Kinematics with Robotics Toolbox


```
clc; clear all; close all;
```

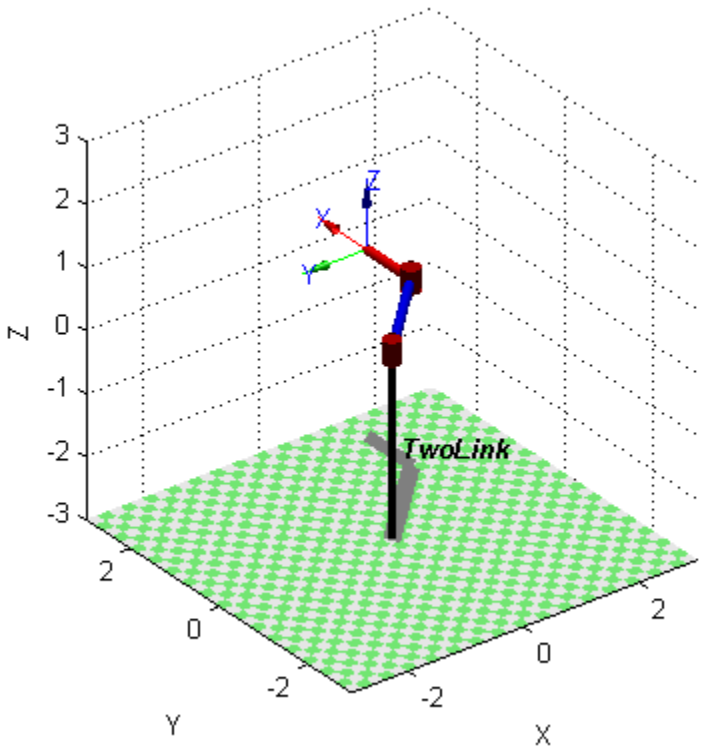
(1) Two Link Manipulator:

[1.1] Construction of the robot:

Define each link of the manipulator with its own DH parameters:

```
L(1) = Link([0 ,0, 2,0, 0]); % [theta, d, a, alpha, P=1/R=0]
L(2) = Link([0, 0, 1, 0, 0]); % theta, d, a, alpha, P=1/R=0
% We can find the properties of each link like its length, transformation
% matirx, alpha,.. etc
L(1).A(pi/2);  % transformation matrix for link 1 i.e. A1 at theta=pi/2
```

```
% Define the two links manipulator with function SerialLink
twoLinkRobot = SerialLink(L,'name','TwoLink'); % the robot name will apear on its plot
 twoLinkRobot.plot([pi/4 pi/4]); % plot the robot at certain configuration (theta1 and
theta2)
```



[1.1] Forward Kinematics:

find the pose (rotation matrix and translation) matrix for a given angles values. This matrix is the transformation between the endeffector and the robot base:

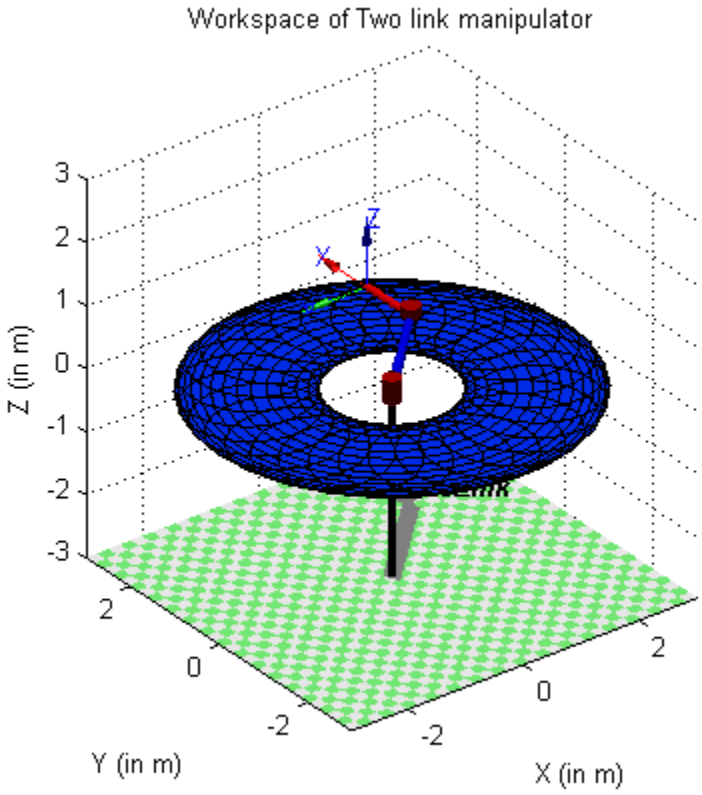
```
T1 = twoLinkRobot.fkine([pi/4 pi/4]); % T1 is the transformation matrix
T2 = twoLinkRobot.fkine([pi pi/3]); % T2 is the transformation matrix
```

