1 Introduction

The word "robot" came from Czech "robota," meaning hard work or slavery. When talking about a robot, general audience may picture some humanoid type seen in movies, like the Terminator (who took control of California for some time), or a real walking robot ASIMO from Honda. Students perhaps think about mobile robots with wheels, wings, or rotor blades that could move and follow a set of rules in a competition. There are nano-robots small enough to get into our arteries and do some medical work like removing clogging. Nevertheless, in this book we focus only on one type, known as robot manipulator, or simply robot arm. Generally speaking, this is just an automated mechanical device consisting of links and joints to mimic the movement of human arm.

According to formal definition by The Robotic Industries Association (RIA), "A robot is a *reprogrammable*, *multifunctional* manipulator designed to move material, parts, tools or specialized devices through variable programmed motions for the performance of a variety of tasks." The keywords "reprogrammable," and "multifunctional" makes a robot differ from other types of automated machines like CNC milling machine, which is limited in functionality. We cannot use a CNC milling to perform arc welding task, say.

1.1 Robot Arm Structure

Figure 1.1 shows typical mechanical components of a robot manipulator. Joints are where actuators are installed to command arm movement. They are connected by links into a particular robot configuration. Compared to human anatomy, links correspond to upper and lower arms and joints are wrist, elbow, shoulder, and waist. The arm is then attached to a base, which may be anchored on the floor, wall, ceiling, or even mobile. The end-effector is the part where desired operation is carried out; i.e., where a tool is mounted. The figure shows a gripper type though it can be a spot-welding or stray painting tool, or else.

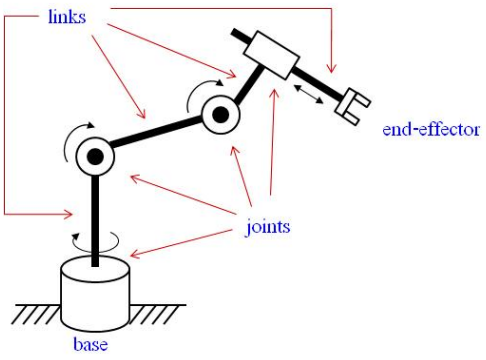

Figure 1.1 main components of a robot

A typical robot joint can be one of the following

- (R) revolute joint
- (P) prismatic joint

In this book we will use symbol (R) and (P) to represent joint types. Type (R) is a joint that rotates around its joint axis. The circular movement is natural just like our elbow and the joint can be driven directly by an electric motor. In contrast, joint type (P) moves along a straight line, so some transmission is needed for the joint to be driven by a rotating machine. Figure 1.2 shows a robot diagram used in this book. Revolute and prismatic joints are drawn using cylindrical and rectangular shapes, respectively.

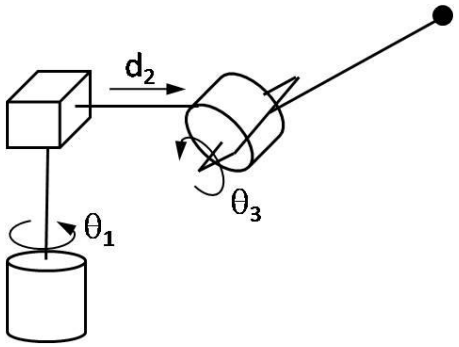

Figure 1.2 symbolic robot diagram

In Figure 1.2 we also see the displacement of each joint is indicated by a joint variable, where θ_i and d_i represent joint angle and linear distance, respectively. When talking about a robot joint without specifying its type, we may use q_i as a generalized joint variable. On each joint we should be able to indicate the *joint axis*. For (R), joint axis is the axis of rotation. For (P), joint axis is aligned with the linear movement of that joint.

A common way to specify a robot configuration is by its joint type from base to top. The robot in Figure 1.2, for example, can be called an RPR robot. This name convention can be ambiguous sometimes from the fact that a joint axis can be placed parallel, perpendicular, or at any angle with another joint axis.

1.2 Coordinate Systems and Frames

A fundamental requirement for robot analysis is to represent the position and orientation of objects in some environment. Normally an object is assumed to be a rigid body. So a robot arm is nothing more than as set of rigid objects connected from base to tool. The movement of a part can then be described as position and orientation in a coordinate system. Cartesian coordinate, with standard mutually perpendicular XYZ axes is the most used.

