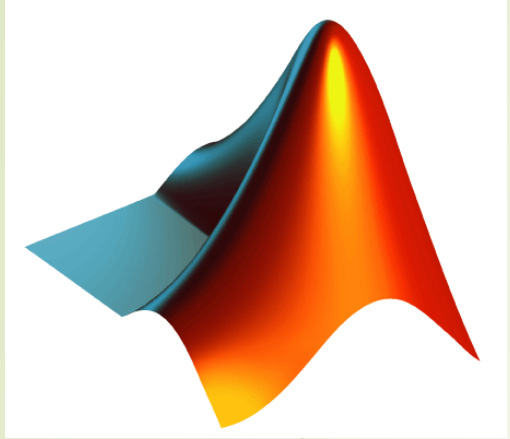
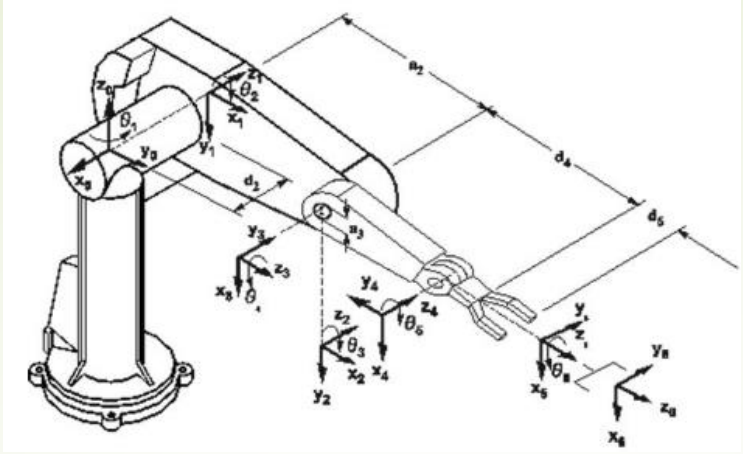
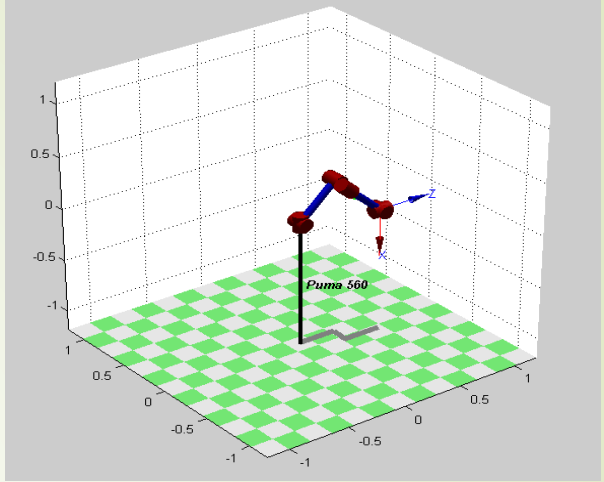






Robotics Toolbox for Matlab

Dr. Nader A. Mansour

naderabdelwahab@gmail.com

Department of Mechanical Engineering

Outlines

- Robotics Toolbox Download and Setup
- Position and Orientation Representation
- Robot Arm Kinematics
- References

Robotics Toolbox Download and Setup

- Robotics Toolbox Download and Setup

https://petercorke.com/wordpress/toolboxes/robotics-toolbox

Robotics Toolbox Download and Setup

Robotics Toolbox Download and Setup

Note: version 10.3 released 19 August 2018

- Download RTB-10.3.1 mltbx format (23.2 MB) in MATLAB toolbox format (.mltbx)
  - From within the MATLAB file browser double click on this file, it will install and configure the paths correctly
- Download RTB-10.3.1 zip format (22.5 MB) as a zip file (.zip)
  - Unpack the archive which will create the directory (folder) rvctools, and within that the directories robot, simulink, and common.
  - Adjust your MATLABPATH to include rvctools
  - Execute the startup file rvctools/startup_rvc.m and this will place the correct directories in your MATLAB path.

Position and Orientation Representation

- Homogenous transformation in 2D using the function (**SE2**)

```
>> T1 = se2(1, 2, 30*pi/180)
T1 =
0.8660          -0.5000          1.0000
0.5000          0.8660           2.0000
0               0                1.0000
```

which represents a translation of (1, 2) and a rotation of 30°.

We can plot this, relative to the world coordinate frame, by

```
>> axis([0 5 0 5]);
>> trplot2(T1, 'frame', '1', 'color', 'b')
```

Similarly:

Position and Orientation Representation

- Homogenous transformation in 2D using the function (**SE2**)

```
>> T2 = se2(2, 1, 0)
T2 =
     1     0     2
     0     1     1
     0     0     1
>> T3 = T1*T2
T3 =
    0.8660   -0.5000    2.2321
    0.5000    0.8660    3.8660
         0         0    1.0000
```

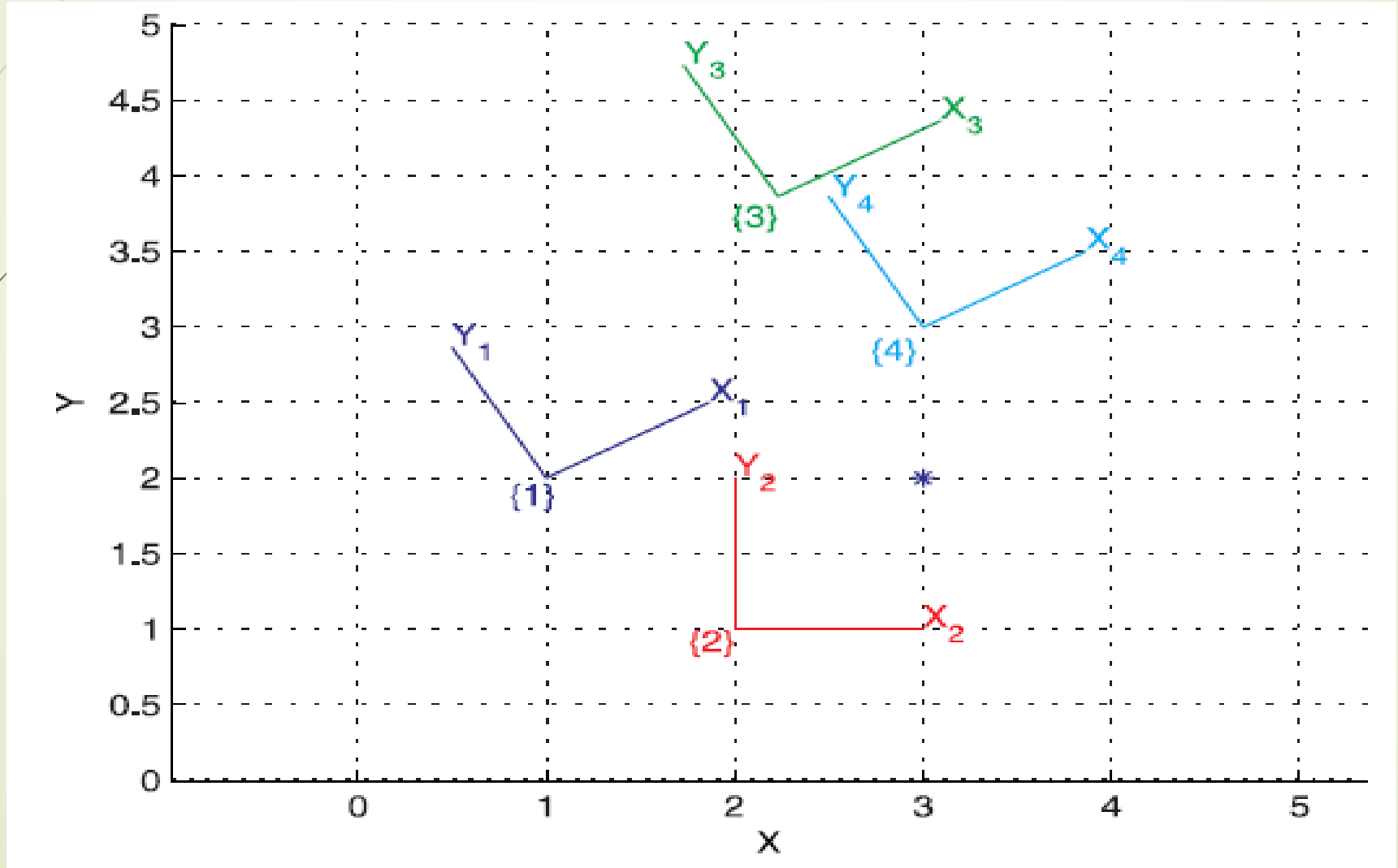
Position and Orientation Representation

- Homogenous transformation in 2D using the function (**SE2**)

```
>> hold on
>> trplot2(T2, 'frame', '2', 'color', 'r');
>> trplot2(T3, 'frame', '3', 'color', 'g');
>> T4 = T2*T1;
>> trplot2(T4, 'frame', '4', 'color', 'c');
```

Position and Orientation Representation

- Homogenous transformation in 2D using the function (**SE2**)



Position and Orientation Representation

- Rotation by angle θ around x axis using the function (**rotx**)

```
>> R = rotx(pi/2)
R =
    1.0000         0         0
         0    0.0000   -1.0000
         0    1.0000    0.0000
```

The corresponding coordinate frame can be displayed graphically

```
>> trplot(R)
```

