# 9

# Hardware-in-the-loop Simulation and Real-time Control

In the previous chapters, Simulink was used to model complicated systems. Single variable and multivariable systems, linear and nonlinear systems, continuous, discrete and hybrid systems, time invariant and time varying systems, engineering and non-engineering systems can all be simulated by Simulink. With the use of S-function and Stateflow techniques, complicated systems, as well as discrete event systems can also be modeled and simulated.

So far, numerical simulation approaches in Simulink have been studied. However, the interaction with the real world has yet to be considered. For many practical processes, accurate mathematical models cannot be obtained, so exact Simulink models cannot be constructed. Sometimes due to the complexity of actual systems, the Simulink models established may not be accurate. The actual system should somehow be embedded in the simulation loop to get more accurate simulation results. This kind of simulation is usually referred to as a hardware-in-the-loop simulation. Since this kind of simulation is often performed in real-time, it is sometimes referred to as real-time simulation.

Real-Time Workshop provided by MathWorks can translate the Simulink models into C code, and the standalone executable files can also be generated using this tool, so that real-time control can be performed. Also, third-party software and hardware also provide interfaces to Simulink. Good examples of these products are dSPACE, with its Control Desk and Quanser plus WinCon (which can be used to implement hardware-in-the-loop simulation and real-time control experiments). MATLAB and Simulink support many products from well-known hardware manufacturers such as Motorola, Texas Instruments etc, and can directly generate executable code for them from Simulink models. A low-cost NIAT tool can also be used to implement hardware-in-the-loop simulation and experiments. Moreover, a much lower cost platform using Arduino is introduced. In this chapter, these tools will be demonstrated with applications in real-time control.

## 9.1 Simulink and Real-Time Workshop

### 9.1.1 *Introduction to Hardware-in-the-loop Techniques*

In hardware-in-the-loop simulation systems, part of the simulation loop is composed of computer software, while the rest is the actual hardware systems.

In practical control systems, the hardware in the loop can either be the controllers or the plant. In aerospace and military systems, the hardware in the system is usually the controllers. However, in process control systems, the hardware is usually the plant, and the controllers can be the Simulink models.

One advantage of hardware-in-the-loop simulation is that the validation of control results is very straightforward. In industrial control, hardware-in-the-loop simulation techniques can significantly reduce the time required to design controllers and can increase the reliability of the systems.

Fast prototype design is a new approach in controller design and implementation. The controllers can be constructed with Simulink and Stateflow, and the executable code for the controllers can be generated easily, and the parameters of the controllers can be tuned on-line, according to the actual control behavior. The designed controller can be regarded as a prototype, and once the control results are satisfactory, the code can be downloaded to the real controllers and the control actions can be carried out without the use of MATLAB or Simulink.

MathWorks provides the following tools to support Simulink controllers:

- **Real-Time Workshop, RTW**: Optimized C and Ada code can be generated automatically from the Simulink model. The executable file is much faster than the Simulink model.
- **Real-Time Workshop Embedded Coder**: This can be used to develop embedded C programs.
- **Real-Time Windows Target** and **xPC Windows Target**: The controller described by Simulink can be used to access the input and output ports of I/O boards, such as A/D and D/A converters, so that real-time control systems can be established.

### *9.1.2 Standalone Code Generation*

Sometimes the simulation process of Simulink model can be very slow, so we may need to speed up the process. On the other hand, we may sometimes need the simulation process to be executed independently, without support from MATLAB environment. Thus, standalone executables need to be considered.

In real-time simulation, fixed step size simulation algorithms have to be used. The details of setting up Simulink algorithms can be found in Fig. 4.34(b) in Chapter 4. Examples are given below to demonstrate the use of the real-time tools.

**Example 9.1** *Consider the Simulink model in Example 8.60, with the model file c8fstr1.mdl. Selecting a fixed step size of 0.001 s, and the algorithm* ***ode5(Dormand-Prince)****, the following statements can be used for assigning the values of the parameters*

```
M=0.01; K=1; Fsliding=1; Fstatic=1;
```

*These commands can be written into the* ***PreLoadFcn*** *property of the model, and saved in a new file c9fstr1.mdl. Running the model*

```
>> tic, [t,x,y]=sim('c9fstr1'); toc
```

*it can be seen that the execution time required is 11.32 s on the testing machine. The menu* ***Tools*** *→* ***Real-Time Workshop*** *→* ***Build Model*** *can be used, and an executable file c9fstr1.exe generated. This file is a standalone file, which can be executed without MATLAB. The following commands can be used, and the execution time can be measured as 0.37 s. This is much faster than the model running under MATLAB.*

```
>> tic, !c9fstr1
   toc % note that this command cannot be used in the previous line
```

*The executable file saves the results to the data file c9fstr1.mat. With the* `load` *command, the data obtained can be loaded into the MATLAB workspace. The two variables* `rt_tout` *and* `rt_yout` *are returned, which store respectively the time and output signal of the system. For instance, the simulation curves can be drawn with the following statements, as shown in Fig. 9.1.*

```
>> load c9fstr1; plot(rt_tout,rt_yout)
```

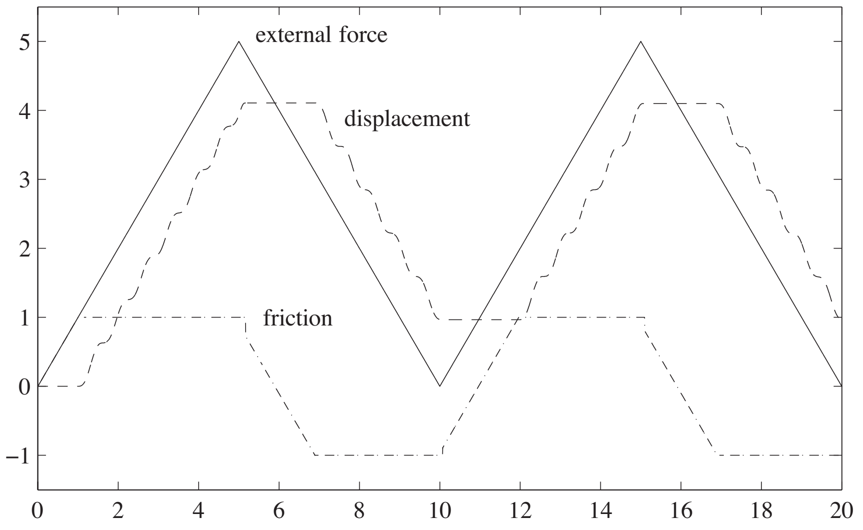


**Figure 9.1** Simulation results obtained with the executable file.

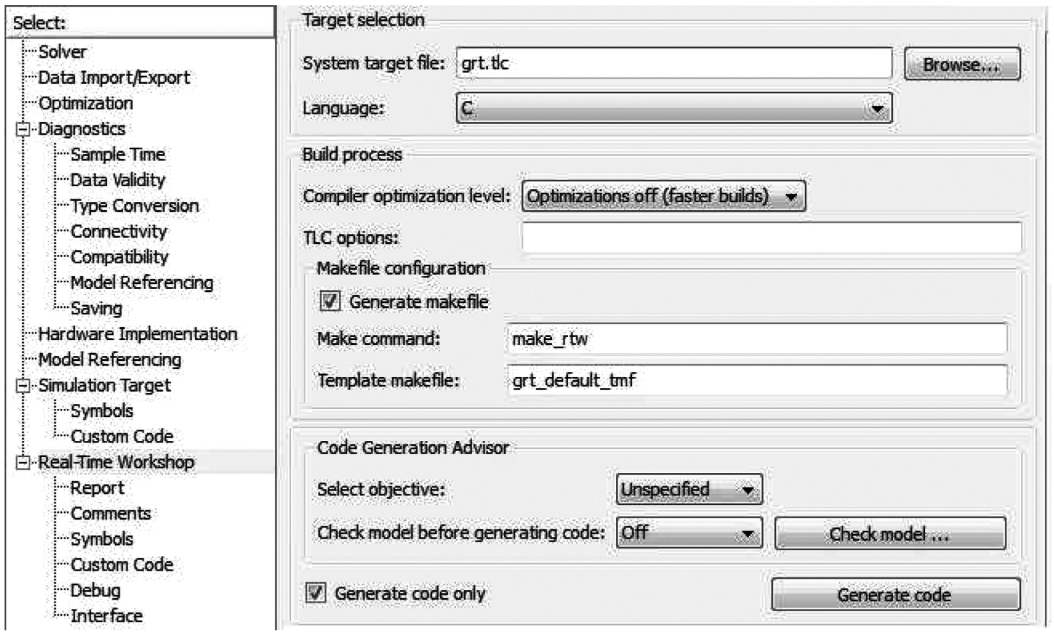
*It should be noted that the simulation results are obtained with the fixed step algorithm, but the results are the same as the variable step algorithm. With the* ***Simulation*** *→* ***Configuration Parameters*** *menu in the Simulink model window, the* ***Real-Time Workshop*** *pane of the dialog box is shown in Fig. 9.2. If only the C code is needed, then the* ***Generate code only*** *checkbox should be selected. The* ***Tools*** *→* ***Real-time workshop*** *→* ***Build Model*** *menu item can be selected to generate the source code. The files *.c, *.mk and *.h can be generated in the* `c9fstr1_grt_rtw` *folder. The executable file can then be constructed. Compiling parameters can be selected from the dialog box. If the executable file is to be generated, deselect the* ***Generate code only*** *checkbox.*

*If standalone executable is not needed, and the only requirement is to speed up the simulation process, there is no need to use fixed step algorithms. The menu* ***Simulation*** *→* ***Accelerator*** *can be used to construct dynamic link library files. The model can then be saved to a new file c9fstr2.mdl. Executing the file, the time required is 0.35 s. So it can be seen that the efficiency of the simulation model has again been significantly improved.*

```
>> tic, [t,x,y]=sim('c9fstr2'); toc
```

The Simulink models with S-functions written in C language can be converted into standalone executables, while the one containing the MATLAB version of S-functions cannot be manipulated in this way.

JWST357-c09 JWST357-Xue Printer: Yet to Come August 8, 2013 13:18 Trim: 7in × 10in



Target selection

System target file: grt.tlc

Language: C

Build process

Compiler optimization level: Optimizations off (faster builds)

TLC options:

Makefile configuration

[✓] Generate makefile

Make command: make_rtw

Template makefile: grt_default_tmf

Code Generation Advisor

Select objective: Unspecified

Check model before generating code: Off

[✓] Generate code only

**Figure 9.2** Simulink parameter setting dialog box.

### *9.1.3 Real-time Simulation and Target Computer Simulation*

Hardware-in-the-loop simulation and fast prototype design can be performed directly with Real-Time Windows. This allows the use of a computer as a host and a target in simulation [1]. When the Simulink model has been constructed, external simulation mode can be used to generate executable files.

In ordinary off-line control system design, the mathematical models of the plant model should be extracted first. Then, based on the mathematical model of the plant, a controller can be designed. However, when the controller is used in real-time control, the behavior of the system may not be as good; indeed, the control results may be very poor. This is because, in numerical simulation, many of the factors of the plant models may have to be neglected, such as the accuracy of the mathematical models, external disturbances, model parameter variations, measurement noise and so on. In actual systems, all these may affect the behavior of the practical systems.

Thus, hardware-in-the-loop simulation techniques are very important. Since the controller designed can be used to control the actual plant, the control behavior can be assessed directly. xPC is a low-cost hardware-in-the-loop simulation software package, and the major input and output boards are supported.

Here we shall concentrate on presenting the real-time simulation techniques based on Real-Time Windows Target. The environment should be set first. The command

```
rtwintgt -install
```

can be used to complete the required specifications.

```
You are going to install the Real-Time Windows Target kernel.
Do you want to proceed? [y] :
```

If error messages are obtained, it may indicate that the current operating systems are not supported. The current version of Real-Time Windows Target only supports operating systems such as 32-bit Windows XP and Vista, while Windows 7 and other 64-bit operating systems are not supported.

As soon as Real-Time Windows Target is installed, it can be used. However, to successfully run Real-Time Windows Target, both Microsoft Visual C++ and the Watcom C compiler should also be installed.

The concepts of "host" and "target" computers will be presented first in this section. The host here is normally the computer running MATLAB and Simulink, while the target computer refers to the computer running the standalone executables generated by Simulink. The RS232 interface and TCP/IP protocol can be used to connect the target with the host computer, and together they can complete real-time simulation tasks. It is not necessary to run the target executables on a different computer, but this application can only be used on the same computer.

