

A Final Year Project Report on

Fabrication & Simulation of VTOL based UAV swarms for Disaster Relief, Military and Commercial Applications

Submitted in Partial Fulfilment of the Requirements for the Award of Degree of

Bachelor of Technology
In
Mechanical Engineering



Submitted by

Anshul Awasthi
2016UME1274

Devnath S. Nair
2016UME1015

Nimesh Khandelwal
2016UME1270

Under the supervision of

Dr. Dinesh Kumar
Associate Professor
Department of Mechanical Engineering
MNIT, Jaipur

Department of Mechanical Engineering
Malaviya National Institute of Technology, Jaipur
May 2020



**DEPARTMENT OF MECHANICAL ENGINEERING
MALAVIAYA NATIONAL INSTITUTE OF TECHNOLOGY,
JAIPUR (RAJASTHAN)-302017**

DECLARATION

We declare that the training report titled “Fabrication & Simulation of VTOL based UAV swarms for Disaster Relief, Military and Commercial Applications” being submitted by us in partial fulfilment of the degree of B.Tech. (Mechanical Engineering) is a project work carried out by us under the supervision of Dr. Dinesh Kumar, Associate Professor, Department of Mechanical Engineering of Malaviya National Institute of Technology and the contents of this Project work, in full or in parts, have not been submitted to any other institute or University for the award of any certificate or diploma. We also certify that no part of this project work has been copied or borrowed from anyone else. In case any type of plagiarism is found out, we will be solely and completely responsible for it.

Date:	Anshul Awasthi	Devnath S. Nair	Nimesh Khandelwal
Place:	2016UME1274	2016UME1015	2016UME1270



**DEPARTMENT OF MECHANICAL ENGINEERING
MALAVIAYA NATIONAL INSTITUTE OF TECHNOLOGY,
JAIPUR (RAJASTHAN)-302017**

CERTIFICATE

This is to certify that report titled “Fabrication & Simulation of VTOL based UAV swarms for Disaster Relief, Military and Commercial Applications” that is being submitted by **Anshul Awasthi (2016UME1274), Devnath S. Nair (2016UME1015)** and **Nimesh Khandelwal (2016UME1270)** in partial fulfilment of the degree of Bachelor of Technology, (Mechanical Engineering) submitted to the Mechanical Engineering Department, Malaviya National Institute of Technology, Jaipur, is found to be satisfactory and is hereby approved for submission.

Date:

Place:

Dr. Dinesh Kumar

Associate Professor

Department of Mechanical Engineering

MNIT, Jaipur

Acknowledgements

Projects are essential to gain the practical exposure which otherwise, is not possible when studying about the theory in classrooms. This research experience has taught us immensely, but without the support of the faculty members of the *Department of Mechanical Engineering, Malaviya National Institute of Technology*, this would not have been possible.

It has been a wonderful and a very informative experience while doing this project. However, this would not have been so without the aid of the experienced professionals and many greatly talented people. Thus, it gives us immense pleasure to thank the people whose contributions and help in this project saved a lot of time and effort and taught us a great deal about the subject.

Foremost, we would like to thank Dr. Dinesh Kumar for giving us the opportunity to work on the project and for providing constant motivation, support and help while we carried out this project. Without his efforts, this project would not have reached the level it has.

We would also like to extend our sincere thanks to various developers that have developed various tools that were used in this project. Without their help and guidance on how to solve various problems and roadblocks we encountered while doing this project, we would possibly have been stuck or the project would have not made this much progress in the given time.

Lastly, it is never without our parents, who shower their unconditional love on us, we could have completed our project during this period.

Authors:

Anshul Awasthi	Devnath S. Nair	Nimesh Khandelwal
2016UME1274	2016UME1015	2016UME1270

Abstract

In this project report, we present the design, analysis and 3D simulation model of a dual-rotor tailsitter VTOL UAV. The VTOL UAV combines the advantages of a multicopter, having vertical takeoff and landing characteristics as well as hovering stability at a stationary point, with that of a fixed wing UAV, having efficient level flight and payload capacity with much lesser power consumption. We will discuss the evolution and use of VTOL UAV systems for military as well as commercial purposes followed by discussion of its mechanical CAD design and the methodologies used for that. Following sections detail the structural and aerodynamic analysis of various parts of the UAV. A new motor mount design has been analyzed in that section that acts as a fuse in case of a crash. The last section describes the approach and methodology used to setup a Gazebo simulation model of the vehicle, with a discussion of the aerodynamic forces acting on the vehicle. The simulation can be used as a test bed for various type of algorithmic testing.

Contents

Certificate	
Declaration	
Acknowledgements.....	i
Abstract.....	ii
List of Figures.....	v
List of Tables.....	vi
1. Chapter 1 – Introduction to VTOL UAVs.....	1
1.1. Introduction.....	1
1.2. Classification of VTOL UAVs.....	2
1.3. Project Aim.....	3
2. Chapter 2 - Mechanical Design.....	4
2.1. Design Methodology.....	4
2.1.1. Material Selection.....	5
2.1.2. SolidWorks Model.....	7
2.2. Fabrication and Assembly.....	7
2.2.1. Wing Fabrication.....	7
2.3. GPS Orientation Stabilization.....	8
2.4. Result.....	8
3. Chapter 3 - Mechanical and Flow Analysis.....	9
3.1. Introduction.....	9
3.2. The Black Box.....	10
3.3. Discussion on Aim #1.....	11
3.3.1. Mission Statement.....	11
3.3.2. Meshing.....	11
3.3.3. Normal Stress and Total Deformation Analysis.....	13
3.3.4. Diameter Range for Design.....	16
3.4. Discussion on Aim #2.....	18
3.4.1. Mission Statement.....	18
3.4.2. Meshing.....	18

3.4.3. Flow Analysis by ANSYS Fluent.....	20
3.4.4. Results from Flow Analysis.....	23
3.5. Conclusions.....	24
3.5.1. Aim #1.....	24
3.5.2. Aim #2.....	24
4. Chapter 4 - 3D Simulation Modelling and Testing	25
4.1 Introduction to Simulation.....	25
4.2 Unmanned Aerial Vehicle (UAV) Simulation.....	26
4.3 Different approaches to simulating UAVs.....	26
4.4 Achieving actuator control via ROS.....	27
4.5 Aerodynamic Forces on the Vehicle.....	29
4.6 Calculation of Aerodynamic Forces.....	29
4.6.1 Propulsive Forces.....	29
4.6.2 Aerodynamic Forces due to motion.....	33
4.7 Further Development Plans.....	35
5. Chapter 5 Conclusion and discussions.....	36
6. Appendix A.....	38
7. Appendix B.....	40
8. Appendix C.....	46
References.....	56

List of Figures

Figure	Caption	Page
1.1	Tiltrotor VTOL UAV	2
1.2	Multirotor VTOL UAV	2
1.3	Hybrid VTOL UAV	2
2.1	High Level MAV Architecture	4
2.2	Eppler E-168 Aerofoil Data	5
2.3	Designed Wing and Elevon profile	5
2.4	3D printed Motor Mount	6
2.5	Aluminium Reinforcement Spar design	6
2.6	Different Views of the MCAD model	7
2.7	Wing Fabrication	7
3.1	Behind the BLACK BOX	10
3.2	Motor Mount Multizone Meshing	12
3.3	Mesh Nodes and Elements	12
3.4	Skewness Plot	13
3.5	Skewness Quality	13
3.6	Varying Thrust Force	14
3.7	Normal Stress Variation	15
3.8	Total Strain Deformation	15
3.9	Motor Mount Analogy with Rod	16
3.10	Orthogonal Mesh Quality Check	19
3.11	Orthogonal & Skewness Quality	19
3.12	Contour of Static Pressure on an Aerofoil	20
3.13	Scaled Residual Plot	21
3.14	Coefficient of Drag Plot	22
3.15	Coefficient of Lift Plot	22
3.16	Force Results in X-Direction	23
3.17	Force Results in Y-Direction	23
3.18	Force Results in Z-Direction	23
4.1	Flow chart showing actuator control routine	28
4.2	Comparison of BET predicted Thrust values and experimental values as in manufacturer datasheet	31

List of Tables

Table	Caption	Page
1.1	Different types of VTOL UAVs	2

CHAPTER 1 Introduction to VTOL UAVs

1.1 Introduction

Unmanned Aerial Vehicles (UAV) are aircrafts which are controlled remotely via direct radio frequency communication or are equipped with autonomous devices to perform autonomous mission flights. The active development in the field of UAV in recent years has led to its increase in versatility in terms of its functionality and range of application.

Early UAV development is generally divided into two separate types, Fixed Wing UAV (FWUAV), and Rotorcraft UAV (RUAV). FWUAVs typically have longer endurance and thus can reach further than RUAV in a single flight, but require runways for take-offs and landings. RUAVs on the other hand, are more manoeuvrable and able to take-off and land vertically without any runway but has much lesser endurance due to their low efficiency in power consumption and low speed.

Therefore, to bridge the performance gap between RUAVs and FWUAVs, the solution is to have a **hybrid** Vertical-Take-Off-Landing (VTOL) aircraft. This type of aircraft will have the advantages of both FWUAV and RUAV and to a substantial extent should be able to eliminate their shortcomings.

In the aviation industry, major companies have already successfully mass-produced and commercialized numerous models of VTOL aircraft. To name a few:

- Bell Boeing V-22 Osprey (tilt rotor mechanism)
- Quantum Systems Trinity F90+
- Quantum Systems Tron F9 (electric-VTOL)
- ALTI's fleet of VTOL UAVs (Ascend, Transition, Reach)
- PD-1 Pro VTOL by UKR-SPEC systems




There are many more aviation companies investing heavily in VTOL aircrafts due to their dynamic nature and easy maneuverability. In academia also, these UAV systems are being researched due to their immense potential.

Currently, the major issue that the researchers are facing is to figure out an efficient transition maneuver for the transition from the multirotor mode to fixed wing mode during the flight.

1.2 Classification of VTOL UAVs

On the basis of their design and propulsion mechanism, the common VTOL UAVs can be classified into three types:

Table 1.1 Different types of VTOL UAVs

TYPE 1	TYPE 2	TYPE 3
<p>These are fixed wing aircraft with tilting rotor mechanism.</p>  <p><i>Figure 1.1 Tiltrotor VTOL UAV</i></p>	<p>They have no wings and have fixed vertical rotor(s)</p>  <p><i>Figure 1.2 Multirotor VTOL UAV</i></p>	<p>They have wings as well as vertical rotor(s)</p>  <p><i>Figure 1.3 Hybrid VTOL UAV</i></p>

Each of these designs has its own complexity, with the TYPE1 being most complex than others. Moreover, due to the difficulty in optimizing the transition flight from vertical to horizontal flight and vice versa, the VTOL UAV may lose its balance during the transition process and can lose a lot of altitudes. Therefore, it is vital to design the VTOL UAV that is lightweight, able to perform transition flight safely and also relatively simple yet efficient propulsion system.

1.3 Project Aim

The aim of this project is to design and develop an autonomous dual-rotor tailsitter VTOL UAV system that is capable of surveying, payload dropping, and reconnaissance. The key outcomes of this project are:

- Development of a Mechanical CAD (MCAD) model keeping in mind the results of the other VTOL UAVs already available in the market and the different projects being carried out at various universities.
- Stress and flow analysis on the motor mounts and wings of the vehicle respectively, the results from which will be used to calculate suitable design constraint for both.
- A 3D simulation model will be developed for the same MCAD model to be used with the Gazebo simulator., that can be used to test various algorithms before implementing them on the real-world model. It could also be used as a test bed for testing various control, path planning, and swarm algorithms for the developed platform.

This report details our original work done for achieving the outcomes of the project.

CHAPTER 2 Mechanical CAD Design of VTOL

Mechanical design means the design of components and systems of a mechanical nature—machines, products, structures, devices and instruments. For the most part mechanical design uses mathematics, materials, and the engineering-mechanics sciences.

2.1 Design Methodology:

In any mechanical design process, a higher-level block diagram is the first step, followed by the detailed design of the components. For our UAV, the MAV architecture is given in the below figure:

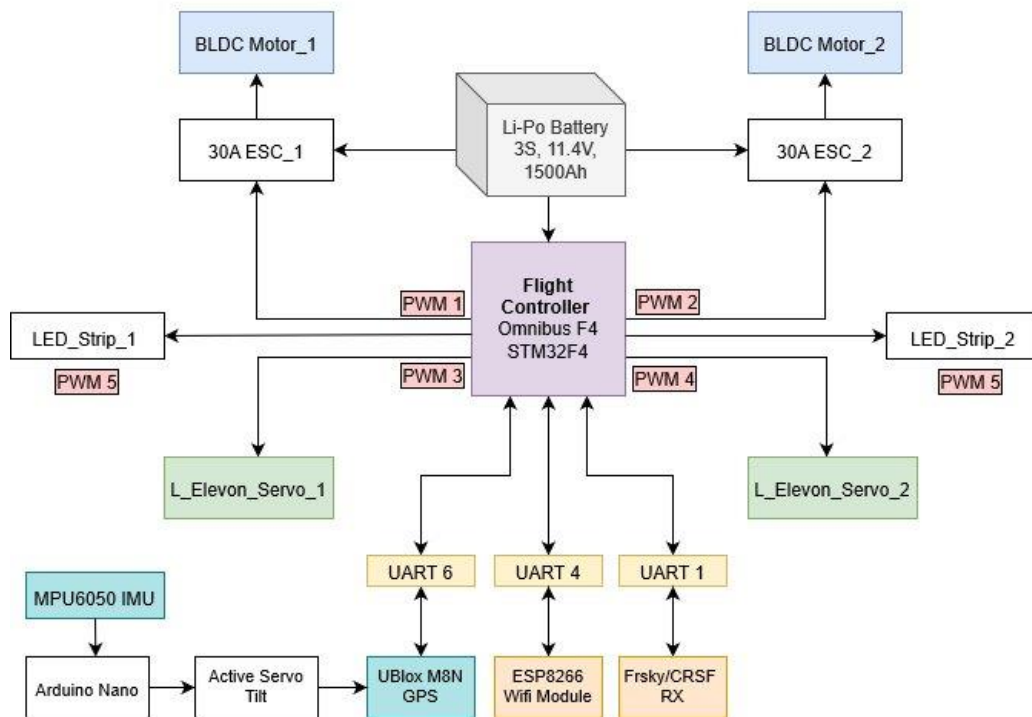


Figure 2.1 High Level MAV Architecture

The most important and critical aspect of any fixed wing aircraft is the aerofoil that it used. For that, we used an open source software called OpenVSP. The Eppler-168 (Figure 2.2) design was found to be suitable for our purposes as it is a symmetrical aerofoil and this can be modelled and debugged easily in the initial phases.

