Department of Engineering Science, Faculty of Engineering and Science,
Chatham Maritime, Kent, ME4 4TB

FINAL PROJECT REPORT 2024/2025

Title: Robotic Arm for Education Purposes

Syed Shafi Ahmed [001248523]
BEng (Hons) Computer Engineering Cybernetics

Supervisor: **Jan Krabicka**

Word Count: **9230**

Abstract

This report presents the design, development, and evaluation of a 3-DOF robotic arm intended for educational use in schools, colleges, and universities. The primary aim was to create a cost-effective, modular, and interactive learning tool to teach basic concepts in robotics, programming, and motion control. The project builds on existing research in educational robotics, which emphasizes the importance of hands-on learning and low-cost solutions for effective STEM education.

The methodology involved designing a lightweight, 3D-printed robotic arm controlled by an Arduino Uno R3, using servo motors and programmed through the Arduino IDE. Functions such as manual control, moveTo(), and inverse kinematics were implemented to allow real-time movement and coordinate-based control of the end effector. The design was prototyped first using a breadboard and later refined with a custom PCB and 3D-printed parts sourced from open-source designs.

The robotic arm demonstrated reliable performance in basic object manipulation and 3D positioning, with smooth movement achieved through linear interpolation. While some challenges were encountered—such as servo failures and mechanical fitting issues—the overall functionality met the project's educational goals. The findings underscore the potential of such systems in engineering education, and future improvements could include AI integration, additional degrees of freedom, and enhanced control algorithms for broader application and greater learning depth.

Table of Contents

List of Figures

List of Tables

List of Abbreviations and Mathematical Symbols

STEM	Science, Technology, Engineering and Mathematics
DOF	Degrees of Freedom
PID	Proportional-Integral-Derivative Control
D	Dimension
θ	Angle
PWM	Pulse Width Modulation
AI	Artificial Intelligence
Number”	Number Inches
V	Volt
A	Ampere
Θ	Theta Degree

Acknowledgements

I would first like to express my sincere gratitude to my supervisor Dr. Jan Krabicka for his continuous guidance and support throughout this project. His insights, feedback, and encouragement played a crucial role in helping me understand the techniques and steps required to complete the work successfully.

I would also like to thank Dr. Michael Okereke for providing valuable video resources, which greatly assisted me in writing and structuring this report effectively.

My appreciation also goes to Bruce Hassan from the HAWK Lab at the University of Greenwich for his assistance in 3D printing the components essential to this project.

CHAPTER ONE

INTRODUCTION

1.1 Background of Project

In recent years, robotics has become a cornerstone of innovation in multiple fields, from automation and manufacturing to healthcare and entertainment. However, the application of robotics in education has seen significant growth. It has become essential for students to equip themselves with the skills and knowledge necessary to navigate the modern world with the rapid advancement of technology. In this case, Robotics plays a crucial role.

The choice to build a 6DOF robotic arm for educational purposes is driven by the need to provide an interactive learning tool that can teach fundamental concepts in robotics, programming, circuit designing, embedded systems, mechanics and PID control. A robotic arm serves the simplest and the most practical way to introduce beginners to complex topics such as motion control, sensor integration, circuit designing, Arduino programming and kinematics, allowing students to visualize and interact with the abstract concepts.

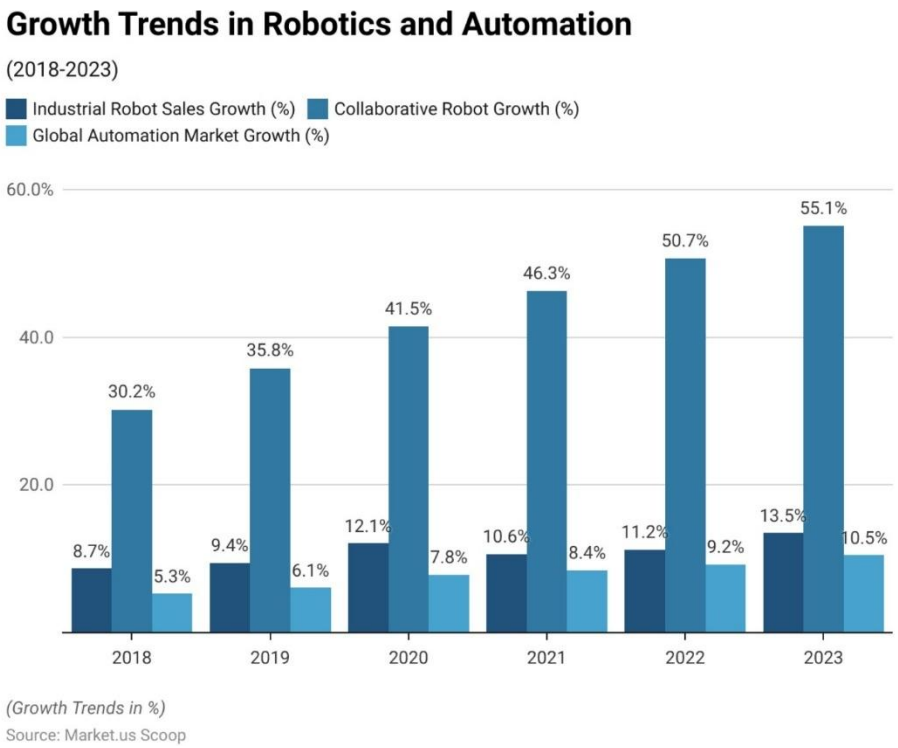


Figure 1:Global Trends in Robotics and Automation. (Tajammul Pangarkar, 2025)

In figure 1, Collaborative Robot Growth is drastically increasing over the years. Industrial Robot Sales increased by 13.5% in 2023. In this context of the growing demand for STEM skills, robotics studies are highly relevant. Industries are evolving and becoming more technology-driven, there is a growing need for individuals with a strong foundation in robotics. According to studies by the University of Oxford Department of Engineering, 47% of jobs are at risk of being lost to automation. (Frey, Carl Benedict, and Michael Osborne, 2013). Robotics, being an interdisciplinary field that draws on concepts from all the STEM subjects, offers a unique opportunity to bridge the gap between theory and practical application. By building and programming a robotic arm, students gain exposure to

engineering design, coding, and problem-solving techniques that are directly applicable in many modern industries. (Ramanand Amy Bernstein Anand, 2017)

Robotics plays a significant role in engaging students at various educational levels - schools, college, and university. For school students, a robotic arm can introduce the exciting world of robotics and engineering. At college, students can begin their journey of programming and using embedded systems. College students can be introduced to motion control and basic ideas of kinematics. University students can explore the construction of robotic arms from choosing motors, degree of freedom and circuit designing. They can carry out various types of projects implementing sensors, studying kinematics and introducing trajectory in motion control. (S. Jadhav, S. Iyer and K. Arya, 2024)

One of the key challenges is the affordability of robotic arms. Traditional educational methods often lean on passive learning, where students are absorbing information by going through lectures and textbooks. Educational robotics should be more practical and experimental as it provides project-based learning, fosters deeper understanding and skill development.

In this project, the aim is to design and develop a robotic arm that is not only functional but also easy to use and cost-effective. Making it accessible to educational institutions. By designing such a robot, students will gain exposure to important concepts like

kinematics, motion control, coding, control systems, servo motors, sensors and circuit design. They will also develop a better understanding of the use of robots in practical and industrial applications. This project will be a bridge between theoretical learning and hands-on experience, preparing students for a future in which robotics will play a crucial role.

1.2 Aims and Objectives of the Project

1.2.1 Project Aims

The aim of this project is to design and develop a cost-effective, easy-to-use 6-DOF robotic arm to serve as an educational tool for teaching robotics, motion control, circuit designing and programming to students at various educational levels.

1.2.2 Project Objectives

To Achieve the above aim, the project will focus on several specific objectives:

1. Design a 3DOF Robotic Arm using open-source design and selecting proper Servo Motors.
2. Design a detachable end effector with 3DOF Mechanical Gripper along with keeping options for various end effectors.
3. Designing Circuit using Breadboard for Motor connections and Power Supply, create a circuit board to minimize jumper wires.
4. Implement Kinematics for Motion Trajectory in 3D space and Locating Coordinates in 3D space.

5 Implement Interpolation for Smooth Movement.

6 Create Functions for motion control system and automation.

7 Create Manual Control System, where robotic arms are digitally controlled by using terminals.

8 Provide Tutorials and User Instructions.

9 Ensure Cost-Effectiveness and Scalability.

10 Enable Interactive Learning and Engagement.

11 Test and Evaluate Educational Effectiveness.

1.3 Definition of relevant terms

DOF (Degrees of Freedom): A robotic arm with degrees of freedom indicates the ability to move in 3-dimensional space, rotational movement and translational movements.

Joints: The movement axis or the motor location are indicated as joints. Similar to Human anatomy of arm.

Links: The extension or the support which connects each joint. In other words, the skeletal system for robotic arms.

Actuator: A machine that moves or controls components in a system by converting energy into physical motion.

End Effector: The tool attached at the end of a Robotic arm. End Effectors are usually a Mechanical claw/gripper replicating human finger. Depending on the function and purpose of the arm, end effector can vary.

Forward Kinematics: The process of calculating axis in 2D or 3D coordinates based on given angles. In this case the angles of Joints are calculated to pinpoint the end effector location. (Donald L. Pieper, 1968)

Inverse Kinematics: The process of calculating the angles based on the orientation of an object (End Effector in this case) in 2D or 3D space. This is in contrast to forward kinematics. (Donald L. Pieper, 1968)

Servo Motor: A small motor designed for precise control of angular position, commonly used in robotic systems to move joints and achieve angles.

Arduino UNO: A popular microcontroller board used in a wide range of electronics and robotics projects. It is programmed with Arduino language and IDE. Used for embedded systems and other robotic applications. It serves as the brain of the robotic arm.

Pulse Width Modulation (PWM): PWM is a technique used to control the speed and position of motors. The width of the pulse determines how long the motor will stay in a particular position or move at a specific speed. (Butterfield, Andrew J.; Szymanski, John, 2018)

Interpolation: A method used to estimate intermediate values between two known values in a set of data points. In robotics, interpolation is used to create smooth and continuous motion path between two points of the end effector. (Ben Moshe, Nir 2025)

Linear Interpolation: Movement Between two points are achieved linearly varying the position, speed and other parameters over time. (Ben Moshe, Nir 2025)

1.4 Scope of the Project

The scope of this project is to design and develop a 6-DOF robotic arm that serves as an educational tool for teaching robotics, control systems, and programming to students at various levels of education. Primarily the project focuses on providing an easy interactive platform where students can engage with fundamental robotic concepts such as motion control, servo control, kinematics and circuit design. The robotic arm will allow students to explore how mechanical systems are controlled through software and will serve as a practical introduction to robotics for school, college, and university students. However, the project comes with several key limitations

First of all, it will not incorporate artificial intelligence (AI) or machine learning algorithms. This project emphasizes teaching basic robotics principles, rather than advanced decision making or task recognition. Although students are free to explore and incorporate machine learning or AI.

The robotic arm is not designed for commercial use or industrial use. The dimension of the robotic arm is 10.21 inchx9.3 inchx5.27 inch [Length, Width, Height], constructed with 3D printed base, links and end effectors. The servo motors used in the project are cost effective motors. The robotic arm cannot lift anything wider than 6 centimeters and heavier than 100 grams. Anything more than this limit will damage the motors or break any fragile part. The replacement of parts is cheap but time consuming.

Additionally, the robotic arm will not include extensive sensor integration, such as force feedback sensors or vision systems. These sensors can be added to the design. The

control system will be basic, relying on standard servo motors for movement and position control. The robotic arm will be controlled using coding, modifying functions and using Arduino IDE terminal.

Lastly, the robotic arm has limited movement scope, each servo can rotate from 0 degree to 180 degrees. Changing the servos will not be possible as the 3D printed links were designed to fit MG996R and SG90 servo motors only. Programming beyond the movement limit can cause damage to the servos or the links.

In conclusion, this project will provide an easy-to-use, cost-effective educational tool for teaching robotics and control systems, with clear boundaries set around its capabilities.

1.5 Report Outline

Chapter 1: initiates with an introduction to the project's background, aims, and objectives.

Chapter 2: discusses the literature review, theory, and implementation approach utilized in this study.

Chapter 3: outlines the research methodology necessary for project completion.

Chapter 4: presents the study's findings and provides a thorough discussion of the results.

Chapter 5: addresses conclusions drawn from the study and outlines avenues for future research.

CHAPTER TWO

LITERATURE REVIEW

2.1 Literature Review

The robotic arm designed for this project aims for a hands-on and interactive learning platform for students from different educational levels. The purpose of this literature review is to examine some existing research of educational robotic arms and explore their design, control system and integration into educational platforms. Through reviews of academic papers, journals, books and articles, challenges associated with implementing educational robotic arms in schools, colleges and universities have been taken in account and the basis of the project has been formulated.

2.2 Mechanical Design of Educational Robots

Mechanical Design of the Robotic arm plays a critical role in ensuring the robot is both functional and accessible to the students. Many educational robotic arms are designed with 2-DOF or 6-DOF configurations. A 6-DOF robotic arm allows students to explore complex kinematics and control systems. It offers the students a realistic simulation of the human arm movement.

