

DESIGN AND FABRICATION OF AUTONOMOUS VOICE NAVIGATED 6-DOF MANIPULATOR

A PROJECT REPORT

Submitted by

**SURENDHAR M
BARANI KUMAR S**

**210420115052
210420115004**

in partial fulfillment for the award of the degree

of

BACHELOR OF ENGINEERING

in

MECHATRONICS ENGINEERING



**CHENNAI INSTITUTE OF TECHNOLOGY (AUTONOMOUS)
(Affiliated to Anna University, Chennai)
CHENNAI-600 069**



**ANNA UNIVERSITY: CHENNAI-600 025
APRIL-2023**

ANNA UNIVERSITY: CHENNAI-600 025

BONAFIDE CERTIFICATE

Certified that this project report “**DESIGN AND FABRICATION OF AUTONOMOUS VOICE NAVIGATED 6-DOF MANIPULATOR**” is the bonafide work of **SURENDHAR M (210420115052)**, **BARANI KUMAR S (210420115004)** who carried out the project work under my supervision.

SIGNATURE

Mr.P.SHANMUGASELVAM M.E(Ph.D)
HEAD OF THE DEPARTMENT
Assistant Professor
Department of Mechatronics
Engineering,
Chennai Institute of Technology,
Kundrathur,
Chennai-600069.

SIGNATURE

Mr.P.VINOTH KUMAR M.E
SUPERVISOR
Assistant Professor
Department of Mechatronics
Engineering,
Chennai Institute of Technology,
Kundrathur,
Chennai-600069.

Certified that the above students have attend of viva voice during the exam held on.....

INTERNAL EXAMINER

EXTERNAL EXAMINER

ACKNOWLEDGEMENT

We express our gratitude to our Chairman **Shri.P.SRIRAM** and all trust members of Chennai institute of technology for providing the facility and opportunity to do this project as a part of our undergraduate course.

We are grateful to our Principal **Dr.A.RAMESH M.E, Ph.D.** for providing us the facility and encouragement during the course of our work.

We sincerely thank our Head of the Department, **Mr.P.SHANMUGASELVAM M.E (Ph.D)** Department of Mechatronics Engineering for having provided us valuable guidance, resources and timely suggestions throughout our work.

We sincerely thank our Project Guide, **Mr.P.VINOTH KUMAR M.E** Professor, Department of Mechatronics Engineering for having provided us valuable guidance, resources and timely suggestions throughout our work.

We would like to extend our thanks to our **Faculty coordinators of the Department of Mechatronics Engineering**, for their valuable suggestions throughout this project.

We wish to extend our sincere thanks to all **Faculty members of the Department of Mechatronics Engineering** for their valuable suggestions and their kind cooperation for the successful completion of our project.

We wish to acknowledge the help received from the **Lab Instructors of the Department of Mechatronics Engineering** and others for providing valuable suggestions and for the successful completion of the project.

ABSTRACT

A 6-dof manipulator-based autonomous voice-controlled robot is the goal of this project. By recognizing voice commands from the user, the robot can perform a variety of tasks such as picking and placing objects, opening doors, and turning switches. The main controller for the robot is a Raspberry Pi 4, and the 6-dof manipulator is a Dynamixel XM430-W350-R servo motor. The robot likewise has coordinated abilities with Amazon® Alexa with the end goal of route and way arranging. The project shows that using voice as a natural and easy-to-use interface for human-robot interaction is possible and possible. The inverse kinematics controller receives input signals from the voice recognition module using a deep learning model. The opposite kinematics regulator utilizes another profound learning model to take care of the backwards kinematics issue of the 6-DOF controller and create joint plots for the ideal end-effector position and direction. The framework can be applied to different areas like modern advanced mechanics, administration advanced mechanics, and space mechanical technology. The task means to show the practicality and viability of involving man-made consciousness calculations for voice-based control of repetitive controllers. The project also aims to investigate the difficulties and limitations of such a system, such as the robustness of the system, human-robot interaction, voice recognition accuracy, and network training time. This project is for people who don't want to learn the fundamentals of robotics or come from a different field, like medicine, business, or something else. This not only enables experts from outside the field to concentrate on their research and applications, but it also enables them to program robots by simply using voice commands. Voice commands can be used to simplify complex robot programming. This also ensures a person's safety by preventing them from standing near the manipulator while programming it. With the integration of Amazon Alexa, wireless programming of the manipulator is also possible.

TABLE OF CONTENTS

CHAPTER NO	TITLE	PAGE NO
	ACKNOWLEDGEMENT	iii
	ABSTRACT	iv
	TABLE OF CONTENTS	v
1	INTRODUCTION	1
	1.1 Manipulators	1
	1.2 Types of Manipulators	2
	1.3 Applications of Robots	3
	1.4 Raspberry PI	4
2	LITERATURE REVIEW	5
	2.1 Intro	5
	2.2 Voice Navigation for Bots	5
	2.3 Navigation for Manipulators	5
	2.4 Challenges	6
	2.5 Security	6
	2.6 Integration Complexity	6
	2.7 Context Awareness	7
	2.8 Complex Commands	7
	2.9 Robot Integration	7

	2.10 Kinematics Complexity	8
	2.11 Limited Vocabulary	8
	2.12 Non-Linear Equations	8
	2.13 Multiple Solutions	8
	2.14 Joint Limits	9
	2.15 Conclusion	9
3	MANIPULATOR SYSTEMS	10
	3.1 Manipulator Introduction	10
	3.2 Manipulator Joint Types	11
	3.2.1 Revolute Joints	11
	3.2.2 Cylindrical Joints	12
	3.2.3 Prismatic Joints	13
	3.2.4 Planar Joints	14
4	THE MICROCONTROLLER – RPI	15
	4.1 CPU Specifications	15
	4.2 The Master-Slave Configuration	16
	4.3 The Voice Module	17
5	RESOLVING JOINT KINEMATICS	18
	5.1 The Inverse Kinematics Approach	18
	5.2 Numerical Inverse Kinematic Solutions	19
	5.3 Analytical Inverse Kinematic Solutions	20

6	DEVANIT HARTENBERG SOLUTION	21
	6.1 Denavit Hartenberg Analysis	21
	6.2 The Denavit Hartenberg Parameters	23
	6.3 Denavit Hartenberg Matrix	25
	6.4 Kinematics in Denavit Hartenberg	26
7	DYNAMICS OF MANIPULATOR	28
	7.1 Dynamical Analysis	28
	7.2 Modified D-H Parameters	29
8	DESIGN AND MODELLING	30
	8.1 Modelling of the Manipulator Arm	30
9	CIRCUIT CONSTRUCTION AND CONFIGURATION	33
	9.1 The Master Slave Configuration	33
	9.2 Arduino I2C Program	34
	9.3 I2C Data Monitoring	34
	9.4 I2C Protocol	35
	9.5 Voice Module Integration	37
10	ANALYSIS	39
	10.1 Voice Recognition Analysis	39
	10.2 Mic Array Visualization and Test	40
	10.3 Arduino Manipulator System Design	41

	10.4 Circuit Mapping of Servo	43
	10.5 Manipulator Torque Analysis	43
	10.6 Joint Torque Calculation	45
	10.7 Analysis Results	48
11	MATERIALS & COMPONENTS USED	51
	11.1 Arduino UNO R3	51
	11.2 Raspberry PI Model 3B+	52
	11.3 MAX4466 Mic Module	52
	11.4 Buck Converter	53
	11.5 MG995 Servo Motors	54
	11.6 12v SMPS	55
	11.7 Jumper Cables	56
12	CODE SNIPPETS	57
	12.1 Program to Check Mic Connection	57
	12.2.1 Mic Array Visualization Test	57
	12.2.2 Creating PyAudio Streaming Object	58
	12.2.3 Grabbing Data from Buffer	58
	12.2.4 Functions of Plotting Data	59
	12.2.5 Main Loop Program	60
	12.3 Main Driver Program (Master Side)	61
	12.4 PID Program (Slave Side)	62

13	CONCLUSION	66
14	REFERENCES	67

CHAPTER -1

INTRODUCTION

1.1 Manipulators

A robotic manipulator arm is a reprogrammable and multifunctional mechanical device responsible for moving materials, parts, objects, or tools through programmed motions in order to perform various tasks. These mechanical devices are composed of a series of jointed segments that form an arm-like manipulator. A robotic manipulator is capable of moving or handling objects automatically depending upon its given number of degrees of freedom. Those degrees of freedom are also known as axes. Each axis of a robotic manipulator correlates to the number of motors within the robot. Robotic manipulators can range from two axes to ten or more. Most robots used in industrial settings have between four to six axes. Six-axis robotic manipulators are the most common since their range of motion is similar to the human arm. This provides the flexibility needed to automate many industrial processes with robots.

The general construction of a robotic manipulator consists of rigid links that are connected by joints. One end of the manipulator is fixed to a base while the other end is free and used for performing different robotic applications. The structure of the robotic manipulator will determine the reach of its end-effector and its work envelope.

Robotic manipulators are typically divided into two parts; the arm/body and the wrist. The arm and body of the manipulator control the movement of objects within the robot's work envelope. For instance, when a FANUC M710ic/50 moves a part onto a conveyor. While the wrist controls the movement of the end-effector,

allowing the manipulator to carry out the task it has been programmed for, such as the FANUC M-20ia picking up a part with its gripper.

1.2 Types of Manipulators



Fig 1.1 – Types of Manipulators

There are several types of robotic manipulators. These types vary based upon the combination of joints and include cartesian, cylindrical, polar, articulated, SCARA, and delta configurations. Articulated manipulators are the most widely used and recognized type in manufacturing. They feature a revolute shoulder joint that allows for rotation. The FANUC Lr Mate 200id and the ABB IRB 2600 are two examples of articulated robots. Cartesian manipulators consist of prismatic or sliding joints, providing a rectangular work envelope. Cylindrical and polar manipulators both consist of revolute joints. The work envelope of cylindrical robots is cylindrical, while polar robots operate within a spherical envelope. SCARA and delta manipulators both utilize parallel joint configurations.

1.3 Basic Applications of Manipulators



Fig 1.2 – Applications of Manipulators

All robotic manipulators include a controller and teach pendant. A teach pendant is used to program the robotic manipulator while the controller functions as the “brain,” allowing the robot to interpret and carry out the desired operations of the program. How the manipulator operates depends upon its payload capacity, speed, and repeatability.

Robot manipulators are capable of automating a number of different types of applications. Some of the most common include automated welding, robotic assembly, material removal, material handling, painting, robotic palletizing, and automated pick and place. Advancements in technology has greatly improved the accuracy and precision of robotic manipulators, allowing for the automation of new applications such as robotic 3D printing. As manipulators become more sophisticated so does the scope of robotic applications. Manufacturing processes become more efficient, reliable, and productive with industrial robot automation

1.4 Raspberry PI

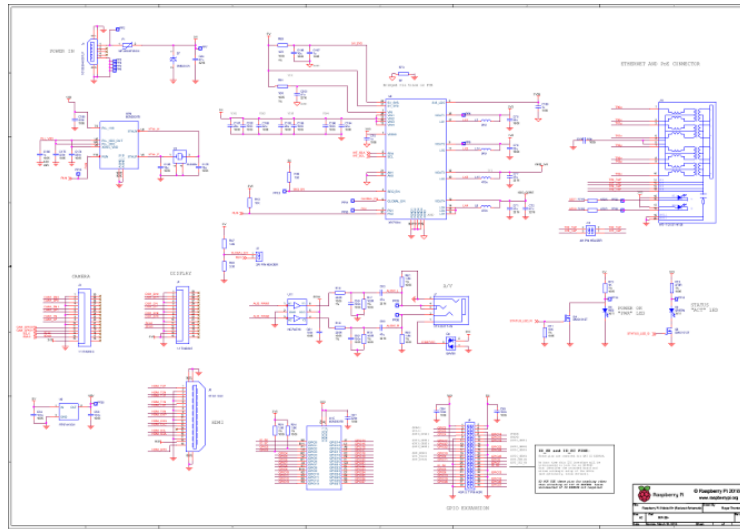


Fig 1.3 – Schematics of Raspberry PI 4 Microcontroller

The Raspberry Pi is a credit card-sized computer. The Raspberry Pi 3 Model B is the third generation Raspberry Pi. It is based on the BCM2837 system-on-chip (SoC), which includes a 1.2 GHz quad-core ARMv8 64bit processor and a powerful VideoCore IV GPU. The Raspberry Pi can run a full range of ARM GNU/Linux distributions, including Snappy Ubuntu Core, Debian, Fedora, and Arch Linux, as well as Microsoft Windows 10 IoT Core.

The Raspberry Pi 3 Model B is the first Raspberry Pi to feature onboard wireless and Bluetooth connectivity.

The Raspberry Pi was designed by the Raspberry Pi Foundation to provide an affordable platform for experimentation and education in computer programming. The Raspberry Pi can be used for many of the things that a normal desktop PC does, including word-processing, spreadsheets, high-definition video, games, and programming. USB devices such as keyboards and mice can be connected via the board's four USB ports

CHAPTER – 2

LITERATURE REVIEW

2.1 Introduction

Voice navigation is an emerging field in robotics, where robots can be controlled through human voice commands. With the rise of voice assistants like Alexa and Siri, voice navigation is becoming an increasingly popular means of controlling robots. This literature review aims to explore the current state of research on voice navigated robots.

2.2 Voice Navigation for Mobile Robots

One area of research in voice navigation is focused on mobile robots. Researchers have developed voice control systems for autonomous mobile robots that enable them to perform various tasks. For example, a study conducted by Li et al. (2020) proposed a voice control system for mobile robots that enables them to navigate, recognize objects, and avoid obstacles. Similarly, a study by Sankaranarayanan et al. (2019) proposed a voice navigation system for a wheelchair that allows users to control the wheelchair using voice commands.

2.3 Voice Navigation for Industrial Robots

Voice navigation has also been studied in the context of industrial robots. Researchers have proposed voice control systems for industrial robots to improve worker safety and productivity. For instance, a study by Paredes et al. (2019) proposed a voice navigation system for an industrial robot that enables workers to control the robot without having to use a remote control, which can be cumbersome and time-consuming.

2.4 Challenges in Voice Navigation

Despite the potential benefits of voice navigation, there are several challenges that need to be addressed. One of the main challenges is the robustness of the voice recognition system. Voice recognition systems can be affected by background noise, accents, and variations in pronunciation. Another challenge is the need for natural language processing algorithms that can understand complex commands and respond appropriately.

2.5 Security

One of the main challenges in integrating Alexa with an industrial robot is security. Industrial robots are often used in critical applications, and any security breach can have serious consequences. Integrating Alexa with an industrial robot requires connecting the robot to the internet, which can create potential security vulnerabilities. To address this challenge, researchers have proposed various security measures, such as using encryption and authentication protocols.

2.6 Integration Complexity

Integrating Alexa with an industrial robot requires a high level of technical expertise, and the process can be complex. The integration process involves connecting the robot to the internet, setting up the Alexa Skills Kit, and developing custom Alexa skills. To address this challenge, researchers have proposed simplified integration processes that use pre-built modules and templates. Another challenge in integrating Alexa with an industrial robot is ensuring the accuracy of voice recognition. Industrial robots are often used in noisy environments, and background noise can affect the accuracy of voice recognition. To address this

challenge, researchers have proposed using noise-canceling microphones and improving the accuracy of voice recognition algorithms.

