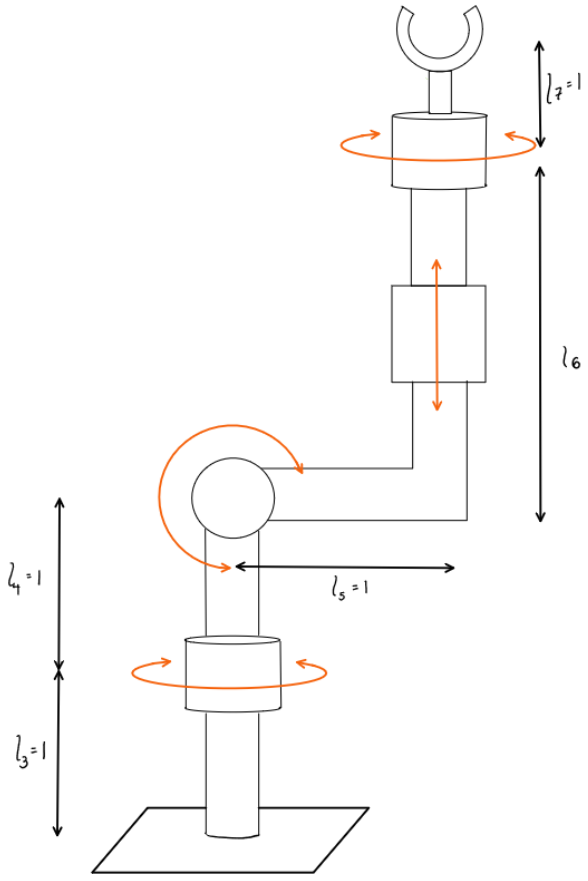
Dhanush Karthikeyan

12/3/21

MATLAB Simulation Project

In this project, we will be observing and experimenting with the forward (2.36) and inverse (2.39) kinematics for a 4-DOF robotic arm. We will assign the desired dimensions of the links and create our desired trajectory within the robot's workspace. This will then be simulated in MATLAB, where our desired and actual trajectories will be superimposed to identify any deviations in trajectories.



For the scope of this project, we will adhere to the following constraint: the trajectory will not entail any changes in orientation. Therefore, our *n*, *o*, *a* vectors will be fixed values. It is important noting that there is a translation from the universal of lengths L1 and L2, which will be reintroduced later into our matrix calculations.

The dimensions of the robotic arm are determined in the code and assigned to variables, so that the code can be dynamically changed if any future dimensional changes are necessitated.

Creating the Robotic Arm

```
clc;

% Robotic Arm Links
l1 = 4;
l2 = 4;
l3 = 1;
l4 = 1;
l5 = 1;
l7 = 1;
```

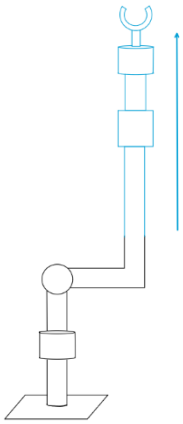
We now determine our desired trajectory. To show the diverse motions available to the robotic arm, I have chosen a two-stage trajectory where each motion can be isolated upon visual inspection. This occurs over a period of 20 seconds, with a sample time of 10ms for each calculated coordinate.

Desired 2-Stage Trajectory

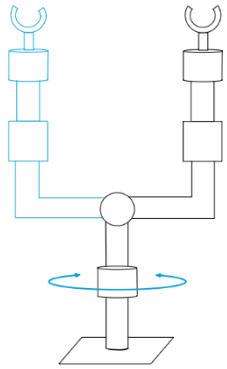
```
% Sample time
h=0.01;

t0 = 0:h:10; % t is a vector from 0-10 secs at a sample time of 0.01 sec
t1 = 10+h:h:20; % t is a vector from 0-20 secs at a sample time of 0.01 sec
t = [t0 t1];

% First Stage
Px1 = l1 + 0*t0;
Py1 = l2 + 0*t0 - l5;
Pz1 = 0.1*t0 + 4;

% Second Stage
Px2 = l1 + 1*sin(pi/2*t1);
Py2 = l2 + 1*cos(pi/2*t1);
Pz2 = 0.1*t1 + 4;

% Total 2-Stage Motion Trajectory
Px = [Px1 Px2];
Py = [Py1 Py2];
Pz = [Pz1 Pz2];
```

The first trajectory will be a simple movement about the z-axis, where the values in the x and y axes remain the same. This movement is accomplished through elongation of Link L_6, which is variable to change. If properly executed, there will be minimal deviation in the movement and the only change will be in L_6.

The second trajectory will be a simple spiral rotation, where movement about the z-axis occurs incrementally as the arm rotates. This movement will combine a rotation element (through manipulation of θ_1) and the first trajectory, which result in a spiral movement.



