ÇUKUROVA UNIVERSITY

ENGINEERING FACULTY

DEPARTMENT OF ELECTRICAL AND ELECTRONICS ENGINEERING

GRADUATION THESIS

DESIGN AND IMPLEMENTATION OF A LOW-COST ROBOTIC ARM FOR BASIC REPETITIVE AUTOMATION

By

2018513018 - Yusuf Serdar Çalışkan

Advisor

Prof. Dr. Mustafa Gök

January 2023

Adana

ABSTRACT

BACHELOR THESIS

There are several potential benefits to using a robotic arm in the home environment. For example, a robotic arm could assist with tasks such as loading and unloading the dishwasher, organizing items in a pantry or closet, helping with the cooking process, or even assisting with household chores such as vacuuming or cleaning windows. In this way, a robotic arm could provide a level of convenience and efficiency in the home, as well as potentially reducing the risk of injury for certain tasks. Overall, the use of a robotic arm in the home environment could potentially improve quality of life and make everyday tasks easier and more efficient.

The goal of this thesis is to design and build a functional robotic arm using low cost components and Arduino UNO as a microcontroller unit. This robotic arm is intended for use in home applications, and as such, it is important that it be both functional and affordable. The research presented in this thesis focuses on the development of a low-budget robotic arm that can perform various tasks, including grasping and manipulation. The design and construction of the robotic arm are described in detail, as well as the challenges and solutions that were encountered during the development process. The results of the project demonstrate that it is possible to build a functional robotic arm using low-cost components.

ACKNOWLEDGMENTS

I would like to express my sincere gratitude to my thesis advisor, Professor Doctorate Mustafa Gök, for their guidance, support, and encouragement throughout this process. Their expertise and insights were invaluable to the development and completion of this thesis.

I am grateful to my friends and family for their support and encouragement throughout this journey.

CONTENTS PAGE

TABLE AND FIGURE LIST

1- INTRODUCTION

In this chapter the general aim of the project and objectives of the thesis are presented.

1.1. Research Motivation

Robotic arms are a valuable tool in industrial settings because they can perform tasks with a high degree of accuracy and efficiency. However, the cost of purchasing and maintaining a robotic arm for use in a home environment is a significant barrier. These devices are often expensive to purchase and require specialized knowledge and expertise to operate and maintain. Despite these challenges, there is still significant potential for using robotic arms in domestic settings.

A robotic arm could be used to assist individuals with disabilities in performing tasks that require fine motor skills, such as grasping objects, writing, or using a computer mouse. It could also be used to perform simple tasks around the house, such as turning off lights, helping with the cooking process or retrieving items from hard to reach places.

This project aims to demonstrate that it is possible to build a low cost robotic arm using low cost components and to showcase the potential for using these types of devices in home applications. The arm we design will be capable of performing basic repetitive automation tasks and will serve as a proof of concept for the feasibility of using inexpensive robotic arms in domestic settings.

1.2. Objectives

Objectives of a robotic arm that has been built for demonstration purposes for this project are as follows.

- Ability of grasping objects.
- Position manipulation of grasped objects.
- Building a simple computer application for controlling the robotic arm.
- Making repetitive automation that moves an object from one place to another.
- Implementing safety features.

2- EQUIPMENTS USED IN THIS PROJECT

In this chapter equipment and materials used in creation of robotic arm are listed. Each of the equipment's working principles and what they do in this system are explained.

2.1. Arduino Uno

Arduino Uno is a microcontroller board based on the ATmega328 microcontroller. It is a popular platform for building electronic projects because it is inexpensive, easy to use, and has a large community of users who share their knowledge and resources.

The ATmega328 microcontroller is a central processing unit (CPU) that executes instructions stored in its memory. It contains two types of memory: volatile memory (RAM) and non-volatile memory (ROM). The ROM stores the program code, which tells the microcontroller what to do, and the RAM stores temporary data that the microcontroller uses while it is executing the program.

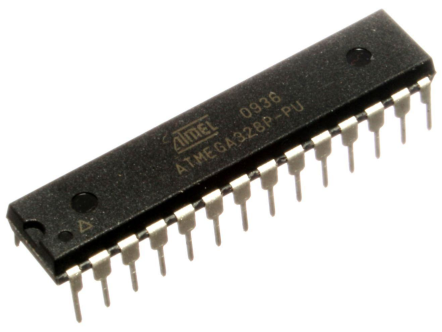


Figure 2.1.1 ATmega328 microcontroller

The Arduino Uno has several input and output (I/O) pins that allow it to communicate with other devices. These pins can be used to read data from sensors, control motors, and display information on a screen. The pins can be programmed to perform different functions, such as reading a temperature sensor or controlling the speed of a motor.

The Arduino Uno can be programmed using a computer and a USB cable. The Arduino Integrated Development Environment (IDE) is a software tool that allows users to write, upload, and debug their code. Code in IDE is written in C++ programming language.