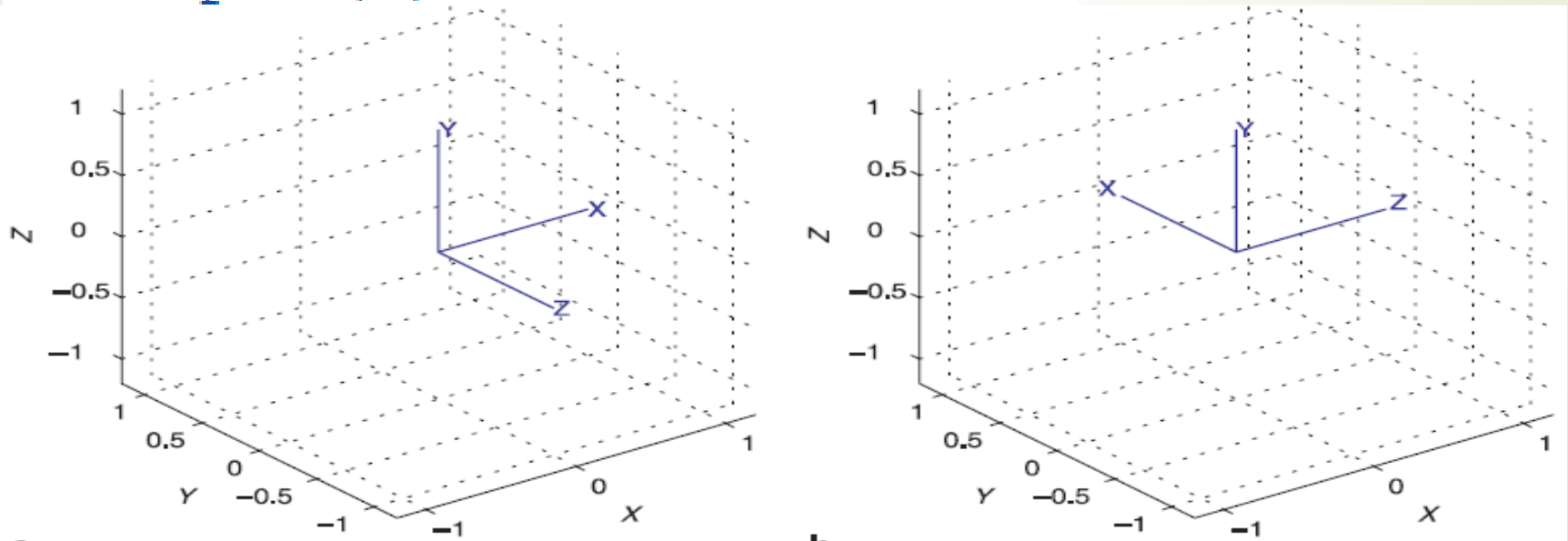
Toolbox function `tranimate` which animates a rotation

```
>> tranimate(R)
```

Position and Orientation Representation

- Rotation by angle θ around x axis using the function (**rotx**)

```
>> rotx(pi/2) * roty(pi/2)
ans =
    0.0000         0    1.0000
    1.0000    0.0000   -0.0000
   -0.0000    1.0000    0.0000
>> trplot(R)
```



Position and Orientation Representation

- Composition of transforms of (translations & rotations)

```
>> T = transl(1, 0, 0) * trotx(pi/2) * transl(0, 1, 0)
T =
    1.0000         0         0    1.0000
         0    0.0000   -1.0000    0.0000
         0    1.0000    0.0000    1.0000
         0         0         0    1.0000
```

The rotation matrix component of `T` is

```
>> t2r(T)
ans =
    1.0000         0         0
         0    0.0000   -1.0000
         0    1.0000    0.0000
```

and the translation component is a vector

```
>> transl(T)'
ans =
    1.0000    0.0000    1.0000
```

Robot Arm Kinematics

- Robots are usually defined based on the links using the object (**Link**) as follows

```
L = Link([0 1.2 0.3 pi/2]);
L = Link('revolute', 'd', 1.2, 'a', 0.3, 'alpha', pi/2);
L = Revolute('d', 1.2, 'a', 0.3, 'alpha', pi/2);
```

- The input parameters for the function Link are basically the DH parameters of each link and they are usually in the order θ, d, a, α.
- If not specified, the link is set to be a revolute by default.

Robot Arm Kinematics

```
>>L.A(pi/2); % returns transformation matrix for (theta=pi/2)
>>L.type  % returns the robot type (revolute or prismatic)
>>L.a        % returns the value a
>>L.d        % returns the value d
>>L.alpha  % returns the value of alpha
```

Robot Arm Kinematics

- For the following two link robot

```
>> L(1) = Link([0 0 1 0]);
>> L(2) = Link([0 0 1 0]);
>> L
L =
 theta=q1, d=0, a=1, alpha=0 (R,stdDH)
 theta=q2, d=0, a=1, alpha=0 (R,stdDH)
```

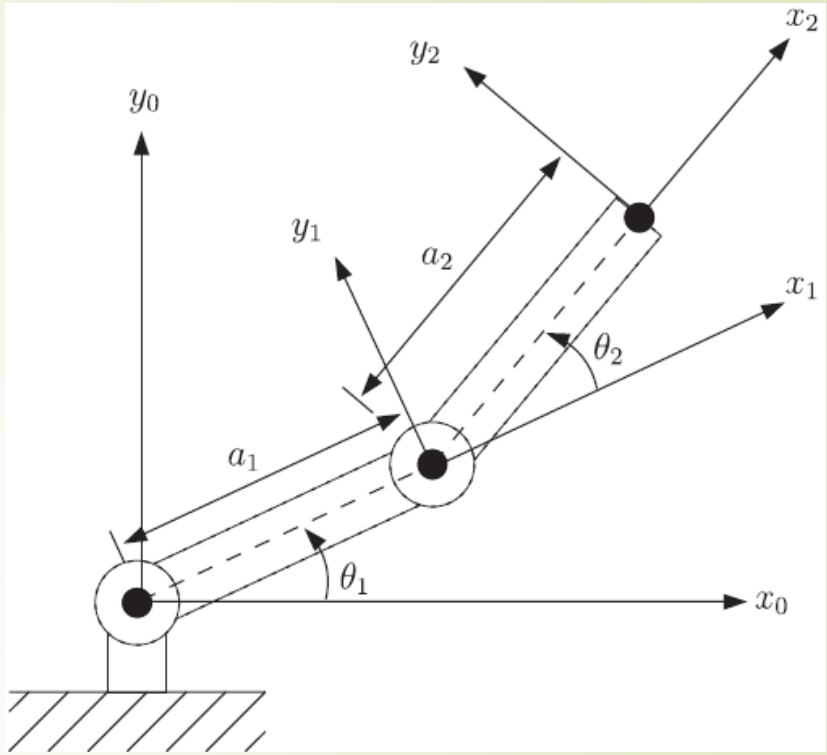
which are passed to the constructor `SerialLink`

```
>> two_link = SerialLink(L, 'name', 'two link');
```

