ROCO224 - Simulation of a 6DOF Robotic Arm

Coursework 2020

Student No. 10618407

Thursday 9th April, 2020

Contents

Chapter 1

6DOF (6 degrees of freedom) Robotic Arm

The 6DOF robotic arm is used throughout this project. It has six revolute joints, allowing it to have six degrees of freedom. The robot arm design is shown below, along with the subsequent coordinate frames and distances between them. These were used throughout the coursework. The 6DOF robotic arm model created in Matlab was based off this design:

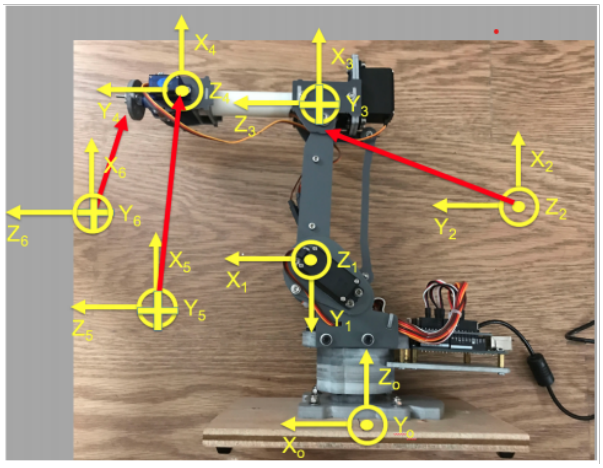


Figure 1.1: Coordinate frames for the 6DOF robot arm

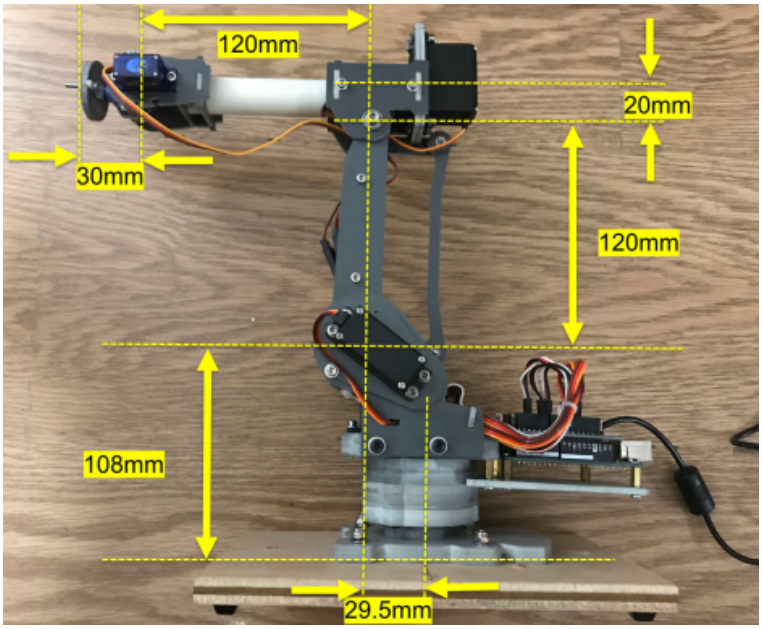


Figure 1.2: Distance between coordinate frames for the 6DOF robot arm

Chapter 2

Analysis of 6DOF robotic arm

2.1 Transformation between frame {1} to frame {0}

To transform between these coordinate frames, two movements are required. These are a rotation and a translation.

The rotation is 90° about the x-axis. This utilises the rotation matrix of:

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & c\theta & -s\theta \\ 0 & s\theta & c\theta \end{bmatrix}$$

The rotational matrix of the translation between frame {1} and frame {0} is:

$$\begin{matrix} x_1 \\ y_1 = \\ z_1 \end{matrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & -1 & 0 \end{bmatrix} \begin{bmatrix} x_0 \\ y_0 \\ z_0 \end{bmatrix}$$

The translation between these two frames is 29.5mm in the x-axis and 108mm in the z-axis. The translation matrix for this is:

$$\begin{bmatrix} t_{\mathrm{x}} \\ t_{\mathrm{y}} \\ t_{\mathrm{z}} \end{bmatrix} = \begin{bmatrix} 29.5 \\ 0 \\ 108 \end{bmatrix}$$

The rotation and translation matrices can be combined together to make a single homogeneous matrix. In Matlab these can be done both symbolically and numerically. Homogeneous equations allow both operations to be simultaneously calculated, as shown below:

```
function [ F01 ] = symbFrame01()

    %Symbols for translation matrix
    tx = sym('tx');
    ty = sym('ty');
    tz = sym('tz');

    %Symbols for rotation matix
    cosine  = sym('c(theta)');
    sinep   = sym('s(theta)');
    sinen   = sym('-s(theta)');

    %Rotation matrix
    Rotation = [ 1 0 0; 0 cosine sinen; 0 sinep cosine ];

    %Translation matrix
    Translation = [ tx; ty; tz ];

    %Full homogeneous transformtaion matrix
    F01 = [ Rotation Translation; 0 0 0 1 ];

end
```

Figure 2.1: Code for transforming between frame 1 and frame 0 symbolically in Matlab

```
function [F01] = zeroToOne ()
%angle to rotate by in radians
theta = -pi/2;
t = theta;
%matrix for the rotation
rotation = [1 0 0; 0 cos(t) -sin(t); 0 sin(t) cos(t)];
%matrix for the translation
translation = [29.5; 0; 108];
%matrix for the homogenous tranformation
F01 = [ rotation translation; 0 0 0 1;];
end
```

Figure 2.2: Code for transforming between frame 1 and frame 0 numerically in Matlab

For the transformation between frame $\{1\}$ and frame $\{0\}$ the homogeneous equation is as follows:

$$F01 = \begin{bmatrix} 1 & 0 & 0 & 29.5 \\ 0 & 0 & 1 & 0 \\ 0 & -1 & 0 & 108 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

```
>> zeroToOne ()

ans =

    1.0000         0         0   29.5000
         0    0.0000    1.0000         0
         0   -1.0000    0.0000  108.0000
         0         0         0    1.0000
```

Figure 2.3: Output transformation between frame 1 and frame 0 from Matlab

2.2 Transformation between frame {2} to frame {1}

To transform between these coordinate frames, two movements are required. These are a rotation and a translation.

