Hochschule Bonn-Rhein-Sieg

Mathematics for Robotics and Control, SS17

Assignment 1 - Frames

This week's assignment is about frames of reference. As you have learned in the lecture, the concept of frames is of great importance in robotics.

Let us consider a mobile robot (e.g. a youBot) that delivers packages in a lab. The robot is equipped with several sensors, including a camera for perceiving its environment and a gripper for grasping objects. You will use your knowledge of frames to help our robot complete its tasks.

Let us first setup this notebook so that figures and plots can be shown in the notebook page. Once you run the following cell, you don't have to import any of the packages in the subsequent code cells, as they will be available to all of them.

In [48]:

```
import numpy as np
import numpy.linalg as linalg
import matplotlib.pyplot as plt

from IPython.core.pylabtools import figsize, getfigs
np.set_printoptions(suppress=True)
```

Hint: You might want to check the the NumPy manual [1] before you start. In particular, read and understand the following functions:

```
array()
asarray()
sin()
cos()
tan()
radians()
hstack()
vstack()
dot()
delete()
linalg.inv()
linalg.det()
```

[1] http://docs.scipy.org/doc/numpy/genindex.html

Picking up a package for the lab

The robot's task for today is to go to the reception and pick up a package that is lying on a cabinet. To do so, the robot has to complete a few subtasks.

Locate the pose of the reception's door relative to the robot's base frame

Assume that our robot is located in a hallway that leads to the reception. In order to go inside the reception, the robot needs to know the pose of the door $\{D\}$ relative to the base frame $\{B\}$, i.e. we need the transform ${}^B_D T$; however, we are given the pose of the door relative to the camera's frame $\{C\}$, as we are using a camera for detecting the reception's door.

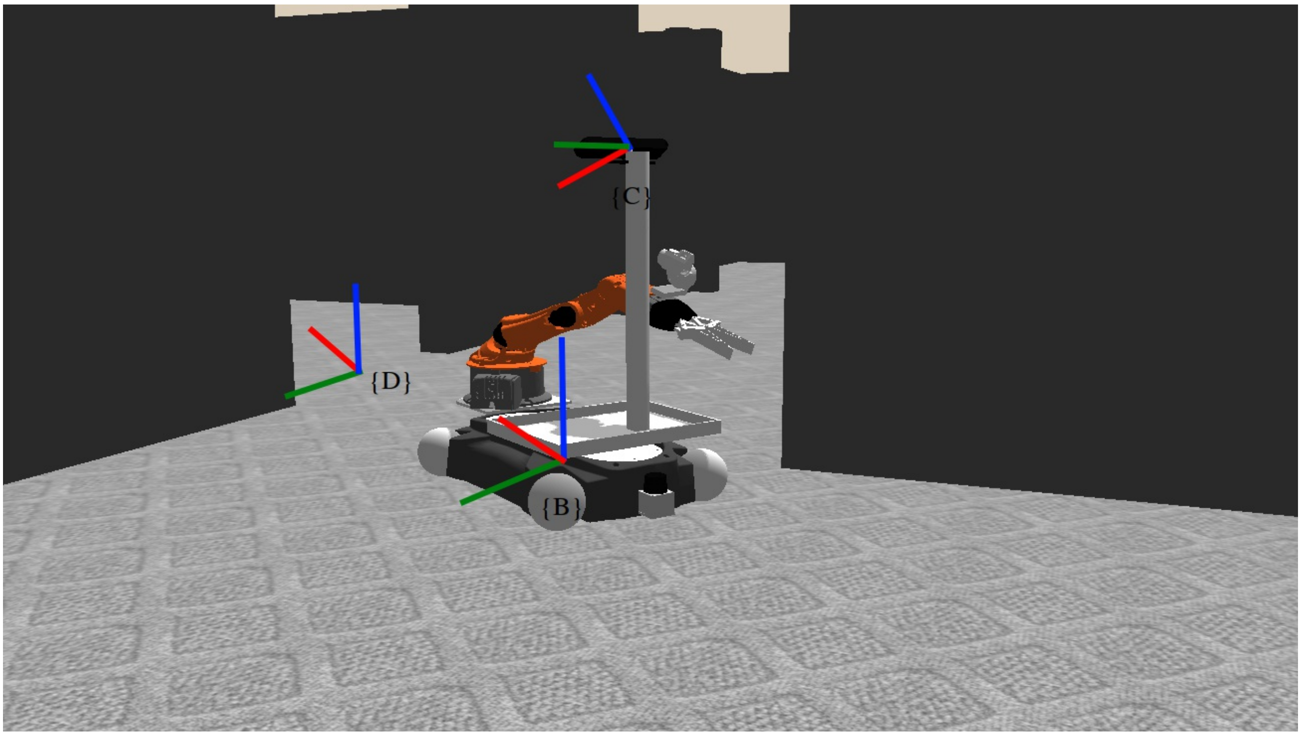
With respect to the camera, the door frame $\{D\}$ is rotated -13.215^o about Z and -28.647^o about Y (this a rotation about Z first, followed by a rotation in Y using the *Z-Y-X*-Euler-angle convention) and has a relative translation of $(1.533, -0.354, 0.197)$ meters in X, Y, and Z respectively. We also know the pose of the camera relative to the base, ${}^B_C T$: $\{C\}$ is located $(-0.176, 0.035, 0.563)$ meters away from the base frame and is rotated 28.647^o about $\{B\}$'s Y axis.

