Introduction to Robotics ID6040-Assignment 1

Siddharth Nayak EE16B073

The outputs for the code have been printed below.

```
In [1]: import numpy as np
```

```
In [2]: axes = input('Please enter the number of axes: ')
        print('Please enter the Matrix in the form :')
        print('Please enter all angles in degrees')
        print('[Theta , d , a , alpha ]')
```

```
Please enter the number of axes:  6


Please enter the Matrix in the form :
Please enter all angles in degrees
[Theta , d , a , alpha ]
```

```
In [3]: dh=[]
        # take user input
        for i in range(int(axes)):
            theta = float(input('theta_'+str(i+1)+': '))
            theta = float(np.deg2rad(theta))
            d = float(input('d_'+str(i+1)+': '))
            a = float(input('a_'+str(i+1)+': '))
            alpha = float(input('alpha_'+str(i+1)+': '))
            alpha = float(np.deg2rad(alpha))
            dh.append(np.array([theta,d,a,alpha]))
```

```
theta_1:  0
d_1:  200
a_1:  0
alpha_1:  90
theta_2:  0
d_2:  0
a_2:  100
alpha_2:  0
theta_3:  0
d_3:  0
a_3:  150
alpha_3:  0
```

```
theta_4:  0
d_4:  0
a_4:  100
alpha_4:  -90
theta_5:  -90
d_5:  0
a_5:  0
alpha_5:  -90
theta_6:  -90
d_6:  250
a_6:  0
alpha_6:  0
```

```
In [4]: class DH_transform():
            '''
            Class for all functions related to the DH parameters
            '''
            def __init__(self,dh):
                self.dh = dh

            def link_tf(self):
                '''
                Returns T(k-1 to k)
                '''
                dh = self.dh
                T = []
                for i in range(int(axes)):
                    T_i = np.zeros((4,4))
                    theta_k = dh[i][0]
                    d_k = dh[i][1]
                    a_k = dh[i][2]
                    alpha_k = dh[i][3]
                    # first row
                    T_i[0][0] = np.cos(theta_k)
                    T_i[0][1] = -np.cos(alpha_k)*np.sin(theta_k)
                    T_i[0][2] = np.sin(alpha_k)*np.sin(theta_k)
                    T_i[0][3] = a_k*np.cos(theta_k)

                    # second row
                    T_i[1][0] = np.sin(theta_k)
                    T_i[1][1] = np.cos(alpha_k)*np.cos(theta_k)
                    T_i[1][2] = -np.sin(alpha_k)*np.cos(theta_k)
                    T_i[1][3] = a_k*np.sin(theta_k)

                    # third row
                    T_i[2][0] = 0
                    T_i[2][1] = np.sin(alpha_k)
```

```python
            T_i[2][2] = np.cos(alpha_k)
            T_i[2][3] = d_k

            # fourth row
            T_i[3][0] = 0
            T_i[3][1] = 0
            T_i[3][2] = 0
            T_i[3][3] = 1

            T.append(T_i)
        return T

    def inverse_link_tf(self):
        '''
        Returns T(k to k-1)
        '''
        dh = self.dh
        T = []
        for i in range(int(axes)):
            T_i = np.zeros((4,4))
            theta_k = dh[i][0]
            d_k = dh[i][1]
            a_k = dh[i][2]
            alpha_k = dh[i][3]
            # first row
            T_i[0][0] = np.cos(theta_k)
            T_i[0][1] = np.sin(theta_k)#-np.cos(alpha_k)*np.sin(theta_k)
            T_i[0][2] = 0 #np.sin(alpha_k)*np.sin(theta_k)
            T_i[0][3] = -a_k#*np.cos(theta_k)

            # second row
            T_i[1][0] = -np.cos(alpha_k)*np.sin(theta_k)
            T_i[1][1] = np.cos(alpha_k)*np.cos(theta_k)
            T_i[1][2] = np.sin(alpha_k)#*np.cos(theta_k)
            T_i[1][3] = -d_k*np.sin(alpha_k)

            # third row
            T_i[2][0] = np.sin(alpha_k)*np.sin(theta_k)
            T_i[2][1] = -np.sin(alpha_k)*np.cos(theta_k)
            T_i[2][2] = np.cos(alpha_k)
            T_i[2][3] = -d_k*np.cos(alpha_k)

            # fourth row
            T_i[3][0] = 0
            T_i[3][1] = 0
            T_i[3][2] = 0
            T_i[3][3] = 1
```

```
            T.append(T_i)
        return T


    def tool_base(self,T):
        '''
        Takes in all the transformation matrices from base to tip
        and returns the base to tip transformation matrix
        '''
        T_i = T[0]
        for i in range(int(axes)-1):
            T_i = np.dot(T_i,T[i+1])
        return T_i

    def coords(self,T):
        '''
        Takes in matrix and splits it
        into translation vector and rotation matrix
        '''
        pos_vec = T[0:3,-1]
        theta_vec = T[0:3,0:3]
        print('Position vector wrt base:')
        print(pos_vec)
        print()
        print('Orientation(rotation matrix) Vector wrt base:')
        print(theta_vec)
```

```
In [5]: DH = DH_transform(dh)
```

