Trajectory planning of end-effector to avoid obstacles while spot welding

Sai Manohar Vangara, saimanoh@buffalo.edu

Sai Manikanta Servirala, sservira@buffalo.edu

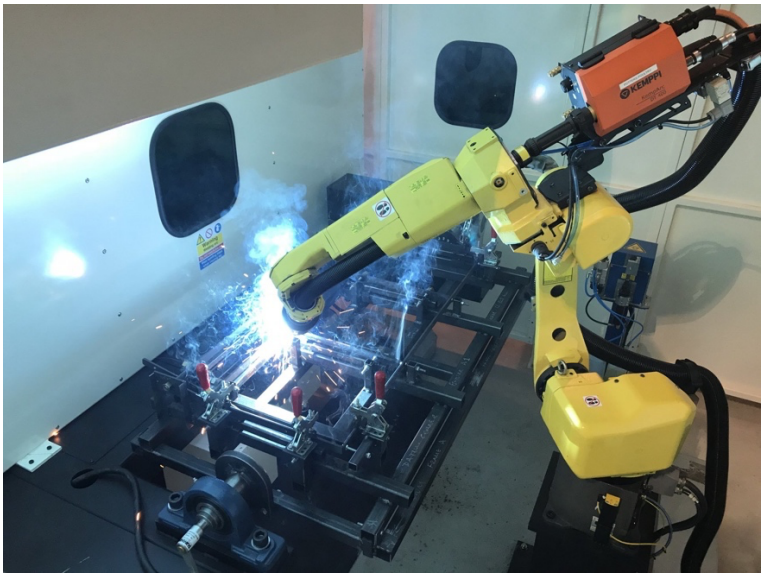
Abstract

Manufacturing automation is a process which automates the equipment used in the manufacturing process. One of the primary components used in manufacturing automation is a robotic arm for spot wielding. Controlling Robotic arm trajectory without jerks is a factor to be considered because the accurate moment of the arm results accurate end-product. So, modelling a robotic arm joint parameters is the first step. There are several kinematic operations called forward and inverse kinematics that can solve for joint parameters which define the moment in a robotic arm. Along with the efficient interpolation method for the trajectory planning the other factor that needs to considered is the safety of humans and objects around the robotic arm. Safety is the main issue while working with robotic arms which can bump into the objects/human if bought into the path of the robotic arm end-effector. These arises an obstacle avoidance problem. The main objective of the paper to provide the efficient trajectory planning method along with performing Bezier segmentation to avoidance the obstacle in the path. The task of trajectory planning, which involves the robot determining the best path to complete a specific task, has been a central focus in the field of robotics. Along with this it will also discuss and examine the trends of mean position, mean velocity, and mean acceleration characters of the robotic arm end-effector at different trajectory paths.

1 Introduction

Industrial robots have been used in automatic production lines in place of human workers for a decade. These robust machines are rapidly changing to autonomous machines, to complete specific duties. Such a machine is flexible how it operates and versatile in the range of duties it can complete, thanks to computer software to control it. However, getting the most out of a robot depends on how well the trajectory design is done. Trajectory planning and management are the two main stages that make up the process of programming industrial robots. A vital field of research in robotics and control systems is trajectory planning. It includes planning and carrying out a series of actions that a robot or other autonomous entity must carry out to advance toward its destination while avoiding obstacles and abiding by limitations. Finding a path in the configuration space that meets certain criteria, such as minimizing time, energy, or distance, can be defined as

the trajectory planning problem. In manufacturing and assembly, accuracy is the main factor to be considered along with the safety of the workers. Using a robotic arm for assembling activates is the new mode of manufacturing. When it comes to the accuracy robotic arms are unbeatable compared to what a human can achieve. So, automating robotic arm can put out the full potential of the equipment However, In the assemble line wielding is a complex job that is done by human due to the complex curve and bends along the parts that would take a great labor. To reduce the cost, increase accuracy and increase safety an autonomous robot is needed. To archive an autonomous path for a robotic arm trajectory planning is the way to achieve it.



There have been numerous techniques developed to ensure path planning avoids collisions, and these can be applied to both vehicles and mobile robots. Genetic algorithms have been successfully used to find the optimal path for mobile robots. Other techniques, such as visibility graphs and Bezier curves, have been used to construct smooth, collision-free trajectories, particularly for wheeled mobile robots. Specialized motion planning algorithms have been created for robot manipulators, yielding smoother and shorter paths. Curve-based methods are another solution for path planning. They generate a smooth, collision-free path between the starting and goal points amidst obstacles. Bezier and Hermite curves, which are types of parametric curves, that have proven particularly useful. When it comes to robotic arms, it is crucial to accurately determine the position, velocity, and acceleration during motion. When the arm is required to follow a predetermined path, there might be multiple trajectories.

Bezier curves and cubic splines have been a focus due to their simplicity and flexibility in path planning. However, conventional Bezier curves lack the flexibility to control the shape of the curve, which has led to the creation of modifiable Bezier curves that include shape parameters. Inverse kinematics has also been used to calculate the joint parameters needed to position the end of the robotic arm at a desired location in space. This is a mathematical technique that assists in

directing the motion and orientation of a robot arm. The objective is to enhance the proficiency and effectiveness of the robot's movement, minimize the steps taken from beginning to end, and ensure the pathway traversed is smooth and secure.

Literature review

There are several algorithms for the trajectory planning of the robot manipulator. The algorithm that can generate smooth and no jerks will be ideal for the wielding process. [1].Visioli A used simulations of a three-degree-of-freedom planar robot manipulator to verify the proposed technique. The outcomes demonstrate that the generated trajectories are smooth, continuous, and within the robot's limits. The findings show that the suggested method is better in terms of smoothness and accuracy when compared to other trajectory planning techniques.[2] Boryga M, Graboś proposed a method which is divided into two steps. In the first step, the starting and final positions, speeds, and accelerations of the manipulator are specified to describe the desired trajectory. The trajectory is produced in the second step using higher-degree polynomials. Several techniques for producing these polynomials are presented, including the use of the Bernstein basis, the Bezier curve, and the Chebyshev polynomial.[3] The Proposed method by Dyllong E, Visioli A is a modified method based on cubic splines which is used to modify the trajectory in real-time. The modified algorithm generates a new trajectory that complies with the robot's constraints and avoids obstacles by considering its current position, velocity, and intended final position. The method proposed by Dyllong E, Visioli A will be used in the project as it takes current position along with velocity which can be used for the robotic manipulators that can generate continuous, smooth trajectories which are the exact requirements for a wield to be good.

2 Method

The method involves three main stages which are design of the described path using different interpolation methods, creating an environment and finally simulation. The path's design was based on trajectory planning using two curve methods that are Cubic spline with Hermit Interpolation and Bezier curve along with Bezier segmentation for avoiding the object.