2.7 Context Awareness

Industrial robots often perform complex tasks that require context awareness. For example, a robot may need to understand the context of a task before executing it. Integrating Alexa with an industrial robot requires developing custom skills that can understand the context of a task and respond appropriately. To address this challenge, researchers have proposed developing natural language processing algorithms that can understand complex commands and respond appropriately.

2.8 Complex Commands

Voice-controlled robots need to be able to understand and execute complex commands. For example, a robot may need to understand the context of a task before executing it. Resolving complex commands into kinematics points requires the development of natural language processing algorithms that can understand complex commands and respond appropriately.

2.9 Robot Integration Complexity

Integrating voice control with the robot's kinematics requires a high level of technical expertise, and the process can be complex. The integration process involves connecting the voice control system to the robot's kinematics system and developing custom software that translates voice commands into kinematics points. To address this challenge, researchers have proposed simplified integration processes that use pre-built modules and templates.

2.10 Kinematics System Complexity

The robot's kinematics system can be complex and vary depending on the robot's design and function. Resolving voice signals into kinematics points requires a deep understanding of the robot's kinematics system, which can be challenging. To address this challenge, researchers have proposed using modular kinematics systems that are more adaptable to different robots and tasks.

2.11 Limited Vocabulary

The vocabulary used in programming a robot via voice control can be limited, which can be a challenge when dealing with complex tasks. To address this challenge, researchers have proposed developing custom vocabularies for specific tasks or using machine learning algorithms to learn new commands over time.

2.12 Non-Linear Equations

The equations involved in resolving inverse kinematics for a robot arm are non-linear, which can make the process challenging. The non-linearity arises due to the trigonometric functions used in the equations. Solving these equations requires the use of numerical methods, which can be time-consuming and computationally expensive.

2.13 Multiple Solutions

There can be multiple solutions to the inverse kinematics problem, which can lead to challenges in determining the correct solution. In some cases, multiple solutions can lead to singularities, which can make the arm unstable. To address this

challenge, researchers have proposed using optimization algorithms to find the best solution.

2.14 Joint Limits

Robot arms have joint limits that restrict the range of motion of each joint. Resolving inverse kinematics for a robot arm requires ensuring that the joint angles are within the joint limits. This can be challenging, particularly when the desired end-effector position and orientation is outside the reachable workspace of the arm.

2.15 Conclusion

Voice navigation is an emerging field in robotics that has the potential to revolutionize the way robots are controlled. The current research in this field has focused on developing voice control systems for mobile and industrial robots. However, there are several challenges that need to be addressed, such as improving the robustness of voice recognition systems and developing natural language processing algorithms that can understand complex commands. With further research and development, voice navigation has the potential to become a widely adopted means of controlling robots. Resolving voice signals into kinematics points is a complex process that requires addressing several challenges, such as noise and ambiguity, complex commands, integration complexity, and kinematics system complexity. With further research and development, these challenges can be addressed, and voice-controlled robots can become a more widely adopted means of controlling robots. Improved voice recognition algorithms, natural language processing algorithms, and simplified integration processes can make voice control more accessible and reliable for a broader range of robotic applications.

CHAPTER – 3

MANIPULATOR SYSTEMS

3.1 Manipulator Introduction

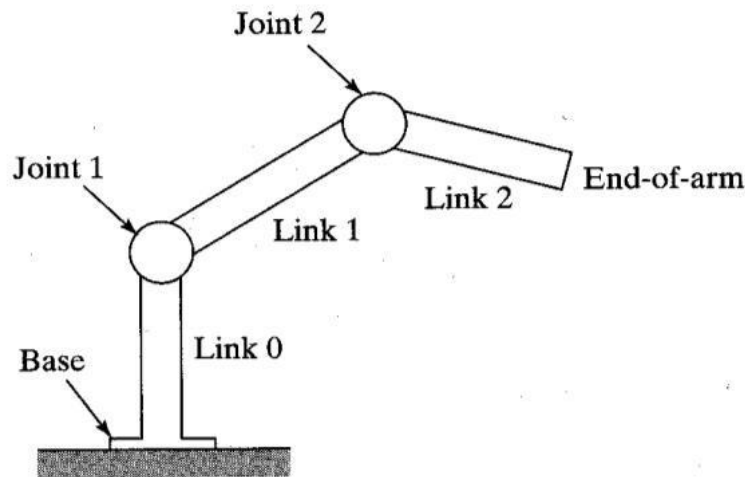


Fig 3.1 Standard 6-DOF Manipulator

Industrial robots are robotic arms that can move in several directions and can be programmed to carry out many different types of tasks in different environments. These industrial robots can work:

- ❖ In varying environments like high-pressure and vacuum chambers.
- ❖ In clean workrooms and in terribly dirty areas.
- ❖ In dangerous areas where threats of explosions, infections, radiation, or other extreme hazards are present and dangerous to humans.

Robotic arms can be equipped with specialized grippers to work with delicate and fragile objects, while other robotic arms can have grippers that can grasp and lift loads weighing several tons.

3.2 Manipulator Joint Types

When it comes to the mechanical joints featured in robotic arms there are five principal types that you need to consider. Two of the joints are linear which means the relative motion between the adjacent links is translational. On the other hand, the other three are rotary which means the relative motion of the links involves rotations between them.

3.2.1 Revolute Joints

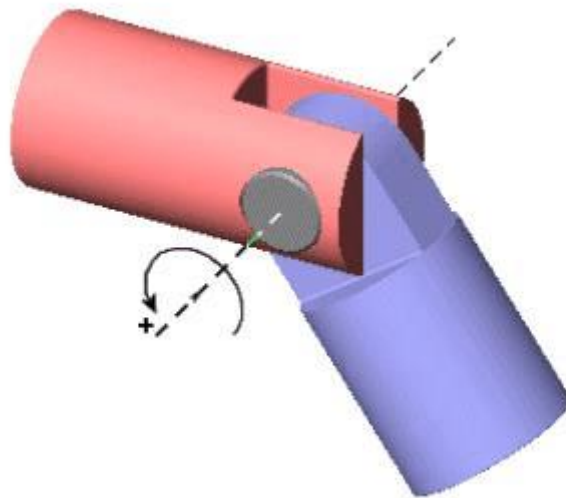


Fig 3.2 – Revolute Joint

A revolute joint (also called pin joint or hinge joint) is a one-degree-of-freedom kinematic pair used frequently in mechanisms and machines. The joint constrains the motion of two bodies to pure rotation along a common axis. The joint does not allow translation, or sliding linear motion, a constraint not shown in the diagram. Almost all assemblies of multiple moving bodies include revolute joints in their designs. Revolute joints are used in numerous applications such as door hinges, mechanisms, and other uni-axial rotation devices.

A revolute joint is usually made by a pin or knuckle joint, through a rotary bearing. It enforces a cylindrical contact area, which makes it a lower kinematic pair, also called a full joint. However, If there is any clearance between the pin and hole (as there must be for motion), so-called surface contact in the pin joint actually becomes line contact.

The contact between the inner and outer cylindrical surfaces is usually assumed to be frictionless. But some use simplified models assume linear viscous damping in the form $T = B\omega$, where T is the friction torque, ω is the relative angular velocity, and B is the friction constant. Some more complex models take stiction and stribeck effect into consideration.

3.2.2 Cylindrical Joints

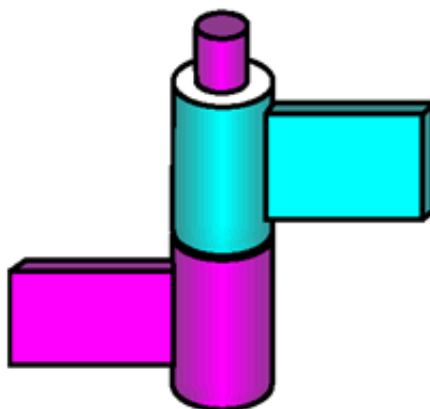


Fig 3.3 – Cylindrical Joint

A cylindrical joint is a two-degrees-of-freedom kinematic pair used in mechanisms. Cylindrical joints constrain two bodies to a single axis while allowing them to rotate about and slide along that axis. This can be pictured by an unsecured axle mounted

on a chassis, as it may freely rotate and translate. An example of this would be the rotating rods of a table football.

3.2.3 Prismatic Joints

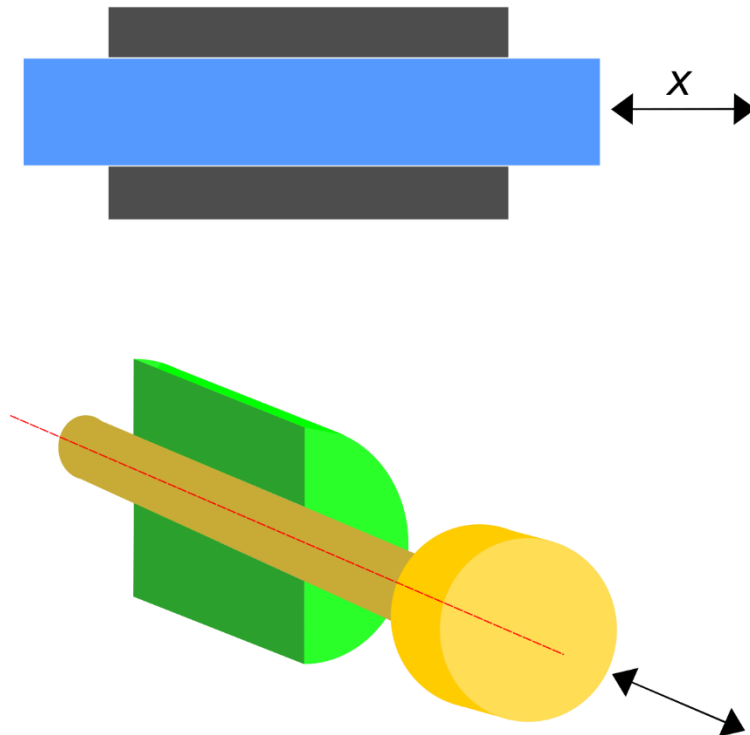


Fig 3.4 – Prismatic Joint

A prismatic joint is a one-degree-of-freedom kinematic pair which constrains the motion of two bodies to sliding along a common axis, without rotation; for this reason, it is often called a slider (as in the slider-crank linkage) or a sliding pair. They are often utilized in hydraulic and pneumatic cylinders.

A prismatic joint can be formed with a polygonal cross-section to resist rotation. Examples of this include the dovetail joint and linear bearings.

3.2.4 Planar Joints

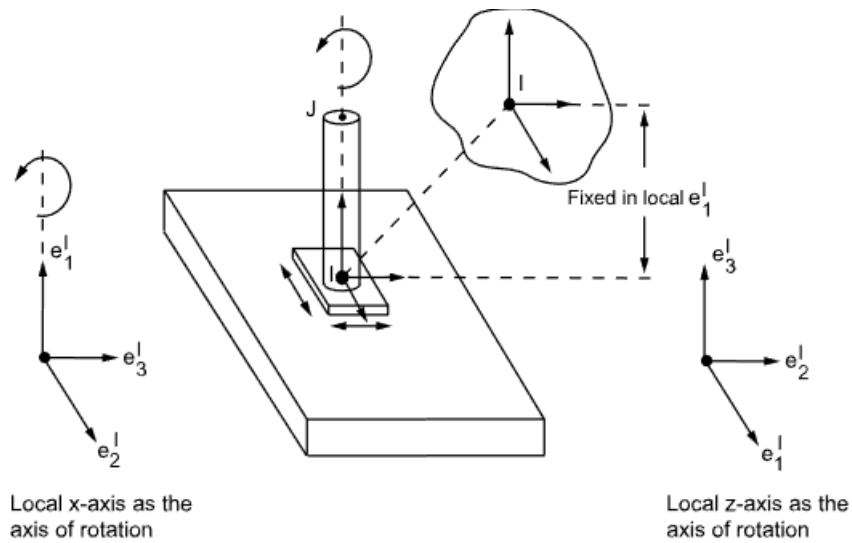


Fig 3.5 – Planar Joints

A planar joint is a three degree-of-freedom constraint. It constrains a plane on one body (Body 1) to remain in a plane defined on the other body (Body 2) connected by the joint. The planes are defined by the X and Y axes of the markers defining the joint. Body 1 can rotate about Z axis and translate along the X and Y axes of the marker which is used to define the constraint.

CHAPTER – 4

THE MICROCONTROLLER – RASPBERRY PI

4.1 CPU Specifications

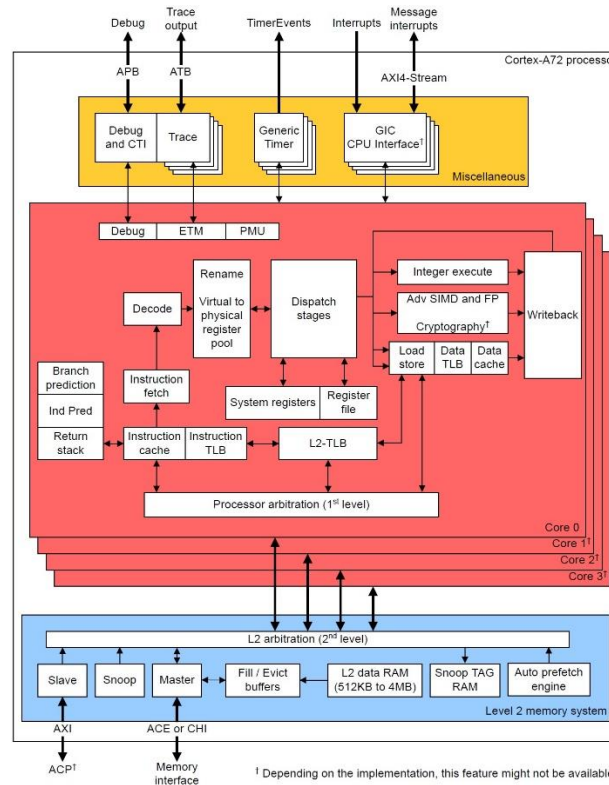


Fig 4.1 – CPU Architecture of RPI 3b

- Broadcom BCM2711, Quad core Cortex-A72 (ARM v8) 64-bit SoC @ 1.8GHz
- 1GB, 2GB, 4GB or 8GB LPDDR4-3200 SDRAM (depending on model)
- 2.4 GHz and 5.0 GHz IEEE 802.11ac wireless, Bluetooth 5.0, BLE
- Gigabit Ethernet
- 2 USB 3.0 ports; 2 USB 2.0 ports.

- Raspberry Pi standard 40 pin GPIO header (fully backwards compatible with previous boards)
- 2 × micro-HDMI ports (up to 4kp60 supported)
- 2-lane MIPI DSI display port
- 2-lane MIPI CSI camera port
- 4-pole stereo audio and composite video port
- H.265 (4kp60 decode), H264 (1080p60 decode, 1080p30 encode)
- OpenGL ES 3.1, Vulkan 1.0
- Micro-SD card slot for loading operating system and data storage
- 5V DC via USB-C connector (minimum 3A*)
- 5V DC via GPIO header (minimum 3A*)
- Power over Ethernet (PoE) enabled (requires separate PoE HAT)
- Operating temperature: 0° – 50°C ambient

* A good quality 2.5A power supply can be used if downstream USB peripherals consume less than 500mA in total.

