Gesture Control Robot Using Image Processing and Object Tracking Algorithm

Submitted By
Muhammad Tayyab
F19603018
Saad Ahmad
F19603036
Muhammad Waqas
F19603023
Muhammad Haseeb Khalil
F19603016

Supervised By
Dr. Abdullah Waqas

Electrical Engineering Department
National University of Technology (NUTECH)
Islamabad, Pakistan

2023

ACKNOWLEDGMENT

In the name of Allah Almighty, the foremost gracious and therefore the most merciful. First and foremost, we are glad that Allah Almighty has given us the strength, knowledge, ability, and opportunity to undertake this study and complete it satisfactorily.

Secondly, we would like to thank and express our heartfelt gratitude to our supervisor Dr. Abdullah Waqas for his continuous support, zeal, patience, valuable advice, encouragement, and unwavering guidance that helped us exceptionally in our Final Year Project. His unparalleled knowledge, insightful suggestions, intense experience, and professional competence have empowered us to accomplish this successfully. Without him, this would not have been possible. We would not have imagined having a better supervisor and mentor for our BS study.

We would also be thankful to the university staff for their support and assistance. Most importantly, we would like to thank our friends, colleagues, and lab mates for their continuous support throughout the process.

DEDICATED TO

It is impossible to extend enough thanks to our families, especially our parents, who gave us the encouragement we needed through the process.

TABLE OF CONTENTS

Table of Figures

LIST OF ABBREVIATIONS

ABSTRACT

These days, robotic arms are employed in a variety of contexts, including pick-and-place tasks in artificial automation operations and in the military, defense, and medical fields. The Replicating Gesture Robotic arm is a servo-controlled robotic arm that imitates human arm motions in three dimensions. This design's primary goal is to manage the robotic arm.

Using mortal gestures. The robotic arm moves and completes the task while the system mimics the actions of human hands. It is coded so that the expected behavior for the fatal gesture is carried out. Consequently, the model we suggested will be quite useful.

Keywords: Servo motor, Arduino Mega, and robotic arm

Chapter 1

INTRODUCTION

1.1 Background

The development of computer vision and robotics has led to the creation of gesture control robots that use image processing and object tracking algorithms. While object tracking algorithms concentrate on tracking the movements of certain objects within a scene, image processing entails extracting information from digital stills or moving pictures. Robotic systems now have these technologies built in to help them understand human motions and offer simple control schemes.

Gesture control robots record real-time images of the user's movements by integrating cameras. These images are subjected to analysis by image processing algorithms, which track and recognize things like the user's hands. The robot can then comprehend the gestures and react appropriately thanks to object tracking algorithms, which keep track of the object's position and movement.

Robots with gesture controls have found use in a variety of fields, including manufacturing, healthcare, and gaming. By doing away with the use of physical controllers or complicated interfaces, these robots provide a more comfortable and natural method for people to connect with machines.

The robustness and accuracy of gesture recognition, as well as the adaptation of these robots to various surroundings, are all being improved through ongoing developments. In the future, it is anticipated that gesture control robots will revolutionize human-robot interaction and have even more diverse applications as technology advances.

1.2 Problem Statement

To design a robot that can imitate human gesture. And will be replacing human being in such places where human life is in danger and where human access is limited.

1.3 Design Overview

The overall all system consist of eight servo motor. The first five servo’s control the figures movement and next servo control the shoulder and next control the elbow movement and last one control the wrist movement as shown in figure 1.0.1.

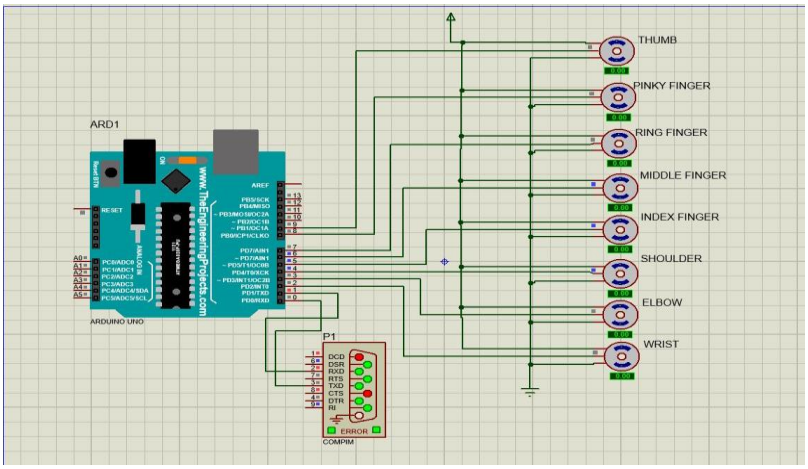

Figure 1.0.1 Proteus Design

1.4 Novelty

We control the robotic arm using image processing and object tracking algorithm. Python used to detect and to give data points of human joints. Which are then communicated to Arduino. We give signal to the Arduino, Arduino generate PWM to operate the servo, and these servos control the arm movement.

1.5 Time Frame

1.5.1 Semester VII

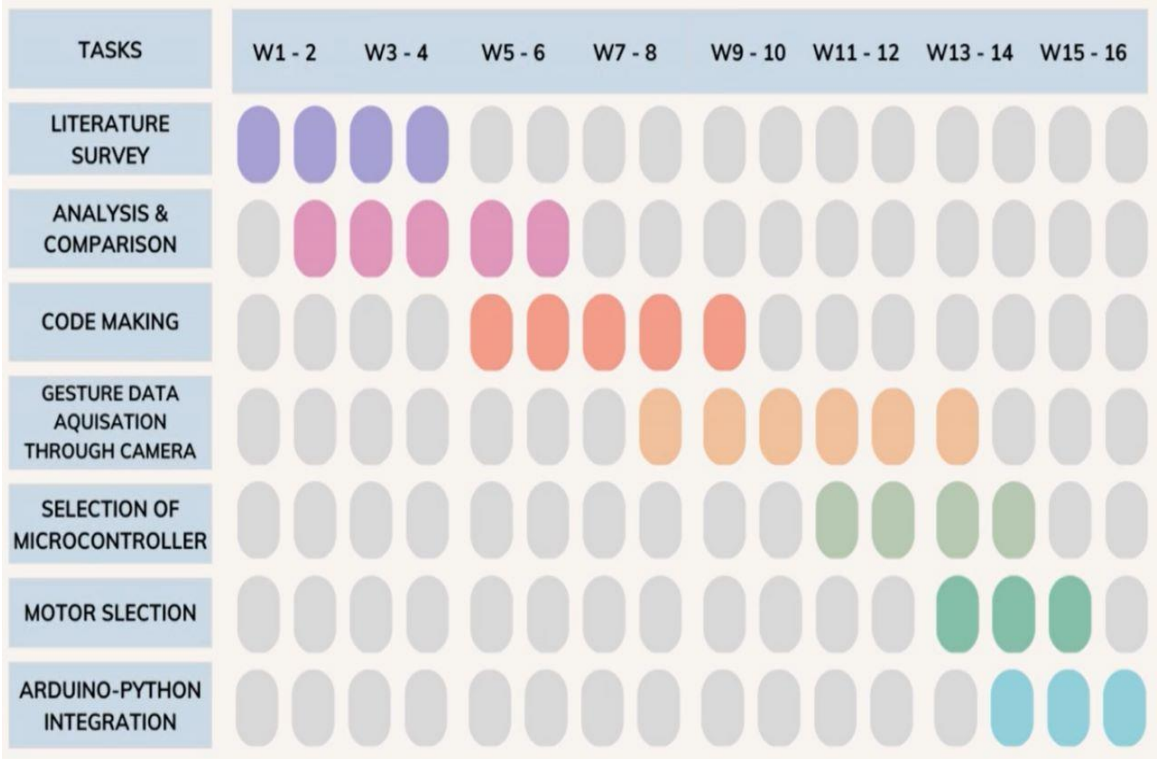

Figure 1.0.2 Gantt chart 1

1.5.2 Semester VIII

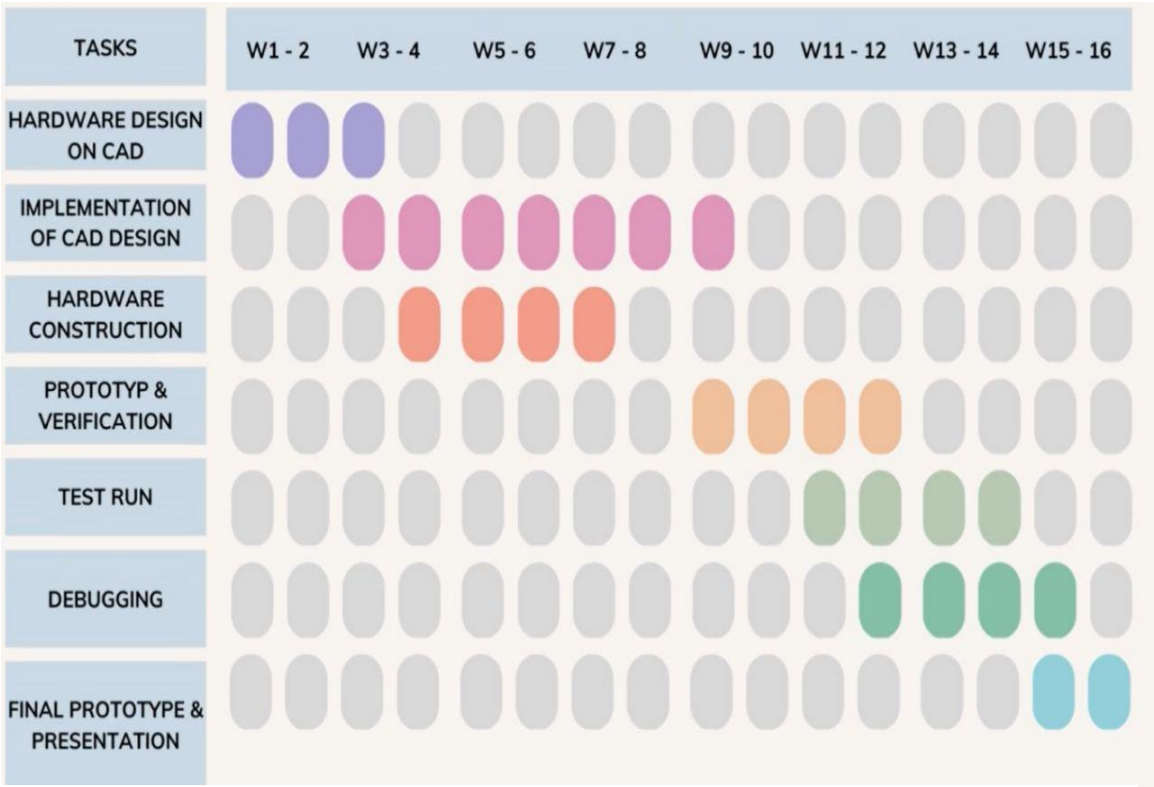

Figure 1.0.3 Gantt chart 2

1.6 Resources Required

1.6.1 Components

- MG996R
- MG945
- Arduino UNO
- ESP WIFI Module
- Battery
- Fishing Wire
- Male/Female Wire
- Dummy
- Transistor (LM7805)
- Vero Board
- Connector

1.6.2 Equipment

- PC
- 3D Printer
- DMM
- Power supply
- Portable Grinder

Chapter 2

LITERATURE REVIEW

The purpose of a literature review is to identify the current state of knowledge, identify gaps or controversies in the existing literature, and provide a theoretical and conceptual framework for the project.