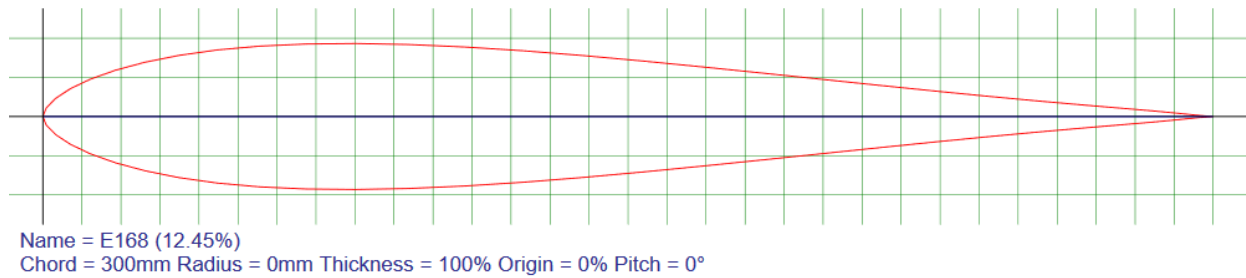


Figure 2.2 Eppler E-168 Aerofoil data

During the designing phase, the chosen aerofoil shape was integrated in the wings as well as the elevons to maintain the shape of the aerofoil over the entire air flow region. The SolidWorks design of the aerofoil is given in Figure 2.3

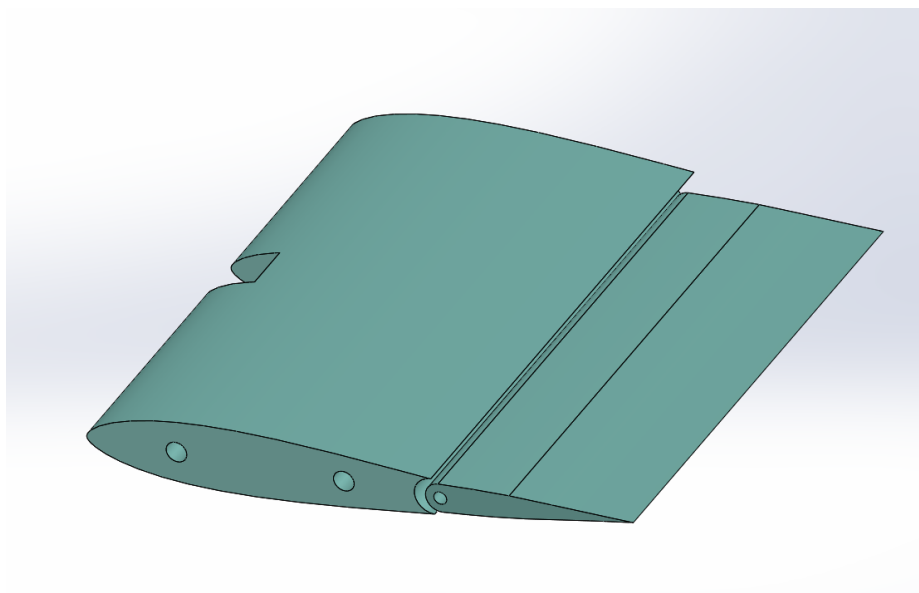


Figure 2.3 Designed Wing and Elevon profile

2.1.1 Material Selection:

For wings, we chose to use XPS (Polystyrene sheets) as they have a very high toughness and are very light as compared to other materials used in DIT, hobbyist

aeromodelling (Coroplast, etc.). The landing gear and motor mounts were made of PLA and were 3D printed in house to save the cost. The material selection here was due to the fact that these are the intended failure areas in the vehicle in the event of a crash. Therefore, being able to print these parts in house gives us the flexibility to test the vehicle to it's limit where it might even crash. The suitable working range of values for the motor mount has been discussed and calculated in the next chapter.

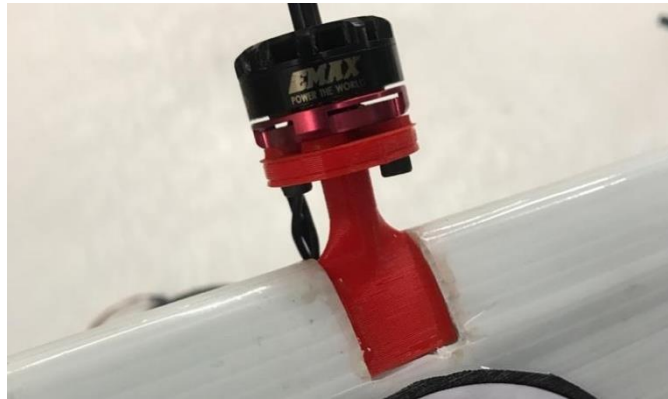


Figure 2.4 3D printed Motor Mount

For frame reinforcement, thin (10mm dia) aluminium tubes were used as spars to provide structural stability to the frame.

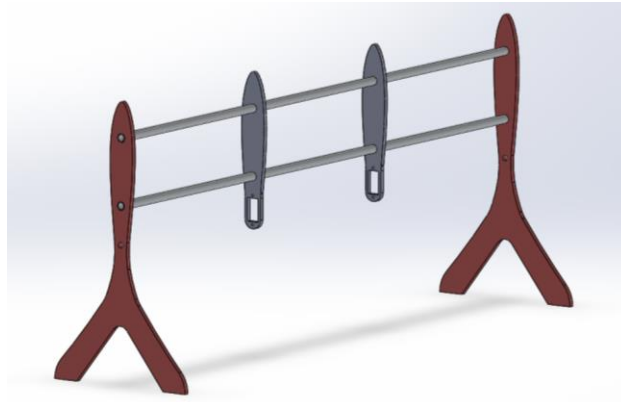


Figure 2.5 Aluminium Reinforcement Spar design

The materials are chosen to make the vehicle as light weight as possible while having acceptable working strength.

2.1.2 SolidWorks Model:

For CAD modelling of the vehicle, SolidWorks was chosen as it is the industry standard in the mechanical design industry. The different views of the developed model are shown in figure.

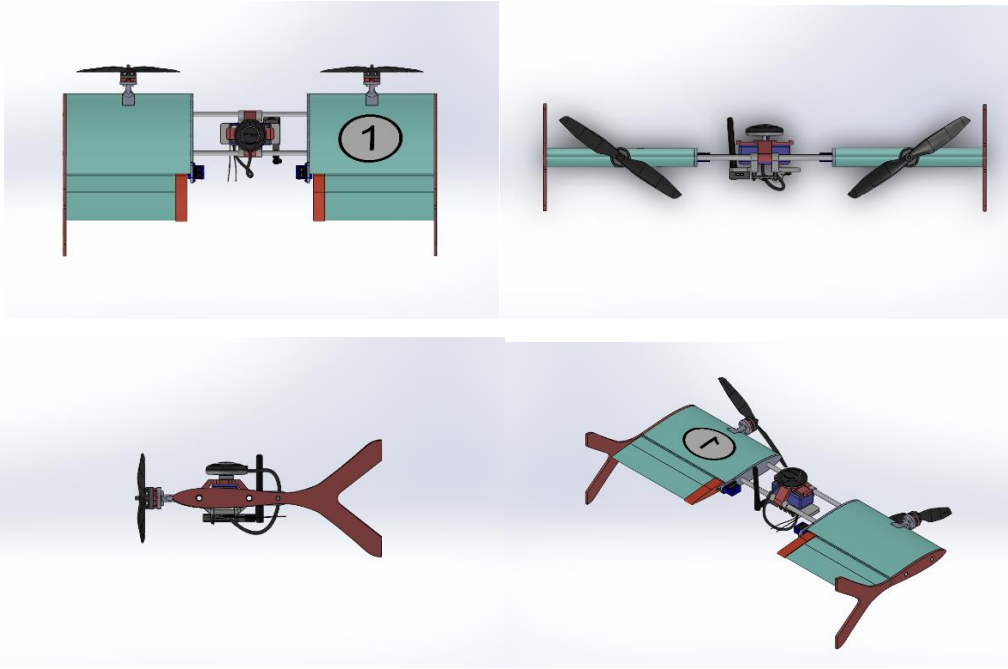


Figure 2.6 Different Views of the MCAD model

2.2 Fabrication and assembly:

2.2.1 Wing Fabrication:

Since XPS is a brittle material, it cannot be machined or cut using conventional cutting tools. For this, a hot wire cutter (Figure) was made and used. The resulting wing profile was found to be acceptable and was covered with coroplast sheets to compensate for the rough surface of the XPS to reduce turbulence drag over the wing.



Figure 2.7 Wing Fabrication

2.3 GPS orientation stabilizer:

Since we will be using GPS for position data of our model, it is crucial to keep the GPS flat all the time to ensure proper reception of the satellite signal by the patch antennae of GPS module. For this, an auto-stabilizer was developed that keeps the GPS flat irrespective of the pitch of the vehicle. Since pitch is the maneuver that the vehicle will do mostly, only one direction control was sufficient.

To filter out the high frequency noise due to vibrations of the vehicle a low pass filter was applied to the IMU data of the GPS stabilizer. This greatly helped stabilize as well as smoothen the motion of the stabilizer.

2.4 Result:

The MCAD model was made using SolidWorks. The part files were exported in different formats to be used for analysis as well as for development of the simulation model. Fabrication of the First prototype of the vehicle had been completed before March, 2020.

CHAPTER 3 MECHANICAL & FLOW ANALYSIS

Aim #1

To perform normal stress and total deformation analysis of motor mounts made up of PLA using ANSYS MECHANICAL & subsequently finding out the range of diameters for designing the neck of motor mount.

Aim #2

To perform flow analysis of the aerofoil using ANSYS FLUENT & consequently finding out the drag and lift forces on the aerofoil.

3.1 Introduction

The Ansys Workbench platform is the framework upon which the industry's broadest and deepest suite of advanced engineering simulation technology is built. An innovative project schematic view ties together the entire simulation process, guiding the user through even complex Multiphysics analysis with drag-and-drop simplicity. With bi-directional CAD connectivity, an automated project level update mechanism, pervasive parameter management and integrated optimization tools, the Ansys Workbench Platform delivers unprecedented productivity, enabling simulation driven product development.

Ansys Mechanical Enterprise is the flagship mechanical engineering software solution that uses finite element analysis (FEA) for structural analysis using the Ansys Mechanical interface. It covers an enormous range of applications and comes complete with everything you need from geometry preparation to optimization and all the steps in between. With Mechanical Enterprise you can model advanced materials, complex environmental loadings and industry-specific requirements in areas such as offshore hydrodynamics and layered composite materials.

Fluent software contains the broad, physical modelling capabilities needed to model flow, turbulence, heat transfer and reactions for industrial applications. These range from air flow over an aircraft wing to combustion in a furnace, from bubble columns to oil platforms, from blood flow to semiconductor manufacturing and from clean room design to wastewater treatment plants. Fluent spans an expansive range, including special models, with capabilities to model in-cylinder combustion, aero-acoustics, turbomachinery and multiphase systems.

3.2 The Black Box

A certain set of steps are followed in Ansys Workbench platform:

1. Pre-Analysis
2. Geometry
3. Mesh
4. Model Setup
5. Numerical Solution
6. Numerical Results
7. Verification & Validation

The following flow chart tells that what happens under the Black Box:

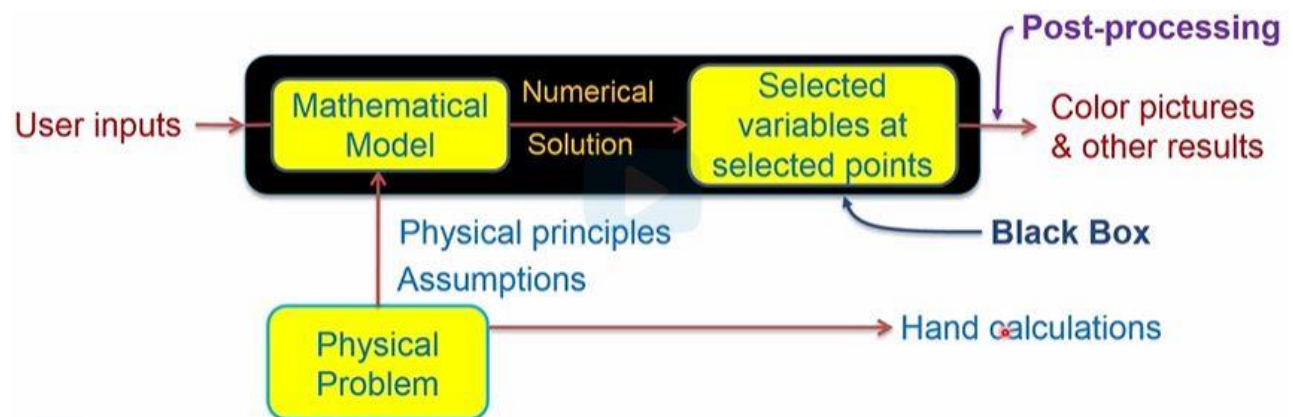


Figure 3.1 Behind the BLACK BOX

3.3 Discussion on Aim #1

3.3.1 Mission Statement

“To perform normal stress and total deformation analysis of motor mounts made up of PLA using ANSYS MECHANICAL & subsequently finding out the range of diameters for designing the neck of motor mount.”

The key targets of this part included:

- Performing Normal Stress Analysis
- Performing Total Deformation Analysis
- Finding the minimum diameter of the neck of motor mount with the help of maximum normal stress value obtained from stress analysis
- Finding the maximum diameter of the neck of motor mount (PLA Material) by considering the breakage of motor mount before the wings (XPS Material) break during a crash.