Stage 1



Stage 2

Fixed Orientation

```
% fixed orientation
nx=1;ny=0;nz=0;
ox=0;oy=1;oz=0;
ax=0;ay=0;az=1;

% number of columns
n_col = max(size(t));
```

As established by the project parameters, the orientation of the hand will remain constant. Therefore, we will set these as constants that remain the same throughout the trajectory.

Inverse Kinematics:

From our desired trajectory, we can calculate combinations of joint angles and variable lengths (L_6) that can accomplish the movement. We make use of inverse kinematics, which are calculated from the matrix multiplications of the A-matrices derived from our DH table.

$\#i$	θ_i	d_i	a_i	α_i
0 – 1	θ_1	L_4	0	90^o
1 – 2	θ_2	0	L_5	-90^o
2 -3	$180^o + \theta_3$	$L_6 + L_7$	0	0

$$\text{from } S_1 a_x - C_1 a_y = 0 \text{ we get}$$

$$\theta_1 = \tan^{-1}\left(\frac{P_y}{P_x}\right)$$

$$\text{from } S_2 = -(C_1 a_x + S_1 a_y) \text{ and } C_2 = a_z \text{ we get}$$

$$\theta_2 = \text{atan2}(S_2, C_2)$$

$$\text{from } S_3 = S_1 n_x - C_1 n_y \text{ and } C_3 = S_1 o_x - C_1 o_y \text{ we get}$$

$$\theta_3 = \text{atan2}(S_3, C_3)$$

These equations can then be used to determine our joint angles and length of L_6, which are dynamically written into the code base. Through a loop, we traverse the time period (20s) and populate the vectors for the four variables: $\theta_1, \theta_2, \theta_3, and\ L_6$.

Inverse Kinematics:

```
% Initialize vectors for variables
theta1=zeros(1,n_col);
theta2=zeros(1,n_col);
theta3=zeros(1,n_col);
l6 = zeros(1,n_col);

for i=1:n_col
    theta1(i) = atan2(Py(i) - l2,Px(i) - l1);
    C1 = cosd(theta1(i)); S1 = sind(theta1(i));

    theta2(i) = atan2d(-1*(C1*ax + S1*ay), az);
    C2 = cosd(theta2(i)); S2 = sind(theta2(i));

    theta3(i) = atan2d((S1*nx - C1*ny),(S1*ox - C1*oy));

    l6(i) = ((Pz(i)-l3-l4-l5*S2)/C2)-l7;
end
```

Forward Kinematics:

```
% Code to Calculate the Forward Kinematics

% ai    alphai   di     thetai
A1=DH_MatrixA(str2sym('[0,pi/2,l4,th1]'));
A2=DH_MatrixA(str2sym('[l5,-(pi/2),0,th2]'));
A3=DH_MatrixA(str2sym('[0,0,l6+l7,pi+th3]'));
A0=str2sym('[1,0,0,l1;0,1,0,l2;0,0,1,l3;0,0,0,1]');
AT = A0*A1*A2*A3
```

AT =

$$\begin{pmatrix} \sin(th_1)\sin(th_3) - \cos(th_1)\cos(th_2)\cos(th_3) & \cos(th_3)\sin(th_1) + \cos(th_1)\cos(th_2)\sin(th_3) & -\cos(th_1)\sin(th_2) & l_1 - \cos(th_1)\sin(th_2)\,(l_6+l_7) + l_5\cos(th_1)\cos(th_2) \\ -\cos(th_1)\sin(th_3) - \cos(th_2)\cos(th_3)\sin(th_1) & \cos(th_2)\sin(th_1)\sin(th_3) - \cos(th_1)\cos(th_3) & -\sin(th_1)\sin(th_2) & l_2 - \sin(th_1)\sin(th_2)\,(l_6+l_7) + l_5\cos(th_2)\sin(th_1) \\ -\cos(th_3)\sin(th_2) & \sin(th_2)\sin(th_3) & \cos(th_2) & l_3 + l_4 + \cos(th_2)\,(l_6+l_7) + l_5\sin(th_2) \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

To calculate the forward kinematics, we derive the A matrices from our DH table, which are multiplied together to determine our total transformation matrix. We include the matric A_0, which represents the initial translation from the universal to joint 0. This translation was removed from the DH table to simplify the matrix calculations but will need to be included in the forward kinematics to calculate the exact final coordinates with respect to the universal.

From this matrix, we can calculate our positional coordinates, as detailed in the fourth column. These equations yield the x, y, and x coordinates when using the provided joint angles and link lengths. By using the forward kinematics in tandem with the inverse kinematics, we can determine if the joint angles were properly calculated and if the desired trajectory is mathematically feasible.