Observe the following figure for a visual description of the frames. The X axis is represented by the red line, the Y axis by the green line, and the Z axis by the blue line.

In [2]:

```
import IPython
IPython.core.display.Image("images/youbot_and_door.png", embed=True)
```

Out[2]:



Calculate ${}^B_D T$ by completing the following function.

In [11]:

```
def get_rotation_matrix_z_axis(theta):
    A_B_R = np.array([[np.cos(theta), -np.sin(theta), 0.],
                      [np.sin(theta), np.cos(theta),  0.],
                      [0.,            0.,             1.]])
    return A_B_R

def get_rotation_matrix_x_axis(theta):
    A_B_R = np.array([[1., 0.,            0.],
                      [0., np.cos(theta), -np.sin(theta)],
                      [0., np.sin(theta), np.cos(theta)]])
    return A_B_R

def get_rotation_matrix_y_axis(theta):
    A_B_R = np.array([[np.cos(theta),  0., np.sin(theta)],
                      [0.,             1., 0.],
                      [-np.sin(theta), 0., np.cos(theta)]])
    return A_B_R

def direct_transform(Rzyx, t):

    """
    This function returns a homogenous transformation describing
    """

    #transposing the translate vector to get a column vector
    t = t[np.newaxis].T

    # stacking translation horizontally
    T = np.hstack((Rzyx, t))

    # stacking  vertically
    T = np.vstack((T, np.array([0., 0., 0., 1])))

    return T


print""
print""
print "Step 1"
print "homogeneous transformation of {D} with respect to {C}"
print "-------------------------------------------------------"

C_D_theta_z=np.deg2rad(-13.215)
C_D_theta_y=np.deg2rad(-28.647)

#rotation matrix {D} with respect to {C}
C_D_Rz = get_rotation_matrix_z_axis(C_D_theta_z) #rotaion of frame D with r
eference to C about Z-axis
C_D_Ry = get_rotation_matrix_y_axis(C_D_theta_y) #rotaion of frame D with r
eference to C about Y-axis

C_D_Rzyx = np.dot(C_D_Rz,C_D_Ry) #final rotaion of frame D with reference t
o C

# translation vector
C_D_t = np.array([1.533, -0.354, 0.197])

#homogeneous transformation of {D} with respect to {C}
C_D_T=direct_transform(C_D_Rzyx, C_D_t)
```

```
print C_D_T


print""
print""
print "Step 2"
print "homogeneous transformation of {C} with respect to {B}"
print "-------------------------------------------------------"

B_C_theta_y=np.deg2rad(28.647)

#rotation matrix {C} with respect to {B}
B_C_Ry = get_rotation_matrix_y_axis(B_C_theta_y) #rotaion of frame C with r
eference to B about Y-axis

# translation vector
B_C_t = np.array([-0.176,0.035,0.563])

#homogeneous transformation of {D} with respect to {C}
B_C_T=direct_transform(B_C_Ry, B_C_t)

print B_C_T

print""
print""
print "Step 3"
print "homogeneous transformation of {D} with respect to {B}"
print "-------------------------------------------------------"


B_D_T= np.dot(B_C_T,C_D_T)
print B_D_T
```

```
Step 1
homogeneous transformation of {D} with respect to {C}
-------------------------------------------------------
[[ 0.85435062  0.22860574 -0.46671665  1.533     ]
 [-0.20062212  0.97351909  0.10959632 -0.354     ]
 [ 0.47941191  0.          0.87759001  0.197     ]
 [ 0.          0.          0.          1.        ]]


Step 2
homogeneous transformation of {C} with respect to {B}
-------------------------------------------------------
[[ 0.87759001  0.          0.47941191 -0.176     ]
 [ 0.          1.          0.          0.035     ]
 [-0.47941191  0.          0.87759001  0.563     ]
 [ 0.          0.          0.          1.        ]]


Step 3
homogeneous transformation of {D} with respect to {B}
-------------------------------------------------------
[[ 0.97960535  0.20062212  0.01114124  1.26378963]
 [-0.20062212  0.97351909  0.10959632 -0.319     ]
 [ 0.01114124 -0.10959632  0.99391374  0.00094677]
 [ 0.          0.          0.          1.        ]]
```

