Aalto University
School of Electrical
Engineering

Inverse kinematics (numerical solution); Trajectories

ELEC-C1320 Robotics

Pekka Forsman

Topics

- Inverse kinematics (numerical solution)
  - Redundant mechanisms
- Trajectories
  - Joint-space motion
  - Cartesian motion
  - Motion through a singularity
- Example problems

Inverse kinematics (numerical solution)

For the case of robots which do not have six joints and a spherical wrist it could be the best (or only) option to use an iterative numerical solution to calculate inverse kinematics.

We can think of the numerical inverse kinematics problem as one of adjusting the joint coordinates until the forward kinematics matches the desired pose. More formally this is an optimization problem – to minimize the error between the forward kinematic solution and the desired pose ξ^*. (Corke, p. 206)

$$\boldsymbol{q}^* = \arg\min_{\boldsymbol{q}} \left\| \mathcal{K}(\boldsymbol{q}) \ominus \xi^* \right\|$$

For example for a two link planar manipulator with (x,y)-position coordinates as the task space the optimization equation becomes

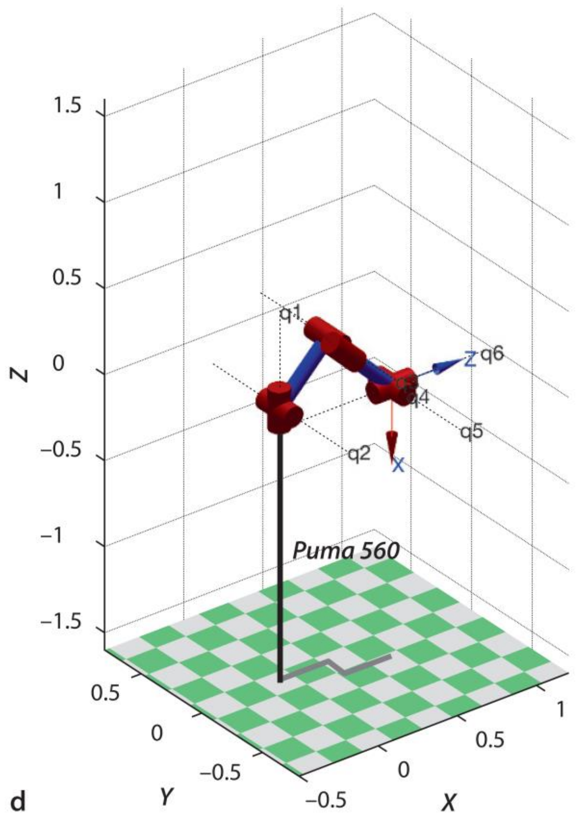
$$E(q) = \left\| [\mathcal{K}(\boldsymbol{q})]_t - \left(x^* \ y^*\right)^T \right\|$$

An example, Puma 560 robot:

For p560, the nominal, dexterous working joint configuration **qn** is (q1=0, q2=0.7854, q3=3.1416, q4=0, q5=0.7854, q6=0)

For which the end-effector pose in form of a arm matrix **T** is

```
>> T = p560.fkine(qn)
T =
   -0.0000    0.0000    1.0000    0.5963
   -0.0000    1.0000   -0.0000   -0.1501
   -1.0000   -0.0000   -0.0000   -0.0144
         0         0         0    1.0000
```



Now, if we compute (numerically) the joint configuration corresponding to the arm matrix, T, we get a guite different set of joint values compared to the original one:

```
>> qi = p560.ikine(T)
qi =
     0.0000    -0.8335     0.0940    -0.0000    -0.8312     0.0000
```

which is quite different from the original value

```
>> qn
qn =
          0     0.7854     3.1416          0     0.7854          0
```

Again, if we calculate the forward kinematics by using these values we get the same result for the arm matrix as before:

```
>> p560.fkine(qi)
ans =
   -0.0000     0.0000     1.0000     0.5963
   -0.0000     1.0000    -0.0000    -0.1501
   -1.0000    -0.0000    -0.0000    -0.0144
         0          0          0     1.0000
```

here qi is in fact the elbow down configuration for the same tool frame pose T.

A limitation of this general numeric approach is that it does not provide explicit control over the arm's kinematic configuration as did the analytic approach – the only control is implicit via the initial values of joint coordinates (which default to zero).

In the example, if we specify the initial joint coordinates properly, the solution converges to the elbow-up configuration:

As would be expected the general numerical approach of ikine is **considerably slower** than the analytic approach of ikine6s. However it has the great advantage of being able to work with manipulators at singularities and manipulators with less than six or more than six joints

However, for **under-actuated mechanisms** (i.e. nb. of DOF < 6), it is a good idea to restrict the task space such that an analytical inverse kinematics solution can be acquired (*compare slides of lecture 6*)

Redundant mechanisms

A redundant manipulator is one with more than six joints.

The extra Degrees Of Freedom (DOF) can be utilized for expanding the robot task/motion space or for avoiding singular configurations of the robot arm.

A typical example of redundant mechanisms is a six axes industrial robot mounted on a one or two axes platform.

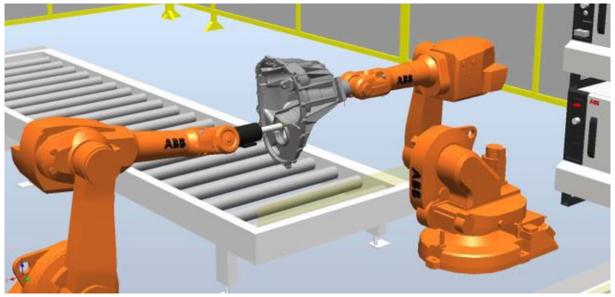
A criteria for having a true redundant mechanism is that all degrees of freedom can be controlled syncronously for moving the robot tool frame. Control units of most of the industrial robot brands have indeed io-connections and software functionalities to expand the number of DOFs in this way.

In a simpler case, the platform on which the robot has been mounted is used just to transport the robot between work stations. Then, within the work stations the robot operates as a normal 6 DOF robot arm.

Examples of redundant mechanisms include:

- 6 DOF-robot mounted on a linear track
- 6 DOF-robot mounted on a mobile platform
- Dexterous robot hands (e.g. Barrett robot hand)

- Team of cooperating manipulators



Programming teams of cooperating robots

source: ABB Application manual MultiMove, RobotWare 6.05

Available approaches for motion synchronization:

1. **Independent movements**

If the different task programs, and their robots, work independently, no synchronization or coordination is needed.

Sometimes, even if the movements do not need to be coordinated, the task programs can have dependencies. For example, if one robot leaves an object that a second robot will pick up, the first robot must finish with the object before the second robot can grab it.

These interactions can be solved for example with:
• the instruction WaitSyncTask
• I/O signals

2. Semi coordinated movements

Several robots can work with the same work object, without synchronized movements, as long as the work object is not moving.

A positioner can move the work object when the robots are not coordinated to it, and the robots can be coordinated to the work object when it is not moving.

The positioner must know when the work object can be moved, and the robots must know when they can work on the work object. However, it is not required that every move instruction is synchronized.