**Example 9.2** *Consider the Van der Pol equation in Example 4.3. In the Simulink model, an* ***XY Scope*** *block was used, as shown in Fig. 4.44. For convenience, the model is saved again to c9fvdp1.mdl file, as shown in Fig. 9.3.*

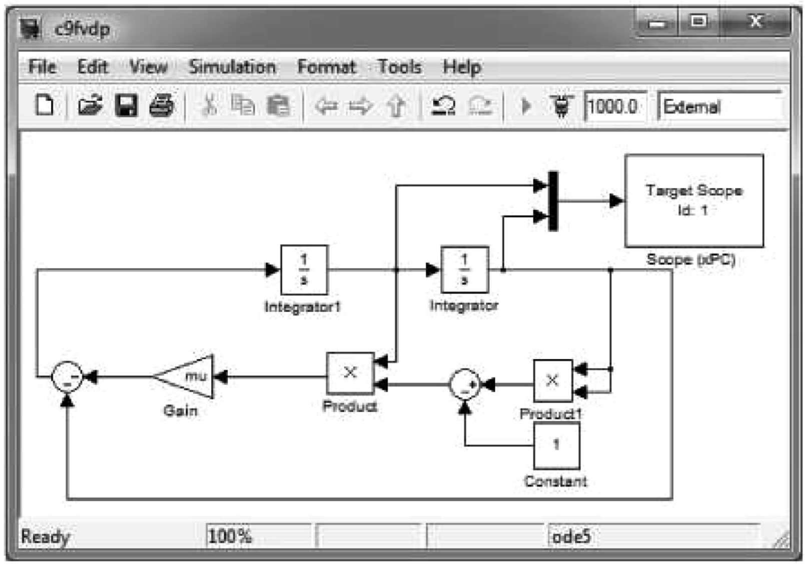
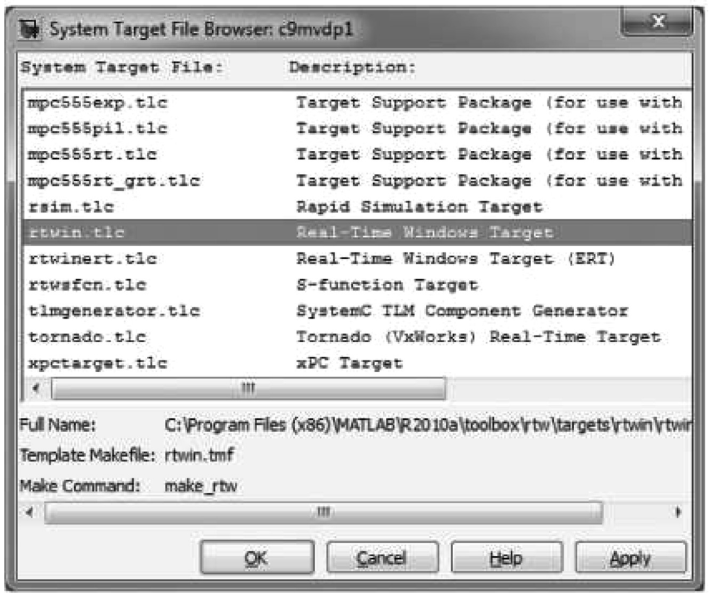


**Figure 9.3** Modified Simulink model for the Van der Pol equation (model name: c9fvdp1).

The **Simulation** → **Mode** menu supports the following simulation modes.

- **Normal simulation mode**: This is normally used in off-line numerical solutions of systems. This mode is the default one. In the **Simulation** menu, the **Normal** item should be checked.
- **Simulation accelerator mode**: When the menu **Simulation** → **Accelerator** is checked, Simulink will automatically translate the Simulink model into C code, and establish an executable *.mexw32 file. This model can be called from Simulink to speed up the simulation process. In this mode, variable step simulation algorithms can be selected, but it is not a standalone program. It has to be executed in the MATLAB environment.

- **External simulation mode**: The menu item **Simulation → External** is checked. Simulink models can be executed on the host with no MATLAB installed. In order to get the external function, **Simulation → Configuration parameters** or **Tools → Real-Time Workshop → Option** menus should be selected. In the dialog box, the **System Target File** listbox should be assigned as **Real-Time Windows Target**, as shown in Fig. 9.4. Selecting **Tools → Real-Time Workshop → Build model** menu, an executable Windows Target code can be generated automatically.
- Simulation also supports hardware-in-the-loop (**HIL**) and processor-in-the-loop (**PIL**) simulation.


System Target File Browser: c9mvdp1

| System Target File: | Description: |
|---|---|
| mpc555exp.tlc | Target Support Package (for use with |
| mpc555pil.tlc | Target Support Package (for use with |
| mpc555rt.tlc | Target Support Package (for use with |
| mpc555rt_grt.tlc | Target Support Package (for use with |
| rsim.tlc | Rapid Simulation Target |
| rtwin.tlc | Real-Time Windows Target |
| rtwinert.tlc | Real-Time Windows Target (ERT) |
| rtwsfcn.tlc | S-function Target |
| tlmgenerator.tlc | SystemC TLM Component Generator |
| tornado.tlc | Tornado (VxWorks) Real-Time Target |
| xpctarget.tlc | xPC Target |

Full Name: C:\Program Files (x86)\MATLAB\R2010a\toolbox\rtw\targets\rtwin\rtwin

Template Makefile: rtwin.tmf

Make Command: make_rtw

**Figure 9.4** Parameter setting dialog box.

From the listbox in Fig. 9.4, it can be seen that, in addition to the ordinary **Real-Time Windows Target** option, a large number of other target formats are also supported.

- **DOS(4GW) Real-Time Target**: The target program can be generated for MS-DOS systems. However, the code can only be executed in the DOS environment, and cannot be executed in a DOS window under the Windows system.
- **Real-Time Workshop Embedded Coder**
- **Rapid Simulation Target**: The rapid simulation code can be generated, mainly used for frequently performed simulation processes, such as the Monte Carlo simulation.
- **Real-Time Windows Target**
- **Tornado (VxWorks) Real-Time Target**

When the compiler is set up, the menu **Tools → Real-Time Workshop → Build model** can be selected to compile and link the model, such that an executable file c9fvdp1.exe is generated.

Selecting the **Tools** → **External mode control panel** menu item, the dialog box shown in Fig. 9.5 will be opened. Click the **Connect** button, and the executable file can be executed in real-time.

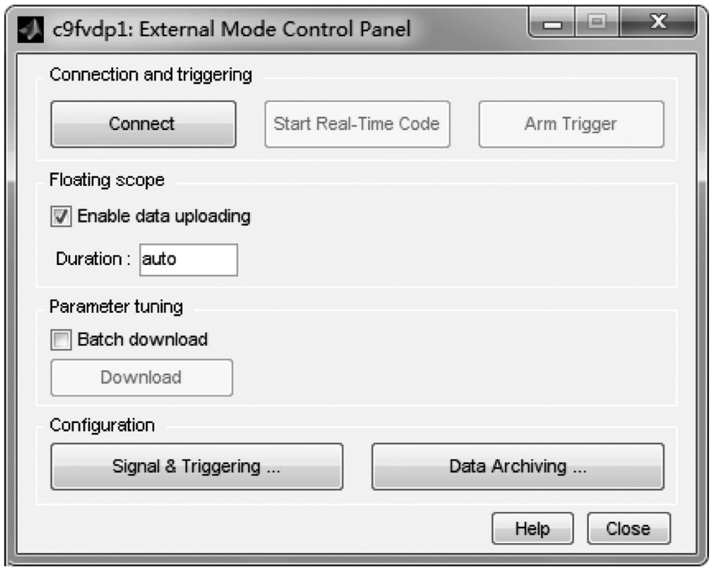


**Figure 9.5** Control panel of the external simulation mode.

In a similar way, the target executable file can be generated for MS DOS system. The Watcom C compiler should be used to compile and link the code, and the executable generated can then be run directly under MS DOS.

Real-Time Windows Target also provides its own blockset. The blockset can be opened with either the Simulink model library, or the MATLAB command `rtwinlib`, as shown in Fig. 9.6. Various blocks suitable for real-time simulation are provided.

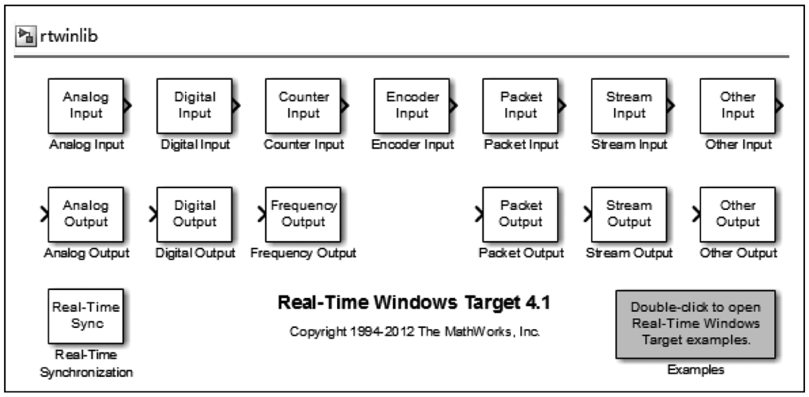


**Figure 9.6** Real-Time Windows Target blockset.

JWST357-c09 JWST357-Xue Printer: Yet to Come August 8, 2013 13:18 Trim: 7in × 10in

### *9.1.4 Hardware-in-the-loop Simulation with xPC Target*

xPC Target is the fast prototype design tool provided by MathWorks. It can be used in real-time testing and development of controllers. Also, C compilers, such as Microsoft Visual C++ or MATCOM C/C++, are required. Through the compatible compilers, an executable file can be generated and used in real-time control.

The major characteristics of xPC Target are that the real-time executable code generated by Simulink can run on the target machine. The maximum sample rate can be as high as 100 kHz. Various standard input and output devices are supported, and interactive parameter tuning on the host or target computers can also be carried out. The communication with RS232 and TCP/IP protocol between the host and target computers can be established. Desktop, laptop computers and PC/104, PC/104+, CompactPCI, single board or single-chip computers can be used as target computers to perform real-time control tasks.

On the host computer, MATLAB and Simulink should be installed, and a C compiler can be used as a development tool. Real-time application programs can be generated. To run the executable program, a disk containing xPC Target real-time kernel should be used to boot the target computer. Once the target computer is booted, the real-time application program can run on that computer.

In the MATLAB command window, execute the `xpclib` command, or click the **xPC Target** icon in the Simulink model library, and the xPC group shown in Fig. 9.7 will open. In the group, there are commonly used device icons such as **A/D**, **D/A**, **Counter** and so on. Users can select appropriate devices as needed.

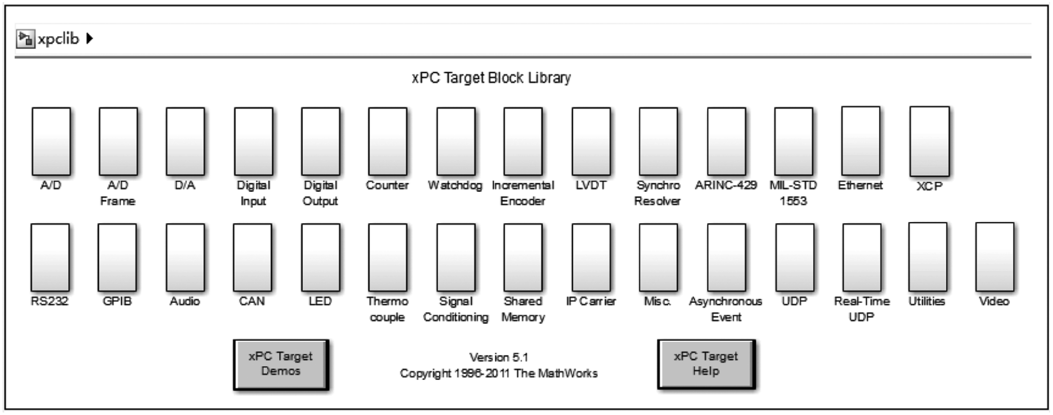


**Figure 9.7** xPC Target blockset.

For instance, double click the **A/D** icon, and the A/D converter group will open, as shown in Fig. 9.8, where all the well-known A/D converters are provided. Double click the **Digital Input**, and then the **Advantech** icon, and the Advantech A/D device group shown in Fig. 9.9 is displayed. Suitable A/D devices from the group can be selected, and the controllers can be connected to the control terminals.

Here the Advantech input/output board PCL 1800 is used to introduce the application of the xPC tools. PCL 1800 is an ISA input/output board and it can be used on the main board of the computer. On this board, 16 12-bit analog input channels and two 12-bit D/A converters are provided.