2.1 Gesture Recognition for Human-Machine Interaction

Gesture recognition facilitates the establishment of a seamless rapport between mankind and machinery, obviating the necessity for mechanical or electro-mechanical contrivances. It empowers users to engage with computers and exert dominion over them by means of indicative gesticulations directed towards the display. The advent of gesture recognition technology potentially heralds the supplanting of conventional input apparatuses, including mice, keyboards, and touch screens. The attainment of gesture recognition is predicated upon the convergence of computer vision and image processing methodologies. Hand interactive systems, as commonly implemented, exhibit a tripartite structure, encompassing the realms of detection, tracking, and recognition. [1]

2.2 Three Layers of Gesture Recognition

The three layers of gesture recognition systems are detecting, tracking, and recognizing. The detection layer extracts visual features associated with the presence of hands in the camera's field of view. The tracking layer establishes temporal data association between successive image frames to track the movement of hands. The recognizing layer groups the spatiotemporal data extracted from the previous layers and assigns labels to specific gesture classes. [2]

2.3 Cost-effective Approach: Arduino UNO & OpenCV

DSP processors are highly qualified for signal processing but not cost-effective for the project. Arduino Uno, a low-cost single board computer, was used instead. Python programming language is utilized with the OpenCV library for gesture recognition. OpenCV is an open-source library that requires fewer resources compared to costly MATLAB software. [1]

2.4 Wireless Control

The system uses wireless control for better portability. An ESP8266 module is used to make it wireless The ESP8266 module allows microcontrollers to establish a connection with 2.4 GHz Wi-Fi networks, utilizing the IEEE 802.11 ban standard.

Chapter 3

THEORY

Once the subject was chosen for the bachelor's thesis project the initial research was done. Previous bachelor's theses regarding a Robotic Hand Controlled by Glove Using Wireless Communication [3], Gesture Replication Robo-Arm [4] , Robotic Telekinesis: Learning a Robotic Hand Imitator [5], gesture based wireless control of robotic hand using image processing [1] and The Manipulation of Real-Time Kinect-Based Robotic Arm Using Double-Hand Gestures [6] were needed for the project.

3.1 Anatomy of Human Hand Wrist

3.1.1 The Bones and Joints

The carpus and five fingers make up the human hand Figure. The grouping of eight tiny bones known as the carpus is located between the wrist and the tips of the fingers. The Navicular, Lunate, Triquetrum, Pisiform, Greater Multangular, Lesser Multangular, Capitate, and Hamate are among these bones. The Radiocarpal (RC) joint helps to link the hand to the fingers. The four joints, three phalanges, and one metacarpal bone (M) make up each finger, except for the thumb. The first phalanx (FP), second phalanx (SP), and third phalanx (TP) are the three phalanges. The distal interphalangeal joint (DIP), proximal interphalangeal joint (PIP), metacarpophalangeal joint (MP), and either carpometacarpal joint (CM) or intercarpal joint (IC) are the four joints. [7]

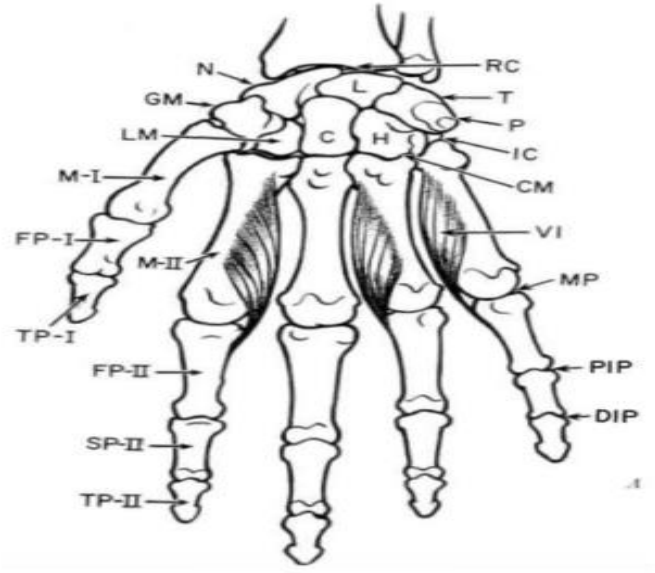

Figure 3.0.1 Bones & Joints of Human Hand [7]

Bones &oints of Human Hand [7]

3.1.2 The Tendons

Flexor tendons: Located on the palm side, responsible for flexing the fingers and thumb. Extensor tendons: Located on the back of the hand, responsible for extending the fingers and thumb. Tendons run through sheaths, allowing smooth gliding motion. [7]

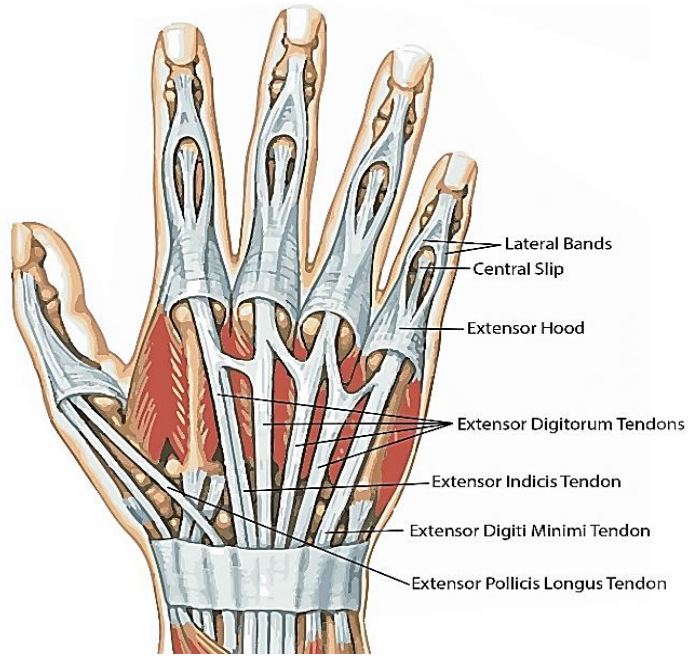

Figure 3.0.2 Human Hand Tendons working [8]

3.2 Arduino Uno

The open-source electronics platform known as Arduino offers the Arduino Uno, an open-source microcontroller. This microcontroller is programmed in the Arduino programming language, which has its origins in the open-source Wiring microcontroller architecture. The Arduino Integrated Development Environment (IDE), which is based on the Processing software, is used to write the code. The Arduino Uno microcontroller is linked to a battery for operation, and a USB cable (USB-A to USB-B) is used to upload the compiled programming to the controller. The table below contains important details about the Arduino Uno microcontroller. Data from the glove is received by the Arduino Uno, which is then utilized to control the fingers of the robotic hand and wrist. [9]

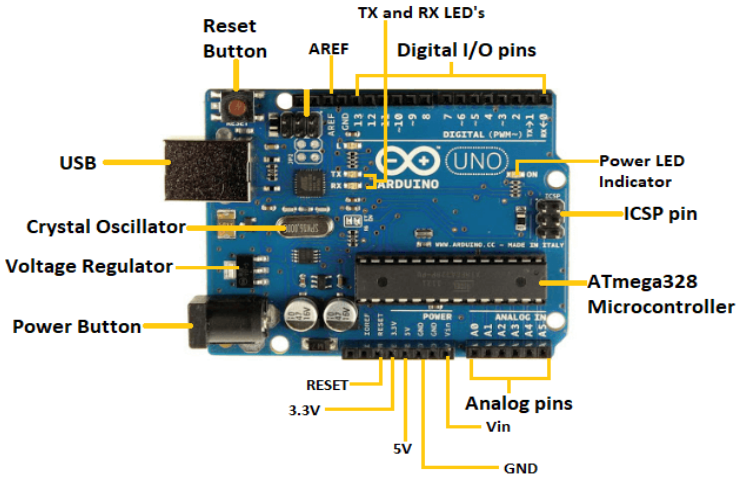

Figure 3.0.3 Arduino UNO [10]

Table 3.1: Tech specs of the Arduino-Uno microcontroller. [10]

Microcontroller name	ATmega328P
Operating Volts	5 Volt
Input Volt	6-20 Volt
Digital Input/Output Pins	14 (6 are PWM out)
PWM Digital In/Out Pins	6
Analog Input signal Pins	6
DC Current per In/Out Pin	20 milli Amps
DC Current for 3.3Volt Pin	50 milli Amps
Flash Memory storage	32 Kilo Bytes
Static RAM	2 Kilo Bytes
Electrically Erasable Programmable ROM	1 Kilo Bytes
Clock Speed in MHz	16
LED_BUILT-IN	13
Length in mm	68.6
Width in mm	53.4
Weight in grams	25

3.3 MG996R Servo Motor

The MG996R servo motors, shown in the image, weigh 55 g and have a torque of 11 kg/cm at 6 V and 9.4 kg/cm at 4.8 V. This means that, depending on the power source, the motor can lift either 11 kg or 9.4 kg at a distance of 1 cm perpendicular to the shaft. The kg/cm to N/m conversion factor is 0.981. The operational voltage ranges from 4.8 to 6.6 V, and the rotation is 180°. The microcontroller and the power supply must be linked to the three wires on the motor. The red wire should be connected to the battery (power source), the orange wire to an Arduino pin that can send the pulse width modulated (PWM) signal to the motor, and the brown wire is the ground wire and should be linked to the Arduino's ground input. The robotic hand's fingers may bend and stretch thanks to the servo motors. [11]

Figure 3.0.4 MG996R Servo Motor [12]

Because servo motors include internal controls that can adjust both position and speed, they are used because of their ease of installation. In contrast to stepper motors, which work through an open loop and require a given number of steps, this enables the specification of a desired position. Simple servo motors only require three wires because of the internal self-regulating nature of servo motors. [13]

3.3.1 Pulse Width Modulation

The method of pulse width modulation makes it possible to use digital outputs on analogue circuits by producing analogue signals from digital signals. Instead of using straight analogue signals, PMW on a digital signal can save system expenses and power usage. Utilising PMW can also reduce the effects of system noise. The method is based on altering the duty cycle, which is the duration of the digital signal applied in relation to a square wave's period. For instance, the analogue signal produced by a 5 V supply and a 10% duty cycle is 0.5 V. Figure shows signals with various duty cycles. [14]

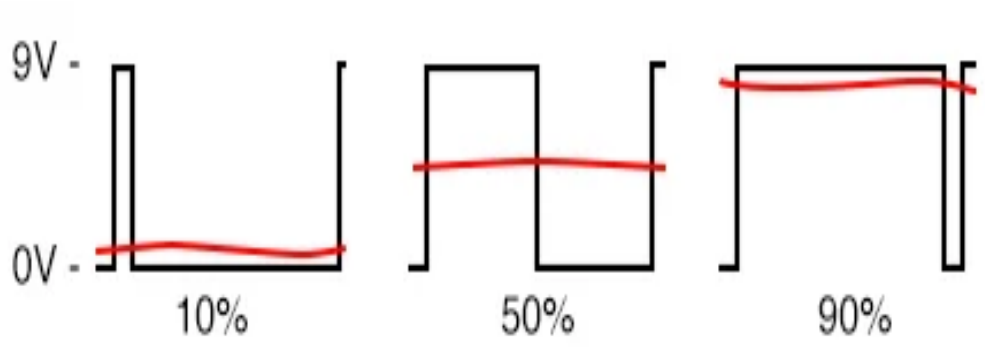

Figure 3.0.5 Pulse Width

3.4 Battery Bank

A battery bank of 7.4-8.4V is used having total 12Ah charge. It was made using lithium cell of 3.7V. 6 cells are used to make such a battery bank. This battery bank contains more energy than required as we need approximately 8Amp. However this bank can provide 12amps this increases the battery time of robot.