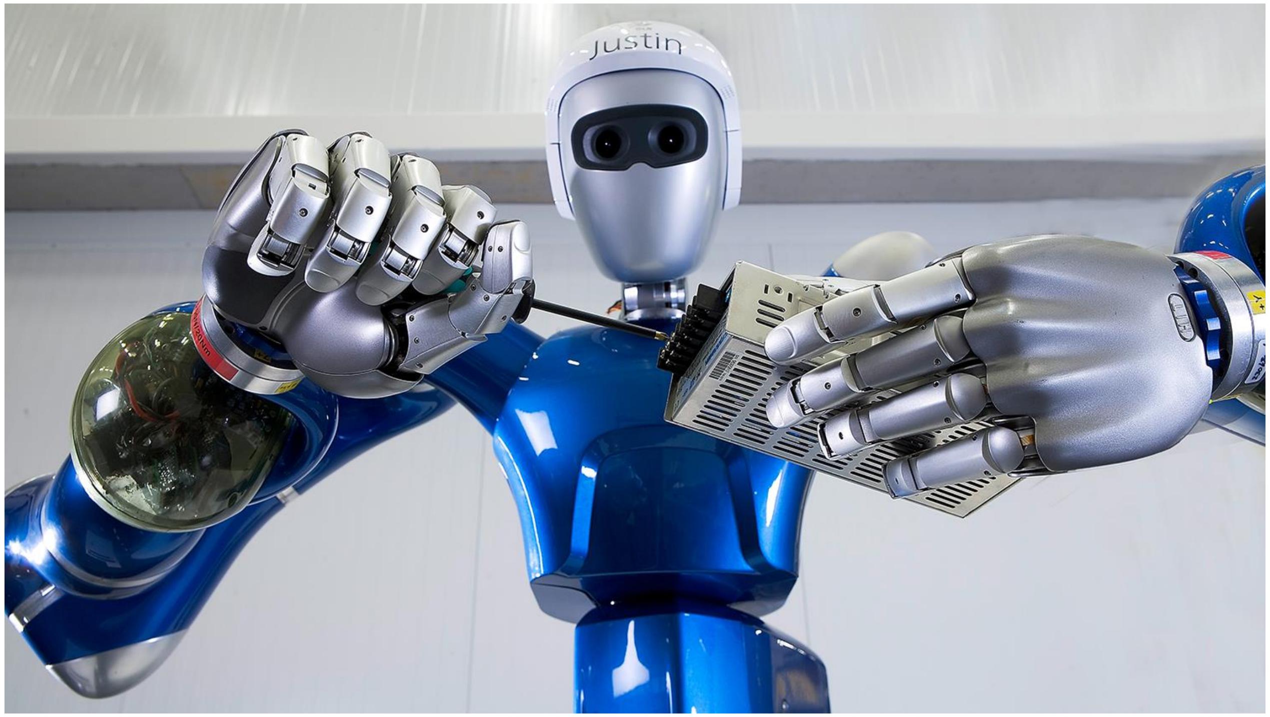
3. Coordinated synchronized movements

Several robots can work with the same moving work object.
The positioner or robot that holds the work object and the robots that work with the work object must have synchronized movements.

Coordinated synchronized movements usually save cycle time since the robots do not have to wait while the work object is being moved.

https://www.youtube.com/watch?v=SOESSCXGhFo

- Humanoid robots (e.g. Justin the robot (53 DOF)- source: DLR - Institute for robotics and mechatronics, Germany)



https://www.youtube.com/watch?v=3107ELGw4JA

For redundant mechanisms there are usually an infinite number of joint configurations to reach the desired tool pose. Optimization objectives to choose one of the joint configurations include:

1. Joint range (give more weight for the joint configurations that remain closer to the middle of the motion space of each robot joint – minimize a weighted sum of the distances)

2. Manipulability (maximize the distance of joint control values w.r.t singular configurations)

3. Obstacle avoidance (maximize the clearance between all parts of the robot arm and the workspace obstacles – requires knowledge of work cell object locations and dimensions)

Trajectories - Joint-space motion

One of the most common requirements in robotics is to move the end-effector smoothly from pose A to pose B. Building on what we learnt in Sec. 3.1 (compare slides of lecture 4) we will discuss two approaches for generating such trajectories:

Straight lines in joint space (also called **joint interpolated motion**) and straight lines in Cartesian space (also called **linear interpolated motion**).

We can now demonstrate joint interpolated motion with the robotics toolbox:

First specify the Cartesian start and finish poses for the trajectory and calculate the corresponding joint configurations with the inverse kinematics method of p560 robot

```
>> T1 = transl(0.4, 0.2, 0) * trotx(pi);
>> T2 = transl(0.4, -0.2, 0) * trotx(pi/2);
```

```
>> q1 = p560.ikine6s(T1);
>> q2 = p560.ikine6s(T2);
```

Create a time vector for a 2 s motion in 50 ms steps

```
>> t = [0:0.05:2]';
```

Create a smooth joint space trajectory and plot it – see fig (a) on the next slide

```
>> q = p560.jtraj(T1, T2, t)
```

```
>> qplot(t, q);
```

