PROJECT KINEMATIC ANALYSIS, TRAJECTORY PLANNING AND WORKSPACE REPRESENTATION FOR 6-DOF ROBOT MANIPULATOR

Prepared for

The Society of Automotive Engineers

Collegiate Club Number - SAEICCBIS022

National Institute of Technology Karnataka

Prepared by

Pradeep Singh Solanki – 191ME259 (Project Mentor)

Sumanth N Hegde – 191ME284 (Project Mentor)

Vabilisetti Prakhya Viswa Vidya Nischayi – 201ME363

April 10, 2022

ABSTRACT

In the era of 4.0 technology, robotic arms are becoming more and more popular in modern industries. Today, every small and large industry uses the robotic manipulator to complete various tasks like picking and placing, welding process, painting, and material handling. Therefore, the research and simulation of robotic arms significantly improve the efficiency of using this tool in all sectors. The Unimation PUMA 560 serially linked robot manipulator was used as a basis because this robot manipulator is widely used in industry and academia. It has nonlinear and uncertain dynamic parameters serial link 6 degrees of freedom robot manipulator. The position and orientation of the robotic arm end-effector are generated by the Denavit Hartenburg method, and the inverse kinematics equations have been derived using analytical method. Cubic and quintic polynomial interpolations have been used to study trajectory planning, and the Monte Carlo method is proposed to analyze the robotic arm's workspace.

Table of Contents

SECTION – I INTRODUCTION 3
- Timeline 3
- Tools and Technologies 3
- Literature Review 3

SECTION – II CONCEPT DEVELOPMENT AND EVALUATION 4
- Methodology 4
- Results and Discussion 14
- Conclusion 14
- Future Scope 14

SECTION – I
INTRODUCTION

Timeline

Month	Week	Task Accomplished
Nov-21	Three	Introductory Meet – Project Start
Nov-21	Four	Completed MATLAB Onramp Course from MathWorks
Dec-21	One	Learnt Fusion 360 Design Software
Dec-21	Two	Learnt Fundamentals Of Robotics from NPTEL videos
Dec-21	Three	Design Of PUMA 560 Robot Arm
Dec-21	Four	Manual Code For Forward Kinematics
Jan-22	Two	Manual Code For Inverse Kinematics
Feb-22	One	Manual Code For Trajectory
Feb-22	Two	Manual Code For Workspace
Feb-22	Three	Dynamic Simulation Using Matlab Simulink
Mar-22	One	Documentation and Submission of Final Report

Tools and Technologies

Sl. No	Tools/Technologies Used	Remark
1.	Solidworks	Designing CAD Model Of Robotic Arm
2.	Matlab, Simulink, Robotic Toolbox	Manual Code For Forward and Inverse Kinematics, Trajectory, Workspace, Dynamic Simulation

Literature Review

Serdar Kucuk [3] carried out the kinematic analysis of stanford manipulator. Analytical and geometrical approaches have been used to determine the inverse kinematic equations. Xiaojie Zhao [5] focused on the application of cubic and quintic polynomial in the robotic arm trajectory planning and obtained the corresponding curves by MATLAB simualtion. Jinliang Luo [6] established the kinematic model of a 7 DOF robotic by DH method and figured out the workspace by Monte Carlo method in MATLAB environment. Similar approach has been used in the project to derive the equations and the resulting graphs were verified with the above papers.

SECTION – II
CONCEPT DEVELOPMENT AND EVALUATION

Methodology

Design

Programmable Universal Machine for assemble PUMA 560 robot manipulator is a industrial robotic arm developed by Victor Scheinman at pioneering robot company Unimation. The essence of the design is represented in three categories; 200, 500, and 700 series. The 200 series was used for the first robotic stereotactic brain biopsy in 1985. The 500 Series and can reach almost 2 meters up and is the more recognizable configuration. The 700 series is the largest of the group and was intended for assembly line, paint, and welding work. All designs consist of two main components: the mechanical arm and the control system. These are typically interconnected by one or two large multi-conductor cables. When two cables are used, one carries power to the servo motors and brakes while the second carries the position feedback for each joint back to the control system. Solidworks software has been used to do the 3D model of the robot arm and the dimensions for this have been taken from Figure [2]. It is then imported into simulink using simmechanics tool where the robot arm is controlled by changing the joint angles. Additionally, PID controllers could be used to look into the response time and to compare the input signal and part response.