Reaching for a package

Once the robot has successfully entered the reception and approached the cabinet where the package is, it uses an object detection and recognition module for finding the package relative to the camera frame $\{C\}$. The module reports that, relative to frame $\{C\}$, the package is located at $(1.124, -0.060, 0.473)$ meters in X, Y, and Z, respectively, and is rotated by -28.647^o about Y. This information corresponds to the transform ${}^C_PT$.

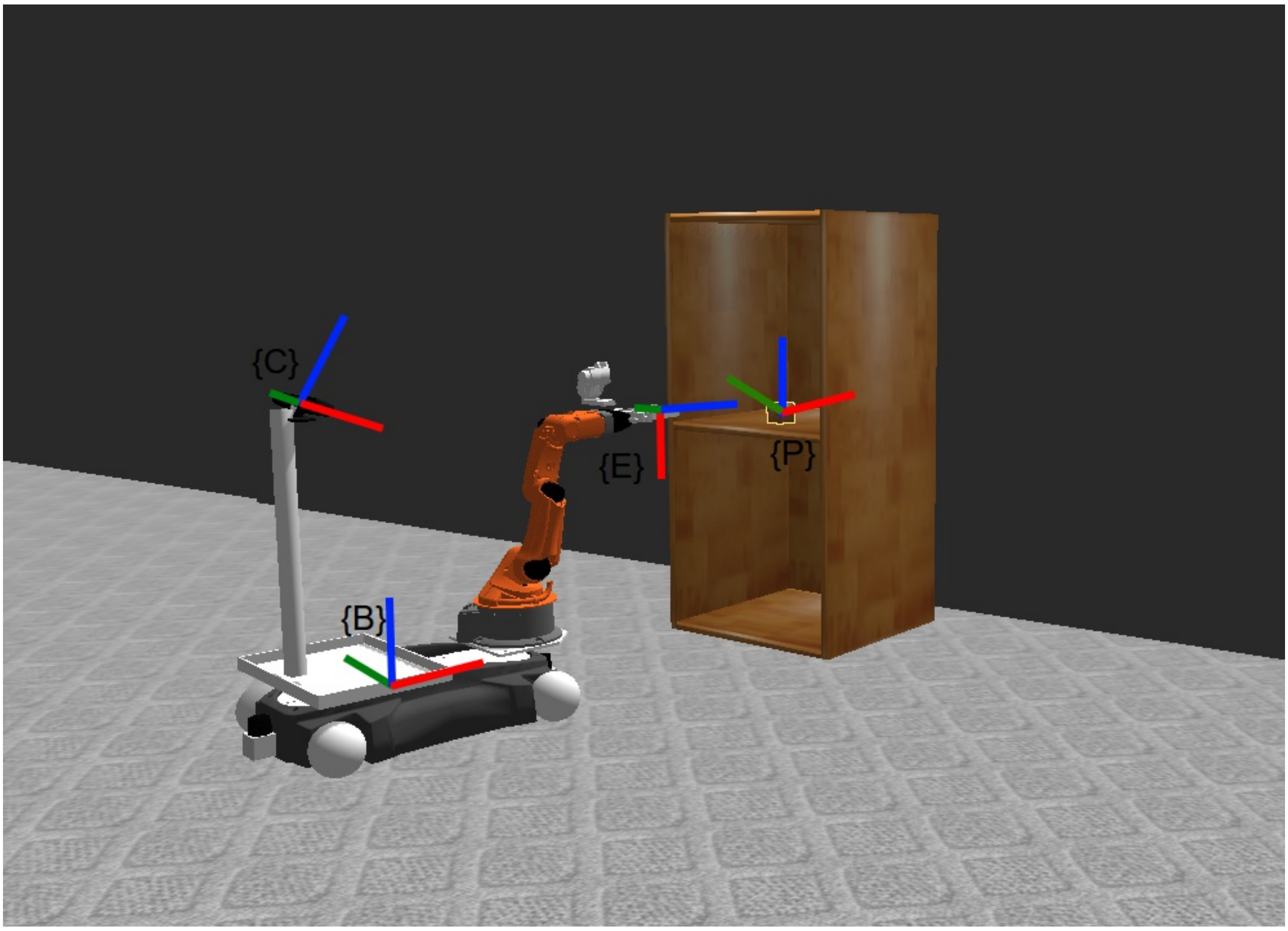
From the previous subtask, we know the location of the camera relative to the robot's base (given by ${}^B_CT$). Furthermore, using the robot's kinematics, we can calculate ${}^B_ET$, the transform describing the frame of the manipulator's end-effector $\{E\}$ relative to the base frame $\{B\}$; this is given by a translation of $(0.679, -0.019, 0.445)$ meters in X, Y, and Z respectively and a rotation of 90^o about Y.

