Lab 7 Exercises

Sunday, 2 October 2022 9:36 PM

1 Redundant Robot

θ_j	d_j	a_j	α_j
q_1	0	l_1	0
q_2	0	l_2	0
q_3	0	l_3	0

1.1 Derive the Jacobian, by hand for a 3-Link Planar Robot using position variables, and the rotation about the z-axis variable, as the joint angle, changes.

1.2 Now input the equation for into Matlab using symbolic values, e.g. syms l1 l2 l3 x y phi q1 q2 q3 Jq;

1.3 Using the Matlab subs function: subs(subs(subs(Jq,l1,1),l2,1),l3,1))where show the Jacobian, Jq is now

```
[ - sin(q1 + q2 + q3) - sin(q1 + q2) - sin(q1), - sin(q1 + q2 + q3) - sin(q1 + q2), -sin(q1 + q2 + q3)]
[   cos(q1 + q2 + q3) + cos(q1 + q2) + cos(q1),   cos(q1 + q2 + q3) + cos(q1 + q2),  cos(q1 + q2 + q3)]
[                                            1,                                  1,                  1]
```

1.4 Either by looking at the matrix above or by again using the Matlab subs function, substitute in for variables q_1= q_2= q_3=0 and determine what is the instantaneous velocity in of the end-effector as changes? Confirm using mdl_planar3; p3.jacob0([0,0,0])

```
%% Q1 Derive 3-link Jacobian and use Matlab symbolic solver

% 1.2 From the derived Jacobian equation
syms l1 l2 l3 x y phi q1 q2 q3 Jq;

x = l1*cos(q1) + l2*cos(q1+q2) + l2*cos(q1+q2+q3);
y = l1*sin(q1) + l2*sin(q1+q2) + l2*sin(q1+q2+q3);
phi = q1 + q2 + q3;

% Compute the Jacobian
Jq = [diff(x,q1),diff(x,q2),diff(x,q3) ...
   ; diff(y,q1),diff(y,q2),diff(y,q3) ...
   ; diff(phi,q1),diff(phi,q2),diff(phi,q3)];

% 1.3 Solve for the link lengths being 1
JqForLength1 = subs(subs(subs(Jq,l1,1),l2,1),l3,1)

% 1.4 Solve for all joint angles being 0. By observation x velocity is 0
subs(subs(subs(JqForLength1,q1,0),q2,0),q3,0)

% Confirm this by using the toolbox
mdl_planar3;                                % Load 2-Link Planar Robot
p3.jacob0([0,0,0])
```

```
JqForLength1 =

[ - sin(q1 + q2 + q3) - sin(q1 + q2) - sin(q1), - sin(q1 + q2 + q3) - sin(q1 + q2), -sin(q1 + q2 + q3)]
[   cos(q1 + q2 + q3) + cos(q1 + q2) + cos(q1),   cos(q1 + q2 + q3) + cos(q1 + q2),  cos(q1 + q2 + q3)]
[                                            1,                                  1,                  1]


ans =

[ 0, 0, 0]
[ 3, 2, 1]
[ 1, 1, 1]


ans =

     0     0     0
     3     2     1
     0     0     0
     0     0     0
     0     0     0
     1     1     1
```

2 Dealing with Singularities

2.1 Load a 2-Link planar robot, and assign parameters for the simulation. Assign variable for small angle change

```
mdl_planar2;
t = ...;                    % Total time in seconds
steps = ...;                % No. of steps
deltaT = t/steps;           % Discrete time step
deltaTheta = 4*pi/steps;    % Small angle change
qMatrix = zeros(steps,2);   % Assign memory for joint angles
x = zeros(2,steps);         % Assign memory for trajectory
m = zeros(1,steps);         % For recording measure of manipulability
errorValue = zeros(2,steps);% For recording velocity error
```

2.2 Create a trajectory

```
for i = 1:steps
    x(:,i) = [1.5*cos(deltaTheta*i) + 0.45*cos(deltaTheta*i)
              1.5*sin(deltaTheta*i) + 0.45*cos(deltaTheta*i)];
end
```

