Summer Internship 2022-23

Title: Wafer Handling 4 Degree of Freedom robot

Project Report by

Saptarshi Rajan
B.Tech Undergraduate in Mechanical Engineering
Birla Institute of Science and Technology Pilani, Hyderabad
Under the Guidance of:

Prof. Cheruvu Siva Kumar
Department of Mechanical Engineering
Indian Institute of Technology Kharagpur
Duration – 10/06/2022-11/08/2022

Table of contents-

Acknowledgement-

I would like to express my special thanks and gratitude to my supervisor, Prof. Cheruvu Shiva Kumar as well as my mentors, Singhmar Harbaksh Singh and Diwakar Gurung who gave me the golden opportunity to do this wonderful project on the topic Wafer handling '4 DoF Robot' in the Robotics domain, which also helped me in doing a lot of research and I came to know about so many new things.

I am really thankful to them.
Secondly, I would also like to thank my friends and the other fellow MTech and Phd Scholars in the Robotics & Automation Lab who has helped me in finishing this project in limited time.

It really helped me increase my knowledge and skills.

Thanks For everyone.

SAPTARSHI RAJAN

Introduction

Finding innovative, more effective techniques to automate the handling of semiconductor wafers is a frequent challenge for semiconductor producers. Wafers come in a variety of sizes and shapes, necessitating various handling procedures. Therefore, finding a suitable handling solution can occasionally be a time-consuming and expensive task for producers. An automated wafer handler is a robotics system designed to automate and facilitate the handling of silicon wafers during the semiconductor manufacturing process. In order to complete various manufacturing activities during the course of the semiconductor production lifecycle, wafer handlers are integrated with semiconductor process tools. A good wafer handling robot is characterized not only by application-specific properties but also by:

- Quiet running
- Precision
- Long life
- User-friendliness

In addition to the wafer handlers, the end effector is the most elementary component since it touches the substrate directly as an attachment of the wafer handler.

Figure 1: Wafer Handling robots

In this project, we have chosen SCARA Robot for pick and place operations as a reference, where high speed and high accuracy is required. A SCARA robot often has a greater operating speed and an optional cleanroom specification. Currently available SCARA robots may attain repeatability tolerances of less than 10 microns, as opposed to a six-axis robot's tolerance of 20 microns. The SCARA robot is made to be used in applications with a tiny operating area and in places with

limited floor space. Their compact size also makes them easier to move to temporary or outlying locations. Highly flexible, precise and compact SCARA robots can help manufacturers attain production results, while requiring minimal capital investment, creating momentum for an array of tasks across multiple industries, boosting production and accelerating return on investment in a fluctuating market.