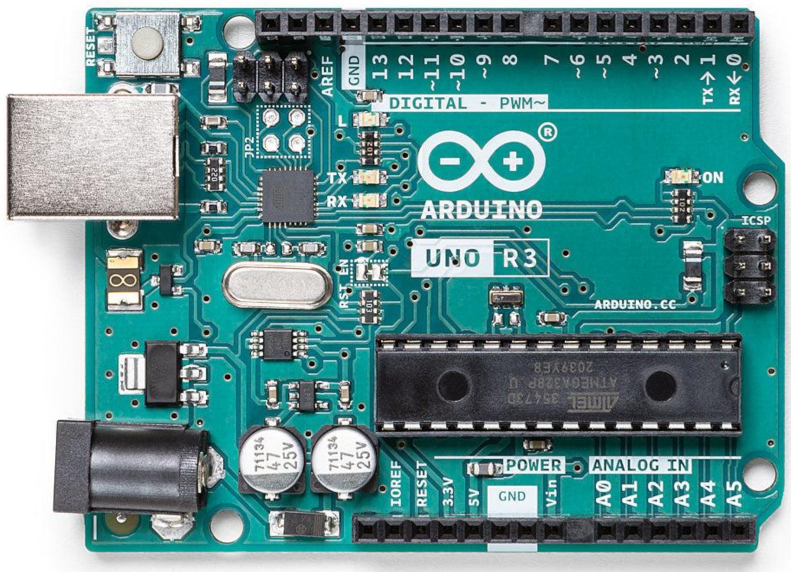


Figure 2.1.2: Arduino Uno

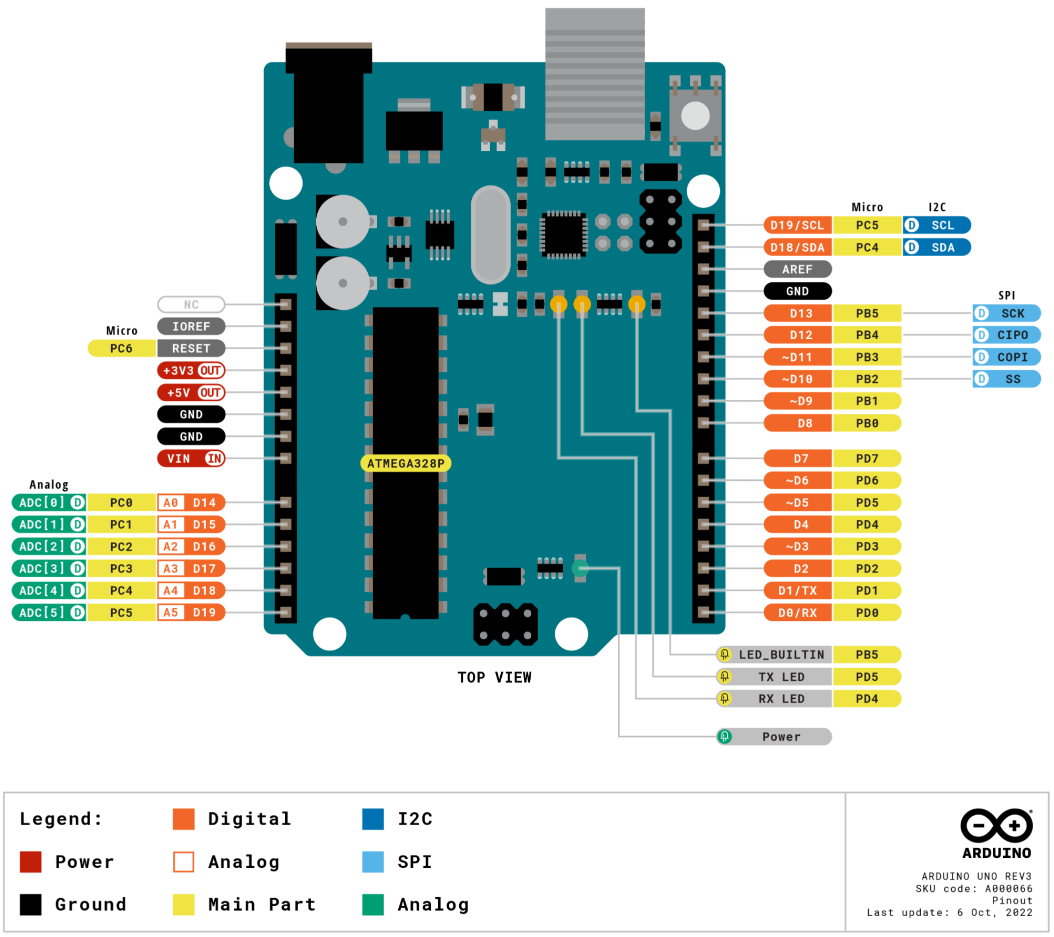


Figure 2.1.3: Arduino Uno Pin Layout

Board	**Name**	Arduino UNO R3
	SKU	A000066
Microcontroller	ATmega328P	
USB connector	USB-B	
Pins	**Built-in LED Pin**	13
	Digital I/O Pins	14
	Analog input pins	6
	PWM pins	6
Communication	**UART**	Yes
	I2C	Yes
	SPI	Yes
Power	**I/O Voltage**	5V
	Input voltage (nominal)	7-12V
	DC Current per I/O Pin	20 mA
	Power Supply Connector	Barrel Plug
Clock speed	**Main Processor**	ATmega328P 16 MHz
	USB-Serial Processor	ATmega16U2 16 MHz
Memory	**ATmega328P**	2KB SRAM, 32KB FLASH, 1KB EEPROM
Dimensions	**Weight**	25 g
	Width	53.4 mm
	Length	68.6 mm

Table 2.1.1: Arduino Uno Tech Specs

2.1.1. Connections of Arduino Uno

The Arduino Uno is a microcontroller board based on the ATmega328 microcontroller. It has a total of 20 digital input/output pins, 6 of which can be used as pulse width modulation (PWM) outputs and 6 of which can be used as analog inputs. The digital pins can be used to interface with a variety of sensors, actuators, and other devices, while the analog pins can be used to read analog signals from sensors or to generate analog outputs using pulse width modulation.

The digital pins on the Arduino Uno are labeled 0-13 and are located on the left side of the board. Digital pins 0 and 1 are reserved for serial communication, while the remaining digital pins can be used as general purpose input/output (GPIO) pins. The digital pins can be configured as input or output, and they can be used to read digital signals or to control devices that use digital signaling.

The analog pins on the Arduino Uno are labeled A0-A5 and are located on the right side of the board. These pins can be used to read analog signals from sensors or to generate analog outputs using pulse width modulation. The analog pins can be configured as either inputs or outputs, and they can be used to read analog signals such as temperature, light, or sound, or to control devices that use analog signaling.

In addition to the digital and analog pins, the Arduino Uno also has a number of other pins and connectors, including a power connector, a USB connector, and a reset button. These features allow the Arduino Uno to be powered and programmed using a variety of different methods and to be easily integrated into a wide range of projects and applications.

In the robotic arm project arduino uno is responsible for controlling servo motors by processing the data sent by the serial communication. Total of 4 pins are digital pins used for controlling 4 servo motors in the robotic arm. One ground pin is connected to the negative sides of alkaline batteries through a breadboard to create common ground. The USB connector is connected to a computer for both powering arduino uno and using serial communication to control robotic arm.

2.1.2 Communication with Arduino Uno

The Arduino Uno is equipped with a hardware serial port, which is located on digital pins 0 (RX) and 1 (TX) and USB connection. This serial port can be used to communicate with other devices that use serial communication, such as computers, GPS units, and sensors.

There are a few important considerations to keep in mind when using the serial port on the Arduino Uno:

1. The baud rate must be the same on both the transmitting and receiving devices. If the baud rates are different, the devices will not be able to communicate with each other.
2. The serial port can only transmit one character at a time, so if you want to send a longer message, you will need to send it one character at a time and wait for the transmission to complete before sending the next character.
3. The serial port uses a buffer to store incoming data, so you may need to use a function to check if there is data available to be read before calling the a read function.

In this project communication is made between a computer with windows operating system and arduino uno. Purpose of this communication is to control robotic arm behavior with computer keyboards. With this you can control 4 axis of the robotic arm with “W”, “A”, “S”, “D” and arrow keys. It is also possible to control automation of robotic arms via other keys, explained in “3.2. Software Architecture and Design”.

2.2. SG90 Servo Motor

The SG90 is a small, low-cost servo motor that is commonly used in hobby robotics and other applications. It consists of a motor, gear train, and control circuit, all housed in a plastic case.

The motor rotates a shaft, which is connected to a gear train that reduces the speed and increases the torque of the motor. The gear train is connected to a control horn, which is a small lever that protrudes from the servo case. When the motor rotates, it moves the control horn back and forth. Shaft can rotate 180 degrees in total.

The control circuit of the servo motor consists of a pulse width modulation (PWM) control signal, which is sent to the servo from a microcontroller or other control device. The pulse width of the control signal determines the position of the control horn. For example, a pulse with a width of 1.5 milliseconds will cause the servo to move the control horn to its neutral position, while a pulse with a width of 1.0 milliseconds will cause the servo to rotate the control horn to its maximum clockwise position, and a pulse with a width of 2.0 milliseconds will cause the servo to rotate the control horn to its maximum counterclockwise position.

To move the control horn to a specific position, the microcontroller or control device sends a series of pulses to the servo at a regular interval, usually around 50 times per second. The servo uses the control circuit to compare the position of the control horn to the position specified by the pulse width of the control signal, and adjusts the motor accordingly to move the control horn to the desired position.

