3 Robot Kinematics

In this chapter we discuss kinematics for serial link manipulators. Roughly speaking, a kinematics problem refers to the study of motion resulting from arm geometry, regardless of any force or torque acting on the system. While a typical textbook likes to separate the position/orientation from velocity analysis, we combine the two issues into this chapter. The study relies on the math basics from chapter 2, in particular, the homogeneous transformation. Essentially, we attach coordinate frames to each joint of a robot like shown in Figure 3.1. Motion of the end-effector w.r.t the reference frame, usually the base frame, can then be computed by multiplying each homogeneous matrix from base upwards. For the RPR robot in Figure 3.1, for example,

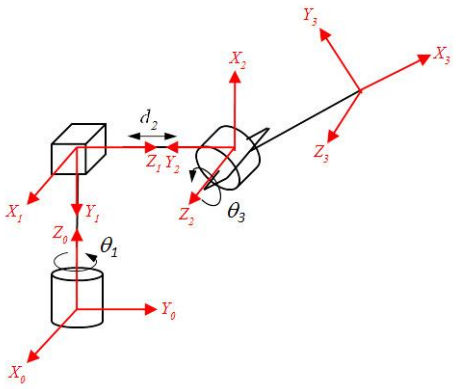


Figure 3.1 coordinate frames attached to a robot.

$$^0T_3 = {}^0T_1\,{}^1T_2\,{}^2T_3 \tag{3.1}$$

This is known as a *forward kinematics* problem, in contrast to an *inverse kinematics* problem where the goal of the latter is to solve for joint variables from a given robot configuration. In the first part of this chapter, we discuss forward kinematics and develop a systematic way to solve for a solution.

For a serial link robot manipulator, one might attempt to attach a frame to each joint arbitrarily and compute the overall homogeneous transformation such as in (3.1). The complexity increases with the number of joints, since generally there are 6 parameters between each pair of joints: 3 for translation and 3 for rotation. So we want to find a description that depends on joint variables only, leaving other parameters as constant. Moreover, the total number of parameters needs to be kept minimal.

Let q_i denote the i^{th} joint variable. We write the homogenous transformation of this joint w.r.t the previous joint as

$$A_i = A_i(q_i) = {}^{i-1}T_i \tag{3.2}$$

which has the structure

$$A_i = \begin{bmatrix} {}^{i-1}R_i & {}^{i-1}o_i \\ 0 & 1 \end{bmatrix} \tag{3.3}$$

Hence the homogeneous transformation of the end-effector w.r.t base can be expressed as

$${}^{0}T_n = A_1(q_1)A_2(q_2)\ldots A_n(q_n) \tag{3.4}$$

3.1 The Denavit-Hartenberg Convention

A commonly-used scheme for attaching frames to robot joints is the so-called Denavit-Hartenberg (DH) convention, introduced by Jacques Denavit and Richard S. Hargenberg in 1955. The scheme has an advantage of reducing parameters between a pair of joints from 6 down to 4. Most standard robot texts explain in detail about the basics and derivation of the DH convention. Here we focus on its application to any robot configuration.

In essence, for a robot with frames attached conforming to DH convention, it can be shown that each of the homogeneous transformation A_i is a product of 4 basic transformation matrices

$$\begin{aligned} A_i &= Trot_{z,\theta_i} Transl_{z,d_i} Transl_{x,a_i} Trot_{x,\alpha_i} \\ &= \begin{bmatrix} \mathrm{c}_{\theta_i} & -\mathrm{s}_{\theta_i} & 0 & 0 \\ \mathrm{s}_{\theta_i} & \mathrm{c}_{\theta_i} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & a_i \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \mathrm{c}_{\alpha_i} & -\mathrm{s}_{\alpha_i} & 0 \\ 0 & \mathrm{s}_{\alpha_i} & \mathrm{c}_{\alpha_i} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \\ &= \begin{bmatrix} \mathrm{c}_{\theta_i} & -\mathrm{s}_{\theta_i}\mathrm{c}_{\alpha_i} & \mathrm{s}_{\theta_i}\mathrm{s}_{\alpha_i} & a_i\mathrm{c}_{\theta_i} \\ \mathrm{s}_{\theta_i} & \mathrm{c}_{\theta_i}\mathrm{c}_{\alpha_i} & -\mathrm{c}_{\theta_i}\mathrm{s}_{\alpha_i} & a_i\mathrm{s}_{\theta_i} \\ 0 & \mathrm{s}_{\alpha_i} & \mathrm{c}_{\alpha_i} & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}. \end{aligned} \tag{3.5}$$

The four DH parameters $\theta_i, d_i, a_i, \alpha_i$ are called joint angle, link offset, link length, and link twist, respectively. Only one of these parameters is a joint variable; the other three are constants. In particular, θ_i is a joint variable for joint type (R), as d_i is for (P).

In order to conform to the DH convention, some rules must be followed. But this does not mean the DH frame structure for a robot is unique. Two persons might get slightly different frame assignments that are both valid. Nevertheless, if the base and tool frames for the two are identical, the forward kinematics solutions should match.

3.1.1 The DH Frame Assignment Procedure

Here we describe step-by-step how to attach coordinate frames to a robot conforming to the DH convention. Without loss of generality, the base frame {0} is assumed to be at the lowermost joint of robot, with its Z axis aligned with the rotational or linear axis of the joint for type (R) and (P), respectively. In practice, if the robot is installed at different location/orientation in the room, such as on a wall, ceiling, say, a base homogeneous transformation can be applied to compensate for the mounting configuration change.

Step 1: Attach z_i to all joint axes, starting from base upwards. There are 2 cases according to joint types, as shown in Figure 3.2. If joint $i+1$ is of type (R), z_i must coincide with the joint axis of rotation. For joint type (P), z_i is along the translation axis. The arrow direction of z_i is arbitrary. Notice that the joint index is one more than the frame index; i.e., {0} is attached to joint 1, {1} to joint 2, and so on.

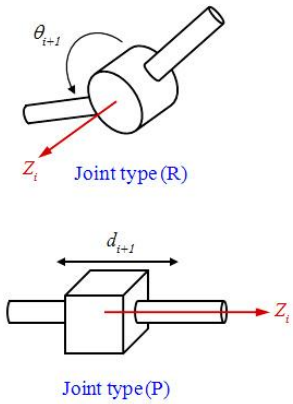


Figure 3.2 attaching z_i on robot joints

Step 2: Determine a location and orientation of base frame. This depends on user's preference, as long as the origin lies on z_i chosen from step 1 and all principal axes follow right-hand rule. Remember that the base frame is typically our reference for forward kinematics analysis, so the solution depends on its position/orientation.