4.2 The Master – Slave Configuration

For serial communication over the I2C protocol, the Broadcom processor of Raspberry Pi has Broadcom Serial Controller (BSC). This standard-mode master BSC controller is NXP Semiconductor's I2C compliant and supports a data transfer rate of 400 kbps. The BSC controller supports both 7-bit as well as 10-bit addressing. This I2C interface is accessible at pins GPIO2 (Board Pin No. 3) and GPIO3 (Board Pin No. 5). GPIO2 is Serial Data (SDA) line, and GPIO3 is a Serial Clock (SCL) line of the I2C1. These I2C pins are internally pulled up to 3.3V via

1.8 k Ω resistors. That is why these pins cannot be used for general-purpose I/O where pull-up is not required.

4.3 The Voice Module



Fig 4.2 – Analog Voice Module 537473

The Raspberry Pi is capable of recording audio through its USB 2.0 ports using the advanced Linux sound architecture (ALSA). The RPi can sample at 48kHz at a bit depth of 16-bits, which allows the user to record and playback fairly good quality audio. For this tutorial, I will demonstrate the process of enabling a USB audio device and using it to record and analyze acoustic signals using Python 3.x. The Python audio analysis is a great tool for engineers interested in acoustic or audio processing and even signal processing techniques. Depending on the application, the user may want to buy a nicer microphone (higher bit-depth, larger dynamic frequency range, higher sampling audio card, etc.) or a cheaper one may suffice. Cheaper and mid-tier microphone links are listed below. The microphone in our store is a fairly inexpensive microphone that is compatible with Raspberry Pi and also boasts a frequency response of 20 Hz - 16 kHz, so it is ideal for most acoustic applications.

CHAPTER – 5

RESOLVING THE JOINT KINEMATICS

5.1 The Inverse Kinematics Approach

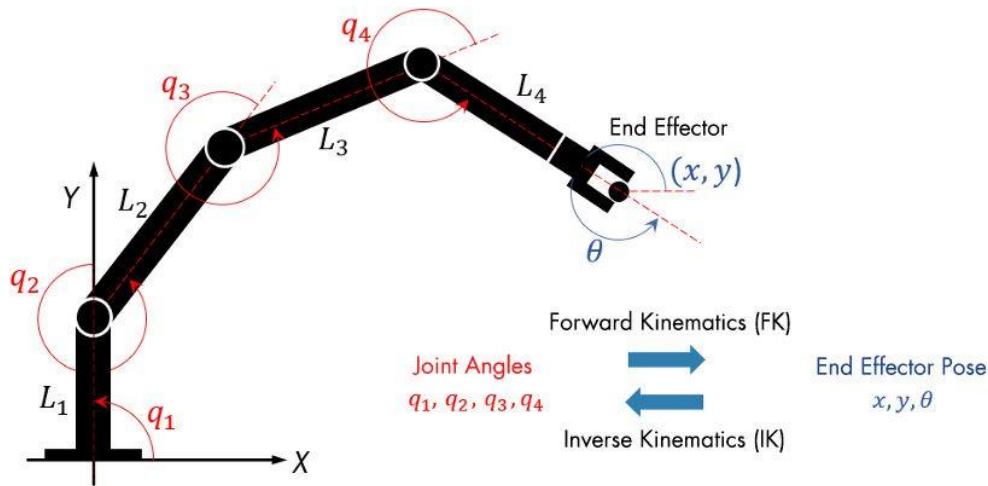


Fig 5.1 – Inverse Kin. Joint Specifications

Once the robot's joint angles are calculated using the inverse kinematics, a motion profile can be generated using the Jacobian matrix to move the end-effector from the initial to the target pose. The Jacobian matrix helps define a relationship between the robot's joint parameters and the end-effector velocities.

In contrast to forward kinematics (FK), robots with multiple revolute joints generally have multiple solutions to inverse kinematics, and various methods have been proposed according to the purpose. In general, they are classified into two methods, one that is analytically obtained (i.e., analytic solution) and the other that uses numerical calculation.

5.2 Numerical Inverse Kinematic Solutions

In order to approximate a robot configuration that achieves specified goals and constraints for the robot, numerical solutions can be used. Each joint angle is calculated iteratively using algorithms for optimization, such as gradient-based methods.

Numerical IK solvers are more general but require multiple steps to converge toward the solution to the non-linearity of the system, while analytic IK solvers are best suited for simple IK problems. Numerical IK is more versatile in that robot kinematic constraints can be specified and external constraints, like an aiming constraint for a camera arm to point at a target location, can be set to IK solvers. Determining which IK solver to apply mainly depends on the robot applications, such as real-time interactive applications, and on several performance criteria, such as the smoothness of the final pose and scalability to redundant robotics systems.

5.3 Analytical Inverse Kinematics Solutions

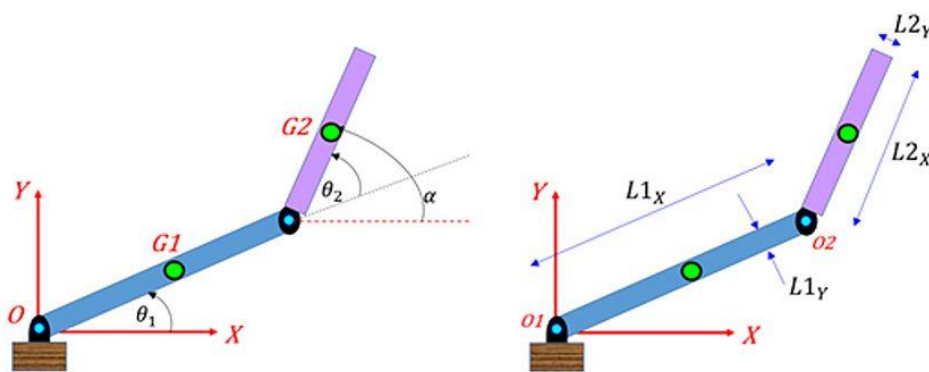


Fig 5.2 - A two-linkage robot arm with the joint angles θ_1 and θ_2 and the joint parameters to calculate the inverse kinematics solutions.

Each joint angle is calculated from the pose of the end-effector based on a mathematical formula. By defining the joint parameters and end-effector poses symbolically, IK can find all possible solutions of the joint angles in an analytic form as a function of the lengths of the linkages, its starting posture, and the rotation constraints.

Analytical IK is mainly used for robots with low degrees of freedom (DoF) due to the nonlinearity of the kinematics equations and the lack of scalability for redundant robot configurations.

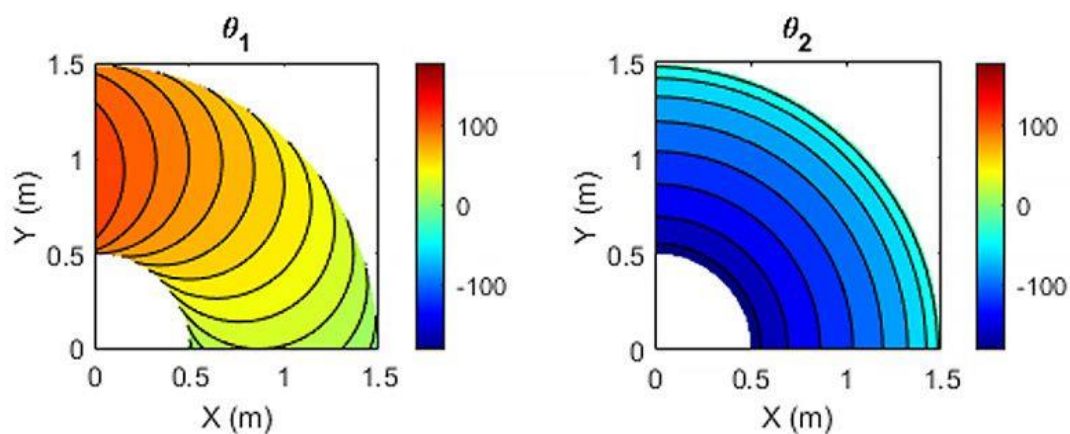


Fig 5.3 - Analytic inverse kinematic solutions of the joint angles θ_1 and θ_2 at the desired end-effector pose.

CHAPTER – 6

THE DENAVIT HARTENBERG SOLUTIONS

6.1 Denavit Hartenberg Analysis

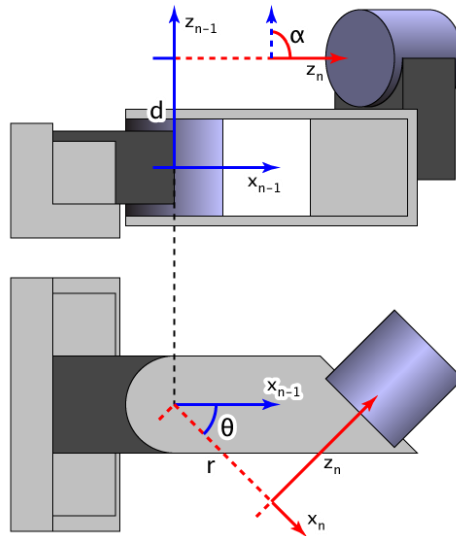


Fig 6.1 – Denavit Hartenberg Analysis of a Revolute Joint

In this convention, coordinate frames are attached to the joints between two links such that one transformation is associated with the joint, $[Z]$, and the second is associated with the link $[X]$. The coordinate transformations along a serial robot consisting of n links form the kinematics equations of the robot,

$$[T] = [Z_1][X_1][Z_2][X_2] \dots [X_{n-1}][Z_n][X_n],$$

where $[T]$ is the transformation locating the end-link.

In order to determine the coordinate transformations $[Z]$ and $[X]$, the joints connecting the links are modeled as either hinged or sliding joints, each of which have a unique line S in space that forms the joint axis and define the relative movement of the two links. A typical serial robot is characterized by a sequence of six lines $S_i, i = 1, 2, \dots, 6$, one for each joint in the robot. For each sequence of lines S_i and S_{i+1} , there is a common normal line $A_{i, i+1}$.

The system of six joint axes S_i and five common normal lines $A_{i, i+1}$ form the kinematic skeleton of the typical six degree of freedom serial robot. Denavit and Hartenberg introduced the convention that z-coordinate axes are assigned to the joint axes S_i and x-coordinate axes are assigned to the common normals $A_{i, i+1}$.

This convention allows the definition of the movement of links around a common joint axis S_i by the screw displacement,

$$[Z_i] = \begin{bmatrix} \cos \theta_i & -\sin \theta_i & 0 & 0 \\ \sin \theta_i & \cos \theta_i & 0 & 0 \\ 0 & 0 & 1 & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix},$$

where θ_i is the rotation around and d_i is the slide along the z-axis—either of the parameters can be constants depending on the structure of the robot. Under this convention the dimensions of each link in the serial chain are defined by the screw displacement around the common normal $A_{i,i+1}$ from the joint S_i to S_{i+1} , which is given by,

$$[X_i] = \begin{bmatrix} 1 & 0 & 0 & r_{i,i+1} \\ 0 & \cos \alpha_{i,i+1} & -\sin \alpha_{i,i+1} & 0 \\ 0 & \sin \alpha_{i,i+1} & \cos \alpha_{i,i+1} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix},$$

where $\alpha_{i,i+1}$ and $r_{i,i+1}$ define the physical dimensions of the link in terms of the angle measured around and distance measured along the X axis.

6.2 The Denavit Hartenberg Parameters

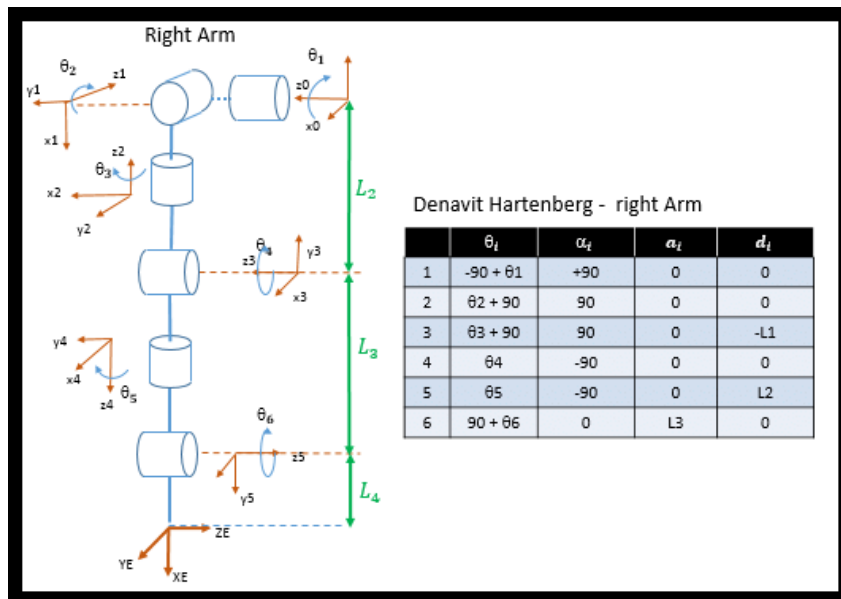


Fig 6.2 – Denavit Hartenberg Parameters for a Manipulator Arm

The following four transformation parameters are known as D–H parameters:.[4]

- d : offset along previous z to the common normal
- θ : angle about previous z, from old x to new x

- r : length of the common normal (aka a , but if using this notation, do not confuse with α). Assuming a revolute joint, this is the radius about previous z .
- α : angle about common normal, from old z axis to new z axis.

There is some choice in frame layout as to whether the previous x axis or the next x points along the common normal. The latter system allows branching chains more efficiently, as multiple frames can all point away from their common ancestor, but in the alternative layout the ancestor can only point toward one successor. Thus the commonly used notation places each down-chain x axis collinear with the common normal, yielding the transformation calculations shown below.