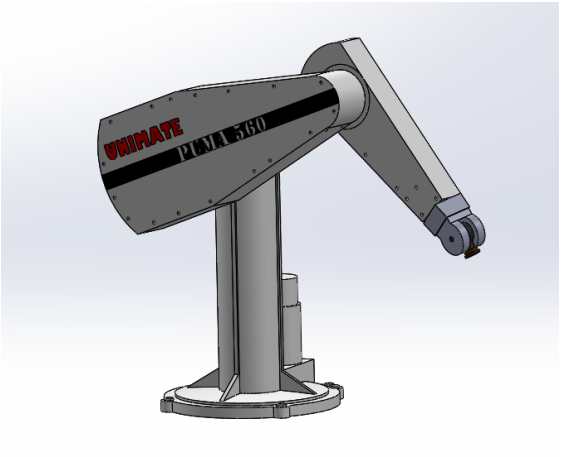

Figure 1: 3D Model of a PUMA 560 Robot Arm

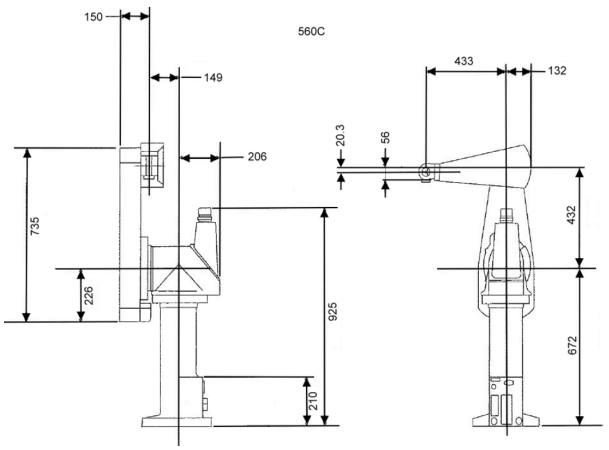

Figure 2: PUMA 560 manipulator segment measurements

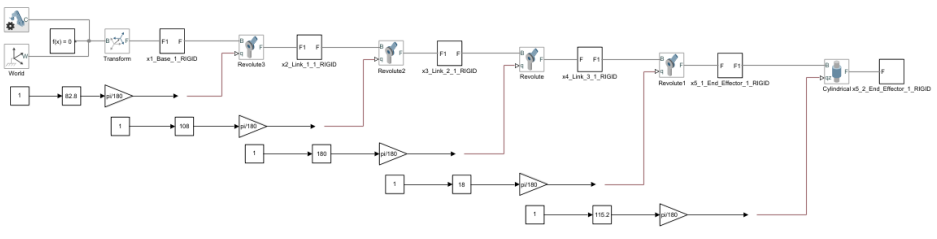

Figure 3: Simulink block diagram

Forward Kinematics

Kinematics is the science of motion that treats motion without regard to the forces which cause it. Within the science of kinematics, one studies position, velocity, acceleration, and all higher order derivatives of the position variables (with respect to time or any other variables. A manipulator is composed of serial links which are connected to each other by revolute or prismatic joints from the base frame through the end-effector. Calculating the position and orientation of the end-effector in terms of the joint variables is called as forward kinematics.

Denavit-Hartenberg (DH) kinematic model that uses four parameters is the most common method for describing the robot kinematics. These parameters a_{i-1}, α_{i-1}, d_i and θ_i are the link length, link twist, link offset and joint angle, respectively. A coordinate frame is attached to each joint to determine DH parameters. The distance from Z_{i-1} to Z_i measured along X_{i-1} is assigned as a_{i-1}, the angle between Z_{i-1} and Z_i measured along X_i is assigned as α_{i-1}, the distance from X_{i-1} to X_i measured along Z_i is assigned as d_i and the angle between X_{i-1} to X_i measured about Z_i is assigned as θ_i. The DH parameters for PUMA 560 robotic arm are listed below

Frame	Joint Angle	Twist Angle	Link Length	Link Offset
1	θ1	90	0	0
2	θ2	0	0.4318	0
3	θ3	-90	0.0203	0.15
4	θ4	90	0	0.4318
5	θ5	-90	0	0
6	θ6	0	0	0

Table 1: DH Parameters of PUMA 560 robot manipulator

The general homogeneous matrix consist the rotation and displacement vectors for a given joint angles. We define the homogeneous forward transformation matrix at each frame as

$$T_n^{n-1} = \begin{bmatrix} \cos\theta_n & -\sin\theta_n \cdot \cos\alpha_n & \sin\theta_n \cdot \sin\alpha_n & r_n\cos\theta_n \\ \sin\theta_n & \cos\theta_n \cdot \cos\alpha_n & -\cos\theta_n \cdot \sin\alpha_n & r_n\sin\theta_n \\ 0 & \sin\alpha_n & \cos\alpha_n & d_n \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The rotation and displacement matrix are given by,

$$R_n^{n-1} = \begin{bmatrix} \cos\theta_n & -\sin\theta_n \cdot \cos\alpha_n & \sin\theta_n \cdot \sin\alpha_n \\ \sin\theta_n & \cos\theta_n \cdot \cos\alpha_n & -\cos\theta_n \cdot \sin\alpha_n \\ 0 & \sin\alpha_n & \cos\alpha_n \end{bmatrix} \quad d_n^{n-1} = \begin{bmatrix} P_x \\ P_y \\ P_z \end{bmatrix} = \begin{bmatrix} r_n\cos\theta_n \\ r_n\sin\theta_n \\ d_n \end{bmatrix}$$

The forward kinematics of the end-effector with respect to the base frame is determined by multiplying all of the T_n^{n-1} matrices.