which returns a `SerialLink` object that we can display

```
>> two_link
two_link =
two link (2 axis, RR, stdDH)
+---+-----------+-----------+-----------+-----------+
| j |     theta |         d |         a |     alpha |
+---+-----------+-----------+-----------+-----------+
|  1|         q1|          0|          1|          0|
|  2|         q2|          0|          1|          0|
+---+-----------+-----------+-----------+-----------+

grav =     0  base = 1   0   0   0    tool =   1   0   0   0
           0           0   1   0   0             0   1   0   0
        9.81           0   0   1   0             0   0   1   0
                       0   0   0   1             0   0   0   1
```



Link	θ_i	d_i	a_i	α_i
1	q_1	0	1	0
2	q_2	0	1	0

Robot Arm Kinematics

- Forward kinematics using (**fkine**)

Now we can put the robot arm to work. The forward kinematics are computed using the `fkine` method. For the case where $q_1 = q_2 = 0$

```
>> twolink.fkine([0 0])
ans =
     1     0     0     2
     0     1     0     0
     0     0     1     0
     0     0     0     1
```

the method returns the homogenous transform that represents the pose of the second link coordinate frame of the robot, T_2. For a different configuration the tool pose is

```
>> twolink.fkine([pi/4 -pi/4])
ans =
    1.0000         0         0    1.7071
         0    1.0000         0    0.7071
         0         0    1.0000         0
         0         0         0    1.0000
```

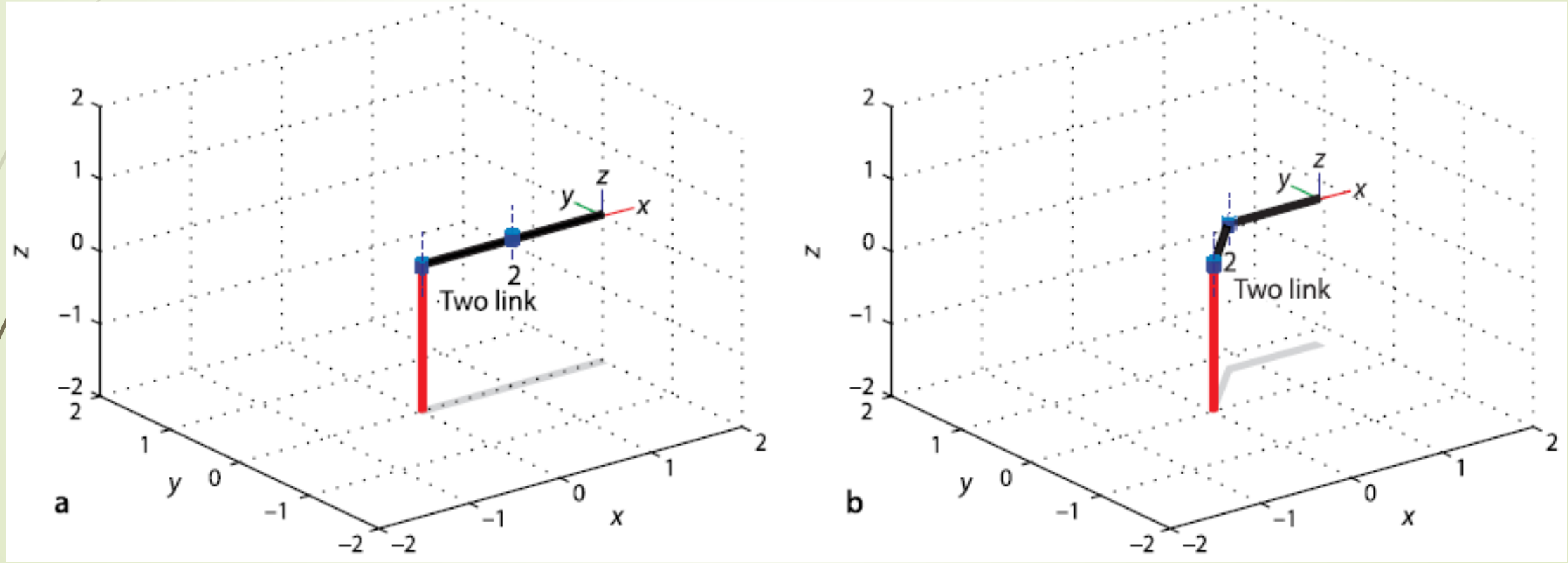
By convention, joint coordinates with the Toolbox are row vectors.