3.5 ESP8266

The ESP8266 is an affordable and versatile microcontroller module equipped with built-in Wi-Fi capabilities. It is commonly used in IOT applications to connect microcontrollers to Wi-Fi networks and enable internet communication. With its compact size, powerful processor, and ample memory, the ESP8266 supports the IEEE 802.11 b/g/n Wi-Fi standards. It can be programmed using various development environments and languages like Arduino IDE or NodeMCU Lua firmware. Due to its cost-effectiveness and flexibility, the ESP8266 has gained popularity among hobbyists, makers, and developers for a wide range of IoT projects.

Chapter 4

PYTHON PROGRAMMING

4.1 Introduction

Python is a high level and all-purpose programming language. With the off-side rule and substantial indentation, its design philosophy prioritizes readability of the code. It is an interpreter language. Latest version of python is 3.11.4 as of June 2023.

Python`s memory management system combines reference counting and cycle detecting garbage collector with dynamic typing. Method and variable names bound via dynamic name resolution, which takes place while the program is running.

4.1.1 Syntax and Semantics

Python is made easy to read. Its formatting is distinct and usually using English keywords. Unlike other language, it does not use curly brackets to limit blocks, and not even semicolon is used to terminate a statement.

4.1.2 Typing

Duck typing is used in python, which has typed objects but un-typed variable names. Type restriction aren`t verified at build time; instead, actions on an object could fail, indicating that it is not the right type.

4.1.3 Libraries

Python supported a wide variety of libraries and this is one of the strengths of python. For internet facing applications, protocols and standard such as Multipurpose Internet Mail Extensions (MIME) and Hypertext Transfer Protocol (HTTP) are also supported.

4.1.4 Integrated Development Environment

Python has a number of Integrated Development Environment (IDE). It comes with a default IDE. One of the main distributions for python IDEs is Anaconda, which includes Spyder, Jupiter, PyCharm and many more IDEs.

4.2 Spyder IDE

It is a powerful IDE for python programming. It is an open-source software can be

downloaded independently or can be used through Anaconda distribution.

In this project we used Spyder IDE for python programming.

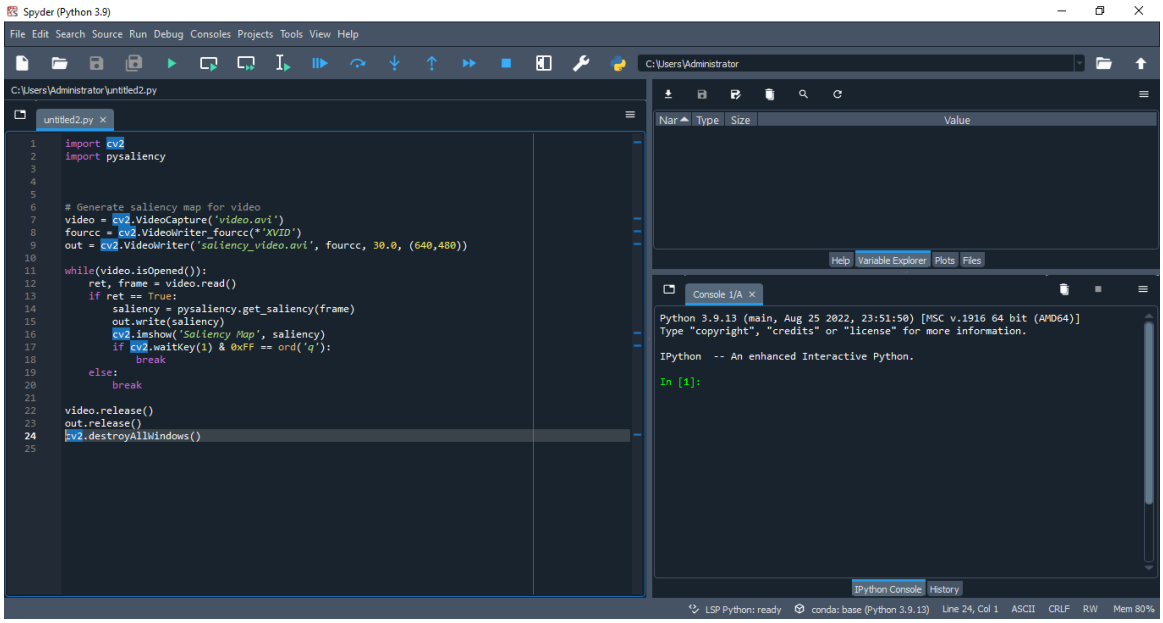

Figure 4.0.1 Spyder IDE

4.3 Open Computer Vision Library (OpenCV)

OpenCV is a programming library, which is mainly used for real-time computer vision. It is a cross platform and licensed as free and open source. It features GPU acceleration for the real-time operations.

We have used it to get real time data from live feed of camera. Which is then processed to get the desired results. It opens the webcam of the pc or laptop and get the real time data from it.

4.4 Media pipe

Media pipe is framework containing a wide range of machine learning solutions. It contains everything required in order to customize and deploy easily.

We are using Media pipe holistic solution which gives us the detection of different joints of human body, which includes shoulder, elbows, fingers, arms, hip joints and etc.

4.5 Python Code

4.5.1 Libraries

```
import mediapipe as mp
import cv2
import numpy as np
from cvzone.HandTrackingModule import HandDetector
import time
import socket
```

Figure 4.0.2 Libraries

Importing different libraries in order to perform desired task.

- **Media pipe**: It is a Python framework used for building machine-learning pipelines that process multimedia data like images and videos.
- **Cv2**: This library, known as OpenCV, provides functions for image and video processing.
- **NumPy**: It is a powerful Python library used for numerical computing, offering support for large arrays and matrices, along with mathematical functions.
- **CV Zone**: This appears to be a custom module or package called cvzone, which likely includes a class called Hand Detector for hand tracking in images or videos.
- **Time**: A standard Python module that offers functions related to time, such as measuring intervals and introducing delays in code execution.
- **Socket**: A module used for low-level network programming in Python, enabling network connections and communication with other machines.

4.5.2 Wireless Communication

```
#Assigning IP Address
ESP_IP = '192.168.15.4'
ESP_PORT = 8888
```

Figure 4.0.3 IP Address

The code assigns values to two variables, ESP_IP and ESP_PORT, which are utilized for defining the IP address and port number used to establish communication with an ESP device.

The variable ESP_IP is set to '192.168.15.4', representing the specific IP address assigned to the ESP device. The actual IP address may differ based on the network

configuration and setup of the ESP device.

The variable ESP_PORT is assigned the value 8888, which denotes the port number employed for the communication. The port number 8888 is commonly utilized for network communication, although it can be modified to any available port based on the project's requirements or network configuration.

4.5.3 Models

```
#Getting Models
detector=HandDetector(detectionCon=0.8,maxHands=1)
mp_drawing = mp.solutions.drawing_utils
mp_holistic = mp.solutions.holistic
mp_pose=mp.solutions.pose
mp_Hand=mp.solutions.hands
hands=mp_Hand.Hands(static_image_mode=False,max_num_hands=2,min_detection_confidence=0.8)
```

Figure 4.0.4 Models

The provided code segment obtains various models or modules for hand detection, pose estimation, and holistic analysis using the mediapipe library.

- **Detector = HandDetector (detectionCon=0.8, maxHands=1)**: This line creates a HandDetector object from the cvzone.HandTrackingModule module. It sets the detection confidence threshold to 0.8 and restricts the maximum number of detected hands to 1.
- **mp_drawing = mp.solutions.drawing_utils**: This line imports the drawing_utils module from mediapipe.solutions and assigns it to the variable mp_drawing. It provides functions for drawing visualizations of the results.
- **mp_holistic = mp.solutions.holistic**: This line imports the holistic module from mediapipe.solutions and assigns it to the variable mp_holistic. It allows for holistic analysis, encompassing body pose estimation, face analysis, and hand tracking.
- **mp_pose = mp.solutions.pose**: This line imports the pose module from mediapipe.solutions and assigns it to the variable mp_pose. It enables body pose estimation through the Pose class.
- **mp_Hand= mp.solutions.hands**: This line imports the hands module from mediapipe.solutions and assigns it to the variable mp_Hand. It enables hand tracking via the Hands class.
- **Hands = mp_Hand. Hands (static_image_mode=False, max_num_hands=2, min_detection_confidence=0.8)**: This line initializes a Hands object for hand tracking.

It sets static_image_mode to False, indicating that real-time video frames will be processed. Additionally, it configures max_num_hands to 2, allowing the detection of a maximum of 2 hands, and min_detection_confidence to 0.8, representing the minimum confidence threshold for hand detection.

These models and objects will be used later in the code to perform tasks such as hand detection, pose estimation, and holistic analysis.

4.5.4 Webcam

```
#Opening WebCam
cap = cv2.VideoCapture(1)
```

Figure 4.0.5 Webcam Command

The provided code segment initializes the webcam or video capture device using the OpenCV library.

Cap = cv2.VideoCapture (1): This line creates a Video Capture object named cap to access the webcam. The argument 1 represents the index or ID assigned to the specific webcam to be used. The index may vary depending on the available camera devices. In this case, the value 1 indicates the second webcam device.

4.5.5 Function to Calculate Angle

```
# Defining Function to Calculate angles
def calculate_angle(a,b,c,d):
    a=np.array(a)
    b=np.array(b)
    c=np.array(c)
    radians=np.arctan2(c[1]-b[1],c[0]-b[0])-np.arctan2(a[1]-b[1],a[0]-b[0])
    angle =np.abs(radians*180/np.pi)
    if d==0:
        if angle >180:
            angle=360-angle
        angle=int(angle)
    elif d==1:
        angle=int(angle)
    return angle
```

Figure 4.0.6 Function to Calculate Angle

A function called calculate angle that determines the angle between four points: a, b, c, and d.

The function takes four parameters: a, b, c, and d, representing the coordinates of the four points. The coordinates a, b, and c are converted into NumPy arrays. The radians variable

calculates the angle between the vectors ba and bc using the arctan2 function.

The angle variable stores the absolute angle in degrees, obtained by converting the radians to degrees. The function checks the value of the parameter d. If it is 0, the angle is adjusted to be within the range of 0 to 180 degrees by subtracting it from 360 degrees if it exceeds 180. The angle is then converted to an integer. If d is 1, the angle is directly converted to an integer. Finally, the function returns the calculated angle.

4.5.6 Function to Pad Zeros

```
# Defining Function to Pad Zeros
def pad_zeros(arr):
    send=""
    for i in range(len(arr)):
        if len(str(arr[i])) < 3:
            send = send+str(arr[i]).zfill(3)
        else:
            send=send+str(arr[i])
    return send
```

Figure 4.0.7 Function to Pad Zeros

A function named pad_zeros that adds leading zeros to the elements of an array.