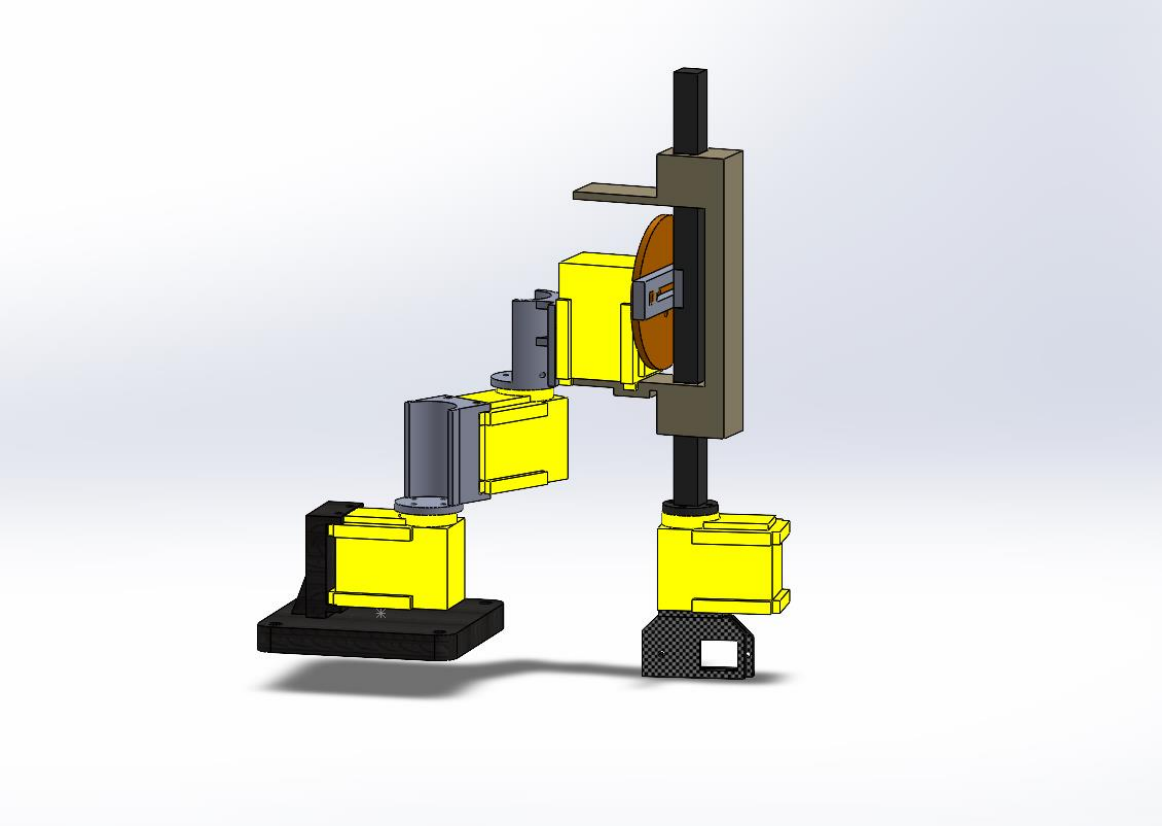

Figure 2: CAD model of SCARA Robot

Literature Review

Now-a-days tasks needed to be performed by the robots are defined using analytical or off-line programming tools also. Therefore, there must be a precise correlation between the robot model and the robot as it operates. With the aid of a geometric model, the relationship between the robot's joint and task spaces is established. Denavit-Hartenberg (DH), the most used nomenclature, is typically offered by robot manufacturers in the form of robot specifications. The parameters of the robot in action may differ due to component manufacture and assembly faults, wear and tear, etc., necessitating the use of measurement equipment with extremely high long-range precision. The cost of position measurement equipment that can take data at a great distance is significant. Therefore, it is necessary to determine the precise model of the robot using a technique that can make use of smaller workspace measurement inputs. Eliminating the discrepancy between the actual robot geometry and the geometric representation recorded in the controller is the most crucial step in reducing positioning error. Other techniques for representing geometric models include Hayati parameters, which add a parameter to account for the influence of consecutive joint axes that are parallel or nearly parallel in order to prevent numerical instability during estimation. It was suggested to extract the DH parameters analytically using vector algebra. The method provided motivation to develop a novel methodology using Dual Vector Algebra in Automatic Extraction of DH Parameters of Serial Manipulators using Line Geometry, which has compact and elegant representation. The autonomous identification method is proposed which is only based on the joint sensor readings obtained through the robot sensory interface of an industrial robot. The position of the end-effector caused by the movement of only one joint keeping the others locked is noted, and the process was repeated by progressing towards the inward joints starting from the last one. The method is useful in identifying the DH parameters of the robot within a small workspace, by giving a limited angular motion to each joint. As a result, one can use measurement devices with lesser measuring volume, thus increasing the accuracy and reducing the cost of the device.

The most widely used notation for the geometric modeling of robots is DH notation. The travel from the base frame to the end-effector frame is achieved by moving across two consecutive frames placed at the joints. The set of four parameters relates the transformation between Frame i to Frame i+1 by d_i , θ_i , a_i and α_i.

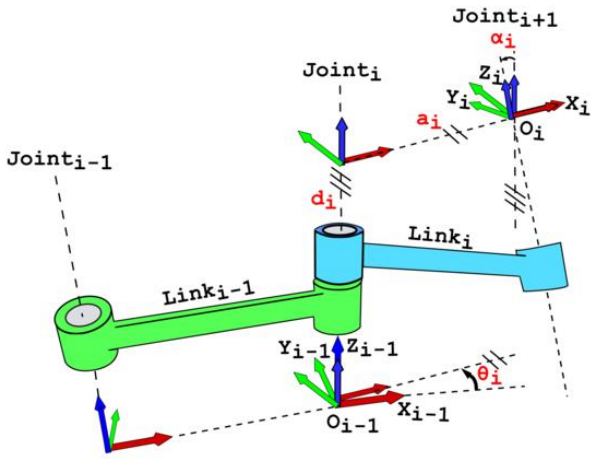

Figure 3: Robot kinematics nomenclature

An industrial robot's manufacturer often provides the DH parameters in the form of engineering drawings or specifications. It was suggested to use an analytical technique to extract the DH parameters from a robot's CAD model. The approach was created as a plug-in or add-on for the Autodesk Inventor CAD program. It required two inputs: the direction of the joint axes and the coordinates of the circle's centre, as measured in a frame attached to the robot's base. The software's Application Programming Interface (API) was used to determine the joint axes, or the direction (Zi) and point (Ci) on the axis. These axes were represented as Dual Vectors and the relationship between two consecutive joint axes, i.e., if they are parallel, intersecting or skewed, was determined using Dual Vector Algebra and the DH parameters were determined using line geometry. The methodology is used here to determine the DH parameters of a real robot. This proposed method showed that a small range of the end-effector did not change the identified results for the DH parameters. Hence the robustness of the algorithm has been established.

With a high degree of flexibility and versatility SCARA robots has been in the electronics and automotive industry for quite a long time. It is a highly non-linear, strong coupling and time varying system. This studies the robot In this the robot joints 1, 2, and 4 that rotate are studied, along with joints 3 that are straight and use an AC servo motor and ball screw drive. An incremental photoelectric encoder can determine the robot's location. To protect the safety of the robot body and the operator, 1, 2, and 3-axis joint control are also mounted on the photoelectric limit switch, which is not assured beyond its joint axis travel range. SCARA robot has three rotational joints, θ_{124} variable for the joint. A movement of the joint, for the joint variables.

We use the DH method to link the coordinate system established at each joint odd transformation matrix that represents its relationship with the previous link coordinate system set with the robot coordinate system is connected to the base reference coordinate system, you can get the appropriate posture transformation matrix.

For inverse kinematics we always describe the position and orientation of the robot tool's end when creating the robot control program using the global coordinate system. We must move the robot to the end of each joint tool pose and position from the current position in order to make

it reach the end of the tool defined location and with the required posture. Any point (within the scope of the robot's movement) can be fed into the control interface to move from a known position (such as zero) to another point, and then values can be coordinated in accordance with this point (constant) calculated the value of the joint angle at that time, then this is the starting point of the joint angle value with the value of joint angle can be drawn differently requiring to rotate each joint of the robot angle increment.

The simulation realization of robot end is expected to from the initial point A to point B. SCARA robot 3 d simulation graph generated by MATLAB. We input parameters of the robot. Connecting rod in the first four elements of α, a, θ, d, the last element for joint types: 0 for the rotational joint, 1 for mobile joints.

Through the outcomes of the simulation, we saw the robot go from point A to point B and from point B to point C with normal joint action, as indicated in tables 2 and 3. Experiments that support the connecting rod parameters' complete rationalism conclude that a simple design can reach the parameters of a predefined aim.

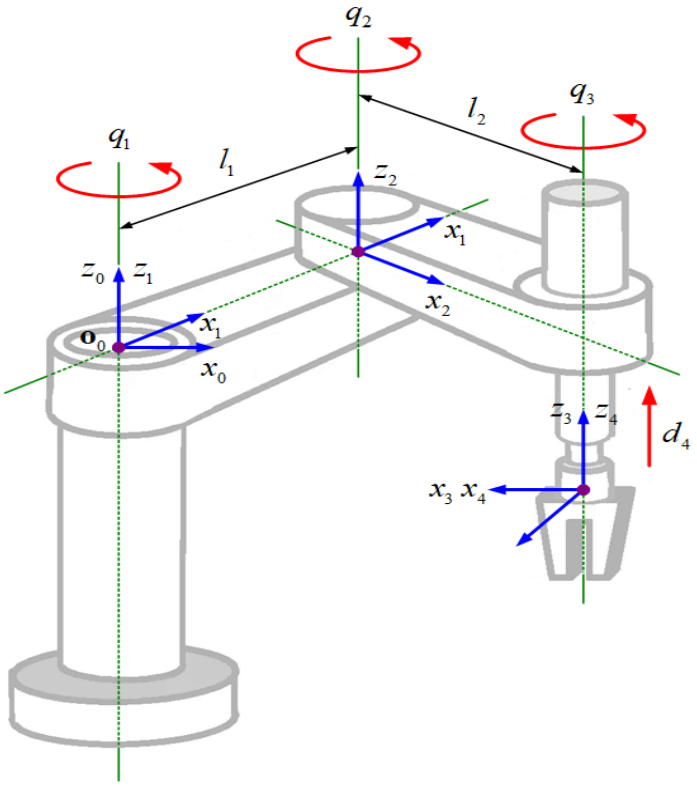

Figure 4: SCARA robot and its D-H notation

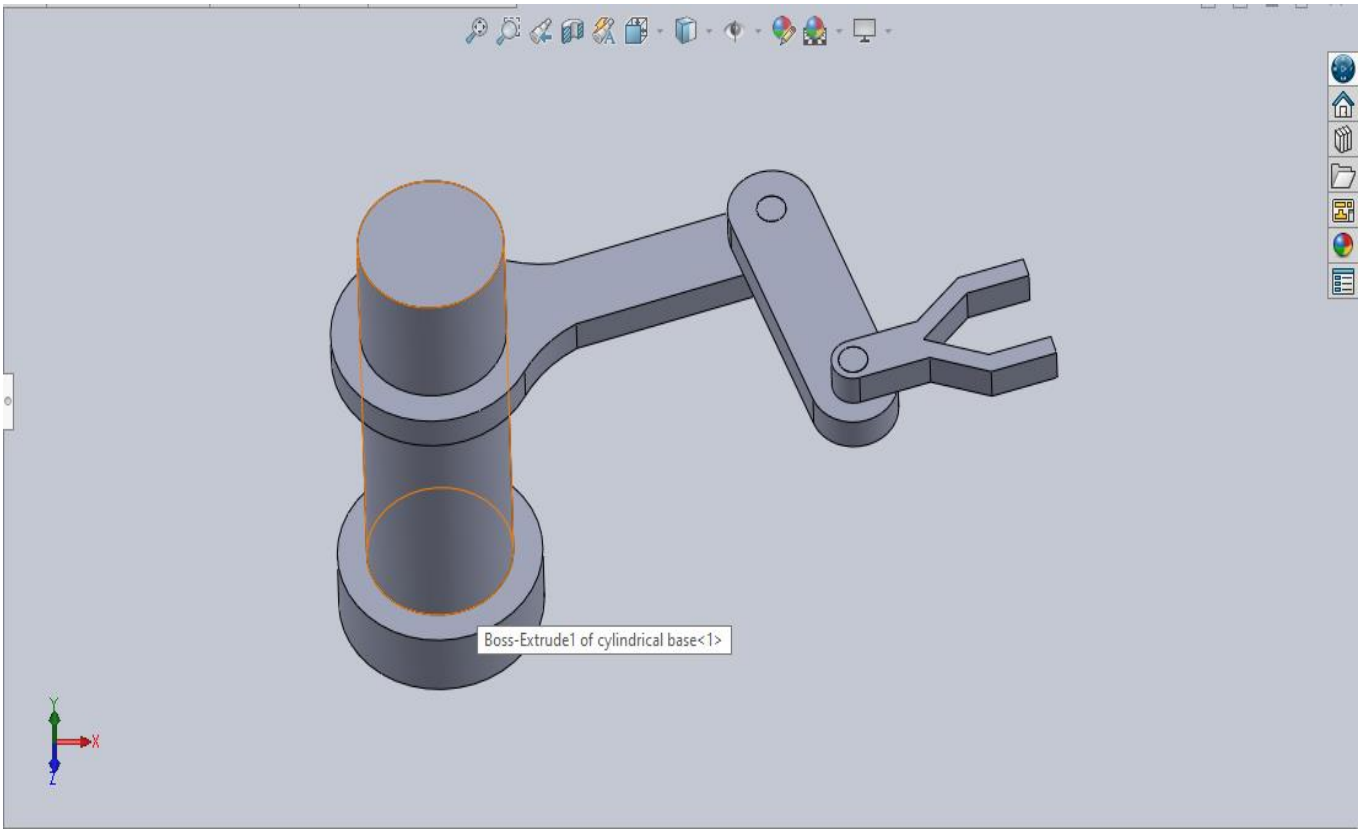

Figure 5: CAD model of 4 DoF Wafer Handling Robot

Our Wafer Handling Robot is a RPRR type robot with 4 DoF. The above attached picture is from the CAD modelling which I have done in SolidWorks. It comprises of 5 parts-

- The base which is purely for support purpose
- The cylinder upon which a link slides up & down. It acts a cylindrical type of joint.
- Link 1- which can rotate as well as slide up and down.
- Link 2- is a revolute joint, which helps in better positioning and placing of wafers even in tight space.
- Gripper- for handling the wafers. It's also a revolute joint.