3.3.2 Meshing

Meshing is an integral part of the computer-aided engineering simulation process. The mesh influences the accuracy, convergence and speed of the solution. Once the best design is found, meshing technologies from ANSYS provide the flexibility to produce meshes that range in complexity from pure hex to highly detailed hybrid; a user can put the right mesh in the right place and ensure that a simulation will accurately validate the physical model.

Meshing methods available in Ansys are:

1. [Automatic Meshing Method](#) - If we select the automatic method control, the body will be swept if possible. Otherwise, Tetrahedrons (Patch Conforming) is used.
2. [Tetrahedrite/Hybrid Meshing Method](#) - where an all tetrahedral mesh is created.
3. [Hex Dominant Meshing Method](#) - where a free hex dominant mesh is created. This option is recommended for bodies that cannot be swept.

4. **Sweep Meshing Method** - A sweep mesh is forced on “sweepable” bodies (including axis-sweepable bodies, which are not displayed when you use the show sweepable bodies feature).

5. **Multizone Meshing Method** – It automatically generates a pure hexahedral mesh where possible and then fills the more difficult to capture regions with unstructured mesh. The Multizone has capabilities that make it more suitable for a class of problems for which the Sweep method would not work without extensive geometry decomposition.

Meshing of Motor Mount:

Multizone Mesh Method is used for the motor mount that is able to fill the whole volume with hexahedral elements (Hex20) which are more efficient for the same level of accuracy. The following photo shows the meshed motor mount:

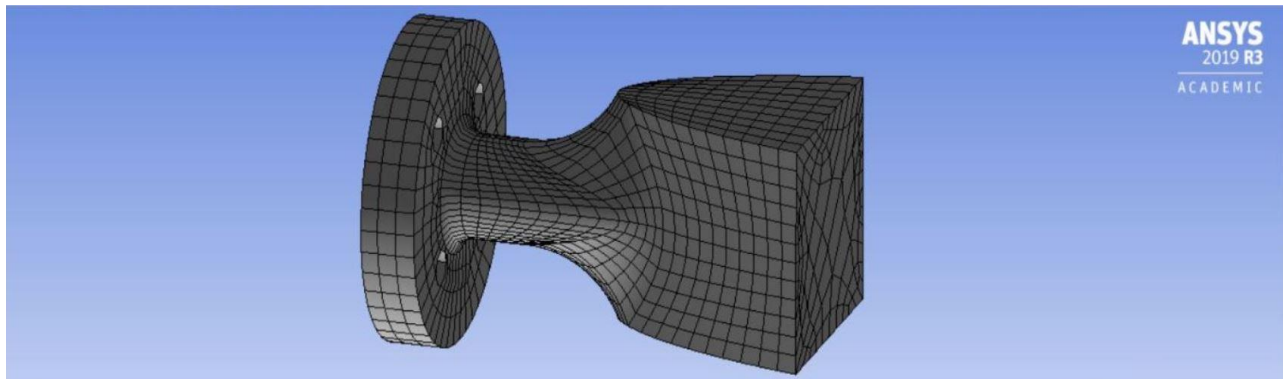


Figure 3.2 Motor Mount Multizone Meshing

This component has in total 15687 nodes and 3234 elements.

Details of "Mesh"	
[-] Display	
Display Style	Use Geometry Setting
[-] Defaults	
Physics Preference	Mechanical
Element Order	Program Controlled
<input type="checkbox"/> Element Size	Default
+ Sizing	
+ Quality	
+ Inflation	
+ Advanced	
[-] Statistics	
<input type="checkbox"/> Nodes	15687
<input type="checkbox"/> Elements	3234

Figure 3.3 Mesh Nodes and Elements

Another factor comes into picture that's very important to note is the Skewness of the meshing. **Skewness** is defined as the difference between the shape of the cell and the shape of an equilateral cell of equivalent volume. Highly skewed cells can decrease accuracy and destabilize the solution.

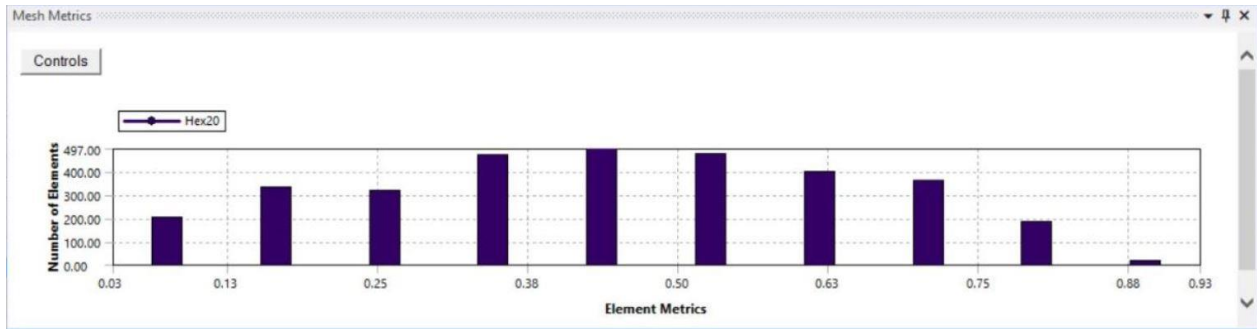


Figure 3.4 Skewness Plot

The table below lists the range of skewness values and corresponding cell quality:

The following table lists the range of skewness values and the corresponding cell quality.

Value of Skewness	Cell Quality
1	degenerate
0.9 — <1	bad (sliver)
0.75 — 0.9	poor
0.5 — 0.75	fair
0.25 — 0.5	good
>0 — 0.25	excellent
0	equilateral

According to the definition of skewness, a value of 0 indicates an equilateral cell (best) and a value of 1 indicates a completely degenerate cell (worst). Degenerate cells (slivers) are characterized by nodes that are nearly coplanar (collinear in 2D).

Figure 3.5 Skewness Quality

3.3.3 Normal Stress and Total Deformation Analysis

The five forces that act on the blades of an aircraft propeller in motion, they are:

1. **Thrust bending force:** Thrust loads on the blades act to bend them forward.

2. **Centrifugal twisting force:** Acts to twist the blades to a low, or fine pitch angle.
3. **Aerodynamic twisting force:** as the centre of pressure of a propeller blade is forward of its centreline the blade is twisted towards a coarse pitch position.
4. **Centrifugal force:** the force felt by the blades acting to pull them away from the hub when turning.
5. **Torque bending force:** Air resistance acting against the blades, combined with inertial effects causes propeller blades to bend away from the direction of rotation.

The above forces lead to reaction forces on the motor mount. Out of all these, Thrust Bending Force leads to the maximum bending moment impact on the motor mount. MATLAB Coding was used here for the extraction of values of Thrust Forces and Resisting Torques at varying values of Coefficient of Drag & Coefficient of Lift with varying motor speed. This varying thrust force is hence used in normal stress analysis & tabulated in Ansys as follows:

(Reference - Appendix A: MATLAB code for BET calculations)

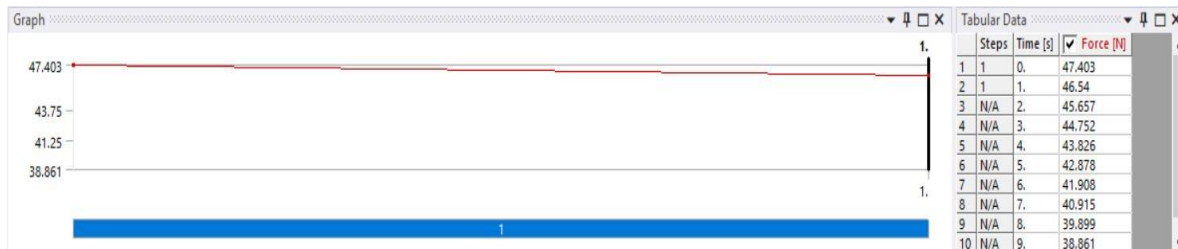


Figure 3.6 Varying Thrust Force

After conducting the normal stress analysis, the conclusion is made that the motor mount which we have designed could bear the maximum stress without any trouble because maximum stress value comes out to be lesser than the Yield Strength of PLA. The following is the photo showing stress results:

From results obtained,

Maximum stress = **8.5716 MPa**

Maximum strain = **3.0864e-04**

Yield Strength of PLA = 35.9 MPa

Since, from above: **Maximum Stress (8.5716 MPa) < Y.S. (35.9 MPa)**

Hence, our design is valid according to stress considerations.

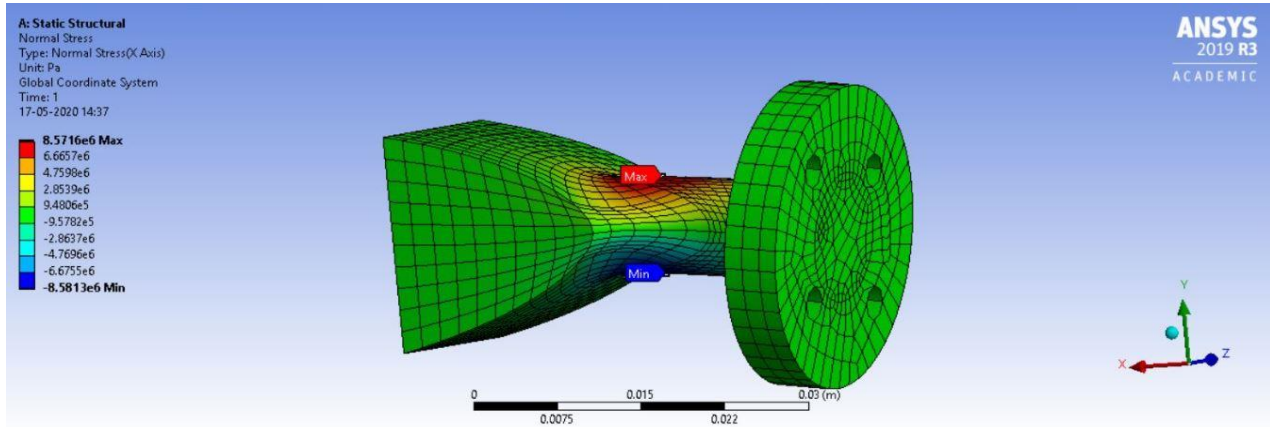


Figure 3.7 Normal Stress Variation

And from Total Deformation Analysis, we can conclude that the maximum deformation value (normal strain) comes out to be lesser than that obtained when the motor mount is subjected to maximum normal stress value.

From results obtained,

Young's Modulus of PLA = 2.3 GPa

Maximum Stress obtained above = 8.5716 MPa

Therefore corresponding deformation (strain) = (Maximum Stress/Young's Modulus)

Theoretical Strain = $(8.5716 \times 10^6) / (2.3 \times 10^9) = 3.7267 \times 10^{-3}$

Since the maximum strain obtained is less than strain corresponding to max. stress

That is, **Maximum Strain (3.0864×10^{-4}) < Theoretical Strain (3.7267×10^{-3})**

Hence, our design is also valid according to strain considerations.

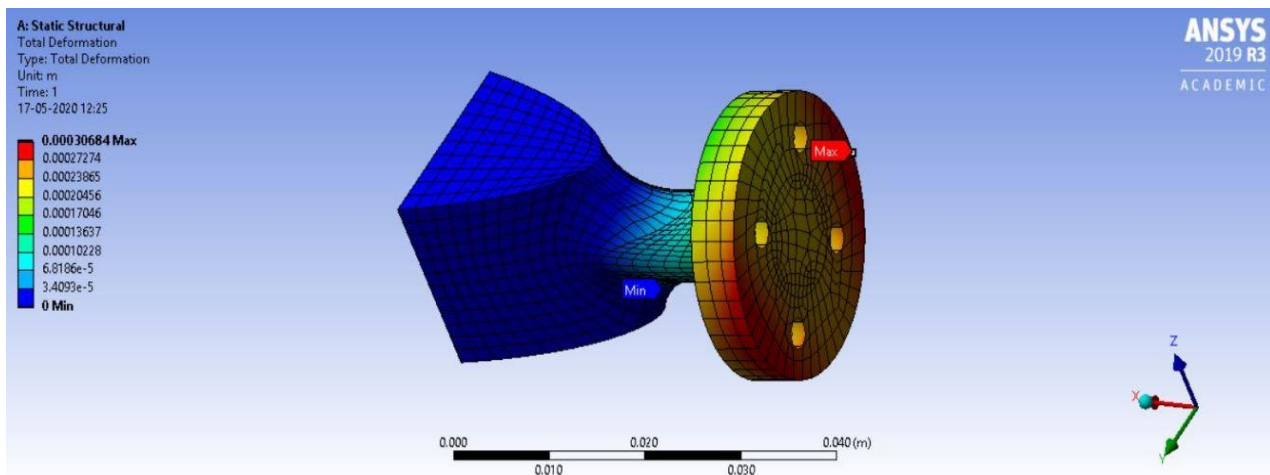


Figure 3.8 Total Strain Deformation

3.3.4 Diameter Range for Design

- ✚ Finding the minimum diameter of the neck of motor mount with the help of maximum normal stress value obtained from stress analysis.

Maximum Stress obtained above = 8.5716 MPa

Maximum Thrust Force value = 47.402 N

Since Thrust Force is acting axially so,

Maximum Stress = (Max. Thrust Force / Cross-section Area)

Cross-section is circular so let's suppose it as "Dmin"

Cross-section Area = $(\pi * (Dmin^2) / 4)$

$8.5716e+06 = (47.402 / \text{Cross-section Area})$

$Dmin = \{(4 * 47.402) / ((8.5716e+06) * \pi)\}^{(0.5)}$

Therefore, **Dmin = 2.6535 mm**

- ✚ Finding the maximum diameter of the neck of motor mount (PLA Material) by considering the breakage of motor mount before the wings (XPS Material) break during a crash.