The robot can be visualized graphically

```
>> twolink.plot([0 0])
>> twolink.plot([pi/4 -pi/4])
```

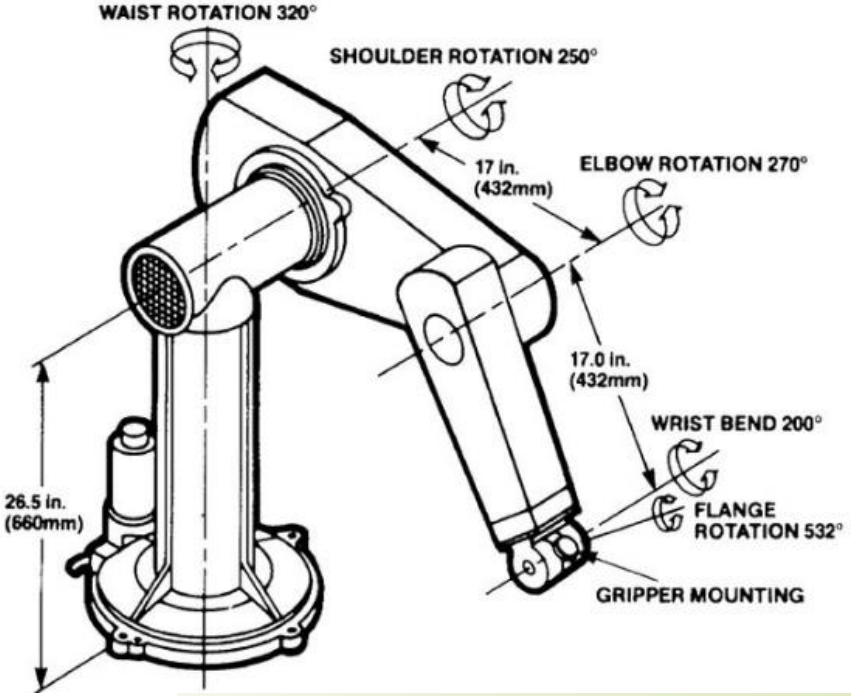
Robot Arm Kinematics

- Forward kinematics using (**fkine**)
  - Matlab Model



Robot Arm Kinematics

- Puma 560 model




```
>> mdl_puma560
```

which creates a `SerialLink` object, `p560`, in the workspace

```
>> p560
p560 =
Puma 560 (6 axis, RRRRRR, stdDH)
 Unimation; viscous friction; params of 8/95;
+---+-----------+-----------+-----------+-----------+
| j |     theta |         d |         a |     alpha |
+---+-----------+-----------+-----------+-----------+
|  1|         q1|          0|          0|      1.571|
|  2|         q2|          0|     0.4318|          0|
|  3|         q3|       0.15|     0.0203|     -1.571|
|  4|         q4|     0.4318|          0|      1.571|
|  5|         q5|          0|          0|     -1.571|
|  6|         q6|          0|          0|          0|
+---+-----------+-----------+-----------+-----------+
```

