(Established under Karnataka Act No. 16 of 2013)
100-ft Ring Road, Bengaluru– 560 085, Karnataka, India

5-DOF Arm Robot

6th Semester

Robotics Modeling and Control Project

(UE21EE342BC7)

PES University

Name	SRN	Contribution Level	Contribution
A Sanath Hoysala	PES1UG21EE002	High	Calculated DH Parameters, Inverse kinematics using NR method, Analytical method, Geometrical method, Arduino code
Aneesh CD	PES1UG21EE013	High	Calculated DH Parameters, Inverse kinematics using NR method, Analytical method, Geometrical method and python code, Arduino code

Contents

List of Figures

1 Introduction

Robotic arms are integral components in various industries, including manufacturing, healthcare, and research, due to their precision, versatility, and efficiency in performing repetitive and complex tasks. A 5-degree-of-freedom (DOF) robotic arm is a versatile device that can move and position its end effector in three-dimensional space through a combination of five rotational joints. Ths project aims to develop an efficient algorithm wth the help of inverse kinematics computations which will control the motors precisely and hence enable accurate positionings based on user defined target coordinates.

1.1 Problem Statement

1. Use the Denavit-Hartenberg (DH) parameters to define the robotic arm's geometry.
2. Calculate the Forward kinematics of the robot using DH Parameters.
3. Calculate the Inverse kinematics of the robot using the Forward kinematics equations to get the joint angles.
4. Implement the Arduino code to control the stepper motors and servos of the robotic arm based on the received joint angles.
5. Integrate the inverse kinematics calculations with the Arduino control system.
6. Ensure precise movement and positioning of the robotic arm to reach the target coordinates as calculated by the Inverse kinematics.

1.2 Scope

The scope of this project encompasses the design, development, and testing of a control system for a 5-degree-of-freedom (DOF) robotic arm. The project includes the following key areas:

- Inverse Kinematics Computation
    1. Algorithm Development
    2. Validation

- System Integration
    1. Python Script for Computation
    2. Serial Communication
- Motor Control
    1. Hardware Setup
    2. Arduino Programming

2 Technical Details

2.1 Block Diagram

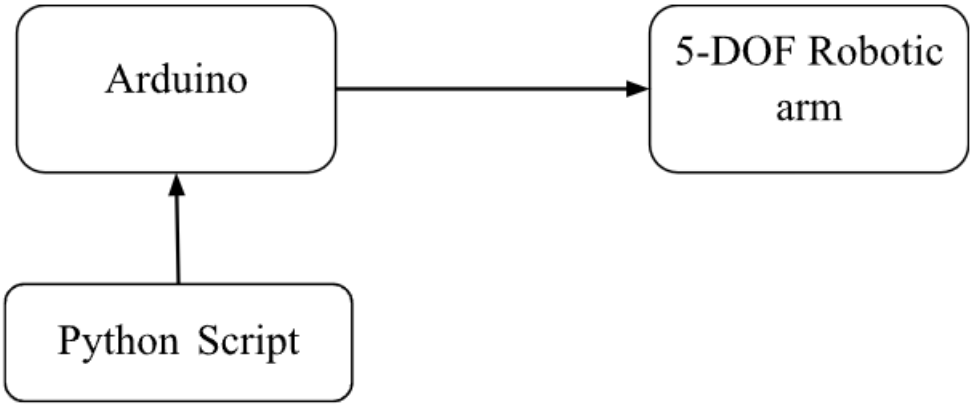


Figure 2.1: Block Diagram

2.1.1 Components

- Robotic Arm

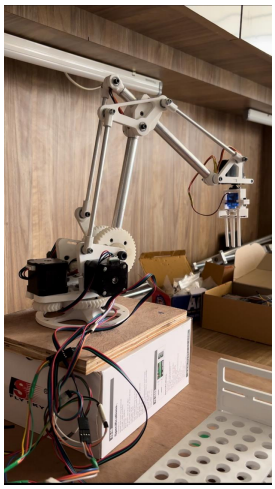


Figure 2.2: Image of 5-DOF Arm Robot

- Arduino : Acts as the main control unit for the robotic arm. It receives commands from the Python script via serial communication and controls the motors accordingly.

- CNC Shield : A CNC (Computer Numerical Control) shield is an expansion board for the Arduino microcontroller, designed to make it easier to control stepper motors.

