Gesture-Controlled Robotic Arm using Arduino

Dissertation

Submitted in partial fulfilment of the requirements
for the award of degree of

Bachelor of Technology

By

Mr. Prokash Pegu (ECE/20/03)
Mr. Keisham Biswadeep Singh (ECE/20/14)
Mr. Vijay Dui (ECE/20/09)

Under the supervision of:

Dr. Sahadev Roy

Assistant Professor

Department of Electronics & Communication Engineering

DEPARTMENT OF ELECTRONICS & COMMUNICATION ENGINEERING

National Institute of Technology

(Established by Ministry of Human Resource Development, Govt. of India)
(Deemed University)
Jote, District: Papumpare, Arunachal Pradesh 791123

National Institute of Technology

(Established by Ministry of Human Resource Development, Govt. of India)
Jote, District: Papum Pare, Arunachal Pradesh 791113

Phone: 0360-2001582, Fax: 0360-2284972, Email nitarunachal@gmail.com

Web-site: http://www.nitap.ac.in

CERTIFICATE FROM SUPERVISOR

This is to certify that the dissertation entitled " Gesture-Controlled Robotic Arm using Arduino " submitted by

Mr. Prokash Pegu (ECE/20/03)
Mr. Keisham Biswadeep Singh (ECE/20/14)
Mr. Vijay Dui (ECE/20/09)

as a partial fulfilment of his B.Tech. Degree in Electronics & Communication Engineering of the Institute is absolutely based upon his own work, carried out during the period from January 2024- May 2024 under my supervision. Neither this dissertation nor any part of it has been submitted for the award of any other degree of this Institute or any other Institute/University.

(Dr. Sahadev Roy)

Assistant Professor, Dept. of ECE

NIT Arunachal Pradesh

Supervisor

Date

National Institute of Technology

(Established by Ministry of Human Resource Development, Govt. of India)
Jote, District: Papum Pare, Arunachal Pradesh 791113

Phone: 0360-2001582, Fax: 0360-2284972, Email nitarunachal@gmail.com

Web-site: http://www.nitap.ac.in

CERTIFICATE OF APPROVAL

The dissertation entitled **"Gesture-Controlled Robotic Arm using Arduino,"** submitted by

Mr. Prokash Pegu (ECE/20/03)

Mr. Keisham Biswadeep Singh (ECE/20/14)

Mr. Vijay Dui (ECE/20/09)

is presented in a satisfactory manner to warrant its acceptance as a pre-requisite for the degree of Master of Technology in **VLSI & Embedded System Design** from the department of Electronics and Communication Engineering of National Institute of Technology, Arunachal Pradesh. It is understood that by this approval the undersigned do not necessarily endorse or approve any statement made, opinion expressed or conclusion drawn therein, but only for the purpose for which it has been submitted.

BOARD OF EXAMINERS:

ABSTRACT

This thesis describes the design and implementation of a gesture controlled robotic arm using gesture sensing technology in conjunction with an Arduino microcontroller. The project aims to create an intuitive and interactive control interface for robotic tasks, allowing users to control the robotic arm with gestures, contributing to its versatility and usability in various applications. The proposed system is expected to offer an intuitive and efficient control interface for robotic manipulation tasks, enabling users to interact with the robotic arm seamlessly through hand gestures. The project outcomes will contribute to the advancement of human-robot interaction technologies and open up new possibilities for applications in industries, healthcare, education, and entertainment. The development of a manually controlled robotic arm using hand gesture sensing technology and Arduino offers a great way to improve the usability and usability of robotic systems. This application paves the way for new applications in many areas by providing users with the ability to control the operation of the robot using hand movements.

ACKNOWLEDGEMENT

Dated

Foremost, we would like to express my sincere gratitude to my project supervisor Dr. Sahadev Roy, Assistant Professor, Department of Electronics and Communication Engineering for his constant guidance, motivation, enthusiasm, and immense knowledge. His guidance helped us in all the time of research and writing of this thesis. We could not have imagined having a better supervisor and mentor for our bachelor`s study.

Besides our supervisor, we would like to thank my thesis advisor: Mr. Himangshu Bora, PhD, Department of Electronics and Communication Engineering for his encouragement, insightful comments, and support at every step.

Last but not the least, we would like to thank our families for their love, constant support, and encouragement to pursue our goals and to make this thesis possible.

Prokash Pegu
(ECE/20/03)

Keisham Biswadeep Singh
(ECE/20/14)

Vijay Dui
(ECE/20/09)

LIST OF FIGURES

LIST OF ABBREVIATIONS

The abbreviation used in this report are preserved below: -

LAN	Local Area Network
MEMS	Micro-electromechanical system
ARM	Advanced reduced instruction set computer (RISC) machine
RF	Radio Frequency
DOF	Degree Of Freedom
AM	Additive Manufacturing
CAD	Computer-aided Design
FDM	Fused Deposition Modelling
ABS	Acrylonitrile Butadiene Styrene
PLA	Poly Lactic Acid
ADC	Analog-to-digital converter
PWM	Pulse Width Modulation
GND	Ground
IDE	Integrated Development Environment

LIST OF CONTENTS

TABLE OF CONTENTS

Chapter 1

Introduction

Robotic arms have long been an important tool in many industrial, scientific, and daily applications, providing high performance and efficiency in many tasks from assembly line manufacturing to the operating room. However, traditional interventions for controlling these robotic arms often require special training and lack the desired interactivity. As technology continues to advance, the need for natural and intuitive controls that can bridge the gap between humans and machines increases.

A promising solution that allows users to interact with robotic devices with simple hand gestures and gestures. Using advances in computer vision and machine learning, these machines can interpret and respond to human movements, providing users with a more intuitive and practical experience. The machines present an exciting opportunity to improve the control of robotic arms.

In the realm of robotics, the ability to seamlessly control movements with intuitive gestures is no longer the stuff of science fiction. With advancements in technology, particularly through the versatile Arduino platform and sophisticated hand sign detection algorithms, we now have the capability to interact with robotic arms in a manner reminiscent of orchestrating an ensemble with the wave of a conductor's baton.

This introduction serves as a gateway into the captivating world of gesture-controlled robotic arms, where Arduino serves as our conductor, translating hand signals into precise and dynamic movements. Let's embark on this journey to explore the fusion of hardware and software, where innovation meets human intuition.

Unveiling the Components

At the heart of our endeavour lies the Arduino microcontroller, a staple in the maker community renowned for its flexibility and ease of use. Paired with sensors capable of recognizing hand gestures, Arduino becomes the conduit through which our intentions are communicated to the robotic arm.

The robotic arm itself embodies the culmination of mechanical engineering prowess, designed to execute a myriad of tasks with precision and grace. With the addition of gesture control, its utility expands beyond mere automation, offering a mode of interaction that feels almost telepathic.

Understanding Hand Sign Detection

Central to the functionality of our gesture-controlled system is the ability to accurately interpret hand signals. This is achieved through a combination of hardware and software, wherein sensors capture the nuances of our gestures, while algorithms analyze and translate them into actionable commands.

Whether it's a simple wave, a clenched fist, or a delicate pinch, each hand sign carries unique significance, serving as the building blocks of our robotic communication. Through meticulous calibration and training, our system learns to distinguish between these gestures, enabling fluid and intuitive control of the robotic arm.

The Promise of Gesture-Controlled Robotics

Beyond the realm of novelty, gesture-controlled robotic arms hold immense potential in various fields, ranging from manufacturing and logistics to healthcare and beyond. By harnessing the power of human gestures, we bridge the gap between man and machine, simplifying complex tasks and enhancing collaboration.

In the pages that follow, we will delve deeper into the intricacies of building and programming a gesture-controlled robotic arm using Arduino, exploring the technical challenges and creative possibilities that lie ahead. Together, we will unlock the full potential of this transformative technology, ushering in a new era of human-robot interaction.

1.1 Objectives

- Design and implement a robotic arm system based on Arduino architecture.
- Develop algorithms for hand sign detection and recognition using computer vision techniques.
- Establish communication protocols between the hand sign detection module and the Arduino controller.
- Integrate the hand sign detection module with the robotic arm to enable real-time control.
- Evaluate the performance and usability of the gesture-controlled robotic arm in practical scenarios.

1.2 Research Problem

The research problem for a gesture-controlled robotic arm using Arduino and hand sign detection involves several aspects:

- **Gesture Recognition Accuracy**: One key challenge is achieving high accuracy in recognizing hand gestures. Research needs to focus on developing algorithms that can reliably interpret various hand signs with minimal errors and robustness to environmental factors like lighting conditions and background clutter.

- **Real-Time Performance**: Another important consideration is ensuring real-time performance of the gesture recognition system. Researchers need to explore efficient algorithms and hardware implementations that can process hand gestures swiftly to provide seamless control of the robotic arm without noticeable delays.

- **Adaptability to Different Users**: The system should be able to adapt to different users with varying hand sizes, shapes, and gestures. This requires designing algorithms that can generalize well across different individuals and accommodate a wide range of hand movements while maintaining accuracy and reliability.

- **Robotic Arm Control**: Integrating the gesture recognition system with the robotic arm controlled by Arduino introduces challenges related to motor control, kinematics, and dynamics. Researchers need to address how to translate recognized hand gestures into precise movements and actions of the robotic arm, considering factors such as joint limitations, workspace constraints, and collision avoidance.

- **User Interface Design**: Designing an intuitive user interface for interacting with the gesture-controlled robotic arm is crucial for usability and user experience. Research should explore different interface modalities, feedback mechanisms, and user interaction paradigms to ensure that the system is user-friendly and accessible to a wide range of users.

- **Robustness and Reliability**: The system should be robust and dependable in various operating conditions to ensure safe and efficient operation. This involves testing the system under different scenarios, including challenging environments and user behaviours, and identifying potential failure modes or performance degradation factors to mitigate them effectively.

Addressing these research challenges will contribute to the development of advanced gesture-controlled robotic arm systems that are accurate, responsive, adaptable, and user-friendly, paving the way for applications in areas such as assistive robotics, industrial automation, and interactive human-robot collaboration.

1.3 Scope of Implementation

The **main concept** of this project is to put into action a robotic arm. Although it can be executed in multiple ways, when **different parameters** are involved. Implementing a gesture-controlled robotic arm using Arduino offers a wide scope of applications across various fields, including but not limited to:

- **Industrial Automation**: Gesture-controlled robotic arms can be used in manufacturing plants for tasks such as pick-and-place operations, assembly, welding, and packaging. They can improve efficiency and flexibility in production lines by allowing operators to control the robot's movements intuitively (Fig 1.1).

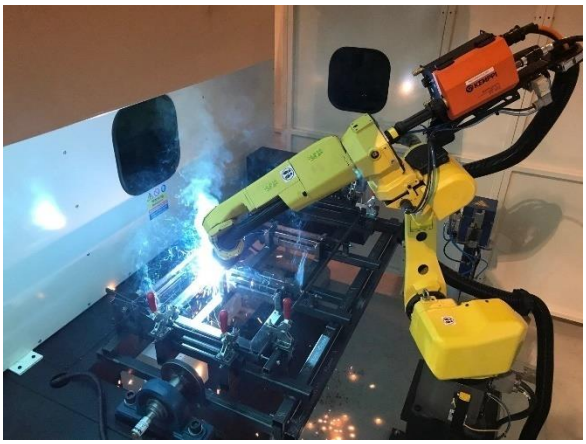

Fig. 1.1 Industrial Robo Arm

- **Healthcare**: In healthcare settings, gesture-controlled robotic arms can assist with tasks such as surgical procedures, rehabilitation exercises, and patient care. Surgeons can use gesture commands to manipulate robotic arms during minimally invasive surgeries, enhancing precision and reducing fatigue (Fig 1.2).

Fig.1.2 Surgical Robo Arm

- **Assistive Technology**: Gesture-controlled robotic arms can be employed as assistive devices for individuals with disabilities, enabling them to perform daily tasks independently. These devices can help with activities such as eating, dressing, and personal hygiene, enhancing quality of life for users (Fig 1.3).

Fig-1.3 Prosthetic Arm

- **Education and Research**: Arduino-based gesture-controlled robotic arms provide an accessible platform for teaching robotics, programming, and human-machine interaction concepts. They can be used in educational institutions to engage students in hands-on learning experiences and research projects (fig 1.4) .

Fig 1.4 Robo arm for education

- **Entertainment and Gaming**: Gesture-controlled robotic arms can be integrated into interactive entertainment systems and gaming platforms. Users can manipulate virtual objects or control in-game characters using hand gestures, providing immersive gaming experiences (fig 1.5).

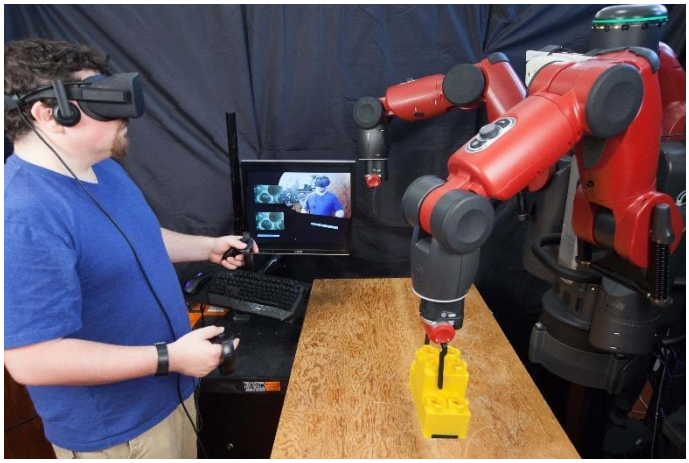

Fig 1.5 Gaming Robo arm

- **Home Automation**: Gesture-controlled robotic arms can be incorporated into smart home systems for tasks such as object manipulation, home security, and household chores. Users can control the robotic arm's movements through simple gestures, making home automation more intuitive and user-friendly (fig 1.6).

Fig 1.6 Household use Robo arm

- **Agriculture**: In agricultural applications, gesture-controlled robotic arms can be used for tasks such as crop harvesting, planting, and irrigation. They can increase efficiency and productivity in farming operations while reducing the physical strain on workers (fig 1.7).

Fig 1.7 Agriculture Robo arm

- **Search and Rescue**: In emergency situations, gesture-controlled robotic arms can be deployed for search and rescue operations in hazardous environments. They can be remotely operated by rescue teams to navigate through rubble and debris to locate and extract survivors (fig 1.8).

Fig 1.8 Rescue Robot

Chapter 2

Literature Review

There are various ways in which a robotic arm may be managed. Inside the beyond there were many researchers running to manipulate the robotic arm via pc terminals, joysticks, even interfacing them with the net so they can be managed from anywhere inside the globe.

Accelerometer-based control of an industrial robotic arm technique was proposed by **Pedro Neto** et al.,[1]. It is operated using 3-axis wireless accelerometers, attached to the human arms, capturing its behavior (gestures and postures). When compared with other common input devices, especially the teach pendant, this approach using accelerometers is more intuitive and easier to work. It is also more reliable. It has a response time is 160 milliseconds. The future scope is the implementation of a gyroscope into the system.

Malav Atul Doshi et al.,[2] proposed and analyzed the performance of a wireless robotic hand using flex sensors. The design and development of a robotic hand with real time control is precise and cost-effective. These five fingered robotic arms mimic a small degree of dexterity and is used for other applications such as prosthesis for leprosy patients. Robotic hand for tele-surgery using haptic technology was also implemented. But the major limiting factor was the time delay between the instructions.

Gourab Sen Gupta et al.,[3] proposed a Wi-Fi based control of a robotic arm with remote vision. The design of a controller intended for tele-operation can control an anthropomorphic robotic arm through a LAN or via the internet. The user can controlthe robotic arm remotely and access its sensory feedback signals as well. The camera mounted on the robot arm takes images and transmits them to the control station. The robot arm was controlled using a master-slave control methodology. The system was operated using PC based hardware and a combination of the old and new master rig. The robotic arm movement is essentially instantaneous and simultaneous and the original rotation encoding method using potentiometers has been used.

K. Brahmaniet al.,[4] used the technique of MEMS technology for controlling a robotic arm. It comprised of controlling of robotic arm powered with ARM7 based LPC1768 core. The LC1768 core was used and interfaced with DC motors of robotic arm to control their movements. MEMS, a three-dimensional accelerometer sensor is used to capture the gestures of human-arm and produce three analog output voltages in three dimensional axes. Two flex sensors are also used to control the gripper movement using a 2.4 GHz RF Module.

A survey on Arduino Controlled Robotic Arm by **Ankur Bhargava**et al.,[5]. In this paper a 5 Degree of Freedom (DOF) robotic arm have been developed. It is controlled by an Arduino Uno microcontroller which accepts input signals from a user by means of a set of potentiometers. The arm is made from four rotary joints and end effector also, where rotary motion is provided by a servomotor. Each link has been first designed using Solid works Sheet Metal Working Toolbox and then fabricated using a 2mm thick Aluminum sheet. The servomotors and links thus produced assembled with fasteners produced the final shape of the arm. The Arduino has been programmed to provide rotation to each servo motor corresponding to the amount of rotation of the potentiometer shaft. A robot can be defined according to the nature of the relative movements between the links that constitute it.

Review on development of industrial robotic arm by **Rahul Gautam**et al.,[6] This selective operation robotic control method needs to be overcome the problem such as placing or picking object that at distant from the worker. The robotic arm has been developed successfully as the movement of the robot can be controlled precisely. It is expensive to change the cable and therefore the designing to reduce the friction on table, is crucial to increase time between maintenance.

Survey on Design and Development of competitive low-cost Robot Arm With Four Degrees of Freedom by **Ashraf Elfasahany**et al.,[7] In this paper the representation of the design, development and implementation of robot arm is done, which has the ability to perform simple tasks, such as light material handling. The robotic arm is designed and made from acrylicmaterial where servo motors are used to perform links between arms. The servo motors consist of encoder so that no need to use controller. However, the rotation range of the motor is less than 180º span, which greatly decreases the region reached by the arm and the possible positions. The design of the robot arm was for four degrees of freedom. The end effector is not considered while designing because a readily available gripper is used as it is much easier and economical to use a commercial.

S. Pachaiyappan, **M. Micheal Balraj** and **T. Sridhar** have published a journal that contains complete data related to articulated arm robots for industrial applications [8]. They have developed an advanced technique in working of these robots from hazardous conditions and how the human can intervene into the robotic work zone. The ultimate motive in the research was to save human lives and in addition to increasing the productivity and quality of product with good and high technology environment. Nonetheless they have focused on safety and to create a good and healthy environment in the industry with use of advanced technology.

2.1 Additive Manufacturing

Additive manufacturing (AM), also known as 3D printing, is a manufacturing process that builds objects layer by layer from digital 3D models. Unlike traditional subtractive manufacturing methods, which involve cutting away material from a solid block, additive manufacturing adds material in a controlled manner to create the desired shape.

In additive manufacturing, the process typically begins with the creation of a digital 3D model using computer-aided design (CAD) software. This model is then sliced into thin horizontal layers using slicing software. Each layer is sent to the 3D printer, which deposits material (such as plastic, metal, or composite powders) layer by layer, based on the instructions from the sliced model.

There are several different technologies used in additive manufacturing, each with its own advantages and limitations. Some common additive manufacturing techniques include:

1. **Fused Deposition Modelling** (FDM): This method involves heating and extruding thermoplastic filament through a nozzle, which deposits the material layer by layer to create the object.
2. **Stereolithography** (SLA): SLA uses a liquid photopolymer resin that is selectively cured by a laser or UV light source, solidifying the resin layer by layer to build the object.
3. **Selective Laser Sintering** (SLS): In SLS, a high-powered laser selectively fuses powdered material (such as plastics, metals, or ceramics) together to create the object layer by layer.
4. **Direct Metal Laser Sintering** (DMLS): Similar to SLS, DMLS uses a laser to selectively fuse metal powders together, creating metal parts with high precision and strength.
5. **Binder Jetting**: In this process, a liquid binding agent is selectively deposited onto a bed of powdered material, binding the particles together to form the object layer by layer.

The additive manufacturing **process** begins with designing in CAD software based on designs and designs creat ed by experts in the field. The next step is to save the file in stl format. This file **only** describes the **geometry** of the 3D object **and does not store** information about texture, color, or other **CAD** model **properties**. The stl file is then processed **by** another software **which** is **responsible for cutting** the model **as there is no need** to print t he **model on** the additive manufacturing machine **(often called** 3D **printer paper).**

2.1.1 Advantages of Additive Manufacturing

Additive Manufacturing (AM), also known as 3D printing, offers numerous advantages across various industries:

- **Design flexibility**: Additive manufacturing enables complex geometry that is difficult or impossible to ac hieve using traditional methods. This design freedom enables innovative product designs and optimization for performance.
- **Cost-Effective Prototyping**: Traditional prototyping methods can be costly and time-consuming. AM allows for rapid prototyping, reducing lead times and costs associated with tooling and setup.
- **Customization and Personalization**: AM enables mass customization, allowing products to be tailored to individual customer needs without significantly impacting production costs.
- **Reduced Material Waste**: Traditional subtractive manufacturing methods often generate significant material waste. AM builds parts layer by layer, minimizing material waste and leading to more sustainable manufacturing processes.
- **On-Demand Manufacturing**: AM enables on-demand production, reducing the need for large inventories and storage space. This can lead to shorter supply chains and faster response times to market demands.
- **Complexity at No Extra Cost**: With AM, complexity doesn't necessarily increase production costs, unlike traditional manufacturing methods where intricate designs can be expensive to produce.
- **Improved Performance**: AM allows for the creation of lightweight structures and optimized components, leading to improved performance in terms of strength-to-weight ratio, thermal efficiency, and other key metrics.
- **Supply Chain Resilience**: AM can mitigate risks associated with supply chain disruptions by enabling local production and reducing reliance on global suppliers.
- **Tool-less Production**: Unlike traditional manufacturing processes that require expensive molds, dies, or fixtures, AM often requires little to no tooling, reducing upfront costs and lead times.

2.1.2 Disadvantages of Additive Manufacturing

While **additive manufacturing** (AM) offers numerous advantages, it also comes with some disadvantages:

- **Limited Material Selection**: While the range of available materials for AM is expanding, it still may not offer the same breath as traditional manufacturing processes. Some specialized materials may not be suitable for AM, limiting its applicability in certain industries [5].
- **Surface Finish and Resolution**: AM parts may have rougher surface finishes compared to parts produced using traditional methods. Achieving high-resolution finishes may require additional post-processing steps, increasing time and cost[6-9].
- **Build Time**: AM processes can be relatively slow compared to traditional manufacturing methods, especially for large or complex parts. Longer build times can impact production schedules and time-to-market [10].
- **Quality and Consistency**: Achieving consistent quality across AM parts can be challenging, particularly with certain AM technologies and materials. Variations in layer adhesion, porosity, and dimensional accuracy may require additional quality control measures[11].
- **Limited Scaling for Mass Production**: While AM is well-suited for low-volume production and prototyping, scaling up to high-volume production can be challenging. The speed and efficiency of AM processes may not be competitive with traditional mass production methods[12].
- **Equipment Costs**: Initial investment costs for AM equipment can be high, especially for industrial-grade machines capable of producing large or high-quality parts. Additionally, ongoing maintenance and material costs can contribute to the total cost of ownership[13].
- **Post-Processing Requirements**: AM parts often require post-processing steps such as machining, sanding, or surface treatment to achieve desired tolerances, surface finishes, or mechanical properties [14-16].
- **Design Constraints**: While AM offers design freedom, certain design constraints must be considered to ensure successful printing, such as overhangs, support structures, and minimum feature sizes. Designing for AM may require specialized knowledge and expertise.
- **Environmental Concerns**: Some AM processes involve the use of materials that can be hazardous to health or the environment if not handled properly. Additionally, energy consumption during printing and post-processing steps can contribute to environmental impacts [17].
- **Intellectual Property Risks**: The ease of sharing digital design files in AM raises concerns about intellectual property (IP) protection. Unauthorized reproduction of designs or the potential for design piracy is a risk in the AM ecosystem [18].

2.2Fused Deposition Modelling

FDM or Fused Filament Fabrication (FFF) is an additive manufacturing process that produce parts in thermoplastics polymers by utilizing the material extrusion technique. In this technique, the filament of thermoplastic polymer that comes in the form spool is pushed through a nozzle after being passed through a heated chamber. The FDM is machine is responsible to selectively deposit the melted filament in a layer-by-layer manner according to the designed part. The FDM technology is the most famous and widely used 3D printing technique in the world [19].

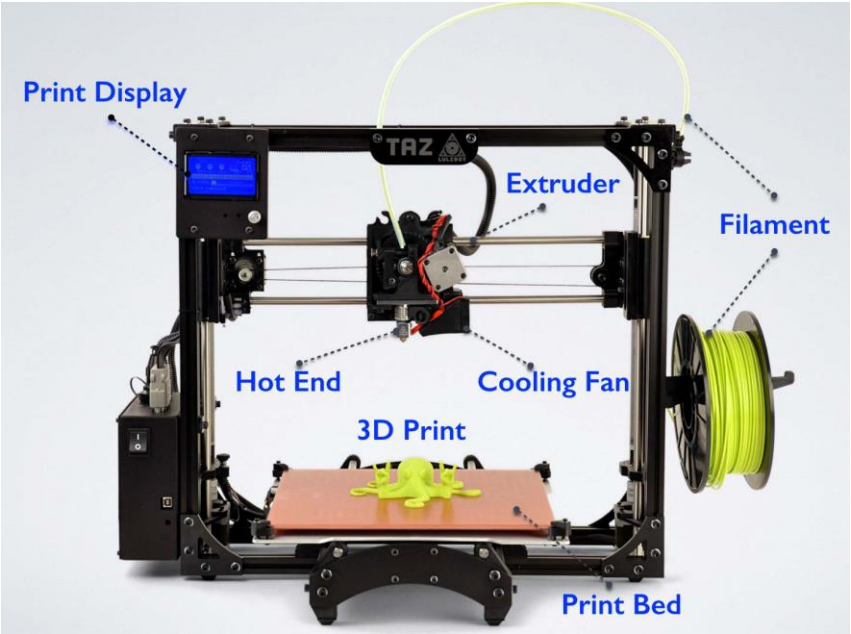

Fig-2.1 FDM Machine

2.2.1 Parameters of an FDM Machine

Some common FDM machine parameters are:

- **Temperature**: The temperature of nozzle and the build platform (bed).
- **Build speed**: The maximum and minimum speeds achievable in the x, y and z directions.
- **Layer height**: The minimum height deposited by the nozzle of the filament. The lower the height, the better the accuracy of the part.
- **Build size**: The size of the bed where the part will be printed.
- **Speed of cooling fans**: The option to turn the cooling fan on or off or the option to change the speed according to requirements.

2.2.2 Design Parameters for FDM

Design parameters for FDM include various factors that influence the quality, strength, and dimensional accuracy of printed parts. Here are some key design parameters to consider:

- **Layer Height**: Layer height refers to the thickness of each layer deposited during printing. Smaller layer heights typically result in smoother surface finishes and finer details but increase print time.

- **Extrusion Temperature**: The extrusion temperature determines the viscosity and flow characteristics of the filament material. It should be set according to the material manufacturer's recommendations to ensure proper extrusion and adhesion between layers.

- **Bed Temperature**: The bed temperature affects adhesion between the first layer and the build platform. It varies depending on the filament material, with higher temperatures often required for materials like ABS and lower temperatures for PLA. Using a heated bed can help prevent warping and improve print quality.

- **Print Speed**: Print speed influences the time taken to complete a print and can affect surface quality and part strength. Higher print speeds may result in rougher surface finishes and reduced detail resolution but can decrease print time. Optimal print speeds depend on factors such as material, layer height, and printer capabilities.