Robot Arm Kinematics

- Puma 560 model

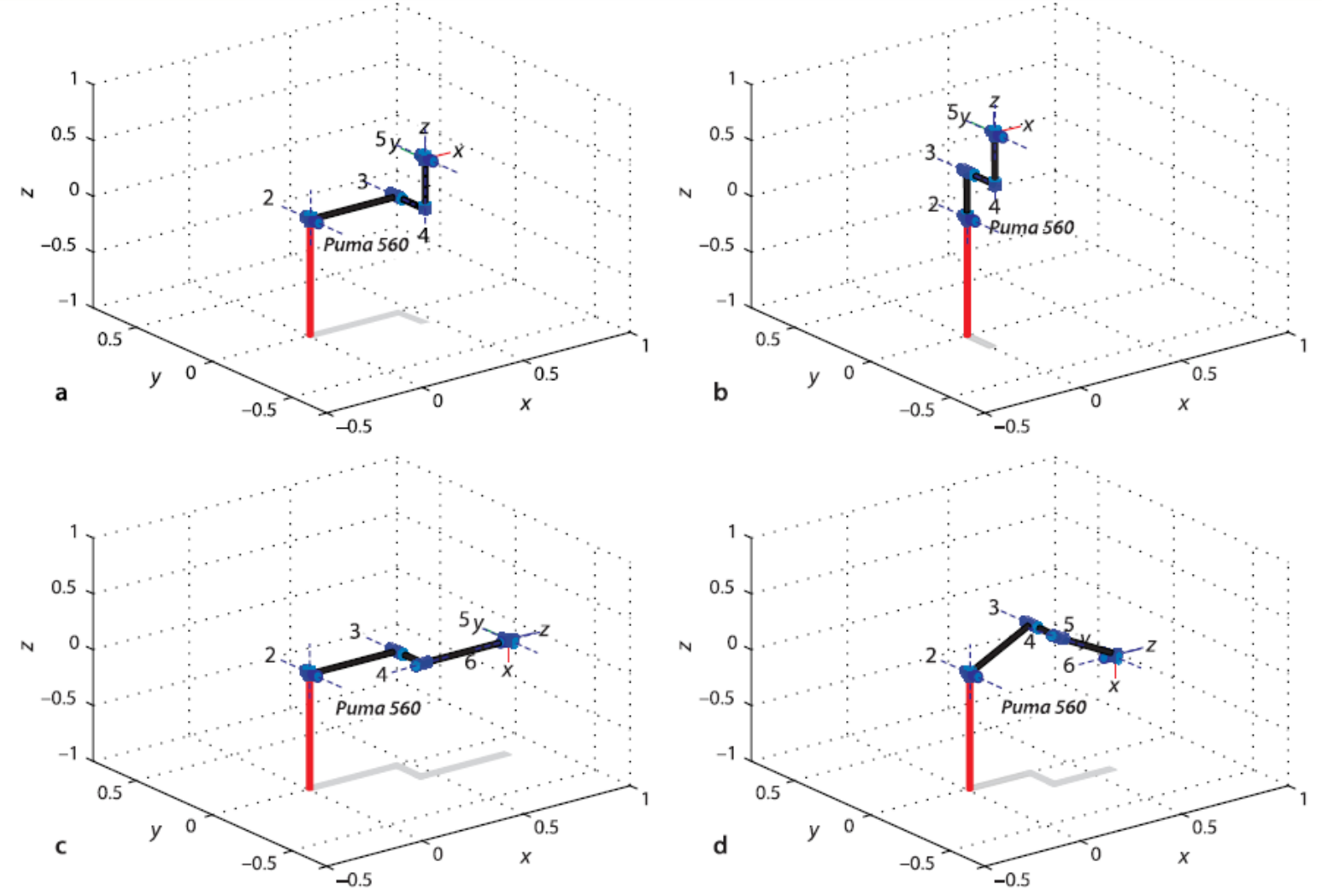
```
grav =     0   base = 1   0   0   0    tool =   1   0   0   0
           0            0   1   0   0             0   1   0   0
        9.81            0   0   1   0             0   0   1   0
                        0   0   0   1             0   0   0   1
```

qz	$(0, 0, 0, 0, 0, 0)$	*zero angle*
qr	$(0, \frac{\pi}{2}, -\frac{\pi}{2}, 0, 0, 0)$	*ready*, the arm is straight and vertical
qs	$(0, 0, -\frac{\pi}{2}, 0, 0, 0)$	*stretch*, the arm is straight and horizontal
qn	$(0, \frac{\pi}{4}, -\pi, 0, \frac{\pi}{4}, 0)$	*nominal*, the arm is in a dextrous working pose

```
>> p560.plot(qz)
```

Robot Arm Kinematics

- Puma 560 model (Forward kinematics)



Robot Arm Kinematics

- Puma 560 model (Inverse kinematics)

At the *nominal* joint coordinates

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

the end-effector pose 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
```

Since the Puma 560 is a 6-axis robot arm with a spherical wrist we use the method `ikine6s` to compute the inverse kinematics using a closed-form solution. The required joint coordinates to achieve the pose `T` are

```
>> qi = p560.ikine6s(T)
qi =
    2.6486   -3.9270    0.0940    2.5326    0.9743    0.3734
```

Robot Arm Kinematics

- Puma 560 model (Inverse kinematics)

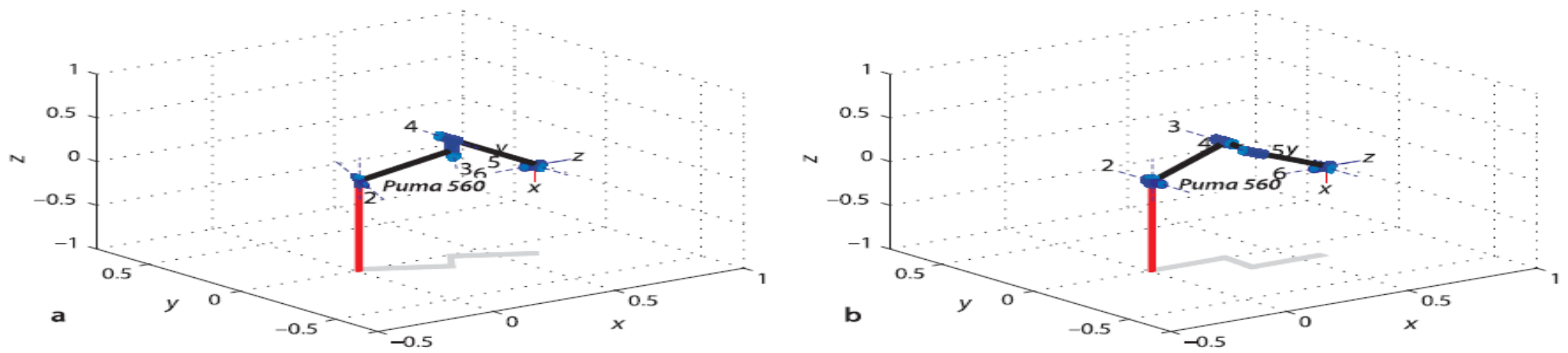
Surprisingly, these are quite different to the joint coordinates we started with. However if we investigate a little further