The transformation matrix from base to elbow matches with the one calculated in question 2

```
In [6]: T=DH.link_tf()
        print('Transformation matrix from base to elbow')
        print(T[0]@T[1])
```

```
Transformation matrix from base to elbow
[[ 1.00000000e+00   0.00000000e+00   0.00000000e+00   1.00000000e+02]
 [ 0.00000000e+00   6.12323400e-17  -1.00000000e+00   0.00000000e+00]
 [ 0.00000000e+00   1.00000000e+00   6.12323400e-17   2.00000000e+02]
 [ 0.00000000e+00   0.00000000e+00   0.00000000e+00   1.00000000e+00]]
```

All the transformation matrices

```
In [14]: for i in range(len(T)):
             print("Transformation matrix from link_"+str(i) +' to link_'+str(i+1))
             print(T[i])
```

```
Transformation matrix from link_0 to link_1
[[  1.00000000e+00  -0.00000000e+00   0.00000000e+00   0.00000000e+00]
 [  0.00000000e+00   6.12323400e-17  -1.00000000e+00   0.00000000e+00]
 [  0.00000000e+00   1.00000000e+00   6.12323400e-17   2.00000000e+02]
 [  0.00000000e+00   0.00000000e+00   0.00000000e+00   1.00000000e+00]]
Transformation matrix from link_1 to link_2
[[   1.   -0.    0.  100.]
 [   0.    1.   -0.    0.]
 [   0.    0.    1.    0.]
 [   0.    0.    0.    1.]]
Transformation matrix from link_2 to link_3
[[   1.   -0.    0.  150.]
 [   0.    1.   -0.    0.]
 [   0.    0.    1.    0.]
 [   0.    0.    0.    1.]]
Transformation matrix from link_3 to link_4
[[  1.00000000e+00  -0.00000000e+00  -0.00000000e+00   1.00000000e+02]
 [  0.00000000e+00   6.12323400e-17   1.00000000e+00   0.00000000e+00]
 [  0.00000000e+00  -1.00000000e+00   6.12323400e-17   0.00000000e+00]
 [  0.00000000e+00   0.00000000e+00   0.00000000e+00   1.00000000e+00]]
Transformation matrix from link_4 to link_5
[[  6.12323400e-17   6.12323400e-17   1.00000000e+00   0.00000000e+00]
 [ -1.00000000e+00   3.74939946e-33   6.12323400e-17  -0.00000000e+00]
 [  0.00000000e+00  -1.00000000e+00   6.12323400e-17   0.00000000e+00]
 [  0.00000000e+00   0.00000000e+00   0.00000000e+00   1.00000000e+00]]
Transformation matrix from link_5 to link_6
[[  6.12323400e-17   1.00000000e+00  -0.00000000e+00   0.00000000e+00]
 [ -1.00000000e+00   6.12323400e-17  -0.00000000e+00  -0.00000000e+00]
 [  0.00000000e+00   0.00000000e+00   1.00000000e+00   2.50000000e+02]
 [  0.00000000e+00   0.00000000e+00   0.00000000e+00   1.00000000e+00]]
```

The cartesian space co-ordinates: position vector and orientation

```
In [15]: tool = DH.tool_base(T)
         coords_ = DH.coords(tool)
```

```
Position vector wrt base:
[  6.00000000e+02   1.53080850e-14   2.00000000e+02]

Orientation(rotation matrix) Vector wrt base:
[[ -6.12323400e-17   6.12323400e-17   1.00000000e+00]
 [ -6.12323400e-17  -1.00000000e+00   6.12323400e-17]
 [  1.00000000e+00  -6.12323400e-17   6.12323400e-17]]
```