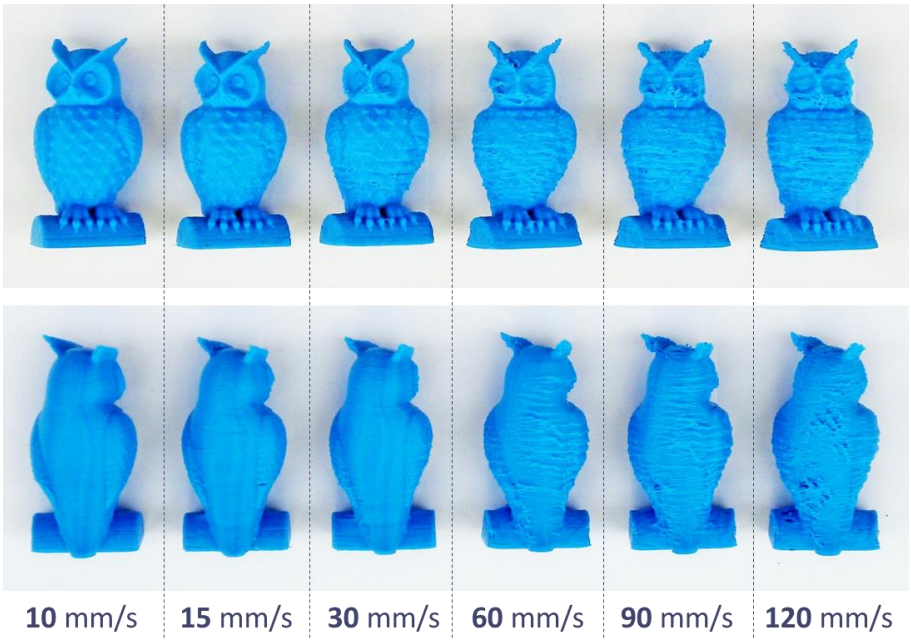

Fig 2.2 Print speed vs. Quality

- **Infill Density**: Infill density refers to the percentage of the internal volume of the printed part that is filled with material. Higher infill densities increase part strength but also increase material usage and print time. Common infill densities range from 10% to 100%, with 20% to 30% being typical for general-purpose prints.

- **Infill Pattern**: FDM printers offer various infill patterns, such as honeycomb, grid, and rectilinear. Each pattern has different properties in terms of strength, weight, and print time. Choosing the right infill pattern depends on the specific requirements of the part.

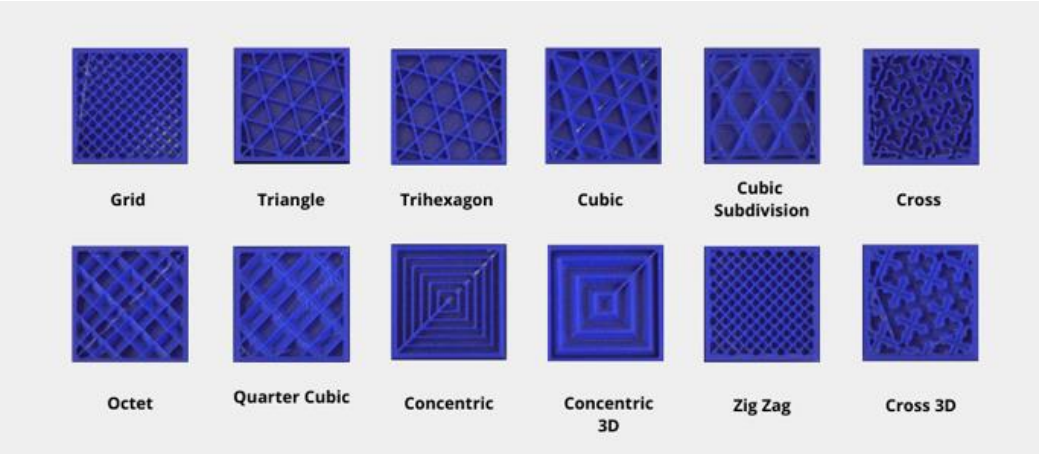

Fig 2.3 Types of Infill Pattern

- **Shell Thickness**: Shell thickness, also known as perimeter or wall thickness, refers to the number of perimeter layers around the part. Increasing shell thickness improves part strength and surface quality but also increases print time and material usage.

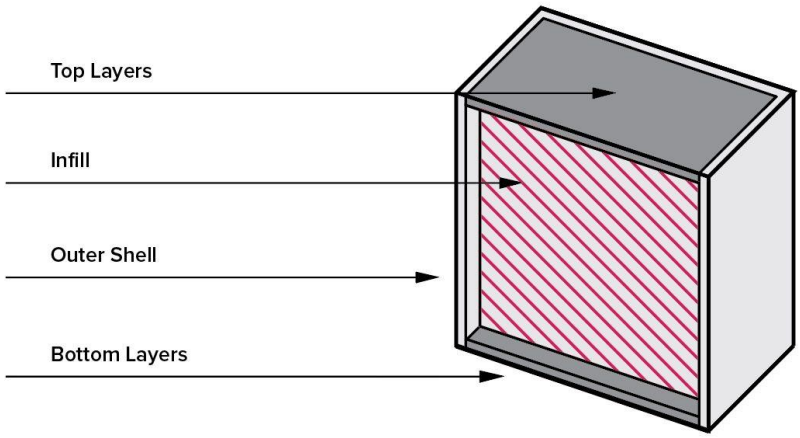

Fig 2.4 Section of a 3D print object

- **Print Orientation**: The orientation of the part on the build platform can affect strength, surface quality, and dimensional accuracy. Orienting the part to minimize overhangs and support material usage is important. Additionally, considering layer adhesion and mechanical properties along different axes is crucial for load-bearing parts.

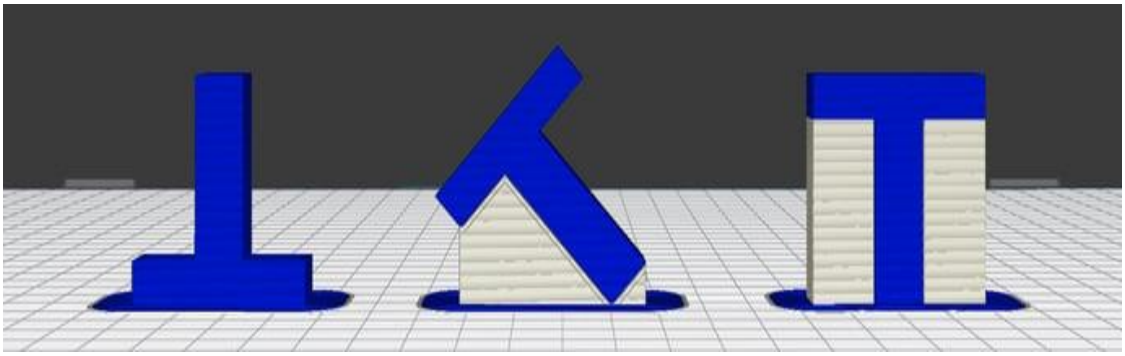

Fig2.5 Types of Orientation

- **Support Structures**: Overhangs and bridges may require support structures to prevent sagging during printing. Support material adds material and time to the print and requires post-processing removal.

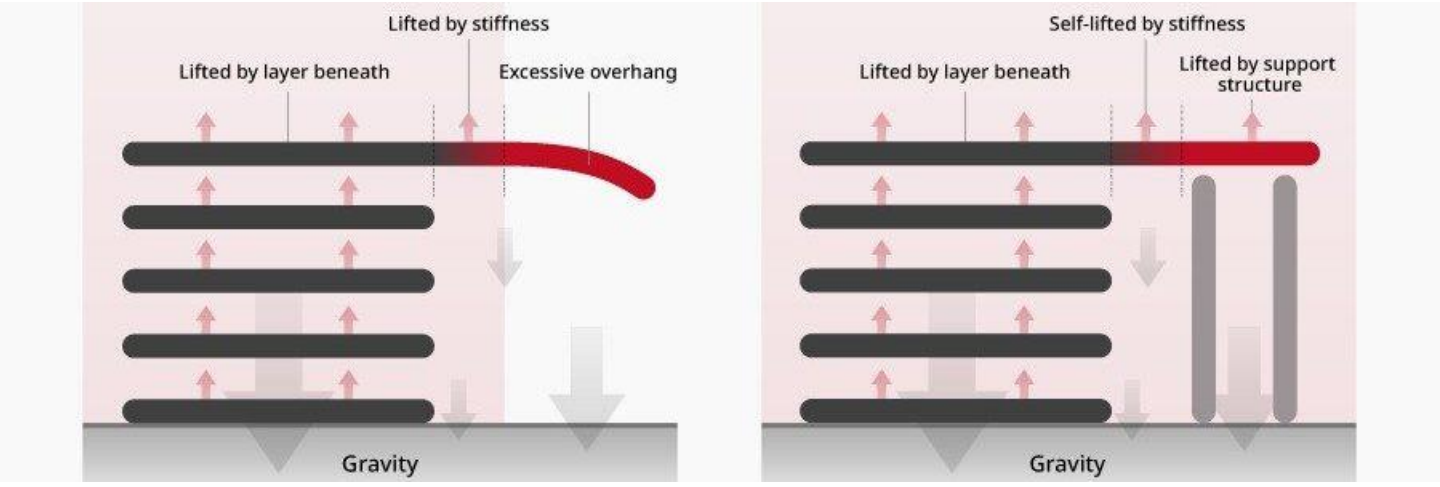

Fig2.6 Support Structures

- **Cooling**: Proper cooling is essential to prevent overheating and warping, particularly for small features and overhangs. FDM printers often use fans to cool the printed layers. Adjusting fan speed and direction can help improve print quality and dimensional accuracy.

2.2.3 Problems in FDM

While **Fused Deposition Modeling** (FDM) is a popular and accessible 3D printing technology, it also comes with its own set of challenges and limitations. Some common problems encountered in FDM printing include:

- **Poor Adhesion to Build Plate**: One of the most common issues is poor adhesion between the first layer of the print and the build plate. This can result in warping, lifting, or even failed prints. Factors contributing to poor adhesion include improper bed leveling, inadequate bed temperature, and surface contamination.

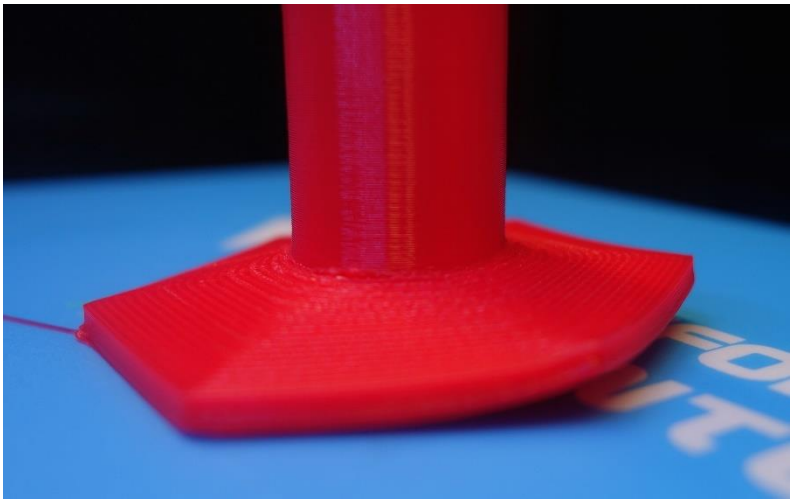

Fig 2.7 Poor Adhesion

- **Warped or Deformed Parts**: Warping occurs when different parts of the print cool at different rates, causing uneven contraction and distortion. This problem is often exacerbated by large temperature variations, insufficient cooling, and inadequate bed heating. Warping can lead to dimensional inaccuracies and structural weaknesses in the printed part.

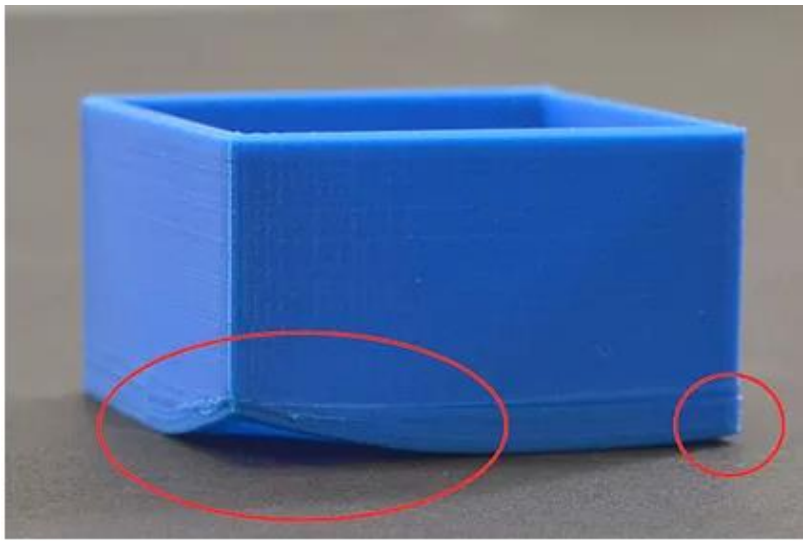

Fig 2.8 Warping

- **Layer Adhesion Issues**: Weak layer adhesion can result in delamination or splitting between layers, compromising the strength and integrity of the print. Factors such as improper extrusion temperature, inadequate cooling, and poor filament quality can contribute to this problem.

Fig 2.9 Cracking

- **Stringing and Oozing**: Stringing refers to the formation of thin strands or threads of filament between different parts of the print, while oozing occurs when excess filament leaks out of the nozzle during non-printing movements. These issues can lead to messy prints, rough surface finishes, and functional problems with moving parts.

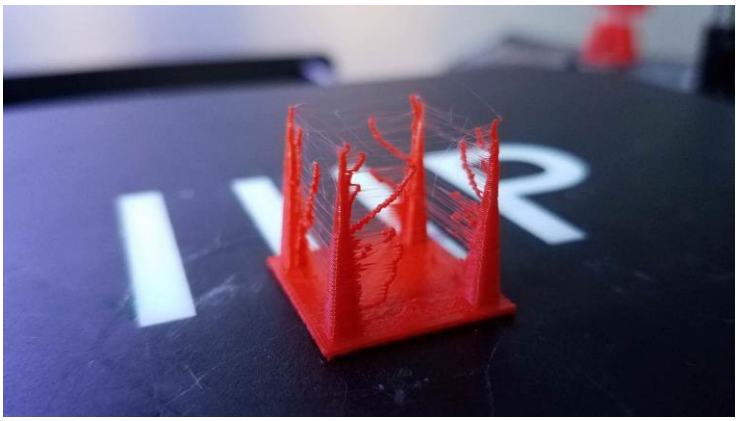

Fig 2.10 Stringing

- **Poor Surface Quality**: FDM prints often exhibit visible layer lines and surface imperfections, detracting from the overall aesthetics of the print. Achieving smooth surfaces and fine details can be challenging, particularly with larger layer heights and faster print speeds.
- **Overhangs and Bridging**: FDM printers struggle with printing overhangs and bridging without the use of support structures. Unsupported features may sag or deform during printing, resulting in poor surface quality and dimensional inaccuracies.
- **Extrusion Problems**: Inconsistent extrusion can cause under-extrusion (gaps or missing layers) or over-extrusion (bulging or blobbing), leading to print defects and functional issues. Factors such as clogged nozzles, filament diameter variations, and improper extrusion settings can contribute to extrusion problems.
- **Nozzle Clogging**: Nozzle clogging occurs when foreign particles or melted filament residue accumulate and block the extrusion nozzle, disrupting filament flow and causing print defects. Regular maintenance, proper filament storage, and using high-quality filaments can help prevent nozzle clogging.

2.2.4 FDM Material

The material we are using during our project is PLA.Polylactic Acid (PLA) is a biodegradable thermoplastic polymer derived from renewable resources such as corn starch, tapioca roots, or sugarcane. It is one of the most commonly used materials in additive manufacturing (3D printing) due to its ease of use, biodegradability, and wide availability.

Fig 2.11 PLA Material

Here are some key characteristics of PLA material:

- **Biodegradability:** PLA is biodegradable under certain conditions, making it an environmentally friendly alternative to traditional petroleum-based plastics. It can decompose into carbon dioxide and water in industrial composting facilities.
- **Ease of Printing:** PLA has excellent printability characteristics, including low warp and good adhesion to build platforms. It can be printed at relatively low temperatures compared to other materials, making it compatible with a wide range of 3D printers.
- **Low Toxicity:** PLA is considered safe for home use and generally does not produce toxic fumes or odors during printing. However, it's always advisable to operate 3D printers in a well-ventilated area.
- **Wide Color Range:** PLA filament is available in a wide variety of colors and finishes, including opaque, translucent, and metallic finishes, making it suitable for a range of aesthetic applicat

- **Strength and Rigidity:** While PLA is not as strong or heat resistant as some other thermoplastics like ABS (Acrylonitrile Butadiene Styrene), it still offers good strength and rigidity for many applications, especially when printed with infill patterns and higher layer heights.
- **Post-Processing:** PLA can be easily post-processed using techniques such as sanding, painting, and chemical smoothing to achieve desired surface finishes and textures.
- **Applications:** PLA is commonly used for prototyping, hobbyist projects, educational purposes, and low-stress functional parts. It's also used in applications where biodegradability is a desirable trait, such as food packaging, disposable utensils, and medical implant

Chapter 3

Selected Materials and Components

3.1 Microcontroller Board

Arduino is an **open**-source **electronic** platform based on simple hardware and software. It consists of a microcontroller board (the "Arduino board") and a development environment (Arduino IDE) used for writing, compiling, and uploading code to the board. Here are some key features of Arduino microcontroller boards:

- **Microcontroller:** At the heart of every Arduino board is a microcontroller chip. The most common microcontroller used in Arduino boards is the Atmel AVR series, particularly the ATmega328P, although other models are also used in some variants. These microcontrollers come with built-in flash memory, RAM, EEPROM, and various peripherals like digital I/O pins, analog-to-digital converters (ADCs), timers, and communication interfaces.

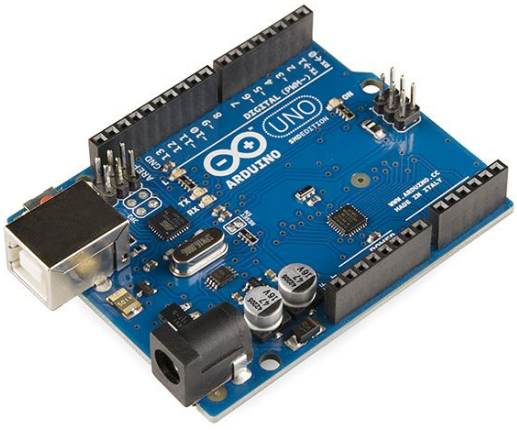

Fig 3.1 Arduino UNO.

- **Digital and Analog I/O:** Arduino boards typically feature a set of digital input/output (I/O) pins and analog input pins. These pins can be used to interface with various electronic components and sensors, allowing the Arduino to interact with the physical world.

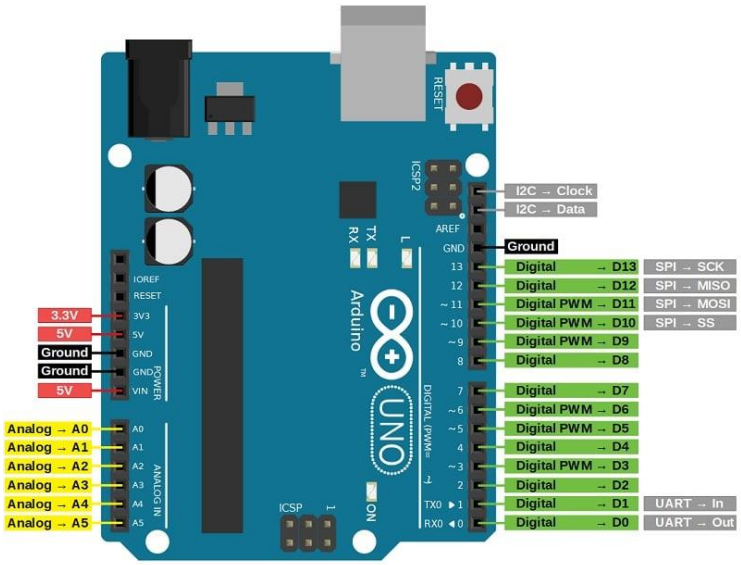

Fig 3.2 Analog and Digital pin

- **Communication Interfaces:** Arduino boards support various communication protocols, including UART (Serial), I2C, SPI, and USB. These interfaces enable communication with other devices such as sensors, actuators, displays, and computers.

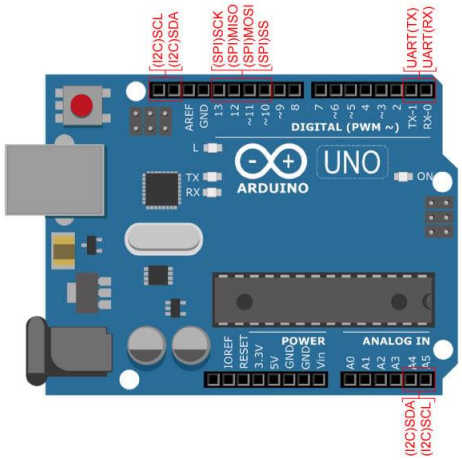

Fig 3.3 Communication Interfaces

3.2 Micro/Mini 180-degree Rotation Servo motor

A **micro-180-degree servo** motor is a type of electric motor commonly used in robotics, automation, and hobbyist projects. Unlike standard motors that rotate continuously in one direction, servo motors can precisely control the position of their output shaft within a limited range of motion, typically up to 180 degrees.

These motors consist of a small **DC motor** mechanically coupled to a gear reduction system and a **feedback** mechanism, such as a potentiometer or an encoder. The feedback mechanism provides positional information to the motor controller, allowing it to accurately adjust the motor's position based on the desired target angle.Micro 180-degree servo motors are **compact** in size and lightweight, making them suitable for applications where space is limited, or weight is a concern. They are commonly available in various torque ratings, speed specifications, and operating voltages to suit different application requirements.

Controlling a micro servo motor typically involves sending it a **pulse-width modulated** (PWM) signal from a microcontroller or other electronic device. The width of the PWM signal determines the desired position of the servo motor's shaft. By adjusting the width of the PWM signal, the motor controller can accurately position the servo motor to the desired angle within its operational range.

These motors find applications in a wide range of fields, including robotics, RC vehicles, model airplanes, camera gimbals, and animatronics. They are prized for their precision, reliability, and ease of use, making them a popular choice for both beginners and experienced enthusiasts alike.

Fig 3.4 Micro Servo Motor

3.3 Metal Gear Servo Motor MG995-180º

The **Metal Gear Servo Motor** MG995-180º is a specific type of servo motor known for its **durability**, strength, and precision. As its name suggests, it features metal gears, which provide increased torque and reliability compared to plastic gear servos, making it suitable for applications that require higher loads or more robust operation.

This servo motor has **a rotational range** of 180 degrees, meaning it can rotate its output shaft within this specified angular range. This range allows for precise control over the position of attached mechanisms or devices, making it ideal for various robotics, automation, and hobbyist projects.

The MG995-180º servo motor typically operates on a voltage range of around **4.8V** to **7.2V**, although it may be compatible with higher voltages within certain limits. It can be controlled using pulse-width modulated (PWM) signals from a microcontroller or servo controller, with the width of the PWM signal determining the desired position of the motor's shaft.

Due to its metal gear construction, the MG995-180º servo motor offers increased durability and resistance to wear, making it suitable for demanding applications where reliability is paramount. It is commonly used in robotics, RC vehicles, mechanical arms, camera gimbals, and other projects that require precise motion control and high torque capabilities.

Fig 3.5 Metal Gear Servo Motor

3.4 Nuts and bolts

In our robotic arm, 20mm nuts and bolts are used as part of the mechanical structure to assemble various components together. Here's how they might be utilized:

1. **Joint Connections**: The robotic arm's joints are typically constructed using a combination of brackets, plates, and shafts. 20mm bolts could be used to fasten these components together, providing the necessary strength and stability for the arm's movement.

2. **Linkages**: The links or segments of the robotic arm are often connected using bolts and nuts. These connections allow for articulation and movement between the segments while maintaining structural integrity. 20mm bolts may be used to secure the links together, ensuring smooth and precise motion.

3. **End Effector Mounting**: The end effector, or tool attached to the end of the robotic arm, such as a gripper or a welding torch, may be mounted using bolts and nuts. The 20mm bolts could be used to attach the end effector securely to the arm, allowing it to perform its intended tasks with precision.

4. **Base and Mounting**: The base of the robotic arm, which is often attached to a stationary platform or the chassis of a robot, may also require sturdy fasteners. 20mm bolts could be used to securely mount the robotic arm to its base, providing a stable foundation for its operation.

Fig 3.6 Nuts and Bolts

3.5 Jumper wires

Jumper wires are **short, insulated** wires with connectors at each end, commonly used to create temporary electrical connections on a breadboard or between electronic components. They allow for quick and easy prototyping and experimentation in electronics projects, eliminating the need for soldering. Jumper wires come in various lengths, **colors**, and types of connectors (such as male-to-male, male-to-female, and female-to-female), making them versatile for different applications.

Fig 3.7 Male to Male Jumper Wires

Fig 3.8 Female to Female Jumper Wires

Fig 3.9 Female to Male Jumper Wires

3.6 Breadboard

Using a breadboard in connection with an Arduino for a robotic arm project provides a flexible and convenient platform for prototyping and testing the electrical components and control logic of the arm.Connect the power rails of the breadboard to the Arduino to provide power to the components. Use jumper wires to connect the 5V and GND pins on the Arduino to the positive (+) and negative (-) power rails on the breadboard, respectively.

Our robotic arm includes DC motors, stepper motors, or servo motors, we connect them to motor drivers or servo controllers on the breadboard. We will use jumper wires to connect the control pins of the motor drivers or servo controllers to the Arduino's digital or PWM pins, depending on the motor type and control requirements.

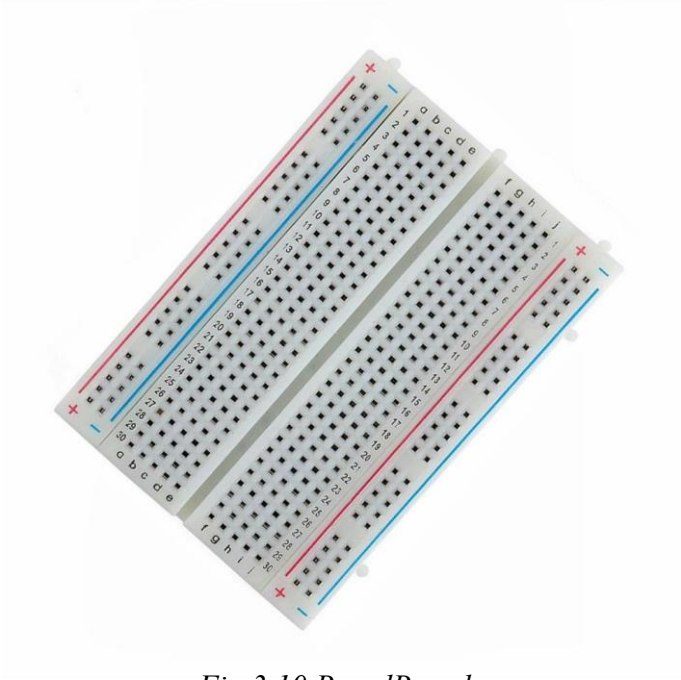

Fig 3.10 BreadBoard

Chapter 4

Methodology

4.1 SOLIDWORKS

4.1.1 Installation of SOLIDWORKS

Installing the app requires several instructions to be followed as mentioned below: -

1. Go to the SolidWorks Download Page https://www.solidworks.com/sw/support/downloads.htm.
2. If you are visiting the downloads page for the first time you will need to pick log in.

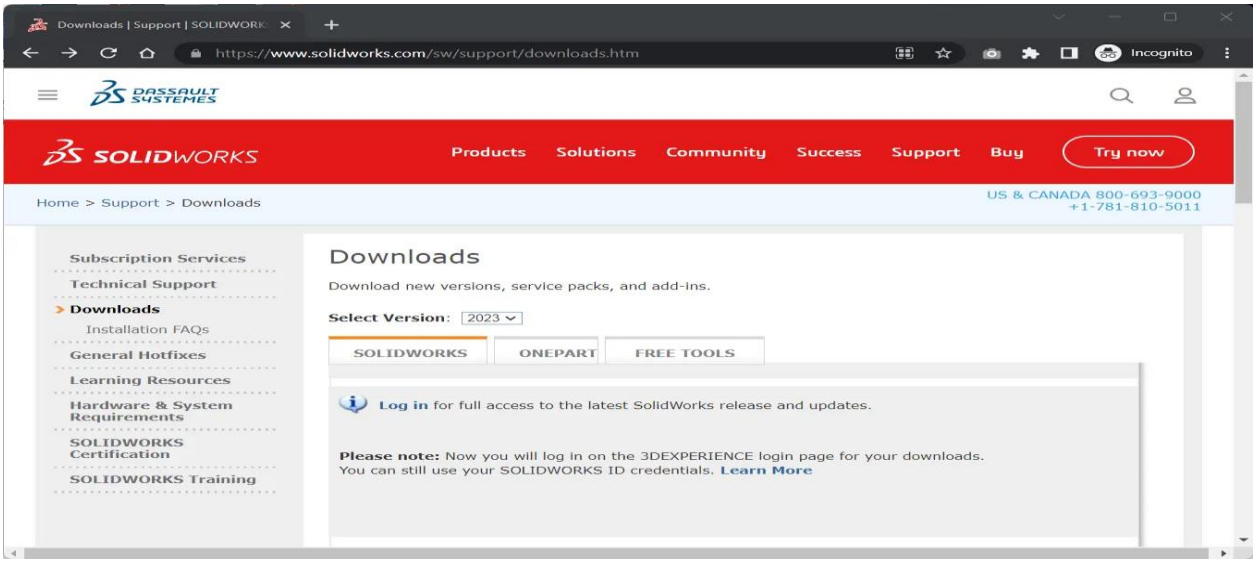

Fig 4.1 Download page for SolidWorks Application

3. On the 3DEXPERIENCE ID login page (DSXClient) enter your email address and password and pick the log in button. If you do not have a 3DEXPERIENCE ID you can pick the Create your 3DEXPERIENCE ID link and use your email address to create a new account.
4. After successfully logging into the DSXClient, you will be redirected to the downloads page on the SOLIDWORKS website.