```
>> p560.fkine(qi)
ans =
   -0.0000    0.0000    1.0000    0.5963
    0.0000    1.0000   -0.0000   -0.1500
   -1.0000    0.0000   -0.0000   -0.0144
         0         0         0    1.0000
```

- These two different sets of joint coordinates result in the same end-effector pose.

Robot Arm Kinematics

- Puma 560 model (Inverse kinematics)



- Two solutions to the inverse kinematic problem, ***left-handed*** and ***right-handed*** solutions. The shadow shows clearly the two different configurations

Robot Arm Kinematics

- Puma 560 model (Inverse kinematics)

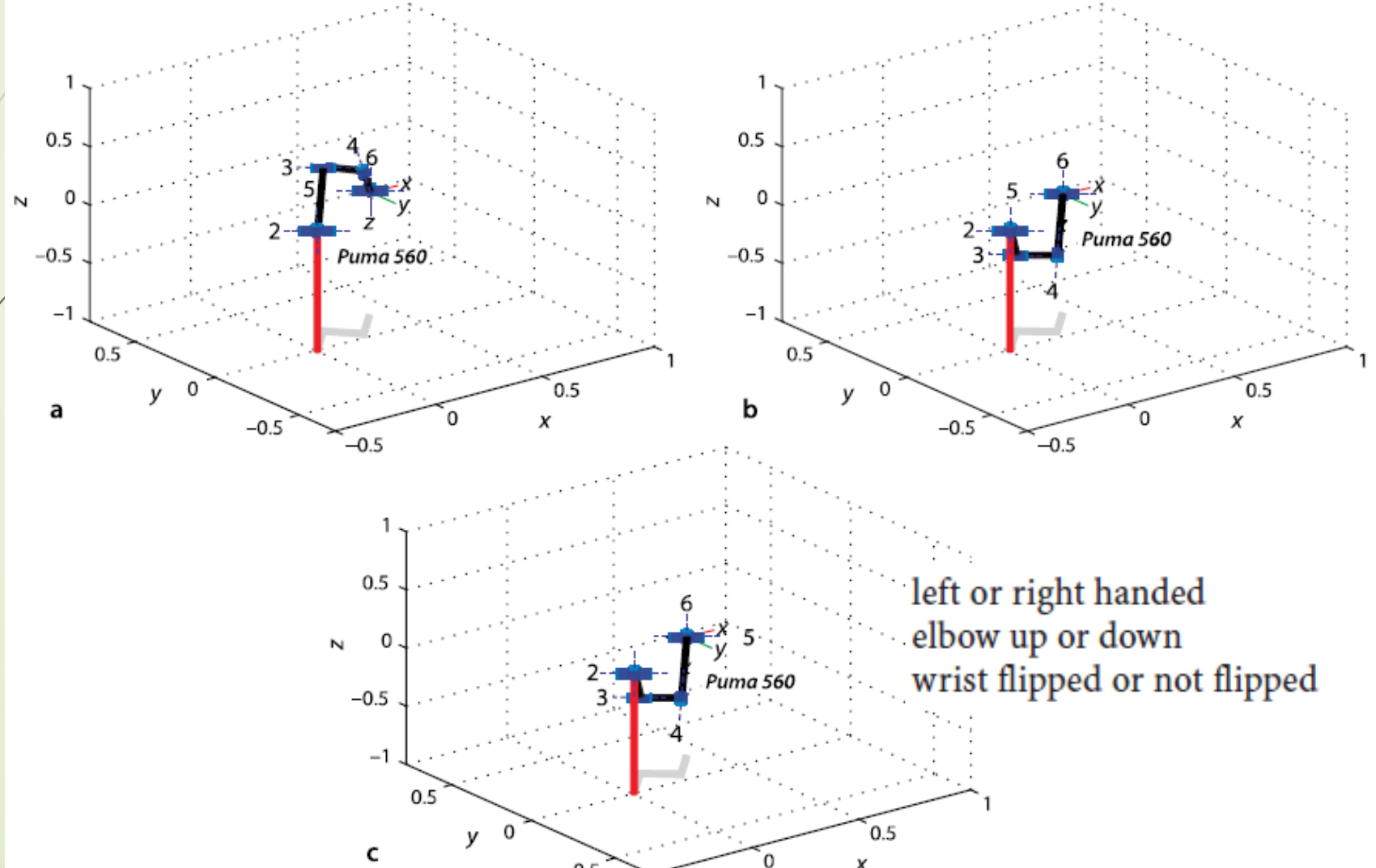
We can *force* the right-handed solution