The problem is mainly possessing axial thrust loading and torsional loading on a circular rod of diameter 'Dmax' as shown in the simplified figure below:

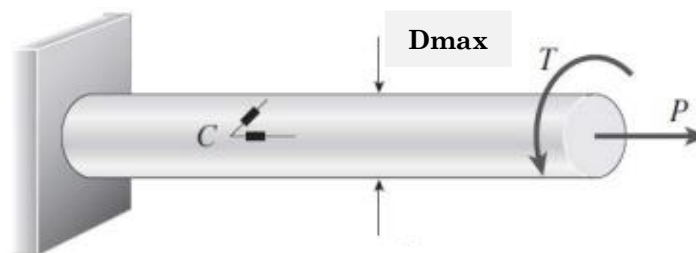


Figure 3.9 Motor Mount Analogy with Rod

$$\sigma = (F \cdot y / I) = (F \cdot 32 / (\pi \cdot (D_{\max}^3))) = (47.4029 \cdot 32 / (\pi \cdot (D_{\max}^3)))$$

$$\tau = (T \cdot r / J) = (T \cdot 16 / (\pi \cdot (D_{\max}^3))) = (0.5238 \cdot 16 / (\pi \cdot (D_{\max}^3)))$$

$$\sigma_1 = (16 / (\pi \cdot (D_{\max}^3))) \cdot (F + ((F^2) + (T^2))^{0.5})$$

$$\sigma_1 = (16 / (\pi \cdot (D_{\max}^3))) \cdot (47.4029 + ((47.4029^2) + (0.5238^2))^{0.5})$$

$$\text{Therefore, } \sigma_1 = (482.8567 / (D_{\max}^3))$$

$$\sigma_2 = (16 / (\pi \cdot (D_{\max}^3))) \cdot (F - ((F^2) + (T^2))^{0.5})$$

$$\sigma_2 = (16 / (\pi \cdot (D_{\max}^3))) \cdot (47.4029 - ((47.4029^2) + (0.5238^2))^{0.5})$$

$$\text{Therefore, } \sigma_2 = (-0.0148 / (D_{\max}^3))$$

Yield Point for XPS (σ_y) = 3300 MPa

According to Maximum Shear Stress Theory;

$$|\sigma_1 - \sigma_2| \leq \sigma_y$$

$$|(482.8567 / (D_{\max}^3)) + (0.0148 / (D_{\max}^3))| \leq 3300e+06$$

Therefore, $D_{\max} \leq 5.269 \text{ mm}$.

According to Von Mises Criteria;

$$\{(\sigma_1 - \sigma_2)^2 + (\sigma_2 - \sigma_3)^2 + (\sigma_3 - \sigma_1)^2\} / 2 \leq \sigma_y^2$$

$$\{(482.8715 / (D_{\max}^3))^2 + (482.8567 / (D_{\max}^3))^2 + (0.0148 / (D_{\max}^3))^2\} \leq (3300e+06)^2$$

Therefore, $D_{\max} \leq 5.269 \text{ mm}$.

Hence, the Diameter Range is given as:

$$D_{\min} \leq d \leq D_{\max}$$

$$2.6535 \text{ mm} \leq d \leq 5.269 \text{ mm}$$

3.4 Discussion on Aim #2

3.4.1 Mission Statement

“To perform flow analysis of the aerofoil using ANSYS FLUENT & consequently finding out the drag and lift forces on the aerofoil.”

The key targets of this part included:

- Finding out the Lift Force on the aerofoil
- Finding out the Drag Force on the aerofoil
- Obtaining values and plotting graphs of Coeff. of Drag and Lift with 100 iterations

3.4.2 Meshing

Here, fine meshing is done which influences the accuracy, convergence and speed of the simulation. Some Mesh Quality Metric such as orthogonality, skewness, aspect-ratio are seen as conceptual means in the evaluation of mesh quality and its impact on obtaining an accurate solution. A check on Orthogonality (A Mesh Quality Metric) becomes necessary here.

Mesh Quality Metrics:

Orthogonality -

The concept of mesh orthogonality relates to how close the angles between adjacent element faces (or adjacent element edges) are to some optimal angle (depending on the relevant topology). The orthogonality measure ranges from 0 (bad) to 1 (good).

Skewness -

Skewness in tetrahedral elements is best captured by the deviation from an optimal (equilateral) volume.

Aspect ratio -

The aspect ratio metric would be defined as length to height ratio in 2D or as the radius ratio of circumscribed to the inscribed circles in 3D (sometimes also the area ratio).

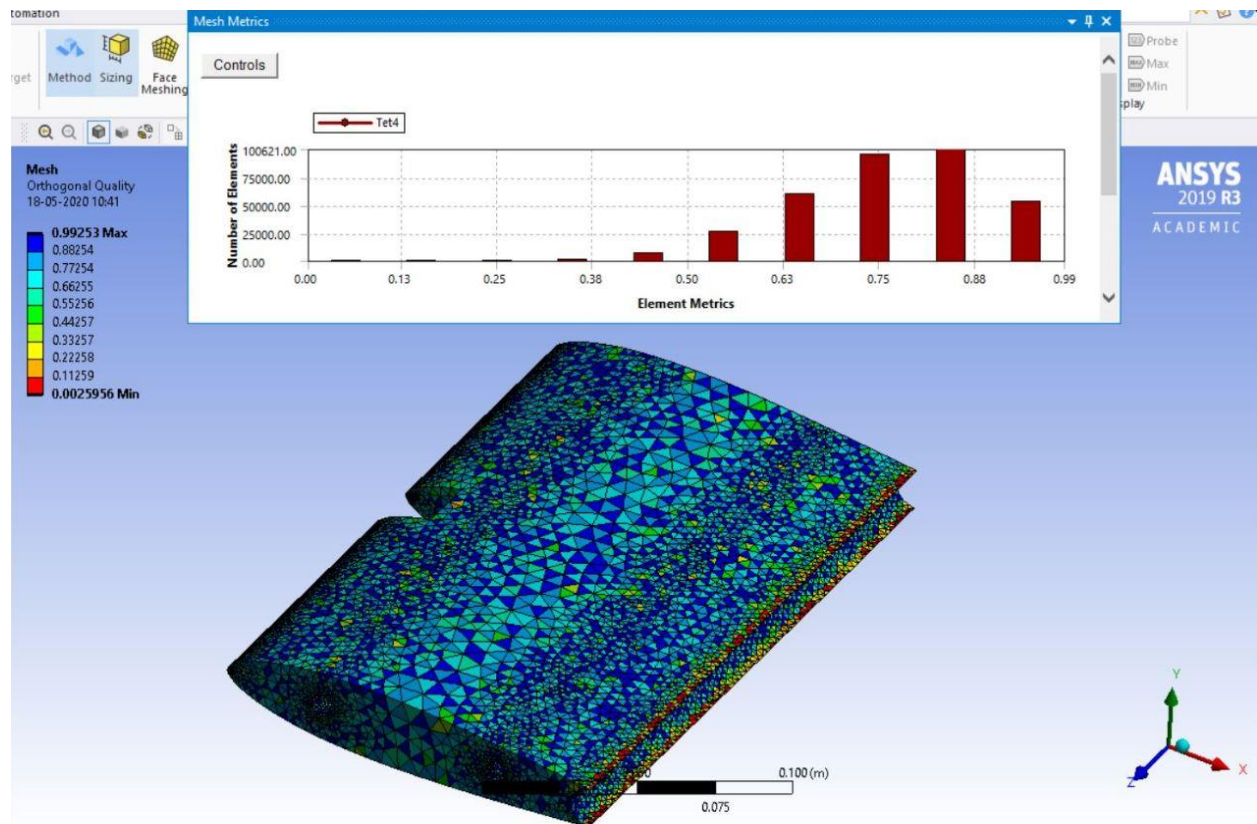


Figure 3.10 Orthogonal Mesh Quality Check

From above orthogonal meshing style, it can be concluded that maximum number of elements lie in the acceptable range. This can be concluded from the table below:



Figure 3.11 Orthogonal & Skewness Quality

3.4.3 Flow Analysis by Ansys Fluent

Assuming the properties of air at 1 atm and at a temperature 25 degree Celsius;

Density = 1.184 kg/m³

Coefficient of Viscosity = 1.849e-05 kg/m-s

Cruising Speed/ Maximum speed at which UAV could glide = 10 m/s

Chord Width = 0.256 m

Reynold's Number (Rn)= (Density * Max. Speed * Chord Width)/ Coeff. of viscosity

$Rn = (1.184 * 10 * 0.256 / (1.849e-05))$

Therefore, **Rn = 1,63,892**

Hence, we can conclude that the flow is turbulent in nature.

Now after substituting all the given data and all the other properties in Ansys Fluent, we obtain the following contour plot of Static Pressure over the wall of the aerofoil.

Contours of Static Pressure on the aerofoil is shown in snapshot below:

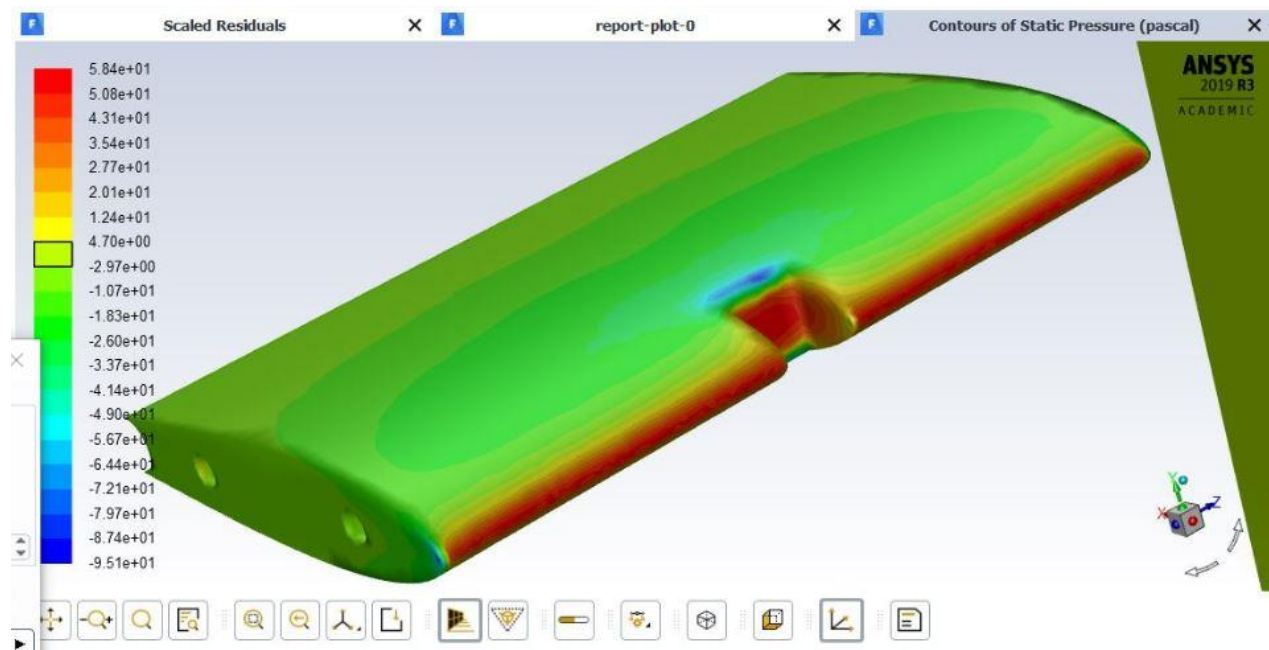


Figure 3.12 Contour of Static Pressure on an Aerofoil

From the above snapshot, it can be observed that due to uniform air flow over the surfaces of the aerofoil, the maximum static pressure is obtained at the stagnation points, i.e. where air velocity becomes zero (RED REGION), negative pressure is

generated due to back flow of air (BLUE REGION) and remaining (GREEN REGION) shows slight variations in static pressures over aerofoil surfaces.

Various Graph Plots with 100 iterations performed:

1. Scaled Residual Plot -

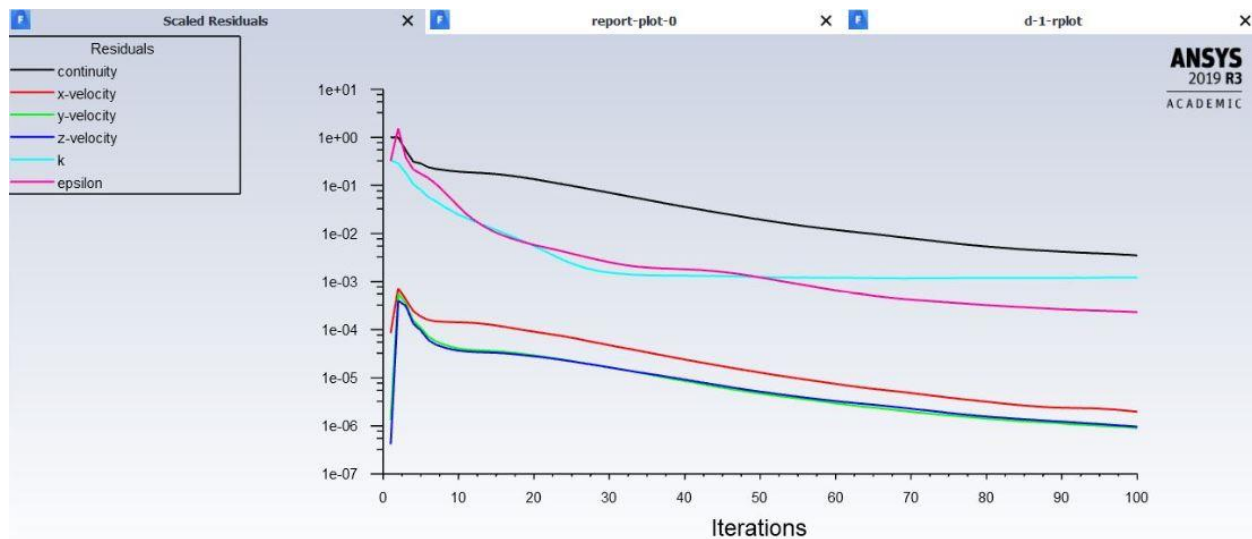


Figure 3.13 Scaled Residual Plot

In a CFD Analysis, the residual measures the local imbalance of a conserved variable in each control volume. In an iterative numerical solution, the residual will never be exactly zero, However, the lower the residual value is, the more numerically accurate the solution.

The above graph represents variations in Continuity, velocity in X-direction, velocity in Y-direction and velocity in Z-direction.