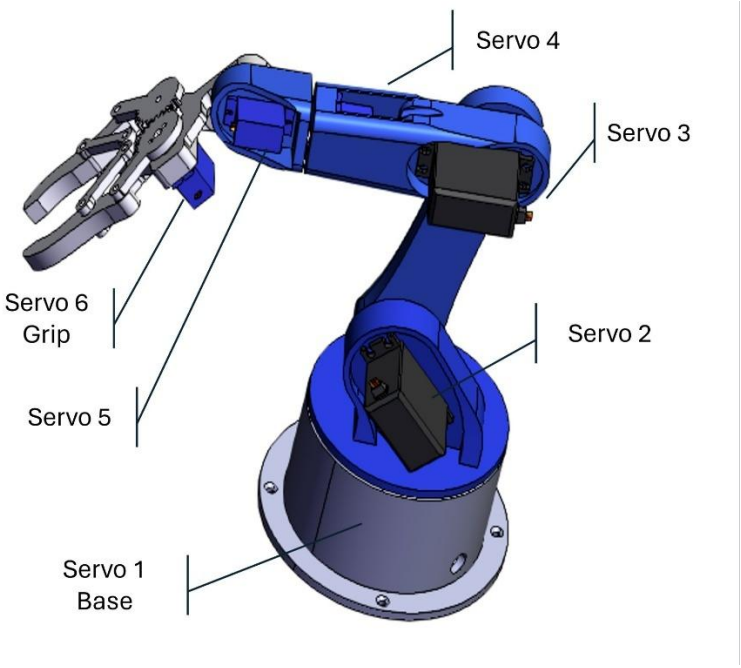


Figure 2:6-DOF robotic arm basic structure

According to Rajasekaran and Srinivasan (2019), deep understanding of both mechanical and computational aspects of robotics can be achieved by a well-designed 6-DOF robotic arm. The design consists of cost-effective, lightweight, durable materials such as plastic (3D printed) or aluminum. These materials provide sufficient stability for educational purposes. Using only a single type of material simplifies the construction process without the need for expensive equipment. (Tan and Toh, 2021)

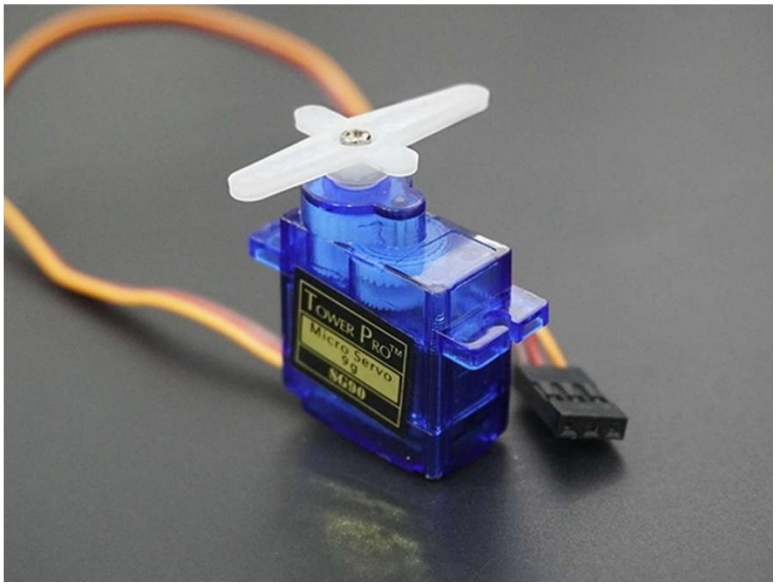


Figure 3:Servo Motor SG90

As for the choice of actuator for educational robotic arms, Servo motors are most commonly used due to its affordability, ease of control and ability to provide precise movement. Hwang and Kim (2019), in their design argued that servo motors enable accurate positioning and are widely used in educational robotic arms. Servo motors offer the simplest yet effective way to control robotic arms. DC motors can also be considered but servo motors remain the most practical due to their cost-effectiveness. Various libraries for servo motors are available which makes it easy to integrate simple control systems. A good design ensures that the arm is durable and capable of handling repetitive movements, crucial for longevity.

2.3 Control Systems for Robotic Arms

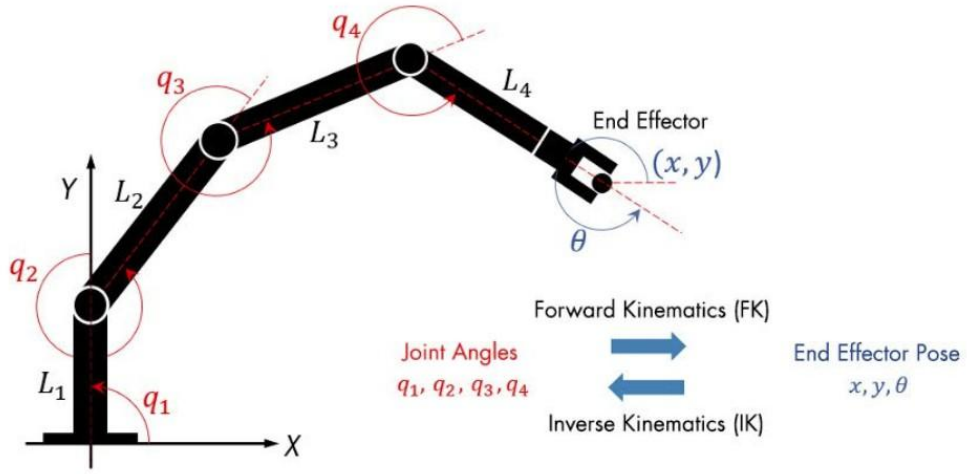


Figure 4:Application of Inverse Kinematics in Robotic Arms. (MathWorks, Inverse Kinematics: What Is Inverse Kinematics? - MATLAB & Simulink)

The fundamental of robotic arms functionality is the control system. Kinematics, inverse kinematics, forward kinematics, linear interpolations, PID controllers, radio controllers and mechanical controllers are some of the several methods developed to control the movement of robotic arms. Hwang and Kim (2019) demonstrated how inverse kinematics algorithms are implemented to calculate the joint angles for a 6-DOF robotic arm. Inverse Kinematics is essential for computing the necessary joint angles required to position the end effector at a desired point in space.

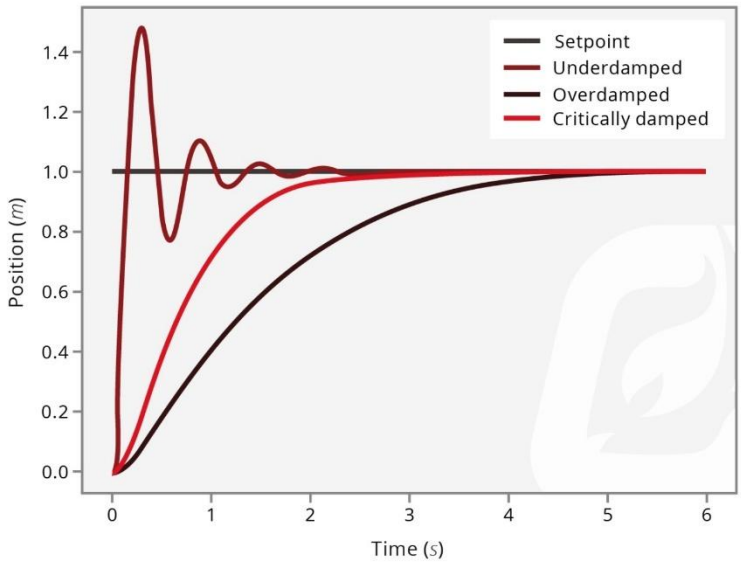


Figure 5:PID in Speed Control (Available: What is a PID (Proportional, Integral, Derivative) Controller?)

PID is a feedback loop control mechanism which minimizes the error between the desired and actual position of the robotic arm. It is another important aspect of robotic arm control. PID ensures smooth movement and precise movement to the target position of the end effector. It is an ideal control method for educational applications. (Ismail and Khanis,2017). PID control adjusts the motor's behavior based on 3 parameters: Proportional(P), integral(I) and derivative(D). This control system stabilizes the system and reduces oscillations or delays in movement. Anjaneyulu and Kumar (2018) suggest that PID controllers are widely used in educational robotic arms as they provide simple and effective control of motor position.

2.4 Integration Platform

Figure 6: Arduino Uno

Arduino and Raspberry Pi are very popular integration platforms used in educational robots as it is easier for students to program and control the robot. These platforms are affordable and versatile. They offer a simple interface for students to interact with hardware and software. According to Martinez and Reye (2017), Arduino and Raspberry Pi provide an excellent learning environment for students to explore programming, sensor integration and control systems. These platforms allow students to write code and experiment with the robotic arm's movement in a hands-on, interactive way, making the process of learning robotics more accessible and enjoyable.

2.5 Educational Impact of robotics

Integration of robotic arms in education has been shown to have significant benefits in enhancing students' engagement with STEM subjects. Educational Robotic Arm offers

students the opportunity to interact with scientific concepts in a practical and visual way. This method improves students' understanding of complex concepts. According to Khan et al. (2019), the use of robotics in educational institutions has been proven to improve students' motivation, increased interest in STEM careers and improved problem-solving skills. By designing and programming robotic systems, students not only learn about mechatronics and control systems but also develop essential skills in programming, critical thinking and teamwork.

McComb and Philips (2018), in their studies, heightened the ability of robotics to foster creativity and innovation in students. The hands-on nature of robotics projects encourages students to experiment, make mistakes and solve problems, all of which are crucial skills in the modern workforce. Furthermore, robotics requires knowledge from physics, engineering, and computer science. Such interdisciplinary trait offers students a holistic learning experience that can enhance their understanding of how different STEM fields are interconnected. Harnandez and Morales (2020) argue that robotics provides an interactive and immersive experience that not only makes learning more engaging but also helps bridge the gaps between theoretical knowledge and real-world applications.

Robotic arms are useful teaching aids in classrooms for subjects like programming, motion control, and sensor integration. In addition to teaching technical subjects, robotics can be used to teach more general abilities like perseverance, teamwork, and communication. Students who work with robots are more likely to be interested in and driven to pursue STEM-related careers, according to research. (McComb and Phillips, 2018)

2.6 Challenges in educational Robotics

Educational robotics has many advantages, but there are still some drawbacks, namely in the areas of affordability, usability, and accessibility. The cost of robotics is one of the main barriers preventing its widespread use in classrooms. The cost of high-quality robotic systems can prevent schools with tight budgets from purchasing them. Tan and Toh (2021), in their studies pointed out, this financial barrier limits the ability of many schools to implement robotics programs. Rajasekaran & Srinivasan (2019) mentions the progress in developing low-cost alternatives using open-source hardware, designs and platforms like Arduino and Raspberry Pi. These systems provide affordable solutions while maintaining functionality and educational value.

Easy to use is another significant challenge. Robotic systems are usually complex and may be difficult for school or college students. Therefore, it is crucial to design user-friendly interfaces that allow students to interact with the system without being overwhelmed by the technicalities of programming or control. Santos and Lopes (2018) addresses that simplifying control systems and making the hardware modular can help face the challenge.

The scalability of robotic systems across various educational levels is another challenge. The complexity needed for university-level students may not be there in a robotic arm designed for secondary school use. To maximize its impact, a robotic system must be designed to be modified at various educational levels (Khan et al., 2019). Furthermore, because not all schools have the infrastructure or resources necessary to implement robotics programs, accessibility is still an issue. To make robotics instruction more

accessible to a larger range of schools, solutions including open-source software, internet resources, and reasonably priced hardware are crucial (Santos & Lopes, 2018).

CHAPTER THREE

METHODOLOGY

1.1 Research Methodology

The design, construction, and evaluation of a 6-DOF robotic arm for instructional purposes are the goals of this project's research methodology. The process focuses on creating a working robotic arm with a detachable end effector, putting control systems in place, and making sure it is both affordable and scalable. The approach entails constructing hardware, writing software, creating a control system, and analysing the educational impact of the system.

The project follows a systematic design and implementation process, broken down into several phases: design, development, integration, testing, and evaluation. Each phase is aimed at achieving the specific objectives of the robotic arm, while ensuring that the system is easy to use, cost-effective, and suitable for educational environments

1.2 Materials

1.2.1 Arduino UnoR3 Microcontroller:

The Arduino Uno R3 was chosen as the central processing unit for the robotic arm due to its widespread use in educational settings and ease of integration with sensors and motors. The open-source nature of the Arduino platform ensures that students can modify and extend the system with minimal effort. Additionally, the Arduino IDE simplifies the process of writing and uploading code, making it an ideal choice for an educational project.

1.2.2 Servo Motors:

The Servo Motor SG996R was used for driving the main joints of the robotic arm, while the SG90 servo motor was selected for constructing the mechanical gripper. Both SG996R and SG90 are widely recognized for their cost-effectiveness, making them suitable choices for an educational-purpose robotic arm project. Despite their affordability, these motors provide sufficient torque output to handle the required range of motion and load for lightweight tasks.

In terms of power requirements, both motors operate efficiently within a 12V 2A power supply, ensuring stable performance throughout usage. They are also known for their durability and consistent functionality, which adds to the overall reliability of the system. Additionally, their compact dimensions and lightweight build contribute to the robotic arm's ease of assembly and structural balance, making them ideal components for a low-cost, educational robotics platform.

1.2.3 3.3.3 3D Printed Components:

An open-source 3D model of a robotic arm and end effector was sourced for this project. The design is fully compatible with MG996R and SG90 servo motors, ensuring seamless integration of the mechanical and electronic components. The lengths of the links are appropriately proportioned, allowing for effective weight distribution across the structure. This ensures that the load remains within the operational capacity of each servo motor, contributing to stable and efficient movement.

The overall design is compact, lightweight, and mechanically durable, making it well-suited for educational and demonstration purposes. All 3D components were printed at Hawk Laboratories, University of Greenwich, using high-quality materials to ensure strength and dimensional accuracy.

1.2.4 Power Supply:

A 12V 2A power supply was selected to provide sufficient power for the servo motors used in the robotic arm. This supply ensures stable operation and delivers the required voltage and current during extended use. The Arduino Uno R3, on the other hand, is powered separately via the laptop’s USB terminal.

1.2.5 Breadboard and PCB:

The initial circuit was prototyped using a breadboard to allow flexibility during the design and testing phases. Once the circuitry was finalized, a custom printed circuit board (PCB)

was developed to enhance reliability and reduce the reliance on jumper wires. The PCB was fabricated by JLCPCB specifically for this project.

1.3 Methods

This section outlines the steps involved in the design, construction, and testing of the robotic arm, as well as the methods used to implement the control system, kinematics, motion control, and user interface.

1.3.1 Mechanical Design and Construction

1.3.1.1 Robotic Arm Skeletal Design and Construction:

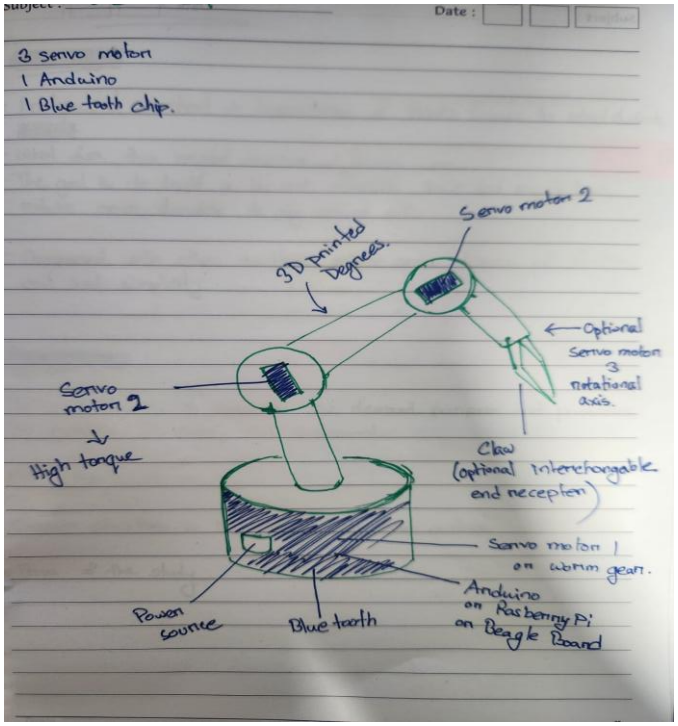


Figure 7: Hand Drawn Design of Robotic Arm

The initial concept for this project began with a hand-drawn design, which provided a preliminary overview of the robotic arm's structure. This sketch illustrated the

placement of the servo motors and clearly defined the number of joints and links within the system. The initial sketch outlines the use of three primary servo motors assigned to control the base, shoulder, and elbow joints of the robotic arm. In addition to these, the design includes provisions for three optional servo motors intended for wrist articulation and gripper operation. This configuration allows for enhanced flexibility and increased degrees of freedom, supporting more complex and precise movements if the optional motors are implemented. Although a Bluetooth module was referenced in the original design, it was ultimately not included in the final implementation of the project.

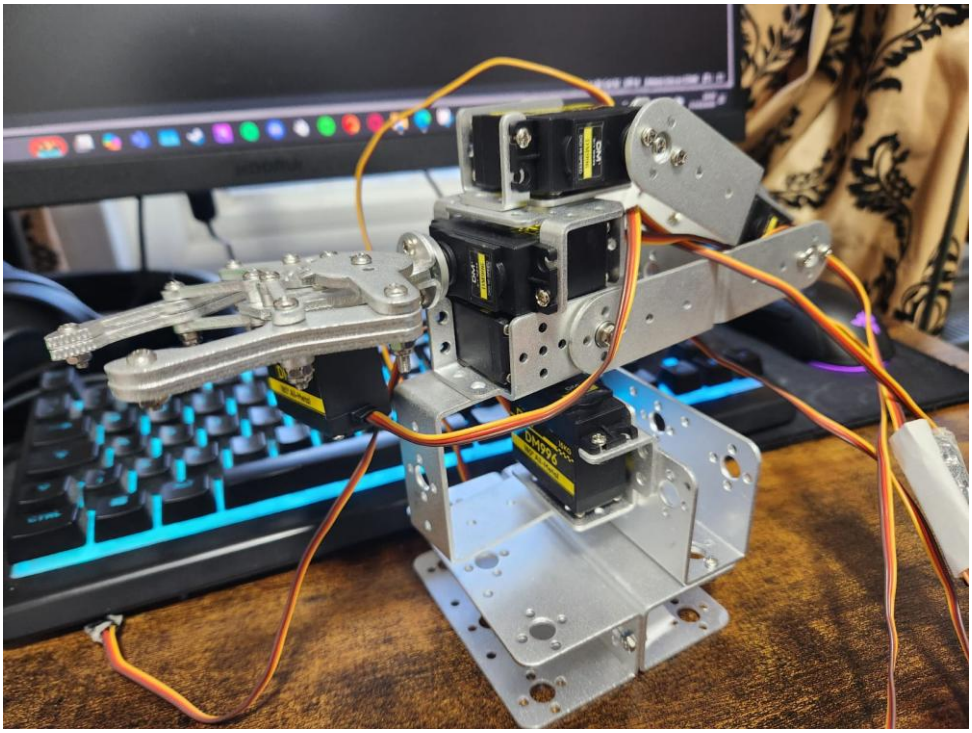


Figure 8: Aluminum based Robotic Arm, PROTOTYPE 1

Based on the initial hand-drawn design, an aluminium robotic arm kit was purchased from eBay for the development of Prototype 1. The mechanical structure of the product closely matched the design specifications outlined in the sketch. However, this prototype ultimately proved to be unsuccessful due to several critical issues.

The primary issue was the excessive weight of the aluminium structure, which exceeded the load capacity of the cost-effective servo motors used in the project. Each link of the arm had approximately the same weight, resulting in poor weight distribution. This was particularly problematic for the shoulder joint, where the corresponding servo motor was required to support the entire upper body of the arm. The motor was unable to manage this load, leading to instability and operational failure.

Additionally, the use of identical servo motors throughout the structure contributed further to the imbalance. All servo motors, including those placed in the upper segments, had the same weight, unnecessarily increasing the load on the lower joints. During testing, the arm exhibited highly unstable motion, and there was a significant risk of permanent damage to the servo motors due to overloading and strain. These issues highlighted the need for a lighter and more appropriately balanced design for the subsequent prototypes.

Figure 9: 3D model of Robotic Arm (Available: 🤖 Arduino based Robot Arm ▪ STEP File for ▪ Cults)

Considering the shortcomings of the initial prototype, an open-source design was adopted for the final version of the project. This design was sourced from the YouTuber *How To Mechatronics*, who provided the model through his official website under the project title "🤖 Arduino based Robot Arm ▪ STEP File for ▪ Cults". The open-source design closely aligns with the specifications outlined in the original hand-drawn sketch, ensuring consistency with the intended structure and functionality.

The STL files associated with the design were submitted to the HAWK Laboratory at the University of Greenwich for 3D printing. This approach allowed for the creation

of a lighter, well-balanced robotic arm that met the project's technical and educational objectives.

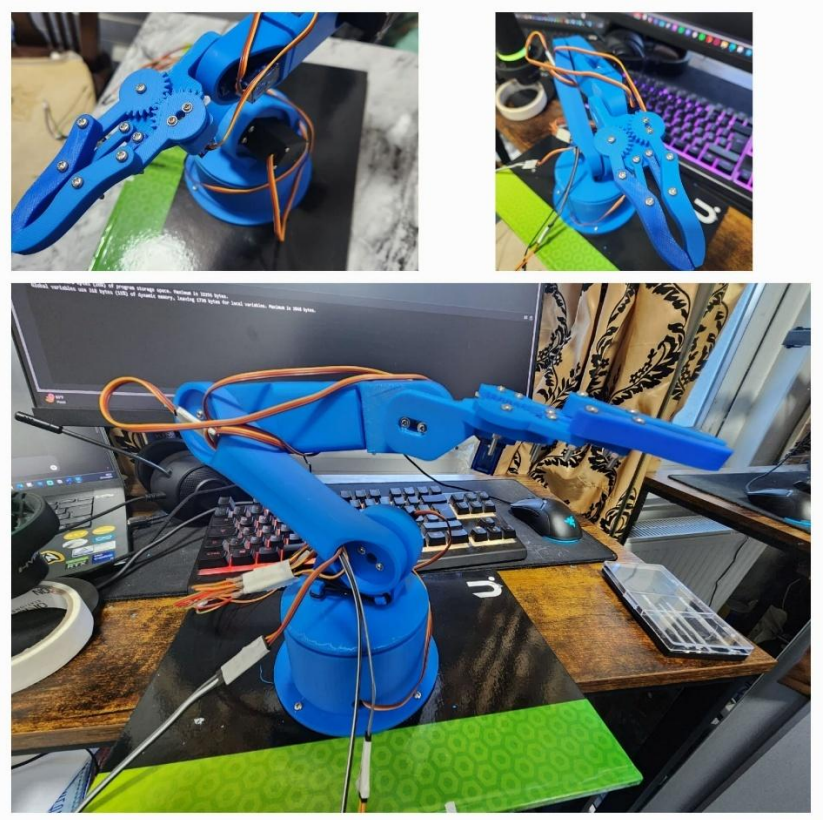


Figure 10: Printed Robotic Arm (Eagle views and Side View)

The 3D-printed robotic arm was selected as the final design for the project, as it aligned perfectly with the initial hand-drawn sketch in terms of structure and functionality. This design utilizes three MG996R servo motors and three SG90 servo motors, optimally distributed to balance performance and weight. The final 3D-printed robotic arm has overall dimensions of 291 mm × 180 mm × 239.8 mm, making it compact and suitable for desktop use. The length of Link 1 (shoulder segment) measures 151 mm, while Link 2 (elbow segment) measures 110 mm. The total weight

of the assembled arm is approximately 250 grams, contributing to its lightweight and efficient performance. This manageable weight, combined with the thoughtfully proportioned link lengths, supports smooth motion and ensures that the mechanical load remains within the capacity of the servo motors.

The MG996R servos, with dimensions of 40.7 mm × 19.7 mm × 42.9 mm, are used to control the base, shoulder, and elbow joints—areas that require higher torque. In contrast, the SG90 servos, which are significantly lighter at 22.2 mm × 11.8 mm × 31 mm, are designated for the wrist and gripper. This configuration ensures that the upper section of the robotic arm remains lightweight compared to the shoulder link, reducing the mechanical load on the shoulder servo.

Additionally, the upper body links and the gripper are intentionally designed to be slimmer and lighter, further contributing to better weight distribution. This design consideration allows the shoulder-mounted MG996R, which provides a thrust of up to 11 kg/cm at 6V, to operate efficiently without strain. As a result, the robotic arm demonstrates improved stability and functionality, making it well-suited for educational and prototype development purposes.

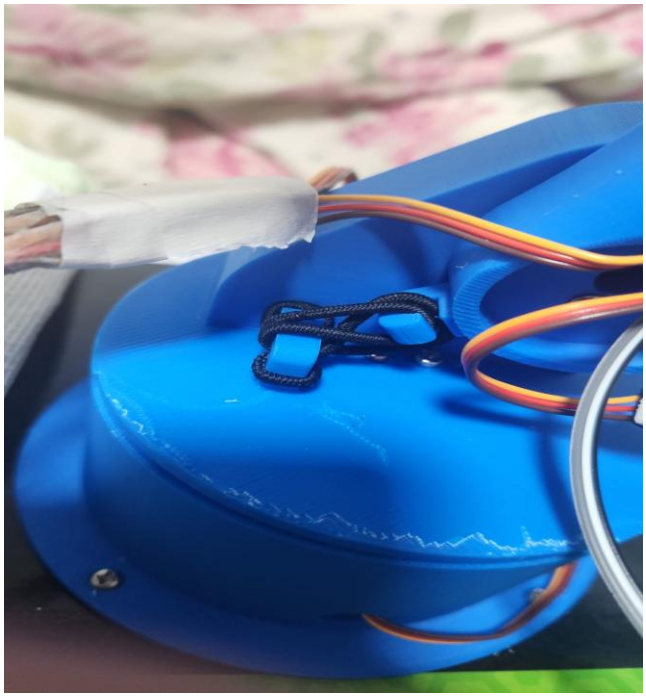


Figure 11: Extra feature (Supporting Band for Shoulder), In the robotic arm design.

An additional advantage of this design is the inclusion of a practical mechanical feature: two integrated hooks—one on the base and another on the shoulder joint—designed specifically for attaching a rubber band. This rubber band acts as a passive support mechanism, assisting the shoulder joint in managing the weight of the upper body.

By partially offsetting the load, the rubber band reduces the torque demand on the shoulder servo motor, allowing it to operate more smoothly and with less strain. This results in improved movement stability and responsiveness. Moreover, the added support serves as a protective measure by minimizing the risk of mechanical stress or potential damage to the shoulder servo in case of sudden movements or

disturbances to the upper body. This thoughtful feature enhances both the reliability and longevity of the robotic arm.

Overall, this design features well-considered weight distribution that enables smooth and stable movement across all joints. The allocation of components and structural elements ensures that the load on each servo motor remains within its thrust capacity, particularly at critical points such as the shoulder joint. This balanced configuration enhances performance, reduces mechanical strain, and contributes to the overall durability and efficiency of the robotic arm during operation.

However, despite its advantages, the 3D-printed design does present a few notable drawbacks. One of the primary issues is the difficulty involved in assembling the components, particularly when securing screws. The process can be quite challenging and time-consuming due to tight spaces and limited accessibility in certain areas.

Additionally, some of the holes intended for attaching servo motors to the links are slightly narrow in diameter, leading to a loose or imprecise fit. This misalignment can affect the overall stability of the joints and compromise the secure attachment of the servo motors. As a result, assembling and handling the robotic arm requires extra caution. Improper handling or excessive force during setup may cause the printed components to crack, become damaged, or detach from the servo motors. These issues highlight the need for careful assembly and potentially minor modifications to improve fit and durability.

1.3.1.2 Assembly and Component Integration:

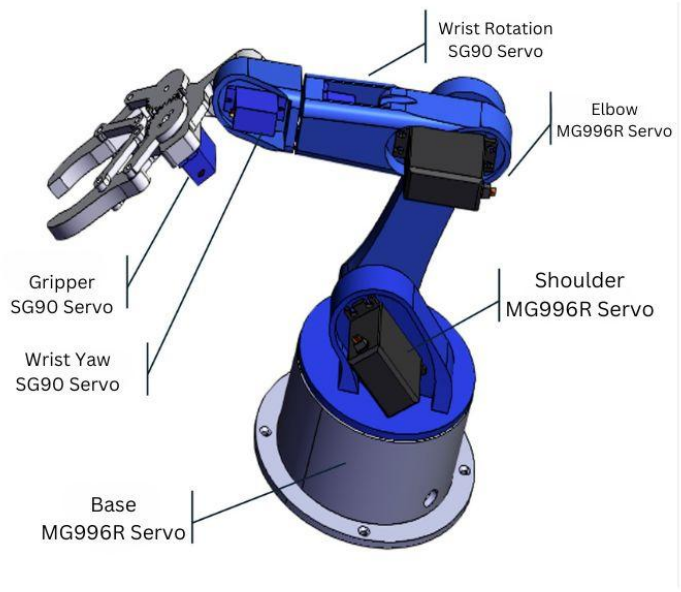


Figure 12: Assembly and Component Integration

The assembly of the 3D-printed robotic arm involved the use of various M3 screws, selected based on the specific requirements of each part. The base, shoulder, and elbow sections include dedicated slots for the MG996R servo motors, which were securely mounted using M3 screws. The servo horns and attaching modules that came with the MG996R servos were used to connect the links to the motor shafts, ensuring a firm and stable joint.