2.3 Create the Transformation Matrix, solve the joint angles

```
T = [eye(3) [x(:,1);0];zeros(1,3) 1];
qMatrix(1,:) = p2.ikine(T,[0 0],M);
```

2.4 Use Resolved Motion Rate Control to solve joint velocities at each time step that will make the end-effector follow the Cartesian trajectory created.

```
for i = 1:steps-1
    T = ...                    % End-effector transform at current joint state
    xdot = ...                  % Velocity to reach next waypoint
    J = ...                     % Get Jacobian at current state (use jacob0)
    J = J(1:2,:);               % Take only first 2 rows
    m(:,i) = sqrt(det(J*J'));   % Measure of Manipulability
    qdot = ...                  % Solve the RMRC equation
    errorValue(:,i) = xdot - J*qdot; % Velocity error
    qMatrix(i+1,:)= ...         % Update the joint state
end
```

2.5 Now plot the trajectory and the error.

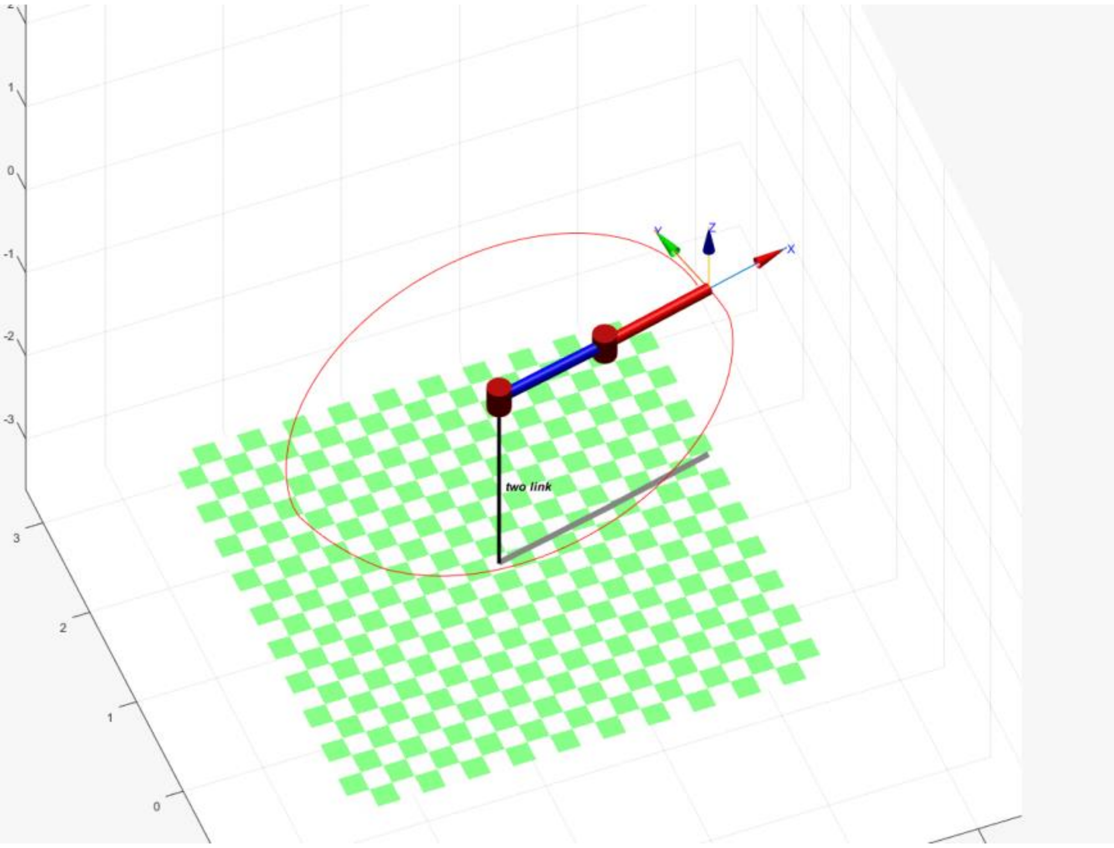
```
figure(1);
p2.plot(qMatrix,'trail','r-');       % Animate the robot
figure(2);
plot(m,'k','LineWidth',1);           % Plot the Manipulability
title('Manipulability of 2-Link Planar')
ylabel('Manipulability')
xlabel('Step')
figure(3)
plot(errorValue','Linewidth',1);          % Plot the velocity error
ylabel('Error (m/s)')
xlabel('Step')
legend('x-velocity','y-velocity');
```

