Course No. M4609301
Introduction to Robotics
機器人學概論

Matlab Chapter 7
Robot Arm Kinematics

Shuhao Liang
Graduate Institute of Intelligent Manufacturing Technology
National Taiwan University of Science and Technology
2024

Part III Arm-Type Robots

Chapter 7 Robot Arm Kinematics

Chapter 8 Velocity Relationships

Chapter 9 Dynamics and Control

Part

III

Arm-Type Robots

Arm-type robots or robot manipulators are a very common and familiar type of robot. We are used to seeing pictures or video of them at work in factories doing jobs such as assembly, welding and handling tasks, or even in operating rooms doing surgery.

The first robot manipulators started work nearly 60 years ago and have been enormously successful in practice – many millions of robot manipulators are working in the world today. Many products we buy have been assembled, packed or handled by a robot.

Different types of robot manipulator

Fig. III.1.
a A 6DOF serial-link manipulator. General purpose industrial manipulator (source: ABB).
b SCARA robot which has 4DOF, typically used for electronic assembly (photo of Adept Cobra s600 SCARA robot courtesy of Adept Technology, Inc.).
c A gantry robot; the arm moves along an overhead rail (image courtesy of Güdel AG Switzerland | Mario Rothenbühler | www.gudel.com). **d** A parallellink manipulator, the end-effector is driven by 6 parallel links (source: ABB)

End-effectors

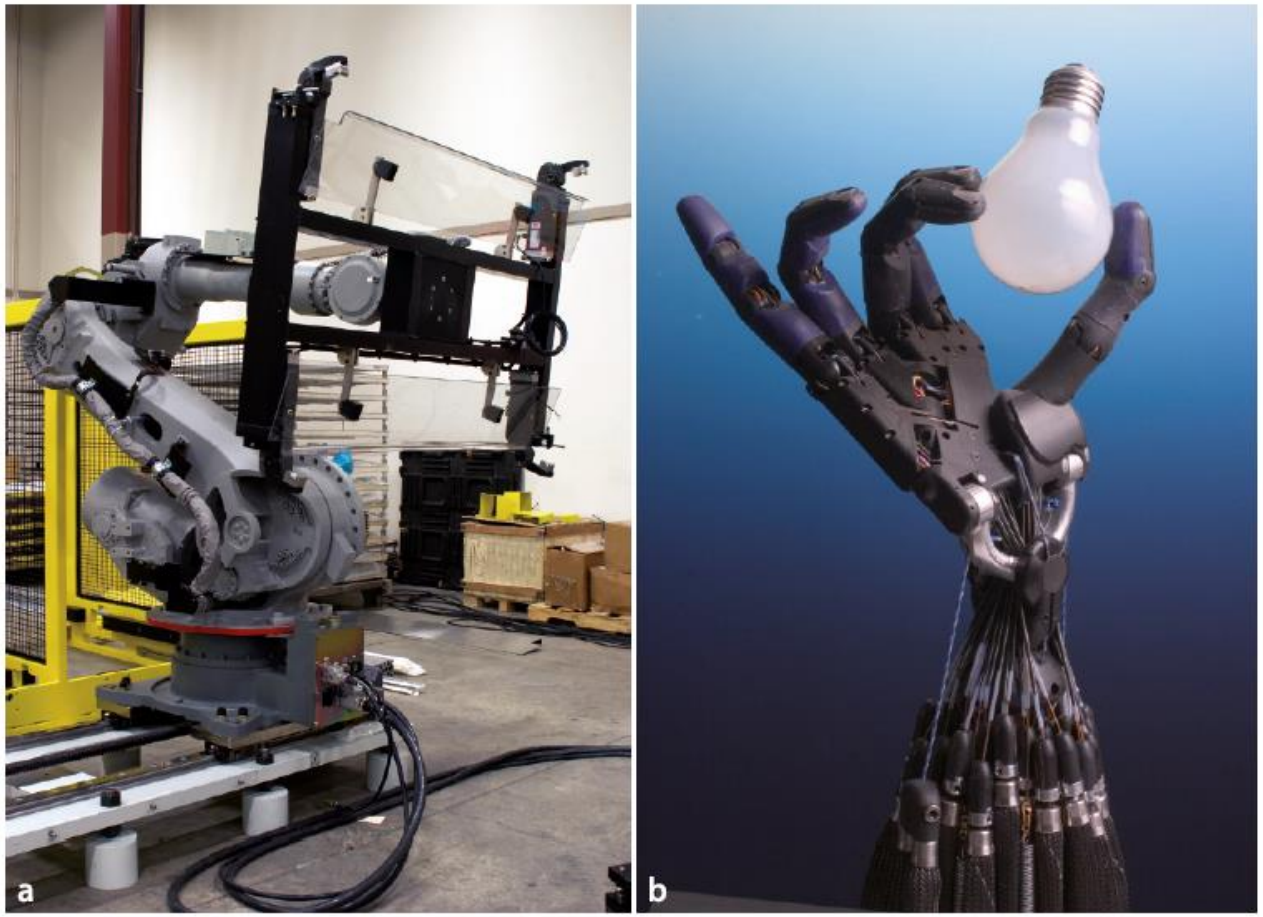


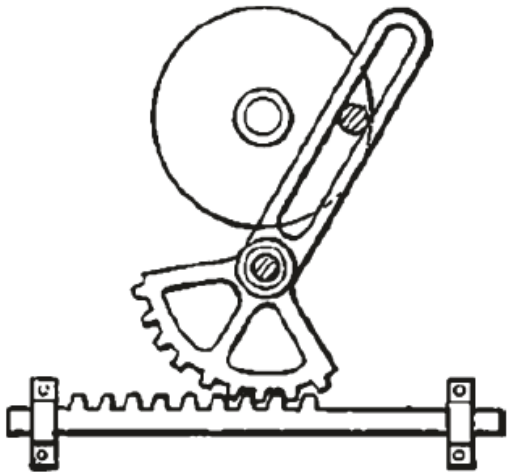
Fig. III.2.
Robot end-effectors. **a** A vacuum gripper holds a sheet of glass. **b** A human-like robotic hand (© Shadow Robot Company 2008)

Chapter

7 Robot Arm Kinematics

Take to kinematics. It will repay you.
It is more fecund than geometry; it adds a fourth dimension to space.
Chebyshev to Sylvester 1873

Kinematics is the branch of mechanics that studies the motion of a body, or a system of bodies, without considering its mass or the forces acting on it.



A robot arm, more formally a serial-link manipulator, comprises a chain of rigid links and joints. Each joint has one degree of freedom, either translational (a sliding or prismatic joint) or rotational (a revolute joint). Motion of the joint changes the relative pose of the links that it connects. One end of the chain, the base, is generally fi xed and the other end is free to move in space and holds the tool or end-effector that does the useful work.

Figure 7.1 shows two modern arm-type robots that have six and seven joints respectively.

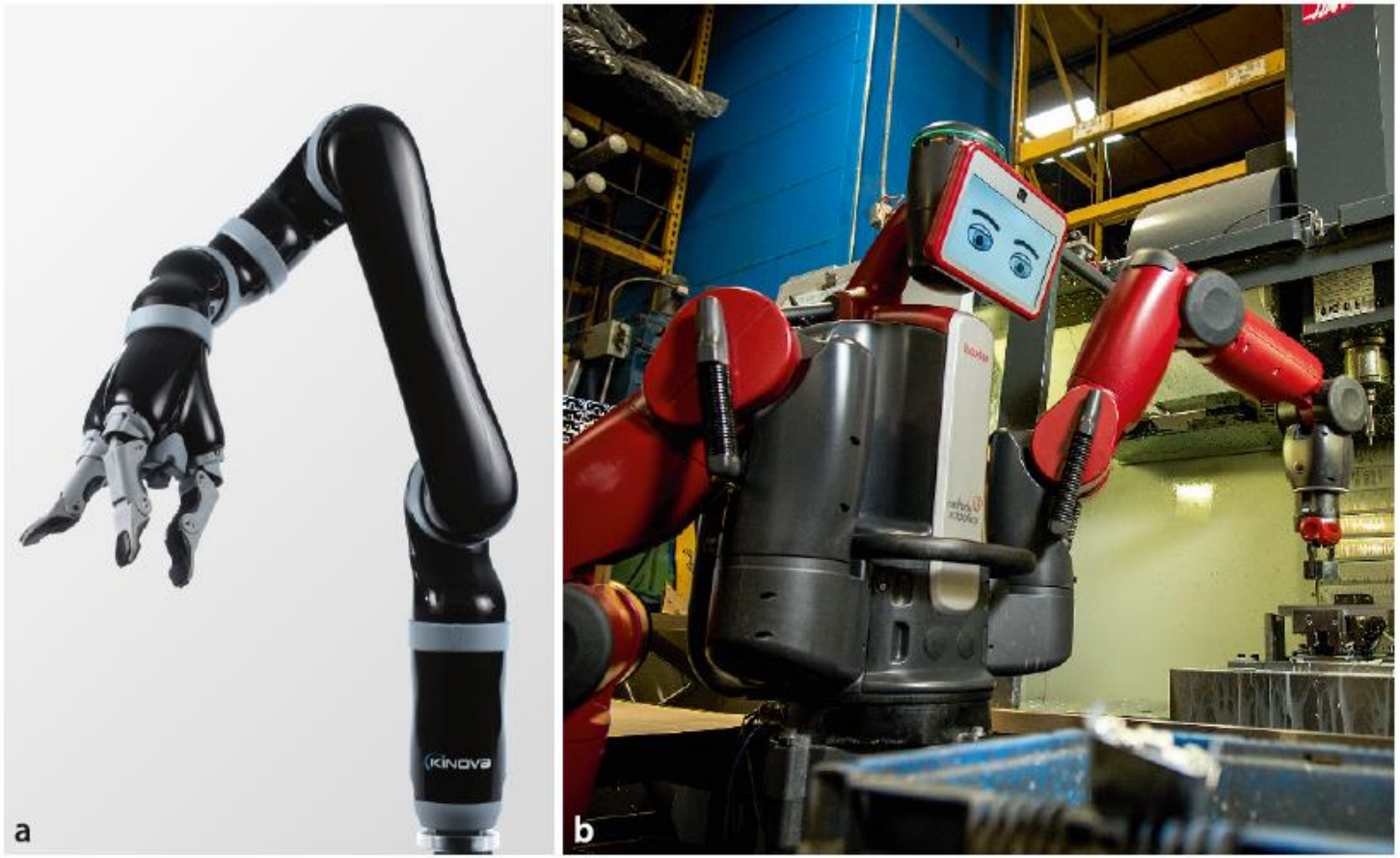


Fig. 7.1.
a Mico 6-joint robot with 3-fi ngered hand (courtesy of Kinova Robotics). b Baxter 2-armed robotic coworker, each arm has 7 joints (courtesy of Rethink Robotics)

Pose, position of each joint, smooth paths

Clearly the pose of the end-effector will be a complex function of the state of each joint and **Sect. 7.1** describes how to compute the pose of the end-effector. **Section 7.2** discusses the inverse problem, how to compute the position of each joint given the end-effector pose. **Section 7.3** describes methods for generating smooth paths for the end-effector. The remainder of the chapter covers advanced topics and two complex applications: writing on a plane surface and a fourlegged walking robot whose legs are simple robotic arms.

Outline

7.1 Forward Kinematics
7.2 Inverse Kinematics
7.3 Trajectories
7.4 Advanced Topics
7.5 Applications

7.1.1 2-Dimensional (Planar) Robotic Arms

Consider the simple robot arm shown in Fig. 7.2a which has a single rotational joint. We can describe the pose of its end-effector – frame {*E*} – by a sequence of relative poses: a rotation about the joint axis and then a translation by a_1 along the rotated *x*-axis

$$\xi_E(q) = \mathcal{R}_z(q_1) \oplus \mathcal{T}_x(a_1)$$

The Toolbox allows us to express this, for the case $a_1 = 1$, by

```
>> import ETS2.*
>> a1 = 1;
>> E = Rz('q1') * Tx(a1)
```

which is a sequence of `ETS2` class objects. The argument to `Rz` is a string which indicates that its parameter is a joint variable whereas the argument to `Tx` is a constant numeric robot dimension.

The forward kinematics for a *particular* value of $q_1 = 30$ deg

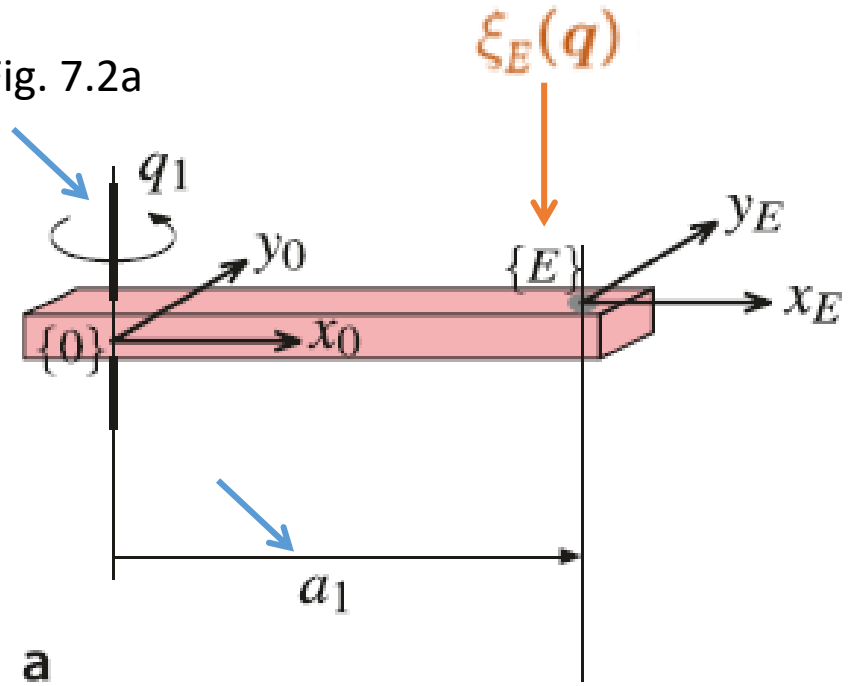
```
>> E.fkine( 30, 'deg')
ans =
    0.8660   -0.5000    0.866
    0.5000    0.8660      0.5
         0         0        1
```

is an SE(2) homogeneous transformation matrix representing the pose of the end-effector – coordinate frame {*E*}.