[1.2] Plot the workspace:

To plot the workspace, we can vary theta2 and theta3 to calculate the end-effector's position and plot it using the surf function.

```
xlabel('X (in m)');
ylabel('Y (in m)');
zlabel('Z (in m)');
title('Workspace of Two link manipulator');
```

```
hold on;
N=30;
for i= 1:N+1
for j= 1:N+1
TR = twoLinkRobot.fkine([2*pi*(i-1)/N 2*pi*(j-1)/N]);
workspace(i,j,:) = TR(:,4);
end
end
surf(workspace(:,:,1), workspace(:,:,2), workspace(:,:,3));
```



[1.3] Inverse Kinematics:

Numerical solver for the inverse kinematics

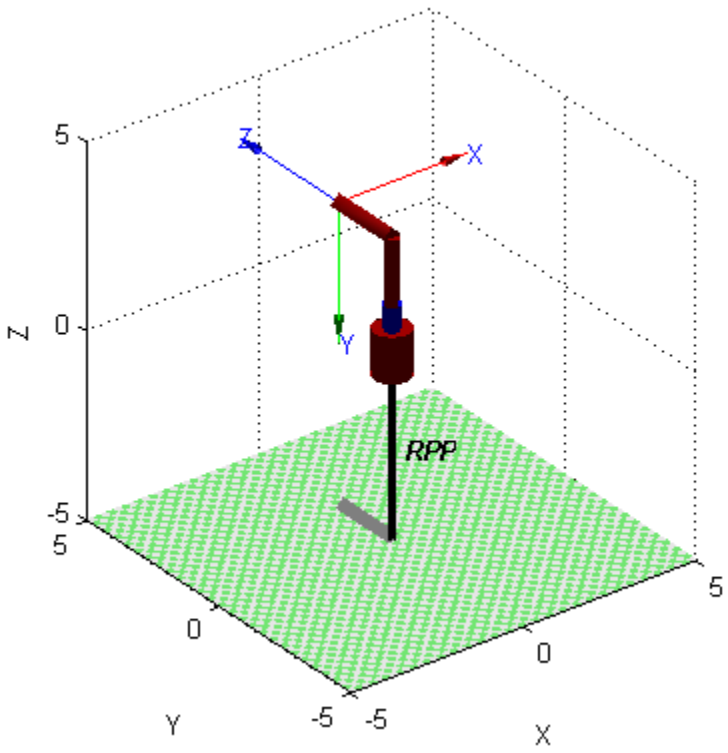
```
q = twoLinkRobot.ikine(T1,[0 0],[1 1 0 0 0 0]);
% T1 is the desired pose
% [0 0] is the inital values for the joints theta1 and theta2
% [1 1 0 0 0 0] is the error mask for x, y, z, roll, pith and yaw.
% Since the given T is over constrained for the two link manipulator,
% we specifiy the interest components to be considered. Here we specify to search in x
and y and ignore others.
%--------------------------------------------------------------------------%
clc; clear all; close all;
```

```
Warning: Initial joint configuration results in a
(near-)singular configuration, this may slow convergence
```

(2) three links RPP Robot:

[2.1] Robot Construction:

```
L(1) = Link([0, 1, 0, 0, 0]); % theta, d, a, alpha, P=1/R=0
L(2) = Link([0, 0, 0, -pi/2, 1]); % theta, d, a, alpha, P=1/R=0
L(3) = Link([0, 0, 0, 0, 1]); % theta, d, a, alpha, P=1/R=0
%
threeLinkRobot = SerialLink(L,'name','RPP');
% Since the prismatic joint d value can take any value to infinity, when we have
% a prismatic joint, we should specify the workspace range.
threeLinkRobot.plot([0 2 2],'workspace', [-5 5 -5 5 -5 5]);  % workspace [Xmin, Xmax,
Ymin, Ymax, Zmin, Zmax]
```