The rotation is 90°about the z-axis. This utilises the rotation matrix of:

$$\begin{bmatrix} c\theta & -s\theta & 0 \\ s\theta & c\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

The rotational matrix of the translation between frame {2} and frame {1} is:

$$\begin{matrix} x_1 \\ y_1 \\ z_1 \end{matrix} = \begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ y_0 \\ z_0 \end{bmatrix}$$

The translation between these two frames is -120mm in the y-axis. The translation matrix for this is:

$$\begin{bmatrix} t_{\mathrm{x}} \\ t_{\mathrm{y}} \\ t_{\mathrm{z}} \end{bmatrix} = \begin{bmatrix} 0 \\ -120 \\ 0 \end{bmatrix}$$

The rotation and translation matrices can be combined together to make a single homogeneous matrix. In Matlab these can be done both symbolically and numerically. Homogeneous equations

allow both operations to be simultaneously calculated, as shown below:

```
function [ F12 ] = symbFrame12()

    %Symbols for translation matrix
    tx = sym('tx');
    ty = sym('ty');
    tz = sym('tz');

    %Symbols for rotation matix
    cosine  = sym('c(theta)');
    sinep   = sym('s(theta)');
    sinen   = sym('-s(theta)');

    %Rotation matrix
    Rotation = [ cosine sinen 0; sinep cosine 0; 0 0 1 ];

    %Translation matrix
    Translation = [ tx; ty; tz ];

    %Full homogeneous transformtaion matrix
    F12 = [ Rotation Translation; 0 0 0 1 ];

end
```

Figure 2.4: Code for transforming between frame 2 and frame 1 symbolically in Matlab

```
function [F12] = oneToTwo ()
%angle to rotate by in radians
theta = -pi/2;
t = theta;
%matrix for the rotation
rotation = [cos(t) -sin(t) 0; sin(t) cos(t) 0; 0 0 1];
%matrix for the translation
translation = [0; -120; 0];
%matrix for the homogenous tranformation
F12 = [rotation translation; 0 0 0 1];
end
```

Figure 2.5: Code for transforming between frame 2 and frame 1 numerically in Matlab

For the transformation between frame $\{2\}$ and frame $\{1\}$ the homogeneous equation is as follows:

$$F12 = \begin{bmatrix} 0 & 1 & 0 & 0 \\ -1 & 0 & 0 & -120 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

```
>> oneToTwo

ans =

    0.0000    1.0000         0         0
   -1.0000    0.0000         0 -120.0000
         0         0    1.0000         0
         0         0         0    1.0000
```

Figure 2.6: Output transformation between frame 2 and frame 1 from Matlab

2.3 Transformation between frame {3} to frame {2}

To transform between these coordinate frames, two movements are required. These are a rotation and a translation.

The rotation is 90° about the x-axis. This utilises the rotation matrix of:

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & c\theta & -s\theta \\ 0 & s\theta & c\theta \end{bmatrix}$$

The rotational matrix of the translation between frame {3} and frame {2} is:

$$\begin{matrix} x_1 \\ y_1 \\ z_1 \end{matrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & -1 & 0 \end{bmatrix} \begin{bmatrix} x_0 \\ y_0 \\ z_0 \end{bmatrix}$$

The translation between these two frames is 20mm in the x-axis. The translation matrix for this is:

$$\begin{bmatrix} t_{\mathrm{x}} \\ t_{\mathrm{y}} \\ t_{\mathrm{z}} \end{bmatrix} = \begin{bmatrix} 20 \\ 0 \\ 0 \end{bmatrix}$$

The rotation and translation matrices can be combined together to make a single homogeneous matrix. In Matlab these can be done both symbolically and numerically. Homogeneous equations

allow both operations to be simultaneously calculated, as shown below:

```
function [ F23 ] = symbFrame23()

    %Symbols for translation matrix
    tx = sym('tx');
    ty = sym('ty');
    tz = sym('tz');

    %Symbols for rotation matix
    cosine  = sym('c(theta)');
    sinep   = sym('s(theta)');
    sinen   = sym('-s(theta)');

    %Rotation matrix
    Rotation = [ 1 0 0; 0 cosine sinen; 0 sinep cosine ];

    %Translation matrix
    Translation = [ tx; ty; tz ];

    %Full homogeneous transformtaion matrix
    F23 = [ Rotation Translation; 0 0 0 1 ];

end
```

Figure 2.7: Code for transforming between frame 3 and frame 2 symbolically in Matlab

```
function [F23] = twoToThree ()
%angle to rotate by in radians
theta = -pi/2;
t = theta;
%matrix for the rotation
rotation = [1 0 0; 0 cos(t) -sin(t); 0 sin(t) cos(t)];
%matrix for the translation
translation = [20; 0; 0];
%matrix for the homogenous tranformation
F23 = [rotation translation; 0 0 0 1];
end
```

Figure 2.8: Code for transforming between frame 3 and frame 2 numerically in Matlab

For the transformation between frame {3} and frame {2} the homogeneous equation is as follows:

$$F23 = \begin{bmatrix} 1 & 0 & 0 & 20 \\ 0 & 0 & 1 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

```
>> twoToThree

ans =

    1.0000         0         0   20.0000
         0    0.0000    1.0000         0
         0   -1.0000    0.0000         0
         0         0         0    1.0000
```

Figure 2.9: Output transformation between frame 3 and frame 2 from Matlab

2.4 Transformation between frame {4} to frame {3}

To transform between these coordinate frames, two movements are required. Thse are a rotation and a translation.

The rotation is 90° about the x-axis. This utilises the rotation matrix of:

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & c\theta & s\theta \\ 0 & -s\theta & c\theta \end{bmatrix}$$

The rotational matrix of the translation between frame {4} and frame {3} is:

$$\begin{matrix} x_1 \\ y_1 \\ z_1 \end{matrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & -1 & 0 \end{bmatrix} \begin{bmatrix} x_0 \\ y_0 \\ z_0 \end{bmatrix}$$

The translation between these two frames is 120mm in the z-axis. The translation matrix for this is:

$$\begin{bmatrix} t_{\mathrm{x}} \\ t_{\mathrm{y}} \\ t_{\mathrm{z}} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 120 \end{bmatrix}$$

The rotation and translation matrices can be combined together to make a single homogeneous matrix. In Matlab these can be done both symbolically and numerically. Homogeneous equations

allow both operations to be simultaneously calculated, as shown below:

```
function [ F34 ] = symbFrame34()

    %Symbols for translation matrix
    tx = sym('tx');
    ty = sym('ty');
    tz = sym('tz');

    %Symbols for rotation matix
    cosine  = sym('c(theta)');
    sinep   = sym('s(theta)');
    sinen   = sym('-s(theta)');

    %Rotation matrix
    Rotation = [ 1 0 0; 0 cosine sinen; 0 sinep cosine ];

    %Translation matrix
    Translation = [ tx; ty; tz ];

    %Full homogeneous transformtaion matrix
    F34 = [ Rotation Translation; 0 0 0 1 ];

end
```

Figure 2.10: Code for transforming between frame 4 and frame 3 symbolically in Matlab

```
function [F34] = threeToFour ()
%angle to rotate by in radians
theta = pi/2;
t = theta;
%matrix for the rotation
rotation = [1 0 0; 0 cos(t) -sin(t); 0 sin(t) cos(t)];
%matrix for the translation
translation = [0; 0; 120];
%matrix for the homogenous tranformation
F34 = [rotation translation; 0 0 0 1];
end
```

Figure 2.11: Code for transforming between frame 4 and frame 3 numerically in Matlab

For the transformation between frame {4} and frame {3} the homogeneous equation is as follows:

$$F34 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 120 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

```
>> threeToFour

ans =

    1.0000         0         0         0
         0    0.0000   -1.0000         0
         0    1.0000    0.0000  120.0000
         0         0         0    1.0000
```

Figure 2.12: Output transformation between frame 4 and frame 3 from Matlab

2.5 Transformation between frame {5} to frame {4}

To transform between these coordinate frames, two movements are required. These are a rotation and a translation.

The rotation is 90° about the x-axis. This utilises the rotation matrix of:

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & c\theta & s\theta \\ 0 & -s\theta & c\theta \end{bmatrix}$$

The rotational matrix of the translation between frame {5} and frame {4} is:

$$\begin{matrix} x_1 \\ y_1 \\ z_1 \end{matrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & -1 & 0 \end{bmatrix} \begin{bmatrix} x_0 \\ y_0 \\ z_0 \end{bmatrix}$$

For the transformation between frame 5 and frame 4 there is no translation between the two coordinate frames, therefore the translation matrix is equal to 0.

$$\begin{bmatrix} t_{\mathrm{x}} \\ t_{\mathrm{y}} \\ t_{\mathrm{z}} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

The rotation and translation matrices can be combined together to make a single homogeneous matrix. In Matlab these can be done both symbolically and numerically. Homogeneous equations allow both operations to be simultaneously calculated, as shown below:

```
function [ F45 ] = symbFrame45()

    %Symbols for translation matrix
    tx = sym('tx');
    ty = sym('ty');
    tz = sym('tz');

    %Symbols for rotation matix
    cosine  = sym('c(theta)');
    sinep   = sym('s(theta)');
    sinen   = sym('-s(theta)');

    %Rotation matrix
    Rotation = [ 1 0 0; 0 cosine sinen; 0 sinep cosine ];

    %Translation matrix
    Translation = [ tx; ty; tz ];

    %Full homogeneous transformtaion matrix
    F45 = [ Rotation Translation; 0 0 0 1 ];

end
```

Figure 2.13: Code for transforming between frame 5 and frame 4 symbolically in Matlab

```
function [F45] = fourToFive ()
%angle to rotate by in radians
theta = -pi/2;
t = theta;
%matrix for the rotation
rotation = [1 0 0; 0 cos(t) -sin(t); 0 sin(t) cos(t)];
%matrix for the translation
translation = [0; 0; 0];
%matrix for the homogenous tranformation
F45 = [rotation translation; 0 0 0 1];
end
```

Figure 2.14: Code for transforming between frame 5 and frame 4 numerically in Matlab

For the transformation between frame $\{5\}$ and frame $\{4\}$ the homogeneous equation is as follows:

$$F45 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

```
>> fourToFive

ans =

    1.0000         0         0         0
         0    0.0000    1.0000         0
         0   -1.0000    0.0000         0
         0         0         0    1.0000
```

Figure 2.15: Output transformation between frame 5 and frame 4 from Matlab

2.6 Transformation between frame {6} to frame {5}

To transform between these coordinate frames, two movements are required. These are a rotation and a translation.

There is no rotation between the two coordinate frames for this transformation. Because of this, an identity matrix is used:

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

The rotational matrix of the translation between frame {6} and frame {5} is:

$$\begin{matrix} x_1 \\ y_1 \\ z_1 \end{matrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ y_0 \\ z_0 \end{bmatrix}$$

The translation between these two frames is 30mm in the z-axis. The translation matrix for this is:

$$\begin{bmatrix} t_{\mathrm{x}} \\ t_{\mathrm{y}} \\ t_{\mathrm{z}} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 30 \end{bmatrix}$$

The rotation and translation matrices can be combined together to make a single homogeneous matrix. In Matlab these can be done both symbolically and numerically. Homogeneous equations

allow both operations to be simultaneously calculated, as shown below:

```
function [ F56 ] = symbFrame56()

    %Symbols for translation matrix
    tx = sym('tx');
    ty = sym('ty');
    tz = sym('tz');

    %Symbols for rotation matix
    cosine  = sym('c(theta)');
    sinep   = sym('s(theta)');
    sinen   = sym('-s(theta)');

    %Rotation matrix
    Rotation = [ 1 0 0; 0 cosine sinen; 0 sinep cosine ];

    %Translation matrix
    Translation = [ tx; ty; tz ];

    %Full homogeneous transformtaion matrix
    F56 = [ Rotation Translation; 0 0 0 1 ];

end
```

Figure 2.16: Code for transforming between frame 6 and frame 5 symbolically in Matlab

```
function [F56] = fiveToSix ()
%angle to rotate by in radians
theta = 0;
t = theta;
%matrix for the rotation
rotation = [1 0 0; 0 cos(t) -sin(t); 0 sin(t) cos(t)];
%matrix for the translation
translation = [0; 0; 30];
%matrix for the homogenous tranformation
F56 = [rotation translation; 0 0 0 1];
end
```

Figure 2.17: Code for transforming between frame 6 and frame 5 symbolically in Matlab

For the transformation between frame $\{6\}$ and frame $\{5\}$ the homogeneous equation is as follows:

$$F56 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 30 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

```
>> fiveToSix

ans =

     1     0     0     0
     0     1     0     0
     0     0     1    30
     0     0     0     1
```

Figure 2.18: Output transformation between frame 6 and frame 5 in Matlab

2.7 Forward kinematics: frame {6} to frame {0}

To get from frame {6} to frame {0}, you need to multiply together the homogeneous matrices of the coordinate frames already calculated.

$$F06 = \begin{bmatrix} 0 & 0 & 1 & 179.5 \\ 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 248 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

```
function [ M ] = sixToZero(F01, F12, F23, F34, F45, F56)

%To get the overall homoegenous matrix times all of the frames together
M = F01 * F12 * F23 * F34 * F45 * F56;

end
```