A similar assembly process was followed for the wrist and gripper sections, where SG90 servo motors were installed using M3 screws and their respective

attachment components. To maintain a clean and organized setup, all servo wires were routed through the built-in cable management slots provided in the 3D-printed design. This not only enhances the overall appearance but also reduces the risk of wire entanglement or damage during operation.

1.3.1.3 Circuit Design:

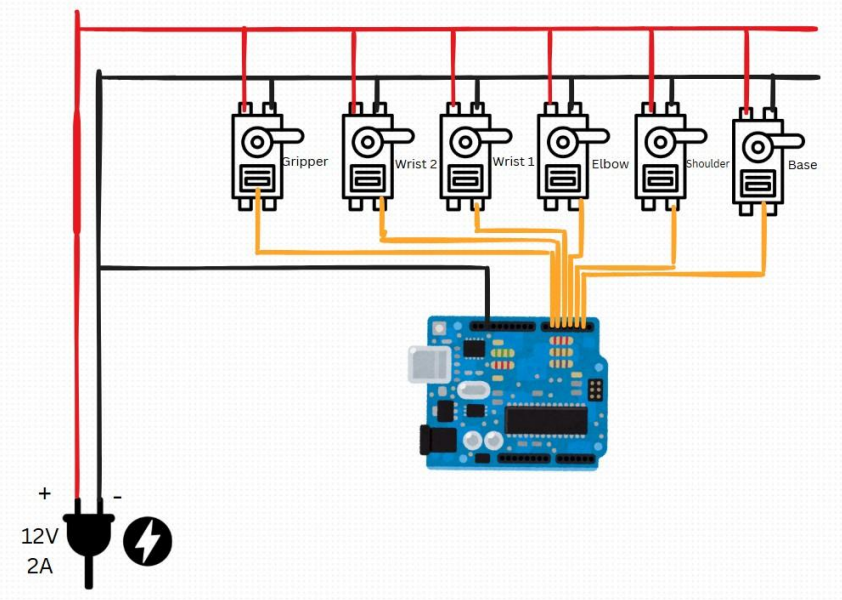


Figure 13: Circuit Layout for servos and microcontroller.

The circuit for the robotic arm was designed in accordance with the layout presented in Figure 13. A 12V power supply is used to power the servo motors. The positive terminal of the power supply is connected to the VCC pins of all the servo motors, while the negative terminal is shared as a common ground. This ground is also connected to the GND pin of the Arduino Uno R3, ensuring a unified ground reference across the entire system.

Each servo motor is connected to the Arduino through its corresponding PWM (Pulse Width Modulation) pin. Specifically, the base, shoulder, elbow, wrist servo 1, wrist servo 2, and gripper servo are connected to PWM pins 2, 3, 4, 5, 6, and 7 respectively. This configuration allows the Arduino to independently control each degree of freedom using PWM signals, enabling precise and synchronized motion across all joints of the robotic arm.

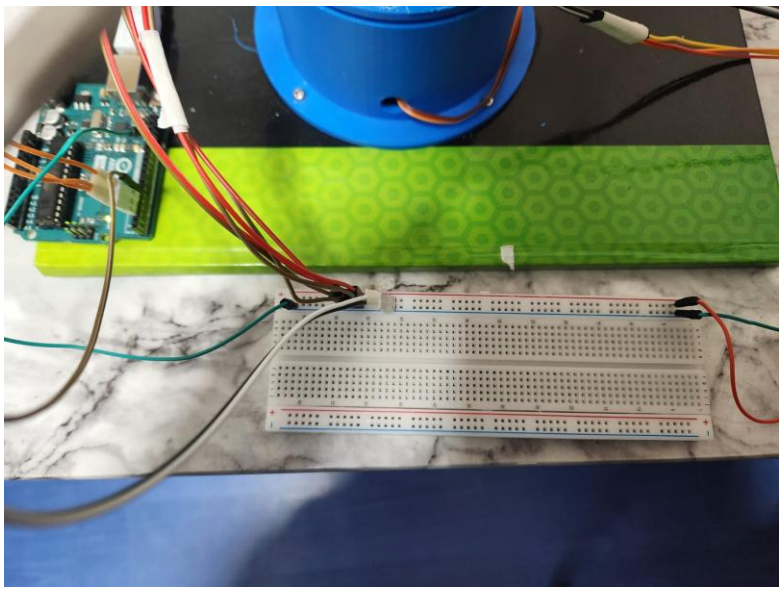


Figure 14: Breadboard Connections

The circuit described above was implemented on a breadboard, as illustrated in Figure 14. This setup allowed for flexible prototyping and easy adjustments during the development phase. All servo motor connections, including power, ground, and PWM signal lines, were organized on the breadboard. The 12V power supply was distributed through the power rails, with the positive rail supplying voltage to all servo motors and the negative rail serving as the common ground.

The Arduino Uno R3 was connected to the breadboard, with its ground pin linked to the shared negative rail to maintain a common reference point. PWM pins 2 through 7 were routed to their respective servo signal lines, enabling control over the base, shoulder, elbow, wrist (2 servos), and gripper movements. This breadboard configuration provided a reliable platform for testing and refining the control logic before transitioning to a more permanent setup.

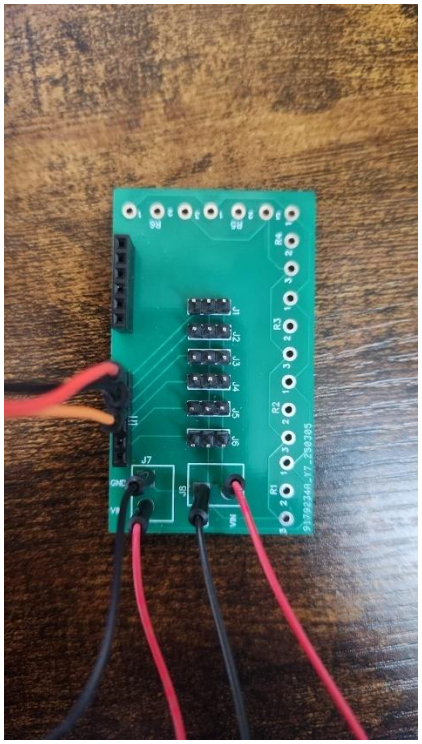


Figure 15: Custom PCB

A custom PCB was developed based on the breadboard prototype to create a more compact and reliable circuit layout. However, due to a connection design error in the PCB, the circuit did not function as intended. This design flaw led to incorrect voltage or signal routing, which ultimately resulted in the failure of the PCB.

Unfortunately, the faulty connections caused damage to two servo motors during testing, making the PCB implementation unsuccessful. This setback highlighted the importance of thorough verification and simulation before finalizing PCB designs, especially when interfacing with sensitive components such as servo motors.

1.3.2 Control System Implementation

1.3.2.1 Arduino Programming:

The control system for the robotic arm was developed using the Arduino Uno R3 and programmed via the Arduino IDE. Two primary libraries were utilized in the code: Servo.h and Math.h. The Servo.h library facilitated the control of the servo motors by allowing PWM signals to be easily translated into specific angular positions, while Math.h provided mathematical functions needed for calculations related to motion and positioning.

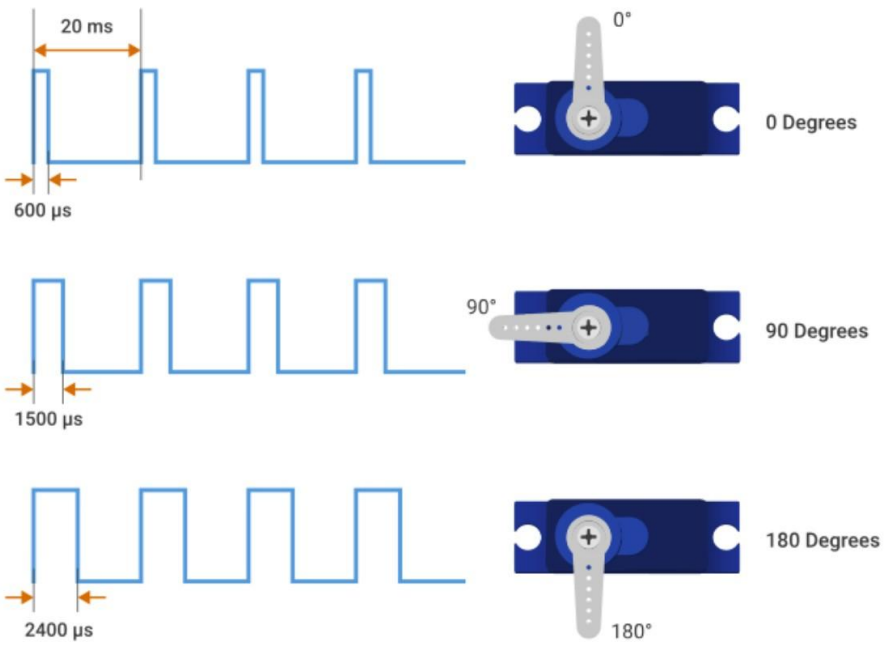


Figure 16: Translation of PWM to Angle Degrees(circuitgeeks,2024)

Each servo motor was connected to a designated PWM pin on the Arduino, enabling precise control of individual joints. The program interprets user-defined commands or input values and converts them into corresponding PWM signals. These signals determine the angle of rotation for each servo, allowing for coordinated and accurate movements of the base, shoulder, elbow, wrist, and gripper joints.

1.3.2.2 Kinematics Implementation:

```
void forwardKinematics(float &X, float &Y, float &Z) {
    float thetaBase = degToRad(baseAngle);
    float theta1 = degToRad(shoulderAngle);
    float theta2 = degToRad(elbowAngle);

    float r = L1 * cos(theta1) + L2 * cos(theta1 + theta2);
    X = r * cos(thetaBase);
    Y = r * sin(thetaBase);
    Z = L1 * sin(theta1) + L2 * sin(theta1 + theta2);
}
```

Figure 17: Implementation of Forward Kinematics in Robotic arm movement

The code shown in Figure 16 implements forward kinematics to determine the position of the end effector in 3D space, based on the angular positions of the base, shoulder, and elbow servo motors. This approach uses the known joint angles and link lengths to calculate the Cartesian coordinates (X, Y, Z) of the end effector relative to the base of the robotic arm.

Accurate link lengths are crucial for these calculations, as they directly affect the computed position. In this implementation, trigonometric functions from the Math.h library are used to resolve the arm's configuration into spatial coordinates. By applying standard forward kinematics equations for a 3-joint planar manipulator, the code enables real-time tracking of the end effector's location, which is essential for motion planning, path following, and verification of movement accuracy. [J. J. Craig, 2017]

Shoulder Link, L1 = 155 mm

Elbow Link, L2 = 110 mm

The arm's planar reach in the shoulder-elbow plane is calculated as:

$$r = L1 \cdot \cos(\theta_1) + L2 \cdot \cos(\theta_1 + \theta_2)$$

This distance r is then projected into the XY-plane based on the rotation of the base:

$$X = \cos(\theta base)$$

$$Y = r \cdot \sin(\theta base)$$

The vertical position Z is determined by the elevation of both arm segments:

$$Z = L1 \cdot \sin(\theta_1) + L2 \cdot \sin(\theta_1 + \theta_2)$$

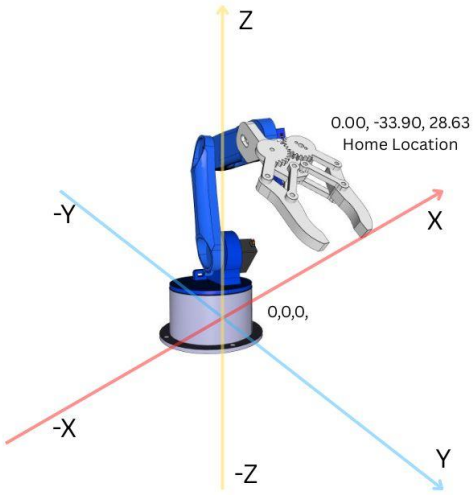
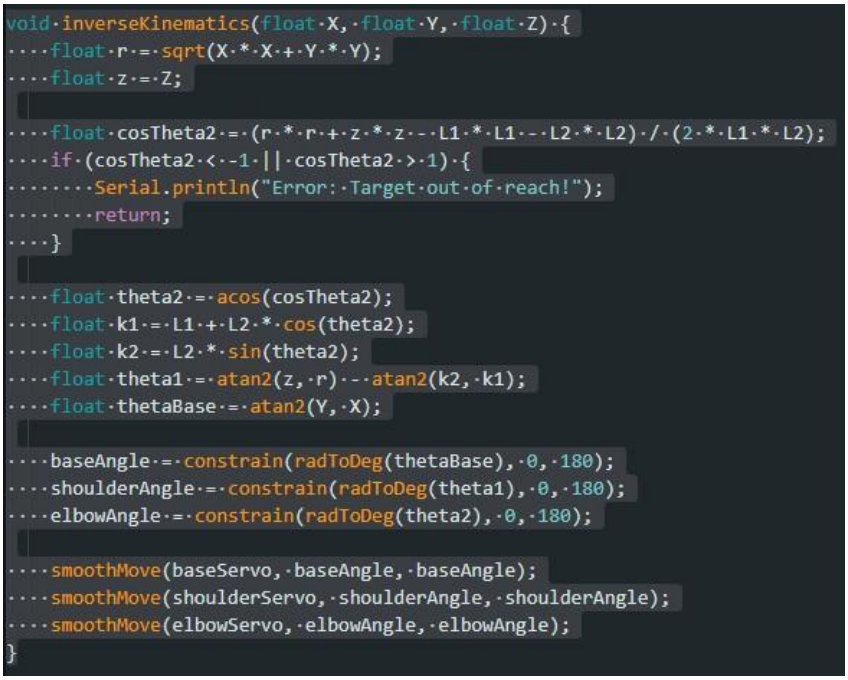


Figure 18: 3D spaces Created Using Forward Kinematics Function

This mathematical calculation effectively establishes a 3D coordinate space for the end effector of the robotic arm. Using the angles of the base, shoulder, and elbow

servos—along with the known lengths of the arm's links—the forward kinematics equations calculate the precise X, Y, and Z coordinates of the end effector.