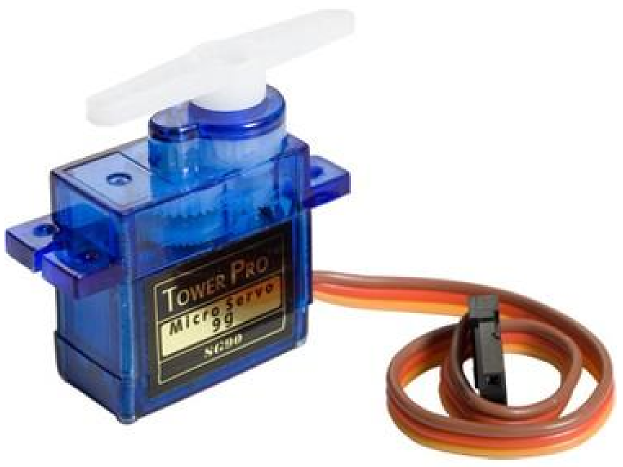


Figure 2.2.1: SG90 Servo Motor

In this project servo is controlled by arduino uno with help of built in servo library and arduino pins with PWM capability. SG90 servo motors can draw around 360 mA in moments of stall or high load. Since arduino 5 volt pins can only supply as high as 0.8 mA, these 4 servo motors are powered with 2 parallelly connected 6 volt batteries (x4 AA 1.5 volt Alkaline battery).

Model	SG90
Weight(gm)	9
Operating Voltage (VDC)	3.0 ~ 7.2
Operating Speed @4.8V	0.10sec/60°
Stall Torque @ 4.8V (Kg-Cm)	1.2
Stall Torque @6.6V (Kg-Cm)	1.6
Operating Temperature (°C)	-30 to 60
Dead Band Width (μs)	7
Gear Type	Glass Fiber
Rotational Degree	180º
Servo Plug	JR
Cable Length (cm)	25
Length (mm)	22.8
Width (mm)	12.6
Height (mm)	34.5
Shipment Weight	0.014 kg
Shipment Dimensions	6 × 6 × 4 cm

Table 2.2.1: SG90 Specifications

2.3. Breadboard

A breadboard is a prototyping tool used to build and test circuits. It consists of a plastic board with a series of holes in it, into which you can insert wires and electronic components to build a circuit. The breadboard has a series of metal clips underneath the holes, which hold the components and wires in place and allow them to be easily connected together.

The breadboard is an essential tool for prototyping and experimenting with different circuit designs. It allows you to easily build and modify circuits without the need for soldering. In this project breadboard is used for wiring and powering components.

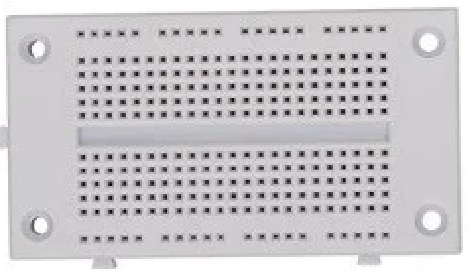


Figure 2.3.1: Breadboard

2.4. Jumper Cable

A jumper cable is a short wire with connectors on either end that is used to make temporary connections between electronic components or points on a circuit. Jumper cables are often used with breadboards, which are prototyping tools that allow you to build and test circuits without the need for soldering. Usage of jumper cables in the project makes it easier to make connections to both arduino and servo motors.

Figure 2.4.1: Jumper Cables

2.5. AA 1.5 Volt Alkaline Battery

An AA 1.5 Volt Alkaline Battery is a type of battery that is commonly used in portable electronic devices such as radios, flashlights, and remote controls. It is called an "AA" battery because of its size and shape, which is similar to that of a standard AA dry cell battery. The "1.5 Volt" designation refers to the voltage of the battery, which is the electrical potential difference between its two terminals.

Alkaline batteries are a type of primary battery, which means that they are not rechargeable and must be replaced when they are depleted. They are called "alkaline" because they use an alkaline electrolyte solution as their cathode material, which allows them to provide a higher voltage and longer shelf life than other types of primary batteries.

AA 1.5 Volt Alkaline Batteries are known for their high energy density and long shelf life, making them a popular choice for portable electronic devices. They are widely available and can be found at most electronics stores and online retailers.

In this project a total of 8 of these batteries are used. Batteries are connected in series of 4 to create near 6 voltage power sources. As mentioned before, servo motors can draw more than 1A of current so in order to solve power bottleneck these batteries that are connected in series of 4 are connected in parallel to power the system. Standart x4 battery holder is used to connect these batteries in series.

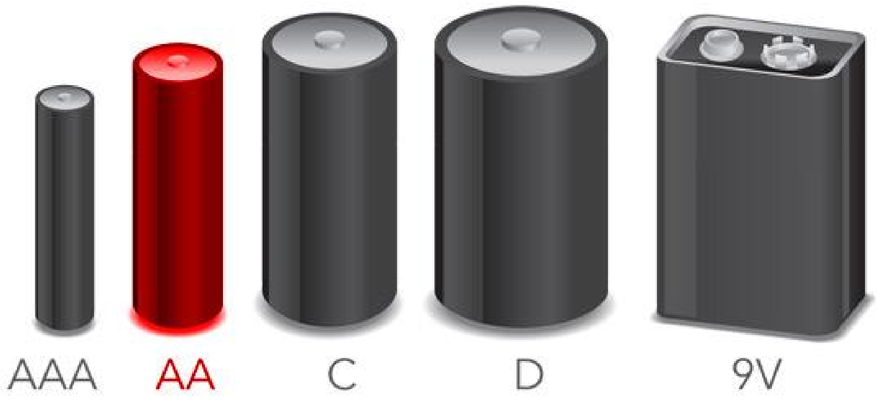


Figure 2.5.1: Classifications of batteries

Figure 2.5.2: x4 AA Battery Holder

2.6. Computer with Windows OS

In this project robotic arm is controlled via computer keyboard. Arduino Uno is connected to the computer via USB connector and serial communication is made between the arduino and the computer. A python script is used for taking inputs from a computer keyboard and sending them via serial communication. Computer require python installed to run the script, “pyserial” python library for serial communication, “win32api” python library for watching which keyboard keys are pressed by user and windows operating system for this to work.

2.6.1 Python

Python is a high-level, interpreted programming language that is widely used for web development, artificial intelligence, data analysis, and scientific computing. It was created by Guido van Rossum in the late 1980s and has a design philosophy that emphasizes code readability and a syntax that allows programmers to express concepts in fewer lines of code than would be possible in languages such as C++ or Java.

Figure 2.6.1.1: Python Logo

Python is an interpreted language, which means that it is not compiled into machine code that can be directly executed by a computer. Instead, it is executed by an interpreter, which reads the Python code and executes the corresponding instructions. This makes Python a popular choice for rapid prototyping and development, as it allows developers to quickly write and test code without the need to go through a separate compilation step.

Python is a general purpose programming language that is used in a wide variety of applications, including web development, scientific computing, data analysis, machine learning, and artificial intelligence. It has a large and active community of users and developers, and it is supported by a wealth of libraries and frameworks that make it easier to develop a wide range of applications. Due to this very reason it's used in this project.

2.6.2 Pyserial

PySerial is a Python library that provides support for serial communication using the Python programming language. It allows Python scripts to communicate with devices that use a serial interface, such as microcontrollers, sensors, and other devices that use a serial protocol to communicate with a host computer.

Figure 2.6.2.1: Pyserial Logo

PySerial provides a set of functions and classes that allow Python scripts to access the serial port on a computer and exchange data with connected devices. It includes functions for reading and writing data, setting the baud rate and other communication parameters, and checking the status of the serial port.