```
>> qi = p560.ikine6s(T, 'ru')
qi =
   -0.0000    0.7854    3.1416    0.0000    0.7854   -0.0000
```

- **Ikine6s** functions is used for closed form solution
- **Ikine6s** can’t be used with robot of less than 6 dof
- Sometimes, inverse kinematics fails if the point is unreachable.
- Make sure the pose of interest is already within the workspace.

Robot Arm Kinematics

- Inverse kinematics (Different possible solutions)



Robot Arm Kinematics

- Inverse kinematics (Numerical Solution & under actuated)
  - Using **ikine** function
  - Depends on optimization techniques to find the inverse kinematics solution
  - Ikine(T, qi, ‘mask’, [1 1 1 0 0 0]);

Where T is the desired pose, qi is initial vector required for the optimization problem, number of 1’s of the mask refers to the number of dof of the robot actuated links.

- Matlab Demo

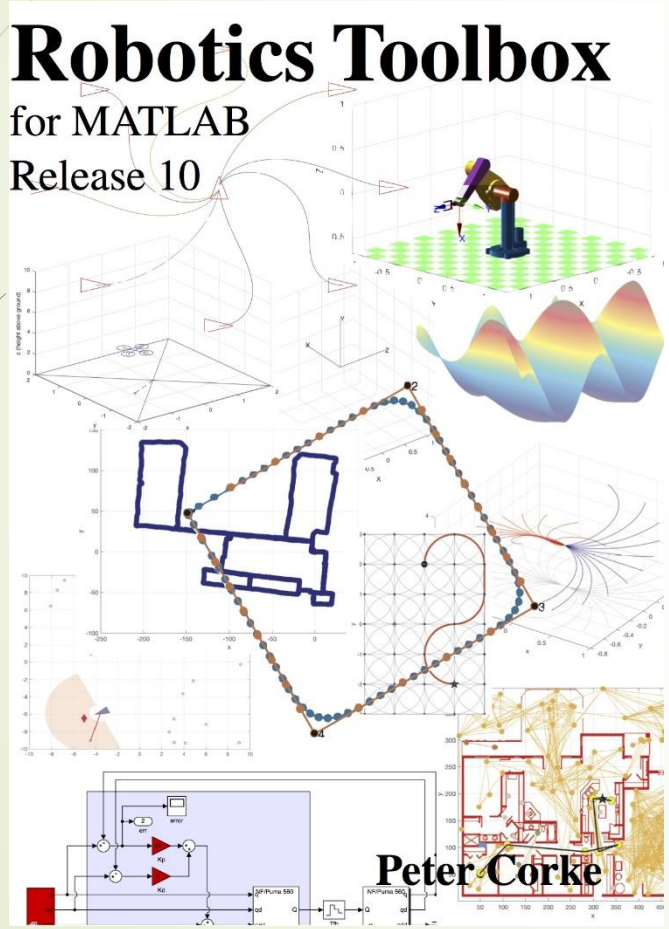
Robot Arm Kinematics

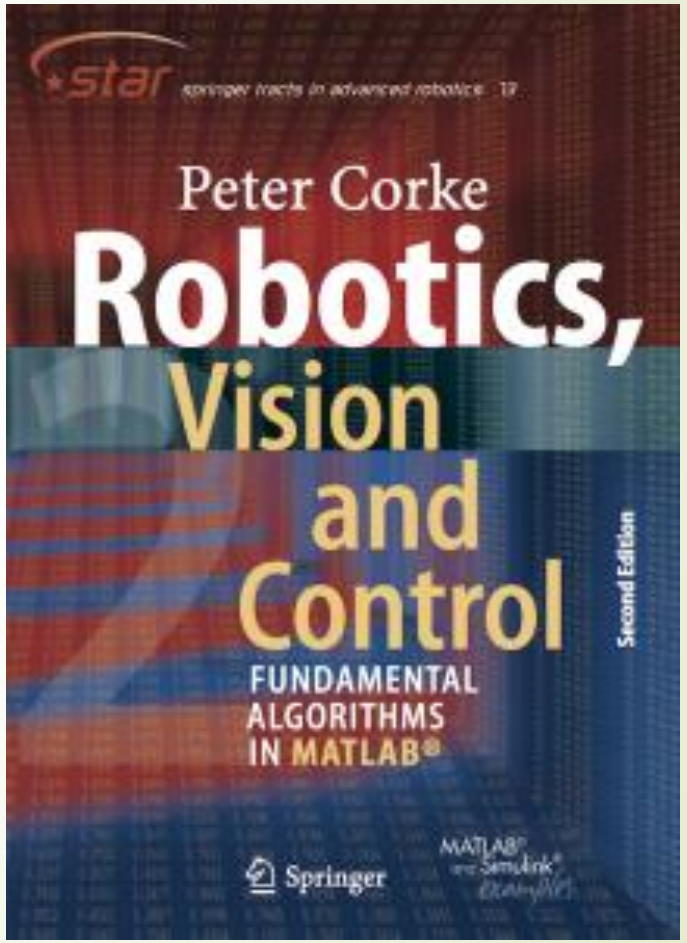
- Trajectory planning (Joint Space)
  - Using **jtraj** function
  - A joint-space trajectory is formed by smoothly interpolating between two joints' configurations q1 and q2.
  - Matlab Demo

Robot Arm Kinematics

- Trajectory planning (cartesian Space)
  - Using **ctraj** function
  - For many applications we require straight-line or obstacle avoidance motion in Cartesian space which is known as Cartesian motion
  - Depends mainly on interpolating between two poses of the end effector in the cartesian space in the start and end points T1 and T2.
  - Matlab Demo

References


Robotics Toolbox
for MATLAB
Release 10
Peter Corke


star
springer tracts in advanced robotics
Peter Corke
Robotics,
Vision
and
Control
FUNDAMENTAL
ALGORITHMS
IN MATLAB®
Second Edition
Springer
MATLAB®
and Simulink®