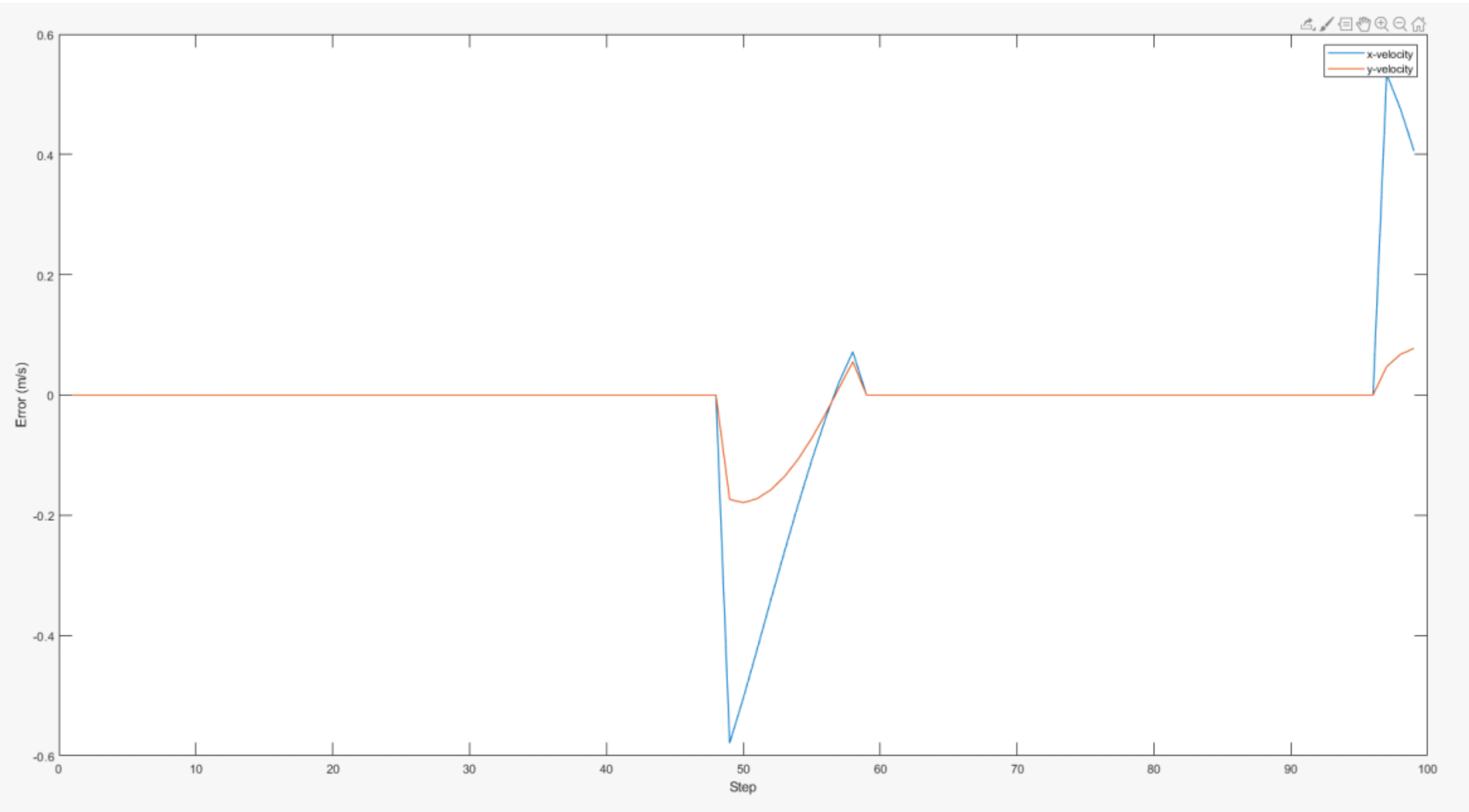
2.6 What do you notice about the manipulability when there is trajectory error?