Kinematics of Wafer handling Robot

1. Link Parameters

Any robot can be described kinematically by giving the values four quantities for every link. Two define the link itself, and two describe the link's connection to a neighboring link. In the usual case of a revolute joint, θ_i is called the joint variable, and the other three quantities would be fixed link parameters. For prismatic joints, d_i is the joint variable, and the other three quantities are fixed link parameters. The definition of mechanisms by means of these quantities is a convention usually called the D-H notation. For the derivation c = cos(.) and s = sin(.)

- θ_i is the joint angle. It is measured as the angle from X_{i-1} to X_i measured about Z_i.
- d_i is the link offset. It is the distance from X_{i-1} to X_i measured along Z_i
- α_i is the link twist angle. It is the angle from Z_i to Z_{i+1} measured along X_i
- a_i is the link length. It is the distance from Z_i to Z_{i+1} measured along X_i

Link	a_i	α_i	d_i	θ_i
1	0	0	a_1	Θ_1
2	a_2	0	d_2	0
3	a_3	0	0	Θ_3
4	0	-90	0	Θ_4

Table 1: D-H parameters of our wafer handling robot

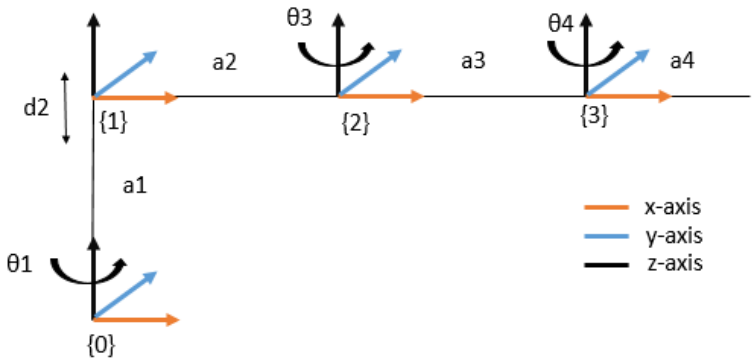

Figure 6: Different frames and joints in our robot

2. Forward Kinematics

$$T_1 = \begin{bmatrix} c1 & -s1 & 0 & 0 \\ s1 & c1 & 0 & 0 \\ 0 & 0 & 1 & d1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad T_2 = \begin{bmatrix} 1 & 0 & 0 & a2 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d2 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T_3 = \begin{bmatrix} c3 & -s3 & 0 & a3c3 \\ s3 & c3 & 0 & a3s3 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad T_4 = \begin{bmatrix} c4 & 0 & -s4 & 0 \\ s4 & 0 & c4 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T_1T_2 = \begin{bmatrix} c1 & -s1 & 0 & c1a2 \\ s1 & c1 & 0 & s1a2 \\ 0 & 0 & 1 & a1 + d2 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T_1T_2T_3 = \begin{bmatrix} c1c3 - s1s3 & -(c1s3 + s1c3) & 0 & c1a2 + a3c13 \\ s1c3 + c1s3 & -s1s3 + c1c3 & 0 & s1a2 + a3s13 \\ 0 & 0 & 1 & a1 + d2 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Where c13= cos (θ_1+ θ_3), s13= sin (θ_1+ θ_3)