Step 3: Determine position/orientation of each joint frame, from {1} upwards. Do this step recursively until the uppermost joint is reached. Indeed, we only need to locate the origin o_i and the direction of x_i, since from a choice of o_i and x_i, y_i is achieved by right-hand rule. How to choose o_i and x_i depends on the frame just below it, in particular, z_{i-1} and o_{i-1}. There are basically 3 cases as follows:

Case 1: z_{i-1} and z_i are not coplanar.

If z_{i-1} and z_i do not lie on the same plane, there exist a unique, shortest perpendicular line between them as shown in Figure 3.3. Choose o_i at the intersection of this line and z_i, and choose x_i in any direction along this line. y_i follows RH rule.

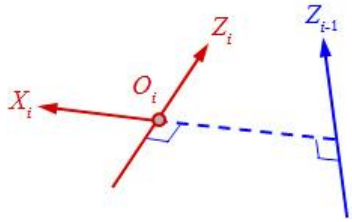


Figure 3.3 z_{i-1} and z_i are not coplanar

The other two cases happen when z_{i-1} and z_i are coplanar.

Case 2: z_{i-1} and z_i are parallel. There are infinitely many lines perpendicular to both axes. The most convenient way is to select the line that also passes through o_{i-1} as shown in Figure 3.4, to get rid of the link offset parameter. Choose o_i at the intersection of this line and z_i , and choose x_i in any direction along this line. y_i follows RH rule.

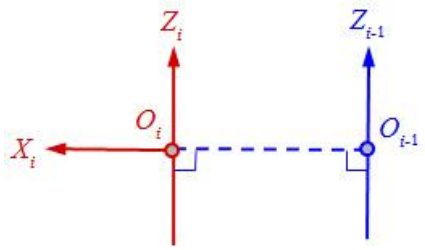


Figure 3.4 z_{i-1} and z_i are parallel.

Case 3: z_{i-1} and z_i intersects. When z_{i-1} and z_i are coplanar but not parallel, they should intersect at a point, as shown in Figure 3.5. Choose o_i at the intersection and choose x_i in any direction normal to the plane that contains z_{i-1} and z_i. y_i follows RH rule.

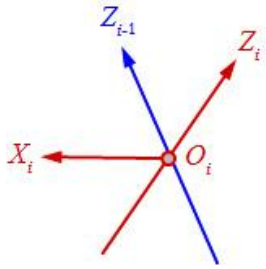


Figure 3.5 z_{i-1} and z_i intersects.

Step 4: Attach an end-effector frame. This can be done quite freely as long as it conforms to DH requirements*. Figure 3.6 shows some guideline for a gripper tool. The letter abbreviation stands for n = normal, o = orientation (some text uses s = sliding), and a = approach. Care must be taken since in some case this suggestion might violate the DH requirements and then (3.5) cannot be used. Alternately, homogenous transformation for the end-effector frame can be derived independently and post-multiplied to the forward kinematics equation afterwards.

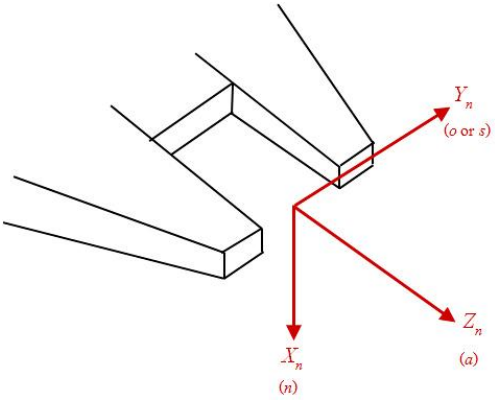


Figure 3.6 suggestion for end-effector frame

* For each two consecutive frames *i* and *i*-1, DH convention requires that x_i intersect and perpendicular to z_{i-1}

Step 5: Construct DH parameter table. After the frame attachment steps are completed, a table is created to contain the 4 DH parameters $\theta_i, d_i, a_i, \alpha_i$ for each link. An example in figure 3.7 illustrates how to determine the DH parameters. Figure 3.8 is used to find the correct sense of angles α_i and θ_i .This can be summarized as follows

- a_i: distance along x_i from o_i to the intersection of x_i and z_{i-1} (constant)
- d_i: distance along z_{i-1} from o_{i-1} to the intersection of x_i and z_{i-1} (variable for joint type (P)/ constant for (R))
- α_i: angle between z_{i-1} and z_i measured around x_i (constant, with sense according to Figure 3.8)
- θ_i: angle between x_{i-1} and x_i measured around z_{i-1} (variable for joint type (R)/constant for (P), with sense according to Figure 3.8)

Step 6: Substitute the DH parameters from Step 5 into (3.5)

Step 7: Compute forward kinematics ${}^0T_n = A_1 A_2 \dots A_n$

We illustrate the procedure by a few examples below.

