Design And Fabrication of Unmanned Aerial Manipulative System

A Final Year Project Report

Presented to

SCHOOL OF MECHANICAL & MANUFACTURING ENGINEERING

Department of Mechanical Engineering

NUST

ISLAMABAD, PAKISTAN

In Partial Fulfillment

of the Requirements for the Degree of

Bachelor of Mechanical Engineering

By

Muhammad Sheraz Amin

Asad Ullah

Rana Muhammad Hunain

June 2023

EXAMINATION COMMITTEE

We hereby recommend that the final year project report prepared under our supervision by:

Muhammad Sheraz Amin	289384
Asad Ullah	307158
Rana Muhammad Hunain	288959

Titled: "Design and Fabrication of Unmanned Aerial Manipulative System" be accepted in partial fulfillment of the requirements for the award of Mechanical Engineering degree.

Supervisor: Dr. Muhammad Jawad Khan, Assistant Professor, Department of Robotics and Intelligent Machine Engineering, SMME	Dated:
Committee Member:	
Committee Member:	

(Head of Department) (Date)

COUNTERSIGNED

Dated:

(Dean / Principal)

Abstract

As usage of multi-purpose drones is skyrocketing, many innovations are a foot. The aerial manipulator, a combination of a robotic arm and a flying platform, has emerged as a cutting-edge technology with diverse applications in various fields, such as disaster response, environmental monitoring, and infrastructure inspection. In this report, we investigate the design and implementation of a 3-DOF robotic arm mounted on a hexacopter, simulated using the Robot Operating System (ROS) and structurally analyzed using ANSYS. We focus on the applications of this technology in active environments, where tasks are hazardous or difficult for human operators.

The aerial manipulator can be equipped with sensors, cameras, and other tools, making it an ideal solution for tasks such as search and rescue, inspection of pipelines and power lines, and monitoring of forest fires. In addition, the aerial manipulator can reach locations that are inaccessible to ground-based robots, making it a valuable asset in disaster response scenarios.

To assess the market potential of aerial manipulators, we have analyzed data from various sources, including industry reports and academic publications. The market for aerial manipulators is projected to grow significantly in the coming years, driven by the increasing demand for unmanned aerial vehicles (UAVs) in various industries. According to a report by MarketsandMarkets, the global market for UAVs is expected to reach $55.8 billion by 2025, with aerial manipulators being one of the fastest-growing segments.

Our simulations using ROS have demonstrated the feasibility of the proposed design, highlighting the importance of optimizing the configuration and control of the aerial manipulator to achieve maximum efficiency and accuracy. The major capability of aerial manipulator is Pick and Place. We believe that the aerial manipulator has the potential to revolutionize the way we perform tasks in hazardous and inaccessible environments and automotive tasks as delivery service.

In conclusion, our study has demonstrated the potential of the aerial manipulator in various applications, including those where human intervention is limited or impossible. With the market for aerial manipulators projected to grow rapidly in the coming years, we believe that this technology will play a vital role in shaping the future of unmanned aerial systems.

Acknowledgements

Firstly, we give Allah, the Almighty, praise and thanks for His bounties that enabled us to successfully complete the research for our Final Year project.

We would like to express our heartfelt gratitude to our research supervisor, Dr. Muhammad Jawad Khan, Assistant Professor and Head, Aerial Robotics Lab SINES, NUST for his invaluable guidance and support throughout the entire duration of our Final Year Project. Without his expertise, dedication, and encouragement, we would not have been able to successfully complete our research project.

Dr. Muhammad Jawad Khan has consistently provided us with constructive feedback, insightful suggestions, and timely assistance whenever we needed it. He was always there to answer our questions, clarify our doubts, and steer us in the right direction. His expertise and experience have been crucial in helping us navigate the various challenges we encountered during the course of our project.

Furthermore, he has been an excellent role model, demonstrating the importance of hard work, diligence, and perseverance. We have learned so much from his/her example and are grateful for the opportunity to work under his/her mentorship.

In conclusion, we would like to extend our heartfelt appreciation to Dr. Muhammad Jawad Khan for his unwavering support and guidance throughout our project. We are truly grateful for everything he has done for us and will always remember his contribution to our success.

We are particularly appreciative of the School of Mechanical and Manufacturing Engineering for giving us the chance to do research in this rapidly expanding area. We owe a debt of gratitude to the academics and members of our thesis committee for their invaluable advice, support, and direction during this effort.

We would like to express our gratitude to our friends and fellow undergraduate mechanical engineering students at SMME NUST for their advice and guidance as we informally discussed our projects during lunch breaks.
For their ongoing, unconditional financial and emotional support, without which our initiative could not have been feasible, we are grateful to our parents and relatives.
Finally, we would like to express our gratitude to everyone who helped us, directly or indirectly, to finish the study project.

Muhammad Sheraz Amin

Asad Ullah

Rana Muhammad Hunain

report

ORIGINALITY REPORT

7%	5%	3%	4%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	Submitted to Higher Education Commission Pakistan Student Paper	1%
2	Xiangdong Meng, Yuqing He, Feng Gu, Liying Yang, Bo Dai, Zhong Liu, Jianda Han. "Design and implementation of rotor aerial manipulator system", 2016 IEEE International Conference on Robotics and Biomimetics (ROBIO), 2016 Publication	1%
3	www.ncbi.nlm.nih.gov Internet Source	1%
4	dokumen.pub Internet Source	<1%
5	dspace.mit.edu Internet Source	<1%
6	Submitted to Cebu Institute of Technology Student Paper	<1%
7	www.coursehero.com Internet Source	<1%

Table of Contents

LIST OF TABLES

LIST OF FIGURES

Chapter 1: Introduction

1.1 Motivation

Aerial manipulators have gained significant attention in recent years due to their ability to perform a wide range of tasks in various industries. The global market for aerial manipulators is projected to witness substantial growth in the coming years, with a CAGR of 17.2% from 2021 to 2028.

The market for aerial manipulators is driven by several factors, including the growing demand for automation and the need to increase efficiency and safety in various industries. Additionally, advancements in technology, such as the development of lightweight materials and sensors, are contributing to the increased adoption of aerial manipulators.

The market is expected to continue its growth trajectory in the coming years, with the Asia-Pacific region projected to be the fastest-growing market due to the high demand for automation in industries such as manufacturing and agriculture. However, challenges such as high costs and regulatory hurdles may impede the market's growth. North America and Europe are expected to adopt the aerial manipulator market faster, owing to the increasing adoption of UAVs for various applications, including military and defense and oil and gas. Ukraine war has demonstrated the efficiency of drones on battlefield where drones being used from reconnaissance to bomb dropping.

Aerial manipulator market presents a significant opportunity for manufacturers and investors looking to capitalize on the growing demand for automation.

1.2 Versatility

Main feature of aerial manipulators is the versatility they possess credited to a unique combination of mobility of hexacopter and interaction of robotic arm with surroundings. The versatility of aerial manipulators is one of their most significant advantages. With the ability to fly and maneuver in three-dimensional space, they can access and perform tasks in locations that are challenging or impossible for ground-based robots. Additionally, they can carry various tools and sensors, enabling them to perform a wide range of tasks, from inspection and maintenance to disaster response and environmental monitoring. This versatility makes aerial manipulators an attractive solution for industries such as agriculture, construction, and energy, where tasks often involve difficult-to-reach locations or hazardous conditions.

1.3 Problem Statement

To design and develop a UAM platform in which robotic arm is integrated on UAV such that external disturbances caused by the arm on UAV are minimum.

Our energies are focused on following three design consideration:

1.3.1 Light Weight/ Low Inertia Arm

The significance of employing materials and manufacturing processes that minimize the weight of the robotic arm cannot be overstated, as it decreases the resistance to displacement and maneuverability. This, in turn, necessitates the use of less powerful actuators, which, in conjunction with a smaller battery, can contribute to a lighter overall weight of the robot.

By prioritizing lightweight construction, we can enhance the efficiency and agility of the robotic arm, enabling it to operate with greater precision and speed. This approach is not only practical but also essential in ensuring the robot's longevity and functionality.

Moreover, the benefits of lightweight construction extend beyond the robot itself, as it can contribute to the overall sustainability and cost-effectiveness of the project. By reducing the size and weight of components, we can minimize resource consumption and waste generation while optimizing the efficiency of the system.

1.3.2 COG Mechanism

As the robotic arm traverses through a 3D Euclidean space, the displacement of mass is spread over a greater distance. This, in turn, causes a displacement of the combined center of gravity (COG) of the aerial manipulator, leading to an imbalanced weight distribution and thrust misplacement that generates moments on the drone platform. Such distortions can compromise the stability of the drone during its operations and pose a risk to its reliability.

To mitigate these challenges, careful consideration must be given to the design and placement of the robotic arm and its components. Furthermore, the selection of materials must be optimized to ensure a balance between strength and weight. The use of advanced algorithms and control systems can also aid in maintaining the stability and trajectory of the drone.

1.3.3 Manufacturing Cost

Our manufacturing process will utilize easily accessible techniques such as 3D printing, employing materials that are both economical and manageable. We have opted for PLA, a material with attributes that fulfill the aforementioned requirements, to construct the robotic arm. This methodology will enable us to curtail both expenses and time. The assembly of the drone will also be a streamlined procedure, with an emphasis on the construction of the frame and controls.

By embracing novel methods, we can improve the quality of our outputs while minimizing the impact on our resources. Thus, we are confident that our approach will enable us to achieve our objectives in a sustainable, cost-effective, and timely manner.

Chapter 2: Literature Review

A thorough summary of the literature review is noted down below and useful concepts which we found helpful in our project are listed down and expounded upon.

2.1 Centralized and Decentralized Control:

In the context of aerial manipulator control, a centralized approach involves a single controller that coordinates the motion and manipulation of the aerial vehicle and its end-effector. This controller receives feedback from various sensors and calculates the required control inputs to achieve the desired task.

On the other hand, a decentralized approach involves multiple controllers that work together to achieve the desired task. In this approach, each controller is responsible for a specific task, such as controlling the aerial vehicle or the end-effector and communicates with other controllers to coordinate their actions.

Both approaches have their advantages and disadvantages. A centralized approach can provide more precise and coordinated control of the aerial manipulator, but it can also be more complex and less resilient to failures.

A decentralized approach can be more robust to failures and easier to implement, but it may be less precise and coordinated than a centralized approach. The choice between a centralized and decentralized approach will depend on the specific requirements and constraints of the aerial manipulator task.

2.2 Adaptive Sliding Control:

Adaptive sliding control is a control technique that is used to control uncertain systems, particularly those with unknown or varying parameters. Adaptive sliding control can handle uncertainties while ensuring convergence to the desired trajectory. Aerial manipulation is a complex task that involves picking and releasing objects while the aerial vehicle is hovering. One of the crucial aspects of accurate manipulation is position holding, which can be disrupted by sudden changes in the quadrotor's attitude caused by the additional torque generated during object grabbing or releasing. Therefore, the controller for aerial manipulation must be able to address challenges such as battery drainage, miscalculated mechanical properties, measurement bias, and noise. It is essential to ensure that the controller can handle these issues to maintain the stability and precision of the aerial manipulator.

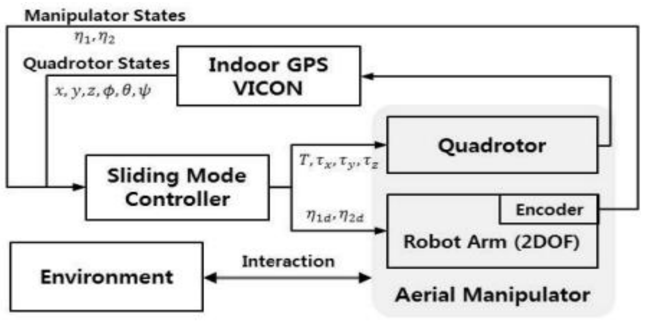


Figure 2.1: Adaptive Sliding Control

2.3 Topology Optimization:

One of the main remedies of COG shifting problem is the topology optimization of robotic arm/manipulator. Topology optimization is a design optimization technique that aims to find the optimal distribution of material within a given design domain to achieve a desired set of performance criteria. The technique uses mathematical algorithms to iteratively optimize the topology or shape of a structure, while satisfying constraints on performance, such as maximum stress, minimum weight, or maximum stiffness in our case lightweight frame without compromising structural integrity.

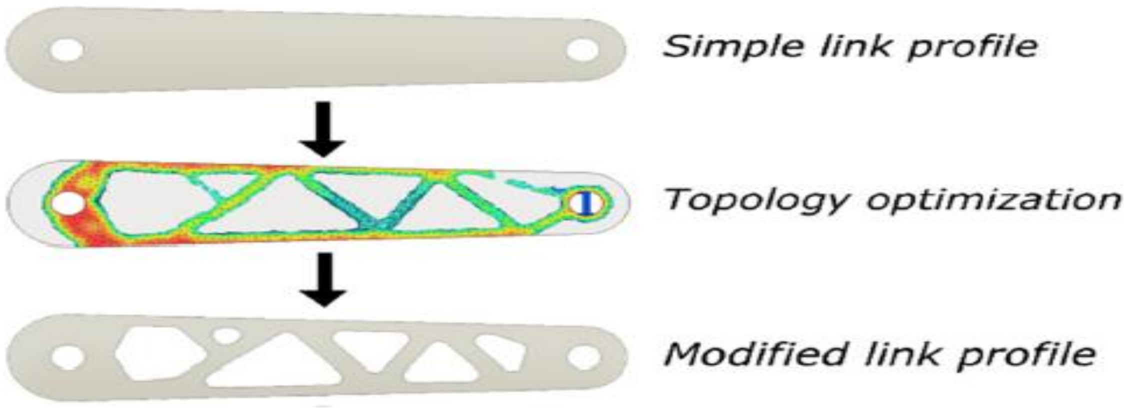


Figure 2.2: Topology Optimization

2.4 Quadcopter Dynamics:

A quadcopter is operated by controlling the RPM of its rotors, which, in turn, regulates the lift, torque, and thrust of the rotors. Compared to other UAVs, quadcopters are relatively small, allowing them to perform complex aerial maneuvers. However, due to their size, precise control of the quadcopter's angles is necessary for performing these complex maneuvers accurately. Therefore, careful handling of the angles is crucial to achieving the required precision in flight control.

2.4.1 Take off:

To lift off the ground, a quadcopter must generate force equal to or greater than gravity. This is achieved by controlling the engine and propeller direction to manage the lift and descent of the aircraft. As the propeller blades spin, they push air downwards, and according to Newton's Third Law, an equal and opposite reaction force is generated, pushing the rotor upwards. Thus, the speed of the rotor determines the amount of lift generated - faster rotation leads to more lift and vice versa.

2.4.2 Hovering:

To achieve hovering, net thrust of all six rotors must be equal to weight of drone and weight of the robotic manipulator with all its accessories.

2.4.3 Thrust:

Rotating the propeller of a quadcopter generates an orthogonal force, known as thrust, which propels the aircraft in the direction of the force. Equation of thrust:

$$T = \frac{\pi}{4} D^2 \rho v \Delta v$$

However, the magnitude of thrust depends on the real-time air density, velocity (v), and diameter (D) of the propeller. Neglecting changes in air density can compromise the performance of the rotor, as it plays a critical role in determining the amount of thrust generated. Therefore, it is crucial to consider environmental conditions when estimating the thrust produced by the propeller.

2.5 Flight Controls:

The hex-rotor aerial robot system consists of four sub-systems, including the flight control sub-system, sensor sub-system, communication sub-system, and ground station sub-system.

2.5.1 PixHawk

Pixhawk is an open-source autopilot platform designed for unmanned aerial vehicles (UAVs) and other robotic systems. The Pixhawk platform consists of a main processor board, peripheral sensors, and an array of input/output connectors that can be customized to suit specific needs. It is equipped with a suite of sensors, including accelerometers, gyroscopes, magnetometers, barometers, and GPS, which provide accurate and reliable data for navigation, stabilization, and control of the UAV.

Pixhawk runs on the ArduPilot firmware, which is a popular open-source autopilot software that provides a wide range of features and functions, including autonomous flight, mission planning, and telemetry. It supports various communication protocols, such as MAVLink, which enables real-time communication between the UAV and the ground station.