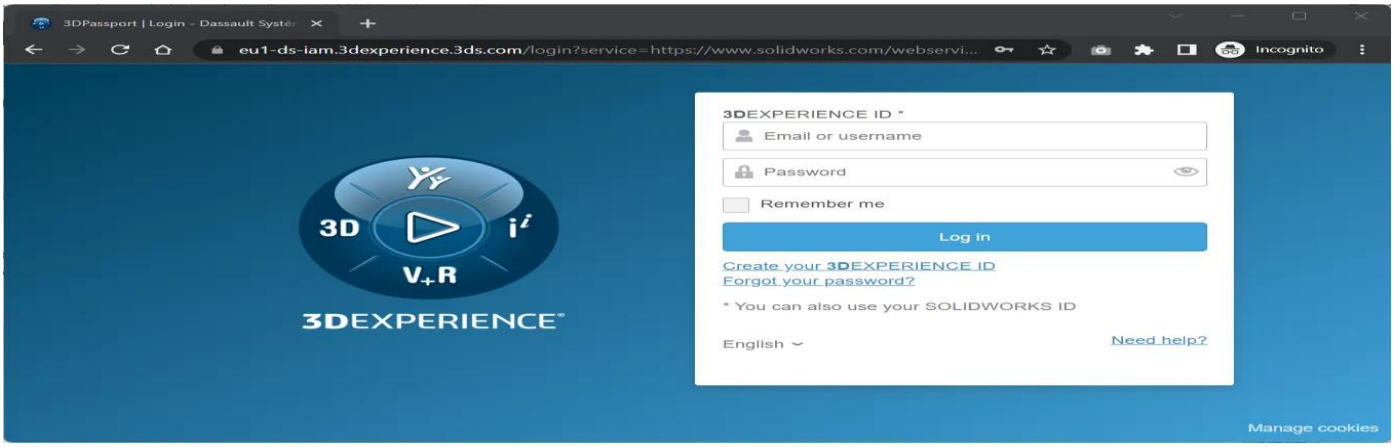

Fig 4.2 Log In page for SolidWorks

5. You can now access SOLIDWORKS desktop software directly from the SOLIDWORKS website. You have the option to download from the four most recent versions/releases of SOLIDWORKS and select the specific Service Pack (SP) required for your needs.
6. In Downloads page select the version you require from the dropdown menu
7. Pick the SOLIDWORKS Products link adjacent to the service pack you require (pick the highest number SP if you are a new customer):

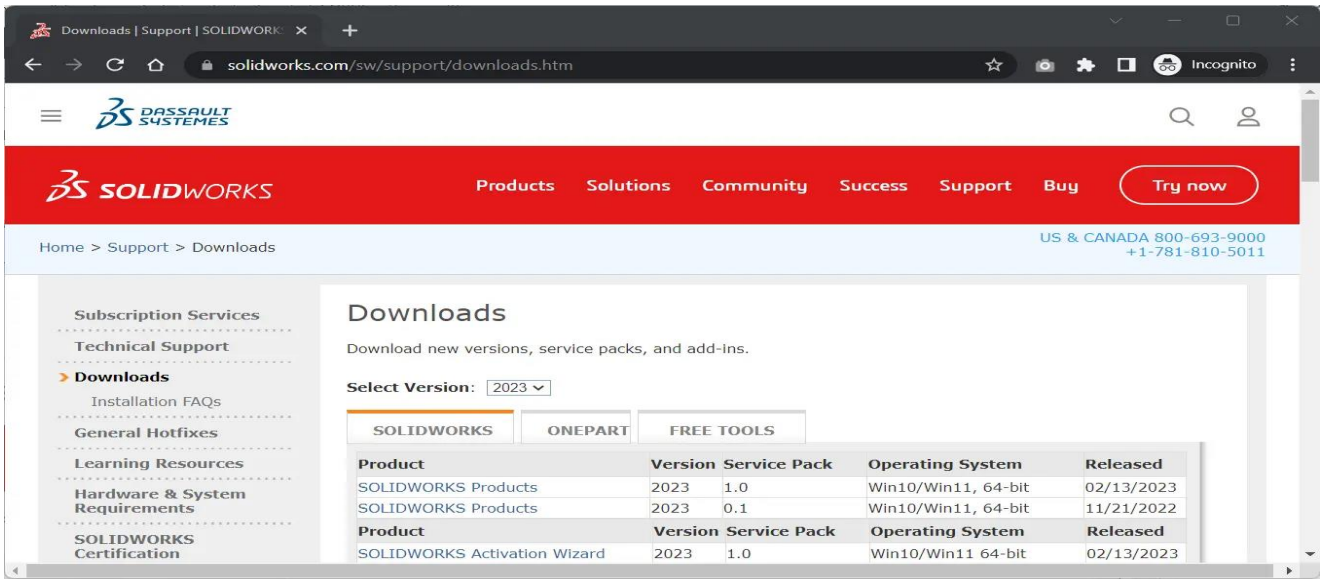

Fig 4.3 Download Page

8. You then need to accept the license agreement to download the software by picking the Accept Agreement and Continue button (Fig 4.4)

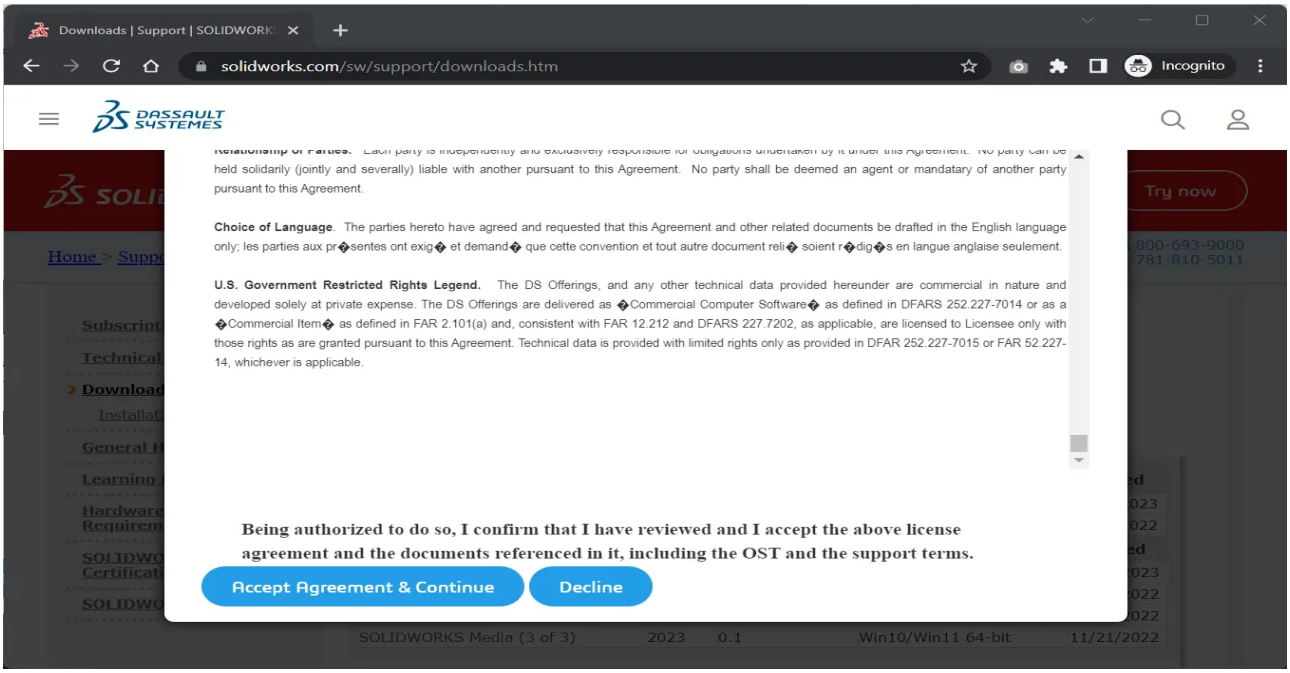

Fig 4.4 Agreement page

You can now download the SOLIDWORKS Installation Manager to begin the installation process.:

9. Select the Download link in the page and a zip file will download from the SOLIDWORKS web site. This 32MB file automates the installation process, sparing you the need to manually download files.
10. After the SolidWorksSeup.exe zip file has completed downloading run the SolidWorksSeup.exe file from your local machine
11. You may be asked to authorize the app to make changes on your device, select Yes

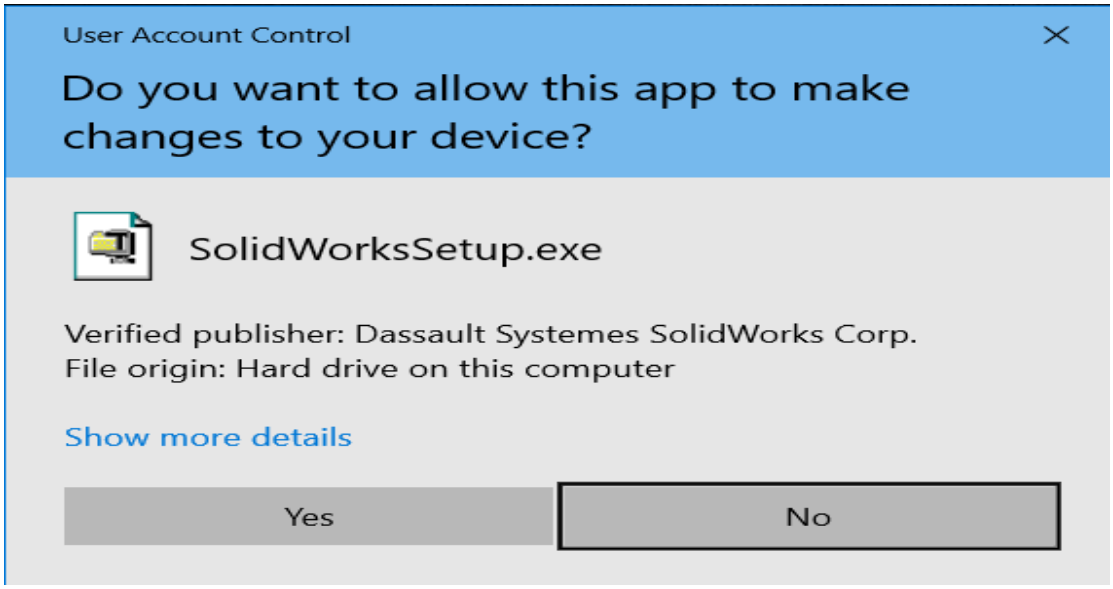

Fig 4.5 Authorization Page

12. You can then Browse to the location where you want the files to unzip using the browse button on the WinZip dialog or use the default location

NOTE: not all the SOLIDWORKS installation files will download at this point, just the Installation Manager

13. Select Unzip and the files will be extracted and the SOLIDWORKS Installation Manager will load

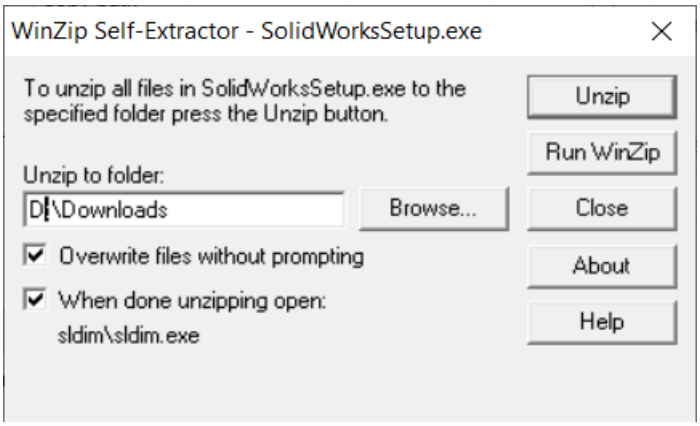

Fig 4.6 Extraction Page and SOLIDWORKS installation Manager

14. After the files have extracted select OK and the SOLIDWORKS Installation Manager will start automatically

IMPORTANT NOTE: During installation the SOLIDWORKS Installation Manager will determine the files you need based on the serial number you enter, this could by up to 16GB. Prepare yourself for a substantial download and ensure that your machine has sufficient space to accommodate both the downloaded files and the installation process.

15. The initial step in the installation procedure involves selecting the type of installation needed from four available options. For this demonstration, we will opt for the widely chosen "Install on this computer" option. Additional alternatives encompass.
16. Administrative image enables the download of a collection of installation files for deployment across multiple machines.
17. Server components facilitate network licensing installations.
18. Download and share all files feature enables downloading all installation files for future deployment

19. After selecting Install on this computer, proceed by clicking Next.

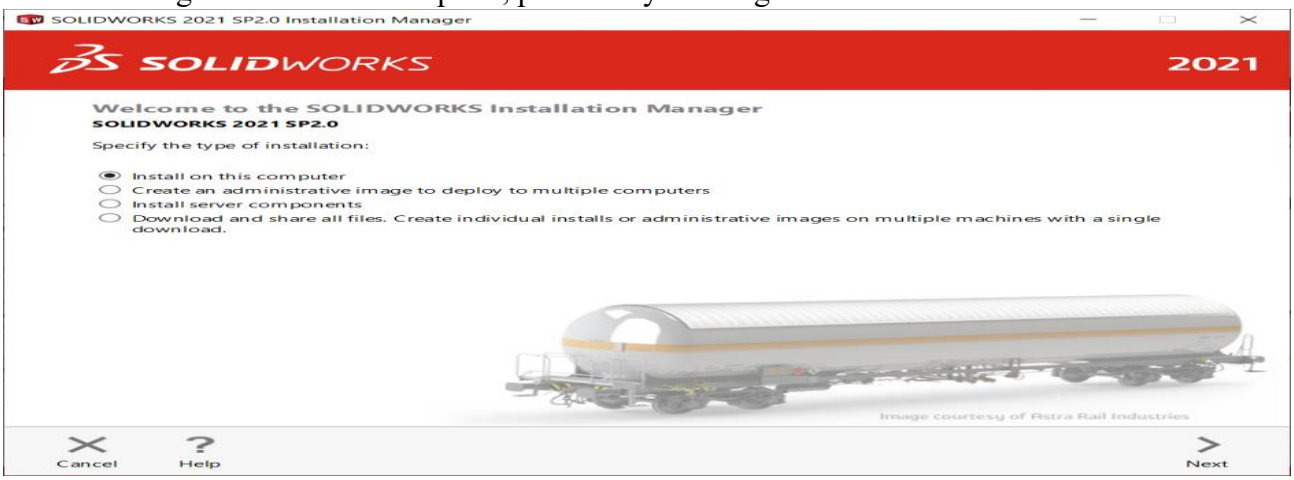

Fig 4.7 Installation page

20. Serial Number Entry
21. As part of the installation process, you will be prompted to enter the serial number[s] for your purchased SOLIDWORKS products. This stage allows you to install any licenses you have acquired, including those for 3D Design, Visualization, Simulation, CAM, Technical Communication, and Electrical Design.
22. After selecting the products, you wish to install, you will proceed to enter the corresponding SOLIDWORKS serial number[s]. You have the flexibility to install only the desired products at this stage, with the option to modify the installation later if needed. Please be aware that automatically populating the serial numbers using the Log In button necessitates the creation of a SOLIDWORKS Admin Portal account. Alternatively, you can simply enter the serial numbers manually, as provided by your local reseller.

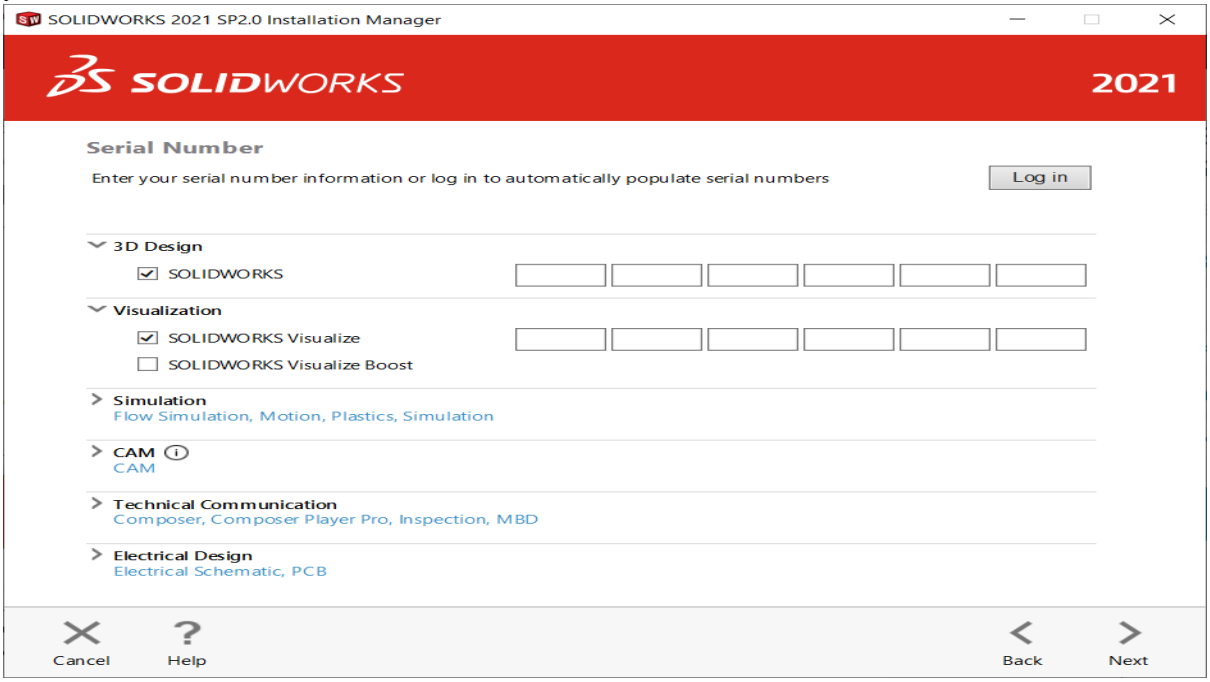

Fig 4.8 Enter Product Key

23. Click "Next" to proceed, and SOLIDWORKS will establish a connection to the online installation server. It will then determine the products you have purchased based on the entered serial number. If an existing version of SOLIDWORKS is detected on your machine, the installation manager will prompt you to choose between upgrading your current version or installing a new separate instance.
24. After completing the previous steps, the final installation phase involves downloading and installing the software onto your machine. You will find the estimated installation file size listed alongside the download file size.
25. Ensure that you check the box indicating your acceptance of the SOLIDWORKS license agreement terms.
26. Once you have checked the acceptance box, select the "Download and Install" option to initiate the installation process.

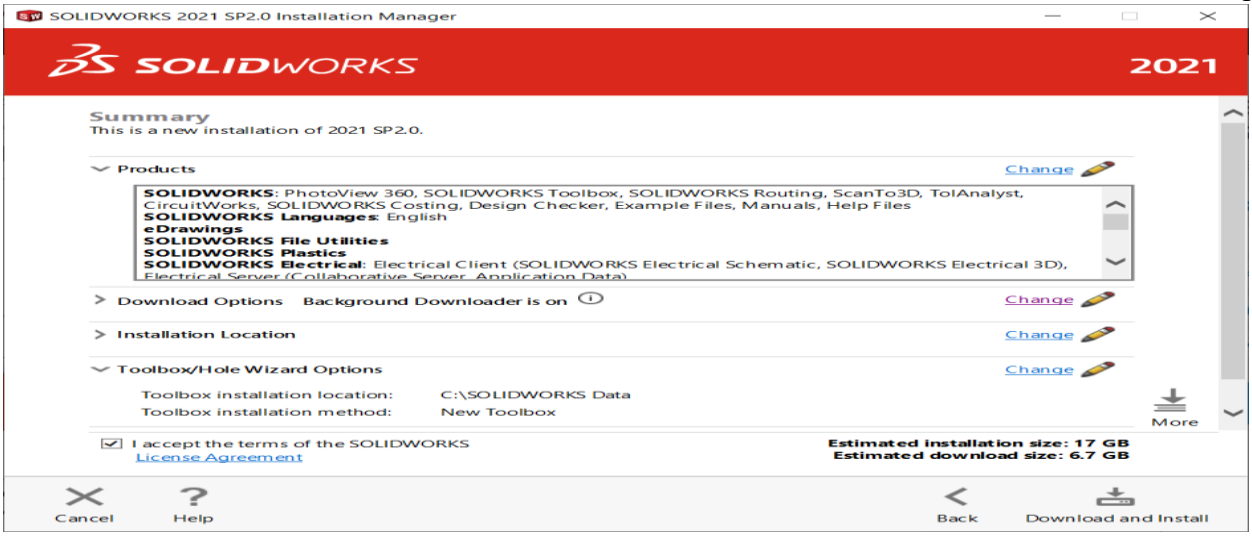

Fig 4.9 Run Installation

27. The SOLIDWORKS Installation Manager will proceed to download the necessary files and subsequently install SOLIDWORKS on your machine The length of this process may fluctuate based on the speed of your internet connection.

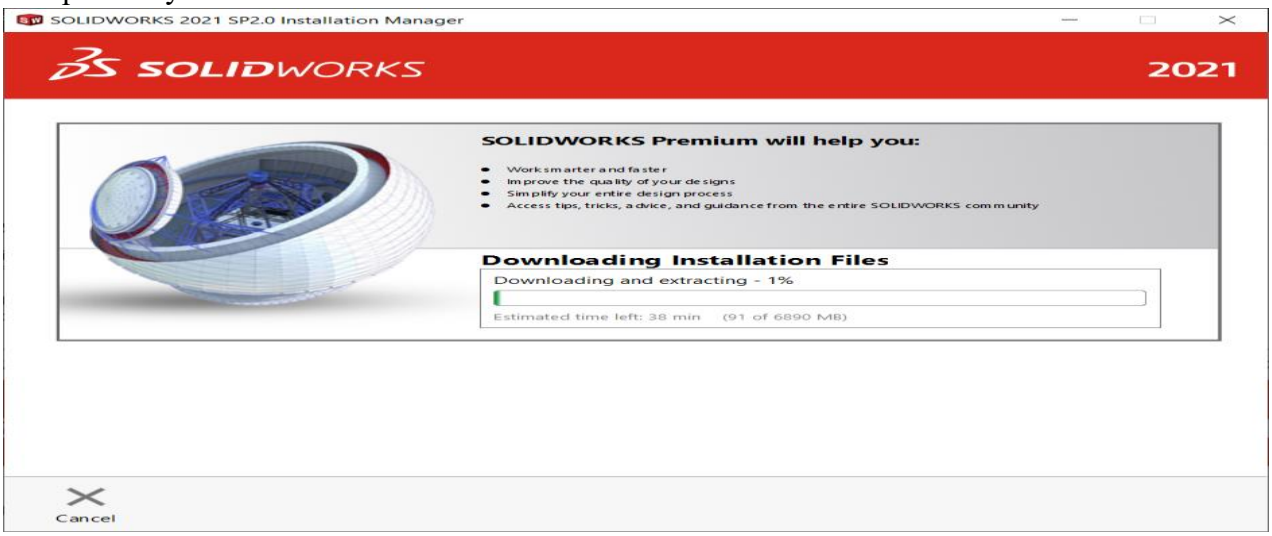

Fig 4.10 Downloading and Extracting

28. Once the installation of SOLIDWORKS is finished, you'll be presented with several options, including the opportunity to join the SOLIDWORKS Customer Experience Improvement Program. Opting to join this program by selecting "Yes, I want to join" can greatly benefit DS SolidWorks Corp. Rest assured that this program is confidential and will not impact your system's performance. To discover more about the SOLIDWORKS Customer Experience, refer to our dedicated article.
    Afterward, simply click on "Finish," and you'll discover the SOLIDWORKS application icons conveniently placed on both your desktop and under the Start menu.

