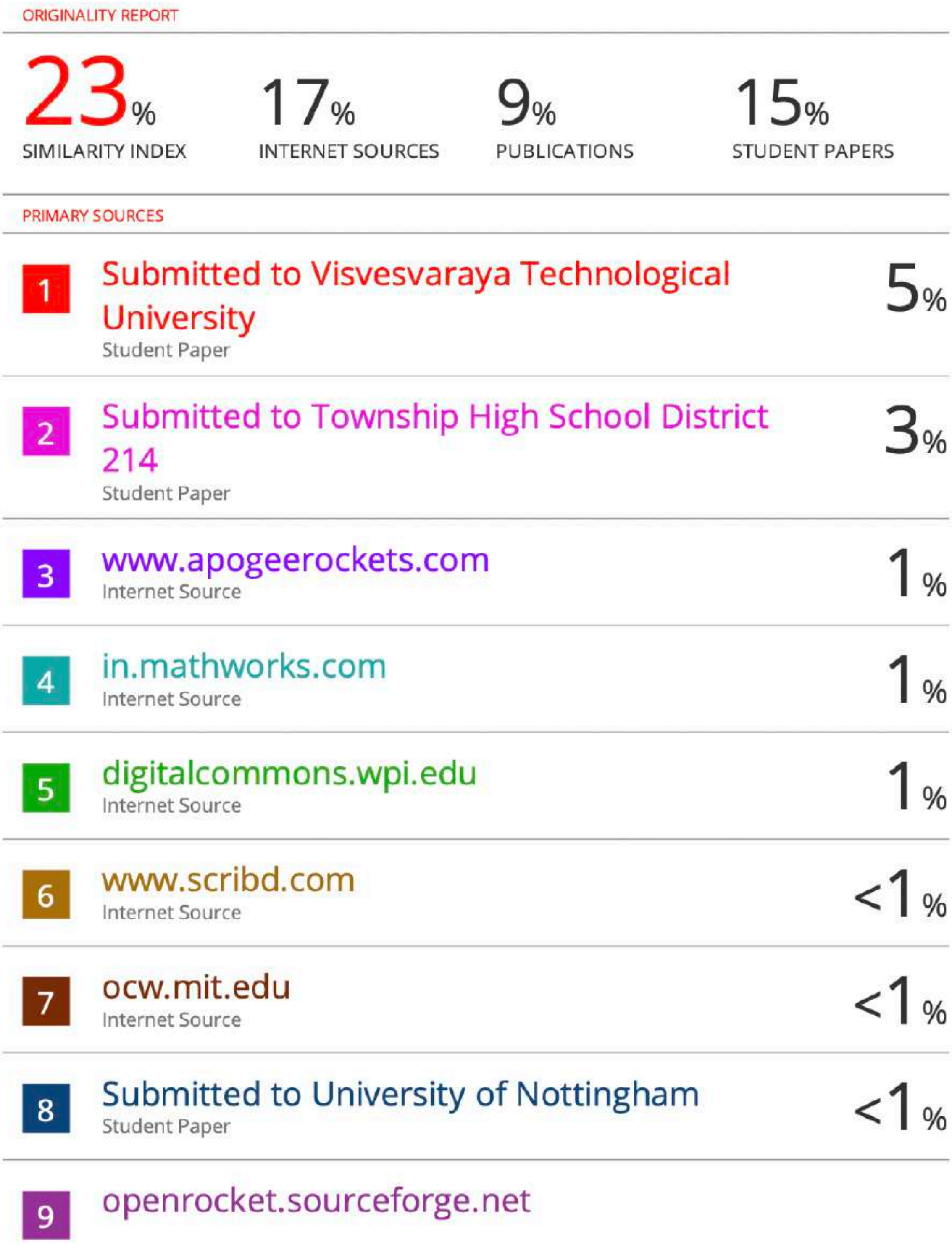
```
from vpython import *
from time import *
import numpy as np
import math
import serial
ad=serial.Serial('com4',9600)
sleep(1)

#.........................................................................................................
#import the stl file
def stl_to_triangles(fileinfo): # specify file
    # Accept a file name or a file descriptor; make sure mode is 'rb' (read binary)
    fd = open(fileinfo, mode='rb')
    text = fd.read()
    tris = [] # list of triangles to compound
    if False: # prevent executing code for binary file
        pass

    else:
        fd.seek(0)
        fList = fd.readlines()

        # Decompose list into vertex positions and normals
        vs = []
        for line in fList:
            FileLine = line.split( )
            if FileLine[0] == b'facet':
                N = vec(float(FileLine[2]), float(FileLine[3]), float(FileLine[4]))
            elif FileLine[0] == b'vertex':
                vs.append(vertex(pos=vec(float(FileLine[1]),float(FileLine[2]),float(FileLine[3])),normal=
                if len(vs) == 3:
                    tris.append(triangle(vs=vs))
                    vs = []

    return compound(tris)

if __name__ == '__main__':
    #man = stl_to_triangles('STLbot.stl')
    #man.pos = vec(-200,0,0)
    #man.color = color.cyan
    rocket = stl_to_triangles('Rocket_Assembly-ascii.stl')
    #part.size *= 200
    rocket.pos = vec(0,0,0)
    rocket.color = color.green
    #rocket.opacity = 0.5
```



```
#.................................................................................................
scene.range=5
#scene.background=color.yellow
toRad=2*np.pi/360
toDeg=1/toRad
scene.forward=vector(-1,-1,-1)

scene.width=1200
scene.height=1080

xarrow=arrow(lenght=2, shaftwidth=.1, color=color.red,axis=vector(1,0,0))
yarrow=arrow(lenght=2, shaftwidth=.1, color=color.green,axis=vector(0,1,0))
zarrow=arrow(lenght=4, shaftwidth=.1, color=color.blue,axis=vector(0,0,1))

frontArrow=arrow(length=4,shaftwidth=.1,color=color.purple,axis=vector(1,0,0))
