MOTION BASED CONTROLLER FOR HUMANOID ROBOT

A project report submitted in partial fulfillment of the requirements for the award of the degree of

BACHELOR OF TECHNOLOGY
IN
COMPUTER SCIENCE AND ENGINEERING

Submitted by

K.L.S.AKASH	**221910312015**
K.S.P. VIJAY VARDHAN	**221910312019**
M.D.V.KASHYAP	**221910312032**
SREEKANTH SYAMALA	**221910312052**

Under the esteemed guidance of

M. R. ARCHANA

Assistant Professor

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

GITAM SCHOOL OF TECHNOLOGY

(Deemed to be University)

HYDERABAD

April 2023

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

GITAM SCHOOL OF TECHNOLOGY

GITAM

(Deemed to be University)

DECLARATION

We, hereby declare that the project report entitled "**MOTION BASED CONTROLLER FOR HUMANOID ROBOT**" is an original work done in the Department of Computer Science and Engineering, GITAM School of Technology, GITAM (Deemed to be University) submitted in partial fulfillment of the requirements for the award of the degree of B.Tech. in Computer Science and Engineering. The work has not been submitted to any other college or University for the award of any degree or diploma.

Date: 30th March 2023

Registration No.	Name	Signature
221910312015	**K.L.S.AKASH**	
221910312019	**K.S.P. VIJAY VARDHAN**	
221910312032	**M.D.V.KASHYAP**	
221910312052	**SREEKANTH SYAMALA**	

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

GITAM SCHOOL OF TECHNOLOGY

GITAM

(Deemed to be University)

CERTIFICATE

This is to certify that the project report entitled "**MOTION BASED CONTROLLER FOR HUMANOID ROBOT**" is a bonafide record of work carried out by **K.L.S. AKASH (2219103012015), K.S.P. VIJAY VARDHAN (221910312019), M.D.V. KASHYAP (221910312032), SREEKANTH SYAMALA (221910312052)** students submitted in partial fulfillment of the requirement for the award of the degree of Bachelors of Technology in Computer Science and Engineering.

Project Guide	Project Coordinator	Head of the Department
M. R. Archana	**Dr. S. Aparna**	**Dr. K. Sudeep**
Assistant Professor	**Assistant Professor**	**Professor & HOD**
Dept. of CSE	**Dept. of CSE**	**Dept. of CSE**

ACKNOWLEDGEMENT

Our project would not have been successful without the assistance of numerous people. We would like to thank the personalities who were part of our project in numerous ways, those who gave us outstanding support from the project's birth.

We are extremely thankful to our honorable Pro-Vice Chancellor, **Prof. D Sambasiva Rao**, for providing the necessary infrastructure and resources to accomplish our project. We are highly indebted to **Prof. N. Seetharamaiah**, Principal, School of Technology, for his support during the project's tenure.

We are very much obliged to our beloved **Prof. K. Sudeep**, Head of the Department of Computer Science & Engineering, for providing the opportunity to undertake this project and for encouragement in the completion of this project.

We hereby wish to express our deep sense of gratitude to **Dr. S Aparna**, Project Coordinator, Department of Computer Science and Engineering, School of Technology and to our guide, **M. R. Archana** , Assistant Professor, Department of Computer Science & Engineering, School of Technology for the esteemed guidance, moral support, and invaluable advice for the success of the project report.

We are also thankful to all the Computer Science and Engineering department staff members who have cooperated in making our project successful. We would like to thank all our parents and friends who extended their help, encouragement, and moral support in our project work, directly or indirectly.

Sincerely,

K.L.S.AKASH (221910312015)

K.S.P. VIJAY VARDHAN (221910312019)

M.D.V.KASHYAP (221910312032)

SREEKANTH SYAMALA (221910312052)

Table Of Contents

ABSTRACT

As the modern-day technologies advances every day, the access to harmful environments grows exponentially. The need for artificial remote controllers to manipulate an environment is paramount. With controllers, the complexity for precise and accurate movement is high but using a ML framework that can automatically detect the movement made by an individual and subsequently send the data to a robot which mimics the entire movement reduces the complexity for any precise and accurate movement desired by the user. This project aims to simply control a robot by providing a motion based system using multiple sensors and cameras to map out its current environment and to allow it to manipulate its local environment. Additionally, this project also will present the challenges faced when operating a robot with traditional guided controllers as compared to a motion based controller. This project will attempt to fully control a robot in order to completely emulate a human for the advancements of performing human actions in an environment that may or may not be suitable for human survival.

LIST OF FIGURES

LIST OF TABLES

SYMBOLS AND ABBREVIATIONS

AI - Artificial Intelligence

ML - Machine Learning

mAh - milliampere hour

θ - Theta

I2C - Inter-integrated Circuit

GPIO - General Purpose Input/output

PWM - Pulse Width Modulation

MQTT - Message Queueing Telemetry Transport

WLAN - Wide Local Area Network

POT - Potentiometer

IoT - Internet of Things

SSh - Secure Shell

DVR - Digital Video Recorder

CLI - Command Line Interface

IP - Internet Protocol

GND - Ground

VCC - Voltage Common Collector

SDA - Serial Data

SCL - System Clock

CHAPTER 1

INTRODUCTION

1.1 ORIGIN OF ROBOTS

The original purpose of robotics was to relieve humans of some of their more menial tasks. The world's first robot, a hydraulic arm created by George Devol in the 1950s, was the first ever machine of its kind. It was given the name the Unimate, and its primary function was to move things that were heavy. It was sold to General Motors, which was a landmark event in the automotive industry since it considerably decreased the amount of time it took to make automobiles on the assembly line.

Figure 1.1*: The unveiling of the first industrial robotic arm in the 1950's*

In the years that followed, other iterations of robots, along with developments in science and technology, sensors and cameras were integrated into robots, which resulted in an improvement in the capabilities and usefulness of these robots. The Stanford Research Institute was eventually responsible for the creation of the world's first mobile robot in the year 1964. It was the very first general-purpose computer that could evaluate commands and carry out the processing of those directives bit by bit without the requirement for people to provide it with specific instructions.

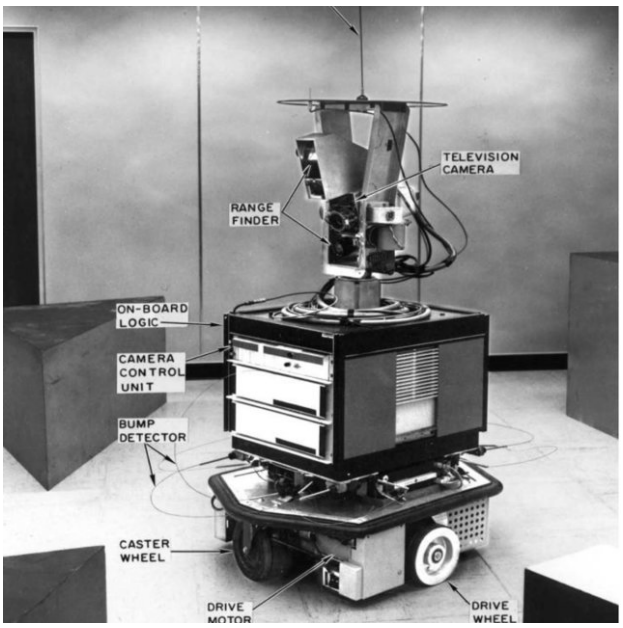


Figure 1.2: *World's first General Purpose Mobile Robot*

1.2 MOTIVATION:

Minimalism and simplicity have always been the core of modern tools and technologies like rust and python. Reducing and abstracting the complexity involved in systems to give the user an easier time to focus on the task has always been a critical design factor in multilevel systems where many modules and subsystems interact. Today's robotic industry relies on heavy AI models to run the machinery with high precision and speed. However, it lacks tools to control robots easily in real-time situations, especially when the robots have the same degrees of freedom as a human body.

Luckily, science fiction authors and filmmakers have identified these problems very early on and have always been ahead of current technologies. It is very common for engineers to take inspiration from these sources and devise ingenious ways to replicate the technology using present-day tools. We, too, can turn to movies like Real Steel and Pacific Rim to look for inspiration for a system to control robots that would otherwise be very difficult to control with precision and speed. The environments the robots operate in are heavily real-time focused, with split-second movements crucial for its operation and achieving its purpose. These movies also give us two types of control systems to achieve the desired control. The first method uses sensors and

mechanical structures attached to the operator's limbs. The system uses the data from the sensors to figure out the exact position of the operator in 3D space. The second method uses a vision system to estimate and mimic the pose the human operator takes.

1.3 PROBLEM IDENTIFICATION

The primary challenge that needs to be addressed, and one that this problem intends to address, is the problem of operating a robot without the need of physical controllers in order to obtain accurate motions for the goal of altering the environment in which the robot operates. The majority of robots that are employed in today's society to aid people typically come equipped with directed controllers or some kind of logical artificial intelligence model or machine learning model. Nevertheless, these take a significant amount of time in addition to a substantial amount of money in order to train the AI models to a degree of autonomy. In the case of guided controllers, it is necessary for the user to provide input for each individual step that is wanted before the robot would carry out the next action. In circumstances where there is a time constraint, the accuracy of the user's movements and the movements themselves will not be optimal for the situation at hand. On the other hand, logical artificial intelligence models and machine learning models are able to make judgments with a relative degree of autonomy without the requirement for inputs from users. The issue with these artificial intelligence models is that in order for them to be able to make decisions on their own, they need not only a large dataset in which they can train themselves, but also a significant amount of time in order to become accurate in their decisions. A motion-based controller should take the place of guided controllers and artificial intelligence models whenever possible, as this will allow for more effective and time-efficient robots. This will result in a reduction in the amount of time required for the robot to make the movements that are wanted, as well as an increase in the robot's accuracy and precision when interacting with and manipulating its surroundings.

1.3 LIMITATION

During the process of developing this project, there have been a number of constraints that have been encountered. These restrictions were the result of a number of different variables, all of which worked together to prevent the robot from reaching its full potential. These limitations are:

- **Mobility**: It has not been possible to retrofit the robot with a versatile movable platform that may expand the robot's applicability. This limitation has prevented this modification from being implemented. The robot cannot move on its own and is therefore confined to a stationary posture. Even if it were possible to install it, the power supply would not be sufficient to enable mobility for the robot even if it was implemented.
- **Power**: The Raspberry Pi and the servo motors are each connected to their own separate power banks that have a capacity of 10000 mAh. However, this power source is unreliable since it is unable to supply a steady voltage, which is necessary for use over an extended period of time. The Raspberry Pi can only function in a condition that consumes a lower amount of power, and the robot's battery life is severely constrained as a result.
- **Cost**: The resources required when developing the hardware for this robot may not be accessible to everyone as it requires a specific set of hardware components in order to function at a reasonable rate.
- **Latency**: Because the humanoid robot is unable to complete its tasks with zero latency, there will always be a little delay between the user's input and the robot's response to that input. This also takes into account the fact that there is a strong network connection established between the user and the robot. The humanoid robot's ability to carry out activities is, in all likelihood, constrained by its connection to the internet.

1.4 OBJECTIVES

The core objectives that this project has are :

- Demonstrate the feasibility and practicality of a motion based controller over a guided controller.
- Present an alternate solution to the existing solutions to the way humans interface with robots
- Reduce the complexity in controlling robots over traditional controllers
- Eliminate time taken for the humanoid robot to interact with its environment
- Provide near limitless accessibility to the humanoid robot by allowing it to be accessed on a virtual private network
- Represent a relatively inexpensive humanoid robot that can manipulate its environment while being practical simultaneously.

1.5 APPLICATIONS

There are a multitude of various applications that this humanoid robot can be used in. The primary application would be when a situation that arises that requires modification of any environment but poses a danger to humans. This humanoid robot eliminates the risk to humans by breaking the biological barriers that we have and can perform the tasks that are required. There are also several other applications such as:

- **Manufacturing**: In the industrial industry, the humanoid robot can be employed to carry out jobs that would normally need the dexterity and ability of a human worker. These tasks which are physically and mentally intensive include the assembly of small parts, painting, and welding.
- **Scientific Research**: Experiments and investigations in the field of science that were once thought to be dangerous to human life can now be carried out by the robot. The humanoid robot's capabilities in terms of applicability and capability can be expanded thanks to the fact that it can be outfitted with end effectors that can hold objects.

1.6 OVERVIEW

It is more difficult to move with greater accuracy when using Controllers. This can be resolved utilizing straightforward machine learning frameworks that compute the required angles and coordinates based on real-time detection of our body movements. The robot is then controlled by these variables without the assistance of a person.

This project seeks to develop a motion-based system to manage the robotic components in order to acquire more exact movements when handling hostile situations or toxic materials. This project also would like to show the disparaging differences between human guided controllers and motion based controllers. This robot can be used in a challenging area and be controlled remotely without the need for human assistance and offers a greater accuracy and precision when interacting with its environment compared to the traditional and existing methods that are currently being used.

CHAPTER 2

LITERATURE REVIEW

2.1 LITERATURE SURVEY

In recent years, there has been a growth in the production of robots that can interact with people in settings including homes, offices, and hospitals. The anthropomorphism, pleasant design, mobility, and interaction with human living environments, among other factors, make the research of humanoid robotics more pertinent. Little robots are typically better since they are easier to handle and less dangerous to people. Robots' physical features, like momentum, alter when they grip an object, depending on what it is. Hence, the learning mechanism is also introduced in humanoid robots. Two models were utilized in a modular control strategy that was described; on one side, the forward model projected the next state from the present state, and on the other, the inverse model predicted the motor command from both the predicted and present states. This motor prediction distinguishes between self-generated movements and disturbances from the outside world.

2.2 ROBOTIC ARM DESIGN

The base research paper used in this project "*Design and implementation of Arduino based robotic arm*" written by Hussein Mohammed Ali, Yasir Hashim, and Ghadah Alaadden Al-Sakkal, implements a 6 degree of freedom robotic arm that is controlled using an app built using the mit app builder to control their arm. The app connects to the HC05 Bluetooth module present on the Arduino to send instructions from a mobile phone to the Arduino uno board. The app provides a slider to control each motor individually from the wait to the gripper. The robot features a 6 degree of freedom movement set. The first three joints, the waist, the shoulder, and the elbow, the MG945 servo motors are used. While the other three joints, the waist roll, the pitch roll and the gripper use the smaller MG90S micro servo motors.

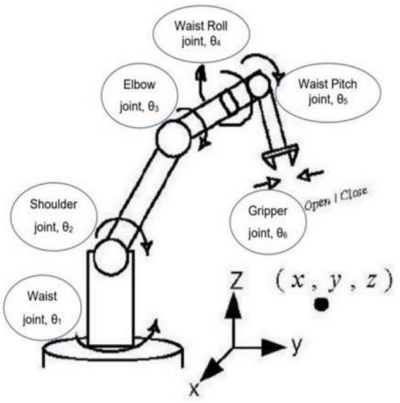


Figure 2.2.1 Design of robotic arm

Figure 2.2.2 Complete design of robotic arm

Parameter	Value
Weight	100 g
Stall Torque	10 kg/cm (4.8 V): 12 kg/cm (6 V)
Operating Speed	0.23 sec/60 degree (4.8 V): 0.2 sec/60 degree (6.0 V)
Operating Voltage	4.8 V a 7.2 V
Dead Band Width	5 micro s
Dimension	40.7x19.7x42.9mm
Temperature Range	0-55 Degree

Table 2.2.1 Specifications of MG945 Servo Motors

Parameter	Value
Weight	13.4 g
Dimension	22.8x12.2x28.5mm
Operating Speed	0.1 sec/60 degree (4.8 v), 0.08 sec/60 degree (6 v)
Operating Voltage	4.8-6.0 Volts
Dead Band Width	7usec
Stall Torque	1.8 kg/cm (4.8 V), 2.2 kg/cm (6 V)
Temperature Range	-30 to+60 Degree C

Table 2.2.2 Specifications of MG90S Servo Motors

This figure explains the complete process of the designed Arduino controlled robotic arm. When the operation starts, the robotic arm is in its initial condition. If a command order from the mobile application user is not valid, then the arm will stay at

its initial condition. However, if the command order is a valid order, it will be sent to the specific motor and the motor will move accordingly. After the order is complete, the robotic arm returns to its initial condition until a new order is received allowing the process to repeat in a continuous loop until the power is turned off.

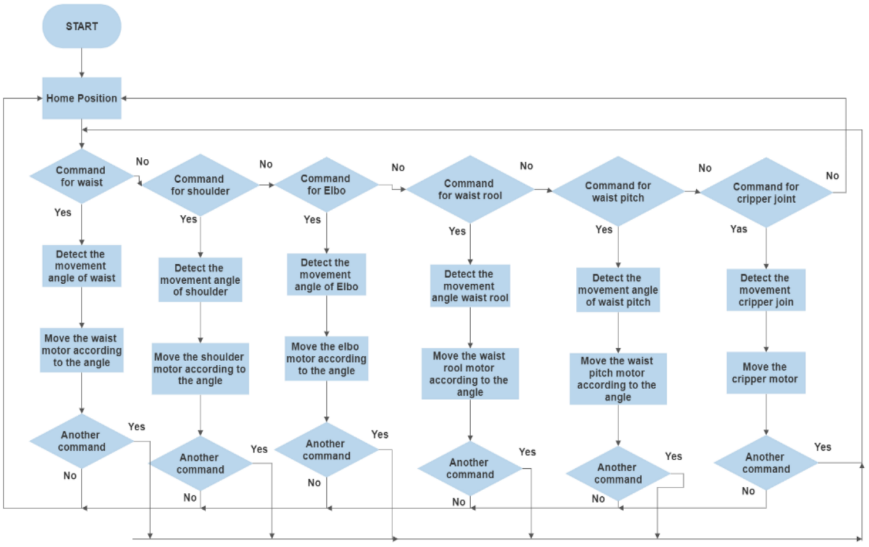


Figure 2.2.3 Flow chart of the designed robotic arm

Waist motor movement is to rotate the whole robotic arm on its base from 0 and 180 degree of angle θ1 according to the servo motors rotation range. Shoulder motor movement requires the most strength in the robotic arm as it carries the entire arm's weight. It moves up and down depending on angle θ2. Elbow motor movement is the movement type for more comfortable maneuvering for the robotic arm. Waist roll motor movement is for the rotation of the arm in its axis between 0 and 180 degrees of angle θ4. Waist pitch motor is the up and down maneuver of that joint. Gripper motor movement is the end effector of the arm that reacts with the environment; it only has the ability to grasp objects making its movement only opening and closing.

A type of medium must exist to transmit the user's commands to the robotic arm in order for the two parties to establish communication. A Bluetooth module has been employed in this study to enable communication between the mobile user and the robotic arm. The user instructs the Arduino using a smartphone application, which communicates with it via Bluetooth, and the Arduino creates the correct movements

using the motors on the robotic arm. It must be coded such that the Arduino can enable the Bluetooth on the mobile device in order for the configuration to work. For each servo motor to be controlled with the appropriate motion, the robot's programming must be precise. The sliders in the application's design are linked to individual motor movements, so as the user adjusts the sliders, the motors respond appropriately.

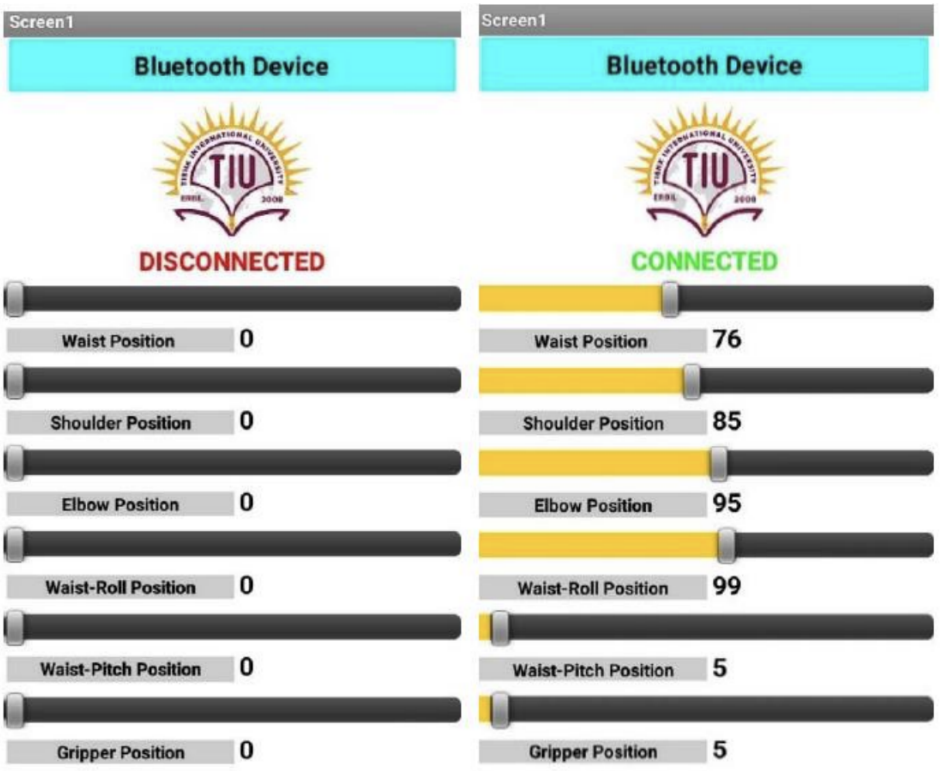


Figure 2.2.4 Before and after connecting mobile applications to Arduino (home position)

2.3 CHANGES IN RESEARCH PAPER

The research paper's model is straightforward to use and effective. However it was too complicated for a simple demonstration. The model used in these projects reduced the degrees of freedom available for the arm making it easier to move using a pose estimation model. The sliders shown in figure 2.2.4 can be hard to control which makes the process of operating the robot slow. The arm's overall performance will be impacted by how well trained and adapted the operator is and since the arm's functionality depends on the app, it might be hard to visualize the path of the arm when multiple sliders are used simultaneously.

The major goal of our project is to control the arm only using our own poses and do away with physical gadgets. Our approach is to build a machine learning model that can recognise the user's actual poses, compute the user's coordinates and angle, and then operate the arm based on the results. It will be much simpler to control the arm more precisely because our model does not require any additional equipment. This model greatly decreases the time complexity and will greatly simplify it. The project's ultimate goals are accessibility, simplicity, and efficiency. In our earlier model, we restricted the robot to only one arm since it was more portable and easy to use. The design is very straightforward and easy to comprehend.

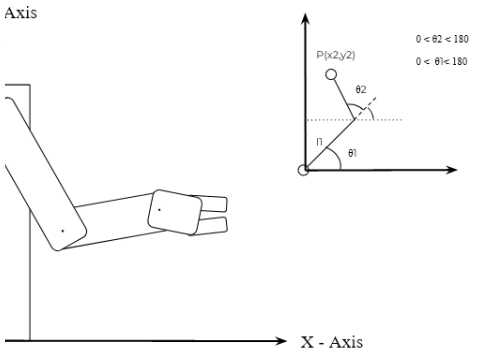


Figure 2.3.1 Design of Robotic Arm

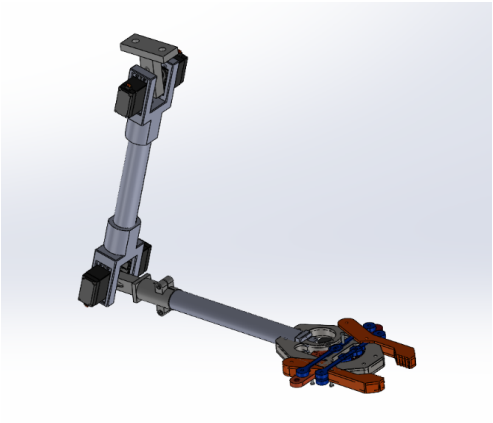


Figure 2.3.2. Complete design of arm

The robotic arm used a raspberry pi as a controller and used I2C protocol to interact with the PCA 9685. The arm's kinematics were calculated by understanding its geometry which explains its movement characteristics and orientation with respect to its start points and end points. The kinematics consisted of two parts, forward and backwards. The general design of the arm is shown in Figure 2.3.1. The figure explains the various joint positions and motor positions with their constraints. The arm consisted of 5 motors, in which, the shoulder joint used 2, the elbow joint used 2 and the end effector used 1 which are controlled as individual sections using a servo motor driver.

Parameter	Value
Weight	55g
Dimensions	20.7x19.7x42.9mm
Stall torque	9.4kg/cm (4.8V); 11kg/cm (6.0V)
Operating Speed	0.19s/60deg (4.8V); 0.15s/60deg (6.0V)
Operating Voltage	4.8-6.6V
Dead band width	1us

Table 2.3.3 Specifications of MG996R Servo Motors

The Raspberry pi has 40 pins on it, in which, two are 5V input, and another two are 3.3V power pins and contain 8 ground pins[1]. Rest of the pins are general purpose input/output(GPIO) pins starting from GPIO0 to GPIO28. The PCA9685 Servo Driver board provides 16 12-bit PWM channels while taking up only two GPIO pins on the raspberry pi[2]. Figure 2.3.4 shows the pin diagram of raspberry pi. The figure 2.3.4 shows PCA9685.

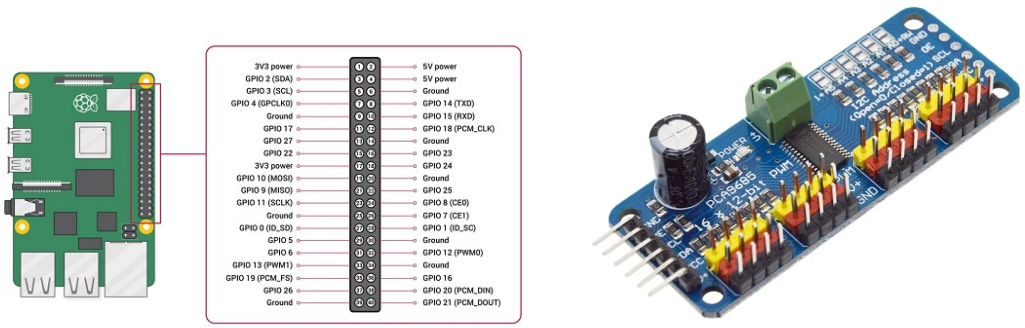


Figure 2.3.4 Pin Diagram of raspberry pi and PCA9685

When the raspberry pi starts executing the driver code, the arm takes its initial position. When the MQTT publisher sends data to the Raspberry pi, the data is read and validated. If the change in angle position falls in the error rate; the arm remains in its initial position. If the change in angle position falls outside the error rate, the arm uses the data to angle the joints. The arm remains in its augmented position which is then taken as its initial position in a continuous loop until the arm is powered down.

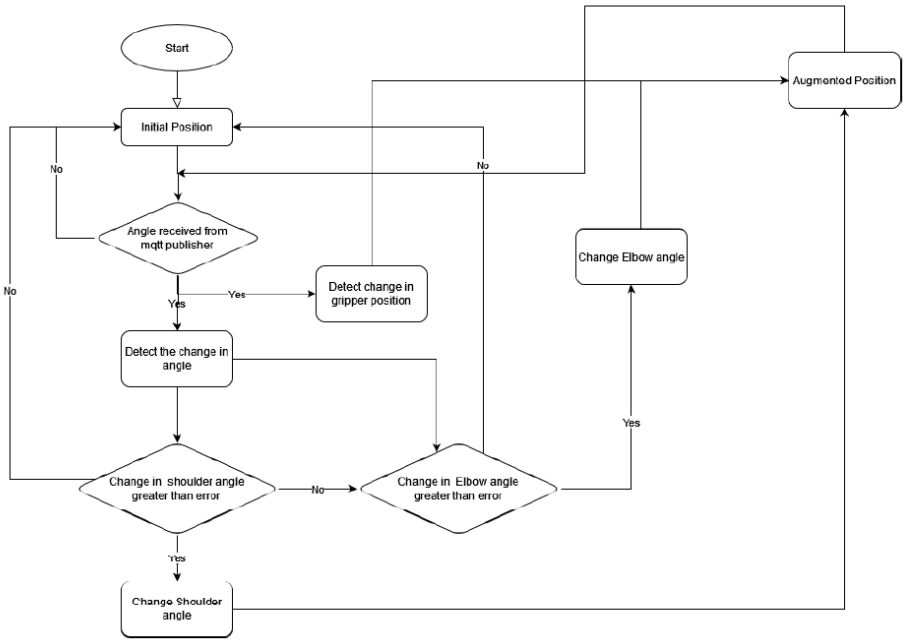


Figure 2.3.5 Flow chart of the robotic Arm

Figure 2.3.1 illustrates the circuit diagram of the robotic arm. The raspberry pi acts as the central hub for communication between the pose detection software and the servo motors. To reduce the number of physical GPIO pins occupied on the raspberry pi by the servo motors, we connect a PCA9685 servo driver board. The PCA9685 servo driver board only takes up two physical GPIO pins and one 5v power pin and one ground pin, which it then multiplexes into sixteen 12-bit channels. Each individual channel includes a data line, a power line, and a ground line. An external 5V power supply is attached to the PCA9685 servo driver board to power the servo motors.

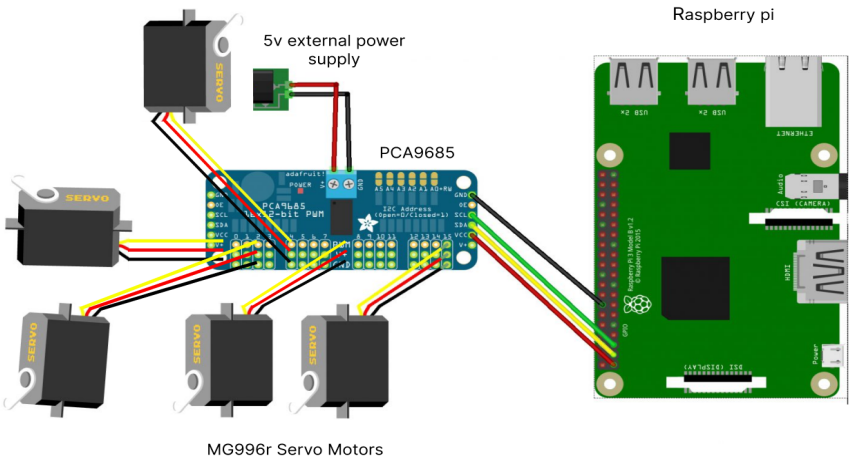


Figure 2.3.6 Circuit diagram of the raspberry pi

To control the end effector of the robotic arm, we had to use a gripper controller. This gripper controller uses a ESP32 board, and a spring loaded B10K potentiometer enclosed in a plastic encasing. Each fingertip contains an end cap with is linked to the potentiometer using a string which moves the potentiometer proportional to the movement of the finger. The movement is captured by the ESP32 board which translates the potentiometer value into servo angle. ESP32 communicates wirelessly over WLAN with the raspberry pi using MQTT protocol, acting as a publisher, to deliver the data.