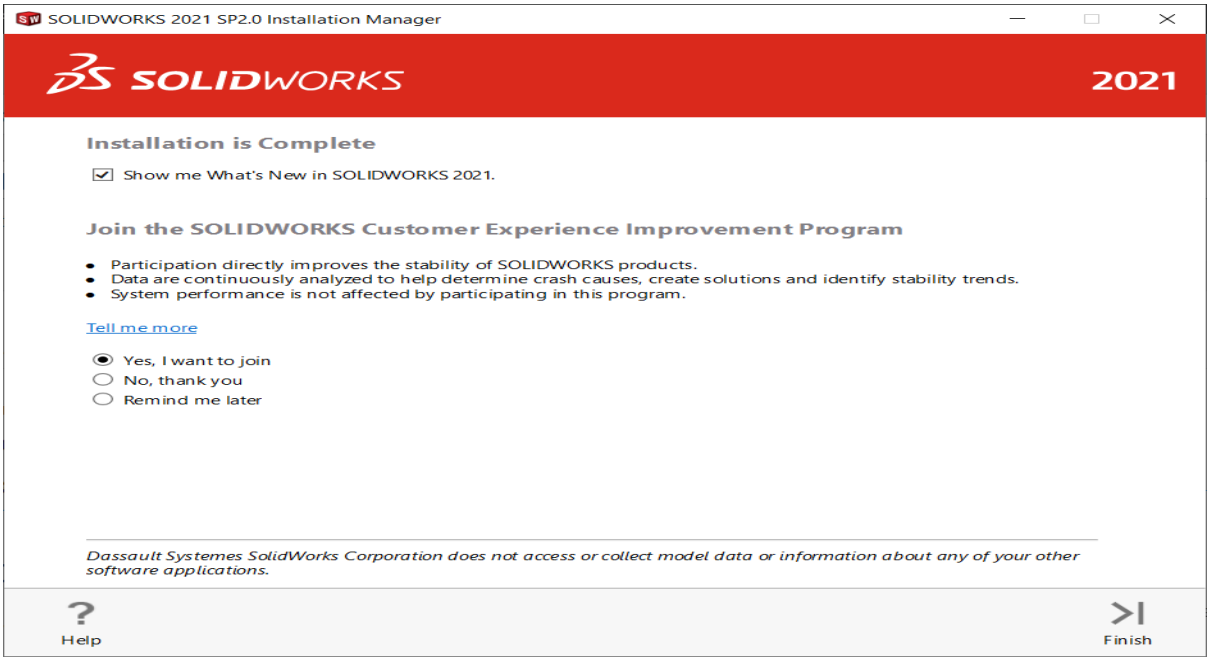

Fig 4.11 Installation Complete

4.1.2 Designing Robotic Arm Components: A SolidWorks Exploration

1. **Base**

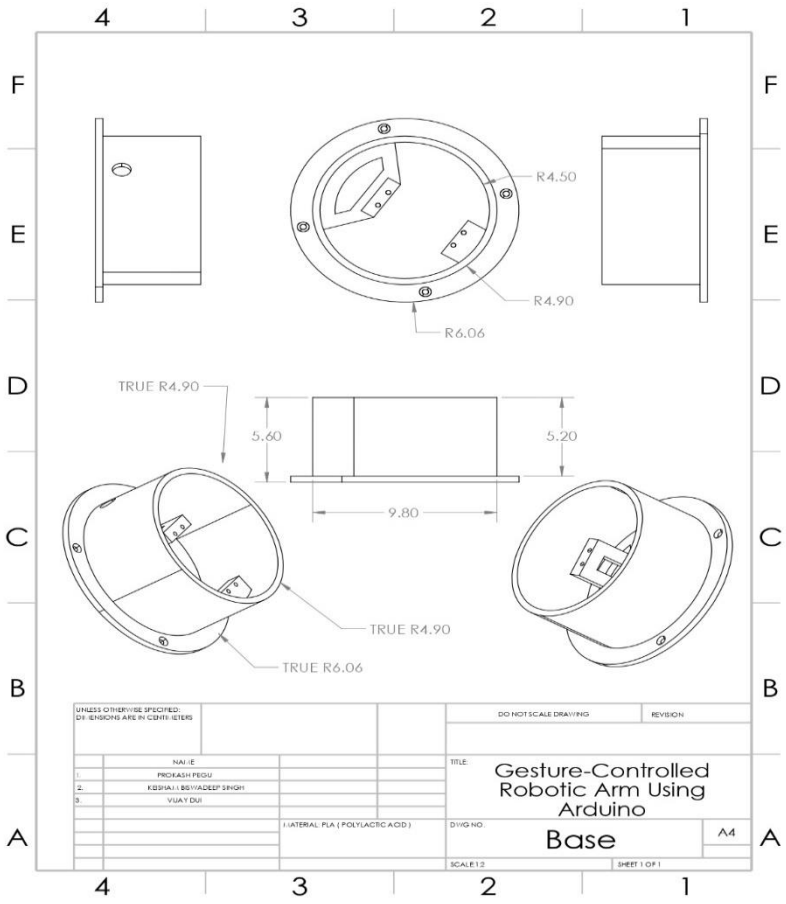

Fig 4.12 Drawing of Base

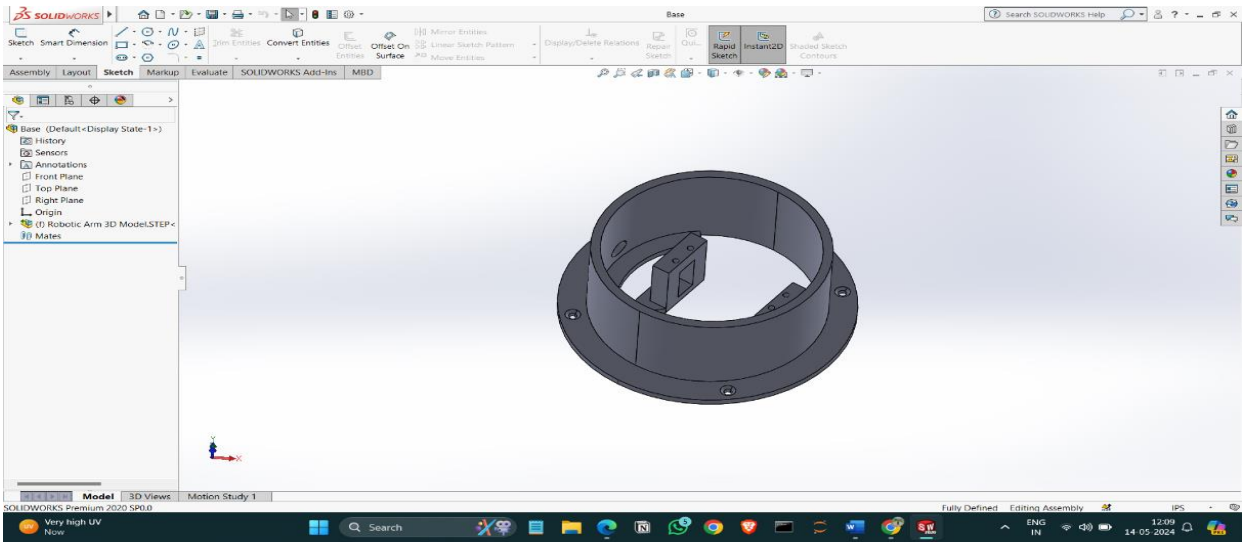

Fig 4.13 SolidWorks Model of Base

2. WAIST

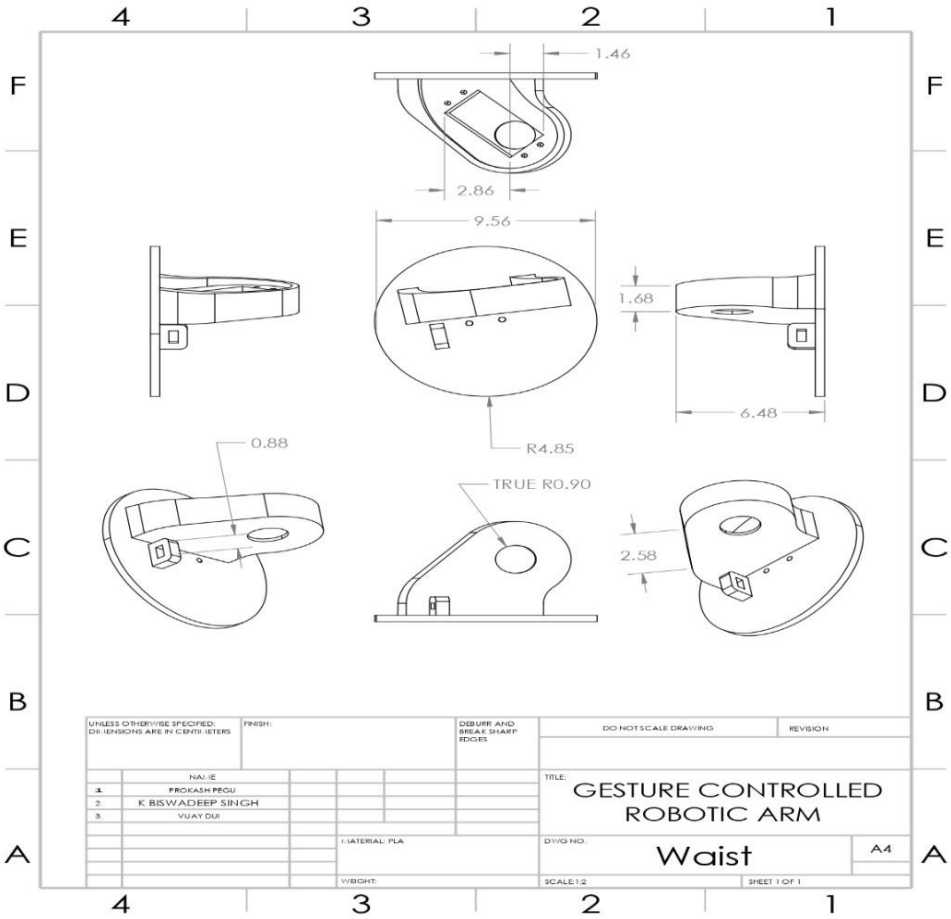

Fig 4.14 Drawing of Waist

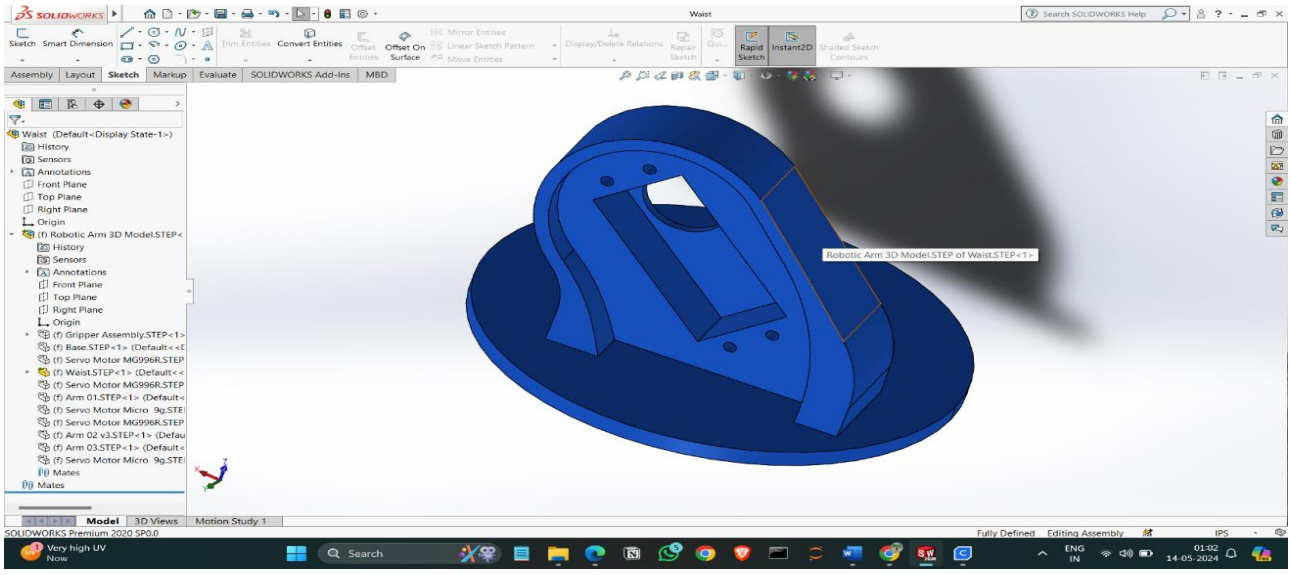

Fig 4.15 SolidWorks Model of Waist

3. Arm 1

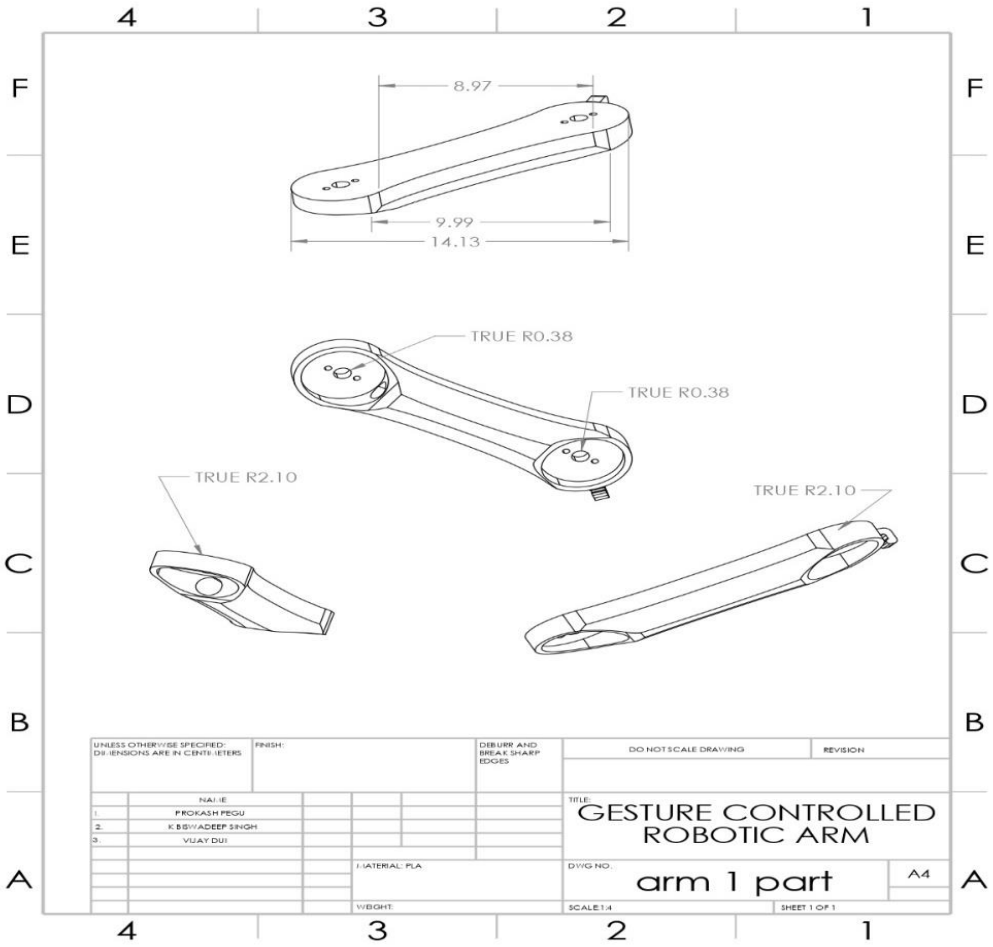

Fig 4.16 Drawing of Arm 1

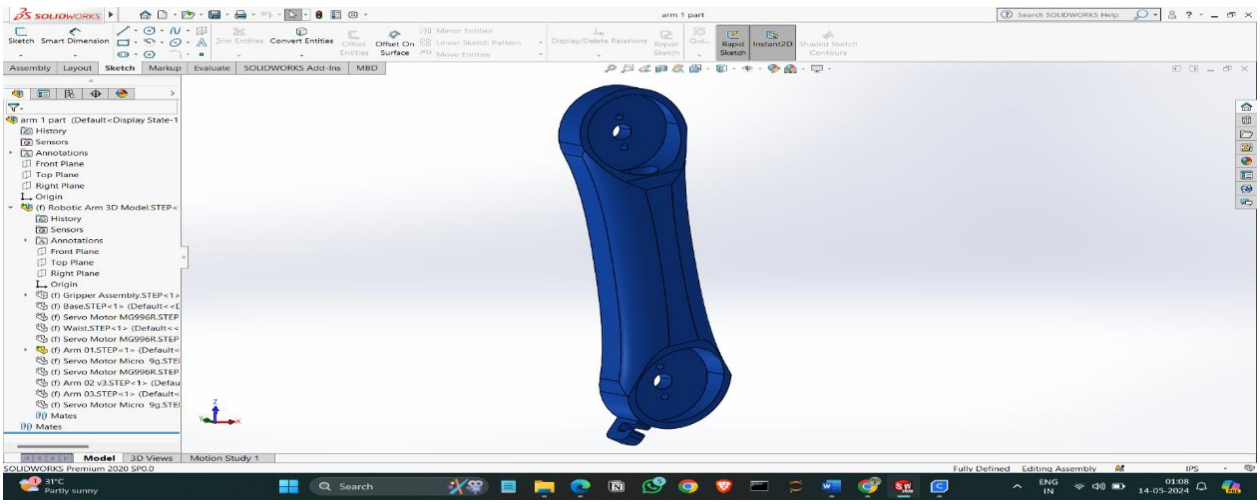

Fig 4.17 SolidWorks Model of Arm

4. Arm 2

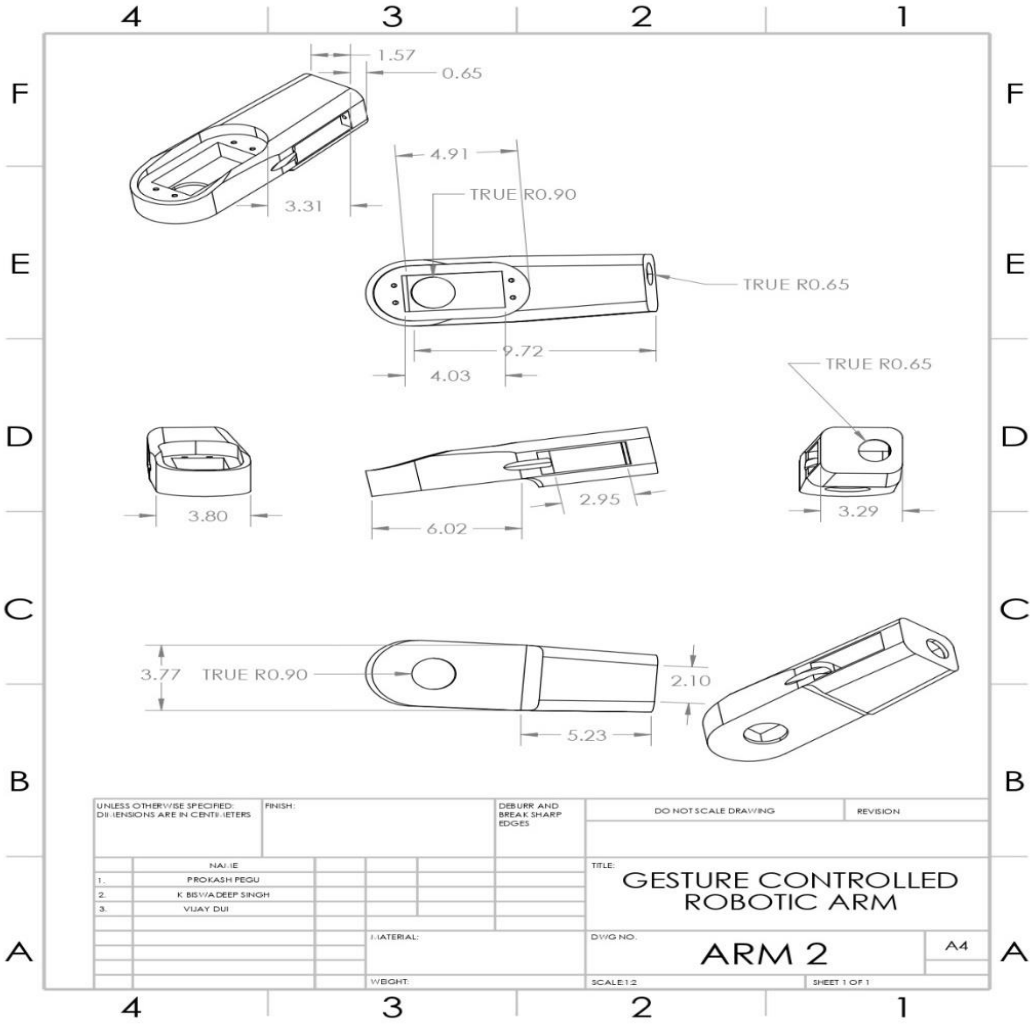

Fig 4.18 Drawing of Arm 2

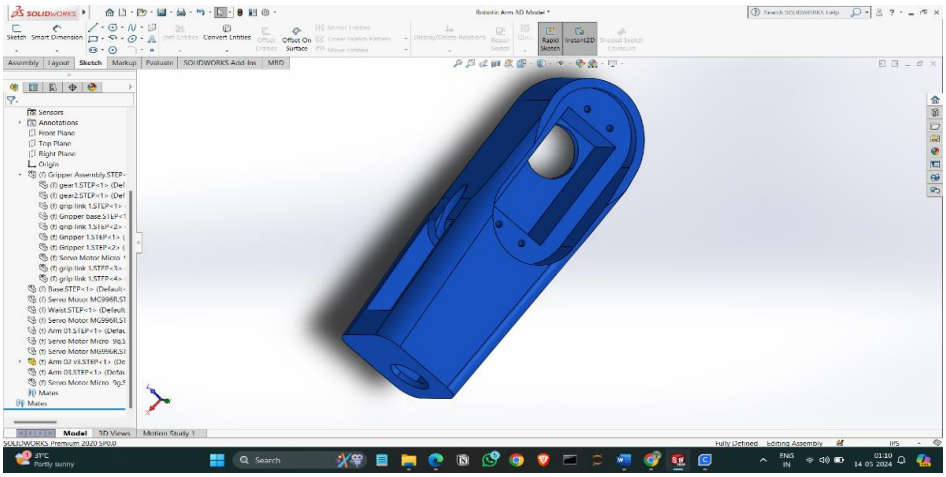

Fig 4.19 SolidWorks Model of Arm

5. Gripper Base

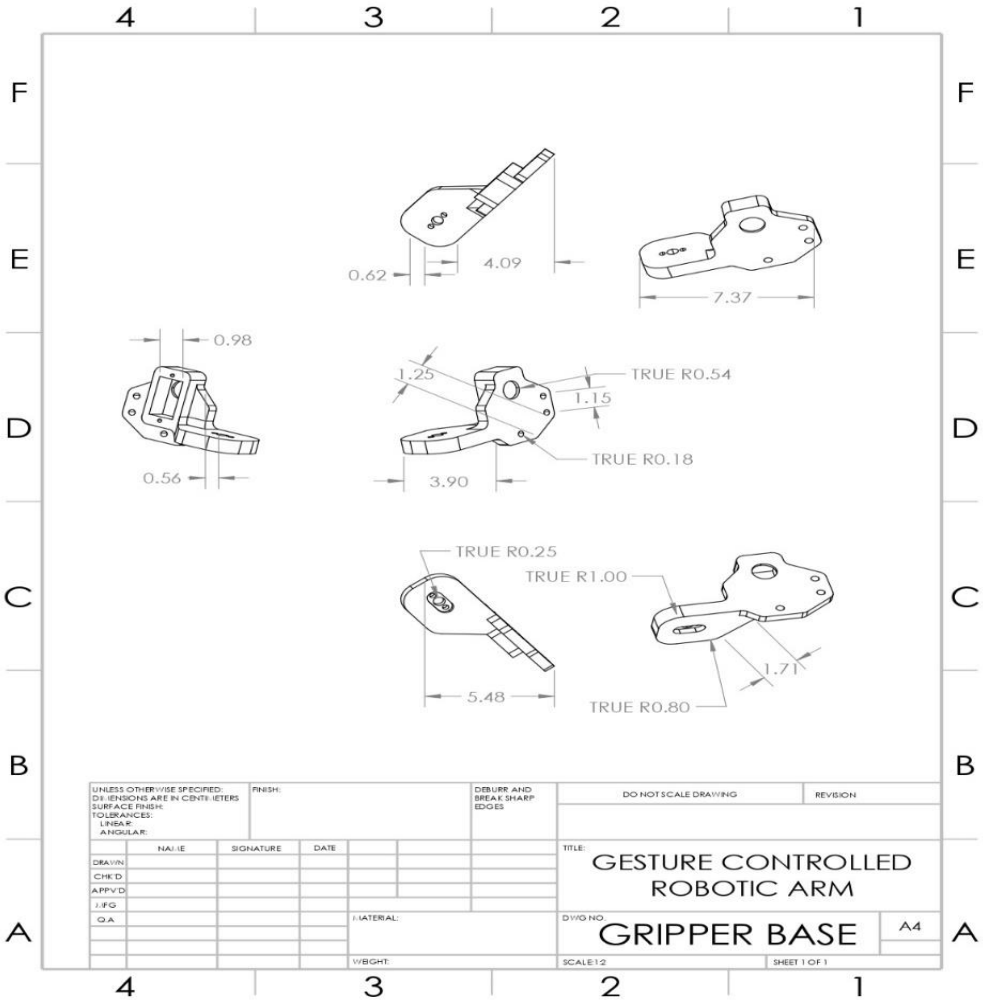

Fig 4.20 Drawing of Gripper Base

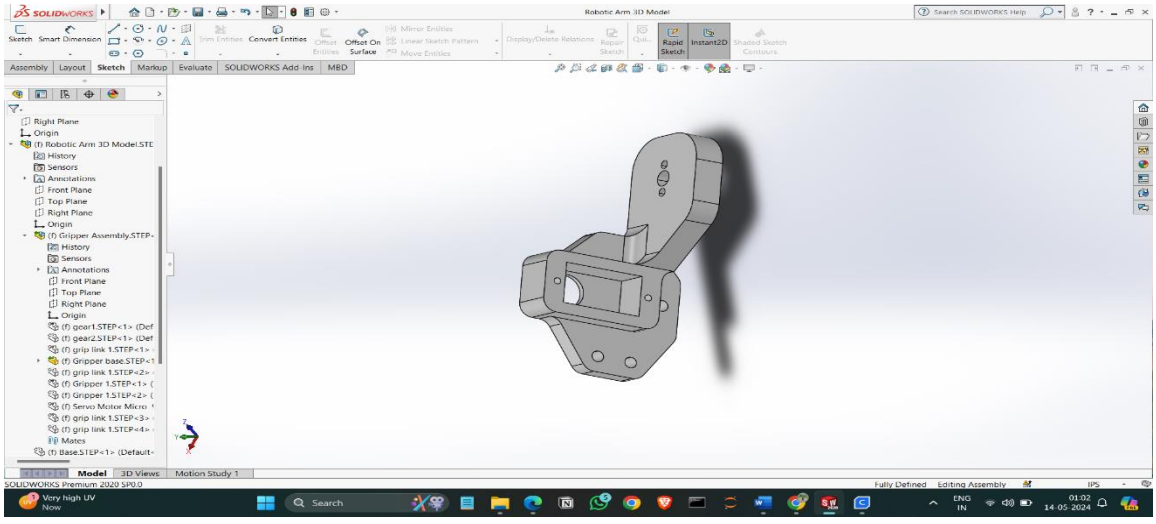

Fig 4.21 SolidWorks Model of Gripper Base

6. GEAR

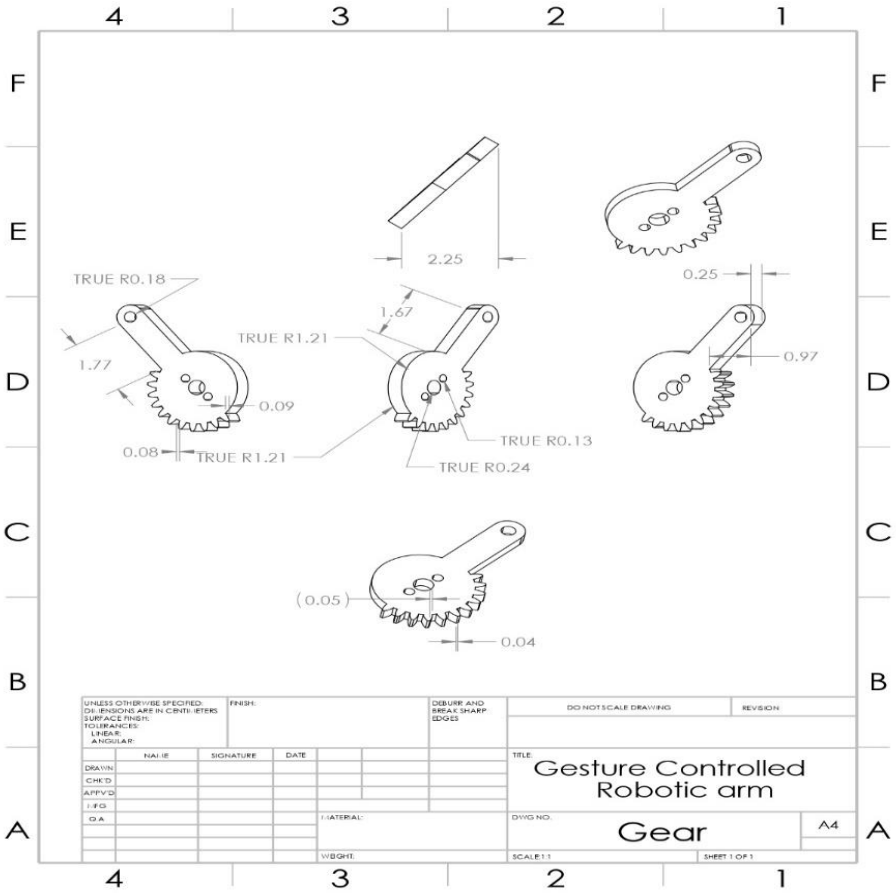

Fig 4.22 Drawing of Gear

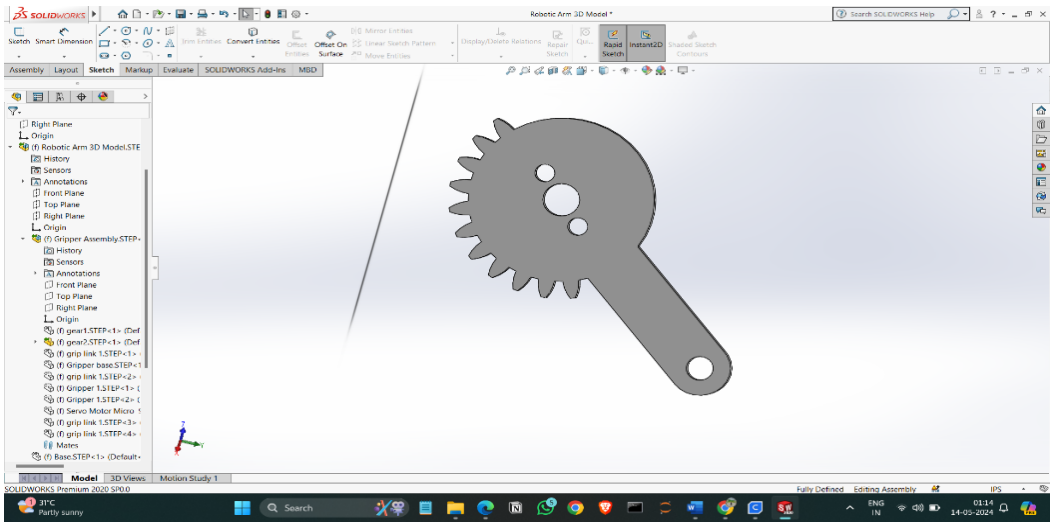

Fig 4.23 SolidWorks Model of Gear

7. Link

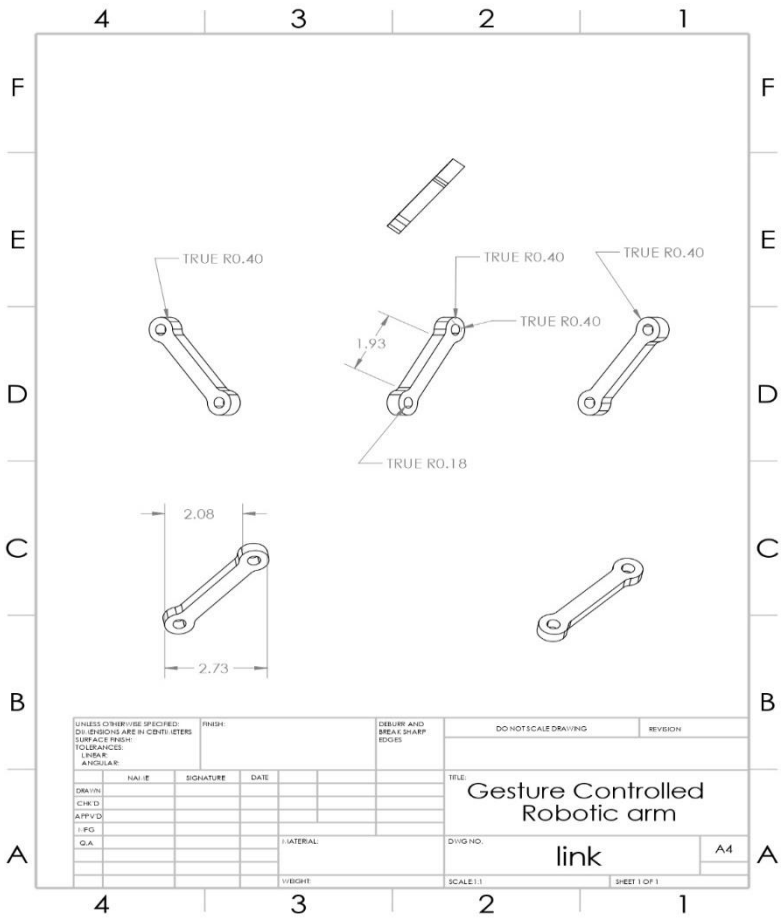

Fig 4.24 Drawing of Link

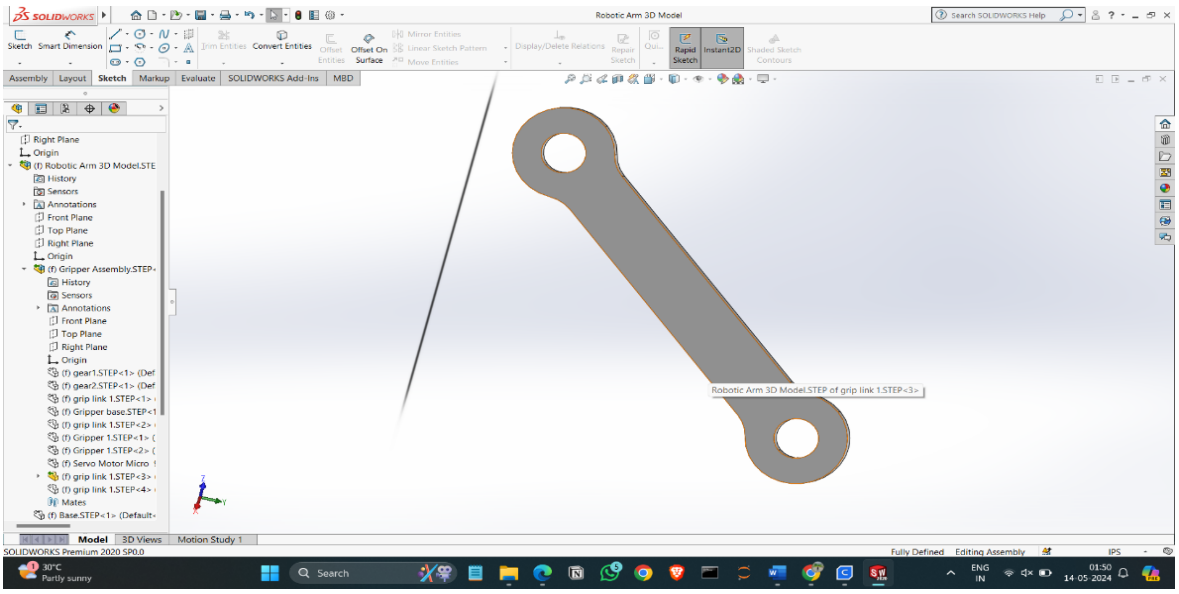

Fig 4.25 SolidWorks Model of Link

8. Gripper

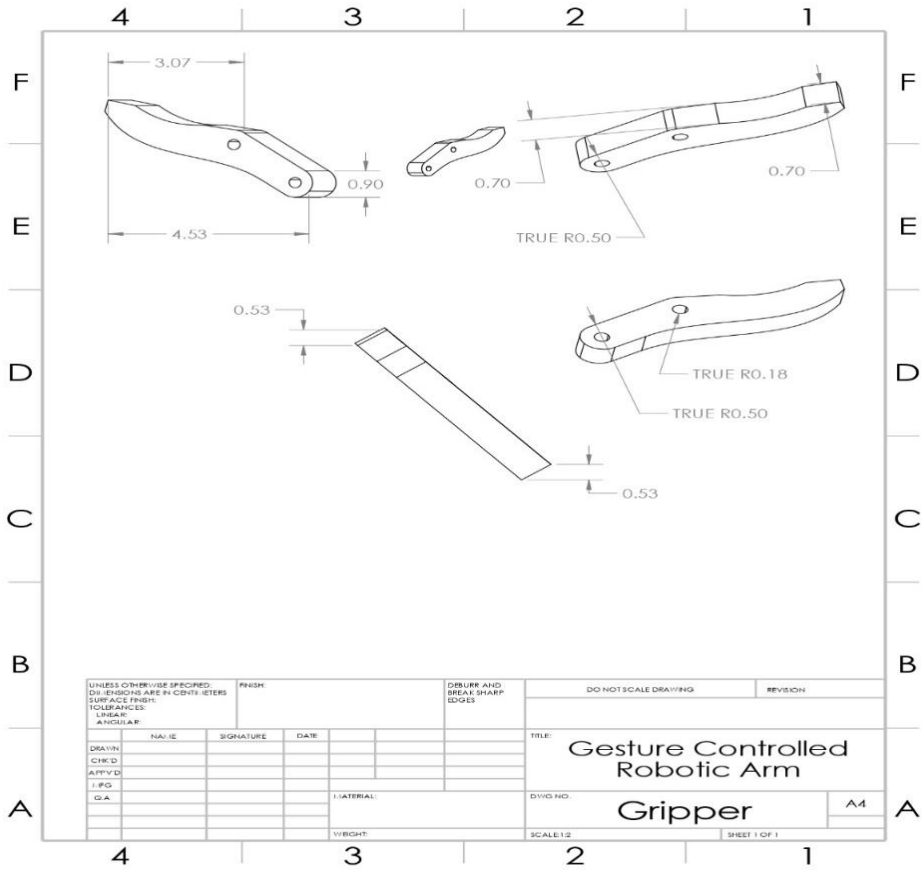

Fig 4.26 Drawing of Gripper

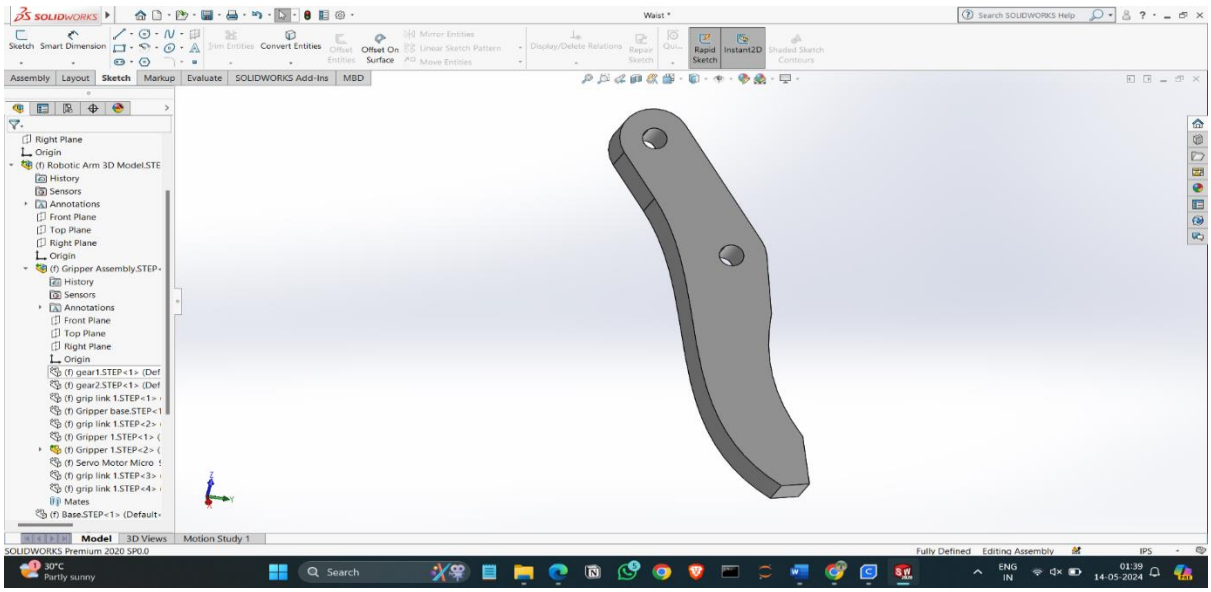

Fig 4.27 SolidWorks Model of Gripper

4.1.3 Assembly of the Robotic Arm Component in SolidWorks

Having examined both the detailed drawings and SolidWorks models of the various components comprising the robotic arm, we are now well-prepared to proceed with the assembly process in SolidWorks. By leveraging these comprehensive visualizations, we can seamlessly integrate each individual part into a cohesive whole, ensuring precision and accuracy throughout the assembly. Once complete, the result will be a fully functional and meticulously designed robotic arm, ready to undertake its intended tasks with efficiency and reliability.

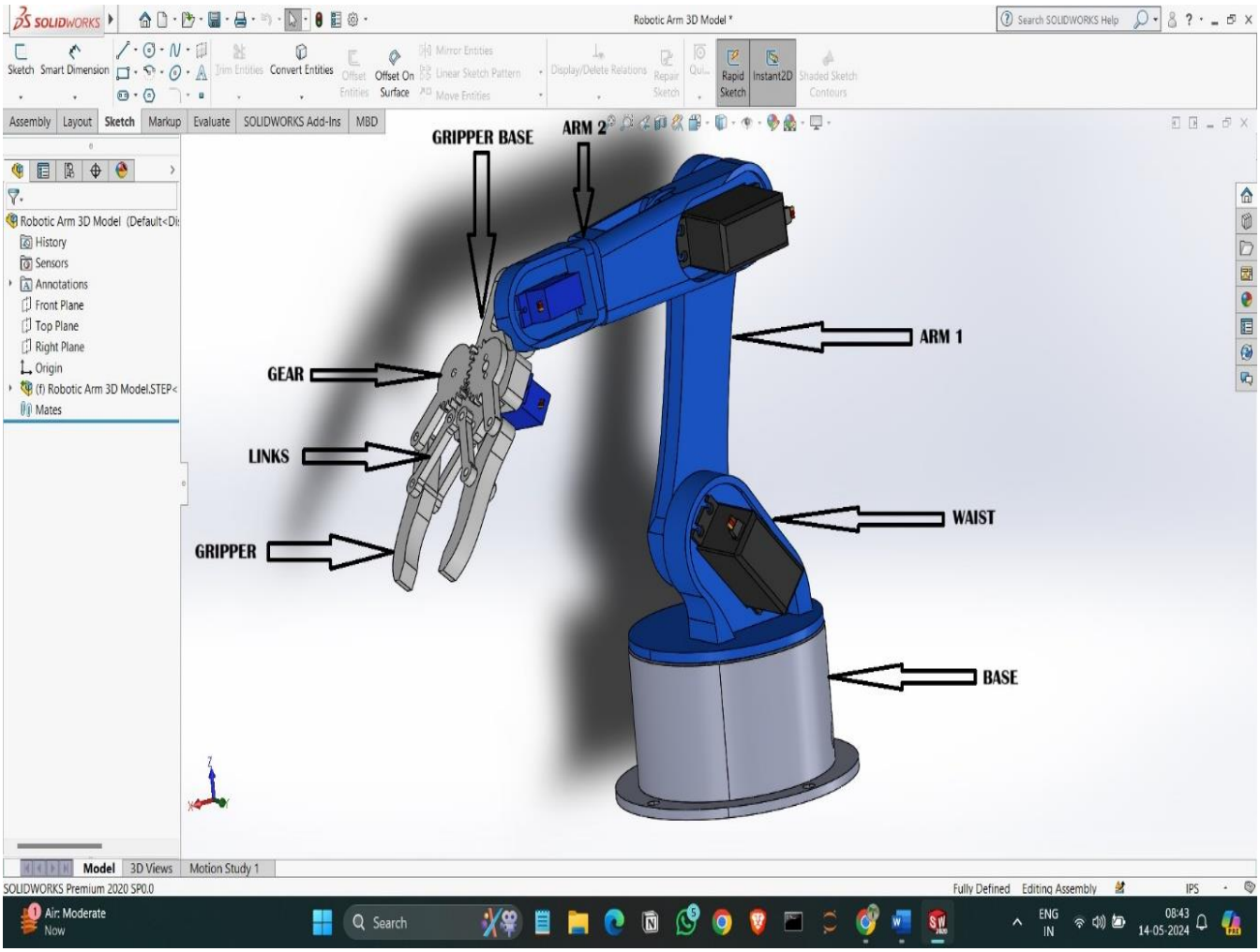

Fig 4.28 Assembled Robotic Arm in SolidWorks.

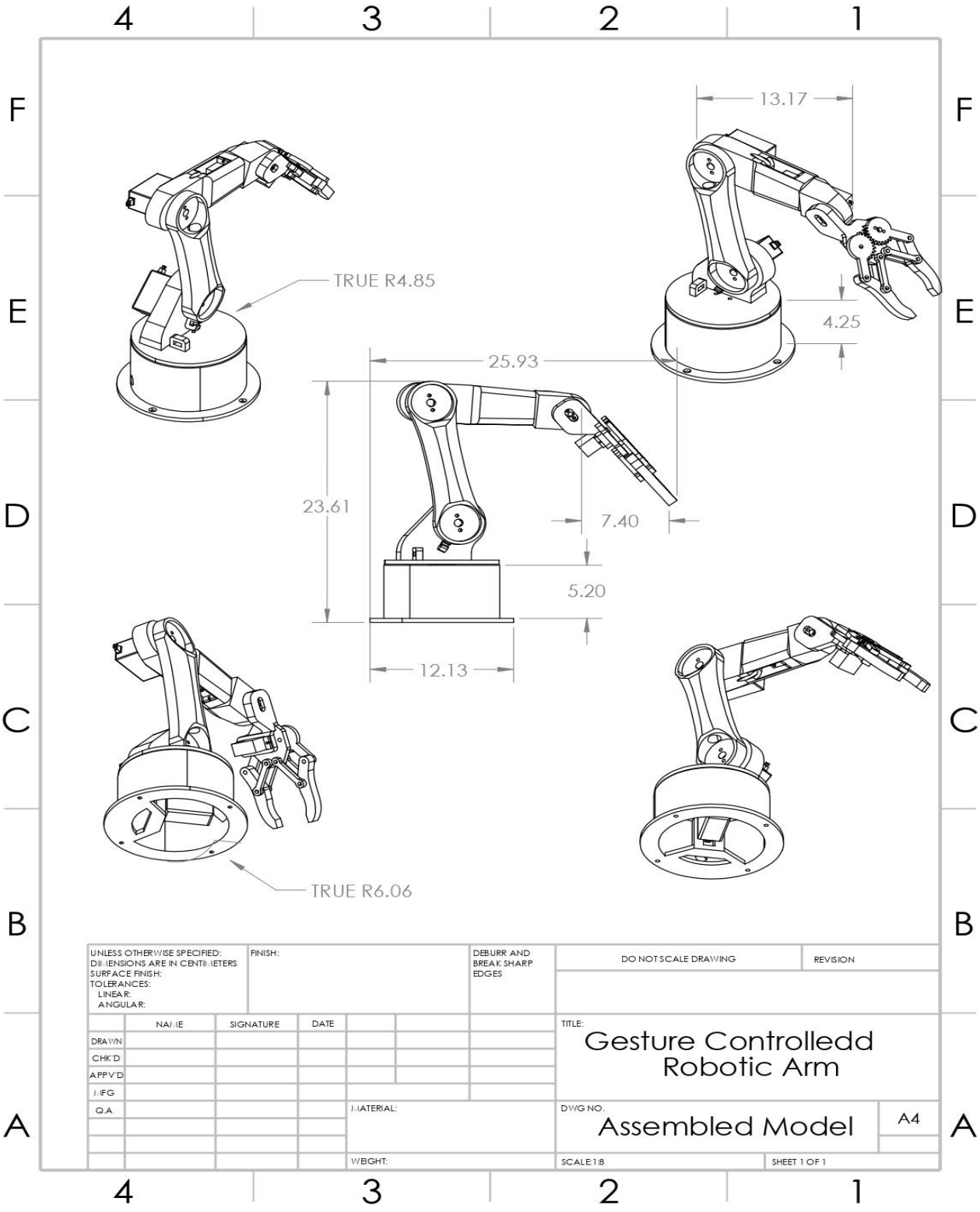

Fig 4.29 Drawing of Assembled Robotic Arm in SolidWork

4.2 UltiMaker Cura

4.2.1 Installation of UltiMaker Cura

Here are step-by-step instructions to install Ultimaker Cura on your PC:

1. **Download Ultimaker Cura:** Visit the official Ultimaker website at https://ultimaker.com/software/ultimaker-cura and navigate to the download section. Choose the appropriate version of Cura for your operating system (Windows) and click on the download button.

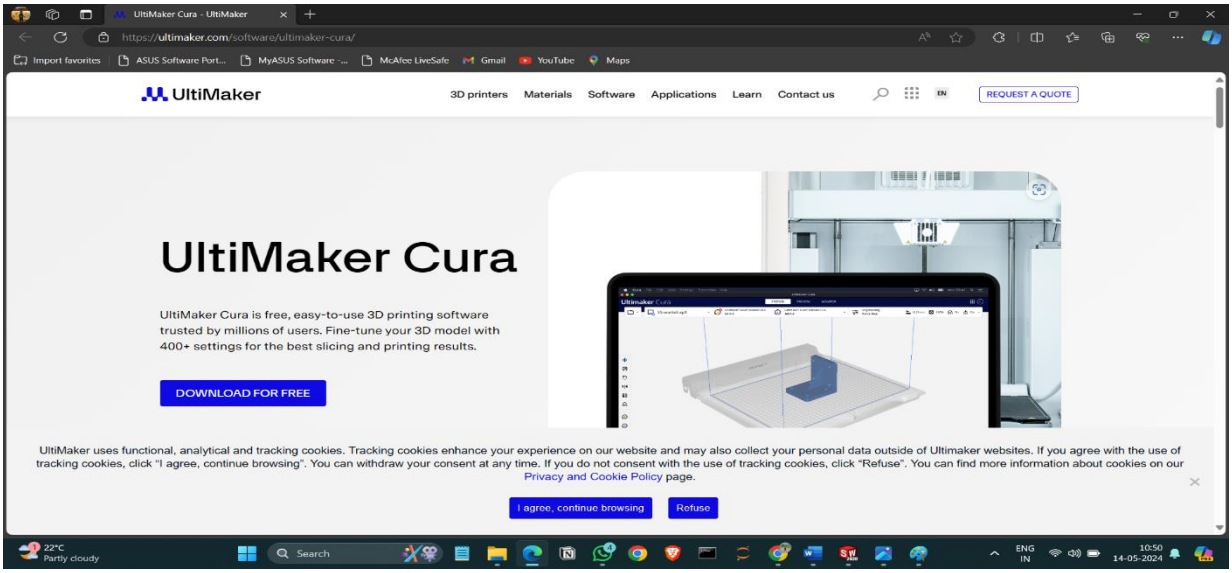

Fig 4.30 UltiMaker Cure Page

2. Run the Installer: Once the download is complete, locate the downloaded file (typically in your Downloads folder) and double-click on it to run the installer.

3.Select Installation Language: The installer will prompt you to select the installation language. Choose your preferred language and click "OK" to proceed.