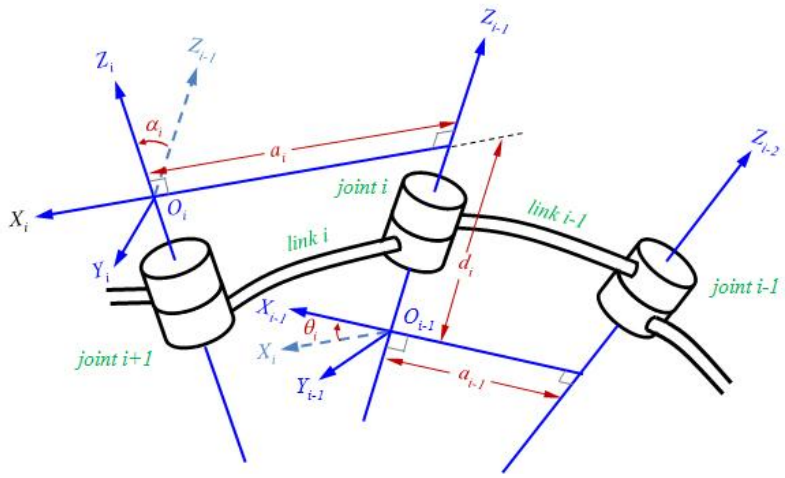


Figure 3.7 determine DH parameters from robot frame attachment

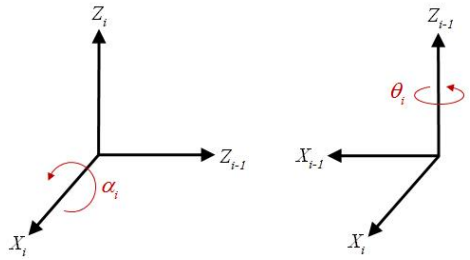
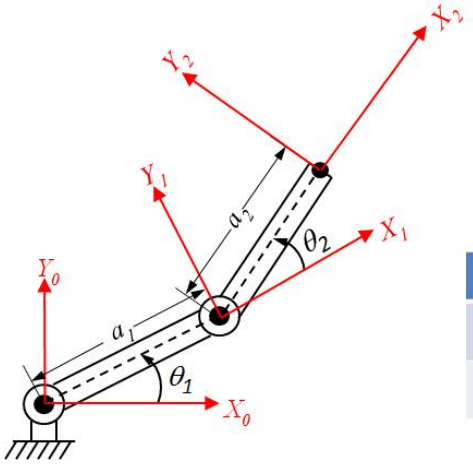


Figure 3.8 determine correct senses for a α_i and θ_i

Ex 3.1: Figure 3.9 shows DH frame attachment to a two-link manipulator that results from applying the above procedure. All z axes point out of the page so they are parallel (case 2 of step 3). The DH parameter table is shown on the right. It is customary to denote joint variables with *. From these values, we compute A_i by using (3.5)



link	θ_i	d_i	a_i	α_i
1	θ_1^*	0	a_1	0
2	θ_2^*	0	a_2	0

Figure 3.9 DH frames attached to a two-link manipulator

$$A_1 = \begin{bmatrix} c_1 & -s_1 & 0 & a_1c_1 \\ s_1 & c_1 & 0 & a_1s_1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \qquad A_2 = \begin{bmatrix} c_2 & -s_2 & 0 & a_2c_2 \\ s_2 & c_2 & 0 & a_2s_2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

and the corresponding forward kinematics equation

$$^0T_2 = A_1A_2 = \begin{bmatrix} c_{12} & -s_{12} & 0 & a_1c_1 + a_2c_{12} \\ s_{12} & c_{12} & 0 & a_1s_1 + a_2s_{12} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.6}$$

with abbreviations $c_1 = \cos(\theta_1)$, $s_1 = \sin(\theta_1)$, $c_{12} = \cos(\theta_1 + \theta_2)$, $s_{12} = \sin(\theta_1 + \theta_2)$.

Note that trigonometric identities are applied to achieve (3.6).

To construct a robot model from DH link parameters with RTSX is easy with commands `Link` and `SerialLink`. First a link data structure is created with successive calls to `Link`, with DH parameters for each link passed as a vector argument $[\theta_i, d_i, a_i, \alpha_i]$.

Let's illustrate this with the two-link manipulator. Assume $a_i = 1, i = 1,2$.

```
-->L(1)=Link([0 0 1 0]);
-->L(2)=Link([0 0 1 0]);
```

Remarks:

- It does not matter what initial values are passed to the joint variables θ_i. In this example we just assign 0 to them.
- Default joint type is (R). For a (P) joint, option `'p'` must be specified explicitly after the parameter vector.
- While most RTSX commands can be typed using all lowercase letters, `Link` is an exception. Typing `link` invokes a linker in Scilab that has nothing to do with RTSX at all. Doing so results in some error message that might leave you perplex.

After filling in DH parameters to all links, we assemble them with `SerialLink` command.

```
-->twolink=SerialLink(L,'name','2-link robot');
```

Robot name is optional. Now we have our 2-link robot model contained in a data structure `twolink`. Use `Robotinfo` command to display information

```
-->Robotinfo(twolink)

================= Robot Information =====================
Robot name: 2-link robot
Manufacturer: N/A
Number of joints: 2
Configuration: RR
Method: Standard DH
+---+-----------+-----------+-----------+-----------+
| j |   theta   |     d     |     a     |   alpha   |
+---+-----------+-----------+-----------+-----------+
| 1 |    q1     |    0.00   |   1.00    |   0.00    |
| 2 |    q2     |    0.00   |   1.00    |   0.00    |
+---+-----------+-----------+-----------+-----------+

Gravity =
    0.
    0.
    9.81

Base =
    1.    0.    0.    0.
    0.    1.    0.    0.
    0.    0.    1.    0.
    0.    0.    0.    1.

Tool =
    1.    0.    0.    0.
    0.    1.    0.    0.
    0.    0.    1.    0.
    0.    0.    0.    1.
```

which also shows information we have not yet discussed. At this point we focus on the upper part of information, particularly the DH parameter table, to verify that it is actually the desired two-link manipulator.

After basic data checking, we can visualize the `twolink` robot model with commands `PlotRobot` and `PlotRobotFrame`, which plot the symbolic diagram and coordinate frame configuration of the robot, respectively. Figure 3.10 shows the results from issuing the following commands

```
-->PlotRobot(twolink,[pi/3 pi/4]);
-->PlotRobotFrame(twolink,[pi/3 pi/4]);
```

Note that in addition to the robot model, joint variable values for the configuration must be passed as second argument.