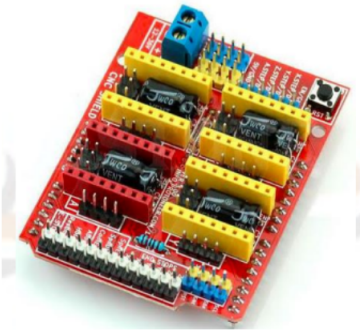


Figure 2.3: CNC Shield [1]

- Motor Drivers (2A Pololu RepRap Stepper motor Driver): Controls the current sent to the stepper motors, enabling precise control of the motor's speed, direction, and position.

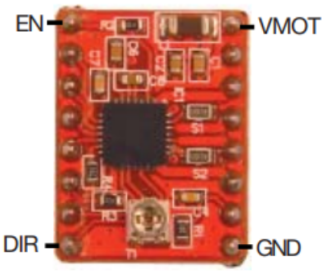


Figure 2.4: 2A Pololu RepRap Stepper motor Driver [2]

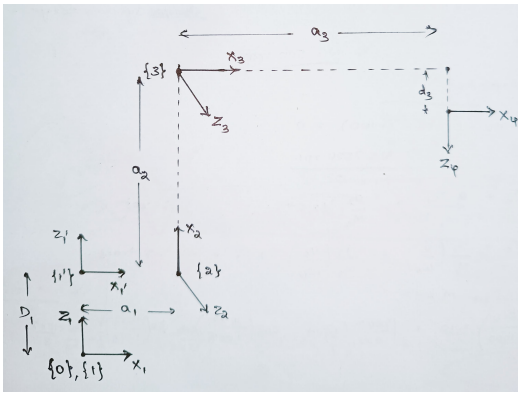
- Stepper Motors: Precisely control the angular position of the robotic arm's joints and enable smooth and accurate movements for the arm's positioning.

Figure 2.5: Stepper Motor [3]

- Servo Motors: Controls the end effector's movements such as gripping and releasing object and provide fine control over small angles, useful for delicate tasks.

Figure 2.6: Servo Motor [4]

2.2 Denavit-Hartenberg (DH) Parameters



Link	α	a	d	θ
1	0	0	0	θ_1
1'	0	0	0	d_1
2	$\frac{\pi}{2}$	a_1	0	$\theta_2 + \frac{\pi}{2}$
3	0	a_2	0	$\theta_3 - \pi/2$
4	$\frac{\pi}{2}$	a_3	d_3	0

Figure 2.7: DH Parameters

Here 1' is taken as the intermediate frame while moving from 1 to 2 as the translation cannot happen directly. 4th frame is the end effector's frame. The directions of the x and z axes followed while calculating the DH Parameters are as shown and the below table shows the DH Parameters obtained for the robot shown in Figure:**??**.
The measurements of the links are as shown:

- d1 = 62.5 mm
- a3 = 170.35097 mm
- a2 = 178.45003439 mm
- d3 = 58.409 mm
- a1 = 41.20509591 mm

