CONSTRUCTION OF A ROBOTIC ARM

Thesis · June 2024

DOI: 10.13140/RG.2.2.29191.00167

Muhammed Hamza Sefa
Işık University

UNIVERSITY OF THE BASQUE COUNTRY

FACULTY OF ENGINEERING, GIPUZKOA

GRADUATION PROJECT

eman ta zabal zazu

Universidad del País Vasco Euskal Herriko Unibertsitatea

CONSTRUCTION OF A ROBOTIC ARM

Student : Muhammed Hamza Sefa

Director : Eneko Solaberrieta Mendez

Semester : 2022/23

Abstract

This article presents the design and control of an industrial robotic arm using Arduino microcontroller and Bluetooth communication. The aim of this study is to develop a cost-effective and versatile robotic arm that can be controlled wirelessly via a mobile application, to better understand industrial automation and to have knowledge and experience on this subject individually. Robotic arm, Arduino UNO R3, HC05 Bluetooth module, MG996R and SG90 servo motors are key components in the design of the robotic arm. In the conclusion part, the system architecture, circuit diagram and the design of the robotic arm are explained in detail through visuals. Control of the arm is provided through an Android app that allows users to manipulate the movements of the arm. The coding process is explained in detail. The designed system offers a cost-effective Mini Robotic Arm design that offers flexibility and ease of control for industrial automation.

Table of Contents:

This page intentionally left blank

1. Introduction:

The aim of this study is to be familiar with robot arms by having knowledge and experience about industrial robots. In order to achieve this goal more successfully, to build a Robot Arm that will reveal the knowledge and skills on it. In this way, it is aimed to put the knowledge learned theoretically into practice.

1.1 Industrial Robot Definition

An industrial robot is defined as "an automatically controlled, reprogrammable, multipurpose manipulator programmable in three or more axes, which may be either fixed in place or mobile for use in industrial automation applications."[1] Industrial robots are becoming an essential component of industrial manufacturing facilities. This system, which is created using various electric, pneumatic, and hydraulic motor systems, is utilized throughout the business. Briefly, the robotic arm can be characterized as a collection of mechanical components that can be programmed or as a component of a sophisticated robot. It makes a substantial contribution to today's technology by requiring less labor, having a smaller margin of error, and producing more. Typical production applications of industrial robots include spot welding, material transfer, machine loading, spray painting, and assembly.

What makes robotic arms so significant, and why have they recently become so popular in the industry? Industrial robots are significant both commercially and technologically for the following reasons:

- Humans can be replaced by robots in risky or unwelcoming workplaces.
- Humans are unable to achieve the consistency and reproducibility that a robot achieves in performing its work cycle.
- Robots are programmable. A robot can be reprogrammed and given the required tools to do an entirely different task after the production run of the current operation is complete.
- Since computers operate robots, computer integrated manufacturing can be accomplished by connecting them to other computer systems.

Robotic arms are quite common, especially in the automotive industry. However, it is not yet able to completely replace human beings and has disadvantages. For example, a few of them are as follows:

- Cost of capital, industrial robots are already costly tools and can cause a high capital cost, so the expectation of return on investment should be well measured before deciding on the application of industrial robots. For this reason, this practice is more common in countries where labor is expensive than in countries where labor is cheap.
- The initial setup of industrial robots, like any other type of technology, requires a lot of training and expertise.
- Flexibility is particularly important in non-mass production plants where production volumes are relatively low and industrial robots are not yet suitable for this.

1.2. Robot Anatomy

Many components are required to manufacture an industrial robot. These are the Mechanical parts that make up the robot skeleton system, the actuators that provide the joint movement of the robot, the sensors that collect real-time data from the environment, the control unit that will process the data and create a reaction, and the end effector, which differs according to the application to be made and performs the desired operation.

1.2.1. Mechanical Part:

The mechanical part consists of joints and connections. Links are rigid members between joints. There are two types of joints, linear and rotary. Each joint provides "degree-of-freedom" and most robots in the industry have 5 or 6 degrees-of-freedom. Degree-of-freedom is also called axis. Currently, there are industrial robots with 1 to 7 axes on the market. More axes mean more functionality, but more axes also mean more expense. Many different robots can be used to solve a problem. For this reason, it is very important to understand the programming details and requirements before starting an application and to choose the robot with the required number of axes.

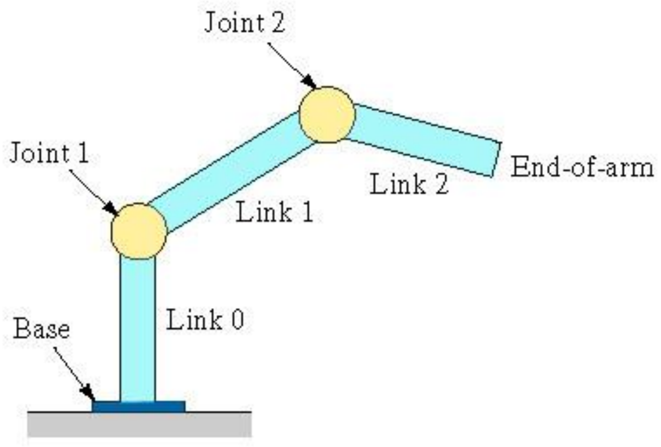


Figure 1. Robot manipulator - a series of joint-link combinations

1.2.1.1. Types of Manipulator Joints

Joint Notation Scheme, Uses the joint symbols (L, O, R, T, V) to designate joint types used to construct robot manipulator and Separates body-and-arm assembly from wrist assembly using a colon (:). Such as TLR : TR

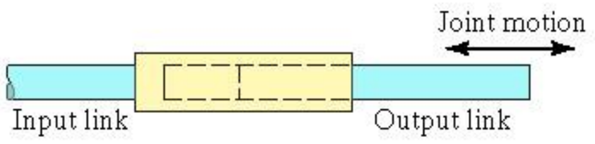


Figure 2. Linear Joint (type L)

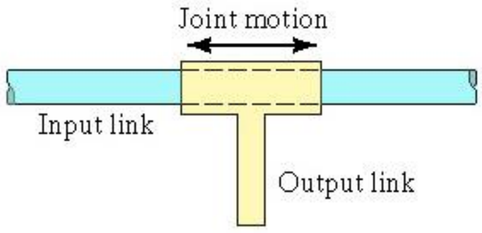


Figure 3. Orthogonal Joint (type O)

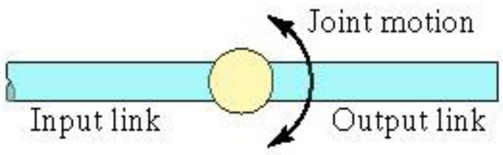


Figure 4. Rotational Joint (Type R)

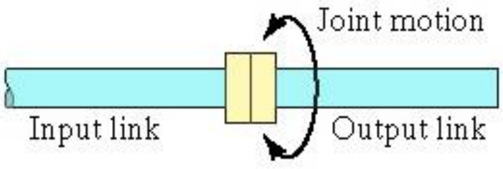


Figure 5. Twisting Joint (Type T)

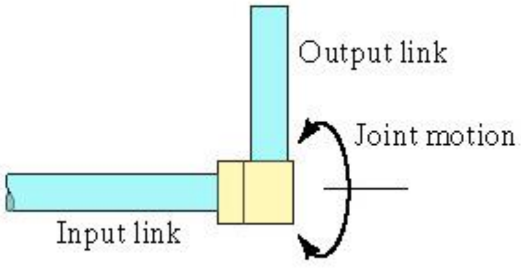


Figure 6. Revolving Joint (type V)

1.2.1.2. Types of Industrial Robots

The movement and design of articulated robots resembles a human arm, and it is important to know the types of manipulator joints because robot types are classified according to robot body-and-arm configurations. There are five common body-and-arm configurations for industrial robots. Function of body-and-arm assembly is to position an end effector (e.g., gripper, tool) in space.

Polar Robots: They are also called spherical robots. Their designs create a spherically shaped workspace. The main disadvantages of Polar robots are their incompatibility with other robots. Its configuration consists of a sliding arm (L joint) actuated relative to the body, which can rotate about both a vertical axis (T joint) and a horizontal axis (R joint). Polar robots are used for arc welding, assembly applications, spot welding, painting, injection, and molding.

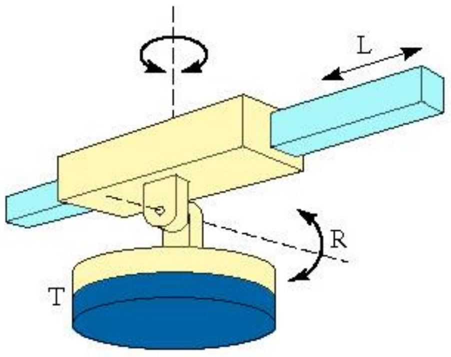


Figure 7. Polar Robot (Notation: TRL)

Articulated Robots: It has a general human shoulder and arm layout and is also referred to as a jointed-arm robot. It consists of an upright body that swivels about the base using a T joint. At the top of the body is a shoulder joint (R joint), whose output link connects to an elbow joint (another R joint).Used for assembly operations, diecasting, fettling machines, gas welding, arc welding and spray-painting. It is a robot whose arm has at least three rotary joints.

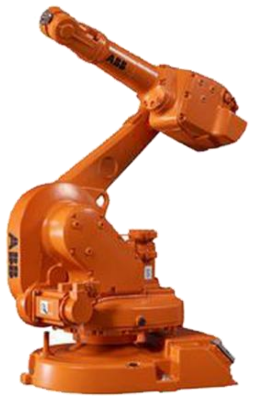


Figure 8. Articulated Robot (Notation: TRR)

Cylindrical Robots: Cylindrical robots are robots that consist of a vertical column, relative to which an arm assembly is moved up or down and the arm can be moved in or out relative to the column. This design requires a cylindrical shaped work area and therefore limits them to only two degrees of freedom. Therefore, the area in which they can operate is limited. Cylindrical robots are typically used in tight workspaces and are an excellent choice for objects that need to have round symmetry (e.g., wires, pipes). Grinding, assembly, spot welding applications benefit from cylindrical robots.

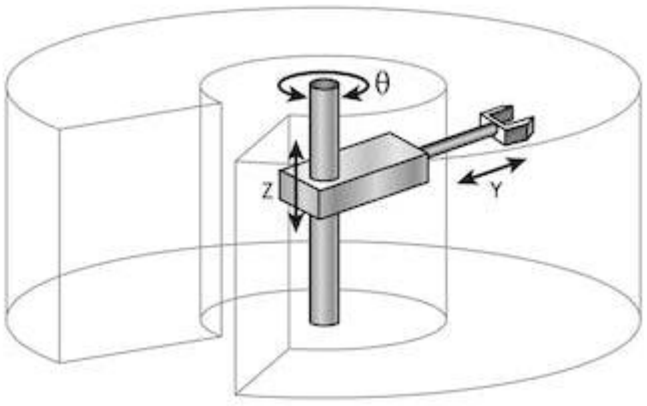


Figure 9. Cylindrical Robot (Notation: TLO)

Cartesian Robots: These robots, also called gantry robots, use the Cartesian Coordinate system (X, Y and Z) for linear movements in three axes. Consists of three sliding joints, two of which are orthogonal. All three joints are prismatic, so movement is limited to linear. Cartesian robots are mainly used for CNC machining and 3D printing, but they can also be used in assembly or pick-and-place applications with no limitations on the size of parts.

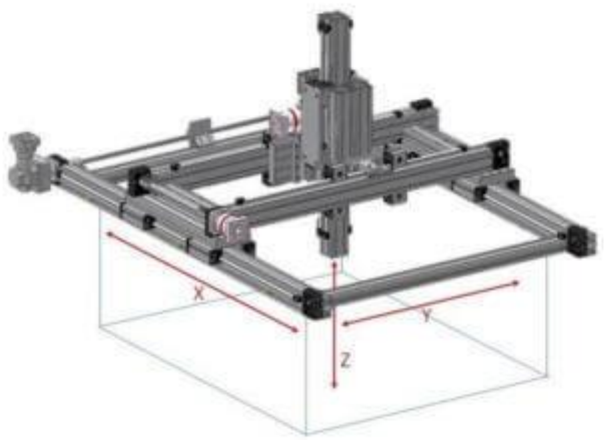


Figure 10. Cartesian Robot (Notation: LOO)

SCARA Robots: SCARA stands for Selectively Compliant Assembly Robot Arm, and they are similar to Articulated and Cartesian Robots in that they move in three axes, but they can also rotate. They are generally faster and more flexible, but less precise than Cartesian robots. In practice it involves assembling, palletizing, and placing small parts such as screws.

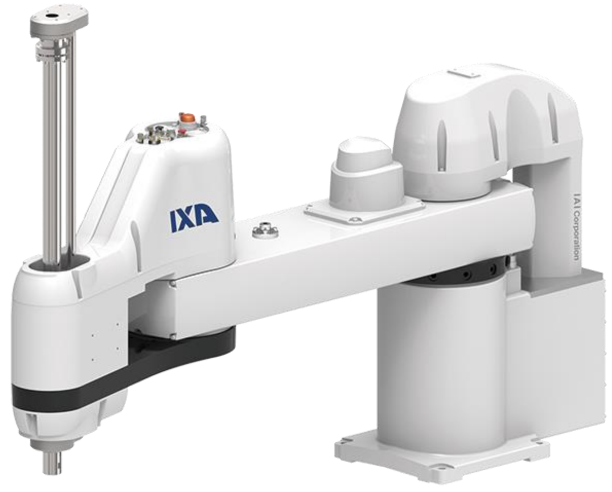


Figure 11. SCARA Robot (Notation: VRO)

1.2.1.3. Wrist Configurations

Robot Wrist mechanism is an important part of Industrial robots. Wrist assembly is attached to end-of-arm and end effector is attached to wrist assembly. Function of wrist assembly is to orient end effector, which does the main work, and supplies the opportunity to make different angles and different applications. Body-and-arm determines global position of end effector. Typical wrist assembly has two or three degrees-of-freedom which are roll, pitch and yaw (Figure 12.)