For picking up this package, the robot needs to know the package's position and orientation with respect to its end-effector. Your task is to calculate the pose of the package relative to the manipulator's end-effector, namely compute ${}^E_PT$. The following figure shows a description of the frames involved.

In [4]:

```
IPython.core.display.Image("images/youbot_and_package.png", embed=True)
```

Out[4]:



Calculate ${}^E_PT$ by completing the following function.

In [50]:

```
def inverse_transform(Rzyx, t):
    """
    This function returns a homogenous transformation describing
    the pose of frame {P} relative to frame {E}.

    """
    Rzyx_T = Rzyx.T #Transpose of B_E_Ry

    #transposing the translate vector to get a column vector
    t = t[np.newaxis].T

    t=np.dot(-Rzyx_T,t)

    # stacking translation horizontally
    T = np.hstack((Rzyx_T, t))

    # stacking  vertically
    T = np.vstack((T, np.array([0., 0., 0., 1])))

    return T

print""
print""
print "homogeneous transformation of {P} with respect to {C}"
print "-------------------------------------------------------"

C_P_theta_y=np.deg2rad(-28.647)

#rotation matrix {P} with respect to {C}
C_P_Ry = get_rotation_matrix_y_axis(C_P_theta_y) #rotaion of frame P with r
eference to C about Y-axis

# translation vector
C_P_t = np.array([1.124,-0.060,0.473])

#homogeneous transformation of {P} with respect to {C}
C_P_T=direct_transform(C_P_Ry, C_P_t)

print C_P_T


print""
print""
print "homogeneous transformation of {C} with respect to {B}"
print "-------------------------------------------------------"

print B_C_T

print""
print""
print "homogeneous transformation of {E} with respect to {B}"
print "-------------------------------------------------------"

B_E_theta_y=np.deg2rad(90)

#rotation matrix {E} with respect to {B}
B_E_Ry = get_rotation_matrix_y_axis(B_E_theta_y) #rotaion of frame P with r
eference to C about Y-axis

#translation vector
```

```
#translation vector
B_E_t = np.array([0.679,-0.019,0.445])

#homogeneous transformation of {E} with respect to {B}
B_E_T=direct_transform(B_E_Ry, B_E_t)

print B_E_T

print""
print""
print "homogeneous transformation of {B} with respect to {E}"
print "-------------------------------------------------------"

#homogeneous transformation of {E} with respect to {B}
E_B_T=inverse_transform(B_E_Ry, B_E_t)

print E_B_T

'''
print ""
print ""
print"**********************************************************************
*******"
print"OR, the same transform can be found by directly taking an inverse of
B_E_T"
print"**********************************************************************
*******"
print "homogeneous transformation of {B} with respect to {E}--II"
print "-------------------------------------------------------"

E_B_T= linalg.inv(B_E_T)
print E_B_T
'''
print""
print""
print"````````````````````````````````````````````````````````````"
print"````````````````````````````````````````````````````````````"
print"Finally,"
print "homogeneous transformation of {P} with respect to {E}"
print "-------------------------------------------------------"

E_P_T=np.dot(E_B_T,B_C_T,C_P_T)

print E_P_T
```

```
homogeneous transformation of {P} with respect to {C}
-------------------------------------------------------
[[ 0.87759001  0.         -0.47941191  1.124     ]
 [ 0.          1.          0.         -0.06      ]
 [ 0.47941191  0.          0.87759001  0.473     ]
 [ 0.          0.          0.          1.        ]]


homogeneous transformation of {C} with respect to {B}
-------------------------------------------------------
[[ 0.87759001  0.          0.47941191 -0.176     ]
 [ 0.          1.          0.          0.035     ]
 [-0.47941191  0.          0.87759001  0.563     ]
 [ 0.          0.          0.          1.        ]]
```

```
homogeneous transformation of {E} with respect to {B}
-----------------------------------------------------
[[ 0.      0.      1.      0.679]
 [ 0.      1.      0.     -0.019]
 [-1.      0.      0.      0.445]
 [ 0.      0.      0.      1.   ]]


homogeneous transformation of {B} with respect to {E}
-----------------------------------------------------
[[ 0.      0.     -1.      0.445]
 [ 0.      1.      0.      0.019]
 [ 1.      0.      0.     -0.679]
 [ 0.      0.      0.      1.   ]]