4. Start Installation: Click on the "Install" button to begin the installation process. You may be prompted by User Account Control (UAC) to confirm the installation, so click "Yes" or enter your administrator password if prompted.

5. Choose Installation Options: Follow the on-screen instructions to proceed with the installation. You may be presented with options such as choosing the installation directory and creating shortcuts. Make any desired selections and click "Next" to continue.

6. Install: Once you have reviewed and confirmed your installation settings, click on the "Install" button to start the installation process. The installer will now copy the necessary files to your computer.

7. Complete Installation: Once the installation is complete, you will see a confirmation message indicating that Ultimaker Cura has been successfully installed on your PC. Click "Finish" to exit the installer.

4.2.2 Setup Ultimaker Cura for Anycubickobra Neo (3D Printer)

Now that Ultimaker Cura is installed on your computer, the next step is to set it up for your 3D printer, the Anycubic Cobra Neo. Setting up Ultimaker Cura for your specific printer involves configuring the printer settings to ensure compatibility and optimal printing results. To begin, launch Ultimaker Cura and navigate to the printer settings section. Here, you will need to add your Anycubic Cobra Neo as a new printer and input its specifications, such as build volume, nozzle diameter, and printing temperature. Refer to the user manual or specifications provided by Anycubic for accurate information. Once the printer is added and configured, you can proceed to customize print settings according to your preferences and project requirements. Adjust parameters such as layer height, infill density, and print speed to achieve the desired print quality. With Ultimaker Cura properly set up for your Anycubic Cobra Neo, you're now ready to import your 3D models, slice them, and start 3D printing with confidence and precision.

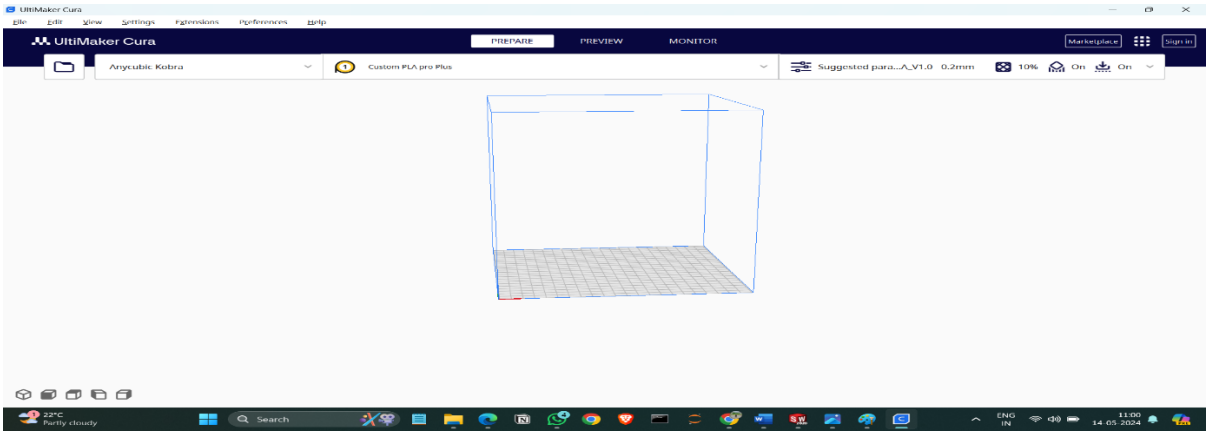

Fig 4.31 Home Page of the UltiMaker Cura

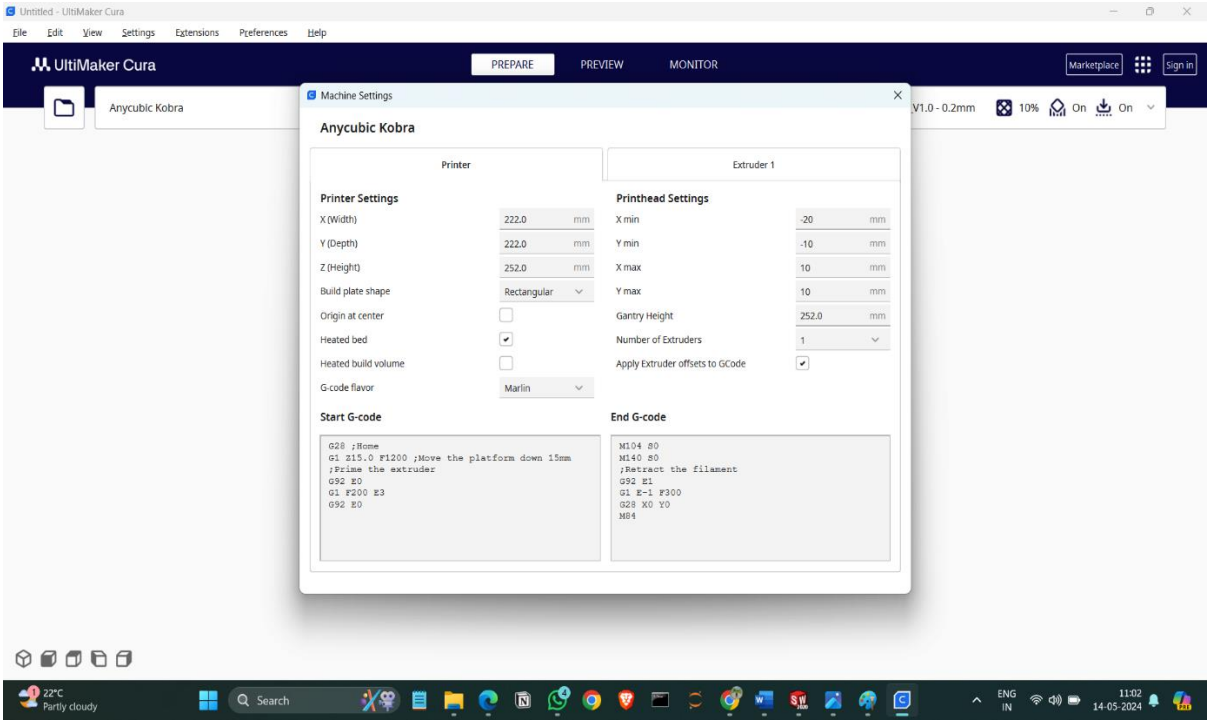

Fig 4.32 Settings for the Anycubic Kora Neo

4.2.3 Slicing using UltiMaker Cura

Before initiating the 3D printing process, it's imperative to slice the components selected from SolidWorks using Ultimaker Cura. Slicing involves breaking down the 3D model into layers and generating the corresponding toolpaths that the 3D printer will follow during printing. In Ultimaker Cura, you'll navigate to the slicing settings where you can fine-tune parameters such as layer height, infill density, print speed, and support structures. These settings dictate the quality, strength, and overall characteristics of the printed parts. By configuring the print settings meticulously, you can optimize the printing process to achieve precise and reliable results. Once the print settings are configured to your satisfaction, you can proceed to slice the components and generate the G-code files necessary for 3D printing. This ensures that the 3D printer will execute the printing job according to your specifications, resulting in high-quality and accurate prototypes or products.