The base servo angle determines rotation in the horizontal plane (affecting the X and Y coordinates), while the shoulder and elbow angles define the vertical positioning and extension of the arm (affecting the Z coordinate as well as the reach in the X-Y plane). As a result, the program can accurately map the end effector's position in 3D space, enabling precise control for tasks such as object manipulation or path planning. [J. J. Craig, 2017]



```
void inverseKinematics(float X, float Y, float Z) {
    float r = sqrt(X * X + Y * Y);
    float z = Z;

    float cosTheta2 = (r * r + z * z - L1 * L1 - L2 * L2) / (2 * L1 * L2);
    if (cosTheta2 < -1 || cosTheta2 > 1) {
        Serial.println("Error: Target out of reach!");
        return;
    }

    float theta2 = acos(cosTheta2);
    float k1 = L1 + L2 * cos(theta2);
    float k2 = L2 * sin(theta2);
    float theta1 = atan2(z, r) - atan2(k2, k1);
    float thetaBase = atan2(Y, X);

    baseAngle = constrain(radToDeg(thetaBase), 0, 180);
    shoulderAngle = constrain(radToDeg(theta1), 0, 180);
    elbowAngle = constrain(radToDeg(theta2), 0, 180);

    smoothMove(baseServo, baseAngle, baseAngle);
    smoothMove(shoulderServo, shoulderAngle, shoulderAngle);
    smoothMove(elbowServo, elbowAngle, elbowAngle);
}
```

Figure 19: Inverse Kinematics Function for Robotic Arm

Figure 20, represents the inverse kinematics function that is used for Robotic Arm. The math has been represented below.

Using the cosine law, the angle at the elbow joint is calculated as:

$$\cos(\theta_2) = \frac{r^2 + z^2 - L_1^2 - L_2^2}{2L_1L_2}$$

$$\theta_2 = \arccos(\cos\theta_2)$$

Then, the shoulder angle is determined by resolving the triangle formed by the arm links and the line connecting the base to the end effector:

$$k1 = L1 + L2\cos(\theta_2),$$

$$k2 = L2\sin(\theta_2)$$

$$\theta_1 = \text{arc. tan2}\left(\frac{z}{r}\right) - \text{arc. tan2}\left(\frac{k1}{k2}\right)$$

The base angle is simply the angle from the X-axis to the XY projection:

$$\theta base = \text{arc. tan2}\left(\frac{Y}{X}\right)$$

After acquiring the X, Y, and Z coordinates of the end effector through user input or target position, the program utilizes inverse kinematics to calculate the required joint angles for the base, shoulder, and elbow servos. This function determines the appropriate rotational values that each joint must achieve to position the end effector accurately at the specified 3D location.

The base angle is computed using the arctangent of the Y and X coordinates to determine horizontal rotation. The shoulder and elbow angles are calculated using

trigonometric relationships based on the arm's link lengths and the vertical distance to the target point. These angles are then converted to PWM signals and sent to the corresponding servos, enabling precise and coordinated movement of the robotic arm toward the desired position. [J. J. Craig, 2017]

1.3.2.3 Interpolation:

```
void smoothMove(Servo &servo, int &currentAngle, int targetAngle) {
    targetAngle = constrain(targetAngle, 0, 180);
    float stepSize = (targetAngle > currentAngle) ? 0.001 : -0.001;
    int stepsCount = abs(targetAngle - currentAngle) * 2; // Twice the resolution

    for (int i = 0; i < stepsCount; i++) {
        currentAngle += stepSize;
        servo.write(currentAngle);
        delay(10 * delayTime);  // Slightly increased delay for smoothness
    }
    currentAngle = targetAngle;
    servo.write(currentAngle);
}
```

Figure 20: Linear Interpolation used for Smooth Movement

To ensure smooth and controlled motion of the robotic arm, linear interpolation was implemented. This technique allows the servos to transition gradually between angles, avoiding sudden or jerky movements. The interpolation function is integrated into all motion-related functions, ensuring consistency across the entire control system.

The function works by first determining the direction of movement for each servo based on the difference between the current angle and the target angle. It then sets a small step size to define how much the angle should change in each iteration. Using the difference between the current and target angles, the function calculates

the number of incremental steps required for the transition. The servo angles are then updated step-by-step, creating a smooth and visually continuous movement until the final position is reached. This approach significantly improves the precision and realism of the robotic arm's operation. [A. Adept,2016]

1.3.2.4 Motion Control:

```
void manualControl() {
    if (Serial.available() > 0) {
        char command = Serial.read();
        command = toupper(command);

        bool angleUpdated = false;

        switch (command) {
            case 'W': // Decrease Shoulder & Decrease Elbow
                shoulderAngle = constrain(shoulderAngle - 5, 0, 180);
                elbowAngle = constrain(elbowAngle - 5, 0, 180);
                angleUpdated = true;
                break;
            case 'S': // Increase Shoulder & Increase Elbow
                shoulderAngle = constrain(shoulderAngle + 5, 0, 180);
                elbowAngle = constrain(elbowAngle + 5, 0, 180);
                angleUpdated = true;
                break;
            case 'A': // Increase Base
                baseAngle = constrain(baseAngle + 5, 0, 180);
                angleUpdated = true;
                break;
            case 'D': // Decrease Base
                baseAngle = constrain(baseAngle - 5, 0, 180);
                angleUpdated = true;
                break;
```

Figure 21: Manual Control Motion Control Function

The primary motion control mechanism for the robotic arm is implemented through the Manual Control function. This function receives input from the keyboard via the Arduino Serial Monitor and responds by adjusting the joint angles accordingly. It

allows the user to control the movement of the robotic arm in real-time by issuing specific character commands.

Each input corresponds to a predefined case that modifies one or more servo angles, effectively moving the end effector in the desired direction. The angle of each relevant servo motor is incremented or decremented by 5 degrees per input to ensure controlled, stepwise movement.

The motion control function translates these angle changes into real servo movements, enabling directional control over the robotic arm. The corresponding input cases are listed below:

w – Moves the end effector **forward**

Action: Decreases shoulder angle, decreases elbow angle

s – Moves the end effector **backward**

Action: Increases shoulder angle, increases elbow angle

a – Rotates the end effector **left**

Action: Increases base angle

d – Rotates the end effector **right**

Action: Decreases base angle

q – Moves the end effector **up**

Action: Decreases elbow angle

e – Moves the end effector **down**

Action: Increases elbow angle

z – Moves the end effector **up**

Action: Increases shoulder angle

x – Moves the end effector **down**

Action: Decreases shoulder angle

```
void moveTo(float X, float Y, float Z) {
    Serial.print("Moving to X: "); Serial.print(X);
    Serial.print(", Y: "); Serial.print(Y);
    Serial.print(", Z: "); Serial.println(Z);

    inverseKinematics(X, Y, Z);
```

Figure 22: Move to Motion Control Function

The Manual Control function also integrates inverse kinematics to enhance control precision. When the user provides target X, Y, Z coordinates—either directly or through cumulative movement commands—the function calculates the required joint angles for the base, shoulder, and elbow servos to position the end effector accurately in 3D space.

This process involves converting the spatial coordinates into angular values using trigonometric relationships based on the arm's geometry and link lengths. The resulting angles are then used to move the corresponding servos smoothly using linear interpolation. This integration of inverse kinematics ensures that the robotic

arm responds accurately to positional commands and maintains realistic, coordinated movement.

```
void jumpTo(float X, float Y, float Z) {
    Serial.println("Executing jumpTo command...");

    // Move to a safe height first
    float currentX, currentY, currentZ;
    forwardKinematics(currentX, currentY, currentZ);
    moveTo(0.0, -36.54, 28.63);
    delay(3000);
    // Move to final target Z
    moveTo(X, Y, Z);
```

Figure 23: Jump to Motion Control Function

This function was intended to operate similarly to the Move To function, with the added feature of maintaining a safe height while following a smooth trajectory to the target position. The goal was to avoid collisions with objects or the base by first lifting the end effector to a designated height, then moving horizontally, and finally descending to the target coordinate.

However, the implementation did not perform as expected in all scenarios. While a few specific coordinate inputs resulted in successful movements, the majority failed due to inaccuracies in trajectory calculation or inverse kinematics resolution. These issues led to unstable or unreachable arm configurations, causing the end effector to miss the target or behave unpredictably. As a result, the function could not be reliably used in the final demonstration and was considered unsuccessful in its current state.

1.3.3 User Interface

1.3.3.1 Manual Control

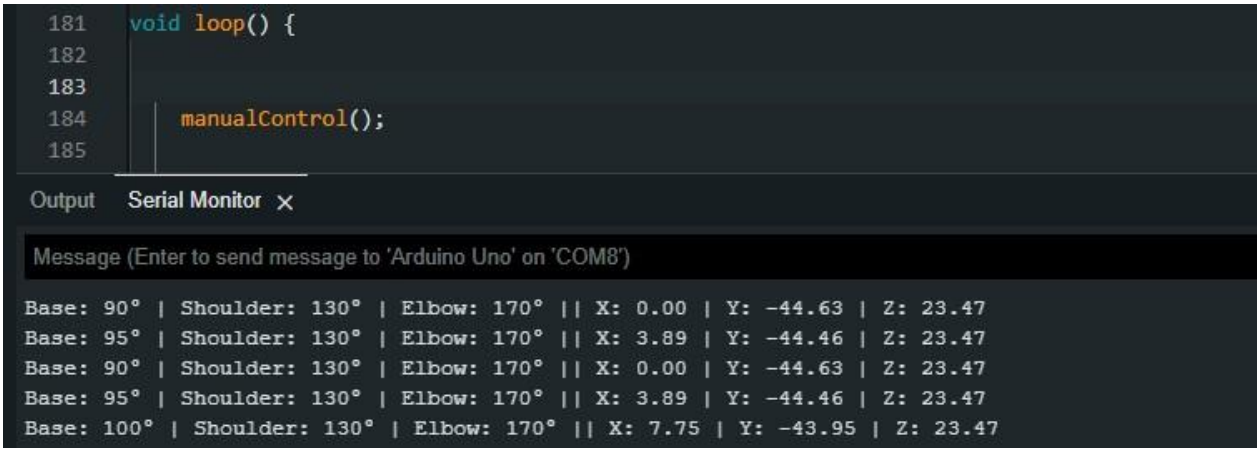


Figure 24: Manual Control

The program begins execution with the **manualControl()** function, which activates the manual operation mode of the robotic arm. The Serial Monitor in the Arduino IDE serves as the interface for controlling the arm's movement. Users can input the designated case keys, as outlined in Section 3.3.2.3, directly into the message field of the Serial Monitor.

Upon receiving an input key, the program adjusts the corresponding joint angles by 5 degrees and moves the end effector accordingly. The Serial Monitor then provides real-time feedback by printing the updated servo angles and the current X, Y, Z coordinates of the end effector. This live feedback allows users to observe the arm's motion and make precise adjustments during operation.

1.3.3.2 Controlling Using Functions

```
180
181  void loop() {
182
183
184      manualControl();
185
186
187      moveTo(5.99, -5.02, 44.32);
188      delay(2000);
189
190      gripServo.write(90);
191      delay(2000);
192      gripServo.write(0);
193      delay(2000);
```

Output Serial Monitor ×

Message (Enter to send message to 'Arduino Uno' on 'COM8')

```
Moving to X: 5.99, Y: -5.02, Z: 44.32
Moving to X: 0.00, Y: -33.90, Z: 28.63
Error: Target out of reach!
Moving to X: -7.81, Y: 0.00, Z: 44.32
Moving to X: 5.99, Y: -5.02, Z: 44.32
Moving to X: 0.00, Y: -33.90, Z: 28.63
```

Figure 25: Moving Using Functions

The end effector can also be positioned using the moveTo() function, which enables direct movement to a specified target coordinate in 3D space. Coordinates can be obtained using the manualControl() function, which allows for precise adjustment and identification of desired X, Y, Z positions. Once the target coordinate is determined, it can be passed into the moveTo() function, as demonstrated in Figure 27, to guide the end effector to that location using inverse kinematics.

For gripper control, a simple servo command is used to open or close the gripper mechanism. The command gripServo.write(angle) sets the position of the gripper servo:

- gripServo.write(0); – Closes the gripper
- gripServo.write(90); – Opens the gripper

This straightforward approach allows easy integration of gripping functionality within both manual and automated routines.

CHAPTER FOUR

RESULTS AND DISCUSSIONS

1.4 Test and Evaluation

1.4.1 Mechanical Test and Design Evaluation

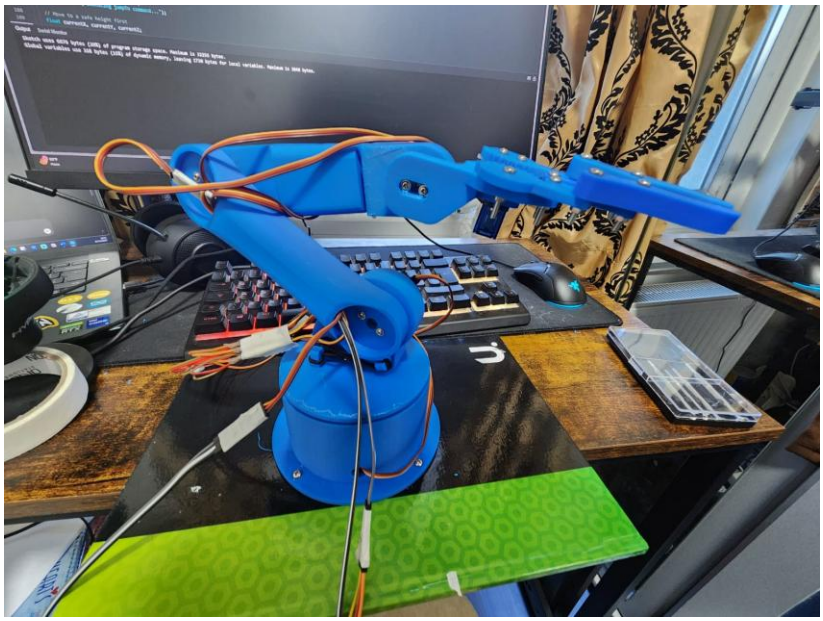


Figure 26: 3D printed Robotic arm

The 3D-printed robotic arm weighs approximately 250 grams, making it lightweight, durable, and highly cost-effective—ideal for educational and prototyping purposes. The overall design meets the project's structural and functional requirements, with proper weight distribution that ensures stability during movement.