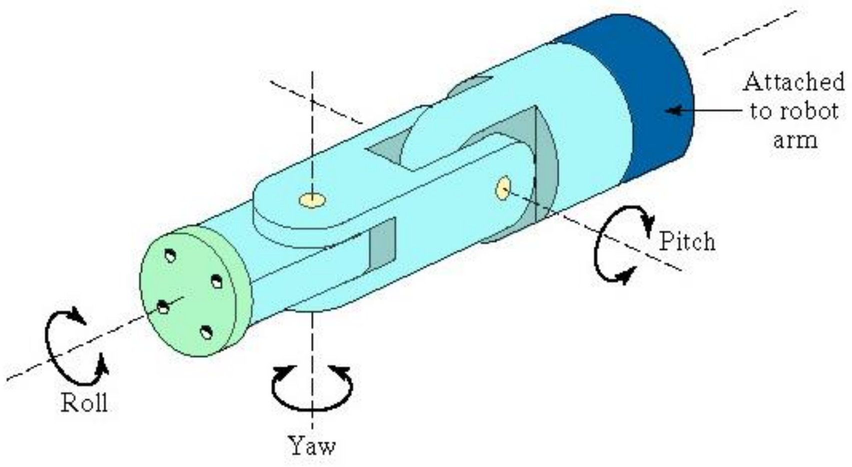


Figure 12. Robot Wrist (Notation: RRT)

1.2.1.4. End Effectors

End effectors are necessary for manipulators and robotic arms to carry out their intended functions. End effectors enable this equipment to interact with a subject by connecting to manipulator arms or a robot's wrist. Briefly, end effectors are the specialized equipment that a robot uses to carry out a particular task.

There are two types of end effectors, one is grippers that grasp and manipulate objects during work cycle such as pick and place, assembly tasks. According to the needs, grippers can be of various types, including electric grippers, pneumatic grippers, suction cups, magnetic grippers, and mechanical grippers. Another type is Tools. Tools to perform a process e.g., spot welding and spray painting.

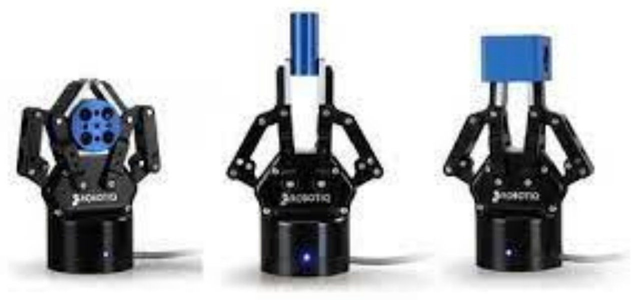


Figure 13.Grippers as End Effector

Figure 14. Tools as End Effector

1.2.1.4. Joint Drive Systems

To move the manipulator's body, arm, and wrist as desired requires control over the actions of each individual joint. The drive system that the robot is powered by provides the. A drive system is usually used to determine the capacity of a robot. Actuators driven by a specific kind of drive system move the joints. Electric drive, hydraulic drive, and pneumatic drive are commonly used drive systems in robotics.

- **Electric drive system:** Robots can be moved by electric drive systems at high speeds or with high power. Either DC servo motors or DC stepping motors can be used to actuate this kind of robot. Both rotational joints and linear joints can benefit from it. Small robots and precise applications will benefit greatly from the electric drive system. Most significantly, it is more accurate and repeatable. This system's slight price premium is its only drawback. Preferred drive system in today's robots.

- **Hydraulic Drive System:** The large-sized robots are the only ones for which the hydraulic drive systems are designed. Compared to electric drive systems, it can produce high power and lift capacity. The main drawback of this drive is thought to be hydraulic oil leakage.

- **Air-Powered Drive System:** The small type robots, which have fewer than five degrees of freedom, are specifically used with pneumatic drive systems. It has the capacity to provide both fine accuracy and quickness. When compared to the hydraulic drive, this system is less expensive. The system is typically limited to smaller robots and simple material transfer applications.

1.3. Robot Control Systems

Robot control systems regulate and direct the robot's operations to produce the desired outcome. The robot itself is an autonomous device with three main components that all function well together. The controller, actuator, and sensors are the three main components. All of the movements of the robot are directed by the controller using a feedback control system that is controlled by a computer program. For various applications, various types of control are needed. Four groups can be made for robot controllers. These are limited-sequence control, playback with point-to-point control, playback with continuous path control, and intelligent control.

- **Limited sequence control** – pick-and-place operations using mechanical stops to set positions.
- **Playback with point-to-point control** – records work cycle as a sequence of points, then plays back the sequence during program execution.
- **Playback with continuous path control** – greater memory capacity and/or interpolation capability to execute paths (in addition to points)
- **Intelligent control** – exhibits behaviour that makes it seem intelligent, e.g., responds to sensor inputs, makes decisions, communicates with humans.

Each joint has its own feedback control system thanks to the controller's hierarchical organization, as shown in Figure 15, and a supervisory controller coordinates the combined actuations of the joints in accordance with the robot program's sequence.

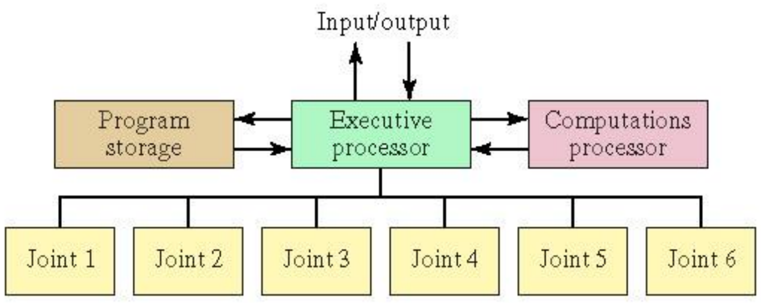


Figure 15. Hierarchical control structure of a robot microcomputer controller

1.4. Robot Programming

Any robotic system needs to be programmed. It provides an industrial robot with crucial information that enables it to function and complete specific tasks. Programming for robots was once thought to be difficult due to the complicated lines of code required. Fortunately, advances in robotics have prompted the creation of more user-friendly programming techniques that are accessible to even the most inexperienced operators. Nowadays, the most common methods to program industrial robots are Teach Method, Hand Guiding/Lead-Through Programming and Offline Robot Programming.

1.4.1. Teaching Method

A handheld control device is used to program and control the robot's positioning in pendant teaching, the most common technique for programming robots. By using the pendant controller to direct the robot through each step of the operation, the operator can record each motion.

In contemporary pendants, a joystick, keypads, and a display are frequently utilized. The workpiece orientation and positioning of the robot are controlled by the joystick. The keypad can be used to enter data and perform predefined functions. On the display screen, operator messages and soft control keys are displayed. Teach pendants are ideal for simple movements like painting a straight line or a large flat surface.

1.4.2. Lead-Through Programming

Lead-through is the simplest of all robot teaching methods. With the servo controls turned off, the operator can manually move the robot arm to its desired path. Significant points can be saved for later use when the robot travels along the path taught.

Although lead-through is a low-precision method, the operator may have editing options to improve precision once the robot has been taught its desired path. Lead-through programming promotes skilled workers who know how to do the job manually with high precision rather than technical skills and thus does not require workers to have any knowledge of computer programming.

The method is most preferred for modern collaborative robots used in smaller industrial installations. It is faster than other methods, but this method is not suitable for situations that require precision.

1.4.2. Offline Robot Programming

Offline programming is another robot programming technique that can be used when automating with robots. Simulation, another name for offline programming, is primarily used in robotics research. In order to ensure that control algorithms work properly before being tested with the robot itself, offline programming is advantageous. With this technique, the desired application and an industrial robot are virtually simulated. Before launching the program live, the application simulation can be modified and reconfigured to ensure programming accuracy. The program can be downloaded to the robot once a technician is happy with it. Offline programming is growing in popularity despite being still relatively new to robotics because it causes the least amount of disruption to manufacture operations. In this method, since the programming process can be performed while the robot is running, robots can be more productive with this method.

2. Features and Materials:

As it has been summarized in the introduction section, according to the information it has been obtained, the Robotic Arm Project will be as follows. According to the information obtained, Industrial Robotic Arms are very costly products, the aim in this project is to design a Mini Robotic Arm which has a low cost, also called the Desktop Robotic Arm, to become familiar with this type of systems and to experience their working principles. For this reason, a model is designed with 5-axis and gripper as end effector, which is widely used in the industry.

The anatomy of the robot arm that is going to be builded is as follows. There are 5 joints in total, 3 on the body and 2 on the wrist. Since each joint means an axis, the robot arm will be 5-axis and 1 servo will be used for the gripper. Such a structure is in the Articulated Robot class and its notation is TRR : RT.

Since the motors to be used in the joints require precise control and feedback so there will be an electric drive system and servo motors will be used for the robotic arm. Arduino, a very popular and widely used controller used to control servo motors for this reason it will also be used as the controller because they are easy to control and provide great position control.

Since relatively stronger motors are required for the 3 joints in the body, 3 MG996R servos[15] and 3 SG90 micro servos[16] for the end effector will be used. These servos will be used for Waist- Shoulder- Elbow- Wrist Roll- Wrist Pitch- Gripper, respectively. These motors are popular and widely used motors that work with Arduino and have been used in other projects before.

For programming, the Teaching method is going to be used. It is going to be done using the HC05 Bluetooth Module and Arduino IDE. The advantage of using Bluetooth module is being able to connect with the robot via the phone and this will obtain to the user mobility and flexibility. In case user want to program the robotic arm but not have computer, it makes them able to program the robotic arm.

More powerful servos could have been preferred for the end effector to have much torque to grip, but in the current situation, Arduino Uno is sufficient as the controller, in that case Arduino Mega will be required. Consequently, considering that those servos and Arduino Mega are more costly than Arduino Uno and micro servos, this situation will make more sense for the purpose of the project.

2.1. Material and Cost List

Material	Arduino Uno Rev3(17)	MG996R Servo Motor(18)	SG90 Micro Servo Motor(19)	HC-05 Bluetooth Module(20)	5V 2A DC Power Supply(21)	Cables (24)	PCB Breadboard (25)	Total
Price (€)	**29,28**	**15,99**	**9,29**	**9,99**	**14,99**	**0,73**	**0,99**	**81,26**

Table 1. Cost List

2.2. Characterization of Components

In this section, the main components used in the construction of the Robotic Arm are introduced and information about their characteristics is given.

2.2.1. Arduino UNO R3 (22)

Arduino Uno is a microcontroller board based on the ATmega328 microcontroller. It has 14 digital input/output pins, 6 analog inputs, a 16 MHz crystal oscillator, a USB connection, a power jack, an ICSP header, and a reset button. It is programmed using the Arduino programming language and the Arduino Integrated Development Environment (IDE). The Arduino Uno can be used to control a variety of electronic devices such as LEDs, motors, and sensors. It is widely used in DIY electronics projects and is an open-source hardware platform.

Figure 16. Arduino Uno

2.2.2. HC05 Bluetooth Module (23)

The HC-05 is a Bluetooth module that can be used to add wireless communication to a microcontroller board. It operates in slave or master mode and can transmit and receive data over a distance of up to 10 meters. It is commonly used in DIY electronics projects.

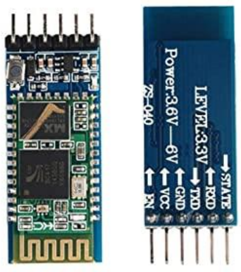


Figure 17. HC-05 Bluetooth Module

2.2.3. MG996R Servo Motor (12,15)

A typical kind of hobbyist servo motor used in a broad range of applications, including robotics, RC cars and aircraft, and other do-it-yourself electronics projects, is the MG996R servo. It is a digital servo motor that can rotate 120 degrees and can produce up to 11kg/cm of torque.

The MG996R servo's metal gear structure, which makes it especially robust and resistant to wear and tear, is one of its distinguishing characteristics. Additionally, it has a quick response time and excellent precision, which makes it ideal for robotics and other applications that call for precise motions.

The MG996R servo operates within the 4.8–7.2V voltage range and uses around 500mA- 900mA (6V) of current when loaded. Additionally, a potentiometer that may be adjusted is included for fine-tuning the servo's center point.

In general, the MG996R servo is a dependable and adaptable motor that is suitable for a variety of DIY and hobbyist electronics applications.

Figure 18. MG996R Servo

2.2.4. SG90 Micro Servo Motor (12,16)

A typical small-sized servo motor used in robotics and other hobbies is the SG90 servo. It is a lightweight, inexpensive servo with a plastic gear train that makes it perfect for modest applications. The SG90 servo is appropriate for regulating minor motions in a range of applications since it has a torque output of around 2.5kg/cm and can spin up to 180 degrees. It uses very little electricity to function and is relatively simple to regulate using a microcontroller or other control board.

When under load, the SG90 servo uses a voltage range of 3.5 to 6V and consumes about 200 to 250 mA of current. It is commonly offered from electronics merchants and online marketplaces, has a common three-pin connection for simple cabling, and is inexpensive. Overall, the price, use, and adaptability of the SG90 servo make it a well-liked option for DIY and hobbyist electronics applications.