2.1 Design of the path

The Curve was designed to pass through P_1 [-2,0,1], P_2 [-1.5, -2, -1], P_3 [1, 2, 0] and P_4 [2 ,0, -1] in the 3D space which are locations for spot welds. The first method for the curve that is used is Cubic spline. It is designed such that the end point tangents are zero, signifying that the path's initial and final velocity are zero using the matrix seen in Fig 1. . By using Hermit interpolation, it is plotted as shown in Fig 2. Then the path was designed using the Bezier curve placing the intermediate points P2 and P3 on the path by placing the control points at the right location. The

main purpose for redesigning the Bezier curve that passes through the same points was to test the avoidance obstacles and kinematic characteristics of the robotic arm. Assuming there is a static object at P_4 [2 ,0, -1], the curve was segmented using Bezier segmentation to P_4' [3, 0 , -1] without changing the rest of the path that is passing through the rest of three points which can be seen in Fig 3. Even though both cubic spline-hermit interpolation and Bezier passing their path differs as seen in Fig 4.

$$\underbrace{\begin{pmatrix} 1 & 4 & 1 & 0 & \cdots & 0 \\ 0 & 1 & 4 & 1 & \cdots & 0 \\ & & & \ddots & \ddots & \vdots \\ 0 & \cdots & \cdots & 1 & 4 & 1 \end{pmatrix}}_{n} \begin{pmatrix} \mathbf{P}_1^t \\ \mathbf{P}_2^t \\ \vdots \\ \mathbf{P}_n^t \end{pmatrix} = \begin{pmatrix} 3(\mathbf{P}_3 - \mathbf{P}_1) \\ 3(\mathbf{P}_4 - \mathbf{P}_2) \\ \vdots \\ 3(\mathbf{P}_n - \mathbf{P}_{n-2}) \end{pmatrix}.$$

Figure 1: condition for the Cubic Spline

Figure 1 describes condition for a cubic spline, in this case there are 4 points. Equation 1 describes the mathematical representation of the Hermit interpolation using a cubic spline which uses the hermit interpolation matrix M to interpolate the points of the path passing through points $P_1\ P_2\ P_3\ P_4$. Equation 2 describes parametric form of the mathematical model. The t ranges from 0 to 1.

$$P(t) = T(t) \times M \times p \text{.....(1)}$$

$$P(t) = [t^3 \quad t^2 \quad t \quad 1] \begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} P_1 \\ P_2 \\ P_3 \\ P_4 \end{bmatrix} \text{.....(2)}$$

$$0 \leq t \leq 1$$

Equation 3 gives the mathematical form for the Bezier curve; Equation 4 describes the parametric equation of the Bezier curve and the points. The t ranges from 0 to 1.

$$P(t) = T(t) \times N \times p \text{.....(3)}$$

$$P(t) = [t^3 \quad t^2 \quad t \quad 1]\begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}\begin{bmatrix} P_1 \\ P_2 \\ P_3 \\ P_4 \end{bmatrix} \ldots\ldots\ldots\ldots(4)$$

$$0 \leq t \leq 1$$

Equation 5 is used to the segment the curve for avoiding the obstacle on the path of the robotic arm end-effector where alpha value was set 0.75 to allow the new curve pass through the first three points. This divides the curve into two segments based on the alpha value that signifies the ratio at which the curve is turned into two segments. The fourth is set such that the path avoids the path passing the through the location of the object. As obstacle is located at the fourth point. The last point of the second segment was changed to P_4' defined previously. The curve is illustrated on Fig 3 along with the obstacle.

$$\begin{pmatrix} \mathbf{P}_0 \\ \mathbf{P}_{01} \\ \mathbf{P}_{012} \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ \alpha & 1-\alpha & 0 \\ \alpha^2 & 2\alpha(1-\alpha) & (1-\alpha)^2 \end{pmatrix} \begin{pmatrix} \mathbf{P}_0 \\ \mathbf{P}_1 \\ \mathbf{P}_2 \end{pmatrix}$$
$$\begin{pmatrix} \mathbf{P}_{012} \\ \mathbf{P}_{12} \\ \mathbf{P}_2 \end{pmatrix} = \begin{pmatrix} \alpha^2 & 2\alpha(1-\alpha) & (1-\alpha)^2 \\ 0 & \alpha & 1-\alpha \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \mathbf{P}_0 \\ \mathbf{P}_1 \\ \mathbf{P}_2 \end{pmatrix}$$

Equation 5: Bezier Segmentation matrix form and the Alpha matrix.

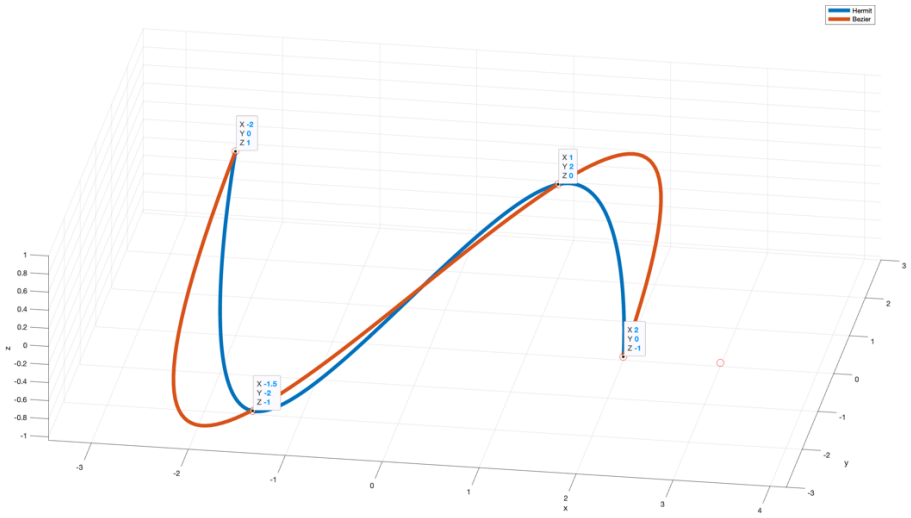


Figure 2: Plot of Cubic Spline-Hermit interpolation and Bezier curve.

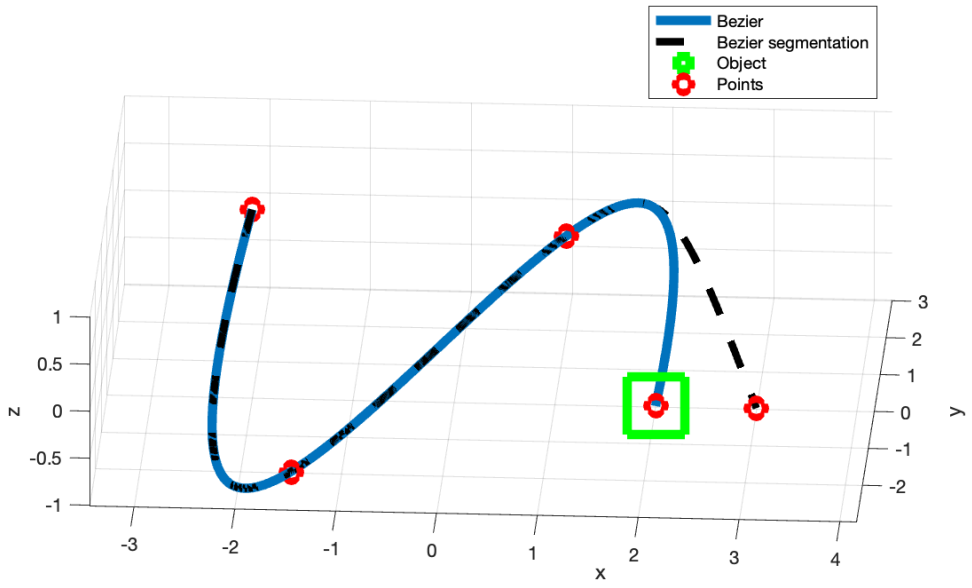


Figure 3: Plot of Object location Bezier curve and Bezier Segmentation.