Each servo motor operates reliably and can support the weight of the specific link it controls. Notably, the shoulder servo—which bears the weight of the entire upper body—functions effectively within its rated thrust capacity. The robotic arm demonstrates smooth

and coordinated motion, although it is noted that movement could be improved further with higher-quality servos. However, the current servo motors were chosen to maintain a low-cost solution.

Additionally, incorporating capacitors in the circuit could help enhance the stability and smoothness of servo movements by filtering voltage fluctuations and reducing noise during operation.

In alignment with project goals, the end effector is detachable, allowing for easy replacement or integration of alternative tools or attachments. This flexibility increases the arm's adaptability for a wide range of applications and future upgrades.

1.4.2 Movement Test

Each servo motor used in this project was individually tested using a separate program that utilized the basic servo command **Servo.write(angle).** This testing ensured that all servos functioned correctly across their full range of motion, from 0 to 180 degrees. On average, it was observed that each servo takes approximately 1.25 seconds to complete this range, providing a reasonable balance between speed and control.

However, during the development process, two wrist servos were damaged due to incorrect wiring. As a result, these servos were not used in the final implementation of the robotic arm. Fortunately, the wrist servos are optional components primarily associated with the end effector, and their absence did not significantly impact the core functionality

of the project. Additionally, since these servos are modular and easily replaceable, they can be reinstalled in future iterations or upgrades of the system if needed

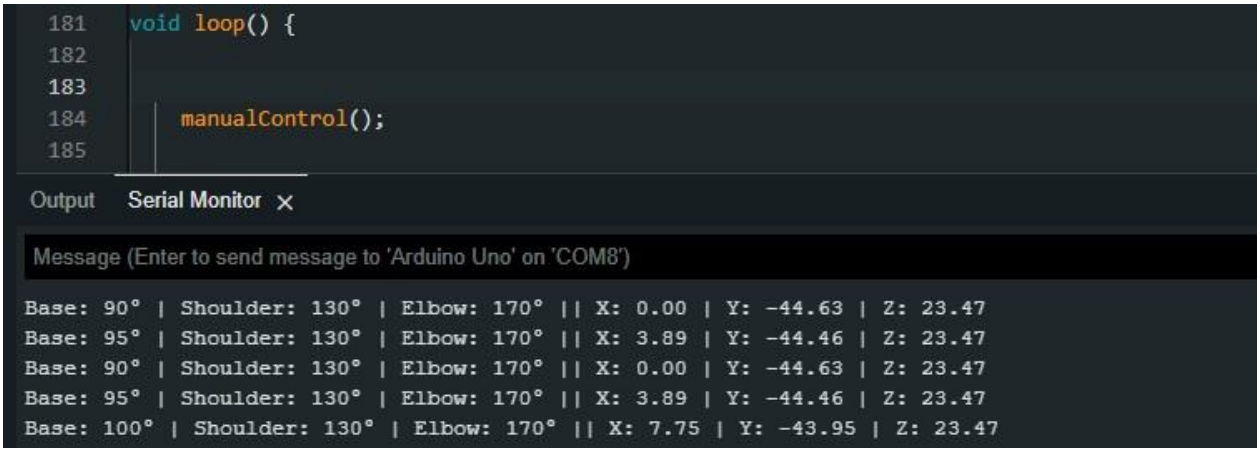


Figure 27: Manual Control Function

The **manualControl()** function performs as described in Section 3.3.3.1, allowing the user to control the movement of the end effector through keyboard inputs. Each key press updates the servo angles and recalculates the corresponding X, Y, Z coordinates, which are displayed in the Serial Monitor for real-time feedback.

While the function generally operates as intended, it occasionally fails to respond to input commands. These inconsistencies are primarily due to unstable current input and limitations in the step size, which can cause the servos to miss small movements or respond irregularly. Additionally, the coordinate calculations—although mostly reliable—are not always precise, with an observed accuracy rate of approximately 90%. This margin of error is acceptable for general control and demonstration purposes but highlights areas for potential refinement in future iterations, such as improved power management and more precise kinematic calculations.

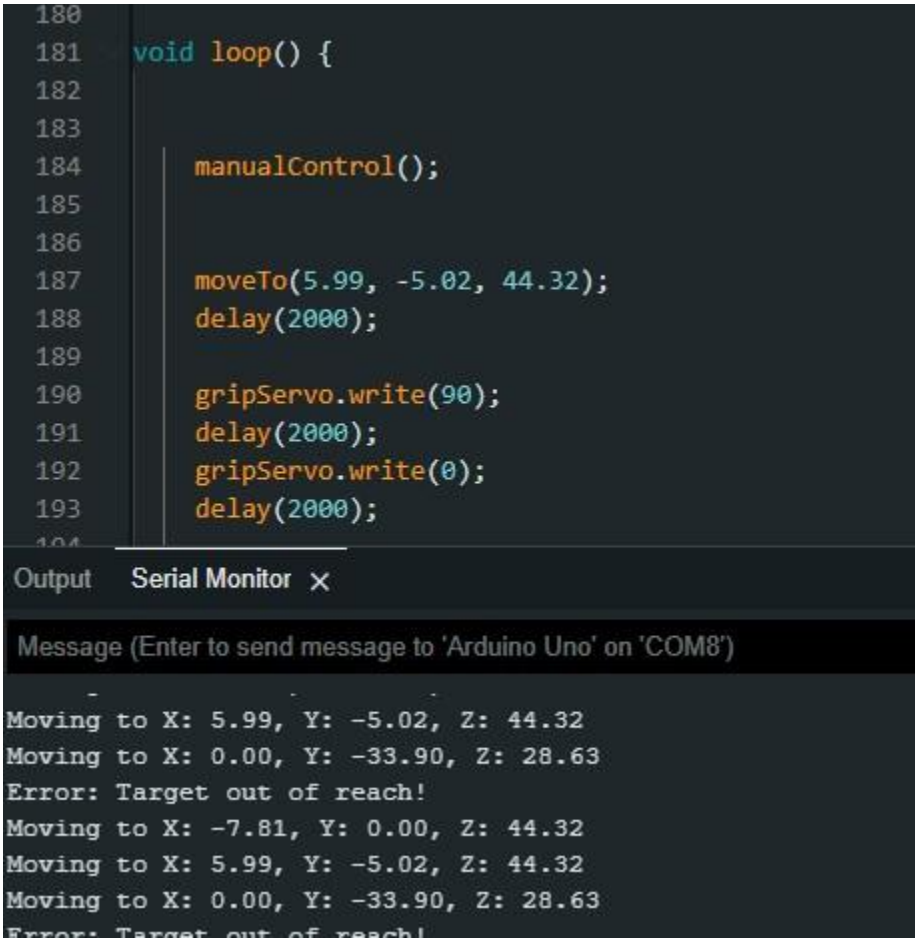


Figure 28: Move To Control Function

The **moveTo()** control function utilizes the X, Y, Z coordinates generated by the manualControl() function to move the end effector to a specific target position in 3D space. The function applies inverse kinematics to convert these coordinates into the corresponding joint angles for the base, shoulder, and elbow servos.

In practice, the **moveTo()** function performs reliably in approximately 75% of cases, accurately moving the end effector to the intended location. However, in about 20% of the cases, the end effector deviates from the target path by 10–15 degrees, likely due to rounding errors, limitations in resolution, or inconsistencies in angle calculation. In rare

instances—about 5% of the time—the motion is reversed or significantly inaccurate, resulting from miscalculations in inverse kinematics or ambiguous coordinate values.

These issues suggest that while the **moveTo()** function is functional and effective for basic demonstrations, further refinement in the mathematical model and error handling is necessary to achieve greater accuracy and reliability in real-world applications

The **jumpTo()** function was intended to provide rapid movement of the end effector to a specified coordinate without following a gradual or interpolated trajectory. However, this function failed during testing and was not included in the final implementation due to its unpredictable and irregular behaviour.

The function caused sudden and unstable servo movements, which not only disrupted the intended motion but also led to minor structural damage to the robotic arm. As a result, no visual or performance data has been documented for the **jumpTo()** function, and it was deemed unsuitable for safe and reliable operation. This outcome highlights the importance of incorporating smooth motion planning and power-safe control logic when dealing with multi-joint robotic systems.

1.5 Educational Effective

This project was specifically designed to be educational and adaptable across **three academic levels**—school, college, and university—making it a versatile tool for introducing and exploring the field of robotics.

School Level

At the school level, students can interact with the robotic arm using the **manualControl()** function under the guidance of an instructor. This function was intentionally designed with simplicity in mind, mimicking the common WASD video game control scheme, making it intuitive and engaging for younger learners. The visibility of the servo motors and their movements provides a hands-on opportunity for students to observe the fundamental mechanics of robotic motion. Through this interaction, students are introduced to the basic concepts of robotics in an accessible and enjoyable way.

College Level

College students, who typically have a foundational understanding of C++ programming, can explore more advanced control features such as the **moveTo()** function. Using these functions, they can develop and implement simple projects like box stacking or object manipulation, which reinforce concepts in motion control, programming logic, and practical problem-solving. These activities serve as a bridge between theoretical learning and hands-on implementation, strengthening their technical skillset.

University Level

At the university level, students can delve deeper into the project by studying the underlying code architecture and exploring the application of forward and inverse kinematics in robotic systems. This level encourages experimentation, such as designing custom robotic arms and developing new control functions tailored to those designs, given that kinematic models differ across structures. Additionally, students can expand the project by integrating sensors (e.g., ultrasonic, IR, or vision systems) to enable

autonomous behaviour, opening the door to more complex and research-oriented applications.

Overall, this project serves as a scalable educational platform, introducing fundamental robotics concepts to beginners while offering advanced learners the opportunity to innovate, experiment, and build upon a functional robotic system.

1.6 Cost of Product

Table 1: Bill of Materials

Component	Quantity	Unit Price (GBP)	Total Price (GBP)
Arduino Uno R3	1	10	10
MG996R Servo Motors	3	5.5	16.5
SG90 Servo Motors	3	2.5	7.5
3D Printing (PLA/ABS - HAWK Lab)	1 Set	12	12
Breadboard	1	3	3
Jumper Wires (Male-Female)	1 Set	2	2

Power Supply (12V 2A Adapter)		1	8	8
Custom PCB (JLCPCB)		1	5	5
M3 Screws and Mounting Hardware	1 Set		3	3
Damaged/Replaced Servo Motors		2	5	10

Total: £77

1.7 Performance Summary

A video has been uploaded with demonstrating the movement function in the link below:

https://youtu.be/6I-tcn5AtSY

Copy and Paste the link in browser if the link is unclickable.

Table 2: Performance Summary

Performance Metric	Result
Full Rotation Time	1.25 Seconds
Task Completion Time (average)	5-7 seconds
Accuracy of Positioning	±10 degrees
Calculation and Output	90% success rate
Student Engagement and Learning	80% confidence in concepts
Durability (Continuous Use)	Minor damage during experiments.

CHAPTER FIVE

CONCLUSIONS AND FUTURE WORK

1.8 Key Findings

1.8.1 Design Functionality and Educational Value

The robotic arm's design turned out to be both practical and ideal for teaching. In addition to offering sufficient versatility to teach fundamental ideas like kinematics, motion control, and inverse kinematics, the 3-DOF architecture allowed students to interact with a tolerable degree of complexity. The mechanical structure's ease assembly and modification—made possible by its lightweight and robust 3D-printed components—is essential for an instructional tool that encourages students to experiment and explore the system. Furthermore, by enabling students to engage with a variety of tools and tasks, such as drawing and object handling, the modular design—which includes the removable end effector—proved beneficial in augmenting the educational experience.

1.8.2 Performance Results

The robotic arm demonstrated strong performance in both 3D positioning and basic object manipulation, achieving a position accuracy of ±10 degrees—suitable for most educational tasks. Thanks to linear interpolation, movements between target points were smooth and stable. The arm completed a full rotation in 1.25 seconds and typically reached target positions within 5–7 seconds. User feedback highlighted the simplicity and intuitiveness of the manual control system, which allowed students to quickly understand

and operate the arm via the Serial Monitor or graphical interface. Overall, the system provided a hands-on, engaging learning experience that effectively introduced core robotics concepts.

1.8.3 Challenges Faced and solutions

One major challenge was ensuring the durability of the 3D-printed components, especially under repetitive motion. Materials like PLA and ABS were chosen for their strength, but long-term testing is still recommended. Another challenge was achieving smooth servo control. Early attempts with a PID control algorithm led to oscillations, which were resolved by tuning the parameters for smoother and more stable movement. Additionally, some students with no programming background found the manual control setup difficult to understand. To improve accessibility, clear documentation and tutorials were created, making it easier for users to operate the robotic arm and engage with the system.

1.9 Conclusion

The project successfully achieved its primary objective of designing and constructing a 3-DOF robotic arm for educational purposes. The arm demonstrated reliable functionality, smooth motion, and ease of use, making it an effective tool for teaching robotics, programming, and control concepts. Its modular and scalable design accommodates a wide range of student skill levels, from beginners to advanced learners. The system fosters hands-on learning and encourages engagement, providing practical experience in real-world applications. Its performance in key areas such as motion accuracy and user

interaction confirms its suitability for integration into school and university STEM curricula, supporting interactive and project-based education.

1.10 Future Work

While the current design and implementation of the 3-DOF robotic arm are effective for educational purposes, several areas offer potential for enhancement and future development. A key improvement would be increasing the degrees of freedom—for example, upgrading to a 6-DOF configuration. This would enable the arm to perform more complex tasks and provide students with opportunities to explore advanced concepts in kinematics and motion planning.

Further refinement of the control algorithms could also improve responsiveness and accuracy. While the existing PID system was functional, implementing advanced techniques such as adaptive control or model predictive control (MPC) would enhance performance, especially in precision-driven applications.

Integrating AI and machine learning opens possibilities for autonomous operations, such as object recognition, adaptive grip strength, and task learning. This would introduce students to real-world applications of intelligent robotics.

Enhancing the gripper design and introducing more interchangeable end effectors would increase the system's versatility for different use cases, such as drawing, writing, or small-scale assembly. Additionally, using stronger, more durable materials or transitioning to injection-moulded parts would improve long-term usability in educational settings.

Finally, integration with platforms like Raspberry Pi or cloud-based systems could support remote collaboration and advanced data analysis, broadening the project's educational impact.

References

Tajammul Pangarkar., 2025. Robot Statistics 2025 by Technology, Machine, Uses. Growth Trends in Robotics and Automation. Available: Robot Statistics and Facts (2025).

