Lab 6 Exercises

1 Convert range measurements to a point cloud

1.1 Consider a laser tape measure (LTM)[1] that projects a laser and measures a single distance to an object in the environment Pretend it is affixed to the end effector of the Schunk robot.

```
robot = SchunkUTSv2_0();
q = [0,pi/2,0,0,0,0];
robot.plot3d(q);
view(3);
camlight;
hold on;
```

1.2 Assume the LTM (max range 3m) has an identical start location as the end-effector location, and the ray is parallel to the Z axis. Plot a red line 1.9594m long from end effector parallel with the Z axis

```
tr = robot.fkine(q)
startP = tr(1:3,4);
endP = tr(1:3,4) + 1.9594 * tr(1:3,3);
line1_h = plot3([startP(1),endP(1)],[startP(2),endP(2)],[startP(3),endP(3)],'r');
plot3(endP(1),endP(2),endP(3),'r*');
```

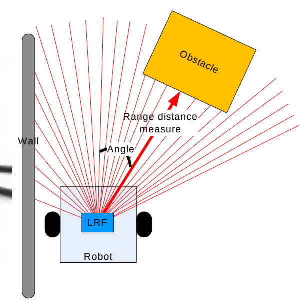
1.3 If the robot is in a pose [0,pi/2,0,0,0,0] and the laser tape measure returns a measurement (as plotted above), the robot is robot is moved twice more and a measurement is taken. Plot this information

Robot Joint State	LTM Measurement
[pi/10, pi/2, 0, 0, 0, 0]	2.4861m
[-pi/10, 5*pi/12, 0, 0, 0, 0]	1.9132m

1.4 Given now these three points, make a mesh of the possible location of a wall (only use value of the wall which are above 0m). Remember, where the triangle vertices are [v1,v2,v3]

```
triangleNormal = unit(cross((v1-v2),(v2-v3)))
```

1.5 Consider mounting a motor to rotate (roll) the LTM 40' around the X axis of the end effector, and look at the wall. Put the robot in a pose [0,pi/2,0,0,0,0] and rotate the LTM by increments of 1' from -20' to 20' (total of 41 readings). Essentially this is a Laser Range Finder (LRF)[2].



1.6 Consider mounting a second motor to rotate the LTM around both the X and the Y axis from -20' to 20' in each (i.e. 41 readings in each row and 41*41 = 1681 readings all together). Note the Field of View is 40' by 40'. Essentially this is a tilting LRF or 3D LiDAR[3] used in many driverless cars. Use it to look at the wall you found.

2 More complex collision detection for 3-link planar robot

The textbook (Appendix E) talked about ellipses. Now, we will create a 3D ellipse called an ellipsoid. Note that the equations of this ellipsoid is

$$\left(\frac{x-x_c}{r_x}\right)^2+\left(\frac{y-y_c}{r_y}\right)^2+\left(\frac{z-z_c}{r_z}\right)^2=1$$

2.1 Create vertices that represent an ellipsoid with radii (rx=3,ry=2,rz=1) centered at [xc,yc,zc] = [0,0,0].

```
centerPoint = [0,0,0];
radii = [3,2,1]
[X,Y,Z] = ellipsoid(centerPoint (1),centerPoint (2),centerPoint
(3),radii(1),radii(2),radii(3));
```

[1] Stanley TLM 100 FatMax Tru-Laser Distance Measurer 0020http://kk.org/cooltools/trulaser-distan/
[2] Hokuyo UTM-30LX https://www.hokuyo-aut.jp/02sensor/07scanner/utm_30lx.html
[3] Velodyne LiDAR http://velodynelidar.com/news.php

2.2 Now plot it

```
surf(X,Y,Z)
```

2.3 Put a cube with sides 1.5m in the environment that is centered at [2,0,-0.5]. Use mesh so as to create a high density mesh that has many vertices (either create in blender and load, or use 6 planes from `meshgrid`). Note: create a single plane of a cube centered at the origin like follows:

```
[Y,Z] = meshgrid(-0.75:0.05:0.75,-0.75:0.05:0.75);
X = repmat(0.75,size(Y,1),size(Y,2));
```

2.4 Check how many point and which points are inside the ellipsoid, using the equation. Note that points that are inside have an algebraic distance (AD) < 1, on the surface AD = 1 and outside AD > 1

2.5 Transform the ellipsoid by translating it [1,1,1], do this by changing the values of [xc,yc,zx], then check which points are inside the ellipsoid

2.6 This time, using the original centered-at-the-origin ellipse, notice how you can transform the points in the environment by `inv(transl(1,1,1))` and then check the original equation to see which have an algebraic distance less than 0. The points inside should be the same as when using the previous method.