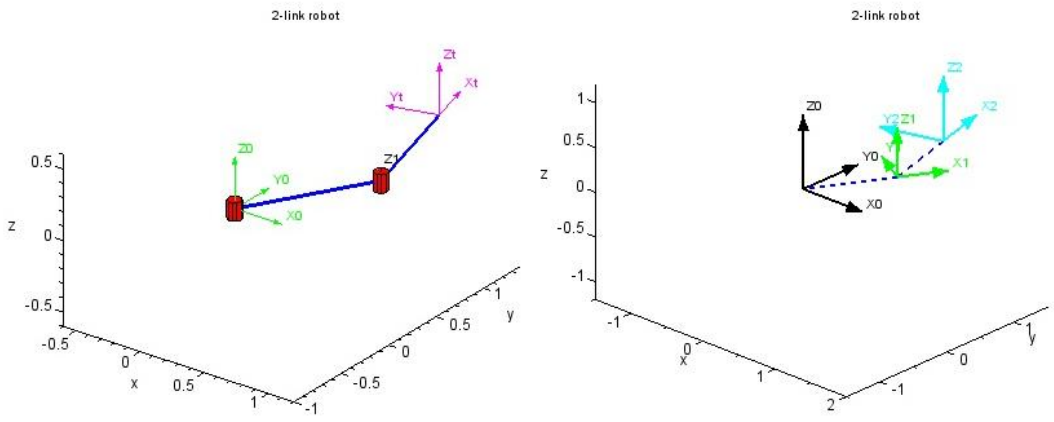
Instead of plotting each in separate window, we may choose to impose the coordinate frames onto the robot diagram by adding an option 'hold' .

```
-->PlotRobot(twolink,[pi/3 pi/4]);
-->PlotRobotFrame(twolink,[pi/3 pi/4],'hold');
```

To see how this robot moves corresponding to a series of joint variable values, use `AnimateRobot`. The joint variables must be passed as a matrix, with number of rows equal to number of data points for robot motion. Try this ad-hoc method to drive each joint of the two-link manipulator 360 degrees simultaneously.

```
-->AnimateRobot(twolink,[2*pi*(0:0.01:1)',2*pi*(0:0.01:1)']);
```

Of course, this only demonstrates a basic way to move a robot. In practice, we might want to command its motion using data from a command generation algorithm, which is the main topic of Chapter 4.



(a) PlotRobot (b) PlotRobotFrame

Figure 3.10 two-link manipulator plot results

For any robot, a forward kinematics solution of the end-effector frame w.r.t base at a given configuration is computed by RTSX command `fkine`. The desired configuration is chosen by passing joint variables as argument. For the two-link manipulator

```
-->fkine(twolink,[pi/3,-pi/2])
 ans  =
    0.8660254    0.5          0.    1.4660254
  - 0.5          0.8660254    0.    0.5392305
    0.           0.           1.    0.
    0.           0.           0.    1.
```

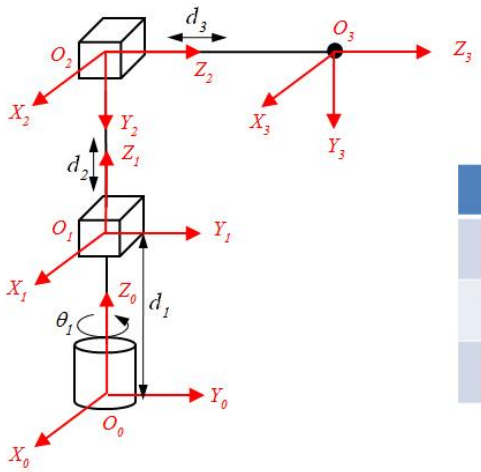
gives a solution at configuration $\theta_1 = 60^o$, $\theta_2 = -90^o$. One might want to verify by plugging these values into (3.6).

Though the two-link manipulator is a simple 2D planar robot that might look like a toy problem, most textbooks present it at the beginning of each topic for the reader to understand the fundamentals before moving on to more advanced structure. This document chooses to follow the same tradition.

Tips: Script files for various robot models are included in subdirectory ./models under the root of RTSX. Assume you are now at root directory and want to construct the two-link manipulator model, for example, simply issue the command

```
-->exec('./models/mdl_twolink.sce',-1);
```

Ex. 3.2: A cylindrical robot diagram with DH frame attachment and parameter table is shown in Figure 3.11. The corresponding forward kinematics solution can be computed as



link	θ_i	d_i	a_i	α_i
1	θ_1^*	d_1	0	0
2	θ_2^*	d_2^*	0	-π/2
3	0	d_3^*	0	0

Figure 3.11 DH frames attached to a cylindrical robot

$$A_1 = \begin{bmatrix} c_1 & -s_1 & 0 & 0 \\ s_1 & c_1 & 0 & 0 \\ 0 & 0 & 1 & d_1 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad A_2 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & -1 & 0 & d_2 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad A_3 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_3 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$^0T_3 = A_1 A_2 A_3 = \begin{bmatrix} c_1 & 0 & -s_1 & -s_1 d_3 \\ s_1 & 0 & c_1 & c_1 d_3 \\ 0 & -1 & 0 & d_1 + d_2 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad (3.7)$$

Assume that $d_1 = 1$, this set of commands construct a cylindrical robot

```
-->L(1)=Link([0 1 0 0]);
-->L(2)=Link([0 1 0 -pi/2],'p'); // 'p' indicates a prismatic joint
-->L(3)=Link([0 1 0 0],'p');
-->cylind_robot = SerialLink(L,'name','Cylindrical Robot');
```

Alternately, running `mdl_cylindrical.sce` provided in `./models` would give identical result. Plotting this robot at some configuration

```
-->PlotRobot(cylind_robot,[0,1,0.5]);
```

yields the guy shown in Figure 3.12. Prismatic joints are drawn with rectangular shapes and links attached to them are shown in yellow. Try `PlotRobotFrame` and compare DH frame attachment to the diagram in Figure 3.11. Finally, animate a motion with `AnimateRobot`. For example,

```
-->q1=pi*(0:0.01:1)';
-->q2=(0:0.01:1)';
-->q3=(0:0.02:2)';
-->AnimateRobot(cylind_robot,[q1,q2,q3])
```

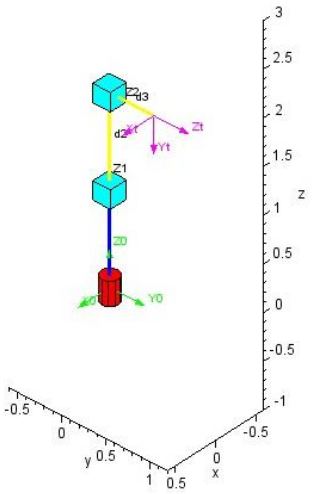


Figure 3.12 a cylindrical robot

As before, a forward kinematics solution can be computed using `fkine`. In some case, in addition to the relationship of end-effector frame w.r.t base, we may also want to know the values of each A_i matrix between each pair of joints. RTSX has a specific command `Robot2AT` for such purpose.

```
-->[A,T]=Robot2AT(cylind_robot,[pi/4, 0.5, 0.8]);
```

returns a hyper-matrix `A` of size 4 x 4 x n, where A_i is contained in `A(:,:,i)`. `T` is the forward kinematics solution identical to the one returned by `fkine`.