PySerial is commonly used to interface with microcontrollers and other devices that use a serial protocol for communication, such as those based on the Arduino platform. It is also used in a variety of other applications that require serial communication, including scientific computing, data acquisition, and automated testing. PySerial is available as a free and open source library, and it is compatible with most versions of Python.

2.6.3 Win32api

The win32api module is a Python library that provides access to the Win32 API (Application Programming Interface) on Microsoft Windows operating systems. The Win32 API is a set of functions and data types that allows developers to create Windows-based applications.

The win32api module provides a thin wrapper around the Win32 API, exposing a subset of the API's functions and data types to Python scripts. It allows Python scripts to perform a variety of tasks that are typically associated with the Win32 API, such as interacting with the operating system, working with files and directories, and creating and manipulating windows.

The win32api module is typically used in Python scripts that need to interact with the Windows operating system or perform tasks that are specific to the Windows platform. It is part of the pywin32 package, which is a collection of Python modules that provide access to various aspects of the Win32 API.

In this project it is used for watching with keyboard input given by the user using the ability to read windows virtual key codes. Virtual key codes are codes that represent the keys on a keyboard or other input device. In the Windows operating system, virtual key codes are used to identify the keys that are pressed or released in a keyboard event.

3- SYSTEM DESIGN AND IMPLEMENTATION

In this chapter design and implementation details of the project are covered.

3.1. Robotic Arm Mechanical Structure

Industrial robotic arms are complex mechanical systems that are used to automate a wide range of tasks in manufacturing, assembly, and other industrial applications. These systems are designed to be robust and precise, and they typically consist of several key components that work together to provide the necessary motion and control.

At the heart of an industrial robotic arm is the mechanical structure, which provides the framework for the other components and defines the range of motion and capabilities of the arm. The mechanical structure typically consists of a series of links, or segments, that are connected by joints. These links and joints are typically made of metal, such as aluminum or steel, and they are designed to be strong and durable.

The links and joints of an industrial robotic arm are typically actuated by motors or hydraulic cylinders, which provide the necessary force to move the arm. The motors or cylinders are controlled by a computer or microcontroller, which receives input from sensors and calculates the necessary control signals to move the arm.

In addition to the mechanical structure, an industrial robotic arm may also include a number of other components, such as end effectors (tools or grippers), sensors, and vision systems. These components work together with the mechanical structure to enable the arm to perform a wide range of tasks, such as welding, assembly, painting, and material handling.

Since this project's main objective is to create proof of concept that low cost robotic arms can perform basic tasks, the structure of the robotic arm is made out of lightweight wooden pieces that are glued together with servo motors. And since only repetitive automation is aimed, there is no need for sensors or vision systems. In order to grab an object a basic claw is used.

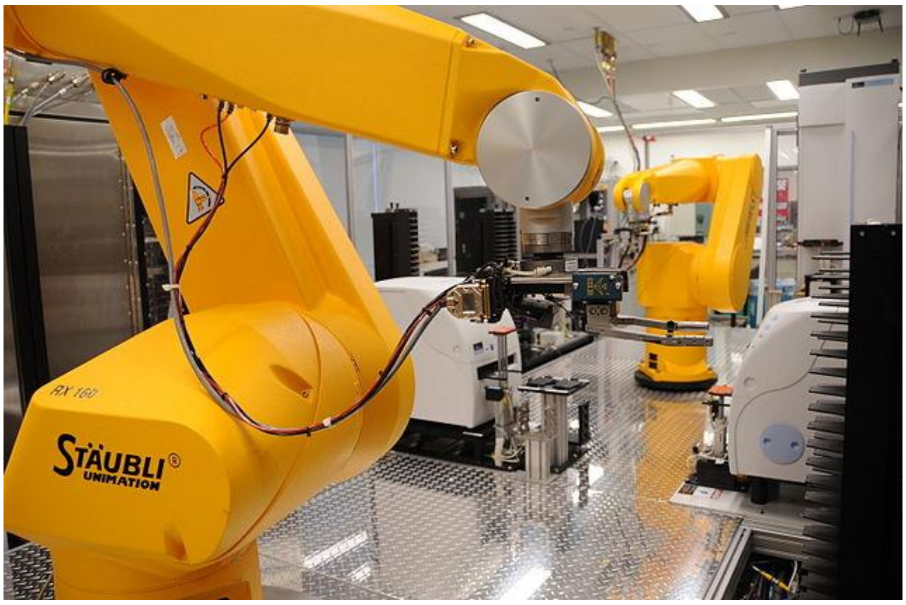


Figure 3.1.1: A Industrial Robotic Arm

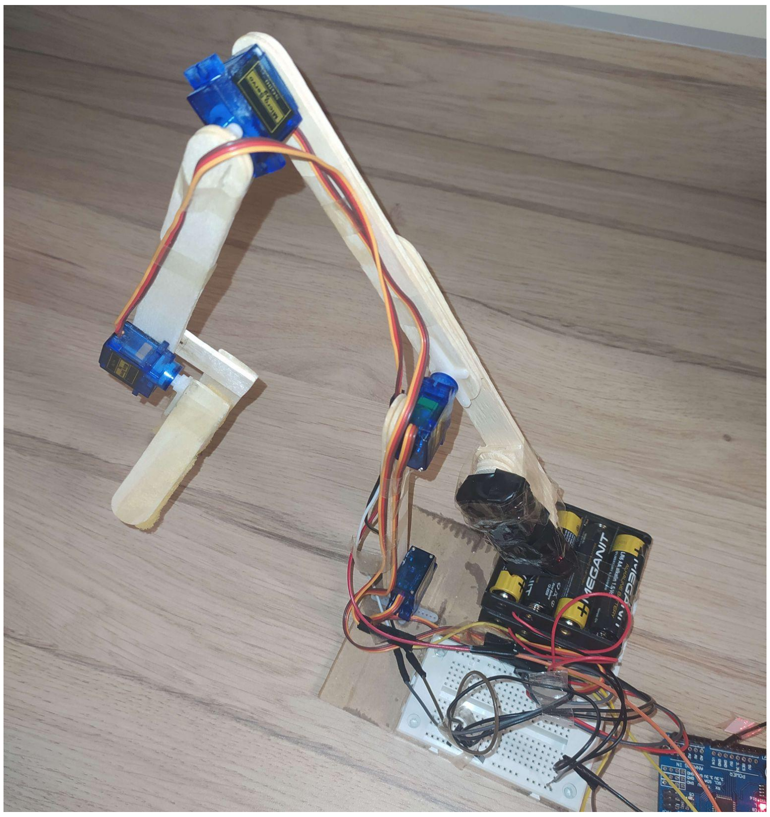


Figure 3.1.2: Photograph of robotic arm

3.1.1 Structure of the Claw

A simple robotic arm claw is a mechanical device that is used to grasp and manipulate objects. It consists of a frame that is mounted on a robotic arm, and it has a series of fingers or jaws that are used to grasp the object.

In this project 2 wooden pieces are used to create claws for the robotic arm. One of them is directly glued to the main arm while the other one is connected to a servo motor to pinch an object between itself and another wooden piece that is glued to the main arm.

In order to increase friction between the object that is going to be grasped and the claw, a small piece of sponge is attached to the end of the claw. After prototyping and testing it can be said this increased ability of grasping objects significantly.