Figure 2.19: Output transformation between frame 6 and frame 0 in Matlab

The matrix below is the homogeneous transformation matrix for the transformation between the frames {0} and {6}:

```
>> sixToZero(F01, F12, F23, F34, F45, F56)

ans =

    0.0000    0.0000    1.0000  179.5000
   -0.0000   -1.0000    0.0000    0.0000
    1.0000   -0.0000   -0.0000  248.0000
         0         0         0    1.0000
```

Figure 2.20: Output transformation between frame 6 and frame 0 in Matlab

Chapter 3

Analysis of 6DOF arm using classical Denavit-Hartenberg Parameters

3.1 Build the Denavit-Hartenberg table

3.1.1 What is Denavit-Hartenberg?

In mechanics, the Denavit-Hartenberg (DH) parameters are one convention for attaching links of a robot manipulator to a reference frame. Coordinate frames are attached to the joints between links using a transformation in reference to both [X] and [Z]. These can be used for every joint of a robot consisting of n links.

$$[T] = [Z_1][X_1][Z_2][X_2]...[Z_{\text{n-1}}][X_{\text{n-1}}][Z_{\text{n}}][X_{\text{n}}]$$

These reference frames are laid out in the following way:

1. z-axis: direction of the joint
2. x-axis: parallel to the common normal
3. y-axis: follows the right hand rule and is dictated by the z-axis and the x-axis

3.1.2 Denavit-Hartenberg Table

The Denavit-Hartenberg table, for the 6DOF robotic arm used in this coursework, is shown below:

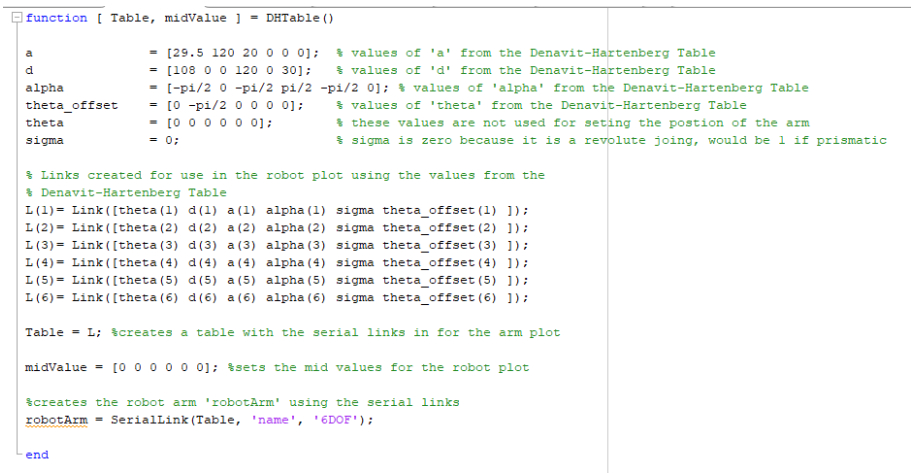
	θ	α	a	d
1	0	-π/2	29.5	108
2	-π/2	0	120	0
3	0	-π/2	20	0
4	0	π/2	0	120
5	0	-π/2	0	0
6	0	0	0	30

Where:

- θ is the angle around the previous z-axis, from the old x to the new x
- α is the angle around the common normal, from the old z to the new z
- a is the length of the common normal
- d is the offset along the previous z-axis

3.2 Forward kinematics

The following code is used to setup both a Denavit-Hartenberg table in Matlab and an environment for the robotic arm to be displayed. The Denavit-Hartenberg table creates all the serial links for the robotic arm. The mid values (θ) of 0 set the robotic arm to its default position.



```
function [ Table, midValue ] = DHTable()

a              = [29.5 120 20 0 0 0];  % values of 'a' from the Denavit-Hartenberg Table
d              = [108 0 0 120 0 30];   % values of 'd' from the Denavit-Hartenberg Table
alpha          = [-pi/2 0 -pi/2 pi/2 -pi/2 0]; % values of 'alpha' from the Denavit-Hartenberg Table
theta_offset   = [0 -pi/2 0 0 0 0];    % values of 'theta' from the Denavit-Hartenberg Table
theta          = [0 0 0 0 0 0];        % these values are not used for seting the postion of the arm
sigma          = 0;                    % sigma is zero because it is a revolute joing, would be 1 if prismatic

% Links created for use in the robot plot using the values from the
% Denavit-Hartenberg Table
L(1)= Link([theta(1) d(1) a(1) alpha(1) sigma theta_offset(1) ]);
L(2)= Link([theta(2) d(2) a(2) alpha(2) sigma theta_offset(2) ]);
L(3)= Link([theta(3) d(3) a(3) alpha(3) sigma theta_offset(3) ]);
L(4)= Link([theta(4) d(4) a(4) alpha(4) sigma theta_offset(4) ]);
L(5)= Link([theta(5) d(5) a(5) alpha(5) sigma theta_offset(5) ]);
L(6)= Link([theta(6) d(6) a(6) alpha(6) sigma theta_offset(6) ]);

Table = L; %creates a table with the serial links in for the arm plot

midValue = [0 0 0 0 0 0]; %sets the mid values for the robot plot

%creates the robot arm 'robotArm' using the serial links
robotArm = SerialLink(Table, 'name', '6DOF');

end
```

Figure 3.1: Code to setup the Denavit-Hartenberg table and the robotic arm

```
>> robotArm = SerialLink(Table, 'name', '6DOF')

robotArm =

6DOF (6 axis, RRRRRR, stdDH, fastRNE)

+---+-----------+-----------+-----------+-----------+-----------+
| j |     theta |         d |         a |     alpha |    offset |
+---+-----------+-----------+-----------+-----------+-----------+
|  1|         q1|        108|       29.5|     -1.571|          0|
|  2|         q2|          0|        120|          0|     -1.571|
|  3|         q3|          0|         20|     -1.571|          0|
|  4|         q4|        120|          0|      1.571|          0|
|  5|         q5|          0|          0|     -1.571|          0|
|  6|         q6|         30|          0|          0|          0|
+---+-----------+-----------+-----------+-----------+-----------+

grav =    0  base = 1  0  0  0   tool =  1  0  0  0
          0          0  1  0  0           0  1  0  0
       9.81          0  0  1  0           0  0  1  0
                     0  0  0  1           0  0  0  1
```