2. Coefficient of Drag Plot –

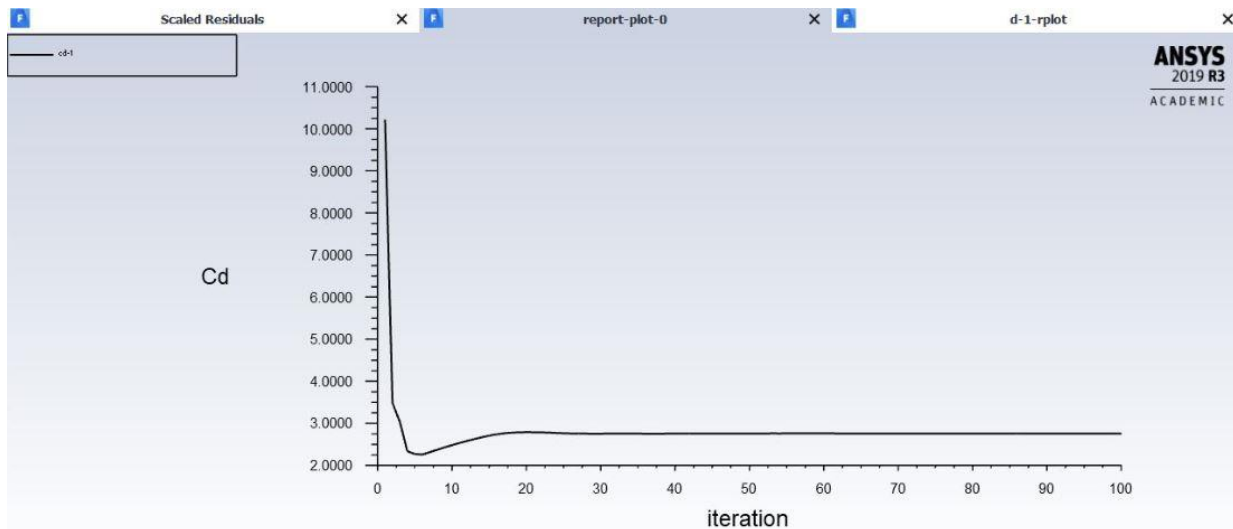


Figure 3.14 Coefficient of Drag Plot

From the above graph, it can be observed that with increase is the number of iterations the value of drag coefficient decreases in an almost exponential manner.

3. Coefficient of Lift Plot –

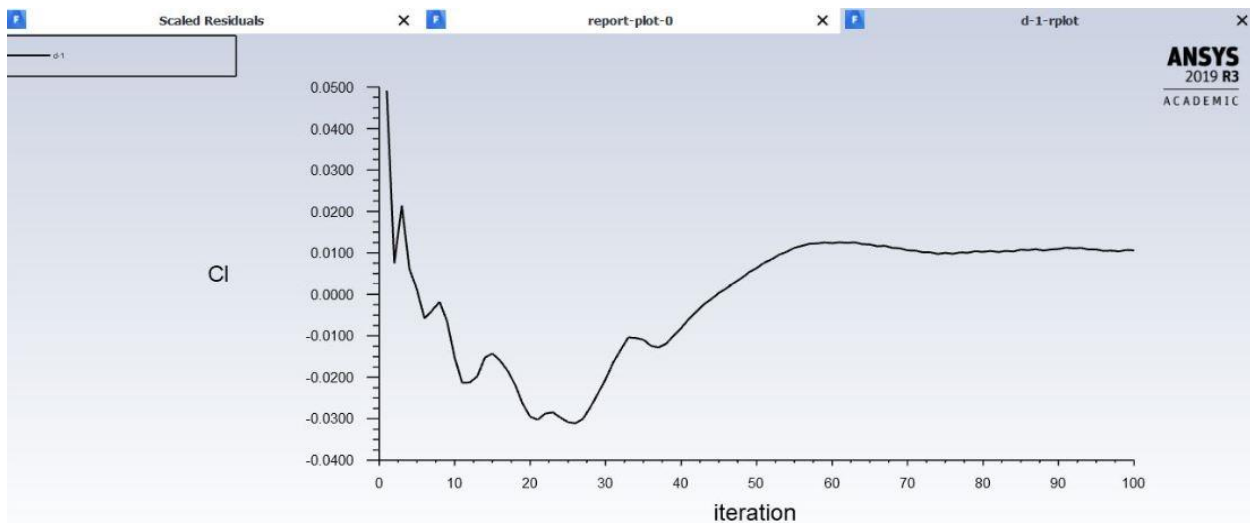


Figure 3.15 Coefficient of Lift Plot

From the above graph, it can be observed that with increase is the number of iterations the value of lift coefficient first decreases then attains a minimum value then starts increasing in an irregular manner.

3.4.4 Results from Flow Analysis

Force Calculations in X-Direction:

Forces								
Coefficients								
Zone	Pressure			Viscous			Total	
Pressure	Viscous			Total			Total	
wall-surrounding_flow_channel	0.066998236	0.006409952	0.022766242	(2.8274133	8.8313165e-05	-0.00044381601)	(2.8944116	
0.0064982651	0.022322426)	(0.10938487	0.010465228	0.037169374)	(4.616185	0.00014418476	-0.00072459757)	(4.7255699
0.010609412	0.036444777)	-----						
Net	(0.066998236	0.006409952	0.022766242)	(2.8274133	8.8313165e-05	-0.00044381601)	(2.8944116	
0.0064982651	0.022322426)	(0.10938487	0.010465228	0.037169374)	(4.616185	0.00014418476	-0.00072459757)	(4.7255699
0.010609412	0.036444777)	-----						
Forces - Direction Vector (1 0 0)								
Coefficients								
Zone	Pressure			Viscous			Total	
Pressure	Viscous			Total			Total	
wall-surrounding_flow_channel	0.066998236	2.8274133	2.8944116	0.10938487	4.616185	4.7255699		
Net	0.066998236	2.8274133	2.8944116	0.10938487	4.616185	4.7255699		

Figure 3.16 Force Results in X-Direction

Force Calculations in Y-Direction:

Forces								
Coefficients								
Zone	Pressure			Viscous			Total	
Pressure	Viscous			Total			Total	
wall-surrounding_flow_channel	0.066998236	0.006409952	0.022766242)	(2.8274133	8.8313165e-05	-0.00044381601)	(2.8944116	
0.0064982651	0.022322426)	(0.10938487	0.010465228	0.037169374)	(4.616185	0.00014418476	-0.00072459757)	(4.7255699
0.010609412	0.036444777)	-----						
Net	(0.066998236	0.006409952	0.022766242)	(2.8274133	8.8313165e-05	-0.00044381601)	(2.8944116	
0.0064982651	0.022322426)	(0.10938487	0.010465228	0.037169374)	(4.616185	0.00014418476	-0.00072459757)	(4.7255699
0.010609412	0.036444777)	-----						
Forces - Direction Vector (0 1 0)								
Coefficients								
Zone	Pressure			Viscous			Total	
Pressure	Viscous			Total			Total	
wall-surrounding_flow_channel	0.006409952	8.8313165e-05	0.0064982651	0.010465228	0.00014418476	0.010609412		
Net	0.006409952	8.8313165e-05	0.0064982651	0.010465228	0.00014418476	0.010609412		

Figure 3.17 Force Results in Y-Direction

Force Calculations in Z-Direction:

Forces								
Coefficients								
Zone	Pressure			Viscous			Total	
Pressure	Viscous			Total			Total	
wall-surrounding_flow_channel	0.066998236	0.006409952	0.022766242)	(2.8274133	8.8313165e-05	-0.00044381601)	(2.8944116	
0.0064982651	0.022322426)	(0.10938487	0.010465228	0.037169374)	(4.616185	0.00014418476	-0.00072459757)	(4.7255699
0.010609412	0.036444777)	-----						
Net	(0.066998236	0.006409952	0.022766242)	(2.8274133	8.8313165e-05	-0.00044381601)	(2.8944116	
0.0064982651	0.022322426)	(0.10938487	0.010465228	0.037169374)	(4.616185	0.00014418476	-0.00072459757)	(4.7255699
0.010609412	0.036444777)	-----						
Forces - Direction Vector (0 0 1)								
Coefficients								
Zone	Pressure			Viscous			Total	
Pressure	Viscous			Total			Total	
wall-surrounding_flow_channel	0.022766242	-0.00044381601	0.022322426	0.037169374	-0.00072459757	0.036444777		
Net	0.022766242	-0.00044381601	0.022322426	0.037169374	-0.00072459757	0.036444777		

Figure 3.18 Force Results in Z-Direction

From above results, it can be concluded that:

Drag Force = 0.0066998 N

Lift Force = 0.006409952 N

Therefore, observing the above values of drag and lift forces, it can be concluded that their effects will be negligible as the length of the aerofoil is small.

3.5 Conclusions

3.5.1 Aim #1 –

- **Maximum Stress (8.5716 MPa) < Y.S. (35.9 MPa) so *our design is valid according to stress considerations.***
- **Maximum Strain (3.0864e-04) < Theoretical Strain (3.7267e-03) hence, *our design is also valid according to strain considerations.***
- **Diameter Range: 2.6535mm <= D <= 5.269mm**

3.5.2 Aim #2 –

- **Drag Force = 0.0066998 N**
- **Lift Force = 0.006409952 N**
- **It is concluded that Lift and Drag force effects will be negligible as the length of the aerofoil is small.**
- **Scaled Residuals of Continuity, X-Direction Velocity, Y-Direction Velocity, Z-Direction Velocity increases in the starting, reaches a peak then decreases with increase in the number of iterations.**
- **Lift coefficient first decreases then attains a minimum value then starts increasing in an irregular manner.**
- **Drag coefficient decreases in an almost exponential manner.**

CHAPTER 4 - 3D Simulation Modelling and testing

A 3D simulation model has been developed in order to test control algorithms before implementing them on the prototype. A near accurate aerodynamics model of the vehicle has been developed for this purpose. The developed 3D model and simulation environment can now be used as a test bed for anyone who wants to research in the field of dual-rotor VTOLs to implement and test various algorithms (Control, Path Planning, etc.).

4.1 Introduction to Simulation

According to Wikipedia, “*A simulation is an approximate imitation of the operation of a process or system; that represents its operation over time*”. Simulation is used in many contexts, such as simulation of technology for performance tuning or optimizing, safety engineering, testing, training, education, and video games.

A simulation model can be called a test model on which we can test our inputs for the response without the need to engage real-world hardware. The real-world hardware cannot be directly employed due to various reasons: it may not be accessible, it may be dangerous or unacceptable to engage, it is being designed but not yet built, or it may simply not exist!

For our project, we have built a real-world model but it is too dangerous for the model and the operator both to run various tests that might cause it to become unstable and crash. Using the simulation model, we can first form a near accurate understanding of how the model is going to behave to a given input, rectifying the mistakes that might be made along the way, and then test those inputs on the real-world model. This approach is clearly both time and cost efficient as well as much safer than directly using the real hardware.

4.2 Unmanned Aerial Vehicle (UAV) Simulation

UAV simulation involves artificially re-creating aircraft flight characteristics and the environment in which it flies for design, or other purposes. It includes replicating the equations that govern how aircraft fly, how they react to applications of flight controls, the effects of other aircraft systems, and how the vehicle reacts to external factors such as air density, turbulence, wind shear, cloud, precipitation, etc.

The current simulation model that has been developed comes under this category of simulation, and is being used for testing different control schemes, swarm behavior, path planning, SLAM, etc. There are numerous simulators in the market that can be used for robotics as well as UAV specific simulations. Ex. UAV Simulator by Quantum3D, VT MAK, Dynautics, Gazebo, etc. to name a few.

Out of the simulators mentioned above, we decided to use the Gazebo simulator as it is an open source software and is being used by the academia for robotics for quite some time now. Also, it can be used in conjunction with ROS (Robot Operating System), a set of software libraries and tools that enable development and testing of various robots.

The robot model is written in the SDF (Simulation Description Format) file format as well as in URDF (Universal Robot Description Format). Both file formats can be used with the simulator of our choice.

4.3 Different approaches to simulating UAVs

For the simulation purpose, we broadly tried two different approaches. The first one was to use the SITL (Software In The Loop) model of an already existing flight control stack (like PixHawk, ArduCopter, etc.) with our model, and then use some middleware framework (like MAVROS) to send high level commands to it via ROS. The second approach includes writing a custom flight control stack along with

an aerodynamic response model of the vehicle. The former approach is clearly simpler but the latter provides more flexibility as well as control over the behaviour of the vehicle.

While testing the first approach, we used the PX4 SITL model and used MAVROS framework to communicate with the SITL using ROS to send commands. While we were able to get basic control of the model using the SITL approach, there were some clear drawbacks in this approach.

- The higher-level control of the dual-rotor VTOL UAV had problems in the autonomous mode, and autonomy of the vehicle is a basic requirement for the future plans of this project.
- Getting individual actuator control should have been possible using the SITL simulator, and there were different MAVROS topics too for this purpose, but none of them worked for the dual-rotor VTOL.

Getting the actuator control was crucial for this project, as without that we could not test custom control schemes on the developed platform.

To solve this issue, custom Gazebo Plugins were developed, written in C++, to access individual actuator control and to apply appropriate aerodynamic forces on the vehicle during the simulation. The code for both is given in Appendix B and C.

4.4 Achieving actuator control via ROS

For controlling the actuators of a robot, ROS has a pre-built package called ROS Control that contains libraries and tools for this purpose. It uses a generic control loop feedback mechanism, typically a PID controller, to control the output, typically effort, sent to your actuators. It has multiple available controller plugins, out of which the controller plugin of our interest is the “effort_controllers” plugin as we want the vehicle to experience the reaction forces when the inputs change.

For testing this controller, the model was described in the URDF format. For the elevons, “joint_position_controller” under the effort_controller class was used.

After appropriate PID tuning, the elevon actuators behaved properly as required. For the rotors, “joint_velocity_controller” under the effort_controller class was used. This did not work. Even after tuning the PID values of the controller, it did not respond to the given inputs.

Since the ROS Control plugin did not work for the rotors, a custom actuator control plugin was developed. The plugin reads the actuator input values from the user, calculates the error, and applies the torque on the joint until error is reduced to 0.1% of the input value. The PD control scheme was used for both the type of actuation (Position for elevons and Velocity for rotors). The torque applied on the joint is calculated as:

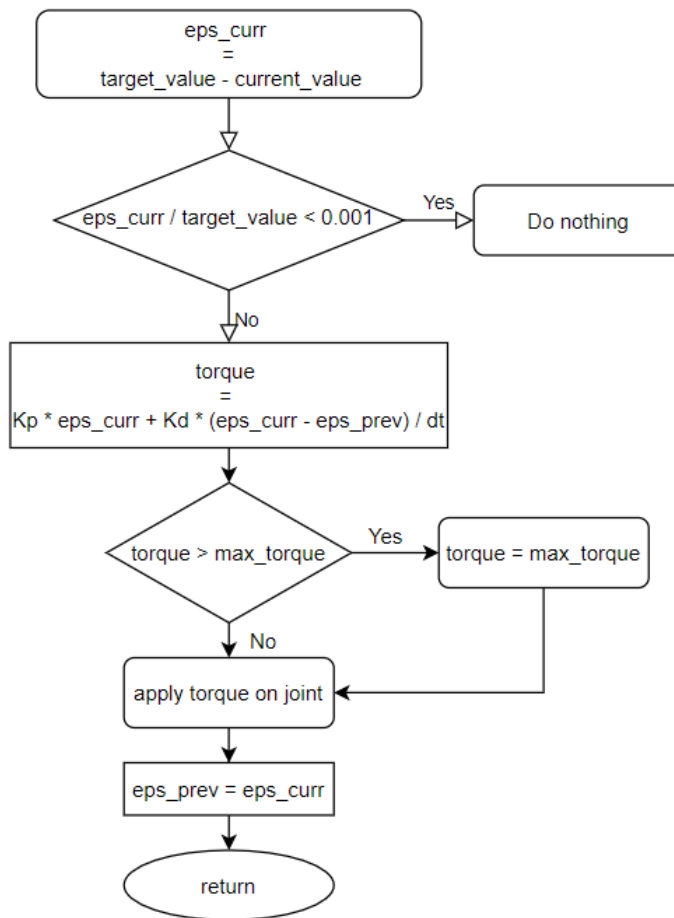


Figure 4.1 Flow chart showing actuator control routine

The values of K_p and K_d were found to be:

- $K_p = 30.0$ & $K_d = 1.0$ for the elevon joints
- $K_p = 0.01$ & $K_d = 0.00001$ for the rotor joints

The inputs for rotor are given in RPM by the user but they are converted to rad per sec before calculating the error for the rotors. The inputs for elevons are given in radians.

4.5 Aerodynamic forces on the vehicle

Another important aspect of the simulation model is the application of forces on the vehicle due to its motion as well as propulsive forces. A near accurate aerodynamic model of the vehicle along with a plugin to apply those forces on the vehicle was developed. The aerodynamic forces on the vehicle can be broadly classified into two categories:

1. **Propulsive Forces:** The propulsive forces are the forces produced on the vehicle due to the propulsion system of the vehicle. These include the thrust forces on the rotors, resistive torque on the rotors, reaction forces on the elevons due to deflection of the air stream coming from the propeller, and the lift force on the wing (in case of asymmetrical airfoil wings).
2. **Aerodynamic drag forces due to motion:** These are the forces that the vehicle experiences due to its motion in the fluid media (air). As the air strikes the surface of the vehicle and gets deflected, there is a resultant drag force on the vehicle body. These forces are produced due to both the translational as well as the rotational motion of the vehicle.

4.6 Calculation of the Aerodynamic forces

4.6.1 Propulsive Forces:

For the calculation of the propulsive forces, we need to calculate the thrust force, resistive torque on the rotor, downstream velocity of the air going through the propeller, as well as the angle by which the stream gets deflected by the elevons.

For calculating the thrust and torque produced by the propeller, Blade Element Theory (BET) was used. This includes discretizing the propeller blade into small elements and then iteratively calculating the net thrust and torque produced by the propeller of given specification going through each element. A MATLAB script (Appendix A) was developed for this purpose. Using the script, the relationship between the thrust/torque and the RPM of the propeller was established using a quadratic polynomial fit. The fitted thrust(T)/torque(Q) and RPM relationship is given by the equations:

$$\mathbf{T} = (\mathbf{T2} * \mathbf{RPM}^2 + \mathbf{T1} * \mathbf{RPM} + \mathbf{T0})$$

$$\mathbf{Q} = (\mathbf{Q2} * \mathbf{RPM}^2 + \mathbf{Q1} * \mathbf{RPM} + \mathbf{Q0})$$

here,

$$\mathbf{T2} = 1.2061\text{e-}07$$

$$\mathbf{T1} = -4.1556\text{e-}07$$

$$\mathbf{T0} = -1.9043\text{e-}06$$

$$\mathbf{Q2} = 1.3114\text{e-}09$$

$$\mathbf{Q1} = -2.18595\text{e-}10$$

$$\mathbf{Q0} = -2.3435\text{e-}08$$

The thrust also depends on the forward velocity of the propeller, and at a critical velocity (**Vcr**) the net thrust produced by the propeller reduces to zero. This critical velocity is calculated as:

$$\mathbf{Vcr} = \mathbf{pitch} * \mathbf{RPM} / \mathbf{60} \text{ (m/sec)}$$

A separate MATLAB script was used to establish the thrust/torque and forward velocity relationship and was found to be linearly decreasing. The velocity correction factor (**f**) is then calculated as:

$$\mathbf{f} = (\mathbf{1} - \mathbf{Vz} / \mathbf{Vcr})$$

here,

Vz: Forward velocity of the vehicle

Vcr: Critical Velocity at given RPM

The final expression for the net thrust (T_{net}) and net torque (Q_{net}) is then given as

$$\begin{aligned} T_{net} &= T * f = (T_2 * RPM^2 + T_1 * RPM + T_0) * (1 - V_z / V_{cr}) \\ Q_{net} &= Q * f = (Q_2 * RPM^2 + Q_1 * RPM + Q_0) * (1 - V_z / V_{cr}) \end{aligned}$$

To validate these equations, the experimental thrust values for a 5045BN propeller (at $V_z = 0$) were used as reference. The graph comparing the thrust predicted by BET and the experimental values of thrust for the given propeller is given below:

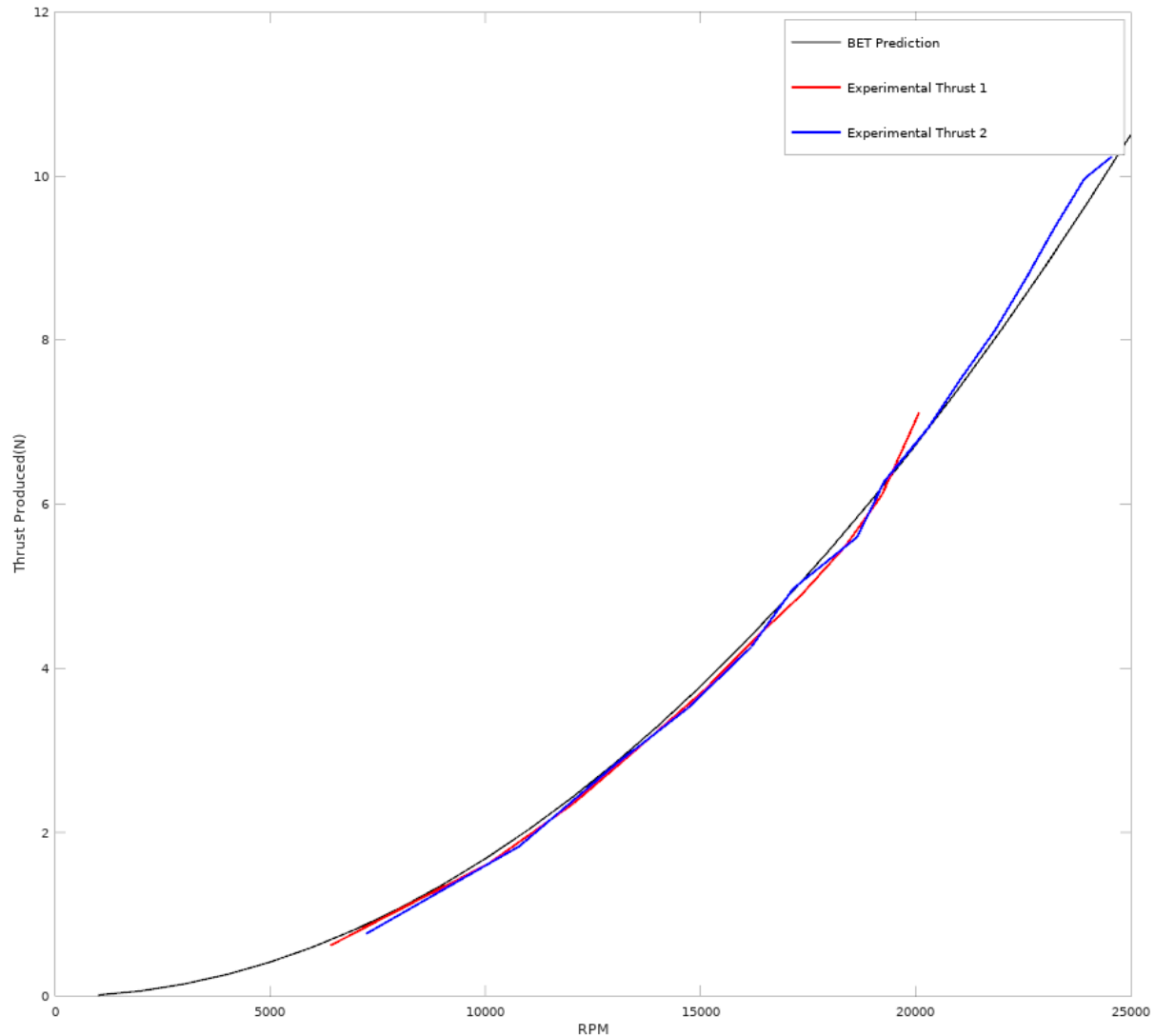


Figure 4.2 Comparison of BET predicted Thrust values and experimental values as in manufacturer datasheet

We can clearly see that the predicted thrust values are in excellent agreement with the experimental values, thus, the formed equations are correct and can be used to simulate thrust forces on our simulated model.

The forces on elevons due to deflection of the air stream is given by the equations:

$$F_{zR} = -1 * \rho * A_s * V_s^2 * \sin(\delta) * \sin(\delta)$$

$$F_{xR} = -1 * \rho * A_s * V_s^2 * \sin(\delta) * \cos(\delta)$$

Here,

F_{zR} : Force on elevon due to rotor air stream in the Z-direction

F_{xR} : Force on elevon due to rotor air stream in the X-direction

ρ : Density of air

A_s : Area over which the air stream acts on the elevon

V_s : Slipstream velocity of the air stream at the elevon

δ : deflection angle of the elevon

The slipstream velocity (V_s) of the air flow is calculated as:

$$V_s = (V_z + V_{err}) * (1 + a)$$

Here,

V_z : Forward velocity of the vehicle

a : Axial inflow factor for the propeller (depends on RPM)

V_{err} : This is a **fictional** parameter taken as a correction term to simulate the upstream velocity of the air flow even when the vehicle is at rest (chosen to be 0.1 m/sec).

The axial inflow factor depends on the rotation speed of the propeller (RPM) as well as the forward velocity of the vehicle. This dependence of this factor on RPM was calculated using the MATLAB script given in Appendix A and was found to be linear, and was found to be inversely proportional to the forward velocity so the final fitted polynomial equation for the axial inflow factor becomes:

$$a = a_1 * RPM / (V_z + V_{err}) + a_0$$

here,

a_1 and a_0 are the fitted polynomial coefficients with values:

$$a_1 = 0.00286$$

$$a_0 = 2.9948$$

There is no lift on the wings as they are symmetrical in this case.

4.6.2 Aerodynamic Forces due to motion:

As the vehicle moves through air with some velocity, it deflects the air it comes in contact with, that creates drag forces on the surface of the vehicle. These are broken in two parts:

- 1) Forces due to Translational Motion
- 2) Forces due to Rotational Motion

1) Forces due to Translational Motion:

The forces imparted on various surfaces of the vehicle can be generalized as the movement of a flat plate through a fluid. The forces in any direction can be calculated by change in momentum of the relative fluid flow as it strikes the surface of the plate.

The force that the plate experiences is given by the equation:

$$\mathbf{F}_i = -1 * \rho * A_i * \text{abs}(\mathbf{V}_i) * \mathbf{V}_i$$

Here,

F_i: Force in the i-direction (i= {x, y, z})

rho: density of air

A_i: Area of the surface

V_i: Velocity of the vehicle in i-direction in the body frame of reference

The **abs()** function has been used here to incorporate the change in direction due to change in the velocity direction directly without calculating the normal vector again.

For wings, A_i is given as:

$$A_x = 2 * A_{wx}$$

$$A_y = A_{wy}$$

$$A_z = 2 * A_{wz}$$

Here,

A_{wx}: area of the surface that has x-axis as its normal (in body frame)

A_{wy}: area of the surface that has x-axis as its normal (in body frame)

A_{wy}: area of the surface that has x-axis as its normal (in body frame)

For elevons, the force due to translational motion is given by:

$$F_{zT} = -1 * \rho * A_e * V^2 * \cos(\phi)^2 * \sin(\delta)$$

$$F_{xT} = -1 * \rho * A_e * V^2 * \cos(\phi)^2 * \cos(\delta)$$

Here,

A_e: Elevon surface Area

V: Net velocity of the vehicle

delta: deflection angle of the elevon relative to the body

phi: angle that the striking air makes with the elevon normal

$$\phi = \arccos(|V_x| / |V|) - \delta$$

2) Forces due to rotational motion

The aerodynamic forces on the wings due to rotational motion can be calculated simply by change in momentum of the air stream after striking the wing.

Currently, only the resultant force/torque due to rotation along the z-axis only has been considered. As this is the only dominant force out of the three, we focus on this only.

This condition is similar to a stream of air hitting a flat plate perpendicularly. The resultant total torque can be calculated by the change in momentum occurring, which is given by:

$$T_z = -2 * \rho * A_{wx} * L^3 * \text{abs}(W_z) * W_z$$

Here,

L: The distance of the point of application of force on both wings with the central axis of rotation.

W_z: Angular Velocity of the vehicle about the z-axis in the body frame

4.7 Further development plans

The simulation model developed can now be used as a test bed for testing new algorithms for dual-rotor VTOL UAV platforms. Our plan is to implement and test conventional control schemes on this model. Once that is achieved, we intend to implement Machine Learning Based control strategies, for which the development of this test bed was absolutely necessary (as we cannot let the real model fail and crash as many times as it wants!).

The plan is to use RNN based strategies to form a data-driven controller without any pre mathematical modelling of the vehicle being fed to the controller. The controller will have to figure out the optimal control strategy to go from point A to point B while following the given policies and constraints (time and safety limits). The scope of this approach is huge, as this opens a wide range of possibilities for forming different algorithms for path planning, swarm behavior, etc.

Due to current time limitation, only the development of the simulation model was completed as of the day of writing.

CHAPTER 5 – Results and Conclusions

For the Mechanical design part, the MCAD model was made using SolidWorks. The part files were exported in different formats to be used for analysis as well as for development of the simulation model. Fabrication of the First prototype of the vehicle had been completed before March, 2020.

Structural and flow analysis of the wings and motor mounts was performed and the following conclusions were drawn:

For Structural Analysis of Motor Mounts:

- Maximum Stress (8.5716 MPa) < Y.S. (35.9 MPa) so our design is valid according to stress considerations.
- Maximum Strain (3.0864e-04) < Theoretical Strain (3.7267e-03) hence, our design is also valid according to strain considerations.
- Diameter Range: 2.6535mm \leq D \leq 5.269mm

For flow analysis of Wings:

- Drag Force = 0.0066998 N
- Lift Force = 0.006409952 N
- It is concluded that Lift and Drag force effects will be negligible as the length of the aerofoil is small.
- Scaled Residuals of Continuity, X-Direction Velocity, Y-Direction Velocity, Z-Direction Velocity increases in the starting, reaches a peak then decreases with increase in the number of iterations.
- Lift coefficient first decreases then attains a minimum value then starts increasing in an irregular manner.
- Drag coefficient decreases in an almost exponential manner.

The simulation model developed can now be used as a test bed for testing new algorithms for dual-rotor VTOL UAV platforms. Our plan is to implement and test conventional control schemes on this model. Once that is achieved, we intend to implement Machine Learning Based control strategies, for which the development of this test bed was absolutely necessary (as we cannot let the real model fail and crash as many times as it wants!).

The future plan for this project includes implementing and testing efficient control schemes with an integrated transition behaviour. After that, swarm behaviour of this class of UAV can be tested. Further mechanical improvements include modifying the aerofoil design for higher payload capacities, with minimal change in the form factor of the vehicle.

Appendix A: MATLAB code for BET calculations

```
%% Calculates Thrust/Torque vs RPM relation for a given propeller using Blade
Element Theory
%% the t & q variables at the end represent the second order polynomial
coefficients for the same

clear, clc
%chord length of blade assumed constant with radius (meters)
chord=0.02;
%pitch distance in meters.
pitch=0.0254 * 4.5;
%diameter of the propeller (meters)
dia=0.0254 * 10;
%tip radius
R=dia/2.0;
%thickness to chord ratio for propeller section (constant with radius)
tonc=0.12*chord;
%standard sea level atmosphere density (kg/m^3)
rho=1.225;
% 2 Bladed Propeller
B=2;
% use 10 blade segments (starting at 10% R (hub) to R)
xs=0.1*R;
xt=R;
rstep=(xt-xs)/10;
r1=[xs:rstep:xt];
%calculate results for a range of velocities from 1 to 60 m/s
V = 0.1;
rpm = 1000:1000:20000;
for i=1:length(rpm)
    RPM = rpm(i);
    n = RPM / 60.0;
    omega = 2.0 * pi * n;
    %initialise sums thrust (N) torque (Nm)
    thrust=0.0;
    torque=0.0;
    a_sum = 0.0;
    %loop over each blade element
    for j=1:size(r1,2)
        rad=r1(j);
        %calculate local blade element setting angle
        theta=atan(pitch/2/pi/rad);
        %calculate solidity
        sigma=2.0*chord/2.0/pi/rad;
        %guess initial values of inflow and swirl factor
        a=0.1;
        b=0.01;
        %set logical variable to control iteration
        finished=false;
        %set iteration count and check flag
        sum=1;
        itercheck=0;
```

```

while (~finished)
    %axial velocity
    V0=V*(1+a);
    %disk plane velocity
    V2=omega*rad*(1-b);
    %flow angle
    phi=atan2(V0,V2);
    %blade angle of attack
    alpha=theta-phi;
    % lift coefficient
    cl=6.2 * alpha;
    %drag coefficient
    cd=0.008-0.003*cl+0.01*cl*cl;
    %local velocity at blade
    Vlocal=sqrt(V0*V0+V2*V2);
    %thrust grading
    DtDr=0.5*rho*Vlocal*Vlocal*B*chord*(cl*cos(phi)-cd*sin(phi));
    %torque grading
    DqDr=0.5*rho*Vlocal*Vlocal*B*chord*rad*(cd*cos(phi)+cl*sin(phi));
    %momentum check on inflow and swirl factors
    tem1=DtDr/(4.0*pi*rad*rho*V*V*(1+a));
    tem2=DqDr/(4.0*pi*rad*rad*rad*rho*V*(1+a)*omega);
    %stabilise iteration
    anew=0.5*(a+tem1);
    bnew=0.5*(b+tem2);
    %check for convergence
    if (abs(anew-a)<1.0e-5)
        if (abs(bnew-b)<1.0e-5)
            finished=true;
        end
    end
    a=anew;
    b=bnew;
    %increment iteration count
    sum=sum+1;
    %check to see if iteration stuck
    if (sum>500)
        finished=true;
        itercheck=1;
    end
end
a_sum = a_sum + a;
thrust=thrust+DtDr*rstep;
torque=torque+DqDr*rstep;
end
A(i) = (a_sum) / sum;
THRUST(i) = thrust;
TORQUE(i) = torque;
end
A_poly = polyfit(rpm, A, 1)
t = polyfit(rpm, THRUST, 2)
q = polyfit(rpm, TORQUE, 2)
plot(rpm, THRUST, 'b')
figure
plot(rpm, TORQUE, 'r')

```


Appendix B: The actuator control Gazebo Plugin Code

```
#ifndef _TERN_PLUGIN_HH_
#define _TERN_PLUGIN_HH_

#include <gazebo/gazebo.hh>
#include <gazebo/physics/physics.hh>
#include <gazebo/transport/transport.hh>
#include <gazebo/messages/messages.hh>
#include <gazebo/common/Timer.hh>
#include <vector>

#include <thread>
#include "ros/ros.h"
#include "ros/callback_queue.h"
#include "ros/subscribe_options.h"
#include "std_msgs/Float32.h"
#include "std_msgs/Time.h"
#include "roscpp_msgs/Clock.h"

#define MAX_SERVO_TRQ 10

namespace gazebo
{
  /// \brief A plugin to control a Velodyne sensor.
  class TernPlugin : public ModelPlugin
  {
    /// \brief A node use for ROS transport
    private: std::unique_ptr<ros::NodeHandle> rosNode;

    /// \brief A ROS subscriber
    private: ros::Subscriber rosSub;

    /// \brief A ROS callbackqueue that helps process messages
    private: ros::CallbackQueue rosQueue;

    /// \brief A thread the keeps running the rosQueue
    private: std::thread rosQueueThread;

    /// \brief A node used for transport
    private: transport::NodePtr node;

    /// \brief A subscriber to a named topic.
    private: transport::SubscriberPtr sub;

    /// \brief Pointer to the model.
    private: physics::ModelPtr model;

    // Pointer to the update event connection
    private: event::ConnectionPtr updateConnection;
  }
}
```

```

private:
    float actuator_input[4];
    physics::Joint_V actuator_joint;

    /// \brief A PID controller for the joint.
    private: common::PID pid;

    float curr_time, prev_time;

    // vector<vector<float>> PID_c;
    float Kp[4], Kd[4], Ki[4];

    ros::Subscriber actuators_sub[4];

    ros::Subscriber timer;

    rosgraph_msgs::Clock time_var;

    /// \brief Constructor
    public: TernPlugin() {}

    /// \brief The load function is called by Gazebo when the plugin is
    /// inserted into simulation
    /// \param[in] _model A pointer to the model that this plugin is
    /// attached to.
    /// \param[in] _sdf A pointer to the plugin's SDF element.
    public: virtual void Load(physics::ModelPtr _model, sdf::ElementPtr _sdf)
    {
        // Safety check
        if (_model->GetJointCount() == 0)
        {
            std::cerr << "Invalid joint count, tern plugin not loaded\n";
            return;
        }

        // PID constants for actuators (Kp, Kd, Ki)
        // PID_c = {{10.0, 1.0, 0.0},
        //           {10.0, 1.0, 0.0},
        //           {10.0, 1.0, 0.0},
        //           {10.0, 1.0, 0.0}};
        for (int j = 0; j < 2; j++) {
            Kp[j] = 0.01;
            Kd[j] = 0.00001;
            Ki[j] = 0.0;
            actuator_input[j] = 0;
        }

        for (int j = 2; j < 4; j++) {
            Kp[j] = 30.0;
            Kd[j] = 1.0;
            Ki[j] = 0.0;
            actuator_input[j] = 0;
        }

        // Store the model pointer for convenience.
        this->model = _model;
    }

```

```

// Get all the joints in the model as a Vector
this -> actuator_joint = _model -> GetJoints();

// Create the node
this->node = transport::NodePtr(new transport::Node());
#if GAZEBO_MAJOR_VERSION < 8
this->node->Init(this->model->GetWorld()->GetName());
#else
this->node->Init(this->model->GetWorld()->Name());
#endif

if (!ros::isInitialized())
{
int argc = 0;
char **argv = NULL;
ros::init(argc, argv, "gazebo_client",
ros::init_options::NoSigintHandler);
}

// Create our ROS node. This acts in a similar manner to
// the Gazebo node
this->rosNode.reset(new ros::NodeHandle("gazebo_client"));

// Create a named topic, and subscribe to it.
ros::SubscribeOptions solr;

// Creating the subscriber topics to get input values from
for (int i = 0; i < 4; i++) {
solr = ros::SubscribeOptions::create<std_msgs::Float32>(
"/" + this -> model -> GetName() + "/" + this ->
actuator_joint[i] -> GetName(),
1,
boost::bind(&TernPlugin::get_act_data, this, _1, i),
ros::VoidPtr(), &this -> rosQueue);
this -> actuators_sub[i] = this -> rosNode -> subscribe(solr);
}

solr = ros::SubscribeOptions::create<rosgraph_msgs::Clock>(
"/clock",
1,
boost::bind(&TernPlugin::time_cb, this, _1),
ros::VoidPtr(), &this -> rosQueue);
this -> timer = this -> rosNode -> subscribe(solr);

// Spin up the queue helper thread.
this->rosQueueThread =
std::thread(std::bind(&TernPlugin::QueueThread, this));

this->updateConnection = event::Events::ConnectWorldUpdateBegin(
std::bind(&TernPlugin::updateJointStates, this));

this -> seq = 0;
this -> err_fraction = 0;
}

```

```

/// \brief ROS helper function that processes messages
private: void QueueThread()
{
    static const double timeout = 0.01;
    while (this->rosNode->ok())
    {
        this->rosQueue.callAvailable(ros::WallDuration(timeout));
    }
}

void get_act_data(const std_msgs::Float32::ConstPtr& msg, int i) {
    this -> actuator_input[i] = msg -> data;
}

void time_cb(const rosgraph_msgs::Clock::ConstPtr& msg) {
    time_var = *msg;
}

float target_rpm[2];
float curr_error[4];
float prev_error[4];
float delta_t;
float control_torque;
float control_torque_tmp;
float err_fraction;
int seq;

float get_control_torque_elevon(int id) {
    this -> control_torque_tmp = this -> Kp[id] * this -> curr_error[id]
+ this -> Kd[id] * (float)(this -> curr_error[id] - this -> prev_error[id]) /
this -> delta_t;
    if (this -> control_torque_tmp > MAX_SERVO_TRQ) {
        return MAX_SERVO_TRQ;
    } else if (this -> control_torque_tmp < -MAX_SERVO_TRQ) {
        return -MAX_SERVO_TRQ;
    } else {
        return this -> control_torque_tmp;
    }
}

float diff_val;

void rotorControlRoutine(int id) {
    this -> diff_val = -1 * (this -> actuator_joint[id] -> GetVelocity(0)
- this -> actuator_input[id] * (2 * M_PI) / 60);
    this -> target_rpm[id] += diff_val / 10;
    // if (this -> target_rpm[id] < this -> actuator_input[id]) {
    //     this -> target_rpm[id] += 10.0;
    // } else if (this -> target_rpm[id] > this -> actuator_input[id]) {
    //     this -> target_rpm[id] -= 10.0;
    // }
    this -> curr_error[id] = -1 * (this -> actuator_joint[id] ->
GetVelocity(0) - this -> target_rpm[id] * (2 * M_PI) / 60);
    // std::cout << "Current Error: " << this -> curr_error[id] << "\n";
    // std::cout << "Current RPM: " << this -> actuator_joint[id] ->
GetVelocity(0) * 60 / (2 * M_PI) << std::endl;
    // std::cout << "Target RPM: " << this -> target_rpm[id] <<
std::endl;
}