We can note constraints on the relationships between the axes:

- the x_n -axis is perpendicular to both the z_{n-1} and z_n axes
- the x_n -axis intersects both z_{n-1} and z_n axes
- the origin of joint n is at the intersection of x_n and z_n
- y_n completes a right-handed reference frame based on x_n and z_n

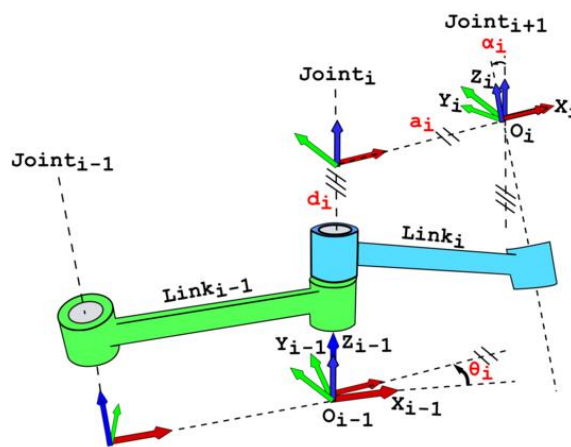


Fig 6.3 – Finalized Joint Layout

6.3 Denavit Hartenberg Matrix

It is common to separate a screw displacement into product of a pure translation along a line and a pure rotation about the line, so that

$$[Z_i] = \text{Trans}_{Z_i}(d_i) \text{Rot}_{Z_i}(\theta_i),$$

And,

$$[X_i] = \text{Trans}_{X_i}(r_{i,i+1}) \text{Rot}_{X_i}(\alpha_{i,i+1}).$$

Using this notation, each link can be described by a coordinate transformation from the concurrent coordinate system to the previous coordinate system.

$${}^{n-1}T_n = [Z_{n-1}] \cdot [X_n]$$

Note that this is the product of two screw displacements, The matrices associated with these operations are:

$$\text{Trans}_{z_{n-1}}(d_n) = \left[\begin{array}{ccc|c} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_n \\ \hline 0 & 0 & 0 & 1 \end{array} \right]$$

$$\text{Rot}_{z_{n-1}}(\theta_n) = \left[\begin{array}{ccc|c} \cos \theta_n & -\sin \theta_n & 0 & 0 \\ \sin \theta_n & \cos \theta_n & 0 & 0 \\ 0 & 0 & 1 & 0 \\ \hline 0 & 0 & 0 & 1 \end{array} \right]$$

$$\text{Trans}_{x_n}(r_n) = \left[\begin{array}{ccc|c} 1 & 0 & 0 & r_n \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ \hline 0 & 0 & 0 & 1 \end{array} \right]$$

$$\text{Rot}_{x_n}(\alpha_n) = \left[\begin{array}{ccc|c} 1 & 0 & 0 & 0 \\ 0 & \cos \alpha_n & -\sin \alpha_n & 0 \\ 0 & \sin \alpha_n & \cos \alpha_n & 0 \\ \hline 0 & 0 & 0 & 1 \end{array} \right]$$

This gives,

$${}^{n-1}T_n = \left[\begin{array}{ccc|c} \cos \theta_n & -\sin \theta_n \cos \alpha_n & \sin \theta_n \sin \alpha_n & r_n \cos \theta_n \\ \sin \theta_n & \cos \theta_n \cos \alpha_n & -\cos \theta_n \sin \alpha_n & r_n \sin \theta_n \\ 0 & \sin \alpha_n & \cos \alpha_n & d_n \\ \hline 0 & 0 & 0 & 1 \end{array} \right] = \left[\begin{array}{c|c} R & T \\ \hline 0 & 0 & 0 & 1 \end{array} \right]$$

where R is the 3×3 submatrix describing rotation and T is the 3×1 submatrix describing translation.

In some books, the order of transformation for a pair of consecutive rotation and translation (such as d_n and θ_n) is replaced. However, because matrix multiplication order for such pair does not matter, the result is the same. For example:

$$\text{Trans}_{z_{n-1}}(d_n) \cdot \text{Rot}_{z_{n-1}}(\theta_n) = \text{Rot}_{z_{n-1}}(\theta_n) \cdot \text{Trans}_{z_{n-1}}(d_n)$$

6.4 Kinematics in Denavit Hartenberg Analysis

Further matrices can be defined to represent velocity and acceleration of bodies. The velocity of body i with respect to body j can be represented in frame k by the matrix.

$$W_{i,j(k)} = \left[\begin{array}{ccc|c} 0 & -\omega_z & \omega_y & v_x \\ \omega_z & 0 & -\omega_x & v_y \\ -\omega_y & \omega_x & 0 & v_z \\ \hline 0 & 0 & 0 & 0 \end{array} \right]$$

Where ω is the angular velocity of body j with respect to body i and all the components are expressed in frame k ; v is the velocity of one point of the body j with respect to body i (the pole).

The acceleration matrix can be defined as the sum of the time derivative of the velocity plus the velocity squared

$$H_{i,j(k)} = \dot{W}_{i,j(k)} + W_{i,j(k)}^2$$

It is also possible to prove that

$$\dot{M}_{i,j} = W_{i,j(i)} M_{i,j}$$

$$\ddot{M}_{i,j} = H_{i,j(i)} M_{i,j}$$

Velocity and acceleration matrices add up according to the following rules

$$W_{i,k} = W_{i,j} + W_{j,k}$$

$$H_{i,k} = H_{i,j} + H_{j,k} + 2W_{i,j}W_{j,k}$$

in other words the absolute velocity is the sum of the parent velocity plus the relative velocity; for the acceleration the Coriolis' term is also present.

CHAPTER – 7

DYNAMICS OF THE MANIPULATOR

7.1 Dynamical Analysis

For the dynamics three further matrices are necessary to describe the inertia J , the linear and angular momentum Γ and the forces and torques Φ applied to a body.

Inertia J :

$$J = \left[\begin{array}{ccc|c} I_{xx} & I_{xy} & I_{xz} & x_g m \\ I_{yx} & I_{yy} & I_{yz} & y_g m \\ I_{zx} & I_{zy} & I_{zz} & z_g m \\ \hline x_g m & y_g m & z_g m & m \end{array} \right]$$

Where m is the mass, x_g, y_g, z_g represent the position of the center of mass, and the terms I_{xx}, I_{xy}, \dots represent inertia and are defined as

$$I_{xx} = \iint x^2 dm$$

$$I_{xy} = \iint xy dm$$

$$I_{xz} = \dots$$

$$\vdots$$

7.2 Modified D-H Parameters

Compared with the classic DH parameters, the coordinates of frame O_{i-1} is put on axis $i-1$, not the axis i in classic DH convention. The coordinates of O_i is put on the axis i , not the axis $i+1$ in classic DH convention.

Another difference is that according to the modified convention, the transform matrix is given by the following order of operations:

$${}^{n-1}T_n = \text{Rot}_{x_{n-1}}(\alpha_{n-1}) \cdot \text{Trans}_{x_{n-1}}(a_{n-1}) \cdot \text{Rot}_{z_n}(\theta_n) \cdot \text{Trans}_{z_n}(d_n)$$

Thus, the matrix of the modified DH parameters becomes

$${}^{n-1}T_n = \left[\begin{array}{ccc|c} \cos \theta_n & -\sin \theta_n & 0 & a_{n-1} \\ \sin \theta_n \cos \alpha_{n-1} & \cos \theta_n \cos \alpha_{n-1} & -\sin \alpha_{n-1} & -d_n \sin \alpha_{n-1} \\ \sin \theta_n \sin \alpha_{n-1} & \cos \theta_n \sin \alpha_{n-1} & \cos \alpha_{n-1} & d_n \cos \alpha_{n-1} \\ \hline 0 & 0 & 0 & 1 \end{array} \right]$$

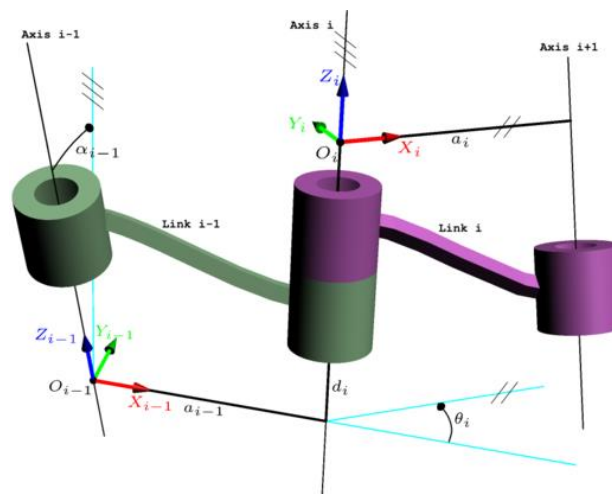


Fig 7.1 – Modified D-H Parameters

CHAPTER – 8

DESIGN AND MODELING

8.1 Modelling of the Manipulator Arm

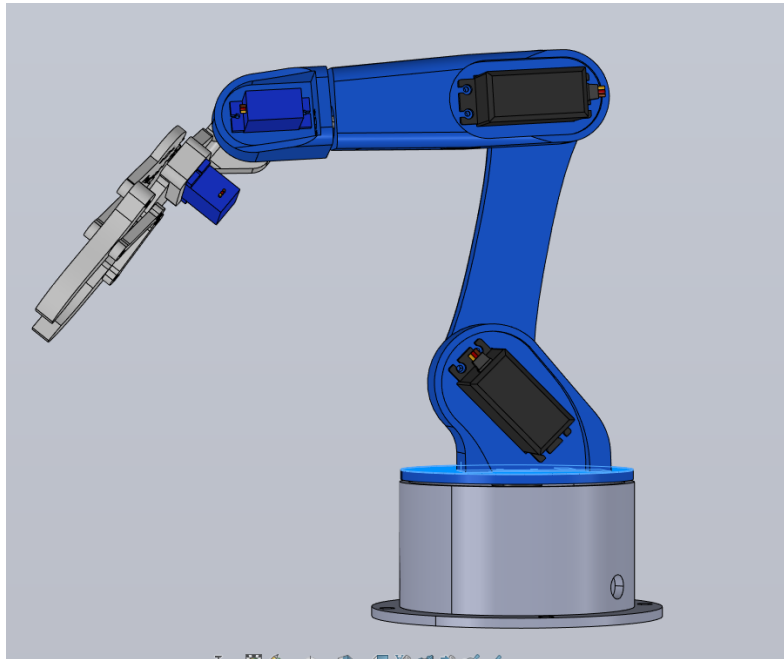


Fig 8.1 – Side View of the Manipulator

Before starting the modeling process, it is essential to gather information about the robot's specifications, including the dimensions, axis movements, and any other relevant details. This information will help in creating an accurate model.

The first step in modeling a 6-axis robot is to create the base. The base provides the robot with stability and support. In SolidWorks, create a new part file and sketch the base's outline using the sketch tools. Next, use the extrude feature to create a 3D model of the base.

The next step is to create the first axis of the robot. The first axis is responsible for vertical movement. In SolidWorks, create a new part file and sketch the outline of the first axis using the sketch tools. Next, use the extrude feature to create a 3D model of the first axis.

The second axis is responsible for horizontal movement. In SolidWorks, create a new part file and sketch the outline of the second axis using the sketch tools. Next, use the extrude feature to create a 3D model of the second axis.

The third axis is responsible for rotational movement. In SolidWorks, create a new part file and sketch the outline of the third axis using the sketch tools. Next, use the extrude feature to create a 3D model of the third axis.

The fourth axis is responsible for horizontal movement. In SolidWorks, create a new part file and sketch the outline of the fourth axis using the sketch tools. Next, use the extrude feature to create a 3D model of the fourth axis.

The fifth axis is responsible for rotational movement. In SolidWorks, create a new part file and sketch the outline of the fifth axis using the sketch tools. Next, use the extrude feature to create a 3D model of the fifth axis.

The sixth axis is responsible for vertical movement. In SolidWorks, create a new part file and sketch the outline of the sixth axis using the sketch tools. Next, use the extrude feature to create a 3D model of the sixth axis.

Once all the axes are created, it is time to assemble them. In SolidWorks, create a new assembly file and insert each axis into the assembly. Use the mate feature to align and connect each axis.

The last step is to add the gripper to the robot. The gripper is responsible for holding and manipulating objects. In SolidWorks, create a new part file and sketch the outline of the gripper using the sketch tools. Next, use the extrude feature to create a 3D model of the gripper. Insert the gripper into the assembly and use the mate feature to connect it to the sixth axis.

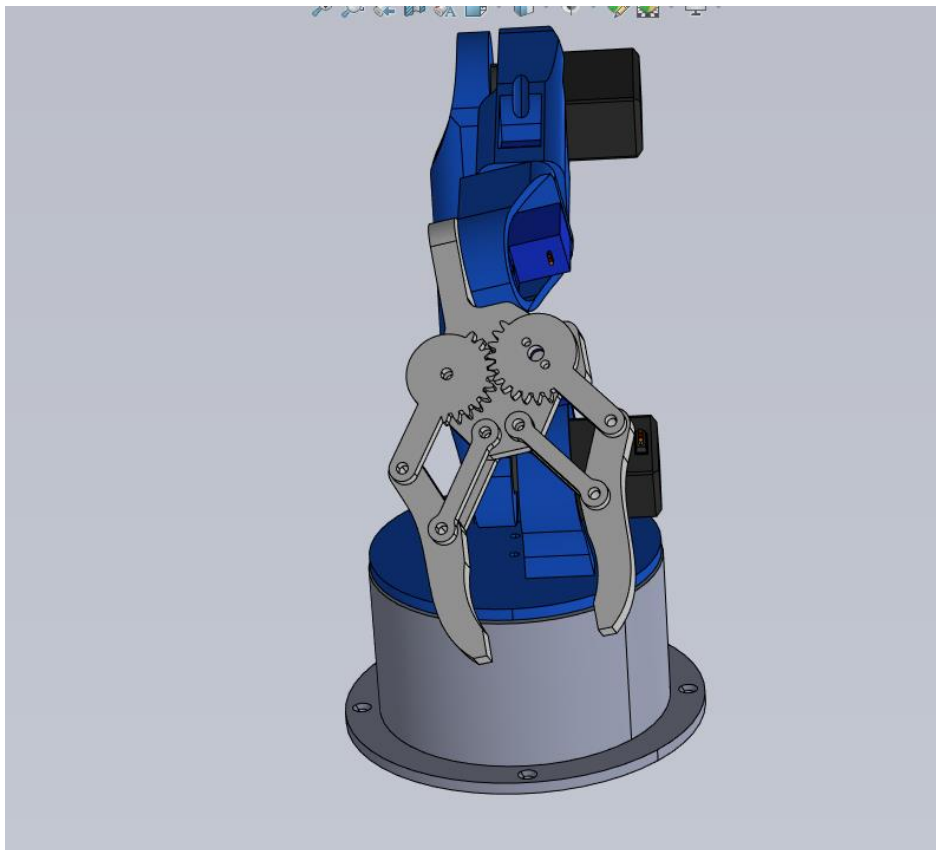


Fig 8.2 – Front View of the Manipulator

CHAPTER – 9

CIRCUIT CONSTURCUCTION AND CONFIGURATION

9.1 – The Master-Slave Configuration

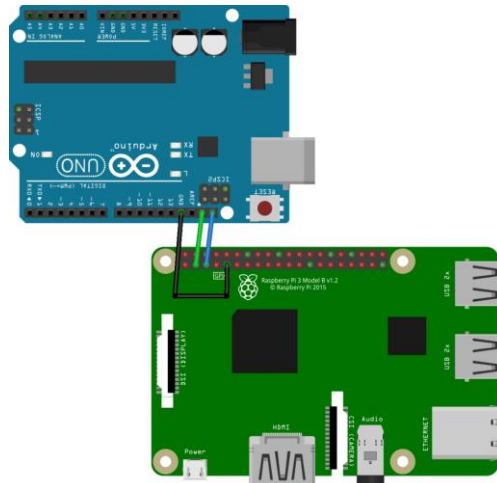


Fig 9.1 – Master-Slave Config of RPI3

- Link the GND of the Raspberry Pi to the GND of the Arduino.
- Connect the SDA (I2C data) of the Pi (pin 2) to the Arduino SDA.
- Connect the SCL (I2C clock) of the Pi (pin 3) to the Arduino SCL.

Important note: the Raspberry Pi 4 (and earlier) is running under 3.3V, and the Arduino Uno is running under 5V!

If the Raspberry Pi is configured as a master and the Arduino as a slave on the I2C bus, then you can connect the SDA and SCL pins directly. To make it simple, in this scenario the Raspberry Pi will impose 3.3V, which is not a problem for the Arduino pins.