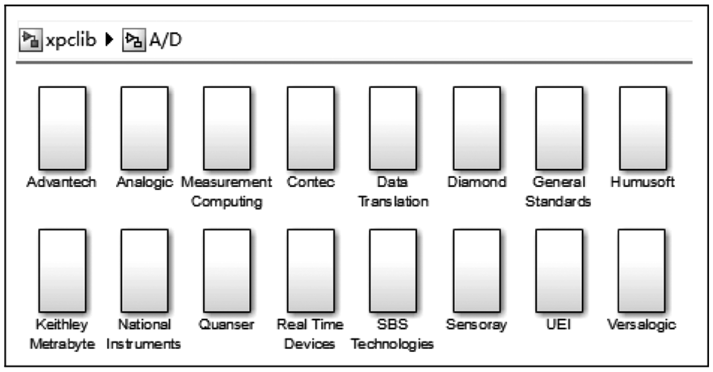


**Figure 9.8** A/D converter supported by xPC.

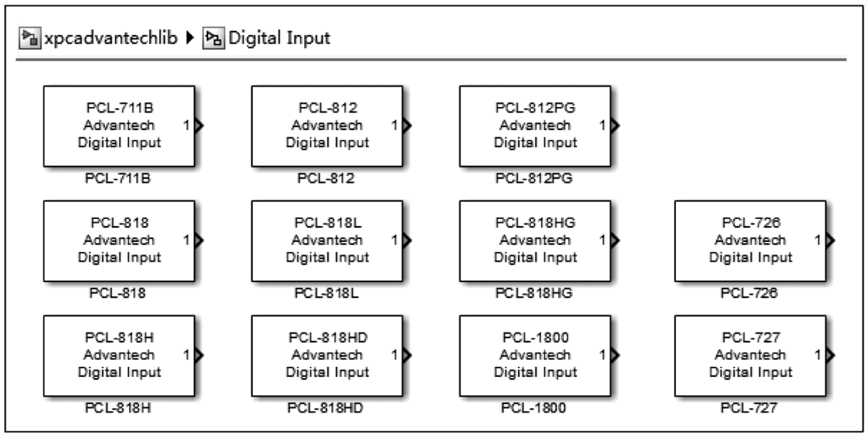

xpcadvantechlib ▸ Digital Input
PCL-711B Advantech Digital Input
PCL-711B
PCL-812 Advantech Digital Input
PCL-812
PCL-812PG Advantech Digital Input
PCL-812PG
PCL-818 Advantech Digital Input
PCL-818
PCL-818L Advantech Digital Input
PCL-818L
PCL-818HG Advantech Digital Input
PCL-818HG
PCL-726 Advantech Digital Input
PCL-726
PCL-818H Advantech Digital Input
PCL-818H
PCL-818HD Advantech Digital Input
PCL-818HD
PCL-1800 Advantech Digital Input
PCL-1800
PCL-727 Advantech Digital Input
PCL-727

**Figure 9.9** AdvanTech A/D converter.

**Example 9.3** *Consider the PI control block diagram studied in Example 5.25. In the system, the PI controller can be designed with the mathematical model of the plant. The Simulink block diagram is obtained as shown in Fig. 9.10.*

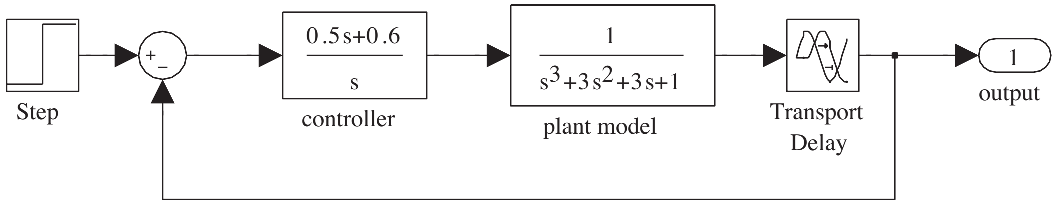


**Figure 9.10** Numerical simulation of a control system model (model name: c9mpi).

*It has been pointed out that the simulation results are made on the mathematical model of the plant, and the simulation results may not be the same when the actual plant is used in the system. Sometimes the errors can be misleading. In order to test the control behavior of the actual plant, the* *mathematical model of the plant must first be deleted from the Simulink model, and then replaced with the actual plant. This can be done using the D/A and A/D converters of xPC, for instance, with the Advantech PCL 1800, as shown in Fig. 9.11.*

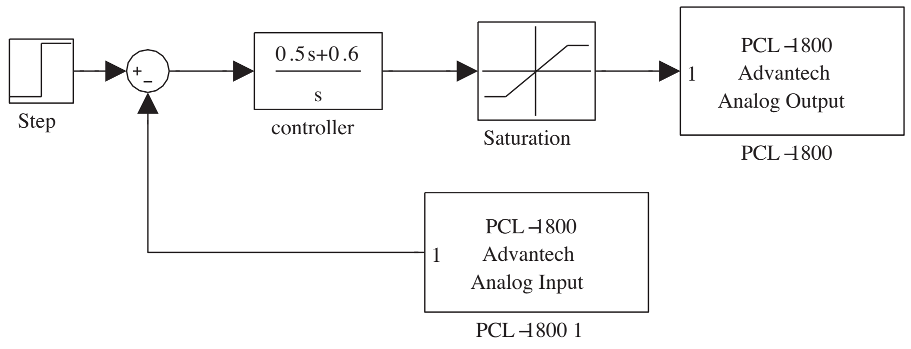
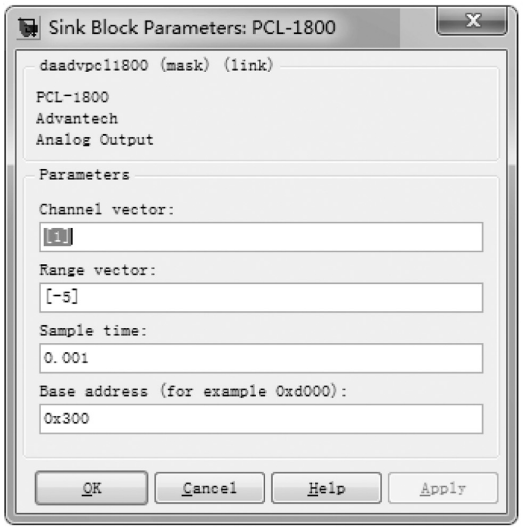


**Figure 9.11** Hardware-in-the-loop PI control system (model name: c9mpi1).

With the Simulink model, the actual plant can be connected to the computer through the Advantech PCL 1800 card. The input terminal of the plant can be connected to the D/A port to accept control signals. The output terminal of the plant can be connected to the A/D port, which returns the measured output to the computer. Double click the A/D block and D/A block and the dialog boxes are respectively shown in Figs 9.12(a) and (b).


Sink Block Parameters: PCL-1800

daadvpcl1800 (mask) (link)

PCL-1800
Advantech
Analog Output

Parameters

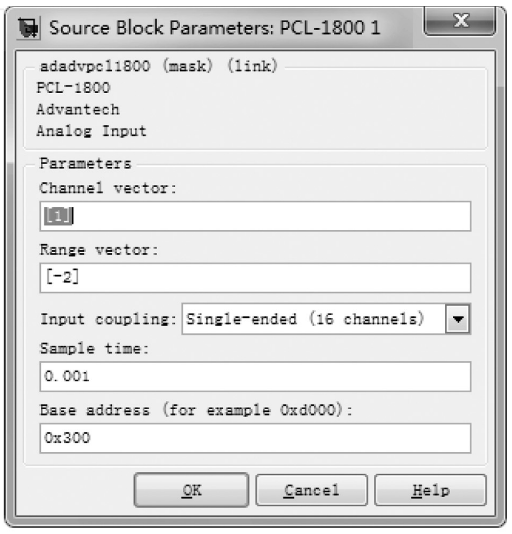
Channel vector: [1]

Range vector: [-5]

Sample time: 0.001

Base address (for example 0xd000): 0x300

(a) A/D converter dialog box


Source Block Parameters: PCL-1800 1

adadvpcl1800 (mask) (link)

PCL-1800
Advantech
Analog Input

Parameters

Channel vector: [1]

Range vector: [-2]

Input coupling: Single-ended (16 channels)

Sample time: 0.001

Base address (for example 0xd000): 0x300

(b) D/A converter dialog box

**Figure 9.12** Dialog boxes of the PCL 1800 AD and DA converters.

- **Channel vector**: If it is connected to a particular channel, the number of it should be entered. For instance, the entry **1** here is the channel number. If many signals are connected to the D/A converters, the channel vector should be specified.
- **Range vector**: Each input and output signal has its own range, which can be expressed by `[`$v_1$`,`$v_2$`,···]`. If the range of the $i$th signal is (−5, +5), then it is denoted as $v_i = -5$. However, if the range is (0, +5), it is denoted by $v_i = 5$.
- **Sample time**: This is the sampling interval of the block and it should be the same as the step size of the simulation algorithm.
- **Base address**: The specification is the same as the one in the input and output cards.

**Example 9.4** *It looks as if the system given in Fig. 9.11 is not a closed-loop system; rather it looks like an open-loop structure. In fact, the connection is indeed a closed-loop structure, when the actual plant is considered. Since the plant is part of the simulation system, such a system is referred to as the "hardware-in-the-loop" simulation structure.*

*If the input, output and intermediate signals are needed, the simulation model can be redrawn as shown in Fig. 9.13. In such a system, the signal obtained can be saved into data files. The data files can be loaded into the MATLAB workspace, or you can use an x-y scope block, such that real-time display of the results is possible.*

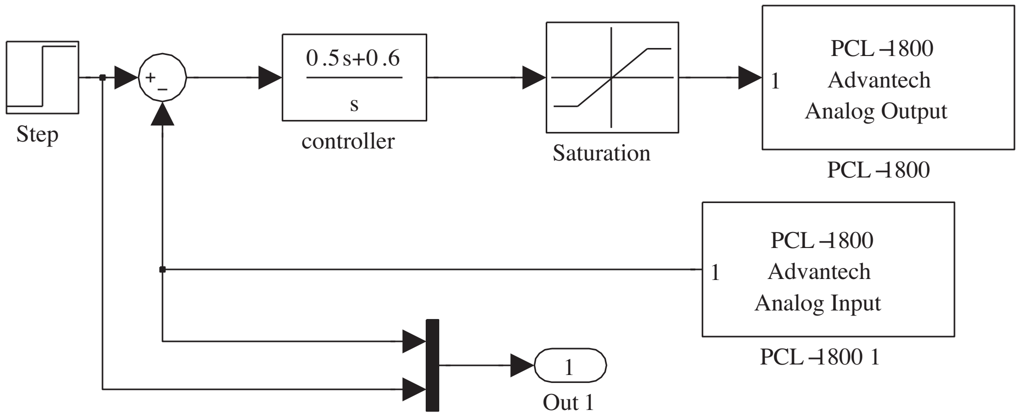


**Figure 9.13** Hardware-in-the-loop simulation block diagram with signal detection (c9mpi2).

## 9.2 Introduction to dSPACE and its Blocks

### *9.2.1 Introduction to dSPACE*

dSPACE (digital Signal Processing And Control Engineering) is the real-time simulation and testing platform, and it can be used along with MATLAB/Simulink. The dSPACE real-time system has high-speed computation capabilities in hardware systems, with processes and I/O ports, and it can easily be used in real-time code generation, downloading and testing [2].

dSPACE real-time control systems have certain advantages. The blocks provided are adequate for constructing real-time control systems, and the real-time facilities are ideal, since the on-board PowerPC processor can be used directly. Seamless connection with MATLAB and Simulink can be used to directly convert the numerical simulation structure to real-time control. dSPACE systems are now widely used in automotive, aerospace, robots, industrial automation and other fields. By using dSPACE systems, product development cycles can be significantly reduced, and the control quality significantly increased.

### *9.2.2 dSPACE Block Library*

Software and hardware environments of the dSPACE real-time simulation system are presented below. The widely used hardware in education and scientific research is the ACE1103 and ACE1104 R&D controller boards. Real-time control software Control Desk, real-time interface RTI and real-time data acquisition interface MTRACE/MLIB are provided in the dSPACE package, and it is very flexible and convenient to use. The DS1104 R&D controller board, which is equipped with a PCI interface and a PowerPC processor, is a cost-effective entry-level control system design product. Here, the DS1104 R&D controller board is used to illustrate the applications in hardware-in-the-loop simulation.

When the hardware and software of dSPACE are installed, there will be a dSPACE group in the Simulink model library; double click its icon, and the group will be opened as shown in Fig. 9.14.

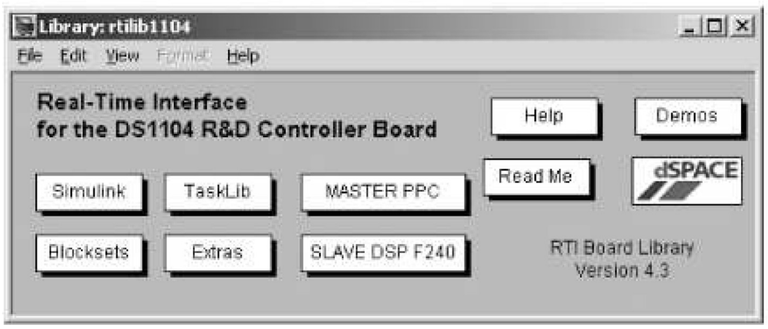


**Figure 9.14** dSPACE 1104 blockset.

Double click the **MASTER PPC** block, and the model library shown in Fig. 9.15 will be opened. It can be seen that a lot of components on the board, such as A/D converter, are represented by blocks in the group. Also, double clicking the **Slave DSP F240** icon will open the slave DSP F240 blocks shown in Fig. 9.16. Many practical servo control blocks, such as PWM signal generator and frequency sensor are provided in the group, and they can be dragged to Simulink models. The signals generated by the computer can also be used to drive the actual plant, so that hardware-in-the-loop simulation can be completed.

## 9.3 Introduction to Quanser and its Blocks

### *9.3.1 Introduction to Quanser*

Quanser products are developed by Quanser Inc., who provides various plants for control education. Quanser has interfaces to MATLAB/Simulink and to LabView of National Instruments. It also provides real-time control software WinCon (now named as QUARC), which is similar to Control Desk from dSPACE. Quanser products are suitable for control education and laboratory research and experimentation, where different control algorithms are being tested. Quanser also offers industrial solutions for hardware-in-the -loop rapid prototyping of real time control systems.