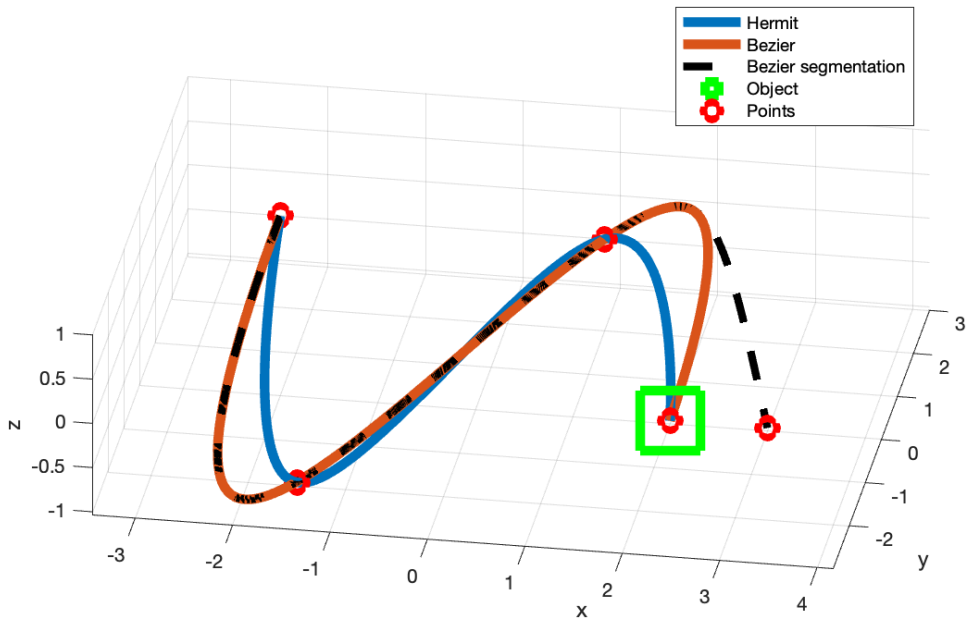


Figure 4: Plot of Cubic Spline-Hermit interpolation Bezier curve, and Bezier Segmentation.

2.2 Creating environment for simulation

The robotic arm selected for the application was Kuka LBR iiwa(Intelligent Industrial Work Assistant) 14. The LBR iiwa is the first sensitive robot ever produced in series that is also HRC-compatible. Intelligent industrial work assistant (iiwa) and lightweight robot (LBR-"Leichtbauroboter") are both German words. With the LBR iiwa, humans and robots can collaborate closely to complete sensitive tasks, creating the potential for new applications and improving efficiency and cost-effectiveness. It has 7 Degree of Freedom. It is commonly used for mechanical and industrial operations such as Mechanical machining, surface polishing Assembly and painting etc. Using the Command in Appendix 1 the robotic arm was imported to the environment by built-in MATLAB Toolbox called "Robot toolbox" as illustrated in Fig 5. To fit into the environment the curve was placed in the unit space of [-1 1] in the 3D as seen in Fig 6.

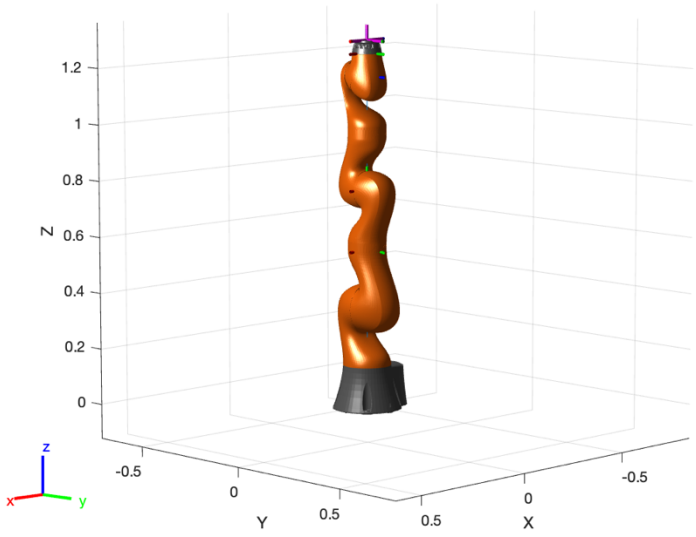


Figure 5: Importation of Robotic arm Kuka LBR iiwa 14 in MATLAB.

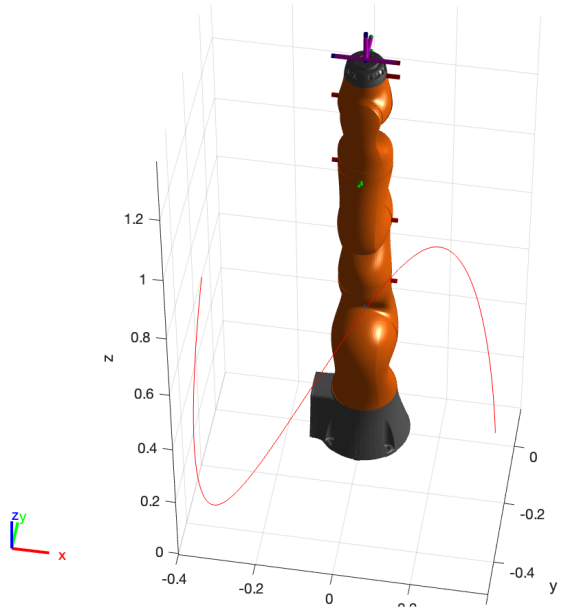


Figure 6: Robotic arm and the desired oath of the end-effector

2.3 Simulation

The simulation is carried out for a duration 10 seconds with a sampling time of 0.01 sec between each point on the path of the end-effector. A Simulink model was used to integrate the points and the robotic arm using inverse kinematics based on the rigid body tree of the selected robotic arm.

SIMULINK model

A Simulink model in Appendix 2 was used to calculate the inverse kinematics of the robotic arm. The mode uses pose, initial values, and the desired path points of the curve.

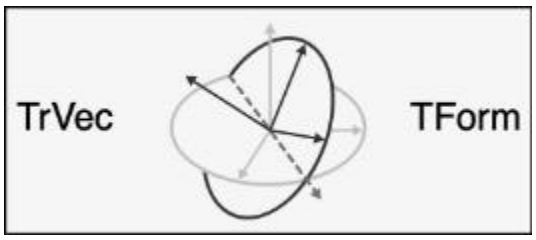


Figure 7: Coordinate Transformation Conversion block

To solve the inverse kinematics, the x, y, and z coordinates of the path that are defined in MATLAB code of Appendix 3 are made a translation matrix by a mux that converts into homogeneous transformations by Coordinate Transformation Conversion block (Fig 7) and input as the desired Pose.

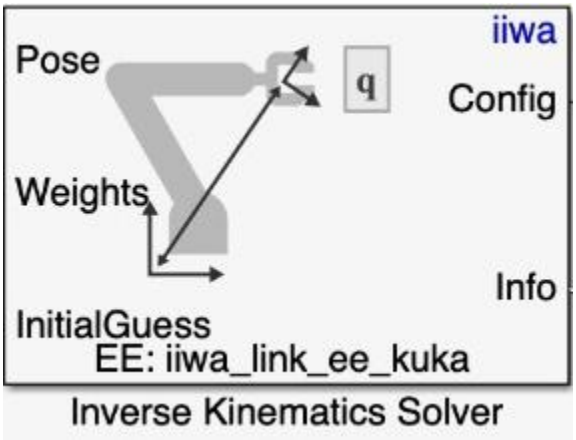


Figure 8: Inverse Kinematics Solver

There are three inputs to the inverse kinematics solver block seen in Fig 8, The end-effector pose, which is represented as a 4x4 homogeneous transformation matrix, is the first pose. This Matrix monitors the rigid body's desired position and orientation as specified by the end-effector parameter. Secondly the Weights, which are six-element vectors representing the pose tolerance weights. The first three components of this vector are related to the weights on the target pose's orientation error. The concluding three elements of the vector are associated with the weights on the xyz positional error for the intended pose. Lastly, Initial Guess which is defined as a vector of joint positions. The number of positions is the same as the Rigid body tree parameter's non-fixed joint count. To assist in leading the solver to the desired robot configuration, the initial guess is used. However, there is no guarantee that the answer will resemble this initial one.

The output of the Inverse kinematics solver is Configs. And Information. The following are the contents each output of the solver.

Configs - Robot configuration that, when given as a vector, resolves the desired end-effector pose. For the rigid body tree model, a robot configuration is a vector of joint positions. The number of positions is the same as the Rigid body tree parameter's non-fixed joint count.

Information - Solution information, returned as a bus. The solution information bus contains these elements:

- Iterations - Number of iterations run by the algorithm.
- PoseErrorNorm - The magnitude of the error between the pose of the end effector in the solution and the desired end-effector pose.
- ExitFlag - Code that gives more details on the algorithm execution and what caused it to return. For the exit flags of each algorithm type

The mean velocity and acceleration were plotted using the mean position of the end-effector. Appendix 3 shows the Simulink model for calculating the velocity and acceleration. A MATLAB function was used to find the mean of position from the origin ,these signifies that the velocity and acceleration measured are relative to the origin rather than relative to the end-effector or links or joints. A continuous derivative block was used to find the end-effector mean velocity and mean acceleration.

3 Results and discussion

3.1 Cubic-Spline Hermit interpolation

As seen from Fig 9, The end-effector has followed the desired path that is Cubic Spline Hermit interpolation which was plotted together in the operating environment. Fig 10 illustrates the mean position, mean velocity, and mean acceleration of the robotic arm end-effector. The position, velocity and acceleration were calculated in the Position Derivatives Simulink model with a

sampling time of 0.01 Sec. The duration of the simulation was 10 Sec. As the cubic spline was designed wanting the initial and final tangents to be zero. The velocity at the start and the end was zero.

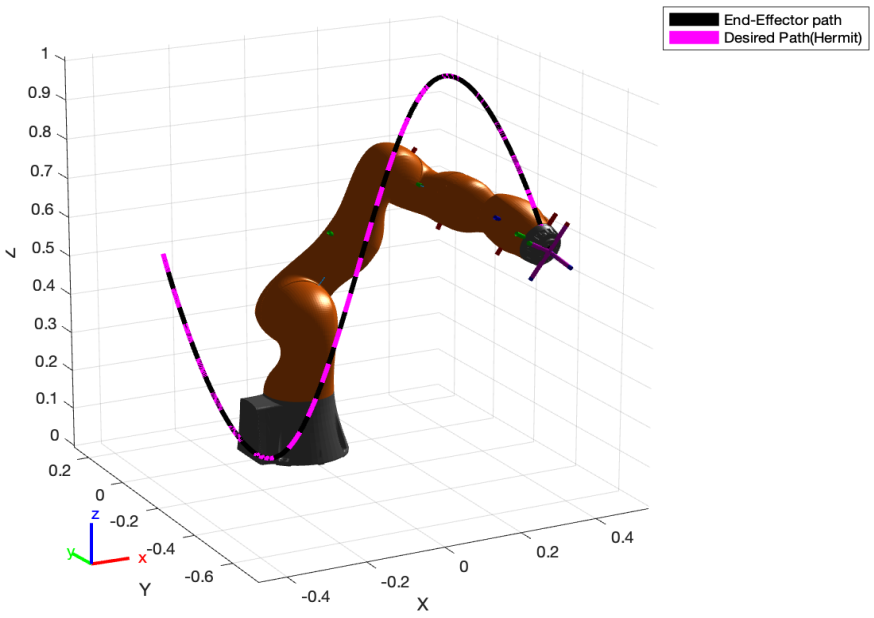


Figure 9: Comparing the desired path(Cubic spline-Hermit) vs the End-effector path.

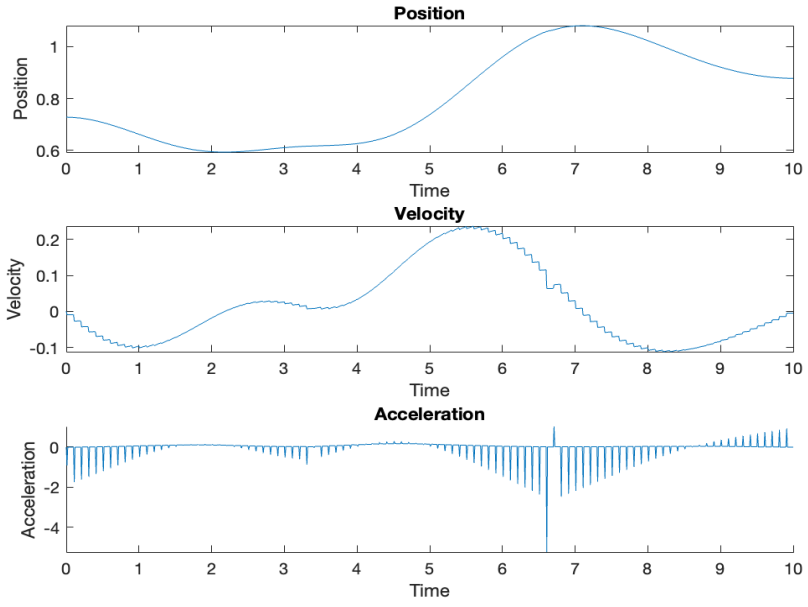


Figure 10: Mean position, velocity, and acceleration of the end-effector(Cubic spline-Hermit)

Note : This is not a continuous plot, it is a discrete plot.

3.2 Bezier Curve

As seen from Fig 11, The end-effector has followed the desired path that is Cubic Spline Hermit interpolation which was plotted together in the operating space. The plot is discrete with a sampling time of 0.01 Sec and not continuous, but we should take the peak as the value at that point. plot in Fig 12 there is a smooth position, velocity, and acceleration that states that Bezier curve performed better than cubic splines without any jerks in the acceleration of the robotic arm end-effector. Using the Simulink model, the interpolation and kinematics of the robot was linked as described in the later section. The duration of the simulation was 10 Sec.

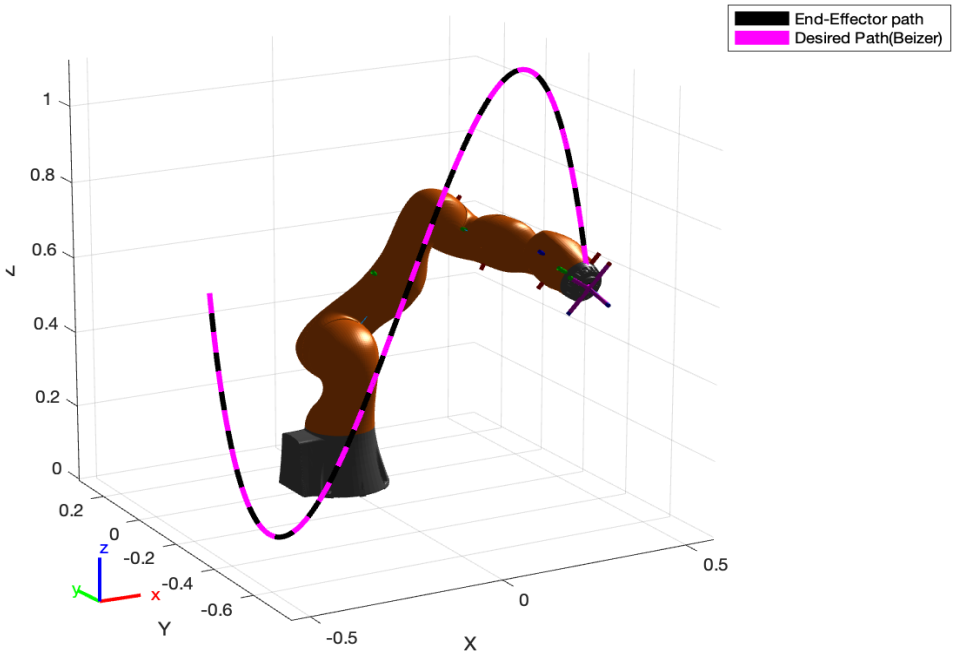


Figure 11: Comparing the desired path(Bezier) vs the End-effector path.

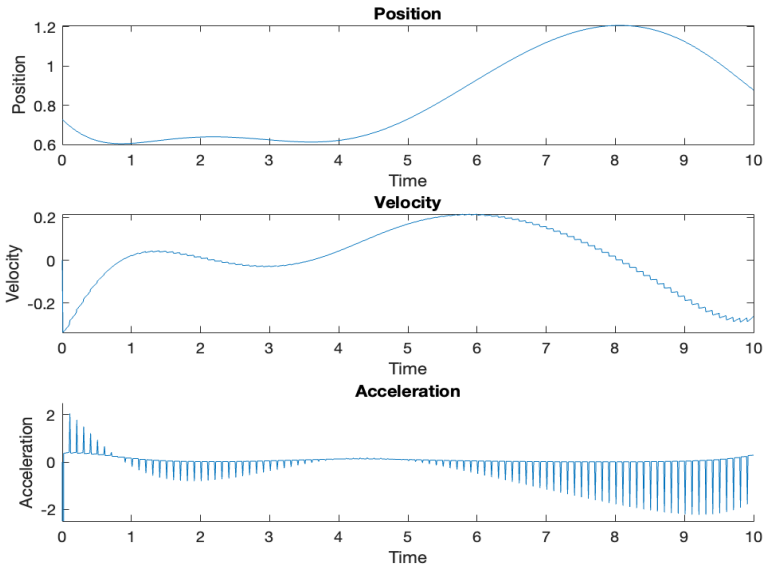


Figure 12: Mean position, velocity, and acceleration of the end-effector(Bezier).

Note : This is not a continuous plot, it is a discrete plot.

3.3 Bezier Segmentation

Fig 13 illustrates the Bezier Segmentation curve in which the path is segmentated with an alpha value of 0.75 such that the curve passes through the three points. The robotic arm end-effector final position along with the end-effector path and desired path which was segmented has been plotted. The velocity and acceleration were plotted by taking the derivatives of the position in the Simulink model in the code (Position derivatives). The duration of the simulation was 10 Sec.

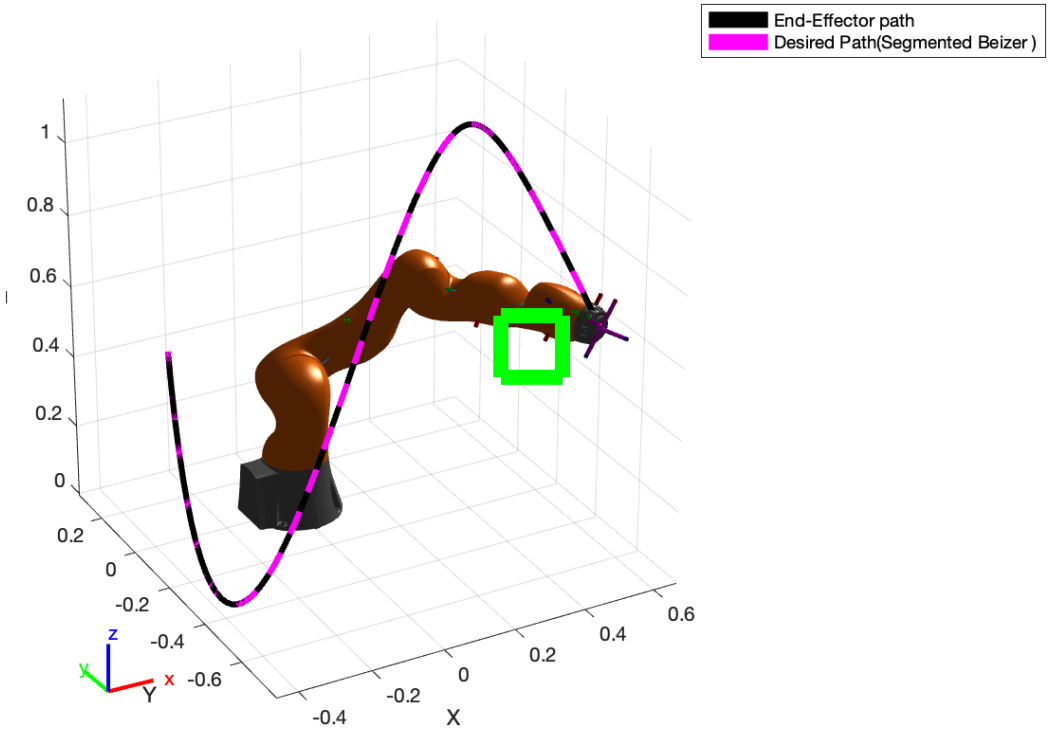


Figure 13: Comparing the desired path(Bezier Segmentation) vs the End-effector path.

