EENG 428 Introduction to Robotics Laboratory

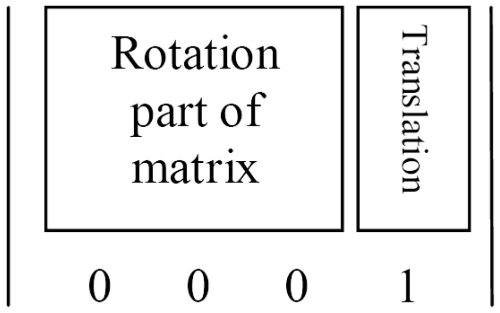
EXPERIMENT 5

Robotic Transformations

Objectives
This experiment aims on introducing the homogenous transformation matrix that represents rotation and translation in the space. Next, the ability of converting the transformation matrix into Euler angles format.
Besides, students will learn how to build a robotic system under MATLAB environment, including the robot definition and any other required specifications. Moreover, students will be able to visualize the robot configuration and to perform visual animations to explore its functionalities.

1. Transformation Matrix

In the field of robotics there are many possible ways of representing positions and orientations, but the homogeneous transformation is well matched to MATLAB's powerful tools for matrix manipulation. Homogeneous transformations describe the relationships between Cartesian coordinate frames in terms of translation and orientation. The transformation matrix has the following structure:



In the MATLAB Robotics toolbox, the homogenous transformation matrix of translation of a (x, y, z) distance and rotation of an angle θ around the x, y and z axes is evaluated by the commands$transl(x, y, z)$, $rotx(\theta)$, $roty(\theta)$ and $rotz(\theta)$, respectively.

* A pure translation of 0.5m in the X direction is represented by

```
>>    transl(0.5, 0.0, 0.0)
ans =
   1.0000         0         0    0.5000
        0    1.0000         0         0
        0         0    1.0000         0
```

```
         0         0         0    1.0000
```

a rotation of 90degrees about the Y axis by

```
>>     roty(pi/2)
ans =
    0.0000         0    1.0000         0
         0    1.0000         0         0
   -1.0000         0    0.0000         0
         0         0         0    1.0000
```

and a rotation of -90degrees about the Z axis by

```
>> rotz(-pi/2)
ans =
    0.0000    1.0000         0         0
   -1.0000    0.0000         0         0
         0         0    1.0000         0
         0         0         0    1.0000
```

these may be concatenated by multiplication

```
>> t = transl(0.5, 0.0, 0.0) * roty(pi/2) * rotz(-pi/2)
t =
    0.0000    0.0000    1.0000    0.5000
   -1.0000    0.0000         0         0
   -0.0000   -1.0000    0.0000         0
         0         0         0    1.0000
```

It is important to note that transform multiplication is in general not commutative as shown by the following example

```
>> rotx(pi/2) * rotz(-pi/8)
ans =
    0.9239    0.3827         0         0
   -0.0000    0.0000   -1.0000         0
   -0.3827    0.9239    0.0000         0
         0         0         0    1.0000
>> rotz(-pi/8) * rotx(pi/2)
ans =
    0.9239    0.0000   -0.3827         0
   -0.3827    0.0000   -0.9239         0
         0    1.0000    0.0000         0
```

```
        0         0         0    1.0000
```

2. Transformation matrix conversion to Euler Angles:

Command :***tr2eul***
Purpose: Convert a homogeneous transform to Euler angles
Synopsis: [a b c] = tr2eul(T)
Description: tr2eul returns a vector of Euler angles, in radians, corresponding to the rotational part of the homogeneous transform T.

* the orientation of the new coordinate frame of the last example may be expressed in terms of Euler angles

```
>>     tr2eul(t)

ans =

         0    1.5708   -1.5708
```

3. Robot Object Definition

Command: **robot**

Synopsis r = robot

r = robot(rr)

r = robot(link ...)

r = robot(DH ...)

r = robot(DYN ...)

Description:

robot is the constructor for a robot object. The first form creates a default robot, and the second form returns a new robot object with the same value as its argument. The third form creates a robot from a cell array of link objects which define the robot's kinematics and optionally dynamics. The fourth and fifth forms create a robot object from legacy DH and DYN format matrices. The last three forms all accept optional trailing string arguments which are taken in order as being robot name, manufacturer and comment.

5. Defining a Link

Command: **link**
Purpose: Link object

Synopsis:
L = link
L = link([alpha, a, theta, d])
L = link([alpha, a, theta, d, sigma])
L = link(dyn row)
A = link(q)
Description:
The link function constructs a link object. The object contains kinematic and dynamic parameters as well as actuator and transmission parameters. The first form returns a default object, while the second and third forms initialize the kinematic model based on Denavit and Hartenberg parameters. By default the standard Denavit and Hartenberg conventions are assumed but a flag (mdh) can be set if modified Denavit and Hartenberg conventions are required. The dynamic model can be initialized using the fourth form of the constructor where dyn row is a 1 20 matrix which is one row of the legacy dyn matrix.

4. Plotting a Robot

Command: plot
Purpose: Graphical robot animation
Synopsis: plot(robot, q)

Example: P arm using modified DH parameters

```
L1=link([0 0 0 0 1], 'modified');
L2=link([0 1 0 0 0], 'modified'); %Lee: no joint var for mod DH params
r=robot({L1 L2});
r.name='Our First Robot';
plot(r, [1 0])
```

The output is depicted in Figure 1.