Figure 3.2: Output Denavit-Hartenberg table for the robot arm in Matlab

3.3 Display default configuration

The code below is used to create a static model of the 6DOF robotic arm. It utilises the serial links created using the Denavit-Hartenberg table. The robot arm is in its 'natural' position (with mid values equal to zero). The robot arm is plotted in a work space that is 1000x1000x600. This allows the robot to extend its full range of movements without going outside the bounds of the work

space.

```
function [ Table, midValue ] = DHTable()

a               = [29.5 120 20 0 0 0];  % values of 'a' from the Denavit-Hartenberg Table
d               = [108 0 0 120 0 30];   % values of 'd' from the Denavit-Hartenberg Table
alpha           = [-pi/2 0 -pi/2 pi/2 -pi/2 0]; % values of 'alpha' from the Denavit-Hartenberg Table
theta_offset    = [0 -pi/2 0 0 0 0];    % values of 'theta' from the Denavit-Hartenberg Table
theta           = [0 0 0 0 0 0];        % these values are not used for seting the postion of the arm
sigma           = 0;                    % sigma is zero because it is a revolute joint, would be 1 if prismatic

% Links created for use in the robot plot using the values from the
% Denavit-Hartenberg Table
L(1)= Link([theta(1) d(1) a(1) alpha(1) sigma theta_offset(1) ]);
L(2)= Link([theta(2) d(2) a(2) alpha(2) sigma theta_offset(2) ]);
L(3)= Link([theta(3) d(3) a(3) alpha(3) sigma theta_offset(3) ]);
L(4)= Link([theta(4) d(4) a(4) alpha(4) sigma theta_offset(4) ]);
L(5)= Link([theta(5) d(5) a(5) alpha(5) sigma theta_offset(5) ]);
L(6)= Link([theta(6) d(6) a(6) alpha(6) sigma theta_offset(6) ]);

Table = L; %creates a table with the serial links in for the arm plot

midValue = [0 0 0 0 0 0]; %sets the mid values for the robot plot

%creates the robot arm 'robotArm' using the serial links
robotArm = SerialLink(Table, 'name', '6DOF');

%plots the robot arm 'robotArm' into a plot using the midValues. The size
%of the work space is 1000x1000x600 (this is the area in which the robot operates
title('Student No: 10614838 - Static 6DOF arm')
robotArm.plot(midValue, 'noshadow', 'workspace', [-500 500 -500 500 0 600]);
end
```

Figure 3.3: Code to setup and plot the robotic arm

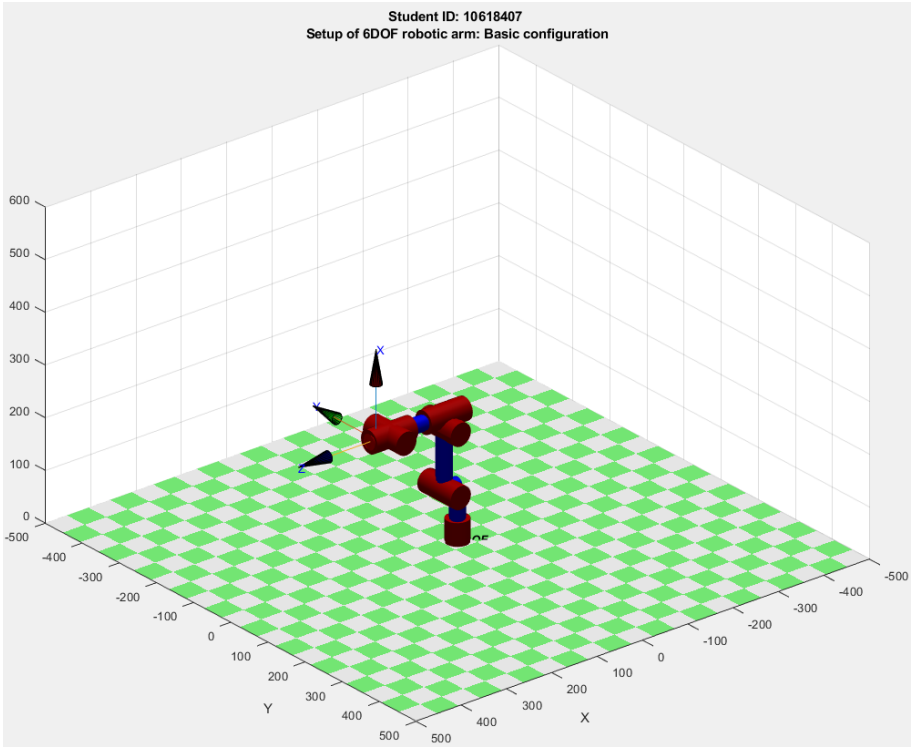


Figure 3.4: Setup of 6DOF robotic arm: Basic (natural) configuration

3.4 Build interactive model

The interactive mode allows the robot's joints to be manipulated using various sliders along the left hand side of the work space. This allows the range of the robot's movement to be analysed in conjunction with one another.

```
function [ Table, midValue ] = DHTable()

a               = [29.5 120 20 0 0 0];  % values of 'a' from the Denavit-Hartenberg Table
d               = [108 0 0 120 0 30];   % values of 'd' from the Denavit-Hartenberg Table
alpha           = [-pi/2 0 -pi/2 pi/2 -pi/2 0]; % values of 'alpha' from the Denavit-Hartenberg Table
theta_offset    = [0 -pi/2 0 0 0 0];    % values of 'theta' from the Denavit-Hartenberg Table
theta           = [0 0 0 0 0 0];        % these values are not used for seting the postion of the arm
sigma           = 0;                    % sigma is zero because it is a revolute joint, would be 1 if prismatic

% Links created for use in the robot plot using the values from the
% Denavit-Hartenberg Table
L(1)= Link([theta(1) d(1) a(1) alpha(1) sigma theta_offset(1) ]);
L(2)= Link([theta(2) d(2) a(2) alpha(2) sigma theta_offset(2) ]);
L(3)= Link([theta(3) d(3) a(3) alpha(3) sigma theta_offset(3) ]);
L(4)= Link([theta(4) d(4) a(4) alpha(4) sigma theta_offset(4) ]);
L(5)= Link([theta(5) d(5) a(5) alpha(5) sigma theta_offset(5) ]);
L(6)= Link([theta(6) d(6) a(6) alpha(6) sigma theta_offset(6) ]);

Table = L; %creates a table with the serial links in for the arm plot

midValue = [0 0 0 0 0 0]; %sets the mid values for the robot plot

%creates the robot arm 'robotArm' using the serial links
robotArm = SerialLink(Table, 'name', '6DOF');

%plots the robot arm 'robotArm' into a plot using the midValues. The size
%of the work space is 1000x1000x600 (this is the area in which the robot operates
title('Student No: 10614838 - Static 6DOF arm')
robotArm.teach(midValue, 'noshadow', 'workspace', [-500 500 -500 500 0 600]);
end
```