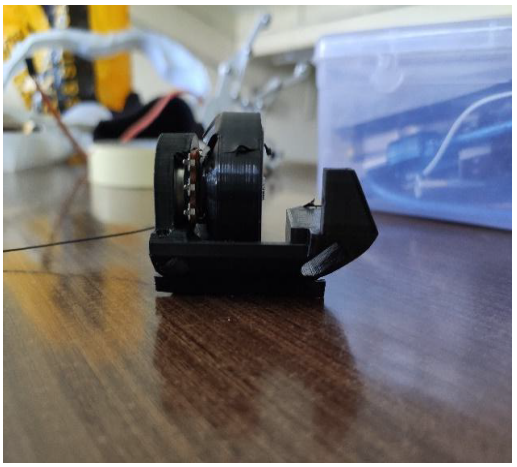


Figure 2.3.7 Pot spool front

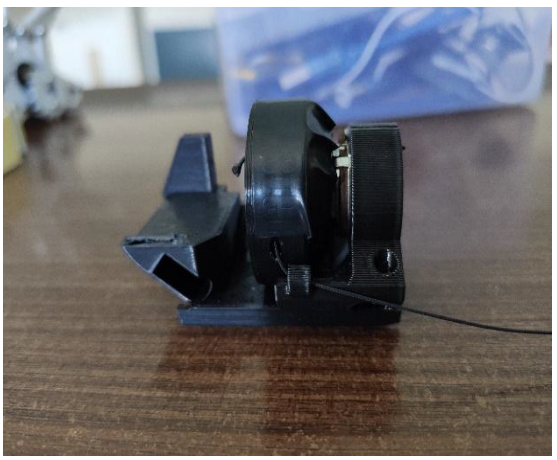


Figure 2.3.8 Pot spool back

Figure 2.3.5 Illustrates the circuit diagram of the ESP32 board. When the board is turned on, it starts reading the value of the potentiometer. After verifying the values received from the potentiometers, ESP32 tries to establish communication with the raspberry pi

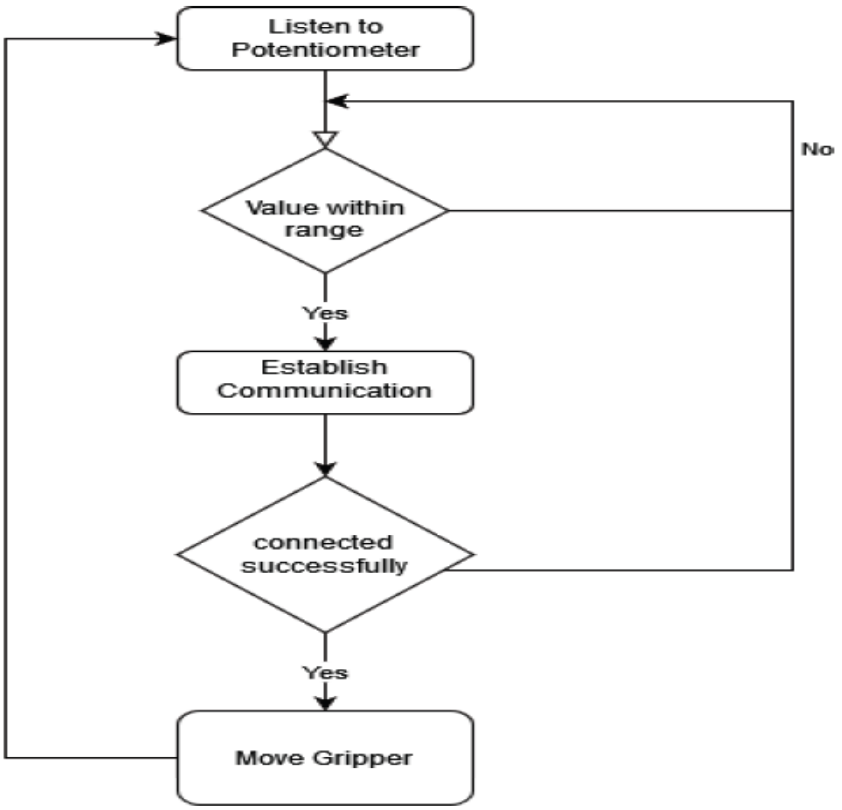


Figure 2.3.9 Flow chart of ESP32

2.4 LIMITATIONS IN PREVIOUS MODEL

The previous model was easy to use and built but it had some obvious drawbacks mainly its lack of a range of motion that was achieved by the base model . Because it had only two degrees of freedom, the robotic arm was limited to the XY axis. Due to the robotic arm's lack of wheels, it is completely immobile. The arm could only be connected to and used by systems using the same local network and internetwork operation was not possible. The system running the pose detection model had to be in the same network as the arm’s controller making it harder for the arm to be deployed in areas far away from the control center. The pose estimation algorithm was limited to only the XY plane and could not accurately predict the

coordinates in 3D world space. The operator had no way of looking at the actions or the surroundings the arm is deployed in. There was no failsafe in place for when the pose estimation model does not work as intended.

2.5 CHANGES MADE IN THE CURRENT MODEL

A complete robot was designed with two arms, a head. The three-degrees-of-freedom arms finally allows the arm to move in 3D space and not be restricted to only a single curve and provides a greater range of motion. The robot was designed to be compatible with a mobility module by providing a snap fit hole at the bottom of the body. The networking has been changed so that the operator can access the robot from any network and from anywhere. A camera module has been added to provide vision to the operator so that they can see the performing actions and the surroundings. The complexity and the confidence level of the pose estimation model has been increased and the output is updated to provide 3D world coordinates. Controller support was added to provide a backup in case the pose estimation model fails.

CHAPTER 3

PROBLEM ANALYSIS

3.1 SCOPE AND METHODOLOGY

Our project aims to bridge the gap between the current stage of integrating robotics with AI and ML and the future vision of robots achieving full autonomy, by providing a framework for human intuitive control of robotics. The objective of this project is to imagine a future for robotics that can be seamlessly integrated into our daily lives and remove the existing biological barriers.

3.2 PROBLEM STATEMENT

Robots in industries currently require an operator using a controller, such as a joystick or a remote, which limits their flexibility as their movements are pre-programmed as well as impacting their efficiency. This process necessitates a well-trained operator with sufficient knowledge of the controller which results in a lengthy training period.

3.3 EXISTING SYSTEM

A diverse range of controllers and AI models are utilized by contemporary systems to guide robots in performing designated tasks. The most prevalent tool is an assisted controller that leverages AI models to generate intermediate tasks based on human instructions. Logical AI models are employed in scenarios where real-time human operation is not feasible, but Machine Learning and Deep Learning models have recently superseded these logic-based systems.

3.4 PROPOSED SYSTEM

We are substituting a joystick-like controller with a motion-based controller, enabling a humanoid robot to track and replicate the movements of the operator. This approach has the advantage of eliminating the costly and time-intensive training of human operators on the controller's functionality, and reducing the time taken to move the robot by 47.3% ultimately leading to a more efficient system.

3.5 SOFTWARE REQUIREMENTS

Operating System: Windows 10, PiOS

Framework: Mediapipe

Language: PYTHON

IDE: Jupyter , VSCode,

3.6 HARDWARE REQUIREMENTS

Processor: INTEL

Hard disc: 500GB

RAM: 4GB

System with all standard accessories like monitor, keyboard, mouse, etc.

CHAPTER 4

DESIGN

4.1 SYSTEM ARCHITECTURE

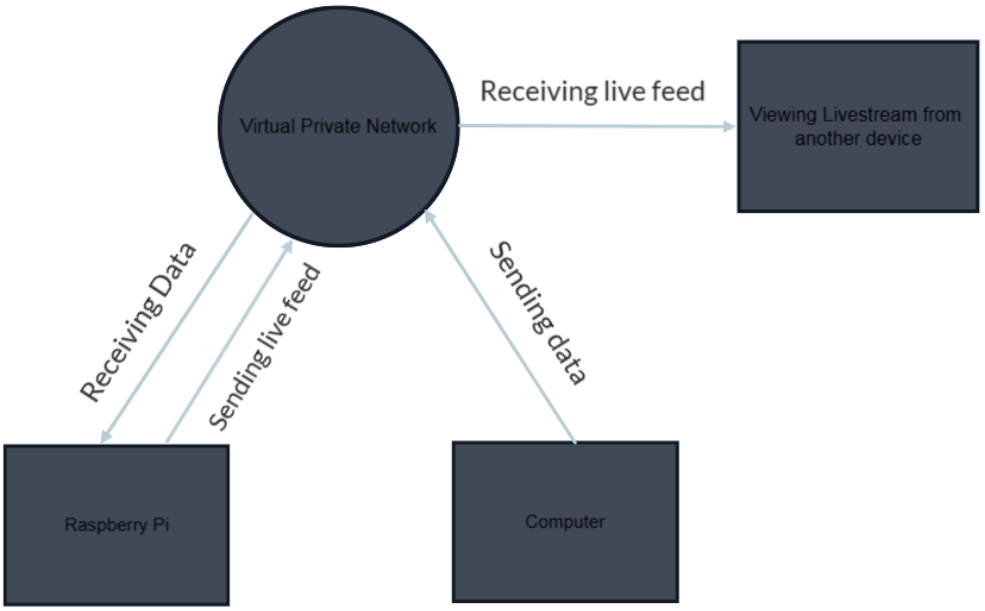


Figure 4.1.1 illustrates the network design of the humanoid robot.

To enable the robot to receive and process data sent from the PC, we need to establish a connection between the PC and the Raspberry Pi. As the Raspberry Pi falls under the category of IoT devices, there exists specific methods and protocols for communication among such devices. In this case, we are using a Virtual Private Network which is hosted through ZeroTier's global networking platform which enables limitless access between the user and the robot [3].

From the following figure, the computer will capture the movements from the user and send those movements to the Raspberry Pi through the virtual private network via a lightweight communication protocol for IoT devices named MQTT. The Raspberry Pi will capture the movements and send the movements to the servo motors. The Raspberry Pi will simultaneously capture its environment in real-time to

a web server which can be accessed using a web browser on any device as to increase the applicability and simplicity of the program.

Figure 4.1.1 accurately depicts the design of all the components and how they will communicate with each other in order for the robot to execute its task.

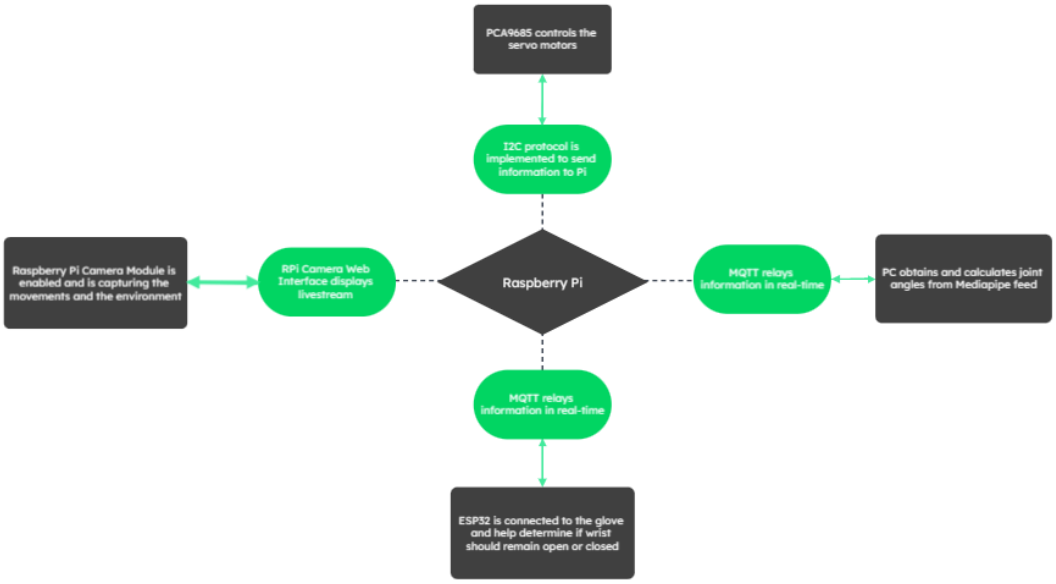


Figure 4.1.2 represents the System Architectural design of the humanoid robot.

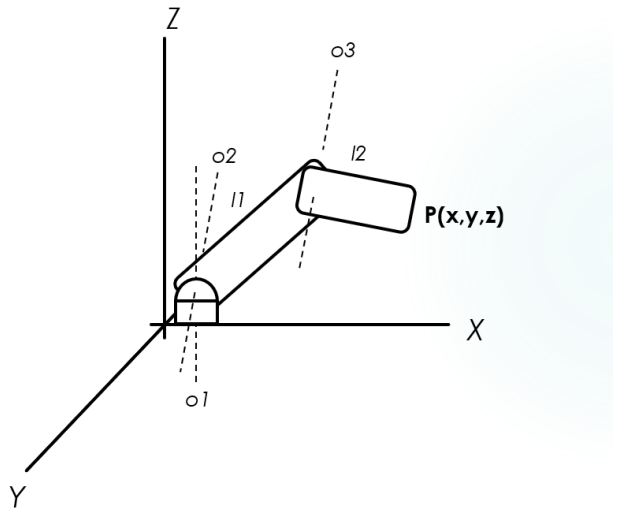
From figure 4.1.2, we can visualize how the Raspberry Pi will interact with various softwares and hardware. In this case, Raspberry Pi receives and processes the angles calculated by the user and sends that information to the PCA9685 which assists in the movement of the servo motors. The Raspberry Pi will also use RPi Cam Web Interface to broadcast its surroundings using its onboard camera module.

The Raspberry Pi acts like the brain of the robot such that all the processes required for the robot to receive and produce an output must be processed from the Raspberry Pi. The Raspberry Pi uses wired connection from its General-Purpose Input Output or GPIO Pins to control the movement of the servo motors with the assistance of PCA9685. It also uses wireless connections via Wi-Fi/Ethernet to establish the livestream video of the robot's point of view as well as the relay of information in real-time via Message Queuing Telemetry Transport (MQTT) communication protocol [4].

4.2 HUMANOID ROBOT DESIGN

The robotic arm uses a raspberry pi as a controller and uses I2C protocol to interact with the PCA 9685 servo driver board. The arm's kinematics are calculated by understanding its geometry which explains its movement characteristics and orientation with respect to its start points and end points[5].

The kinematics consist of two parts, forward and backwards. The general design of the robot is shown in Figure 1. The figure explains the various joint positions and motor positions with their constraints. The arm consists of 5 motors, in which, the shoulder joint uses 2, the elbow joint uses 2 and the end effector uses 1 which are controlled as individual sections using a servo motor driver.



Forward Kinematics:

$X = \cos(o1)X(l1\cos(o2)+l2\cos(o3))$

$Y = \sin(o1)X(l1\cos(o2)+l2\cos(o3))$

$Z = l1\sin(o2)+l2\sin(o3)$

Figure 4.2.1 3D Kinematics

The mechanical design of the robot consists of designing parts around motors such that they can lift the mechanical parts and well as carry their own weight. The project uses Fusion 360 to design the mechanical parts of the arm. The joins consist of Three parts, the first joint provides an articulation point about the XY plane and ZY plane and is designed to mimic the motion at the shoulder. The motors are attached to 3d printed links of length 160mm which also acts as a housing for the elbow actuator..

The mechanical joins are configured for 3D printing using cura software using 100% infill and a complete infill ensuring the rigidity and strength of the printed parts.