$T_1T_2T_3T_4$=

$$\begin{bmatrix} c13c4 - s13s4 & 0 & -(c13s4 + s13c4) & c1a2 + a3c13 \\ s13c4 + c13s4 & 0 & -s13s4 + c13c4 & s1a2 + a3c13 \\ 0 & -1 & 0 & a1 + d2 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

3. Inverse Kinematics

Procedure:-

1. First, with given position and orientation about Z0 (alpha) of end effector the position of base of link four is derived using the equation below.
   X=Xee+a4*sin (alpha)
   Y=Yee-a4*cos (alpha)
   Z=Zee
2. Using modified X, Y, Z and alpha ,Transformation matrix is found
   $T=\begin{bmatrix} r11 & r12 & r13 & r14 \\ r21 & r22 & r23 & r24 \\ r31 & r32 & r33 & r34 \\ r41 & r42 & r34 & r44 \end{bmatrix}$
3. First three joints are used to reach the required position of end effector. So, theta1,d2,theta3 is found by equation the (1,4) , (2,4) ,(3,4) of T30 and T.

$$\text{So, } T_3^0=\begin{bmatrix} c13 & -s13 & 0 & c1a2 + a3c13 \\ s13 & c13 & 0 & s1a2 + a3s13 \\ 0 & 0 & 1 & a1 + d2 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

We found out that $a_1+d_2=r_{34}$, so $d_2=r_{34}- a_1$. Squaring and adding T(1,4), T(2,4) =>

- $a_3^2c_{13}^2 + a_2^2c_1^2 + 2a_2a_3c_{13}c_1 + a_3^2s_{13}^2 + a_2^2s_1^2+2a_3a_2s_{13}^2s_1 = r_{14}^2+r_{24}^2$
- $a_3^2+a_2^2+2a_3a_2c_3=r_{14}^2+r_{24}^2$,
- $\theta_3= \cos^{-1}[(r_{14}^2+r_{24}^2-a_2^2-a_3^2)/ 2a_2a_3]$

Let $\cos\theta_3= k$,
Now to find θ_1 replace θ_3 in T(1,4).

- $a_3(c_1c_3-s_3s_1)+ a_2c_1= r_{14}$,
- $(a_3c_3+a_2)c_1-s_3s_1=r_{14}$,

We will use the entity by dividing by $\text{sqrt}((a_2+a_3c_3)^2+(a_3s_3)^2)$.

- $\text{Sin}\varphi c1- \cos\varphi s1= \frac{r14}{\text{sqrt}((a2+a3c3)^2+(a3s3)^2)}$
- $\text{Sin}(\varphi- \theta_1)= \frac{r14}{\text{sqrt}((a2+a3c3)^2+(a3s3)^2)}$
- $\theta_1= \varphi- \sin^{-1}[\frac{r14}{\text{sqrt}((a2+a3c3)^2+(a3s3)^2)}]$,
  Where $\varphi= \frac{a2+a3c3}{\text{sqrt}((a2+a3c3)^2+(a3s3)^2)}$.

4. Last Joint (J4) is for orientation of end effector. It is found by equation the elements of R40(1,1) and T(1,1).

$R_4^0=\begin{bmatrix} c134 & -s134 & 0 \\ s134 & c134 & 0 \\ 0 & 0 & 1 \end{bmatrix}$, where c134= cos ($\theta_1+ \theta_3+ \theta_4$) and s134= sin($\theta_1+ \theta_3+ \theta_4$).

Using R (1,1) we get, $\theta_4 = \cos^{-1}(r_{11}) - (\theta_1 + \theta_3)$.

4. Jacobian

- $P_{i-1} = {}^{0}R_{i-1}\hat{u}$,
- ${}^{i-1}P_n = {}^{0}T_n O_n - {}^{0}T_{i-1} O_n$
- $J_i = \begin{bmatrix} P(i-1) * (i-1)Pn \\ Pi - 1 \end{bmatrix}$ for revolute joints
- $J_i = \begin{bmatrix} P(i-1) \\ 0 \end{bmatrix}$ for prismatic joints

$$P_0 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

$${}^{0}P_4 = \begin{bmatrix} c1a2 + a3c13 \\ s1a2 + a3c13 \\ d1 + d2 \\ 1 \end{bmatrix} - \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

After solving the equations we get,

$$J_1 = \begin{bmatrix} -s1a2 - a3s13 \\ c1a2 + a3c13 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}, \quad J_2 = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \quad J_3 = \begin{bmatrix} -a3s13 \\ a3c13 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}, \quad J_4 = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

$$J = \begin{bmatrix} -s1a2 - a3s13 & 0 & -a3s13 & 0 \\ c1a2 + a3c13 & 0 & a3c13 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 \end{bmatrix}$$