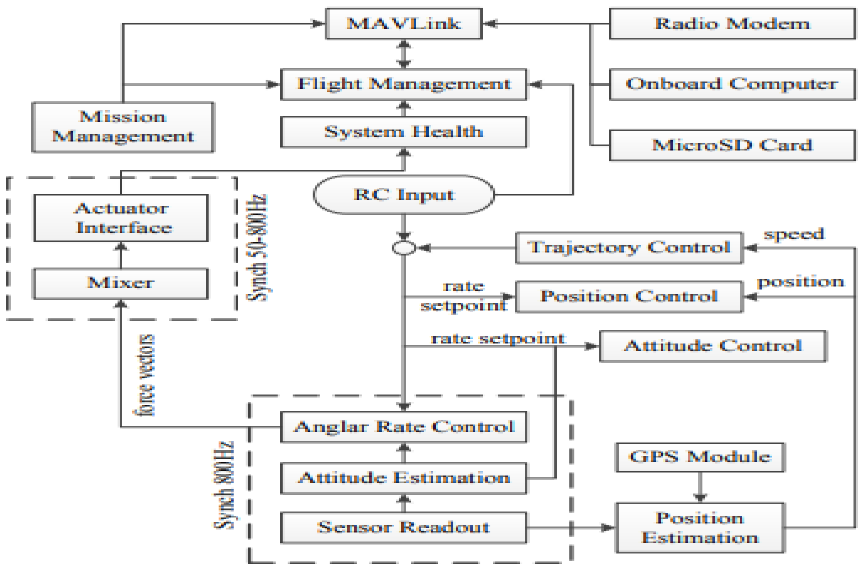


Figure 2.3: Flight Control Layout

2.5.2 Overall Control Configuration

RAMS (Rotor Aerial Manipulator System) consists of an onboard part and a ground part that communicates wirelessly. The onboard part stabilizes the RAR(rotor aerial robot), performs aerial manipulation tasks, and transmits data to the ground. The Pixhawk autopilot is the core component, providing real-time flight state information and recording various data. The ground part includes a ground station and operator, responsible for commanding the aerial manipulator, performing ground operations, and switching to manual control during emergencies. Effective communication between the two parts is crucial for secure and efficient aerial manipulation.

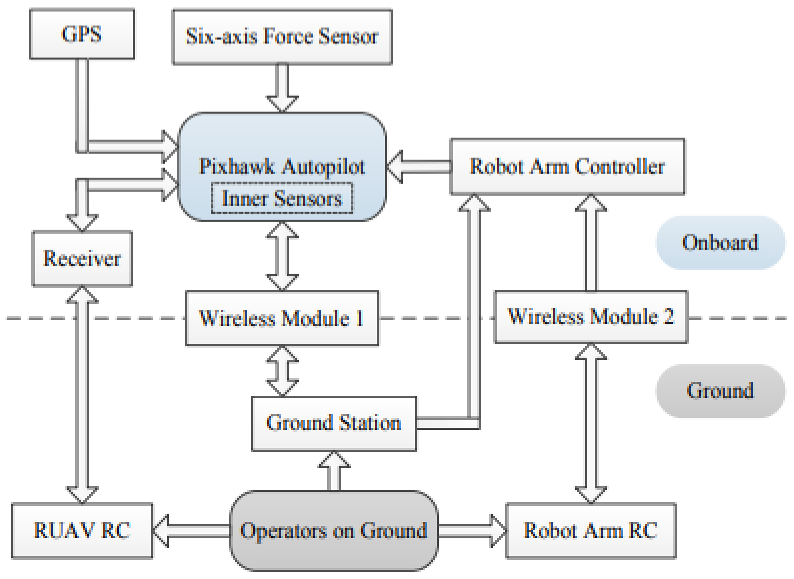


Figure 2.4: Overall Control Configuration

2.6 Dynamics of 3DOF manipulator

Dynamics of manipulators are quite a complex task involving various facets of mathematics like tensor calculus. This leads to many novel terminologies.

2.6.1 Euler- Lagrange Approach

The Euler-Lagrange approach is a mathematical framework used to derive the equations of motion for a mechanical system. It is commonly used in control systems. The approach is based on the principle of stationary action, which states that the actual motion of a system is the one that minimizes the action integral over all possible paths. The action is defined as the integral of the Lagrangian, which is a function that describes the kinetic and potential energy of the system.

$$\mathcal{L} = \mathcal{K} - \mathcal{U}$$

Lagrangian is first defined for the system, and then the Euler-Lagrange equations are applied. These equations are a set of partial differential equations that describe the motion of the system in terms of the Lagrangian.

Euler-Lagrangian approach yields the following expression for n-link manipulator.

$$\sum_{j} d_{kj}(q)\ddot{q}_j + \sum_{i,j} C_{ijk}(q)\dot{q}_i \dot{q}_j + \phi_k(q) = \tau_k$$

In more general form this is written as,

$$D(q)\ddot{q} + C(q,\dot{q})\dot{q} + G(q) = \tau$$

Where D(q) represents all inertial effects of all links, C$(q, \dot{q})$ denotes the centripetal and Coriolis force effects on link motions also known as velocity-product term and G(q) is the effect of gravity on every link mass as it changes position in Euclidean space.

$$D(q) = \sum_{i=1}^{n} m_i {J^T}_{v_i} J_{v_i} + \sum_{i=1}^{n} {J^T}_{w_i} R_i I_b {R^T}_i J_{w_i}$$

Masses, rotation matrices and inertia tensors of every link are required along with transrational and rotational Jacobians. MATLAB is used to compute all matrix multiplications and additions.

G(q) is easily computable. It is a gravity vector which includes potential energies of every link.

$$P = \sum_{i=1}^{n} P_i = \sum_{i=1}^{n} g r_{ci} m_i$$

Here r_{ci} is the height of center of mass of each link. But we use angular displacements(ɵ) of links as its substitute. Because both are analogous to each other.

2.6.2 Christoffel Number:

In the context of dynamic modeling of an n-link manipulator, the Christoffel symbols represent the effects of the Coriolis and centripetal forces on the motion of the system. These forces arise from the fact that the motion of each link is affected by the motion of all the other links, due to the geometry of the system.

$$C_{ijk} = \frac{1}{2}\{\frac{\partial d_{kj}}{\partial \theta_i} + \frac{\partial d_{ki}}{\partial \theta_j} - \frac{\partial d_{ij}}{\partial \theta_k}\}$$

C_{ijk} represents the “Christoffel symbols” of degree 1 . It is derived using tensor calculus. It is also called coefficient of velocity product. In above mentioned equation, i=j case is centripetal while i≠j is Coriolis effect of velocities.

Chapter no 3: Methodology

The previous chapter conducted a literature review in which various typed of models for aerial manipulators were explored. The major research problem which was found the existing research was the proper integration of UAV with Robotic Arm and to reduce the external disturbance caused by the robotic manipulator on the aerial platform stability.

We saw in previous papers that various techniques were used to achieve above mentioned objective. In one paper a timing belt mechanism was used to increase the mass distribution near the base. In one six axis torque and force sensors were utilized to give the feedback to control loop to minimize the disturbances.

In our design, using all these results of previous research conducted we will practically demonstrate a prototype implementation of this concept.

3.1 UAV Design Calculations:

3.1.1 Thrust Calculations:

From Bench Test we found the following thrust value at certain RPMs:

Table 3.1: Propeller Data

RPMs	Current (A)	Voltage (V)	Thrust (g)
3925	2.03	11.23	212
7204	13.81	10.82	770
6360	9.3	10.97	620

By using numerical method of divided difference to approximate a polynomial to estimate thrust or rpm at any given point, following relation is generated:

$$RPM = 3925 + 5.968137(T - 212) - 0.0006119540024(T - 212)(T - 620)$$

At Different RPMs we calculated the values of Thrust using that equation

Table 3.2: Thrust Calculations

Estimated Values		APC Values		Error
RPMs	Thrust (g)	RPMs	Thrust	
4000	224.07647	4000	259.6306	13.69
5000	387.9367	5000	406.6945	4.61
6000	557.4645	6000	585.8412	4.84
7000	733.29179	7000	798.9568	8.20

The Thrust values we calculated are almost equal to the actual Thrust values from manufacturers website with some error.

Now Thrust calculated by APC propeller's website:

At 6000 RPM

$$Total\ Thrust = 3515.04\ (g)$$

$$Hovering\ Thrust = Total\ Thrust/2$$

$$= 1757.52\ (g)$$

At 7000 RPM

$$Total\ Thrust = 4793.74\ (g)$$

$$Hovering\ Thrust = Total\ Thrust/2$$

$$= 2396.87\ (g)$$

3.1.2 Flight Time Calculations:

The estimated flight time for our UAV can be calculated using various parameters specified by our selected components:

$$Battery = 5200mAH$$

$$= 5.2AH$$

$$Maximum\ current\ drawn\ by\ one\ motor = 13A$$

$$Total\ current\ Drawn\ by\ 6\ Motors\ =\ 78\ A$$

$$\text{Flight Time} = \frac{Battery\ Capacity(AH) * 60}{Total\ Current\ Drawn}$$

$$Flight\ Time = \frac{5.2 * 60}{78}$$

$$Flight\ Time = 4\ min$$

$$Recommended\ Flight\ Time\ =\ Factor\ of\ Safety *\ Flight\ Time$$

$$=\ 0.7 * 4$$

$$Recommended\ Flight\ Time =\ 2.8\ minutes$$

3.2 Design of Robotic Arm:

The Manipulator is crucial aspect of aerial manipulator. We designed the arm using the iterative approach.

3.2.1 Initial Design:

The initial design of the robotic arm was designed using SolidWorks. This is a 3 Degree of Freedom arm with 1 DOF End effector. The arm design had various features such as low weight, distribution of mass at optimum position. The connecting rod mechanism to place the actuator near the base of the UAV. The arm is shown in figure as below:

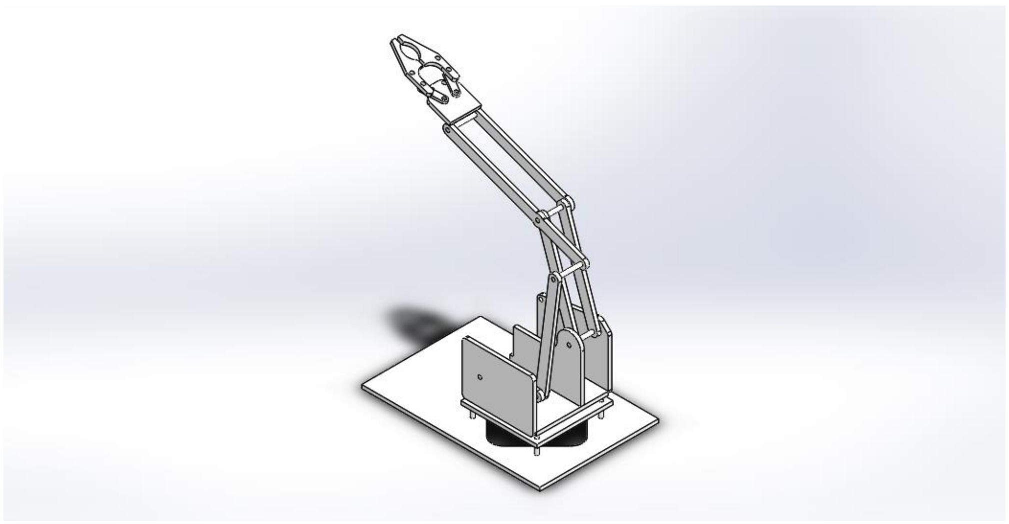


Figure 3.1: Initial CAD Design

The problem with this design is that the structural integrity of arm is very less. The arm has low structural strength making it very unlikely to withstand various impacts during the testing phase.

We need our robotic arm to be structurally rigid to make it able to withstand the impacts. We have to choose a middle way between mass distribution and structural strength.

3.2.2 Finalized design:

The final design of the robotic manipulator is designed by considering all the necessary factors into account. This is 3 Degree of Freedom Arm. It contains three revolute joints and an end effector mounted at the end.

The main features of final design are:

1. For uniform weight distribution links are attached on both sides, in this way UAV's stability would not be affected as weight distribution is equal on both sides.
2. The structural integrity of the arm is increased.

The SolidWorks CAD model of the arm is shown as below:

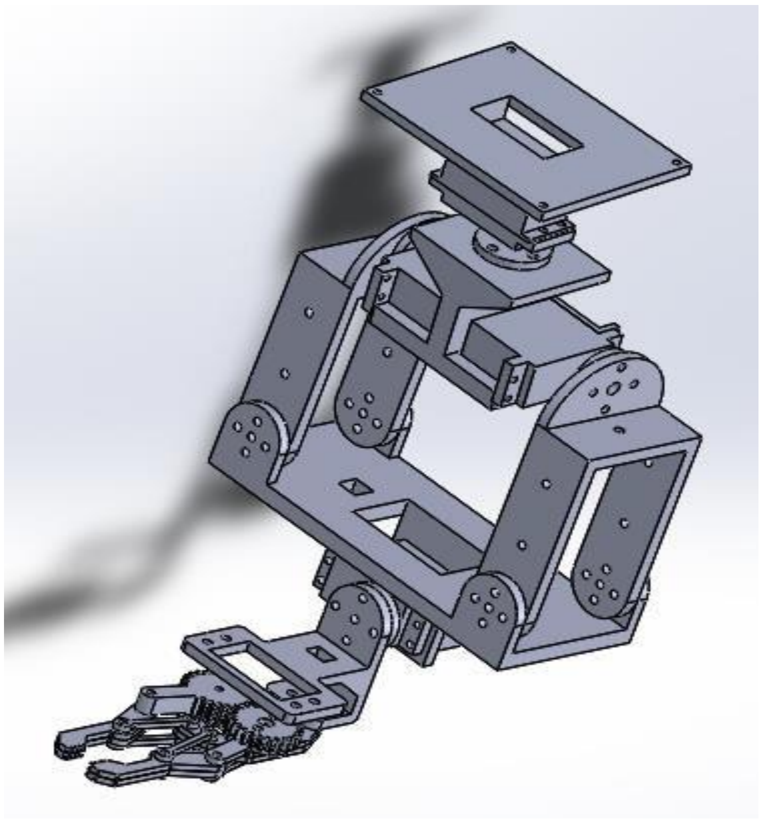


Figure 3.2: Final CAD Model of Robotic Arm

The robotic Manipulator has two main components:

1) Robotic Arm
2) End Effector

3.2.3 End Effector:

The end effector design is Mechanically Robust. It can grasp fairly large objects and have a strong grip. The parent gear will be derived from servo motor, this gear then transmits the motion further.

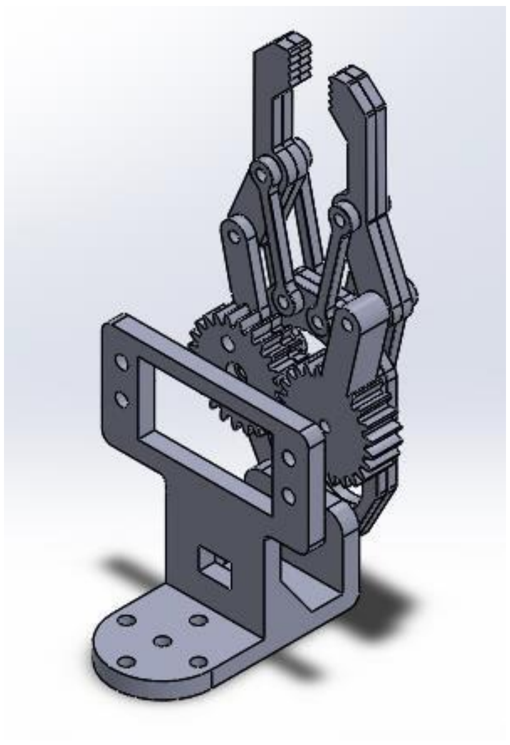
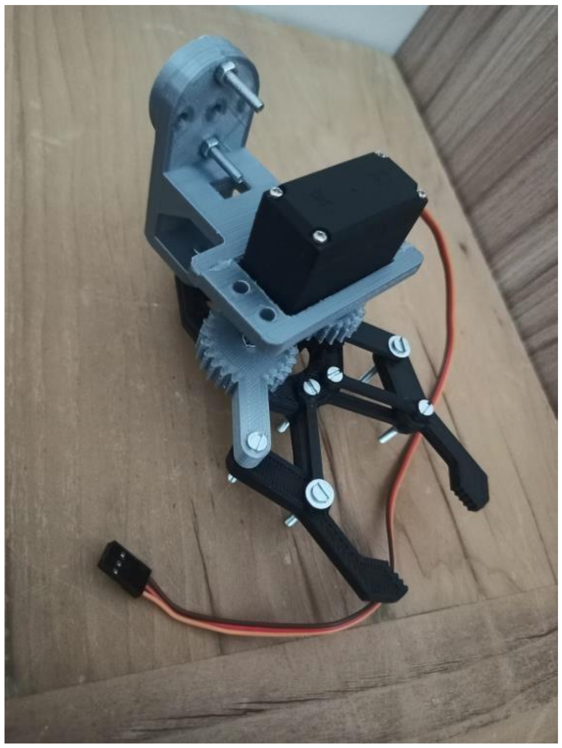



Figure 3.3: End Effector

3.3 Design Calculations:

3.3.1 Forward kinematics:

Using forward kinematics, we will find the position and velocity of end effector given joint angles and velocities.