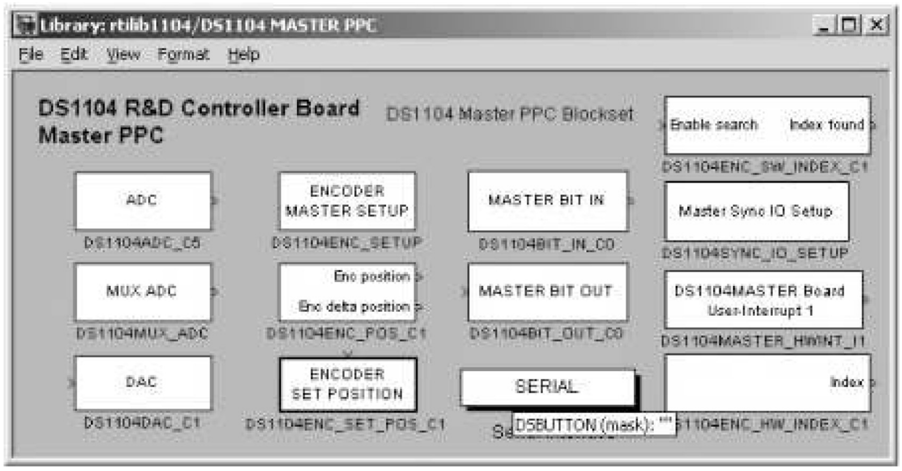


**Figure 9.15** **MASTER PPC** group.

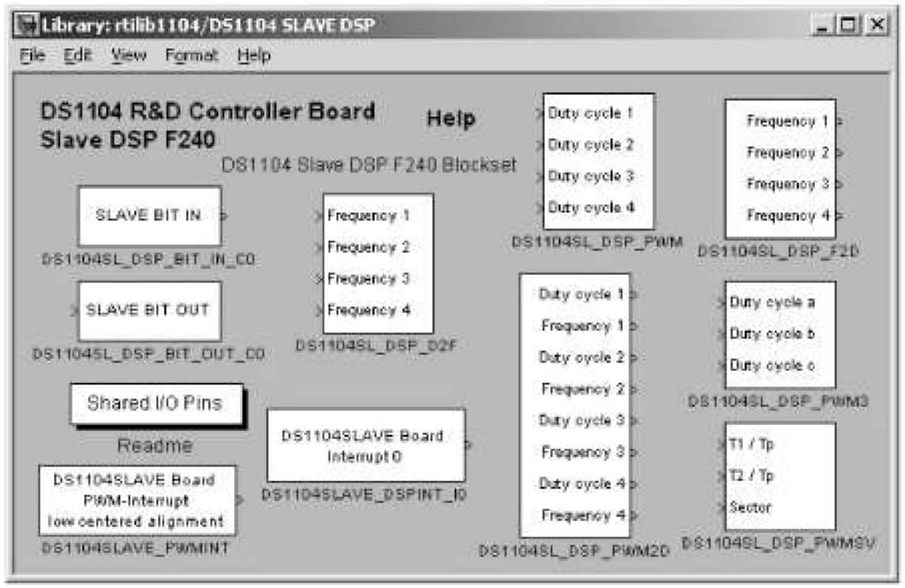


**Figure 9.16** **Slave DSP F240** group.

Quanser experimental plants include linear motion control series, rotary series and other special experimental devices. WinCon software enables Simulink modes to directly control the actual plants.

## 9.3.2 *Quanser Block Library*

MultiQ3, Q3, Q4, Q8 and other kinds of interface boards are provided in Quanser series products. The boards all provide D/A and A/D converters, and motor encoder input and output ports, such that the actual plants can be connected to the computers to construct closed-loop control structures.

WinCon is a Microsoft Windows based application program implementing real-time control. The program can be used to execute code generated by Simulink models and exchange data with the MultiQ card, to achieve real-time control. When WinCon is installed, a **WinCon Control Box** group will appear in the Simulink model library, as shown in Fig. 9.17, where the subgroups for different Q-series boards are provided.

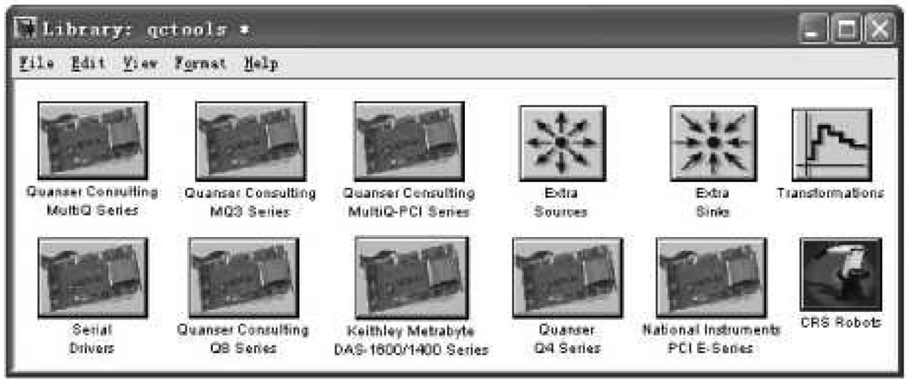

Library: qctools *
File Edit View Format Help
Quanser Consulting MultiQ Series
Quanser Consulting MQ3 Series
Quanser Consulting MultiQ-PCI Series
Extra Sources
Extra Sinks
Transformations
Serial Drivers
Quanser Consulting Q8 Series
Keithley Metrabyte DAS-1600/1400 Series
Quanser Q4 Series
National Instruments PCI E-Series
CRS Robots

**Figure 9.17** **WinCon Control Box** group.

Here the Q4 board is used as an example for further demonstration. Double click the **Quanser Q4 Series** icon in Fig. 9.17 and the Simulink library for the Q4 board is opened as shown in Fig. 9.18. It can be seen that the blocks **Analog Input** and **Analog Output** are used to implement the A/D and D/A converters respectively.

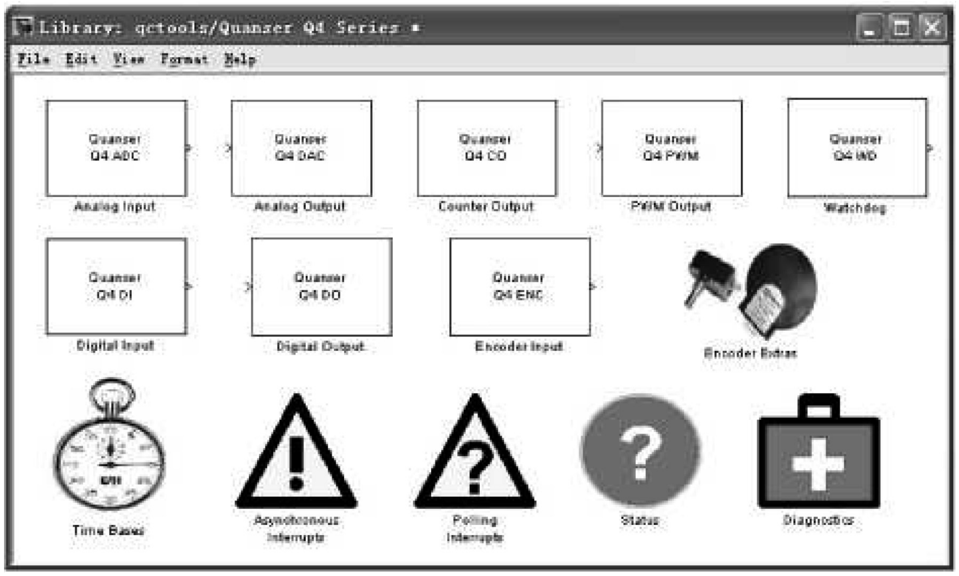
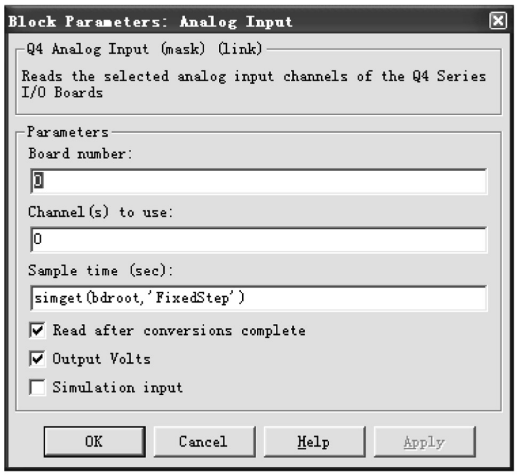


**Figure 9.18** Blocks in the MultiQ4 group.

Double clicking the icons of the **Analog Input** and **Analog Output** blocks, the parameter dialog boxes will be opened as shown in Figs 9.19(a) and (b), respectively. The key parameter **Channel** can


Block Parameters: Analog Input

Q4 Analog Input (mask) (link)

Reads the selected analog input channels of the Q4 Series I/O Boards

Parameters

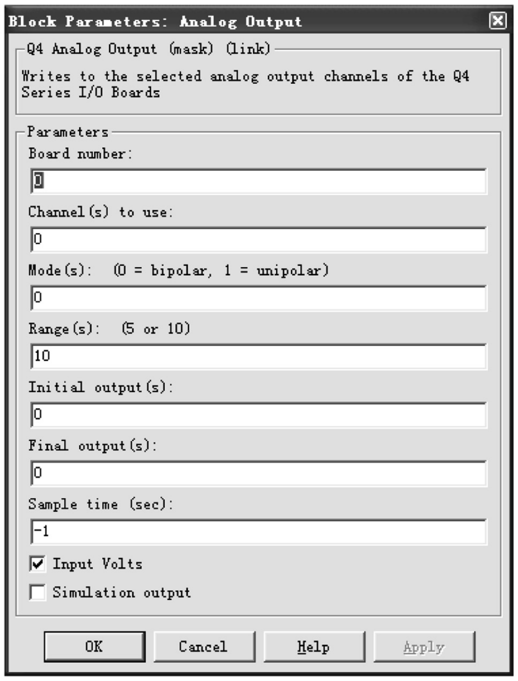
Board number:
0

Channel(s) to use:
0

Sample time (sec):
simget(bdroot,'FixedStep')

☑ Read after conversions complete
☑ Output Volts
☐ Simulation input

(a) **Analog Input** block setting


Block Parameters: Analog Output

Q4 Analog Output (mask) (link)

Writes to the selected analog output channels of the Q4 Series I/O Boards

Parameters

Board number:
0

Channel(s) to use:
0

Mode(s): (0 = bipolar, 1 = unipolar)
0

Range(s): (5 or 10)
10

Initial output(s):
0

Final output(s):
0

Sample time (sec):
-1

☑ Input Volts
☐ Simulation output

(b) **Analog Output** block setting

**Figure 9.19** Dialog boxes of the WinCon blocks.

be set in the dialog box, and they should be the same as the one that is actually connected, otherwise the block cannot be used properly.

### *9.3.3 Plants in Quanser Rotary Series*

Quanser experimental plants are mainly the linear motion series and the rotary motion series. The components in the linear series can be used to compose different experiments such as linear speed and position servo control, linear inverted pendulum, double inverted pendulum. In the rotary series, experiments such as rotary inverted pendulum, planar inverted pendulum, ball and beam flexible joint and flexible link can be composed. Other plants such as helicopter attitude control and magnetic levitation systems can also be composed. Some of the plants in the Quanser series system are shown in Fig. 9.20.

## 9.4 Hardware-in-the-loop Simulation and Real-time Control Examples

### *9.4.1 Mathematical Descriptions of the Plants*

The ball and beam system is one of the experimental systems in the rotary series provided by Quanser. A photograph of this system is shown in Fig. 9.21(a), and the principal structure is shown in Fig. 9.21(b). The principle of the control of the ball and beam system is that the angle $\theta$ is

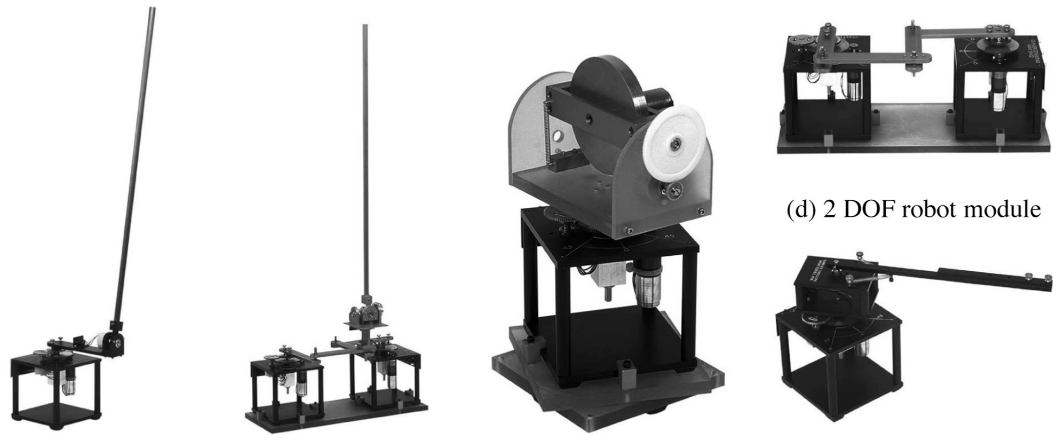


**Figure 9.20** Some of the plants in the rotary series (IP is inverted pendulum).

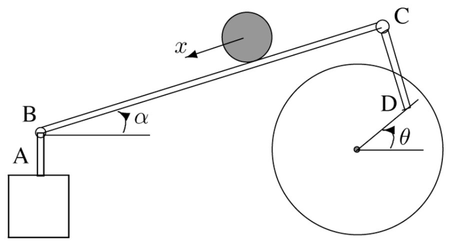


**Figure 9.21** Ball and beam system.

controlled by the bar, driven by the motor, such that the angle $\alpha$ of the beam BC can be adjusted to the expected position rapidly. The bar AB is a fixed supporting arm.