Ex 3.3: PUMA (Programmable Universal Machine for Assembly) from Unimation are well-known robots and often used as examples in many textbooks, especially PUMA 560 model that closely mimics a human arm. It can be described as an articulated robot (Figure 1.4) equipped with a spherical wrist. This makes it a 6 DOF robot with 6 joints of type (R). An important advantage that makes it a prime example is its inverse kinematics can be solved analytically (more on this later).

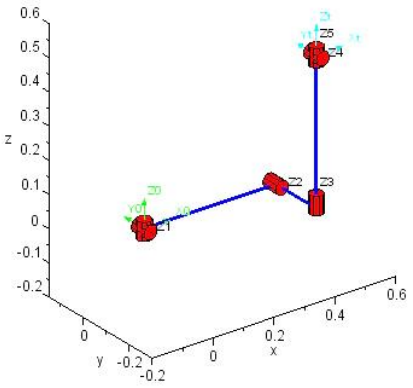


Figure 3.13 a PUMA560 robot

Figure 3.13 shows a PUMA560 robot that can be created by running the script

```
-->exec('./models/mdl_puma560.sce',-1);
```

This script file also initializes some variables for commonly-used configurations, such as `q_z = [0,0,0,0,0,0]` (zero), `q_r = [0,pi/2,-pi/2,0,0,0]` (ready), `q_s = [0,0,-pi/2,0,0,0]` (stretch), and `q_n = [0,pi/4,-pi,0,pi/4,0]` (nominal). The robot diagram in Figure 3.13, for example, is in `q_z` configuration, which is plotted by

```
 -->PlotRobot(p560, q_z);
```

where p560 is the robot model created from the script file. Forward kinematics at this configuration can then be computed as

```
-->T=FKine(p560, q_z)
 T  =
    1.    0.    0.    0.4521
    0.    1.    0.  - 0.15005
    0.    0.    1.    0.4318
    0.    0.    0.    1.
```

Furthermore, in this example we also want to demonstrate how to adjust the end-effector and/or base frame. First, notice in the original p560 in Figure 3.13 that the

tool frame is at the intersection of the 3 wrist axes, which is inside the wrist. In practice, the tool frame can be moved outside by passing to `AttachTool` a homogeneous matrix that describes the offset of the tool

```
-->p560 = AttachTool(p560,transl([0, 0, 0.1]));
```

This moves the tool frame 0.1 meters along z_5. Likewise, a base transformation can be performed using AttachBase command. Suppose we want to mount our `p560` robot on a 3-meters high ceiling. Issuing the following commands

```
-->p560=AttachBase(p560, transl([0,0,3])*trotx(pi));
-->PlotRobot(p560,q_z);
```

yields the plot in Figure 3.14. To summarize, `AttachBase` and `AttachTool` are convenient and safe commands to adjust the base and tool frames to user needs. We advice against directly editing any field in a robot data structure such as `p560.base` or `p560.tool` because such practice is error-prone. Note also that while FKine takes the base and tool transformation into account, Robot2AT does not. Finally, to remove the added base and tool transformation

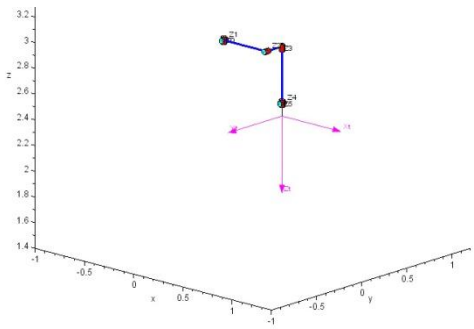


Figure 3.14 p560 with base and tool transformation.

```
-->p560=DetachBase(p560);
-->p560=DetachTool(p560);
```

Tips: The size of a robot diagram displayed by `PlotRobot` is adjusted automatically in proportion to its workspace. Use *zoom* function in Scilab graphic window to expand the part you want to examine up close.

3.2 Inverse Kinematics

The above discussion covers only the forward kinematics analysis of a robot. In this part we consider the other way around; i.e., *inverse kinematics* involves finding values of joint variables $q_0,\ldots q_n$ given a homogeneous matrix from tool to base ${}^0T_n$. This problem is harder to solve in general, because there are 3 possibilities

- unique solution
- no solution
- multiple solutions

depending on the location and orientation of the end-effector described by the given ${}^0T_n$. For example, if it is outside the robot workspace, no solution exists. Multiple solution case happens inside the workspace when more than one configurations lead to the same tool location and orientation.

Another issue that complicates an inverse kinematics problem of some robots even more is the feasibility to find a closed-form solution. One might have to solve the problem using a numerical method, which takes more time depending on how many iterations needed for a given configuration. If a closed-form solution exists, there is no systematic way to tackle the problem since it depends on the robot type that one typically has to analyze using algebraic/geometric means. Procedure for even a simple case like a two-link manipulator could be quite involved. See [Craig05] for details. In this document we omit inverse kinematics analysis and focus on the use of RTSX commands. The examples closely resemble those given in Chapter 7 of [Corke11].

PUMA560 model is used exclusively in this part because the existence of closed-form solution. Create the `p560` model from `mdl_puma560.sce` as in Ex.3.3. Consider forward kinematics solution ${}^0T_6$ at nominal configuration `q_n`

```
-->T=FKine(p560,q_n)
 T  =
    0.    0.    1.    0.5963031
    0.    1.    0.  - 0.15005
  - 1.    0.    0.  - 0.0143543
    0.    0.    0.    1.
```

`IKine6s` is a specific RTSX command to compute inverse kinematics of 6 DOF robot with a spherical wrist like PUMA560. Given the above ${}^0T_6$

```
-->q_i=IKine6s(p560,T)
 q_i  =
    2.6485  - 3.9269    0.0939    2.5325    0.9743    0.3733
```

Observe that `IKine6s` gives an inverse kinematics solution `q_i`, which is different from `q_n`. However, by computing forward kinematics from `q_i`

```
-->FKine(p560,q_i)
 ans  =
    0.    0.    1.    0.5963031
    0.    1.    0.  - 0.15005
  - 1.    0.    0.  - 0.0143543
    0.    0.    0.    1.
```

we see that it gives identical ${}^0T_6$ as q_n. What happens is `q_n` and `q_i` are joint variables of the so-called "left-arm" and "right-arm" configuations of the PUMA560 robot, respectively. This is plotted in Figure 3.15 by using 'hold' option to impose the second plot onto the same window

```
-->PlotRobot(p560,q_n);
-->PlotRobot(p560,q_i,'linklinestyle','.','hold');
```

Notice that the links of robot with configuration `q_i` are shown as dotted lines. As expected, the two different configurations share the same tool frame {6}.