The first transformation matrix from base to link 1:

$$ {}^{0}_{1}T = \begin{bmatrix} cos(\theta_1) & -sin(\theta_1) & 0 & 0 \\ 0 & 0 & -1 & 0 \\ sin(\theta_1) & cos(\theta_1) & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

Second transformation matrix:

$$ {}^{1}_{2}T = \begin{bmatrix} cos(\theta_2) & -sin(\theta_2) & 0 & (a_2) \\ sin(\theta_2) & cos(\theta_2) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

And similarly, the third transformation matrix can also be obtained by:

$$ {}^{2}_{3}T = \begin{bmatrix} cos(\theta_3) & -sin(\theta_3) & 0 & (a_3) \\ sin(\theta_3) & cos(\theta_3) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

The final transformation matrix from base link to end effector can be calculated from the following expression:

$$ {}^{0}_{3}T = {}^{0}_{1}T \times {}^{1}_{2}T \times {}^{2}_{3}T $$

The final transformation matrix we will obtain after multiplying the above three matrices will give us our position and orientation of our end effector.

3.3.2 Inverse Kinematics:

Using the kinematic equations derived above we can find the desired motion of our robotic arm to reach our desired position. There are various approaches to determine inverse kinematics. We will use geometric approach to figure out aur inverse kinematics.

The calculated joint angles are:

$$ \theta_1 = tan^{-1}(\frac{y}{x}) $$

$$ \theta_2 = cos^{-1}(\frac{{l_2}^2 + k^2 - {l_3}^2}{2kl_2}) + tan^{-1}(\frac{(z - l_1)\cos\theta}{x}) $$

$$\theta_3 = \sin^{-1}\left(\frac{k}{l_3}\sin\left(\cos^{-1}\left(\frac{cos\theta_1{}^2\left(l_2 - l_3{}^2 + (z - l_1)^2\right) + x^2}{2l_2\cos\theta_1\sqrt{(z - l_1)^2 cos\theta_1{}^2 + x^2}}\right)\right)\right)$$

These are the desired angles we will use to determine the motion of our robotic arm.

3.3.4 Dynamics:

The dynamic model is developed using Euler-Lagrange Formulation,

$$D(q)\ddot{q} + C(q,\dot{q})\dot{q} + G(q) = \tau$$

Where,

$$D(q) = Inertia\ Matrix$$

$$C(q,\dot{q}) = Coriolis\ Matrix$$

$$G(q) = Gravity\ Vector$$

3.4 Material Selection:

We will be 3D printing our robotic arm. PLA (Polylactic Acid) is one of the most commonly used thermoplastic materials in 3D printing. It is a biodegradable and bioactive polymer made from renewable resources such as cornstarch or sugarcane. Considering the cost, structural strength, and other limiting factors we will be using PLA to 3D print our parts.

The material properties of PLA are:

Table 3.3: Material Properties

Material Property	Value
Density	$1.25\ g/cm^3$
Melting Temperature	$160\ °C$
Yield Strength	$45\ MPa$

Ultimate Tensile Strength	55 MPa
Young's Modulus	2.7 GPa
Poisson's Ration	0.3
Bulk Modulus	2.25 GPa
Shear Modulus	1.0385 GPa

3.5 Finite Element Analysis:

The Finite Element Analysis of the developed robotic arm was done using Ansys Mechanical to test its structural strength and integrity. The material properties of the Polylactide (PLA) were defined in Ansys material definition Module.

After defining the appropriate Geometry and Mesh Generation, the Von Mises stress and deformation were plotted and visualized. Maximum values of stress and deformation were noted. These maximum values noted were less than the critical values which the material can withstand.

3.5.1 Base Plate:

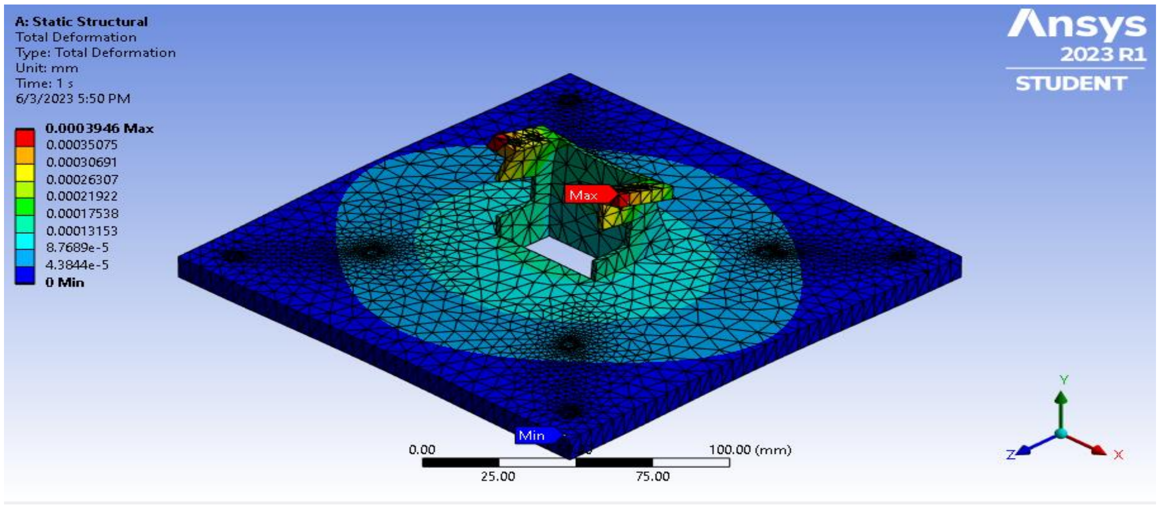


Figure 3.4: Total Deformation in base plate

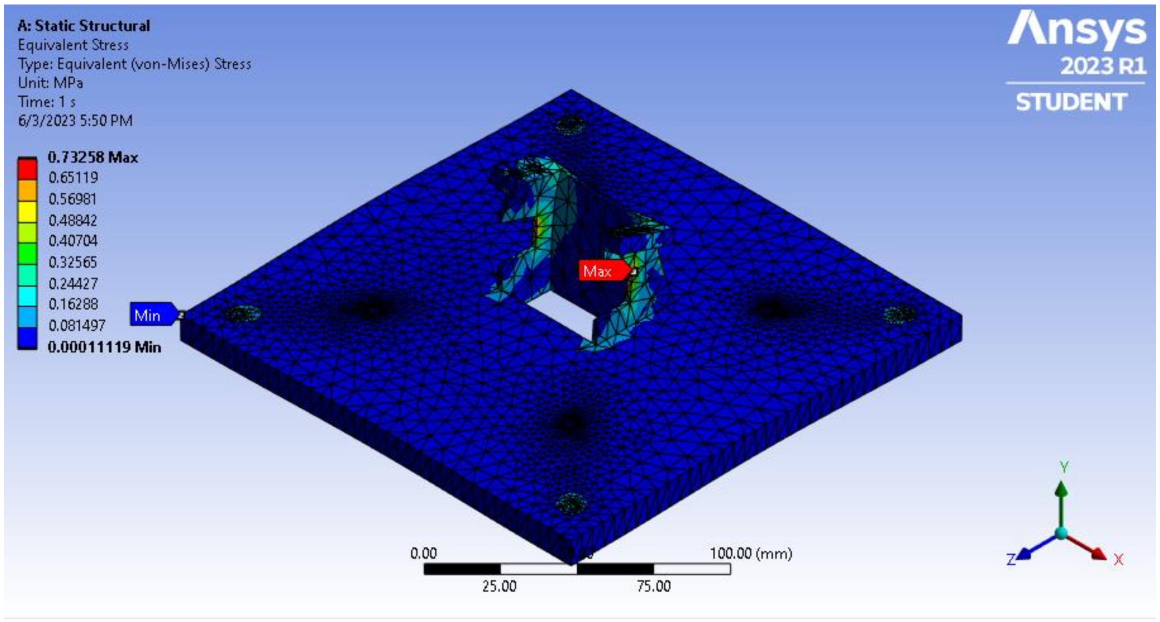


Figure 3.5: Equivalent Von Mises stresses in base plate.

3.5.2 Motor Casing:

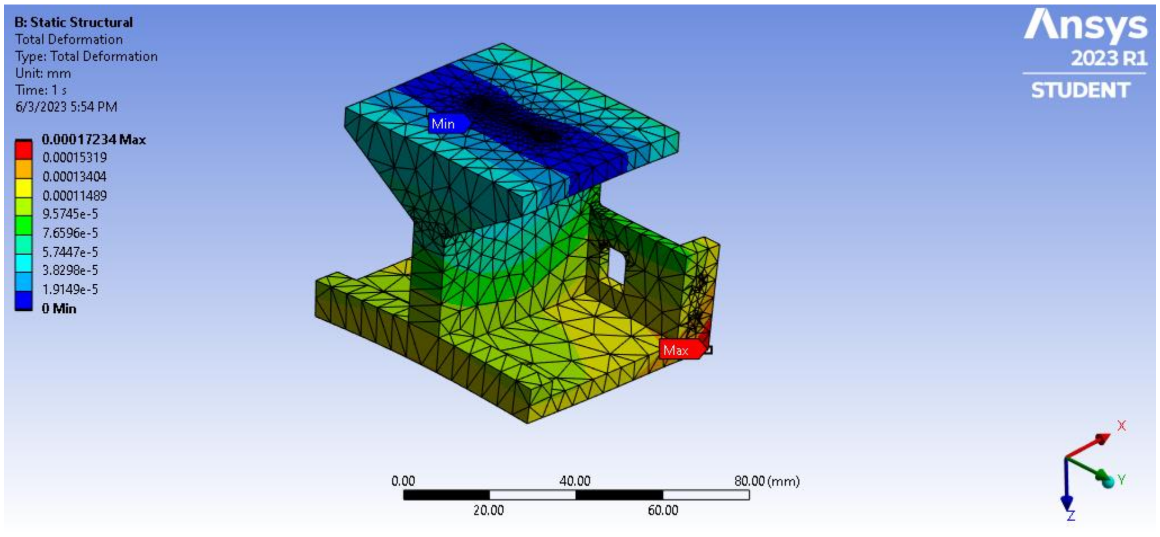


Figure 3.6: Motor Casing Total Deformation

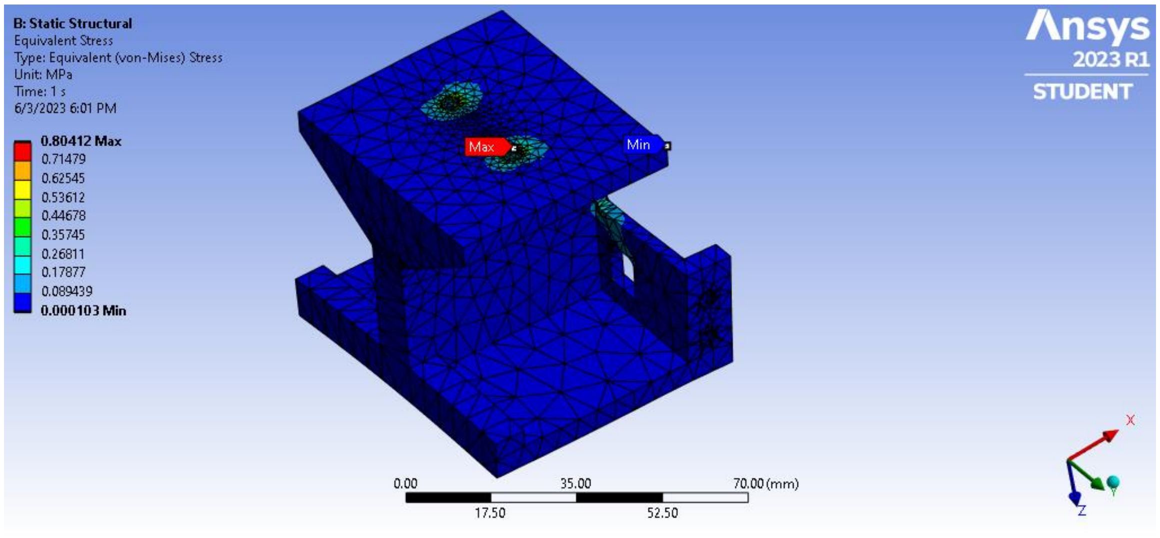


Figure 3.7: Equivalent Von Mises Stresses in Motor Casing

3.5.3 Connecting Link:

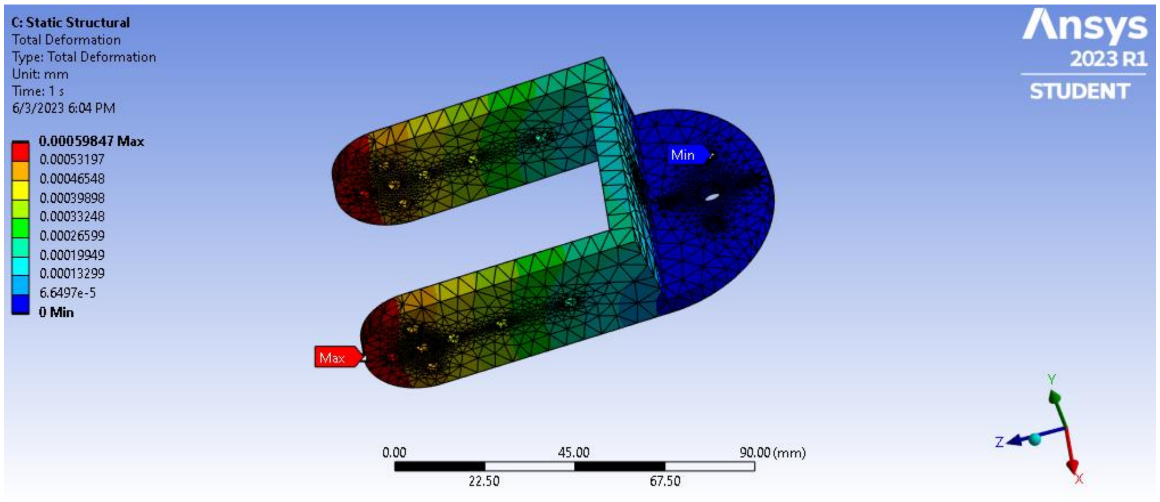


Figure 3.8: Total Deformation in Connecting link.