An easy and intuitive way to understand how this simple robot behaves is interactively

```
>> E.teach
```

Fig. 7.2a

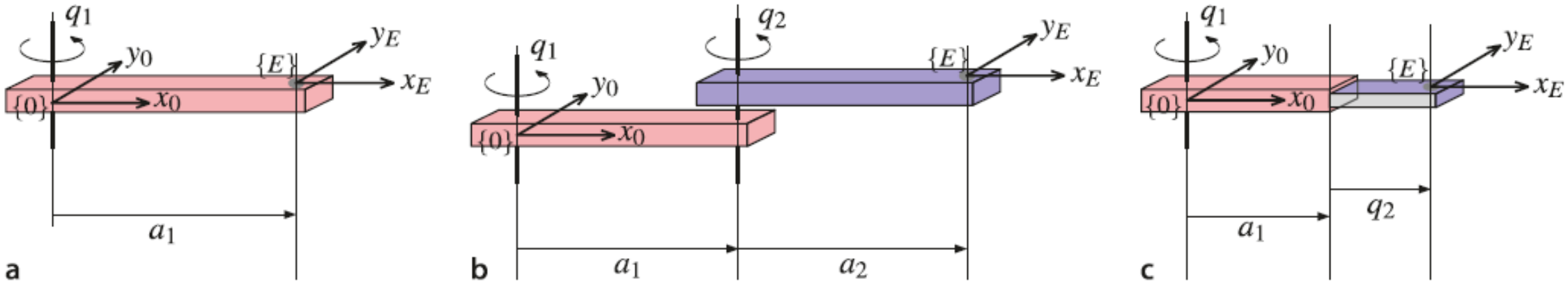


We use the symbols $\mathcal{R}, \mathcal{T}_x, \mathcal{T}_y$ to denote relative poses in SE(2) that are respectively pure rotation and pure translation in the *x*- and *y*-directions.

planar robotic arms.

Fig. 7.2. Some simple planar robotic arms. a Planar arm with one rotational joint; b planar arm with two rotational joints; c planar arm with two joints: one rotational and one prismatic. The base {0} and end-effector {E} coordinate frames are shown. The joint variables, angle or prismatic extension, are generalized coordinates and denoted by q_j

We use the symbols $\mathcal{R}, \mathcal{T}_x, \mathcal{T}_y$ to denote relative poses in SE(2) that are respectively pure rotation and pure translation in the x- and y-directions.



Planar arm with one rotational joint

planar arm with two rotational joints

planar arm with two joints: one rotational and one prismatic.

Toolbox depiction of 1-joint planar robot

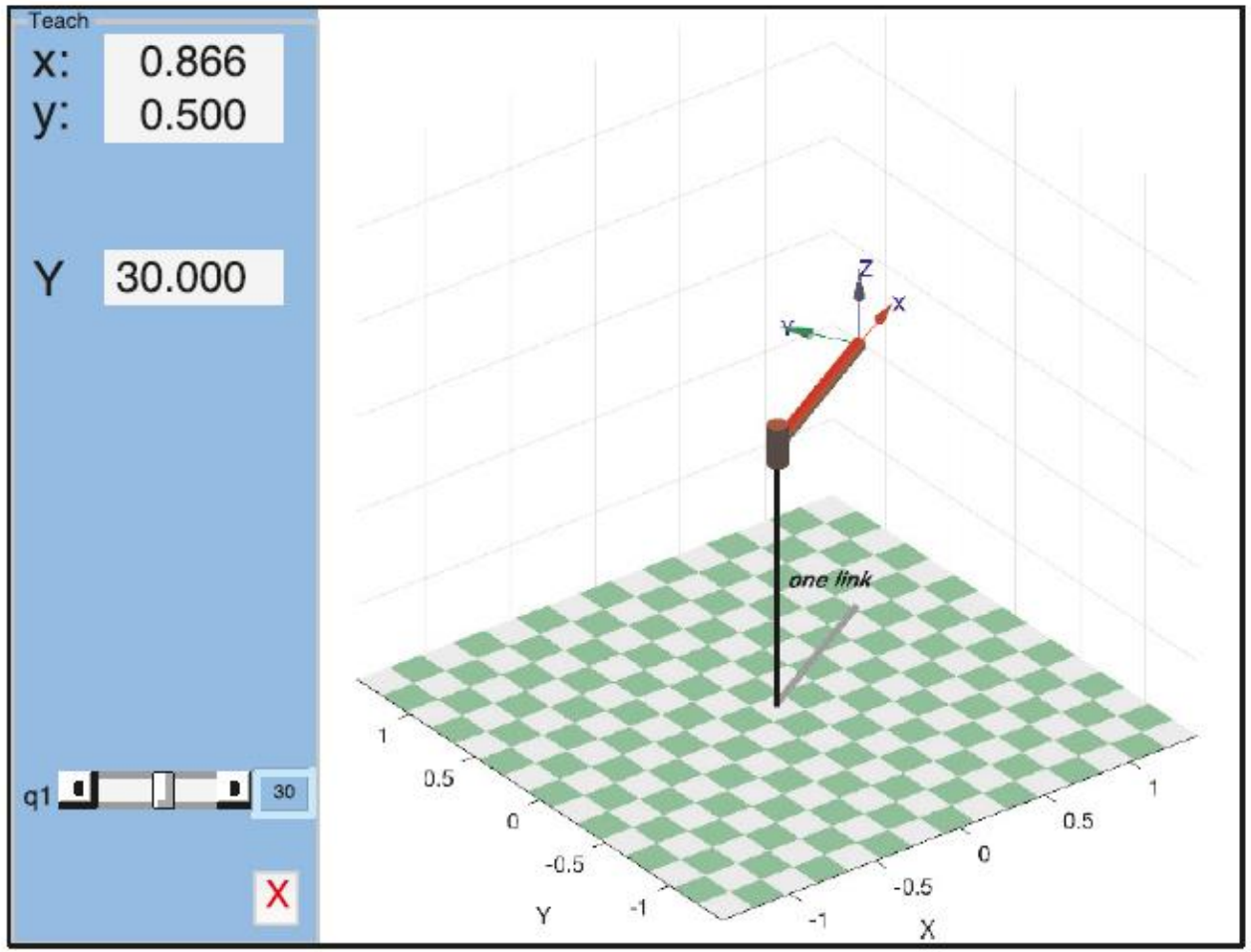


Fig. 7.3.
Toolbox depiction of 1-joint planar robot using the `teach` method. The *blue panel* contains the joint angle slider and displays the position and orientation (yaw angle) of the end-effector (in degrees)

Prismatic joints .

Prismatic joints. Robot joints are commonly revolute (rotational) but can also be prismatic (linear, sliding, telescopic, etc.). The SCARA robot of Fig. III.1b has a prismatic third joint while the gantry robot of Fig. III.1c has three prismatic joints for motion in the x-, y- and z-directions.

The Stanford arm shown here has a prismatic third joint. It was developed at the Stanford AI Lab in 1972 by robotics pioneer Victor Scheinman who went on to design the PUMA robot arms. This type of arm supported a lot of important early research work in robotics and one can be seen in the Smithsonian Museum of American History, Washington DC. (Photo courtesy Oussama Khatib)

7.1.2 3-Dimensional Robotic Arms

Random, casual

Truly useful robots have a task space $\mathcal{T} \subset \mathbf{SE}(3)$ enabling arbitrary position and orientation of the end-effector. This requires a robot with a configuration space dim $\mathcal{C} \geq \dim \mathcal{T}$ which can be achieved by a robot with six or more joints. In this section we will use the Puma 560 as an exemplar of the class of all-revolute six-axis robot manipulators with $\mathcal{C} \subset (\mathbb{S}^1)^6$.

We can extend the technique from the previous section for a robot like the Puma 560 whose dimensions are shown in Fig. 7.4. Starting with the world frame {0} we move up, rotate about the waist axis (q_1), rotate about the shoulder axis (q_2), move to the left, move up and so on. As we go, we write down the transform expression

$$\xi_E = \mathcal{T}_z(L_1) \oplus \mathcal{R}_z(q_1) \oplus \mathcal{R}_y(q_2) \oplus \mathcal{T}_y(L_2) \oplus \mathcal{T}_z(L_3) \oplus \mathcal{R}_y(q_3) \oplus \mathcal{T}_x(L_4) \oplus \mathcal{T}_y(L_5) \oplus \mathcal{T}_z(L_6) \oplus \underbrace{\mathcal{R}_z(q_4) \oplus \mathcal{R}_y(q_5) \oplus \mathcal{R}_z(q_6)}_{\text{wrist}}$$

SE means "**Special Euclidean**" group, e.g. **SE(3)** , which is shorthand for "the special Euclidean group of rigid body displacements in three dimensions".

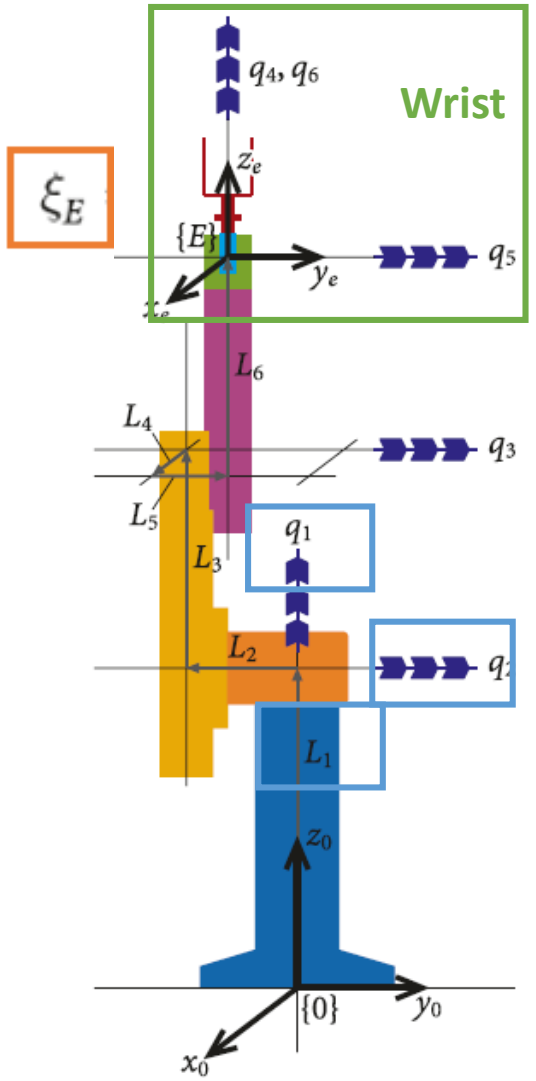


Fig. 7.4. Puma robot in the zero-joint-angle configuration showing dimensions and joint axes (indicated by *blue triple arrows*) (after Corke 2007)

3-dimensional version of the Toolbox class

The marked term represents the kinematics of the robot's wrist and should be familiar to us as a *ZYZ* Euler angle sequence from Sect. 2.2.1.2 – it provides an arbitrary orientation but is subject to a singularity when the middle angle $q_5 = 0$.

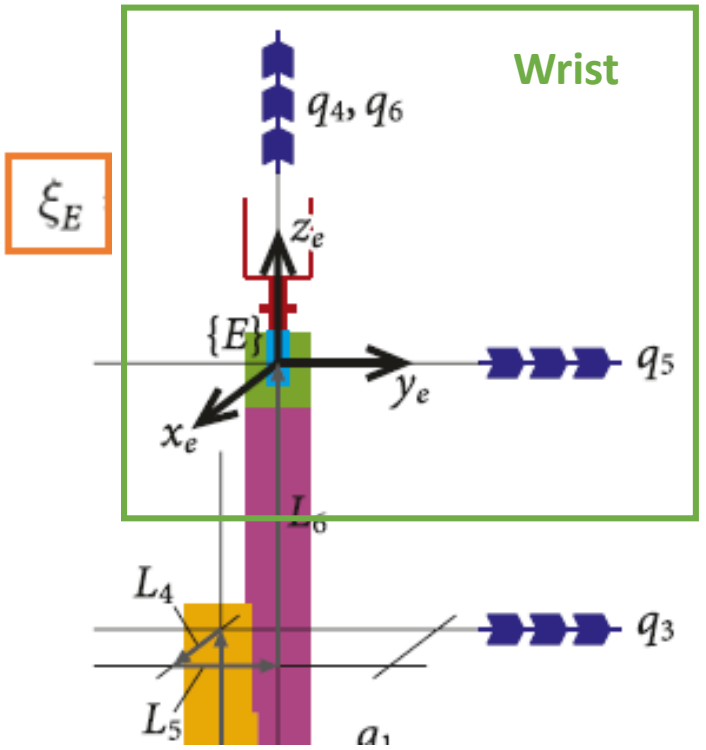
We can represent this using the 3-dimensional version of the Toolbox class we used previously

```
>> import ETS3.*
>> L1 = 0; L2 = -0.2337; L3 = 0.4318; L4 = 0.0203; L5 = 0.0837; L6 = 0.4318;
>> E3 = Tz(L1) * Rz('q1') * Ry('q2') * Ty(L2) * Tz(L3) * Ry('q3') ↵
    * Tx(L4) * Ty(L5) * Tz(L6) * Rz('q4') * Ry('q5')  * Rz('q6');
```

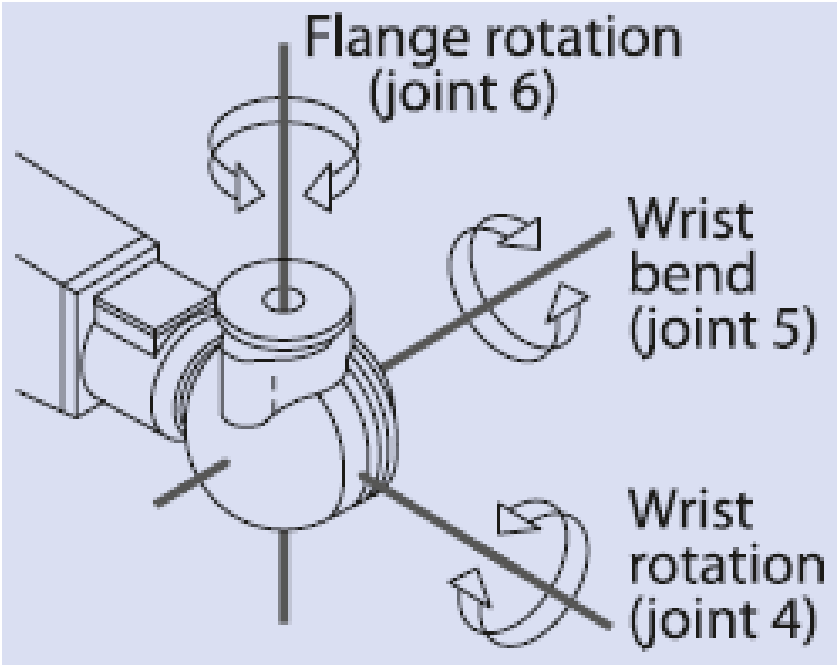
We can use the interactive teach facility or compute the forward kinematics

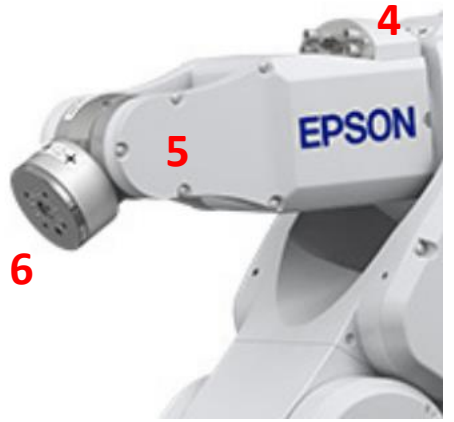
```
>> E3.fkine([0 0 0 0 0 0])
ans =
         1         0         0    0.0203
         0         1         0     -0.15
         0         0         1    0.8636
         0         0         0         1
```

While this notation is intuitive it does becomes cumbersome as the number of robot joints increases. A number of approaches have been developed to more concisely describe a serial-link robotic arm: Denavit-Hartenberg notation and product of exponentials.



Spherical wrists





Spherical wrists are a key component of almost all modern arm-type robots. They have three axes of rotation that are orthogonal and intersect at a common point. This is a gimbal-like mechanism, and as discussed in Sect. 2.2.1.3 and will have a singularity.

The robot end-effector pose, position and an orientation, is defined at the center of the wrist. Since the wrist axes intersect at a common point they cause zero translation, therefore the position of the end-effector is a function only of the first three joints. This is a critical simplification that makes it possible to find closed-form inverse kinematic solutions for 6-axis industrial robots. An arbitrary end-effector orientation is achieved independently by means of the three wrist joints.

(Review 2.2)

2.2 Working in Three Dimensions (3D)

The 3-dimensional case is an extension of the 2-dimensional case discussed in the previous section. We add an extra coordinate axis, typically denoted by z, that is orthogonal to both the x- and y-axes. The direction of the z-axis obeys the *right-hand rule* and forms a *right-handed coordinate frame*. Unit vectors parallel to the axes are denoted $\hat{\boldsymbol{x}}$, $\hat{\boldsymbol{y}}$ and $\hat{\boldsymbol{z}}$ such that

In all these identities, the symbols from left to right (across the equals sign) are a cyclic rotation of the sequence *xyz*.

$$\hat{\boldsymbol{z}} = \hat{\boldsymbol{x}} \times \hat{\boldsymbol{y}}, \quad \hat{\boldsymbol{x}} = \hat{\boldsymbol{y}} \times \hat{\boldsymbol{z}}; \quad \hat{\boldsymbol{y}} = \hat{\boldsymbol{z}} \times \hat{\boldsymbol{x}} \tag{2.12}$$

A point P is represented by its x-, y- and z-coordinates (x, y, z) or as a bound vector

$$\boldsymbol{p} = x\hat{\boldsymbol{x}} + y\hat{\boldsymbol{y}} + z\hat{\boldsymbol{z}}$$

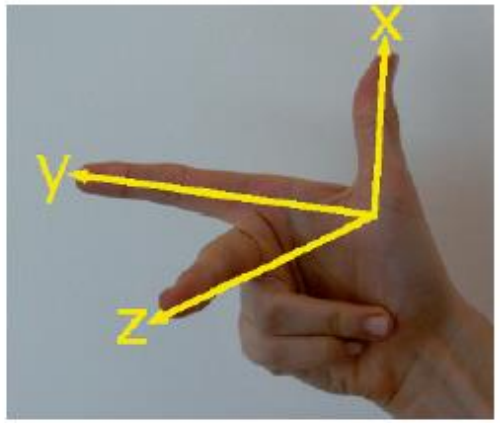


Figure 2.10 shows a red coordinate frame $\{B\}$ that we wish to describe with respect to the blue reference frame $\{A\}$. We can see clearly that the origin of $\{B\}$ has been

Right-hand rule. A right-handed coordinate frame is defined by the first three fingers of your right hand which indicate the relative directions of the x-, y- and z-axes respectively.

(Review 2.2.1.2)

2.2.1.2 Three-Angle Representations

Euler's rotation theorem requires successive rotation about three axes such that no two successive rotations are about the same axis. There are two classes of rotation sequence: Eulerian and Cardanian, named after Euler and Cardano respectively.

The Eulerian type involves repetition, but not successive, of rotations about one particular axis: XYX, XZX, YXY, YZY, ZXZ, or ZYZ. The Cardanian type is characterized by rotations about all three axes: XYZ, XZY, YZX, YXZ, ZXY, or ZYX.

> It is common practice to refer to all 3-angle representations as Euler angles but this is underspecified since there are twelve different types to choose from. The particular angle sequence is often a convention within a particular technological field.

The ZYZ sequence

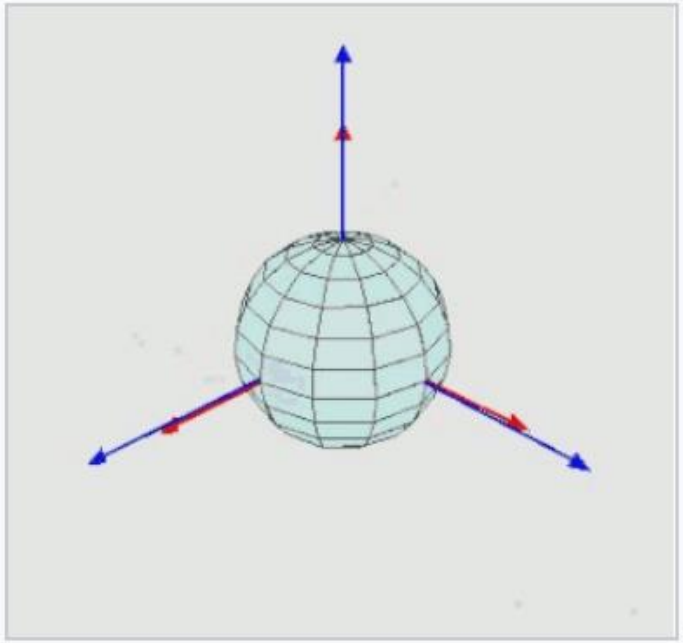
$$\boldsymbol{R} = \boldsymbol{R}_z(\phi)\boldsymbol{R}_y(\theta)\boldsymbol{R}_z(\psi) \tag{2.14}$$

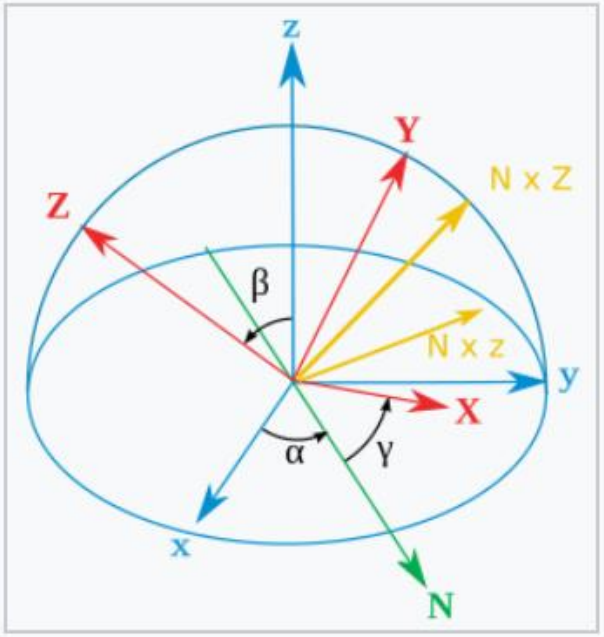
is commonly used in aeronautics and mechanical dynamics, and is used in the Toolbox. The Euler angles are the 3-vector $\boldsymbol{\Gamma} = (\phi, \theta, \psi)$.

Euler angles

https://en.wikipedia.org/wiki/Euler_angles

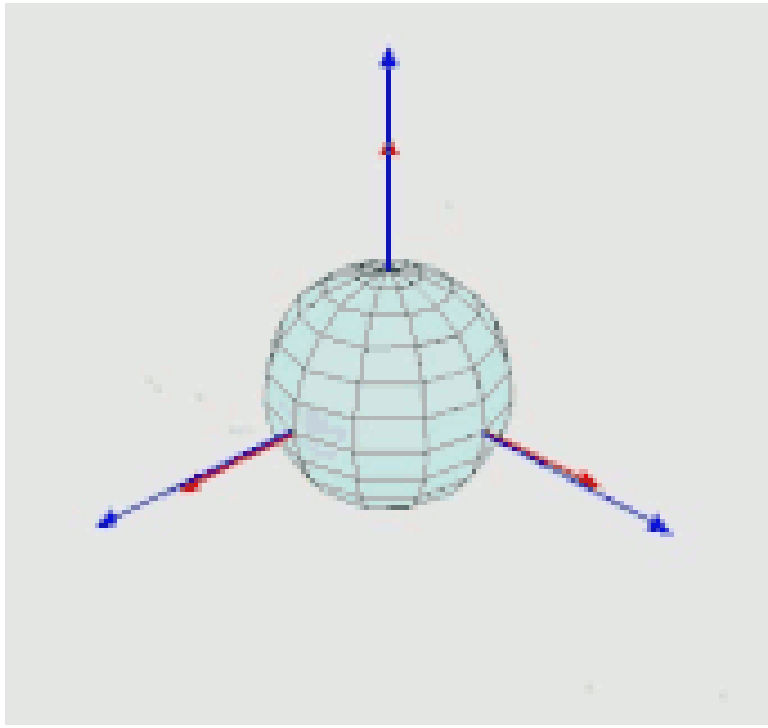
- Euler angles are typically denoted as α, β, γ, or ψ, θ, φ. Different authors may use different sets of rotation axes to define Euler angles, or different names for the same angles. Therefore, any discussion employing Euler angles should *always* be preceded by their definition.





Any target orientation can be reached, starting from a known reference orientation, using a specific sequence of intrinsic rotations, whose magnitudes are the Euler angles of the target orientation. This example uses the *z-x′-z″* sequence.

The three elemental rotations may be extrinsic (rotations about the axes *xyz* of the original coordinate system, which is assumed to remain motionless), or intrinsic (rotations about the axes of the rotating coordinate system *XYZ*, solidary with the moving body, which changes its orientation with respect to the extrinsic frame after each elemental rotation).



- **Proper Euler angles** (z-x-z, x-y-x, y-z-y, z-y-z, x-z-x, y-x-y)
- **Tait–Bryan angles** (x-y-z, y-z-x, z-x-y, x-z-y, z-y-x, y-x-z).

Tait–Bryan angles are also called **Cardan angles**; **nautical angles**; **heading**, **elevation, and bank**; or **yaw, pitch, and roll**. Sometimes, both kinds of sequences are called "Euler angles". In that case, the sequences of the first group are called *proper* or *classic* Euler angles.

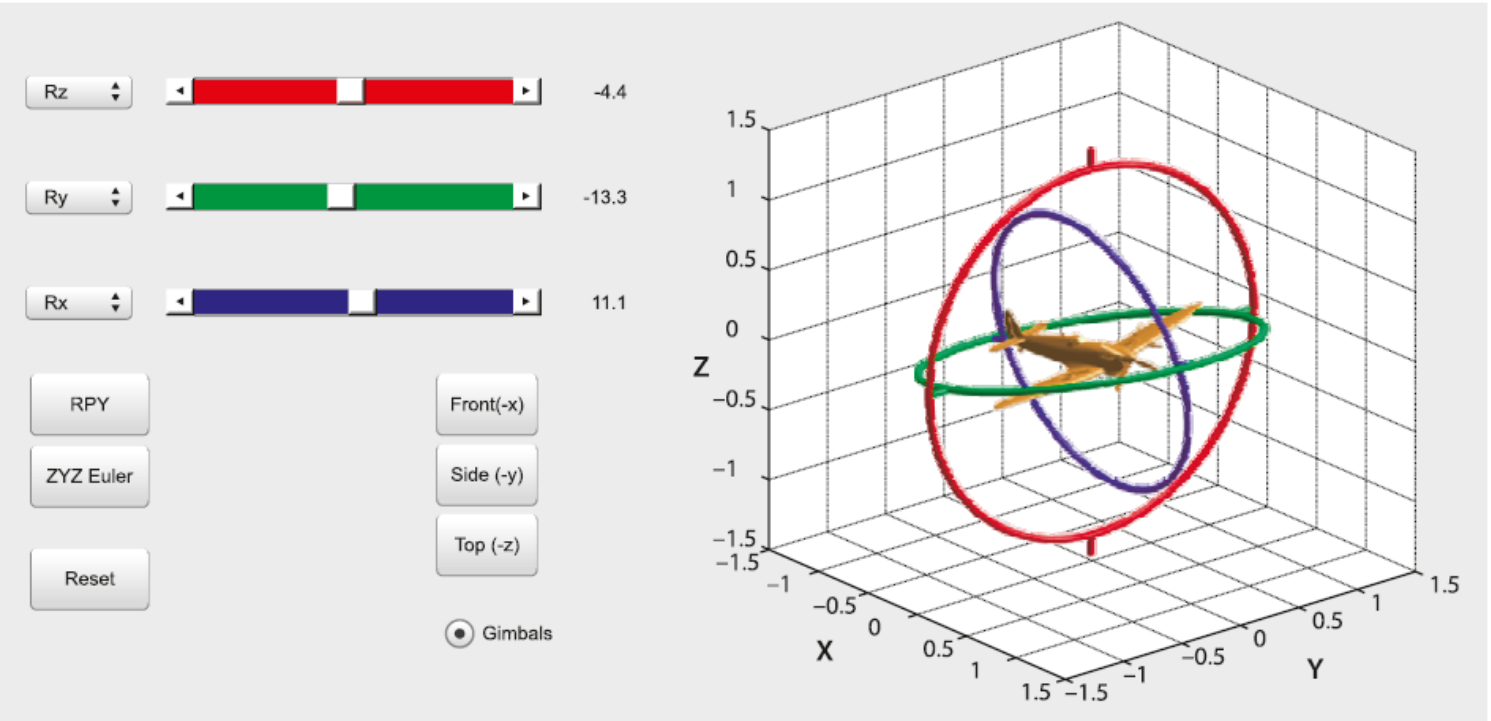


Fig. 2.14.
The Toolbox application `tripleangle` allows you to experiment with Euler angles and roll-pitch-yaw angles and see how the attitude of a body changes