Figure 19. SG90 Micro Servo

2.2.5. Breadboard

A breadboard is a tool used in electronics to create and test circuits without soldering. It is like a reusable board with a grid of holes where you can plug in electronic components and connect them using wires. The holes are arranged in rows and columns, and they allow you to easily experiment and modify your circuit. There are two main sections on a breadboard: the terminal strip and the bus strip. The terminal strip lets you connect components in different ways, while the bus strip provides power and ground connections. Breadboards are great for beginners and hobbyists to quickly build and test circuits without the need for soldering skills.

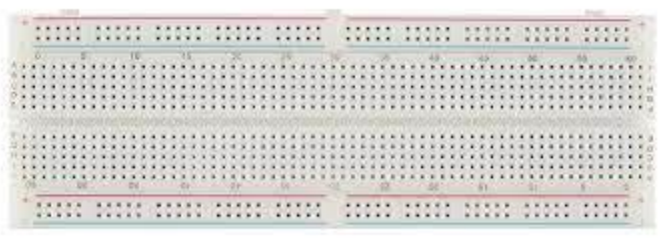


Figure 20. Breadboard

3.Results:

In the results part, the system architecture, circuit diagram, robot arm design, control android application and coding parts that are required for the construction of a robotic arm suitable for individual use, which is easy to manufacture, practical and low-cost, with the achievements obtained from the theory and information parts of the report.

3.1. The System Architecture

System architecture refers to the overall structure, design, and organization of a computer system or software application. It defines how the different components and modules of a system interact with each other to achieve the desired functionality and performance. The overall structure, design and organization of the Robotic Arm, and the interactions of the components and modules with each other are shown in Figure 21.

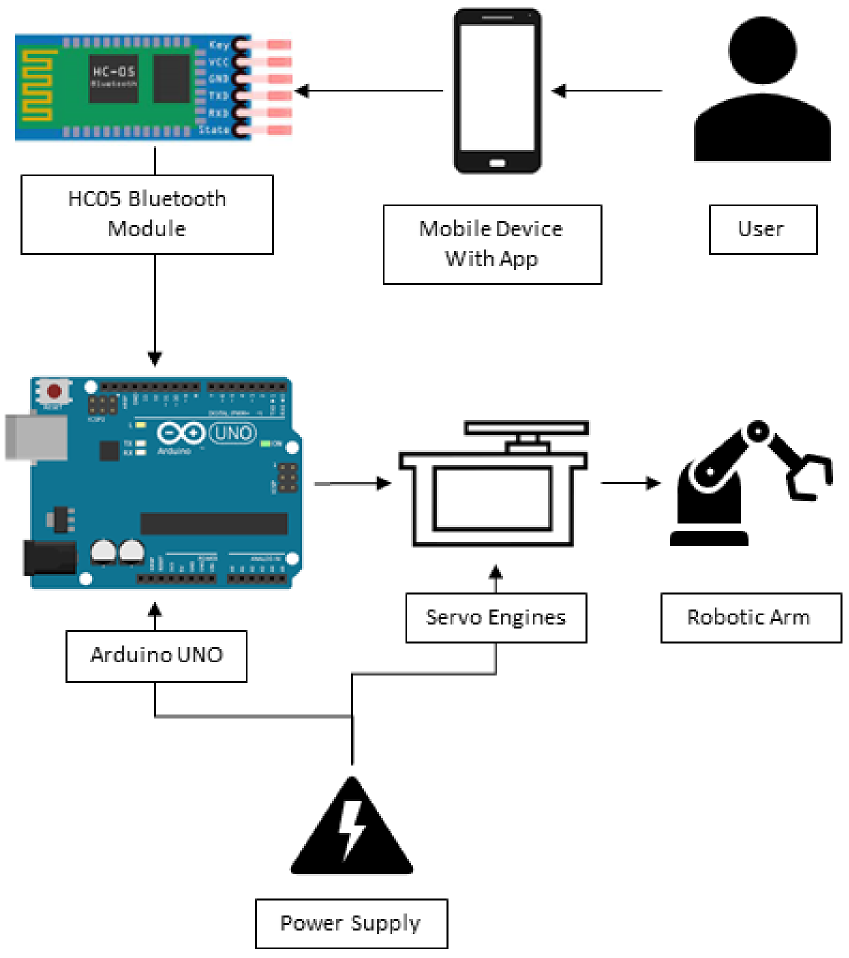


Figure 21. The System Architecture

3.2. Circuit Diagram

Fritzing was used to obtain the circuit diagram (Figure 22). Fritzing is a computer program that helps users design and draw pictures of electronic circuits. It offers a user-friendly interface where individuals can select electronic components and place them on a virtual board. The components can be connected using virtual wires to represent the connections in the circuit. Fritzing provides different views, allowing users to visualize the circuit in various ways, such as a depiction of the components on a board, a circuit diagram, or a layout for creating an actual circuit board.

Connecting the electronics, the circuit diagram of this project is quite simple. We just need an Arduino Uno and on HC-05 Bluetooth module for the communication with the smartphone. The control pins of the six servo motors are connected to six digital pins of the Arduino board for powering the servos. We need 5 volts, but this must come from an external power source because the Arduino is not able to handle the amount of current that all of them can draw. The power source must be able to handle at least 2 amps of current. The power source to be used in the project is the adapter and 9V battery is used as a representative in this diagram.

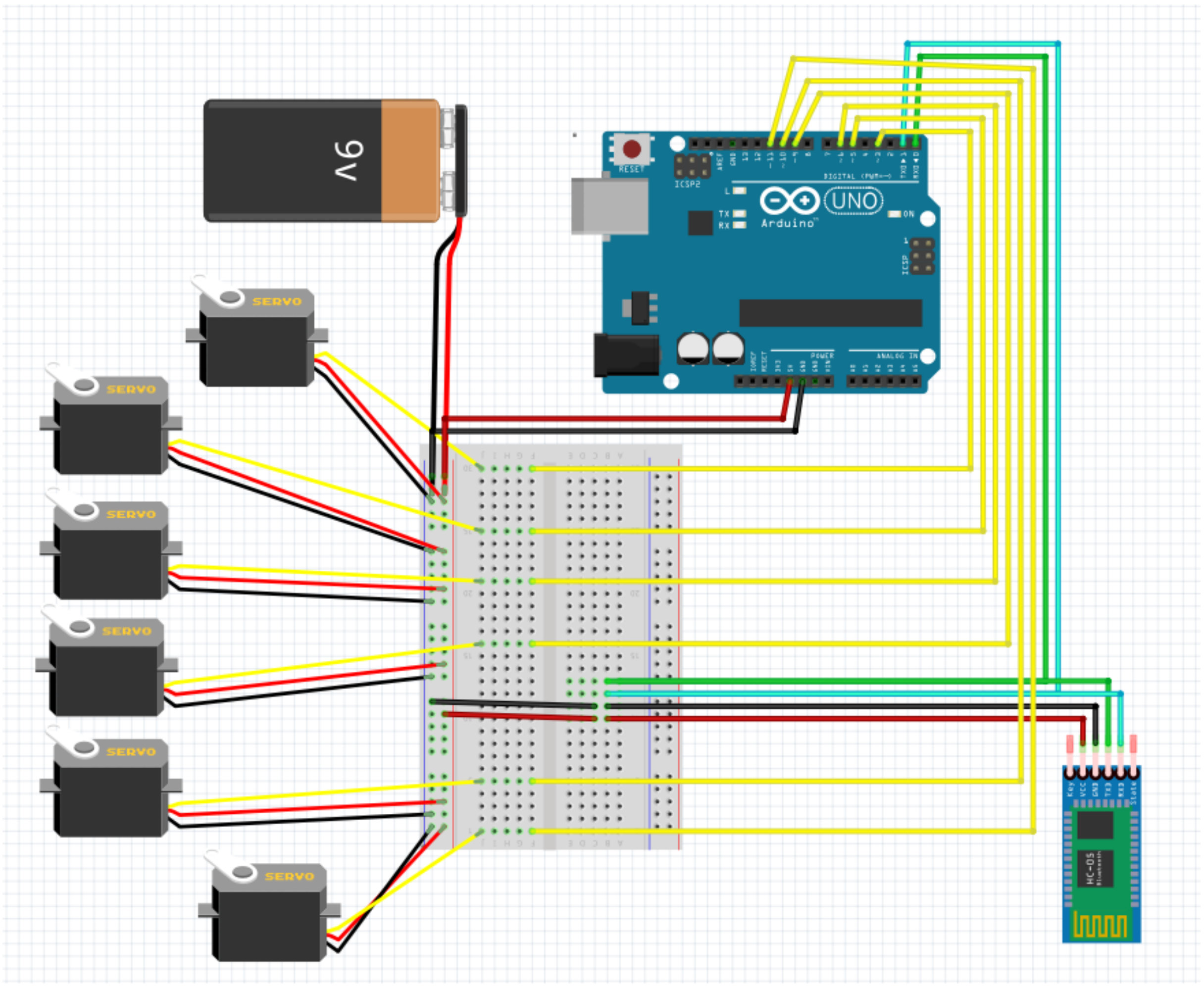


Figure 22. Fritzing Scheme

3.3. Robotic Arm Design

In Robotic Arm manufacturing, many designs can be made according to its purpose. Almost all the robot arms used in the industry are manufactured using sheet metal. However, in line with the environment in which this project will be used and the target of the project, it will be costly and very laborious as a manufacturing process to produce a product in this way. For these reasons, the most suitable manufacturing technique is 3D printing. In manufacturing with 3D printing, the parts of the product will be manufactured with 3D printing and will be suitable for assembly afterwards. Although it is not essential, optional trimming can be done, no other external process is required. It is a quite simple and non-skilful manufacturing technique.

The robotic arm was designed to be created using 3D printing as a manufacturing technique by using SolidWorks 3D modeling software. Since it is aimed to have a different and characteristic appearance, a hexagon-based and relatively sharp-line design has been made. The design consists of 6 main parts in total, as follows, from the base to the end: Waist- Shoulder- Elbow- Wrist Roll- Wrist Pitch- Gripper. In addition, the Gripper part consists of 9 parts assembling (Figure 23, Figure 24). For the first 3 axis, the waist, the shoulder, and the elbow, the MG996R servos is used, and for the other 2 axis, the wrist roll and wrist pitch, as well as the gripper, the smaller SG90 micro servos was used. Such a structure is in the Articulated Robot class and its notation is TRR : RT.