The basic procedure that one must be familiar with in robotic study is to attach a set of XYZ axes, called *coordinate frames*, or *pose,* to each joint of the robot, like shown in Figure 1.3 for an RPR robot. Then a kinematic problem might ask to find an expression of tool position and orientation w.r.t (with respect to) the base frame, or w.r.t the world frame. The problem could be complicated if the adjacent frames are attached randomly, so some rules are invented, such as the Denavit-Hartenberg convention discussed in Chapter 3.

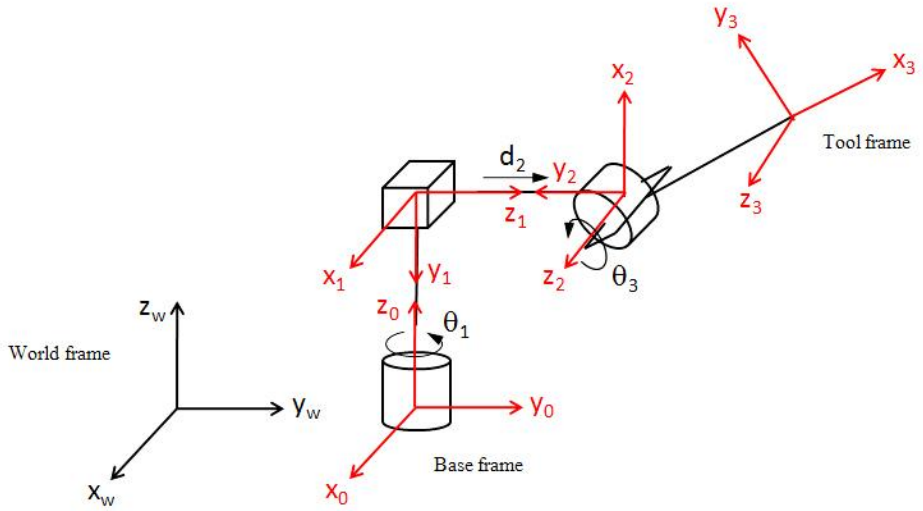

Figure 1.3 Coordinate frames attached to robot parts

1.3 Some Common Robot Manipulators

In this section we list a few robot configurations normally used in industrial applications.

1.3.1 Articulated Robot (RRR)

This robot shown in Figure 1.4 is also called a revolute, elbow, or anthropomorphic. The joint axes from base to tool are designated as waist, shoulder, and elbow, and the links are body, upper arm, and forearm, respectively. This is one of the most widely-used robots in industry. Examples are PUMA 560 from Unimation, IRB1400 from ABB.

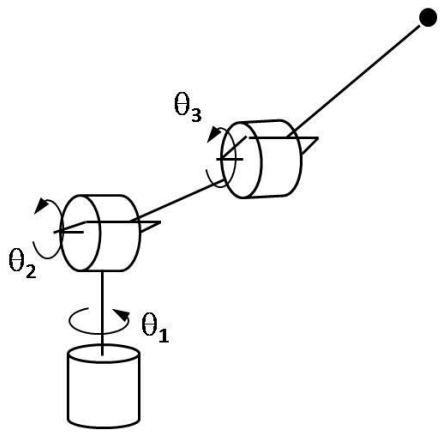

Figure 1.4 articulated robot (RRR)

1.3.2 Spherical Robot (RRP)

Quite often that a robot is named from the shape of workspace its end-effector could span. Figure 1.5 shows a symbolic diagram of a spherical robot that has RRP joint configuration. A well-known example of spherical robot is the Standford Arm.

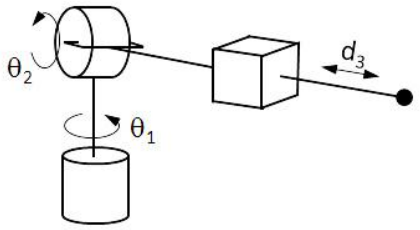

Figure 1.5 spherical robot (RRP)

1.3.3 SCARA Robot (RRP)

Though SCARA (Selective Compliant Articulated Robot for Assembly) robot in Figure 1.6 also has RRP configuration, the arrangement differs from a spherical robot. The three joint axes of SCARA robot are mutually parallel. The design aims for table top assembly and pick-and-place applications.