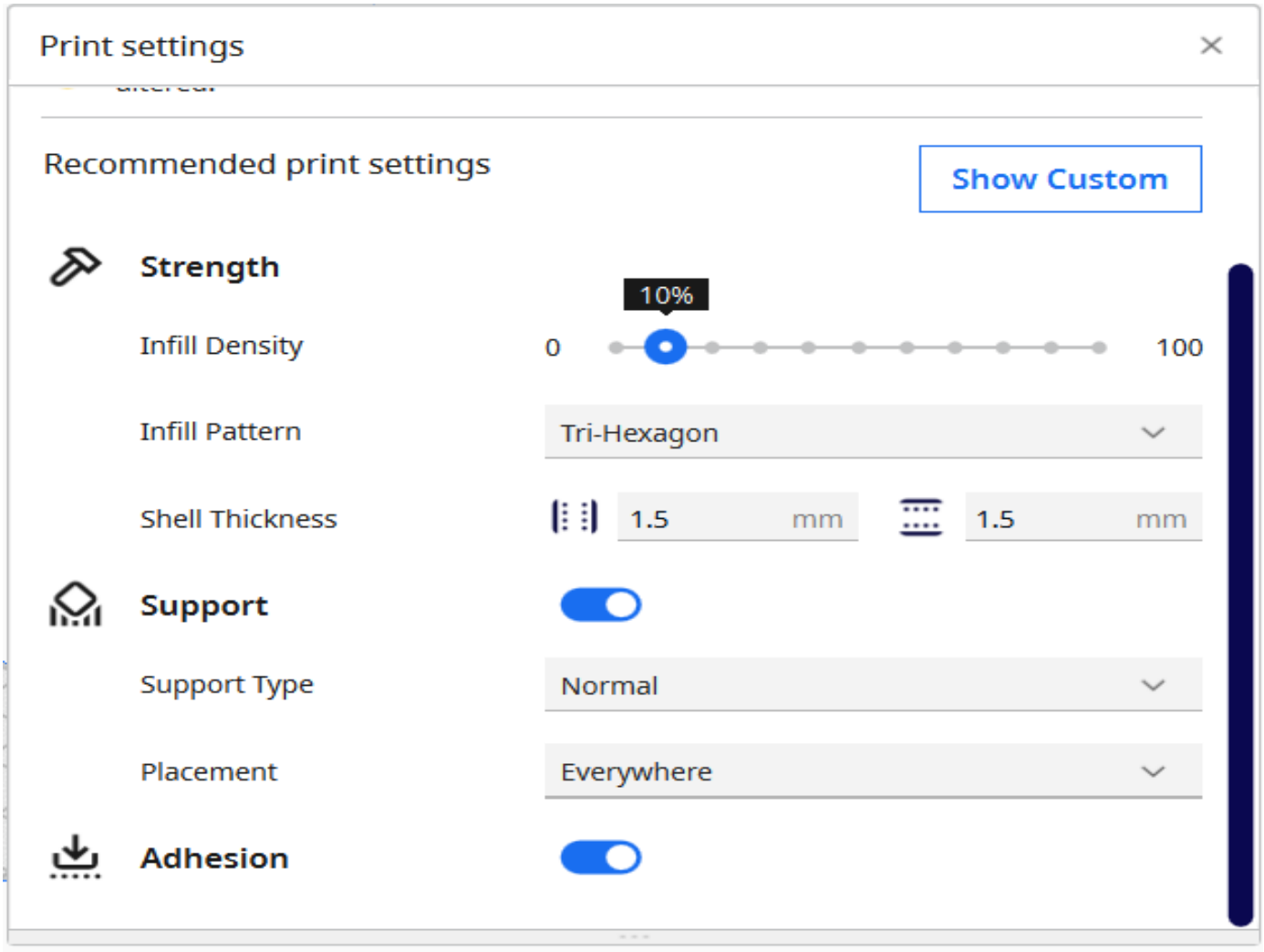

Fig 4.33 Print Settings

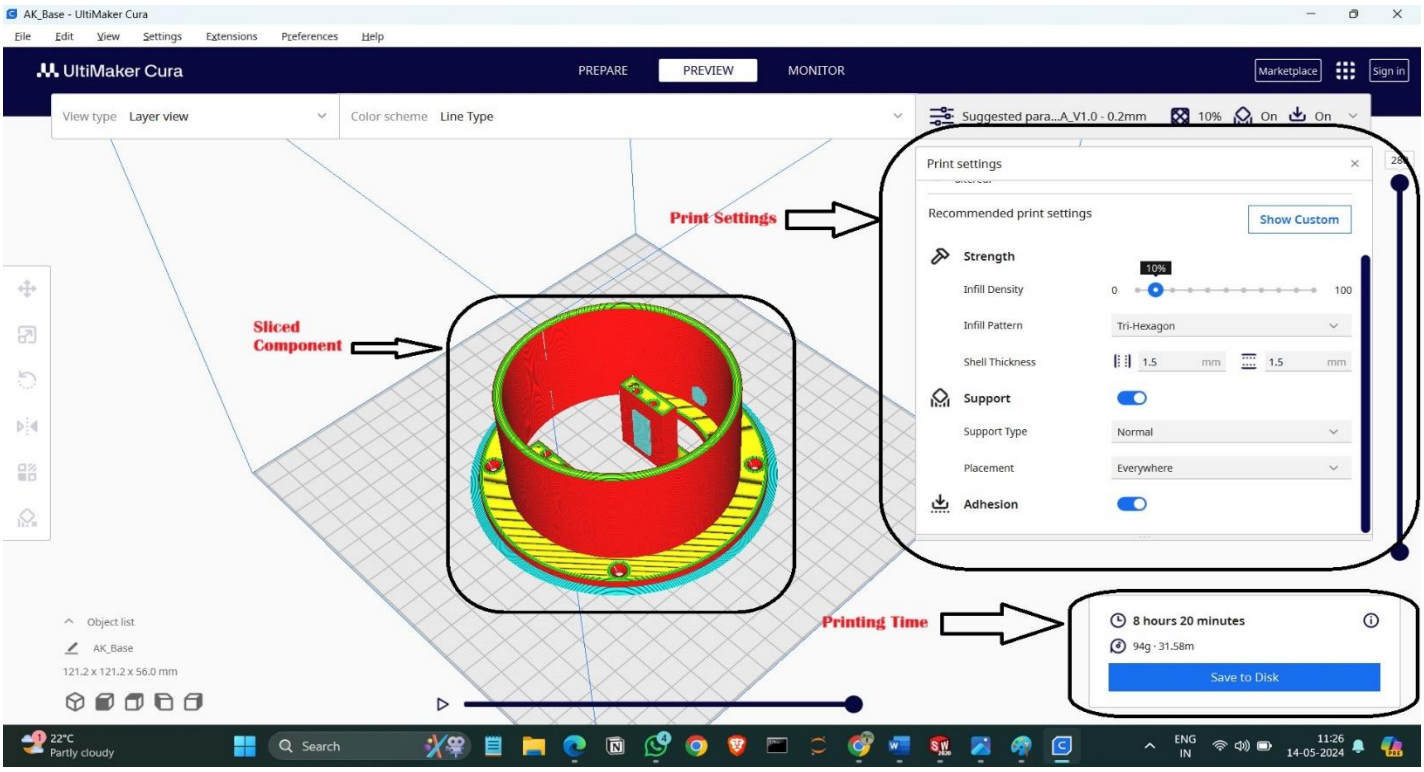

Fig 4.34 Slicing Page of UltiMaker Cura

- After slicing the components in Ultimaker Cura and configuring the print settings to perfection, the next step is to save the sliced model onto a micro-SD card. This micro-SD card serves as the medium for transferring the sliced data to the Anycubic Kobra 3D printer for printing. Once the slicing process is complete, you'll navigate to the option to save the sliced model. Choose the micro-SD card as the destination for saving the sliced file, ensuring compatibility with the Anycubic Kobra 3D printer. By saving the sliced model onto the micro-SD card, you're preparing the file for seamless data transfer and printing. This method facilitates an efficient workflow, allowing you to easily transfer the sliced data from the computer to the 3D printer without the need for a direct connection. With the sliced model safely stored on the micro-SD card, you're ready to initiate the printing process on the Anycubic Kobra 3D printer, bringing your digital designs to life with precision and accuracy.

4.3 AnyCubic Kobra Neo (3D printer)

4.3.1 Setup AnyCubic Kobra Neo

Setting up the Anycubic Kobra Neo printer is a straightforward process that ensures optimal performance and precise printing results. Upon unboxing the printer, the first step is to assemble its components following the manufacturer's instructions carefully. Once assembled, the printer should be placed on a stable surface in a well-ventilated area. Next, connect the printer to a power source and turn it on. After powering up, navigate through the printer's menu to calibrate the Z offset, a crucial parameter that determines the distance between the print bed and the printer's nozzle. The Z offset calibration ensures that the first layer of the print adheres properly to the bed, resulting in accurate and reliable prints. For our specific requirements, the Z offset should be set to 1.58. Adjusting the Z offset to this value ensures optimal adhesion and layer deposition, resulting in high-quality prints. Once the Z offset is calibrated, the Anycubic Kobra Neo printer is ready for use, allowing you to embark on your 3D printing journey with confidence and precision.

Fig 4.35 Setup Screen of AnyCubic Kobra Neo

After completing the setup process for the Anycubic Kobra Neo printer, the next step is to initiate the printing process. This begins by inserting the micro-SD card containing the sliced data of the component into the designated slot on the printer. Once the SD card is securely inserted, navigate through the printer's menu to access the list of available files. Select the file corresponding to the component you

wish to print from the menu. Before proceeding with printing, it's essential to review the print settings and parameters to ensure they align with your requirements and preferences. Once satisfied with the settings, confirm the selection, and the printer will begin the printing process. Throughout the printing process, monitor the printer's progress and make any necessary adjustments as needed to ensure optimal print quality. By following these steps, you can effectively initiate the printing of your desired component with the Anycubic Kobra Neo printer.

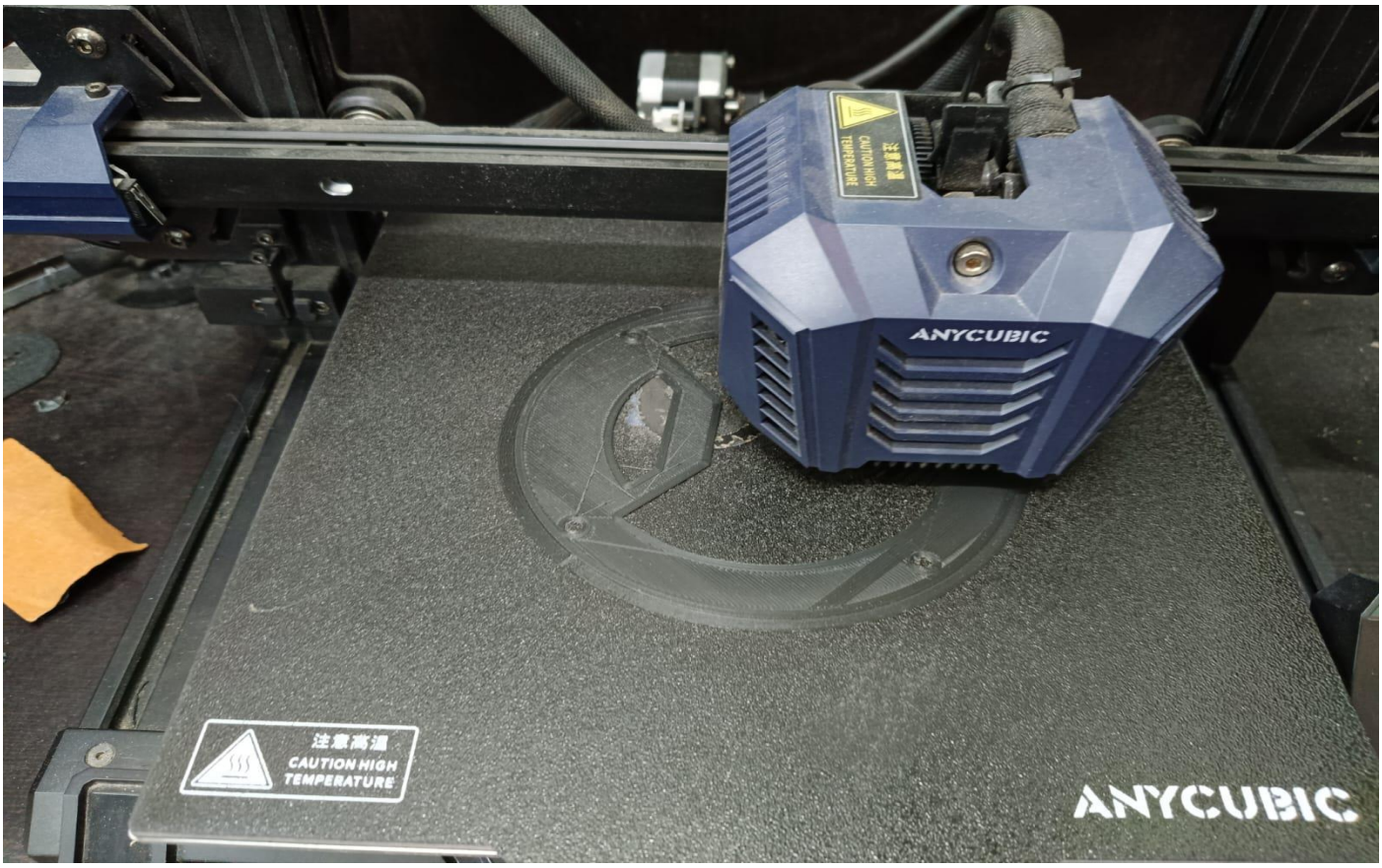

Fig 4.36 Ongoing Printing of Base Component

4.3.2 Total print time for all the components

- Below is the list of parts and their printing time:

Part Name		File size	Print time	Quantity	Printed
Gripper Base		611 kb	2 h 20 min	1	Yes
Waist		749 kb	4h 30 min	1	Yes

Gripper		175 kb	1 hr	2	Yes
Grip Link		205 kb	20 min	4	Yes
Gear		394 kb	50 min	2	Yes
Arm_1		1574 kb	5 h	1	Yes
Arm_2		985 kb	4 h	1	Yes
Base		896 kb	8h 55 min	1	Yes
Total Time Taken			**1 Day 2 Hr 55 min**		

4.3.3 Results

- **Base**

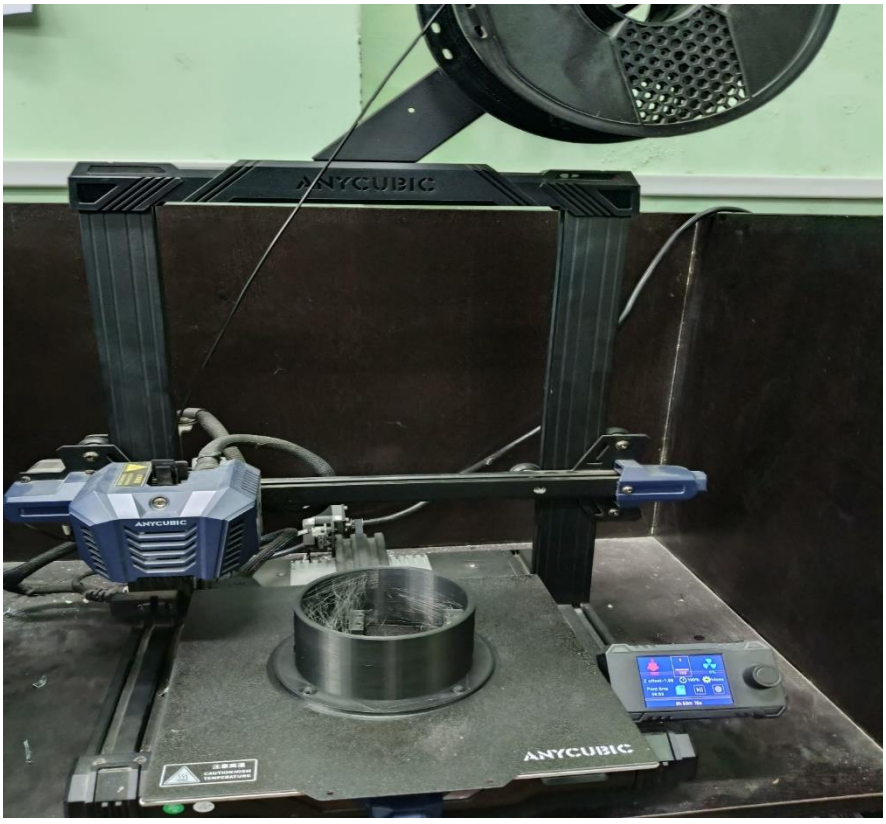

Fig 4.37 Base

- **Waist**

Fig 4.38 Waist

- **Arm 1**

Fig 4.39 Arm 1

- **Arm 2**

Fig 4.40 Arm 2

- **Gripper Base**

Fig 4.41 Gripper Base

- **Gears**

Fig 4.42 Gears

- **Links**

Fig 4.43 Links

- **Gripper**

Fig 4.44 Gripper

4.4 Assembly

After meticulously printing the components of the robotic arm using a 3D printer, we embarked on the assembly process, employing servo motors, screws, nuts, and bolts to bring the design to life. With precision and attention to detail, each piece was carefully fitted together, ensuring seamless functionality and structural integrity. The culmination of our efforts resulted in the successful assembly of the robotic arm, a testament to our dedication to craftsmanship and innovation. Now, with the completed arm at our disposal, we stand ready to explore its capabilities and potential applications in various domains[20-23].

Here is the Result: -

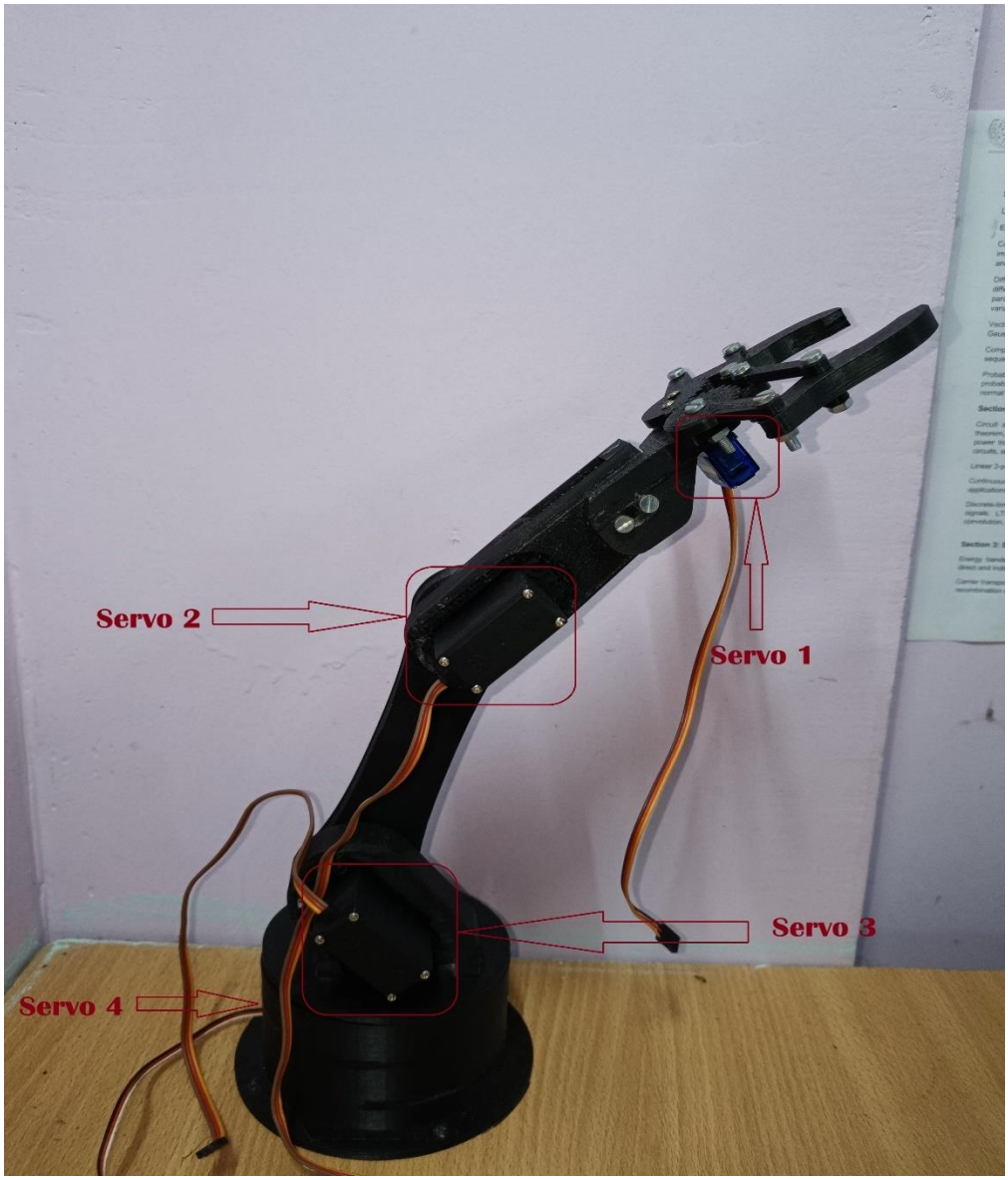

Fig 4.45 Side View of the Robotic Arm

Fig 4.46 Top View of Robotic Arm

4.5 Circuit Diagram

With the assembly of all the components of the robotic arm complete, the next step involves connecting the servo motors to the Arduino board and the power supply. Given the high-power requirements of the servo motors, it becomes evident that the Arduino alone cannot adequately supply the necessary power. Therefore, a dedicated power supply unit capable of meeting the demands of the servo motors is employed. By establishing this robust electrical connection, we ensure that the servo motors receive sufficient power to operate effectively, enabling precise control and smooth movement of the robotic arm [24-27].

- **Here is the Circuit Diagram of the Arduino and the Servo Motors**

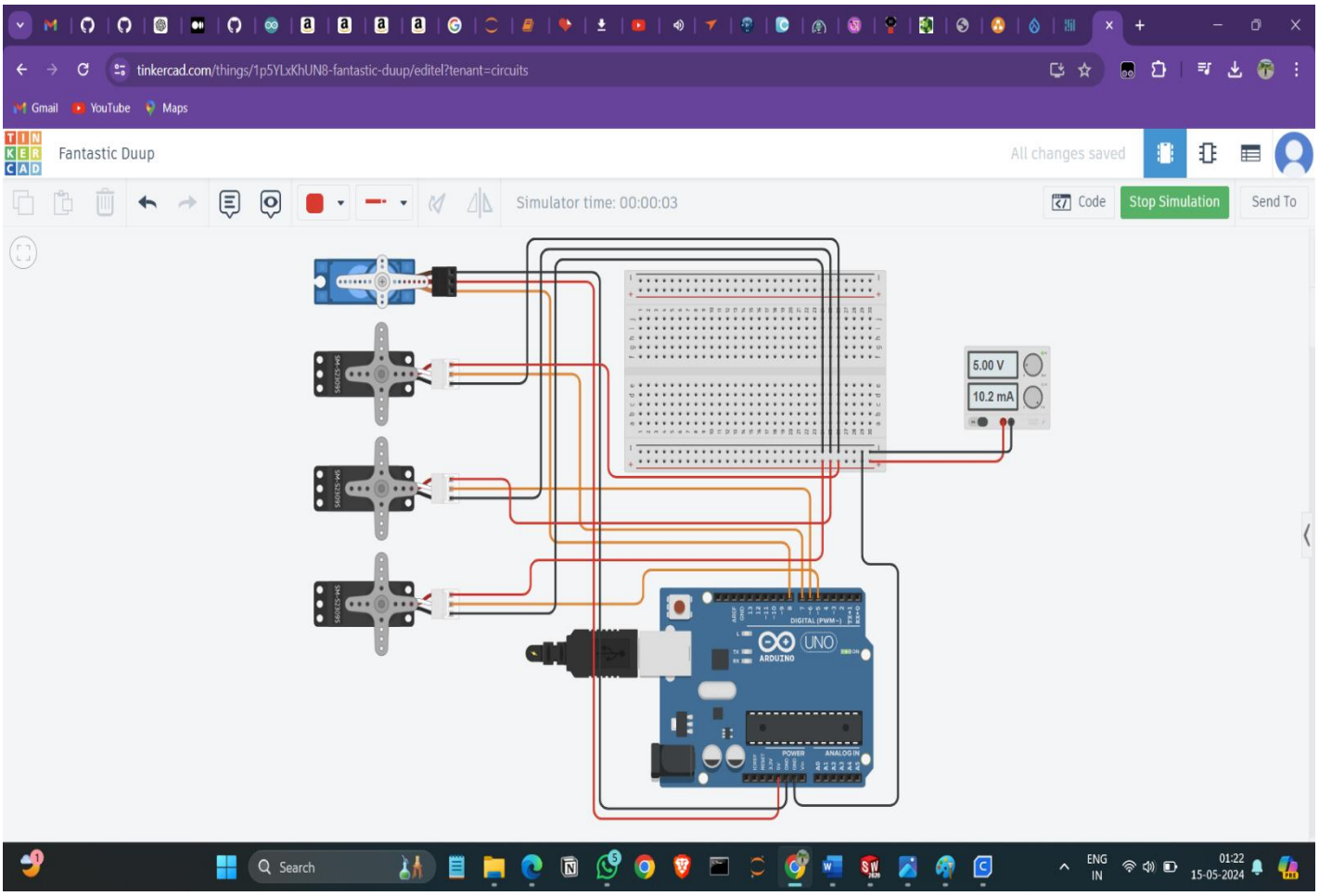

Fig 4.47 Circuit Diagram

4.6 Software Program for the Robotic arm

4.6.1 Flowchart of the complete Gesture Controlled Robotic Arm using Arduino

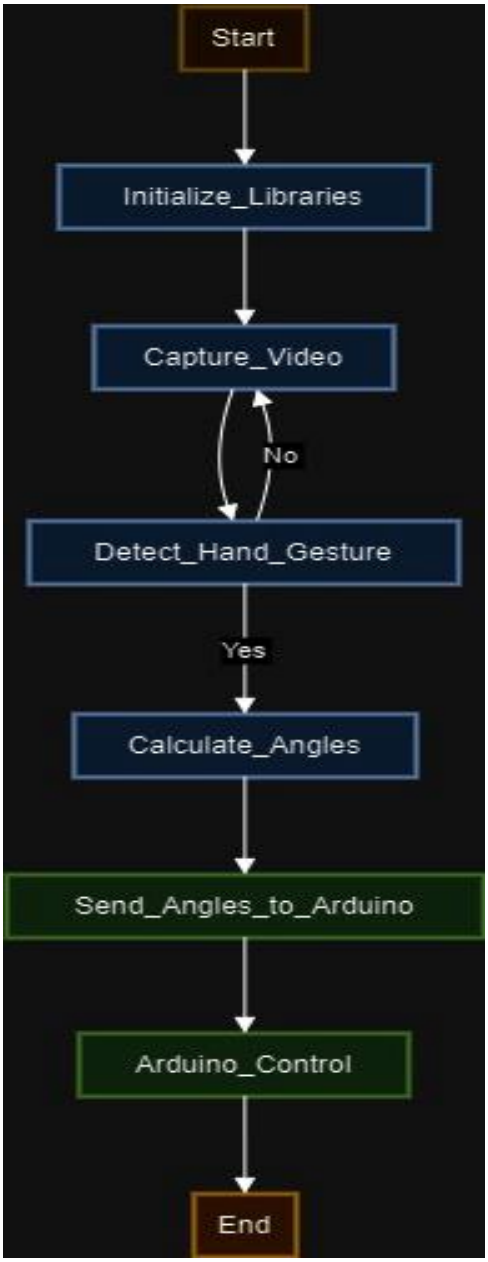

Fig 4.48 Flowchart

Here's a detailed explanation of each step in the flowchart:

1. **Start:** The beginning of the process.

2. **Initialize variables and libraries (Python, Mediapipe, serial):** In this step, we set up the necessary variables and import the required libraries in Python. This includes initializing the Mediapipe library for hand gesture detection and the serial library for communication with the Arduino.

3. **Capture video stream from webcam:** This step involves accessing the webcam feed to capture video frames in real-time. The webcam feed will be used for hand gesture detection.

4. **Process video frames to detect hand gestures using Mediapipe:** Each frame captured from the webcam is processed using the Mediapipe library to detect hand gestures. This includes identifying landmarks on the hand, such as the position of fingers and palm.

5. **Calculate servo motor angles based on hand gesture:** Once hand gestures are detected, the program calculates the corresponding angles for the servo motors based on the detected hand positions. For example, closing fingers may correspond to different servo motor positions compared to an open hand gesture.

6. **Send servo motor angles to Arduino via serial communication:** After calculating the servo motor angles, the Python program sends this data to the Arduino board using serial communication. The Arduino board is connected to the PC via USB and is responsible for controlling the servo motors of the robotic arm.

7. **Arduino reads servo motor angles and controls robotic arm:** Upon receiving the servo motor angles from the Python program, the Arduino board reads this data and translates it into commands to control the servo motors of the robotic arm. The robotic arm moves accordingly based on the received instructions.

8. **End:** The end of the process.

Decision Points:

- Is hand gesture detected? This decision point checks if a hand gesture is detected in the video frame captured from the webcam.

- **Yes:** If a hand gesture is detected, the program proceeds to calculate servo motor angles.

- **No:** If no hand gesture is detected, the program continues capturing video frames to detect hand gestures.

This flowchart outlines the sequence of steps involved in the project, from capturing hand gestures using a webcam to controlling a robotic arm based on the detected gestures.