3 Kinematics

3.1 Forward Kinematics

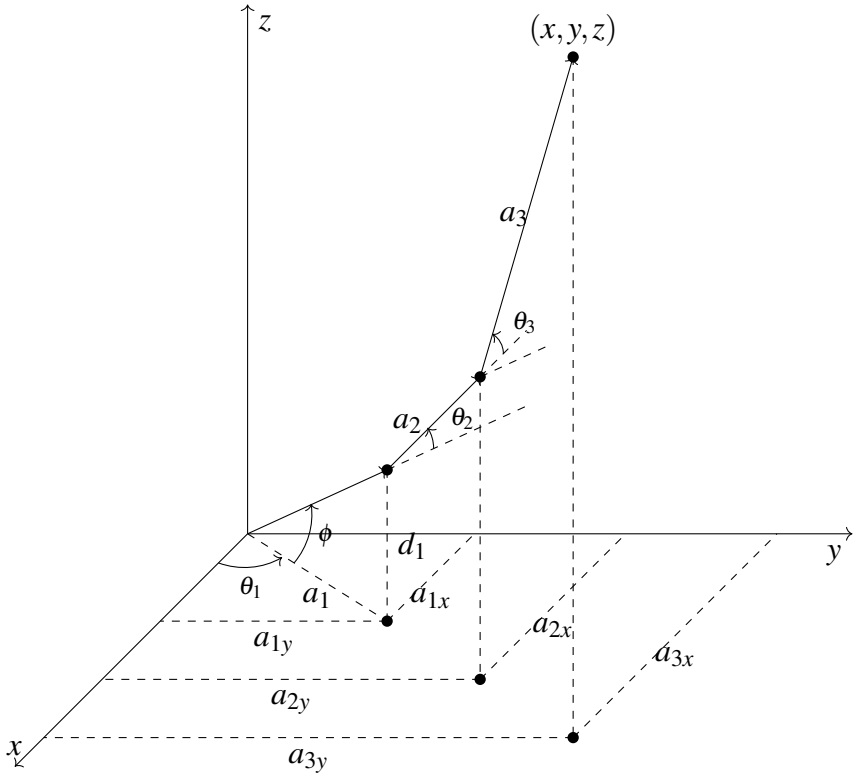


Figure 3.1: Plot used to calculate forward kinematics

Equations:

$$x = \cos(\theta_1)\left(a_1 + a_2\cos(\theta_2+\phi) + a_3\cos(\theta_2+\theta_3+\phi)\right) \tag{3.1}$$

$$y = \sin(\theta_1)\left(a_1 + a_2\cos(\theta_2+\phi) + a_3\cos(\theta_2+\theta_3+\phi)\right) \tag{3.2}$$

$$z = d_1 + a_2\sin(\theta_2+\phi) + a_3\sin(\theta_2+\theta_3+\phi) \tag{3.3}$$

After simplifying the above equations we arrive at the following set of equations representing the inverse kinematics.

3.2 Inverse Kinematics

3.2.1 Analytical Method

$$\theta_1 = \tan^{-1}(y,x) \tag{3.4}$$

$$\beta = \frac{y}{\sin(\theta_1)} - a_1 - d_3\cos(\alpha) \tag{3.5}$$

$$\gamma = z - d_1 \tag{3.6}$$

$$K = \frac{a_2}{2a_3} + \frac{a_2}{3a_3^2} \tag{3.7}$$

$$L = \frac{\beta}{a_3^2} \tag{3.8}$$

$$M = \frac{\beta^2}{2a_2a_3^2} + \frac{\gamma^2}{2a_2a_3} - \frac{a_2}{2a_3} - \frac{a_3}{2a_2} \tag{3.9}$$

$$S = 4a_2^2(a_3^2 - \beta^2) \tag{3.10}$$

$$T = 4(a_2^2\beta^2 - (a_2a_3)^2 - a_2^4) \tag{3.11}$$

$$U = 8\beta a_2^3 \tag{3.12}$$

$$V = 4a_2^4 \tag{3.13}$$

$$A = K^2 - V \tag{3.14}$$

$$B = U - 2KL \tag{3.15}$$

$$C = 2KM + L^2 - T \tag{3.16}$$

$$D = 2LM \tag{3.17}$$

$$E = M^2 - S \tag{3.18}$$

$$p_1 = 2C^2 - 9BCD + 27AD^2 + 27EB^2 - 72ACE \tag{3.19}$$

$$p_2 = p_1 + \sqrt{-4(C^2 - 3BD + 12AE)^3 + p_1^2} \tag{3.20}$$

$$p_3 = \frac{(C^2 - 3BD + 12AE)}{3A\left(\frac{p_2}{2}\right)^{1/3}} + \frac{\left(\frac{p_2}{2}\right)^3}{3A} \tag{3.21}$$

$$p_4 = \sqrt{\frac{B^2}{4A^2} - \frac{2C}{3A} + p_3} \tag{3.22}$$

$$p_5 = \frac{B^2}{2A^2} - \frac{4C}{3A} - p_3 \tag{3.23}$$

$$p_6 = \frac{-\frac{B^3}{A^3} + \frac{4BC}{A^2} - \frac{8D}{A}}{4p_4} \tag{3.24}$$

$$s_1 = -\frac{B}{4A} - \frac{p_4}{2} - \frac{\sqrt{p_5 - p_6}}{2} \tag{3.25}$$

$$s_2 = -\frac{B}{4A} - \frac{p_4}{2} + \frac{\sqrt{p_5 - p_6}}{2} \tag{3.26}$$

$$s_3 = -\frac{B}{4A} + \frac{p_4}{2} - \frac{\sqrt{p_5 - p_6}}{2} \tag{3.27}$$

$$s_4 = -\frac{B}{4A} + \frac{p_4}{2} + \frac{\sqrt{p_5 - p_6}}{2} \tag{3.28}$$

(3.29)