Grasping ability can be further increased by using higher quality gripper, force sensor for gripper, vision system and closed loop feedback system to adjust gripper accordingly to maintain a precise grip.

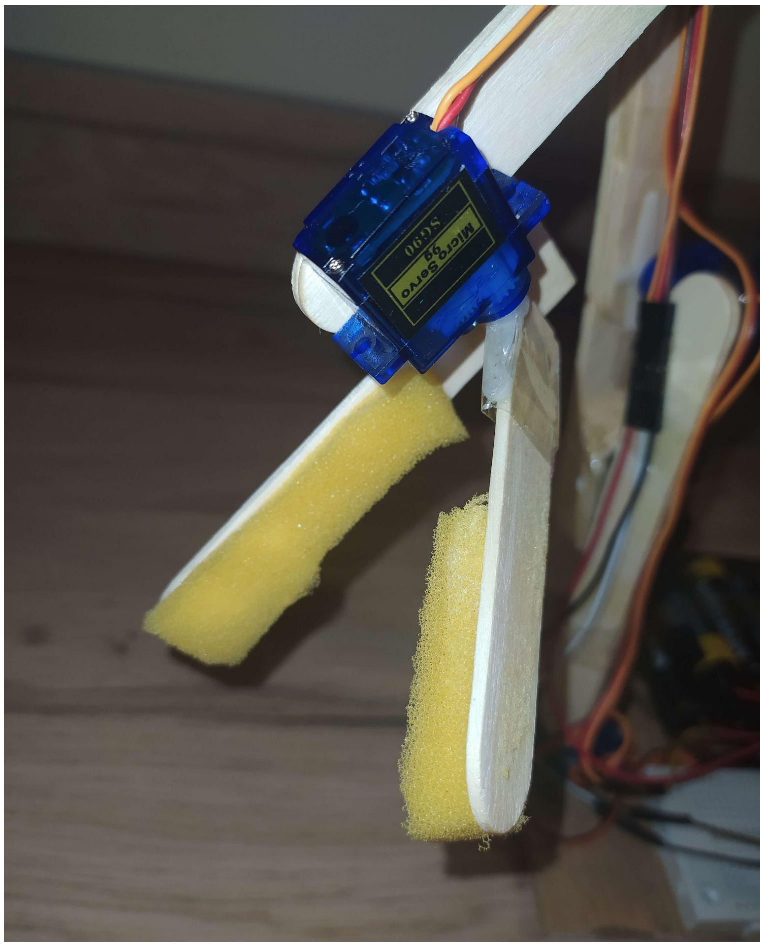


Figure 3.1.1.1: Photograph of the claw

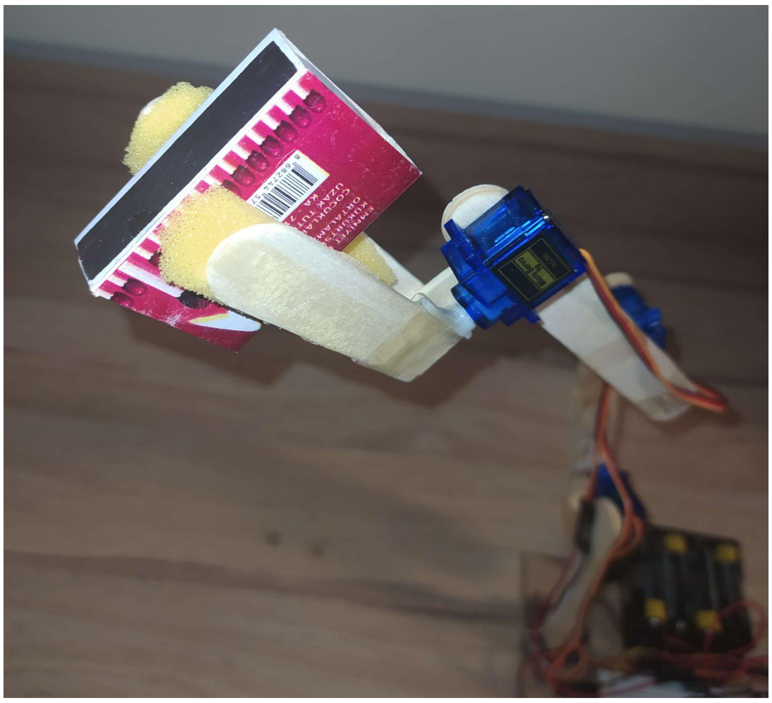


Figure 3.1.1.2: Photograph of the claw holding a matchbox

3.1.2 Structure of the Base

The base of a robotic arm is the foundation of the mechanical structure and provides the necessary stability and support for the arm to operate. The design of the base depends on the size and weight of the arm, as well as the type of application it will be used for.

In general, the base of a robotic arm is made of a sturdy and rigid material, such as steel or aluminum, and it is typically designed to be stable and resistant to vibrations and external forces. The base may also include mounting points or fixtures for attaching the arm to a workbench or other surface.

The base of a robotic arm is typically connected to the rest of the arm through a series of joints or bearings, which allow the arm to move and rotate. These joints may be actuated by motors or hydraulic cylinders, which provide the necessary force to move the arm.

In addition to providing stability and support, the base of a robotic arm may also include features such as cable management systems and safety guards to protect the arm and its surroundings. The base is an important part of the mechanical structure of a robotic arm and plays a crucial role in its overall performance and reliability.

In this project a servo motor glued to the base and the robotic arm shaft makes it possible to rotate the whole robotic arm by 180 degrees. This is the weakest part of the design as a simply glued servo motor is carrying the whole weight of the robotic arm and the load. Since this robot arm should be portable for presentation of this thesis it's made so it doesn’t require any connections such as screwing it to a fixed place. It solely relies on the weight of the components placed on the base such as batteries, arduino uno.

3.1.3 Structure of the Arm

The number of axes, or degrees of freedom, in a robotic arm refers to the number of dimensions in which the arm can move. The structure of a robotic arm is designed to provide the necessary range of motion and control to enable the arm to perform a variety of tasks.

A typical robotic arm has at least three axes of motion:

1. Base rotation: This axis allows the arm to rotate around a vertical axis, which is typically located at the base of the arm.
2. Shoulder: This axis allows the arm to move up and down, or to pivot around a horizontal axis that is located near the shoulder of the arm.
3. Elbow: This axis allows the arm to bend or flex at the elbow, which is typically located near the middle of the arm.

Some robotic arms may have additional axes of motion, such as a wrist that allows the arm to rotate around its own axis or a gripper that allows the arm to open and close. The number of axes in a robotic arm is determined by the requirements of the application and the capabilities of the arm.

The structure of a robotic arm consists of a series of links or segments that are connected by joints or bearings. The links and joints are typically made of metal, such as aluminum or steel, and they are designed to be strong and durable. The joints may be actuated by motors or hydraulic cylinders, which provide the necessary force to move the arm. The arm is typically mounted on a base, which provides stability and support, and it may include features such as cable management systems and safety guards to protect the arm and its surroundings.