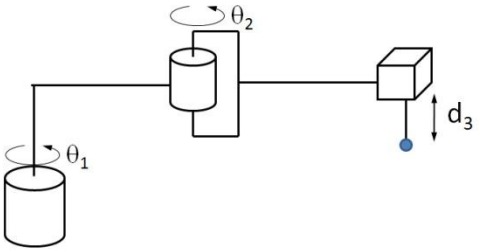

Figure 1.6 SCARA robot (RRP)

1.3.4 Cylindrical Robot (RPP)

Figure 1.7 shows a symbolic diagram of a cylindrical robot. The first joint is revolute and produces a rotation about the base, while the second and third joints are prismatic. it has cylindrical shape workspace, hence the name. Cylindrical robots are often used in materials transfer tasks. Some example is RT3300 Robot made by Seiko.

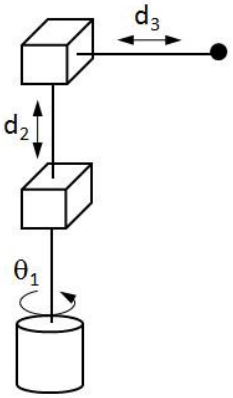

Figure 1.7 cylindrical robot (RPP)

1.3.5 Cartesian Robot (PPP)

A Cartesian robot has all three joints of prismatic type, as shown in Figure 1.8. The joint variables of this robot are the Cartesian coordinates of the end effector w.r.t the base. That renders its workspace a rectangular shape. Cartesian robots are useful for assembly, pick and place operations, and for transfer of material or cargo. The design allows increased structural rigidity and hence higher precision.

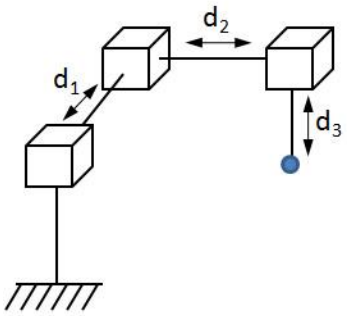

Figure 1.8 Cartesian robot (PPP)

1.4 Getting Started with Scilab and RTSX

In the past, robot study had to be done with pencil and paper. That was quite a challenge for students. The analysis and illustrations were often restricted to a simple robot configuration known as a planar two-link manipulator in Figure 1.9. Nowadays, while many books still like to use this two-link robot as the first example, the computation can be easily extended to more complicated robots with the help of suitable computer software.

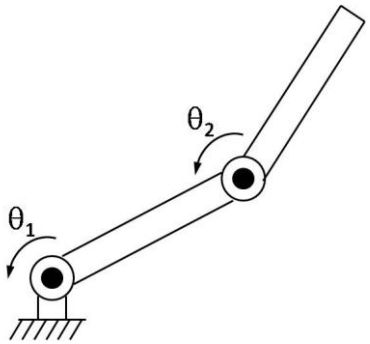

Figure 1.9 a two-link planar robot

The main purpose for this book is to illustrate how to use Scilab and RTSX as study aids for robotics. We assume the reader has some familiarity with Scilab, since there is no way to cover the details from the ground up. Just like other open source package, the software has community support where you could post your question and get help. Below we describe Scilab usage in a nutshell.

1.4.1 Scilab basics

If you have not done so, go to www.scilab.org and download the latest release for your operating system (version 5.4.1 by the time of this writing), and install with all the default options. Then launch it by clicking on the icon on your desktop, you should see the working environment as shown in Figure 1.10

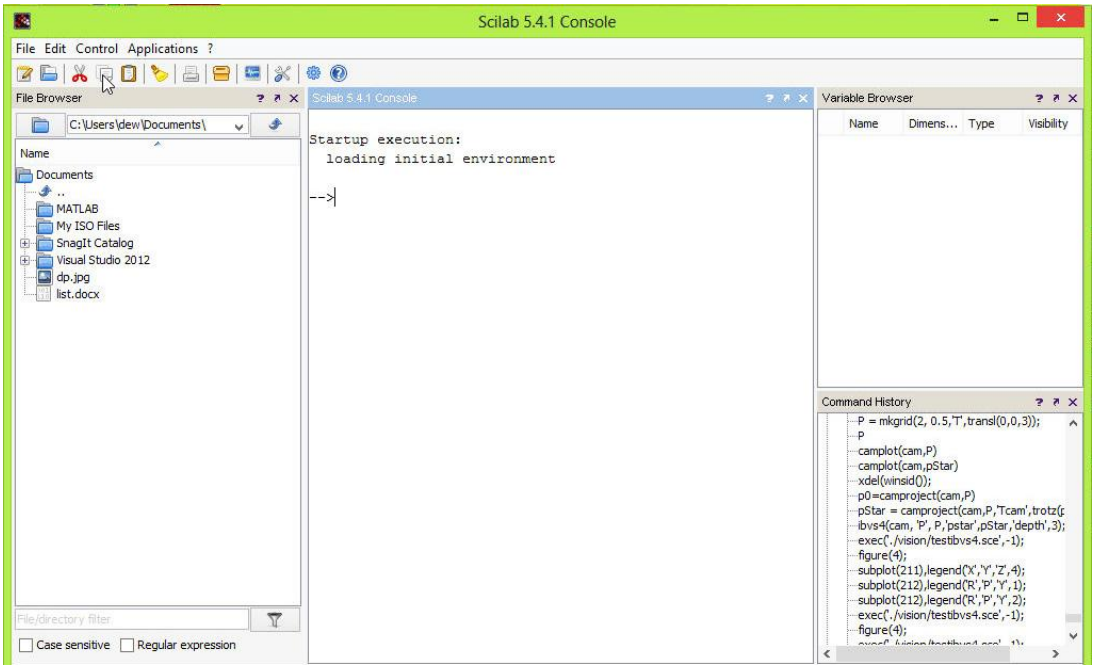

Figure 1.10 Scilab main console

Note: All the figures in this book corresponds to MS Windows version of Scilab, though the appearance and functionality among all OS versions are quite similar.

The Scilab console consists of 4 sub-windows. File Browser in the leftmost shows all the files in current directory, where you can click to edit a file (more on this later). You can also change working directory using the combo box or buttons on the upper tab. The upper right window is the Variable Browser, a handy tool to examine the values of all variables in Scilab workspace. The lower right window is Command History; i.e., it keeps all the commands you typed in a session. The window of most interest is the middle one, where we will be dealing with it interactively by typing a Scilab command at the `-->` prompt.

As an example, try some simple commands to create a matrix and compute its inverse

```
-->A = [1 2 -1;2 0 4;-2 3 1]
 A  =
    1.    2.  - 1.
    2.    0.    4.
  - 2.    3.    1.
```

```
-->inv(A)
 ans  =
    0.3157895    0.1315789  - 0.2105263
    0.2631579    0.0263158    0.1578947
  - 0.1578947    0.1842105    0.1052632
```

and see that Scilab returns the answer immediately. For some command that you don't want a response, end the line with ; to tell Scilab to shut up.

This kind of interactive dialog is handy when we want to compute a simple math problem. For a more complicated situation that involves a long command sequence, it is more convenient to put the commands in a script file. Scilab comes with its editor named SciNotes, shown in Figure 1.11 which you can launch by clicking on the Applications menu.

Scilab has its own syntax for functions and control statements, but basically the structure is not different from other programming language. There is no way you can remember all the command usage; that's what the ? menu is for. Alternatively, you can type help follow by the command in doubt, such as `--> help norm` for information on matrix norm function.