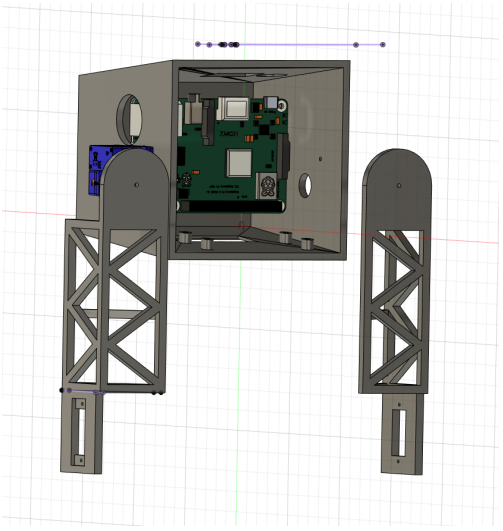


Figure 4.2.2 3D model of Body

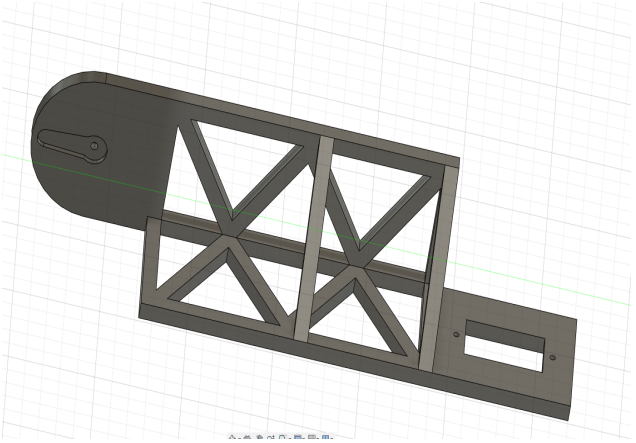


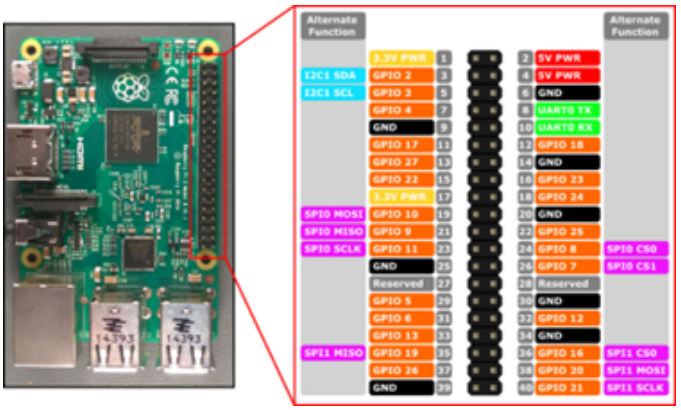
Figure 4.2.3 3D model of Arm

The complete design is depicted in Figure 2. The arm uses 2 motors at the shoulder joint giving 2 degrees of freedom and one motor for the elbow joint totalling 3 degrees of freedom for each arm. The design was made with modularity in mind, the user can freely increase or decrease the degrees of freedom by simply adding or removing articulation points and changing end effectors based on the need. The joints use MG90S servo motors. The Specification of MG90S is given in Table 4.2.1

Parameter	Value
Weight	13.4g
Dimensions	22.8x12.2x28.5mm
Stall torque	1.8 kg/cm (4.8 V), 2.2 kg/cm (6 V)
Operating Speed	0.1 sec/60 degree (4.8 v), 0.08 sec/60 degree (6 v)
Operating Voltage	4.8-6.0 Volts

Table 4.2.1 Specification of MG90S Servo Motors [6]

The Raspberry pi has 40 pins on it, in which, two are 5V input, and another two are 3.3V power pins and contain 8 ground pins. Rest of the pins are general purpose input/output(GPIO) pins starting from GPIO0 to GPIO28. The PCA9685 Servo Driver board provides 16 12-bit PWM channels while taking up only two GPIO pins on the raspberry pi. Figure 3 shows the pin diagram of raspberry pi. The figure 4 shows PCA9685.



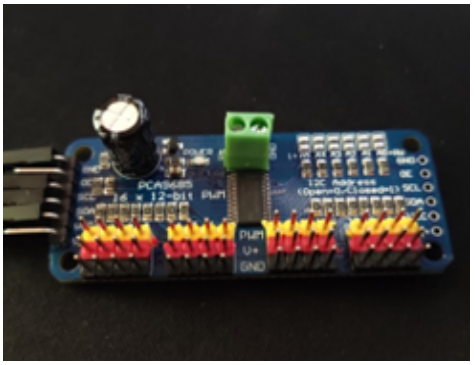


Figure 4.2.4 Pin Diagram of Raspberry pi[2]

Figure 4.2.5 PCA9685

Figure 4.2.6 illustrates the control flow of the robot. When the raspberry pi starts executing the driver code, the robot takes its rest position. When the mqtt publisher sends data to the Raspberry pi, the data is read and validated. If the current mode is set to controller, the driver program increments the position of the arms according to the analog input received from the controller and when the mode is switched to pose estimation, the raspberry pi receives the positions of the joints it's supposed to take. If the change in angle position falls within the error rate, the arm remains as it is. If the change in angle position falls outside the error rate, the arms use the data to angle the joints. The robot remains in its augmented position which is then taken as its initial position in a continuous loop until the robot is powered down.

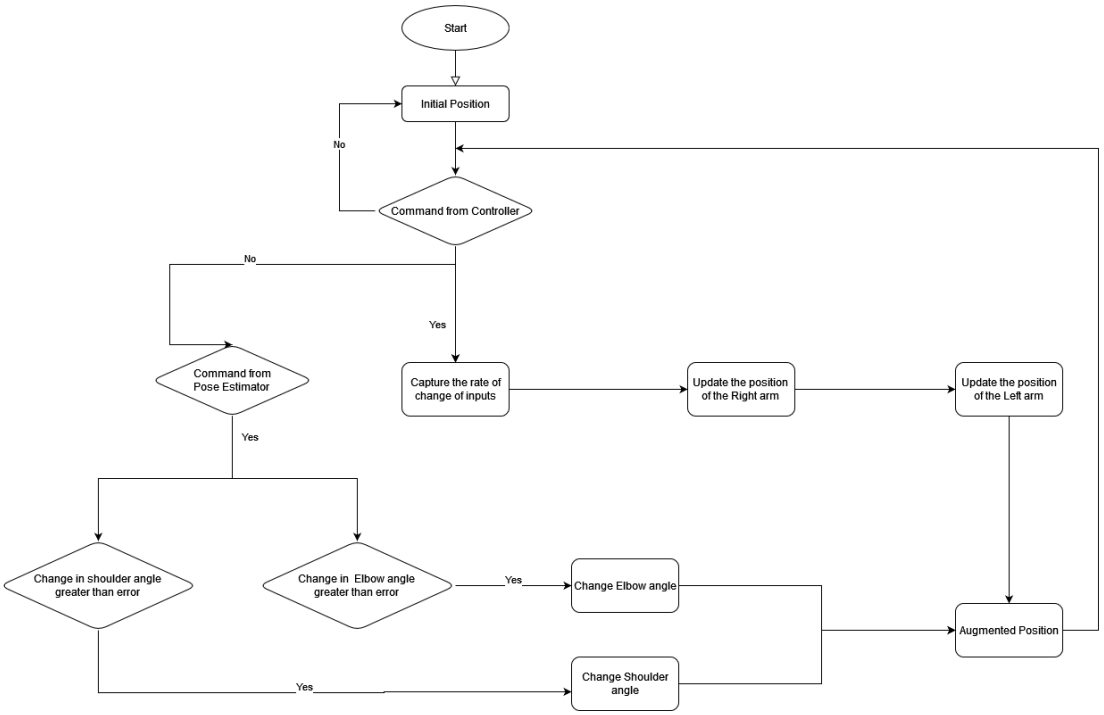


Figure 4.2.6 Flow Chart of the Robot

Figure 4.2.7 illustrates the circuit diagram of the robot. The raspberry pi acts as the central hub for communication between the software and the servo motors. To reduce the number of physical GPIO pins occupied on the raspberry pi by the servo motors, we connect a PCA9685 servo driver board. The PCA9685 servo driver board only takes up two physical GPIO pins and one 5v power pin and one ground pin, which it then multiplexes into sixteen 12-bit channels. Each individual channel includes a data line, a power line, and a ground line. An external 5V power supply is attached to the PCA9685 servo driver board to power the servo motors.

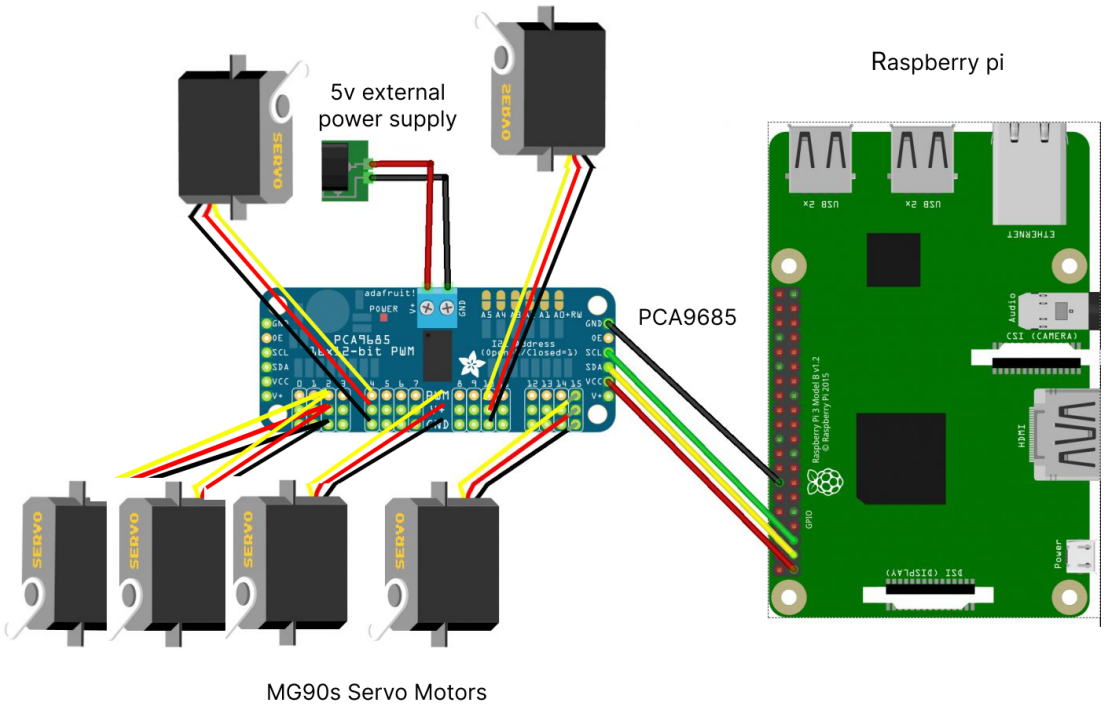


Figure 4.2.7 Circuit diagram of the raspberry pi

4.3 OVERVIEW OF TECHNOLOGIES

4.3.1 TECHNOLOGIES USED

4.3.1.1 RASPBERRY PI:

The Raspberry Pi comprises a set of single-board computers and an open-source ecosystem which finds applications in diverse fields such as programming, hardware projects, home automation, edge computing and many more. Additionally, the Raspberry Pi computer is operated using the Raspberry Pi Operating System which is a lightweight adaptation of the Linux OS to increase versatility and applicability in the various operations that it will face.

4.3.1.2 PYTHON

Python is a high-level interpreted language that serves as a general-purpose programming language. It was developed by Guido van Rossum and made its debut in

1991. Python boasts of a design philosophy that accentuates code readability and a syntax that allows programmers to express concepts in fewer lines of code, making extensive use of significant white space. It also presents constructs facilitating explicit programming regardless of small scale or industrial-scale projects.

4.3.1.3 PUTTY:

A terminal emulator, which is open-source and enables remote connection from devices on the same network through Secure Shell or SSH. PuTTY enables users to remotely connect to other devices or SSH (Secure Shell) into the devices through the network. This reduces the need to have an external monitor to be connected to the Raspberry Pi in order to interface with it, hence reducing the overall complexity any time any future iterations or optimizations need to be implemented.

4.3.1.4 RPi CAM WEB INTERFACE:

The RPi Cam Web Interface functions as a web interface specifically designed for the Raspberry Pi Camera module and can be utilized for diverse applications such as surveillance, DVR recording, and time-lapse photography [7]. This establishes a web-server in which any stream that is broadcasted can be accessed on any web-browser on any device that is connected in the virtual private network. The RPi Cam Web Interface is able to maintain a steady 24-30 frames-per-second while broadcasting as well as significantly reducing the load on the CPU when broadcasting compared to other methods.

4.3.1.5 ZEROTIER:

ZeroTier is a global networking platform that enables users to establish secure virtual networks while providing speed and flexibility to any device with a major OS. ZeroTier allows for the easy flow of data from various applications and systems provided they are connected to the internet. ZeroTier will enable the creation of the Virtual Private Network and offers 256-bit end to end encryption. ZeroTier also maintains a list of devices that are attempting to connect to the network and must be

authorized in order for them to join the network and communicate to other devices. This will increase the security of the network and deter unauthorized nodes to gain access to the network.

4.4 LIBRARIES USED:

4.4.1 NUMPY:

NumPy serves as the fundamental package for scientific computing in Python by introducing support for extensive, multi-dimensional arrays and matrices, complemented by a broad range of high-level mathematical functions that can be applied to manipulate these arrays.

4.4.2 OPENCV:

OpenCV, which is a massive open-source library for computer vision, machine learning, and image processing, has become increasingly crucial in modern systems due to its significant involvement in real-time operations.

4.4.3 TENSORFLOW:

A library for numerical computation called TensorFlow, which is open-source and friendly to Python, enables faster and easier development of neural networks and machine learning.

4.4.4 MEDIAPIPE:

The framework called MediaPipe facilitates the construction of machine learning pipelines that handle time-series data such as audio, video, and so on.

4.4.5 ECLIPSE PAHO:

A python adaptation that is based on the lightweight communication protocol used in many IoT devices MQTT [8]. The communication between a PC and a Raspberry Pi relies on MQTT (Message Queuing Telemetry Transport).

4.5 UML DIAGRAMS

4.5.1 ACTIVITY DIAGRAM

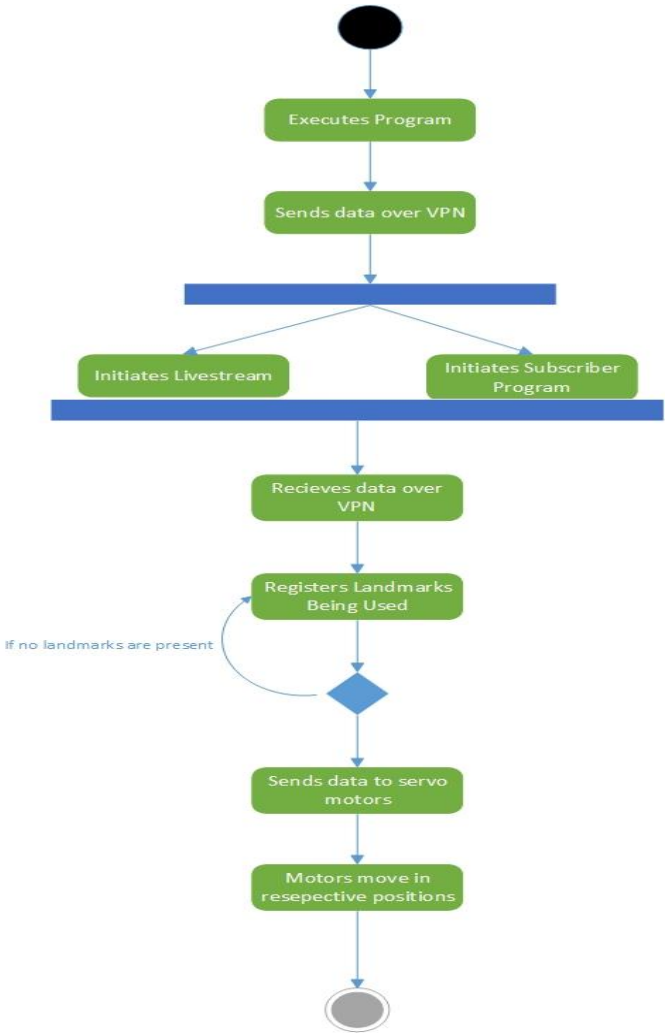


Figure 4.5.1 Activity Diagram

The activity diagram given above represents the various activities that need to take place from the initial state to the final state in order for the desired output to be produced. Each individual process needs to be executed in the correct order for the humanoid robot to execute the given inputs received from the user.