The function accepts an array called arr as input. It initializes an empty string variable called send to store the padded values. The function iterates over each element in the array using a for loop. For each element, it checks if the length of its string representation is less than 3. If the length is less than 3, it pads leading zeros to the string representation using the zfill method. The padded or original string representation of the element is then appended to the send variable. After iterating through all the elements, the function returns the final concatenated string containing the padded values.

4.5.7 Function for Finger Angles

```
# Defining Function For Finger Angles
def finger_angle(val):
    if val==1:
        angle = 0
    elif val==0:
        angle=90
    return angle
```

Figure 4.0.8 Function to Calculate Finger Angle

The function starts by checking if the value of the finger is either 1 or 0. If the finger value is 1, the function returns an angle of 0 degrees. Similarly, if the finger value is 0, the function returns an angle of 90 degrees. However, if the finger value is any other value, the function raises a Value Error exception.

4.5.8 Initializing Finger Values

```
#Assigning Fingers Angle initially
Thumb_angle=0
Index_angle=0
Middle_angle=0
Ring_angle=0
Pinky_angle=0
```

Figure 4.0.9 Initializing Finger Values

This only to initialize the fingers values to zero. This will set the fingers in default open position, as well as helps overcome Value Error exception in above code.

4.5.9 Initializing Holistic and Setting Frame

```
# Initializing Holistic
with mp_holistic.Holistic(min_detection_confidence=0.5, min_tracking_confidence=0.5) as holistic:
    while cap.isOpened():
        ret, frame = cap.read()
        # Recolor Feed
        image = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)

        # Make Detections
        results = holistic.process(image)
        results_1= hands.process(image)

        # Recolor image back to BGR for rendering
        image = cv2.cvtColor(image, cv2.COLOR_RGB2BGR)
        ##

        #extract landmarks
        try:
            landmarks=results.pose_landmarks.landmark
            landmarks_1=results.right_hand_landmarks.landmark
        except:
            pass
```

Figure 4.0.10 Initializing Holistic Model

The provided code executes a continuous loop while the video stream is active. During each iteration, the following steps are performed:

- The next frame is read from the video stream.
- The color space of the frame is converted from BGR to RGB.
- The Holistic object is utilized to detect and track landmarks of the human body in the frame.

- The frame is converted back to the BGR color space for visualization purposes.
- The pose landmarks and right-hand landmarks are extracted from the results obtained by the Holistic object.
- If no landmarks are detected in the frame, the code proceeds to the next iteration.
- The try-except blocks are used to handle situations where no landmarks are detected. In such cases, the code simply moves to the next iteration.

The variables landmarks and landmarks_1 hold the pose landmarks and right hand landmarks, respectively. These variables can be utilized for further processing of the video stream, such as tracking the movement of the user's body or hands.

4.5.10 Detections

```
# Pose Detections
mp_drawing.draw_landmarks(image, results.pose_landmarks, mp_holistic.POSE_CONNECTIONS)
#Frame
cv2.imshow('Gesture Control Robot', cv2.flip(image,1))
if results_1.multi_hand_landmarks:
    for handLandmarks in results_1.multi_hand_landmarks:
        lmList, bbox = detector.findHands(image, handLandmarks)
        if lmList:

            fingers = detector.fingersUp(lmList[0]) # Detect open or closed finger # 1 - Open finger, 0 - Closed
            #Converting Finger angles
            Thumb_angle=finger_angle(fingers[0])
            Index_angle=finger_angle(fingers[1])
            Middle_angle=finger_angle(fingers[2])
            Ring_angle=finger_angle(fingers[3])
            Pinky_angle=finger_angle(fingers[4])
```

Figure 4.0.11 Detections

The code utilizes the mp_drawing module to draw pose landmarks on an image. The POSE_CONNECTIONS variable is a list containing pairs of landmark indices that specify the connections between pose landmarks. The draw landmark's function uses this information to draw lines connecting the corresponding landmarks on the image.

The image is displayed on the screen using the cv2.imshow function. To horizontally mirror the image, the cv2.flip function is employed.

The code checks if there are any hand landmarks detected in the image. If hand landmarks are present, the code iterates through each hand landmark and retrieves the landmark list and the bounding box.

To determine whether the fingers are open or closed, the code employs the detector object. The fingersUp function is used, which returns a list of Boolean values. Each value indicates whether the corresponding finger is open or closed.

The finger_angle function is used to calculate the angle of each finger. This function returns the angle of the finger in degrees.

4.5.11 X and Y parameters

```
#Getting X and Y Parameters

left_middle = [landmarks[mp_pose.PoseLandmark.RIGHT_INDEX.value].x,landmarks[mp_pose.PoseLandmark.RIGHT_INDEX.value].y]
shoulder_R = [landmarks[mp_pose.PoseLandmark.RIGHT_SHOULDER.value].x,landmarks[mp_pose.PoseLandmark.RIGHT_SHOULDER.value].y]
shoulder_L = [landmarks[mp_pose.PoseLandmark.LEFT_SHOULDER.value].x,landmarks[mp_pose.PoseLandmark.LEFT_SHOULDER.value].y]
elbow = [landmarks[mp_pose.PoseLandmark.RIGHT_ELBOW.value].x,landmarks[mp_pose.PoseLandmark.RIGHT_ELBOW.value].y]
wrist = [landmarks[mp_pose.PoseLandmark.RIGHT_WRIST.value].x,landmarks[mp_pose.PoseLandmark.RIGHT_WRIST.value].y]
hip_joint = [landmarks[mp_pose.PoseLandmark.RIGHT_HIP.value].x,landmarks[mp_pose.PoseLandmark.RIGHT_HIP.value].y]
```

Figure 4.0.12 Parameters Extraction

The code retrieves the x and y coordinates of several landmarks: right index finger, right shoulder, left shoulder, right elbow, right wrist, and right hip. These coordinates are stored in a list of lists. The landmarks variable is a list of landmark objects. Each landmark object contains the x, y, z, and visibility coordinates of a specific landmark. The mp_pose. PoseLandmark class defines the names and indices of the pose landmarks. By accessing the value property of a landmark object, the index of the landmark can be obtained. The extracted x and y coordinates of the landmarks are valuable for tracking the movement of the user's body or hands, as well as for gesture detection. For instance, the code could be employed to track the user's right hand and identify when the user forms a fist gesture.

4.5.12 Calculating Angles

```
#Calculating Angles
shoulder_angle_UD=calculate_angle(hip_joint, shoulder_R, elbow,0)
shoulder_angle_SW=calculate_angle(shoulder_L, shoulder_R, elbow,1)
shoulder_angle_SW=270-shoulder_angle_SW
elbow_angle=calculate_angle(shoulder_R, elbow,wrist,0)
elbow_angle=180-elbow_angle
wrist_angle=calculate_angle(elbow,wrist,left_middle,0)
```

Figure 4.0.13 Calculating Joints Angles

The provided code calculates the angles between specific landmarks in the body,

1. Hip joint, right shoulder, and Elbow.
2. Left shoulder, right shoulder, and Elbow.
3. Right shoulder, right elbow, and Wrist
4. Right elbow, right wrist and left middle.

The calculation is performed using the calculate angle function. This function accepts three

arguments: the start landmark, the end landmark, and a flag indicating whether the angle should be measured in degrees or radians. The function computes and returns the angle between the two landmarks in degrees.

By utilizing the calculate angle function with the appropriate landmark pairs, you can determine the angles formed by different body parts. These angles can be useful for various applications, such as tracking body posture or analyzing movement patterns.

4.5.13 Data Transmission

```
#Assiging Data to array
data=[Thumb_angle,Index_angle,Middle_angle,Ring_angle,Pinky_angle,shoulder_angle_UD,elbow_angle,shoulder_angle_SW]
Send=pad_zeros(data)
message = Send
# Create a TCP/IP socket
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
# Connect to the ESP
sock.connect((ESP_IP, ESP_PORT))
# Send the message to the ESP
sock.sendall(str(message).encode())
time.sleep(0.8)
```

Figure 4.0.14 Data Transmission

The code performs the following tasks:

The angles of the fingers and shoulders are assigned to a list.

The list is then padded with leading zeros so that all elements have a length of 3.

The padded list is assigned to the variable message.

Additionally, the code involves the following steps related to TCP/IP socket communication:

A TCP/IP socket is created using the socket module. The socket.AF_INET parameter specifies that the socket should use the IPv4 protocol, and the socket.SOCK_STREAM parameter indicates that it should be a stream socket.

The socket is connected to the ESP device. The IP address of the ESP is specified by the ESP_IP variable, and the port number is specified by the ESP_PORT variable.

The message is sent to the ESP. The str(message).encode() expression converts the message variable to a string and then encodes it as a byte sequence. The sock.sendall() method sends the byte sequence to the ESP.

The code pauses its execution for 0.8 seconds. This delay allows the ESP device to receive the message effectively.

These steps collectively enable the transfer of the padded message to the ESP device using a TCP/IP socket connection.

4.5.14 Terminating Window

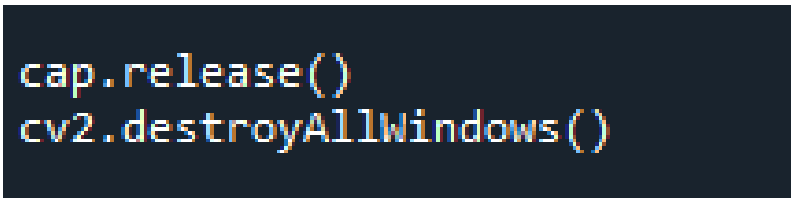

Figure 4.0.15 Terminating Windows

The video capture object is released using the cap.release() function, which also releases any associated system resources. When you are done using the video capture, you should call this.

All open windows produced by OpenCV are closed and destroyed using the cv2.destroyAllWindows() function. To make sure all windows are correctly closed at the end of a program, this technique is frequently employed.

When using OpenCV, both of these function calls are necessary for effective resource management and cleanup.

Chapter 5

ARDUINO CODING

5.1 Arduino IDE

The user-friendly software tool known as the Arduino IDE is used to program Arduino boards. It offers a straightforward and understandable user interface for creating, assembling, and uploading code to Arduino microcontrollers. The Arduino IDE gives users the ability to develop and release their projects more quickly thanks to features like code highlighting, a serial monitor, and sizable library selection. The user-friendly software tool known as the Arduino IDE is used to program Arduino boards. It offers a straightforward and understandable user interface for creating, assembling, and uploading code to Arduino microcontrollers. The Arduino IDE gives users the ability to develop and release their projects more quickly thanks to features like code highlighting, a serial monitor, and a sizable library selection.