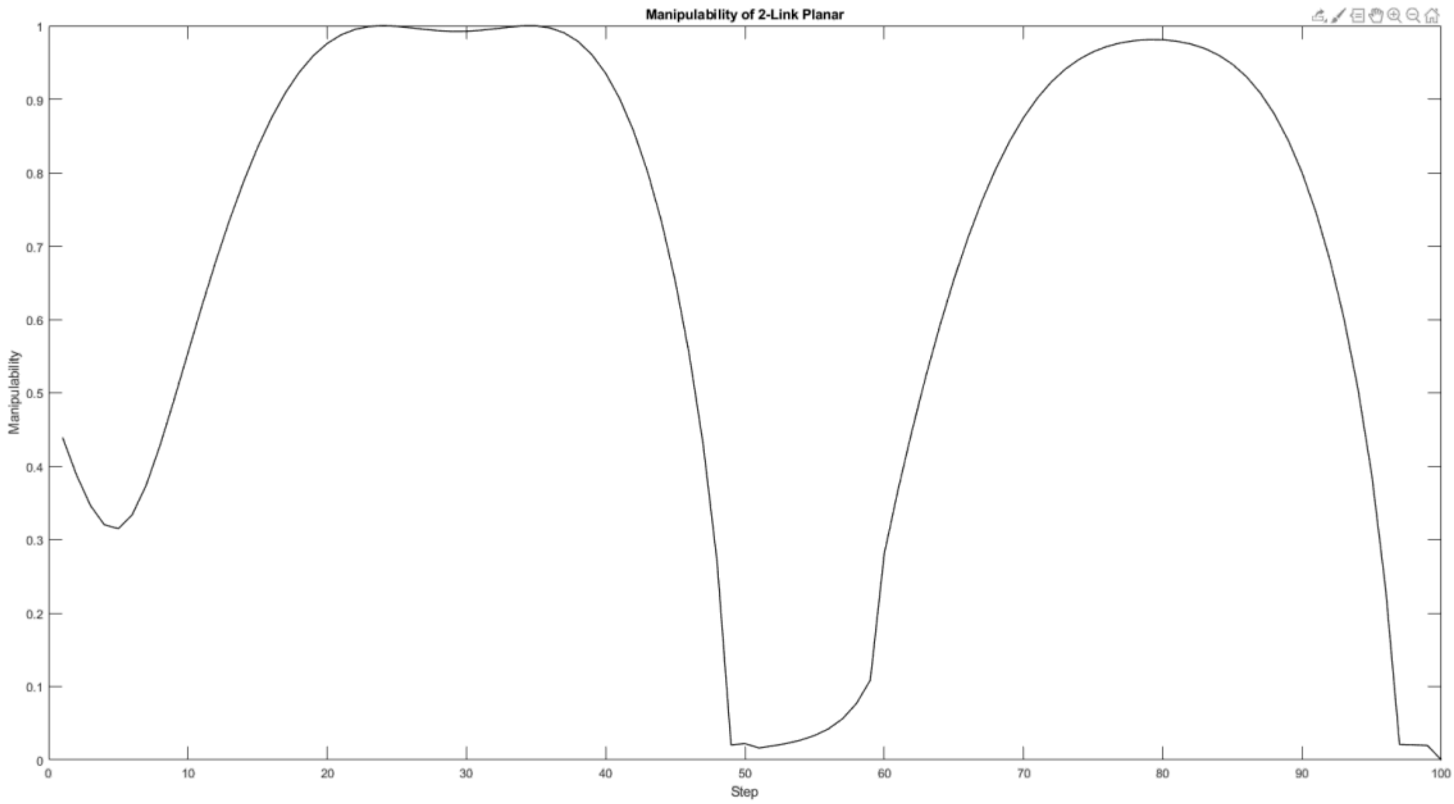
2.7 You will probably notice that the robot arm loses control at certain points. Try applying Damped Least Squares and see what happens.