4.5.2 USE CASE DIAGRAM:

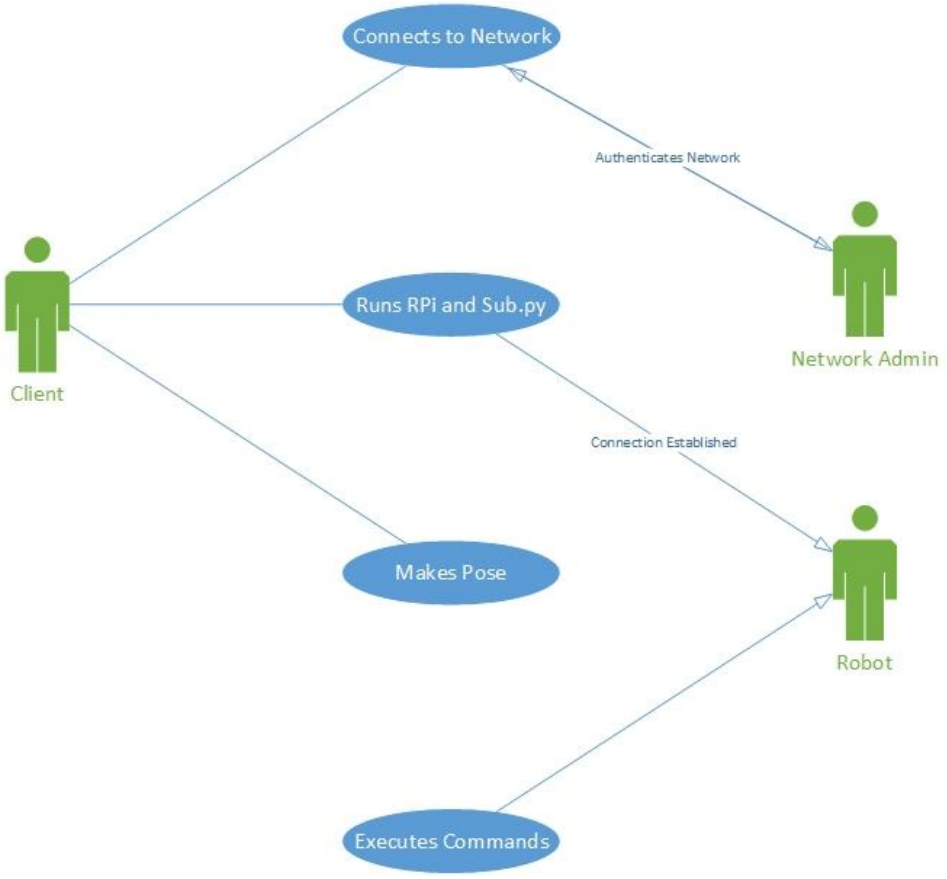


Figure 4.5.2 Use Case Diagram

The use case diagram given below presents a sequence of events that need to take place from various actors to produce a subsequent output. Here the actors are the client or in this case the user, the network administrator, and the humanoid robot itself. The network administrator can also be the user but for the sake of data security and privacy, it is best to keep the roles separate as maintaining multiple networks of various robots may pose a security risk.

4.5.3 SEQUENCE DIAGRAM:

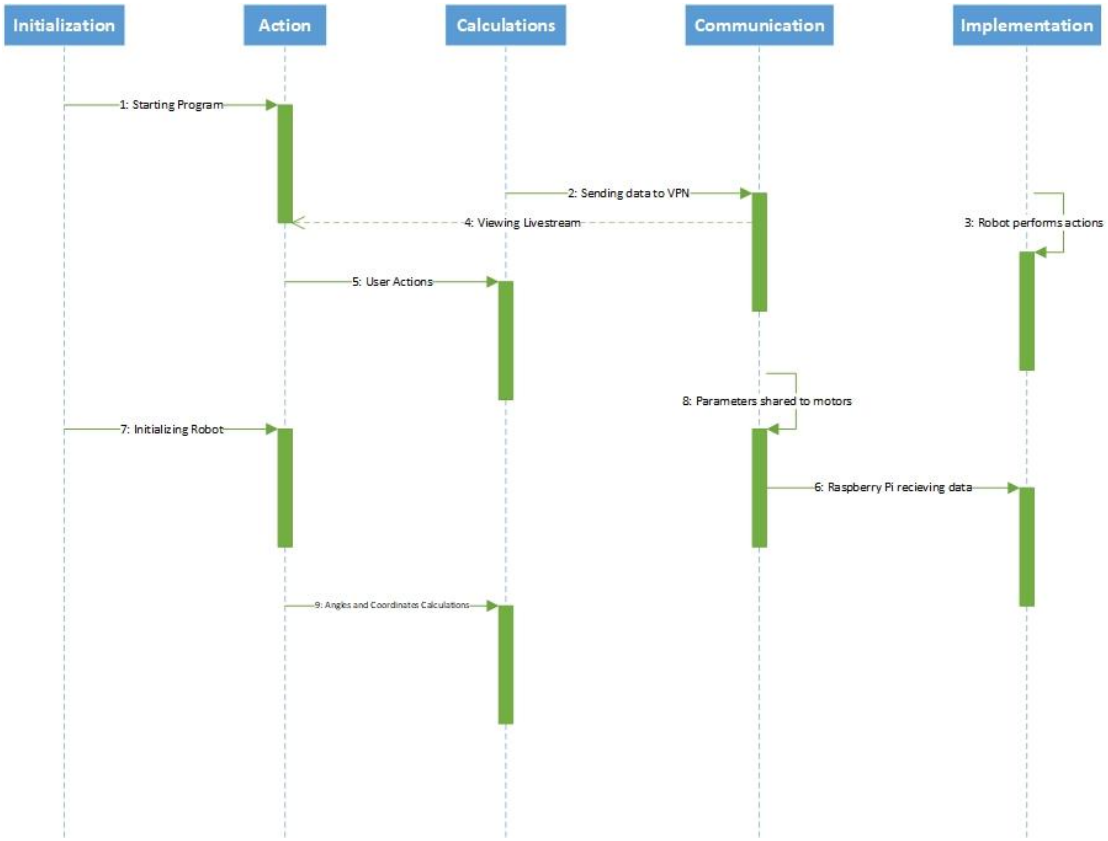


Figure 4.5.3 Activity Diagram

From the given sequence diagram, we can visually see the sequence of events taking place for the humanoid robot to perform any action based on user input.

CHAPTER 5

IMPLEMENTATION

The implementation is split into two parts: the software and the hardware. The software section is again divided into two parts, the frontend and the backend. Frontend section of the code runs on the client's machine which consists of the pose estimation model, the controller driver code, a cli for easier manipulation of the program behavior, and the network programming to send data to the raspberry pi. The backend sanction of the code sits on the raspberry pi which acts as a server and consists of the driver program to control the servo motors via pca 9685 and the network logic.

5.1 CODING

5.1.1 INTRODUCTION

In order for the Humanoid Robot to function as the user desires, A model has to be used in ML for mimicking the arm movements of a human. By doing this, certain outputs will be generated such as coordinates and angles. These outputs are needed for moving the arm in real-time. The program will not be connected directly to the Humanoid Robot but the outputs will be sent to the raspberry pi via a network setup. The angles obtained from the real time motion of our arm will be mimicked by the robotic arm and the necessary action will be implemented by the arm. This task is called Pose Estimation.

5.1.2 PREREQUISITES

There are several modules/frameworks which can be used, such as:

- **Mediapipe**: The Media Pipe Framework is used to create machine learning pipelines for processing time-series data, including audio, video, and other types. With MediaPipe, a developer can create prototypes of cross-platform

applications by quickly and simply combining new and current perceptual components. [9]

- **OpenCV**: OpenCV is a large open-source library for image processing, machine learning, and computer vision. It now plays a significant part in real-time operation, which is crucial in modern systems. [10]
- **TensorFlow**: TensorFlow is an open-source library for numerical computation that is compatible with Python that speeds up and simplifies the creation of neural networks and machine learning algorithms. Neural network model study and application are significantly facilitated and sped up by TensorFlow. [11]
- **JupyterLab**: JupyterLab is the most recent interactive development environment for code, data, and notebooks, which is web-based. Data science, scientific computing, computational journalism, and machine learning workflows may all be configured and organized using the interface's flexibility. The functionality can be expanded and enhanced with expansions thanks to a modular architecture. Several societies in both the scientific and industrial fields have adopted Jupyter Notebooks broadly.[12]

5.1.3 THEORY

Pose Estimation is the task of using an ML model to estimate the pose of a person from an image or a video by estimating the spatial locations of key body joints (key points).

This is typically done by identifying, locating, and tracking several key points on a given object or person. For objects, this could be corners or other significant features. And for humans, these key points represent major joints like an elbow or knee.

Mediapipe is a simpler choice for real-time pose estimation. This is because the instances come inbuilt with the framework and can be adjusted. By analyzing an audio or video input, Mediapipe is able to create Machine Learning pipelines.

The alternatives to Mediapipe are OpenCV and TensorFlow.

Although being excellent alternatives, OpenCV and TensorFlow require adequately trained models using multiple datasets in order to produce the necessary results. These datasets include previously captured images or videos that can be used to train the model. The camera will capture the real-time motion of the human arm and converts it into coordinates,

With these coordinates we can find the angles between several Landmarks or parts of the body.

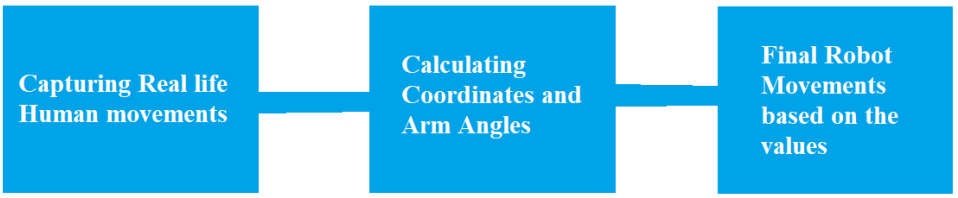


Figure 5.1.3.1 A basic block diagram of Pose Estimation for Humanoid Robot

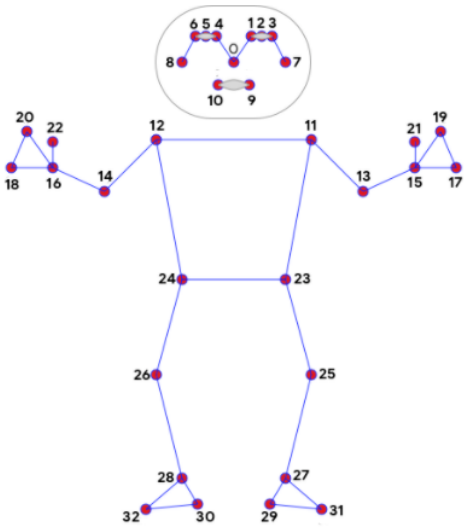
The pose estimation model gives an AP (average precision) score between 68.1 - 73 and a PCK@0.2 (Percentage of Correct Key-points) score between 96.4 - 97.2, which means that while the model can recognize limbs with high accuracy, the precision at which the model can estimate the position of the arms in 3D space is quite average resulting in moderate accuracy of the arms in the ZY axis, especially at the shoulder joints. This mainly occurs because information about the upper arm is blocked when the arms perform specific actions. The elbow angle's accuracy in the ZY axis is better than the shoulder due to a clearer view of the shoulder landmarks. Moving in only the XY axis is highly accurate because the arms are fully visible to the camera which leads to similarly accurate movements from the arms.[13]

5.1.4 DESIGN

There are certain points on the human body called Landmark. A Landmark is a single point which has a particular meaning attached to it such as nose or shoulder.

These landmarks are crucial for recognizing various body parts such that the system can comprehend.

Landmarks are connected to create a grid like structure which are helpful to calculate angles.



0. nose
1. left_eye_inner
2. left_eye
3. left_eye_outer
4. right_eye_inner
5. right_eye
6. right_eye_outer
7. left_ear
8. right_ear
9. mouth_left
10. mouth_right
11. left_shoulder
12. right_shoulder
13. left_elbow
14. right_elbow
15. left_wrist
16. right_wrist
17. left_pinky
18. right_pinky
19. left_index
20. right_index
21. left_thumb
22. right_thumb
23. left_hip
24. right_hip
25. left_knee
26. right_knee
27. left_ankle
28. right_ankle
29. left_heel
30. right_heel
31. left_foot_index
32. right_foot_index

Figure 5.1.4.1 All the landmarks included in the mediapipe framework [14]

For the Humanoid Robot, we will be using only 4 landmarks:

- Hip (left and right)
- Shoulder (left and right)
- Elbow (left and right)
- Wrist (left and right)
- Nose

5.1.5 IMPLEMENTATION

5.1.5.1 Command Line Interface

The command line interface provides several useful flags and options used to set the state of the program.

```
============================================================

 /$$   /$$  /$$$$$$  /$$$$$$$  /$$       /$$$$$$   /$$$$$$
| $$  /$$/ /$$__  $$| $$__  $$| $$      /$$__  $$ /$$__  $$
| $$ /$$/ | $$  \ $$| $$  \ $$| $$     | $$  \ $$| $$  \__/
| $$$$$/  | $$$$$$$$| $$$$$$$/| $$     | $$  | $$|  $$$$$$
| $$  $$  | $$__  $$| $$__  $$| $$     | $$  | $$ \____  $$
| $$\  $$ | $$  | $$| $$  \ $$| $$     | $$  | $$ /$$  \ $$
| $$ \  $$| $$  | $$| $$  | $$| $$$$$$$$|  $$$$$$/|  $$$$$$/
|__/  \__/|__/  |__/|__/  |__/|________/ \______/  \______/

============================================================

Checking for Python Installation...
Python is Installed, Checking if its set up...

You have Python set up.

Useage:
karlos [options]

Available Options:
-n <subscriber-ip>            | -n <local>        : sets the ip address of the pi subscriber    <default: runs on local>
--network <subscriber-ip>     | --network <local> : sets the ip address of the pi subscriber    <default: runs on local>
-p <mqtt-path>                                    : sets the mqtt path of the subscriber
-path <mqtt-path>                                 : sets the mqtt path of the subscriber
-s <controller>               | -s <pose>         : sets the control                            <default: runs with controller>
-start <controller>           | -start <pose>     : sets the control                            <default: runs with controller>
-c                                                : turns on the pi's camera                    <default: runs without camera>
-rf /path/to/file                                 : reads config data from a file
-readfile /path/to/file                           : reads config data from a file
```

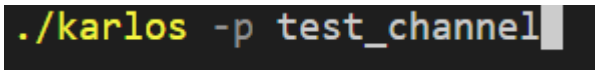
Figure 5.1.5.1 Command Line Interface

The available flags are:

1. -n | --network : takes a string as an input and is used to set the ip address of the server to connect to.

Figure 5.1.5.2 network flag use

2. -n | --path : takes a string as an input and is used to set the mqtt path



.

Figure 5.1.5.3 MQTT Path flag use

3. - s | --start : takes a string as an input and is used to start the program in either controller mode or pose mode.

Figure 5.1.5.4 Start with flag use

4. -c : takes no additional input and is used to enable the video livestream from the raspberry pi.

```
./karlos -c
```

Figure 5.1.5.5 Camera flag use

-rf | --readfile : takes path to a config file as an input and is used to set flags from a config file directly instead of setting it from the CLI.

```
./karlos -rf test.config
```

Figure 5.1.5.6 Readfile flag use

```
1   CONTROLLERFLAG = [bool]    # starts with controller mode if true
2   NETWORKFLAG    = [bool]    # turns on networking if true
3   CAMERAFLAG     = [bool]    # turns on camera if networking is also true
4   CAMERAPATH     = ["str"]   # sets the camera browser app path
5   THREEDMODE     = [bool]    # switches to 2d xy planar coordinates
6   MQTTSERVER     = ["str"]   # sets the server ip. should be used with NETWORKFLAG
7   MQTTPATH       = ["str"]   # sets the server channel. should be used with NETWORKFLAG
```

Figure 5.1.5.7 Structure of the config file

5.1.5.2 POSE ESTIMATION

Posing for the camera enables us to obtain each landmark's coordinates and visibility score. How well the algorithm can comprehend the movements of the landmarks in actual life is determined by the visibility score.

```
landmarks[mp_pose.PoseLandmark.LEFT_ELBOW.value]

x: 0.9282281994819641
y: 1.3837469816207886
z: -0.4659538269042969
visibility: 0.039354272186756134
```

Figure 5.1.5.2.1 The recorded coordinates and visibility score of Left Elbow landmark.