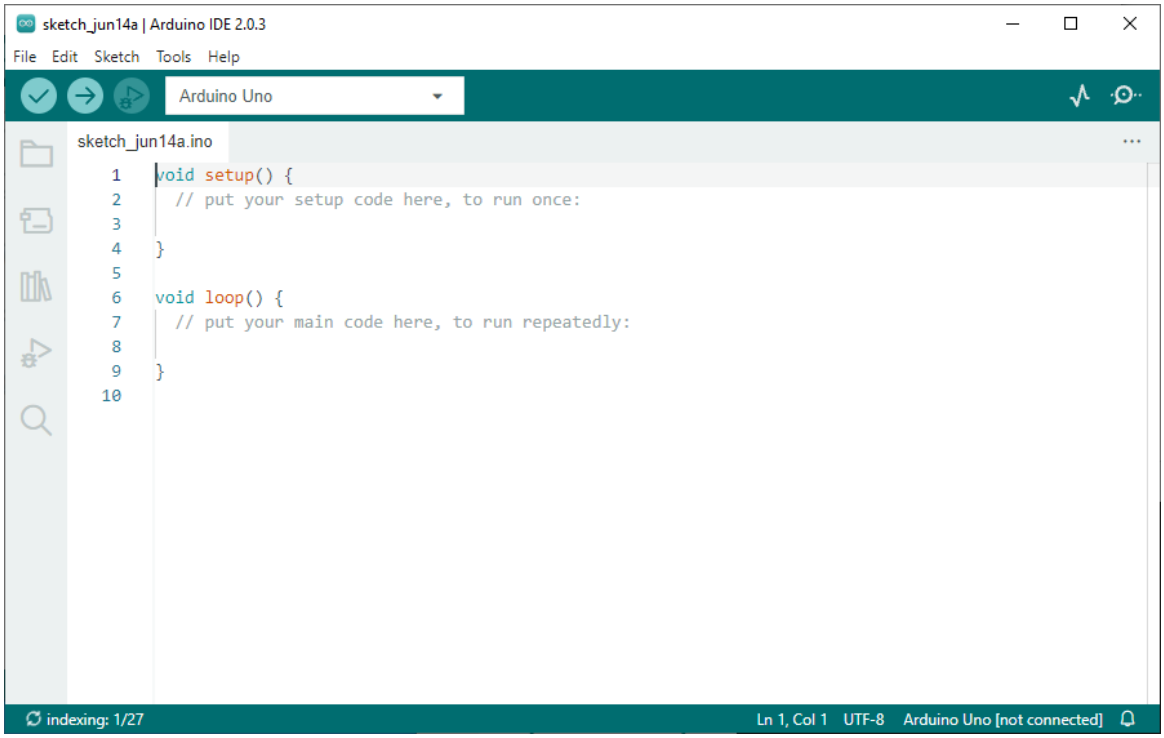

Figure 5.0.1 Arduino IDE

5.1.1 Arduino IDE Language

The Arduino IDE utilizes an adapted version of C++ called the Arduino programming language. It offers a simplified syntax suitable for beginners while maintaining

compatibility with the more advanced features of C++.

5.2 Code for Arduino UNO/MEGA

A program is written which receives data from serial communication and using different techniques to extract angles from the received data. Then use the data to operate servos.

5.2.1 Including Library and defining variables

```
1   #include<Servo.h>
2   #define numOfValsRec 6
3   #define digitsPerValRec 3
4   int valsRec[numOfValsRec];
5
6   Servo Thumb;
7   Servo Index;
8   Servo Middle;
9   Servo Ring;
10  Servo Pinky;
11  Servo Elbow;
12  Servo Shoulder;
13  String Received;
```

Figure 5.0.2 Including Libraries

The `#include<Servo.h>` line imports the Servo library for servo motor control. The `#define` statements establish constants `numOfValsRec` and `digitsPerValRec` with respective values of 6 and 3. An integer array called `valsRec` is declared to store received values. Multiple Servo objects, such as `Thumb`, `Index`, `Middle`, `Ring`, `Pinky`, `Elbow`, and `Shoulder`, are created to control individual servo motors. Lastly, a `String` variable named `Received` is declared to hold received data. In summary, this code sets up the necessary components for managing servo motors and processing received values on an Arduino board.

5.2.2 Setup

```
void setup()
{
  Serial.begin(9600);
  Thumb.attach(3);
  Index.attach(4);
  Middle.attach(5);
  Ring.attach(6);
  Pinky.attach(7);
  Elbow.attach(8);
  Shoulder.attach(9);
  Zero();
}
```

Figure 5.0.3 Setup

The `setup()` function initializes the program by performing the following tasks:

Setting up serial communication with a baud rate of 9600 using `Serial.begin(9600)`.

Associating the servo objects (`Thumb`, `Index`, `Middle`, `Ring`, `Pinky`, `Elbow`, and `Shoulder`) with their corresponding digital pins (3, 4, 5, 6, 7, 8, and 9) using the `attach()` method. Calling the `Zero()` function to position the servos to their initial positions.

In summary, the `setup()` function establishes the necessary configurations for serial communication and servo control, ensuring proper initialization of the program.

5.2.3 Receiving Data

```
void Receiving()
{
  if(Serial.available())
  {
    Received=Serial.readString();
    for(int i=0;i<numOfValsRec;i++)
      {
        int num =(i*digitsPerValRec);
        valsRec[i]= Received.substring(num,num+digitsPerValRec).toInt();
      }
  }
Servo_Control();
Received="";
}
```

Figure 5.0.4 Data Receiving

A function named `Receiving()`, which is responsible for receiving data over the serial communication and processing it to control servos.

The function starts with an `if` statement that checks if there is data available to read from the serial port using `Serial.available()`. If data is available, the following actions are performed:

- The received data is stored in the `Received` variable using `Serial.readString()`.
- A `for` loop is used to iterate through the `valsRec` array.
- Inside the loop, the `num` variable is calculated as the product of `i` and `digitsPerValRec`, which determines the starting index of the substring to extract from `Received`.
- The `substring()` function is used to extract a substring from `Received` starting from `num` and with a length of `digitsPerValRec`. The `toInt()` function is then applied to convert the extracted substring to an integer, which is stored in the corresponding element of the `valsRec` array.
- After processing the received data and populating the `valsRec` array, the `Servo_Control()` function is called to control the servos based on the received values. Finally, the `Received` variable is cleared by setting it to an empty string

5.2.4 Initial Value

```
void Zero()
{
 Thumb.write(0);
 Index.write(0);
 Middle.write(0);
 Ring.write(0);
 Pinky.write(0);
 Elbow.write(0);
 Shoulder.write(0);
}
```

Figure 5.0.5 Zero Position

Function is designed to reset the positions of several servos to zero.

Within the `Zero()` function, each servo object (`Thumb`, `Index`, `Middle`, `Ring`, `Pinky`, `Elbow`, and `Shoulder`) is individually instructed to move to the zero position by

utilizing the `write()` function with a value of 0 as the position argument. This action effectively resets all the servos to their initial zero positions.

In essence, the `Zero()` function ensures that all mentioned servos are uniformly set to the zero position, establishing a consistent starting point for their subsequent movements.

5.2.5 Servo Control

```
void Servo_Control()
{
 Thumb.write(valsRec[0]);
 Index.write(valsRec[1]);
 Middle.write(valsRec[2]);
 Ring.write(valsRec[3]);
 Pinky.write(valsRec[4]);
 Elbow.write(valsRec[5]);
 Shoulder.write(valsRec[6]);
}
```

Figure 5.0.6 Servo Control

Inside the `Servo_Control()` function, each servo object (`Thumb`, `Index`, `Middle`, `Ring`, `Pinky`, `Elbow`, and `Shoulder`) is individually directed to move to a specific position. The desired position for each servo is obtained from the corresponding element in the `valsRec` array.

To achieve this, the `write()` function is utilized for each servo, with the target position passed as an argument. The position value is retrieved from the `valsRec` array using the appropriate index: `valsRec[0]` for `Thumb`, `valsRec[1]` for `Index`, `valsRec[2]` for `Middle`, and so forth.

5.2.6 Loop

```
void loop()
{
  Receiving();
}
```

Figure 5.0.7 Loop

The loop function is continuously calling ‘Receiving’ function. In order to get data from serial and operate the servos.

5.3 Code for ESP8266

ESP is being used to make the system wireless and operate it using wifi.

5.3.1 Declaration

```
#include <ESP8266WiFi.h>
#include <SoftwareSerial.h>

// Wi-Fi configuration
const char* ssid = "NUTECH-Staff";
const char* password = "Nut@ch@765";
const int ledPin = 0;
// UART configuration
const int UART_BAUD_RATE = 9600;
// TCP server configuration
WiFiServer server(8888);
// UART communication
SoftwareSerial uart(1, 3); // RX, TX (Connect ESP8266 TX to Arduino RX and vice versa)
```

Figure 5.0.8 Declaring variables

The code is designed to be used with an ESP8266 module and an Arduino board. It includes necessary libraries for Wi-Fi communication (`ESP8266WiFi`) and software serial communication (`SoftwareSerial`).

The code can be broken down into the following sections.

1. Library Import:

- The code imports the required libraries, `ESP8266WiFi` for Wi-Fi functionality and `SoftwareSerial` for software-based serial communication.

2. Wi-Fi Configuration:

- The code sets the Wi-Fi network credentials, including the network name (SSID) and password. The specified network is "Network SSID".

3. LED and Timing Variables:

- The code defines an LED pin (pin 0) for controlling an LED.

- It also declares two variables (`previousTime` and `totalTime`) for timing purposes.

4. UART Configuration:

- The code sets the baud rate for UART communication to 115200.

- It initializes a software serial object named "uart" on pins 1 (RX) and 3 (TX) to establish communication between the ESP8266 module and the Arduino board.

5. TCP Server Configuration:

- The code creates a TCP server object on port 8888, enabling the ESP8266 module to listen for incoming client connections.

6. UART Communication:

- The code sets up a software serial object named "uart" with specific RX and TX pin numbers.

- The comments suggest connecting the ESP8266 TX pin to the Arduino RX pin and vice versa to establish communication between the two devices.

5.3.2 ESP Code Setup

```
void setup() {
  // Initialize UART communication
  pinMode(ledPin, OUTPUT);
  //uart.begin(UART_BAUD_RATE);
  Serial.begin(9600);
  // Connect to Wi-Fi
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(1000);
    Serial.println("Connecting to WiFi...");
  }
  // Start TCP server
  server.begin();
  Serial.println("TCP server started.");
  Serial.print("IP Address ");
  Serial.println(WiFi.localIP());
  digitalWrite(ledPin, true);
}
```

Figure 5.0.9 ESP Setup

The `setup()` function is responsible for initializing various components and establishing connections. Let's go through the modifications:

1. UART Communication:

The code sets the `ledPin` as an output pin, indicating its usage for controlling an LED. The line `uart.begin(UART_BAUD_RATE);` has been commented out, which means that the software serial communication using the `uart` object is not being initialized.

Instead, `Serial.begin(115200);` is used to initialize the communication through the default

hardware serial port on the Arduino board.

2. Wi-Fi Connection:

The code attempts to connect to the Wi-Fi network specified by the `ssid` and `password` variables. It enters a while loop that waits until the ESP8266 successfully connects to the Wi-Fi network. Within the loop, a delay of 1 second is added, and a message indicating the connection attempt is printed to the Serial monitor.

3. TCP Server:

The code starts the TCP server by calling `server.begin()`. It prints a message to the Serial monitor to indicate that the TCP server has been successfully started. The IP address of the ESP8266 module is also printed to the Serial monitor using `WiFi.localIP()`.

4. LED Control:

The line `digitalWrite(ledPin, true);` turns on the LED connected to the `ledPin`.

5.3.3 ESP Communication

```
void loop() {
  // Check for incoming client connections
  WiFiClient client = server.available();
  if (client) {
    //Serial.println("New client connected.");

    // Read data from the client
    while (client.connected()) {
      while (client.available()) {
        String data = client.readStringUntil('\n');
        data.trim();
        Serial.println(data);
      }

  }
}}
```

Figure 5.0.10 ESP Communication

The `loop()` function continuously performs the following actions:

1. Checking for Incoming Client Connections:

The code uses the `server.available()` function to check if there is a client attempting to connect. If a client is detected, a `WiFiClient` object called `client` is created to handle

the connection.

2. Reading Data from the Client:

The code enters a loop while the client is still connected. Within this loop, it checks if there is any data available from the client using `client.available()`. If there is data available, it reads the data into a `String` variable named `data` using `client.readStringUntil('\n')`. The `'\n'` character is used as the delimiter to read the data until a new line is encountered. The `data` string is then trimmed to remove any leading or trailing whitespace characters using `data.trim()`. Finally, the trimmed `data` is printed to the Serial monitor using `Serial.println(data)`.

Chapter 6

PHYSICAL STRUCTURE

Gael Langerin's open-source library and the MAKERBOT 3D printers were used to 3D print the robotic hand. The forearm was also printed because it included a compartment for the servo motors and the Arduino board. [15]

6.1 The Robotic Hand

The hand consists of five figures, a palm, and a wrist. The hand and forearm were assembled with the help of glue elfy. The fingers attached with the help of copper wire and fingers movement with the help of fishing wire. [15]

Figure 6.0.1 3D Printed Hand

6.1.1 The Fingers

The third phalanx, the second phalanx, the first phalanx (split into two halves), and the metacarpal bone (divided into two sections) made up the six components of the 3D-printed fingers. The third and second phalanges of the open source were fused together using epoxy glue since they lacked a distal interphalangeal junction. Both the two pieces of the metacarpal bone and the two pieces of the first phalanx were cemented together. Except for the third phalanx, which contained the fingertips, each phalange end had two openings, as shown in figure. A 3 mm drill was used to align these holes so that PLA filament could be threaded through them. This served as the joints of the bones by being done in between the

connected phalanges.

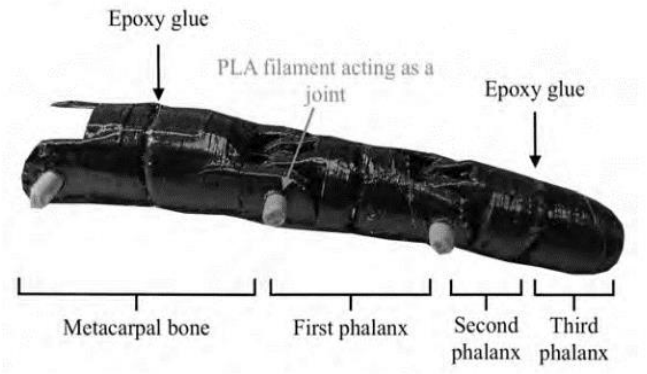

Figure 6.0.2 The side view of a 3D-printed finger, the turquoise parts is the PLA filament acting as joints. [16]

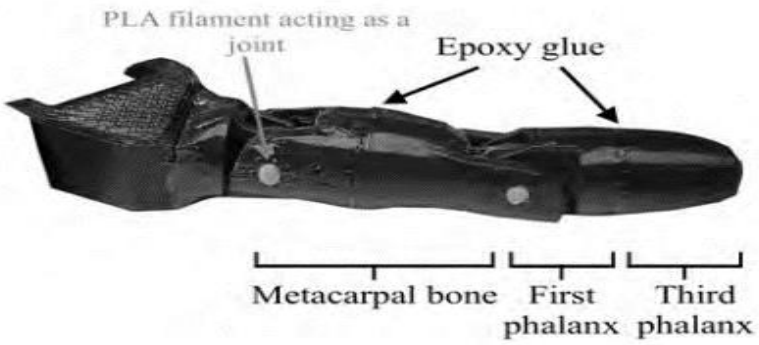

Figure 6.0.3 The side view of the 3D-printed thumb. [16]

The first and third phalanges, as well as the metacarpal bone, make up the thumb's anatomy. The third and first phalanges, the metacarpal bone (in two sections), and a portion that uses a 3D-printed bolt to join the thumb to the palm made up the six components of the 3D-printed thumb. The two pieces of the metacarpal bone as well as the third and first phalanx were attached to one another using epoxy glue. The joints were built in a similar manner to how the other finger joints are.

6.1.2 The Palm

Four sections made up the palm. This made it possible for the hand to imitate human hand motions more faithfully. The four palm structures that extended from the wrist to the carpometacarpal or internarial joints of the fingers had holes in them. To the third phalanx, the fishing lines were fed through these openings. Bolts that were 3D-printed and fitted into holes that had an 8 mm diameter joined the pieces together. For a more aesthetically

pleasing finish, covers were also affixed to the parts.

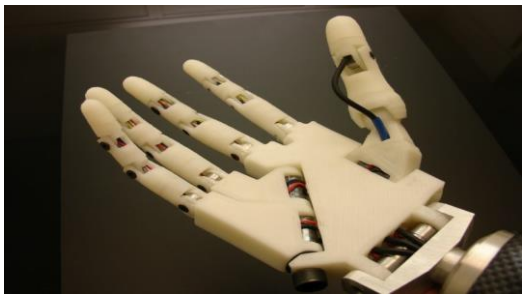

Figure 6.0.4 3D Printed Arm

6.1.3 The Tendons

The wrist, palm, and fingers were connected by ten 75 cm long fishing lines that were threaded from the servo motors. Each finger was strung with two fishing lines, which were then connected at the top of the finger by a knot. The fishing lines were intended to simulate the tendons in the hand and forearm. The finger was stretched out by one fishing line and flexed by another.

Figure 6.0.5 Top view of the hand, without the fingertips. [16]

6.1.4 Wrist

Two gears plus a piece that joins the hand to the forearm make up the wrist. These gears have holes that let fishing wire run through them. The exterior surface of these gears has teeth that essentially regulate wrist movement.

Two gears and a piece that joined the hand to the forearm made up the wrist. A servo motor was employed to simulate the wrist's circular motion. Two gears that were 3D-printed were used for this; see figure 3.5. According to equation 3.1, where Z1 and Z2 were the input and output gears' respective numbers of teeth, the gearing ratio, u, was calculated

to be 6.1 [18]. The smaller gear was attached to the servo motor by using glue to fasten it to the modified rotating attachment piece that was included with the motor. The hand was attached to the larger gear. For added clarity, a cable holder was printed and sewn to the wrist to distinguish the fishing lines from one another.

$$u = {}^{Z2}/_{Z1} \qquad (6.1)$$

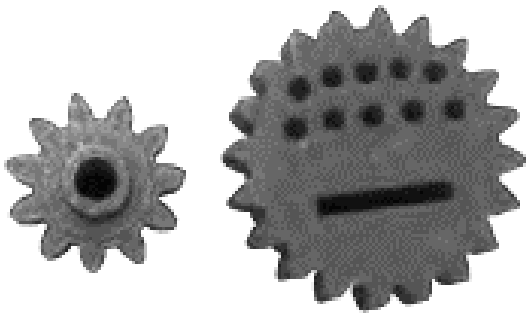

Figure 6.0.6 The small and large gears for the wrist rotational motion. [16]

Figure 6.0.7 The rotational attachment piece for servo motor. [16]

6.2 Forearm

The construction, which was made up of two parts that combined to form the forearm, was enclosed by a cover that was attached to the bottom of the forearm. Five servo motors were also present in the servo bed, another component. Each servo motor was bolted to the servo bed and then connected to each finger of the hand using fishing lines. Five servo pulleys were also 3D printed in order to complete the connection between the motors and the fingers. The modified rotatable attachment components that came with the servo motors were adhered to with epoxy by the servo pulleys. The servo pulleys' holes were threaded with the fishing lines. A screw or the cable holder's perfectly sized holes on the servo bed were used to attach the two 3D-printed cable holder sections to the servo bed. These components were essential to the building process since they made it easier to distinguish between the fishing lines and maintain tension.

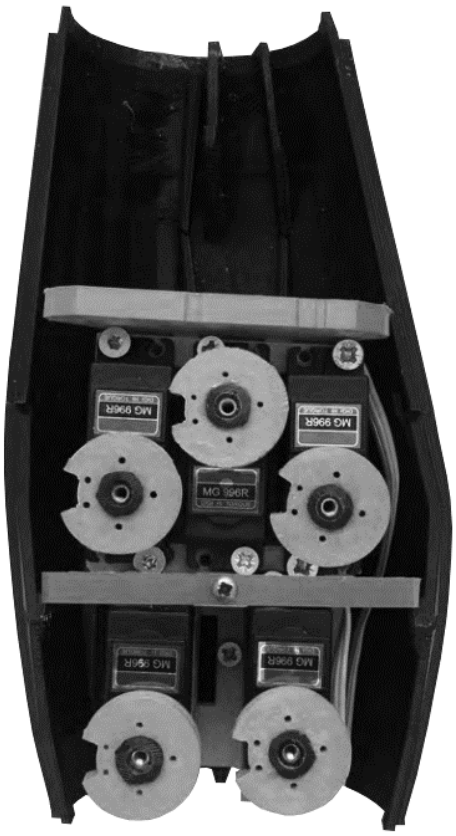

Figure 6.0.8 Forearm with servos. [16]

Figure 6.0.9 3D-printed servo pulley for the servo motor. [16]

The breadboard and the Arduino Uno's analog pins were used to link the servo motors. Servo motors were powered using a breadboard-connected 5 V wall outlet power source and an external 9 V battery supply for the Arduino Uno. It also connected the Arduino Uno and the breadboard to an nRF24L01 transmitter.

6.3 The Humanoid Dummy (Mannequin)

We have purchased a humanoid dummy aka Mannequin from the market to give our robotic arm a sophisticated representation. We have connected the Robotic hand & arm

with the elbow joint of mannequin and using long wires we have placed the circuits, Arduino, ESP module inside the mannequin.

Figure 6.0.10 The Mannequin Connected with the robotic Arm. [15]

Chapter 7

RESULTS AND DISCUSSIONS

7.1 Imitation Test

Real-time imitation of the hand is our main goal of this project. We have tested every hand gesture of the robotic hand. And recorded a video and some snaps are attached here

7.1.1 Fingers Test

First, we performed every finger movement test started by testing the movement of the thumb. In the figure 7.1 we are testing the thumb movement by opening and closing the thumb of our hand (human hand) to check the real-time response of our robotic hand. To check its response time and if it is moving correctly or not. And we have tested its movement successfully.

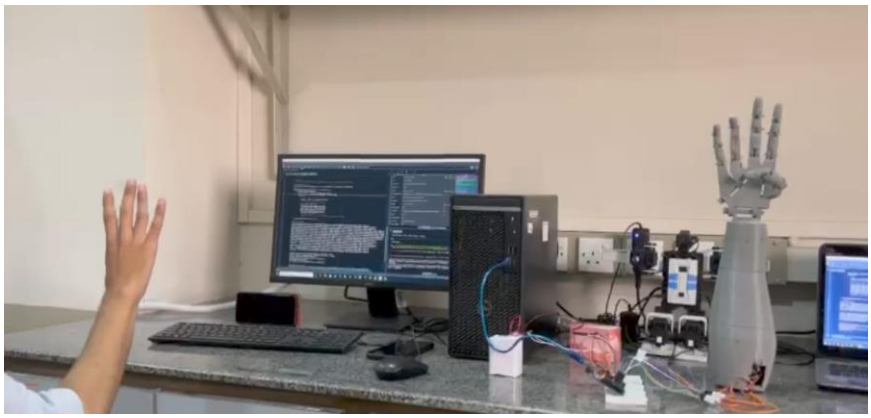

Figure 7.0.1 Thumb testing [15]

Then we tested the fingers one by one in the figure 7.2 we are testing the pinky finger and its flexibility which is working perfectly fine.