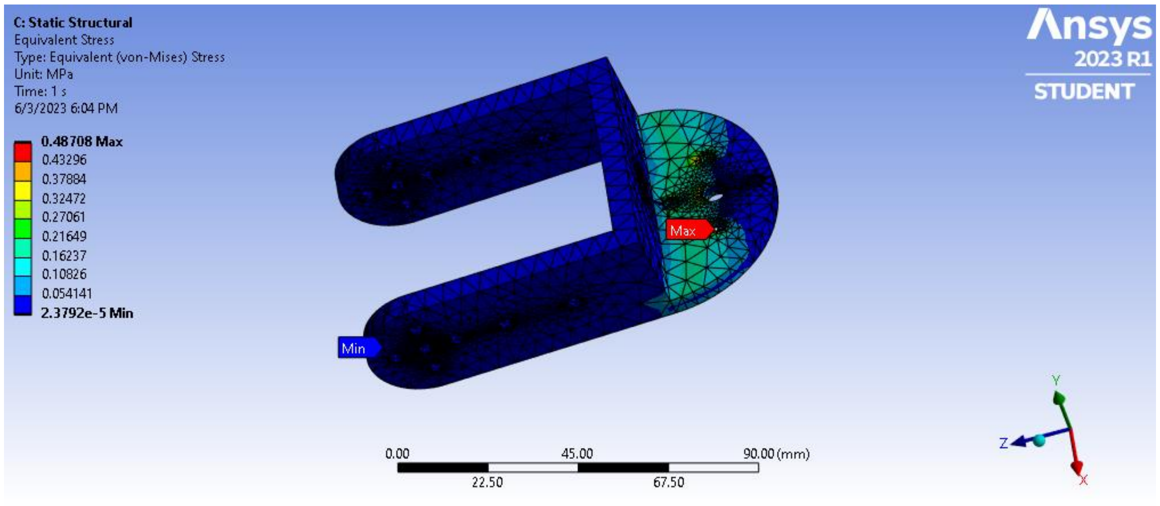


Figure 3.9: Equivalent Von Mises Stresses in Connecting Link

3.5.4 Connecting Link 2:

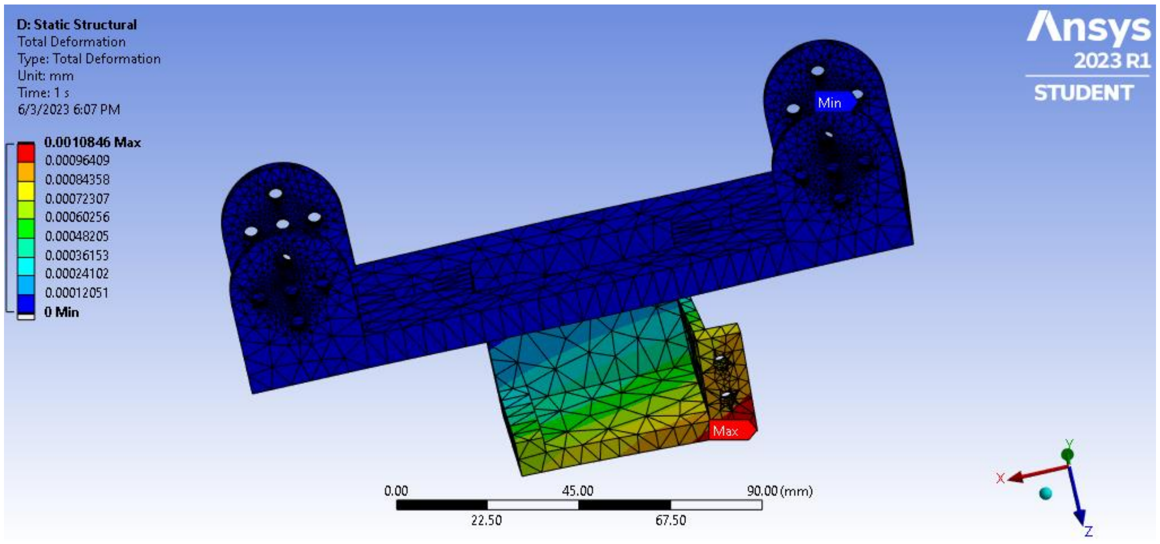


Figure 3.10: Total Deformation in Connecting link 2

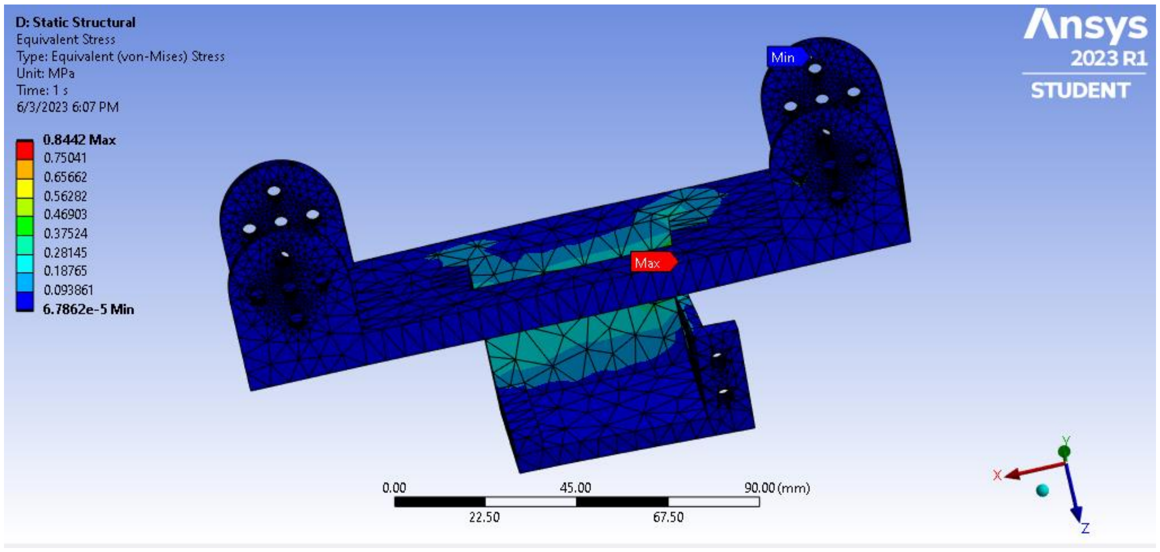


Figure 3.11: Equivalent Von Mises Stresses in Connecting Link

3.6 Control system and Simulation:

3.6.1 Control system of Robotic Manipulator:

The control system for the robotic arm is based on an Arduino microcontroller. It serves as the central processing unit, coordinating the communication between the mobile device and the servos. The Arduino board is responsible for receiving control commands from the mobile application via Bluetooth and translating them into appropriate signals to drive the servos.

The following are the major components of control system:

	Microcontroller	Arduino UNO
	Servo Motors	MG 996R
	Servo Motor driver	Adafruit PCA9685
	Bluetooth Module	HC - 05
	Power Supply	5V - 3A

The control architecture of robotic arm:

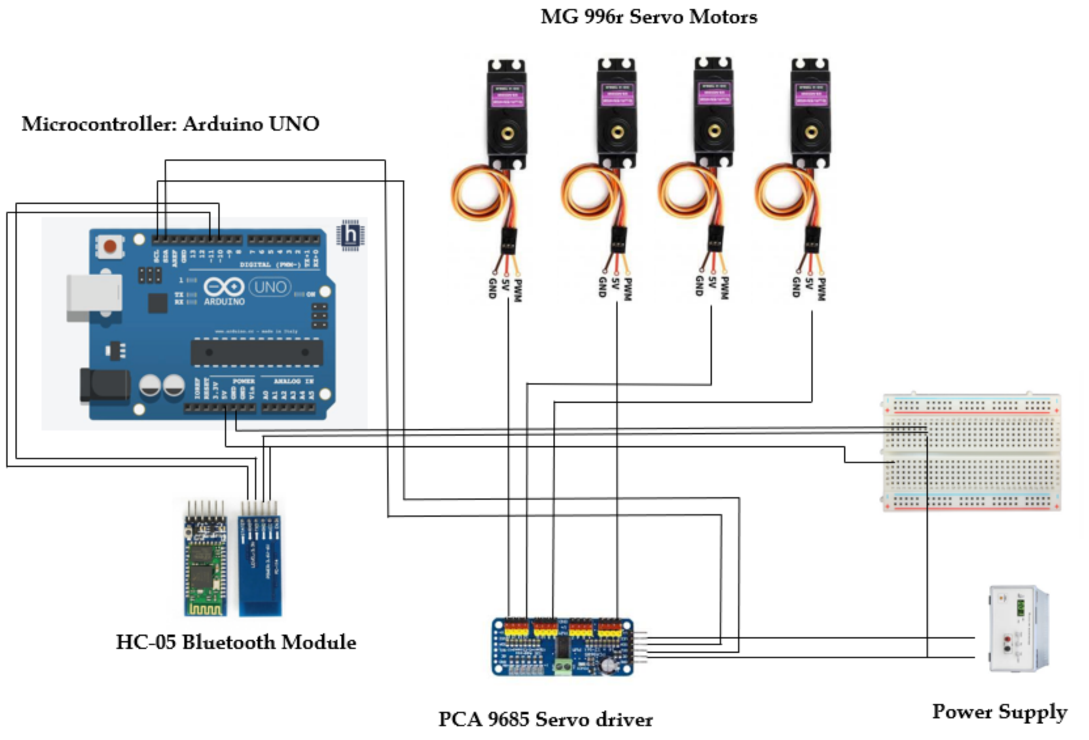


Figure 3.12: Control architecture of robotic arm

The major components of robotic arm are:

1. **Servos and Actuation:**

The MG996R servos are utilized to actuate the three joints of the robotic arm. Each servo motor is connected to the Adafruit PCA9685 servo driver, which is interfaced with the Arduino board. The PCA9685 servo driver allows precise control of the servos by generating Pulse Width Modulation (PWM) signals. The Arduino sends the appropriate PWM signals to the servo driver, instructing the servos to move to the desired positions.

2. **Control Interface:**

The mobile application's user interface is designed to facilitate intuitive control of the robotic arm. The user can interact with the arm by using sliders corresponding to individual servo positions or by using command coordinated movements. The interface provides real-time visual feedback, allowing the user to monitor the arm's position and movements.

3. **Wireless Communication:**

The HC-05 Bluetooth module acts as a wireless communication link between the mobile device and the Arduino board. The Arduino is equipped with the necessary libraries and functions to establish a Bluetooth connection and receive data from the mobile application.

The communication protocol ensures reliable and real-time transmission of control commands, enabling seamless control of the robotic arm.

3.6.2 Manipulator Simulation:

The Manipulator is also simulated in Gazebo combining with ROS control. The manipulation of the robotic arm done using MoveIt ROS Package. This package is the most widely used package for motion planning and manipulation purposes.

First, the URDF File of the robotic arm was generated using SolidWorks CAD Model we have already designed. The workspace was then set up in Ubuntu to generate our package.

We have created package *robot_urdf* and created launch file named *arm_urdf.launch*.

To launch the gazebo world file in the terminal we used the command:

~/moveit_ws$ roslaunch robot_urdf arm_urdf.launch

The robotic arm in our gazebo world is shown:

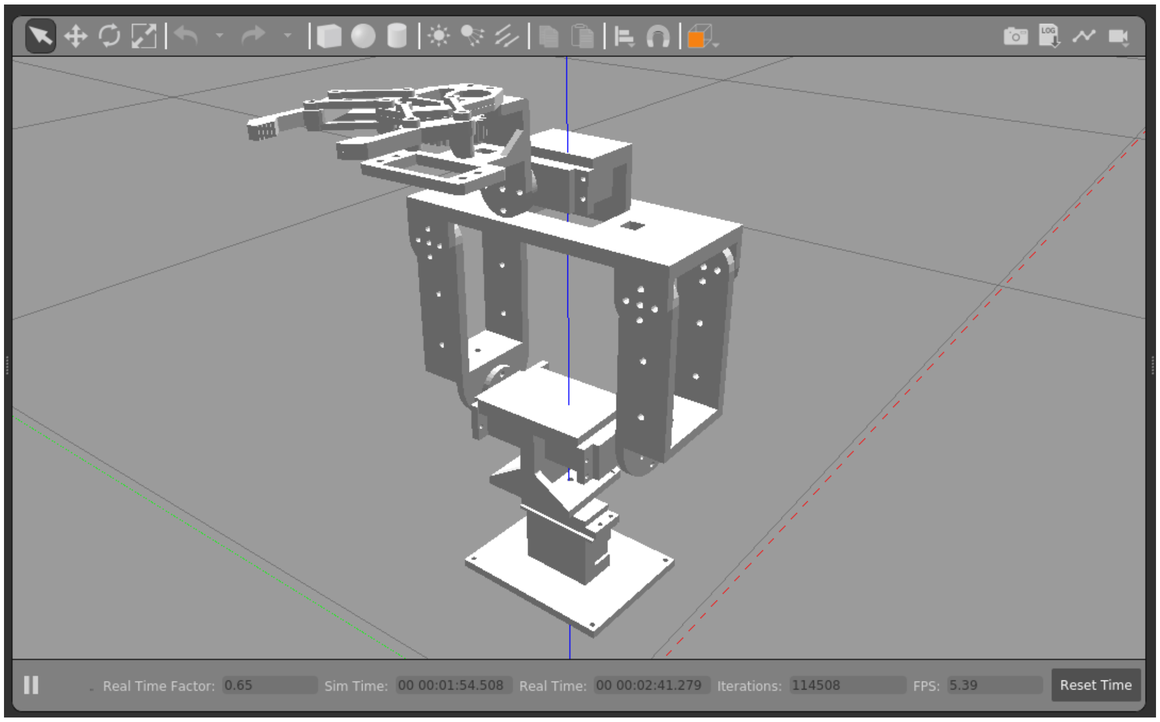


Figure 3.13: Robotic Manipulator in Gazebo ROS

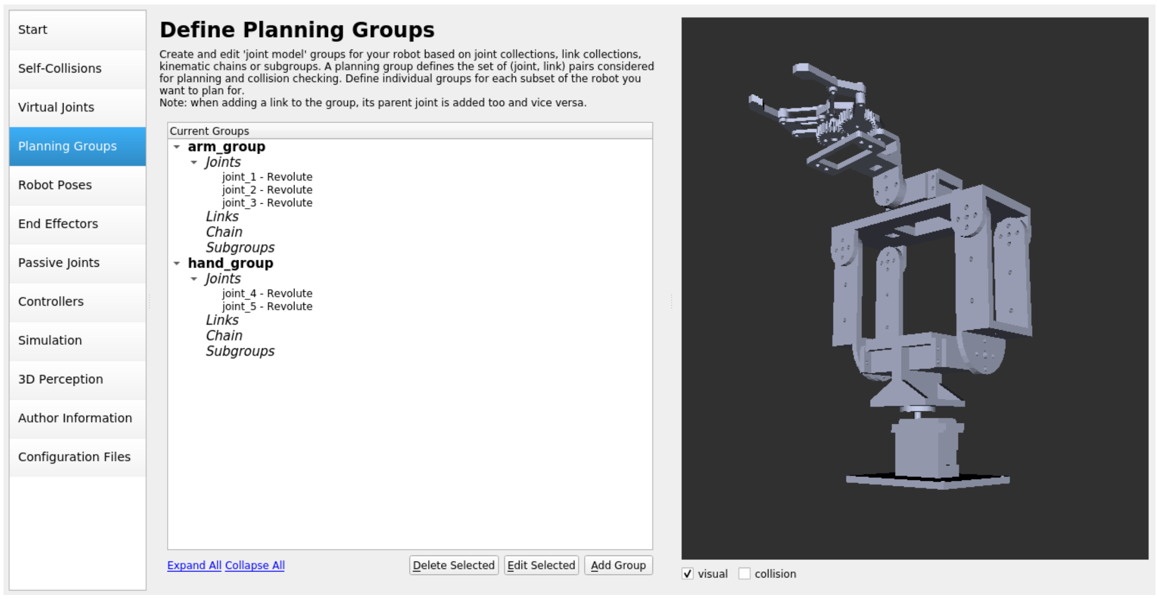
Now, we will launch MoveIt Setup Assistant to control our robotic arm.

Running the following command on the terminal will start the MoveIt Setup Assistant.

~/moveit_ws$ roslaunch moveit_setup_assistant setup_assistant.launch

After loading our created package, we can set up our manipulator and Controllers in MoveIt Setup Assistant.

Following are the some of snapshots of creating and controlling different Robot Poses.