In this project there are a total of 4 servo motors used for 4 axes. These are base rotation, shoulder, elbow and claw of the robotic arm. Maneuverability capabilities are as follows:

1. Base axis: Can rotate 180 degrees whole arm without any restrictions.
2. Shoulder: Can rotate elbow and any part above it by 100 degrees. While the SG90 servo motor is capable of rotation 180 degrees, physical obstacles in robotic arm design restrict this movement.
3. Elbow: Can rotate 180 degrees freely as long as there are not any physical obstacles on the way.
4. Claw: Free moving part of the claw can rotate as much as 180 degrees. This movement can be restricted by any physical obstacle such as an object that is being graped.

12 centimeter lightweight wooden pieces extend from base, shoulder and elbow creating basic structure of the robotic arm. Two pieces are glued in together to create a sturdier structure. A 6 cm long lightweight wooden piece is used for claw structure.

Since the whole weight of the robotic arm relies on the shoulder of the robotic arm a counter weight is placed 5 cm behind the shoulder of the robotic arm to reduce stress and current draw on the shoulder servo motor. A total load of 50 grams is used.

3.2. Software Architecture and Design

Software is separated into two pieces. One being for user input which is a python script that's running on the computer. Other one being arduino code which is used for controlling robotic arms.

3.2.1 Python Script

Python script is runned in a computer with windows operating system. System should have python installed as well as “pyserial” and “win32api” libraries installed in the system.

Details of “pyserial” and “win32api” are explained in “2.6.1 Pyserial” and “2.6.3 Win32api”. In summary pyserial is responsible for handling serial communication and win32api is responsible for collecting keyboard inputs from the user.

Keyboard keys assigned for control of the robotic arm as follows:

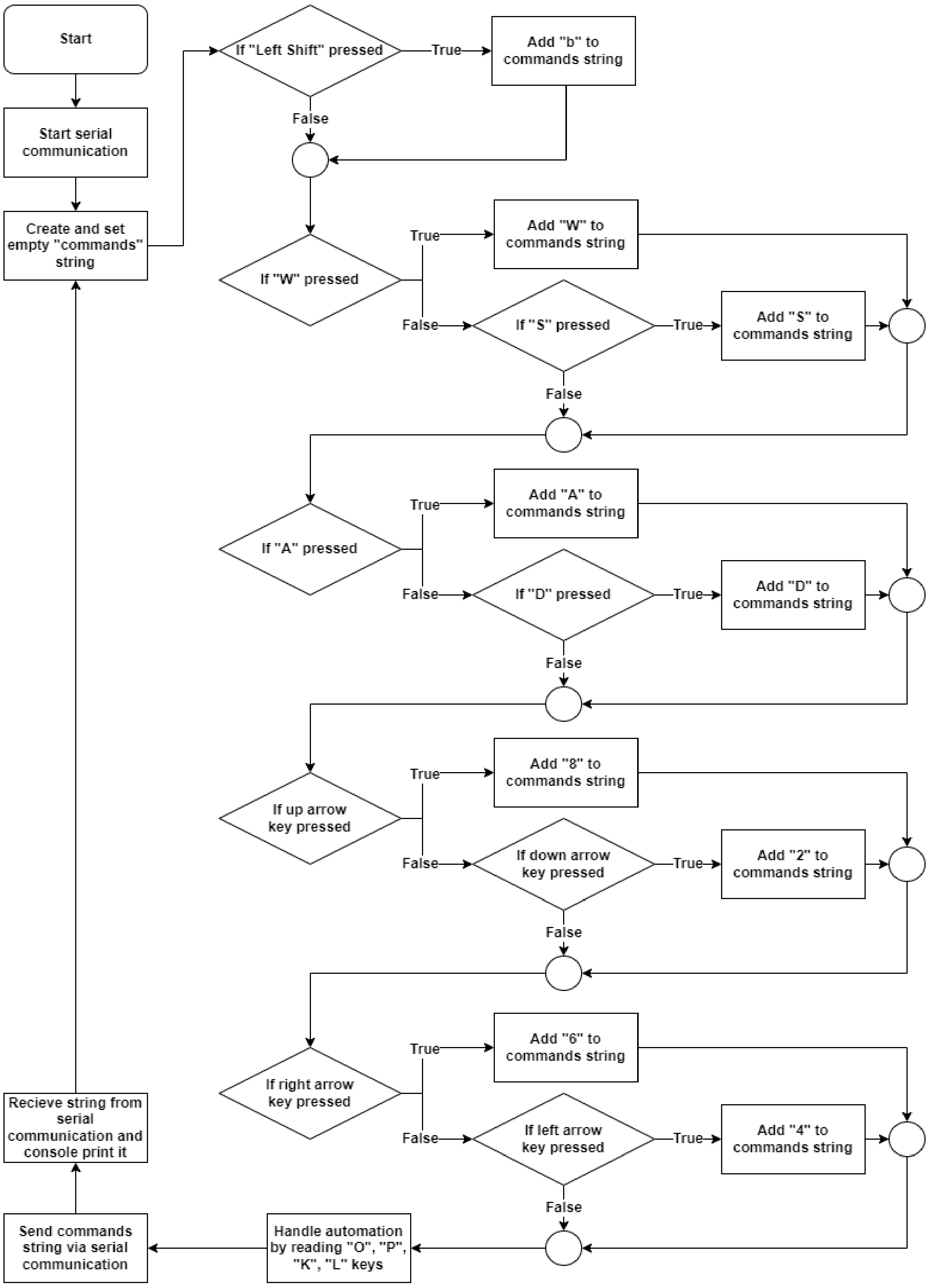
1. Left Shift: Doubles the speed of the robotic arm for all axes as long as pressed.
2. “A” and “D” key: Rotates the base axis.
3. “W” and “S” key: Rotates the shoulder axis.
4. Up and Down arrow key: Rotates the elbow axis.
5. Right and Left arrow key: Rotates the claw axis.
6. “K” key: Save position to memory.
7. “L” key: Clear position memory.
8. “O” key: Start automation.
9. “P” key: Pause automation.

After initializing libraries an infinite loop is called in python script. At the start of each loop a new empty “commands” string is created. Then keys related to robotic arm movement control (without automation) are read and if pressed this letter is added to the “commands” string.

If the “commands” string is still empty after reading all movement related keys then the “startAutomation” boolean is set to “False”. If the automation start key (“O”) is pressed or the “startAutomation” boolean is “True” then the “O” letter will be added to the “commands” string and the “startAutomation” boolean will be set to “True”. This way if any other control key is pressed during automation, the robotic arm will abort the automation process and return to manual control.

After reading the “O” key; “K”, “L” and “P” keys will be read and the “commands” string will be altered in the same way done in previous steps.

Working principle of python code is simplified and summarized in figure 3.2.1.1. Raw version of the python code is included in the appendix.



Start
Start serial communication
Create and set empty "commands" string
If "Left Shift" pressed
True
Add "b" to commands string
False
If "W" pressed
True
Add "W" to commands string
False
If "S" pressed
True
Add "S" to commands string
False
If "A" pressed
True
Add "A" to commands string
False
If "D" pressed
True
Add "D" to commands string
False
If up arrow key pressed
True
Add "8" to commands string
False
If down arrow key pressed
True
Add "2" to commands string
False
If right arrow key pressed
True
Add "6" to commands string
False
If left arrow key pressed
True
Add "4" to commands string
False
Handle automation by reading "O", "P", "K", "L" keys
Send commands string via serial communication
Recieve string from serial communication and console print it

Figure 3.2.1.1: Python flowchart

3.2.2 Arduino Uno Software

Arduino uno software is responsible for controlling the robotic arm movements via sending pulse width modulation signals to servo motors by processing the data sent from serial communication.