The equations for the forward kinematics:

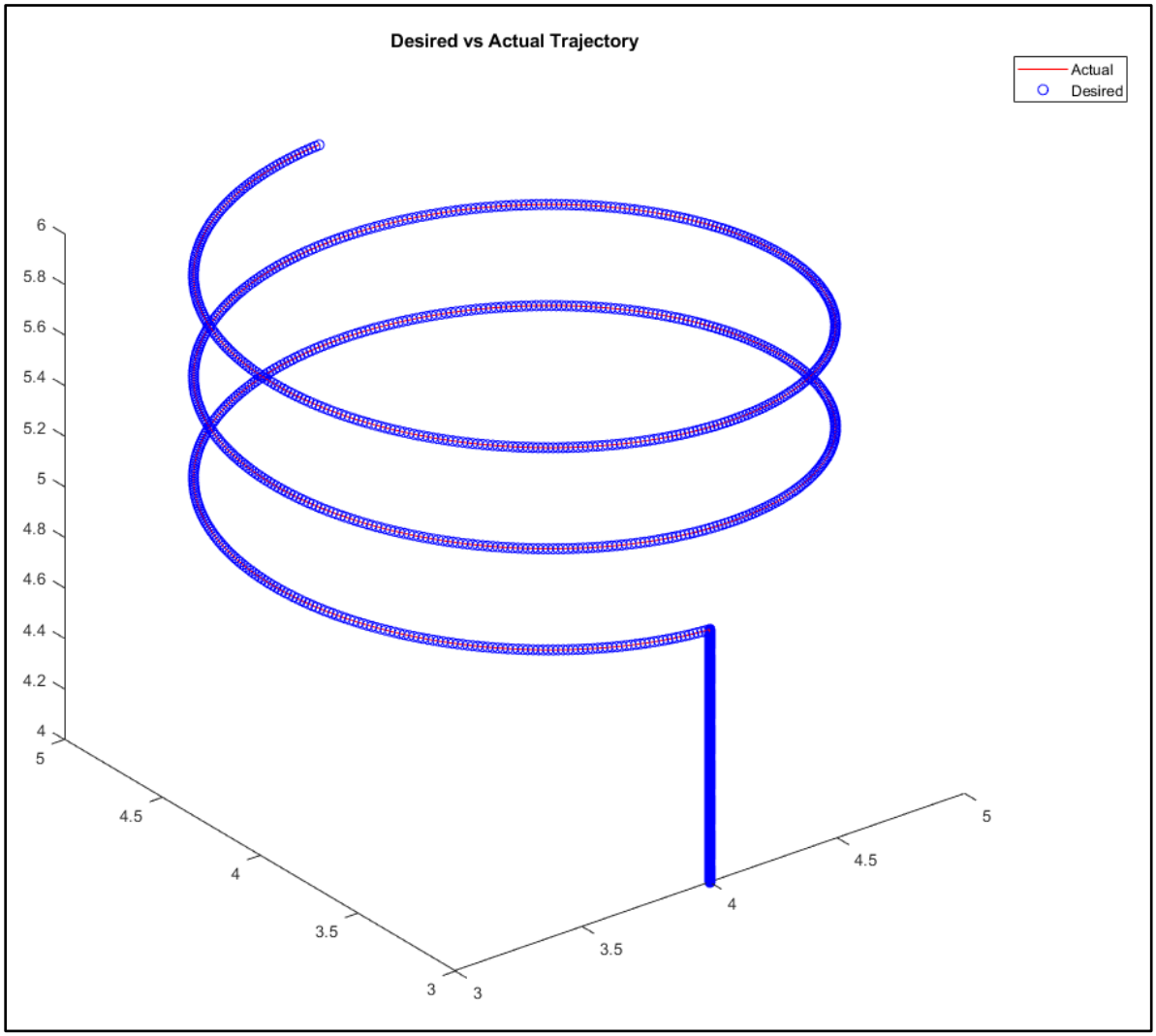
$$\begin{gathered} l_1 - \cos(\mathrm{th}_1)\sin(\mathrm{th}_2)\ (l_6 + l_7) + l_5\cos(\mathrm{th}_1)\cos(\mathrm{th}_2) \\ l_2 - \sin(\mathrm{th}_1)\sin(\mathrm{th}_2)\ (l_6 + l_7) + l_5\cos(\mathrm{th}_2)\sin(\mathrm{th}_1) \\ l_3 + l_4 + \cos(\mathrm{th}_2)\ (l_6 + l_7) + l_5\sin(\mathrm{th}_2) \end{gathered}$$

```
% Initialization of xactual, yactual vectors
xactual=zeros(1,n_col);
yactual=zeros(1,n_col);
zactual=zeros(1,n_col);

for i=1:n_col
   xactual(i)= l5*cos(theta1(i))*cos(theta2(i)) -
cos(theta1(i))*sin(theta2(i))*(l6(i) + l7) + l1;
   yactual(i)= l5*cos(theta2(i))*sin(theta1(i)) -
sin(theta1(i))*sin(theta2(i))*(l6(i) + l7) + l2;
   zactual(i)= l3 + l4 + cos(theta2(i))*(l6(i) + l7) + l5*sin(theta2(i));
end
```