Figure 3.5: Code to setup and plot the robotic arm in teach mode

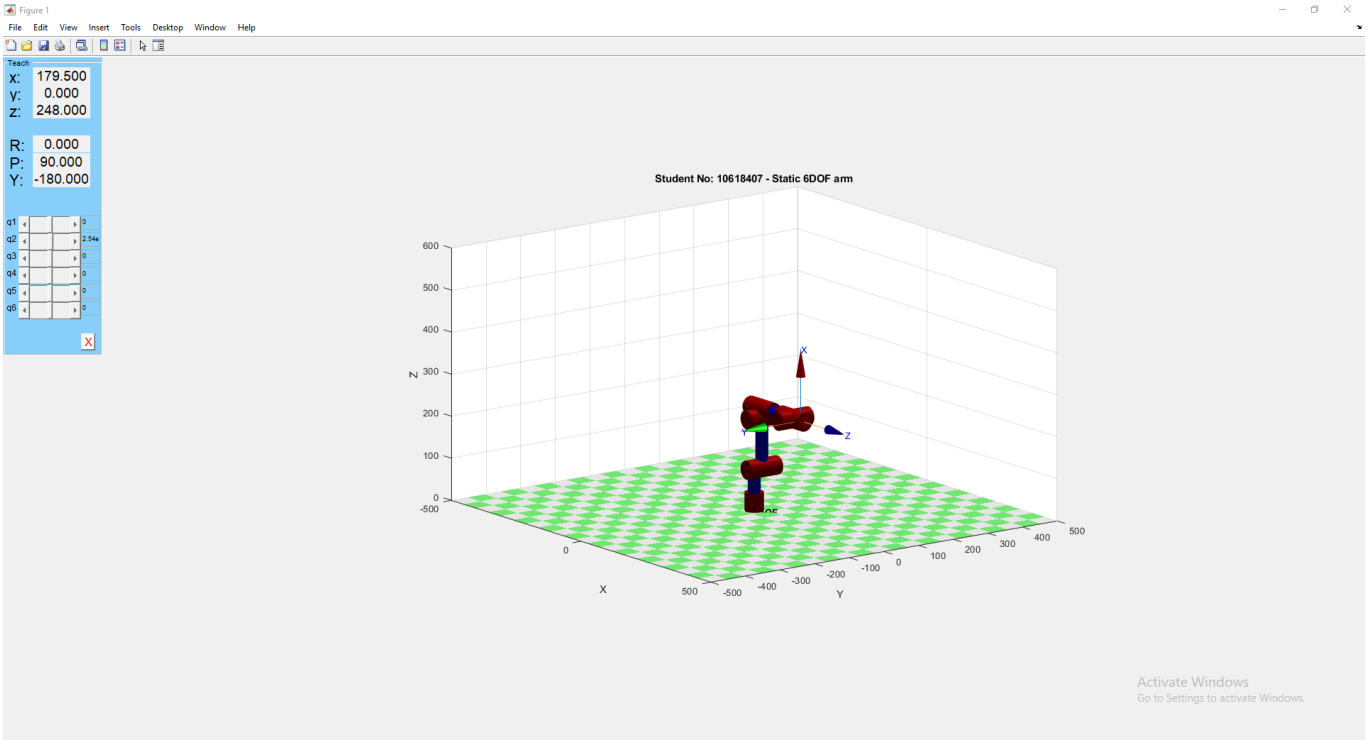


Figure 3.6: Robotic arm in teach mode

To see the teach mode working in a video use the following link: https://youtu.be/RaQ-lbTKuEQ.

3.5 Build a scatter plot of random arm locations

Building a scatter plot of random arm locations allows the evaluation of the positions the robot can be maneuvered into, along with its reach. The following code shows how the random points of the robotic arm were acquired and plotted onto the preexisting model. In total there are 10,000 random points plotted in the figure.

```
function [] = scatterplot(robotArm, midValue)

samples = 10000; %set the number of samples plotted on the diagram
%creating random values for theta to be used to plot the points
thetaRand = pi/2 +(-pi/2 - pi/2).* rand(samples, 6);
[TRand, ALL] = robotArm.fkine(thetaRand);

Rand=TRand.T;
% use squeeze to return an array with all the singletons removed
TP = squeeze(Rand( 1:3, 4, :));

%create a figure to plot the arm and the scattered points
figure

%plot the scattered points
h = plot3(TP(1,:), TP(2,:), TP(3,:), 'r.');
set(h, 'LineWidth', 3);

title('Student No: 10618407 - Scatterplot for 6DOF arm')
%plots the robot arm 'robotArm', along with all of the scatterplot points
%into a plot using the midValues. Thie size of the workspace is
%1000x1000x600 (this is the area in which the robot operates)
robotArm.plot(midValue, 'noshadow', 'workspace', [-500 500 -500 500 0 600]);

end
```

Figure 3.7: Code for plotting the scattered points in Matlab

Below are two views of the scattered points overlaid on the static arm configuration. These show the range of motion that the robot arm can theoretically achieve:

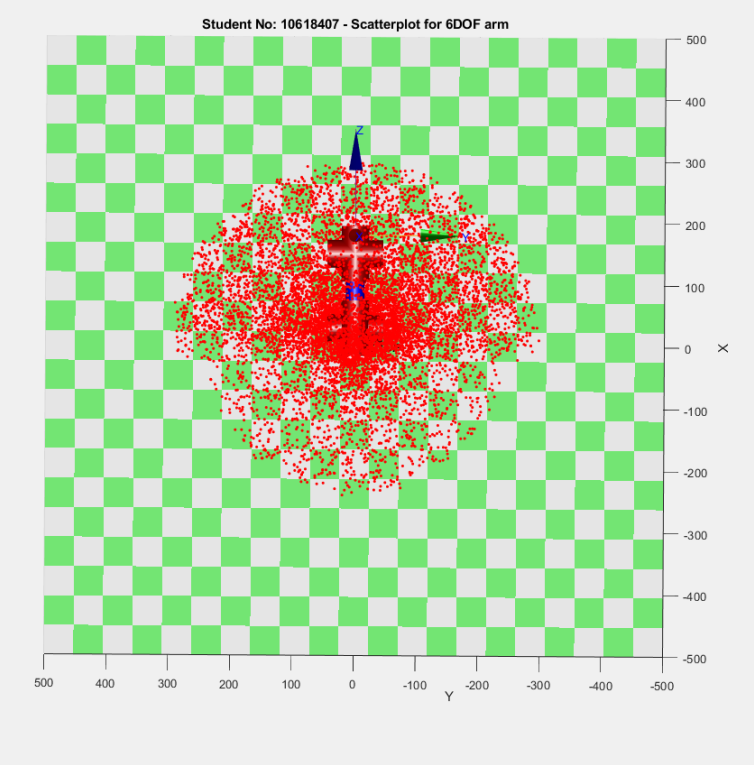


Figure 3.8: Static robotic arm with scattered points (top view)