Then, we calculate the angles based on the coordinates received

The angle calculation algorithm works as follows:

Let **X** and **Y** be the coordinates of landmarks on the body (A landmark can be arm, fingers, or any other part). Coordinates will be extracted from three landmarks

- Shoulder joint (left or right)
- Elbow joint (left or right)
- Wrist joint (left or right)

The angle between these coordinates can be found using the atan function. This function is an extended version of the trigonometric inverse tangent function.

Let us consider shoulder, elbow, and wrist as **a1**, **a2**, **a3** alternatives

The coordinates will be recorded in real time based on the human arm.

The angle p will be calculated based on the obtained coordinates, this is given by

`radians=np.arctan2(c[1]-b[1],c[0]-b[0])-np.arctan2(a[1]-b[1], a[0]-b[0])`

The value of p will change depends on the coordinates of a1, a2 and a3

```
#calculating the angle
def calculate_angle(a,b,c):
    a = np.array(a) # First
    b = np.array(b) # Mid
    c = np.array(c) # End

    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*180.0/np.pi)

    if angle >180.0:
        angle = 360-angle

    return angle
```

Figure 5.1.5.2.2 The code snippet for angles calculation

The **`calculate_angle`** function is used to calculate the angles of all the landmarks which are required to move the robotic arms in a 3-Dimensional space.

Note: Nose landmarks will not be used to control the robotic arms, hence it does not output any angles.

```
    # Calculate right Shoulder angle
    Shoulder_angle_xy_right = calculate_angle(hip_xy_right, shoulder_xy_right, elbow_xy_right)
    Shoulder_angle_yz_right = calculate_angle(hip_yz_right, shoulder_yz_right, elbow_yz_right)
    print("Right Shoulder: ", Shoulder_angle_xy_right, Shoulder_angle_yz_right)
    Right_Shoulder_angles = str(Shoulder_angle_xy_right) + ',' + str(Shoulder_angle_yz_right)

    # Calculate right elbow angle
    Elbow_angle_xy_right = calculate_angle(shoulder_xy_right, elbow_xy_right, wrist_xy_right)
    Elbow_angle_yz_right = calculate_angle(shoulder_yz_right, elbow_yz_right, wrist_yz_right)
    print("Right Elbow: ",Elbow_angle_xy_right, Elbow_angle_yz_right)
    Right_Elbow_angles = str(Elbow_angle_xy_right) + ',' + str(Elbow_angle_yz_right)

    # Calculate left Shoulder angle
    Shoulder_angle_xy_left = calculate_angle(hip_xy_left, shoulder_xy_left, elbow_xy_left)
    Shoulder_angle_yz_left = calculate_angle(hip_yz_left, shoulder_yz_left, elbow_yz_left)
    print("Left Shoulder: ", Shoulder_angle_xy_left, Shoulder_angle_yz_left)
    Left_Shoulder_angles = str(Shoulder_angle_xy_left) + ',' + str(Shoulder_angle_yz_left)

    # Calculate left elbow angle
    Elbow_angle_xy_left = calculate_angle(shoulder_xy_left, elbow_xy_left, wrist_xy_left)
    Elbow_angle_yz_left = calculate_angle(shoulder_yz_left, elbow_yz_left, wrist_yz_left)
    print("Left Elbow: ",Elbow_angle_xy_left, Elbow_angle_yz_left)
    print("Nose: ", nose_xyz)
    Left_Elbow_angles = str(Elbow_angle_xy_left) + ',' + str(Elbow_angle_yz_left)
```

Figure 5.1.5.2.3 Calculating the angles of every landmark (except nose) using the function in 7.4.4

These values will be replicated by the robotic arm to produce the desired output.

```
Right Shoulder:  25.276119619412622 15.505170618374319
Right Elbow:  174.17116804524775 131.94118104935328
Left Shoulder:  27.179919704770533 17.173453588773445
Left Elbow:  162.49573449175904 89.21083160795615
Right Shoulder:  25.02277958542532 16.117075018489125
Right Elbow:  171.2501867228153 128.7818055217157
Left Shoulder:  26.892947837082925 17.051984231795597
Left Elbow:  161.80778363592103 88.8876592521417
Right Shoulder:  24.814423583210186 16.324684844478156
Right Elbow:  170.49784523848965 128.2715161987009
Left Shoulder:  26.643571476761604 16.683415220132954
Left Elbow:  161.89116740193555 88.7072675828876
Right Shoulder:  24.051121331040576 15.853119610712003
Right Elbow:  171.92229680992486 128.840580870155
Left Shoulder:  26.406331489065987 16.32046712728574
Left Elbow:  162.79457435383443 88.42848366391559
```

Figure 5.1.5.2.4 The real-time refreshing of all necessary landmarks' outputted angles.

These angles are rendered in the media pipe feed to check the angles as we pose without the need for tracing back to the program.

5.1.5.3 Controller

```
class Controller(object):
    # Controls the max limit of triggers and joysticks
    MAX_TRIG_VAL = math.pow(2, 8)
    MAX_JOY_VAL = math.pow(2, 8)

    # Inits everything to base position
    def __init__(self):

        self.LeftJoystickY = 0
        self.LeftJoystickX = 0
        self.RightJoystickY = 0
        self.RightJoystickX = 0
        self.LeftTrigger = 0
        self.RightTrigger = 0
        self.LeftBumper = 0
        self.RightBumper = 0
        self.A = 0
        self.X = 0
        self.Y = 0
        self.B = 0
        self.LeftThumb = 0
        self.RightThumb = 0
        self.Back = 0
        self.Start = 0
        self.LeftDPad = 0
        self.RightDPad = 0
        self.UpDPad = 0
        self.DownDPad = 0
        self.ControllerFlag = -1

        # Runs seperately from other processes
        self._monitor_thread = threading.Thread(target=self._monitor_controller, args=())
        self._monitor_thread.daemon = True
        self._monitor_thread.start()
```

Figure 5.1.5.3.1 Controller class

The controller class acts as a state machine, i.e., stores the state of all the buttons, triggers and joysticks at every point in time. The controller object that is created runs on a daemon thread and listens to button presses in the background so as to not hinder any active program running in the foreground.

```
# Return the buttons/triggers state
def read(self):
    left_x       = self.LeftJoystickX
    left_y       = self.LeftJoystickY
    left_trigg   = self.LeftTrigger
    left_bumper  = 1 if self.LeftBumper == 1 else -1
    right_x      = self.RightJoystickX
    right_y      = self.RightJoystickY
    right_trigg  = self.RightTrigger
    right_bumper = 1 if self.RightBumper == 1 else -1
    a            = self.A
    b            = self.B
    y            = self.Y
    x            = self.X

    return [left_x, left_y, left_trigg, left_bumper, right_x, right_y, right_trigg, right_bumper, a, b, x, y]
```

Figure 5.1.5.3.2 Controller read from stream

when the read method is called, the controller returns the state of its object at that point in time. The output from this stage is used to change the state of the program and the rest of the inputs are normalized and are packaged into a payload for the mqtt service.

```
calculate_payload(self, current_inputs):
    #print(current_inputs)
    shoulder_left_dxy   = str(current_inputs[0]     // 12 * -1)                # Normalize to -11 to 10
    shoulder__left_dzy  = str(current_inputs[1]     // 12)                     # Normalize to -11 to 10
    elbow_left          = str((current_inputs[2]    // 25) * -current_inputs[3]) # Normalize to -10 to 10
    shoulder_right_dxy  = str(current_inputs[4]     // 12)                     # Normalize to -11 to 10
    shoulder_right_dzy  = str(current_inputs[5]     // 12)                     # Normalize to -11 to 10
    elbow_right         = str((current_inputs[6]    // 25) * -current_inputs[7]) # Normalize to -10 to 10

    payload = "contr," + shoulder_right_dxy + ',' + shoulder_right_dzy + ',' + elbow_right + ',' + shoulder_left_dxy + ',' + shoulder__left_dzy + ',' + elbow_left
    return payload
```

Figure 5.1.5.3.3 Controller payload calculation

The inputs are normalized to range of either -11 to 10 or -10 to 10. The normalized inputs act as incrementors to the robot's motor parameters.

5.2 Hardware

The body is built in two phases, the robot assembly and then the circuit.

5.2.1 Robot Assembly

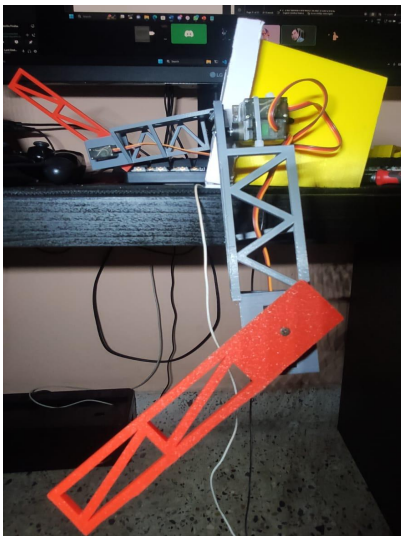


Figure 5.2.1.1 Arm Assembly

The motors at the shoulder joints are first fixed together perpendicular to their plane of motion using zip ties. The shoulder mounting solution provides two degrees of freedom along the YZ axis and the XY axis. The motor cluster is then attached to the body using a gear horn and a single 2 x 8mm screw and the front facing motor is attached to the upper arm in the same method. The elbow joint has a single motor

mount which moves in the ZY axis. The motor is mounted in the upper arm with two M2 X 10 mm screws

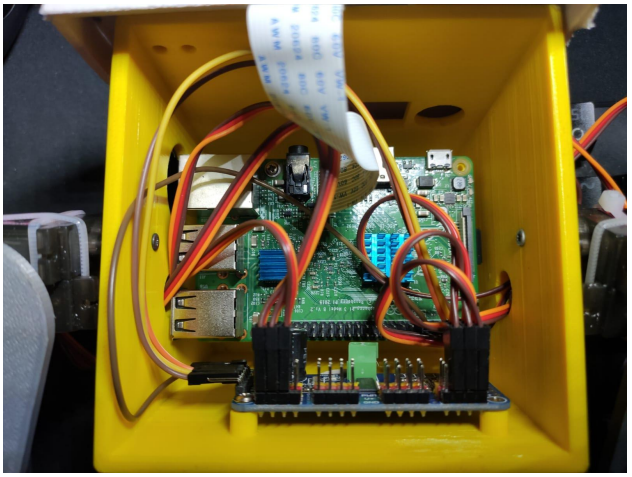


Figure 5.2.1.2 Circuit Assembly

Firstly, the raspberry pi is lowered into the body which provides housing for all the electronics. The raspberry pi sits on 4 mounting legs. The body provides access to the pi's lan port, hdmi port and the power port. A 5V pin and a GND pin is connected to the VCC and GND pin on the pca 9685 respectively. The I2C pins, mainly, SDA(System data) and SCL(System clock) on the raspberry pi are connected to the SDA and SCL on the pca9685. The motors are then inserted into the pins provided on the servo motor driver shield. The 0, 1, and 2 lines are used by the right arm and the 13, 14, 15 lines are used by the left arm in the order of shoulder ZY axis motor, XY axis motor and finally the elbow motor. Separate power lines for the motors are run through the bottom of the robot to connect to a usb socket.

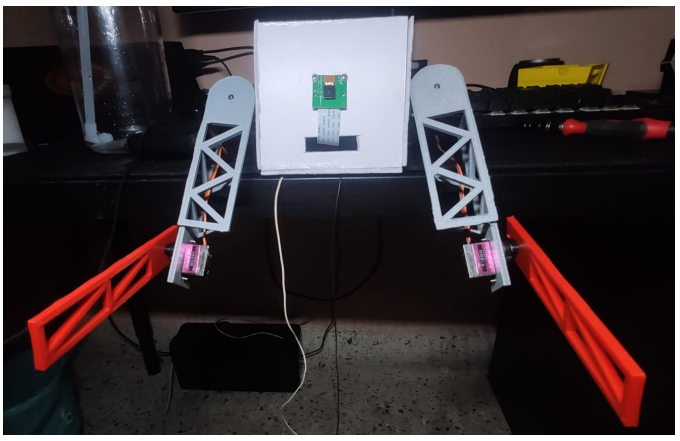


Figure 5.2.1.3 Completed Robot

CHAPTER 6
TESTING AND VALIDATION

6.1 TEST PLAN

Even when all the modules have been integrated successfully, there is still a slight margin of error when testing the program directly. This requires an extensive overview of what modules to be tested and the testing strategy as well as the environment that is needed in order to test the program.

6.1.1 SCOPE

Module Name	Description
karlos.py	Mediapipe Feed: The mediapipe feed must be tested to check whether or not it is able to establish the landmarks on the user as well as calculate the angles in real-time. OpenCV: OpenCV must be tested to see whether or not the camera is able to open or not.
RPi_Cam_Web_Interface	RPi Cam Web Interface must be tested to see if the livestream of the Raspberry Pi's mounted camera is able to capture its environment at a relatively stable frame rate and resolution.
MQTT	The sub.py or subscriber program on the pi must be tested to see if any data is being received as well as if the client computer is able to transmit the data in this case the angles calculated.

PCA9685	This microcontroller should be tested to verify if it is able to receive the angles processed by the Raspberry Pi and if it is able to communicate with the Servo Motors.
Servo Motors	The servo motors need to be tested to see if they are accurately installed to produce the coherent output desired and made by the user.
Controller	The controller acts as a guided controller and must be tested to see if the robot is able to receive the inputs from the guided controller as well as make the subsequent movements inputted by the user.

Table 6.1.1 Modules to be tested

6.1.2 TEST STRATEGIES:

- **Unit Testing:** Individual modules are tested to determine how useful it can be when executing.
- **Integration Testing:** Multiple modules are tested together to oversee how they integrate with each other.
- **System Testing:** All the modules are tested together as a final product/system to inspect any faults or errors.

6.1.3 UNIT TESTING:

1. AngleCalculator.py:

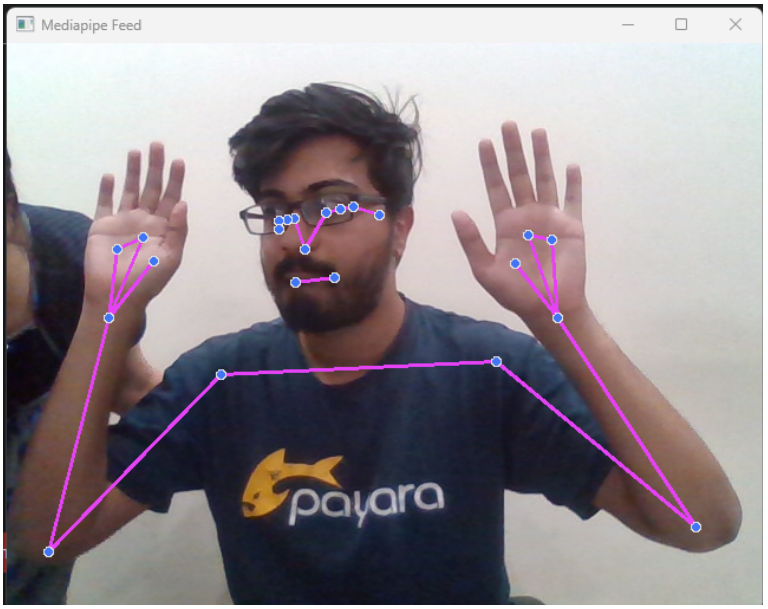


Figure 6.1.3a Unit testing of the anglecalculator.py program

From Figure 6.1.3, we can not only test the accuracy of the landmarks from the user, but also the integrity of the program to not be confused by multiple people in the camera feed. The program is able to detect the main user who is in front of the camera and produce landmarks on them as well as calculate the angles produced by the landmarks.