```

```

this -> err_fraction = (this -> curr_error[id]) / this -> target_rpm[id];
// std::cout << "Fractional Error: " << err_fraction << std::endl;
if (abs(err_fraction) < 0.001) {
    return;
}
this -> delta_t = this -> curr_time - this -> prev_time;
if (this -> delta_t == 0) {
    return;
}
// PD control
// this -> control_torque = get_control_torque_elevon(id);
this -> control_torque = this -> Kp[id] * this -> curr_error[id] +
this -> Kd[id] * (float)(this -> curr_error[id] - this -> prev_error[id]) /
this -> delta_t;
// std::cout << "P: " << this -> Kp[id] * this -> curr_error[id] <<
std::endl;
// std::cout << "D: " << this -> Kd[id] * (float)(this ->
curr_error[id] - this -> prev_error[id]) / this -> delta_t << std::endl;
if (this -> control_torque > 10) {
    this -> control_torque = 10;
} else if (this -> control_torque <= -10) {
    this -> control_torque = -10;
}
// std::cout << "Control Torque: " << this -> control_torque << "\n";
this -> actuator_joint[id] -> SetForce(0, this -> control_torque);
// this->actuator_joint[id]->SetForce(1, 5);
this -> prev_error[id] = this -> curr_error[id];
return;
}

void elevonControlRoutine(int id) {
    this -> curr_error[id] = -1 * (this -> actuator_joint[id] ->
Position() - this -> actuator_input[id]);
// std::cout << "Current Error: " << this -> curr_error[id] << "\n";
if (this -> curr_error[id] < 0.001 && this -> curr_error[id] > -
0.001) {
    return;
}
this -> delta_t = this -> curr_time - this -> prev_time;
if (this -> delta_t == 0) {
    return;
}
// PD control
this -> control_torque = get_control_torque_elevon(id);
this -> actuator_joint[id] -> SetForce(0, this -> control_torque);
this -> prev_error[id] = this -> curr_error[id];
return;
}

public: void updateJointStates() {
// std::cout << "Seq: " << seq++ << std::endl;
this -> curr_time = time_var.clock.sec + (time_var.clock.nsec /
1000000000.0);
rotorControlRoutine(0);
rotorControlRoutine(1);
elevonControlRoutine(2);
elevonControlRoutine(3);
}

```

```
this -> prev_time = this -> curr_time;
    }
};

// Tell Gazebo about this plugin, so that Gazebo can call Load on this
plugin.
GZ_REGISTER_MODEL_PLUGIN(TernPlugin)
}
#endif
```

Appendix C: The Aerodynamics model Gazebo Plugin Code

```
#ifndef _TERN_AERO_PLUGIN_HH_
#define _TERN_AERO_PLUGIN_HH_

#include <gazebo/gazebo.hh>
#include <gazebo/physics/physics.hh>
#include <gazebo/transport/transport.hh>
#include <gazebo/messages/messages.hh>
#include <gazebo/common/Timer.hh>
#include <ignition/math.hh>
#include <vector>

#include <thread>
#include "ros/ros.h"
#include "ros/callback_queue.h"
#include "ros/subscribe_options.h"
#include "std_msgs/Float32.h"
#include "std_msgs/Time.h"
#include "rosgraph_msgs/Clock.h"

#define MAX_SERVO_TRQ 10
#define T2 1.2061e-7
#define T1 -4.1556e-7
#define T0 -1.9043e-6

#define Q2 1.3114e-9
#define Q1 -2.8595e-10
#define Q0 -2.3435e-8

#define D 0.254 // 0.0254 * 10"
#define pitch 0.127 // 0.0254 * 4.5"
#define rho 1.225 // air density in Kg/m^3
#define l 0.2145 // 0.629 / 2 - 0.10
// #define As(r) D^2 * (1 + a(r)) / (1 + 2 * a(r))
#define Verr 0.1
/* For Chosen Verr */
#define A1 0.00286
#define A0 2.9948
/* ***** */
#define As 2.06e-2 // D * 0.0811021
#define Ae 2.4e-2 // 0.0811021 * 0.199357 m^2
#define Awx 3.8e-2 // 0.20 * 0.1914711 m^2
#define Awy 8.177e-3 // 0.03 * 0.2725732 m^2
#define Awz 8.0e-3 // 2 * 0.02 * 0.20

namespace gazebo
{
  /// \brief A plugin to control a Velodyne sensor.
  class TernAeroPlugin : public ModelPlugin
  {
    /// \brief A node use for ROS transport
    private: std::unique_ptr<ros::NodeHandle> rosNode;
```

```

/// \brief A ROS subscriber
private: ros::Subscriber rosSub;

/// \brief A ROS callbackqueue that helps process messages
private: ros::CallbackQueue rosQueue;

/// \brief A thread the keeps running the rosQueue
private: std::thread rosQueueThread;

/// \brief A node used for transport
private: transport::NodePtr node;

/// \brief A subscriber to a named topic.
private: transport::SubscriberPtr sub;

/// \brief Pointer to the model.
private: physics::ModelPtr model;

// Pointer to the update event connection
private: event::ConnectionPtr updateConnection;

private:
    physics::Link_V links;
    physics::Joint_V joint;

/// \brief A PID controller for the joint.
private: common::PID pid;

float curr_time, prev_time;

ros::Subscriber actuators_sub[4];

ros::Subscriber timer;

rosgraph_msgs::Clock time_var;

int seq;

/// \brief Constructor
public: TernAeroPlugin() {}

/// \brief The load function is called by Gazebo when the plugin is
/// inserted into simulation
/// \param[in] _model A pointer to the model that this plugin is
/// attached to.
/// \param[in] _sdf A pointer to the plugin's SDF element.
public: virtual void Load(physics::ModelPtr _model, sdf::ElementPtr _sdf)
{
    // Safety check
    if (_model->GetJointCount() == 0)
    {
        std::cerr << "Invalid joint count, tern plugin not loaded\n";
        return;
    }
    // Store the model pointer for convenience.
    this->model = _model;
}

```



```

// Get the joints and links of the model as a vector
    this -> links = _model -> GetLinks();
    this -> joint = _model -> GetJoints();

    for (int i = 0; i < this -> links.size(); i++) {
        std::cout << "Link: " << this -> links[i] -> GetName() << " at
index: " << i << std::endl;
    }

    // Create the node
    this->node = transport::NodePtr(new transport::Node());
    #if GAZEBO_MAJOR_VERSION < 8
    this->node->Init(this->model->GetWorld()->GetName());
    #else
    this->node->Init(this->model->GetWorld()->Name());
    #endif

    if (!ros::isInitialized())
    {
        int argc = 0;
        char **argv = NULL;
        ros::init(argc, argv, "gazebo_client",
            ros::init_options::NoSigintHandler);
    }
    // Create our ROS node. This acts in a similar manner to
    // the Gazebo node
    this->rosNode.reset(new ros::NodeHandle("gazebo_client"));

    // Create a named topic, and subscribe to it.
    ros::SubscribeOptions solr;

    solr = ros::SubscribeOptions::create<rosgraph_msgs::Clock>(
        "/clock",
        1,
        boost::bind(&TernAeroPlugin::time_cb, this, _1),
        ros::VoidPtr(), &this -> rosQueue);
    this -> timer = this -> rosNode -> subscribe(solr);

    // Spin up the queue helper thread.
    this->rosQueueThread =
    std::thread(std::bind(&TernAeroPlugin::QueueThread, this));

    this->updateConnection = event::Events::ConnectWorldUpdateBegin(
        std::bind(&TernAeroPlugin::updateJointStates, this));

    this -> seq = 0;
}
///  

private: void QueueThread()
{
    static const double timeout = 0.01;
    while (this->rosNode->ok())
    {
        this->rosQueue.callAvailable(ros::WallDuration(timeout));
    }
}

```

```

void time_cb(const rosgraph_msgs::Clock::ConstPtr& msg) {
    time_var = *msg;
}

float omega, RPM;
float thrust, torque;
float Vcr;
ignition::math::Vector3d vel;

void setForceOnRotor(int id) {
    this -> omega = this -> joint[id] -> GetVelocity(0);
    this -> RPM = abs(this -> omega * 60 / (2.0 * M_PI));
    // std::cout << "RPM: " << this -> RPM << std::endl;
    if (abs(this -> RPM) <= 1000) {
        return;
    }
    this -> vel = this -> links[0] -> RelativeLinearVel();
    this -> Vcr = pitch * RPM / 60;
    // std::cout << "RPM: " << this -> RPM << std::endl;
    this -> thrust = (T2 * RPM * RPM + T1 * RPM + T0) * (1 - (vel.Z()) /
Vcr);
    this -> torque = (this -> omega < 0 ? -1 : 1) * (Q2 * RPM * RPM + Q1
* RPM + Q0) * (1 - (vel.Z()) / Vcr);
    // std::cout << "Applied Thrust: " << this -> thrust << std::endl;
    if (!isnan(this -> torque)) {
        std::cout << "Applied Torque: " << this -> torque << std::endl;
    }
    this -> links[2 + id] -> AddRelativeForce(ignition::math::Vector3d(0,
0, thrust));
    // std::cout << "Force on body: " << this -> links[0] ->
RelativeForce() << std::endl;
    this -> links[2 + id] ->
AddRelativeTorque(ignition::math::Vector3d(0, 0, torque));
    // std::cout << "Torque on body: " << this -> links[0] ->
RelativeTorque() << std::endl;
}

double FzR, FxR;
double FzM, FxM;
double Fx, Fz;
double phi;
double delta;
double Vnet;
double Vs;
double a;

void setForceOnElevon(int id, int r_id, int l_id) {
    this -> Fx = this -> FxM = this -> FxR = 0;
    this -> Fz = this -> FzM = this -> FzR = 0;
    this -> torque = 0;
    this -> omega = this -> RPM = this -> delta = this -> Vnet = this ->
Vs = this -> a = 0;

    this -> omega = this -> joint[r_id] -> GetVelocity(0);
    this -> delta = -1 * this -> joint[id] -> Position();
    this -> RPM = this -> omega * 60 / (2.0 * M_PI);

```

```

if (!isnan(this -> RPM)) {
    // std::cout << "RPM: " << this -> RPM << std::endl;
}
// std::cout << "Delta: " << this -> delta << std::endl;
this -> vel = this -> links[0] -> RelativeLinearVel();
this -> Vnet = this -> vel.Length();
this -> a = A1 * RPM / (Vnet + Verr) + A0;
this -> Vs = (this -> vel.Z() + Verr) * (1 + a);
if (this -> RPM != 0) {
    this -> FzR = -1 * rho * As * (this -> Vs * this -> Vs) *
sin(this -> delta) * sin(this -> delta);
    this -> FxR = -1 * rho * As * (this -> Vs * this -> Vs) *
sin(this -> delta) * cos(this -> delta);
}
this -> phi = acos(this -> vel.X() / this -> Vnet) - this -> delta;
this -> FzM = -1 * rho * Ae * (this -> Vnet * this -> Vnet) *
cos(this -> phi) * cos(this -> phi) * sin(this -> delta);
this -> FxM = -1 * rho * Ae * (this -> Vnet * this -> Vnet) *
cos(this -> phi) * cos(this -> phi) * cos(this -> delta);
this -> Fz = this -> FzR + this -> FzM;
this -> Fx = this -> FxR + this -> FxM;
this -> torque = -1 * 0.045 * (Fx * cos(this -> delta) + Fz *
sin(this -> delta));
if (isnan(this -> Fx) || isnan(this -> Fz) || isnan(this -> torque))
{
    return;
}
// std::cout << "Slipstream Velocity: " << this -> Vs << std::endl;
// std::cout << "Z Velocity: " << this -> vel.Z() << std::endl;
// std::cout << "Z force: " << this -> Fz << std::endl;
// std::cout << "X force: " << this -> Fx << std::endl;
// std::cout << "Torque: " << this -> torque << std::endl;
// this -> links[l_id] -> AddLinkForce(ignition::math::Vector3d(Fx,
0, Fz), ignition::math::Vector3d(0, 0, -0.045));
this -> links[l_id] -> AddRelativeForce(ignition::math::Vector3d(this
-> Fx, 0, this -> Fz));
this -> links[l_id] -> AddRelativeTorque(ignition::math::Vector3d(0,
torque, 0));
// std::cout << "Force on Elevon " << id << ": " << this ->
links[l_id] -> RelativeForce() << std::endl;
}

float FbT[3];
float TbR[3];
float thetab[3];
ignition::math::Vector3d body_omega;
void setBodyForces() {
    this -> vel = this -> links[0] -> RelativeLinearVel();
    this -> FbT[0] = -1 * rho * 2 * Awx * abs(this -> vel.X()) * this ->
vel.X();
    this -> FbT[1] = -1 * rho * Awy * abs(this -> vel.Y()) * this ->
vel.Y();
    this -> FbT[2] = -1 * rho * 2 * Awz * abs(this -> vel.Z()) * this ->
vel.Z();
    this -> links[0] -> AddRelativeForce(ignition::math::Vector3d(FbT[0],
FbT[1], FbT[2]));
}

```

```

this -> body_omega = this -> links[0] -> RelativeAngularVel();
    // std::cout << "Body Angular Vel: " << this -> body_omega <<
std::endl;
    this -> TbR[3] = -2 * rho * Awx * l * l * l * abs(this ->
body_omega.Z()) * this -> body_omega.Z();
    this -> links[0] -> AddRelativeTorque(ignition::math::Vector3d(0, 0,
TbR[3]));
    }

    public: void updateJointStates() {
        seq++;
        if (seq < 2000) {
            return;
        }
        // std::cout << "Seq: " << seq++ << std::endl;
        this -> curr_time = time_var.clock.sec + (time_var.clock.nsec /
1000000000.0);
        setForceOnRotor(0);
        setForceOnRotor(1);
        setForceOnElevon(2, 0, 4);
        setForceOnElevon(3, 1, 5);
        setBodyForces();
        // setMotionForces();

        this -> prev_time = this -> curr_time;
    }
};

// Tell Gazebo about this plugin, so that Gazebo can call Load on this
plugin.
GZ_REGISTER_MODEL_PLUGIN(TernAeroPlugin)
}
#endif

```

References:

- 1) <https://www.altiuas.com/#>
- 2) https://www.researchgate.net/publication/323396673_Design_and_Fabrication_of_Small_Vertical-Take-Off-Landing_Unmanned_Aerial_Vehicle
- 3) https://www.researchgate.net/publication/317932848_Design_and_implementation_of_a_quadrotor_tail-sitter_VTOL_UAV
- 4) <https://www.flyingmachinearena.ethz.ch/wp-content/uploads/ritzIEEE17.pdf>
- 5) <https://emaxmodel.com/media/wysiwyg/RS2205-2.jpg>
- 6) [http://staff.fit.ac.cy/eng.fm/classes/amee202/Homework%20Assignment%20\(Momentum%20Equation\).pdf](http://staff.fit.ac.cy/eng.fm/classes/amee202/Homework%20Assignment%20(Momentum%20Equation).pdf)
- 7) <http://www.aerodynamics4students.com/propulsion/blade-element-propeller-theory.php>
- 8) <http://www.aerodynamics4students.com/propulsion/blade-element-propeller-theory.php>
- 9) Glauert H. The Elements of airfoil and airscrew theory. 2nd Edition. (Cambridge, 1947)