$$\theta_{2_1} = \left|\cos^{-1}(s_1)\right| \tag{3.30}$$

$$\theta_{2_2} = \left|\cos^{-1}(s_2)\right| \tag{3.31}$$

$$\theta_{2_3} = \left|\cos^{-1}(s_3)\right| \tag{3.32}$$

$$\theta_{2_4} = \left|\cos^{-1}(s_4)\right| \tag{3.33}$$

$$\theta_3 = \cos^{-1}\left(\frac{\beta - a_2\theta_2}{a_3}\right) \tag{3.34}$$

3.2.2 Geometrical Method

In this second method we will be using another approach to get the inverse kinematics equations.

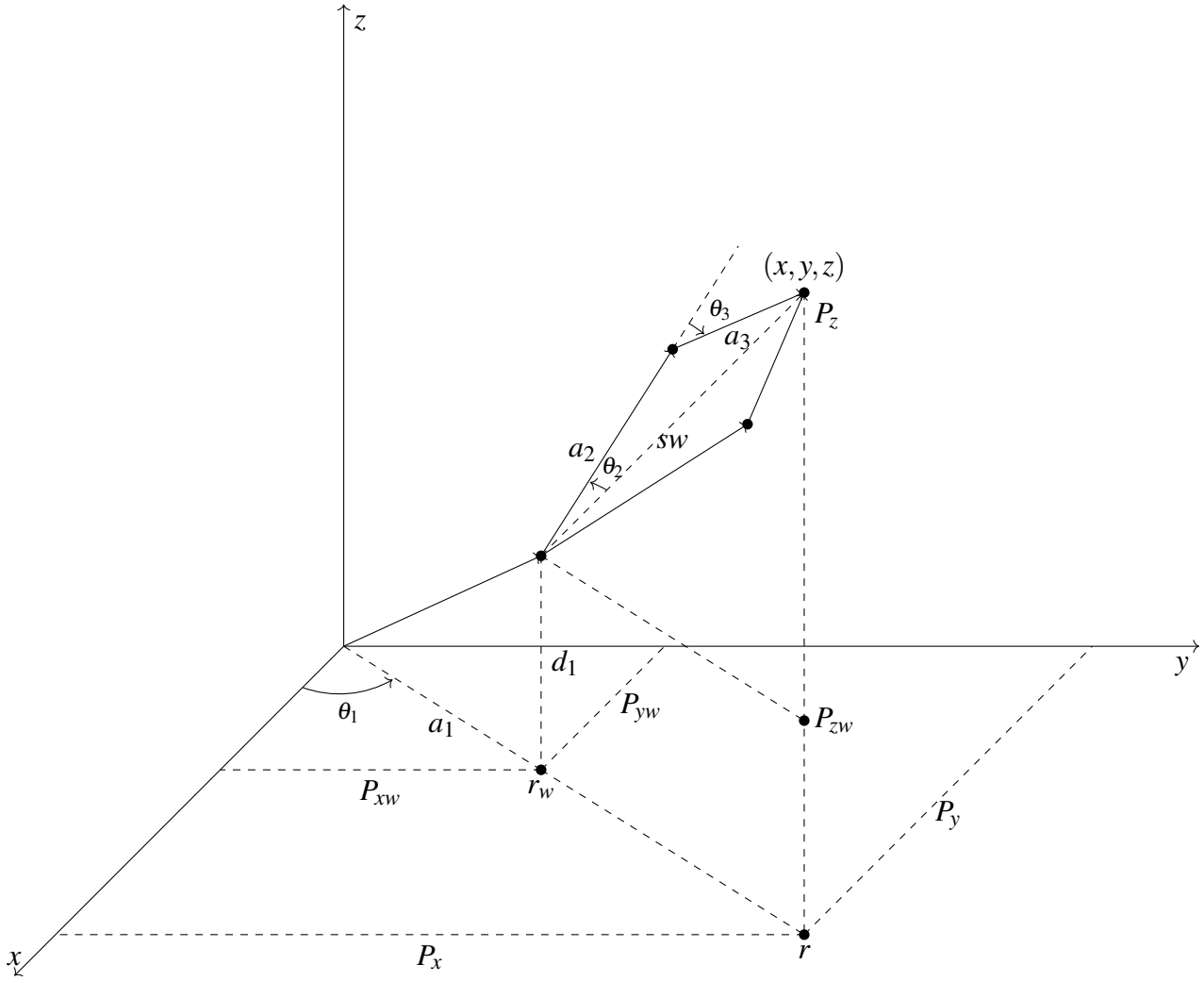


Figure 3.2: Plot to get inverse kinematics equations in Geometrical Method

The following equations are used to find the inverse kinematics. Here is $x = P_x$, $y = P_y$ and $z = P_z$.