$$T_n^0 = T_1^0 \cdot T_2^1 \cdots T_n^{n-1}$$

Matlab Code

```
%standard DH notation
% Joint Angles
J = [0,90,45,0,0,0];
% Twist Angles
T = [90,0,-90,90,-90,0];
% Link Length
r = [0,0.4318,0.0203,0,0,0];
% Link Offsets
d = [0,0,0.1500,0.4318,0,0];

Tr = eye(4);

for n = 1:6
    A = [cosd(J(n)) -sind(J(n))*cosd(T(n)) sind(J(n))*sind(T(n)) r(n)*cosd(J(n));
    sind(J(n)) cosd(J(n))*cosd(T(n)) -cosd(J(n))*sind(T(n)) r(n)*sind(J(n));
    0 sind(T(n)) cosd(T(n)) d(n);
    0 0 0 1];
    fprintf("\nA%d%d\n",n-1,n);
    dispn(A,4);
    Tr = Tr * A;
end
fprintf("\nTransformational matrix \n");
dispn(Tr,4);
Pos_X = Tr(1,4);
Pos_Y = Tr(2,4);
Pos_Z = Tr(3,4);
fprintf("\nPosition Of End Effector: (%.4f,%.4f,%.4f)\n",Pos_X,Pos_Y,Pos_Z);

function [] = dispn(X,N)
[I,J] =   size(X);
   K  = length(N);
if     K == 1
    N = N * ones(1,J);
elseif K ~= J
    disp('ERROR: length(N) must either be 1 or equal to the number of columns of X.')
    return
end

for i = 1:I
    string = '';
    for j = 1:J
        if X(i,j) >= 0
            string = [string,'   %.', num2str( N(j) ) ,'f'];
        else
            string = [string,'  %.' , num2str( N(j) ) ,'f'];
        end
    end
    disp( sprintf( string , X(i,:) ) )
end
end
```

Output

>> Forward_Kinematics

A01

0.9848	-0.0000	0.1736	0.0000
0.1736	0.0000	-0.9848	0.0000
0.0000	1.0000	0.0000	0.0000
0.0000	0.0000	0.0000	1.0000

A12

0.9816	-0.1908	0.0000	0.4239
0.1908	0.9816	-0.0000	0.0824
0.0000	0.0000	1.0000	0.0000
0.0000	0.0000	0.0000	1.0000

A23

0.9781	-0.0000	-0.2079	0.0199
0.2079	0.0000	0.9781	0.0042
0.0000	-1.0000	0.0000	0.1500
0.0000	0.0000	0.0000	1.0000

A34

0.9744	-0.0000	0.2250	0.0000
0.2250	0.0000	-0.9744	0.0000
0.0000	1.0000	0.0000	0.4318
0.0000	0.0000	0.0000	1.0000

A45

0.9703	-0.0000	-0.2419	0.0000
0.2419	0.0000	0.9703	0.0000
0.0000	-1.0000	0.0000	0.0000
0.0000	0.0000	0.0000	1.0000

A56

0.9659	-0.2588	0.0000	0.0000
0.2588	0.9659	-0.0000	0.0000
0.0000	0.0000	1.0000	0.0000
0.0000	0.0000	0.0000	1.0000

Transformational matrix

0.6047	-0.5483	-0.5776	0.2957
0.5768	0.8016	-0.1571	-0.1002
0.5492	-0.2381	0.8011	0.4878
0.0000	0.0000	0.0000	1.0000

Position Of End Effector: (0.2957,-0.1002,0.4878)

Verification

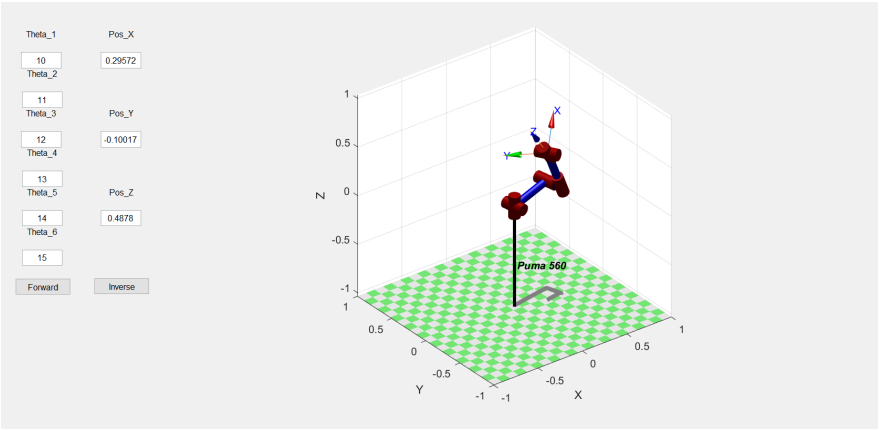

Figure 4: Forward Kinematics using robotic toolbox fkine function

Inverse Kinematics

Tasks to be performed by a manipulator are in the Cartesian space, whereas actuators work in joint space. Cartesian space includes orientation matrix and position vector. However, joint space is represented by joint angles. The conversion of the position and orientation of a manipulator end-effector from Cartesian space to joint space is called as inverse kinematics problem.

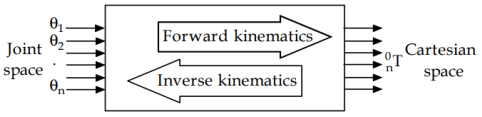

Figure 5: Schematic representation of forward and inverse kinematics

The main solution techniques for the inverse kinematics problem are the algebraic, geometric and numerical methods. For the manipulators with more links and whose arm extends into 3 dimensions the geometric approach gets much more tedious. Hence, in this project, only the algebraic solution of the manipulators is examined. To find the inverse kinematics solution for the first joint as a function of the known elements, the link transformation inverses are premultiplied as follows:

$$\left[{}^{0}_{1}T(q_1)\right]^{-1}\ {}^{0}_{6}T = {}^{1}_{2}T(q_2)\ {}^{2}_{3}T(q_3)\ {}^{3}_{4}T(q_4)\ {}^{4}_{5}T(q_5)\ {}^{5}_{6}T(q_6)$$

$$\left[{}^{0}_{1}T(q_1)\ {}^{1}_{2}T(q_2)\right]^{-1}\ {}^{0}_{6}T = {}^{2}_{3}T(q_3)\ {}^{3}_{4}T(q_4)\ {}^{4}_{5}T(q_5)\ {}^{5}_{6}T(q_6)$$

$$\left[{}^{0}_{1}T(q_1)\ {}^{1}_{2}T(q_2)\ {}^{2}_{3}T(q_3)\right]^{-1}\ {}^{0}_{6}T = {}^{3}_{4}T(q_4)\ {}^{4}_{5}T(q_5)\ {}^{5}_{6}T(q_6)$$

$$\left[{}^{0}_{1}T(q_1)\ {}^{1}_{2}T(q_2)\ {}^{2}_{3}T(q_3)\ {}^{3}_{4}T(q_4)\right]^{-1}\ {}^{0}_{6}T = {}^{4}_{5}T(q_5)\ {}^{5}_{6}T(q_6)$$

$$\left[{}^{0}_{1}T(q_1)\ {}^{1}_{2}T(q_2)\ {}^{2}_{3}T(q_3)\ {}^{3}_{4}T(q_4)\ {}^{4}_{5}T(q_5)\right]^{-1}\ {}^{0}_{6}T = {}^{5}_{6}T(q_6)$$

Matlab Code

```
format long
Pos_X = 0.29572222;
Pos_Y = -0.10017019;
Pos_Z =  0.48779716;
ox = -0.54832378;
oy = 0.80164032;
oz = -0.23814668;
ax = -0.57760124;
ay = -0.15710679;
az = 0.80105822;

% DH Parameters
r2 = 0.4318;
r3 = 0.0203;
d3 = 0.15;
d4 = 0.4318;

Soln_1 = inverse_kinematics(1,1,1,Pos_X,Pos_Y, Pos_Z, ox, oy, oz, ax, ay, az, r2, r3, d3, d4);
Soln_2 = inverse_kinematics(1,-1,1,Pos_X,Pos_Y, Pos_Z, ox, oy, oz, ax, ay, az, r2, r3, d3, d4);
Soln_3 = inverse_kinematics(1,1,-1,Pos_X,Pos_Y, Pos_Z, ox, oy, oz, ax, ay, az, r2, r3, d3, d4);
Soln_4 = inverse_kinematics(1,-1,-1,Pos_X,Pos_Y, Pos_Z, ox, oy, oz, ax, ay, az, r2, r3, d3, d4);
Soln_5 = inverse_kinematics(-1,1,1,Pos_X,Pos_Y, Pos_Z, ox, oy, oz, ax, ay, az, r2, r3, d3, d4);
Soln_6 = inverse_kinematics(-1,-1,1,Pos_X,Pos_Y, Pos_Z, ox, oy, oz, ax, ay, az, r2, r3, d3, d4);
Soln_7 = inverse_kinematics(-1,1,-1,Pos_X,Pos_Y, Pos_Z, ox, oy, oz, ax, ay, az, r2, r3, d3, d4);
Soln_8 = inverse_kinematics(-1,-1,-1,Pos_X,Pos_Y, Pos_Z, ox, oy, oz, ax, ay, az, r2, r3, d3, d4);

Soln = [Soln_1;Soln_2;Soln_3;Soln_4;Soln_5;Soln_6;Soln_7;Soln_8];

fprintf("Joint Angle Solutions\n");
for i = 1:8
    fprintf("Solution %d -> ",i);
    for j = 1:6
        fprintf("%.2f ",Soln(i,j));
    end
    fprintf("\n\n");
end

function Joint_Angle = inverse_kinematics(Number_1,Number_2,Number_3,Pos_X,Pos_Y, Pos_Z, ox, oy, oz, ax, ay, az, r2, r3, d3, d4)
D1 = sqrt(4*d3^2*Pos_X^2-4*(Pos_X^2+Pos_Y^2)*(-Pos_Y^2+d3^2));
A = (2*d3*Pos_X + Number_1*D1)/(2*(Pos_X^2+Pos_Y^2));
Theta_1 = asind(A);
Theta_1;

K3 = cosd(Theta_1)^2*Pos_X^2+sind(Theta_1)^2*Pos_Y^2+Pos_Z^2+2*cosd(Theta_1)*sind(Theta_1)*Pos_X*Pos_Y-r2^2-r3^2-d4^2;
D3 = sqrt((4*K3*r2*d4)^2+4*(4*r2^2*r3^2-K3^2)*(4*r2^2*d4^2+4*r2^2*r3^2));
C = (-4*K3*r2*d4 + Number_2*D3)/(2*(4*r2^2*d4^2+4*r2^2*r3^2));
Theta_3 = asind(C);
Theta_3;

D2 = sqrt((2*Pos_Z*(sind(Theta_3)*d4-cosd(Theta_3)*r3-r2))^2 - 4 * (Pos_Z^2-(cosd(Theta_3)*d4+sind(Theta_3)*r3)^2) * ((sind(Theta_3)*d4-cosd(Theta_3)*r3-r2)^2+(cosd
(Theta_3)*d4+sind(Theta_3)*r3)^2));
B = (-1*(2*Pos_Z*(sind(Theta_3)*d4-cosd(Theta_3)*r3-r2)) + Number_3*D2)/(2*((sind(Theta_3)*d4-cosd(Theta_3)*r3-r2)^2+(cosd(Theta_3)*d4+sind(Theta_3)*r3)^2));
Theta_2 = asind(B);
Theta_2;

Theta_23 = Theta_2 + Theta_3;

Theta_5 = acosd(-cosd(Theta_1)*sind(Theta_23)*ax-sind(Theta_1)*sind(Theta_23)*ay+cosd(Theta_23)*az);

Theta_4 = asind((sind(Theta_1)*ax - cosd(Theta_1)*ay)/sind(Theta_5));

Theta_6 = acosd(-1*(ox*(cosd(Theta_1)*cosd(Theta_23)*sind(Theta_4) + cosd(Theta_4)*sind(Theta_1)) - oy*(cosd(Theta_1)*cosd(Theta_4) - sind(Theta_1)*sind(Theta_4)*cosd
(Theta_23)) + oz*sind(Theta_4)*sind(Theta_23)));

Joint_Angle = [Theta_1,Theta_2,Theta_3,Theta_4,Theta_5,Theta_6];
end
```