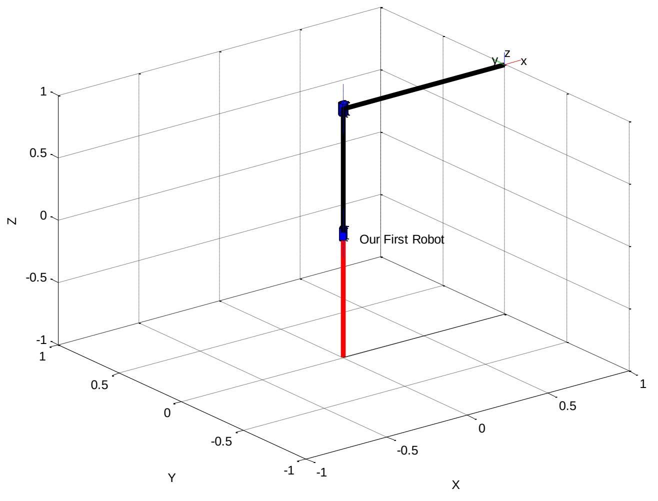


Figure 1 P arm using modified DH parameters

6. Forward robot kinematics for serial link manipulator

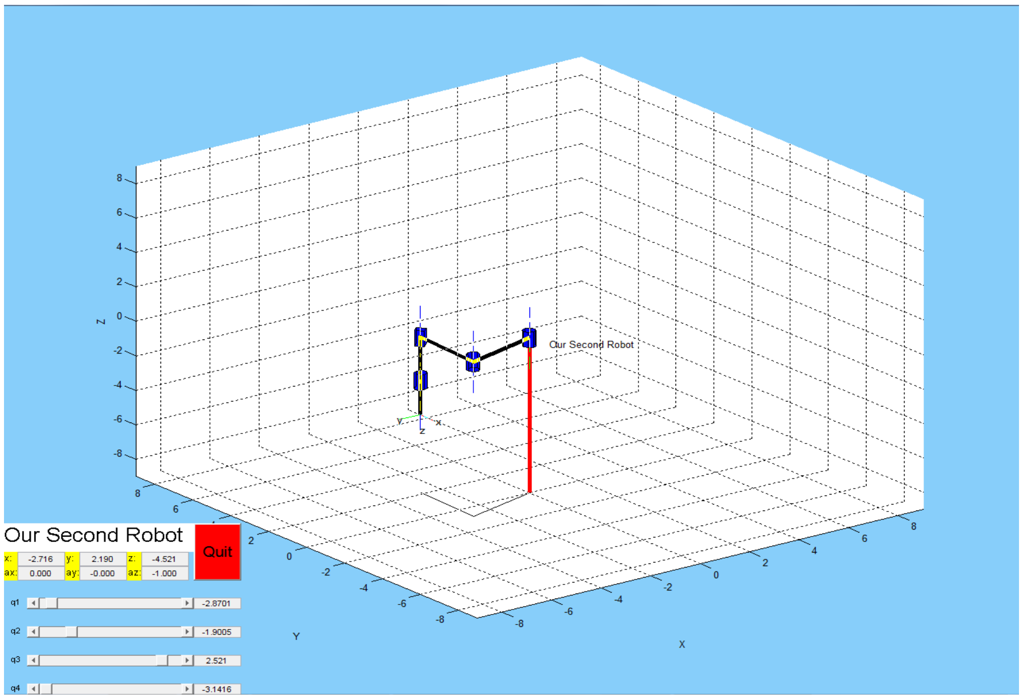
Command: **fkine**
Purpose: Forward robot kinematics for serial link manipulator
Synopsis T = fkine(robot, q)
Description:
fkine computes forward kinematics for the joint coordinate q giving a homogeneous transform for the location of the end-effector. robot is a robot object which contains a kinematic model in either standard or modified Denavit-Hartenberg notation. Note that the robot object can specify an arbitrary homogeneous transform for the base of the robot.

If q is a vector it is interpreted as the generalized joint coordinates, and fkine returns a homogeneous transformation for the final link of the manipulator. If q is a matrix each row is interpreted as a joint state vector, and T is a 4 4 *m* matrix where m is the number of rows in q.

7. Driving a Graphical Robot

Command: drivebot
Purpose: Drive a graphical robot
Synopsis: drivebot(robot)
Description
Pops up a window with one slider for each joint. Operation of the sliders will drive the graphical robot on the screen. Very useful for gaining an understanding of joint limits and robot workspace. The joint coordinate state is kept with the graphical robot and can be obtained using the plot function. The initial value of joint coordinates is taken from the graphical robot.

Example: As a comprehensive example, define and animate the P-arm system depicted in Figure 2.



Our Second Robot

x:	-2.716	y:	2.190	z:	-4.521
ax:	0.000	ay:	-0.000	az:	-1.000

```
q1  -2.8701
q2  -1.9005
q3  2.521
q4  -3.1416
```

Figure 2 A P-Arm Robot

The code is:

```
theta1=30*pi/180;
theta2=60*pi/180;
theta3=90*pi/180;
theta4=90*pi/180;
d3=1;

 % defining the links in terms of  the D-H parameters:
L1=link([0 3 theta1 0 0] );
L2=link([pi 3 theta2 0 0]);
L3=link([0 0 0 d3 1]);
L4=link([0 0 theta4 2 0]);

% defining the robotic objects:
r1=robot({L1});
r2=robot({L2});
r3=robot({L3});
r4=robot({L4});

%creating the complete robot
r=robot({L1 L2 L3 L4});
r.name='Our Second Robot;
t=fkine(r,[0 0 1 0])

% animating the robot
drivebot(r,[0 0 0 0])
```