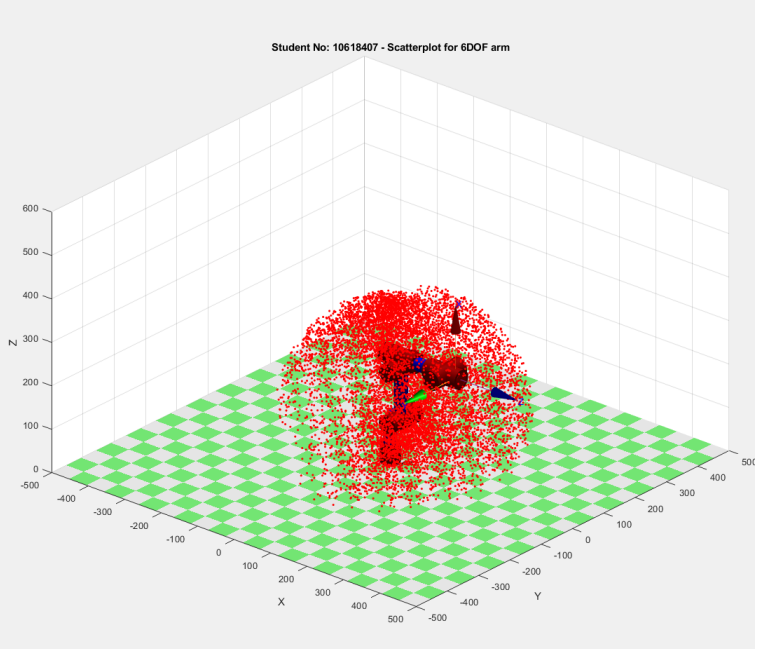


Figure 3.9: Static robotic arm with scattered points (isometric view)

3.6 Animated trajectory following

Animated trajectory following allows you to see how the joints of the robot move when the end effector is moving between given points inside its reach. The code below sets up the points for the robot arm to move between, along with the computation allowing the robot arm's end effector to follow the trajectory. The arm interpolates data between these points allowing it to move in the shortest path possible. All of these positions are put into a matrix and plotted using the robot arm.

```
function [ q ] = robotArmPath (steps, robotArm, robotName)

%define the targets for the square
%cartesian straigtline path between poses
T=[];
T(:,:,1) = transl(100, 200, 100); %defines the start point
T(:,:,2) = transl(100, 200, 100); %defines the start point
T(:,:,3) = transl(100, 200, 200); %defines the start point
T(:,:,4) = transl(100, -200, 200); %defines the start point
T(:,:,5) = transl(100, -200, 100); %defines the start point
T(:,:,6) = transl(100, 200, 100); %defines the start point
T(:,:,7) = transl(100, 200, 100); %defines the start point

%interpolate between targets with steps
TI =[];
for idx = 1 : (length(T)-1)
    offset = (idx - 1) * steps;
    tmp = ctraj(T(:,:,idx), T(:,:,idx + 1) , steps); %computes a cartesian path
    for tidx = 1:steps
        TI(:,:, tidx + offset) = tmp(:,:,tidx); %compute a cartesian path
    end
end

figure %creates a figure for the trajectory plot
TP = squeeze(T(1:3, 4, :)); %remove the singleton dimesions from the matrix
h = plot3(TP(1,:), TP(2,:), TP(3,:), 'k-'); %plot all of the points of the trajectory
set(h, 'LineWidth', 3); %set the width of the line for the trajectory
title('Student No: 10618407 - Path for 6DOF arm')
q = robotArm.ikunc(TI); %create a matrix of all the arm positions
%setting the limit of the robot
robotArm.qlim = [-pi/2 pi/2; -pi/2 pi/2; -pi/2 pi/2; -pi/2 pi/2; -pi/2 pi/2; -pi/2 pi/2;];
save(robotName, 'q'); %save the plotted trajectory points to the robot arm
robotArm.plot(q); %output the points onto the trajectory

end
```

Figure 3.10: Code for plotting the path for the robot arm to follow

Below is a static view of the robot arm and its path of travel:

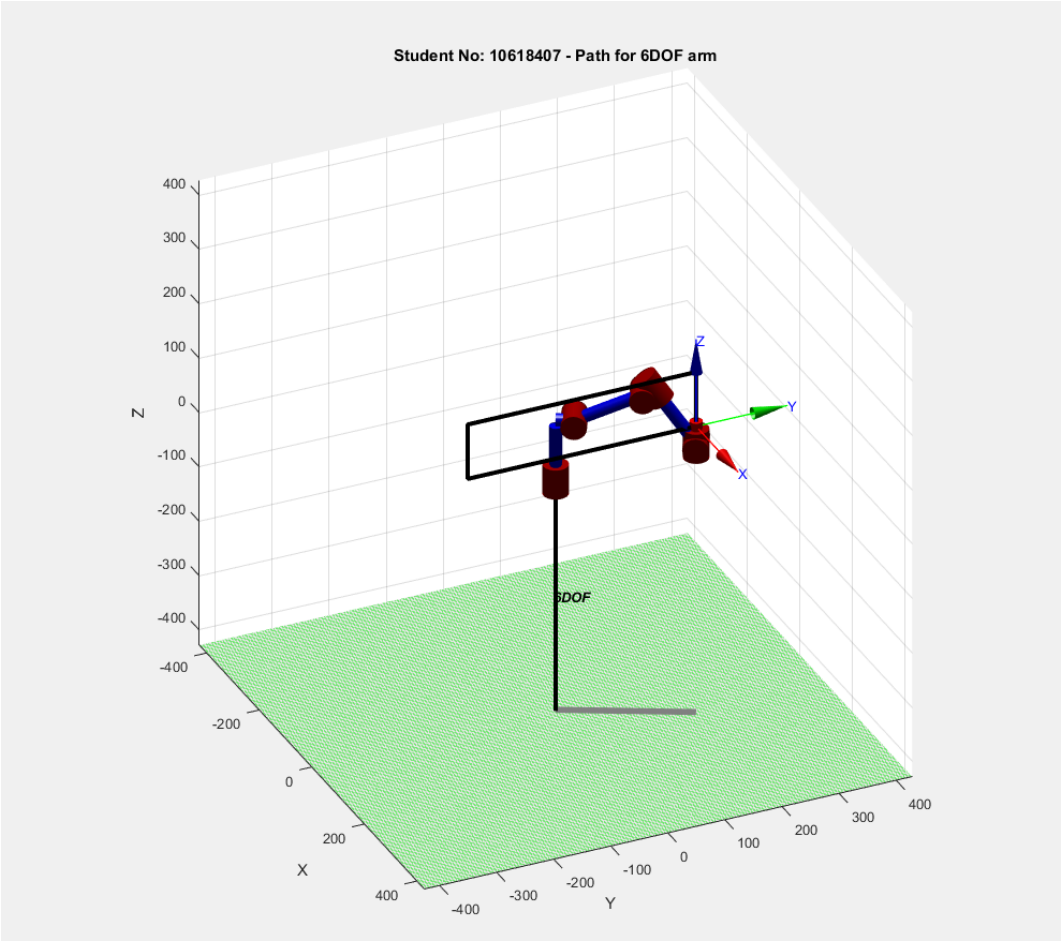


Figure 3.11: Static robotic arm with movement path

To see the robot arm following the path in a video use the following link: https://youtu.be/U7mfT64HJG0