Frey, Carl Benedict, and Michael Osborne. “THE FUTURE OF EMPLOYMENT: HOW SUSCEPTIBLE ARE JOBS TO COMPUTERISATION.” 17 Sept. 2013, pp. 1–72.

Raman, Amy Bernstein Anand. “The Great Decoupling: An Interview with Erik Brynjolfsson and Andrew McAfee.” Harvard Business Review, 13 Mar. 2017, hbr.org/2015/06/the-great-decoupling.

S. Jadhav, S. Iyer and K. Arya, "Enhancing Problem-Solving in Robotics Education: Analysis of Scaffolds and Interactions," *2024 IEEE International Conference on Advanced Learning Technologies (ICALT)*, Nicosia, North Cyprus, Cyprus, 2024, pp. 121-123, doi: 10.1109/ICALT61570.2024.00041.

Donald L. Pieper, The kinematics of manipulators under computer control. PhD thesis, Stanford University, Department of Mechanical Engineering, October 24, 1968.

Butterfield, Andrew J.; Szymanski

, John, eds. (2018). "A Dictionary of Electronics and Electrical Engineering". Oxford

Reference. doi:10.1093/acref/9780198725725.001.0001. ISBN 978-0-19-872572-5.

Ben Moshe, Nir (2025). "A Simple Solution for the Inverse Distance Weighting Interpolation (IDW) Clustering Problem". Sci. **7** (1): 30. doi:10.3390/sci7010030.

Anjaneyulu, J. K. S. S. R., & Kumar, M. R. R. S. (2018). A survey of control techniques for robotic manipulators. *Journal of Robotics and Automation*.

Hwang, T. W., & Kim, S. W. (2019). Design of a 6-DOF robotic arm for educational purposes. *International Journal of Advanced Robotic Systems*.

Ismail, F. A., & Khamis, A. M. (2017). PID control for educational robotic arm using Arduino. *International Journal of Electrical Engineering Education*.

Khan, R. M. M. A., Khan, M. M. A., & Alam, S. S. W. S. (2019). The role of robotics in enhancing STEM education. *International Journal of Engineering Education*.

Martinez, R. J. R., & Reyes, R. T. (2017). Building a low-cost educational robot using Raspberry Pi and Arduino. *IEEE Latin America Transactions*.

McComb, G. D., & Phillips, T. S. (2018). Robotics as a tool to motivate students in STEM education. *Educational Technology Research and Development*.

Rajasekaran, K. R., & Srinivasan, S. (2019). Design and development of a low-cost educational robotic arm for teaching robotics. *IEEE Access*.

Santos, A. E. W. M., & Lopes, M. P. S. (2018). Overcoming barriers to robotics education in schools: Insights from the field. *International Journal of Engineering Education*.

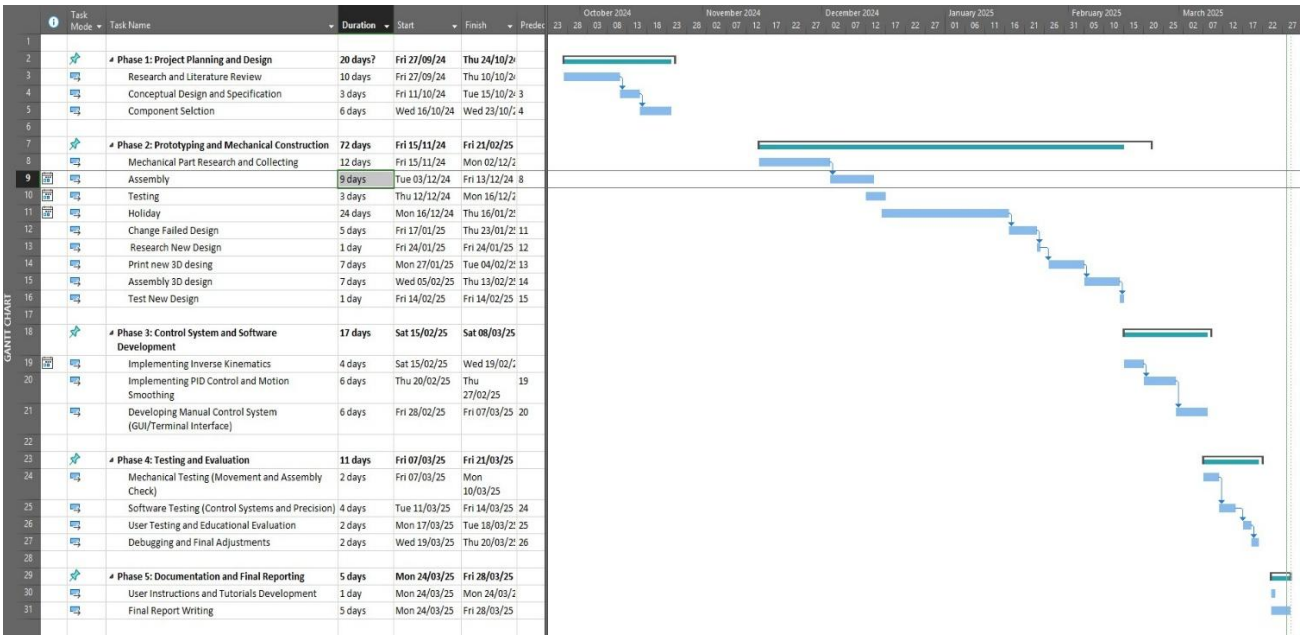
Tan, T. V. D. B., & Toh, C. M. Y. (2021). Challenges in educational robotics: A survey of obstacles and opportunities in the classroom. *Journal of Educational Technology Systems*.

Circuit geeks. Arduino Servo Motor – Complete guide, 2024, available at: Arduino Servo Motors - Complete Guide

J. J. Craig, *Introduction to Robotics: Mechanics and Control*, 4th ed., Pearson, 2017.

A. Adept, *Robotics Programming: Applications and Control*, Springer, 2016.

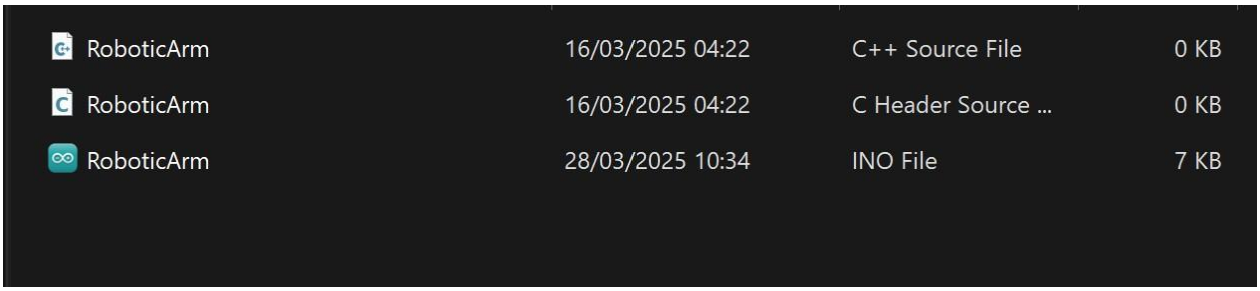
Appendix A: Gantt Chart



	Task Mode	Task Name	Duration	Start	Finish	Predec
1						
2		**Phase 1: Project Planning and Design**	**20 days?**	**Fri 27/09/24**	**Thu 24/10/2**	
3		Research and Literature Review	10 days	Fri 27/09/24	Thu 10/10/2	
4		Conceptual Design and Specification	3 days	Fri 11/10/24	Tue 15/10/2	3
5		Component Selction	6 days	Wed 16/10/24	Wed 23/10/2	4
6						
7		**Phase 2: Prototyping and Mechanical Construction**	**72 days**	**Fri 15/11/24**	**Fri 21/02/25**	
8		Mechanical Part Research and Collecting	12 days	Fri 15/11/24	Mon 02/12/2	
9		Assembly	9 days	Tue 03/12/24	Fri 13/12/24	8
10		Testing	3 days	Thu 12/12/24	Mon 16/12/2	
11		Holiday	24 days	Mon 16/12/24	Thu 16/01/2	
12		Change Failed Design	5 days	Fri 17/01/25	Thu 23/01/2	11
13		Research New Design	1 day	Fri 24/01/25	Fri 24/01/25	12
14		Print new 3D desing	7 days	Mon 27/01/25	Tue 04/02/2	13
15		Assembly 3D design	7 days	Wed 05/02/25	Thu 13/02/2	14
16		Test New Design	1 day	Fri 14/02/25	Fri 14/02/25	15
17						
18		**Phase 3: Control System and Software Development**	**17 days**	**Sat 15/02/25**	**Sat 08/03/25**	
19		Implementing Inverse Kinematics	4 days	Sat 15/02/25	Wed 19/02/2	
20		Implementing PID Control and Motion Smoothing	6 days	Thu 20/02/25	Thu 27/02/25	19
21		Developing Manual Control System (GUI/Terminal Interface)	6 days	Fri 28/02/25	Fri 07/03/25	20
22						
23		**Phase 4: Testing and Evaluation**	**11 days**	**Fri 07/03/25**	**Fri 21/03/25**	
24		Mechanical Testing (Movement and Assembly Check)	2 days	Fri 07/03/25	Mon 10/03/25	
25		Software Testing (Control Systems and Precision)	4 days	Tue 11/03/25	Fri 14/03/25	24
26		User Testing and Educational Evaluation	2 days	Mon 17/03/25	Tue 18/03/2	25
27		Debugging and Final Adjustments	2 days	Wed 19/03/25	Thu 20/03/2	26
28						
29		**Phase 5: Documentation and Final Reporting**	**5 days**	**Mon 24/03/25**	**Fri 28/03/25**	
30		User Instructions and Tutorials Development	1 day	Mon 24/03/25	Mon 24/03/2	
31		Final Report Writing	5 days	Mon 24/03/25	Fri 28/03/25	

Appendix B: User Instructions

1.11 Opening the Robotic Arm Code



RoboticArm	16/03/2025 04:22	C++ Source File	0 KB
RoboticArm	16/03/2025 04:22	C Header Source ...	0 KB
RoboticArm	28/03/2025 10:34	INO File	7 KB

Figure 29: INO File

1. **Navigate to the project folder** containing the robotic arm files on your computer.
2. Look for the file named **RoboticArm.ino** (or similar).

3. **Double-click** the .ino file, or open it directly from the **Arduino IDE** by selecting:
   File → Open → [Your Folder] → RoboticArm.ino
4. Once opened, verify that all tabs and libraries are properly loaded before uploading to the Arduino board.

1.12 Set Up

```
Servo baseServo, shoulderServo, elbowServo, gripServo;

// Set Home Positions
int baseAngle = 90;
int shoulderAngle = 130;
int elbowAngle = 170;
int gripAngle = 90;

//Set up Pins
const int basePin = 2;
const int shoulderPin = 3;
const int elbowPin = 4;
const int gripPin = 5;
```

Figure 30: Pin Set Up

1. Ensure all servo motors are properly connected to their assigned PWM pins as per the configuration. If additional servos such as wrist or gripper are used, connect them to the appropriate pins and update the code accordingly to include them.

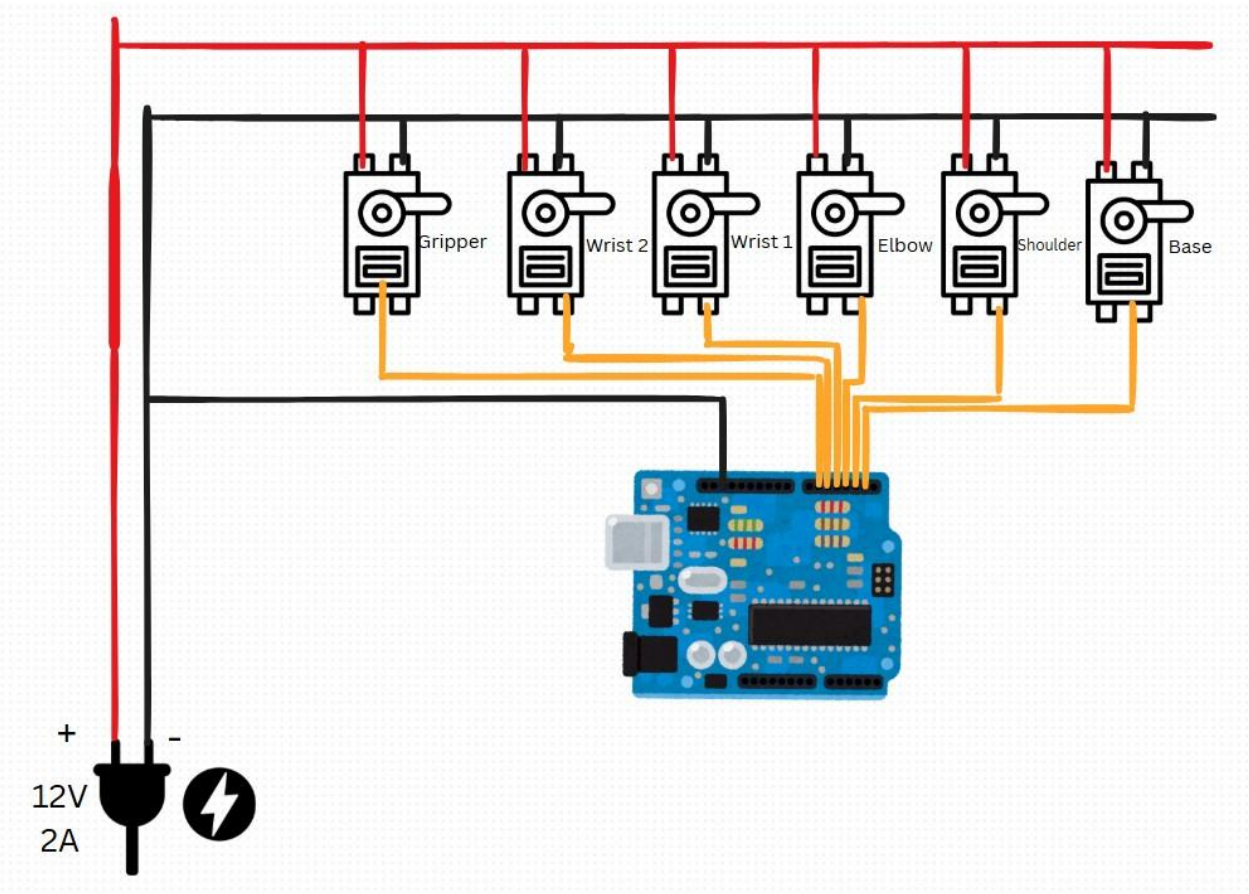


Figure 31: Circuit Diagram

2. Connect the servo motors according to the principles outlined in the circuit diagram above. Ensure each servo is connected to its designated PWM pin, with the power and ground lines properly aligned to the power supply and shared ground.