%% Q2 Dealing with Singularities

```
close all
clc
mdl_planar2;                                  % Load 2-Link Planar Robot
M = [1 1 zeros(1,4)];                           % Masking Matrix
deltaT = 0.05;                               % Discrete time step

minManipMeasure = 0.1;
steps = 100;
deltaTheta = 2*pi/steps;
x = [];

for i = 1:steps
    x(:,i) = [1.5*cos(deltaTheta*i) + 0.45*cos(deltaTheta*i)
          1.5*sin(deltaTheta*i) + 0.45*cos(deltaTheta*i)];
end

T = [eye(3) [x(:,1);0];zeros(1,3) 1];
% Try with ikine
qMatrix(1,:) = p2.ikine(T,[0 0],M)
% Try with ikcon
qMatrix(1,:) = p2.ikcon(T,[0 0])

m = zeros(1,steps);
error = nan(2,steps);
for i = 1:steps-1
    xdot = (x(:,i+1) - x(:,i))/deltaT;                % Calculate velocity at discrete time step
    J = p2.jacob0(qMatrix(i,:));                      % Get the Jacobian at the current state
    J = J(1:2,:);                              % Take only first 2 rows
    m(:,i)= sqrt(det(J*J'));                       % Measure of Manipulability
    if m(:,i) < minManipMeasure
       qdot = inv(J'*J + 0.01*eye(2))*J'*xdot;
    else
       qdot = inv(J) * xdot;                       % Solve velocitities via RMRC
    end
    error(:,i) = xdot - J*qdot;
    qMatrix(i+1,:) = qMatrix(i,:) + deltaT * qdot';            % Update next joint state
end

figure(1)
set(gcf,'units','normalized','outerposition',[0 0 1 1])
p2.plot(qMatrix,'trail','r-')                 % Animate the robot
figure(2)
plot(m,'k','LineWidth',1);                    % Plot the Manipulability
title('Manipulability of 2-Link Planar')
ylabel('Manipulability')
xlabel('Step')
figure(3)
plot(error','Linewidth',1)
ylabel('Error (m/s)')
xlabel('Step')
legend('x-velocity','y-velocity');
```


two link


x-velocity
y-velocity
Error (m/s)
Step




```
qMatrix =

    0.3265   -0.1085

qMatrix =

    0.4913   -0.4548
```

3 Depth Image

3.1 Download the sequence of depth images, "imageData.mat" on UTSOnline captured with an Asus XTion Pro[1]

(a)	(b)	(c)

Fig 1. (a) Asus Xtion Pro camera is similar in design and function to the original Xbox Kinect and the Primesense, (b) Example depth image from the sensor - the higher the pixels intensity, the closer to the camera the surface that reflected the pattern is, (c) The coordinate frame of the camera, note that depth values are the distances along the Z-axis.

3.2 Load the data (i.e. load('imageData.mat')) and iterate through the images and show them with imshow

3.3 The "depth" values are different from "distance" values since they are the distance from the image plane to the reflecting surface, whereas for the laser (in previous lab exercises) it is truly the distance from the center of the sensor. Use surf to plot the depth values in 3D.

3.4 Given that the cameras vertical Field Of View (FOV) is 45' and horizontal FOV is 58' then given the resolution of each image (i.e. 160x120), determine the angular resolution between each distance measurement

3.5 Given a depth image resolution of 320x240 what is the distance between points on a spot directly in front of the camera on a wall 1m away?