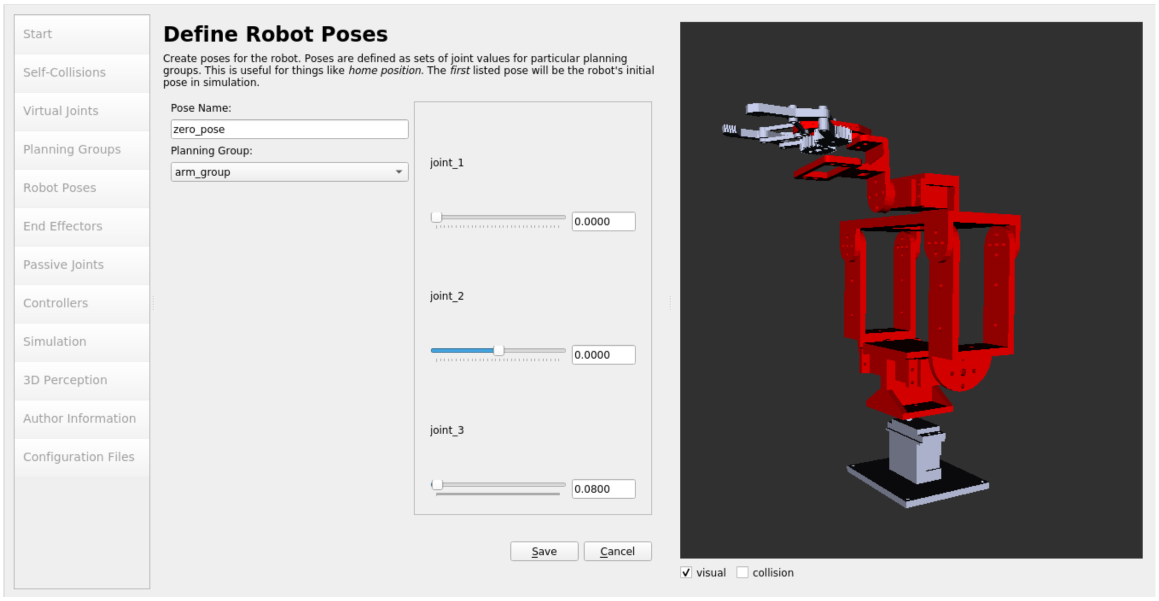


Define Planning Groups

Create and edit 'joint model' groups for your robot based on joint collections, link collections, kinematic chains or subgroups. A planning group defines the set of (joint, link) pairs considered for planning and collision checking. Define individual groups for each subset of the robot you want to plan for.
Note: when adding a link to the group, its parent joint is added too and vice versa.

Current Groups
- **arm_group**
  - *Joints*
    - joint_1 - Revolute
    - joint_2 - Revolute
    - joint_3 - Revolute
  - *Links*
  - *Chain*
  - *Subgroups*
- **hand_group**
  - *Joints*
    - joint_4 - Revolute
    - joint_5 - Revolute
  - *Links*
  - *Chain*
  - *Subgroups*

Figure 3.14: Planning Groups Definition in MoveIt Setup Assistant



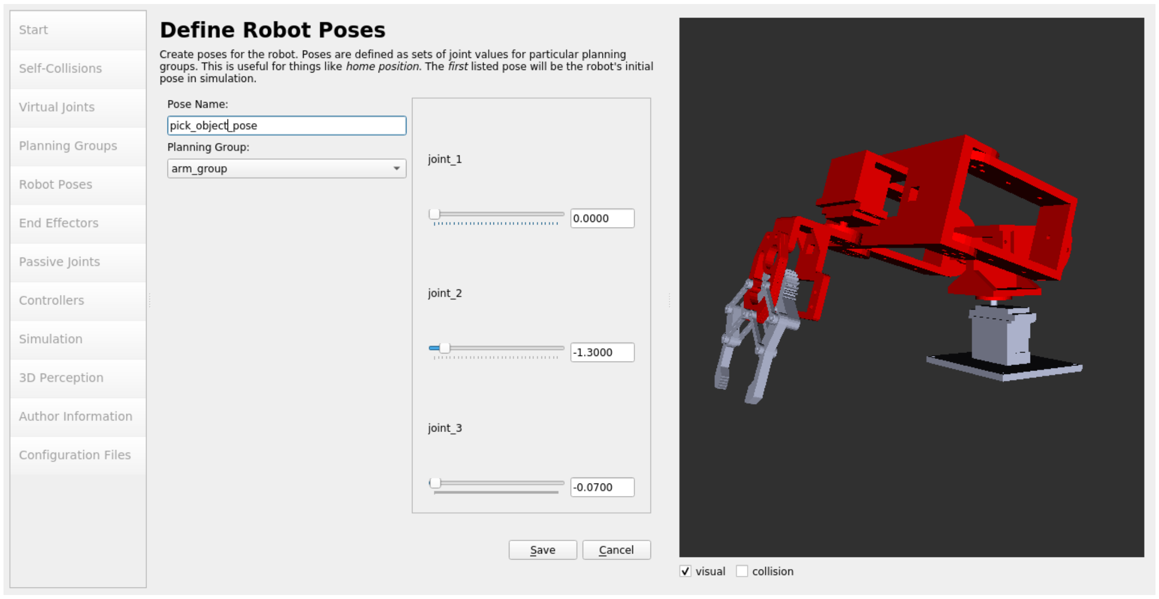
Define Robot Poses

Create poses for the robot. Poses are defined as sets of joint values for particular planning groups. This is useful for things like *home position*. The *first* listed pose will be the robot's initial pose in simulation.

Pose Name: zero_pose
Planning Group: arm_group

joint_1 0.0000
joint_2 0.0000
joint_3 0.0800

Figure 3.15: Robotic Manipulator in zero pose



Define Robot Poses

Create poses for the robot. Poses are defined as sets of joint values for particular planning groups. This is useful for things like *home position*. The *first* listed pose will be the robot's initial pose in simulation.

Pose Name: pick_object_pose

Planning Group: arm_group

joint_1 0.0000

joint_2 -1.3000

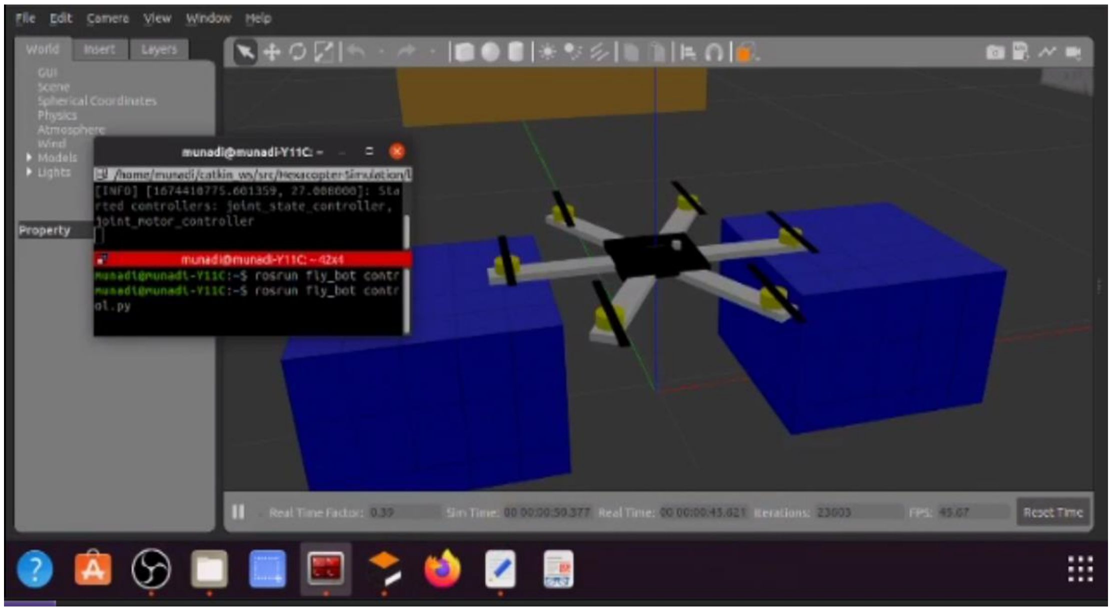
joint_3 -0.0700

Figure 3.16: Robotic Manipulator in picking object pose

3.6.3 UAV Simulation and Control:

The UAV was simulated in **Gazebo** environment using **ROS (Robot Operating System)** Noetic. The drone was spawned in a simulated gazebo environment and was controlled using terminal commands.

The snapshots of drone in **Gazebo** environment are:



```
munadi@munadi-Y11C: ~
/home/munadi/catkin_ws/src/Hexacopter-Simulation/
[INFO] [1674410775.601359, 27.088000]: Sta
rted controllers: joint_state_controller,
joint_motor_controller

munadi@munadi-Y11C: ~ 42x4
munadi@munadi-Y11C:~$ rosrun fly_bot contr
munadi@munadi-Y11C:~$ rosrun fly_bot contr
ol.py
```

Figure 3.17: Hexacopter spawn

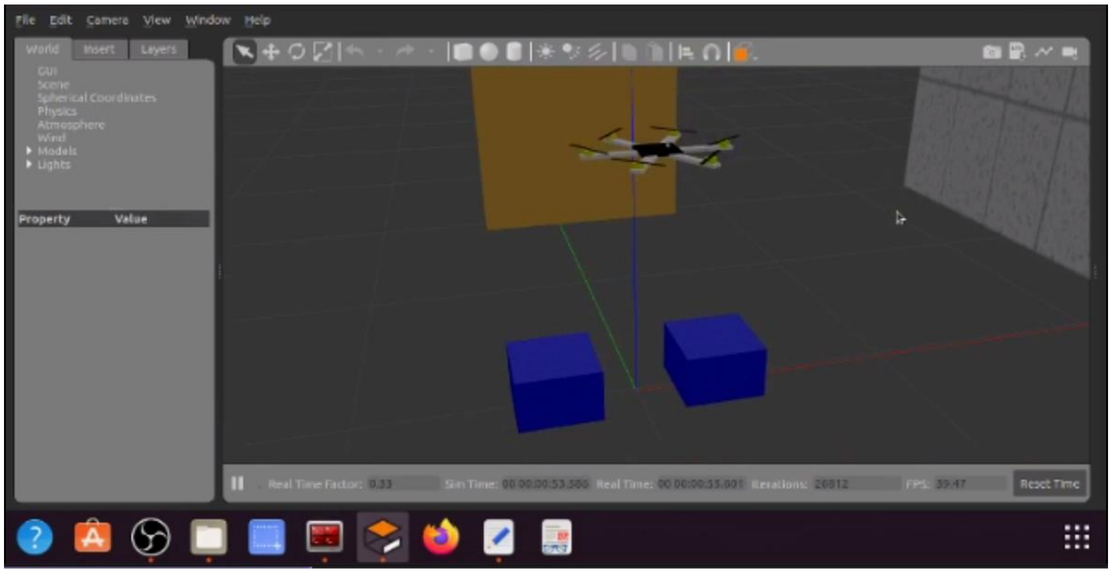


Figure 3.18: Hexacopter hovering

Chapter no 4: Results and Discussions:

This chapter will focus on the results of all the calculations and simulations from the previous chapter. Following that final parametric results will be selected. Also, the final design would be validated. The selection of components will also be discussed.

4.1 UAM Final System Configuration:

4.1.1 UAV Propeller and Motor Selection:

The initial design calculations for the required thrust were done in previous chapter. Following that results will select following motor and propellers.

Table 4.1: Propeller and Motor Specifications

Propeller	APC 1045”
Motor	A2212/13T 1000KV Brushless DC Motor

4.1.2 Frame, Battery, and ESC’s Selection:

According to our requirements we selected the frame of UAV as well as batteries and ESC’s.

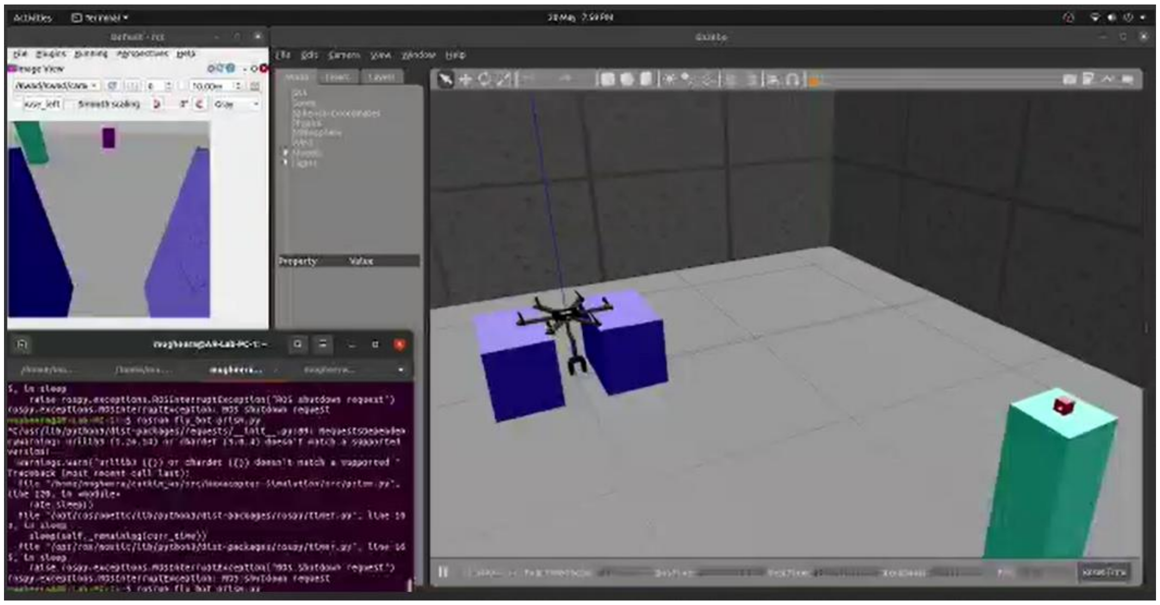
Table 4.2: Frame, Battery and ESC Specs

Frame	DJI F550
ESC	30 Amp
Battery	5200 mAh 3S Lipo 11.1 Volt

4.2 Final Simulation Results:

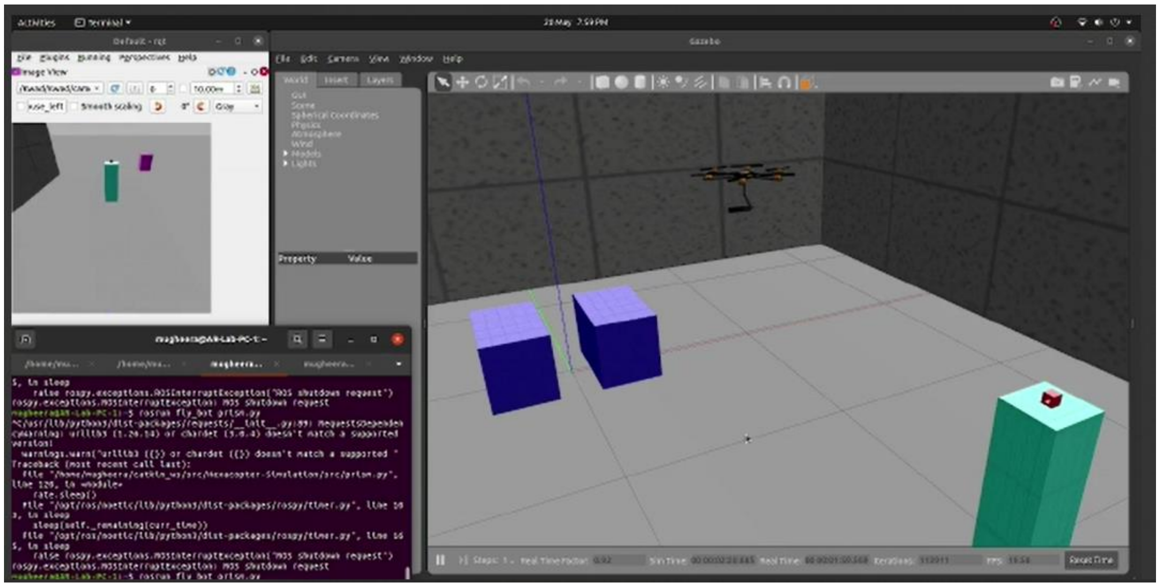
The UAV was simulated in **Gazebo** environment using **ROS (Robot Operating System)** Noetic. The drone was spawned in a simulated gazebo environment and was controlled using terminal commands.