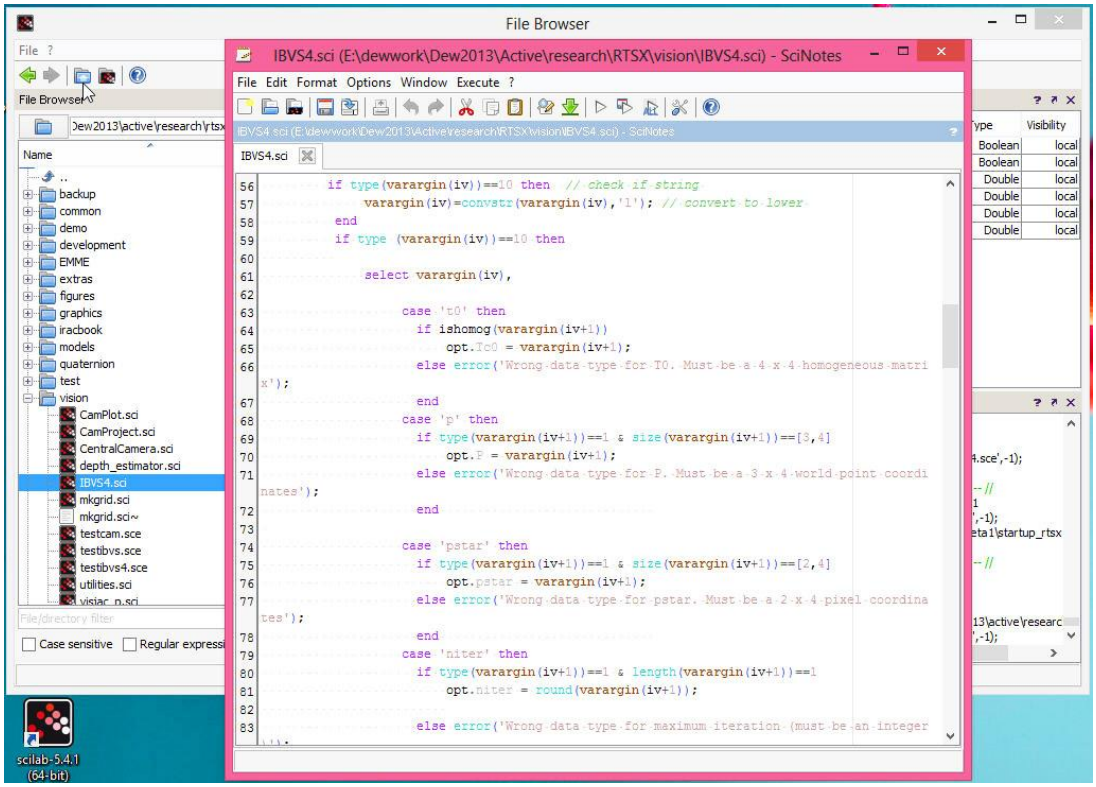

Figure 1.11 SciNotes window for writing a Scilab script/function

There are typically two types of application you would write using Scilab commands: a script, and a function. It is a good practice to save a script using `.sce` ,and a function using `.sci` extension, as Scilab suggests. Both can be called with exec command. For example, suppose you save a file as test.sce, then execute it by typing

```
--> exec('test.sce');
```

or if you prefer not to see the responses from each individual command in the file

```
--> exec('test.sce',-1);
```

A Scilab script is simply a sequence of commands to perform a particular task, which will be executed immediately when you run it by exec command. A Scilab function, in contrast, is only loaded into workspace and can be used afterwards just like any original Scilab command by typing its name and passing argument(s), if any.

To implement a function, put all commands and control statements between `function` and `endfunction` keywords. Let's say you want to build a function `rad2deg` to convert an angle from radian to degree. Put the following code into new SciNotes window

```
// rad2deg.sci convert radian to degree
function deg=rad2deg(rad)
    deg = rad*180/%pi;
endfunction
```

Note that anything after // is a comment. Save the file as rad2deg.sci and then load the function

```
-->exec('rad2deg.sci');
```

Then you can test your new command `rad2deg`

```
-->rad2deg(1.5)
 ans  =
    85.943669
-->rad2deg(%pi/3)
 ans  =
    60.
```

Note: As you may guess, RTSX is just a bunch of functions implemented this way. So you can examine any function and perhaps customize it as you like. Just don't forget to load your modified version with `exec` before use.

1.4.2 Simulation by Xcos

Xcos is the simulation subsystem included in the Scilab package. It is launched by typing xcos at Scilab prompt. A blank window with grid appears where you can create a model from supplied Xcos palettes (or build your own, for an advanced user). Figure 1.12 shows how to construct a simple PID feedback control diagram. What you normally need is some input and activation clock from Sources, output block from Sinks, and, of course, block(s) that describes your system. Connect all the ports together to build a complete diagram. Set up the parameters in each block and in Simulation Setup menu. When finish, choose Simulation – Start (or click the Start button). You'd get a step response like shown in Figure 1.13