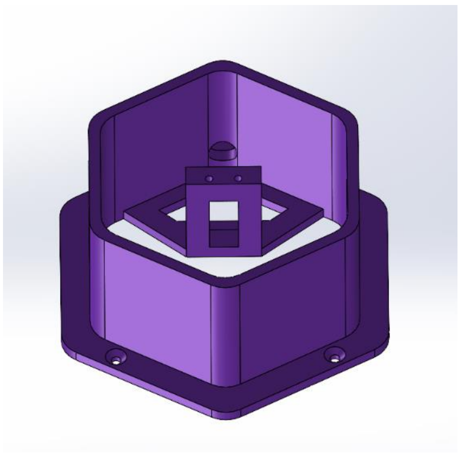


Figure 23. Waist Perspective 1

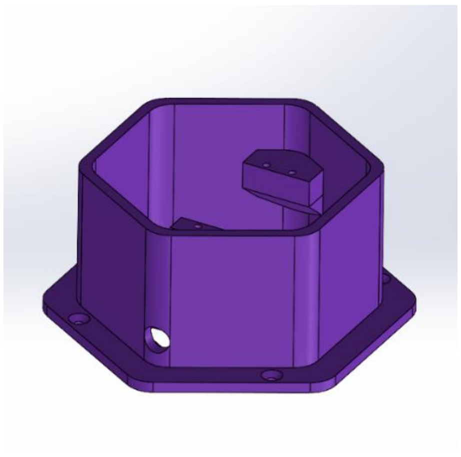


Figure 24. Waist Perspective 2

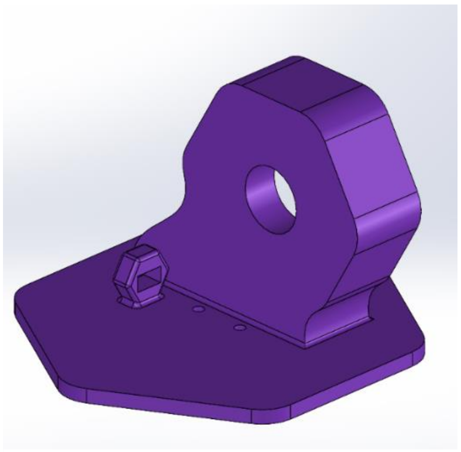


Figure 25. Shoulder Perspective 1

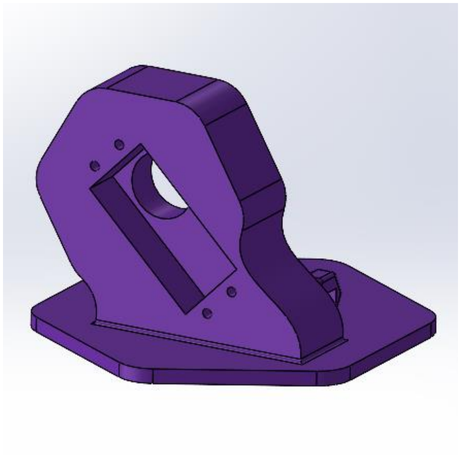


Figure 26. Shoulder Perspective 2

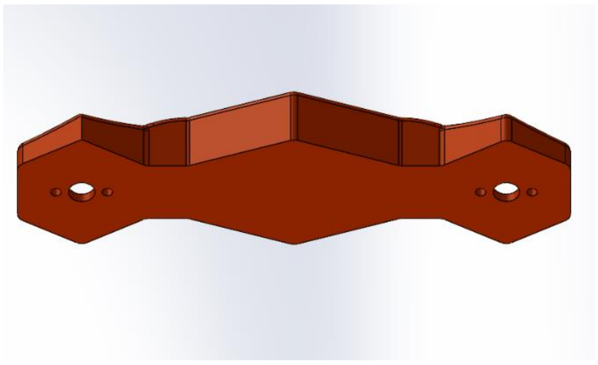


Figure 27. Elbow Perspective 1

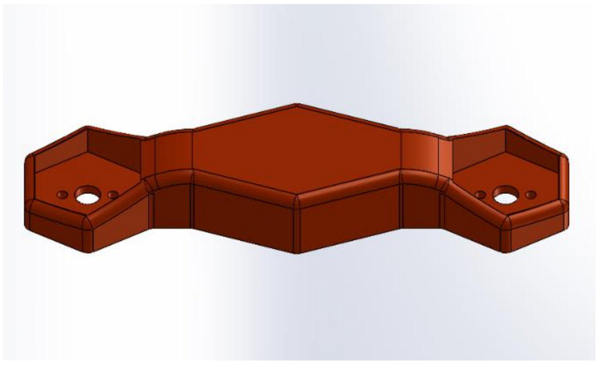


Figure 28. Elbow Perspective 2

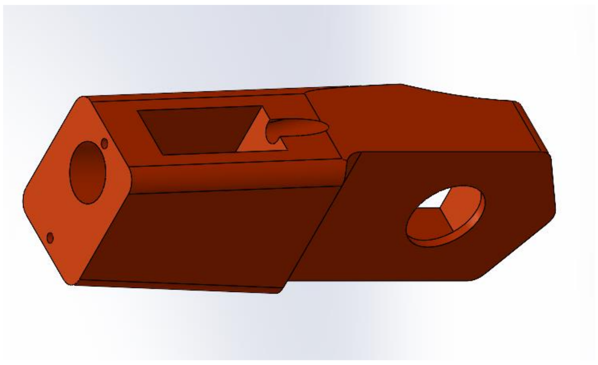


Figure 29. Wrist Roll Perspective 1

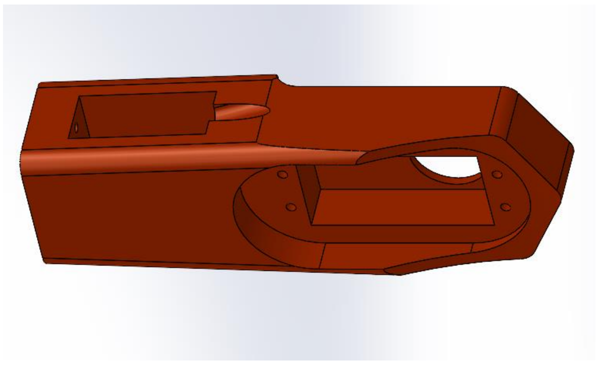


Figure 30. Wrist Roll Perspective 2

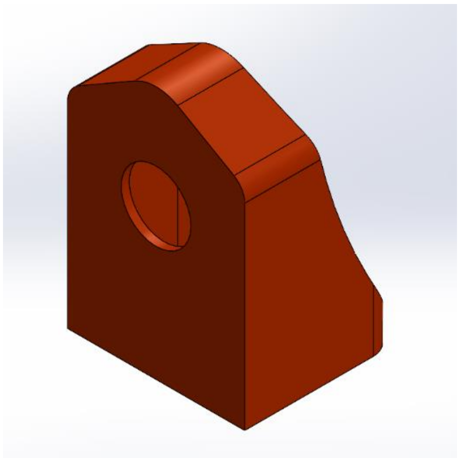


Figure 31. Wrist Pitch Perspective 1

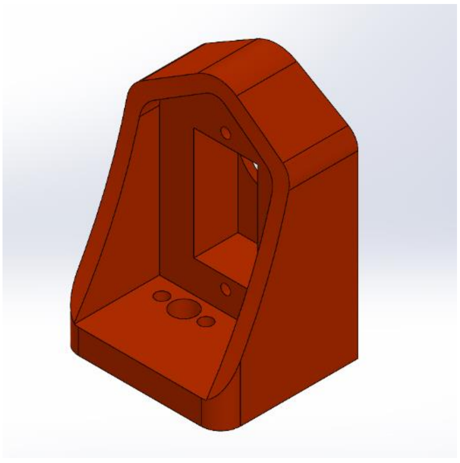


Figure 32. Wrist Pitch Perspective 2

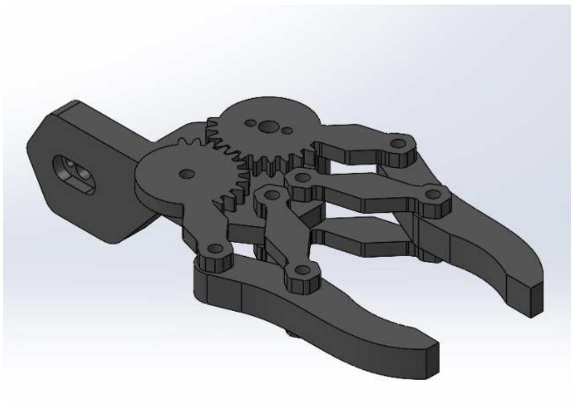


Figure 33. Gripper Perspective 1

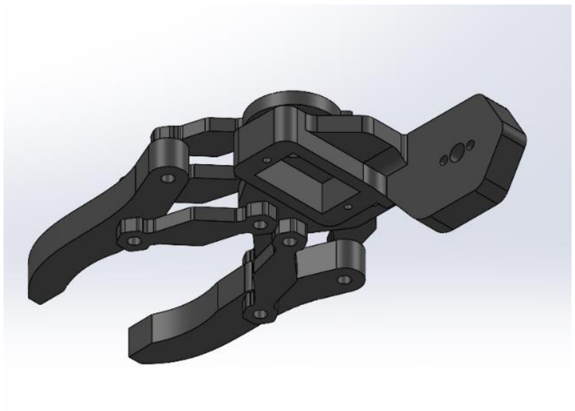


Figure 34. Gripper Perspective 2

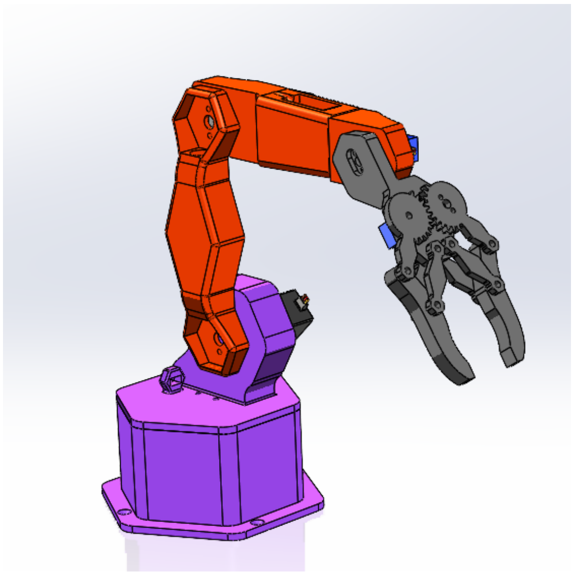


Figure 35. Full Assembly Perspective 1

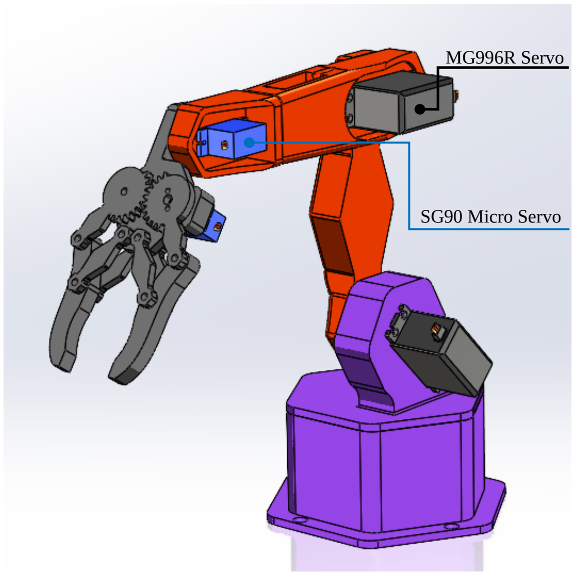


Figure 36. Full Assembly Perspective 2

Exploded images of Assembled parts are given below to help the manufacturing process and simplify the assembly process.

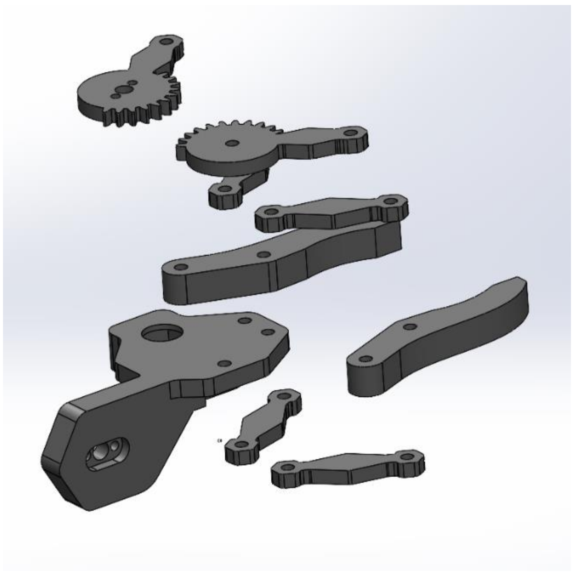


Figure 37. Gripper's Exploded View 1

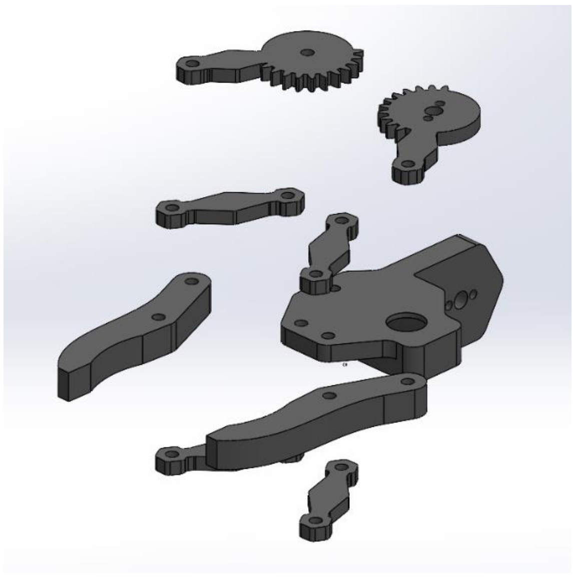


Figure 38. Gripper's Exploded View 2

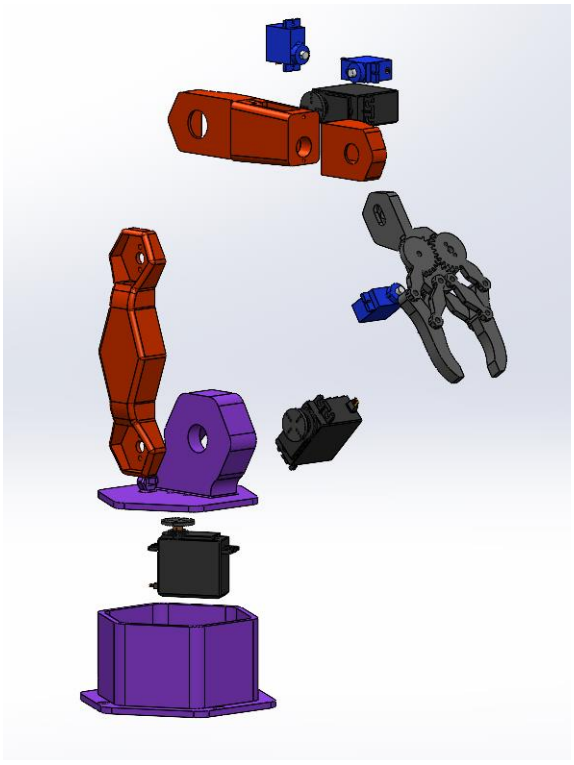


Figure 39. The Robotic Arm's Exploded View 1

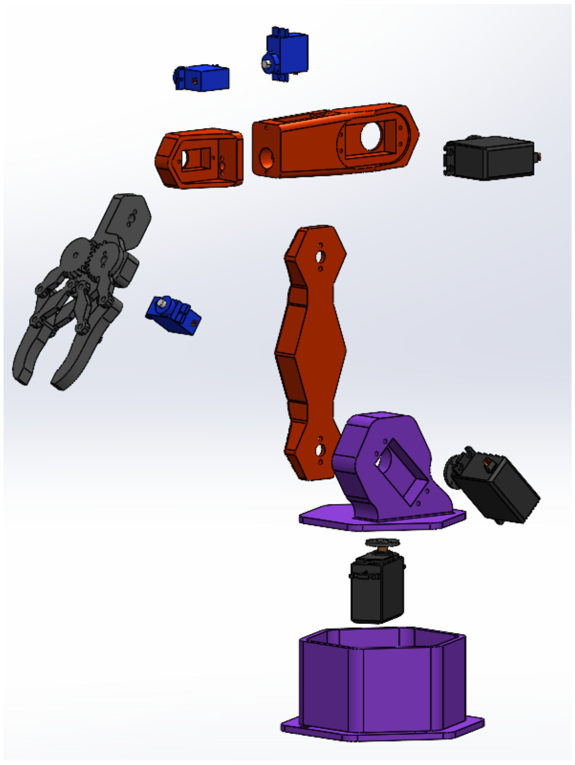


Figure 40. The Robotic Arm's Exploded View 2

Technical drawing is used to show dimensions of the product.