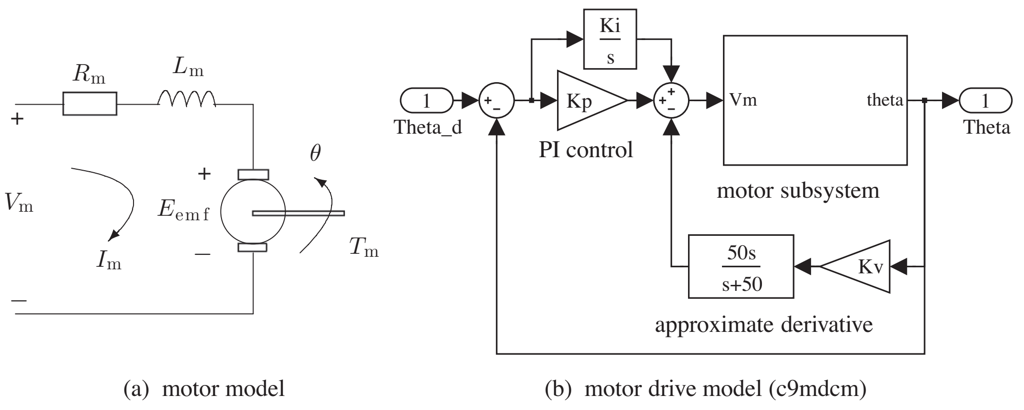
In the ball and beam system, the position $x(t)$ of the ball is the output signal, and the motor voltage $V_{\rm m}(t)$ is the control signal. A controller is needed such that the error $e(t)$ between the expected position $c(t)$ and the detected position signal $x(t)$, that is, $e(t) = c(t) - x(t)$, can be used to calculate the control signal $u(t)$. The steel ball on the beam can be used as a variable resistor, such that the position $x(t)$ can be directly detected through the value of the resistor.

**1) Mathematical model of the motor drive system**

The DC motor model is shown in Fig. 9.22(a). Assume that the electrical efficiency $\eta_{\rm m} = 0.69$, the equivalent resistance of the motor system is $R_{\rm m} = 2.6\,\Omega$, viscous damping coefficient is $B_{\rm eq} = 4\times 10^{-3}\,{\rm N\cdot m\cdot s/rad}$, transmission ratio $K_{\rm g} = 70$, EMF constant $K_{\rm m} = 0.00767\,{\rm V\cdot s/rad}$, torque constant $K_{\rm t} = 0.00767\,{\rm N\cdot m}$, motor moment of inertia of the equivalent load $J_{\rm eq} = 2\times 10^{-3}\,{\rm kg\cdot m^2}$, motor moment of inertia $J_{\rm m} = 3.87\times 10^{-7}\,{\rm kg\cdot m^2}$, gearbox efficiency $\eta_{\rm g} = 0.9$.

Constructing a PID controller for the motor, and using derivative action in the feedback loop, the control system structure is as shown in Fig. 9.22(b). A simple tuning method will be given later,

JWST357-c09 JWST357-Xue Printer: Yet to Come August 8, 2013 13:18 Trim: 7in × 10in



(a) motor model (b) motor drive model (c9mdcm)

**Figure 9.22** Motor and control models.

showing how to design the PID controller parameters to control the angular displacement $\theta$ of the model.

In the Quanser experimental system, the Simulink model of the motor system can be constructed as shown in Fig. 9.23. The transfer function of the motor voltage signal $V_m(t)$ to the angle $\theta$ can be expressed as [3]

$$G_1(s)=\frac{\theta(s)}{V_m(s)}=\frac{\eta_g\eta_m K_t K_g}{J_{eq}R_m s^2+(B_{eq}R_m+\eta_g\eta_m K_m K_t K_g^2)s}=\frac{61.54}{s^2+35.1\,s}. \tag{9.1}$$

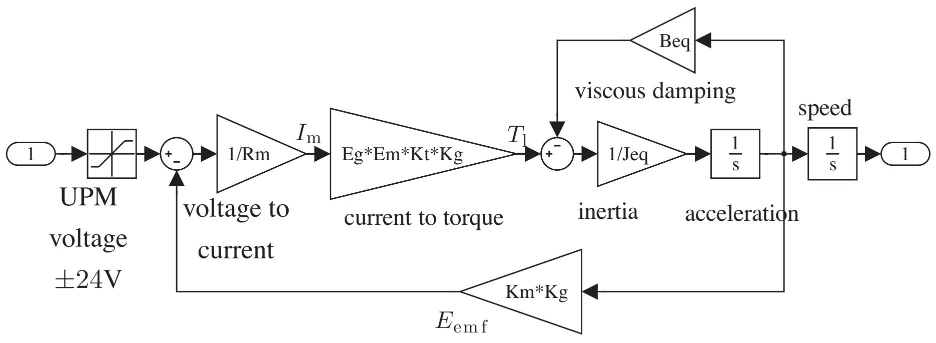


**Figure 9.23** Simulink description of motor model.

**2) Mathematical model of the ball and beam system**

Assume that the length of the beam $l = 42.5$ cm, and the radius of the ball is $R$. The gravity component on x axis is $F_x = mg\sin\alpha$, $m = 0.064$g, and the moment of inertia of the ball is $J = 2mR^2/5$. The dynamical model of the ball is $\ddot{x} = 5g\sin\alpha/7$.

Since the angle $\alpha$ is usually very small, it can be approximated as $\sin\alpha \approx \alpha$, so the original nonlinear model can be approximated by a linear model. The radius of the eccentric disc is $r = 2.54$ cm. The equivalent arc displacement of beam BC can be expressed as $l\alpha = r\theta$, that is, $\theta = l\alpha/r$. Based on the above formula, the Simulink model shown in Fig. 9.24 can be constructed [4].

JWST357-c09 JWST357-Xue Printer: Yet to Come August 8, 2013 13:18 Trim: 7in × 10in

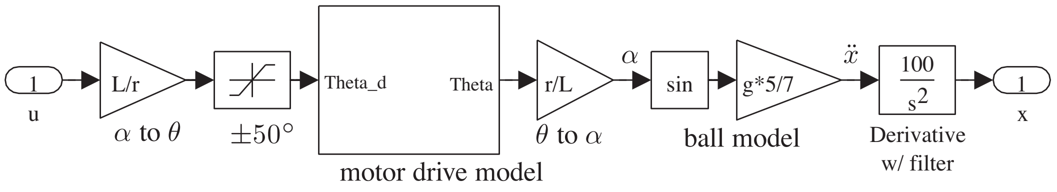
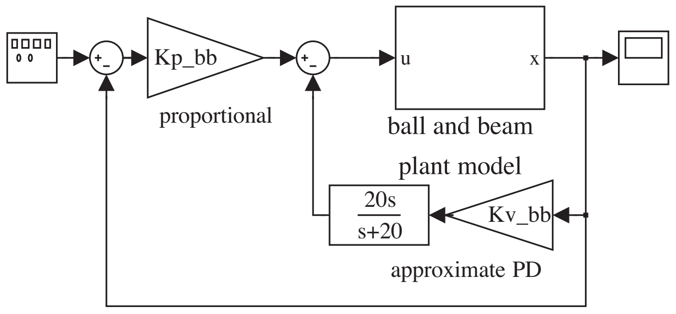


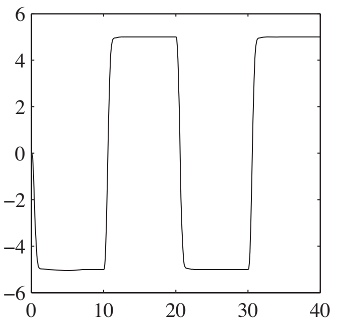
**Figure 9.24** Plant model (model name: c9mball).

The Simulink model of the ball and beam control system can be constructed as shown in Fig. 9.25(a), where the whole system is controlled by a PD controller. The parameters of the models and controllers can be entered with the following statements:

```
clear all;  % filename: c9dat_set.m
Beq=4e-3; Km=0.00767; Kt=0.00767; Jm=3.87e-7; Jeq=2e-3; Kg=70;
Eg=0.9; Em=0.69; Rm=2.6; zeta=0.707; Tp=0.200; num=Eg*Em*Kt*Kg;
den=[Jeq*Rm, Beq*Rm+Eg*Em*Km*Kt*Kg^2 0]; Wn=pi/(Tp*sqrt(1-zeta^2));
Kp=Wn^2*den(1)/num(1); Kv=(2*zeta*Wn*den(1)-den(2))/num(1); Ki=2;
L=42.5; r=2.54; g=9.8; zeta_bb=0.707; Tp_bb=1.5;
Wn_bb=pi/(Tp_bb*sqrt(1-zeta_bb^2)); Kp_bb=Wn_bb^2/7;
Kv_bb=2*zeta_bb*Wn_bb/7; Kp_bb=Kp_bb/100; Kv_bb=Kv_bb/100;
```



(a) ball and beam control (c9mbeam)



(b) simulation result

**Figure 9.25** Control and simulation of the ball and beam system.

Simulating the ball and beam system, the simulation results can be obtained as shown in Fig. 9.25(b). It can be seen that the simulation results are satisfactory. Of course, experimental validation of the controllers can be and should be performed with a hardware-in-the-loop simulation. The hardware-in-the-loop and real-time control of the system are to be demonstrated below.

## 9.4.2 *Quanser Real-time Control Experimentation*

It can be seen that the inner PID controller in the system is used to apply control signal $V_m$ to the motor. The position $x$ of the ball and the angle of the motor $\theta$ are measured. The control signal can be implemented with the **Analog Output** block, and the position $x$ of the ball can be measured with the **Analog Input** block, and the angle $\theta$ of the motor can be measured with the **Encoder Input** block. In real applications, filters can be used, and the real-time control system can be constructed as shown in Fig. 9.26. Note that a fixed step size algorithm is used and the step size is selected as 0.001 s. This can be done by the **Simulation** → **Parameters** menu in the Simulink model, and the **Solver** pane can be set. For instance, the algorithm can be set to **ode4**.

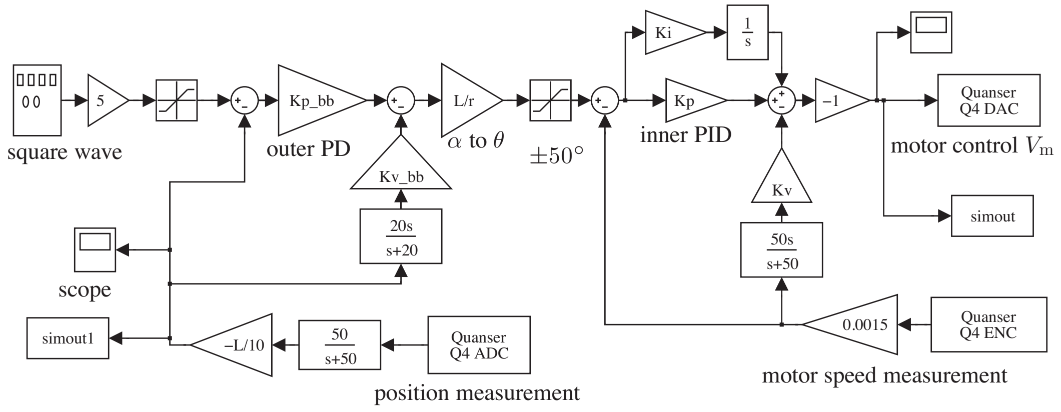


**Figure 9.26** Real-time control Simulink model (model name: c9mbbr).

Selecting the **Tools** → **Real-Time Workshop** → **Build Model** menu item, the original Simulink model can be compiled and the dynamic link library file can be generated automatically. The winCon control interface shown in Fig. 9.27 is opened automatically. The plant is controlled directly from the interface. Click the **START** button and the real-time facilities are started. The **Analog Output** block is used to apply the control signal to the motor, and the angle and ball position are measured in real time and fed back to the computer, so that closed-loop control can be achieved.