Code starts by including libraries that are required for the system. Only library included in the code is the “servo.h” library which is found in Arduino's own IDE. Using the “servo” class from this library, 4 servo objects are created and set to their default angles. Serial communication gets started using 9600 kbps baud rate with 10 ms timeout. An array with length of 80 is created to be used as automation positions memory. This array can hold a total of 20 positions. All array index values are set to “-1” to indicate its empty. After all of these the main loop is called.

At the start of each loop serial connection is checked and if it is not found then the loop is paused until serial connection is achieved. This is done so the robotic arm will stop all of its functions for safety when connection with the user is severed.

First and most important part of the loop is calling the “SetAngleFromKeyboard” function. This function is responsible for reading the “commands” string that is sent by python script via serial communication and extracting its content to related variables.

“SetAngleFromKeyboard” function first checks if automation start or stop commands are sent and sets “runAutomation” boolean according to it. If the save position command is sent and if true then it calls the “SavePostionToMemory” function which adds current robotic arm axis values to the memory array variable. If a command for deleting position memory is called then a function that sets all array index values to “-1” is called. If a manual control command is sent then servo angles will be updated in a variable. This variable will be later written into servo.

Second part of the code is responsible for handling the automation process of the robotic arm. If the “runAutomation” boolean is set “True” in the first part then the “StartAutomation” function will be called. For each iteration of the main loop this function will read target values from the memory array and update angle variables by one for each servo. When the target value is reached, the next target value will be read from the memory array and the same process will repeat itself.

Third and last part of the code is updating PWM signals for servos. This is done in the “SetAngleToServo” function by reading angle variables and using “Servo.write(int value)” command. Just before writing values, variables are clamped between 0 and maximum rotation limit of the servo that is specified in “3.1.3 Structure of the Arm”.

Simplified flow chart diagrams are given below. Raw version of the code is included in the appendix.

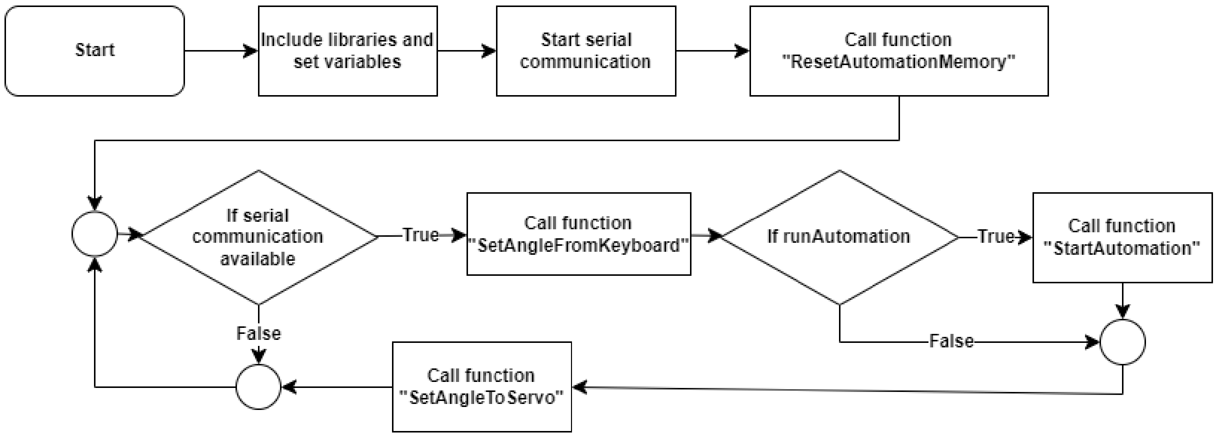

Start
Include libraries and set variables
Start serial communication
Call function "ResetAutomationMemory"
If serial communication available
True
Call function "SetAngleFromKeyboard"
If runAutomation
True
Call function "StartAutomation"
False
False
Call function "SetAngleToServo"

Figure 3.2.2.1: Main loop flowchart

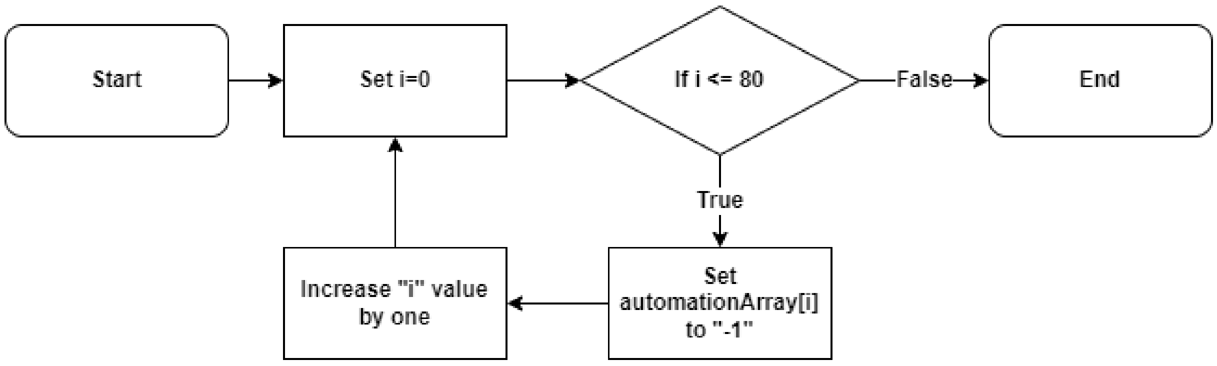


Figure 3.2.2.2: ResetAutomationMemory flowchart

4- CONCLUSION

In this thesis a robotic arm that is able to manipulate position of objects with manual control and repetitive automation is done. Manufacturing a robotic arm with low cost is definitely possible but doing so brings disadvantages such as fragile arm structure, low grasping capability and lower weight lifting capability. Yet all objectives set are achieved such as manual control, automation and safety features. This means production of robotic arms for domestic usage is within reality. These robotic arms can help human daily lives with activities such as cleaning or cooking.

REFERENCES

[1] Official arduino website and documentation, https://www.arduino.cc/

[2] Pyserial python library, https://pypi.org/project/pyserial/

[3] Win32api python library, https://pypi.org/project/pywin32/

[4] Public image source, https://commons.wikimedia.org/wiki/Main_Page

[5] Battery performance tests, https://rightbattery.com/

[6] SG90 servo specifications, https://www.towerpro.com.tw/product/sg90-7/

[7] Official python website and documentation, https://www.python.org/

[8] C++ reference, https://www.w3schools.com/cpp/

[9] Windows virtual keycodes,
https://learn.microsoft.com/en-us/windows/win32/inputdev/virtual-key-codes