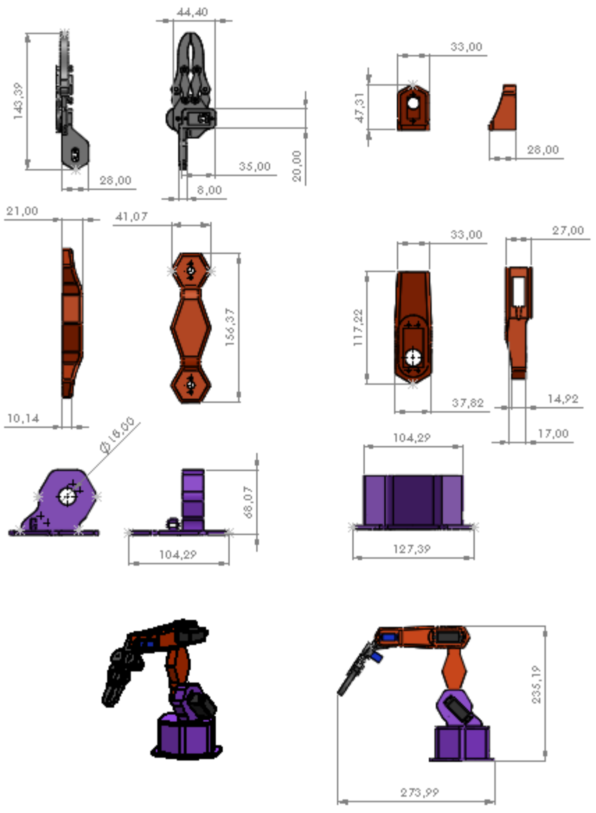


Figure 41. Technical Drawing of All Parts

3.4. Control Android App

Android Application is the step where user send data to Arduino. While designing the application, the MIT App Inventor online application (Figure 42), which has a very simple and user-friendly interface, was used. The application design is as follows (Figure 43). At the top there are two buttons for connecting the smartphone to the HC-05 Bluetooth module. Then on the left side there is an image of the robot arm, and on the right side there are six sliders for controlling the servos and one slider for the speed control. Each slider has different initial, minimum, and maximum value that suits the robot arm joints. At the bottom of the app, there are three buttons, SAVE, RUN and RESET through which user can program the robot arm to run automatically. There is also a label below which shows the number of steps that is saved.

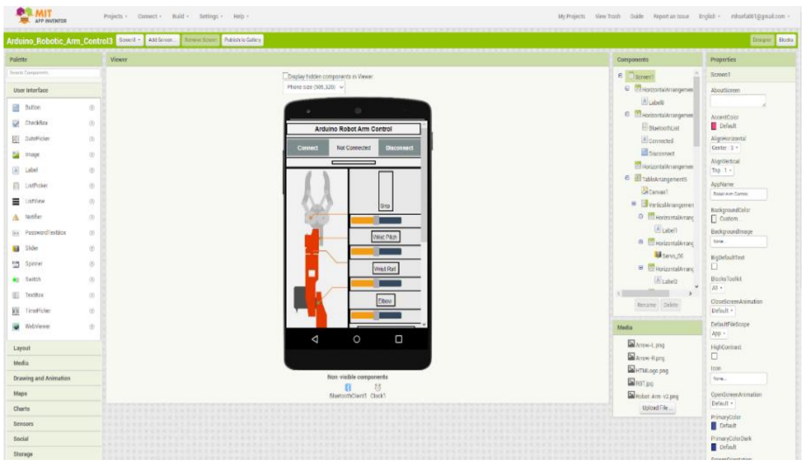


Figure 42. Interface of MIT App Inventor

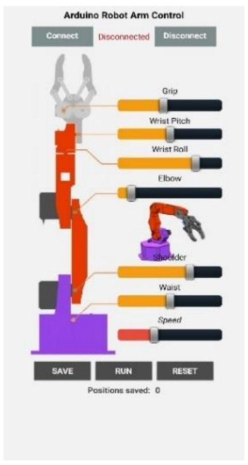
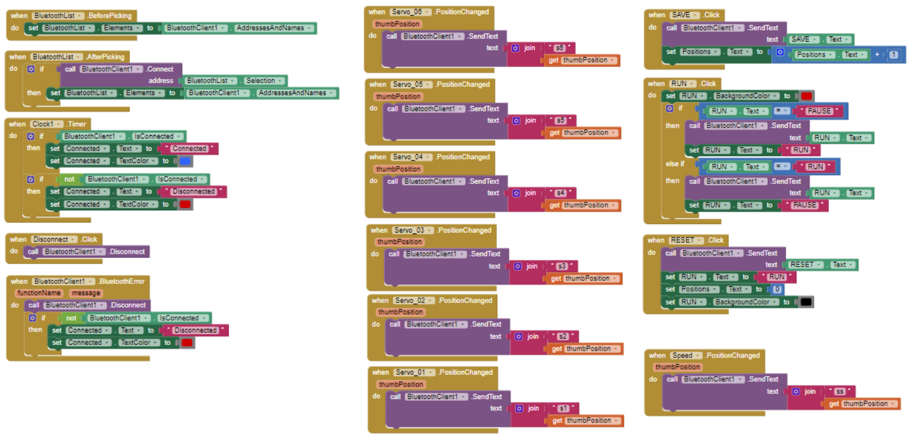


Figure 43. Interface of the App

Background programming is required for an application to work as desired, this is called Back-And. The background blocks for the application are as follows (Figure 44). First, on the left side it has the blocks for connecting the smartphone to the Bluetooth module. Then it has the sliders blocks for the servo position control and the buttons blocks for programming the robot arm. So, if the user changes the position of the slider, using the Bluetooth function **SendText**, the app sends a text to the Arduino. This text consists of a prefix which indicates which slider has been changed as well as the current value of the slider.



```
when BluetoothList .BeforePicking
do  set BluetoothList . Elements to BluetoothClient1 . AddressesAndNames

when BluetoothList .AfterPicking
do  if   call BluetoothClient1 .Connect
                         address BluetoothList . Selection
    then set BluetoothList . Elements to BluetoothClient1 . AddressesAndNames

when Clock1 .Timer
do  if   BluetoothClient1 . IsConnected
    then set Connected . Text to " Connected "
         set Connected . TextColor to 🟦
    if   not BluetoothClient1 . IsConnected
    then set Connected . Text to " Disconnected "
         set Connected . TextColor to 🟥

when Disconnect .Click
do  call BluetoothClient1 .Disconnect

when BluetoothClient1 .BluetoothError
 functionName  message
do  call BluetoothClient1 .Disconnect
    if   not BluetoothClient1 . IsConnected
    then set Connected . Text to " Disconnected "
         set Connected . TextColor to 🟥

when Servo_06 .PositionChanged
 thumbPosition
do  call BluetoothClient1 .SendText
                         text join " s6 "
                                   get thumbPosition

when Servo_05 .PositionChanged
 thumbPosition
do  call BluetoothClient1 .SendText
                         text join " s5 "
                                   get thumbPosition

when Servo_04 .PositionChanged
 thumbPosition
do  call BluetoothClient1 .SendText
                         text join " s4 "
                                   get thumbPosition

when Servo_03 .PositionChanged
 thumbPosition
do  call BluetoothClient1 .SendText
                         text join " s3 "
                                   get thumbPosition

when Servo_02 .PositionChanged
 thumbPosition
do  call BluetoothClient1 .SendText
                         text join " s2 "
                                   get thumbPosition

when Servo_01 .PositionChanged
 thumbPosition
do  call BluetoothClient1 .SendText
                         text join " s1 "
                                   get thumbPosition

when SAVE .Click
do  call BluetoothClient1 .SendText
                         text SAVE . Text
    set Positions . Text to Positions . Text + 1

when RUN .Click
do  set RUN . BackgroundColor to 🟥
    if      RUN . Text = " PAUSE "
    then    call BluetoothClient1 .SendText
                                 text RUN . Text
            set RUN . Text to " RUN "
    else if RUN . Text = " RUN "
    then    call BluetoothClient1 .SendText
                                 text RUN . Text
            set RUN . Text to " PAUSE "

when RESET .Click
do  call BluetoothClient1 .SendText
                         text RESET . Text
    set RUN . Text to " RUN "
    set Positions . Text to 0
    set RUN . BackgroundColor to ⬛

when Speed .PositionChanged
 thumbPosition
do  call BluetoothClient1 .SendText
                         text join " ss "
                                   get thumbPosition
```

Figure 44. The Blocks Behind the Application

3.5. Coding

For programming the robotic arm, Arduino IDE, which is Arduino's own application and uses C base, is used. The code supplies the connection between the application that is in the device and Arduino, via the HC-05 Bluetooth Module and moves the servos of the arm accordingly. The code allows for real time control of individual servos or the execution of pre-defined sequences of servo movements. Additionally, it provides functionality to save and recall specific servo positions, enabling the creation of custom motion patterns for the robot arm. The teaching method was aimed while programming the robotic arm. Because of this method, the code works like that in general. The user saves the path that the robot should move step by step and after the run command the process starts, in this way the desired operation is performed. In the following part, the code is explained part by part.

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

Servo servo01;
Servo servo02;
Servo servo03;
Servo servo04;
Servo servo05;
Servo servo06;

SoftwareSerial Bluetooth(0, 1); // Arduino(RX, TX) - HC-05 Bluetooth (TX, RX)
```

Figure 45.

- This section includes the necessary libraries: **SoftwareSerial** for serial communication with the Bluetooth module and **Servo** for controlling the servo motors.
- Six **Servo** objects are declared to control the six servo motors.
- An instance of **SoftwareSerial** is created to communicate with the HC-05 Bluetooth module, using pins 0 and 1 for RX and TX respectively.

```
int servo1Pos, servo2Pos, servo3Pos, servo4Pos, servo5Pos, servo6Pos; //
current position
int servo1PPos, servo2PPos, servo3PPos, servo4PPos, servo5PPos, servo6PPos; //
previous position
int servo01SP[50], servo02SP[50], servo03SP[50], servo04SP[50], servo05SP[50],
servo06SP[50]; // for storing positions/steps
int speedDelay = 20;
int index = 0;
String dataIn = "";
```

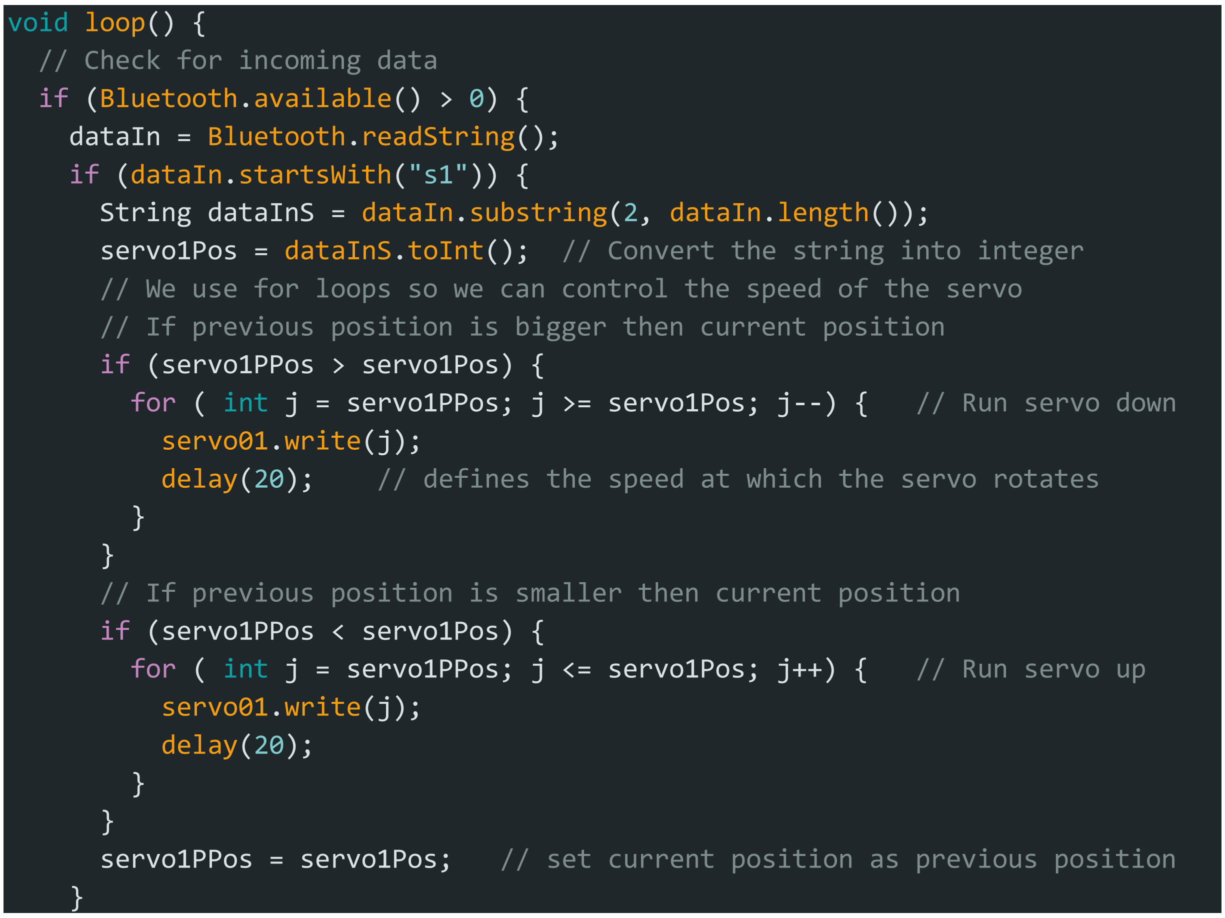
Figure 46.

- These variables are declared to store the current and previous positions of the servos, as well as the positions/steps to be stored for execution.
- **servo1Pos** to **servo6Pos** store the current positions of the respective servos, while **servo1PPos** to **servo6PPos** store the previous positions.
- Arrays **servo01SP** to **servo06SP** are used to store the positions/steps that can be saved and executed later.
- **speedDelay** determines the delay between each step execution, controlling the speed of the servo movements.
- **index** keeps track of the number of saved steps.
- **dataIn** stores the received data from the Bluetooth module.