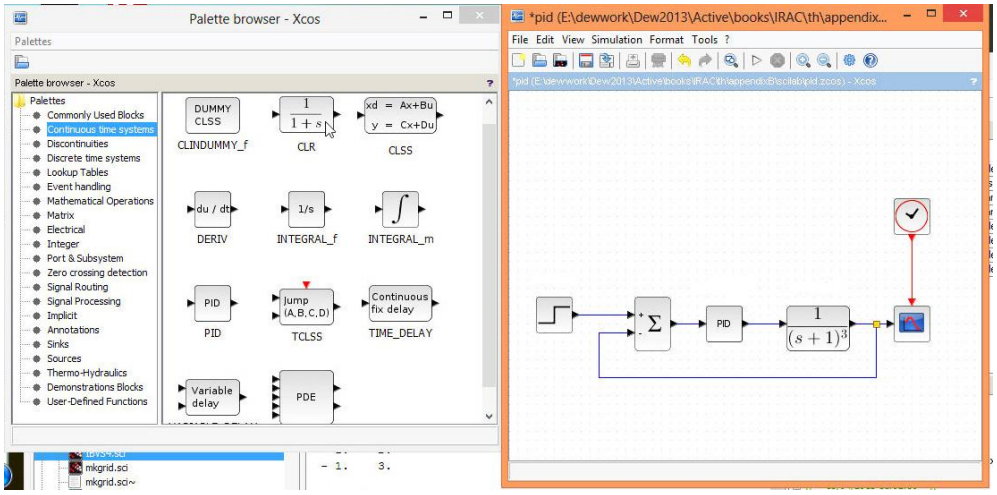

Figure 1.12 PID feedback diagram in Xcos

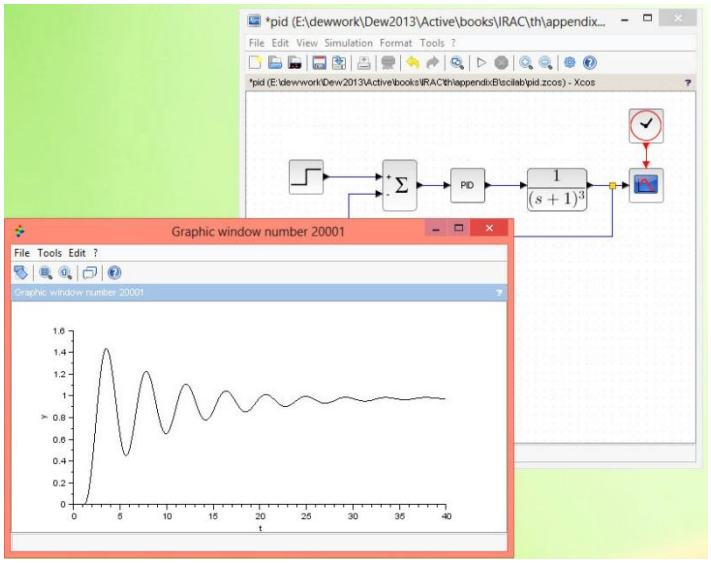

Figure 1.13 Simulation result showing a step response

1.4.3 RTSX Installation

As noted earlier, RTSX is just a collection of functions and scripts written solely in Scilab language together with some Xcos model examples, so it is independent of operating systems. You can use the same package on Windows, Linux, Mac that has Scilab installed. All files are placed in a directory tree and then packed into a single zip file, which you can download the latest version at www.controlsystemslab.com/rtsx/download . The filename should indicate the version number; e.g., RTSX1.0.zip for Version 1.00.

Download the zip file and place it in a working directory of your choice. For example, d:\work. There is no installer. Simply extract the file manually using the utility software (Windows Explorer, 7-zip, WinRAR etc) in your system . Unless you change directory name, you would then have all the files in d:\work\RTSX1.0. From now on, we will refer to this directory as RTSX root directory. If yours is different, just replace the pathname with your choice in all discussion that follows.

Launch Scilab and use the upper part of File Browser to change working directory to d:\work\RTSX1.0. Alternatively, in Scilab console window, type

```
-->cd d:\work\RTSX1.0
```

I prefer this method. You only have to type it once and then this command is kept in the Command History window. Next time you just conveniently click on it to change directory.

Check in the File Browser window that you are now at the root directory of RTSX. It should show RTSX1.0 with a few subdirectories and many *.sci files. Then load all the functions to Scilab workspace by typing

```
-->exec('startup_rtsx.sce',-1);

Robotic Tools for Scilab/Xcos (RTSX) Version 1.00
by Control Systems Lab,  April, 2013
http://www.controlsystemslab.com/rtsx
```

You should see the message as shown above. If instead you see some error message, something is incorrect. A common mistake is when one tries to launch the startup file from any directory other than RTSX root, by typing full pathname to exec command, such as `exec('d:\work\RTSX1.0\startup_rtsx.sce',-1);` This will not work, because startup_rtsx.sce contains a list of exec commands for each of the *.sci files including those in the subdirectories. `startup_rtsx.sce` cannot construct a correct pathnames for all files unless you execute at root directory. Check also for syntax typo.