APPENDIX

1_ Python code

```
## Key codes in:
https://learn.microsoft.com/en-us/windows/win32/inputdev/virtual-key-codes
import time
import win32api
import serial

startAutomation = False
automationKeyPress = 0

# Open and set serial port for
arduino = serial.Serial('com3', 9600, timeout=0.01)

while 1:
        commands = ""
        # Detect "left shift" key for speed boost
        if win32api.GetKeyState(0xA0) < 0:
                commands += "b"

        # Detect "W" key for lower arm
        if win32api.GetKeyState(0x57) < 0:
                commands += "w"
        # Detect "S" key for lower arm
        elif win32api.GetKeyState(0x53) < 0:
                commands += "s"

        # Detect "A" key for base arm
        if win32api.GetKeyState(0x41) < 0:
                commands += "a"
        # Detect "D" key for base arm
```

```
elif win32api.GetKeyState(0x44) < 0:
        commands += "d"

# Detect "UP ARROW" key for upper arm
if win32api.GetKeyState(0x26) < 0:
        commands += "8"
# Detect "DOWN ARROW" key for upper arm
elif win32api.GetKeyState(0x28) < 0:
        commands += "2"

# Detect "RIGHT ARROW" key for claw arm
if win32api.GetKeyState(0x27) < 0:
        commands += "6"
# Detect "LEFT ARROW" key for claw arm
elif win32api.GetKeyState(0x25) < 0:
        commands += "4"



if commands != "":
        startAutomation = False
# Start automation
if win32api.GetKeyStatea(0x4F) < 0 or startAutomation == True:
        commands += "o"
        startAutomation = True
if win32api.GetKeyState(0x50) < 0:
        commands += "p"
        startAutomation = False
if win32api.GetKeyState(0x4C) < 0 and time.time() > automationKeyPress + 1:
        commands += "l"
        startAutomation = False
        automationKeyPress = time.time()
elif win32api.GetKeyState(0x4B) < 0 and time.time() >automationKeyPress+1:
        commands += "k"
        startAutomation = False
        automationKeyPress = time.time()




# If there is key input send this string via serial port.
if commands != "":
        print("sending: "+commands)
        arduino.write(bytes(commands, 'utf-8'))
        data = arduino.readline()
        print(data.decode('utf-8'))
```

2_ Arduino code

```
#include <Servo.h>

Servo baseArm;
Servo lowerArm;
Servo upperArm;
Servo clawArm;

String input;

int baseAngle = 90;
int lowerAngle = 140;
int upperAngle = 90;
int clawAngle = 90;

volatile int turnSpeed = 1;
const int turnBoostConstant = 2;
volatile bool runAutomation = false;
volatile int automationArray[80];
volatile int automationStep = 0;
volatile int nextAutomationSaveStep = 0;

void setup() {
  pinMode(LED_BUILTIN, OUTPUT);

  baseArm.attach(3);
  lowerArm.attach(5);
  upperArm.attach(6);
  clawArm.attach(9);

  baseArm.write(baseAngle);
  lowerArm.write(lowerAngle);
  upperArm.write(upperAngle);
  clawArm.write(clawAngle);

  Serial.begin(9600);
  Serial.setTimeout(10);

  ResetAutomationMemory();
}

void loop() {
  // Wait until there is serial port connection
  while (!Serial.available()) {
    digitalWrite(LED_BUILTIN, HIGH);
  }
  digitalWrite(LED_BUILTIN, LOW);

  // Increase or decrease arm angles by checking input that is been sent by serial port
  SetAngleFromKeyboard();

  if (runAutomation == true) {
    StartAutomation();
  }

  // Write this angles to servo arms.
  SetAngleToServo();
}

void SetAngleFromKeyboard() {
  input = Serial.readStringUntil('\n');

  // Start automation
  if (input.indexOf("p") != -1) {
```

```
  StopAutomation();
} else if (input.indexOf("o") != -1) {
  runAutomation = true;
} else {
  runAutomation = false;
}

// Save automation postion
if (input.indexOf("k") != -1) {
  SavePositionToMemory();
  runAutomation = false;
}
if (input.indexOf("l") != -1) {
  ResetAutomationMemory();
  runAutomation = false;
}

// Stop automation

//set turn speed (boosted if shift is pressed)
if (input.indexOf("b") != -1) {
  turnSpeed = turnBoostConstant;
} else {
  turnSpeed = 1;
}

//check "base arm"
if (input.indexOf("a") != -1) {
  baseAngle += turnSpeed;
  runAutomation = false;
} else if (input.indexOf("d") != -1) {
  baseAngle -= turnSpeed;
```

```
    runAutomation = false;
  }

  //check "lower arm"
  if (input.indexOf("s") != -1) {
    lowerAngle += turnSpeed;
    runAutomation = false;
  } else if (input.indexOf("w") != -1) {
    lowerAngle -= turnSpeed;
    runAutomation = false;
  }

  //check "upper arm"
  if (input.indexOf("8") != -1) {
    upperAngle += turnSpeed;
    runAutomation = false;
  } else if (input.indexOf("2") != -1) {
    upperAngle -= turnSpeed;
    runAutomation = false;
  }

  //check "claw arm"
  if (input.indexOf("6") != -1) {
    clawAngle += turnSpeed;
    runAutomation = false;
  } else if (input.indexOf("4") != -1) {
    clawAngle -= turnSpeed;
    runAutomation = false;
  }
}

void SetAngleToServo() {
  // These make sure angle integer stays between 0 and 180
  if (baseAngle > 180) {
    baseAngle = 180;
  } else if (baseAngle < 0) {
    baseAngle = 0;
  }

  if (lowerAnglessw> 153) {
    lowerAngle = 153;
  } else if (lowerAngle < 0) {
    lowerAngle = 0;l
  }

  if (upperAngle > 180) {
    upperAngle = 180;
  } else if (upperAngle < 0) {
    upperAngle = 0;
  }

  if (clawAngle > 180) {
    clawAngle = 180;
  } else if (clawAngle < 0) {
    clawAngle = 0;
  }

  // Write angles to servos
  baseArm.write(baseAngle);
  lowerArm.write(lowerAngle);
  upperArm.write(upperAngle);
  clawArm.write(clawAngle);
}
```

```
void ResetAutomationMemory() {
  for (int i = 0; i <= 80; i++) {
    automationArray[i] = -1;
  }
  nextAutomationSaveStep = 0;
}

void SavePositionToMemory() {
  automationArray[nextAutomationSaveStep] = baseAngle;
  automationArray[nextAutomationSaveStep+1] = lowerAngle;
  automationArray[nextAutomationSaveStep+2] = upperAngle;
  automationArray[nextAutomationSaveStep+3] = clawAngle;
  nextAutomationSaveStep = nextAutomationSaveStep + 4;
}

void StartAutomation() {



  if (automationArray[0] == -1) {
    return;
  }

  int checkmark = 0;
  if (baseAngle > automationArray[automationStep]) {
    baseAngle--;
  } else if (baseAngle < automationArray[automationStep]) {
    baseAngle++;
  } else {
    checkmark++;
  }

  if (lowerAngle > automationArray[automationStep + 1]) {
    lowerAngle--;
  } else if (lowerAngle < automationArray[automationStep + 1]) {
    lowerAngle++;
  } else {
    checkmark++;
  }

  if (upperAngle > automationArray[automationStep + 2]) {
    upperAngle--;
  } else if (upperAngle < automationArray[automationStep + 2]) {
    upperAngle++;
  } else {
    checkmark++;
  }

  if (clawAngle > automationArray[automationStep + 3]) {
    clawAngle--;
  } else if (clawAngle < automationArray[automationStep + 3]) {
    clawAngle++;
  } else {
    checkmark++;
  }

  if (checkmark == 4) {
    delay(100);
    automationStep = automationStep + 4;
    if (automationArray[automationStep] == -1) {
      automationStep = 0;
    }
  }
}
```

```
void StopAutomation() {
  runAutomation = false;
  automationStep = 0;
}
```