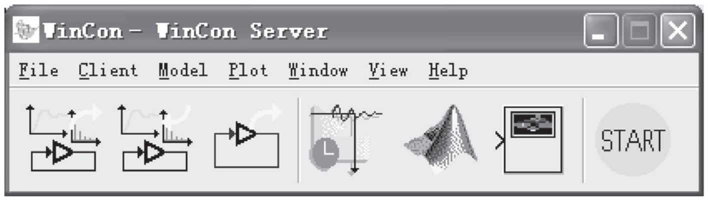
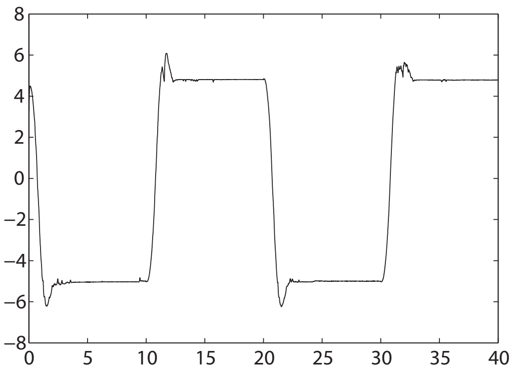


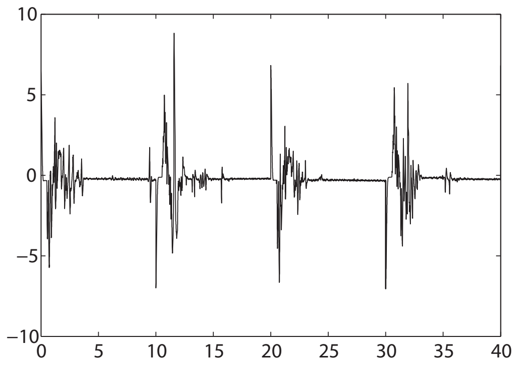
**Figure 9.27** WinCon control interface.

The scope button in the interface can be clicked to display the waveforms of the ball positions and control signals, as shown in Fig. 9.28. It should be noted that there are differences between the actual control results and the numerical simulation results. The differences may be caused by modeling error or other minor discrepancies. With the WinCon real-time control interface, the Simulink model runs in the **External** mode. In this mode, if you modify the variables in the MATLAB workspace, real-time control results will also change.

In real-time control systems, the scope data can be saved to the MATLAB workspace with the **File** → **Save** → **Workspace** menu item. The amount of data is determined by the size of the buffer. The buffer size can be selected with the **Buffer** menu; it is set to 50 s in this example.



(a) ball displacement $\hat{x}(t)$



(b) motor voltage $V_{\mathrm{m}}(t)$

**Figure 9.28** Real-time control of the ball and beam system.

### *9.4.3 dSPACE Real-time Control Experimentation*

From the Simulink model constructed by the interfaces to the Quanser Q4 components shown in Fig. 9.26, a new Simulink model can be constructed, with the corresponding components replaced by dSPACE blocks, as shown in Fig. 9.29. In the dSPACE blockset, the settings of the A/D and D/A converters are different from those in Quanser. We need to multiply the Quanser A/D converter figures by 10, and the D/A figures by 0.1, so that the units are unified. Also, since the motor encode input block of dSPACE is different from that of Quanser, this should be multiplied by 0.006.

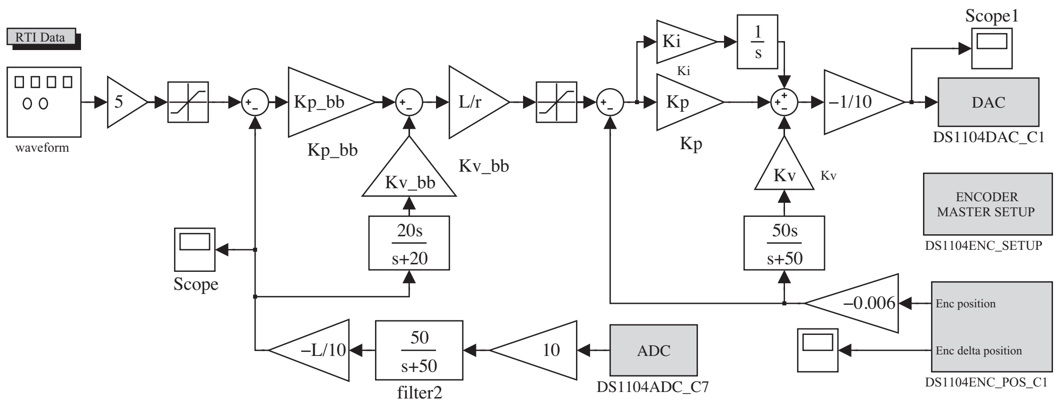


**Figure 9.29** Simulink model with dSPACE (model name: c9mdsp).

From the model established, the menu **Tools** → **Real-Time Workshop** → **Build Model** in the Simulink model window can be selected to compile it, and finally the system description file c9mdsp.ppc can be generated for Power PC. Open the Control Desk software environment window [5], and the menu **File** → **Layout** opens a new virtual instrumentation interface. With the controls on the **Virtual Instruments** toolbar, the control interface can be established. For instance, scroll bars can be used to access the parameters of the PD controllers. Scopes can be used to display the position of the ball and the control signals. The final control interface designed is shown in Fig. 9.30.

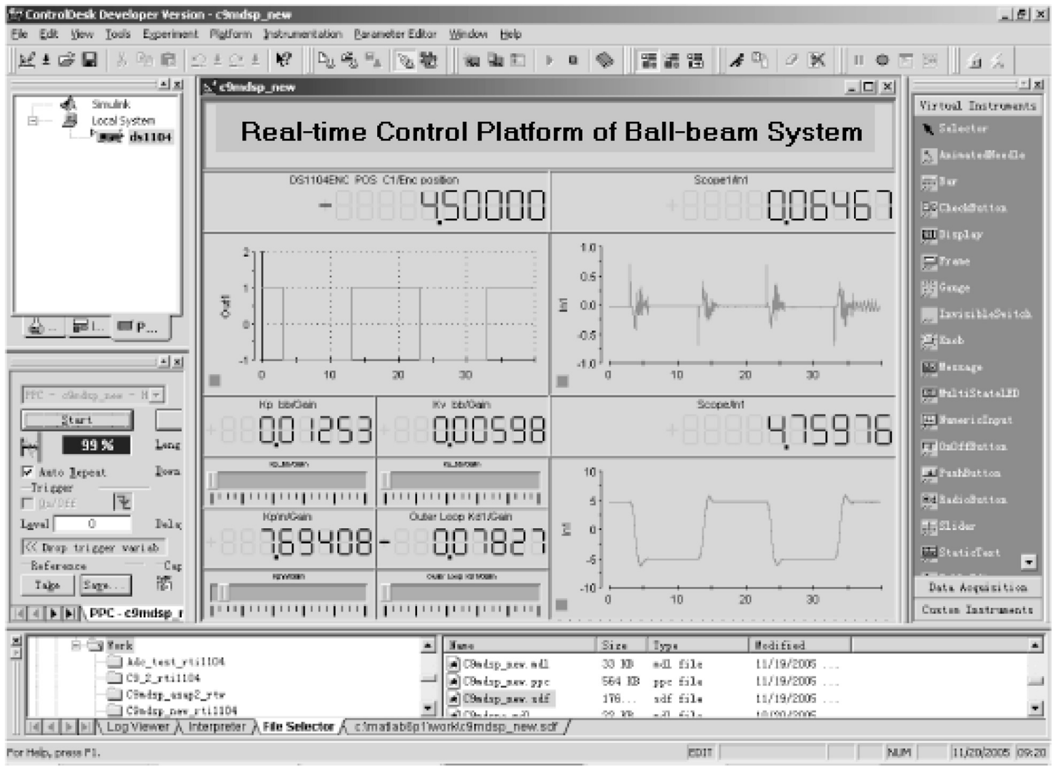


**Figure 9.30** Control interface constructed by Control Desk.

Select the **Platform** pane, and the file c9mdsp.ppc saved earlier can be loaded with the file dialog box. The connection between the interface and the *.ppc file generated with Simulink can be established. The parameters in the Simulink model must be associated with the controls in the interface by dragging the Simulink variable names to the relevant controls in the Control Desk.

With the control interface established, real-time control of the actual plant can be performed. The responses of the ball and beam system are as shown in Fig. 9.30. Note that the control signal obtained here is the actual signal written to the **DAC** block from dSPACE, and this should be multiplied by 10 to get the physical signal in the system, which varies in the interval $(-10, 10)$, and it is similar to that obtained with Quanser.

It can be seen that the controller established in Simulink can be used to directly control the practical plant in real time. Also, on-line controller parameter tuning is also allowed. For instance, in the example, the scroll bars can be used to change the parameters of the PD controller, and the control results can be immediately obtained.

The controller and parameters thus created can be downloaded to the actual controllers, such that control can be achieved without MATLAB or dSPACE environments.

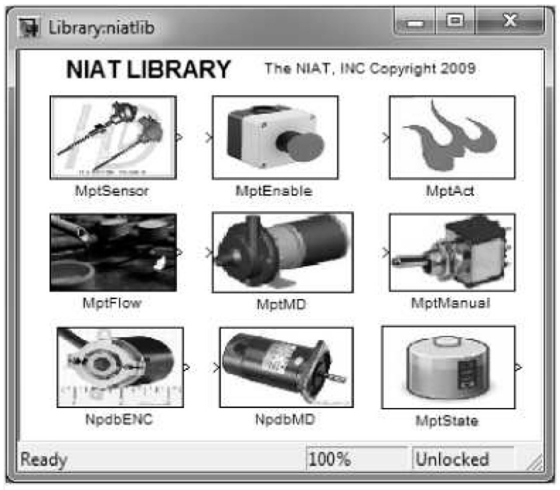
## 9.5 Low Cost Solutions with NIAT

The platform for hardware-in-the-loop simulation and real-time experimentation developed by NIAT is a low-cost solution to experimental education in universities. The Pendubot (double under-actuated inverted pendulum) control system [6] and the water tank control system are ideal research and experimental platforms in teaching real-time control and for MATLAB/Simulink based hardware-in-the-loop simulation. In this section, NIAT blockset is presented first and the modeling and numerical simulation of Pendubot control systems are demonstrated, followed by the hardware-in-the-loop simulation with NIAT.

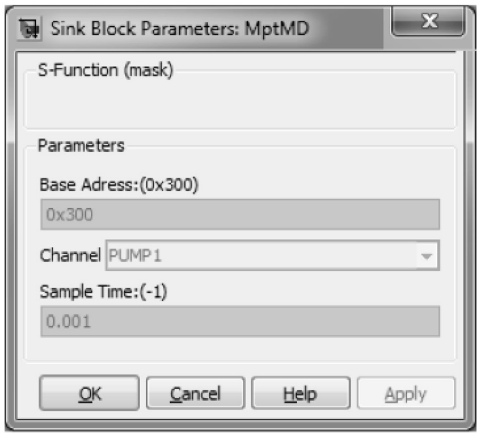
### *9.5.1 Commonly Used Blocks in the NIAT Library*

The plants developed by NIAT include motion control series and process control series devices. Motion control series include different inverted pendulums, Pendubot and manipulators, while process control series include temperature and flow rate control systems, water-level control systems and double water tank systems.

Universal networked controllers are developed in the NIAT platform with D/A converter **MptSensor**, A/D converter **MptMD**, motor encoder input and output **NpdbENC**. These ports can be used to connect the computer to the actual plant, so that a closed-loop structure can be established. NIAT provides MATLAB/Simulink compatible controller and real-time control software EasyControl. This software can be used to activate the executable code generated by Simulink model, so that real-time control tasks can be achieved. When the software EasyControl is installed, the `niatlib` command can be used to display the Simulink model group, as shown in Fig. 9.31(a).



(a) EasyControl blockset


Sink Block Parameters: MptMD

S-Function (mask)

Parameters

Base Adress:(0x300)
0x300

Channel PUMP1

Sample Time:(-1)
0.001

(b) **MptMD** dialog box

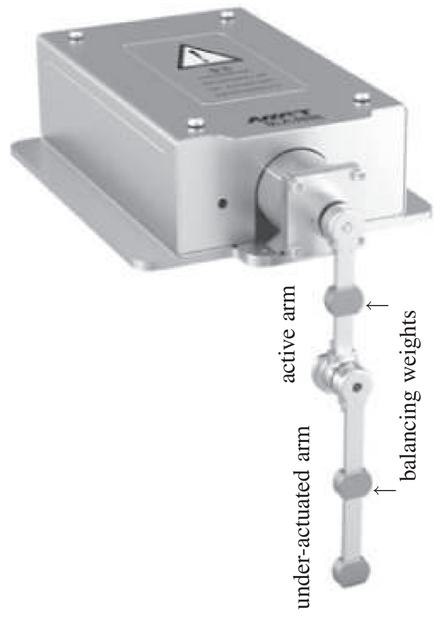
**Figure 9.31** EasyControl blockset and the dialog box.

Double click the **MptMD** motor block, the parameter dialog box can be opened as shown in Fig. 9.31(b). It can be seen that the **Base Address** and **Sample Time** need to be specified.

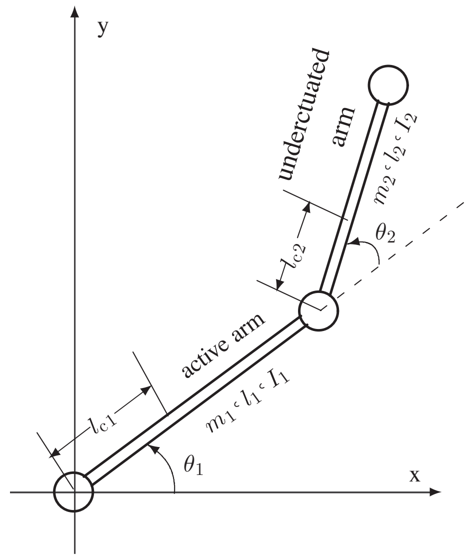
### *9.5.2 Modeling and Simulation of Pendubot Systems*

The photograph of the Pendubot experimental device, developed by NIAT, is shown in Fig. 9.32(a). In the photograph, the Pendubot has settled in downward position. The control objective is to swing up the Pendubot arms such that they can maintain an upright position.