upArrow=arrow(length=1,shaftwidth=.1,color=color.magenta,axis=vector(0,1,0))
sideArrow=arrow(length=2,shaftwidth=.1,color=color.orange,axis=vector(0,0,1))

#bBoard=box(length=6,width=2,height=.2,opacity=.8,pos=vector(0,0,0,))
#bn=box(length=1,width=.75,height=.1, pos=vector(-.5,.1+.05,0),color=color.blue)
#nano=Sphere(radius=0.5, opacity=0.5)
#myObj=compound([bBoard,bn,nano])
while (True):
    try:
        while (ad.inWaiting()==0):
            pass
        dataPacket=ad.readline()
        dataPacket=str(dataPacket,'utf-8')
        #print(dataPacket)
        splitPacket=dataPacket.split(",")
        q0=float(splitPacket[0])
        #print(q0)
        q1=float(splitPacket[1])
        #print(q1)
        q2=float(splitPacket[2])
        #print(q2)
        q3=float(splitPacket[3])
        #print(q3)

        roll=-math.atan2(2*(q0*q1+q2*q3),1-2*(q1*q1+q2*q2))
        print(roll)
        pitch=math.asin(2*(q0*q2-q3*q1))
        print(pitch)
        yaw=-math.atan2(2*(q0*q3+q1*q2),1-2*(q2*q2+q3*q3))-np.pi/2
        print(yaw)

        #rate(50)
        k=vector(cos(yaw)*cos(pitch), sin(pitch),sin(yaw)*cos(pitch))
        y=vector(0,1,0)
        s=cross(k,y)
        v=cross(s,k)
        vrot=v*cos(roll)+cross(k,v)*sin(roll)

        frontArrow.axis=k
        sideArrow.axis=cross(k,vrot)
        upArrow.axis=vrot
        rocket.axis=k
        rocket.up=vrot
        sideArrow.length=2
        frontArrow.length=4
        upArrow.length=1
    except:
        pass
```

15 ANNEXURE

Anti-Plagiarism Report



ORIGINALITY REPORT

23%	17%	9%	15%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	Submitted to Visvesvaraya Technological University Student Paper	5%
2	Submitted to Township High School District 214 Student Paper	3%
3	www.apogeerockets.com Internet Source	1%
4	in.mathworks.com Internet Source	1%
5	digitalcommons.wpi.edu Internet Source	1%
6	www.scribd.com Internet Source	<1%
7	ocw.mit.edu Internet Source	<1%
8	Submitted to University of Nottingham Student Paper	<1%
9	openrocket.sourceforge.net	