Output

```
>> Inverse_Kinematics
Joint Angle Solutions
Solution 1 -> 47.43 50.14 21.35 -28.05 42.73 40.40

Solution 2 -> 47.43 62.88 15.96 -24.88 49.31 38.93

Solution 3 -> 47.43 21.11 21.35 -62.36 21.11 63.90

Solution 4 -> 47.43 13.77 15.96 -82.42 18.78 78.66

Solution 5 -> 10.00 69.62 12.00 4.41 45.05 25.22

Solution 6 -> 10.00 77.19 6.62 4.25 47.23 25.45

Solution 7 -> 10.00 11.00 12.00 13.00 14.00 15.00

Solution 8 -> 10.00 8.81 6.62 8.56 21.44 19.65
```

Figure 6: Joint Angle Solutions

Trajectory

On the basis of kinematics analysis, the trajectory planning and simulation of joint space was carried out. The term trajectory refers to the displacement, velocity and acceleration of the manipulator in motion. The task of the robotic arm trajectory planning is to design the motion function of each joint of the arm based on certain tasks to be accomplished by its end actuators. We can select different joint interpolation functions to generate different trajectories.

Cubic Polynomial Interpolation - The description of the trajectory can be represented by a smooth interpolation function of the start angle and end angle. The joint angle at the start point θ_0 and at the end point θ_f is defined. Time is assumed to vary from 0 to 1 in intermediate steps. The joint velocities at the boundaries is known and it is assumed to be zero in our case. By applying the boundary conditions to the cubic trajectory function $\theta(t)$, we can find the coefficients of the polynomial. Finally, the joint angles will be calculated at different time steps.

$$\theta(0) = \theta_0$$
$$\theta(t_f) = \theta_f$$
$$\dot{\theta}(0) = 0$$
$$\dot{\theta}(t_f) = 0$$

$$\theta(t) = a_0 + a_1 t + a_2 t^2 + a_3 t^3$$
$$\dot{\theta}(t) = a_1 + 2a_2 t + 3a_3 t^2$$

$$a_0 = \theta_0$$
$$a_1 = 0$$
$$a_2 = \frac{3}{t_f^2}(\theta_f - \theta_0)$$
$$a_3 = -\frac{2}{t_f^3}(\theta_f - \theta_0)$$

Quintic Polynomial Interpolation - In the case where the trajectory is more stringent and the constraint condition is increased, the cubic polynomial interpolation can't satisfy the requirement, and the high order polynomial is used for interpolation. For example, when the starting point and the ending point of a certain path are specified for the position, velocity and acceleration of their joints, a quintic polynomial can be used for interpolation. The joint velocities and acceleration is known and it is assumed to be zero in our case. The functional equation of the joint angle, velocity, acceleration, the boundary conditions and the calculated coefficients of the polynomial are shown below.

$$\theta(t_0) = \theta_0 \;\; \theta(t_f) = \theta_f$$
$$\dot{\theta}(t_0) = \ddot{\theta}(t_0) = \dot{\theta}(t_f) = \ddot{\theta}(t_f) = 0$$

$$\theta(t) = a_0 + a_1 t + a_2 t^2 + a_3 t^3 + a_4 t^4 + a_5 t^5$$
$$\dot{\theta}(t) = a_1 + 2a_2 t + 3a_3 t^2 + 4a_4 t^3 + 5a_5 t^4$$
$$\ddot{\theta}(t) = 2a_2 + 6a_3 t + 12a_4 t^2 + 20a_5 t^3$$

$$a_0 = \theta_0$$
$$a_1 = 0$$
$$a_2 = 0$$
$$a_3 = \frac{10(\theta_f - \theta_0)}{t_f^3}$$
$$a_4 = \frac{-15(\theta_f - \theta_0)}{t_f^4}$$
$$a_5 = \frac{6(\theta_f - \theta_0)}{t_f^5}$$

Matlab Code - Cubic Polynomial Interpolation

```
% DH Table - Joint Angle, Link Offset, Link Length, Twist Angle
L(1) = Link([0,0,0,pi/2]);
L(2) = Link([0,0,0.4318,0]);
L(3) = Link([0,0.15,0.0203,-pi/2]);
L(4) = Link([0,0.4318,0,pi/2]);
L(5) = Link([0,0,0,-pi/2]);
L(6) = Link([0,0,0,0]);
puma = SerialLink(L);
puma.name = "Puma 560";

Initial_theta = [0,0,0,0,0,0];
Final_theta = [60,80,100,70,50,120];

% timestep 0.10, time varies from 0 to 1 and it is divided to 10 intervals, joint angles will be computed at these intervals
Time = 0:0.1:1;
% joint space trajectory function 'joint_trajectory' takes initial and final theta, time as input parameters
% cubic (3rd order) polynomial is used with default zero boundary conditions for velocity and acceleration
% joint_trajectory computes the coefficients in cubic polynomial and then finds the joint angles at different time intervals
[Q,QD,QDD] = joint_trajectory(Initial_theta*pi/180,Final_theta*pi/180,Time);
% DH table info is taken from puma, joint angle info is taken from Q and finding out transformation matrix (forward kinematics) at all time intervals
Tr = fkine(puma,Q);

for i = 1:1:length(Time)
    T = Tr(i);
    % function 'transl' extracts the translational matrix(position of end effector) from a transformational matrix
    trs = transl(T);
    % assigning x, y, z coordinate to variable xx, yy, zz respectively
    xx(i) = trs(1);
    yy(i) = trs(2);
    zz(i) = trs(3);
end
```

```
function [] = display(puma,Q,QD,QDD,Time,xx,yy,zz)
    subplot(4,3,[1:9]);
    % plotting robot arm at the joint angle varies
    plot(puma,Q);
    hold on
    % plotting the trajectory
    plot3(xx,yy,zz,'Color',[1 0 0],'LineWidth',1);

    subplot(4,3,10);
    plot(Time,Q);
    xlabel('Time');
    ylabel('Angular Displacement');

    subplot(4,3,11);
    plot(Time,QD);
    xlabel('Time');
    ylabel('Angular Velocity');

    subplot(4,3,12);
    plot(Time,QDD);
    xlabel('Time');
    ylabel('Angular Acceleration');
end

function [Q,QD,QDD] = joint_trajectory(Initial_Theta,Final_Theta,Time)
    Q = zeros(length(Time),length(Initial_Theta));
    QD = zeros(length(Time),length(Initial_Theta));
    QDD = zeros(length(Time),length(Initial_Theta));

    for i = 1:length(Initial_Theta)
    tf = 1;
    a0 = Initial_Theta(i);
    a1 = 0;
    a2 = 3*(Final_Theta(i)-Initial_Theta(i))/tf^2;
    a3 = -2*(Final_Theta(i)-Initial_Theta(i))/tf^3;
    for j=1:length(Time)
        Joint_Angle = a0 + a1*Time(j) + a2*Time(j)^2 + a3*Time(j)^3;
        Joint_Velocity = a1 + 2*a2*Time(j) + 3*a3*Time(j)^2;
        Joint_Acceleration = 2*a2 + 6*a3*Time(j);
        Q(j,i) = Joint_Angle;
        QD(j,i) = Joint_Velocity;
        QDD(j,i) = Joint_Acceleration;
        end
    end
end
```

Output

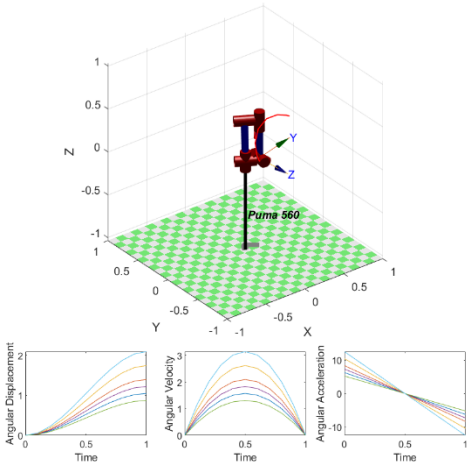

Figure 7: Cubic Polynomial Joint Trajectory Curves

Matlab Code - Quintic Polynomial Interpolation

```
function [Q,QD,QDD] = joint_trajectory(Initial_Theta,Final_Theta,Time)
    Q = zeros(length(Time),length(Initial_Theta));
    QD = zeros(length(Time),length(Initial_Theta));
    QDD = zeros(length(Time),length(Initial_Theta));

    for i =1:length(Initial_Theta)
        tf = 1;
        a0 = Initial_Theta(i);
        a1 = 0;
        a2 = 0;
        a3 = 10*(Final_Theta(i)-Initial_Theta(i))/tf^3;
        a4 = -15*(Final_Theta(i)-Initial_Theta(i))/tf^4;
        a5 = 6*(Final_Theta(i)-Initial_Theta(i))/tf^5;
        for j=1:length(Time)
            Joint_Angle = a0 + a1*Time(j) + a2*Time(j)^2 + a3*Time(j)^3 + a4*Time(j)^4 + a5*Time(j)^5;
            Joint_Velocity = a1 + 2*a2*Time(j) + 3*a3*Time(j)^2 + 4*a4*Time(j)^3 + 5*a5*Time(j)^4;
            Joint_Acceleration = 2*a2 + 6*a3*Time(j) + 12*a4*Time(j)^2 + 20*a5*Time(j)^3;
            Q(j,i) = Joint_Angle;
            QD(j,i) = Joint_Velocity;
            QDD(j,i) = Joint_Acceleration;
        end
    end
end
```