2. RPi_Cam_Web_Interface

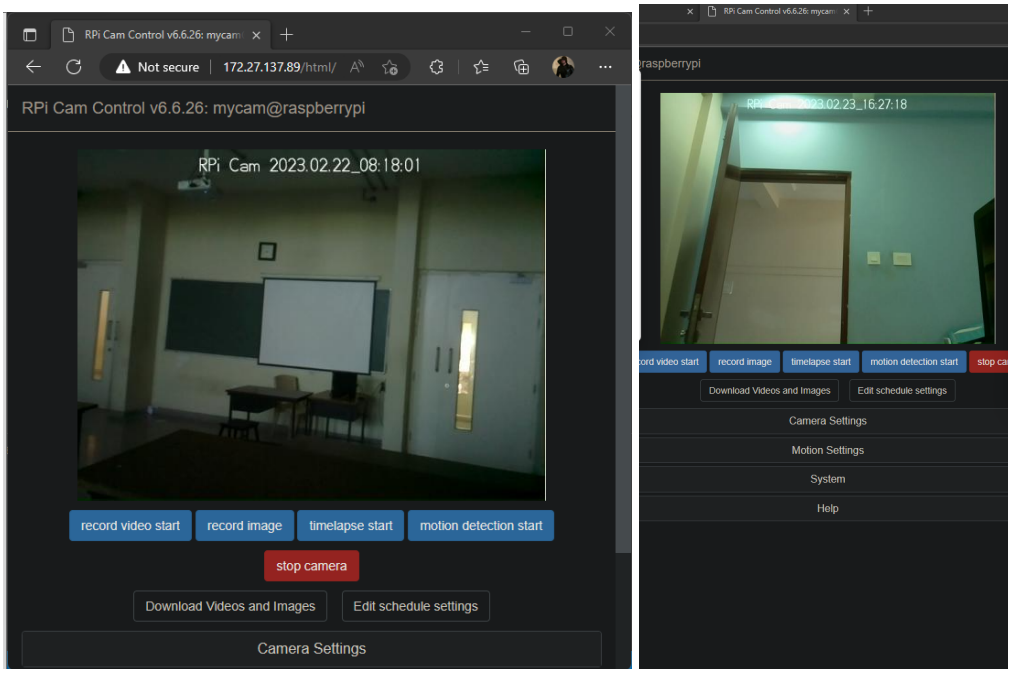


Figure 6.1.3b Unit testing of the RPi_Cam_Web_Interface at different environments

From the above figure, we can validate that the testing of the module is able to broadcast a live feed of the Raspberry Pi's environment regardless of the environment as long as it is connected to the virtual private network.

1. **MQTT**

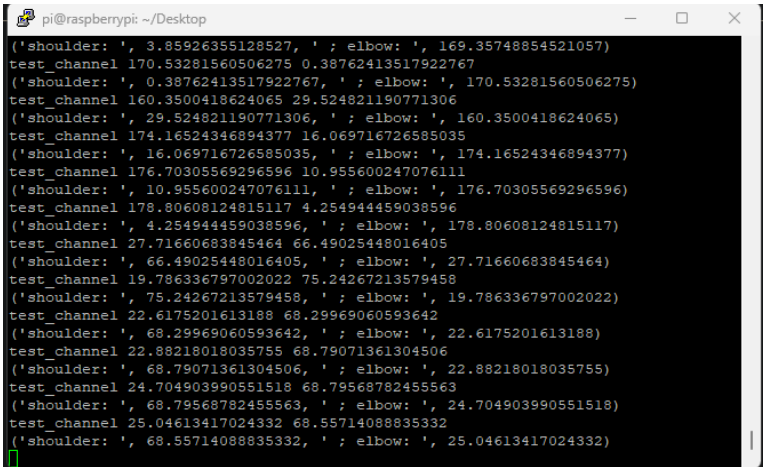


Figure 6.1.3c Unit testing of the subscriber program on the Raspberry Pi

From Figure 6.1.3c, we are able to see that the message queuing telemetry transport communication protocol is able to work and receive angles calculated by the mediapipe feed generated by the user.

2. **Servo Motor**

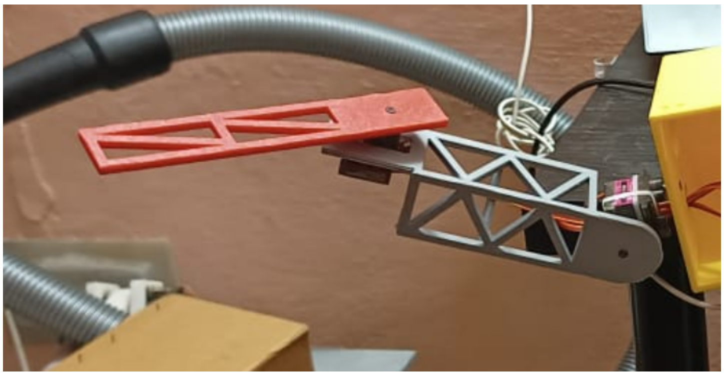


Figure 6.1.3d: Unit testing of the servo motors

From Figure 6.1.3d, we are able to test the motors to confirm that they can operate within the given power constraints as well as operate in the angle range of 0-170 degrees.

3. **Controller**

```
[0.0, 0.0, 0, 0, 0.0, 0.0, 0, 0, 0, 0, 0, 0]
[0.0, 0.0, 0, 0, 0.0, 0.0, 0, 0, 0, 0, 0, 0]
[0.0, 0.0, 0, 0, 0.0, 0.0, 0, 0, 0, 0, 0, 0]
[0.0, 0.0, 0, 0, 0.0, 0.0, 0, 0, 0, 0, 0, 0]
[0.0, 0.0, 0, 0, 0.0, 0.0, 0, 0, 0, 0, 0, 0]
[0.0, 0.0, 0, 0, 0.0, 0.0, 0, 0, 0, 0, 0, 0]
[0.0, 0.0, 0, 0, 0.0, 0.0, 0, 0, 0, 0, 0, 0]
[0.0, 0.0, 0, 0, 0.0, 0.0, 0, 0, 0, 0, 0, 0]
```

Figure 6.1.3e No input

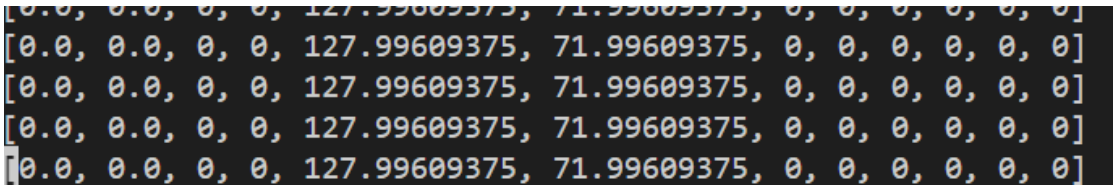


Figure 6.1.3f right thumbstick

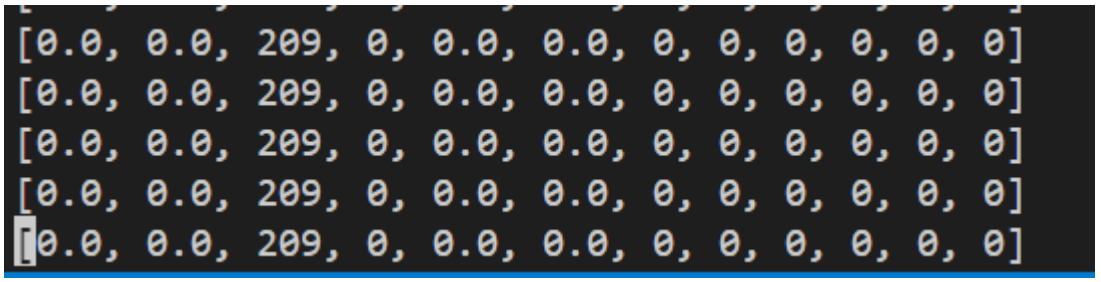


Figure 6.1.3g left trigger

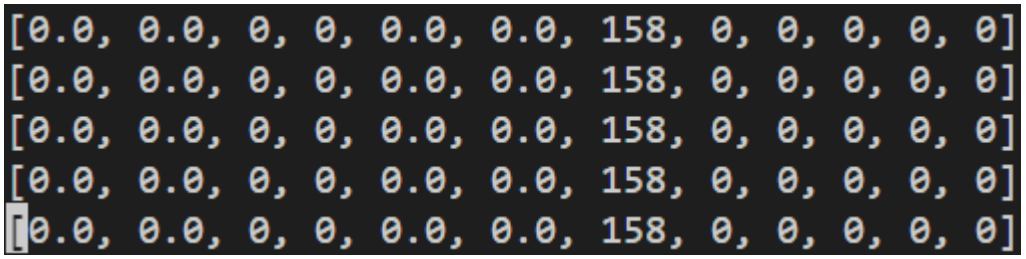


Figure 6.1.3h right trigger

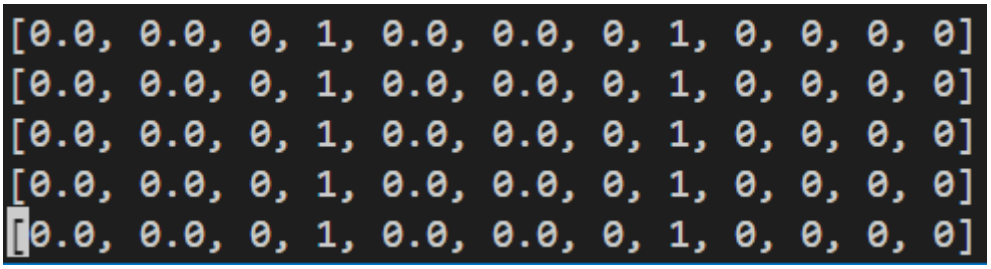


Figure 6.1.3i shoulder buttons

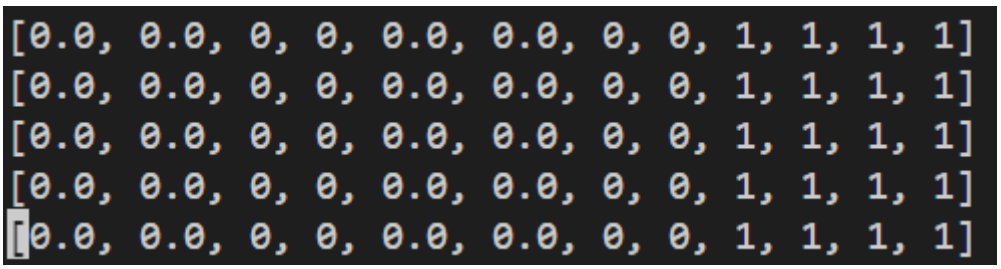


Figure 6.1.3j Face buttons

6.1.4 INTEGRATION TESTING

1. Anglecalculator.py, RPi_Cam_Web_Interface, and MQTT

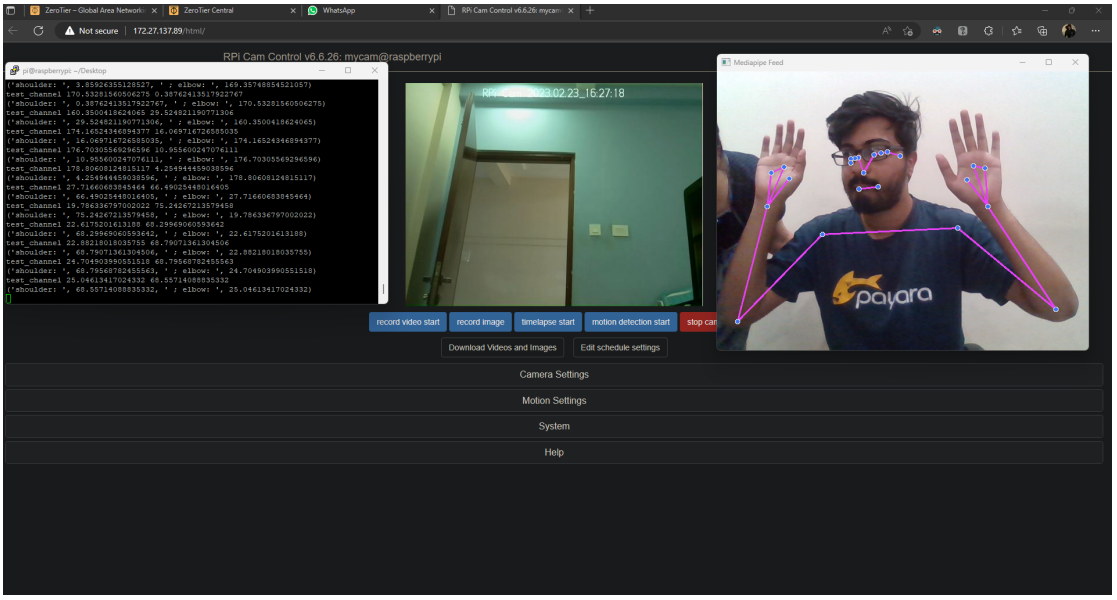


Figure 6.1.4a Integration testing of the first 3 modules

From Figure 6.1.4, we are able to see that the 3 modules, Anglecalculator.py, RPi_Cam_Web_Interface, and the subscriber program using MQTT are able to work simultaneously allowing for the user to view the Raspberry Pi's environment, and make a pose in which the angle calculator program will process and transmit and the Raspberry Pi will able to receive the calculated angles.