As an another way to test whether RTSX is loaded succesfully, issue the command

```
-->rprdemo

RPR Robot Demonstration
Creating robot model and joint variable sequence...
See PlotRobot( ) in window number 1
See PlotRobotFrame( ) in window number 2
See AnimateRobot( ) in window number 3
Generating animation data.....................................
.....................................................................
.........
Enter [Y] to replay,[s] to change speed,  any other key to quit:
========== End of RPR Robot Demonstration =======
```

An RPR robot model is constructed and its symbolic diagram (Figure 1.14) and coordinate frames are plotted. Another graphic window shows simple animation of this robot. Press 'y' and Enter to replay the animation. Press 's' and input a new speed (1- lowest, 10-fastest), or any other key to quit. If you could experience all of these, then RTSX is correctly installed and ready for use.

The demo simply calls RTSX commands Ex. `Link`, `SerialLink`, `PlotRobot`, `PlotRobotFrame`, and `AnimateRobot`. All these commands will be explained later in subsequent chapters.

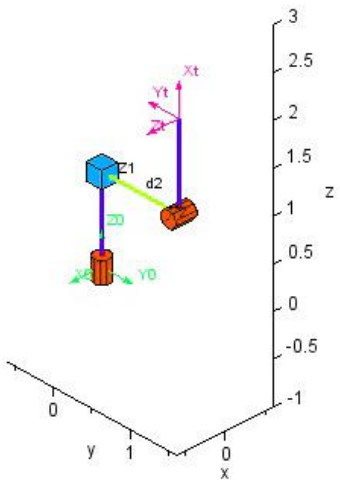

Figure 1.14 an RPR robot model created and plotted by RTSX

Note: In this book, discussion that involves RTSX is indicated by the icon RTSX

1.5 Book Structure

After this introduction, in chapter 2 we start with homogeneous transformation, a math tool used to describe translation and rotation of robot parts in 3-D. This is a basis for kinematics study in chapter 3, where we discuss coordinate frame attachment by DH convention. RTSX functions can be used to construct a robot model from DH link parameters. Inverse kinematics problems are generally more difficult to solve, and sometimes numerical methods have to be applied.

Velocity kinematics are also addressed. We explain how joint and end-effector velocities are related via a configuration-dependent matrix called Jacobian, which is also used to identify singularities of a robot. In chapter 4, various methods for trajectory generation in joint and Cartesian space are discussed. Chapter 5 is about robot dynamics and control. The two common approaches for dynamics analysis are Euler-Lagrange and Newton-Euler. Then we move on to independent joint control, which is the most basic scheme for robot control. The chapter ends with more advanced topics on multivariable, nonlinear control such as inverse dynamics and adaptive control schemes.. Chapter 6 is devoted to some basics of visual-based control for robots. RTSX command reference is provided in Appendix A.

Problems

1-1 The workspace of a robot is the total volume its end-effector could sweep as the robot executes all possible motions. It is constrained by the geometry of the manipulator as well as mechanical limits imposed on the joints. Figure 1.15 shows a sketch of workspace for cylindrical robot, which helps clarify why it is named as such.

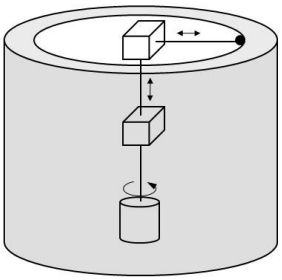

Figure 1.15 workspace of a cylindrical robot

Draw workspaces for all the robot diagrams listed on Section 1.3.

1-2 Download and install Scilab/RTSX. Examine the subdirectory /model where a few script files to create common robot models are included. Click on a file to see how a model is created. If it does not make much sense at this moment, don't worry. The commands will be explained in more detail in Chapter 3.

1-3 Familiarize yourself with Scilab/Xcos by writing a simple script and building a simple simulation model. Use examples provided in RTSX as guidelines.