The snapshots of drone in **Gazebo** environment are:



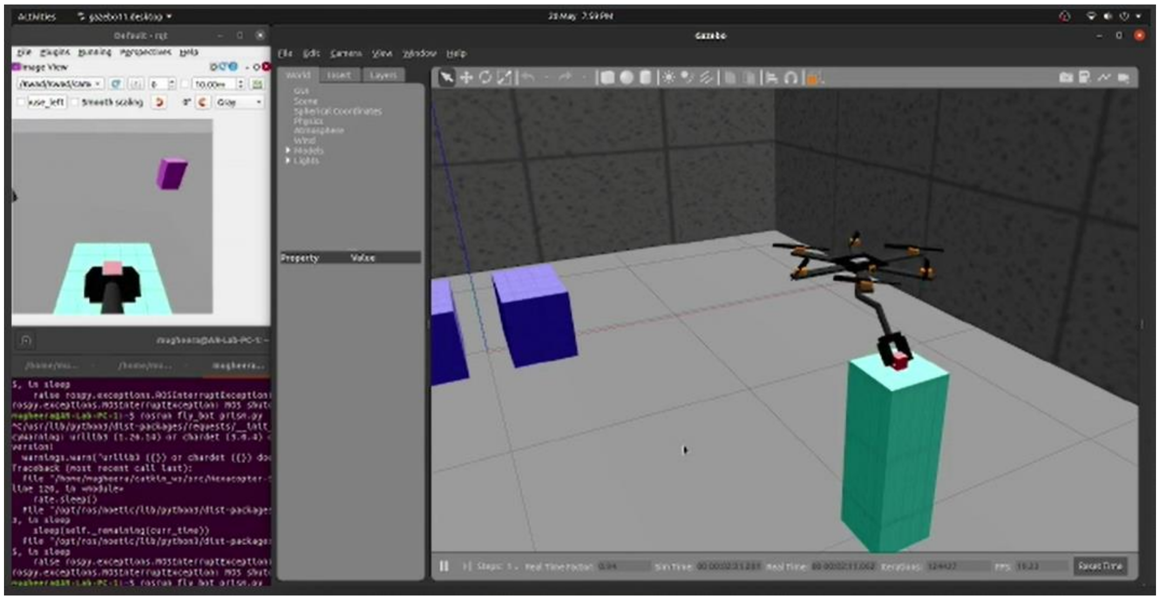
```
5, in sleep
    raise rospy.exceptions.ROSInterruptException("ROS shutdown request")
rospy.exceptions.ROSInterruptException: ROS shutdown request
magheera@AR-Lab-PC-1:~$ rosrun fly_bot prism.py
/usr/lib/python3/dist-packages/requests/__init__.py:89: RequestsDependen
cyWarning: urllib3 (1.26.14) or chardet (5.0.4) doesn't match a supported
version!
  warnings.warn("urllib3 ({}) or chardet ({}) doesn't match a supported "
Traceback (most recent call last):
  File "/home/magheera/catkin_ws/src/hexacopter-Simulation/src/prism.py",
line 120, in <module>
    rate.sleep()
  File "/opt/ros/noetic/lib/python3/dist-packages/rospy/timer.py", line 10
3, in sleep
    sleep(self._remaining(curr_time))
  File "/opt/ros/noetic/lib/python3/dist-packages/rospy/timer.py", line 16
5, in sleep
    raise rospy.exceptions.ROSInterruptException("ROS shutdown request")
rospy.exceptions.ROSInterruptException: ROS shutdown request
magheera@AR-Lab-PC-1:~$ rosrun fly_bot prism.py
```

Figure 4.1: Hexacopter at initial position



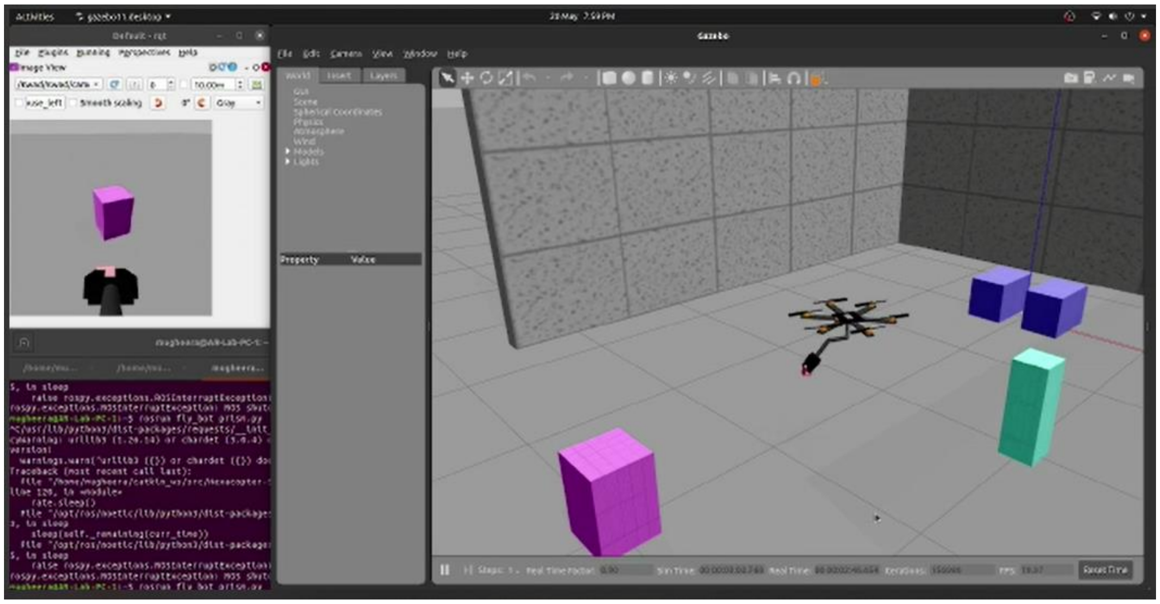
```
5, in sleep
    raise rospy.exceptions.ROSInterruptException("ROS shutdown request")
rospy.exceptions.ROSInterruptException: ROS shutdown request
magheera@AR-Lab-PC-1:~$ rosrun fly_bot prism.py
/usr/lib/python3/dist-packages/requests/__init__.py:89: RequestsDependen
cyWarning: urllib3 (1.26.14) or chardet (5.0.4) doesn't match a supported
version!
  warnings.warn("urllib3 ({}) or chardet ({}) doesn't match a supported "
Traceback (most recent call last):
  File "/home/magheera/catkin_ws/src/hexacopter-Simulation/src/prism.py",
line 120, in <module>
    rate.sleep()
  File "/opt/ros/noetic/lib/python3/dist-packages/rospy/timer.py", line 10
3, in sleep
    sleep(self._remaining(curr_time))
  File "/opt/ros/noetic/lib/python3/dist-packages/rospy/timer.py", line 16
5, in sleep
    raise rospy.exceptions.ROSInterruptException("ROS shutdown request")
rospy.exceptions.ROSInterruptException: ROS shutdown request
magheera@AR-Lab-PC-1:~$ rosrun fly_bot prism.py
```

Figure 4.2: Hexacopter going to pick object



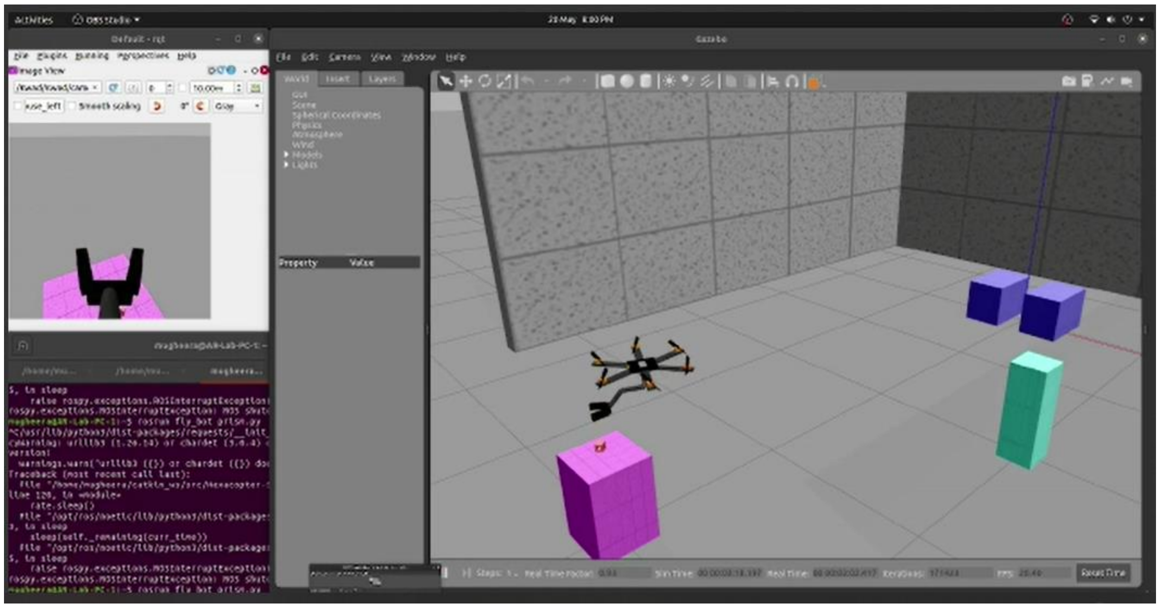
```
5, in sleep
    raise rospy.exceptions.ROSInterruptException(
rospy.exceptions.ROSInterruptException: ROS shut
magheera@AR-Lab-PC-1:~$ rosrun fly_bot prism.py
^C/usr/lib/python3/dist-packages/requests/__init_
cyWarning: urllib3 (1.26.14) or chardet (5.0.4)
version!
  warnings.warn("urllib3 ({}) or chardet ({}) doe
Traceback (most recent call last):
  File "/home/magheera/catkin_ws/src/Hexacopter-
line 120, in <module>
    rate.sleep()
  File "/opt/ros/noetic/lib/python3/dist-package
s, in sleep
    sleep(self._remaining(curr_time))
  File "/opt/ros/noetic/lib/python3/dist-package
5, in sleep
    raise rospy.exceptions.ROSInterruptException(
rospy.exceptions.ROSInterruptException: ROS shut
magheera@AR-Lab-PC-1:~$ rosrun fly_bot prism.py
```

Figure 4.3: Hexacopter while picking the object



```
5, in sleep
    raise rospy.exceptions.ROSInterruptException(
rospy.exceptions.ROSInterruptException: ROS shut
magheera@AR-Lab-PC-1:~$ rosrun fly_bot prism.py
^C/usr/lib/python3/dist-packages/requests/__init_
cyWarning: urllib3 (1.26.14) or chardet (5.0.4)
version!
  warnings.warn("urllib3 ({}) or chardet ({}) doe
Traceback (most recent call last):
  File "/home/magheera/catkin_ws/src/Hexacopter-
line 120, in <module>
    rate.sleep()
  File "/opt/ros/noetic/lib/python3/dist-package
s, in sleep
    sleep(self._remaining(curr_time))
  File "/opt/ros/noetic/lib/python3/dist-package
5, in sleep
    raise rospy.exceptions.ROSInterruptException(
rospy.exceptions.ROSInterruptException: ROS shut
magheera@AR-Lab-PC-1:~$ rosrun fly_bot prism.py
```

Figure 4.4: Hexacopter returning back to drop the object



```
5, in sleep
    raise rospy.exceptions.ROSInterruptException(
rospy.exceptions.ROSInterruptException: ROS shutd
magheera@AR-Lab-PC-1:~$ rosrun fly_bot prism.py
/usr/lib/python3/dist-packages/requests/__init_
: RequestsDependencyWarning: urllib3 (1.26.14) or chardet (3.0.4) d
version!
  warnings.warn("urllib3 ({}) or chardet ({}) doe
Traceback (most recent call last):
  File "/home/magheera/catkin_ws/src/Hexacopter-
line 120, in <module>
    rate.sleep()
  File "/opt/ros/noetic/lib/python3/dist-packages
3, in sleep
    sleep(self._remaining(curr_time))
  File "/opt/ros/noetic/lib/python3/dist-packages
5, in sleep
    raise rospy.exceptions.ROSInterruptException(
rospy.exceptions.ROSInterruptException: ROS shutd
magheera@AR-Lab-PC-1:~$ rosrun fly_bot prism.py
```

Figure 4.5: Hexacopter after dropping the object

4.3 Aerial Manipulation System:

The final design of the Aerial Manipulator will have the integration of the Robotic Arm with our UAV.

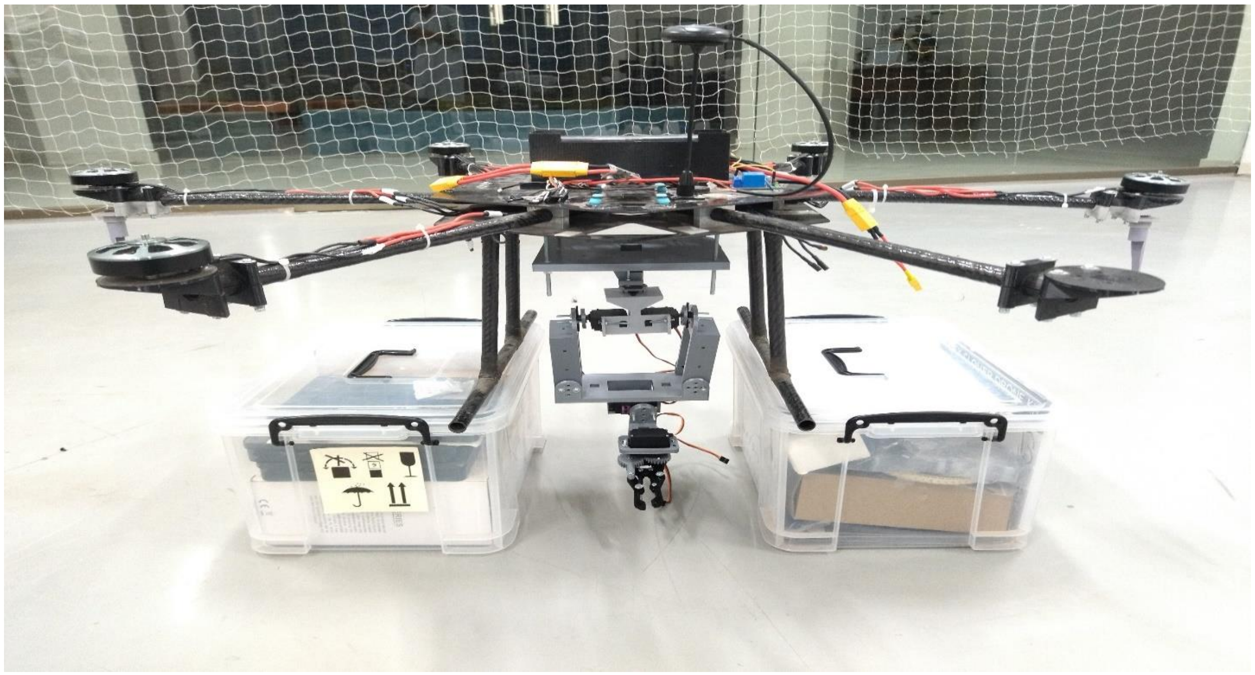


Figure 4.6: Aerial Manipulation System

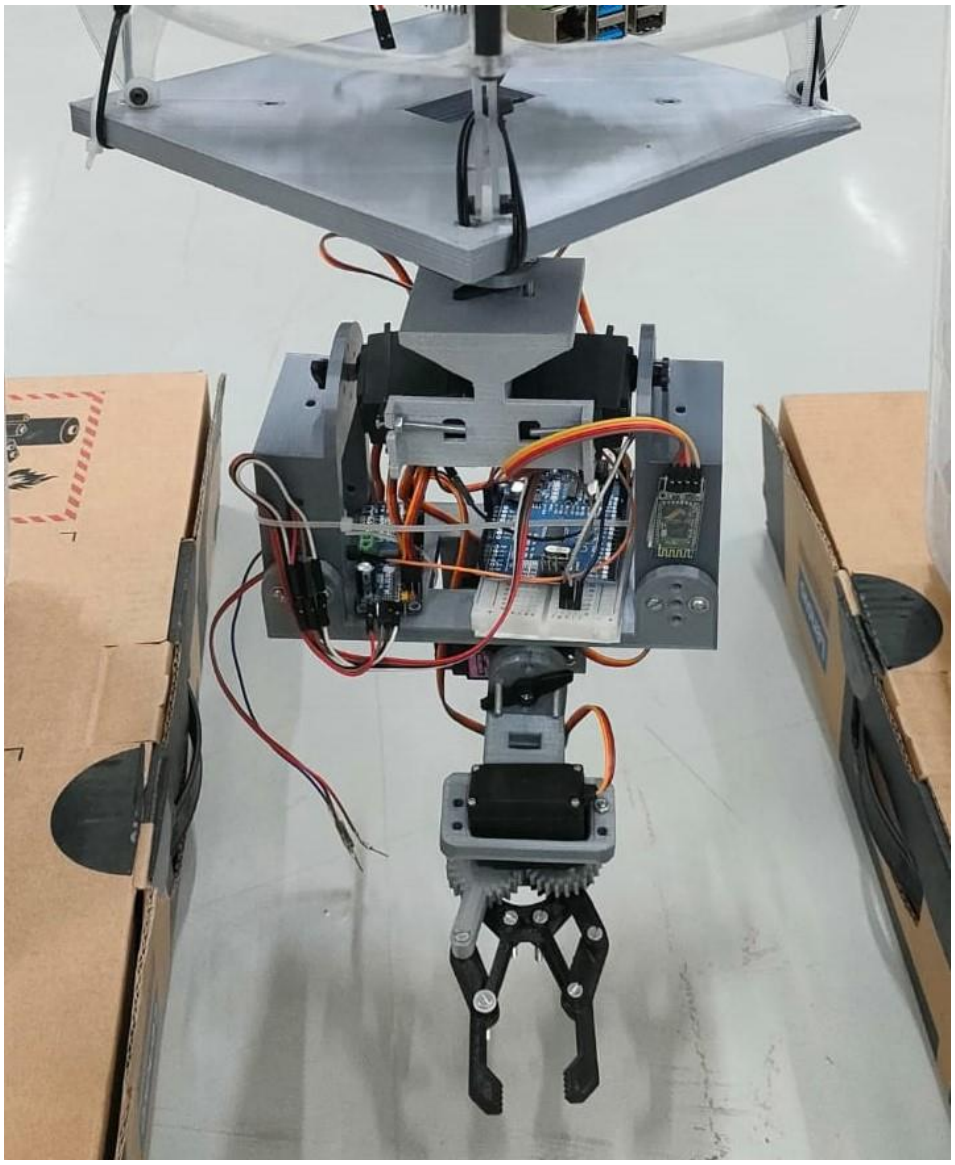


Figure 4.7: Robotic Manipulator

The arm is made up of three revolute joints. Following are the main parameters of the arm joints:

Table 4.3: Parameters of Arm Joints

Joint	Motion Range	Motor Model	Stall Torque	Motor Weight
Base Joint	[0, 360]	Tower Pro MG996r	9.4 kg.cm	55 g
Joint1	[0, 180]	Tower Pro MG996r	9.4 kg.cm	55 g
Joint2	[0, 180]	Tower Pro MG996r	9.4 kg.cm	55 g
Joint 3	[0, 180]	Tower Pro MG996R	9.4 kg.cm	55g

Chapter no 5: Conclusion and Recommendations

This Aerial Manipulation system manufactured as per the requirements. Test Flight has been concluded successfully.

Selection of drone components were quite accurate as the drone system provided thrust as per our requirement. The thrust produced was enough to fly the whole manipulation system along with the mass attached with the arm. At full throttle the maximum flight time is calculated to be 4 minutes. But after applying factor of safety, it reduced to 2.8 minutes.

The selection of PLA material was a good decision which was also proved by the Finite Element Analysis of the arm. PLA is a promising thermoplastic aliphatic polyester with relatively high mechanical strength (flexural strength up to 140 MPa, Young's modulus 5–10 GPa), with excellent optical properties, good processing ability (with low shrinkage not causing product deformation) and complete biodegradation Performing a finite element analysis (FEA) of the robotic arm made of PLA material helped in understanding the structural integrity and potential failure points of the arm.

All the stresses produced in the robotic arm were in the prescribed limits which can be seen through Finite Element Analysis.

5.1 Some of Future Recommendations are follows:

5.1.1 Control System:

The drone-mounted robotic arm can be operated either manually or autonomously. Developing an advanced control system that allows for more precise control and accurate object recognition would improve its efficiency and performance.

5.1.2 Load Capacity:

The current load capacity of the arm is limited by the strength of the material used for 3D printing. To enhance its capability, future designs could use stronger materials, or the arm could be reinforced with metal or other high-strength materials.

5.1.3 Battery Life:

The operation of the drone and robotic arm is dependent on battery power, and the current battery life may limit its usefulness. Innovations in battery technology could help to extend its operational duration, enabling it to perform more tasks.

5.1.3 Sensor Integration:

Integrating sensors like cameras, LIDAR, and ultrasonic sensors into the drone and robotic arm would improve its ability to detect and avoid obstacles, enhancing its safety and accuracy.

5.2 Applications:

The drone-mounted robotic arm has a wide range of applications in various industries, including logistics, agriculture, and construction. It can be used to pick objects from inaccessible places specially in case of emergency. In case of fire in a room or building, this manipulation system can be sent to pick some important stuff from the room like mobile phone or some very important documents. Further research could be conducted to explore and develop more innovative use cases for this technology.

References

[1] *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS).*

Aerial Manipulation Using a Quadrotor with a Two DOF Robotic Arm.

Suseong Kim , Seungwon Choi and H. Jin Kim.

[2] A Versatile Aerial Manipulator Design and Realization of UAV Take-Off from a Rocking Unstable Surface.

Hannibal Paul ,Ryo Miyazaki,Takamasa Kominam ,Robert Ladig and Kazuhiro Shimonomura

[3] Design of a High-Performance Dual Arm Aerial Manipulator

Pedro Grau, Alejandro Suarez, Victor Manuel Vega, Angel Rodriguez-Castaño and Anibal Ollero

[4] Proceedings of the 2016 IEEE International Conference on Robotics and Biomimetics Qingdao, China, December 3-7, 2016

Design and Implementation of Rotor Aerial Manipulator System

Xiangdong Meng, Yuqing He, Feng Gu, Liying Yang, Bo Dai, Zhong Liu and Jianda Han

[5] Ruggiero, F., Lippiello, V., & Ollero, A. (2018). Aerial Manipulation: A Literature Review. IEEE Robotics and Automation Letters, 3(3), 1957–1964. https://doi.org/10.1109/lra.2018.2808541

[6] Imanberdiyev, N., Sood, S., Kircali, D., & Kayacan, E. (2022). Design, development and experimental validation of a lightweight dual-arm aerial manipulator with a COG balancing mechanism. Mechatronics, 82, 102719. https://doi.org/10.1016/j.mechatronics.2021.102719

[7] Paul, H., Miyazaki, R., Kominami, T., Ladig, R., & Shimonomura, K. (2021). A Versatile Aerial Manipulator Design and Realization of UAV Take-Off from a Rocking Unstable Surface. Applied Sciences, 11(19), 9157. https://doi.org/10.3390/app11199157

[8] AlAkhras, A., Sattar, I. H., Alvi, M., Qanbar, M. W., Jaradat, M. A., & Alkaddour, M. (2022). The Design of a Lightweight Cable Aerial Manipulator with a CoG Compensation Mechanism for Construction Inspection Purposes. Applied Sciences, 12(3), 1173. https://doi.org/10.3390/app12031173

Appendix 1: Manipulator Codes

```
#include <Wire.h>

#include <Adafruit_PWMServoDriver.h>


Adafruit_PWMServoDriver pwm = Adafruit_PWMServoDriver(); // use the default address 0x40

void motor(uint8_t servo, int startAngle, int finalAngle, int dA, int dt, int restTime);


#define SERVOMIN  100 // This is the 'minimum' pulse length count (out of 4096)

#define SERVOMAX  450 // This is the 'maximum' pulse length count (out of 4096)

#define USMIN  600 // This is the rounded 'minimum' microsecond length based on the minimum
pulse of 150

#define USMAX  2400 // This is the rounded 'maximum' microsecond length based on the
maximum pulse of 600

#define SERVO_FREQ 50 // Analog servos run at ~50 Hz updates


uint8_t servo1 = 12; int x1 = 100;

uint8_t servo2 = 3; int x2 = 100;

uint8_t servo2b = 8;

uint8_t servo3 = 1; int x3 = 100;

uint8_t servo4 = 15; int x4 = 100;


int dx = 30;

int da = 1;


void setup() {

  Serial.begin(9600);

  Serial.println("4 channel Servo test!");


  pwm.begin();

  pwm.setOscillatorFrequency(27000000);
  pwm.setPWMFreq(SERVO_FREQ);  // Analog servos run at ~50 Hz updates

  delay(10);
}

void loop()
{
   if (Serial.available() > 0)
   {
      char inChar = Serial.read(); // read incoming serial data:
      if(inChar=='a')
      {
       if (x1 >= SERVOMAX) {Serial.println("M1. upper limit: ");  Serial.println(x1);}
       else {motor(servo1, x1, x1+dx, da, 5, 0); x1 = x1 + dx; Serial.println(x1);}
      }
      if(inChar=='z')
      {
       if (x1 <= SERVOMIN) {Serial.println("M1. lower limit: ");  Serial.println(x1);}
       else {motor(servo1, x1, x1-dx, da, 5, 0); x1 = x1 - dx; Serial.println(x1);}
      }

      if(inChar=='s')
      {
       if (x2 >= SERVOMAX) {Serial.println("M2. upper limit: ");  Serial.println(x2);}
       else {
        motor(servo2, x2, x2+dx, da, 5, 0); x2 = x2 + dx/2; Serial.println(x2);
        motor(servo2b, x2, x2-dx, da, 5, 0); x2 = x2 + dx/2;
        }
      }
      if(inChar=='x')
      {
       if (x2 <= SERVOMIN) {Serial.println("M2. lower limit: ");  Serial.println(x2);}
       else {
        motor(servo2, x2, x2-dx, da, 5, 0); x2 = x2 - dx/2; Serial.println(x2);
        motor(servo2b, x2, x2+dx, da, 5, 0); x2 = x2 - dx/2;
        }
      }

      if(inChar=='d')
      {
       if (x3 >= SERVOMAX) {Serial.println("M3. upper limit: ");  Serial.println(x3);}
       else {motor(servo3, x3, x3+dx, da, 5, 0); x3 = x3 + dx; Serial.println(x3);}
      }
      if(inChar=='c')
      {
       if (x3 <= SERVOMIN) {Serial.println("M3. lower limit: ");  Serial.println(x3);}
       else {motor(servo3, x3, x3-dx, da, 5, 0); x3 = x3 - dx; Serial.println(x3);}
      }

      if(inChar=='f')
      { zz
       if (x4 >= SERVOMAX) {Serial.println("M4. upper limit: ");  Serial.println(x4);}
       else {motor(servo4, x4, x4+dx, da, 5, 0); x4 = x4 + dx; Serial.println(x4);}
      }
      if(inChar=='v')
      {
       if (x4 <= SERVOMIN) {Serial.println("M4. lower limit: ");  Serial.println(x4);}
       else {motor(servo4, x4, x4-dx, da, 5, 0); x4 = x4 - dx; Serial.println(x4);}
      }
   }
```

```
}

//-----------------------------------
// Motor Function
//-----------------------------------
void motor(uint8_t servo, int startAngle, int finalAngle, int dA, int dt, int restTime)
{
  // increase the angle
  if (startAngle < finalAngle)
  {
    for (uint16_t pulselen = startAngle; pulselen < finalAngle; pulselen=pulselen+dA)
      {pwm.setPWM(servo, 0, pulselen); delay(dt);}
  }

  // decrease the angle
  else
  {
    for (uint16_t pulselen = startAngle; pulselen > finalAngle; pulselen=pulselen-dA)
      {pwm.setPWM(servo, 0, pulselen); delay(dt);}
  }

  delay(restTime);
}
```

Appendix 2: ROS Simulation Codes

```
=====================================================================
=====================
Drone Controller:
=====================================================================
=====================
joint_state_controller:
  type: joint_state_controller/JointStateController
  publish_rate: 50


joint_motor_controller:
  type: velocity_controllers/JointGroupVelocityController
  joints:
    - joint_front_right_prop
    - joint_front_left_prop
    - joint_left_prop
    - joint_back_left_prop
    - joint_back_right_prop
    - joint_right_prop


=====================================================================
=====================
Arm Controller:
=====================================================================
=====================
arm_controller:
 type: "position_controllers/JointTrajectoryController"
 joints:
   - hip
   - shoulder
   - elbow
   - wrist
```

==
=====================

Launch File 1 - Spawn + Drone Controller:

==
=====================

```
<launch>

	<include file="$(find gazebo_ros)/launch/empty_world.launch">

		<arg name="world_name" value="$(find fly_bot)/worlds/demo.world" />

	<arg name="paused" default="false" />

		<arg name="use_sim_time" default="true" />

		<arg name="gui" default="true" />

		<arg name="headless" default="false" />

		<arg name="debug" default="false" />

	</include>

	<group ns="Hexacopter">

		<param name="robot_description" command="$(find xacro)xacro '$(find
fly_bot)/urdf/HexaCbot.xacro'" />

		<node name="spawn_urdf" pkg="gazebo_ros" type="spawn_model"
respawn="false" output="screen" args="-param robot_description -urdf -model Kwad -x 0 -y 0 -z
0.4" />

		<rosparam file="$(find fly_bot)/config/Kwad_control.yaml" command="load"
ns="/Kwad" />

		<node name="control_spawner" pkg="controller_manager" type="spawner"
respawn="false" output="screen" args="joint_state_controller joint_motor_controller" />

		<node name="robot_state_publisher" pkg="robot_state_publisher"
type="robot_state_publisher" respawn="false" output="screen">

			<param name="publish_frequency" type="double" value="5.0" />

		</node>

	</group>

</launch>
```

==
=====================

Launch File 2 - Manipulator Controller:

===

```
<launch>

	<group ns="Hexacopter">

		<rosparam file="$(find fly_bot)/arm_controllers.yaml" command="load"/>

		<node name="controller_spawner" pkg="controller_manager" type="spawner" 
args="arm_controller"/>

		<node name="robot_state_publisher" pkg="robot_state_publisher" 
type="robot_state_publisher"/>

	</group>


</launch>
```

===

Python Script:

===

```
#!/usr/bin/env python

import sys, rospy, tf, random # moveit_commander

from geometry_msgs.msg import Pose, Point, Quaternion

from trajectory_msgs.msg import JointTrajectory, JointTrajectoryPoint

from math import pi

import time

from std_msgs.msg import Float64MultiArray


rospy.init_node('p_hexacopter', anonymous=True)

pub = rospy.Publisher('/Kwad/armp_controller/command', JointTrajectory, queue_size=10)

velPub = rospy.Publisher('/Kwad/joint_motor_controller/command', Float64MultiArray, 
queue_size=6)

rate = rospy.Rate(1)


traj = JointTrajectory()

traj.joint_names = ["hip", "shoulder", "elbow", "wrist"]
```

```
pointT = JointTrajectoryPoint()
pointT.positions = [0.0, 0.0, 0.0, 0.0]
pointT.time_from_start = rospy.Duration(1.0)
traj.points.append(pointT)

def trajectory(hp, shd, elb, wrs):
        traj.points.pop()
        pointT.positions = [hp, shd, elb, wrs]
        pointT.time_from_start = rospy.Duration(dur)
        traj.points.append(pointT)
        pub.publish(traj)

def wait(t):
        stime = time.time()
        while True:
                if time.time()-stime > t: break

f = Float64MultiArray()
def thrust(F):
        fl_motor_vel = F; fr_motor_vel = F
        l_motor_vel = F;        r_motor_vel = F
        bl_motor_vel = F; br_motor_vel = F
        f.data = [fr_motor_vel, -fl_motor_vel, l_motor_vel, -bl_motor_vel, br_motor_vel, -
r_motor_vel]
        velPub.publish(f)

ctime = time.time()
seq = 0
while not rospy.is_shutdown():
        if seq == 1:
                thrust(40);
        elif seq == 2:
```

```
        wait(1); thrust(40)
        trajectory(0.0,-0.6,-0.7, 0.0)
elif seq == 3:
        trajectory(0.0,-0.6,-0.7, 0.0); wait(1)
elif seq == 4:
        trajectory(0.0,-0.6,-0.7, 0.0)
elif seq == 5:
        trajectory(0.0,-0.6,-0.7, 0.0); wait(1)
elif seq == 6:
        trajectory(0.0,-0.6,-0.7, 0.0)
elif seq == 7:
        trajectory(0.0,-0.6,-0.7, 0.0); wait(1)
elif seq == 8:
        trajectory(0.0,0.0,0.7, 0.0)
elif seq == 9:
        trajectory(0.0,0.0,0.7, 0.0); wait(1)
elif seq == 10:
        trajectory(0.0,0.0,0.7, 0.0); wait(1)
elif seq == 11:
        trajectory(0.0,0.0,0.7, 0.0); wait(1)
elif seq == 12:
        trajectory(0.0,0.0,0.7, 0.0)
elif seq == 13:
        wait(2); trajectory(0.0,0.0,0.7, 0.0)
elif seq == 14:
        wait(3); trajectory(0.0,0.0,0.7, 0.8)
elif seq == 15:
        trajectory(0.0,-0.3,-0.4, 0.79)
seq = seq + 1
rate.sleep()
```

```
============================================================================
=======================World Simulation Setup:
============================================================================
=======================<sdf version='1.6'>
 <world name='default'>
  <light name='sun' type='directional'>
   <cast_shadows>1</cast_shadows>
   <pose frame=''>0 0 10 0 -0 0</pose>
   <diffuse>0.8 0.8 0.8 1</diffuse>
   <specular>0.2 0.2 0.2 1</specular>
   <attenuation>
    <range>1000</range>
    <constant>0.9</constant>
    <linear>0.01</linear>
    <quadratic>0.001</quadratic>
   </attenuation>
   <direction>-0.5 0.1 -0.9</direction>
  </light>
  <model name='ground_plane'>
   <static>1</static>
   <link name='link'>
    <collision name='collision'>
     <geometry>
      <plane>
       <normal>0 0 1</normal>
       <size>100 100</size>
      </plane>
     </geometry>
     <surface>
      <contact>
       <collide_bitmask>65535</collide_bitmask>
       <ode/>
    </contact>
    <friction>
      <ode>
        <mu>100</mu>
        <mu2>50</mu2>
      </ode>
      <torsional>
        <ode/>
      </torsional>
    </friction>
    <bounce/>
  </surface>
  <max_contacts>10</max_contacts>
</collision>
<visual name='visual'>
  <cast_shadows>0</cast_shadows>
  <geometry>
    <plane>
      <normal>0 0 1</normal>
      <size>100 100</size>
    </plane>
  </geometry>
  <material>
    <script>
      <uri>file://media/materials/scripts/gazebo.material</uri>
      <name>Gazebo/Grey</name>
    </script>
  </material>
</visual>
<self_collide>0</self_collide>
<enable_wind>0</enable_wind>
```

```
        <kinematic>0</kinematic>
      </link>
    </model>
    <gravity>0 0 -9.8</gravity>
    <magnetic_field>6e-06 2.3e-05 -4.2e-05</magnetic_field>
    <atmosphere type='adiabatic'/>
    <physics name='default_physics' default='0' type='ode'>
      <max_step_size>0.001</max_step_size>
      <real_time_factor>1</real_time_factor>
      <real_time_update_rate>1000</real_time_update_rate>
    </physics>

   <!-- PLACE URDF CODES HERE FOR OBJECTS IN THE WORLD -->

  </world>
</sdf>
```

==

Hexacopter URDF Xacro Code:

==

=-=

1. Variable Initialization and Imports:

=-=

```
<robot xmlns:xacro="http://www.ros.org/wiki/xacro" name="Kwad">

        <xacro:property name="width" value="0.0315" />
        <xacro:property name="length" value="0.45" />
        <xacro:property name="height" value="0.01" />
        <xacro:property name="mass_pr" value="0.0055" />
```

```
<xacro:property name="pi_value" value="3.14159263" />
<xacro:property name="radiusp" value="0.20" />
<xacro:property name="prop_loc" value="0.15909" />
<xacro:property name="prop_angle" value="0.1" />
<xacro:property name="Ixx_prop" value="0.0" />
<xacro:property name="Iyy_prop" value="0.0" />
<xacro:property name="Izz_prop" value="4.42448e-5" />
<xacro:property name="cyl_mass" value="0.00001" />
<xacro:property name="cyl_ix_iy" value="0.00000022333" />
<xacro:property name="cyl_iz" value="0.00000002" />
<xacro:property name="mass_fr" value="0.5" />
<xacro:property name="frame_ix" value="0.04989/2" />
<xacro:property name="frame_iy" value="0.04989/2" />
<xacro:property name="frame_iz" value="0.24057/2" />

<xacro:include filename="$(find fly_bot)/urdf/material.xacro" />
<xacro:include filename="$(find fly_bot)/urdf/Hexa.gazebo.xacro" />

<xacro:macro name="default_inertial" params="mass p ix_value iy_value iz_value" >
        <inertial>
                <origin xyz="0 0 0" rpy="0 0 ${p}" />
                <mass value="${mass}" />
                <inertia ixx="${ix_value}" ixy="0" ixz="0"
                iyy="${iy_value}" iyz="0" izz="${iz_value}" />
        </inertial>
</xacro:macro>
```

=-=

2. Drone Frame:

=-=

The following code is repeated 3 times to create 3 cuboids for the drone frame:

```
        <link name="base_link1">
                <visual>
                        <origin xyz="0 0 0" rpy="0 0 ${pi_value/6}" />
                        <geometry><box size="${width} ${length} ${height}" /></geometry>
                        <material name="red" />
                </visual>
                <collision>
                        <origin xyz="0 0 0" rpy="0 0 ${pi_value/6}" />
                        <geometry><box size="${width} ${length} ${height}" /></geometry>
                </collision>
                <!--xacro:default_inertial mass="${mass_fr}" p="${pi_value/6}"
ix_value="${frame_ix}" iy_value="${frame_iy}" iz_value="${frame_iz}" /-->
                <inertial>
                        <mass value="${mass_fr}" />
                        <inertia ixx="${frame_ix}" ixy="0" ixz="0"
                                        iyy="${frame_iy}" iyz="0" izz="${frame_iz}" />
                </inertial>

        </link>
```

=-=

3. Drone Plate:

=-=

The following code is repeated 2 times to make the center part of the drone:

```
        <link name="plate1">
                <visual>
                        <origin xyz="0 0 0" rpy="0 0 0" />
                        <geometry><box size="${0.12} ${0.12} ${0.01}" /></geometry>
                        <material name="black" />
                </visual>
```

```
		<collision>
			<origin xyz="0 0 0" rpy="0 0 0" />
			<geometry><box size="${0.12} ${0.12} ${0.01}" /></geometry>
		</collision>
		<!--xacro:default_inertial mass="${0.01}" p="${0}" ix_value="${0}"
iy_value="${0}" iz_value="${2.4-5}" /-->
	</link>
```

=-=

4. Fixed Joints:

=-=

The following code is repeated several times to fix the individual
parts of the drone frame:

```
	<joint name="joint_plate1_frame" type="fixed">
		<parent link="base_link1" />
		<child link="plate1" />
		<origin xyz="${0} ${0} 0.005" rpy="0 0 0" />
		<axis xyz="0 0 1" />
	</joint>
```

=-=

5. Brushless DC Motor:

=-=

The following code is repeated 6 times to make the cylindrical brushless motor:

```
	<link name="cyl1">
		<visual>
			<origin xyz="0 0 0" rpy="0 0 ${pi_value/2}" />
			<geometry><cylinder radius="0.016" length="0.02" /></geometry>
			<material name="golden" />
```

```
                </visual>


                <collision>
                        <origin xyz="0 0 0" rpy="0 0 ${pi_value/2}" />
                        <geometry><cylinder radius="0.016" length="0.02" /></geometry>
                </collision>
                <xacro:default_inertial mass="${cyl_mass}" p="${0}" ix_value="${cyl_ix_iy}"
iy_value="${cyl_ix_iy}" iz_value="${cyl_iz}" />
        </link>
```

=-=

6. Propeller:

=-=

The following code is repeated 2 times to make a single propeller.

This single propeller instance is repeated 6 times for all propellers.

```
        <link name="propel_front_right">
                <visual>
                        <origin xyz="0 ${radiusp*0.2} 0" rpy="0 ${prop_angle} 0" />
                        <geometry><box size="${0.03302/3} ${radiusp*0.4} ${0.001}"
/></geometry>
                        <material name="black" />
                </visual>
                <collision>
                        <origin xyz="0 ${radiusp*0.2} 0" rpy="0 ${prop_angle} 0" />
                        <geometry><box size="${0.03302/3} ${radiusp*0.4} ${0.001}"
/></geometry>
                </collision>
                <xacro:default_inertial mass="${mass_pr}" p="${0}" ix_value="${Ixx_prop}"
iy_value="${Iyy_prop}" iz_value="${Izz_prop}" />
        </link>
```

=-=

7. Propeller Joint:

=-=

The following code is repeated 6 times to implement a continuous joint
for the propellers. This makes the propellers free to rotate.

```
<joint name="joint_front_right_propel" type="continuous">
        <parent link="base_link1" />
        <child link="propel_front_right" />
        <origin xyz="0.1 0.1732 0.021" rpy="0 0 0"/>
        <axis xyz="0 0 1" />
</joint>
```

=-=

8. Propeller Transmission:

=-=

The following code is used to make a velocity-controlled actuation
at the propeller joints:

```
<transmission name="front_right_transmission" >
        <type>transmission_interface/SimpleTransmission</type>
        <joint name="joint_front_right_propel">

<hardwareInterface>hardware_interface/VelocityJointInterface</hardwareInterface>
        </joint>
        <actuator name="brushless_motor1">

<hardwareInterface>hardware_interface/VelocityJointInterface</hardwareInterface>
                <mechanicalReduction>1</mechanicalReduction>
        </actuator>
</transmission>
```

=-=

9. Manipulator - Base:

=-=

```
 <link name="base_arm">
   <visual>
     <geometry> <cylinder length="0.01" radius="0.05"/> </geometry>
     <material name="silver"> <color rgba="0.75 0.75 0.75 1"/> </material>
     <origin rpy="0 0 0" xyz="0 0 0.025" />
   </visual>
   <collision>
     <geometry> <cylinder length="0.01" radius="0.05"/> </geometry>
     <origin rpy="0 0 0" xyz="0 0 0.025" />
   </collision>
   <inertial>
     <mass value="0.1"/>
     <origin rpy="0 0 0" xyz="0 0 0.025"/>
     <inertia ixx="0.00006333" iyy="0.00006333" izz="0.000125" ixy="0" ixz="0" iyz="0"/>
   </inertial>
 </link>
```

=-=

10. Manipulator - Torso:

=-=

```
 <link name="torso">
   <visual>
     <geometry> <cylinder length="0.1" radius="0.01"/> </geometry>
     <material name="silver"/>
     <origin rpy="0 0 0" xyz="0 0 0.0" />
```

```
    </visual>
    <collision>
      <geometry> <cylinder length="0.1" radius="0.01"/> </geometry>
      <origin rpy="0 0 0" xyz="0 0 0.0" />
    </collision>
    <inertial>
      <mass value="0.05"/>
      <origin rpy="0 0 0" xyz="0 0 0.0"/>
      <inertia ixx="0.0000429" iyy="0.0000429" izz="0.0000025"  ixy="0" ixz="0" iyz="0"/>
    </inertial>
  </link>
```

=-=

11. Manipulator - Upper Arm:

=-=

```
  <link name="upper_arm">
    <visual>
      <geometry> <cylinder length="0.15" radius="0.01"/> </geometry>
      <material name="silver"/>
      <origin rpy="0 0 0" xyz="0 0 0.0"/>
    </visual>
    <collision>
      <geometry> <cylinder length="0.1" radius="0.01"/> </geometry>
      <origin rpy="0 0 0" xyz="0 0 0.0"/>
    </collision>
    <inertial>
      <mass value="0.05"/>
      <origin rpy="0 0 0" xyz="0 0 0.0"/>
      <inertia ixx="0.000095" iyy="0.000095" izz="0.0000025"  ixy="0" ixz="0" iyz="0"/>
    </inertial>
  </link>
```

=-=

12. Manipulator - Lower Arm:

=-=

```
 <link name="lower_arm">
   <visual>
     <geometry> <cylinder length="0.1" radius="0.01"/> </geometry>
     <material name="silver"/>
     <origin rpy="0 0 0" xyz="0 0 0.0"/>
     </visual>
   <collision>
     <geometry> <cylinder length="0.1" radius="0.01"/> </geometry>
     <origin rpy="0 0 0" xyz="0 0 0.0"/>
   </collision>
   <inertial>
     <mass value="0.05"/>
     <origin rpy="0 0 0" xyz="0 0 0.0"/>
     <inertia ixx="0.0000429" iyy="0.0000429" izz="0.0000025" ixy="0" ixz="0" iyz="0"/>
   </inertial>
 </link>
```

=-=

13. Manipulator - Gripper:

=-=

The following code implements the palm that holds the gripper joints:

```
 <link name="palm">
   <visual>
     <geometry> <box size="0.03 0.06 0.02"/> </geometry>
     <material name="silver"/>
   </visual>
```

```
    <collision>
      <geometry> <box size="0.03 0.06 0.02"/> </geometry>
    </collision>
    <inertial>
      <mass value="0.1"/>
      <inertia ixx="0.000015" iyy="0.000015" izz="0.000015" ixy="0" ixz="0" iyz="0"/>
    </inertial>
  </link>
```

The following code is repeated 2 times to make the gripper ends:

```
  <link name="grip1">
    <visual>
      <geometry> <box size="0.03 0.02 0.07"/> </geometry>
      <material name="silver"/>
      <origin rpy="0 0 0" xyz="0 0 0.04"/>
    </visual>
    <collision>
      <geometry> <box size="0.03 0.02 0.01"/> </geometry>
      <origin rpy="0 0 0" xyz="0 0 0.04"/>
      <surface>
            <contact>
              <ode>
                <max_vel>0.1</max_vel>
                <min_depth>0.001</min_depth>
              </ode>
            </contact>
            <friction>
              <ode>
                <mu>1000.0</mu>
                <mu2>1000.0</mu2>
              </ode>
            </friction>
          </surface>
    </collision>
    <inertial>
      <mass value="0.1"/>
      <inertia ixx="0.000015" iyy="0.000015" izz="0.000015" ixy="0" ixz="0" iyz="0"/>
    </inertial>
  </link>
```

=-=

14. Manipulator Joints:

=-=

```
 <joint name="hip" type="continuous">
  <axis xyz="0 0 1" />
  <parent link="base_arm" />
  <child link="torso" />
  <origin rpy="0 3.14 0" xyz="0.0 0.0 -0.02" />
 </joint>

 <joint name="shoulder" type="continuous">
  <axis xyz="0 1 0"/>
  <parent link="torso"/>
  <child link="upper_arm"/>
  <origin rpy="0 1.5708 0" xyz="0.066 0.0 0.055"/>
 </joint>

 <joint name="elbow" type="continuous">
  <axis xyz="0 1 0"/>
  <parent link="upper_arm"/>
```

```
   <child link="lower_arm"/>
   <origin rpy="0 -1.5708 0" xyz="-0.05 0.0 0.07"/>
  </joint>

  <joint name="wrist" type="continuous">
   <axis xyz="0 1 0"/>
   <parent link="lower_arm"/>
   <child link="palm"/>
   <origin rpy="0 0 0" xyz="0.0 0.0 0.03"/>
  </joint>

  <joint name="wrist1" type="revolute">
   <axis xyz="1 0 0" />
   <parent link="palm" />
   <child link="grip1" />
   <limit effort="100000.0" lower="-5.0" upper="5.0" velocity="1"/>
   <origin rpy="0 0 0" xyz="0.0 0.03 0.0"/>
  </joint>
  <joint name="wrist2" type="revolute">
   <axis xyz="1 0 0"/>
   <parent link="palm"/>
   <child link="grip2"/>
   <limit effort="100000.0" lower="-5.0" upper="5.0" velocity="1"/>
   <origin rpy="0 0 0" xyz="0.0 -0.03 0.0"/>
  </joint>
```

=-=

15. Manipulator Transmissions:

=-=

The following code is used to make a position-controlled actuation at the manipulator joints:

```
<transmission name="transmission1">
  <type> transmission_interface/SimpleTransmission </type>
  <joint name="hip">
    <hardwareInterface> PositionJointInterface </hardwareInterface>
  </joint>
  <actuator name="motor1">
    <hardwareInterface> PositionJointInterface </hardwareInterface>
    <mechanicalReduction> 1 </mechanicalReduction>
  </actuator>
</transmission>
```

=-=

16. Manipulator Plugin:

=-=

```
<gazebo>
  <plugin name="joint_state_publisher" filename="libgazebo_ros_joint_state_publisher.so">
    <jointName> hip, shoulder, elbow, wrist </jointName>
  </plugin>
</gazebo>
```