4.6.2Include the OpenCV library in Jupyter Notebook

OpenCV (Open-Source Computer Vision Library) is a popular open-source computer vision and machine learning software library. It offers various functions and algorithms to perform tasks such as image processing, object detection, face recognition, and more. OpenCV supports multiple programming languages, including Python, C++, and Java, making it widely used in both academia and industry.

Some key features of OpenCV include:

1. Image Processing: OpenCV provides a wide range of functions for basic and advanced image processing tasks, such as resizing, cropping, filtering, and morphological operations.

2. Object Detection and Tracking: OpenCV includes pre-trained models and algorithms for object detection, recognition, and tracking, allowing developers to build applications for tasks like face detection, pedestrian detection, and object tracking.

3. Machine Learning: OpenCV integrates with popular machine learning libraries like TensorFlow and PyTorch, enabling developers to train custom models for various computer vision tasks.

4. Camera Calibration and 3D Reconstruction: OpenCV offers tools for camera calibration, stereo vision, and 3D reconstruction, making it suitable for applications in augmented reality, robotics, and 3D modeling.

5. Cross-Platform and Real-Time Processing: OpenCV is cross-platform and supports various operating systems like Windows, Linux, macOS, Android, and iOS. It also provides real-time processing capabilities, making it suitable for applications that require low-latency image processing.

Overall, OpenCV is a versatile and powerful library that simplifies the development of computer vision applications by providing a comprehensive set of tools and algorithms.

1. **Install OpenCV**: If you haven't installed OpenCV yet, you can install it using pip. Open a terminal or command prompt and run the following command:
   ```
   pip install opencv-python
   ```

2. **Install Jupyter Notebook:** If you haven't installed Jupyter Notebook, you can install it using pip as well:
   ```
   pip install jupyterlab
   ```

3. **Launch Jupyter Notebook**: Open a terminal or command prompt and navigate to the directory where you want to work with Jupyter Notebook. Then, run the following command:
   ```
   jupyter notebook
   ```

4. **Create a new notebook or open an existing one:** In the Jupyter Notebook interface that opens in your web browser, you can create a new notebook by clicking on the "New" button and selecting "Python 3" under "Notebooks". Alternatively, you can open an existing notebook.
5. **Import OpenCV:** In a code cell in your Jupyter Notebook, you can import OpenCV using the import statement:

```
import cv2
```

6. **Verify the installation:** To verify that OpenCV is installed correctly and working in your Jupyter Notebook, you can write and execute some code that uses OpenCV functions. For example, you can read and display an image from your local filesystem:

```
import cv2
import matplotlib.pyplot as plt

# Read an image from the filesystem
image = cv2.imread('path_to_image.jpg')

# Display the image using matplotlib
plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
plt.axis('off')
plt.show()
```

These steps should help you include and use the OpenCV library in your Jupyter Notebook environment. Make sure to replace 'path_to_image.jpg' with the actual path to an image file on your filesystem.

4.6.3Include the Serial Library in Jupyter Notebook

Serial libraries provide functionality for serial communication between devices, typically over UART (Universal Asynchronous Receiver-Transmitter) or RS-232 connections. These libraries facilitate data transmission and reception between microcontrollers, sensors, actuators, and other hardware components.

Key features of serial libraries include:

1. Serial Port Configuration: They allow configuration of serial port parameters such as baud rate, data bits, stop bits, and parity settings to match the requirements of the connected devices.

2. Data Transmission and Reception: Serial libraries provide functions or methods to send and receive data over the serial port. They handle the low-level communication protocols to ensure reliable data transfer.

3. Buffering: Many serial libraries implement buffering mechanisms to store incoming and outgoing data temporarily. This helps prevent data loss and ensures smooth communication even in cases of occasional delays.

4. Error Handling: Serial libraries often include error detection and correction mechanisms to handle transmission errors, such as parity errors, framing errors, or buffer overflows.

5. Platform Compatibility: Serial libraries are available for various programming languages and platforms, including C/C++, Python, Java, and others. They are commonly used in embedded systems, IoT (Internet of Things) devices, robotics, and other applications requiring device-to-device communication.

Overall, serial libraries play a crucial role in enabling communication between different electronic devices, allowing them to exchange data and commands efficiently.

- To include the `serial` library in Jupyter Notebook, you can follow these steps:

1. Install pySerial: If you haven't already installed the `pySerial` library, you can do so by running the following command in a Jupyter Notebook cell:

```
!pip install pyserial
```

This will install the `pySerial` library in your Jupyter Notebook environment.

2. Import the library: Once `pySerial` is installed, you can import the `serial` module in your Jupyter Notebook code cell using the `import` statement:

```
import serial
```

3. Verify installation: To verify that the library is imported correctly, you can run a simple command to print the version of the `serial` module:

```
print(serial.VERSION)
```

This should output the version number of the `serial` library if it is successfully imported.

By following these steps, you can include the `serial` library in your Jupyter Notebook environment and use it for serial communication within your Python code cells.

4.6.4Include the MediaPipe Library in Jupyter Notebook

MediaPipe is a comprehensive, cross-platform framework developed by Google that provides tools for building machine learning-based applications for various media processing tasks. It offers a wide range of pre-built, customizable machine learning pipelines and components, making it easier for developers to create applications for tasks like object detection, pose estimation, hand tracking, face detection, and more.

Key features of MediaPipe include:

1. Modular Design: MediaPipe is designed with a modular architecture, allowing developers to easily assemble and customize machine learning pipelines for their specific use cases.

2. Cross-Platform Support: MediaPipe supports multiple platforms including Android, iOS, Windows, Linux, and macOS, making it suitable for a wide range of applications across different devices.

3. Ready-to-Use Components: MediaPipe provides a collection of pre-trained models and ready-to-use components for various tasks such as pose detection, hand tracking, face detection, and more, enabling developers to quickly integrate advanced functionality into their applications.

4. Efficient Inference: MediaPipe is optimized for real-time performance and efficient inference, making it suitable for applications that require low-latency processing, such as augmented reality (AR) and virtual reality (VR) applications.

5. Flexible Integration: MediaPipe offers APIs and tools for easy integration with other frameworks and platforms, allowing developers to leverage existing infrastructure and workflows.

Overall, MediaPipe simplifies the development of machine learning-based applications for media processing tasks, enabling developers to create innovative and immersive experiences across various domains.

- **To include the `mediapipe` library in Jupyter Notebook, you can follow these steps:**

1. Install `mediapipe`: If you haven't installed the `mediapipe` library yet, you can install it using pip in your terminal or command prompt:

```
pip install mediapipe
```

2. Open Jupyter Notebook: Launch Jupyter Notebook by typing the following command in your terminal or command prompt:

```
jupyter notebook
```

3. Create a new notebook or open an existing one: Navigate to the directory where you want to create a new notebook or open an existing notebook.

4. Import `mediapipe` in your notebook: In a code cell in your Jupyter Notebook, import the `mediapipe` library.

```
import mediapipe as mp
```

5. Use `mediapipe` functionalities: You can now use the functionalities provided by the `mediapipe` library in your notebook. For example, you can create a `Hands` object to detect hand landmarks:

```
mp_hands = mp.solutions.hands
```

6. Run the code: Execute the code cell in your Jupyter Notebook to import the `mediapipe` library and use its functionalities.

That's it! You have now included the `mediapipe` library in your Jupyter Notebook and can use it for hand gesture detection and other tasks. Make sure to install any other dependencies required for your specific project as well.

4.6.5 Flowchart for Python Code

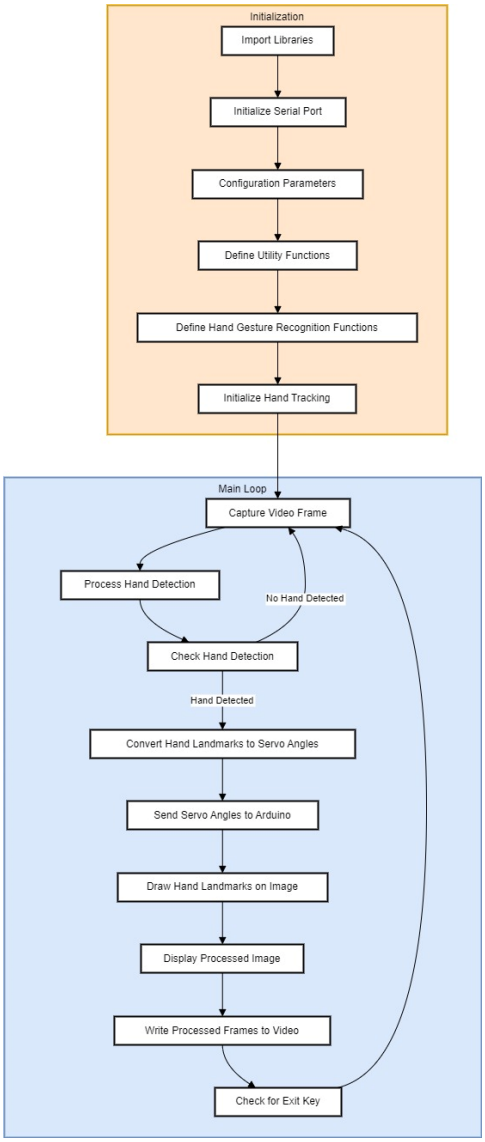

Fig 4.49 Python code Flowchart

4.6.6 Python Code for the Gesture Controlled Robotic Arm Using Arduino

```python
import serial
import cv2
import mediapipe as mp

# Initialize serial port for communication with the Arduino
ser = serial.Serial('COM11', 9600, timeout=0.1)

# config
write_video = True
debug = False
cam_source = 0  # 0 for the built-in webcam

x_min = 0
x_mid = 75
x_max = 150
# use angle between wrist and index finger to control x axis
palm_angle_min = -50
palm_angle_mid = 20

y_min = 0
y_mid = 90
y_max = 180
# use wrist y to control y axis
wrist_y_min = 0.3
wrist_y_max = 0.9

z_min = 10
z_mid = 90
z_max = 180
# use palm size to control z axis
plam_size_min = 0.1
plam_size_max = 0.3

claw_open_angle = 60
claw_close_angle = 0

servo_angle = [x_mid, y_mid, z_mid, claw_open_angle] # [x, y, z, claw]
```

```python
prev_servo_angle = servo_angle
fist_threshold = 7

mp_drawing = mp.solutions.drawing_utils
mp_drawing_styles = mp.solutions.drawing_styles
mp_hands = mp.solutions.hands

cap = cv2.VideoCapture(cam_source)

# video writer
if write_video:
    fourcc = cv2.VideoWriter_fourcc(*'XVID')
    out = cv2.VideoWriter('output.avi', fourcc, 60.0, (640, 480))

clamp = lambda n, minn, maxn: max(min(maxn, n), minn)
map_range = lambda x, in_min, in_max, out_min, out_max: abs((x - in_min) * (out_max - out_min) // (in_max - in_min) + out_min)

# Check if the hand is a fist
def is_fist(hand_landmarks, palm_size):
    # calculate the distance between the wrist and the each finger tip
    distance_sum = 0
    WRIST = hand_landmarks.landmark[0]
    for i in [7,8,11,12,15,16,19,20]:
        distance_sum += ((WRIST.x - hand_landmarks.landmark[i].x)**2 + \
                         (WRIST.y - hand_landmarks.landmark[i].y)**2 + \
                         (WRIST.z - hand_landmarks.landmark[i].z)**2)**0.5
    return distance_sum/palm_size < fist_threshold

def landmark_to_servo_angle(hand_landmarks):
    servo_angle = [x_mid, y_mid, z_mid, claw_open_angle]
    WRIST = hand_landmarks.landmark[0]
    INDEX_FINGER_MCP = hand_landmarks.landmark[5]
    # calculate the distance between the wrist and the index finger
    palm_size = ((WRIST.x - INDEX_FINGER_MCP.x)**2 + (WRIST.y - INDEX_FINGER_MCP.y)**2 + (WRIST.z - INDEX_FINGER_MCP.z)**2)**0.5

    if is_fist(hand_landmarks, palm_size):
        servo_angle[3] = claw_close_angle
    else:
        servo_angle[3] = claw_open_angle

    # calculate x angle
    distance = palm_size
    angle = (WRIST.x - INDEX_FINGER_MCP.x) / distance  # calculate the radian between the wrist and the index finger
    angle = int(angle * 180 / 3.1415926)               # convert radian to degree
    angle = clamp(angle, palm_angle_min, palm_angle_mid)
    servo_angle[0] = map_range(angle, palm_angle_min, palm_angle_mid, x_max, x_min)

    # calculate y angle
    wrist_y = clamp(WRIST.y, wrist_y_min, wrist_y_max)
    servo_angle[1] = map_range(wrist_y, wrist_y_min, wrist_y_max, y_max, y_min)

    # calculate z angle
    palm_size = clamp(palm_size, plam_size_min, plam_size_max)
    servo_angle[2] = map_range(palm_size, plam_size_min, plam_size_max, z_max, z_min)

    # float to int
    servo_angle = [int(i) for i in servo_angle]

    return servo_angle

with mp_hands.Hands(model_complexity=0, min_detection_confidence=0.5, min_tracking_confidence=0.5) as hands:
    while cap.isOpened():
        success, image = cap.read()
        if not success:
            print("Ignoring empty camera frame.")
            # If loading a video, use 'break' instead of 'continue'.
            continue

        # To improve performance, optionally mark the image as not writeable to
        # pass by reference.
        image.flags.writeable = False
```

```
        image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
        results = hands.process(image)

Source Control (Ctrl+Shift+G)  the hand annotations on the image.
        image.flags.writeable = True
        image = cv2.cvtColor(image, cv2.COLOR_RGB2BGR)
        if results.multi_hand_landmarks:
            if len(results.multi_hand_landmarks) == 1:
                # print("One hand detected")
                hand_landmarks = results.multi_hand_landmarks[0]
                servo_angle = landmark_to_servo_angle(hand_landmarks)

                if servo_angle != prev_servo_angle:
                    print("Servo angle: ", servo_angle)
                    prev_servo_angle = servo_angle
                    if not debug:
                        ser.write(bytearray(servo_angle))
            else:
                print("More than one hand detected")
            for hand_landmarks in results.multi_hand_landmarks:
                mp_drawing.draw_landmarks(
                    image,
                    hand_landmarks,
                    mp_hands.HAND_CONNECTIONS,
                    mp_drawing_styles.get_default_hand_landmarks_style(),
                    mp_drawing_styles.get_default_hand_connections_style())
        # Flip the image horizontally for a selfie-view display.
        image = cv2.flip(image, 1)
        # show servo angle
        cv2.putText(image, str(servo_angle), (10, 30), cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 0, 255), 2, cv2.LINE_AA)
        cv2.imshow('MediaPipe Hands', image)

        if write_video:
            out.write(image)
        # Check for exit key (ESC)
        if cv2.waitKey(5) & 0xFF == 27:
            if write_video:
```

```
                    out.release()
                break

    # Close serial port and release resources
    if not debug:
        ser.close()
    cap.release()
    cv2.destroyAllWindows()
```

- **Let's break down the code into sections and explain each part:**

1. Library Import:

- This section imports necessary libraries for the project, including `serial` for serial communication, `cv2` for computer vision tasks, and `mediapipe` for hand gesture detection.

2. Serial Communication Setup:

- The code initializes serial communication with the Arduino board connected to the specified COM port (`COM11`) with a baud rate of 9600.

3. Configuration Parameters:

- This section defines various configuration parameters used throughout the code, such as video recording settings (`write_video`), debugging mode (`debug`), webcam source (`cam_source`), and servo motor control parameters.

4. Utility Functions:

- This part defines utility functions that are used in the code. `clamp` function clamps values within specified ranges, and `map_range` function maps values from one range to another.

5. Hand Gesture Detection:

- The main loop of the code captures frames from the webcam feed, processes them using the MediaPipe library to detect hand landmarks and gestures.

6. Servo Angle Calculation:

- This section calculates servo motor angles based on detected hand landmarks and gestures. It maps hand positions and gestures to servo motor angles for controlling the robotic arm.

7. Arduino Communication and Control:

- The code sends servo motor angles to the Arduino board via serial communication (`ser.write(bytearray(servo_angle))`) to control the robotic arm.

8. Visualization:

- The code visualizes hand landmarks and gestures on the webcam feed using OpenCV (`cv2`), and displays the servo motor angles on the screen.

9. Video Recording:

- Optionally, this section records the webcam feed and hand gesture visualization to a video file (`output.avi`) if `write_video` is set to `True`.

10. Exit Condition:

- The code checks for the ESC key press (`cv2.waitKey(5) & 0xFF == 27`) to exit the program gracefully.

11.Resource Cleanup:

- Finally, this part of the code closes the serial port and releases the webcam feed and OpenCV windows when the program exits.

By breaking down the code into sections, it becomes easier to understand the purpose and functionality of each part, making it more manageable to comprehend and debug if needed.

4.6.7 Installation of Arduino IDE

Here are the steps to install Arduino IDE and set up libraries for servo motor:

Download Arduino IDE: Visit the official Arduino website https://www.arduino.cc/en/software and download the latest version of the Arduino IDE suitable for your operating system (Windows, macOS, or Linux).

Install Arduino IDE: Once the download is complete, run the installer and follow the on-screen instructions to install the Arduino IDE on your computer.

Open Arduino IDE: After installation, open the Arduino IDE from the desktop shortcut or the Start menu.

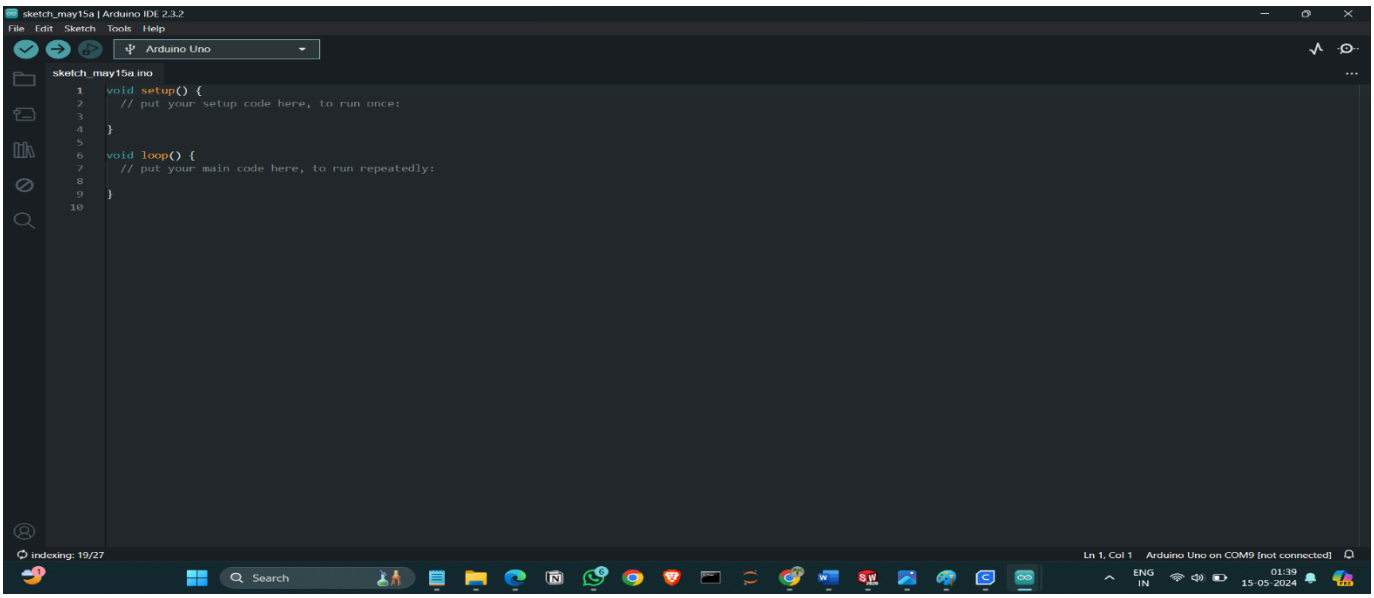

Fig 4.48 Arduino IDE

Set up Board: Go to the "Tools" menu, select "Board," and choose the appropriate Arduino board you are using (e.g., Arduino Uno, Arduino Mega, etc.)

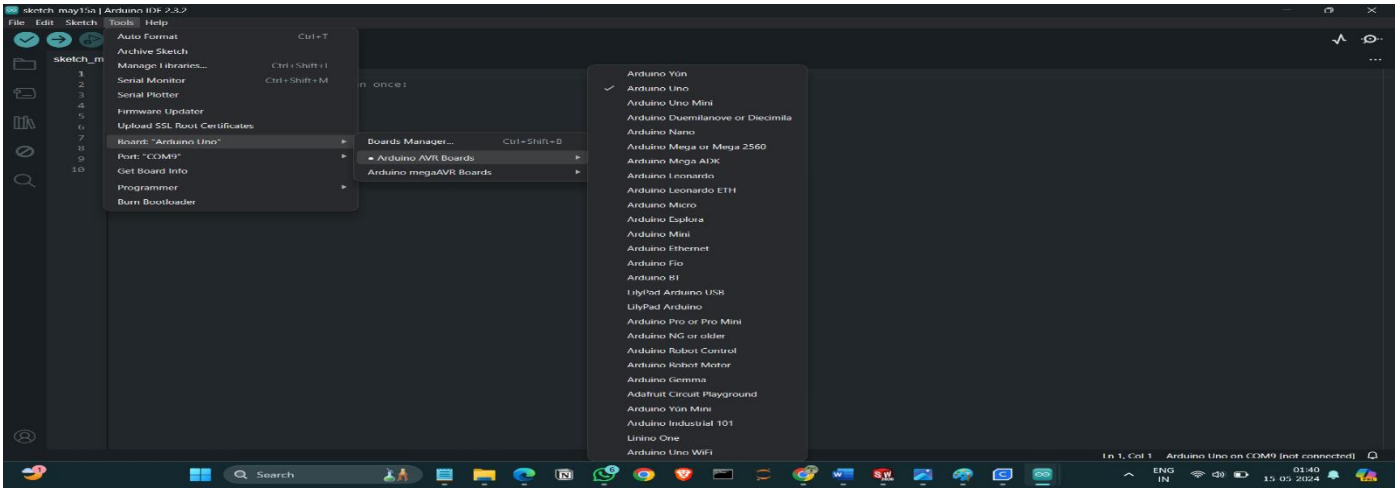

Fig 4.49 Tools Menu

Connect Arduino Board: Connect your Arduino board to your computer using a USB cable. The drivers for the Arduino board will be automatically installed.

Install Servo Library: To control servo motors, you need to install the Servo library. Go to the "Sketch" menu, navigate to "Include Library," and select "Manage Libraries."

Search for Servo Library: In the Library Manager window, type "Servo" in the search bar. The Servo library by Michael Margolis should appear in the list.

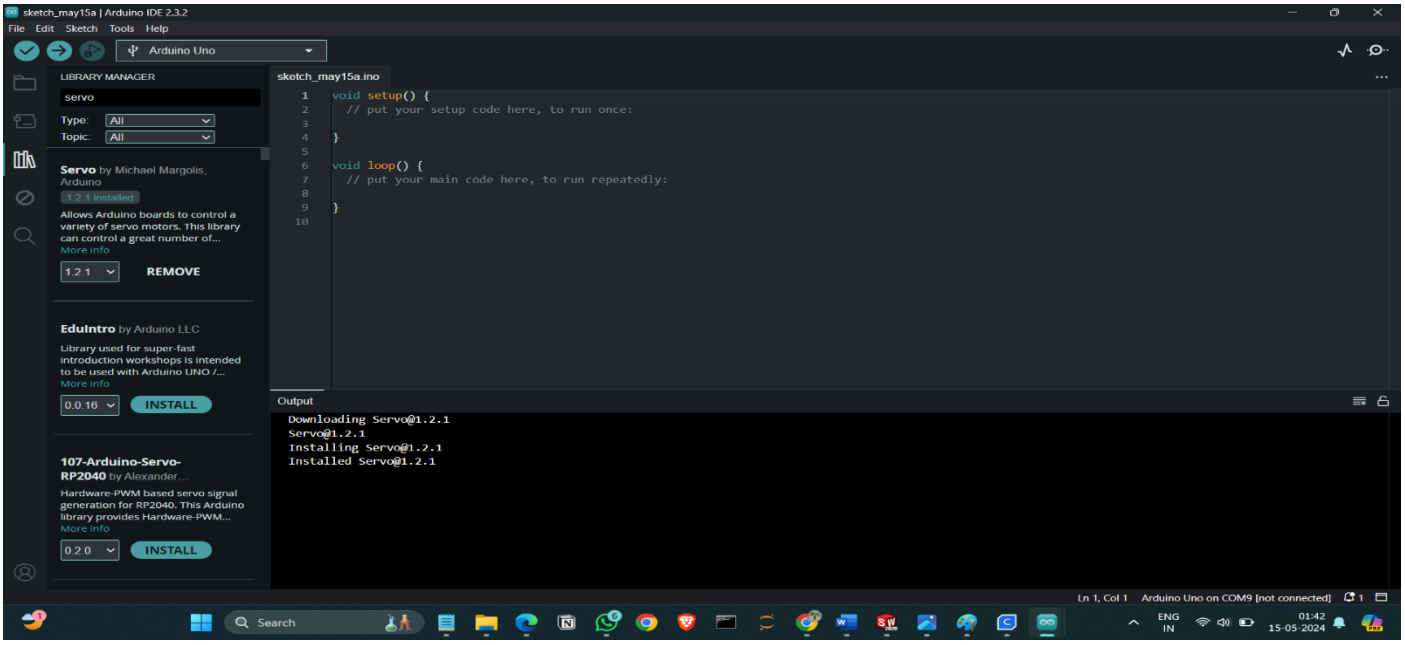

Fig 4.50 Libraries Manager

Install Servo Library: Click on the Servo library, then click the "Install" button to install it. Once installed, close the Library Manager window. By following these steps, you should be able to install Arduino IDE and set up the necessary libraries to control servo motors.

4.6.8 Arduino Code Flowchart

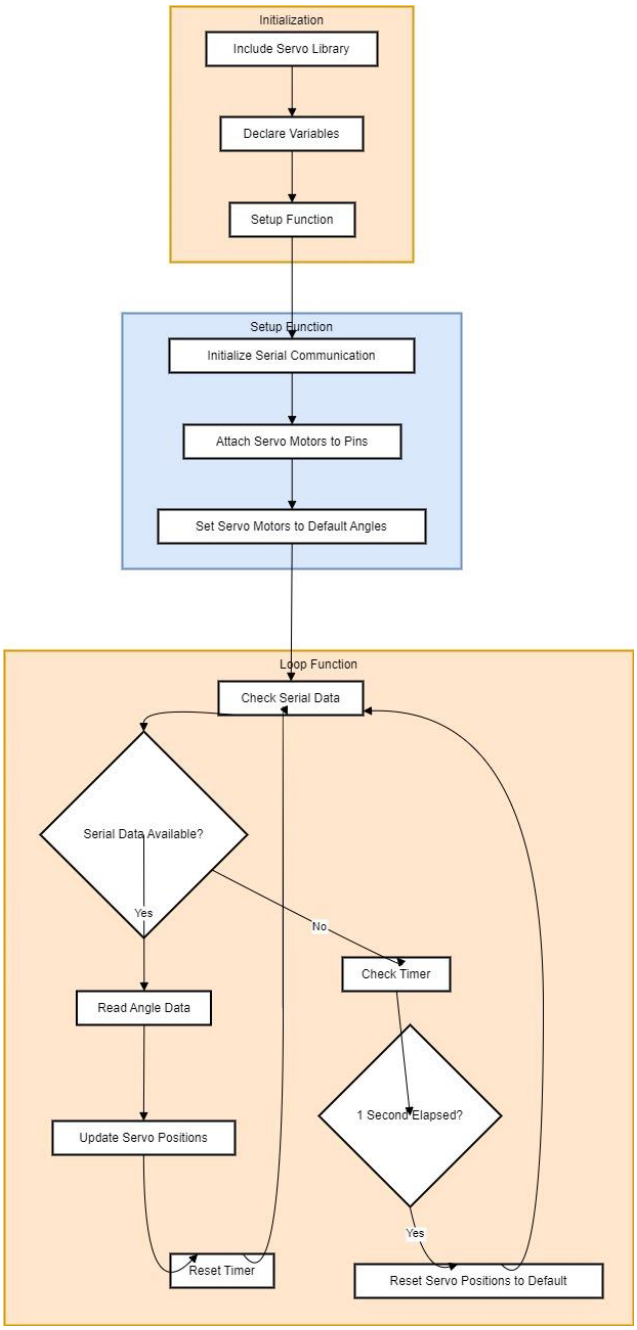

Fig 4.50 Flowchart for Arduino code

4.6.9 Arduino Code to Control Servo Motors Using Hand Gestures

This code is written for an Arduino microcontroller to control four servo motors. Here's an explanation of each part of the code:

1. Including the Servo Library:`#include <Servo.h>` includes the Servo library, which allows the Arduino to control servo motors.

2. Initializing Servo Objects:`Servo servo[4];` declares an array of Servo objects named `servo`. In this case, it's an array of 4 servo motors.

3. Defining Default Angles:`int default_angle[4] = {75, 90, 90, 60};` initializes an array `default_angle` with default angle values for each servo motor.

4. Setup Function: The **`setup()`** function is executed once when the Arduino starts.

- `Serial.begin(9600);` initializes serial communication at a baud rate of 9600.

- `servo[i].attach(pin);` attaches each servo object to a specific digital pin on the Arduino.

- `servo[i].write(angle);` sets the initial position of each servo motor to the default angle.

5. Loop Function: The **`loop()`** function continuously executes the code inside its curly braces.

- `if (Serial.available())` checks if there is data available on the serial port.

- `Serial.readBytes(angle, 4);` reads 4 bytes of data from the serial port into the `angle` array.

- `servo[i].write(angle[i]);` sets the position of each servo motor based on the received angle data.

- `if (millis() - t > 1000)` checks if one second has passed since the last servo movement.

- If one second has passed, it resets each servo motor to its default angle.

- **Note:** This code essentially listens for angle data sent over the serial port and moves the servo motors accordingly. If no new angle data is received within one second, it resets the servo motors to their default position

- **Code**

```
#include <Servo.h>
Servo servo[4];
int default_angle[4] = {75, 90, 90, 60};
void setup()
{
    Serial.begin(9600);
    servo[0].attach(5);
    servo[1].attach(6);
    servo[2].attach(7);
    servo[3].attach(8);

    for (size_t i = 0; i < 4; i++)
    {
        servo[i].write(default_angle[i]);
    }
}
byte angle[4];
byte pre_angle[4];
long t = millis();

void loop()
{
    if (Serial.available())
    {
        Serial.readBytes(angle, 4);
        for (size_t i = 0; i < 4; i++)
        {
            if (angle[i] != pre_angle[i])
            {
                servo[i].write(angle[i]);
                pre_angle[i] = angle[i];
            }
        }
        t = millis();
    }

    if (millis() - t > 1000)
    {
        for (size_t i = 0; i < 4; i++)
        {
            servo[i].write(default_angle[i]);
            pre_angle[i] = default_angle[i];
        }
    }
}
```

Chapter 5

RESULT

5.1 Comprehensive Guide to Robotic Arm Software Setup

5.1.1 Circuit Diagram of Arduino and Servo Motor

With the assembly of all the components of the robotic arm complete, the next step involves connecting the servo motors to the Arduino board and the power supply. Given the high-power requirements of the servo motors, it becomes evident that the Arduino alone cannot adequately supply the necessary power. Therefore, a dedicated power supply unit capable of meeting the demands of the servo motors is employed. By establishing this robust electrical connection, we ensure that the servo motors receive sufficient power to operate effectively, enabling precise control and smooth movement of the robotic arm.

- **Here is the Circuit Diagram of the Arduino and the Servo Motors**

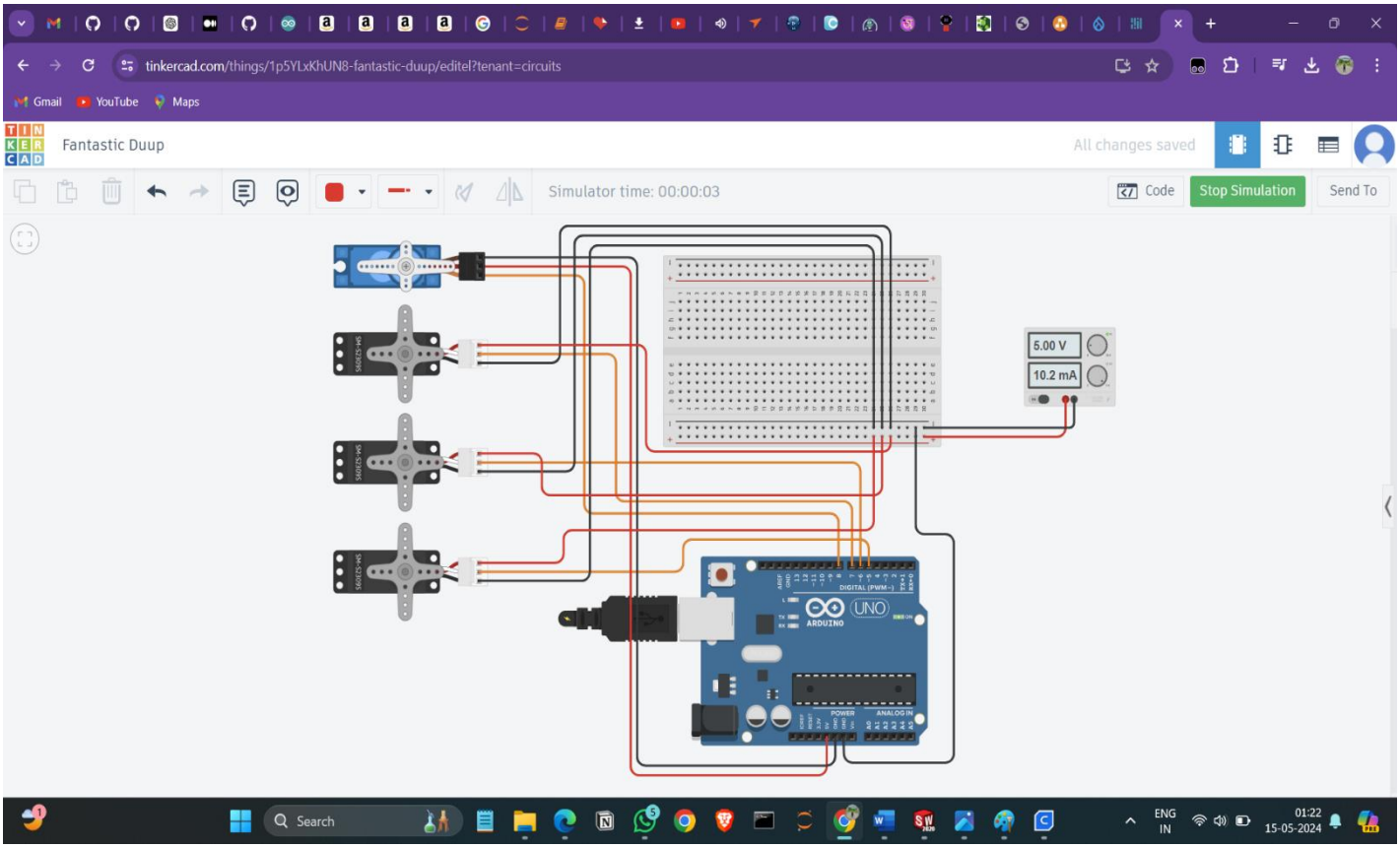

Fig 5.1 Circuit Diagram

5.1.2 Jupyter Notebook Setup (anaconda 3)

After meticulously completing all the necessary library installations and ensuring each circuit connection for the robotic arm is properly configured, we have reached the exciting phase of our project. With all the hardware components in place and the software environment ready, it's now time to run the Python code. This will bring our robotic arm to life, allowing us to test its functionality and execute the programmed tasks. Let's dive into the code execution and watch our project in action.

NOTE: Make sure that the COM port is connected to the Arduino Broad while running the code otherwise it will give errors.

- After code execution

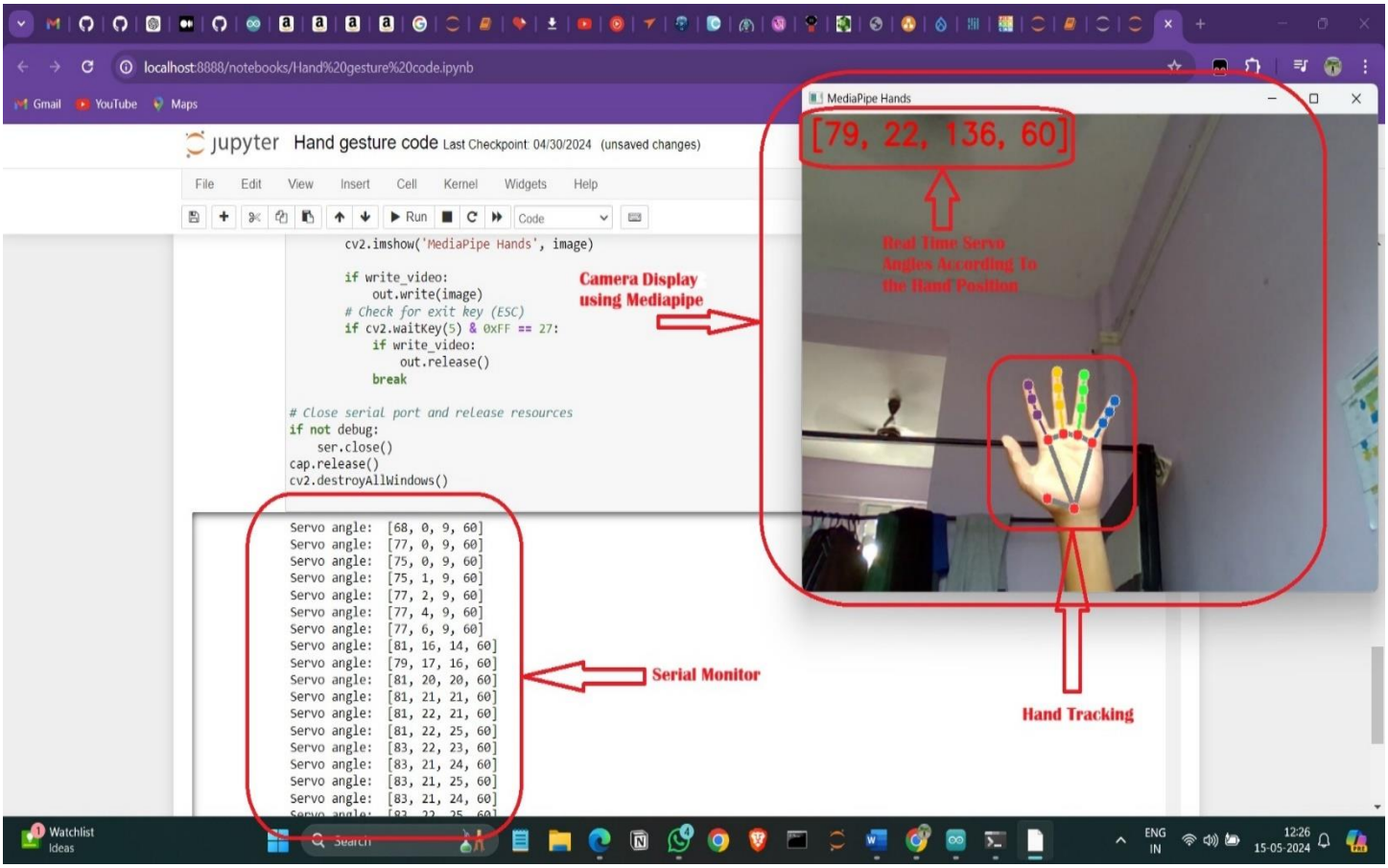

Fig 5.2 Jupyter Notebook while running the code

5.1.3 Uploading Program to Arduino

Fig 5.3 Uploading Program to Arduino

5.2 Implementation

5.2.1 Complete Robotic Arm Circuit Connections

We have now reached the final connection stage of our robotic arm, carefully following the detailed circuit diagram. Every wire and motor have been meticulously connected to ensure proper functionality and integration. With this last step completed, our robotic arm is now fully assembled according to the prescribed schematics. We are ready to proceed with testing and bring our robotic creation to life.

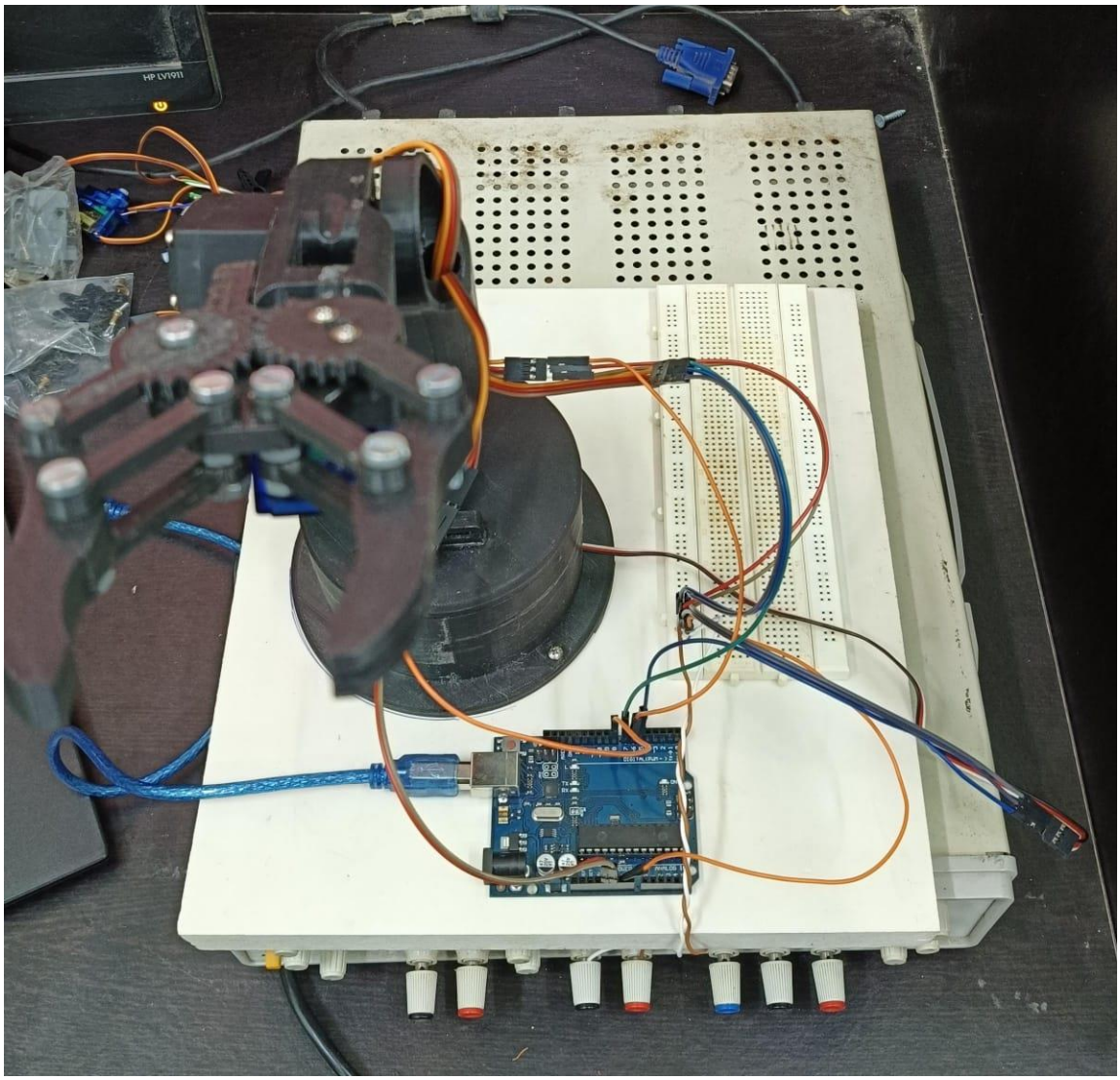

Fig 5.4 Complete Robotic Arm

5.3 Working Robotic Arm

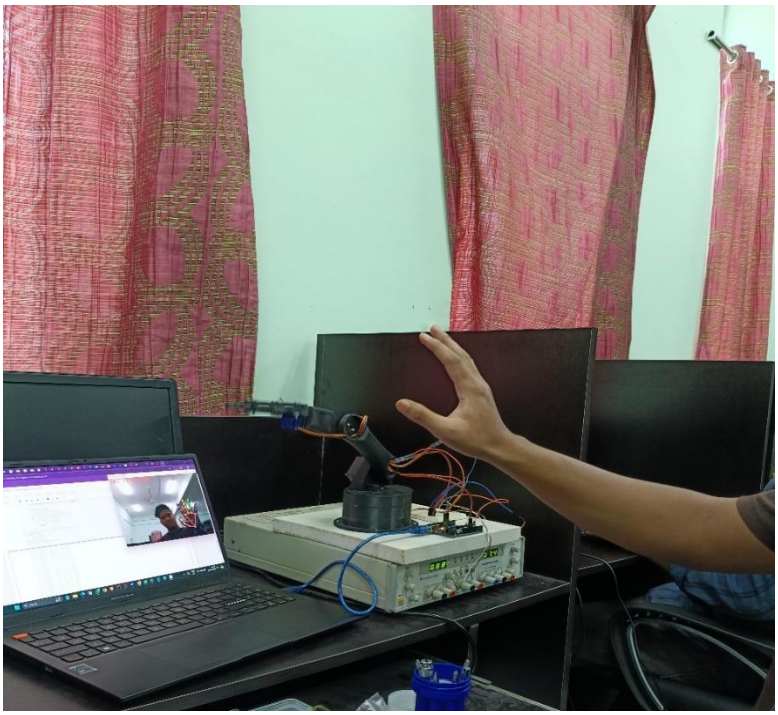

Fig 5.5 Working Robotic Arm with Claw Open

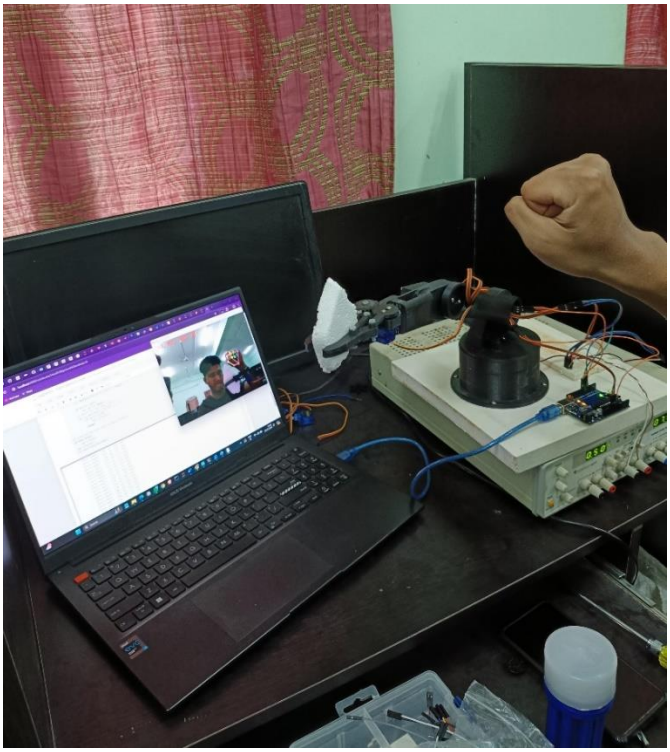

Fig 5.6 Working Robotic Arm with Closed Claw

Chapter 6

Future work and Conclusion

6.1 Future Works

The development of a gesture-controlled robotic arm using Arduino and hand sign detection has opened several avenues for future research and enhancements. Here are some potential directions for future work:

1. **Improved Gesture Recognition Algorithms**:Implementing more advanced machine learning algorithms, such as deep learning models, could enhance the accuracy and robustness of gesture recognition. This could include training on larger, more diverse datasets to better handle variations in lighting, background, and individual hand shapes.

2. **Enhanced Sensor Integration**:Incorporating additional types of sensors, such as infrared sensors, ultrasonic sensors, or cameras with depth sensing capabilities, could improve the precision and range of gesture detection. This would allow the system to function effectively in a wider variety of environments and conditions.

3. **User Interface Development**:Developing a more sophisticated user interface, possibly integrating with mobile devices or computers, could provide users with additional control options and feedback mechanisms. This could include visual representations of the arm's movements and real-time status updates.

4. **Robustness and Durability**:Enhancing the physical robustness and durability of the robotic arm and its components would make the system more suitable for industrial or heavy-duty applications. This could involve using stronger materials, improving the design of joints and linkages, and protecting electronic components from environmental factors.

5. **Machine Learning for Adaptive Control**:Implementing adaptive control algorithms that can learn from user interactions over time could make the system more intuitive and responsive. This could involve using reinforcement learning techniques to optimize the arm's movements based on user preferences and performance feedback.

By pursuing these future work directions, the gesture-controlled robotic arm project can continue to evolve, offering enhanced capabilities, greater reliability, and broader applicability across various fields. These advancements will contribute to making gesture-controlled robotic systems more accessible, efficient, and beneficial for users in diverse contexts.

6.2 Conclusions

In this thesis, we have explored the development of a gesture-controlled robotic arm using Arduino and hand sign detection. The project aimed to create a user-friendly, cost-effective system that leverages intuitive human gestures to control a robotic arm's movements, thereby enhancing the interface between humans and machines. Through the integration of Arduino microcontrollers, sensors, and actuators, we successfully designed a prototype that accurately interprets hand gestures and translates them into precise robotic arm motions. The system's hardware components were selected for their affordability, availability, and compatibility with Arduino, ensuring that the overall setup remains accessible for hobbyists and researchers alike.

The software architecture, developed using the Arduino IDE, demonstrated the capability to process sensor inputs in real-time, execute control algorithms, and manage the movement of the robotic arm. The implementation of machine learning algorithms for gesture recognition significantly improved the accuracy and responsiveness of the system, allowing for a seamless user experience.

Our experimental results showed that the gesture-controlled robotic arm could perform a variety of tasks with a high degree of precision and reliability. The arm successfully responded to a range of predefined gestures, confirming the effectiveness of our design and implementation. Moreover, the system's modularity allows for easy expansion and customization, making it adaptable to different applications and use cases.

In conclusion, this project underscores the potential of combining Arduino with gesture recognition technology to create innovative and practical robotic systems. The gesture-controlled robotic arm developed in this study not only demonstrates the feasibility of such systems but also opens up new possibilities for human-computer interaction in various fields, including assistive technology, industrial automation, and interactive entertainment. Future work could focus on enhancing the robustness of the system, incorporating advanced machine learning techniques, and exploring new application areas to further expand the capabilities and impact of gesture-controlled robotics.

References

[1]. Khajone, S. A., Mohod, S. W., & Harne, V. M., (2015) "Implementation of Wireless Gesture Controlled Robotic Arm", International Journal of Innovative research in Computer and Communication Engineering, 3 (1), pp. 375 – 379.

[2]. Brahmani, K., Roy, K. S., & Ali, M., (2013) "Arm 7 Based Robotic Arm Control by Electronic Gesture Recognition Unit Using MEMS", International Journal of Engineering Trends and Technology, 4 (4), pp. 1245 – 1248.

[3]. Neto, P., Pires, N. J., & Moreira, P. A., (2009) "Accelerometer-Based Control of an Industrial Robotic Arm", International Journal of Electronics, 6, pp. 167 – 173.

[4]. Aggarwal, L., Gaur, V., & Verma, P., (2013) "Design and Implementation of a Wireless Gesture Controlled Robotic Arm with Vision", International Journal of Computer Applications (0975 – 8887), 79 (13), pp. 39 – 43.

[5]. Bhargava and A. Kumar, "Arduino controlled robotic arm," 2017 International conference of Electronics, Communication and Aerospace Technology (ICECA), Coimbatore, India, 2017, pp. 376-380, doi: 10.1109/ICECA.2017.8212837.

[6]. Gautam R, Gedam A, Zade A, Mahawadiwar A, Review on Development of Industrial robotic arm, IRJET, March 2017, Volume 4, Issue 3.

[7]. Elfasakhany A, Yanez E, Design and development of a competitive low cost Robot Arm with four degrees of freedom, MME, Nov 2011.

[8]. M. M. B. S. Pachaiyyapan and T.Sridhar, "Design and Analysis of an Articulated Robotic Arm for various Industrial Applications," 2015.

[9]. R. C. Luo, K.L. Su, .A multi agent multi sensor based real-time sensory control system for intelligent security robot. IEEE International Conference on Robotics and Automation, vol. 2, 2003, pp.2394 .2399.

[10]. Riazollah Firoozian, "Servo Motors and Industrial Control Theory." Springer; 1st edition (December 8, 2008).

[11]. Massimo Banzi. "Getting Started with Arduino", O'Reilly Media, Inc, 1st Edition, pp. 21-22, 2011.

[12]. Wai Mar Myint and Theingi, "Kinematic Control of Pick and Place Robot Arm", *International Journal of Engineering and Techniques*, vol. 1, no. 4, pp. 63-70, July-Aug 2015.

[13]. Ayokunle A. Awelewa, Kenechukwu C. Mbanisi, Samuel O. Majekodunmi, Ishioma A. Odigwe, Ayoade F. Agbetuyi and I. A. Samuel, "Development of a Prototype Robot Manipulator for Industrial Pick-and-Place Operations", *International Journal of Mechanical Mechatronics Engineering*, vol. 5, pp. 20-27, 2013.

[14]. Nimisha Limaye and Siddharth Verma, "5 Degrees of Freedom Robotic Arm Controlled using PS/2 Mouse", *International Journal of Engineering and Technical Research*, vol. 2, no. 11, November 2014.

[15]. Myint Khin Moe, Zaw Min Min Htun and Hla Myo Tun, "Position Control Method For Pick And Place Robot Arm For Object Sorting System", *International Journal of Scientific Technology Research*, vol. 5, no. 6, pp. 63-70, June 2016.

[16]. C. Chandra Mouli, P. Jyothi, K. Nagabhushan Raju and C. Nagaraja, "Design and Implementation of Robot Arm Control Using Lab VIEW and ARM Controller", *IOSR Journal of Electrical and Electronics Engineering*, vol. 6, no. 5, pp. 80-84, July-Aug 2013.

[17]. Escandon, C. Rodrigo and A. Marco Carpio, "Design of a Kinematic Control Model for an Anthropomorphic Robot Arm applied to the Teaching of Industrial Robotics", *16th International Conference on Advanced Robotics (ICAR)*, 2013.

[18]. Ashraf Elfasakhany, Eduardo Yanez, Karen Baylon and Ricardo Salgado, "Design and Development of a Competitive Low-Cost Robot Arm with Four Degrees of Freedom", *Modern Mechanical Engineering*, vol. 1, pp. 47-55, 2011.

[19]. Posadas Hector and Victor Fern, "Affordable Easy-to-use Robotic Arm used in Hardware Description Languages Teaching", *International Symposium on Computers in Education (SIIE)*, pp. 161-166, 2015.

[20]. Agnihotri Payal, V. Kbanga and Er. Gurjeet Singh, "Review of Kinematic Modelling and Control of 5 Degree of Freedom Robotic Arm Using DH Representation", *2nd International Conference on Recent Innovations in Science Engineering and Management*, November 2015.

[21]. Li Wen-Bo, Guang-Zhong Cao, Xiao-Qin Guo and Su-Dan Huang, "Development of a 4-DOF SCARA Robot with 3R1P for Pick-and-Place Tasks", *6th International Conference on Power Electronics Systems and Applications (PESA)*, pp. 1-5, 2015.

[22]. Fernando Gonzalez and Janusz Zalewski, "A Robotic Arm Simulator Software Tool for use in Introductory Robotics Courses", *IEEE Global Engineering Education Conference (EDUCON)*, pp. 861-866, 2014.

[23]. Popovic N., Williams S, Schmitz-Rode T., Rau G., Disselhorst-Klug C.,“Robot-based methodology for a kinematic and kinetic analysis of unconstrained, but reproducible upper extremity movement”, Journal of Biomechanics, Vol. 42 No. 10, 2009.

[24]. Wang Z., Xilun D., Alberto R. and Alessandro G., “Mobility analysis of the typical gait of a radial symmetrical six-legged robot”, Mechatronics, Vol. 21, No. 7, pp. 1133-1146, 2011.

[25]. Man C., Fan X., Li C. and Zhao Z., “Kinematics analysis based on screw theory of a humanoid robot”, Journal of China University of Mining & Technology, Vol. 17 No. 1, pp. 49–52, 2007.

[26]. Cubero S., “An inverse kinematics method for controlling all types of serial-link robot arms, Mechatronics and Machine Vision in Practice, Springer-Verlag, Berlin, pp. 217-232.

[27]. Kuma R., Kalra P and Prakash N., “A virtual RV-M1 robot system”, Robotics and Computer-Integrated Manufacturing, Vol. 26 No. 6, pp. 994-1000, 2011.