As the end effector moves across a linear path, the joint angles change. The joint angles do not move linearly like the end effector. This is due to the nature of the movement of the robot. Because of this, it is important to be able to evaluate how the joint angles move with the end effector. This is used heavily in robotics to judge how an end effector or joints will move. This is shown in the graph below along with the code to produce to trajectory plot graph:

```
function [q1, q2, q3] = trajectoryPlot (q)

figure %create a figure
for i = 1:size(q)
    q1(i,1) = q(i,1);   %get the values for the first joint and put in q1
    q2(i,1) = q(i,2);   %get the values for the second joint and put in q2
    q3(i,1) = q(i,3);   %get the values for the third joint and put in q3
    q4(i,1) = q(i,4);   %get the values for the forth joint and put in q4
    q5(i,1) = q(i,5);   %get the values for the fith joint and put in q5
    q6(i,1) = q(i,6);   %get the values for the sitxh joint and put in q6
    hold on
end


hold on
h = plot(q1, 'r-.');    %plot q1 in red, dash-dot line
h = plot(q2, 'b-');     %plot q2 in blue, solid line
h = plot(q3, 'm--');    %plot q3 in magenta, dashed line
h = plot(q4, 'k:');     %plot q4 in black, dotted line
h = plot(q5, 'c');      %plot q5 in cyan, solid line (default)
h = plot(q6);           %plot line in random colour, solid line(default)

%title for the figure
title('Student No: 10618407 - Angular Trajectories for 6DOF arm')
%set the legend for the grapgh, top right hand corner, vertical
legend({'Q1','Q2','Q3','Q4','Q5','Q6'},'Location','northeast','Orientation','vertical');
%label the x-axis
xlabel('Time[arb]')
%label the y-axis
ylabel('Angle [rads]')

end
```

Figure 3.12: Matlab code to produce the trajectory plot graph

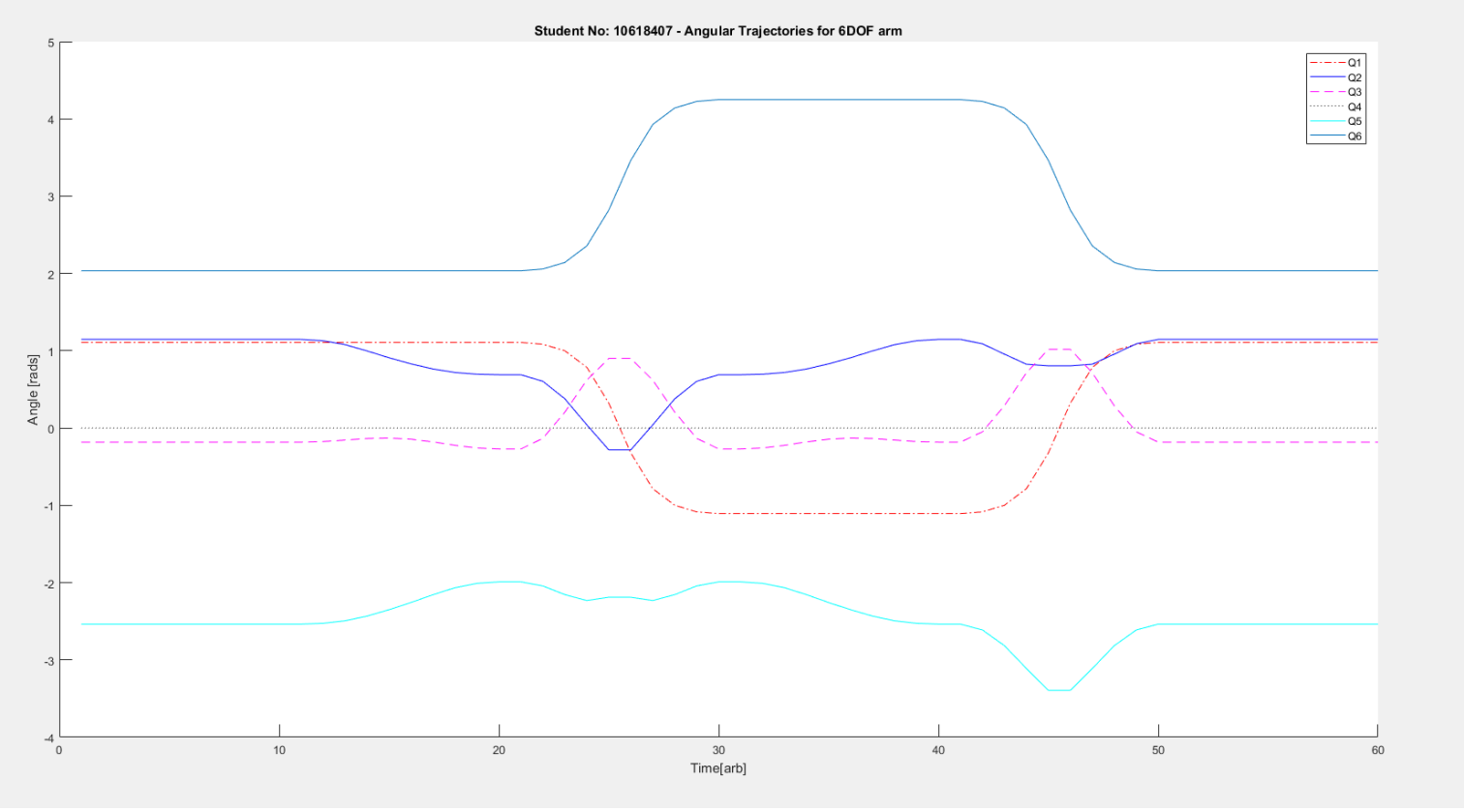


Figure 3.13: Angular trajectory of robot arm joints