```
void setup() {
  servo01.attach(2);
  servo02.attach(5);
  servo03.attach(6);
  servo04.attach(9);
  servo05.attach(10);
  servo06.attach(11);
  Bluetooth.begin(38400); // Default baud rate of the Bluetooth module
  Bluetooth.setTimeout(1);
  delay(20);
  // Robot arm initial position
  servo1PPos = 90;
  servo01.write(servo1PPos);
  servo2PPos = 150;
  servo02.write(servo2PPos);
  servo3PPos = 35;
  servo03.write(servo3PPos);
  servo4PPos = 140;
  servo04.write(servo4PPos);
  servo5PPos = 85;
  servo05.write(servo5PPos);
  servo6PPos = 80;
  servo06.write(servo6PPos);
}
```

Figure 47.

- The **setup()** function is called once when the Arduino is powered on or reset.
- Servos are attached to their respective pins.
- The Bluetooth module is initialized with a baud rate of 38400, and the timeout is set to 1 millisecond.
- A small delay is added for stability.
- Initial positions for all servos are set arbitrarily between 0-180 that is the angle range of the servos.
- The initial positions are also written to the servos using the **write()** method.

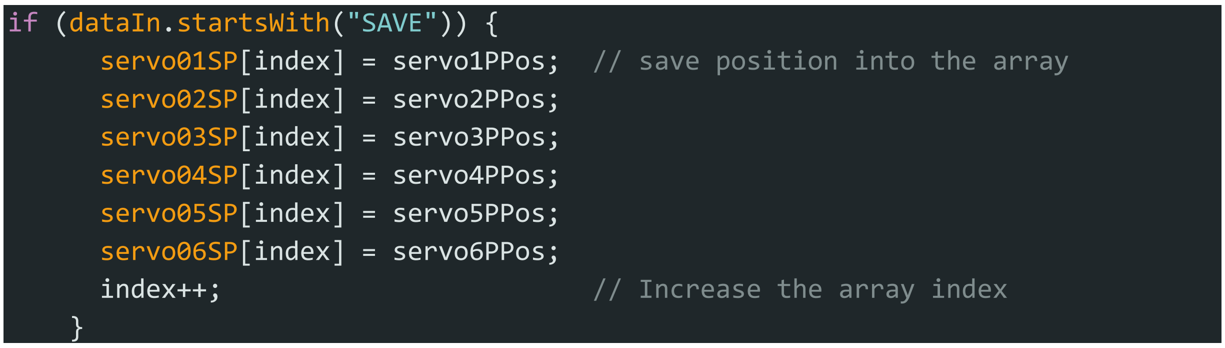


```
void loop() {
  // Check for incoming data
  if (Bluetooth.available() > 0) {
    dataIn = Bluetooth.readString();
    if (dataIn.startsWith("s1")) {
      String dataInS = dataIn.substring(2, dataIn.length());
      servo1Pos = dataInS.toInt();  // Convert the string into integer
      // We use for loops so we can control the speed of the servo
      // If previous position is bigger then current position
      if (servo1PPos > servo1Pos) {
        for ( int j = servo1PPos; j >= servo1Pos; j--) {   // Run servo down
          servo01.write(j);
          delay(20);    // defines the speed at which the servo rotates
        }
      }
      // If previous position is smaller then current position
      if (servo1PPos < servo1Pos) {
        for ( int j = servo1PPos; j <= servo1Pos; j++) {   // Run servo up
          servo01.write(j);
          delay(20);
        }
      }
      servo1PPos = servo1Pos;   // set current position as previous position
    }
```

Figure 48.

- **Bluetooth.available** checks the connection of Bluetooth and reads the data as string.
- This part of the code checks if the received data begins with the prefix "s1", indicating that the value for servo 1 has changed.
- It extracts the numeric value from the data by removing the first two characters ("s1") and converts it to an integer.
- Depending on whether the current position is greater or smaller than the previous position, the servo is moved up or down gradually using a for loop.
- The delay() function is used to control the speed at which the servo moves.
- Finally, the current position is stored as the previous position for future comparisons.

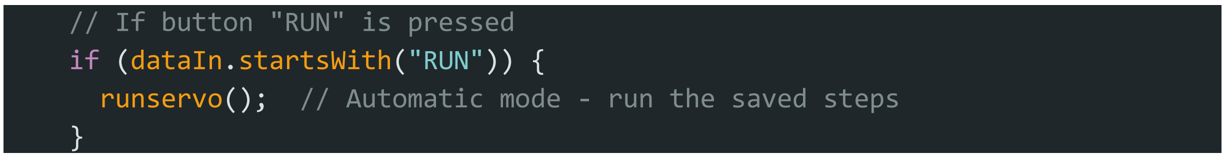
The code then repeats similar blocks of code for servos 2 to 6, checking if their values have changed and moving them accordingly.



```
if (dataIn.startsWith("SAVE")) {
      servo01SP[index] = servo1PPos;  // save position into the array
      servo02SP[index] = servo2PPos;
      servo03SP[index] = servo3PPos;
      servo04SP[index] = servo4PPos;
      servo05SP[index] = servo5PPos;
      servo06SP[index] = servo6PPos;
      index++;                        // Increase the array index
    }
```

Figure 49.

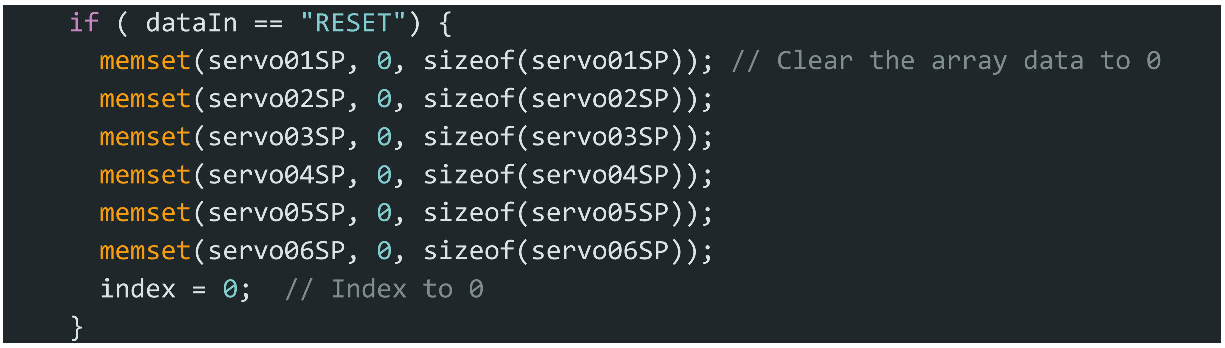
- This part checks if the received data starts with the string "SAVE." If it does, it means the "SAVE" button was pressed. The code then saves the current positions of the servos (**servo1PPos**, **servo2PPos**, etc.) into corresponding arrays (**servo01SP**, **servo02SP**, etc.) at the current index. The **index** variable keeps track of the current position in the array, and it is incremented after saving the positions.



```
    // If button "RUN" is pressed
    if (dataIn.startsWith("RUN")) {
      runservo();  // Automatic mode - run the saved steps
    }
```

Figure 50.

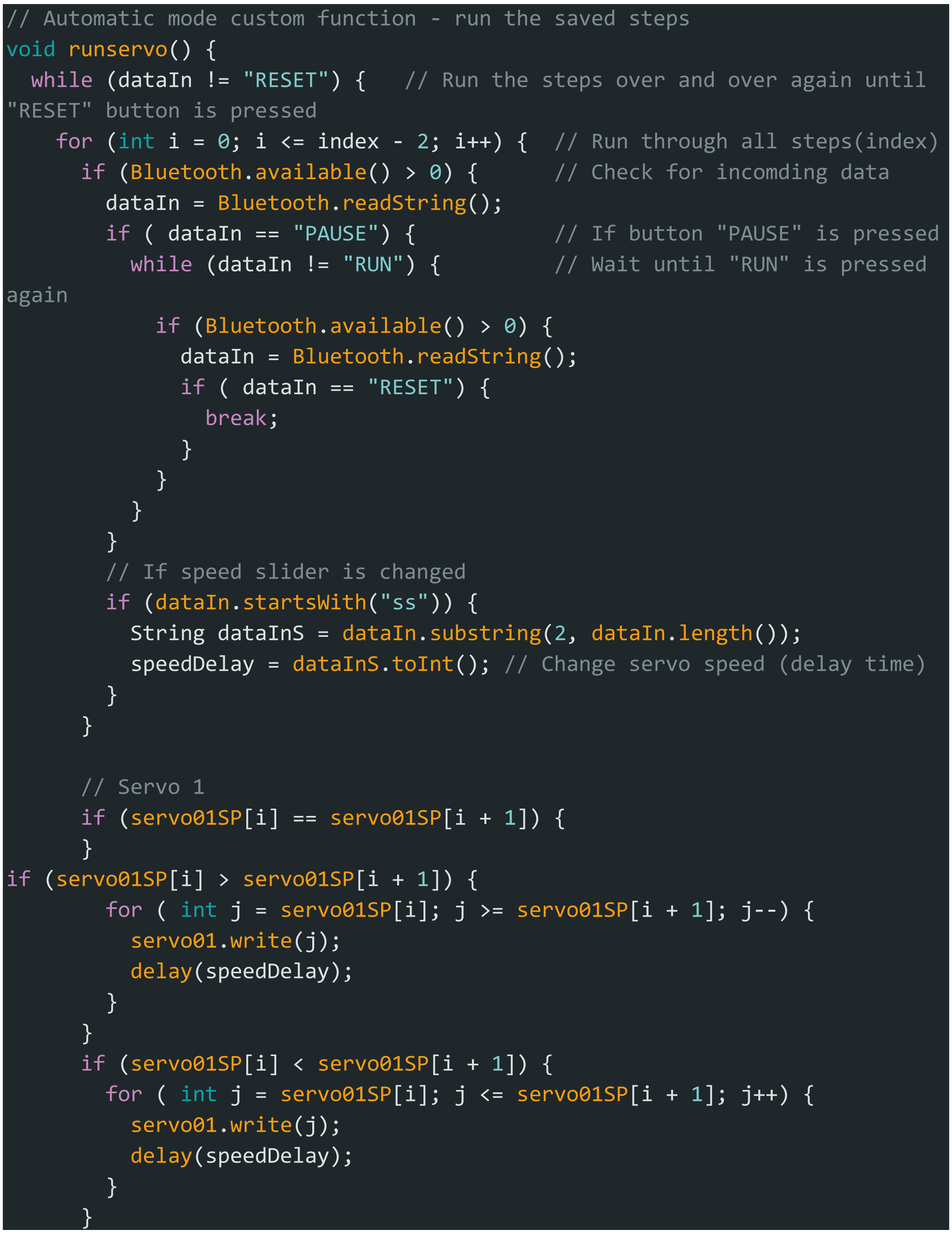
- When the received data starts with the string "RUN," it indicates that the "RUN" button was pressed. The code then calls the **runservo()** function, which runs the saved steps in an automatic mode.



```
    if ( dataIn == "RESET") {
      memset(servo01SP, 0, sizeof(servo01SP)); // Clear the array data to 0
      memset(servo02SP, 0, sizeof(servo02SP));
      memset(servo03SP, 0, sizeof(servo03SP));
      memset(servo04SP, 0, sizeof(servo04SP));
      memset(servo05SP, 0, sizeof(servo05SP));
      memset(servo06SP, 0, sizeof(servo06SP));
      index = 0;  // Index to 0
    }
```

Figure 51.

- This part checks if the received data is equal to the string "RESET." If it is, it means the "RESET" button was pressed. The code then uses the **memset()** function to set all elements of the servo position arrays (**servo01SP**, **servo02SP**, etc.) to 0. Additionally, it sets the **index** variable to 0, indicating that the array is now empty.



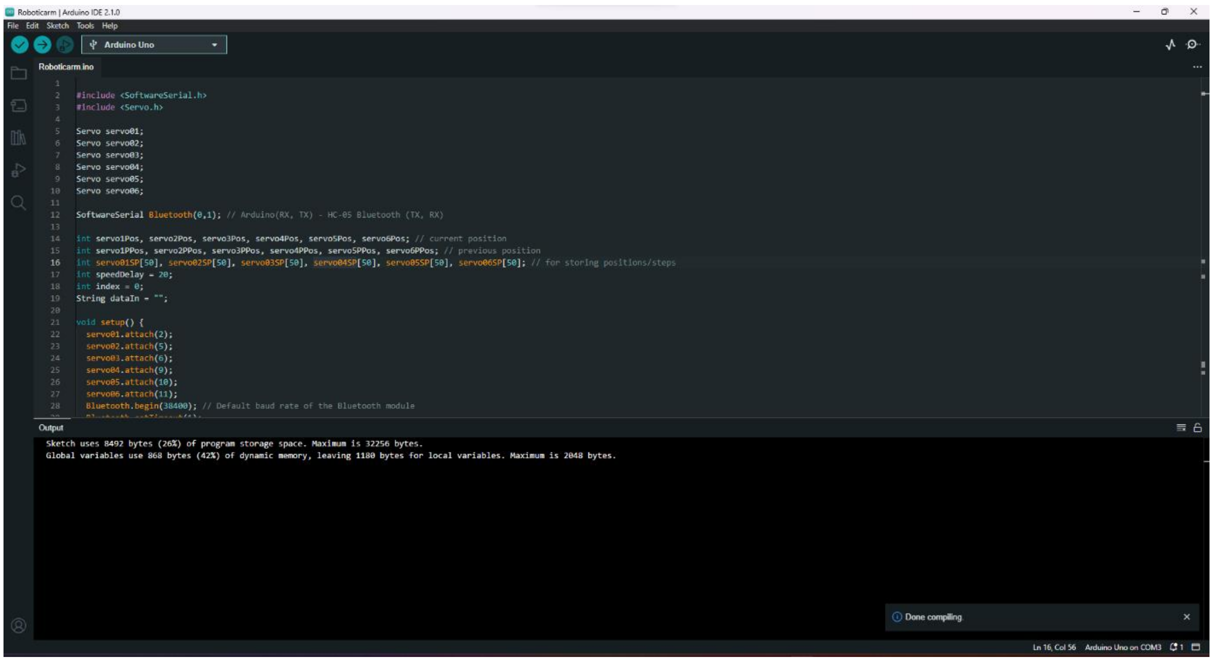
```
// Automatic mode custom function - run the saved steps
void runservo() {
  while (dataIn != "RESET") {   // Run the steps over and over again until
"RESET" button is pressed
    for (int i = 0; i <= index - 2; i++) {  // Run through all steps(index)
      if (Bluetooth.available() > 0) {      // Check for incomding data
        dataIn = Bluetooth.readString();
        if ( dataIn == "PAUSE") {           // If button "PAUSE" is pressed
          while (dataIn != "RUN") {         // Wait until "RUN" is pressed
again
            if (Bluetooth.available() > 0) {
              dataIn = Bluetooth.readString();
              if ( dataIn == "RESET") {
                break;
              }
            }
          }
        }
        // If speed slider is changed
        if (dataIn.startsWith("ss")) {
          String dataInS = dataIn.substring(2, dataIn.length());
          speedDelay = dataInS.toInt(); // Change servo speed (delay time)
        }
      }

      // Servo 1
      if (servo01SP[i] == servo01SP[i + 1]) {
      }
if (servo01SP[i] > servo01SP[i + 1]) {
        for ( int j = servo01SP[i]; j >= servo01SP[i + 1]; j--) {
          servo01.write(j);
          delay(speedDelay);
        }
      }
      if (servo01SP[i] < servo01SP[i + 1]) {
        for ( int j = servo01SP[i]; j <= servo01SP[i + 1]; j++) {
          servo01.write(j);
          delay(speedDelay);
        }
      }
```

Figure 52.

- Here the code runs the store steps over and over again until user press the reset button. Using the for loop the code runs through all positions stored in the RX.
- At the same time **Bluetooth.available()** checks whether it has any incoming data from the smartphone. This data can be the **RUN** or **PAUSE** button which pauses the robot and if clicked again, it continues with the automatic movements.

- In similar way as explained earlier, with these if statements and for loops the user moves the servo to their next position.
- If the received data starts with "**ss**" which means speed slider position is changed, it extracts the speed value from the string and the code will use that value to change the delay time **"delay(speedDelay)"** between each iteration in the for loops below which controls the speed of the servo motors.
- For each servo (servo 1 to 6), the code compares the current position with the next position. Depending on the comparison, it either moves the servo gradually by increasing or decreasing the position value.

The code is compiled to verify that the code works correctly and is applicable. As shown in Figure 53, the code compiles successfully and does not show any errors. The code has been shared completely in the appendix part (Appendix 1).


Roboticarm | Arduino IDE 2.1.0
File Edit Sketch Tools Help
Arduino Uno
Roboticarm.ino
Output
Done compiling.
Ln 16, Col 56 Arduino Uno on COM3

```

#include <SoftwareSerial.h>
#include <Servo.h>

Servo servo01;
Servo servo02;
Servo servo03;
Servo servo04;
Servo servo05;
Servo servo06;

SoftwareSerial Bluetooth(0,1); // Arduino(RX, TX) - HC-05 Bluetooth (TX, RX)

int servo1Pos, servo2Pos, servo3Pos, servo4Pos, servo5Pos, servo6Pos; // current position
int servo1PPos, servo2PPos, servo3PPos, servo4PPos, servo5PPos, servo6PPos; // previous position
int servo01SP[50], servo02SP[50], servo03SP[50], servo04SP[50], servo05SP[50], servo06SP[50]; // for storing positions/steps
int speedDelay = 20;
int index = 0;
String dataIn = "";

void setup() {
  servo01.attach(2);
  servo02.attach(5);
  servo03.attach(6);
  servo04.attach(9);
  servo05.attach(10);
  servo06.attach(11);
  Bluetooth.begin(38400); // Default baud rate of the Bluetooth module
```