9.2 – Arduino I2C slave program

A screenshot of an Arduino IDE window with a yellow background. The code is displayed in a dark-themed editor with syntax highlighting. The code is for an I2C slave program. It includes the Wire.h library, defines the slave address as 0x08, and initializes a byte variable data_to_echo to 0. The setup function initializes the Wire library with the slave address and registers the receiveData and sendData functions. The loop function is empty. The receiveData function reads the received data into data_to_echo. The sendData function writes data_to_echo back to the I2C bus.

```
#include <Wire.h>

#define SLAVE_ADDRESS 0x08

byte data_to_echo = 0;

void setup()
{
  Wire.begin(SLAVE_ADDRESS);

  Wire.onReceive(receiveData);
  Wire.onRequest(sendData);
}

void loop() { }

void receiveData(int bytecount)
{
  for (int i = 0; i < bytecount; i++)
  {
    data_to_echo = Wire.read();
  }
}

void sendData()
{
  Wire.write(data_to_echo);
}
```

Fig 9.2 – Master Slave Program

9.3 I2C Data Monitoring

The I2C data signals obtained from the slave Arduino UNO R3 can be decoded and analyzed by the Pulseview Software. It is noted that the type of signal received from

the slave microcontroller is the Pulse Width Modulation pattern. The screenshots of the analysis are provided below.

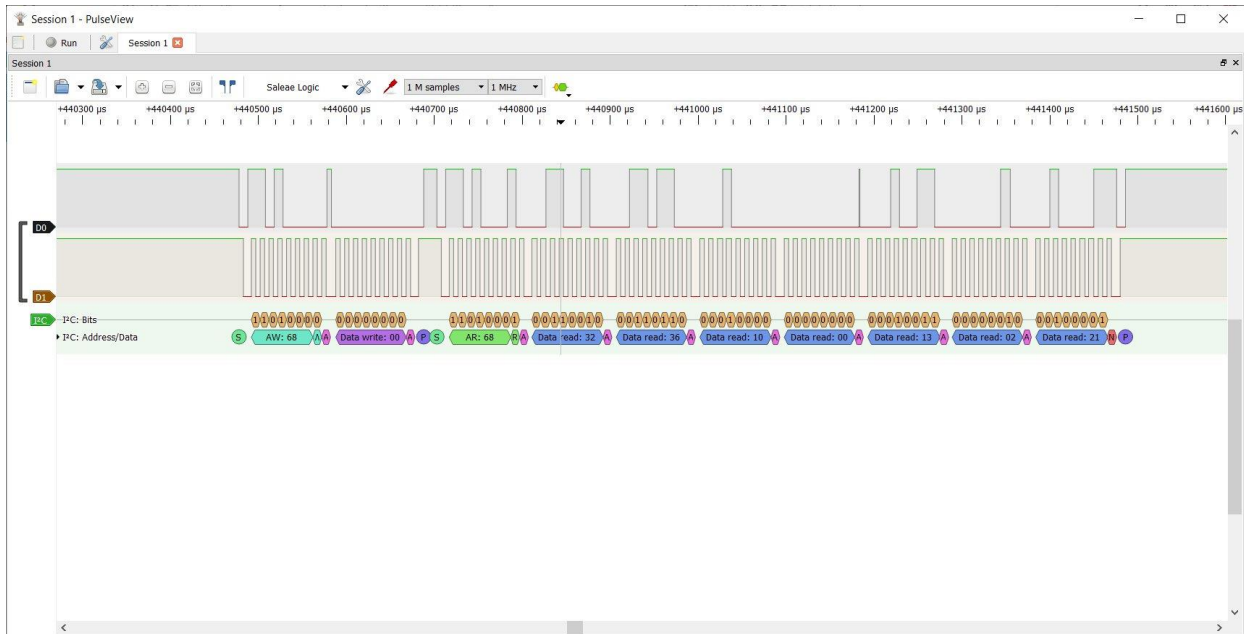


Fig 9.3 – PWM Analysis in PulseView

9.4 I2C Protocol

I2C (Inter-Integrated Circuit) is a communication protocol used by many devices, including Arduino and Raspberry Pi, to exchange data with each other. Here are some theories related to the I2C configuration of Arduino and Raspberry Pi:

- **I2C Bus Configuration:** The I2C bus is a two-wire interface that consists of a data line (SDA) and a clock line (SCL). Both Arduino and Raspberry Pi have dedicated pins for SDA and SCL. In Arduino, the A4 pin is used for SDA, and the A5 pin is used for SCL. In Raspberry Pi, the GPIO 2 pin is used for SDA, and the GPIO 3 pin is used for SCL. These pins can be configured as either master or slave devices using software libraries like Wire.h in Arduino and smbus in Raspberry Pi.

- Addressing: Each I2C device on the bus must have a unique address that is used to identify it. The address is either 7 bits or 10 bits long, depending on the device. In Arduino, the Wire library provides functions like `Wire.beginTransmission()` and `Wire.write()` to set the address and send data to the slave device. In Raspberry Pi, the `smbus` library provides similar functions like `smbus.SMBus()` and `smbus.write_byte()`.

- Clock Speed: The I2C bus supports different clock speeds, ranging from a few kilohertz to several megahertz. The clock speed determines how fast the devices can communicate with each other. In Arduino, the default clock speed is 100 kHz, but it can be changed using the `Wire.setClock()` function. In Raspberry Pi, the default clock speed is 100 kHz, but it can be changed using the `i2cdetect` command.

- Pull-up Resistors: I2C signals are transmitted using open-drain outputs, which require pull-up resistors to maintain the signal voltage. The pull-up resistors can be internal or external, depending on the device. In Arduino, the internal pull-up resistors can be enabled using the `pinMode()` and `digitalWrite()` functions. In Raspberry Pi, the internal pull-up resistors are enabled by default, but they can be disabled using the `i2cdetect` command. External pull-up resistors can also be used if necessary.

- Multi-master Configuration: In a multi-master configuration, multiple devices on the bus can act as master devices and control the communication. This requires careful management of the bus to avoid conflicts and data corruption. In Arduino, the Wire library provides functions like

`Wire.endTransmission()` and `Wire.requestFrom()` to manage multi-master communication. In Raspberry Pi, the `smbus` library provides similar functions like `smbus.write_byte_data()` and `smbus.read_byte_data()`.

9.5 Voice Module Integration

- **Hardware Connection:** To connect a voice module to a Raspberry Pi, you need to identify the appropriate GPIO pins or USB port to connect the module. Most voice modules come with documentation or pinout diagrams that show how to connect them to a Raspberry Pi. For example, if you're using a USB voice module, you can connect it to any available USB port on the Raspberry Pi.
- **Software Configuration:** Once the hardware is connected, you need to configure the software on the Raspberry Pi to communicate with the voice module. Depending on the type of module you're using, this may involve installing drivers or software libraries. For example, if you're using a USB voice module, you may need to install drivers for the module, while if you're using a serial voice module, you may need to install PySerial library.
- **Speech Recognition:** To enable speech recognition, you can use one of several open-source speech recognition libraries available for the Raspberry Pi. Popular speech recognition libraries include PocketSphinx, CMU Sphinx, or Google Cloud Speech-to-Text. These libraries can take an audio signal from the voice module and convert it to text.

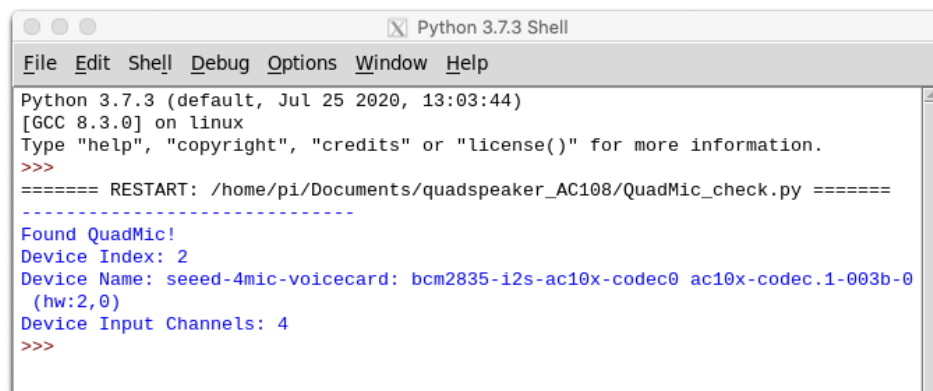
- **Text-to-Speech:** To enable text-to-speech, you can use one of several text-to-speech libraries available for the Raspberry Pi. Popular text-to-speech libraries include Festival or eSpeak. These libraries can take text and convert it to an audio signal that can be played through the voice module.
- **Integration with Other Applications:** Once you have enabled speech recognition and text-to-speech, you can integrate the voice module with other applications running on the Raspberry Pi. For example, you can use voice commands to control GPIO pins, run scripts, or even control a robot.
- **Noise Reduction:** To improve the accuracy of speech recognition, it is important to reduce background noise and other unwanted sounds that the voice module may pick up. This can be achieved by using a microphone with noise-cancelling capabilities or applying noise reduction algorithms to the audio signal.
- **Language Model:** The accuracy of speech recognition also depends on the language model used. You can train a language model on a specific language or domain to improve the accuracy of speech recognition. There are several open-source language models available for the Raspberry Pi, such as the Voxforge speech corpus.

CHAPTER – 10

ANALYSIS

10.1 Voice Recognition analysis

The QuadMic Array is a 4-microphone array based around the AC108 quad-channel analog-to-digital converter (ADC) with Inter-IC Sound (I2S) audio output capable of interfacing with the Raspberry Pi. The QuadMic can be used for applications in voice detection and recognition, acoustic localization, noise control, and other applications in audio and acoustic analysis. The QuadMic will be connected to the header of a Raspberry Pi 4 and used to record simultaneous audio data from all four microphones. Some signal processing routines will be developed as part of an acoustic analysis with the four microphones. Algorithms will be introduced that approximate acoustic source directivity, which can help with understanding and characterizing noise sources, room and spatial geometries, and other aspects of acoustic systems. Python is also used for the analysis.



```
Python 3.7.3 Shell
File Edit Shell Debug Options Window Help
Python 3.7.3 (default, Jul 25 2020, 13:03:44)
[GCC 8.3.0] on linux
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: /home/pi/Documents/quadspeaker_AC108/QuadMic_check.py =====
Found QuadMic!
Device Index: 2
Device Name: seeed-4mic-voicecard: bcm2835-i2s-ac10x-codec0 ac10x-codec.1-003b-0
(hw:2,0)
Device Input Channels: 4
>>>
```

Fig 10.1 – Detection of Quad Mic in Raspberry Pi

The QuadMic device finder routine above will be used to specify the index to be read by the audio streaming function in pyaudio. If the snippet doesn't print out information about a device - then it is likely that one or multiple issues are a result:

- The Raspberry Pi needs to be restarted
- The Sseed voice card was not properly installed
- The QuadMic is improperly wired/aligned to the Raspberry Pi
- The QuadMic is damaged
- The Raspberry Pi Version and/or operating system is not compatible with the Sseed voice card

Under the assumption that the QuadMic has been found by pyaudio on the Raspberry Pi, the next section will test the four microphones and introduce methods for verifying the functionality of each.

10.2 Microphone Array Visualizations and Test

The QuadMic can be tested by taking sample measurements while tapping each microphone. This will allow the user to ensure that each microphone is correlated correctly with the respective measurement and parsing of each data vector.

The code below plots all four microphone responses in real time at a sample rate of 16kHz in chunks of 4000 points. This equates to roughly 250ms recordings for all four microphones simultaneously.

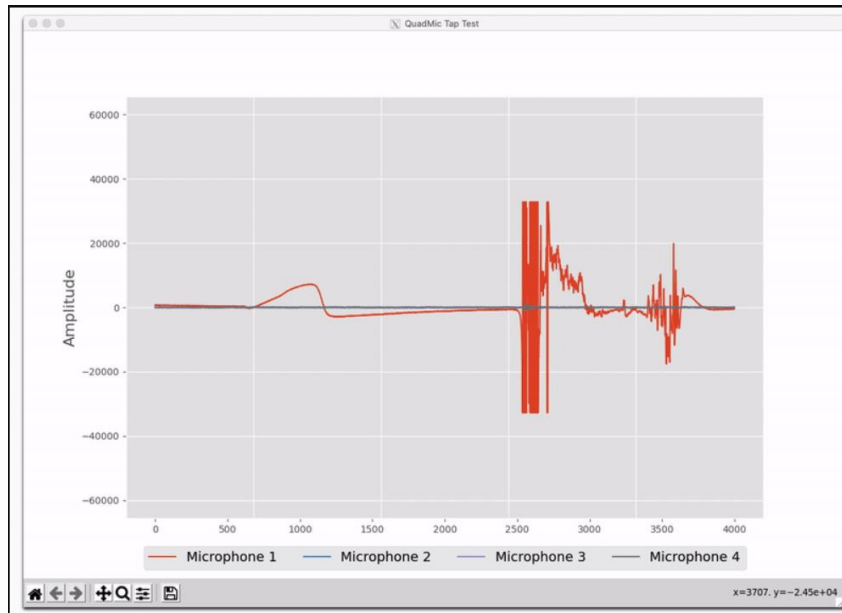


Fig 10.2 – Visualization of Mic Response

One thing to note in the tap test is the peaking of the microphone signal. The microphones peak at 2 raised to the 15th power, both negatively and positively. Anytime the microphone is peaking it can cause damage, and thus, it shouldn't be done often. Therefore, the tap test should serve as a starting method of ensuring that the channels are being parsed correctly. Going forward, only acoustic inputs will be used to force the microphones into a response.

10.3 Arduino Manipulator System Design

Micros Servos are controlled by sending an electrical pulse of variable width, or pulse width modulation (PWM), through the control wire. A servo motor can usually turn 180° in either direction. Robot Arm has 4 degrees of freedom which includes: Shoulder rotation, Elbow rotation, Wrist pitch and roll. Using just one input pin, servos receive the position from the Arduino and they go there. Internally, they have a motor driver and a feedback circuit that makes sure that the servo arm reaches the desired position. It is a square wave similar to PWM. Each cycle in the signal lasts

for 20 milliseconds and for most of the time, the value is LOW. At the beginning of each cycle, the signal is HIGH for a time between 1 and 2 milliseconds. At 1 millisecond it represents 0 degrees and at 2 milliseconds it represents 180 degrees. In between, it represents the value from 0–180. This is a very good and reliable method. A potentiometer is used to measure angular position. The varying voltage is directly proportional to the angular position of the shaft connected to the center of the potentiometer. This allows in obtaining an analog measurement of an angular position slide. A potentiometer is connected to the central axis of the servo motor. The potentiometer allows the control circuitry; monitor the current angle of the servo motor. If the shaft of the servo is at the correct angle, then the engine is off. If the circuit checks that the angle is not correct, the motor will turn in the right direction until the correct angle. Push buttons are used to store the value of the potentiometer which is used during teach mode, and the other is used during play mode that is which performs the continuous action until the next movement of the arm is changed.