2.7 Now, if the ellipsoid where transformed by `transl(1,1,1)*trotx(pi/4)`, which points are inside (note that you will need to transform the points in the environment instead of the ellipsoid formula

2.8 Now create a 3 link planar and use the 3 ellipsoids as the model points and faces. Now use teach to move it around so you should see the ellipsoids move around as well

2.9 **(Bonus)** For a given pose, work out the location of the ellipsoid of the end effector, using `fkine`, and the multiply the points in the environment by the inverse of this transform, and check the algebraic distance

2.10 **(Bonus)** Do this for each of the ellipsoids on the three links. Note: you will need to have your own forward kinematics routine so you can compute the location of each of the ellipsoids

3 Joint Interpolation vs Resolve Motion Rate Control

3.1 Moving from A to B with Joint Interpolation: Load a 2-Link Planar Robot with `mdl_planar2;`

3.2 Create two Transformation Matrices:

$$\boldsymbol{T}_1 = \begin{bmatrix} \boldsymbol{I}_3 & \begin{matrix} 1.5 \\ 1 \\ 0 \end{matrix} \\ \boldsymbol{0}_{1\times3} & \boldsymbol{1} \end{bmatrix} \qquad \boldsymbol{T}_2 = \begin{bmatrix} \boldsymbol{I}_3 & \begin{matrix} 1.5 \\ -1 \\ 0 \end{matrix} \\ \boldsymbol{0}_{1\times3} & \boldsymbol{1} \end{bmatrix}$$

3.3 Use Inverse Kinematics to solve the joint angles required to achieve each pose.

```
M = [1 1 zeros(1,4)];          % Masking Matrix
q1 = p2.ikine(T1,[0 0],M);     % Solve for joint angles
q2 = p2.ikine(T2,[0 0],M);     % Solve for joint angles
```

3.4 Use joint interpolation to move between the two poses. Be sure to plot the end-effector path.

```
p2.plot(qMatrix,'trail','r-');
```

3.5 Moving from A to B with Resolved Motion Rate Control

3.6 Create two sets of points in the X-Y plane

```
x1 = [1.5 1]';
x2 = [1.5 -1]';
deltaT = ...                            % Discrete time step
```

3.7 Create a matrix of waypoints

```
x = zeros(2,steps);                     % Assign memory
```

```
s = lspb(0,1,steps);                        % Create interpolation scalar
for i = 1:steps
      x(:,i) = x1*(1-s(i)) + s(i)*x2;      % Interpolate waypoints
end
```

3.8 Create a matrix of joint angles

```
qMatrix = nan(steps,2);
```

3.9 Set the Transformation for the 1st point, and solve for the joint angles $\boldsymbol{T}_1 = \begin{bmatrix} \boldsymbol{I}_3 & \begin{matrix} 1.5 \\ -1 \\ 0 \end{matrix} \\ \boldsymbol{0}_{1\times3} & \boldsymbol{1} \end{bmatrix}$

```
qMatrix(1,:) = p2.ikine(T1,[0 0],M);
```

3.10 Use Resolved Motion Rate Control to move the end-effector from x_1 to x_2.

```
for i = 1:steps-1
      xdot = ...                            % Velocity to reach next waypoint
      J = p2.jacob0(qMatrix(i,:));          % Get Jacobian at current state
      J = J(1:2,:);                         % Take only first 2 rows
      qdot = ...                            % Solve the RMRC equation
      qMatrix(i+1,:)= ...                   % Update the joint state
end
```