``````````````````````````````````````````````````````````
``````````````````````````````````````````````````````````
Finally,
homogeneous transformation of {P} with respect to {E}
-----------------------------------------------------
[[ 0.47941191  0.          -0.87759001 -0.118     ]
 [ 0.          1.           0.          0.054     ]
 [ 0.87759001  0.           0.47941191 -0.855     ]
 [ 0.          0.           0.          1.        ]]
```

## Gimbal lock

One of the problems with using Fixed or Euler angles for rotations is the Gimbal lock. Read about it here, or watch a video about it here.

In the code below, first complete the `rotate` function to rotate a point using fixed-angles. Next, rotate the point `p`, by two different sets of angles to illustrate the Gimbal lock. Explain what has happened, and why.

In [53]:

```
def rotate(p, alpha, beta, gamma):
    """
    This function rotates the point, p, by
    gamma (about X), beta (about Y) and alpha (about Z) using fixed angles,
    and returns the rotated point

    """
    R_Z=get_rotation_matrix_z_axis(alpha)
    R_Y=get_rotation_matrix_y_axis(beta)
    R_X=get_rotation_matrix_x_axis(gamma)

    R=np.dot(R_Z,R_Y,R_X)

    p_roated=np.dot(R,p)
    print"----------------------"
    print "Rotaion Matrix::::::"
    print R
    print"----------------------"
```

```
    print"Coordinates of P after Rotation "
    return p_roated


p = np.array([1., 1., 1.])
print""
print"Initial coordiantes of P::"
print p
print ""
print "**************************************************"
print ""
#### Specify angles alpha1,... etc. to illustrate the Gimbal lock

###***************************
### We fix the angle beta1 as -90 degrees
###***************************

#First Set of Angles
alpha1=np.pi / 4
beta1=-np.pi / 2
gamma1=np.pi / 6

#Second Set of Angles
alpha2=np.pi / 6
gamma2=np.pi / 3

print ""
print "Rotating p by Angles (alpha,beta,gamma)::",alpha1,",", beta1,",", ga
mma1
print rotate(p, alpha1, beta1, gamma1)

print ""
print "**************************************************"
print ""

print "Rotating p by Angles (alpha,beta,gamma)::",alpha2,",", beta1,",", ga
mma2
print""
print""
print rotate(p, alpha2, beta1, gamma2)
print""
print""
```

```
Initial coordiantes of P::
[ 1.  1.  1.]

**************************************************



Rotating p by Angles (alpha,beta,gamma):: 0.785398163397 , -1.57079632679 ,
0.523598775598
----------------------
Rotaion Matrix::::::
[[ 0.          -0.70710678 -0.70710678]
 [ 0.           0.70710678 -0.70710678]
 [ 1.           0.           0.        ]]
----------------------
Coordinates of P after Rotation
[-1.41421356  0.           1.        ]
```

```
*********************************************

Rotating p by Angles (alpha,beta,gamma):: 0.523598775598 , -1.57079632679 ,
1.0471975512


----------------------
Rotaion Matrix::::::
[[ 0.         -0.5        -0.8660254]
 [ 0.          0.8660254 -0.5       ]
 [ 1.          0.          0.       ]]
----------------------
Coordinates of P after Rotation
[-1.3660254  0.3660254  1.       ]
```