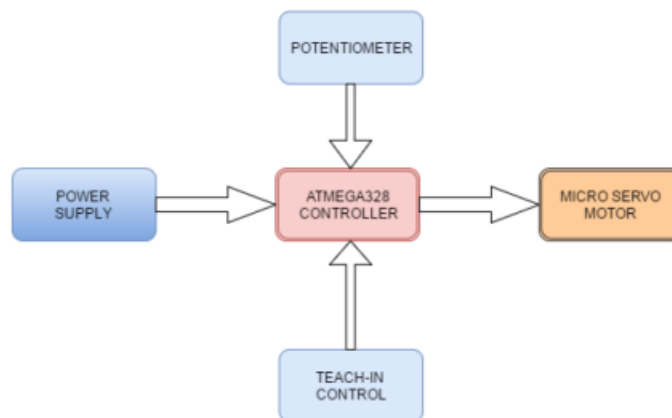


Fig 10.3 – Block diagram for the Manipulator

10.4 Circuit Mapping of Servo

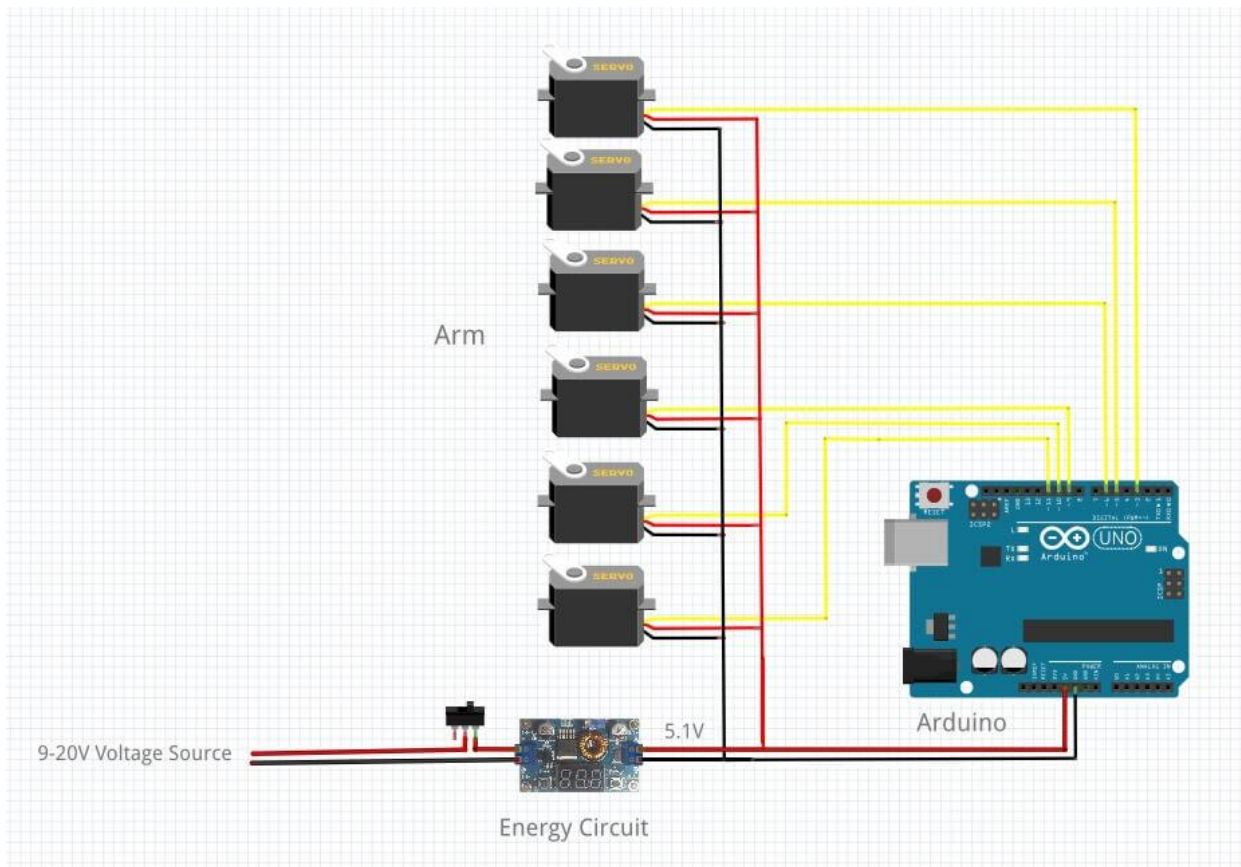


Fig 10.4 – Circuit Diagram of Mapped Servos to the Arduino

10.5 Manipulator Torque Analysis

Torque is defined as the turning or twisting force and it is calculated using the following relationship:

$$\text{Torque } (\tau) = \text{Force } (F) \times \text{Length } (L)$$

$$\tau = F \times L \quad (4)$$

$$F = W = m \times g \quad (5)$$

Where W is the weight, m is the mass, and g is the acceleration due to gravity, therefore,

$$\tau = m \times g \times L \quad (6)$$

$$\tau = W \times L$$

In order to estimate the torque required at each joint, the worst case scenario must be chosen. A link of length L is rotated clockwise. Only the perpendicular component of length between the pivot and the force is taken into account. It was observed that this distance decreases from L_3 to L_1 (L_1 being zero). Since the equation for torque is length (or distance) multiplied by the force, the greatest value will be obtained using L_3 , since F does not change. The link can be similarly rotated counterclockwise and the same effect will be observed.

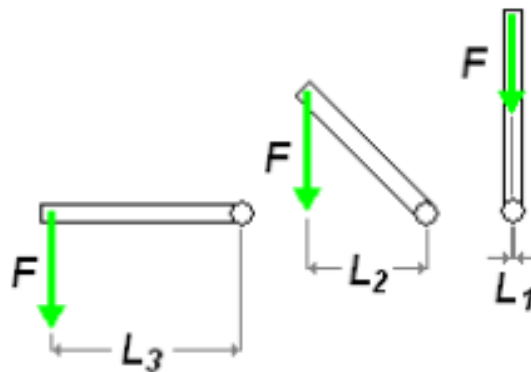


Fig 10.5 – Rotation of Link Length

This means that a joint revolving vertically is not involved in weight lighting and thus has a zero torque.

10.6 Joint Torques Calculations

The major purpose of joint force calculations in figure 2 is for motor selection. In most robot arm designs the weight of the robot arm and the weight of a possible load are considered in choosing a motor in order to achieve a good design that can support the weight of the arm and the load.

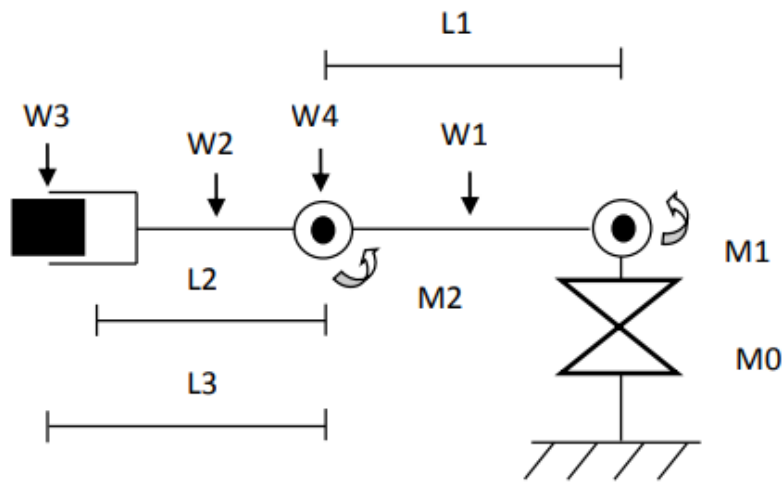


Fig 10.6 – Force Calculation of Joints

To carry out a moment arm calculation, multiplying downward force with the linkage lengths. This calculation must be done for each lifting actuator. The manipulator design in this work has two DOF that requires lifting, and the center of mass of each linkage is assumed to be $\text{Length}/2$.

The remaining degree of freedom does not require lifting. For the manipulator to function properly the following point must be met: The total torque about a joint must be equal or less than the torque produced by the actuator ($\sum \tau_{\text{about a joint}} \leq \tau_{\text{produced by the actuator}}$). The robotic manipulator joint torque is determined as follows:

Torque about Joint 1:

$$M1 = L1 / 2 * W1 + L1 * W4 + (L1 + L2 / 2) * W2 + (L1 + L3) * W3$$

Torque about Joint 2:

$$M2 = L2 / 2 * W2 + L3 * W3$$

Where:

W1 = Weight of link1,

W2 = Weight of link2,

W3 = Weight of load,

W4 = Weight of actuator (servo) 2,

L1 = Length of link1,

L2 = Length of link2,

L_{Load} = Length of the load (end-effector),

L3 = Length2 + (1/2 * L_{Load}),

M0 = Base Actuator (Cylindrical movement),

M1 = Actuator 1 (Joint I),

M2 = Actuator 2 (Joint II).

However, the above equations for torque calculation only deal with the case where the robot arm is being held horizontally (not in motion). For the arm to move from a rest position, acceleration is required. To solve for this added torque, it is known that the sum of torques acting at a pivot point is equal to the moment of inertia (J) multiplied by the angular acceleration (a):

$$\tau = J \times A$$

The following parameters were applied for the joint torque calculation:

$$W1 = 10.9\text{N},$$

$$W2 = 0.6\text{N},$$

$$W3 = 0.12\text{N},$$

$$W4 = 0.04\text{N},$$

$$L1 = 1.25\text{m},$$

$$L2 = 1\text{m},$$

$$L_{\text{Load}} = 0.71\text{m},$$

$$L3 = 1.4\text{m}$$

From the joint torque calculations, the calculated torques at the two joints that require lifting and the selected actuators torques for the respective joints are shown below. With the calculated joint torques, the joint actuators were selected from the actuator stepper motor manufacturer's catalog of NMB Corporation.

Joint	Calculated Torque (g-cm)	Selected Actuator Torque	
I	2056.7	2200	
II	451.7	500	

Table 10.1 – Calculated and Selected Torques for Joint I and II

10.7 Analysis Results

The results of the computation of the robot arm dynamics based on the Lagrange-Euler iteration method. The position, velocity against time graph of the joints I and II show a good trajectory of the robot arm movements. The angular position and velocity graphs of link I and II did not maintain a straight line or zero position; rather they yielded regular change in movement. Secondly, the position graph of link II appears higher than that of link I because the link II makes its angular movement in such a way that it adds to that of link I. These satisfy the performance characteristics of the arm.

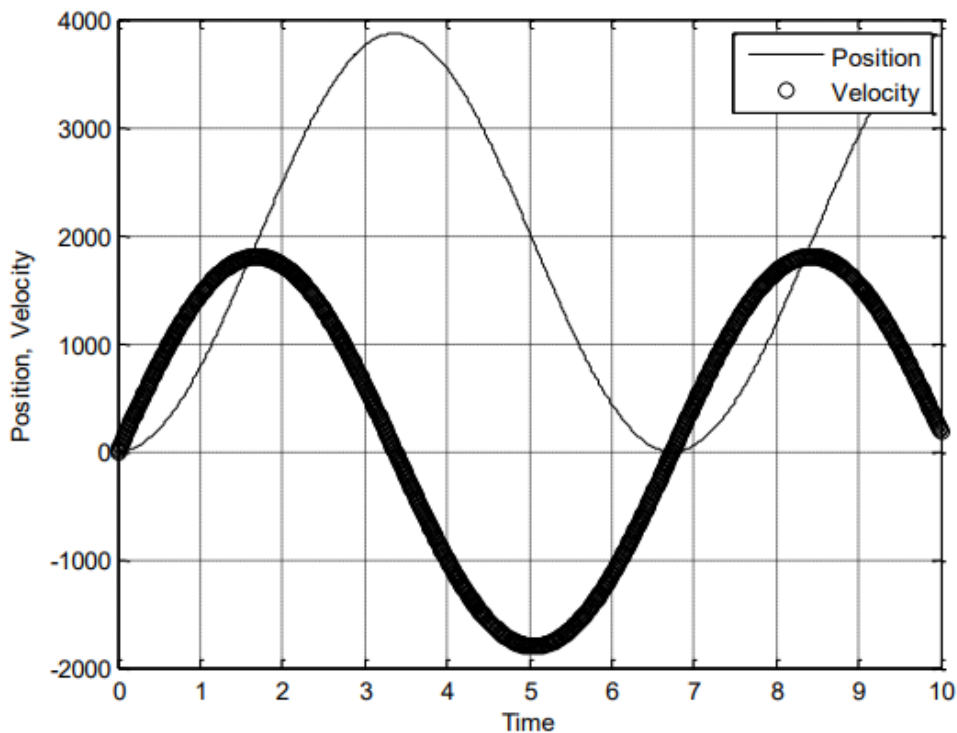


Fig 10.7 – Comparison of Velocity and Position of Manipulator's Joint I

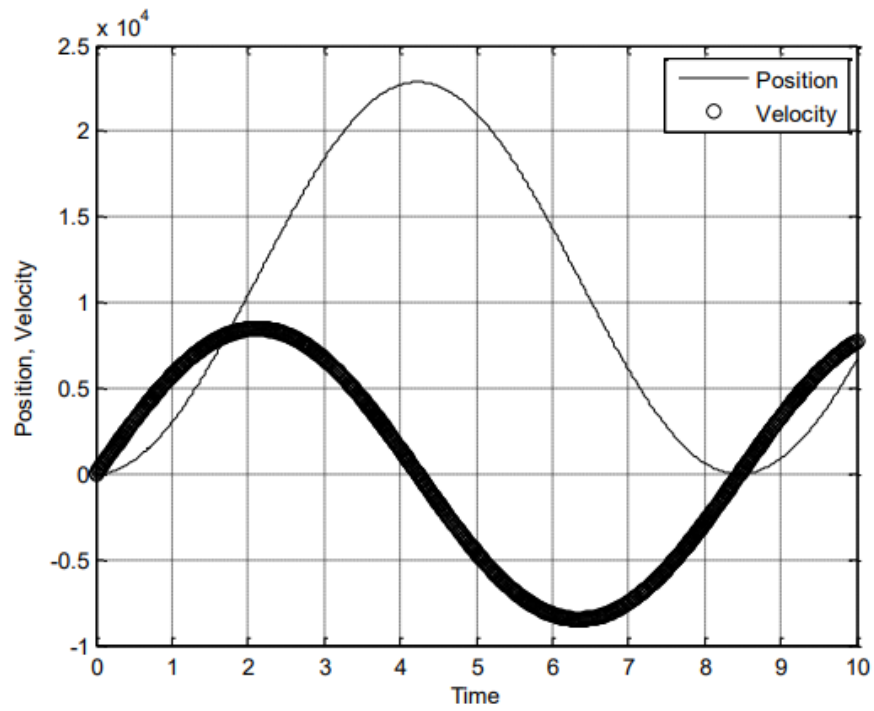


Fig 10.8 – Comparison of Velocity and Position of Manipulator's Joint II

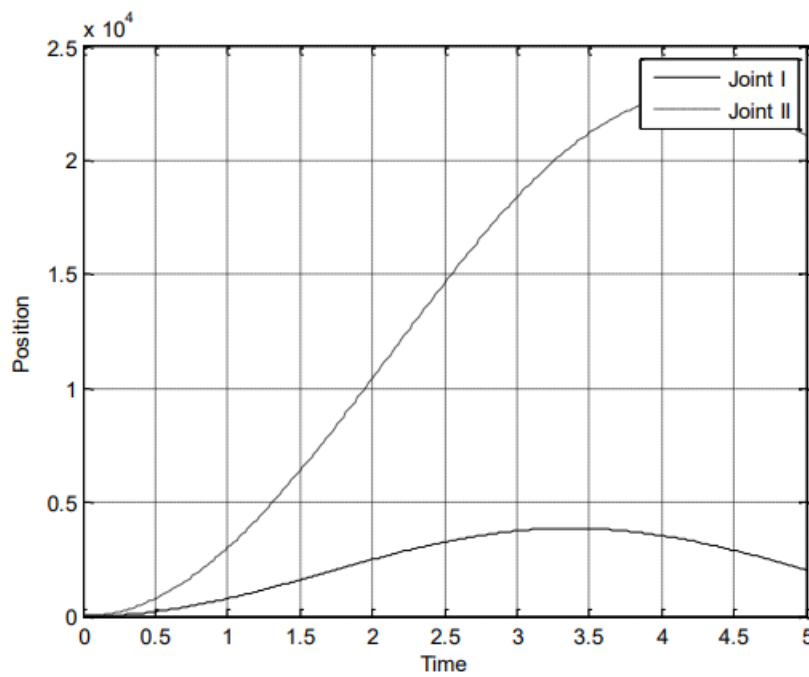


Fig 10.9 - Comparing positions of Joints I and II of the robot arm with initial values; $q1=30$, $q2=30$, $q1=q2=0$