Results and Discussions

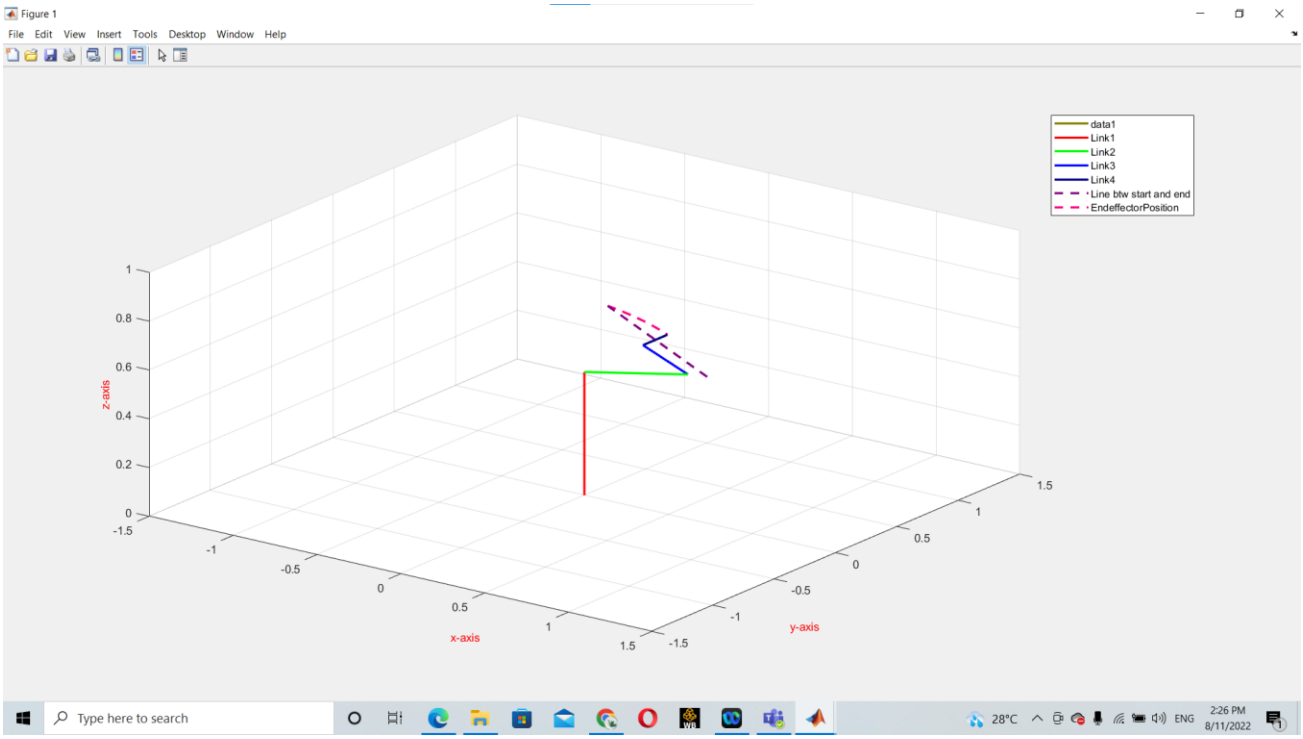

Figure 7: MATLAB simulation of our model

We have calculated the forward & inverse kinematics using transformation matrices as well as the Jacobian matrix of our robot. We have prepared a CAD model as well as a MATLAB simulation based on our code. In the MATLAB, we implemented path planning and through the simulation we could see that the end effector is going from one point to the other in the coordinate space in cubical trajectory motion, also in the way verifying our forward & inverse kinematics of our robot model.