$$r = \sqrt{P_x^2 + P_y^2} \tag{3.35}$$

$$r_w = a_1 = \sqrt{P_{xw}^2 + P_{yw}^2} \tag{3.36}$$

$$S_w = \sqrt{(P_z - d_1)^2 + (r - r_w)^2} \tag{3.37}$$

$$\theta_{3_1} = \cos^{-1}\left(\frac{sw^2 - a_2^2 - a_3^2}{2a_2a_3}\right) \tag{3.38}$$

$$\theta_{3_2} = -\cos^{-1}\left(\frac{sw^2 - a_2^2 - a_3^2}{2a_2a_3}\right) \tag{3.39}$$

$$\theta_2 = \cos^{-1}\left(\frac{sw^2 + a_2^2 - a_3^2}{2a_2sw}\right) \tag{3.40}$$

3.2.3 Newton-Raphson Method

Inverse kinematics involves calculating the joint angles of a robotic arm to achieve a desired end-effector position. The Newton-Raphson method is an iterative numerical technique used to solve non-linear equations, making it suitable for inverse kinematics.

Given the set of non-linear equations representing the robotic arm's kinematics:

$$f_1(\theta_1, \theta_2, \theta_3) = \cos(\theta_1)\left(a_1 + a_2\cos(\theta_2 + \phi) + a_3\cos(\theta_2 + \theta_3 + \phi)\right) - x \tag{3.41}$$

$$f_2(\theta_1, \theta_2, \theta_3) = \sin(\theta_1)\left(a_1 + a_2\cos(\theta_2 + \phi) + a_3\cos(\theta_2 + \theta_3 + \phi)\right) - y \tag{3.42}$$

$$f_3(\theta_1, \theta_2, \theta_3) = d_1 + a_2\sin(\theta_2 + \phi) + a_3\sin(\theta_2 + \theta_3 + \phi) - z \tag{3.43}$$

We can apply the Newton-Raphson method to iteratively solve for the joint angles θ_1, θ_2, and θ_3. The general iterative formula for the Newton-Raphson method is:

$$\theta^{(k+1)} = \theta^{(k)} - \mathbf{J}^{-1}(\theta^{(k)})\mathbf{F}(\theta^{(k)}) \tag{3.44}$$

where:

- $\theta^{(k)} = [\theta_1^{(k)}, \theta_2^{(k)}, \theta_3^{(k)}]^T$ is the vector of joint angles at iteration k.
- $\mathbf{F}(\theta) = [f_1(\theta), f_2(\theta), f_3(\theta)]^T$ is the vector of the non-linear equations.
- $\mathbf{J}(\theta)$ is the Jacobian matrix of $\mathbf{F}$ with respect to θ.

By iterating this process, the joint angles can be adjusted to minimize the difference between the current end-effector position and the desired position, ultimately solving the inverse kinematics problem.

$$J = \begin{bmatrix} J_{11} & J_{12} & J_{13} \\ J_{21} & J_{22} & J_{23} \\ J_{31} & J_{32} & J_{33} \end{bmatrix} \tag{3.45}$$

$$J_{11} = -a_1 \sin(\theta_1) - a_2 \sin(\theta_1)\cos(\theta_2 + \phi) - a_3 \sin(\theta_1)\cos(\theta_2 + \theta_3 + \phi) \tag{3.46}$$

$$J_{12} = -a_2 \sin(\theta_2 + \phi)\cos(\theta_1) - a_3 \sin(\theta_2 + \theta_3 + \phi)\cos(\theta_1) \tag{3.47}$$