Figure 32: First Control Function

3. Unless you intend to modify the kinematic functions or add new functionalities, scroll to the end of the code where the `void loop()` function begins. To enable manual control of the robotic arm, call the `manualControl();` function inside the `loop()` block.

1.13 Manual Control Mode

1. Open the **Serial Monitor** in Arduino IDE (set baud rate: 115200).
2. Use the following keyboard keys to control the end effector manually:

Table 3: Manual Control Keys and Functions

Key	Action
w	Move Forward (↓ Shoulder, ↓ Elbow)
s	Move Backward (↑ Shoulder, ↑ Elbow)
a	Rotate Left (↑ Base)
d	Rotate Right (↓ Base)
q	Move Up (↓ Elbow)
e	Move Down (↑ Elbow)
z	Move Up (↑ Shoulder)
x	Move Down (↓ Shoulder)

3. Servo angles and end-effector coordinates will be displayed in the Serial Monitor.
4. To move the arm in multiple steps, input the key multiple time. For example; instead of entering “a” which will rotate the arm to left by 5 degrees, enter “aaaaa” resulting the rotation of robotic arm to the left side by 25 degrees.

```
181  void loop() {
182
183
184      manualControl();
185
```

Output Serial Monitor ×

Message (Enter to send message to 'Arduino Uno' on 'COM8')

```
Base: 90° | Shoulder: 130° | Elbow: 170° || X: 0.00 | Y: -44.63 | Z: 23.47
Base: 95° | Shoulder: 130° | Elbow: 170° || X: 3.89 | Y: -44.46 | Z: 23.47
Base: 90° | Shoulder: 130° | Elbow: 170° || X: 0.00 | Y: -44.63 | Z: 23.47
Base: 95° | Shoulder: 130° | Elbow: 170° || X: 3.89 | Y: -44.46 | Z: 23.47
Base: 100° | Shoulder: 130° | Elbow: 170° || X: 7.75 | Y: -43.95 | Z: 23.47
```

Figure 33: Manual Control Output

1.14 Move To Coordinates

1. Use the **manualControl()** function to move the arm and record the desired **X, Y, Z** coordinates.

```
moveTo(5.99, -5.02, 44.32);
delay(2000);
```

Figure 34: Move to Function

2. Pass these coordinates to the **moveTo(x, y, z)** function to move the arm directly to that position using inverse kinematics.

1.15 Gripper Control

```
gripServo.write(90);
delay(2000);
gripServo.write(0);
delay(2000);
```

Figure 35: Gripper Control Function

To operate the gripper:

- **Open Gripper:** gripServo.write(90);
- **Close Gripper:** gripServo.write(0);

This can be added to the main code or triggered manually in the IDE.

1.16 Safety and Handling

- Always handle the arm carefully, especially when powered.
- Avoid forcing the joints manually to prevent servo damage.
- Power off the system before making mechanical adjustments.
- Ensure all connections are secure to avoid loose contacts or servo jittering.

Appendix C: Flow Chart

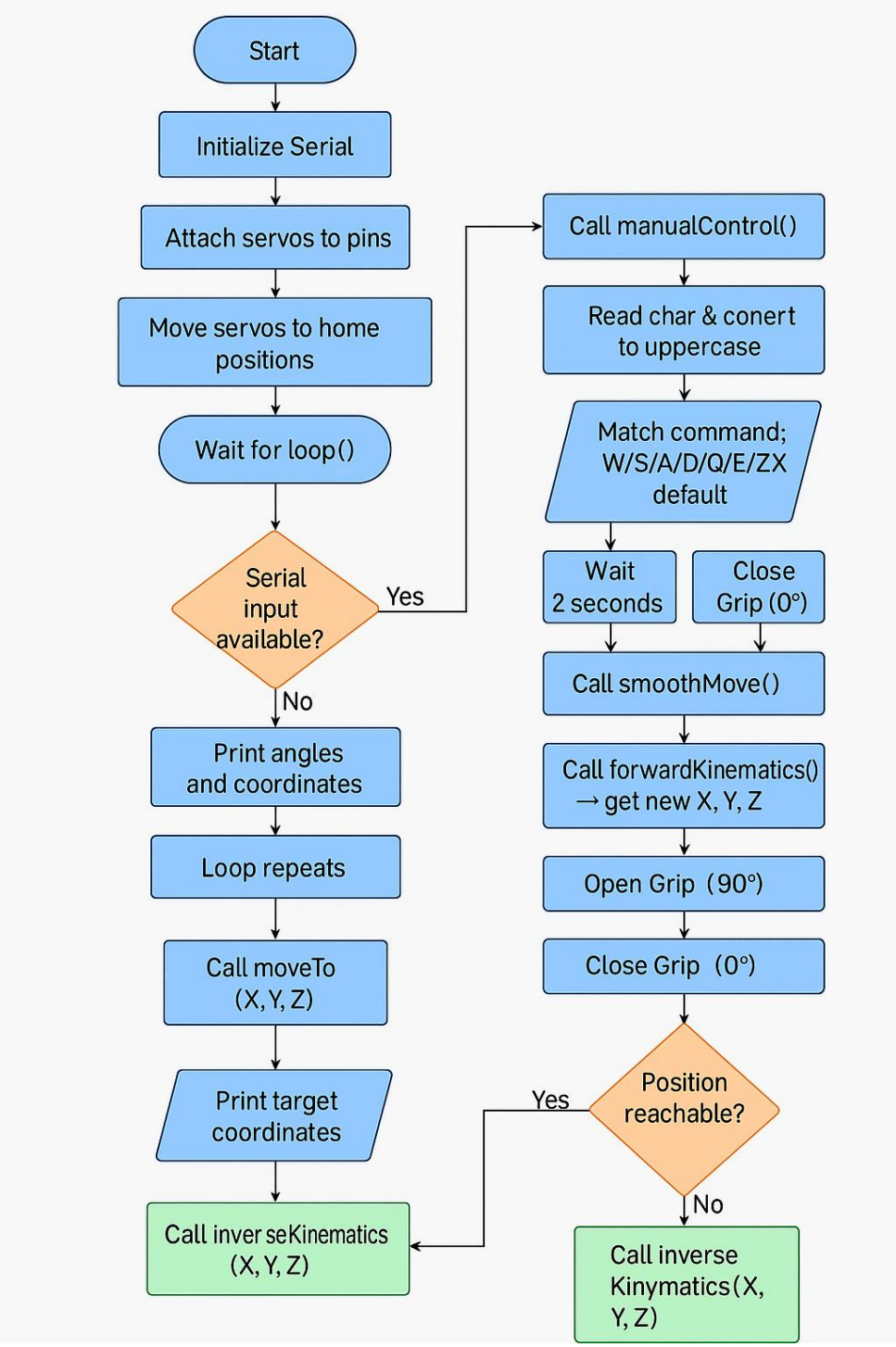


Figure 36: Flow Chart

Appendix D: Full Code

```
#include <Servo.h>
#include <math.h>

Servo baseServo, shoulderServo, elbowServo, gripServo;

// Set Home Positions
int baseAngle = 90;
int shoulderAngle = 130;
int elbowAngle = 170;
int gripAngle = 90;

//Set up Pins
const int basePin = 2;
const int shoulderPin = 3;
const int elbowPin = 4;
const int gripPin = 5;

// Link lengths
const float L1 = 155.0;  // Shoulder to elbow
const float L2 = 110.0;  // Elbow to wrist

// Interpolation settings
const int delayTime = 1000;

//Do not Edit Below Unless Necessary
void setup() {
    Serial.begin(115200);

    baseServo.attach(basePin);
    shoulderServo.attach(shoulderPin);
    elbowServo.attach(elbowPin);
    gripServo.attach(gripPin);

    baseServo.write(baseAngle);
    shoulderServo.write(shoulderAngle);
    elbowServo.write(elbowAngle);
    gripServo.write(gripAngle);
}

```

```
39
40  //Linear Interpolation
41  void smoothMove(Servo &servo, int &currentAngle, int targetAngle) {
42      targetAngle = constrain(targetAngle, 0, 180);
43      float stepSize = (targetAngle > currentAngle) ? 0.001 : -0.001;
44      int stepsCount = abs(targetAngle - currentAngle) * 2; // Twice the resolution
45
46      for (int i = 0; i < stepsCount; i++) {
47          currentAngle += stepSize;
48          servo.write(currentAngle);
49          delay(10 * delayTime);  // Slightly increased delay for smoothness
50      }
51      currentAngle = targetAngle;
52      servo.write(currentAngle);
53  }
54
55  //Conversion of degrees to radians
56  float degToRad(float degrees) {
57      return degrees * (PI / 180.0);
58  }
59
60  //Conversion of radias to degrees
61  float radToDeg(float radians) {
62      return radians * (180.0 / PI);
63  }
64
65
66  //Forward Kinematics Function
67  void forwardKinematics(float &X, float &Y, float &Z) {
68      float thetaBase = degToRad(baseAngle);
69      float theta1 = degToRad(shoulderAngle);
70      float theta2 = degToRad(elbowAngle);
71
72      float r = L1 * cos(theta1) + L2 * cos(theta1 + theta2);
73      X = r * cos(thetaBase);
74      Y = r * sin(thetaBase);
75      Z = L1 * sin(theta1) + L2 * sin(theta1 + theta2);
76  }
77
```

```
void inverseKinematics(float X, float Y, float Z) {
    float r = sqrt(X * X + Y * Y);
    float z = Z;

    float cosTheta2 = (r * r + z * z - L1 * L1 - L2 * L2) / (2 * L1 * L2);
    if (cosTheta2 < -1 || cosTheta2 > 1) {
        Serial.println("Error: Target out of reach!");
        return;
    }

    float theta2 = acos(cosTheta2);
    float k1 = L1 + L2 * cos(theta2);
    float k2 = L2 * sin(theta2);
    float theta1 = atan2(z, r) - atan2(k2, k1);
    float thetaBase = atan2(Y, X);

    baseAngle = constrain(radToDeg(thetaBase), 0, 180);
    shoulderAngle = constrain(radToDeg(theta1), 0, 180);
    elbowAngle = constrain(radToDeg(theta2), 0, 180);

    smoothMove(baseServo, baseAngle, baseAngle);
    smoothMove(shoulderServo, shoulderAngle, shoulderAngle);
    smoothMove(elbowServo, elbowAngle, elbowAngle);
}


//Move To Movement Control Function
void moveTo(float X, float Y, float Z) {
    Serial.print("Moving to X: "); Serial.print(X);
    Serial.print(", Y: "); Serial.print(Y);
    Serial.print(", Z: "); Serial.println(Z);

    inverseKinematics(X, Y, Z);
}
```

```
            case 'Z': // Increase Shoulder
                shoulderAngle = constrain(shoulderAngle + 5, 0, 180);
                angleUpdated = true;
                break;
            case 'X': // Decrease Shoulder
                shoulderAngle = constrain(shoulderAngle - 5, 0, 180);
                angleUpdated = true;
                break;
            default: return;
        }

        if (angleUpdated) {
            smoothMove(baseServo, baseAngle, baseAngle);
            smoothMove(shoulderServo, shoulderAngle, shoulderAngle);
            smoothMove(elbowServo, elbowAngle, elbowAngle);

            float X, Y, Z;
            forwardKinematics(X, Y, Z);

            Serial.print("Base: "); Serial.print(baseAngle);
            Serial.print("° | Shoulder: "); Serial.print(shoulderAngle);
            Serial.print("° | Elbow: "); Serial.print(elbowAngle);
            Serial.print("° || X: "); Serial.print(X, 2);
            Serial.print(" | Y: "); Serial.print(Y, 2);
            Serial.print(" | Z: "); Serial.println(Z, 2);
        }
    }
}
//Do not Edit Above Unless Necessary
```

```
128
129 // Manual control function
130 void manualControl() {
131     if (Serial.available() > 0) {
132         char command = Serial.read();
133         command = toupper(command);
134 
135         bool angleUpdated = false;
136 
137         switch (command) {
138             case 'W': // Decrease Shoulder & Decrease Elbow
139                 shoulderAngle = constrain(shoulderAngle - 5, 0, 180);
140                 elbowAngle = constrain(elbowAngle - 5, 0, 180);
141                 angleUpdated = true;
142                 break;
143             case 'S': // Increase Shoulder & Increase Elbow
144                 shoulderAngle = constrain(shoulderAngle + 5, 0, 180);
145                 elbowAngle = constrain(elbowAngle + 5, 0, 180);
146                 angleUpdated = true;
147                 break;
148             case 'A': // Increase Base
149                 baseAngle = constrain(baseAngle + 5, 0, 180);
150                 angleUpdated = true;
151                 break;
152             case 'D': // Decrease Base
153                 baseAngle = constrain(baseAngle - 5, 0, 180);
154                 angleUpdated = true;
155                 break;
156             case 'Q': // Decrease Elbow
157                 elbowAngle = constrain(elbowAngle - 5, 0, 180);
158                 angleUpdated = true;
159                 break;
160             case 'E': // Increase Elbow
161                 elbowAngle = constrain(elbowAngle + 5, 0, 180);
162                 angleUpdated = true;
163                 break;
164             case 'Z': // Increase Shoulder
165                 shoulderAngle = constrain(shoulderAngle + 5, 0, 180);
166                 angleUpdated = true;
167                 break;
```

```
}
//Do not Edit Above Unless Necessary

//Control Functions are Executed from this section
void loop() {


  manualControl();// After Excecuting Command, Use Serial Monitor and reffer to User Instruction for Controls


  moveTo(5.99, -5.02, 44.32);
  delay(2000);

  gripServo.write(90);
  delay(2000);
  gripServo.write(0);
  delay(2000);

  moveTo(0.00, -33.90, 28.63);
  delay(2000);

  moveTo(-7.81, 0.00, 44.32);
  delay(3000);

  gripServo.write(90);
  delay(2000);
  gripServo.write(0);
  delay(2000);

}
```