2. Controller and the Servo Motors

Figure 6.1.4b Integration testing of Controller and Servo Motors

6.1.5 SYSTEM TESTING:

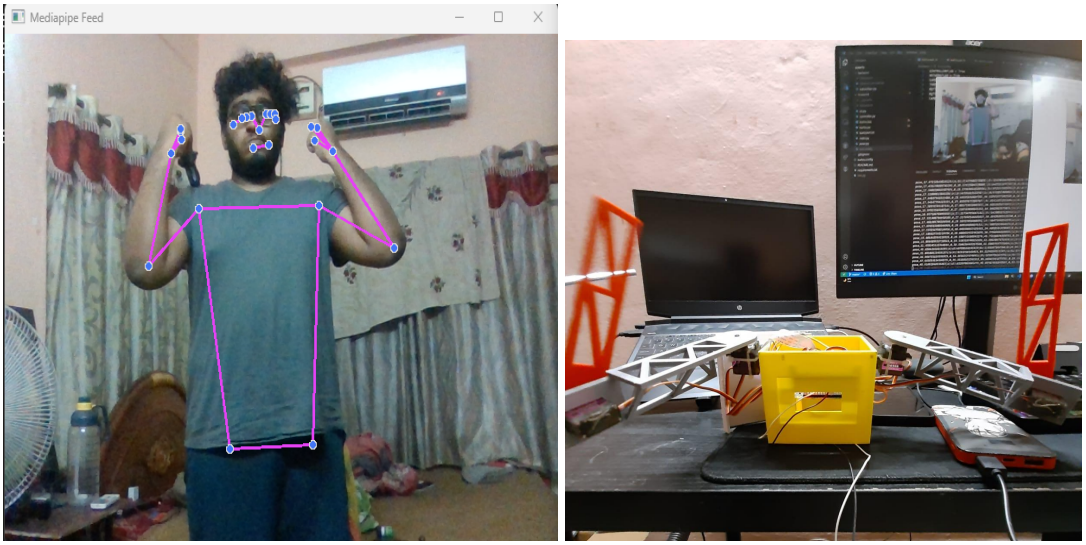


***Figure 6.1.5a*:** Male and robot arms upright

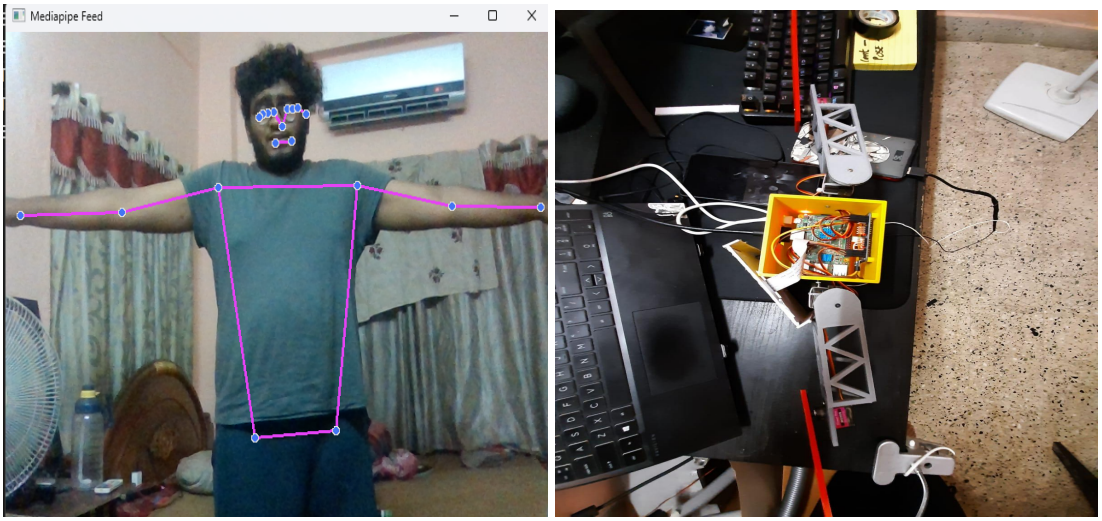
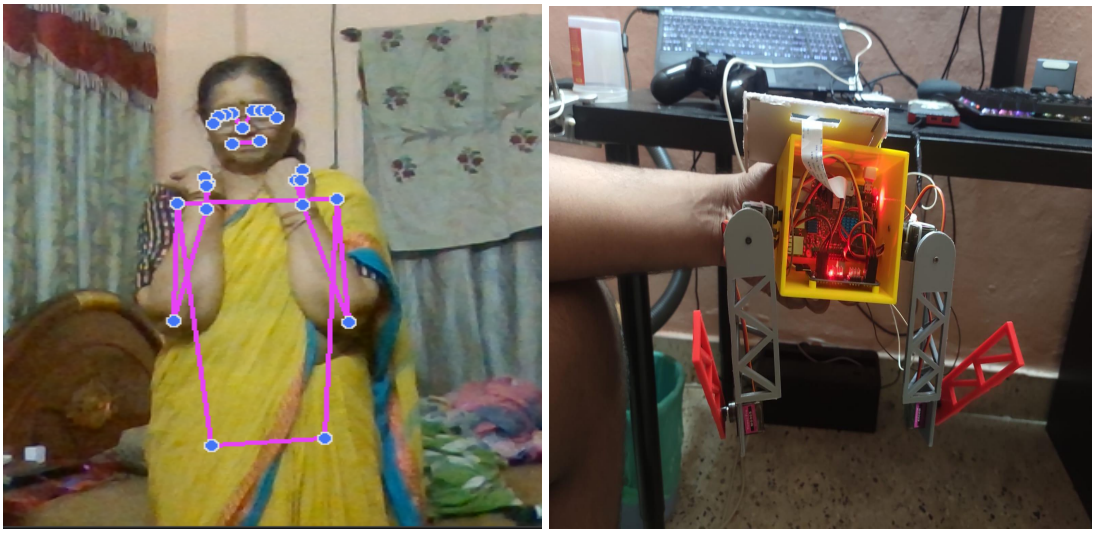


***Figure 6.1.5b*: Male and robot arms are extended**



`

***Figure 6.1.5c*: Female and robots arms are upright**

***Figure 6.1.5d*:** Female and robots arms are extended

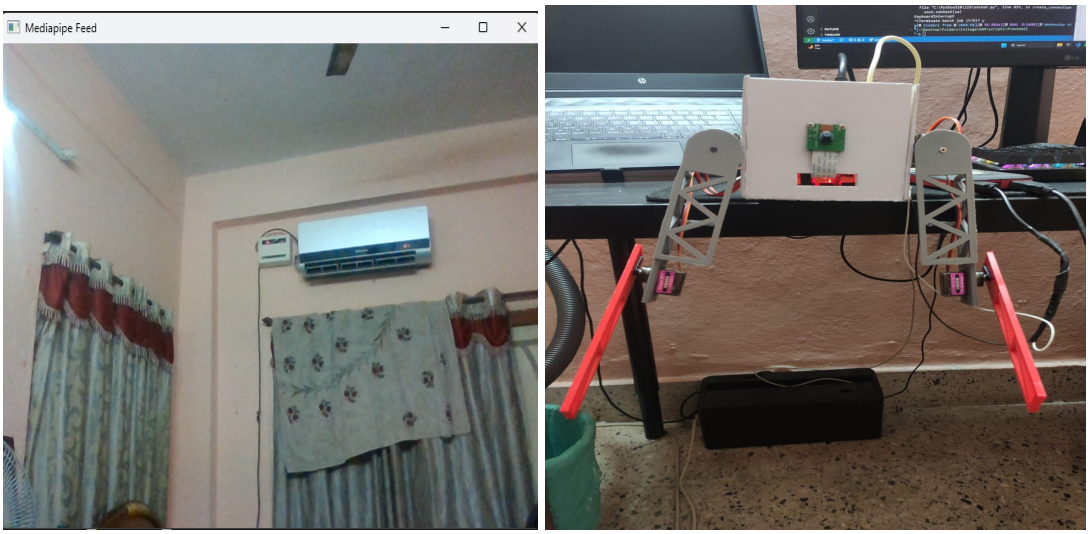


***Figure 6.1.5e*:** No objects detected and therefore robot remains in its default position

6.2 TEST CASES:

Test Case ID	Input Description	SExpected Output	Actual Output
1.	Male placing both of their arms upright	Robot should place both of its arms upright	Robot places both of its arms upright slightly crooked
2.	Male extending both of their arms	Robot should extend both of its arms	Robot extends both of its arms
3.	Female placing both of their arms upright	Robot should place both of its arms upright	Robot places both of its arms upright slightly crooked
4.	Female extending their arm	Robot should extend both of their arms	Robot extends both of its arms
5.	No objects place in front of the camera	Robot should not make any movements	Robot did not make any movements
6.	Controller gives robot certain input	Robot makes the subsequent output	Robot makes the subsequent output

Table 6.2.1 Test cases

6.3 TEST CONCLUSION

All of the given test cases have been tested to see whether or not any errors, issues, or bugs reveal themselves but the overall function is not affected. The individual modules are able to properly function and there is no problem when integrating multiple modules. In the overall system test, the robot is able to perform the desired action based upon user input. Improvements can be made to optimize the time taken to connect, detect, and execute but at a fundamental level, the end result is the same.

Its also noted that on an average, The controller takes 4 seconds to complete a full rotation of the arm and the pose estimation mode takes 2.12 seconds to complete a full rotation of the arm. The average improvement over the controller results in a 47% decrease in the time taken to move the arm back and forth over an axis.

CHAPTER 7

RESULT ANALYSIS

We can agree, on the basis of the test cases that have been provided, that the humanoid robot is capable of striking poses in response to user input. We have shown that a motion-based controller, as opposed to a conventional guided controller, is superior in terms of speed and achieves considerable accuracy. The guided controller that is being used in this project is able to make motions in response to the inputs that are being provided by the user.

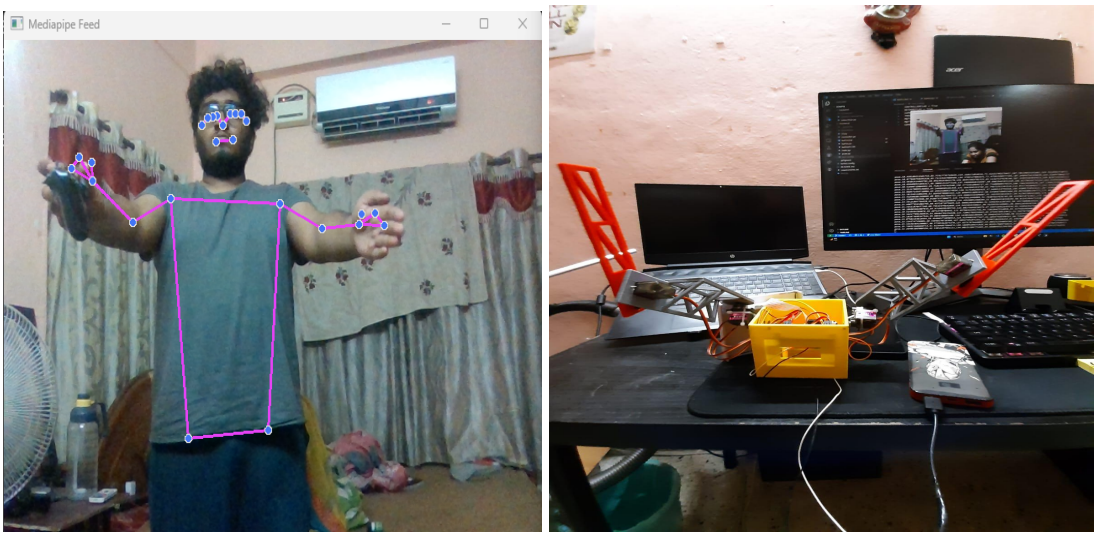


Figure 7.1: User giving an input and the robot produces the desired output using a motion based controller

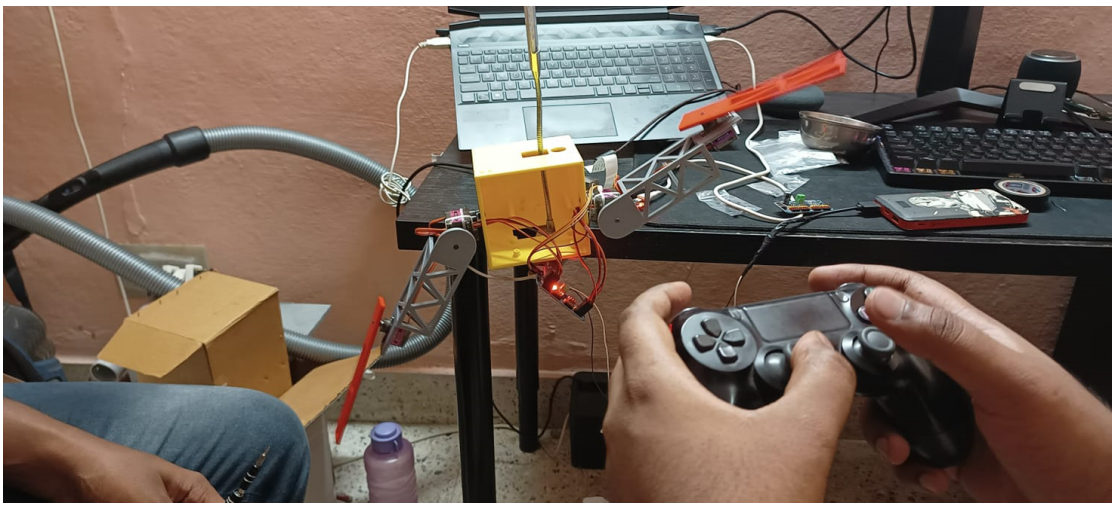


Figure 7.2: User giving an input via controller and robot producing desired output

As you can see from the figures above, the humanoid robot is able to produce a subsequent output given from a user. We are also able to view a stable live feed from the Raspberry Pi which represents what the humanoid robot is seeing in real time. Using SSH, we are able to see visually the data being received from the user via the subscriber program. We are able to see that traditional guided controllers require more time to produce the same output and increase the complexity in the movements compared to a motion based controller but the output is the same nevertheless.

Table 7.1 evaluates the model used in this project with other commonly used pose estimation models with a dataset called 'Yoga'.

Method	Yoga mAP	Yoga PCK@0.2
BlazePose GHUM Heavy	68.1	96.4
BlazePose GHUM Full	62.6	95.5
BlazePose GHUM Lite	45.0	90.2
AlphaPose ResNet50	63.4	96.0

Table 7.1: mAP and PCK score comparison on dataset 'Yoga'

CHAPTER 8

CONCLUSION

This project has aimed to provide a solution for the problem and has successfully accomplished its task while maintaining a relative degree of simplicity as well as establishing the groundwork to build an improved system for future iterations. We have been able to successfully integrate a fully operational robot based off of a motion based controller. We have demonstrated the ease of use of a motion based controller compared to a traditional guided controller as well as the complexities it reduces. This project has established a virtual private network as to which permits the robot to be accessed via the Internet which ultimately increases the applicability of the overall project. This project makes use of common electronics laboratory equipment instead of complex microcontrollers resulting in an increase in practicality and the potential of improved future iterations. This project has executed the tasks which normally require a human to do and while it may be limited in its functionality at the moment, it holds the potential needed to advance the utilization of robots to assist humans. This project is a proof of concept that guided controllers are no longer needed when operating complex advanced robots.The project attempted to solve an issue and has achieved its objective with a certain degree of simplicity, while also establishing the basis for future iterations to expand upon. By automating tasks that typically require human involvement, the project may currently be limited in functionality, but it possesses the potential to advance the use of robots in assisting humans. In the end, the purpose of this project is to serve as a proof of concept that demonstrates the elimination of the need for guided controllers when it comes to the operation of sophisticated robots.

8.1 FUTURE SCOPE:

The project has a considerable potential for future development. Some of the key area of interests that extend the functionality of the robot are:

1. Increasing the AP score of the pose estimation model to achieve better precision.
2. Adding support for modular end effectors at the wrists of both arms.
3. Improving the mechanism of the arms to include more degrees of freedom.
4. Adding mobility for the robot body and a way to control its actions using a similar system.
5. Developing sensors and mechanical structures that attach to the limbs to capture motion.
6. Improving and optimizing power delivery to the servo motors.
7. Adding force feedback from the robot to the operator.
8. Increasing security and adding encryption to the robot's networking.

REFERENCES:

[1] Kamalakannan M a.w.t, “Controlling the Speed of Conveyor Belt using Python – Raspberry Pi 3B+”. Orient.J. Comp. Sci. and Technol, 2019.

[2] Baiwan Liana a.w.t, “Design of 4-DOF manipulator based on Arduino”. Academic Journal of Engineering and Technology Science, 2022.

[3] Cobos Brione a.w.t, “Design and Implementation of a Docker-based Virtual Network in a Software-Defined Networking (SDN) environment using ZeroTier and Raspberry Pi,” Juan Carlos 2021.

[4] U. Hunkeler a.w.t, "MQTT-S — A publish/subscribe protocol for Wireless Sensor Networks," IEEE, 2008

[5] K. Kruthika a.w.t, “Design and development of a robotic arm,” IEEE, 2017.

[6] A. Basu a.w.t, “Mechanical optimization of servo motor,” Journal of Mechanical Design, 2005.

[7] A. Rahman a.w.t, "IoT Based Patient Monitoring System Using ECG Sensor," IEEE, 2019.

[8] Mohamad Khairi a.w.t, “Design and Implementation of Robot Assisted Surgery Based on Internet of Things (IoT)” IEEE, 2018.

[9] Camillo Lugaresi a.w.t MediaPipe_CVPR_CV4ARVR_Workshop_2019.pdf, Google Research (2020)

[10] Ivan Culjak a.w.t, “A brief introduction to OpenCV”, IEEE, 2012.

[11] Bo Pang a.w.t, “Deep Learning With TensorFlow: A Review”, Journal of Educational and Behavioral Statistics, 2019.

[12] João Felipe Pimentel a.w.t, “A Large-Scale Study About Quality and Reproducibility of Jupyter Notebooks”, IEEE, 2019.

[13] Valentin Bazarevsky a.w.t. “BlazePose: On-device Real-time Body Pose tracking”, Google Research, 2020.

[14] Mediapipe for Pose, https://google.github.io/mediapipe/solutions/pose, 2020.