JWST357-c09 JWST357-Xue Printer: Yet to Come August 8, 2013 13:18 Trim: 7in × 10in



(a) Pendubot photograph



(b) Pendubot illustration

**Figure 9.32** Pendubot plant.(a) Pendubot photograph.(b) Pendubot illustration.

The Pendubot is a two-degree-of-freedom under-actuated mechanical system provided by NIAT. In the system, there is an active arm driven by a DC torque motor, and an under-actuated arm, without any driving action. Two VLT12 high-precision photoelectric encoders, with a precision of 1250 pulses/cycle. They are installed respectively on the active joint and the under-actuated joint, and these provide the position feedback signals of the arms. The sketch of the Pendubot is given in Fig. 9.32(b), where $l_1$ is the length of the active arm, $l_{c1}$ is the length from the axle to the center of gravity, $l_{c2}$ is the length from the axle to the center of gravity of the under-actuated arm, $m_1$ and $m_2$ are the masses of the two arms and $I_1$ and $I_2$ are the moments of inertia to the center of gravity of the two arms. The dynamic equation of the Pendubot system can be written as

$$\boldsymbol{D}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{C}(\boldsymbol{q}, \dot{\boldsymbol{q}})\dot{\boldsymbol{q}} + \boldsymbol{G}(\boldsymbol{q}) = \boldsymbol{\tau}, \tag{9.2}$$

where $\boldsymbol{q} = [\theta_1, \theta_2]$' $\boldsymbol{\tau} = [\tau, 0]$' $\tau$ is the torque of the active arm, the coefficient matrices are defined as

$$\boldsymbol{D}(\boldsymbol{q}) = \begin{bmatrix} a_1 + a_2 + 2a_3\cos\theta_2 & a_2 + a_3\cos\theta_2 \\ a_2 + a_3\cos\theta_2 & a_2 \end{bmatrix}, \quad \boldsymbol{G}(\boldsymbol{q}) = \begin{bmatrix} a_4\cos\theta_1 + a_5\cos(\theta_1+\theta_2) \\ a_5\cos(\theta_1+\theta_2) \end{bmatrix} \mathrm{g}, \tag{9.3}$$

$$\boldsymbol{C}(\boldsymbol{q}, \dot{\boldsymbol{q}}) = \begin{bmatrix} -a_3\dot{\theta}_2 & -a_3(\dot{\theta}_2 + \dot{\theta}_1) \\ a_3\dot{\theta}_1 & 0 \end{bmatrix} \sin\theta_2, \tag{9.4}$$

and $a_1 = m_1 l_{c1}^2 + m_2 l_1^2 + I_1$, $a_2 = m_2 l_{c2}^2 + I_2$, $a_3 = m_2 l_1 l_{c2}$, $a_4 = m_1 l_{c1} + m_2 l_1$, $a_5 = m_2 l_{c2}$. The parameters were identified as $a_1 = 0.0106$ kg·m$^2$, $a_2 = 0.00597$ kg·m$^2$, $a_3 = 0.00509$ kg·m$^2$, $a_4 = 0.0751$ kg·m, $a_5 = 0.0367$ kg·m under the current balancing weights. The objective of the balancing control is that when $\theta_1$ approaches the set-point $\theta_{1\mathrm{d}}$, the under-actuated arm maintains an upright position, and $\theta_2$ settles down at the given balancing point $\theta_{2\mathrm{d}}$ such that $\theta_{1\mathrm{d}} + \theta_{2\mathrm{d}} = 90^\circ$.

Since the Pendubot system is driven by a DC motor, in the mathematical model of the actuator and DC motor, the control torque $\tau(t)$ of the active arm can be obtained from

$$\tau(t) = K_1 K_2 u(t), \tag{9.5}$$

where $K_1$ is the torque constant of the motor and $K_2$ is the gain of the actuator. In this control system, these two parameters are 0.4125 N·m/A and 2.68 A/V respectively. The signal $u(t)$ is the input voltage of the actuator, generated directly by the controller.

It can be seen from (9.2) that when the substate vectors are selected as $\boldsymbol{x}_1 = \boldsymbol{q}, \boldsymbol{x}_2 = \dot{\boldsymbol{q}}$, the original system can be written as

$$\begin{bmatrix} \dot{\boldsymbol{x}}_1 \\ \dot{\boldsymbol{x}}_2 \end{bmatrix} = \begin{bmatrix} \boldsymbol{x}_2 \\ \boldsymbol{D}^{-1}(\boldsymbol{x}_1)\left[\boldsymbol{\tau} - \boldsymbol{C}(\boldsymbol{x}_1, \boldsymbol{x}_2)\boldsymbol{x}_2 - \boldsymbol{G}(\boldsymbol{x}_1)\right] \end{bmatrix}. \tag{9.6}$$

More specifically, the state variables can be selected as $x_1 = \theta_1$, $x_2 = \theta_2$, $x_3 = \dot{\theta}_1$, $x_4 = \dot{\theta}_2$. Thus the state space model of the system can be written as

$$\begin{cases} \dot{\boldsymbol{x}}(t) = \boldsymbol{F}(t, \boldsymbol{x}(t), \boldsymbol{u}(t)) \\ \boldsymbol{y}(t) = \boldsymbol{x}(t). \end{cases} \tag{9.7}$$

It can be seen that, the state space model is suitable to be described by an S-function. In the model, there are four continuous state variables and no discrete states. There is one input signal $u(t)$ and four output signals, that is, the four states. There are also four additional variable vectors, the coefficient $\boldsymbol{a} = [a_1, \cdots, a_5]$, the initial state vector $\boldsymbol{x}_0$ and constants $K_1$ and $K_2$. The system model can be described by the following S-function

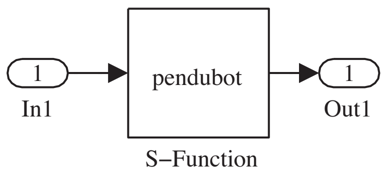
```
function [sys,x0,str,ts]=pendubot(t,x,u,flag,a,x0 s,K1,K2)
switch flag,
case 0
   sizes = simsizes; sizes.NumContStates=4; sizes.NumDiscStates=0;
   sizes.NumOutputs=4; sizes.NumInputs=1;
   sizes.DirFeedthrough=0; sizes.NumSampleTimes=1;
   sys=simsizes(sizes); x0=x0 s; str=[]; ts=[0 0];
case 1
   c1=cos(x(1)); c2=cos(x(2)); c12=cos(x(1)+x(2)); x2=x(3:4);
   D=[a(1)+a(2)+2*a(3)*c2, a(2)+a(3)*c2; a(2)+a(3)*c2, a(2)];
   G=[a(4)*c1+a(5)*c12; a(5)*c12]*9.81;
   C=[-a(3)*x(4), -a(3)*(x(3)+x(4)); a(3)*x(3), 0]*sin(x(2));
   sys=[x2; inv(D)*([K1*K2*u; 0]-C*x2-G)];
case 3, sys=x;
case {2, 4, 9},  sys = [];
otherwise, error(['Unhandled flag=',num2str(flag)]);
end
```

The four additional variables can be assigned with the following statements

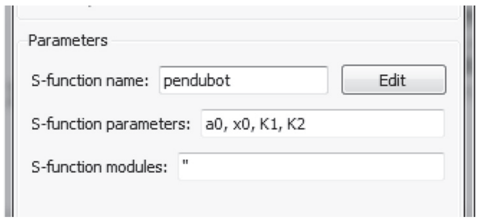
```
>> a0=[0.0106, 0.00597, 0.00509, 0.0751, 0.0367];
   x0=[1.3; 0.3; 0; 0]; K1=0.4125; K2=2.68;
```

The plant model can be linearized, and a controller can be designed from such a model. To get the linearized model, a masked Simulink model can be constructed, as shown in Fig. 9.33(a), where the dialog box of the Pendubot model is given in Fig. 9.33(b). For the equilibrium point $(90^\circ, 0^\circ)$ with zero input, that is, $u_0 = 0$, the linearized model can be obtained with the following MATLAB statements

```
>> G=linearize('c9mniat3',[pi/2; 0; 0; 0],0)
```



(a) Pendubot linearization



Parameters
S-function name: pendubot
S-function parameters: a0, x0, K1, K2
S-function modules: ''

(b) S-function dialog box

**Figure 9.33** Simulink model of the Pendubot system (model name: c9mniat3).

The linear state expression can be obtained as

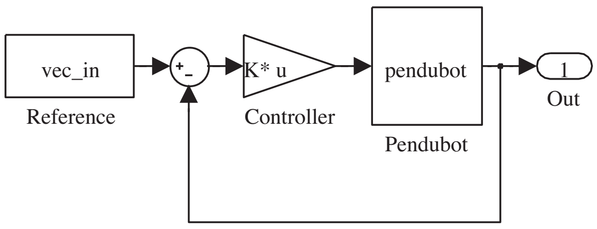
$$\boldsymbol{A} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 68.651 & -49.033 & 0 & 0 \\ -66.876 & 151.14 & 0 & 0 \end{bmatrix}, \quad \boldsymbol{B} = \begin{bmatrix} 0 \\ 0 \\ 176.59 \\ -327.15 \end{bmatrix}, \quad \boldsymbol{C} = \boldsymbol{I}_{4\times 4}, \boldsymbol{D} = \boldsymbol{0}_{4\times 1}.$$

Assume that, at initial time, the Pendubot system is settled around its equilibrium point $(90^\circ, 0^\circ)$. Linearization can be performed on the original system, and based on the results, an optimum linear quadratic controller can be designed, with state feedback vector $\boldsymbol{K} = [-22.3589, -19.8736, -4.0484, -2.4394]$. The Simulink model with state feedback control is constructed as shown in Fig. 9.34(a), with the set-point input vector given by $\boldsymbol{x}_\mathrm{d} = [\pi/2, 0, 0, 0]$.

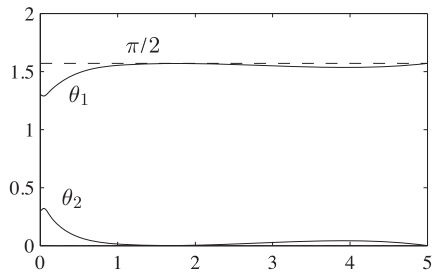
Assume that the control objective is to keep the two arms in an upright position, that is, $\theta_{1\mathrm{d}} = 90^\circ$ and $\theta_{2\mathrm{d}} = 0^\circ$. The following MATLAB statements can be executed and the simulation results are obtained as shown in Fig. 9.34(b).

```
>> vec_in=[pi/2 0 0 0]; K=[-22.3589,-19.8736,-4.0484,-2.4394];
   [t,x,y]=sim('c9mpendsim',[0,5]); plot(t,y(:,1:2))
```

It can be seen that when the initial conditions are near the upright position, that is, $\theta_{10} = 1.3$ rad$= 74.5^\circ$, $\theta_{20} = 0.3$ rad$= 17.2^\circ$, within a very short time the system settles down at its equilibrium point, such that $\theta_2 = 0.5\pi$ rad $= 90^\circ$, $\theta_2 = 0^\circ$.