We can force IKine6s to return joint variables for a particular configuration by by specifying an option. For example,

```
-->q_i=IKine6s(p560,T,'ru')
 q_i  =
    0.    0.7853    3.1415    0.    0.7853    0.
```

qives an answer identical to q_n. There are eight possibilities for PUMA560 that could result from a combination of the following options

- `'l','r'` left or right arm
- `'u','d'` elbow up or down
- `'f','n'` wrist flipped or not

From above, `'ru'` instructs `IKine6s` to give a "right arm, elbow up" solution.

As stated earlier, there is no inverse kinematics solution when the given tool frame is selected outside the robot workspace.

```
-->IKine6s(p560, transl([3,0,0]))
WARNING: Puma 560 -- point not reachable
 ans  =
    Nan    Nan    Nan    Nan    Nan    Nan
```

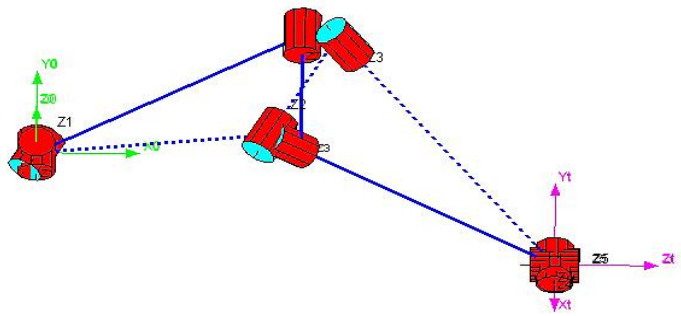


Figure 3.15 robot configurations from joint variables `q_n` and `q_i`

For some robots where a closed-form inverse kinematics solution does not exist, RTSX as a command IKine to compute an answer numerically. This is an iterative algorithm that loops until achieving a solution within some tolerance limit. When running this command, a loop counter is displayed. Let's try it with the PUMA560 so that we could compare the result with the closed-form counterpart.

```
-->T=FKine(p560,q_n)
T  =
    0.    0.    1.    0.5963031
    0.    1.    0.  - 0.15005
  - 1.    0.    0.  - 0.0143543
    0.    0.    0.    1.
-->q_i = IKine(p560,T)
Computing inverse kinematics for robot Puma 560
393
 q_i  =
  - 0.0000  - 0.8335    0.0940    0.0000  - 0.8312  - 0.0000
```

This might be a good time to grab a beer, since the numerical process takes some time depending on how ancient your computer is. The total number of loops executed is 393 before the answer pops up. Again, we get another different configuration from `q_n`. Verify that this is also a valid answer

```
-->FKine(p560,q_i)
 ans  =
    0.0000001  - 0.0000002    1.           0.5963039
  - 5.475D-08    1.           0.0000002  - 0.1500505
  - 1.         - 5.475D-08    0.0000001  - 0.0143543
    0.           0.           0.           1.
```

By issuing `PlotRobot(p560,q_i)`, we get the robot diagram shown in Figure 3.16, which is in an elbow-down configuration. Unlike the closed-form version `IKine6s` , there is no explicit way to specify a particular configuration with `IKine`. However, we can help the algorithm to converge to a configuration by suggesting an initial set of joint variables (from default values of all zeros).

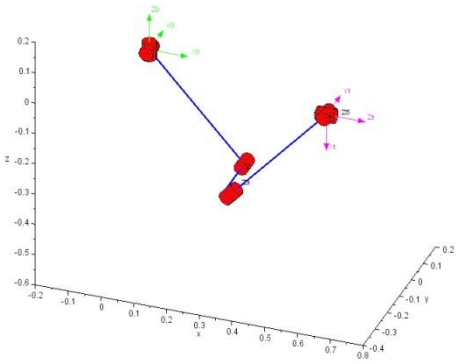


Figure 3.16 `p560` in an elbow-down configuration

Run `IKine` again with initial joint variables `q0 = [0,0,3,0,0,0]`.

```
-->q_i = IKine(p560,T,'q0',[0 0 3 0 0 0])
Computing inverse kinematics for robot Puma 560
378
 q_i  =
  - 0.0000    0.7853    3.1415  - 0.0000    0.7853    0.0000
```

We see that this time the answer equals `q_n`.

3.3 Velocity Kinematics

The last part of robot kinematics analysis in this chapter concerns with relationships between joint and end-effector velocities, termed as a *velocity kinematics* problem. The key math element for this purpose is called a Jacobian, which can be thought of as matrix equivalent of derivative. Essentially, we have

$$\xi = J(q)\dot{q} \tag{3.8}$$

where $\xi = [v_x, v_y, v_z, \omega_x, \omega_y, \omega_z]^T \in \mathbb{R}^6$ represents translational and rotational velocity components of the end-effector, $\dot{q}$ is a vector of joint velocities, and the matrix $J(q) \in \mathbb{R}^{6\times n}$ is the manipulator Jacobian, or the geometric Jacobian. See, for example, [SHV06], for detailed derivation and analysis of Jacobian matrix.

A Jacobian marix that represents a relation between spatial velocity of the tool frame and the joint velocity vector of a robot can be computed by command `jacob0`. We illustrate this with an example.