When describing the attitude of a robot gripper, as shown in Fig. 2.16, the convention is that the z-axis points forward and the x-axis is either up or down. This leads to the XYZ angle sequence

$$\boldsymbol{R} = \boldsymbol{R}_x(\theta_y)\boldsymbol{R}_y(\theta_p)\boldsymbol{R}_z(\theta_r) \tag{2.16}$$

The Toolbox defaults to the ZYX sequence but can be overridden using the `'xyz'`

Denavit-Hartenberg: a serial-link robotic arm

Table 7.1.
Denavit-Hartenberg parameters: their physical meaning, symbol and formal definition

Joint angle	θ_j	the angle between the x_{j-1} and x_j axes about the z_{j-1} axis	revolute joint variable
Link offset	d_j	the distance from the origin of frame $j-1$ to the x_j axis along the z_{j-1} axis	prismatic joint variable
Link length	a_j	the distance between the z_{j-1} and z_j axes along the x_j axis; for intersecting axes is parallel to $\hat{z}_{j-1} \times \hat{z}_j$	constant
Link twist	α_j	the angle from the z_{j-1} axis to the z_j axis about the x_j axis	constant
Joint type	σ_j	$\sigma = R$ for a revolute joint, $\sigma = P$ for a prismatic joint	constant

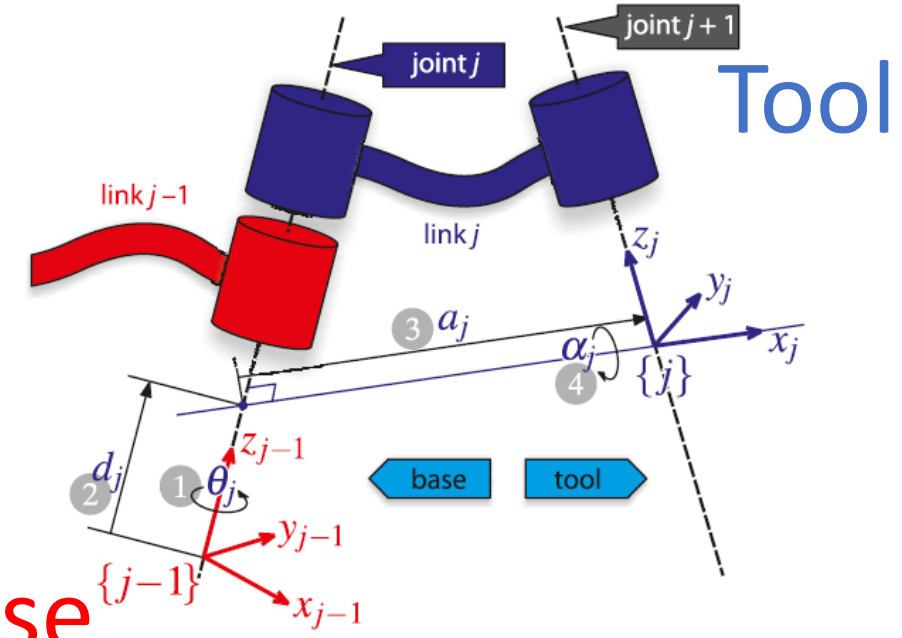


Fig. 7.5.
Definition of standard Denavit and Hartenberg link parameters. The colors red and blue denote all things associated with links $j-1$ and j respectively. The numbers in circles represent the order in which the elementary transforms are applied. x_j is parallel to $z_{j-1} \times z_j$ and if those two axes are parallel then d_j can be arbitrarily chosen

7.1.2.3 6-Axis Industrial Robot

Truly useful robots have a task space $\mathcal{T} \subset \mathbf{SE}(3)$ enabling arbitrary position and attitude of the end-effector – the task space has six spatial degrees of freedom: three translational and three rotational. This requires a robot with a configuration space $\mathcal{C} \subset \mathbb{R}^6$ which can be achieved by a robot with six joints. In this section we will use the Puma 560 as an example of the class of all-revolute six-axis robot manipulators. We define an instance of a Puma 560 robot using the script

```
>> mdl_puma560
```

which creates a `SerialLink` object, `p560`, in the workspace. Displaying the variable shows the table of its Denavit-Hartenberg parameters

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

The **Puma 560 robot** (Programmable Universal Manipulator for Assembly) released in 1978 was the first modern industrial robot and became enormously popular. It featured an anthropomorphic design, electric motors and a spherical wrist – the archetype of all that followed. It can be seen in the Smithsonian Museum of American History, Washington DC.

The Puma 560 catalyzed robotics research in the 1980s and it was a very common laboratory robot. Today it is obsolete and rare but in homage to its important role in robotics research we use it here. For our purposes the advantages of this robot are that it has been well studied and its parameters are very well known – it has been described as the "white rat" of robotics research.

Most modern 6-axis industrial robots are very similar in structure and can be accomodated simply by changing the Denavit-Hartenberg parameters. The Toolbox has kinematic models for a number of common industrial robots from manufacturers such as Rethink, Kinova, Motoman, Fanuc and ABB. (Puma photo courtesy Oussama Khatib)

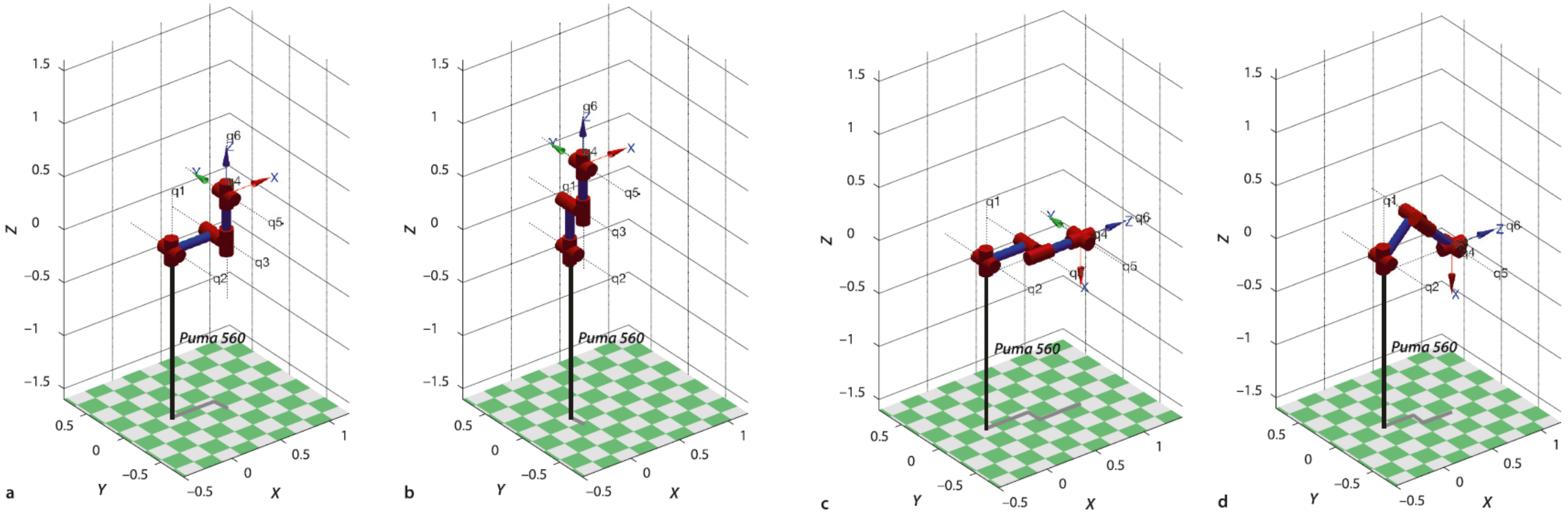
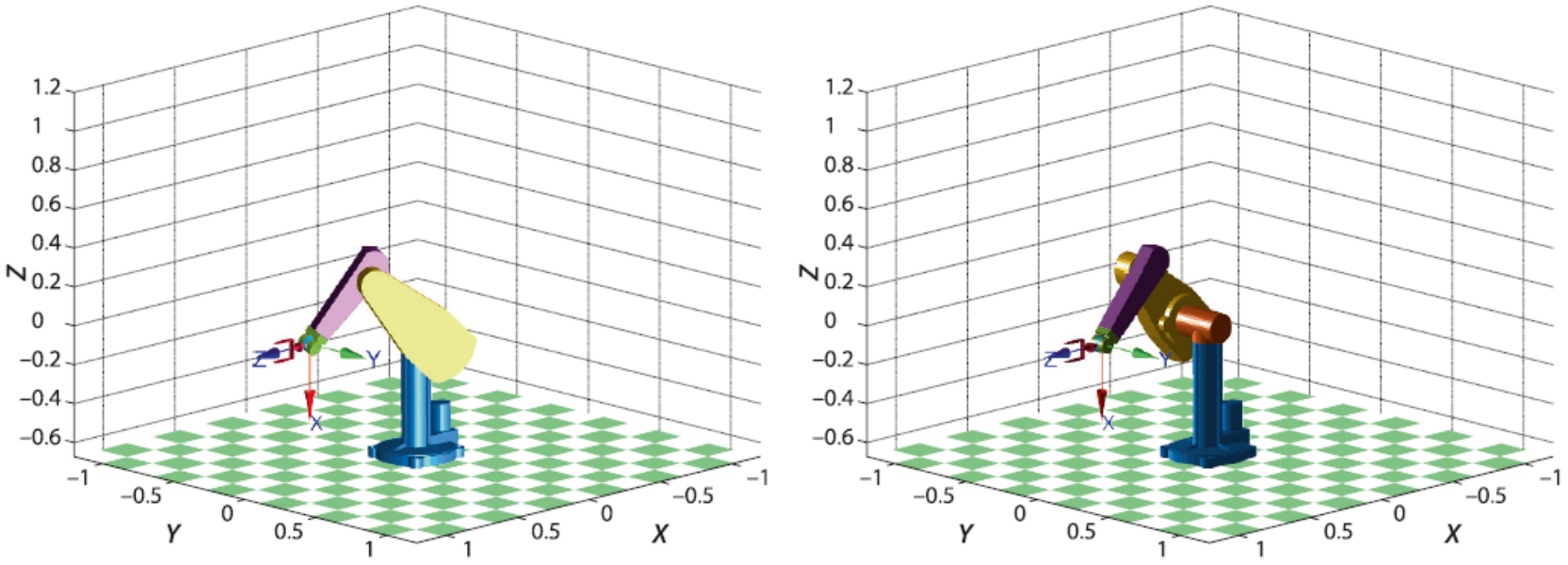


Fig. 7.6. The Puma robot in 4 different poses. **a** *Zero angle*; **b** *ready* pose; **c** *stretch*; **d** *nominal*

Two different robot configurations result in the **same end-effector pose**



where for consistency we have converted the pedestal height to SI units. Now, with both base and tool transform, the forward kinematics are

```
>> p560.fkine(qz)
ans =
    1.0000         0         0    0.4521
         0    1.0000         0   -0.1500
         0         0    1.0000    1.3938
         0         0         0    1.0000
```

Fig. 7.7. These two different robot configurations result in the same end-effector pose. They are called the left- and right-handed configurations, respectively. These graphics, produced using the `plot3d` method, are available for a limited subset of robot models

7.2 Inverse Kinematics

Given the desired pose of the end-effector ξ_E

We have shown how to determine the pose of the end-effector given the joint coordinates and optional tool and base transforms. A problem of real practical interest is the inverse problem: given the desired pose of the end-effector ξ_E what are the required joint coordinates? For example, if we know the Cartesian pose of an object, what joint coordinates does the robot need in order to reach it? This is the inverse kinematics problem which is written in functional form as

$$\boldsymbol{q} = \mathcal{K}^{-1}(\xi) \tag{7.5}$$

and in general this function is not unique, that is, several joint coordinate vectors $\boldsymbol{q}$ will result in the same end-effector pose.

Two approaches can be used to determine the inverse kinematics. Firstly, a closed-form or analytic solution can be determined using geometric or algebraic approaches. However this becomes increasingly challenging as the number or robot joints increases and for some serial-link manipulators no closed-form solution exists. Secondly, an iterative numerical solution can be used. In Sect. 7.2.1 we again use the simple 2-dimensional case to illustrate the principles and then in Sect. 7.2.2 extend these to robot arms that move in 3-dimensions.

1st Closed-form or analytic solution (geometric or algebraic)

2nd Iterative numerical solution

7.2.1 2-Dimensional (Planar) Robotic Arms

We will illustrate inverse kinematics for the 2-joint robot of Fig. 7.2b in two ways: algebraic closed-form and numerical.