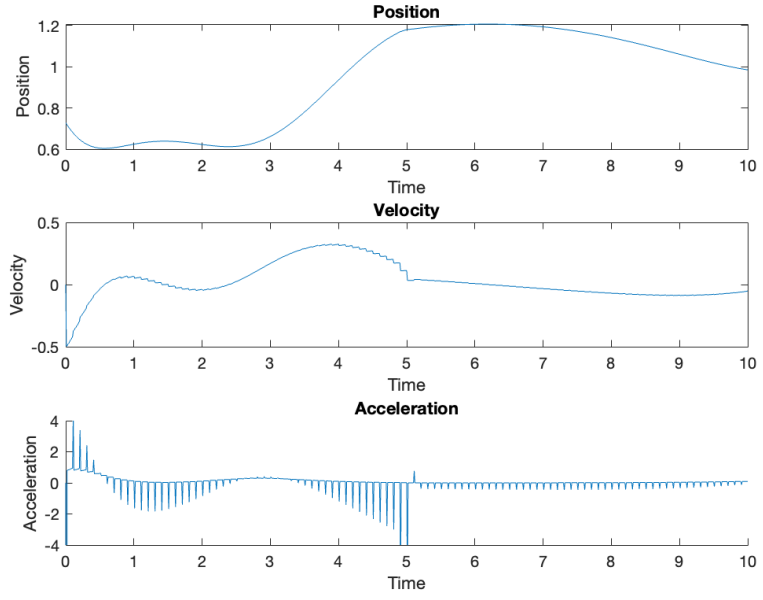


Figure 14: Mean position, velocity, and acceleration of the end-effector(Bezier Segmentation).

Note : This is not a continuous plot, it is a discrete plot

4 Conclusion

In the case of trajectory planning of the specific selected robotic arm. The Bezier curve is better than Cubic spline-Hermit interpolation. This is because the Cubic spline-Hermit interpolation has a narrow trajectory path when compared Bezier curve to the as seen in Fig 4. This feature of the Bezier curve helps the robot joints to rotated smoothly without any jerk. These is reason why we can notice an acceleration peak in the acceleration plot of the Cubic spline-Hermit interpolation whereas the Bezier curve acceleration plot (Fig 12) is smooth which means there are no jerks in the system, Therefore, in this case making Bezier curves feasible for the application in real environment.

Bezier curves is effective in changing the end points that avoid object without affecting the rest of the curve by using the Bezier segmentation method. For Bezier segmentation, the velocity and acceleration remain same as original Bezier curve but as soon as the end-effector entered the new path the acceleration and velocity almost went to zero as in Fig 14 these is because the segmentation makes the tangent zero at the point where the new path was laid. After this the end-effector went smoothly avoiding the object.

The major limitation of the Bezier curve beside its smooth and Collison-free trajectory is its inability to manipulate the tangents as per the application. These is a major problem for the problem involving the controlled trajectory. Coming to the initial and final velocity as Bezier Curve design, does not involve the manipulation tangents of the end points these means there will be an initial and final velocity at the ends. To solve this problem in the trajectory planning two cubic splines must be designed such that one with an initial velocity of zero and final velocity matching the initial velocity of the Bezier curve ,Other with cubic spline with its initial velocity matching the final velocity of the Bezier curve and sure that the splines does not have any major turns as to avoid the jerks in the system. In this manner, the jerks can be avoided leading to a smooth and efficient path.

5 Future Work

There is a need to develop algorithms for the absolute control of the robot arm which must be less costly to compute and work efficiently. Reducing the computing power is important as it requires less energy and saves time. Along with that there must be studies on the characteristics of the robotic arm based on the higher order parametric equation for the path planning and trajectory planning. The robotic arm should include better sensor for good avoidance of the obstacle.

References

[1]. Visioli A. Trajectory planning of robot manipulators by using algebraic and trigonometric splines. Robotica 2000; 18: 611–631.
[2]. Boryga M, Graboś A. Planning of manipulator motion trajectory with higher-degree polynomials use. Mechanism and Machine Theory 2009; 44: 1400–1419.
[3]. Dyllong E, Visioli A. Planning and real-time modifications of a trajectory using spline techniques. Robotica 2003; 1: 475–482
[4] Y. Cai, L. Chen, D. Qin, J. Xie, and X. Xu, "A path and velocity planning method for lane changing collision avoidance of intelligent vehicle based on cubic 3-D Bézier curve," Adv. Eng. Software 132, 65–73 (2019). doi: 10.1016/j.advengsoft.2019.03.007.

[5] Vahide Bulu."The optimal path of robot end effector based on hierarchical clustering and Bézier curve with three shape parameters",Robotica (2022), 40, pp. 3266–3289 doi:10.1017/S0263574722000182

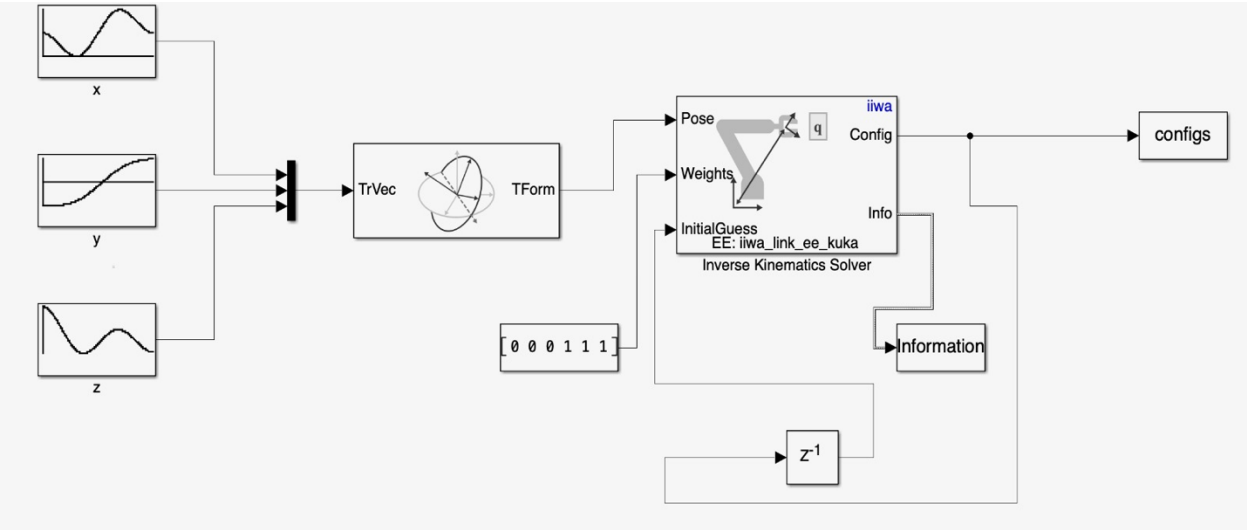
[6] R. Wang, M. Wang, Y. Guan, and X. Li, "Modeling and analysis of the obstacle-avoidance strategies for a mobile robot in a dynamic environment," Math. Probl. Eng. 2015, 1–11 (2015). doi: 10.1155/2015/837259