Ex 3.4: Figure 3.17 shows a cylindrical robot diagram with DH frames attached. A Jacobian matrix for this robot can be described as

$$J = \begin{bmatrix} -s_1 s_2 d_3 & c_1 c_2 d_3 & c_1 s_2 \\ c_1 s_2 d_3 & s_1 c_2 d_3 & s_1 s_2 \\ 0 & s_2 d_3 & -c_2 \\ 0 & s_1 & 0 \\ 0 & -c_1 & 0 \\ 1 & 0 & 0 \end{bmatrix} \tag{3.9}$$

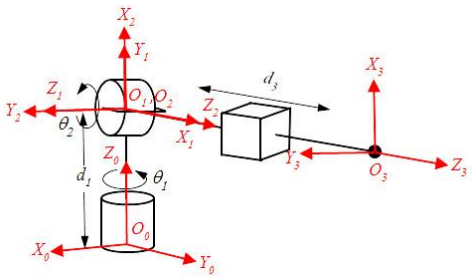


Figure 3.17 A cylindrical robot

Construct this robot model by RTSX.

```
-->L(1)=Link([0 1 0 pi/2]);     // let d1 = 1
-->L(2)=Link([0 0 0 pi/2]);
-->L(3)=Link([0 1 0 0],'p');
-->sphrobot=SerialLink(L,'name','Spherical Robot');
```

and use `jacob0` tocompute Jacobian at $\theta_1 = \pi/4,\ \theta_2 = \pi/3,\ d_3 = 1.5$

```
-->J = jacob0(sphrobot,[pi/4, pi/3, 1.5])
 J  =
  - 0.9185587    0.5303301    0.6123724
    0.9185587    0.5303301    0.6123724
    0.           1.2990381  - 0.5
    0.           0.7071068    0.
    0.         - 0.7071068    0.
    1.           0.           0.
```

Compare this with (3.9) to verify that they give identical result.

```
-->th1=pi/4; th2=pi/3; d3=1.5;
-->s1=sin(th1); c1=cos(th1); s2=sin(th2); c2=cos(th2);
-->Ja=[-s1*s2*d3, c1*c2*d3, c1*s2; c1*s2*d3, s1*c2*d3, s1*s2;
-->0, s2*d3, -c2;0, s1,0; 0, -c1, 0; 1, 0, 0]
 Ja  =
  - 0.9185587    0.5303301    0.6123724
    0.9185587    0.5303301    0.6123724
    0.           1.2990381  - 0.5
    0.           0.7071068    0.
    0.         - 0.7071068    0.
    1.           0.           0.
```

Tips:

- `jacob0` maps joint velocity to spatial end-effector velocity expressed w.r.t world coordinate frame. To obtain spatial velocity w.r.t end-effector frame, use the command `jacobn` instead.
- To compute an analytical jacobian [SHV06,Corke11], add an argument `'eul'`. For example `jacob0(p560,q_n,'eul')`

3.3.1 Robot Singularities and Manipulability

In brief, singularity in a robot is a configuration that constrains its movement in some direction. This normally happens at the workspace boundary, though it could happen somewhere else. Mathematically speaking, singularity refers to a configuration that makes the Jacobian $J(q)$ singular. As a simple example, consider linear velocity in XY plane resulting from two joint variables

$$\begin{bmatrix} \dot{x} \\ \dot{y} \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} \dot{q}_1 \\ \dot{q}_2 \end{bmatrix} \tag{3.10}$$

Obviously, $\dot{y} = 0$ independent of the joint velocities. Hence, at this configuration, movement along the *y* axis is not possible. For the general case of (3.8), singularity at a configuration is determined by checking whether the corresponding Jacobian matrix has full rank, which is constrained by $\text{rank } J \le \min(6, n)$. For a two-link manipulator, $\text{rank } J \le 2$, while for PUMA560, $\text{rank } J \le 6$, for example. Hence singularities of the PUMA560 occur at configurations that cause $\text{rank } J < 6$.

This example below follows Section 8.1.4 of [Corke11], where it is discussed in more detail there. Start from creating a `p560` robot as before.

```
-->exec('./models/mdl_puma560.sce',-1);
```

We want to examine this robot at its "ready" configuration q_r, which is also created by the script file.

```
-->q_r
 q_r  =
    0.    1.5707963  - 1.5707963    0.    0.    0.
```

It can be shown by analysis that, at this configuration, joint 4 and 6 in the spherical wrist are aligned, so the robot loses one degree of freedom. Verify this from the Jacobian

```
-->J = jacob0(p560,q_r)
 J  =
    0.15005  - 0.8636  - 0.4318    0.    0.    0.
    0.0203     0.        0.        0.    0.    0.
    0.         0.0203    0.0203    0.    0.    0.
    0.         0.        0.        0.    0.    0.
    0.       - 1.      - 1.        0.  - 1.    0.
    1.         0.        0.        1.    0.    1.
```

by computing its rank

```
-->rank(J)
 ans  =
    5.
```

As stated above, `p560` should have full rank 6. So this demonstrates a singularity as expected. In fact, a robot might be in trouble already at a configuration close enough to singularity. To investigate this, select a new set of joint variables with slight deviation from `q_r`, say, with a small angle of 5 degrees at joint 5.

```
-->q1 = q_r;
-->q1(5) = 5*pi/180;
-->q1
 q1  =
    0.    1.5707963  - 1.5707963    0.    0.0872665    0.
```

Compute the Jacobian at this new configuration

```
-->J = jacob0(p560, q1);
```

Assume that the end-effector moves slowly with velocity along *z* axis 0.1 meter/sec. We can solve inverse velocity kinematics problem

$$\dot{q} = J^{-1}\xi \tag{3.11}$$

```
-->qd = inv(J)*[0 0 0.1 0 0 0]';
-->qd'
 ans  =
    0.  - 4.9261084    9.8522167    0.  - 4.9261084    0.
```

and see that the elbow (joint 3) has to move at a very high speed of 9.85 rad/s (or 565 degrees/sec). Even though `p560` is not exactly at a singular configuration, the determinant of Jacobian is very small

```
-->det(J)
 ans  =
  - 0.0000155
```

We can also compute the condition number of Jacobian

```
-->cond(J)
 ans  =
    235.24979
```

whose high value indicates that the matrix is poorly-conditioned. Nevertheless, at this same configuration some different movement might not experience any problem at all. For example, suppose we want to move the tool frame around axis *y* at rate 0.2 rad/sec

```
-->qd = inv(J)*[0 0 0 0 0.2 0]';
-->qd'
 ans  =
    0.    0.    0.    0.  - 0.2    0.
```

This results in very small joint rate. This paves the way to the concept of robot manipulability. Consider a set of joint velocity with unit norm $\dot{q}^T\dot{q}=1$. By substituting into (3.11), we get

$$\xi^T(JJ^T)\xi=1 \tag{3.12}$$

This equation expresses the joints on surface of 6-dimensional ellipsoid in the end-effector velocity space. A small radius indicates the end-effector cannot achieve velocity in that direction.