7.2.1.1 Closed-Form Solution

We start by computing the forward kinematics algebraically as a function of joint angles. We can do this easily, and in a familiar way

```
>> import ETS2.*
>> a1 = 1; a2 = 1;
>> E = Rz('q1') * Tx(a1) * Rz('q2') * Tx(a2)
```

but now using the MATLAB Symbolic Math Toolbox™ we define some real-valued symbolic variables to represent the joint angles

```
>> syms q1 q2 real
```

7.2.1.2 Numerical Solution

We can think of the 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^*

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

For our simple 2-link example the error function comprises only the error in the end-effector position, not its orientation

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

7.2.2 3-Dimensional Robotic Arms

7.2.2.1 Closed-Form Solution

Closed-form solutions have been developed for most common types of 6-axis industrial robots and many are included in the Toolbox. A necessary condition for a closed-form solution of a 6-axis robot is a spherical wrist mechanism. We will illustrate closed-form inverse kinematics using the Denavit-Hartenberg model for the Puma robot

```
>> mdl_puma560
```

At the *nominal* joint coordinates shown in Fig. 7.6d

```
>> 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
```

Flip

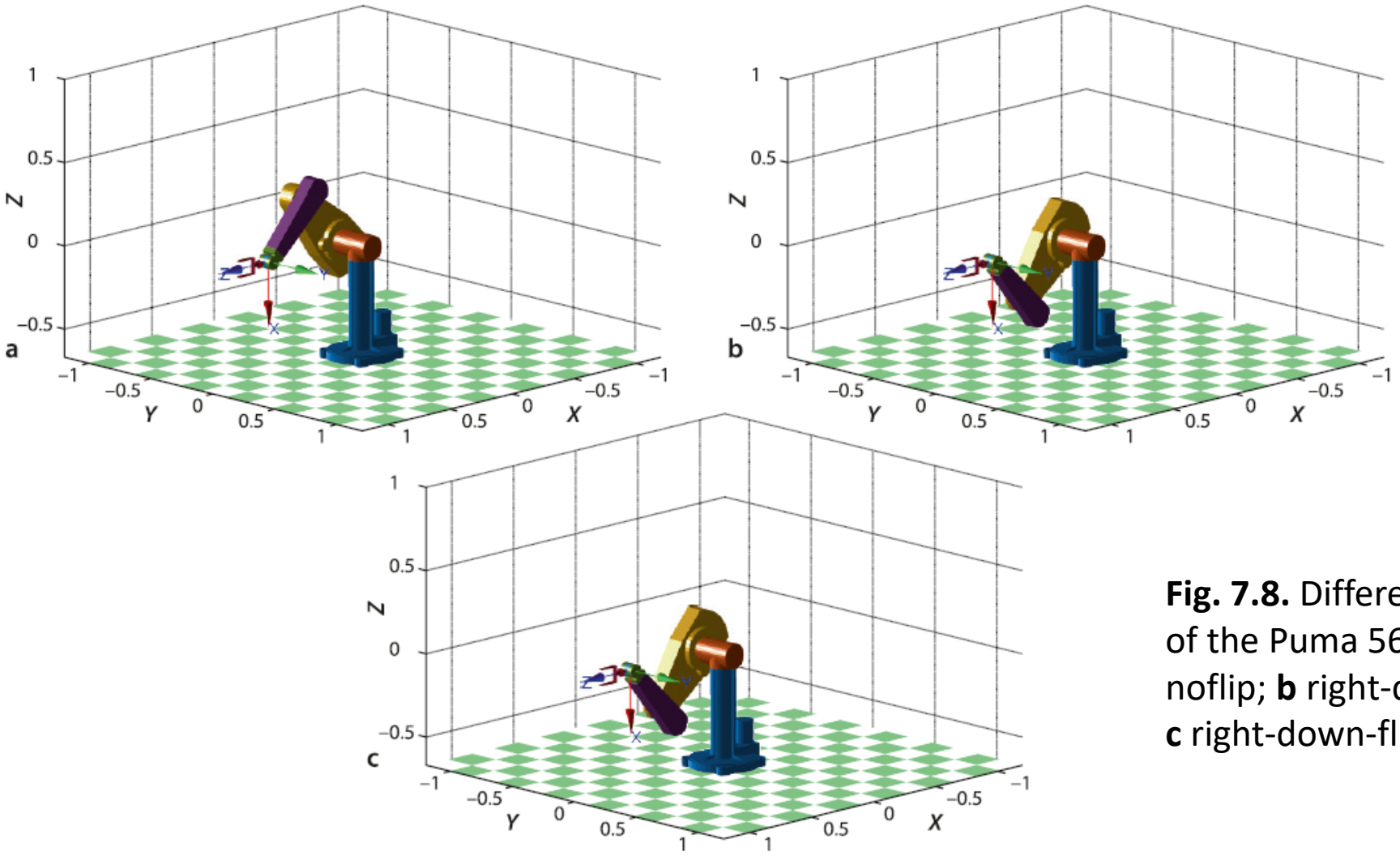


Fig. 7.8. Different configurations of the Puma 560 robot. **a** Right up-noflip; **b** right-down-noflip; **c** right-down-flip

7.3 Trajectories

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 learned in Sect. 3.3 we will discuss two approaches to generating such trajectories: straight lines in configuration space and straight lines in task space. These are known respectively as joint-space and Cartesian motion.

7.3.1 Joint-Space Motion

Consider the end-effector moving between two Cartesian poses◀

```
>> T1 = SE3(0.4,  0.2, 0) * SE3.Rx(pi);
>> T2 = SE3(0.4, -0.2, 0) * SE3.Rx(pi/2);
```

In this robot configuration, similar to Fig. 7.6d, we specify the pose to include a rotation so that the end-effector *z*-axis is not pointing straight up in the world *z*-direction. For the Puma 560 robot this would be physically impossible to achieve in the elbow-up configuration.

which describe points in the *xy*-plane with different end-effector orientations. The joint coordinate vectors associated with these poses are

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

and we require the motion to occur over a time period of 2 seconds in 50 ms time steps

Joint-space motion

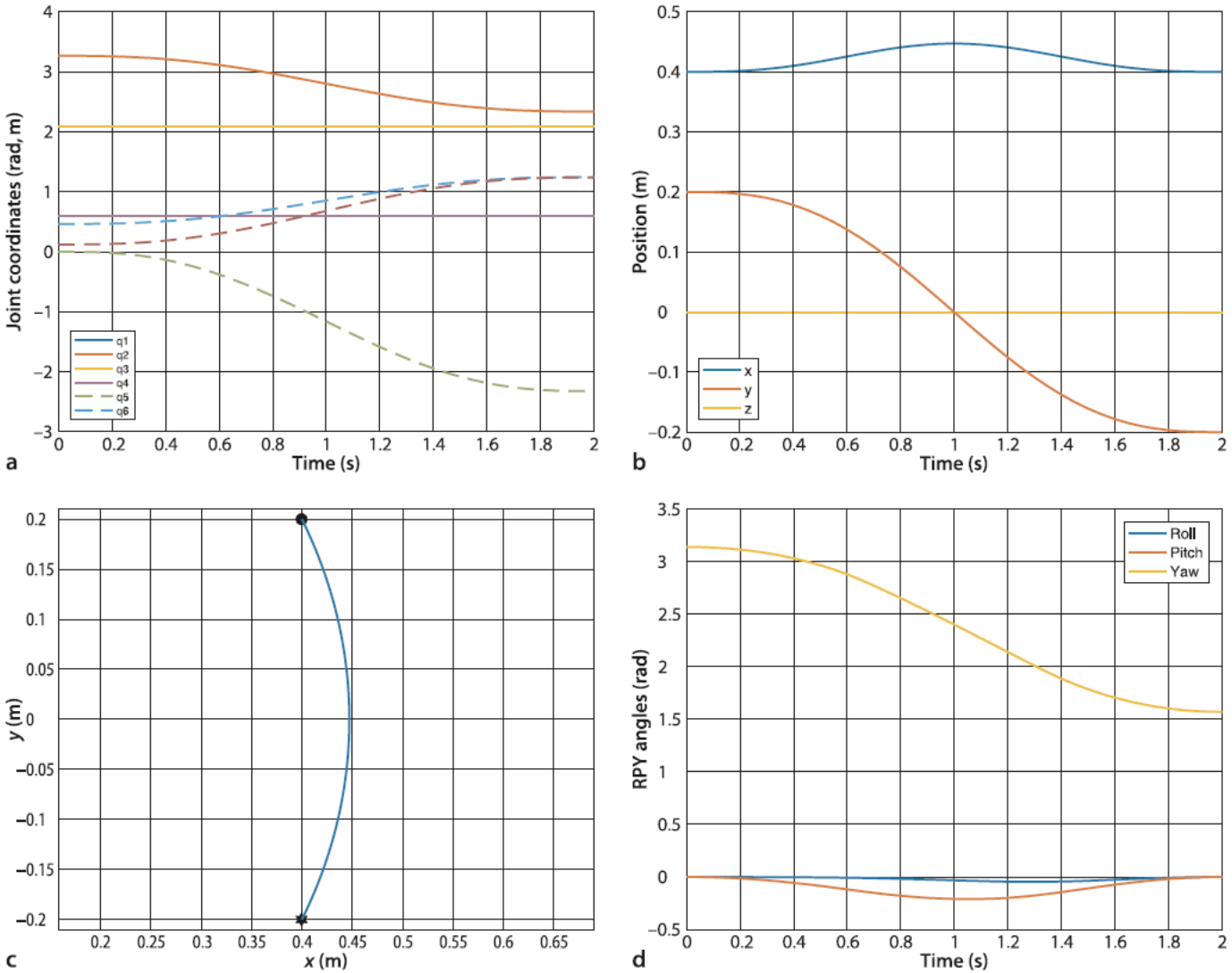


Fig. 7.6. The Puma robot in 4 different poses. **d** *nominal angle*;

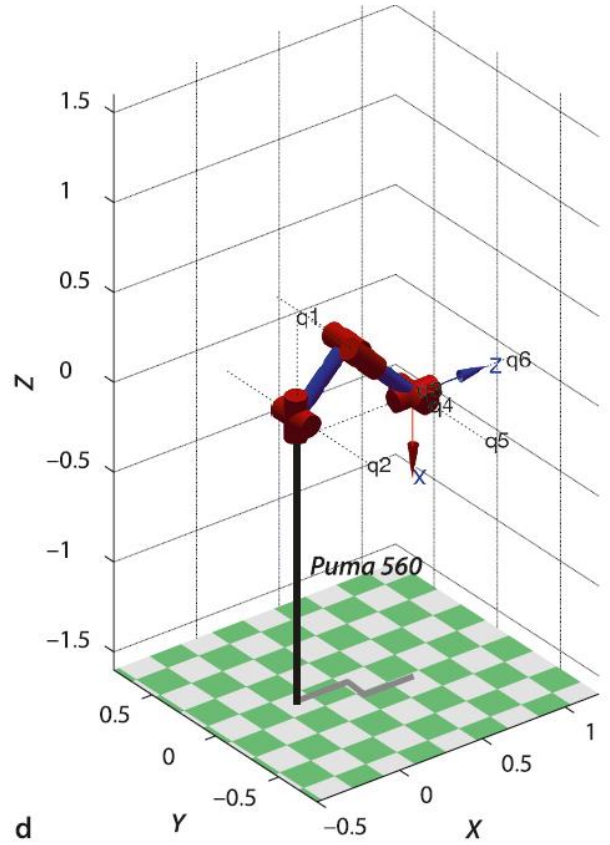


Fig. 7.9. Joint-space motion. **a** Joint coordinates versus time; **b** Cartesian position versus time; **c** Cartesian position locus in the *xy*-plane **d** roll-pitch-yaw angles versus time

7.3.2 Cartesian Motion

For many applications we require straight-line motion in Cartesian space which is known as Cartesian motion. This is implemented using the Toolbox function `ctraj` which was introduced in Sect. 3.3.5. Its usage is very similar to `jtraj`

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

where the arguments are the initial and final pose and the *number of* time steps and it returns the trajectory as an array of `SE3` objects.

As for the previous joint-space example we will extract and plot the translation

```
>> plot(t, Ts.transl);
```

and orientation components

```
>> plot(t, Ts.torpy('xyz'));
```

of this motion which is shown in Fig. 7.10 along with the path of the end-effector in the *xy*-plane. Compared to Fig. 7.9 we note some important differences. Firstly the end-effector follows a straight line in the *xy*-plane as shown in Fig. 7.10c. Secondly the roll and pitch angles shown in Fig. 7.10d are constant at zero along the path.

The corresponding joint-space trajectory is obtained by applying the inverse kinematics

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

and is shown in Fig. 7.10a. While broadly similar to Fig. 7.9a the minor differences are what result in the straight line Cartesian motion.

7.3.3 Kinematics in Simulink

We can also implement this example in Simulink®

```
>> sl_jspace
```

and the block diagram model is shown in Fig. 7.11. The parameters of the `jtraj` block are the initial and final values for the joint coordinates and the time duration of the motion segment. The smoothly varying joint angles are wired to a `plot` block which will animate a robot in a separate window, and to an `fkine` block to compute the forward kinematics. Both the `plot` and `fkine` blocks have a parameter which is a `SerialLink` object, in this case `p560`. The Cartesian position of the end-effector pose is extracted using the `T2xyz` block which is analogous to the Toolbox function `transl`. The `XY Graph` block plots `y` against `x`.