```
Sketch uses 8492 bytes (26%) of program storage space. Maximum is 32256 bytes.
Global variables use 868 bytes (42%) of dynamic memory, leaving 1180 bytes for local variables. Maximum is 2048 bytes.
```

Figure 53. Compiling of the Code

Conclusion

In conclusion, the construction of a robotic arm suitable for individual use has been successfully demonstrated through this project. The aim was to design a robotic arm that is easy to manufacture, practical, and low-cost. The project utilized a combination of hardware and software components to achieve the desired functionality and control of the robotic arm.

The system architecture of the robotic arm was designed to ensure the seamless interaction between its components. The circuit diagram was created using Fritzing, allowing for a clear visualization of the electronic connections and components required for the project. The robotic arm was designed using SolidWorks 3D modeling software, with a focus on a distinctive hexagon-based design that can be easily manufactured using 3D printing techniques. The design consisted of six main parts: Waist, Shoulder, Elbow, Wrist Roll, Wrist Pitch, and Gripper.

To control the robotic arm, an Android application was developed using the MIT App Inventor platform. The application provided a user-friendly interface with buttons and sliders for controlling the servos and programming the arm. The application allowed users to send commands to the Arduino through the HC-05 Bluetooth module, enabling real-time control of the arm's movements.

The coding part involved programming the Arduino using the Arduino IDE and C-based language. The code facilitated the communication between the Android application and the Arduino, ensuring the proper movement of the servos based on the user's commands. The code allowed for real-time control of individual servos as well as the execution of pre-defined sequences of servo movements. It also provided functionality to save and recall specific servo positions, enabling the creation of custom motion patterns for the robotic arm.

Overall, this project successfully demonstrated the design, construction, and control of a robotic arm suitable for individual use. The combination of 3D printing for manufacturing, an Android application for control, and Arduino programming for servo control resulted in a practical and low-cost solution. This robotic arm has the potential for various applications, such as educational purposes, small-scale automation tasks, and hobby projects. Further improvements and enhancements can be made to expand its capabilities and adapt it to specific use cases.

References

1. 2008 Pearson Education, Inc., Upper Saddle River, NJ. "*Automation, Production Systems, and Computer-Integrated Manufacturing,*" Fourth Edition, by Mikell P. Groover.
2. October 11, 2017, www.Asme.org, "*How Many Axes Does Your Robot Need?*", By Crawford Mark
3. www.entes.com, "*What is Industrial Robot? What Tasks Is It Used For?*"
4. https://en.wikipedia.org/wiki/Robotic_arm, Robotic arm
5. 10 December 2021, *"'DEGREES OF FREEDOM' VS 'FUNCTIONS' OF A ROBOTIC ARM"*
6. www.brainkart.com, *"Introduction Robot Drive Systems"*
7. 14 October 2021, www.onlinerobotics.com, *"Introduction to Robotic Control Systems"*
8. 12 January2022, https://www.unidex-inc.com/blog/end-effector-what-it-is-how-it-works-types-applications/ ,
9. 6 May 2022, "Three Methods of Programming Industrial Robots",https://roboticsbiz.com/three-methods-of-programming-industrial-robots/, by Editorial
10. 28 Jan 2021, https://www.ballerstatus.com/2021/01/28/the-3-basic-robot-programming-methods/
11. 21 April 2023, "Servo Motor Basics with Arduino " , https://docs.arduino.cc/learn/electronics/servo-motors, by Arduino
12. 2018, "How to Control Servo Motors with Arduino – Complete Guide", https://howtomechatronics.com/how-it-works/how-servo-motors-work-how-to-control-servos-using-arduino/, by Dejan
13. 2018, "Arduino Servo Motors", https://www.instructables.com/Arduino-Servo-Motors/, by Cornelam
14. "How Servo Motor Works & Interface It With Arduino", https://lastminuteengineers.com/servo-motor-arduino-tutorial/
15. https://www.electronicoscaldas.com/datasheet/MG996R_Tower-Pro.pdf
16. http://www.ee.ic.ac.uk/pcheung/teaching/DE1_EE/stores/sg90_datasheet.pdf
17. https://www.amazon.es/Arduino-UNO-A000066-microcontrolador-ATmega328/dp/B008GRTSV6
18. https://www.amazon.com/WWZMDiB-MG996R-Digital-Control-Helicopter/dp/B0BTBLFTWX/ref=sr_1_50?keywords=servo+motor+sg90&qid=1682281506&sr=8-50
19. https://www.amazon.com/Micro-Servos-Helicopter-Airplane-Controls/dp/B07MLR1498/ref=sr_1_5?keywords=servo%2Bmotor%2Bsg90&qid=1682281506&sr=8-5&th=1
20. https://www.amazon.com/DSD-TECH-HC-05-Pass-through-Communication/dp/B01G9KSAF6/ref=sr_1_1_sspa?crid=D6V6IVUT3G32&keywords=HC-05+Bluetooth+Module&qid=1682281888&sprefix=hc-05+bluetooth+module%2Caps%2C258&sr=8-1-spons&psc=1&spLa=ZW5jcnlwdGVkUXVhbGlmaWVyPUExVVAxTFRKTlhBMVUyJmVuY3J5cHRlZElkPUEwMDUzMDU1MllMSDVUT0hNT1pLQyZlbmNyeXB0Z

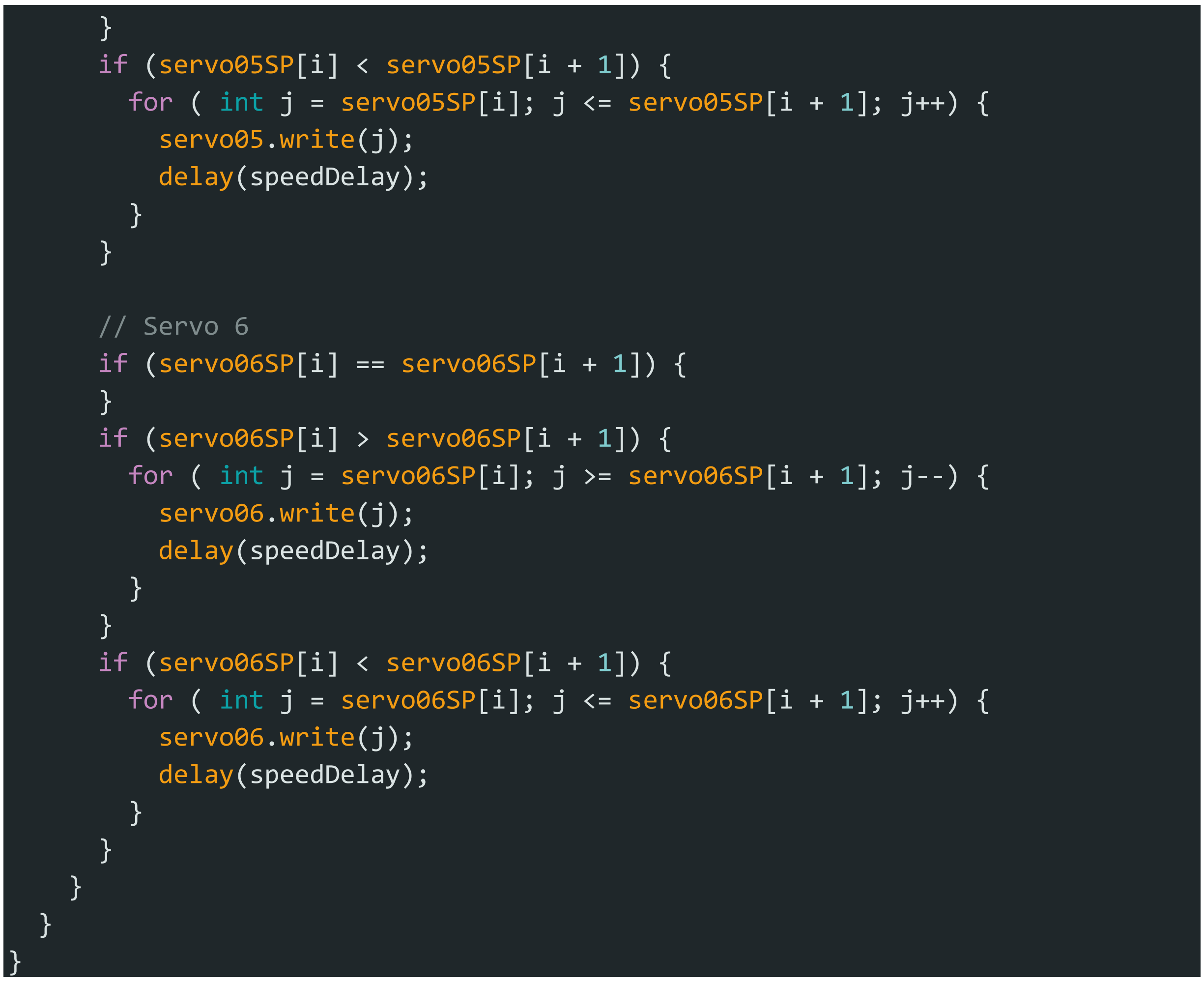
WRBZElkPUEwNTI1OTEyWVEwUkMyQlpVUTFWJndpZGdldE5hbWU9c3BfYXRmJmFjdGlvbj1jbGlja1JlZGlyZWN0JmRvTm90TG9nQ2xpY2s9dHJ1ZQ==

21. https://www.amazon.es/EFISH-Adaptador-Multifuncional-2A-Diferentes/dp/B0825P8TZP/ref=sr_1_6?__mk_es_ES=ÅMÅŽÕÑ&crid=1R8JN366WQMH0&keywords=5V%2B2A%2BDC%2BPower%2BSupply&qid=1682282443&refinements=p_36%3A1323855031&rnid=1323854031&s=electronics&sprefix=5v%2B2a%2Bdc%2Bpower%2Bsupply%2Caps%2C264&sr=1-6&th=1
22. UNO R3 | Arduino Documentation | Arduino Documentation
23. HC05 Bluetooth-Serial Modül Kartı Satın Al | Robotistan
24. https://es.aliexpress.com/item/32951870747.html?spm=a2g0o.productlist.main.55.69f45681ibtBvT&algo_pvid=cd8ff501-beb2-4452-ac6d-326153121175&algo_exp_id=cd8ff501-beb2-4452-ac6d-326153121175-27&pdp_npi=3%40dis%21EUR%210.83%210.73%21%21%21%21%21%402100ba4716858834549846053d074c%2166291478011%21sea%21ES%210&curPageLogUid=aVWKUI6Vu7o3
25. https://www.aliexpress.com/item/1005004891648244.html?spm=a2g0o.productlist.main.11.10c8118d4Ock5J&algo_pvid=98beacd2-35b2-4fb0-8ea2-5e19191d1318&algo_exp_id=98beacd2-35b2-4fb0-8ea2-5e19191d1318-5&pdp_npi=3%40dis%21EUR%213.08%210.99%21%21%21%21%21%40214528be16858837207541995d076c%2112000030913009138%21sea%21ES%210&curPageLogUid=b07sO2saUeUK

Appendix

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

Servo servo01;
Servo servo02;
Servo servo03;
Servo servo04;
Servo servo05;
Servo servo06;

SoftwareSerial Bluetooth(0,1); // Arduino(RX, TX) - HC-05 Bluetooth (TX, RX)

int servo1Pos, servo2Pos, servo3Pos, servo4Pos, servo5Pos, servo6Pos; //
current position
int servo1PPos, servo2PPos, servo3PPos, servo4PPos, servo5PPos, servo6PPos; //
previous position
int servo01SP[50], servo02SP[50], servo03SP[50], servo04SP[50], servo05SP[50],
servo06SP[50]; // for storing positions/steps
int speedDelay = 20;
int index = 0;
String dataIn = "";

void setup() {
  servo01.attach(2);
  servo02.attach(5);
  servo03.attach(6);
  servo04.attach(9);
  servo05.attach(10);
  servo06.attach(11);
  Bluetooth.begin(38400); // Default baud rate of the Bluetooth module
  Bluetooth.setTimeout(1);
  delay(20);
  // Robot arm initial position
  servo1PPos = 90;
  servo01.write(servo1PPos);
  servo2PPos = 150;
  servo02.write(servo2PPos);
  servo3PPos = 35;
  servo03.write(servo3PPos);
  servo4PPos = 140;
  servo04.write(servo4PPos);
  servo5PPos = 85;
  servo05.write(servo5PPos);
  servo6PPos = 80;
  servo06.write(servo6PPos);
}

void loop() {
  // Check for incoming data
  if (Bluetooth.available() > 0) {
    dataIn = Bluetooth.readString();  // Read the data as string

    // If "Waist" slider has changed value - Move Servo 1 to position
    if (dataIn.startsWith("s1")) {
      String dataInS = dataIn.substring(2, dataIn.length()); // Extract only
the number. E.g. from "s1120" to "120"
      servo1Pos = dataInS.toInt();  // Convert the string into integer
      // We use for loops so we can control the speed of the servo
      // If previous position is bigger then current position
      if (servo1PPos > servo1Pos) {
        for ( int j = servo1PPos; j >= servo1Pos; j--) {   // Run servo down
          servo01.write(j);
          delay(20);    // defines the speed at which the servo rotates
        }
      }
      // If previous position is smaller then current position
      if (servo1PPos < servo1Pos) {
        for ( int j = servo1PPos; j <= servo1Pos; j++) {   // Run servo up
          servo01.write(j);
          delay(20);
        }
      }
      servo1PPos = servo1Pos;   // set current position as previous position
    }

    // Move Servo 2
    if (dataIn.startsWith("s2")) {
      String dataInS = dataIn.substring(2, dataIn.length());
      servo2Pos = dataInS.toInt();

      if (servo2PPos > servo2Pos) {
        for ( int j = servo2PPos; j >= servo2Pos; j--) {
          servo02.write(j);
          delay(50);
        }
      }
      if (servo2PPos < servo2Pos) {
        for ( int j = servo2PPos; j <= servo2Pos; j++) {
          servo02.write(j);
          delay(50);
        }
      }
      servo2PPos = servo2Pos;
    }
    // Move Servo 3
    if (dataIn.startsWith("s3")) {
      String dataInS = dataIn.substring(2, dataIn.length());
      servo3Pos = dataInS.toInt();
      if (servo3PPos > servo3Pos) {
        for ( int j = servo3PPos; j >= servo3Pos; j--) {
          servo03.write(j);
          delay(30);
        }
      }
      if (servo3PPos < servo3Pos) {
        for ( int j = servo3PPos; j <= servo3Pos; j++) {
          servo03.write(j);
          delay(30);
        }
      }
      servo3PPos = servo3Pos;
    }
    // Move Servo 4
    if (dataIn.startsWith("s4")) {
      String dataInS = dataIn.substring(2, dataIn.length());
      servo4Pos = dataInS.toInt();
      if (servo4PPos > servo4Pos) {
        for ( int j = servo4PPos; j >= servo4Pos; j--) {
          servo04.write(j);
          delay(30);
        }
      }
      if (servo4PPos < servo4Pos) {
        for ( int j = servo4PPos; j <= servo4Pos; j++) {
          servo04.write(j);
          delay(30);
        }
      }
      servo4PPos = servo4Pos;
    }
    // Move Servo 5
    if (dataIn.startsWith("s5")) {
      String dataInS = dataIn.substring(2, dataIn.length());
      servo5Pos = dataInS.toInt();
      if (servo5PPos > servo5Pos) {
        for ( int j = servo5PPos; j >= servo5Pos; j--) {
          servo05.write(j);
          delay(30);
        }
      }
      if (servo5PPos < servo5Pos) {
        for ( int j = servo5PPos; j <= servo5Pos; j++) {
          servo05.write(j);
          delay(30);
        }
      }
      servo5PPos = servo5Pos;
    }
    // Move Servo 6
    if (dataIn.startsWith("s6")) {
      String dataInS = dataIn.substring(2, dataIn.length());
      servo6Pos = dataInS.toInt();
      if (servo6PPos > servo6Pos) {
        for ( int j = servo6PPos; j >= servo6Pos; j--) {
          servo06.write(j);
          delay(30);
        }
      }
      if (servo6PPos < servo6Pos) {
        for ( int j = servo6PPos; j <= servo6Pos; j++) {
          servo06.write(j);
          delay(30);
        }
      }
      servo6PPos = servo6Pos;
    }
    // If button "SAVE" is pressed
    if (dataIn.startsWith("SAVE")) {
      servo01SP[index] = servo1PPos;  // save position into the array
      servo02SP[index] = servo2PPos;
      servo03SP[index] = servo3PPos;
      servo04SP[index] = servo4PPos;
      servo05SP[index] = servo5PPos;
      servo06SP[index] = servo6PPos;
      index++;                        // Increase the array index
    }
    // If button "RUN" is pressed
    if (dataIn.startsWith("RUN")) {
      runservo();  // Automatic mode - run the saved steps
    }
    // If button "RESET" is pressed
    if ( dataIn == "RESET") {
      memset(servo01SP, 0, sizeof(servo01SP)); // Clear the array data to 0
      memset(servo02SP, 0, sizeof(servo02SP));
      memset(servo03SP, 0, sizeof(servo03SP));
      memset(servo04SP, 0, sizeof(servo04SP));
      memset(servo05SP, 0, sizeof(servo05SP));
      memset(servo06SP, 0, sizeof(servo06SP));
      index = 0;  // Index to 0
    }
  }
```

```
}

// Automatic mode custom function - run the saved steps
void runservo() {
  while (dataIn != "RESET") {   // Run the steps over and over again until
"RESET" button is pressed
    for (int i = 0; i <= index - 2; i++) {  // Run through all steps(index)
      if (Bluetooth.available() > 0) {      // Check for incomding data
        dataIn = Bluetooth.readString();
        if ( dataIn == "PAUSE") {           // If button "PAUSE" is pressed
          while (dataIn != "RUN") {         // Wait until "RUN" is pressed
again
            if (Bluetooth.available() > 0) {
              dataIn = Bluetooth.readString();
              if ( dataIn == "RESET") {
                break;
              }
            }
          }
        }
        // If speed slider is changed
        if (dataIn.startsWith("ss")) {
          String dataInS = dataIn.substring(2, dataIn.length());
          speedDelay = dataInS.toInt(); // Change servo speed (delay time)
        }
      }
      // Servo 1
      if (servo01SP[i] == servo01SP[i + 1]) {
      }
      if (servo01SP[i] > servo01SP[i + 1]) {
        for ( int j = servo01SP[i]; j >= servo01SP[i + 1]; j--) {
          servo01.write(j);
          delay(speedDelay);
        }
      }
      if (servo01SP[i] < servo01SP[i + 1]) {
        for ( int j = servo01SP[i]; j <= servo01SP[i + 1]; j++) {
          servo01.write(j);
          delay(speedDelay);
        }
      }

      // Servo 2
      if (servo02SP[i] == servo02SP[i + 1]) {
      }
      if (servo02SP[i] > servo02SP[i + 1]) {
        for ( int j = servo02SP[i]; j >= servo02SP[i + 1]; j--) {
          servo02.write(j);
          delay(speedDelay);
      }
    }
    if (servo02SP[i] < servo02SP[i + 1]) {
      for ( int j = servo02SP[i]; j <= servo02SP[i + 1]; j++) {
        servo02.write(j);
        delay(speedDelay);
      }
    }

    // Servo 3
    if (servo03SP[i] == servo03SP[i + 1]) {
    }
    if (servo03SP[i] > servo03SP[i + 1]) {
      for ( int j = servo03SP[i]; j >= servo03SP[i + 1]; j--) {
        servo03.write(j);
        delay(speedDelay);
      }
    }
    if (servo03SP[i] < servo03SP[i + 1]) {
      for ( int j = servo03SP[i]; j <= servo03SP[i + 1]; j++) {
        servo03.write(j);
        delay(speedDelay);
      }
    }

    // Servo 4
    if (servo04SP[i] == servo04SP[i + 1]) {
    }
    if (servo04SP[i] > servo04SP[i + 1]) {
      for ( int j = servo04SP[i]; j >= servo04SP[i + 1]; j--) {
        servo04.write(j);
        delay(speedDelay);
      }
    }
    if (servo04SP[i] < servo04SP[i + 1]) {
      for ( int j = servo04SP[i]; j <= servo04SP[i + 1]; j++) {
        servo04.write(j);
        delay(speedDelay);
      }
    }

    // Servo 5
    if (servo05SP[i] == servo05SP[i + 1]) {
    }
    if (servo05SP[i] > servo05SP[i + 1]) {
      for ( int j = servo05SP[i]; j >= servo05SP[i + 1]; j--) {
        servo05.write(j);
        delay(speedDelay);
      }
```



```
      }
      if (servo05SP[i] < servo05SP[i + 1]) {
        for ( int j = servo05SP[i]; j <= servo05SP[i + 1]; j++) {
          servo05.write(j);
          delay(speedDelay);
        }
      }

      // Servo 6
      if (servo06SP[i] == servo06SP[i + 1]) {
      }
      if (servo06SP[i] > servo06SP[i + 1]) {
        for ( int j = servo06SP[i]; j >= servo06SP[i + 1]; j--) {
          servo06.write(j);
          delay(speedDelay);
        }
      }
      if (servo06SP[i] < servo06SP[i + 1]) {
        for ( int j = servo06SP[i]; j <= servo06SP[i + 1]; j++) {
          servo06.write(j);
          delay(speedDelay);
        }
      }
    }
  }
}
```

Appendix 1. The Code