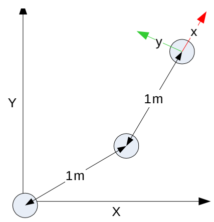
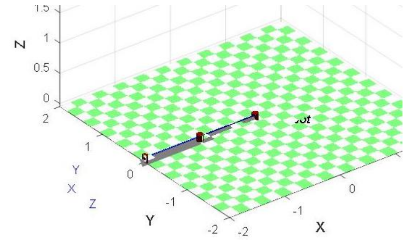
# Lab 4 Exercises

Forward kinematics allows us to work out the end-effector transform given a set of joint angles. Conversely, given an end-effector transform, we can compute a set of joint angles to achieve it.

## 1 Drawing with a 3DOF Planar Arm



### 1.1 Make the 3DOF planar arm model

```
L1 = Link('d',0,'a',1,'alpha',0,'qlim',[-pi pi])
L2 = Link('d',0,'a',1,'alpha',0,'qlim',[-pi pi])
L3 = Link('d',0,'a',1,'alpha',0,'qlim',[-pi pi])
robot = SerialLink([L1 L2 L3],'name','myRobot');
```

### 1.2 Rotate the base around the Y axis so the Z axis faces down ways

```
robot.base = troty(pi);
```

### 1.3 Set workspace, scale and initial joint state, then plot and teach

```
q = zeros(1,3);
robot.plot(q,'workspace',[-2 2 -2 2 -0.05 2],'scale',0.5);
robot.teach;
```

### 1.4 Move the robot around with “teach” and observe that there is no way to affect the Z, roll or pitch values, no matter what joint value you choose.

### 1.5 Consider a pen that is mounted to the Z axis of the final joint. Note how this means we don't care about the yaw angle (if pen where the Z axis rotating it doesn't affect the drawing result)

### 1.6 Get a solution for the end effector at [-0.75,-0.5,0], make sure you mask out the impossible-to-alter values (i.e. z, roll and pitch) and the value we don’t care about (i.e. yaw), thus we need a mask of [1,1,0,0,0,0]:

```
newQ = robot.ikine(transl(-0.75,-0.5,0),q,[1,1,0,0,0,0]);
```

### 1.7 Plot the new joint state and check how close it got to the [x, y] in the goal transform (i.e. transl(-0.75,-0.5,0))

```
robot.plot(newQ)
robot.fkine(newQ)
```

### 1.8 Go through a loop using the previous joint as the guess to draw a line from [-0.75,-0.5,0] to [-0.75,0.5,0] and animate each new joint state trajectory with:

```
robot.animate(newQ);
```

### 1.9 (Bonus) Instead of trusting “animate”, do your own interpolation between joint states and keep track of the [x,y] location of points using fkine. How straight is the actual line drawn if you see every point in between?

### 1.10 (Bonus) Using ikine to get the newQ and fkine to determine the actual point, and animate to move robot “draw” a circle around the robot with a radius of 0.5m

## 2 Inverse Kinematics to Determine Joint Angles of Puma 560

### 2.1 Load a model of the Puma 560 robot

```
mdl_puma560
```

### 2.2 Define our first end-effector pose as a 4x4 Homogeneous Transformation Matrix:

```
T1 = transl(0.5,-0.4,0.5);
```

### 2.3 Solve the inverse kinematics to get the required joint angles

```
q1 = p560.ikine(T1);
```

### 2.4 Define the second end-effector pose as a 4x4 Homogeneous Transformation Matrix:

```
T2 = transl(0.5,0.4,0.1);
```

### 2.5 Solve the inverse kinematics to get the required joint angles

```
q2 = p560.ikine(T2);
```

## 3 Interpolation: Getting from A to B

3.1 Download from UTSOnline and read "Week 4 - Trajectory Interpolation Reference Guide.pdf"

3.2 Load a puma560 robot, use the q1 and q2 from the previous ikine exercise.

3.3 Generate a matrix of interpolated joint angles with 50 steps between q1 and q2 using the following two methods then **do steps 3.4 to 3.10 for each method**

| Method 1: Quintic Polynomial Profile | Method 2: Trapezoidal Velocity Profile |
|---|---|
| `qMatrix = jtraj(q1,q2,steps);` | `s = lspb(0,1,steps);`<br>`qMatrix = nan(steps,6);`<br>`for i = 1:steps`<br>`    qMatrix(i,:) = (1-s(i))*q1 + s(i)*q2;`<br>`end` |

3.4 Animate the generated trajectory and the Puma 560 robot. Note how 'trail','r-' will draw a red line of the end-effector trajectory.

```
figure(1)
p560.plot(qMatrix,'trail','r-')
```

3.5 Create matrices of the joint velocities and accelerations for analysis:

```
velocity = zeros(steps,6);
acceleration  = zeros(steps,6);
for i = 2:steps
    velocity(i,:) = qMatrix(i,:) - qMatrix(i-1,:);
    acceleration(i,:) = velocity(i,:) - velocity(i-1,:);
end
```

3.6 Plot the joint angles, velocities, and accelerations. When plotting the joint angles, we will use qlim to draw reference lines for the upper and lower joint limits .

```
for i = 1:6
    figure(2)
    subplot(3,2,i)
    plot(qMatrix(:,i),'k','LineWidth',1)
    title(['Joint ', num2str(i)])
    xlabel('Step')
    ylabel('Joint Angle (rad)')
    refline(0,p560.qlim(i,1))      % Reference line on the lower joint limit for joint i
    refline(0,p560.qlim(i,2))      % Reference line on the upper joint limit for joint i

    figure(3)
    subplot(3,2,i)
    plot(velocity(:,i),'k','LineWidth',1)
    title(['Joint ', num2str(i)])
    xlabel('Step')
    ylabel('Joint Velocity')

    figure(4)
    subplot(3,2,i)
    plot(acceleration(:,i),'k','LineWidth',1)
    title(['Joint ', num2str(i)])
    xlabel('Step')
    ylabel('Joint Acceleration')
end
```

3.7 Consider the following:

- Joint limitations – Certain inverse kinematic solutions will give infeasible joint angles
- Joint velocities – appreciation for the different between Quintic Polynomials and Trapezoidal Velocity Profiles
- End-effector path – The end-effector will not follow a straight lines from one pose to another