Simulink model sl_jspace

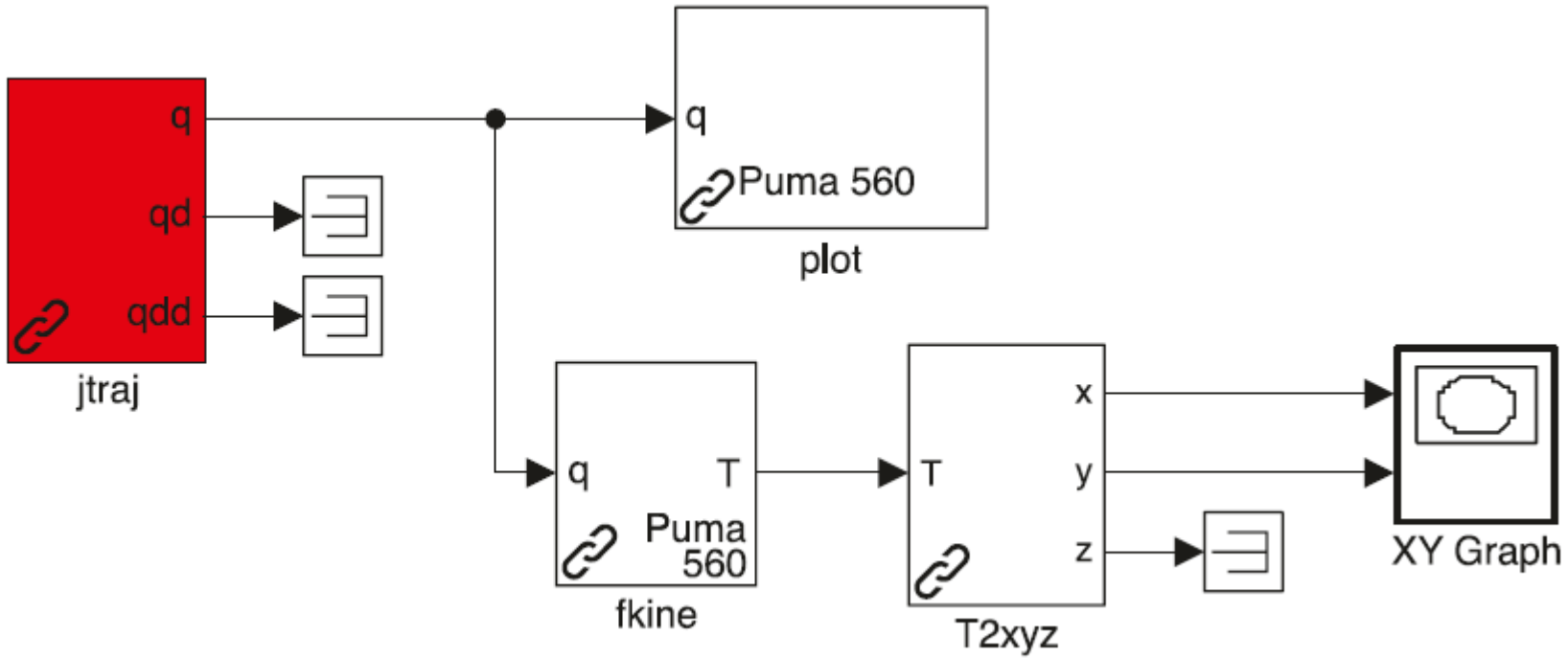


Fig. 7.11.
Simulink model `sl_jspace` for joint-space motion

7.3.4 Motion through a Singularity

We have already briefly touched on the topic of singularities (page 209) and we will revisit it again in the next chapter. In the next example we deliberately choose a trajectory that moves through a robot wrist singularity. We change the Cartesian endpoints of the previous example to

```
>> T1 = SE3(0.5,  0.3, 0.44) * SE3.Ry(pi/2);
>> T2 = SE3(0.5, -0.3, 0.44) * SE3.Ry(pi/2);
```

which results in motion in the y-direction with the end-effector z-axis pointing in the world x-direction. The Cartesian path is

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

which we convert to joint coordinates

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

Cartesian motion through a wrist singularity

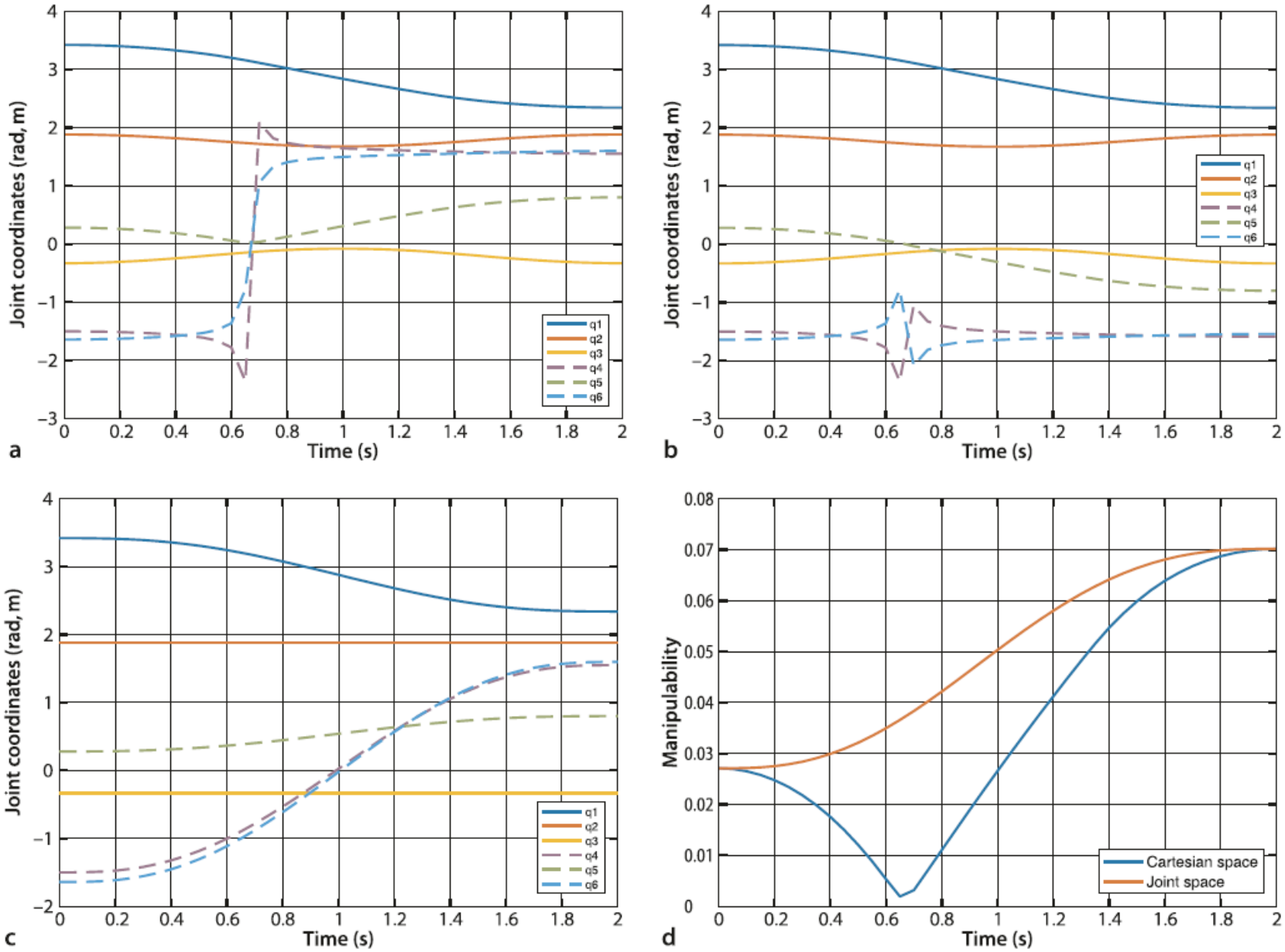


Fig. 7.12. Cartesian motion through a wrist singularity. **a** Joint coordinates computed by inverse kinematics (`ikine6s`); **b** joint coordinates computed by numerical inverse kinematics (`ikine`); **c** joint coordinates for joint-space motion; **d** manipulability

7.4 Advanced Topics

7.4.1 Joint Angle Offsets

The pose of the robot with zero joint angles is an arbitrary decision of the robot designer and might even be a mechanically unachievable pose. For the Puma robot the zero-angle pose is a nonobvious *L-shape* with the upper arm horizontal and the lower arm vertically upward as shown in Fig. 7.6a. This is a consequence of constraints imposed by the Denavit-Hartenberg formalism.

The joint coordinate offset provides a mechanism to set an arbitrary configuration for the zero joint coordinate case. The offset vector, q_0, is added to the user specified joint angles before any kinematic or dynamic function is invoked, for example

$$\xi_E = \mathcal{K}(\boldsymbol{q} + \boldsymbol{q}_0) \tag{7.6}$$

Similarly it is subtracted after an operation such as inverse kinematics

$$\boldsymbol{q} = \mathcal{K}^{-1}(\xi_E) - \boldsymbol{q}_0 \tag{7.7}$$

The offset is set by assigning the `offset` property of the `Link` object, or giving the `'offset'` option to the `SerialLink` constructor.

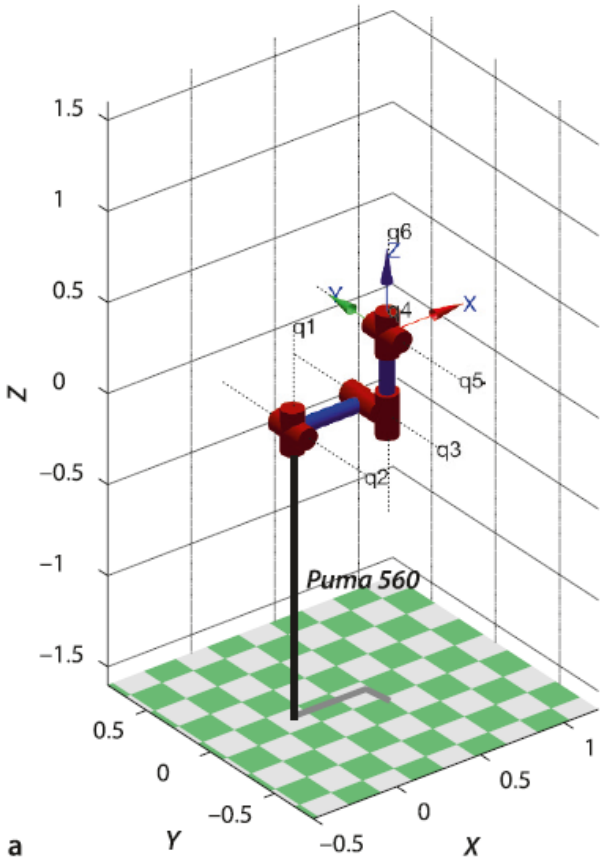


Fig. 7.6. The Puma robot in 4 different poses. a *Zero angle*;

7.4.2 Determining Denavit-Hartenberg Parameters

The classical method of determining Denavit-Hartenberg parameters is to systematically assign a coordinate frame to each link. The link frames for the Puma robot using the standard Denavit-Hartenberg formalism are shown in Fig. 7.14. However there are strong constraints on placing each frame since joint rotation must be about the z-axis and the link displacement must be in the x-direction. This in turn imposes constraints on the placement of the coordinate frames for the base and the end-effector, and ultimately dictates the zero-angle pose just discussed. Determining the Denavit-Hartenberg parameters and link coordinate frames for a completely new mechanism is therefore more difficult than it should be – even for an experienced roboticist.

An alternative approach, supported by the Toolbox, is to simply describe the manipulator as a series of elementary translations and rotations from the base to the tip of the end-effector as we discussed in Sect. 7.1.2. Some of the elementary operations are constants such as translations that represent link lengths or offsets, and

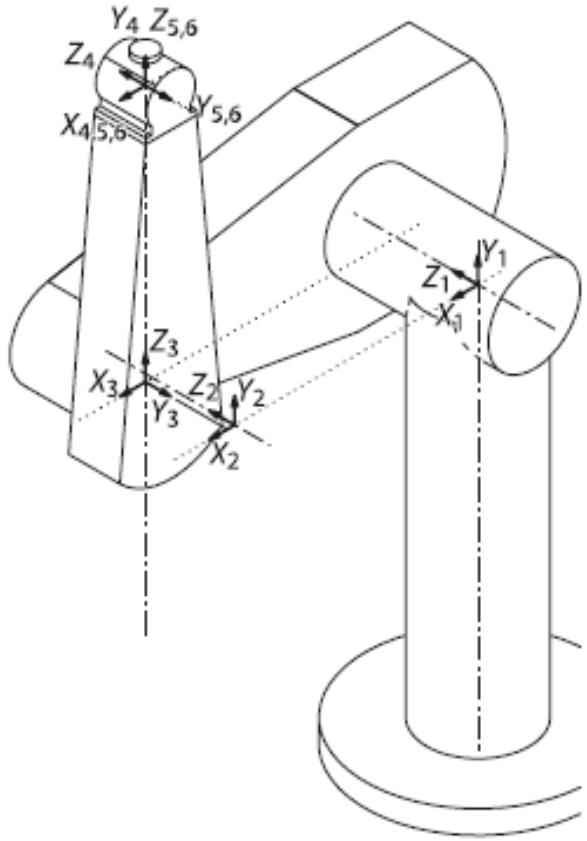


Fig. 7.14. Puma 560 robot coordinate frames. Standard Denavit-Hartenberg link coordinate frames for Puma in the zeroangle pose (Corke 1996b)