Appendix

1 Robotic importing

```
%% importing
iiwa = importrobot('iiwa14.urdf');
iiwa.DataFormat = 'column';
```

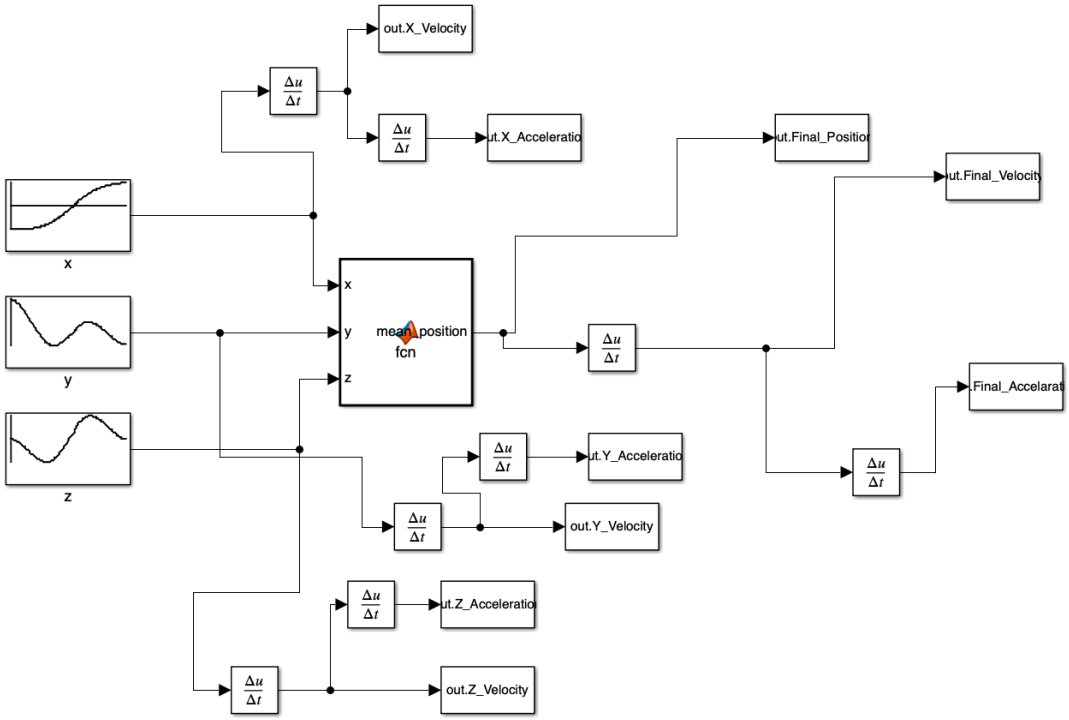
2 Simulink model for inverse Kinematics



3 Cubic-Spline Hermit interpolation

```
%% Curve
Ts=0.01;
t = 0:Ts:1; t=t';
T= [t.^3,t.^2,t,0*t+1];
M = [2 -2 1 1; -3 3 -2 -1; 0 0 1 0; 1 0 0 0];
G=[-2 0 1;-1.5 -2 -1 ;0 0 0;1.7 1.2 -0.8];
P1=T*M*G;
G1=[-1.5 -2 -1;1 2 0;1.7 1.2 -0.8;2.2 1.2 0.2];
P2=T*M*G1;
G2=[1 2 0;2 0 -1;2.2 1.2 0.2;0 0 0];
P3=T*M*G2;
x1=P1(:,1);
y1=P1(:,2);
z1=P1(:,3);
x2=P2(:,1);
y2=P2(:,2);
z2=P2(:,3);
x3=P3(:,1);
y3=P3(:,2);
z3=P3(:,3);
x=0.2*[x1;x2;x3];
z=0.2*[y1;y2;y3];
y=0.2*[z1;z2;z3];
z=z+0.6;
y=y-0.3;
plot3(x,y,z)
xlabel('x')
ylabel('y')
zlabel('z')
grid on
```

3 Simulink Model for Velocity and acceleration



4 Cubic Spline-Hermit interpolation MATLAB code for integrating the curves and the robotic arm

```
clc;
close all;
clear all;
%% importing
iiwa = importrobot('iiwa14.urdf');
iiwa.DataFormat = 'column';
%% Curve
Ts=0.01;
t = 0:Ts:1; t=t';
T= [t.^3,t.^2,t,0*t+1];
M = [2 -2 1 1; -3 3 -2 -1; 0 0 1 0; 1 0 0 0];
G=[-2 0 1;-1.5 -2 -1 ;0 0 0;1.7 1.2 -0.8];
P1=T*M*G;
 G1=[-1.5 -2 -1;1 2 0;1.7 1.2 -0.8;2.2 1.2 0.2];
P2=T*M*G1;
 G2=[1 2 0;2 0 -1;2.2 1.2 0.2;0 0 0];
P3=T*M*G2;
x1=P1(:,1);
y1=P1(:,2);
z1=P1(:,3);
x2=P2(:,1);
y2=P2(:,2);
z2=P2(:,3);
x3=P3(:,1);
```

```
y3=P3(:,2);
z3=P3(:,3);

x=0.2*[x1;x2;x3];
z=0.2*[y1;y2;y3];
y=0.2*[z1;z2;z3];


z=z+0.6;
y=y-0.3;
plot3(x,y,z)
xlabel('x')
ylabel('y')
zlabel('z')
grid on
%% Viewing robot and the path
show(iiwa);
hold all
axis equal
plot3(x,y,z,'r','MarkerSize',20)
xlabel('x')
ylabel('y')
zlabel('z')
%view(135,20)
hold off
%% Simlink implimentation and animation

sim('Inverse_Kinematics_model.slx')
figure('Visible','on');
tform = 1;
j=0;
for i = 1:10:numel(configs.Data)/7
    j=j+1;
    currConfig = configs.Data(:,1,i);
    xlim auto
    ylim auto
    zlim auto
    plot3(x(j),y(j),z(j))
    show(iiwa,currConfig);
    drawnow
    xyz(tform,:) = tform2trvec(getTransform(iiwa,currConfig,'iiwa_link_ee'));
    tform = tform + 1;
end

%% Final Trajectory of the end-effector

figure('Visible','on')
show(iiwa,configs.Data(:,1,end));
hold on
plot3(xyz(:,1),xyz(:,2),xyz(:,3),'-k','LineWidth',3);
plot3(x,y,z,'m--','LineWidth',3)
legend('End-Effector path','Desired Path(Hermit)')
xlim auto
ylim auto
zlim auto
```

```
hold off
%% Velocity and acceleration
H=sim('PositionDerivatives.slx');
time= configs.Time;
position=H.Final_Position.signals.values;
Accelaration=H.Final_Accelaration.signals.values;
Velocity=H.Final_Velocity.signals.values;
vx=H.X_Velocity.signals.values;
ax=H.X_Acceleration.signals.values;
vy=H.Y_Velocity.signals.values;
ay=H.Y_Acceleration.signals.values;
vz=H.Z_Velocity.signals.values;
az=H.Z_Acceleration.signals.values;

figure;
subplot(3, 1, 1);
plot(tout, position);
title('Position');
xlabel('Time');
ylabel('Position');
subplot(3, 1, 2);
plot(tout, Velocity);
title('Velocity');
xlabel('Time');
ylabel('Velocity');
subplot(3, 1, 3);
hold on;
plot(tout, Accelaration);
title('Acceleration');
xlabel('Time');
ylabel('Acceleration');
```