$$J_{13} = -a_3 \sin(\theta_2 + \theta_3 + \phi)\cos(\theta_1) \tag{3.48}$$

$$J_{21} = a_1 \cos(\theta_1) + a_2 \cos(\theta_1)\cos(\theta_2 + \phi) + a_3 \cos(\theta_1)\cos(\theta_2 + \theta_3 + \phi) \tag{3.49}$$

$$J_{22} = a_2 \cos(\theta_2 + \phi)\sin(\theta_1) - a_3 \cos(\theta_2 + \theta_3 + \phi)\sin(\theta_1) \tag{3.50}$$

$$J_{23} = a_3 \cos(\theta_2 + \theta_3 + \phi)\sin(\theta_1) \tag{3.51}$$

$$J_{31} = 0 \tag{3.52}$$

$$J_{32} = a_2 \cos(\theta_2 + \phi) + a_3 \cos(\theta_2 + \theta_3 + \phi) \tag{3.53}$$

$$J_{33} = a_3 \cos(\theta_2 + \theta_3 + \phi) \tag{3.54}$$

3.2.4 Using ikine_LM function from Peter Coorke Toolbox on Python

```
import serial
import time
from roboticstoolbox import *
from spatialmath import SE3
import numpy as np

# Define the robot's DH parameters
d1 = 62.5
a1 = 41.20509591
a2 = 178.45003439
a3 = 170.35097
d3 = 58.409

# Assigning DH
L1 = [
    RevoluteDH(d=0, a=0, alpha=0, qlim=np.array([0, 2*np.pi])),
    RevoluteDH(d=d1, a=0, alpha=0, qlim=np.array([0, 0])),
    RevoluteDH(d=0, a=a1, alpha=np.pi/2, qlim=np.array([11 * (np.pi/180), 68
    * (np.pi/180)])),
    RevoluteDH(d=0, a=a2, alpha=0, qlim=np.array([-57 * (np.pi/180), -8 * (
   np.pi/180)])),
```

```
    RevoluteDH(d=d3, a=a3, alpha=np.pi/2, qlim=np.array([-70 * (np.pi/180), -27 * (np.pi/180)]))
]

arm = SerialLink(L1)

# Desired position
x, y, z = 350, 300, 400

# Calculate inverse kinematics
req_position_arr = [x, y, z]
req_pose_arr = [[1, 0, 0, x], [0, 1, 0, y], [0, 0, 1, z], [0, 0, 0, 1]]
req_pose_se3 = SE3(req_pose_arr)

joint_angles = arm.ikine_LM(req_pose_se3).q

# Convert joint angles to degrees
joint_angles_deg = [angle * (180 / np.pi) for angle in joint_angles]

# Print the angles
print(joint_angles_deg)

# Open a serial connection to Arduino
arduino = serial.Serial('COM15', 9600)  # Adjust 'COM3' to your Arduino's port
time.sleep(2)  # Give some time for the connection to establish

# Send angles to Arduino
angles_str = ','.join(map(str, joint_angles_deg))
arduino.write(angles_str.encode())

# Close the serial connection
arduino.close()
```

4 Progress

Subsystem	Achieved / Not Achieved
DH Parameters	Achieved
Inverse kinematics using python code	Achieved
Getting the joint angles output from python	Achieved
Serial Connection between Python and Arduino	Achieved
Correct Inverse kinematics equations	Not achieved
Efficient control of robot for the given coordinates	Not achieved

Though we solved for inverse kinematics using 4 different methods none of them gave the correct output according to our forward kinematics equations. We wish to look into this in the future.

4.1 Problems Faced

- Deriving the inverse kinematics equations from the forward kinematics using analytical method was tedious as it involved many variables.
- Visualising the geometry in geometrical method was difficult.
- Establishing the seral communication from python to Arduino was generating errors which was later resolved.

4.2 Learning Experience

1. Explored different methods to find inverse kinematics.
2. Understood the working of CNC shield, motor driver and stepper motor with arduino.
3. Implementing serial communication between arduino and laptop through pyserial.
4. Learnt how to apply Newton-Raphson method to find inverse kinematics.

5 Future Work

1. To perform pick and place using camera as an input.
2. Making the control of the robot more accurate.
3. To calculate force and torque required by the limbs using backward propagation equations.
4. To implement trajectory and path planning.

6 References

1. CNC Shield for A4988 datasheet

2. 32395-MP, 2A Pololu ReRap Stepper motor driver datasheet

3. Stepper Motor Image Link

4. Servo motor image link