3.6 If the resolution was increased to 640x480, what is the new distance between points (and point density).

3.7 Consider a depth image. To determine a point cloud (i.e. [x,y,z] points for each depth pixel) , the "depth" value is the Z value and the X, and Y values are determined by the FOV and the index of the depth value. Use a "for" loop to iterate through each depth value, and its index (i.e. implied angle from the Z axis around both X and Y axes), and create a point cloud. The following formula allows for this using pre-computed values[2]

```
X_TO_Z = 1.114880018171494
Y_TO_Z = 0.836160013628620
x = (j /colCount -0.5) * depthValue(i,j) * X_TO_Z;
y = (0.5 -i / rowCount) * depthValue(i,j) * Y_TO_Z;
point = [x, y, depthValue(i,j)];
```

3.8 **(Bonus)** Consider the camera is now mounted to a Schunk robot, move it to a few distinct poses and compute the new point cloud determined from the given pose. See the video of exploration and mapping on Grit-blasting Schunk Robot (currently unavailable, check back soon for an updated link): https://www.youtube.com/watch?v=rmPLRbXgW0g&feature=youtu.be

```
%% Q3
% To get the data from Remote lab (use the appropriate instructions)
try rosinit('138.25.213.85');end
depthRawSub = rossubscriber('/camera/depth/image_raw'); % Depends upon the camera in use
(refer to instructions)
rgbRawSub = rossubscriber('/camera/rgb/image_raw'); % Depends upon the camera in use (refer to
instructions)
pause(2);
depthImg = readImage(depthRawSub.LatestMessage);
rgbImg = readImage(rgbRawSub.LatestMessage);

% To store the data
maxImages = 20;
depthImageData = repmat(depthImg,1,1,maxImages);
rgbImageData = repmat(rgbImg,1,1,1,maxImages);
for i = 1:maxImages
  depthImageData(:,:,i) = readImage(depthRawSub.LatestMessage);
  rgbImageData(:,:,:,i) = readImage(rgbRawSub.LatestMessage);
  pause(0.1);
end
save('imageData.mat','depthImageData','rgbImageData')

%% 3.2 (load)
load('imageData.mat');

%% 3.2 (show)
% Show the data from the Depth and RGB images
for i = 1:maxImages
  imshow(histeq(depthImageData(:,:,i)));
  imshow(rgbImageData(:,:,:,i));
  drawnow();
  pause(0.5);
end

%% 3.3
surf(histeq(depthImageData(:,:,1)));

%% 3.4
widthFOV = 58;
heightFOV = 45;
widthResolutionDegrees = widthFOV / size(depthImageData,2);
heightResolutionDegrees = heightFOV / size(depthImageData,1);

%% 3.5 (point spacing at 1m) Medium res
widthSpacing = tan(deg2rad(widthResolutionDegrees));
heightSpacing = tan(deg2rad(heightResolutionDegrees));

%% 3.6 (point spacing at 1m) High res
widthSpacing = tan(deg2rad(widthResolutionDegrees));
heightSpacing = tan(deg2rad(heightResolutionDegrees));

%% 3.7 Point Cloud
X_TO_Z = 1.114880018171494;
Y_TO_Z = 0.836160013628620;
rowCount = size(depthImageData,1);
colCount = size(depthImageData,2);
pointMillimeters = zeros(colCount*rowCount,3);
for i = 1:rowCount
  for j = 1:colCount
    x = (j /colCount -0.5) * depthImageData(i,j,1) * X_TO_Z;
    y = (0.5 -i / rowCount) * depthImageData(i,j,1) * Y_TO_Z;
    pointMillimeters((i-1) * colCount + j,:) = [x, y, depthImageData(i,j,1)];
  end
end
pointMeters = pointMillimeters / 1000;

%% 3.8 (Bonus) Eye-In-Hand
robot = SchunkUTSv2_0;
q = [0,pi/2,0,0,0,0];
robot.plot3d(q);
camlight
hold on;
plot_h = plot3(0,0,0,'.r');
for joint2Rads = pi/2 : -0.1:0
  q = [0,joint2Rads,0,0,0,0];
  tr = robot.fkine(q);
  transformedPoints = [tr * [pointMeters,ones(size(pointMeters,1),1)]']';
  transformedPoints = transformedPoints(:,1:3);
  plot_h.XData = transformedPoints(:,1);
  plot_h.YData = transformedPoints(:,2);
  plot_h.ZData = transformedPoints(:,3);
  robot.animate(q);
  axis equal
  drawnow();
end
%%
end
```