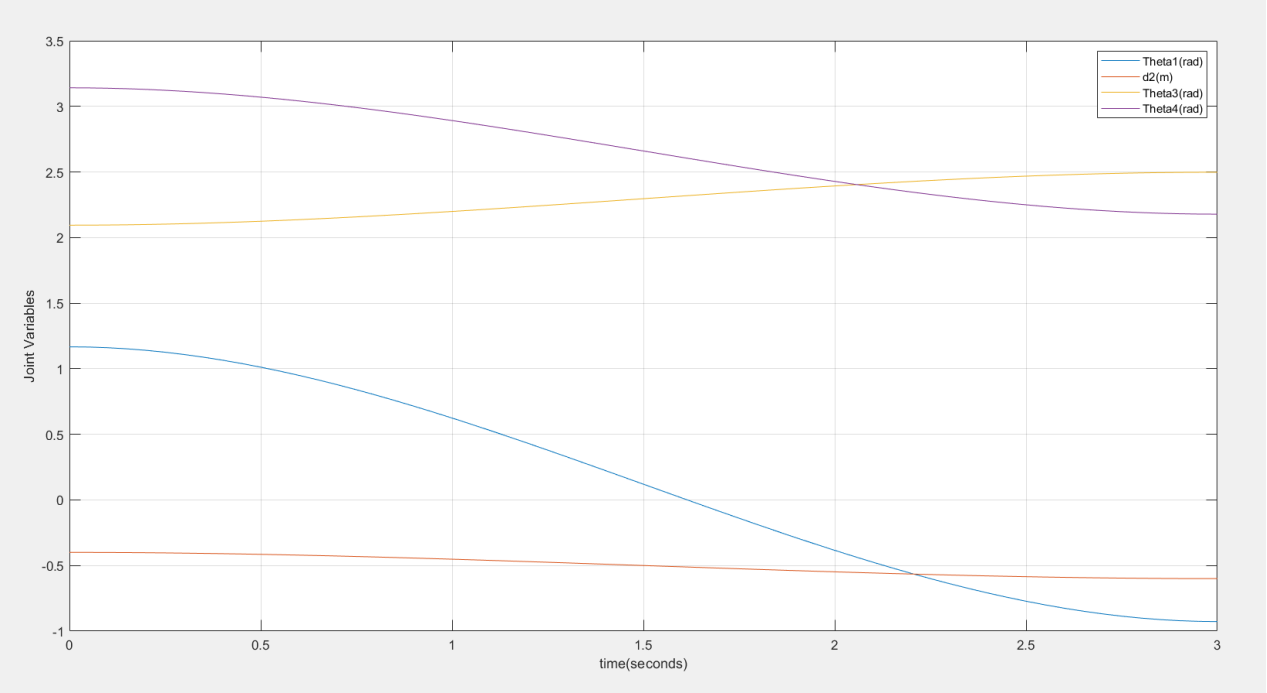

Figure 8: Joint Variables time trajectory

Appendix

MATLAB Codes

8/11/22 2:23 PM C:\Users\harba\OneDr...\FunctionHolder.m 1 of 1

```
classdef  FunctionHolder
    properties
        a1=1;a2=0.5;a3=0.5;a4=0.2;
        realpathPoints=[-0.3 0.3;0.6 0.6;0.5 0.5];   %position of end effector
        OrientZ=[0 0];                               %Orientation of end effector in rad
    end
    methods
        function R=Rotationmatrix(~,alpha,theta)
             R=[cosd(theta) -sind(theta)*cosd(alpha) sind(theta)*sind(alpha);
                sind(theta) cosd(theta)*cosd(alpha) -cosd(theta)*sind(alpha);
                0 sind(alpha) cosd(alpha)];
        end
        function T=TransformationMatrix(a,alpha,theta,d)
            T=[cosd(theta) -sind(theta)*cosd(alpha) sind(theta)*sind(alpha) a*cosd(theta);
                sind(theta) cosd(theta)*cosd(alpha) -cosd(theta)*sind(alpha) a*sind(theta);
                0 sind(alpha) cosd(alpha) d;0 0 0 1];
        end
        function I=InverseKinematics(~,T,a1,a2,a3)
            theta3=acos((T(1,4)^2+T(2,4)^2-a2^2-a3^2)/(2*a2*a3));
            psi=asin((a2+a3*cos(theta3))/sqrt((a2+a3*cos(theta3))^2+(a3*sin(theta3))^2));
            theta1=psi-asin(T(1,4)/sqrt((a2+a3*cos(theta3))^2+(a3*sin(theta3))^2));
            d2=T(3,4)-a1;
            theta4=acos(T(1,1)-theta1-theta3);
            I=[theta1 d2 theta3 theta4];
        end
        function X = CoefficientsOfCubic(~,t1,t2,th1,th2,thd1,thd2)   %for path planning
            syms a0 a1 a2 a3
            eqn1 = a0+t1*a1 +t1^2*a2 + t1^3*a3 == th1;
            eqn2 = a1 +2*t1*a2 + 3*t1^2*a3 == thd1;
            eqn3 =a0+t2*a1 +t2^2*a2 + t2^3*a3 == th2;
            eqn4 = a1 +2*t2*a2 + 3*t2^2*a3 == thd2 ;
            [A,B] = equationsToMatrix([eqn1, eqn2, eqn3, eqn4], [a0, a1, a2 ,a3]);
            X = linsolve(A,B);
        end
    end
end
```

```
robot=FunctionHolder;
a1=robot.a1;a2=robot.a2;a3=robot.a3; a4=robot.a4;
realpathPoints=robot.realpathPoints;
OrientZ=robot.OrientZ;
%% shift of coordinates from end effector to base of Link4
pathPoints=realpathPoints;
for j=1:size(realpathPoints,2)
    pathPoints(1,j)=realpathPoints(1,j)+a4*sin(OrientZ(j));
    pathPoints(2,j)=realpathPoints(2,j)-a4*cos(OrientZ(j));
end
%% Konwn Transformation matrix from end effector to base of robot
Te0=cell(1,size(realpathPoints,2));
for i=1:size(pathPoints,2)
    Te0{i}=[cos(OrientZ(1,i)) -sin(OrientZ(1,i)) 0 pathPoints(1,i);sin(OrientZ(1,i)) ↙
cos(OrientZ(1,i)) 0 pathPoints(2,i);
        0 0 0 pathPoints(3,i);0 0 0 1];
end
velPathPoints=[0 0;0 0;0 0];
%% Trajectory Planning
%From Cartesian coordinates to Joint variable using inverse kinematics
JointVariables=zeros(5,size(pathPoints,2));
JointVariablesdot=zeros(5,size(pathPoints,2));
for i=1:size(pathPoints,2)
    I=robot.InverseKinematics(Te0{i},a1,a2,a3);
    JointVariables(1,i)=I(1,1);
    JointVariables(2,i)=I(1,2);
    JointVariables(3,i)=I(1,3);
    JointVariables(4,i)=I(1,4);
    JointVariables(5,i)=OrientZ(i);    %known orientation of end effector wrt base
end

%% Cubic Polynomial trajectory generation in joint space
T=[0 3] ;                                    % TIme elapsed
t=0:0.05:T(size(T,2));
NoOfJoints=5;                                %joint 5 is not the joint but the ↙
orientation of end effector wrt base.
NoOfCubicPoly=size(pathPoints,2)-1;