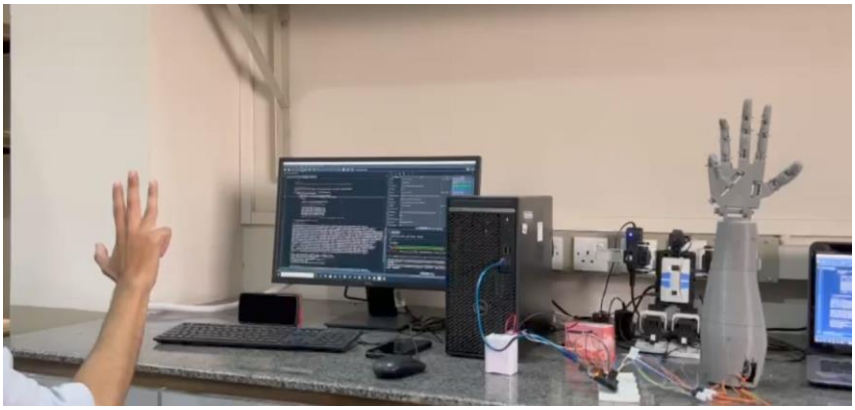

Figure 7.0.2 Pinky Finger testing [15]

Then in this given figure we tested the ring finger response again and again to verify its continuous response and smoothness.

Figure 7.0.3 Ring Finger testing [15]

Now we are testing the movement and response of the index finger in figure 7.4 to check its real-time movements

Figure 7.0.4 Index Finger testing [15]

In figure 7.5 we tested the movement of two fingers at a time and checked their responsive delay

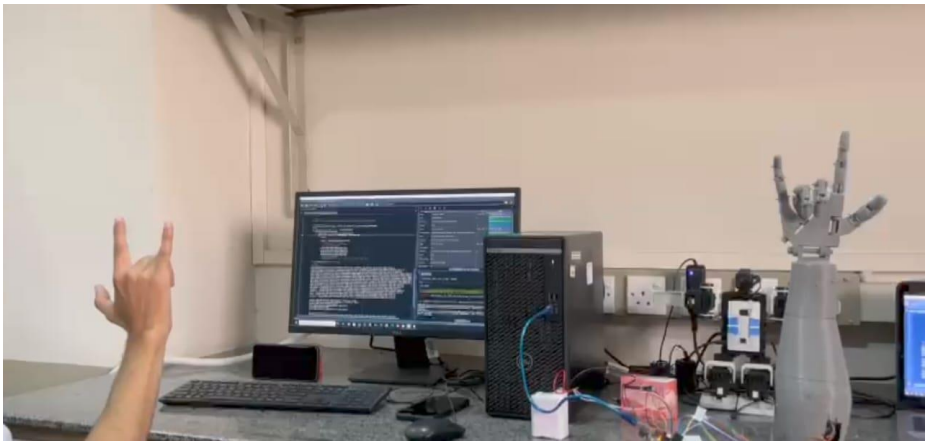

Figure 7.0.5 Ring & Middle finger testing [15]

Then after individual and double fingers test we tested the movement of all fingers at a time. In the figure 7.6 we are testing the whole hand or the palm movements by closing and opening the hand continuously.

Figure 7.0.6 Full Fist to Open hand test [15]

Figure 7.7 is a part of the previous test of the whole hand, figure 7.7 shows the complete hand in a closed state or in a fist shape.

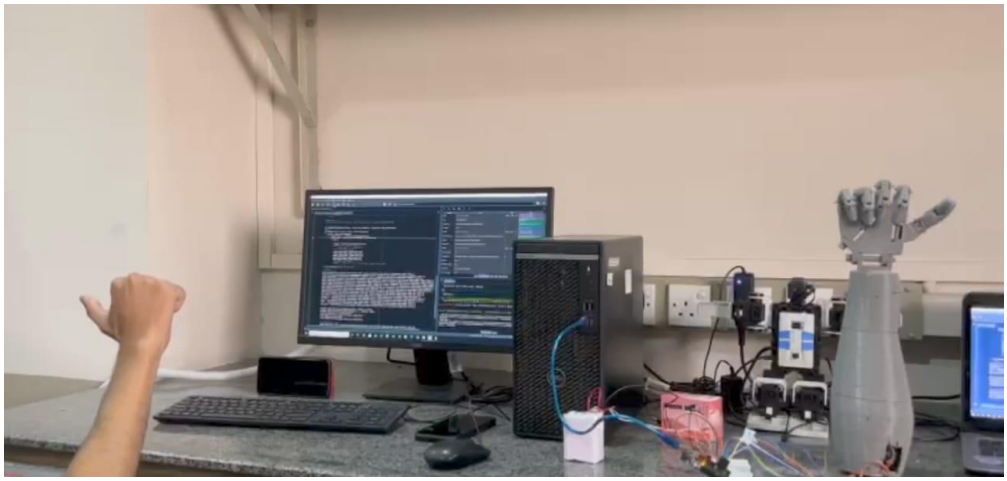

Figure 7.0.7 Fist except Thumb [15]

Chapter 8

CONCLUSIONS AND FUTURE WORK

8.1 Conclusions

The hand gesture driven robotic hand project, demonstrated successfully with Python programming, Servos operating through Arduino, and a successful wireless connection was made using the ESP8266, was, in conclusion, a success. The project's goal of developing a robotic hand that could be operated with hand gestures was successfully accomplished. The experiment also showed how useful gesture control could be for robotic applications.

The project encountered some difficulties, such as the requirement to create a strong gesture detection algorithm and the requirement to make sure that the robotic hand could faithfully replicate the user's actions. These obstacles were overcame, though, and the project was able to succeed.

There are numerous potential uses for the project. The robotic hand, for instance, may be utilised to operate a wheelchair or other assistance device. In industrial contexts, the robotic hand could be utilised to complete activities that are hazardous or challenging for people to complete. It will take the place of people in regions where their lives are in danger and where they have limited access. For instance, in locations at risk of fire, during severe earthquakes, during space exploration missions, in flooded areas, when moving enormous amounts of equipment, etc.The experiment has also shown how useful gesture control could be for upcoming robotic applications. Gesture control is anticipated to become a more popular method of interacting with robots as gesture recognition technology advances.

Here are some additional thoughts on the project:

The team members learned a lot by working on the project. They gained knowledge of the fundamentals of wireless communication, gesture recognition, and robotics. In addition, the endeavor was a lot of fun. The team members appreciated having a challenging project to work on together and seeing their efforts pay off. The undertaking could have a beneficial effect on the world. For instance, the robotic hand could be utilized to provide more independence in the lives of people with disabilities.

8.2 Future Work

Here are some uses of gesture-controlled robots and their future implementations:

Domestic uses:

- **Cleaning**: Housecleaning, laundry, and meal preparation tasks can all be performed by gesture-controlled robots. This might help folks have more time and have an easier time of it. For instance, a gesture-controlled robot may be set up to wash dishes, dust furniture, and vacuum floors.
- **Entertainment**: Video games and virtual reality settings can both be interacted with by gesture-controlled robots. Users might enjoy brand-new, immersive experiences as a result. For instance, a gesture-controlled robot may be used to interact with a virtual reality environment or to control a character in a video game.
- **Security**: Gesture controlled robots can be used for security purposes. For example, a gesture-controlled robot could be used to patrol a home or business and to detect intruders.
- **Assistance for the elderly or disabled:** Robots that respond to gestures can help the elderly or crippled. For instance, a gesture-controlled robot might be utilized as company or to assist a person with a disability about the house.

Industrial uses:

- **Automation**: In industrial environments, tasks can be automated using gesture-controlled robots. Increased productivity and safety in factories may result from this. Robots controlled by gestures could be used to assemble goods or weld components.
- **Inspection**: Robots that can be controlled by gestures can inspect machinery or items. This could aid in raising product quality and spotting possible issues before they cause harm.
- **Packaging**: Gesture controlled robots can be used to package products. This could help to speed up the packaging process and to reduce the risk of errors.

- **Logistics**: Products can be moved around warehouses or distribution centres using gesture-controlled robots. This might contribute to making logistical operations more effective.
- **Maintenance**: Robots that respond to gestures can be used to maintain machinery or other items. This may lessen the requirement for human intervention and increase equipment uptime.
- **Hazardous environments**: In risky situations like nuclear power plants or chemical factories, a robotic arm controlled by gestures might be deployed. By doing this, people may be shielded from dangerous toxins.

Medical uses:

- **Surgery**: Gesture controlled robots can be used to perform surgery. This could help to improve the accuracy and precision of surgery and to reduce the risk of complications.
- **Physical therapy**: Gesture controlled robots can be used to provide physical therapy. This could help patients to recover from injuries or to improve their mobility.
- **Drug delivery**: Gesture controlled robots can be used to deliver drugs to patients. This could help to improve the accuracy of drug delivery and to reduce the risk of side effects.
- **Diagnosis**: Gesture controlled robots can be used to diagnose medical conditions. This could help doctors to identify diseases earlier and to provide more effective treatment.
- **Research**: Gesture controlled robots can be used for research purposes. For example, gesture-controlled robots can be used to study the human brain or to develop new medical treatments.

Future implementations:

- **Virtual assistants**: Gesture controlled robots could be used as virtual assistants. For example, a gesture-controlled robot could be used to control smart home devices, to answer questions, or to provide information.

- **Education**: Gesture controlled robots could be used for educational purposes. For example, a gesture-controlled robot could be used to teach students about science, math, or history.
- **Entertainment**: Gesture controlled robots could be used for entertainment purposes. For example, a gesture-controlled robot could be used to perform in shows or to interact with people in virtual reality environments.
- **Transportation**: Gesture controlled robots could be used for transportation purposes. For example, a gesture-controlled robot could be used to drive a car or to pilot a drone.
- **Space exploration**: Gesture controlled robots could be used for space exploration. For example, a gesture-controlled robot could be used to explore other planets or to conduct experiments in space.

These are just a few examples of the many potential uses of gesture control robots. As the technology continues to develop, we can expect to see even more innovative and exciting applications for these robots in the future.

REFERENCES

[1] S. G. M. C. P. D. M. S. H. S. P. A. Satyam M Achari, "GESTURE BASED WIRELESS CONTROL OF ROBOTIC HAND USING IMAGE PROCESSING," *International Research Journal of Engineering and Technology (IRJET),* pp. 3339-3345, 2018.

[2] S. G. M. C. P. D. M. S. H. S. P. A. Satyam M Achari1, International Research Journal of Engineering and Technology (IRJET), Karnataka, India., 2018.

[3] M. B. MEHNAZ KAZI, "Robotic Hand Controlled by Glove," *KTH ROYAL INSTITUTE OF TECHNOLOGY,* 2020.

[4] S. K. D. K. K. A. P. K. T. K. Dr. (Prof.) Laxmikant Mangate, "Gesture Replication Robo-Arm," *International Journal for Research in Applied Science & Engineering Technology (IJRASET),* pp. 1223-1230, 2022.

[5] K. S. D. P. Aravind Sivakumar, "Robotic Telekinesis: Learning a Robotic Hand," *Carnegie Mellon University,* 2022.

[6] C. Chern-Sheng Lin, "The Manipulation of Real-Time Kinect-Based Robotic Arm," *Hindawi Journal of Sensors,* 2020.