5 Bezier interpolation MATLAB code for integrating the curves and the robotic arm

```
clc;
close all;
clear all;

iiwa = importrobot('iiwa14.urdf');
iiwa.DataFormat = 'column';
%% Curve
P=[-2 0 1;-3.6718 -9.009 -4.1431;2.9206 9.009 2.6556;2 0 -1];
n= size(P,1)-1;
Ts=0.01;
t = 0:Ts:1; t=t';
N = zeros(n);
T = ones(length(t),n+1);
for i=1:n
 T(:,i) = t.^(n-i+1);
end
for r=0:n
 for c=0:n-r
```

```
 N(r+1,c+1) = nchoosek(n,c)*nchoosek(n-c,r)*(-1)^(n-r-c);
 end
end

bez1 =T*N*P;

x1=bez1(:,1);
y1=bez1(:,2);
z1=bez1(:,3);

x=0.2*x1;
z=0.2*y1;
y=0.2*z1;
z=z+0.6;
y=y-0.3;

plot3(x,y,z)
xlabel('x')
ylabel('y')
zlabel('z')
grid on
%% Simlink implimentation and animation

sim('Inverse_Kinematics_model.slx')
figure('Visible','on');
tform = 1;
j=0;
for i = 1:10:numel(configs.Data)/7
    j=j+1;
    currConfig = configs.Data(:,1,i);
    xlim auto
    ylim auto
    zlim auto
    plot3(x(j),y(j),z(j))
    show(iiwa,currConfig);
    drawnow
    xyz(tform,:) = tform2trvec(getTransform(iiwa,currConfig,'iiwa_link_ee'));
    tform = tform + 1;
end

%% Velocity and acceleration
H=sim('PositionDerivatives.slx')
time= configs.Time;
position=H.Final_Position.signals.values;
Accelaration=H.Final_Accelaration.signals.values;
Velocity=H.Final_Velocity.signals.values;
figure;
subplot(3, 1, 1);
plot(tout, position);
title('Position');
xlabel('Time');
ylabel('Position');
subplot(3, 1, 2);
plot(tout, Velocity);
title('Velocity');
```

```
xlabel('Time');
ylabel('Velocity');
subplot(3, 1, 3);
hold on;
ylim([-2.5 2.5])
plot(tout, Accelaration);
title('Acceleration');
xlabel('Time');
ylabel('Acceleration');
%% Final Trajectory of the end-effector

figure('Visible','on')
show(iiwa,configs.Data(:,1,end));
hold on
plot3(xyz(:,1),xyz(:,2),xyz(:,3),'k-','LineWidth',3);
plot3(x,y,z,'m--','LineWidth',3)
legend('End-Effector path','Desired Path(Beizer)')
xlim auto
ylim auto
zlim auto
hold off
```

6 Bezier interpolation MATLAB code for integrating the curves and the robotic arm

```
clc;
close all;
clear all;

iiwa = importrobot('iiwa14.urdf');
iiwa.DataFormat = 'column';
%% Curve
P=[-2 0 1;-3.6718 -9.009 -4.1431;2.9206 9.009 2.6556;2 0 -1];
n= size(P,1)-1;
Ts=0.01;
t = 0:Ts:1; t=t';
N = zeros(n);
T = ones(length(t),n+1);
for i=1:n
 T(:,i) = t.^(n-i+1);
end
for r=0:n
 for c=0:n-r
 N(r+1,c+1) = nchoosek(n,c)*nchoosek(n-c,r)*(-1)^(n-r-c);
 end
end
alpha=0.25;
n = size(P, 1);
Q = zeros(n, n);
```

```
R = zeros(n, n);
G = zeros(n, n);
P1 = zeros(n-1, 2);
P2 = zeros(n-1, 2);
for i = 0:n-1
    for j = 0:i
        Q(i+1, j+1) = nchoosek(i, j) * alpha^(i-j) * (1-alpha)^j;
    end
end
for i = 0:n-1
    for j = 0:i

        R(i+1, j+1) = nchoosek(i, j) * alpha^(j) * (1-alpha)^(i-j);

    end
end
for i=1:n
    for j=1:n
        if i==1
            G(i,j)=Q(n-i+1,j);
        else
        G(i,j)=R(n-i+1,n-j+1);
        end
    end
end

P1 =Q*P;
P2= G*P;

Ns=size(P1,1)-1;
T = ones(length(t),Ns+1);
for i=1:Ns
    T(:,i) = t.^(Ns-i+1);
end

Ps=[P2(1:3,:);3 0 -1];
bezs1=T*N*P1;
bezs2=T*N*Ps;
 x1=[bezs1(:,1);bezs2(:,1)];
 y1=[bezs1(:,2);bezs2(:,2)];
 z1=[bezs1(:,3);bezs2(:,3)];
 x=0.2*x1;
z=0.2*y1;
y=0.2*z1;
z=z+0.6;
y=y-0.3;

O=[0.2*2 0.2*-1-0.3 -0*0.2+0.6];
figure
hold on
 plot3(O(1,1),O(1,2),O(1,3),'g.','Markersize',40)
plot3(x,y,z)
xlabel('x')
ylabel('y')
zlabel('z')
```

```
grid on
%% Simlink implimentation and animation

sim('Inverse_Kinematics_model.slx')
figure('Visible','on');
tform = 1;
j=0;
for i = 1:10:numel(configs.Data)/7
    j=j+1;
    currConfig = configs.Data(:,1,i);
    xlim auto
    ylim auto
    zlim auto
    plot3(x(j),y(j),z(j))
    show(iiwa,currConfig);
    drawnow
    xyz(tform,:) = tform2trvec(getTransform(iiwa,currConfig,'iiwa_link_ee'));
    tform = tform + 1;
end

%% Simulation
figure('Visible','on');
tformIndex = 1;
j=0;
for i = 1:10:numel(configs.Data)/7
    j=j+1;
    currConfig = configs.Data(:,1,i);
    plot3(x(j),y(j),z(j))
    xlim auto
    ylim auto
    zlim auto
    show(iiwa,currConfig);
    drawnow
    xyz(tformIndex,:) =
tform2trvec(getTransform(iiwa,currConfig,'iiwa_link_ee'));
    tformIndex = tformIndex + 1;
end

%% Final Trajectory of the end-effector

figure('Visible','on')
show(iiwa,configs.Data(:,1,end));
hold on
plot3(O(1,1),O(1,2),O(1,3),'gs','Markersize',40,'LineWidth',10)
plot3(xyz(:,1),xyz(:,2),xyz(:,3),'k-','LineWidth',3);
plot3(x,y,z,'m--','LineWidth',3)
legend('End-Effector path','Desired Path(Segmented Beizer )')
xlim auto
ylim auto
zlim auto
hold off

%% Velocity and acceleration
H=sim('PositionDerivatives.slx')
time= configs.Time;
```

```
position=H.Final_Position.signals.values;
Accelaration=H.Final_Accelaration.signals.values;
Velocity=H.Final_Velocity.signals.values;
figure;
subplot(3, 1, 1);
plot(tout, position);
title('Position');
xlabel('Time');
ylabel('Position');
subplot(3, 1, 2);
plot(tout, Velocity);
title('Velocity');
xlabel('Time');
ylabel('Velocity');
subplot(3, 1, 3);
hold on;
ylim([-4 4])
plot(tout, Accelaration);
title('Acceleration');
xlabel('Time');
ylabel('Acceleration');
```