Create a corresponding Cartesian space trajectory and plot its locus of x- and y-coordinates, fig c

```
>> T = p560.fkine(q);     >> p = transl(T);     >> plot(p(:,1), p(:,2))
```

also plot the roll,pitch and yaw-angles

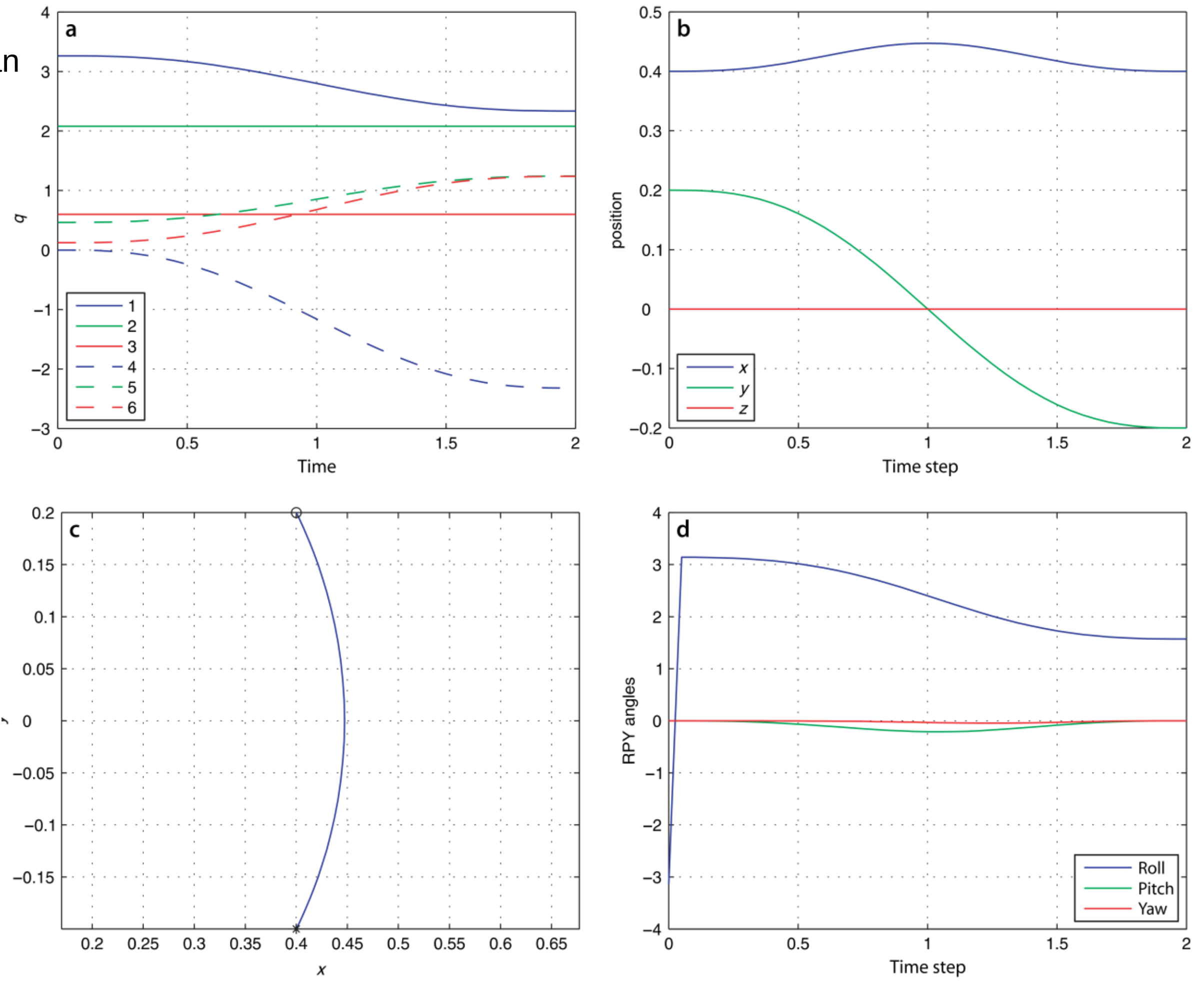
```
>> plot(t, tr2rpy(T))
```

Joint space motion. **a** Joint coordinates versus time; **b** Cartesian position versus time; **c** Cartesian position locus in the xy-plane **d** roll-pitch-yaw angles versus time (Corke, p.154)

The path in fig c is not a straight line. This is to be expected since we only specified the Cartesian coordinates of the end-points. As the robot rotates about its waist joint during the motion the end-effector will naturally follow a circular arc.

Note that the roll angle varies from π to π/2 as was specified.

Also note that the initial roll angle is shown as −π but at the next time step it is π . This is an artifact of finite precision arithmetic and both angles are equivalent on the circle.



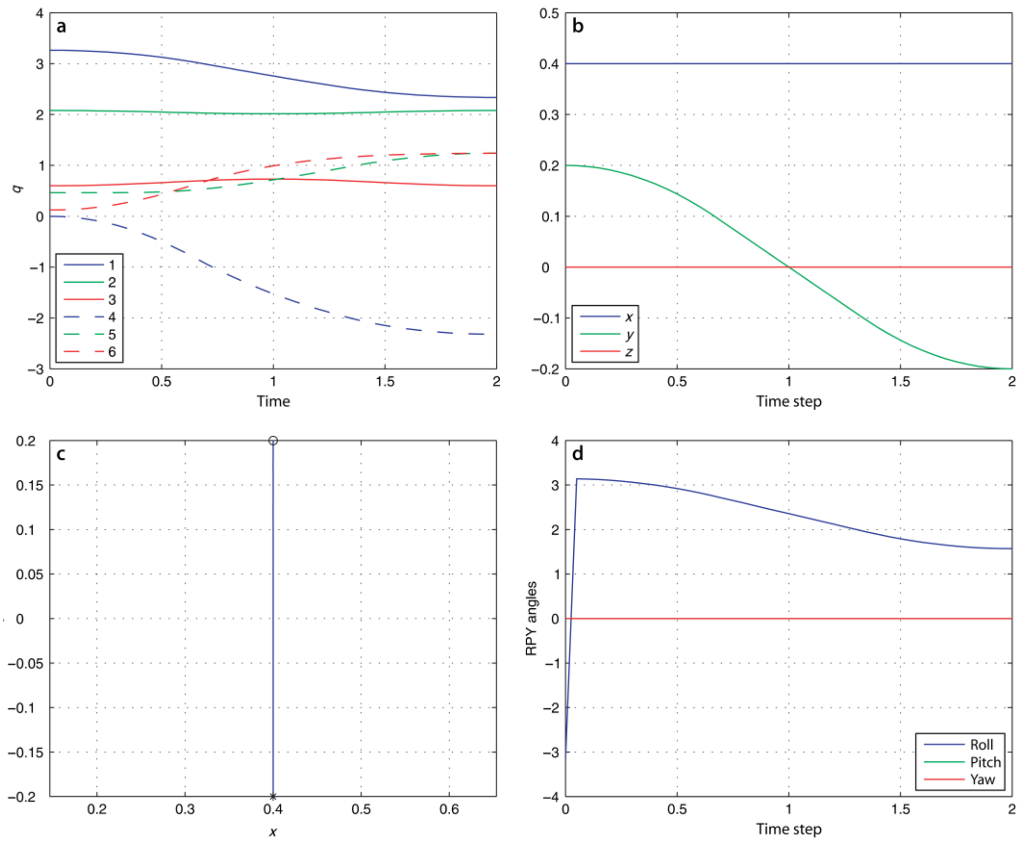
Cartesian motion

In practice, all motions, which require precise, pre-determined motion of the tip of the robot tool must apply Cartesian space linear (or circular etc.) motion.

In industrial robot applications **joint interpolated motion** is typically used for moving from one approach position to another, via an open space without the requirement for moving along a straigth line with a fixed or linearly changing cartesian space orientation.

All other motions, which, for example, move the tool along/parallel to the object contours require linear (or circular) path for the tool frame to follow.

With the robotics toolbox, Cartesian space linear motion can be implemented with ctraj-function, which takes as arguments the initial and final pose, T1 and T2, and the number of time steps, n, and it returns the trajectory as a 3-dimensional matrix, (4 × 4 × n): `>> Ts = ctraj(T1, T2, length(t));`



Cartesian motion. **a** Joint coordinates versus time; **b** Cartesian position versus time; **c** Cartesian position locus in the xy-plane; **d** roll-pitch-yaw angles versus time. (Corke, p.156)

We can see that now the locus of the x,y coordinates is linear (fig c) and that the pitch-angle stays at its constant value (zero) during the whole motion (fig d).

Motion through a singularity

When the Cartesian trajectory of the robot tool frame moves, at the given velocity, near-by or through a singularity, the corresponding, required velocities and accelerations of some of the robot joints may exceed the allowable limits which would disturp or interrupt robot task execution.

See the link below for some more description and a video for the singularity configurations of a 6 axes robot mechanism:

https://www.mecademic.com/en/what-are-singularities-in-a-six-axis-robot-arm

We can also demonstrate motion through a singularity with the toolbox:

We change the Cartesian endpoints of the previous example to

```
>> T1 = transl(0.5, 0.3, 0.44) * troty(pi/2);
>> T2 = transl(0.5, -0.3, 0.44) * troty(pi/2);
```

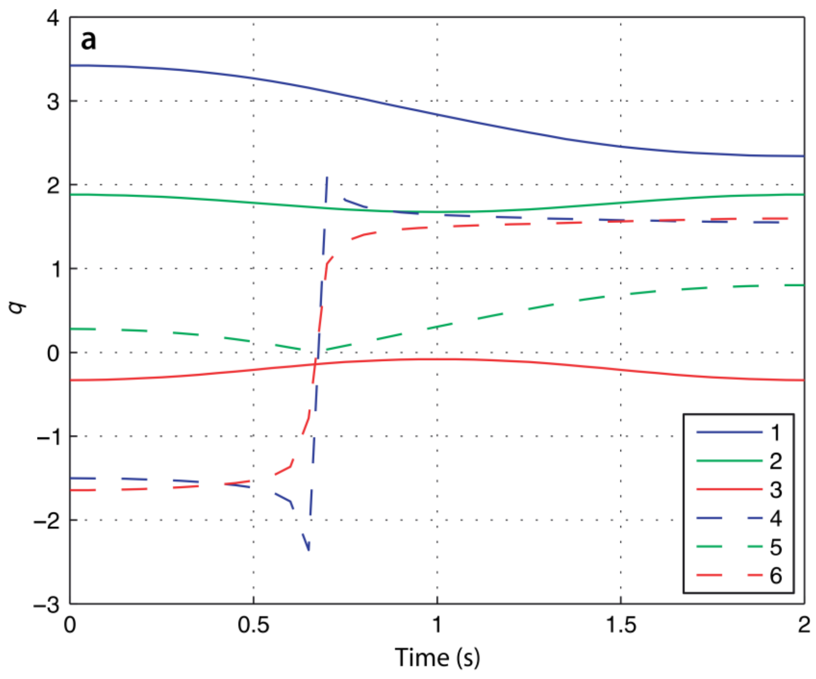
Form the Cartesian-space trajectory and calculate corresponding joint angles with inverse kinematics (see the figure, Corke, p.157)

```
>> Ts = ctraj(T1, T2, length(t));
```

```
>> qc = p560.ikine6s(Ts)
```

At time t ≈ 0.7 s we observe that the rate of change of the wrist joint angles q_4 and q_6 has become very high. The cause is that q_5 has become almost zero which means that the q_4 and q_6 rotational axes are almost aligned, which is a gimbal lock situation or singularity.

(This is actually the problem of closed-form inverse kinematics method, which is generally the preferred approach for calculating the joint angles due to its smaller computational load for the robot control system)



Example problems

1. The task is to demonstrate, with the robotics toolbox, the PUMA 560 robot moving through/nearby a singularity. First you should figure out a **Cartesian space trajectory** for the robot that passes by a singularity. *You are not supposed utilize the same start and end tool frame poses that were used in the example presented in Corke's text book (page 156) and the lecture slides (slide 18).*

For PUMA 560 a typical singularity situation occurs when the 5th joint angle (bending of the wrist) approaches zero, in which case the axes of the 4th and 6th joints become (more and more) parallel, and the robot is about to lose one degree-of-freedom. Note that for calculating values of the robot joints for the intermediate points along the Cartesian space trajectory, you should utilize closed-form inverse kinematics method that has been implemented for PUMA560 by ikine6s (inverse kinematics for 6-axis spherical wrist revolute robot).

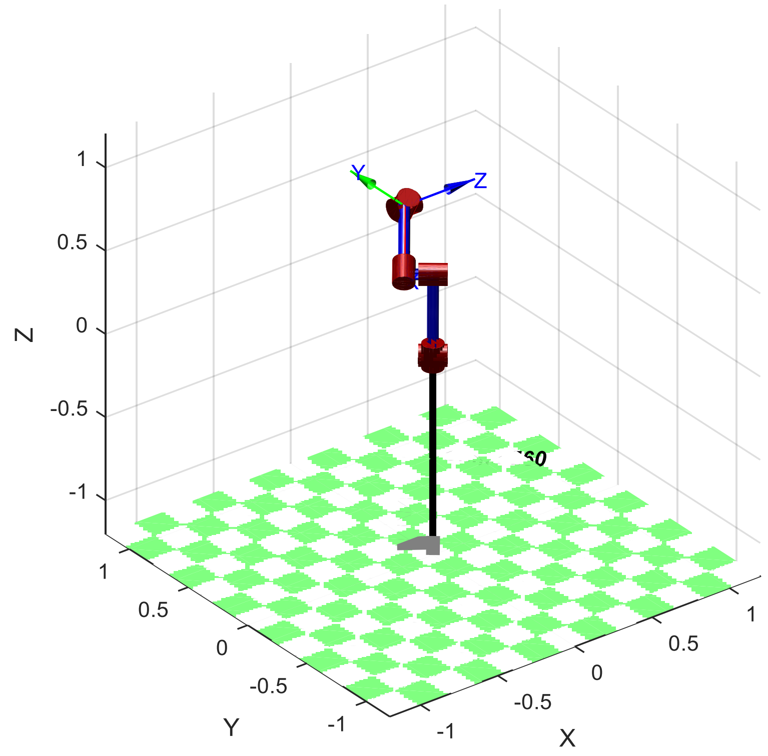
As a solution you should give a plot of the mechanism at the start and end configurations. You should also give a graph of the joint values (six joints) as a function of time (in the graph we should be able to see steep slopes of some of the joint angles when displayed as a function of time, see the book or the lecture slides for a reference). For a comparison, you should also give another graph for the joint values as a function of time when the robot moves from the same start pose to the same end pose by using **joint interpolated motion**-mode.

Solution

There are many possible start and end positions of the tool-frame to demonstrate a motion through/near-by a singularity. A good solution for the problem shows a sharp slope in the plot of the trajectory of one or more joints.

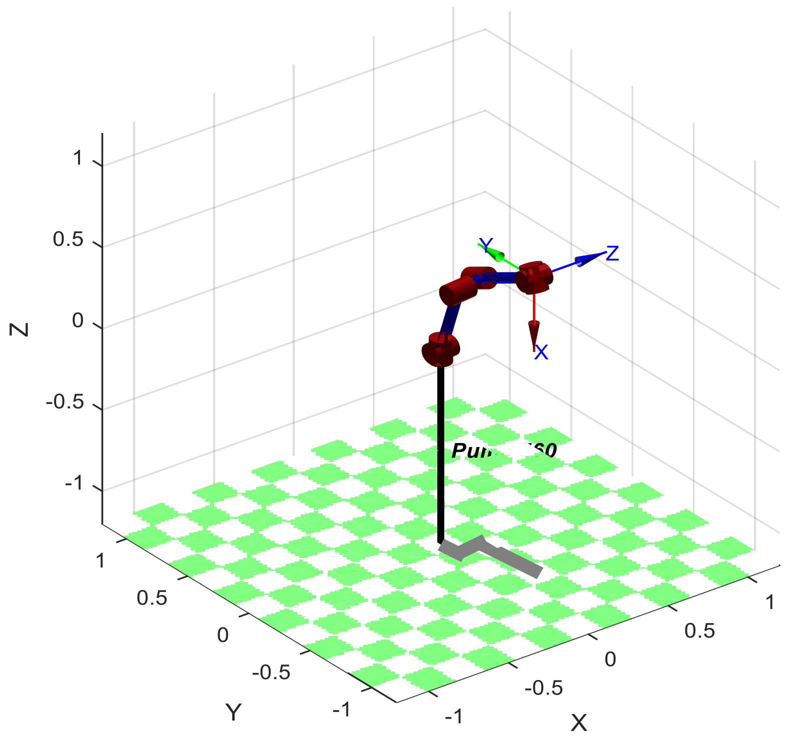
For example:

```
t = [0:0.05:3];
mdl_puma560;
T1 = transl(0.2, 0.5, 0.6) * troty(pi/2);  %Tool frame pose for the start of the motion
T2 = transl(0.2, -0.5, 0.6) * troty(pi/2); %Tool frame pose for the end of the motion
Ts = ctraj(T1,T2,length(t));
qc = p560.ikine6s(Ts);
```



Start pose of the tool-frame T1

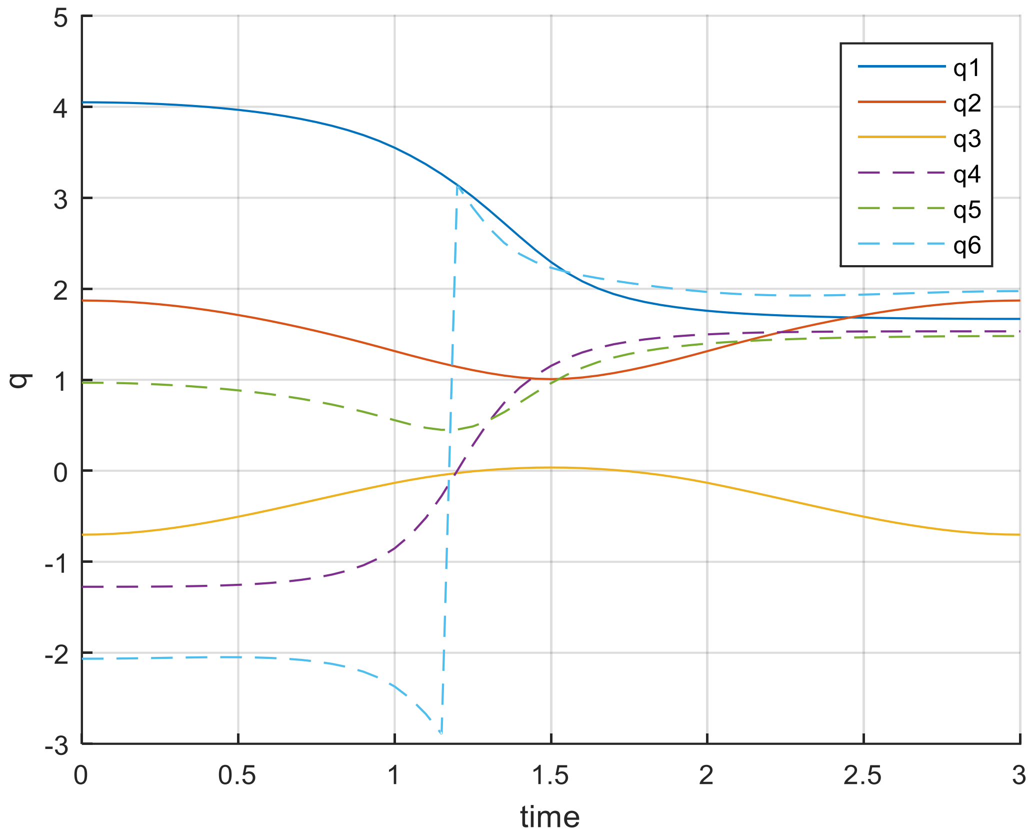
```
p560.plot(qc(1,:));
```



End pose of the tool-frame T2

```
p560.plot(qc(60,:));
```

plot(t,qc); % Plot the trajectory



Recommended reading:

Peter Corke, Robotics, Vision and Control, Fundamental Algorithms in MATLAB, Second Edition, Springer, 2017, pages 206-216.