coefficentOfPoly=cell(1,NoOfJoints);
theta=cell(1,NoOfJoints);
coefficientMatrix=cell(NoOfJoints,NoOfCubicPoly);
for j=1:NoOfJoints
    for k=1:NoOfCubicPoly
        C1=(robot.CoefficientsOfCubic(T(k),T(k+1),JointVariables(j,k),JointVariables ↙
(j,k+1),JointVariablesdot(j,k),JointVariablesdot(j,k+1)));
        coefficentOfPoly{j}(1,k)=C1(1,1);
        coefficentOfPoly{j}(2,k)=C1(2,1);
        coefficentOfPoly{j}(3,k)=C1(3,1);
        coefficentOfPoly{j}(4,k)=C1(4,1);
        coefficientMatrix{j,k}=C1;
```

```
    end
end

for k=1:NoOfCubicPoly
    for j=1:NoOfJoints
        for i=1:length(t)
            if t(i)<=T(k+1) && t(i)>=T(k)
                theta{j}(i)=double(coefficientMatrix{j,k}(1,1)+coefficientMatrix{j,k}(2,1)*t(i)+coefficientMatrix{j,k}(3,1)*t(i)^2+coefficientMatrix{j,k}(4,1)*t(i)^3);
            end
        end
    end
end


%% Main Code
% Forward Kinematics
T10=cell(1,size(t,2));T21=cell(1,size(t,2));T32=cell(1,size(t,2));T43=cell(1,size(t,2));T20=cell(1,size(t,2));
T30=cell(1,size(t,2));T40=cell(1,size(t,2));xm=zeros(1,size(t,2));ym=zeros(1,size(t,2));zm=zeros(1,size(t,2));
for j=1:length(t)
    T10{j}=[cos(theta{1}(j)) -sin(theta{1}(j)) 0 0;sin(theta{1}(j)) cos(theta{1}(j)) 0 0;0 0 1 a1;0 0 0 1];
    T21{j}=[1 0 0 a2;0 1 0 0;0 0 1 theta{2}(j);0 0 0 1];
    T32{j}=[cos(theta{3}(j)) -sin(theta{3}(j)) 0 a3*cos(theta{3}(j));sin(theta{3}(j)) cos(theta{3}(j)) 0 a3*sin(theta{3}(j));0 0 1 0;0 0 0 1];
    T43{j}=[cos(theta{4}(j)) -sin(theta{4}(j)) 0 0;sin(theta{4}(j)) cos(theta{4}(j)) 0 0;0 0 1 0;0 0 0 1];
    T20{j}=T10{j}*T21{j};
    T30{j}=T20{j}*T32{j};
    %modified end effector position
    xm(j)=T30{j}(1,4)-a4*sin(theta{5}(j));
    ym(j)=T30{j}(2,4)+a4*cos(theta{5}(j));
    zm(j)=T30{j}(3,4);
end
%% Animation
plot3([0 0],[0 0],[0 0],'Color',[0.5 0.5 0],'LineWidth',2); %Link1
axis([-1.5 1.5 -1.5 1.5 0 3]);
xlabel('x-axis','FontSize',10,'Color',[1,0,0]);
ylabel('y-axis','FontSize',10,'Color',[1,0,0]);
zlabel('z-axis','FontSize',10,'Color',[1,0,0]);
grid on
hold on
h1=plot3([0 0],[0 0],[0 0],'Color',[1 0 0],'LineWidth',2,'DisplayName','Link1'); %link1
h2=plot3([0 0],[0 0],[0 0],'Color',[0 1 0],'LineWidth',2,'DisplayName','Link2'); %link2
h3=plot3([0 0],[0 0],[0 0],'Color',[0 0 1],'LineWidth',2,'DisplayName','Link3'); %
```

```
link3
h4=plot3([0 0],[0 0],[0 0],'Color',[0 0 0.5],'LineWidth',2,'DisplayName','Link4'); %↙
link4
h5=plot3([realpathPoints(1,1) realpathPoints(1,2)],[realpathPoints(2,1) ↙
realpathPoints(2,2)],[realpathPoints(3,1) realpathPoints(3,2)], ...
    '--','Color',[0.5 0 0.5],'LineWidth',2,'DisplayName','Line btw start and end'); %↙
straight trajectory
h6=plot3(xm(1),ym(1),zm(1),'--','Color',[1 0 0.5],'LineWidth',↙
2,'DisplayName','EndeffectorPosition'); %end effector position
legend;
i=0;
while i<=3
    for j=(2:length(t))
        set(h1,'XData',[0 T10{j}(1,4)],'YData',[0 T10{j}(2,4)],'ZData',[0 T20{j}↙
(3,4)]);
        set(h2,'XData',[T10{j}(1,4) T20{j}(1,4)],'YData',[T10{j}(2,4) T20{j}↙
(2,4)],'ZData',[T20{j}(3,4) T20{j}(3,4)]);
        set(h3,'XData',[T20{j}(1,4) T30{j}(1,4)],'YData',[T20{j}(2,4) T30{j}↙
(2,4)],'ZData',[T20{j}(3,4) T30{j}(3,4)]);
        set(h4,'XData',[T30{j}(1,4) xm(j)],'YData',[T30{j}(2,4) ym(j)],'ZData',[T30↙
{j}(3,4) zm(j)]);
        set(h6,'XData',xm(1:j),'YData',ym(1:j),'ZData',zm(1:j));
        pause(0.1);
    end
end
```