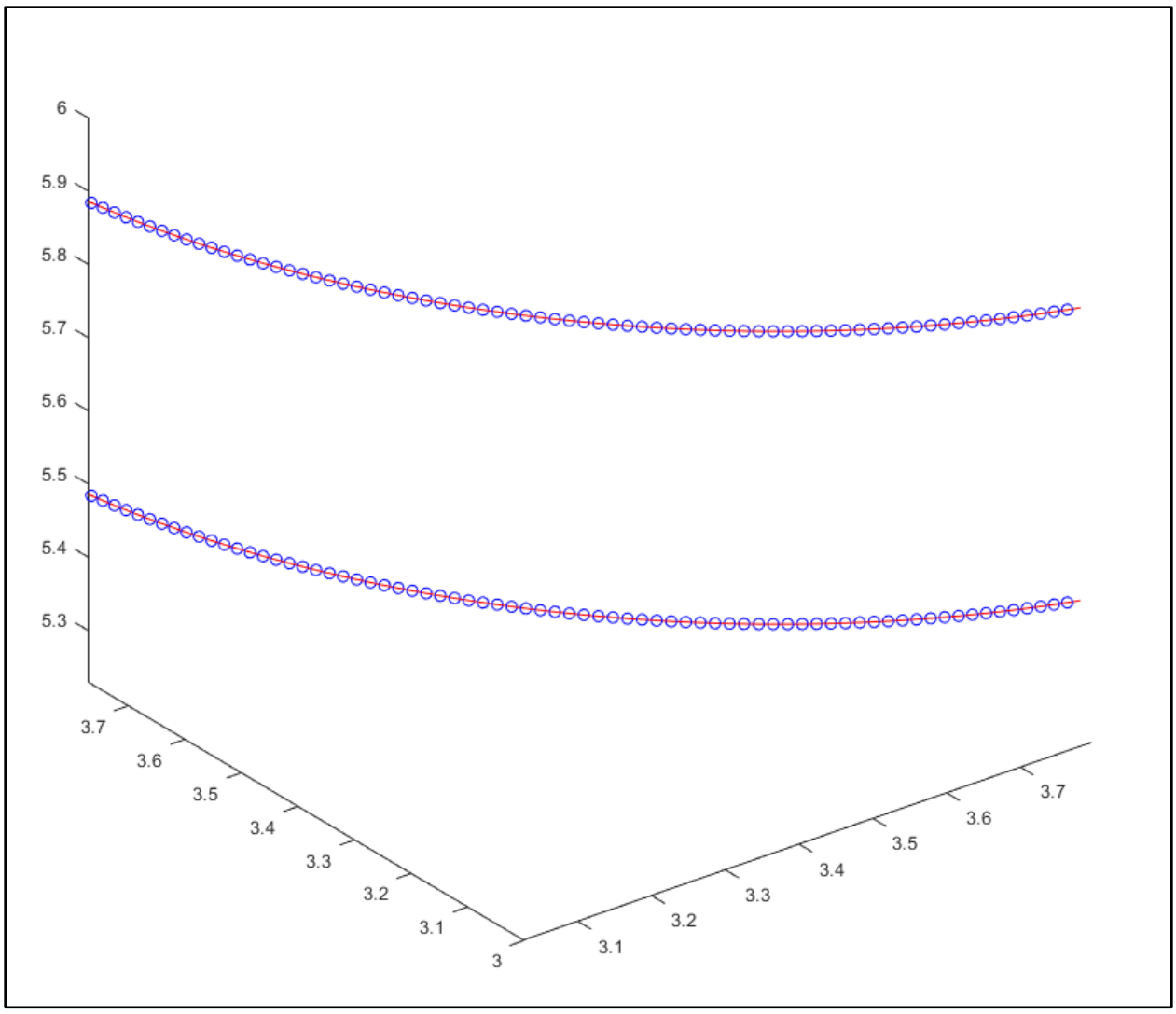
Once the actual trajectory is calculated, we can now superimpose the desired and actual trajectories over each other to determine the accuracy of our calculations. This can be observed in the plots presented below.

Comparison Plot of Desired vs Actual Trajectories

```
plot3(xactual,yactual,zactual,'r',Px,Py,Pz,'bo')
legend('Actual','Desired');
title('Desired vs Actual Trajectory')
```



In the plot above, we can see the two-part trajectory created earlier in this project. The desired trajectory is created with blue circles, so that the actual trajectory can be seen inside of it. As observed, we can accomplish complex trajectories without any obvious errors.



Zoomed Figure of Final Comparison