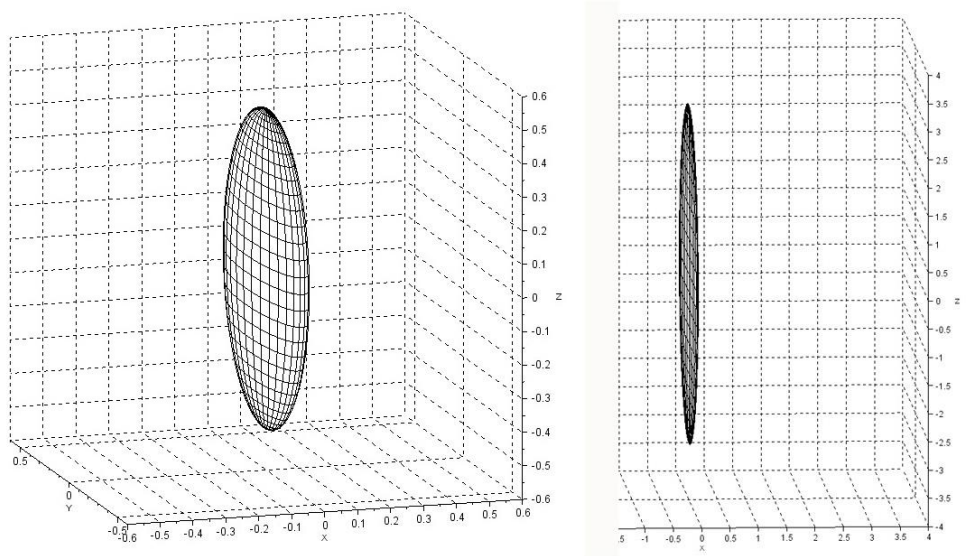
To demonstrate with the p560 robot at the nominal configuration `q_n`, consider only the linear velocity part

```
-->J=jacob0(p560,q_n);
 -->Jv=J(1:3,:);
```

We can plot the velocity ellipsoid with the command

```
-->plot_ellipse(Jv*Jv')
```

that yields Figure 3.18 (a). The resulting ellipsoid has small radius in *x* axis. This implies the velocity is more limited along the *x* direction than in *y* and *z* directions.



(a) linear velocity at `q_n` (b) angular velocity at `q_r`

Figure 3.18 end-effector velocity ellipsoid of `p560` robot

Likewise, we examine angular velocity at the singular configuration `q_r`

```
-->J=jacob0(p560,q_r);
-->Jw=J(4:6,:);
-->plot_ellipse(Jw*Jw')
```

and see from Figure 3.18 (b) that the resulting ellipsoid almost has no thickness in *x* axis. This indicates the end-effector rotation around *x* axis of the base frame is constrained. `PlotRobot` can be used to verify that, at this configuration, rotation around *x* axis is indeed impossible by any joint movement.

Perhaps a more convenient way to determine limitations of robot movements is to get a number that measures how close the ellipsoid to a sphere, the desired shape. The `maniplty` command in RTSX computes Yoshikawa method of robot manipulability

$$m = \sqrt{\det\left(JJ^T\right)} \tag{3.13}$$

which is proportional to the volume of a sphere. We demonstrate by observing `p560` manipulability at configuration `q_r`

```
-->maniplty(p560,q_r,'yoshikawa')
 ans  =
    0.
```

This conforms to our earlier analysis that the robot has poor manipulability at this pose. While at `q_n`

```
-->maniplty(p560,q_n,'yoshikawa')
 ans  =
    0.0786172
```

`p560` achieves better manipulability.

Problems

3-1 Apply DH convention to derive forward kinematics equation for the two-link manipulator in Figure 3.19

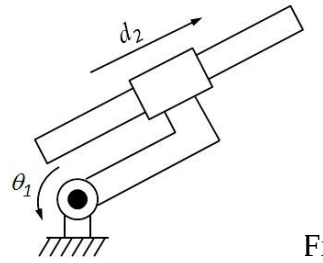


Figure 3.19 two-link planar manipulator for Problem 3-1

3-2 Apply DH convention to derive forward kinematics equation for the articulated (RRR) robot in Figure 3.20

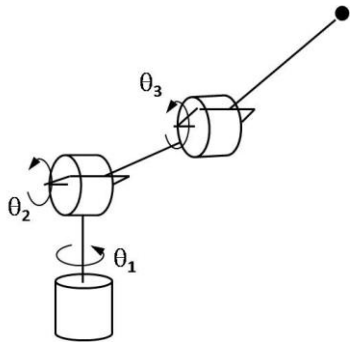


Figure 3.20 the articulated (RRR) robot for Problem 3-2

3-3 Derive forward kinematics equation for the spherical wrist in Figure 3.21

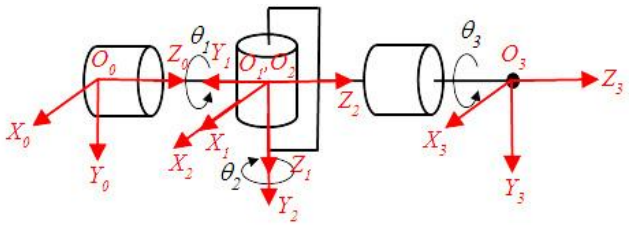


Figure 3.21 a spherical wrist for Problem 3-3

3-4 Verify the answers in Problem 3-1 to 3-3 using RTSX.

3-5 Find inverse kinematics solutions of Problem 3-1 to 3-3 and verify with RTSX.

3-6 Analyze velocity kinematics of the robots in Problem 3-1 to 3-3. Determine their singular configurations and use RTSX to verify your answers.

3-7 Attach DH frames to the robot in Figure 3.22 and derive Jacobian w.r.t base and end-effector frames. Verify with commands `jacob0` and `jacobn`, respectively.

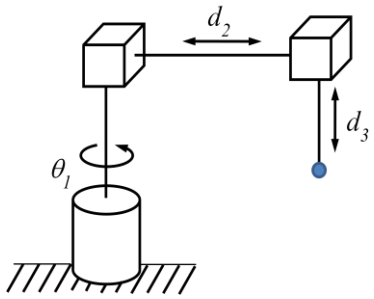


Figure 3.22 robot for Problem 3-7