Review 7.1.2

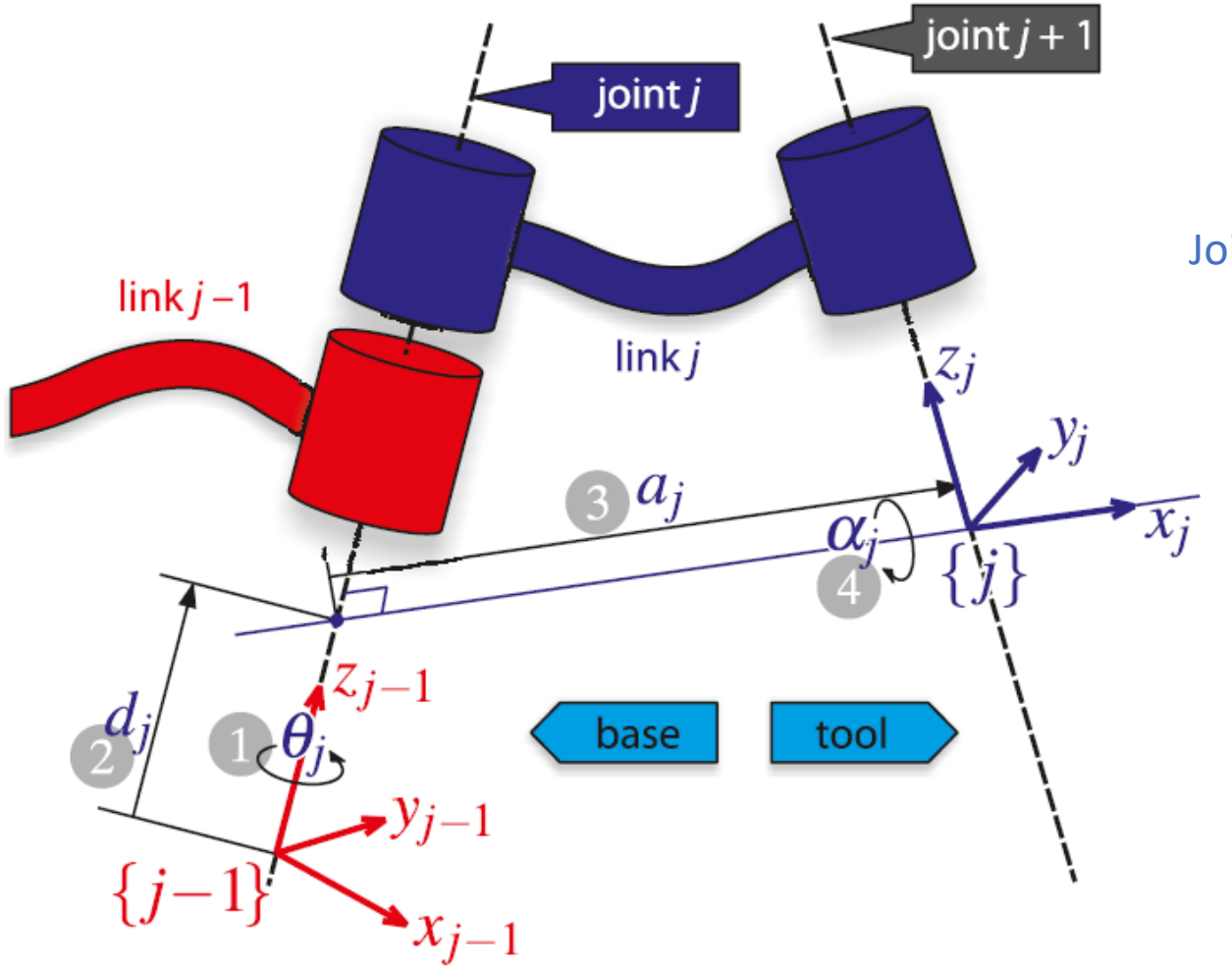
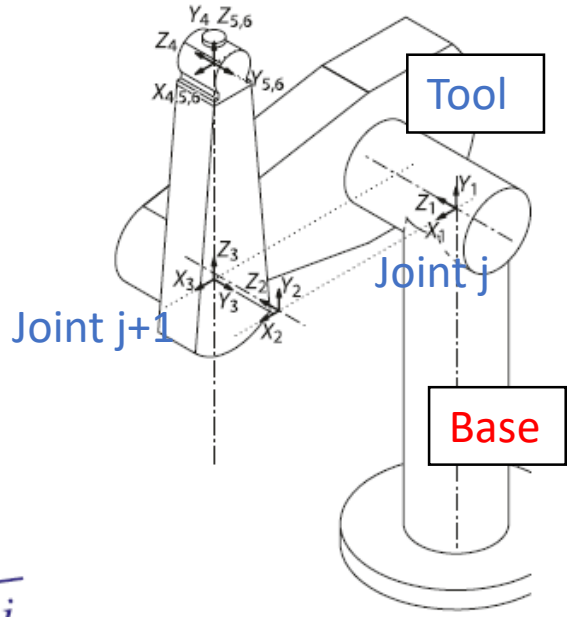


Fig. 7.5. Definition of standard Denavit and Hartenberg link parameters. The colors red and blue denote all things associated with links $j-1$ and j respectively. The numbers in circles represent the order in which the elementary transforms are applied. x_j is parallel to $z_{j-1} \times z_j$ and if those two axes are parallel then d_j can be arbitrarily chosen



7.4.3 Modified Denavit-Hartenberg Parameters

The Denavit-Hartenberg notation introduced in this chapter is commonly used and described in many robotics textbooks. Craig (1986) first introduced the modified Denavit-Hartenberg parameters where the link coordinate frames shown in Fig. 7.15 are attached to the near (proximal), rather than the far (distal) end of each link. This modified notation is in some ways clearer and tidier and is also now commonly used. However its introduction has increased the scope for confusion, particularly for those who are new to robot kinematics. The root of the problem is that the algorithms for kinematics, Jacobians and dynamics depend on the kinematic conventions used. According to Craig's convention the link transform matrix is

$$^{j-1}\xi_j\left(\alpha_{j-1}, a_{j-1}, d_j, \theta_j\right) = \mathscr{R}_x\left(\alpha_{j-1}\right) \oplus \mathscr{T}_x\left(a_{j-1}\right) \oplus \mathscr{T}_z\left(d_j\right) \oplus \mathscr{R}_z\left(\theta_j\right) \tag{7.8}$$

denoted in that book as $^{j-1}_{j}A$. This has the same terms as Eq. 7.2 but in a different order – remember rotations are not commutative – and this is the nub of the problem. a_j is

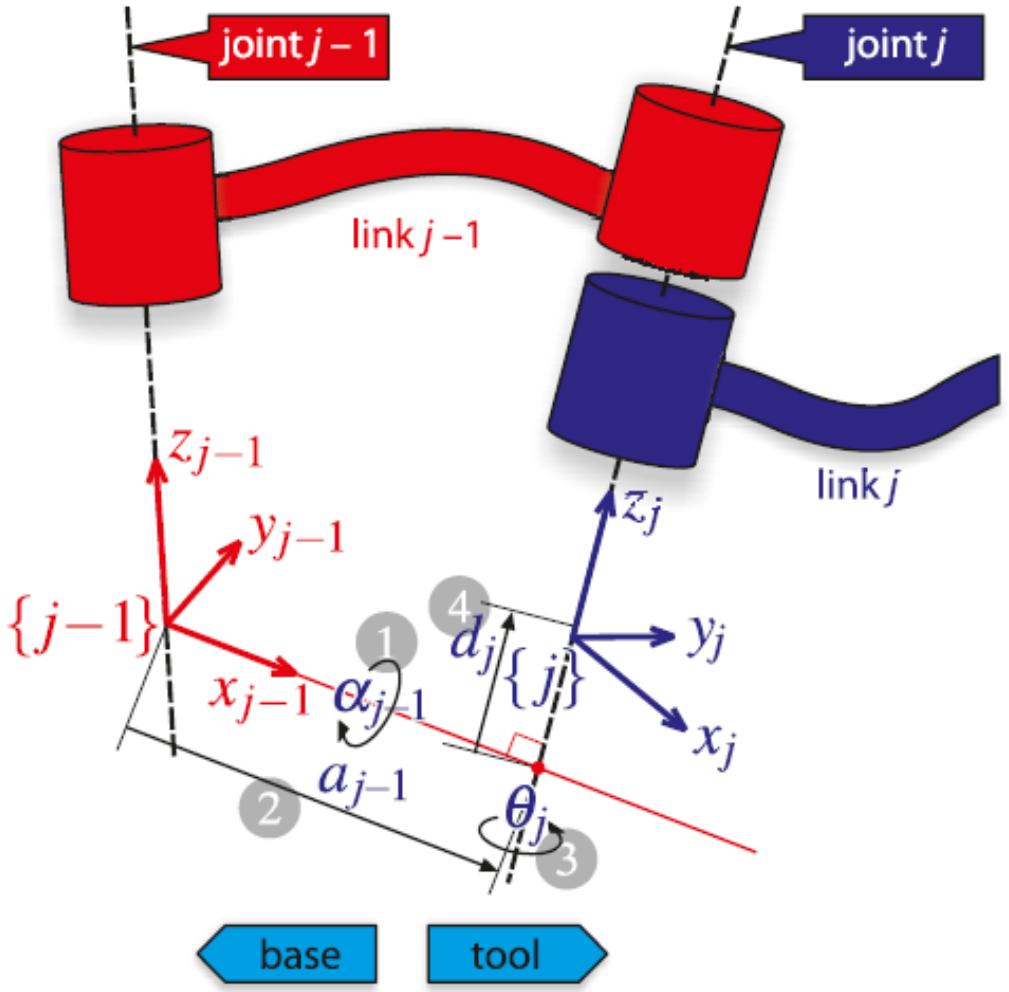


Fig. 7.15. Definition of modified Denavit and Hartenberg link parameters. The colors *red* and *blue* denote all things associated with links $j-1$ and j respectively. The *numbers in circles* represent the order in which the elementary transforms are applied

always the length of link j, but it is the displacement between the origins of frame $\{j\}$ and frame $\{j+1\}$ in one convention, and frame $\{j-1\}$ and frame $\{j\}$ in the other.

end-effector path drawing – "B"

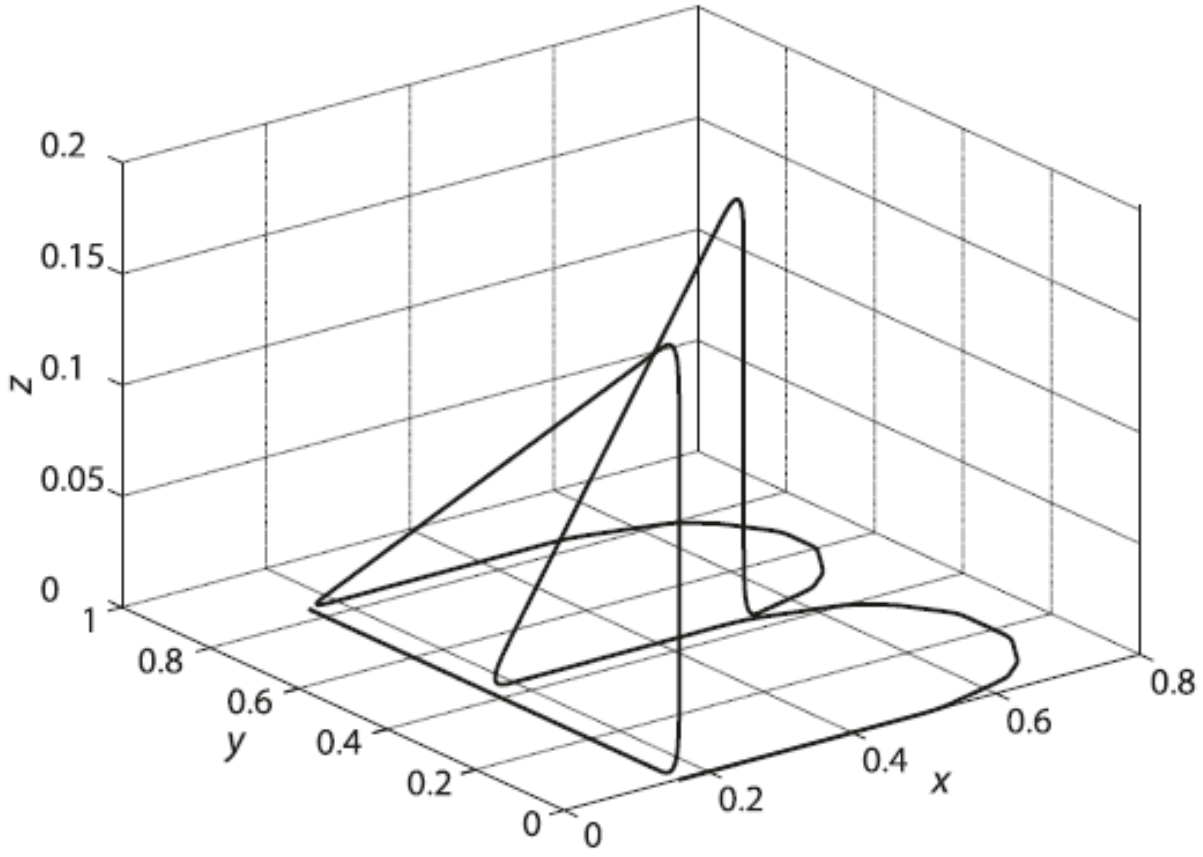


Fig. 7.16.
The end-effector path drawing the letter 'B'

7.5.2 A Simple Walking Robot [examples/walking.m]

Four legs good, two legs bad!
Snowball the pig, Animal Farm by George Orwell

Our goal is to create a four-legged walking robot. We start by creating a 3-axis robot *arm* that we use as a leg, plan a trajectory for the leg that is suitable for walking, and then instantiate four instances of the leg to create the walking robot.

The coordinate frame and axis rotations for the simple leg

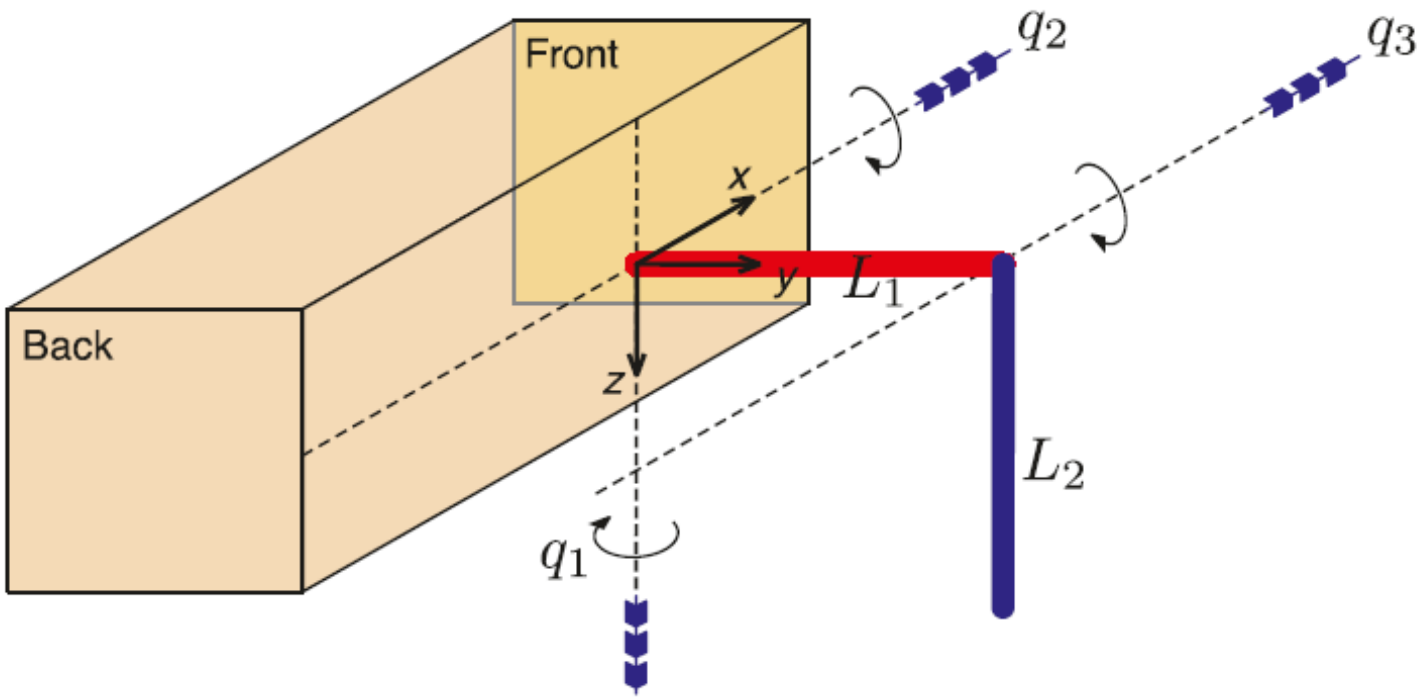


Fig. 7.17.
The coordinate frame and axis rotations for the simple leg. The leg is shown in its zero angle pose