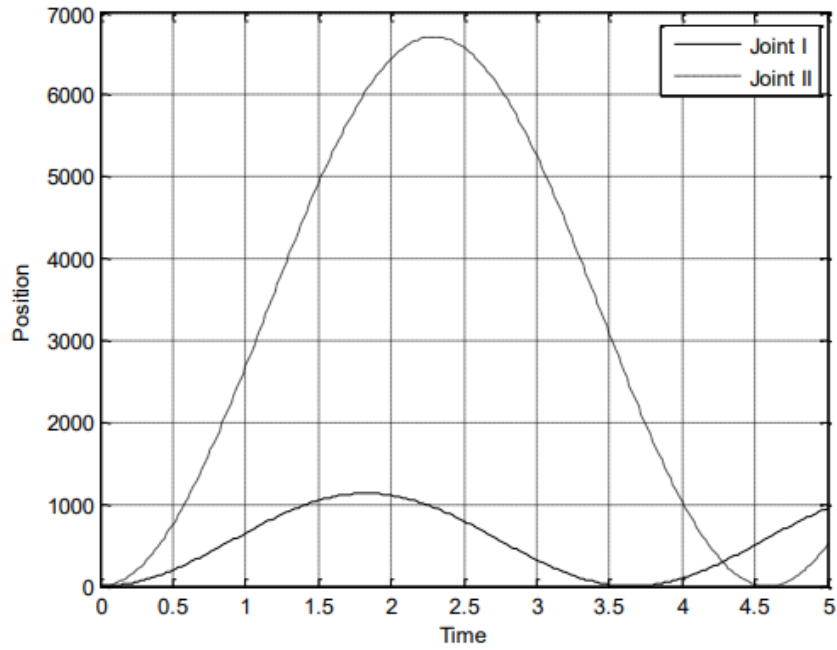


Fig 10.10 - Comparing positions of Joints I and II of the robot arm with initial values; $q1=45$, $q2=45$, $q1=q2=0$

The robotic manipulator is a complex system which has been difficult to model completely; as a result, many researchers have modeled it based on different perspectives and purposes. The aim of the model is to develop a mathematical description of the system which can help in the development and optimization of the characteristics of the system. The Lagrange Euler model was applied here because the purpose of this work is to analyze the working characteristics of the arm based on the selected torque. The applied torques were selected based on the calculated joint torque. From the computation results of the Lagrange Euler iteration the position and velocity graphs of the joints followed the desired trajectories. This method of joint torque calculation and actuator selection should be applied in every robot design for proper functioning of the system.

CHAPTER – 11

MATERIALS & COMPONENTS USED

11.1 Arduino UNO R3

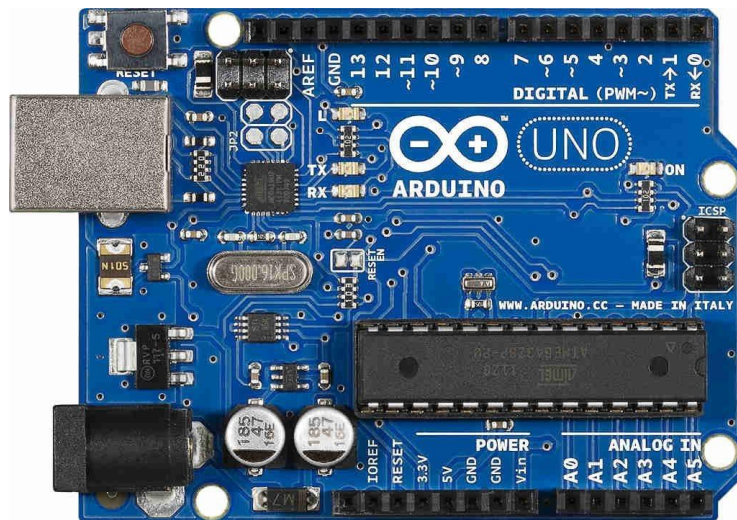


Fig 11.1 – Arduino UNO R3

Arduino UNO is a microcontroller board based on the ATmega328P. It has 14 digital input/output pins (of which 6 can be used as PWM outputs), 6 analog inputs, a 16 MHz ceramic resonator, a USB connection, a power jack, an ICSP header and a reset button. It contains everything needed to support the microcontroller; simply connect it to a computer with a USB cable or power it with a AC-to-DC adapter or battery to get started. You can tinker with your UNO without worrying too much about doing something wrong, worst case scenario you can replace the chip for a few dollars and start over again.

11.2 Raspberry PI Model 3B+

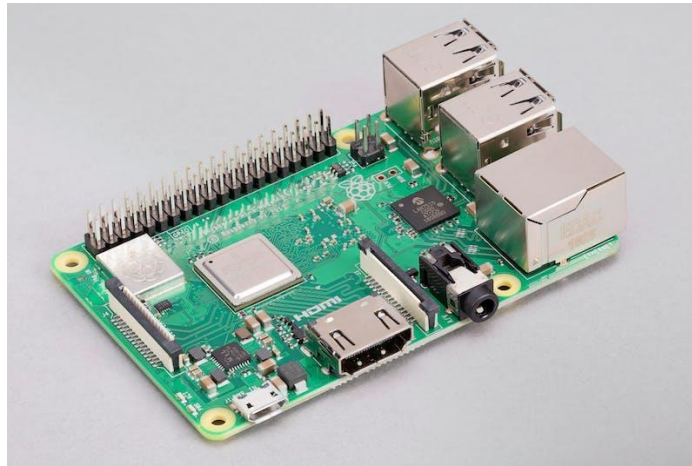


Fig 11.2 – Raspberry PI Model 3B+

The final revision of our third-generation single-board computer.

1.4GHz 64-bit quad-core processor, dual-band wireless LAN, Bluetooth 4.2/BLE, faster Ethernet, and Power-over-Ethernet support (with separate PoE HAT).

11.3 MAX4466 Microphone Module

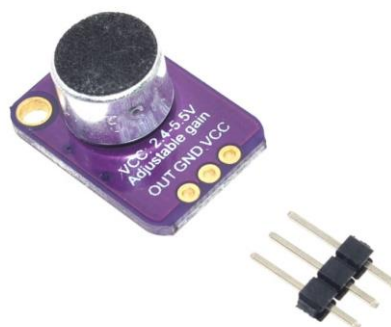


Fig 11.3 – MAX4466 Microphone Module

- +2.4V to +5.5V Supply Voltage Operation
- Versions with 5nA Complete Shutdown Available (MAX4467/MAX4468)
- Excellent Power-Supply Rejection Ratio: 112dB
- Excellent Common-Mode Rejection Ratio: 126dB
- High A_{VOL} : 125dB ($R_L = 100k\Omega$)
- Rail-to-Rail Outputs

11.4 Buck Converter

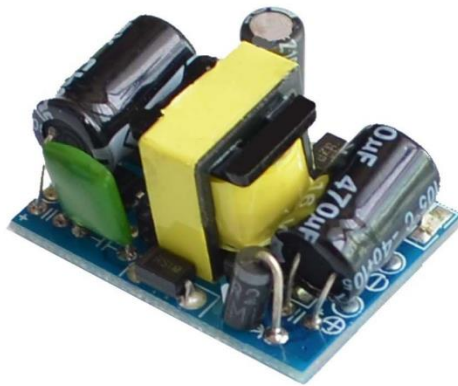


Fig 11.4 – Step Down Rectifier

This is a 5V 700mA power supply module . Which can be used to power any DIY projects that can be operated via 5VDC. The module has a power rating of 3.5W.

11.5 MG995 Servo Motor



Fig 11.5 – MG995 Servo

MG995 is a digital, metal gear, high torque servo for RC robotic models.

- Dimension: 39.5mm x 20.5mm x 40.7mm
- Stall Torque: 9.4kg/cm (4.8v); 11kg/cm (6v)
- Op. speed: 0.20sec/60degree (4.8v); 0.16sec/60degree (6.0v)
- Operating voltage: 4.8~ 6.6v
- Gear Type: Metal gear
- Temperature range: 0- 55deg
- Servo wire length: 30cm
- Rotation angle: 180 degrees.

11.6 12v SMPS



Fig 11.6 – 12v SMPS

Output Specifications:

- DC Voltage: 12V
- Rated Current: 5A
- Current Range: 0-5A
- Rated Power: 60W
- Ripple & Noise (maximum): 150mV
- Voltage Adjustable Range: 10-13.2V
- Voltage Tolerance: $\pm 2.0\%$
- Line Regulation: $\pm 0.5\%$

- Load Regulation: $\pm 0.5\%$
- Setup, Rise, Hold Up Time: 200ms, 50ms, 20ms at full load.

Input Specifications:

- Voltage Range: Rated $\pm 15\%$
- Frequency Range: 47-63Hz
- Efficiency: 74%
- AC Current: 6.5A/115V 4A/230V
- Inrush Current: 25A/115V 50A/230V
- Leakage Current: $<3.5\text{mA}/240\text{VAC}$

11.7 Jumper Cables



Fig 11.7 – Jumper Cables

CHAPTER – 12

CODE SNIPPETS

12.1 Program to Check the Mic Connection

```
#####  
# Mic Device Finder  
#####  
#  
import pyaudio  
audio = pyaudio.PyAudio() # start pyaudio device  
for indx in range(audio.get_device_count()):  
    dev = audio.get_device_info_by_index(indx) # get device  
    if dev['maxInputChannels']==4 and \  
        len([ii for ii in dev['name'].split('-') if ii=='4mic'])≥1:  
        print('-'*30)  
        print('Found Mic!')  
        print('Device Index: {}'.format(indx)) # device index  
        print('Device Name: {}'.format(dev['name'])) # device name  
        print('Device Input Channels: {}'.format(dev['maxInputChannels'])) #
```

12.2.1 Microphone Array Visualization Test

```
import pyaudio,sys,time  
import matplotlib  
matplotlib.use('TkAgg')  
import numpy as np  
import matplotlib.pyplot as  
plt
```

12.2.2 Creating PyAudio Streaming Object

```
#####  
# pyaudio Streaming Object  
#####  
#  
def audio_dev_formatter():  
    stream = audio.open(format=pyaudio_format,rate=samp_rate,  
                        channels=chans,input_device_index=quadmic_indx,  
                        input=True,frames_per_buffer=CHUNK) # audio  
    stream.stop_stream() # stop streaming to prevent overloa  
    return stream
```

12.2.3 Grabbing Data from Buffer

```
#####  
# Grabbing Data from Buffer  
#####  
#  
def data_grabber():  
    stream.start_stream() # start data stream  
    channel_data = [[]]*chans # data array  
    [stream.read(CHUNK,exception_on_overflow=False) for ii in range(0,1)] # clears  
    bufffor frame in range(0,int(np.ceil((samp_rate*record_length)/CHUNK))):  
        if frame==0:  
            print('Recording Started...')  
            # grab data frames from buffer  
            stream_data = stream.read(CHUNK,exception_on_overflow=False)  
            data = np.frombuffer(stream_data, dtype=buffer_format) # grab data from buffer  
            for chan in range(chans): # loop through all channels  
                channel_data[chan] = np.append(channel_data[chan],  
                                                data[chan::chans]) # separate channels  
    print('Recording Stopped')  
    return channel_data
```

12.2.4 Functions for Plotting data

```
#####
# functions for plotting data
#####
#
def plotter():
    #####
    # ---- time series for all mics
    plt.style.use('ggplot') # plot formatting
    fig,ax = plt.subplots(figsize=(12,8)) # create figure
    ax.set_ylabel('Amplitude',fontsize=16) # amplitude label
    ax.set_ylim([-2**15,2**15]) # set 16-bit limits
    fig.canvas.draw() # draw initial plot
    ax_bgnd = fig.canvas.copy_from_bbox(ax.bbox) # get background
    lines = [] # line array for updating
    for chan in range(chans): # loop through channels
        chan_line, = ax.plot(data_chunks[chan],
                             label='Microphone {0:1d}'.format(chan+1)) # initial channel
    plot    lines.append(chan_line) # channel plot array
    ax.legend(loc='upper center',
              bbox_to_anchor=(0.5,-0.05),ncol=chans) # legend for mic labels
    fig.show() # show plot
    return fig,ax,ax_bgnd,lines
```

```
def plot_updater():
    #####
    # ---- time series and full-period FFT
    fig.canvas.restore_region(ax_bgnd) # restore background (for
    speed)
    for chan in range(chans):
        lines[chan].set_ydata(data_chunks[chan]) # set channel data
        ax.draw_artist(lines[chan]) # draw line
    fig.canvas.blit(ax.bbox) # blitting (for speed)
    fig.canvas.flush_events() # required for blitting
    return lines
```

12.2.5 Main Loop Program

```
#####
# Main Loop
#####
#
if __name__=="__main__":
    #####
    # Audio Formatting
    #####
    #
    samp_rate      = 16000 # audio sample rate
    CHUNK          = 4000 # frames per buffer reading
    buffer_format  = np.int16 # 16-bit for buffer
    pyaudio_format = pyaudio.paInt16 # bit depth of audio encoding

    audio = pyaudio.PyAudio() # start pyaudio device
    quadmic_indx,chans = indx_getter() # get QuadMic device index and
channels
    stream = audio_dev_formatter() # audio stream

    record_length = 0.1 # seconds to record
    data_chunks = data_grabber() # grab the data
    fig,ax,ax_bgnd,lines = plotter() # establish initial plot

    while True:
        data_chunks = data_grabber() # grab the data
        lines = plot_updater() # update plot with new data
```

All the programs are written using the Python3 Language. The Raspberry Pi runs on the latest Ubuntu 22.04 Headless OS with python 3.11 installed. Tensorflow generated “.h5” model is used behind the scenes to create the voice recognition module. The raspberry pi is connected with its Arduino slave to move the bot in any direction.

The Raspberry Pi sends serial signals in the form of bytes via the GPIO ports, which are preconfigured in analog configuration. Furthermore, a PID controller is used in order to maintain the loop gain of the servos.

