Class: EML6281

Assignment: Homework 1

Name: Elias Reyes

In [2]:
```
import numpy as np
from numpy import square as sqr
import math
from math import sin
from math import cos
```

Problem 2.3

In [3]:
```
## part a - determine C_T_D
## C_T_D = C_T_A * A_T_B * B_T_D

# C_T_A is given
C_T_A = np.matrix([[np.sqrt(2.0)/2.0, np.sqrt(2.0)/2.0, 0, 0],[np.sqrt(2.0)/2.0, -np.sqrt(2)/2, 
# Calculate A_T_B
B_T_A  = np.matrix([[np.sqrt(3.0)/2.0, 0, 0.5, 20],[0 , -1, 0, 0],[0.5, 0, -np.sqrt(3)/2, 0],[0,
B_P_A0 = B_T_A[0:-1,-1]
A_R_B  = B_T_A[0:3,0:3].transpose()
A_T_B  = np.concatenate((np.concatenate((A_R_B,-A_R_B*B_P_A0),axis=1),[[0, 0, 0, 1]]),axis=0)
# Calculate B_T_D
D_T_B = np.matrix([[1, 0, 0, 0],[0, np.sqrt(3.0)/2, 0.5, 10],[0, -0.5, np.sqrt(3.0)/2, 0],[0, 0,
D_P_B0 = D_T_B[0:-1,-1]
B_R_D  = D_T_B[0:3,0:3].transpose()
B_T_D  = np.concatenate((np.concatenate((B_R_D,-B_R_D*D_P_B0),axis=1),[[0, 0, 0, 1]]),axis=0)

# Calculate C_T_D
C_T_D = np.matmul(np.matmul(C_T_A, A_T_B), B_T_D)
print('Transformation C_T_D:')
print(C_T_D)

## part b
D_P1 = np.matrix([20,-30,5,1]).transpose() # given coordinate, add 1
A_P1 = np.matmul(np.matmul(A_T_B,B_T_D),D_P1)
B_P1 = np.matmul(B_T_D,D_P1)
C_P1 = np.matmul(C_T_D,D_P1)
#
print('Coordinates of point 1 measured in A:')
print(A_P1)
print('Coordinates of point 1 measured in B:')
print(B_P1)
print('Coordinates of point 1 measured in C:')
print(C_P1)
```

```
Transformation C_T_D:
[[  0.61237244  -0.43559574   0.65973961  -7.89149131]
 [  0.61237244   0.78914913  -0.04736717 -20.13894002]
 [ -0.5          0.4330127    0.75        15.66987298]
 [  0.           0.           0.           1.        ]]
Coordinates of point 1 measured in A:
[[-7.83493649]
 [37.14101615]
 [13.57050808]
 [ 1.        ]]
Coordinates of point 1 measured in B:
[[ 20.        ]
 [-37.14101615]
 [-15.66987298]
 [  1.        ]]
Coordinates of point 1 measured in C:
[[ 20.72252766]
 [-31.8028011 ]
 [ -3.57050808]
 [  1.        ]]
```

Problem 2.4

In [4]:

```python
def determine_transformation_about_vector(m, theta):
    # normalize the vector which is being rotated about
    m_hat = m / np.linalg.norm(m)

    mx = m_hat[0]
    my = m_hat[1]
    mz = m_hat[2]
    # convert theta to radians
    theta = theta*math.pi/180
    # term for simplification
    v = 1 - cos(theta)
    # transformation to A from B
    A_R_B = np.array([[sqr(mx)*v + cos(theta), mx*my*v-mz*sin(theta), mx*mz*v+my*sin(theta)],
                      [mx*my*v+mz*sin(theta), sqr(my)*v + cos(theta), my*mz*v-mx*sin(theta)],
                      [mx*mz*v-my*sin(theta), my*mz*v + mx*sin(theta), sqr(mz)*v+cos(theta)]])

    return A_R_B


#A_R_B  = determine_transformation_about_vector([2,4,7],60)
A_P_C0 = np.array([[3,4,-2]]).T
A_R_C  = np.identity(3)
A_T_C  = np.concatenate((np.concatenate((A_R_C,A_P_C0),axis=1),[[0,0,0,1]]),axis=0)

C_R_D = determine_transformation_about_vector([2,4,7],60)
C_P_D0 = np.array([[0,0,0]]).T
C_T_D = np.concatenate((np.concatenate((C_R_D,C_P_D0),axis=1),[[0,0,0,1]]),axis=0)

D_P_B0 = np.array([[-3,-4,2]]).T
D_R_B = np.identity(3)
D_T_B = np.concatenate((np.concatenate((D_R_B,D_P_B0),axis=1),[[0,0,0,1]]),axis=0)

# A_T_B = A_T_C*C_T_D*D_T_B
A_T_B = np.matmul(np.matmul(A_T_C,C_T_D),D_T_B)
print('Transformation that relates A and B:')
print(A_T_B)
```

```
Transformation that relates A and B:
[[ 0.52898551 -0.67182943  0.5184781   5.13731742]
 [ 0.78777146  0.61594203 -0.00561586 -0.83831423]
 [-0.31557955  0.41141296  0.85507246 -0.98876827]
 [ 0.          0.          0.          1.        ]]
```

Problem 2.8

In [5]:
```
A_P_C0 = np.array([[10,20,10]]).T
A_R_C  = np.identity(3)
A_T_C  = np.concatenate((np.concatenate((A_R_C,A_P_C0),axis=1),[[0,0,0,1]]),axis=0)
C_R_D  = determine_transformation_about_vector([1,0,0],30)
C_P_D0 = np.array([[0,0,0]]).T
C_T_D  = np.concatenate((np.concatenate((C_R_D,C_P_D0),axis=1),[[0,0,0,1]]),axis=0)
D_P_B0 = np.array([[-10,-20,-10]]).T
D_R_B = np.identity(3)
D_T_B = np.concatenate((np.concatenate((D_R_B,D_P_B0),axis=1),[[0,0,0,1]]),axis=0)
A_T_B = np.matmul(np.matmul(A_T_C,C_T_D),D_T_B)

# Calculate B_T_A
A_P_B0 = np.array([A_T_B[0:-1,-1]]).T
B_R_A  = A_T_B[0:3,0:3].transpose()

B_T_A  = np.concatenate((np.concatenate((B_R_A,-B_R_A@A_P_B0),axis=1),[[0, 0, 0, 1]]),axis=0)

# Calculate A_T_C
A_R_C   = determine_transformation_about_vector([2,4,6],60)
A_P_C02 = np.array([[0,0,0]]).T
A_T_C   = np.concatenate((np.concatenate((A_R_C,A_P_C02),axis=1),[[0, 0, 0, 1]]),axis=0)

# Calculate B_T_C
B_T_C = np.matmul(B_T_A,A_T_C)

print('Transformation that relates C and B:')
print(B_T_C)
```

```
Transformation that relates C and B:
[[ 0.53571429 -0.6229365   0.57005291  0.        ]
 [ 0.48531316  0.77960099  0.39584523 -2.32050808]
 [-0.69100025  0.06459423  0.71996267 11.33974596]
 [ 0.          0.          0.          1.        ]]
```

In [ ]: