CENG 483 – BEHAVIORAL ROBOTICS

Homework Set #2   Due : 31.10.2023

Cem Tolga MÜNYAS

1. I used MATLAB Robotics Toolbox and wrote functions for theta in degrees for 25, 45, 65, 82.

a) ROTX(theta). This function will give a 4x4 homogenous transformation matrix that will rotate "theta" about the X-axis.

```
function output = ROTX(theta)
    output = trotx(theta, "deg");
end
```

b) ROTY(theta). This function will give a 4x4 homogenous transformation matrix that will rotate "theta" about the Y-axis.

```
function output = ROTY(theta)
    output = trotx(theta, "deg");
end
```

c) ROTY(theta). This function will give a 4x4 homogenous transformation matrix that will rotate "theta" about the Z-axis.

```
function output = ROTZ(theta)
    output = trotx(theta, "deg");
end
```

d) TRANS(x,y,z). This function will give a 4x4 homogenous transformation matrix that will translate "x" units over the X-axis, "y" units over the Y-axis, and "z" units over the Z-axis.

```
function output = TRANS(x, y, z)
    output = transl(x, y, z);
end
```

The code for printing the results is :

```
thetas = [25 45 65 82];

for i=1 : length(thetas)
    rot_x = ROTX(thetas(i));
    rot_y = ROTY(thetas(i));
    rot_z = ROTZ(thetas(i));

    display(rot_x);
    display(rot_y);
    display(rot_z);
end
```

Here is the output matrix for 25 degrees :

```
rot_x =

    1.0000         0         0         0
         0    0.9063   -0.4226         0
         0    0.4226    0.9063         0
         0         0         0    1.0000


rot_y =

    1.0000         0         0         0
         0    0.9063   -0.4226         0
         0    0.4226    0.9063         0
         0         0         0    1.0000


rot_z =

    1.0000         0         0         0
         0    0.9063   -0.4226         0
         0    0.4226    0.9063         0
         0         0         0    1.0000
```

Here is the output matrix for 45 degrees :

```
rot_x =

    1.0000         0         0         0
         0    0.7071   -0.7071         0
         0    0.7071    0.7071         0
         0         0         0    1.0000


rot_y =

    1.0000         0         0         0
         0    0.7071   -0.7071         0
         0    0.7071    0.7071         0
         0         0         0    1.0000


rot_z =

    1.0000         0         0         0
         0    0.7071   -0.7071         0
         0    0.7071    0.7071         0
         0         0         0    1.0000
```

Here is the output matrix for 65 degrees :

```
rot_x =

    1.0000         0         0         0
         0    0.4226   -0.9063         0
         0    0.9063    0.4226         0
         0         0         0    1.0000


rot_y =

    1.0000         0         0         0
         0    0.4226   -0.9063         0
         0    0.9063    0.4226         0
         0         0         0    1.0000


rot_z =

    1.0000         0         0         0
         0    0.4226   -0.9063         0
         0    0.9063    0.4226         0
         0         0         0    1.0000
```

Here is the output matrix for final degrees (82 degrees) :

```
rot_x =

    1.0000         0         0         0
         0    0.1392   -0.9903         0
         0    0.9903    0.1392         0
         0         0         0    1.0000


rot_y =

    1.0000         0         0         0
         0    0.1392   -0.9903         0
         0    0.9903    0.1392         0
         0         0         0    1.0000


rot_z =

    1.0000         0         0         0
         0    0.1392   -0.9903         0
         0    0.9903    0.1392         0
         0         0         0    1.0000
```

2. I used the resulting functions of previous problems to do the following operations :

I) Find a final transformation that translates a frame {N} 2 units in X, 3 unites in Y and -2 units in Z, i.e. (2,3,-2), and then rotate the translated frame 60 degrees over its translated Y axis.

II) Find a final transformation rotates a frame {O} 60 degrees over its translated Y axis and then translates the rotated frame (2,-2, 2).

```
thetas = [25 45 65 82];

N1 = transl(2, 3, -2) * troty(60, "deg");
N2 = troty(60, "deg") * transl(2, -2, 2);
display(N1);
display(N2);
```

Output :

```
N1 =

    0.5000         0    0.8660    2.0000
         0    1.0000         0    3.0000
   -0.8660         0    0.5000   -2.0000
         0         0         0    1.0000


N2 =

    0.5000         0    0.8660    2.7321
         0    1.0000         0   -2.0000
   -0.8660         0    0.5000   -0.7321
         0         0         0    1.0000
```

3. Create a 2D rotation matrix. Visualize the rotation using trplot2. Use it to transform a vector. Invert it and multiply it by the original matrix; what is the result?

Reverse the order of multiplication; what is the result? What is the determinant of the matrix and its inverse?

Here is the code for creating a 2D rotation matrix with the theta angle of 45 degrees.

```
% Create a 2D rotation matrix;
theta = 45;
rad = deg2rad(theta);

rotation1 = [cos(rad) -sin(rad);
             sin(rad) cos(rad)];

display(rotation1);

% Visualization
figure(1);
trplot2(rotation1, "frame", "2", "color", "b");
title("2D Rotation Matrix"); xlabel("X"); ylabel("Y");
```

Here is the visualization of the rotation using trplot2.

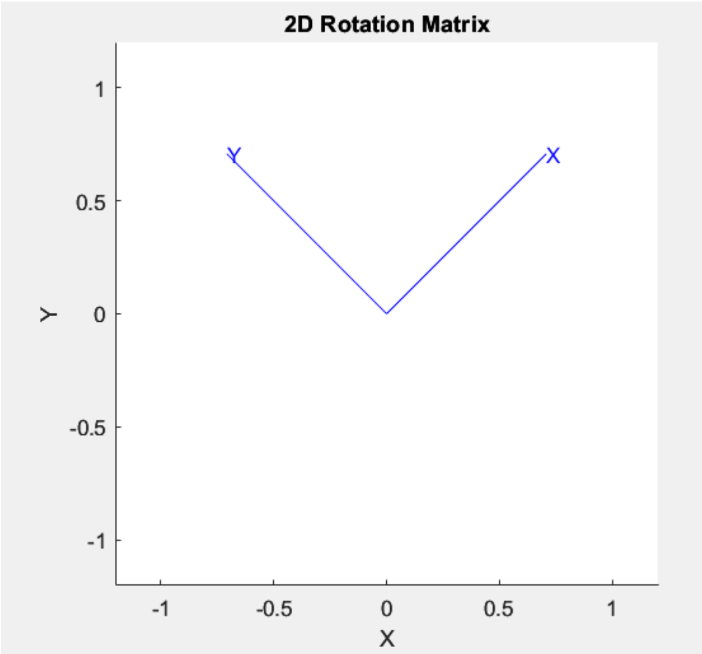

Figure 1.1 Visualization of the rotation using trplot2

Here is the code for finding the inverse of the rotation matrix, multiplying rotation matrix with the inverse matrix by changing the order of them.

```
% Finding inverse of the rotation matrix
inverse = inv(rotation1);
% Multiplication of inverse matrix and rotation matrix
mult = inverse * rotation1;
% Multiplication of rotation matrix and inverse matrix
mult2 = rotation1 * inverse;

fprintf("Inverse of the rotation matrix :");
disp(inverse);
fprintf("Multiplication of inverse matrix and rotation matrix");
disp(mult);
fprintf("Multiplication of rotation matrix and inverse matrix")
disp(mult2);

% Estimating determinant of rotatio matrix
det_rotation = det(rotation1);
% Estimating determinant of inverse matrix
det_inverse = det(inverse);

fprintf("Determinant of rotation matrix"); disp(det_rotation);
fprintf("Determinant of inverse of rotation matrix"); disp(det_inverse);
```

The output is :

```
rotation1 =

    0.7071   -0.7071
    0.7071    0.7071

Inverse of the rotation matrix :
    0.7071    0.7071
   -0.7071    0.7071

Multiplication of inverse matrix and rotation matrix :
     1     0
     0     1

Multiplication of rotation matrix and inverse matrix :
     1     0
     0     1

Determinant of rotation matrix :
     1

Determinant of inverse of rotation matrix :
     1
```

4. Create a 3D rotation matrix. Visualize the rotation *using trplot* or *tranimate*. Use it to transform a vector. Invert it and multiply it by the original matrix; what is the result? Reverse the order of multiplication; what is the result? What is the determinant of the matrix and its inverse?

Here is the code for creating a 3D rotation matrix around x and the code for visualization.

```
% Create a 3D rotation matrix
theta_3d = 60; % rotation angle in degrees
rad_theta = deg2rad(theta_3d);

% 3D Rotation matrix around x
rotation2 = [1 0 0;
             0 cos(theta_3d) -sin(theta_3d);
             0 sin(theta_3d) cos(theta_3d)];

% Visualization
figure(2);
trplot(rotation2, "frame", "2", "color", "b");
title("3D Rotation Matrix"); xlabel("X"); ylabel("Y"); zlabel("Z");
display(rotation2);
```

Here is the visualization of the 3D rotation matrix around x.

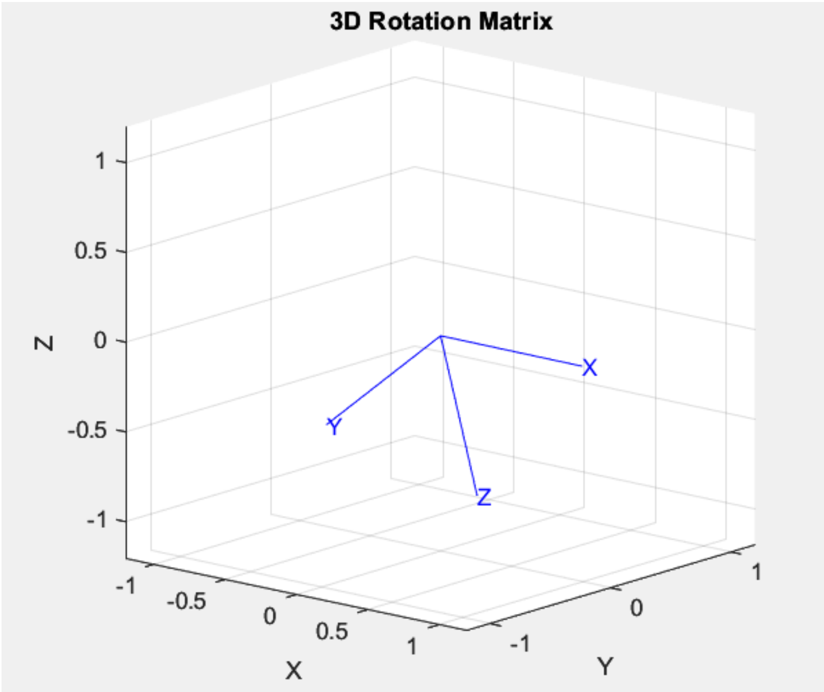

Figure 1.2 Visualization of 3D rotation matrix around x using trplot

Here is the code for finding the inverse of the rotation matrix, multiplying rotation matrix with the inverse matrix by changing the order of them.

```
% Finding inverse of the rotation matrix
inverse = inv(rotation2);
% Multiplication of inverse matrix and rotation matrix
mult = inverse * rotation2;
% Multiplication of inverse matrix and rotation matrix
mult2 = rotation2 * inverse;

fprintf("Inverse of the rotation matrix :\n");
disp(inverse);
fprintf("Multiplication of inverse matrix and rotation matrix :\n");
disp(mult);
fprintf("Multiplication of rotation matrix and inverse matrix :\n");
disp(mult2);

% Estimating determinant of rotation matrix
det_rotation = det(rotation2);
% Estimating determinant of inverse matrix
det_inverse = det(inverse);

fprintf("Determinant of rotation matrix : \n"); disp(det_rotation);
fprintf("Determinant of inverse of rotation matrix : \n"); disp(det_inverse);
```

The output is :

```
rotation2 =

    1.0000         0         0
         0   -0.9524    0.3048
         0   -0.3048   -0.9524

Inverse of the rotation matrix :
    1.0000         0         0
         0   -0.9524   -0.3048
         0    0.3048   -0.9524

Multiplication of inverse matrix and rotation matrix :
    1.0000         0         0
         0    1.0000    0.0000
         0         0    1.0000

Multiplication of rotation matrix and inverse matrix :
    1.0000         0         0
         0    1.0000    0.0000
         0         0    1.0000

Determinant of rotation matrix :
     1

Determinant of inverse of rotation matrix :
     1
```

5. Generate the sequence of plots shown in Fig. 2.12.

```
% Generating the sequence of plots in Fig 2.12
seq1ini = rotx(0);
seq1x = rotx(pi/2);
seq1y = roty(pi/2);
result1 = seq1x * seq1y;
result2 = seq1y * seq1x;

figure(3);
subplot(2,3,1);
trplot(seq1ini); view(10,10);
subplot(2,3,2);
trplot(seq1x); view(10,10);
subplot(2,3,3);
trplot(result1); view(10,10);
subplot(2,3,4);
trplot(seq1ini); view(10,10);
subplot(2,3,5);
trplot(seq1y); view(10,10);
subplot(2,3,6);
trplot(result2); view(10,10);
```

The corresponding figures showing the noncommutativity of rotation. In the top row the coordinate frame is rotated by $\pi/2$ about the x-axis and then $\pi/2$ about the y-axis. In the bottom row the order of rotations has been reversed.

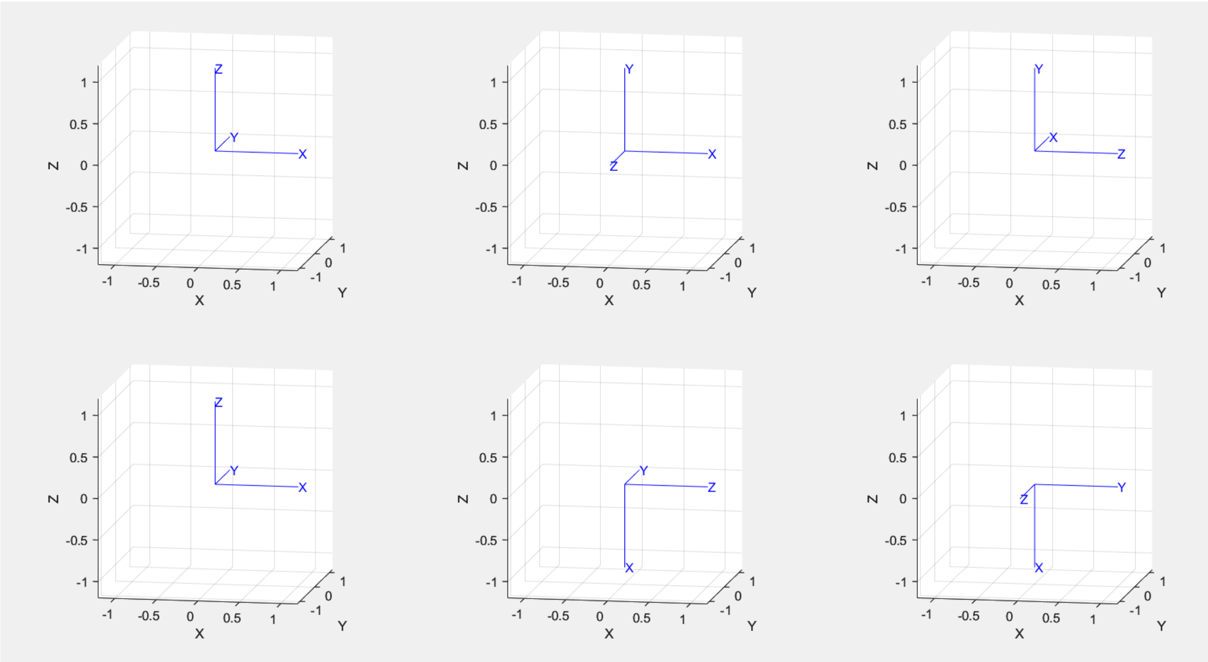

Figure 1.3 Generating the sequence of rotations in textbook

6. For the 3-dimensional rotation about the vector [2, 3, 4] by 0.5 rad compute an SO(3) rotation matrix using: the matrix exponential functions expm() and trexp(), Rodrigues' rotation formula (code this yourself), and the Toolbox function angvec2tr(). Compute the equivalent unit quaternion.

I have applied the Rodriguez formula as a function in code. It is shown in below.

```
% Rodriguez function
function output_matrix = Rodriguez(theta, V)
    I = [1 0 0;
         0 1 0;
         0 0 1;];

    skew_matrix = skew(V);
    output_matrix = I + sin(theta) * skew_matrix
                    + (1 - cos(theta)) * skew_matrix^2;
end
```

Here is the rest of the code by applying exponential functions *expm(), trexp(), angvec2tr(),* and Rodriguez function in order to estimate the rotations. Additionally, I found the equivalent unit quaternion properties.

```
% A 3D rotation matrix with respect to y
rad = 0.5;
theta = rad2deg(rad);
R = [cos(theta) 0 sin(theta);
     0 1 0;
    -sin(theta) 0 cos(theta)];

% Creating the vector and normalization
V = [2; 3; 4];
V_normalized = V / norm(V);

% Rotation with expm()
rot_expm = expm(skew(V_normalized) * rad);
fprintf("Rotation matrix with expm() :\n");
disp(rot_expm);

% Rotation with trexp()
rot_trexp = trexp(skew(V_normalized) * rad);
fprintf("Rotation matrix with trexp() :\n");
disp(rot_trexp);

% Rotation with Rodrigues function
rot_rodriguez = Rodriguez(rad, V_normalized);
fprintf("Rotation matrix with Rodrigues's formula :\n");
disp(rot_rodriguez);

% Rotation with angvec2tr()
rot_tool = angvec2tr(rad, V_normalized);
fprintf("Rotation matrix with angvec2tr() :\n");
disp(rot_tool);

% Equivalent unit quaternion
unit_qua = UnitQuaternion(rot_expm);
fprintf("Equivalent unit quaternion :\n");
disp(unit_qua);
```

The corresponding output is shown below.

```
Rotation matrix with expm() :
    0.8945   -0.3308    0.3009
    0.3814    0.9156   -0.1274
   -0.2333    0.2287    0.9451

Rotation matrix with trexp() :
    0.8945   -0.3308    0.3009
    0.3814    0.9156   -0.1274
   -0.2333    0.2287    0.9451


output_matrix =

    1.0000   -0.3561    0.2671
    0.3561    1.0000   -0.1781
   -0.2671    0.1781    1.0000

Rotation matrix with Rodrigues's formula :
    1.0000   -0.3561    0.2671
    0.3561    1.0000   -0.1781
   -0.2671    0.1781    1.0000

Rotation matrix with angvec2tr() :
    0.8945   -0.3308    0.3009         0
    0.3814    0.9156   -0.1274         0
   -0.2333    0.2287    0.9451         0
         0         0         0    1.0000

Equivalent unit quaternion :
  UnitQuaternion with properties:

    s: 0.9689
    v: [0.0919 0.1378 0.1838]
```