Robot leg in its zero angle pose

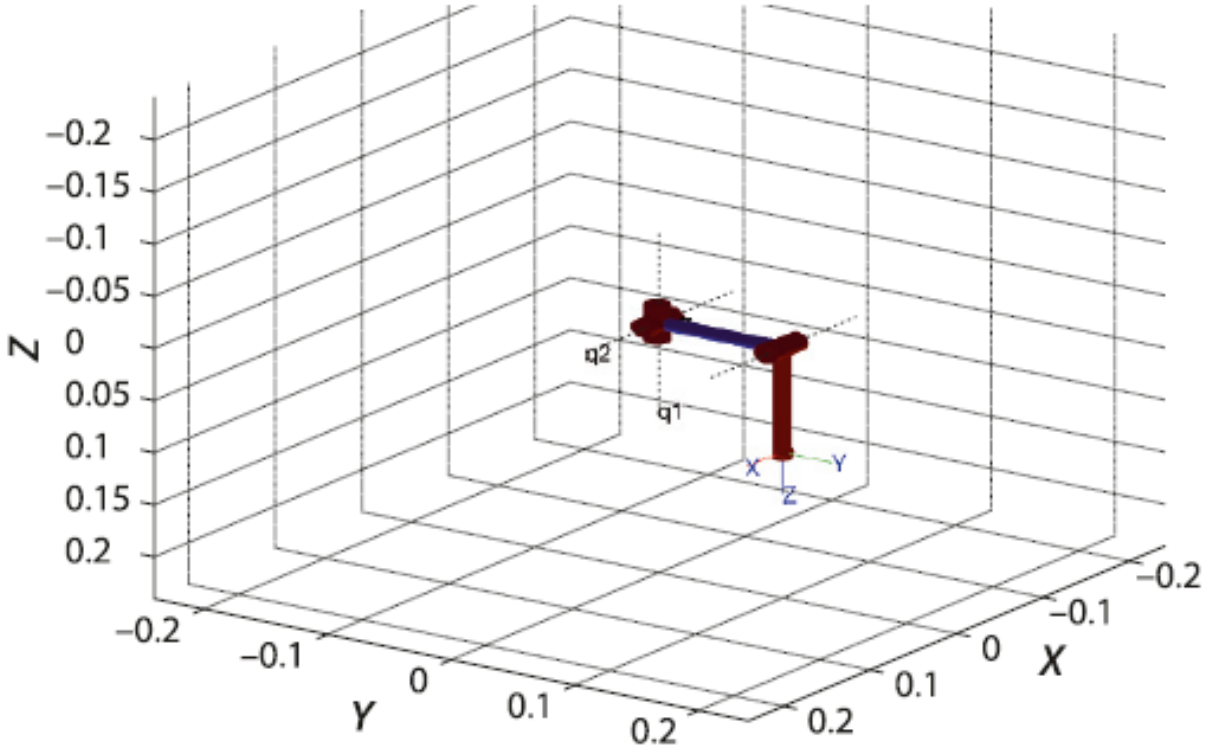


Fig. 7.18.
Robot leg in its zero angle pose. Note that the *z*-axis points downward

Motion of One Leg

The next step is to define the path that the end-effector of the leg, its foot, will follow. The first consideration is that the end-effector of all feet move backwards at the same speed in the ground plane – propelling the robot's body forward without its feet slipping. Each leg has a limited range of movement so it cannot move backward for very long. At some point we must reset the leg – lift the foot, move it forward and place it on the ground again. The second consideration comes from static stability – the robot must have at least three feet on the ground at all times so each leg must take its turn to reset. This requires that any leg is in contact with the ground for ¾ of the cycle and is resetting for ¼ of the cycle. A consequence of this is that the leg has to move much faster during reset since it has a longer path and less time to do it in.

The required trajectory is defined by the via points

```
>> xf = 50; xb = -xf;  y = 50; zu = 20; zd = 50;
>> path = [xf y zd; xb y zd; xb y zu; xf y zu; xf y zd] * 1e-3;
```

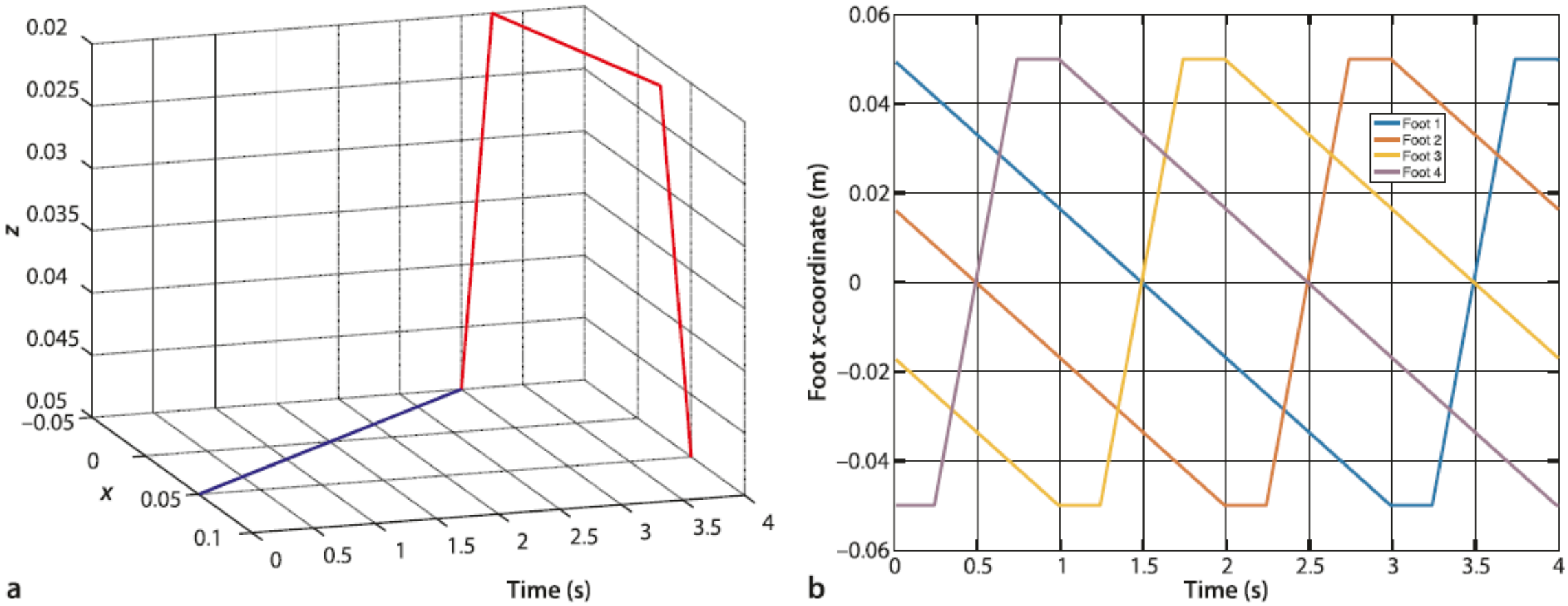


Fig. 7.19. a Trajectory taken by a single foot. Recall from Fig. 7.17 that the *z*-axis is downward. The red segments are the leg reset. b The *x*-direction motion of each leg (offset vertically) to show the gait. The leg reset is the period of high *x*-direction velocity

Motion of Four Legs

Our robot has width and length

```
>> W = 0.1; L = 0.2;
```

We create multiple instances of the leg by cloning the `leg` object we created earlier, and providing different base transforms so as to attach the legs to different points on the body

```
>> legs(1) = SerialLink(leg, 'name', 'leg1');
>> legs(2) = SerialLink(leg, 'name', 'leg2', 'base', SE3(-L, 0, 0));
>> legs(3) = SerialLink(leg, 'name', 'leg3', 'base', SE3(-L, -W, 0) ↵
 *SE3.Rz(pi));
>> legs(4) = SerialLink(leg, 'name', 'leg4', 'base', SE3(0, -W, 0) ↵
 *SE3.Rz(pi));
```

The walking robot

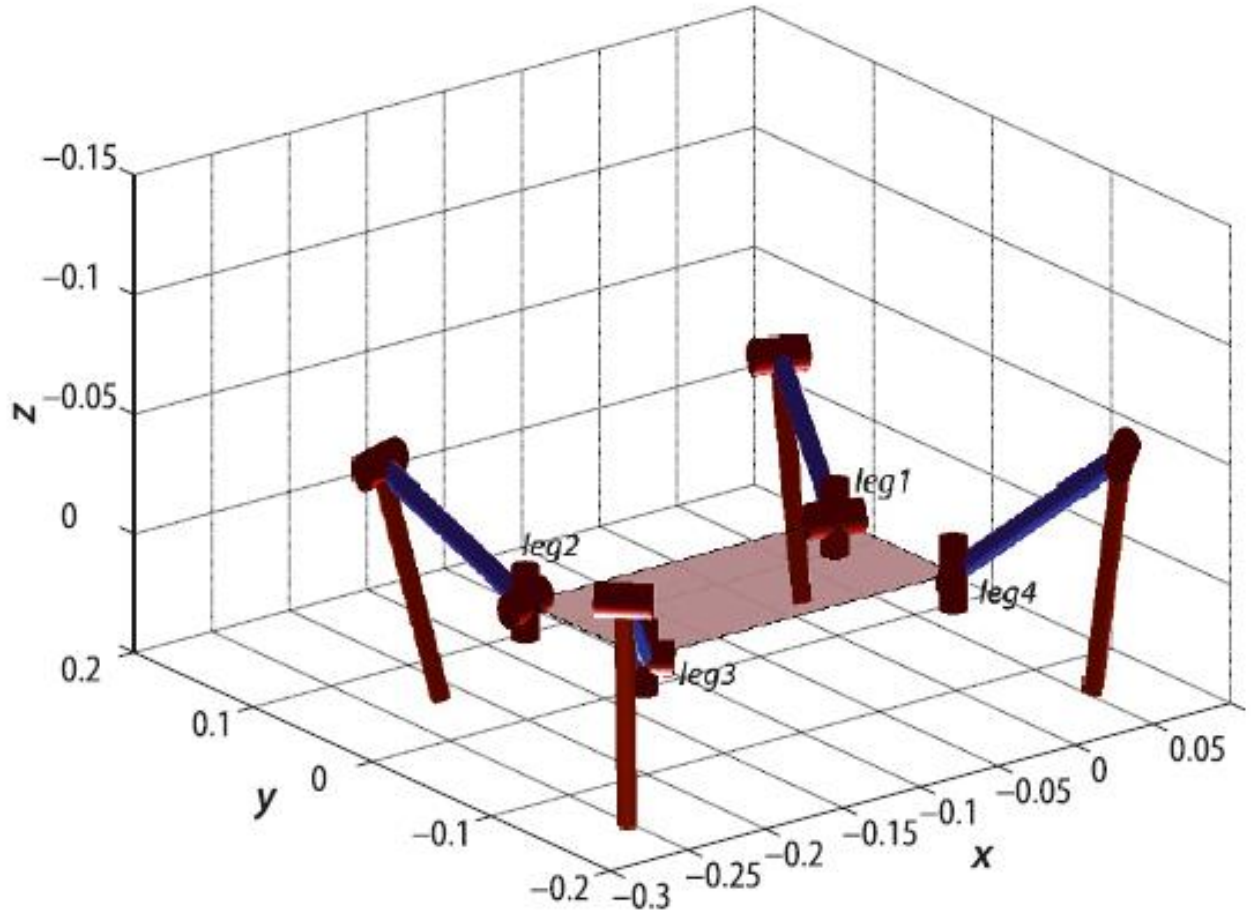


Fig. 7.20.
The walking robot

Exercises

Exercises

1. Forward kinematics for planar robot from Sect. 7.1.1.
   a) For the 2-joint robot use the `teach` method to determine the two sets of joint angles that will position the end-effector at (0.5, 0.5).
   b) Experiment with the three different models in Fig. 7.2 using the `fkine` and `teach` methods.
   c) Vary the models: adjust the link lengths, create links with a translation in the y-direction, or create links with a translation in the x- and y-direction.
2. Experiment with the `teach` method for the Puma 560 robot.
3. Inverse kinematics for the 2-link robot on page 206.
   a) Compute forward and inverse kinematics with a_1 and a_2 as symbolic rather than numeric values.
   b) What happens to the solution when a point is out of reach?
   c) Most end-effector positions can be reached by two different sets of joint angles. What points can be reached by only one set?
4. Compare the solutions generated by `ikine6s` and `ikine` for the Puma 560 robot at different poses. Is there any difference in accuracy? How much slower is `ikine`?
5. For the Puma 560 at configuration `qn` demonstrate a configuration change from elbow up to elbow down.
6. For a Puma 560 robot investigate the errors in end-effector pose due to manufacturing errors.
   a) Make link 2 longer by 0.5 mm. For 100 random joint configurations what is the mean and maximum error in the components of end-effector pose?
   b) Introduce an error of 0.1 degrees in the joint 2 angle and repeat the analysis above.
7. Investigate the redundant robot models `mdl_hyper2d` and `mdl_hyper3d`. Manually control them using the `teach` method, compute forward kinematics and numerical inverse kinematics.