12.3 Main Driver program (Master Side)

```
import serial
import time
import speech_recognition as sr

# Connect to Arduino
ser = serial.Serial('/dev/ttyACM0', 9600, timeout=1)
time.sleep(2)

# Define robot movements
forward = b'F'
backward = b'B'
left = b'L'
right = b'R'
up = b'U'
down = b'D'

# Initialize speech recognition
r = sr.Recognizer()

while True:
    with sr.Microphone() as source:
        print("Say something!")
        audio = r.listen(source)

    # Recognize speech
    try:
        command = r.recognize_google(audio)
        print("You said: " + command)

        # Move the robot based on the recognized command
        if 'forward' in command:
            ser.write(forward)
        elif 'backward' in command:
            ser.write(backward)
        elif 'left' in command:
            ser.write(left)
        elif 'right' in command:
            ser.write(right)
        elif 'up' in command:
            ser.write(up)
        elif 'down' in command:
            ser.write(down)
        else:
            print("Command not recognized")

    except sr.UnknownValueError:
        print("Could not understand audio")
    except sr.RequestError as e:
        print("Could not request results;
        {0}".format(e))
```

12.4 PID Program (Slave Side)

```
#include <PID_v1.h>
#include <Wire.h>
#include <Servo.h>

Servo steer_servo;

//Gyro Variables
float elapsedTime, time, timePrev;
int gyro_error=0;
float Gyr_rawX, Gyr_rawY, Gyr_rawZ;
float Gyro_angle_x, Gyro_angle_y;
float Gyro_raw_error_x, Gyro_raw_error_y;

//Acc Variables
int acc_error=0;
float rad_to_deg = 180/3.141592654;
float Acc_rawX, Acc_rawY, Acc_rawZ;
float Acc_angle_x, Acc_angle_y;
float Acc_angle_error_x, Acc_angle_error_y;

float Total_angle_x, Total_angle_y;

/*working variables for PID*/
unsigned long lastTime;
double Input, Output, Setpoint;
double Iterm, lastInput;
double kp, ki, kd;
int SampleTime = 1000; //1 sec
double outMin, outMax;
bool inAuto = false;

void setup()
{
    // put your setup code here, to run once:
    Wire.begin(); //begin the wire communication

    Wire.beginTransmission(0x68); //begin, Send the slave address (in this case 68)
    Wire.write(0x6B); //make the reset (place a 0 into the 6B register)
    Wire.write(0x00);
    Wire.endTransmission(true); //end the transmission

    //Gyro config
    Wire.beginTransmission(0x68); //begin, Send the slave address (in this case 68)
    Wire.write(0x1B); //We want to write to the GYRO_CONFIG register (1B hex)
    Wire.write(0x10); //Set the register bits as 00010000 (1000dps full scale)
    Wire.endTransmission(true); //End the transmission with the gyro

    //Acc config
    Wire.beginTransmission(0x68); //Start communication with the address found during search.
    Wire.write(0x1C); //We want to write to the ACCEL_CONFIG register
    Wire.write(0x10); //Set the register bits as 00010000 (+/- 8g full scale range)
    Wire.endTransmission(true);

    Serial.begin(9600); //Remember to set this same baud rate to the serial monitor
    time = millis(); //Start counting time in milliseconds
    //end of acc error calculation
```

```

/*Here we calculate the gyro data error before we start the loop
* I make the mean of 200 values, that should be enough*/
if(gyro_error==0)
{
    for(int i=0; i<200; i++)
    {
        Wire.beginTransmission(0x68);           //begin, Send the slave address (in this case 68)
        Wire.write(0x43);                       //First address of the Gyro data
        Wire.endTransmission(false);
        Wire.requestFrom(0x68,4,true);          //We ask for just 4 registers

        Gyr_rawX=Wire.read()<<8|Wire.read();    //Once again we shif and sum
        Gyr_rawY=Wire.read()<<8|Wire.read();

        /*---X---*/
        Gyro_raw_error_x = Gyro_raw_error_x + (Gyr_rawX/32.8);
        /*---Y---*/
        Gyro_raw_error_y = Gyro_raw_error_y + (Gyr_rawY/32.8);
        if(i==199)
        {
            Gyro_raw_error_x = Gyro_raw_error_x/200;
            Gyro_raw_error_y = Gyro_raw_error_y/200;
            gyro_error=1;
        }
    }
} //end of gyro error calculation
steer_servo.attach(9);
}

void loop()
{
    // put your main code here, to run repeatedly:
    accelerometer();
}

void accelerometer()
{
    timePrev = time;                          // the previous time is stored before the actual time read
    time = millis();                          // actual time read
    elapsedTime = (time - timePrev) / 1000;    //divide by 1000 in order to obtain seconds

    //////////////////////////////////////Gyro read////////////////////////////////////

    Wire.beginTransmission(0x68);             //begin, Send the slave address (in this case 68)
    Wire.write(0x43);                         //First address of the Gyro data
    Wire.endTransmission(false);
    Wire.requestFrom(0x68,4,true);            //We ask for just 4 registers

    Gyr_rawX=Wire.read()<<8|Wire.read();       //Once again we shif and sum
    Gyr_rawY=Wire.read()<<8|Wire.read();

    /*Now in order to obtain the gyro data in degrees/seconds we have to divide first
    the raw value by 32.8 because that's the value that the datasheet gives us for a 1000dps range*/
    /*---X---*/
    Gyr_rawX = (Gyr_rawX/32.8) - Gyro_raw_error_x;
    /*---Y---*/
    Gyr_rawY = (Gyr_rawY/32.8) - Gyro_raw_error_y;

    Gyro_angle_x = Gyr_rawX*elapsedTime;

    Gyro_angle_y = Gyr_rawY*elapsedTime;

    Wire.beginTransmission(0x68);             //begin, Send the slave address (in this case 68)
    Wire.write(0x3B);                         //Ask for the 0x3B register- correspond to AcX
    Wire.endTransmission(false);              //keep the transmission and next
    Wire.requestFrom(0x68,6,true);            //We ask for next 6 registers starting withj the 3B
    Acc_rawX=(Wire.read()<<8|Wire.read())/4096.0 ; //each value needs two registres
    Acc_rawY=(Wire.read()<<8|Wire.read())/4096.0 ;
    Acc_rawZ=(Wire.read()<<8|Wire.read())/4096.0 ;
    /*Now in order to obtain the Acc angles we use euler formula with acceleration values
    after that we substract the error value found before*/
    /*---X---*/
    Acc_angle_x = (atan((Acc_rawY)/sqrt(pow((Acc_rawX),2) + pow((Acc_rawZ),2)))*rad_to_deg) -
    Acc_angle_error_x;
    /*---Y---*/
    Acc_angle_y = (atan(-1*(Acc_rawX)/sqrt(pow((Acc_rawY),2) + pow((Acc_rawZ),2)))*rad_to_deg) -
    Acc_angle_error_y;

    Total_angle_x = 0.98 *(Total_angle_x + Gyro_angle_x) + 0.02*Acc_angle_x;

    Total_angle_y = 0.98 *(Total_angle_y + Gyro_angle_y) + 0.02*Acc_angle_y;
}

```



```

void Compute()
{
    if(!inAuto) return;
    unsigned long now = millis();
    int timeChange = (now - lastTime);
    if(timeChange ≥ SampleTime)
    {
        /*Compute all the working error variables*/
        double error = Setpoint - Input;
        ITerm += (ki * error);
        if(ITerm > outMax) ITerm = outMax;
        else if(ITerm < outMin) ITerm = outMin;
        double dInput = (Input - lastInput);

        /*Compute PID Output*/
        Output = kp * error + ITerm - kd * dInput;
        if(Output > outMax) Output = outMax;
        else if(Output < outMin) Output = outMin;

        /*Remember some variables for next time*/
        lastInput = Input;
        lastTime = now;
    }
}

void SetTunings(double Kp = 10, double Ki = 0, double Kd = 0)
{
    double SampleTimeInSec = ((double)SampleTime)/1000;
    kp = Kp;
    ki = Ki * SampleTimeInSec;
    kd = Kd / SampleTimeInSec;
}

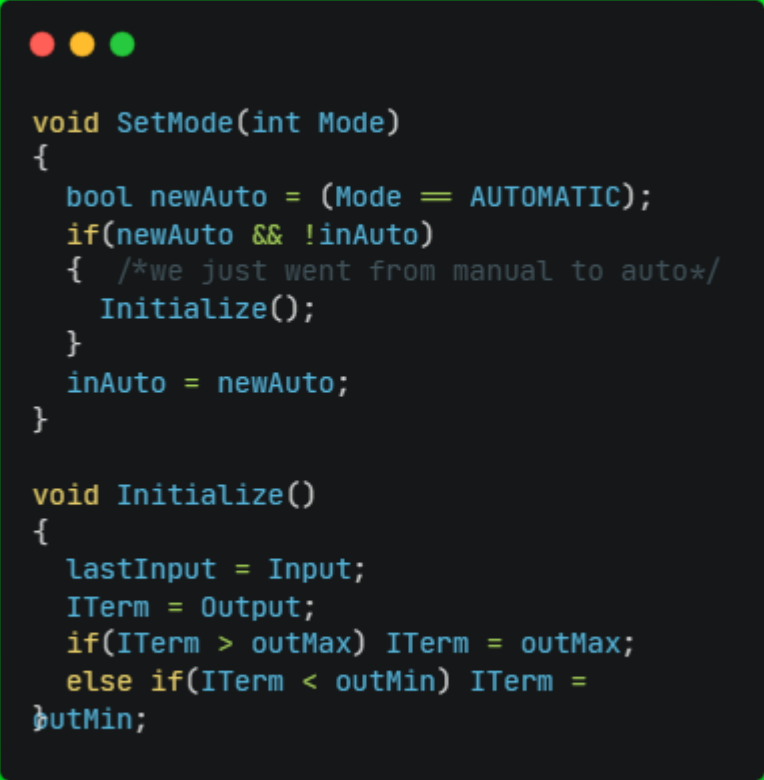
void SetSampleTime(int NewSampleTime)
{
    if (NewSampleTime > 0)
    {
        double ratio =
        (double)NewSampleTime/((double)SampleTime;
        kd /= ratio;
        SampleTime = (unsigned long)NewSampleTime;
    }
}

void SetOutputLimits(double Min, double Max)
{
    if(Min > Max) return;
    outMin = Min;
    outMax = Max;

    if(Output > outMax) Output = outMax;
    else if(Output < outMin) Output = outMin;

    if(ITerm > outMax) ITerm = outMax;
    else if(ITerm < outMin) ITerm = outMin;
}

```



```
void SetMode(int Mode)
{
    bool newAuto = (Mode == AUTOMATIC);
    if(newAuto && !inAuto)
    { /*we just went from manual to auto*/
        Initialize();
    }
    inAuto = newAuto;
}

void Initialize()
{
    lastInput = Input;
    ITerm = Output;
    if(ITerm > outMax) ITerm = outMax;
    else if(ITerm < outMin) ITerm =
outMin;
```

CHAPTER – 13

CONCLUSION

The project of an autonomous voice navigated 6-dof robot arm has the potential to revolutionize the field of robotics and automation. This project involves integrating speech recognition technology with a robotic arm to enable it to perform complex tasks with ease. The idea of controlling a robot arm with voice commands is not new, but this project takes it a step further by making the robot autonomous. The key components of this project include a Raspberry Pi, an Arduino board, and a 6-dof robot arm. The Raspberry Pi acts as the central processing unit and controls the overall functioning of the system. The Arduino board, on the other hand, is used to interface with the robot arm and send commands to it. The robot arm is designed to perform a variety of tasks, such as pick and place operations, welding, painting, and more. The voice navigation feature adds a new dimension to its functionality, making it easier to control and operate. One of the main advantages of an autonomous voice navigated robot arm is that it can be used in hazardous environments, such as factories or nuclear power plants. In such environments, human intervention is risky and can lead to accidents. With an autonomous robot arm, these risks can be minimized, and the overall efficiency of the process can be improved. Another advantage of this project is its potential to make robotics accessible to a wider audience. With voice commands, even those who do not have a technical background can operate and control a robot arm. This could lead to new applications of robotics in fields such as healthcare, education, and entertainment. In conclusion, the autonomous voice navigated 6-dof robot arm project is an innovative application of robotics and automation technology. It has the potential to revolutionize the way we interact with robots and make them more accessible to a wider audience. With further research and development, this project can lead to new breakthroughs in the field of robotics and automation.

REFERENCES

1. John, J. C. (2005). Introduction to Robotics. United State of America, USA: Pearson Education, Inc
2. Technology subjects support service. (2006). Applied Control Technology: Introduction to robotics. Retrieved from <http://www.t4.ie/resources/resources%20by%20Topic/option/applied%20control%20systems/Robotics/introduction%20to%20Robotics.pdf>
3. Steve, H. (2003). Embedded system design. London: Elsevier science
- ROBOINDIA. (2014). Servo Based 5 Axis Robotic Arm Retrieved from <https://www.slideshare.net/mobile/ROBO-INDIA/servo-base-5-axis-robotic-arm-project-report>.
4. D. Katagami, S. Yamada, "Active Teaching for an Interactive Learning Robot", Proceedings of IEEE international Workshop on Robot and Human Interactive Communication Millbrae. California. USA. Oct. 31 - Nov. 2, 2012.
5. Mohd Ashiq Kamaril Yusoffa, Reza Ezuan Saminb, Babul Salam Kader Ibrahimc, "Wireless Mobile Robotic Arm", International Symposium on Robotics and Intelligent Sensors 2012 (IRIS 2012), July 2012.
6. Chanhun Park and Kyoungtaik Park (2013), "Design and Kinematics Analysis of Dual Arm Robot Manipulator for Precision Assembly" IEEE, pp 431- 435.
7. Ayokunle A. Awelewa; Kenechukwu C. Mbanisi, Samuel O. Majekodunmi; Ishioma A (2012), "Development of a Prototype Robot Manipulator for Industrial Pick-and-Place Operations", International Journal of Mechanical & Mechatronics Engineering, Vol: 13.
8. Mohd Ashiq Kamaril Yusoffa, Reza Ezuan Saminb, Babul Salam Kader Ibrahimc, "Wireless Mobile Robotic Arm", International Symposium on Robotics and Intelligent Sensors 2012 (IRIS 2012), July 2012.

9. Mohd Ashiq Kamaril Yusoffa, Reza Ezuan Saminb, Babul Salam Kader Ibrahimc, “Wireless Mobile Robotic Arm”, International Symposium on Robotics and Intelligent Sensors 2012 (IRIS 2012), July 2012.
10. Benson, C., 2013. Robot Arm Torque Tutorial, Retrieved on January 22, 2016 from: <http://www.robotshop.com/blog/en/robot-arm-torque-tutorial-7152>.
11. Chuy, O.Y., Collins, E.G., Shema, A., and Kopinsky, R. 2017. Using Dynamics to Consider Torque Constraints in Manipulator Planning with Heavy Loads, Revised August 30, 2017 from: <http://dynamicsystems.asmedigitalcollection.asme.org/article.aspx?articleid=2582841>
12. Emerich, M., 2007. Design of a six Degree-of-Freedom Articulated Robotic Arm for Manufacturing Electrochromic Nanofilms, Retrieved on October 19, 2016 <http://oldweb.sbc.edu/sites/default/files/Honors/MEmerich.pdf>
13. Farhan, A. S., 2013. Mechatronics Design of Motion Systems; Modeling, Control and Verification, International Journal of Mechanical & Mechatronics Engineering IJMME-IJENS, Vol: 13, No: 02, pp.1-17.
14. Lin, F., and Brandt, R.D., 1997. An Optimal Control Approach to Robust Control of Robot Manipulators, National Science Foundation, pp.1-19.
15. Pachaiyappan, S., Micheal B.M., Sridhar T., 2014. Design And Analysis Of An Articulated Robot Arm For Various Industrial Applications, IOSR Journal of Mechanical and Civil Engineering (IOSR-JMCE), e- ISSN: 2278- 1684, p-ISSN: 2320–334X, PP 42-53.
16. Shweta, P., and Sanjay, L., 2012. Position Control of Pick and Place Robotic Arm, EIE’s 2nd Intl’ Conf. Comp., Energy, Net., Robotics and Telecom.