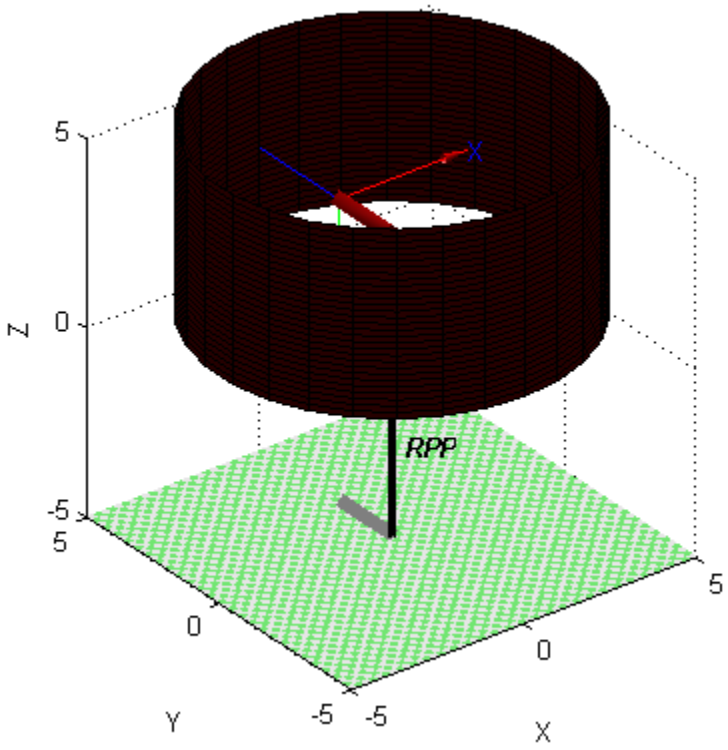


[2.2] Forward Kinematics:

```
T3 = threeLinkRobot.fkine([0 .3 .5]);
T4 = threeLinkRobot.fkine([pi/2 0.3 -0.5]);
```

[2.3] Workspace Kinematics:

```
hold on;
N=30;
for i= 1:N+1
for j= 1:N+1
TR = threeLinkRobot.fkine([2*pi*(i-1)/N 5*(j-1)/N 5]); % 0-->2 pi   , 0--> 5m,  5m
workspace(i,j,:) = TR(:,4);
end
end
surf(workspace(:,:,1), workspace(:,:,2), workspace(:,:,3));
```



[2.4] Inverse Kinematics:

Numerical solver for the inverse kinematics

```
qp = threeLinkRobot.ikine(T3,[0 0 0],[1 1 1 0 0 0]);
% T3 is the desired pose
% [0 0] is the inital values for the joints theta1 and theta2
% [1 1 1 0 0 0] is the error mask for x, y, z, roll, pith and yaw.
% Since the given T is over constrained for the RPP manipulator,
% we specifiy the interest components to be considered. Here we specify to search in
x, y and z and ignore orientation.
```

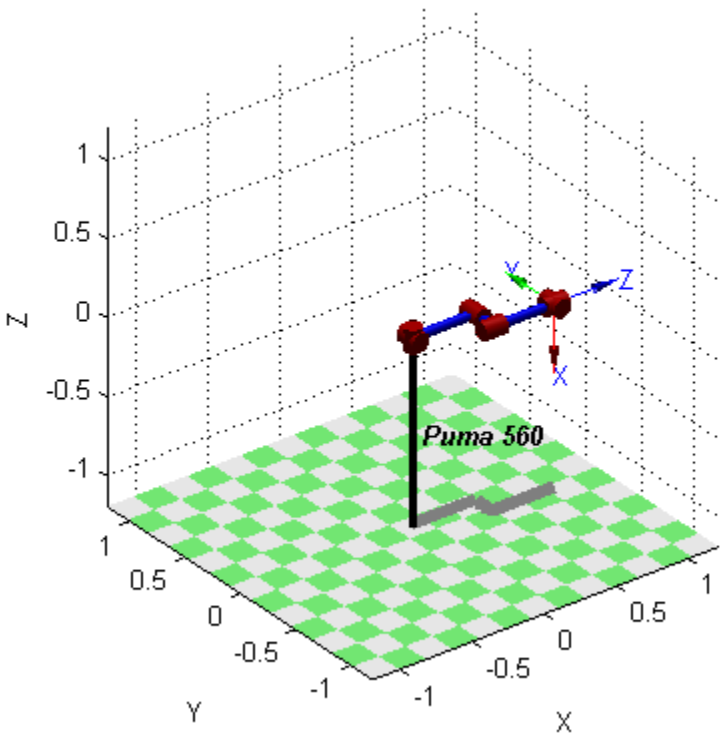
```
%--------------------------------------------------------------------------
clc; clear all; close all;
```

```
Warning: Initial joint configuration results in a
(near-)singular configuration, this may slow convergence
```