[7] R. J. S. a. C. Taylor, "The anatomy and mechanics of the human hand," *Artificial limbs, vol. 2,* p. 22–35, 1955.

[8] Orthobullets, "Tendons," [Online]. Available: https://www.orthobullets.com/hand/6031/flexor-tendon-injuries.

[9] Arduino, "What is arduino?," 03 February 2020. [Online]. Available: https://www.arduino.cc/en/guide/.

[10] Arduino, "Arduino UNO Rev3," 03 February 2020. [Online]. Available: https://store-usa.arduino.cc/products/arduino-uno-rev3?selectedStore=us.

[11] TowerPro, "MG996R Robot Servo 180 Rotation," 1 June 2023. [Online]. Available: https://www.towerpro.com.tw/product/mg995-robot-servo-180-rotation/.

[12] K. Electronics, "MG996R 180° Digital Metal Gear Servo - 11kg/cm - 55g," 01 june 2023. [Online]. Available: https://www.kiwi-electronics.com/en/mg996r-180-digital-metal-gear-servo-11kg-cm-55g-3020.

[13] J. Hieras, *PERMANENT MAGNET MOTOR TECHNOLOGY: DESIGN AND APPLICATIONS,* 2010.

[14] M. Barr, "Pulse width modulation," *Embedded Systems Programming,* p. 2001, 103–104.

[15] G. Langevin, "Inmoov-open source 3d printed life size robot," 2014. [Online]. Available: https://inmoov.fr/.

[16] M. T. H. K. M. W. Saad Ahmad, *Images by Authors,* Islamabad, ICT, 2023.

[17] inmoov, "Palm," 1 june 2023. [Online]. Available: https://inmoov.fr/wp-content/uploads/2015/07/DSC04799.jpg.

[18] KTH, "Maskinelement: Handbok.," *Avd. för maskinelement, Institutionen för maskinkonstruktion, Tekniska högsk,* 2008.

[19] M. K. &. M. Bill, *Images,* 2020.

APPENDIX A

Python Code

```
import mediapipe as mp
import cv2
import numpy as np
from cvzone.HandTrackingModule import HandDetector
import time
import socket
ESP_IP = '192.168.0.112'
ESP_PORT = 8888
#Getting Models

detector=HandDetector(detectionCon=0.8,maxHands=1)
mp_drawing = mp.solutions.drawing_utils
mp_holistic = mp.solutions.holistic
mp_pose=mp.solutions.pose
mp_Hand=mp.solutions.hands

#myserial = Serial('COM7',9600)
cap = cv2.VideoCapture(1)
hands=mp_Hand.Hands(static_image_mode=False,max_num_hands=2,
min_detection_confidence=0.8)
# Initiate holistic model
def calculate_angle(a,b,c,d):
    a=np.array(a)
    b=np.array(b)
    c=np.array(c)
    radians=np.arctan2(c[1]-b[1],c[0]-b[0])-np.arctan2(a[1]-
b[1],a[0]-b[0])
    angle =np.abs(radians*180/np.pi)
    if d==0:
        if angle >180:
     angle=360-angle
            angle=int(angle)
```

```
    elif d==1:
        angle=int(angle)
    return angle
def pad_zeros(arr):
    send=""
    for i in range(len(arr)):
        if len(str(arr[i])) < 3:
            send = send+str(arr[i]).zfill(3)
        else:
            send=send+str(arr[i])
    return send
def finger_angle(val):
    if val==1:
        angle = 0
    elif val==0:
        angle=90
    return angle
Thumb_angle=0
Index_angle=0
Middle_angle=0
Ring_angle=0
Pinky_angle=0
with mp_holistic.Holistic(min_detection_confidence=0.5,
min_tracking_confidence=0.5) as holistic:
   ## myserial.write(str(0).encode())

    while cap.isOpened():
        ret, frame = cap.read()
        # Recolor Feed
        image = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
        # Make Detections
        results = holistic.process(image)
results_1= hands.process(image)
        #  pose_landmarks, left_hand_landmarks,
right_hand_landmarks
```

```
        # Recolor image back to BGR for rendering
        image = cv2.cvtColor(image, cv2.COLOR_RGB2BGR)
        ##

        #extract landmarks
        try:
            landmarks=results.pose_landmarks.landmark

landmarks_1=results.right_hand_landmarks.landmark
#         print(landmarks)
        except:
            pass

        # Right hand
       # mp_drawing.draw_landmarks(image,
results_1.handLandmarks, mp_Hand.HAND_CONNECTIONS)

        # Pose Detections
        mp_drawing.draw_landmarks(image,
results.pose_landmarks, mp_holistic.POSE_CONNECTIONS)
        #Frame
        cv2.imshow('Gesture Control Robot',
cv2.flip(image,1))
        if results_1.multi_hand_landmarks:
            for handLandmarks in
results_1.multi_hand_landmarks:
                lmList, bbox = detector.findHands(image,
handLandmarks)
                if lmList:

                    fingers = detector.fingersUp(lmList[0])
                    # Detect open or closed fingers
                    # 1 - Open finger, 0 - Closed finger
                   # COnverting Finger angles
                    Thumb_angle=finger_angle(fingers[0])
                    Index_angle=finger_angle(fingers[1])
                    Middle_angle=finger_angle(fingers[2])
```

```
                    Ring_angle=finger_angle(fingers[3])
                    Pinky_angle=finger_angle(fingers[4])
                    # Example output format
                    # fingers = [0, 1, 0, 1, 0]  # Thumb,
Index, Middle, Ring, Pinky


        if cv2.waitKey(10) & 0xFF == ord('q'):
            break
        #Calculating Angle


        left_middle =
[landmarks[mp_pose.PoseLandmark.RIGHT_INDEX.value].x,landmar
ks[mp_pose.PoseLandmark.RIGHT_INDEX.value].y]
        shoulder_R =
[landmarks[mp_pose.PoseLandmark.RIGHT_SHOULDER.value].x,land
marks[mp_pose.PoseLandmark.RIGHT_SHOULDER.value].y]
        shoulder_L =
[landmarks[mp_pose.PoseLandmark.LEFT_SHOULDER.value].x,landm
arks[mp_pose.PoseLandmark.LEFT_SHOULDER.value].y]
        elbow =
[landmarks[mp_pose.PoseLandmark.RIGHT_ELBOW.value].x,landmar
ks[mp_pose.PoseLandmark.RIGHT_ELBOW.value].y]
        wrist =
[landmarks[mp_pose.PoseLandmark.RIGHT_WRIST.value].x,landmar
ks[mp_pose.PoseLandmark.RIGHT_WRIST.value].y]
        hip_joint =
[landmarks[mp_pose.PoseLandmark.RIGHT_HIP.value].x,landmarks
[mp_pose.PoseLandmark.RIGHT_HIP.value].y]


        shoulder_angle_UD=calculate_angle(hip_joint,
shoulder_R, elbow,0)
        shoulder_angle_SW=calculate_angle(shoulder_L,
shoulder_R, elbow,1)
        shoulder_angle_SW=270-shoulder_angle_SW
        elbow_angle=calculate_angle(shoulder_R,
elbow,wrist,0)
        elbow_angle=180-elbow_angle


wrist_angle=calculate_angle(elbow,wrist,left_middle,0)
#Assiging Data to array
```

```
data=[Thumb_angle,Index_angle,Middle_angle,Ring_angle,Pinky_
angle,elbow_angle,shoulder_angle_UD]
        Send=pad_zeros(data)
        message = Send



        # Create a TCP/IP socket
        sock = socket.socket(socket.AF_INET,
socket.SOCK_STREAM)

        # Connect to the ESP
        sock.connect((ESP_IP, ESP_PORT))

        # Send the message to the ESP
        sock.sendall(str(message).encode())
        time.sleep(0.8)
cap.release()
cv2.destroyAllWindows()
```

APPENDIX B

Arduino Code

```
#include<Servo.h>
#define numOfValsRec 7
#define digitsPerValRec 3
int valsRec[numOfValsRec];
int stringlength = numOfValsRec*digitsPerValRec;
Servo Thumb;
Servo Index;
Servo Middle;
Servo Ring;
Servo Pinky;
Servo Elbow;
Servo Shoulder;
String Received;
void setup()
{
  Serial.begin(115200);
  Thumb.attach(3);
  Index.attach(4);
  Middle.attach(5);
  Ring.attach(6);
  Pinky.attach(7);
  Elbow.attach(8);
  Shoulder.attach(9);
  Zero();
}
void Zero()
{
  Thumb.write(0);
  Index.write(0);
  Middle.write(0);
  Ring.write(0);
  Pinky.write(0);
  Elbow.write(0);
```

```
Shoulder.write(0);
}
void Receiving()
{
  if(Serial.available())
  {
    Received=Serial.readString();
    for(int i=0;i<numOfValsRec;i++)
      {
        int num =(i*digitsPerValRec);
        valsRec[i]=
Received.substring(num,num+digitsPerValRec).toInt();
      }
  }
 Servo_Control();
 Received="";
}
void Servo_Control()
{
Thumb.write(valsRec[0]);
Index.write(valsRec[1]);
Middle.write(valsRec[2]);
Ring.write(valsRec[3]);
Pinky.write(valsRec[4]);
Elbow.write(valsRec[5]);
Shoulder.write(valsRec[6]);
}
void loop()
{
Receiving();
}
```

APPENDIX C

ESP Code

```
#include <ESP8266WiFi.h>

#include <SoftwareSerial.h>

// Wi-Fi configuration

const char* ssid = "SSID";

const char* password = "Password";

const int ledPin = 0;

unsigned long previousTime =0;

unsigned long totalTime =0;

// UART configuration

const int UART_BAUD_RATE = 115200;

// TCP server configuration

WiFiServer server(8888);

// UART communication

SoftwareSerial uart(1, 3); // RX, TX (Connect ESP8266 TX to
Arduino RX and vice versa)

void setup()

{

  // Initialize UART communication

  pinMode(ledPin, OUTPUT);

  //uart.begin(UART_BAUD_RATE);

  Serial.begin(115200);

  // Connect to Wi-Fi

  WiFi.begin(ssid, password);

  while (WiFi.status() != WL_CONNECTED)

  {

    delay(1000);
```

```
    Serial.println("Connecting to WiFi...");
  }
  // Start TCP server
  server.begin();
  Serial.println("TCP server started.");
  Serial.print("IP Address ");
  Serial.println(WiFi.localIP());
digitalWrite(ledPin, true);
}

void loop()
{
  // Check for incoming client connections
  WiFiClient client = server.available();
  if (client)
  {
    // Read data from the client
    while (client.connected())
    {
      while (client.available())
      {
        String data = client.readStringUntil('\n');
        data.trim();
        Serial.println(data);
      }

    }
```

```
    }

}
```

CONTACT INFORMATION

1) Muhammad Tayyab
Email ID: muhammadtayyabf19@nutech.edu.pk
Mobile No: +92-317-5478535

2) Saad Ahmad
Email ID: saadahmadf19@nutech.edu.pk
Mobile No: +92-348-4548145

3) Muhammad Haseeb Khalil
Email ID: muhammadhaseebf19@nutech.edu.pk
Mobile No: +92-348-4782685

4) Muhammad Waqas
Email ID: muhammadwaqasf19@nutech.edu.pk
Mobile No: +92-344-5447449

Supervisor Name: Dr. Abdullah Waqas
Email ID: abdullah@nutech.edu.pk