**Explanation of Gimbal Lock**

As we can see from the rotation matrix and the coordinates of the roated point, the rotation for both cases is about the same axis. Even though we had fixed only one angle, i.e., *beta*, in effect, we lost two degrees of freedom instead of one.
This is an illustration of the **Gimbal Lock problem**

# Quaternions

By representing rotation angles as [quaternions](), we are able to avoid the Gimbal lock problem. Show this in the code below by using the same angles as in the fixed-angles exercise above.

In [54]:

```
# for euler2quat
from eulerangles import *
# for rotate_vector
from quaternions import *

p = np.array([1., 1., 1.])

### Specify angles alpha1,... etc. (same as above) to show that using quate
rnions
### eliminates the Gimbal lock problem
alpha1=np.pi / 4
beta1=-np.pi / 2
gamma1=np.pi / 6
about
alpha2=np.pi / 6
gamma2=np.pi / 3


q1 = euler2quat(alpha1, beta1, gamma1)
p_a = rotate_vector(p, q1)
print p_a

q2 = euler2quat(alpha2, beta1, gamma2)
```

```
p_b = rotate_vector(p, q2)
print p_b
```

```
[-1.          1.22474487  0.70710678]
[-1.         0.3660254   1.3660254]
```

## Review

The following task is meant to help you review the knowledge you have acquired about frames of reference.

### Properties of a rotation matrix

Create a function, or functions, to verify if a given matrix is a rotation matrix, i.e. show that a given matrix has the properties of a rotation matrix.

Complete the following function to determine if a given matrix is a rotation matrix.

In [57]:

```
def is_rotation_matrix(matrix):
    """
    This function returns True only if the input matrix is a rotation matri
x
    (based on the properties of a rotation matrix), otherwise it returns Fa
lse.

    """
    # for each property you test, write a brief explanation of that propert
y.
    #Rotational matrices are orthogonal, so its inverse is equal to its tra
nspose.
    #As such, (matrix).(transpose of matrix)=Identity matrix
    #Or det(matrix)=1
    flag= False

    shape=np.array(matrix.shape)            #To get dimensions of the matrix.
    if (shape[0]==shape[1]):                #so proceed only for square matr.
ces as orthogonal matrices are square
        n=shape[1]                          #To get the order of the n*n
matrix
        I=np.identity(n, dtype=float)       #To get Identity matrix of order
n
        matrix_T=np.array(matrix.T)
        matrix_check=np.dot(matrix_T,matrix)

        if (np.array_equal(matrix_check, I)): #i.e., matrix).(transpose of
matrix)=Identity matrix
            flag=True

        if ((linalg.det(matrix)==1) and flag==True):
            return True                     #Since det(matrix)=1 and
matrix).(transpose of matrix)=Identity matrix
        else:
            return  False                   #Since above conditions fail, s
```

```
it is not a rotation matrix
    else:
        return False                        #since matrix is not square, so
it cannot be a rotation matrix

#*****PROOF:::::::##
alpha=np.pi / 4
R_roation=get_rotation_matrix_z_axis(alpha)
print ""
print "-----------------------------------------------"
print R_roation
print "Checking if above matrix is a rotation matrix............"
print "ANS: ",is_rotation_matrix(R_roation) #This should be true as we have
used a rotaion matrix of angle alpha

print ""
print "-----------------------------------------------"
print E_P_T #E_P_T is a transform matrix used in earlier subsection
print "Checking if above matrix is a rotation matrix............"
print "ANS: ",is_rotation_matrix(E_P_T) #This should be false
```

```
-----------------------------------------------
[[ 0.70710678 -0.70710678  0.        ]
 [ 0.70710678  0.70710678  0.        ]
 [ 0.          0.          1.        ]]
Checking if above matrix is a rotation matrix............
ANS:  True

-----------------------------------------------
[[ 0.47941191  0.         -0.87759001 -0.118     ]
 [ 0.          1.          0.          0.054     ]
 [ 0.87759001  0.          0.47941191 -0.855     ]
 [ 0.          0.          0.          1.        ]]
Checking if above matrix is a rotation matrix............
ANS:  False
```

# Read the course rules on LEA for submission details