(3) PUMA 560 Manipulator (6 DoF):

[3.1] Loading the model:

```
mdl_puma560; % An already defined model from the robotics toolbox with p560 handler
p560.plot(qs);
% Four configurations are defined in the puma model:
% qz: zero angle (0,0,0,0,0,0)
% qr: ready, arm is straight and vertical (0,90,-90,0,0,0)
% qs: strech, arm is straight and horizontal (0,0,-90,0,0,0)
% qn: nominal, arm is in general workspace (0, 45, -180, 0, 45 0)
```

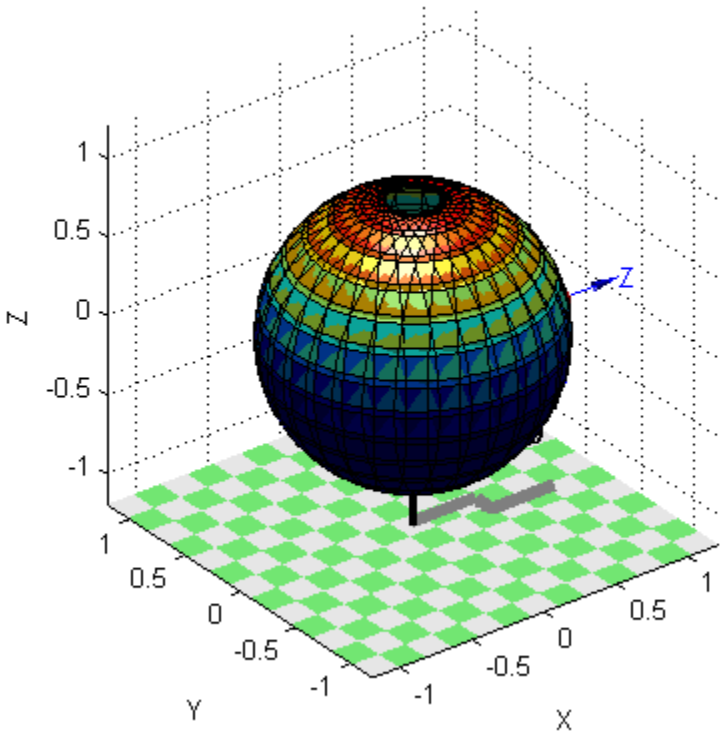


[3.2] Forward Kinematics:

```
T5 = p560.fkine(qn);
T6 = p560.fkine([pi/2 pi/4 0 -pi/6 pi/5 0]);
```

[3.3] Workspace:

```
hold on;
N=30;
for i= 1:N+1
for j= 1:N+1
TR = p560.fkine([2*pi*(i-1)/N 2*pi*(j-1)/N -pi/2 0 0 0]); % 0-->2 pi  ,  0-->2 pi , -pi/2,0 ,0,0
workspace(i,j,:) = TR(:,4);
end
end
surf(workspace(:,:,1), workspace(:,:,2), workspace(:,:,3));
```



[3.4] Inverse Kinematics:

Here we will use the spherical wrist assumption and decouple the position and orientation inverse problem to find a closed form solution for the IK

```
%The available function is ikine6s
q_puma = p560.ikine6s(T6); % different solution of q_puma from the one specified by forward kinematics T6 = p560.fkine([pi/2 pi/4 0 -pi/6 pi/5 0]);
% We can specify which solution we need:
% ru: Right upper
```

```
% lu: Left upper
% rl: Right lower
% ll: Left lower
q_puma = p560.ikine6s(T6,'ru'); % (1)
q_puma = p560.ikine6s(T6,'rl');  % (2)
% (1) and (2) are different joints values for the same required pose.
```

Published with MATLAB® R2013b