Output

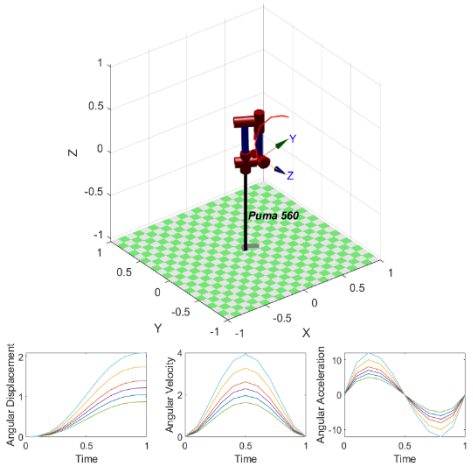

Figure 8: Quintic Polynomial Joint Trajectory Curves

Workspace

The work space of a kinematic structure is defined as the set of all points that it can reach in space. It is important to know the workspace of a kinematic structure, to be able to assess its flexibility and workability. Monte Carlo method, also called statistical simulation method, is a numerical method by means of random sampling to solve mathematical problems, and it is also widely used to describe certain random physical phenomena in engineering. This method is easy to implement graphical display function and calculates fast and simply, it can avoid the complicated mathematical derivation and calculation process. It is suitable for solving workspace of any joint type robotic arm without the limitation for the range of joint variables, and its error is independent of the dimension. Within the range of each joint variable value, N random values from 0 to 1 are generated by function rand. Taking N pseudo-random values of joint variables obtained into the kinematic positive solution equation, and figuring out the end point corresponding position vector of robotic arm.

$$\theta_i = \theta_{imin} + (\theta_{imax} - \theta_{imin}) \times rand(N, 1)$$

The joint limits of the PUMA 560 robot arm is given below:

Joint	θmin	θmax
1	-160	160
2	-45	225
3	-225	45
4	-110	170
5	-100	100
6	-266	266

Table 2: Joint Limits of PUMA 560 manipulator

Matlab Code

```
format long
% DH Table - Joint Angle, Link Offset, Link Length, Twist Angle
L(1) = Link([0,0,0,pi/2]);
L(2) = Link([0,0,0.4318,0]);
L(3) = Link([0,0.15,0.0203,-pi/2]);
L(4) = Link([0,0.4318,0,pi/2]);
L(5) = Link([0,0,0,-pi/2]);
L(6) = Link([0,0,0,0]);
puma = SerialLink(L);
puma.name = "Puma 560";

%Number of random coordinate points
N = 20000;
Angle_min = [-160, -45, -225, -110, -100, -266]*pi/180;
Angle_max = [160, 225, 45, 170, 100, 266]*pi/180;
[xx,yy,zz] = workspace(puma,Angle_min,Angle_max,N);
fprintf('Workspace Range \n x:[%.3f, %.3f] \n y:[%.3f, %.3f] \n z:[%.3f, %.3f] \n\n',min(xx),max(xx),min(yy),max(yy),min(zz),max(zz));

subplot(3,5,[1:2]);
plot3(xx,yy,zz,'.');
label_display();
title('Isometric View');
view(3);

subplot(3,5,[4:5]);
plot3(xx,yy,zz,'.');
label_display();
title('Top View (X-Y Plane)');
view(2);

subplot(3,5,[9:10])
plot3(xx,yy,zz,'.');
label_display();
title('Front View (X-Z Plane)');
view([0 1 0]);

subplot(3,5,[14:15])
plot3(xx,yy,zz,'.');
label_display();
title('Side View (Y-Z Plane)');
view([1 0 0]);
```

```
Angle_min(1) = 0*pi/180;
Angle_max(1) = 0*pi/180;
[xx,yy,zz] = workspace(puma,Angle_min,Angle_max,N);
subplot(3,5,[6:7])
plot3(xx,yy,zz,'.');
label_display();
title('Sectional View (X-Z Plane)');
view([0 1 0]);

Angle_min(1) = 90*pi/180;
Angle_max(1) = 90*pi/180;
[xx,yy,zz] = workspace(puma,Angle_min,Angle_max,N);
subplot(3,5,[11:12])
plot3(xx,yy,zz,'.');
label_display();
title('Sectional View (Y-Z Plane)');
view([1 0 0]);

function [xx,yy,zz] = workspace(puma, Angle_min,Angle_max,N)
for i = 1:6
    a(:,i)=Angle_min(i) + [Angle_max(i)-Angle_min(i)].*rand(N,1);
end
for j = 1:N
    Tr = fkine(puma,a(j,:));
    trs = transl(Tr);
    xx(j) = trs(1);
    yy(j) = trs(2);
    zz(j) = trs(3);
    %puma.plot(a(j,:))
end
end

function label_display()
xlabel('x');
ylabel('y');
zlabel('z');
grid on;
end
```

Output

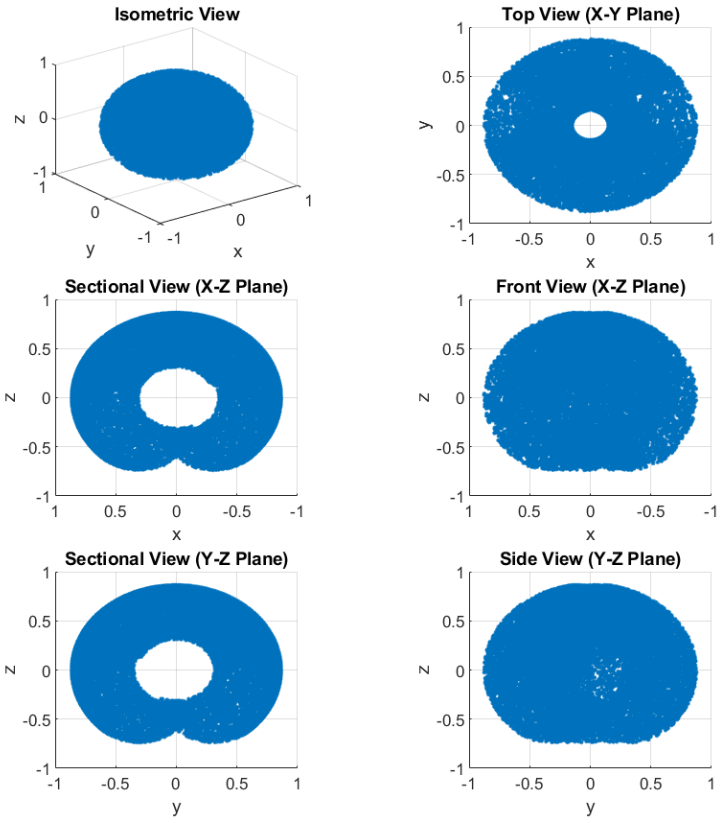

Figure 9: Workspace of PUMA 560 manipulator

Results and Discussion

In the forward kinematics, the manual code takes joint angles as the input and calculates the position of the end effector. The forward kinematics of the end-effector with respect to the base frame is determined by multiplying all the transformation matrix. From Figure [4], it can be seen that the position of the end effector obtained from the manual code matches with the results obtained using the inbuilt robotic toolbox 'fkine' function. Algebraic approach has been used to compute the inverse kinematic equations. For a given position of end effector, all the possible joint angle solutions have been listed in Figure [5].

Figure [7] and [8] shows the angular displacement, angular velocity and angular acceleration curve of each joint obtained by cubic and quintic polynomial respectively. It can be seen from the figures that the plots obtained are continuous, and the movement is more smoother in the case of quintic polynomial interpolation. It indicates that this method is feasible in the application of trajectory planning of similar 6 DOF robotic arm. This method can make the robotic arm to run smoothly, reduce the occurrence of jitter and vibration, improve the stability of system and prolong the life of robotic arm. The workspace of the robotic arm can be obtained from the Figure [9]. The accessible workspace of robotic arm consists of an approximate ellipsoid with $x \in [-0.874, 0.876]$m, $y \in [-0.875, 0.876]$m, $z \in [-0.737, 0.864]$m. The solution process adopted was concise and high-speed, providing the basis for obstacle avoidance planning. The simulated workspace range is accurate and closely matches with the actual workspace of robotic arm.

Conclusion

The manual codes for the robot kinematics, trajectory planning, and workspace were written in MATLAB, and the obtained results were verified using the inbuilt functions and the existing literature. The key takeaways by doing this project is that the project members got a better understanding on the concepts of robot kinematics and the software knowledge gained here will come in use if we are doing robotic simulation projects in the future.

Future Scope

The limitations of the project is that, higher order polynomial interpolation for trajectories provide peaks of velocity in the middle, which is not desirable for multiple point trajectory. As part of future work, trajectory planning can be done using optimization algorithms like PSO, GA and fuzzy PID based dynamic control system can be designed to have a more control over joint angles, velocity and acceleration. The inverse kinematic equations could also be evaluated using the geometric approach or by making use of Adaptive Neuro-Fuzzy Inference System (ANFIS). Furthermore, the position error between model's end-effector position that use angle from ANFIS solution and calculated end-effector's position could be analysed.

References:

- NPTEL Robotics by Dr. DK Pratihar, Dept of Mechanical Engineering, IIT Kharagpur
- Introduction To Robotics By John J. Craig, Third Edition
- Serdar Kucuk and Zafer Bingul. “Robot Kinematics: Forward and Inverse Kinematics”, Industrial Robotics: Theory, Modelling and Control
- Tarun Pratap Singh, Dr. P. Suresh, Dr. Swet Chandan, “Forward and Inverse Kinematic Analysis of Robotic Manipulators”, International Research Journal of Engineering and Technology Volume 04
- Xiaojie Zhao, Maoli Wang, Ning Liu, Yongwei Tang “Trajectory Planning for 6-DOF Robotic Arm Based on Quintic Polynormial”, International Conference on Control, Automation, and Artificial Intelligence (CAAI 2017)
- Jinliang Luo, Qun Wen, Jialai He, Bin Ye, “Workspace Analysis of 7-DOF Humanoid Robotic Arm”, Proceedings of the 2015 International Conference on Intelligent Systems Research and Mechatronics Engineering
- Essam Hashem, “Humanoid arm robot simulation on SIMULINK MATLAB”
- Transformation Matrix and Denavit Hartenburg Convention
- Peter Corke Robotic Toolbox
- https://en.wikipedia.org/wiki/Programmable_Universal_Machine_for_Assembly