(a) Pendubot control model (c9mpendsim)



(b) simulation results

**Figure 9.34** Pendubot control system simulation.

If the control object is to have $\theta_{1\mathrm{d}} = 1$ rad and $\theta_{2\mathrm{d}} = (\pi/2 - \theta_{1\mathrm{d}})$ rad, the following statements can be used and the simulation results are then as shown in Fig. 9.35.

```
>> vec_in=[1 pi/2-1 0 0]; [t,x,y]=sim('c9mpendsim',[0,5]);
   plot(t,y(:,1:2),t,y(:,1)+y(:,2))
```

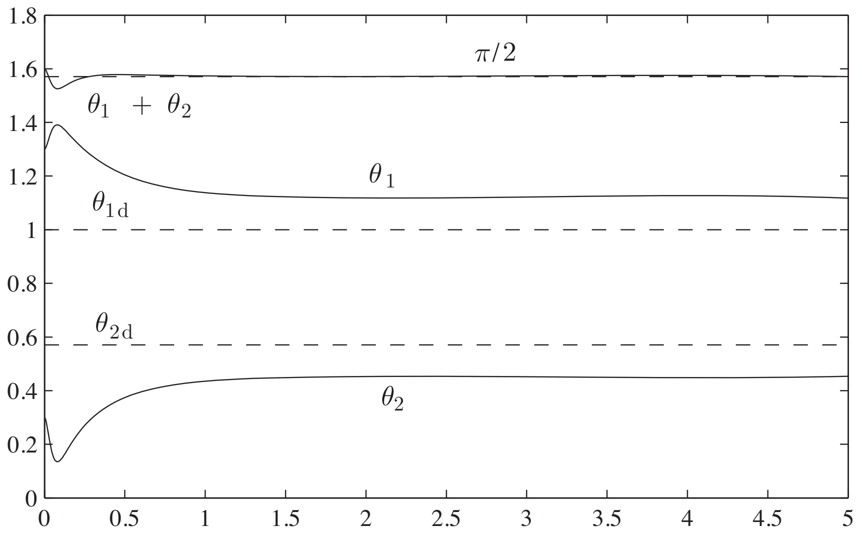
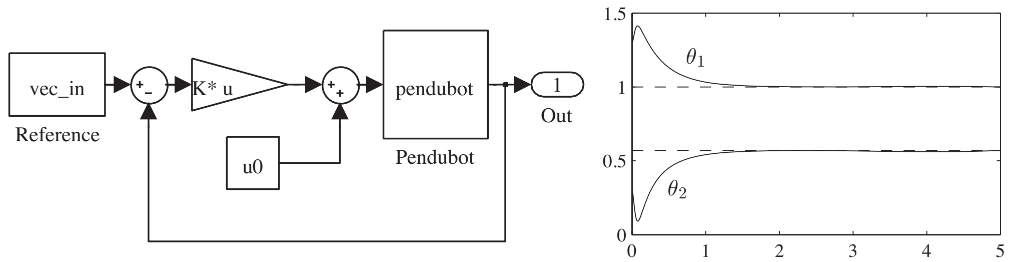


**Figure 9.35** Simulation results when the control objectives are changed.

An optimum linear quadratic regulator alone cannot be used to achieve the expected $\boldsymbol{x}_\mathrm{d}$ equilibrium point. If such a result is to be achieved, gravity compensation needs to be introduced so that $u(t) = \boldsymbol{K}(\boldsymbol{x}_\mathrm{d} - \boldsymbol{x}(t)) + u_0$, where the gravity compensation can be obtained from

$$u_0 = \frac{g}{K_1 K_2}\Big[a_4 \cos\theta_{1\mathrm{d}} + a_5 \cos(\theta_{1\mathrm{d}} + \theta_{2\mathrm{d}})\Big] \tag{9.8}$$

With the gravity compensation technique, the new Simulink model can be constructed as shown in Fig. 9.36(a). The calculation of the gravity compensation constant $u_0$ can be embedded in the **PreLoadFcn** property of the model. For this system, $u_0 = 0.3601$ V. Simulation results under the compensation can be obtained as shown in Fig. 9.36(b). In this way, the set-point of the angle $\theta_{1\mathrm{d}}$ can be arbitrarily chosen. From (9.8), if $\theta_{1\mathrm{d}} = 90^\circ$, $\theta_{2\mathrm{d}} = 0^\circ$, the compensation constant is $u_0 = 0$, which means that the compensation method also works for the original upright control problem.



(a) Pendubot model (model name: c9mpendsim_c) (b) compensated simulation results

**Figure 9.36** Pendubot system with gravity compensation.

## *9.5.3 Hardware-in-the-loop Simulation Experiment of Pendubot Systems*

With the mechanism of hardware-in-the-loop simulation, the actual Pendubot system can be used to substitute the mathematical model, such that the new hardware-in-the-loop Simulink model shown in Fig. 9.37 can be constructed, where the two angles $\theta_1$ and $\theta_2$ can be measured with the **NpdbENC** encoder blocks, and the gain of $2\pi/(4 \times 1250)$ can be used to convert it to radians. The derivatives of the angles can be obtained with the approximate derivative block with $80s/(s + 80)$. The control signal $u$ can be calculated with the **MptMD** block, and fed to the driving motor. The signal can be applied to the driving motor through the A/D converter, to form the closed-loop control system. The model can be used to control directly the real Pendubot system in real-time. The control results shown in Fig. 9.38 can be achieved.

The optimum linear quadratic controller can be used to move the two arms of the Pendubot relatively quickly into the upright position from their initial states, and can remain in that position. This means that the system has clear robustness.

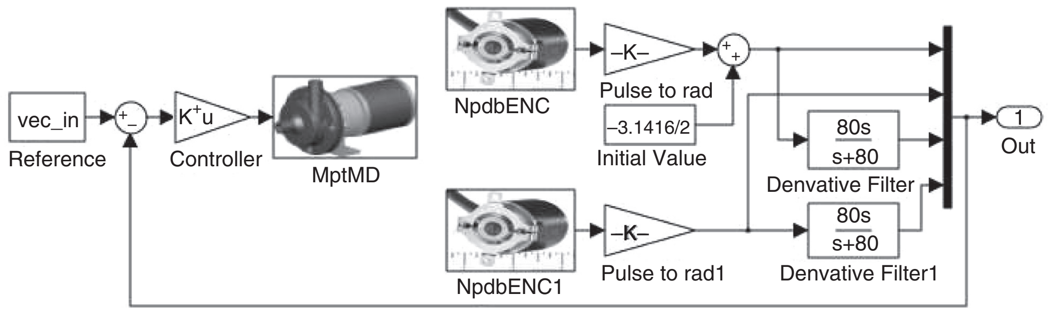
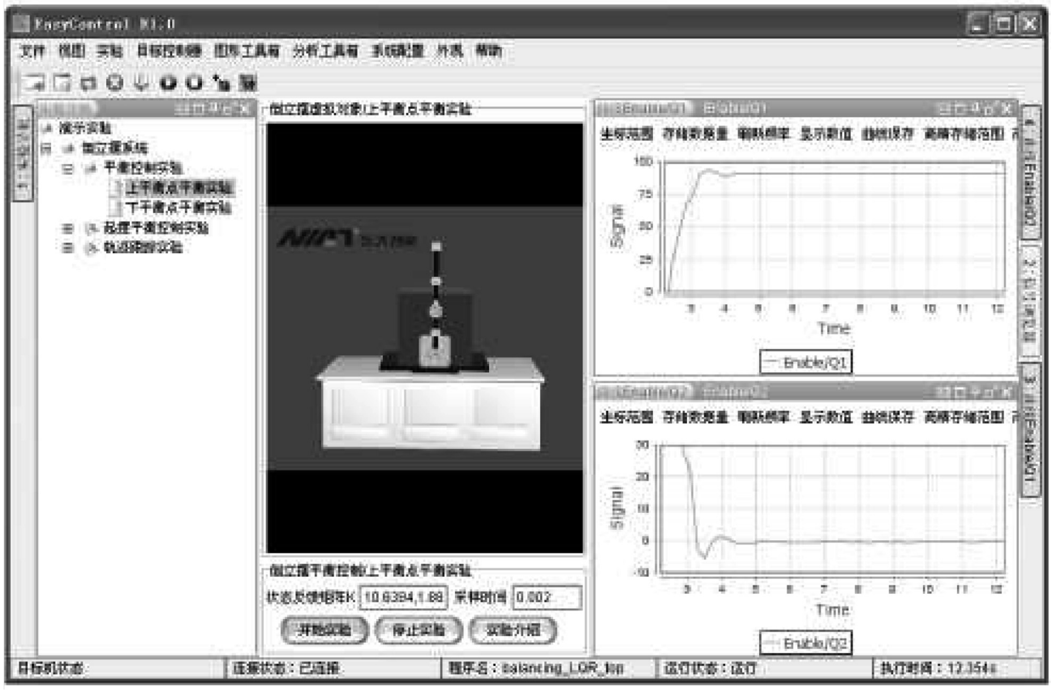


**Figure 9.37** Pendubot hardware-in-the-loop model (model file: balancing_LQG_top).


EasyControl V1.0
文件 视图 实验 目标控制器 图形工具箱 分析工具箱 系统配置 外观 帮助
演示实验
倒立摆系统
平衡控制实验
上平衡点平衡实验
下平衡点平衡实验
起摆平衡控制实验
轨迹跟踪实验
倒立摆虚拟对象|上平衡点平衡实验
坐标范围 存储数据量 刷新频率 显示数值 曲线保存 离线存储范围
Signal
Time
Enable/Q1
Enable/Q2
倒立摆平衡控制上平衡点平衡实验
开始实验
停止实验
实验介绍

状态反馈矩阵K 10.6384,1.88 采样时间 0.002

目标机状态 | 连接状态：已连接 | 程序名：balancing_LQR_top | 运行状态：运行 | 执行时间：12.354s

**Figure 9.38** Real-time control result in Pendubot system.

## 9.6 HIL Solutions with Even Lower Costs

It can be seen from previous sections that HIL and rapid prototyping solutions can be obtained by products such as those from Quanser and NIAT. In this section, we will introduce another HIL solution using low-cost hardware based on Arduino™ for hardware-in-the-loop real-time control rapid prototyping. This solution is not only low cost but also very portable and it can be taken home for doing real-time closed-loop systems control experiments. This is particularly useful in a mobile age.

### *9.6.1 Arduino Interface Installation and Settings*

In the latest versions of MATLAB such as R2012b, Simulink provides support packages for third-party products such as Arduino, LEGO Mindstorms™ and PandaBoard™.

This feature can be accessed in any Simulink model window through the **Tools** → **Run on Target Hardware** → **Install/Update Support Package...** menu item, as shown in Fig. 9.39, or simply by typing `targetinstaller` in the MATLAB command window. Alternatively, in older versions of MATLAB, users can go to the **Tools** menu in a Simulink model window and click **Add-Ons** → **Get Hardware Support Packages** menu item.

After the above action, users will be directed to the **Target Installer** in which the target support packages are listed. For a complete list of the supported products, see [7]. On completing the installation procedure described previously, the "Simulink Arduinolib" is automatically embedded into the Simulink Library Browser and is ready to use. It includes the blocks shown in Fig. 9.40.

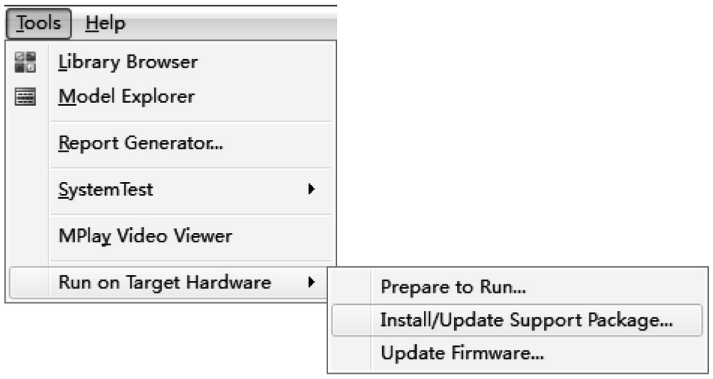


**Figure 9.39** Additional package support menu.

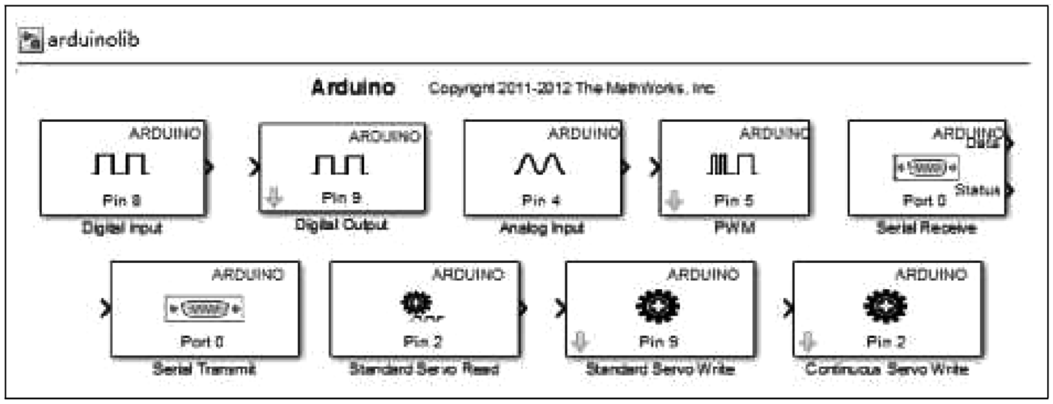


**Figure 9.40** Simulink Arduinolib blockset.

The Simulink model created in this way is set up to run on the target, which is the Arduino board in this case. Hence, pre-settings need to be performed prior to execution. Again, refering to Fig. 9.39, clicking the **Prepare to Run** menu item will pop up the standard Simulink configuration dialog box. From the dialog box, if the **Run on Target Hardware** is selected, the **Arduino Uno** item can be specified from the **Target hardware** list box.

Follow the steps of specifying the serial communication port and Baud rate, then the popup menu appears as in Fig. 9.39. Now, the Simulink model can run on the Arduino target as a stand-alone application.

## *9.6.2 Applications of Arduino Control*

There are two main ways of interfacing with the open-source hardware Arduino:

1) interfacing with it as a normal target through code generation

2) "virtual machine style" host-target communication.

The block library given earlier can be regarded as the first approach, while the Simulink model can be executed directly with the connected hardware.

Here, the second approach is presented as a case study on Arduino Uno. This interfacing approach does not support code generation, meaning that it does not generate code to download to the Arduino target. Instead, an I/O description file needs to be fed into the Arduino in advance as if it were firmware. Afterwards, all the I/Os on the Arduino board can be accessed either through Simulink or through MATLAB commands using serial communication. This is similar to the mechanism of a "Java virtual machine", which manages the bottom layer hardware while providing users with generic APIs.

To use this approach, a third-party Arduino support package is needed which is freely available from MATLAB Central [8], and it can be initiated with `arduino_io_lib`. The blocks in the package are shown in Fig. 9.41.

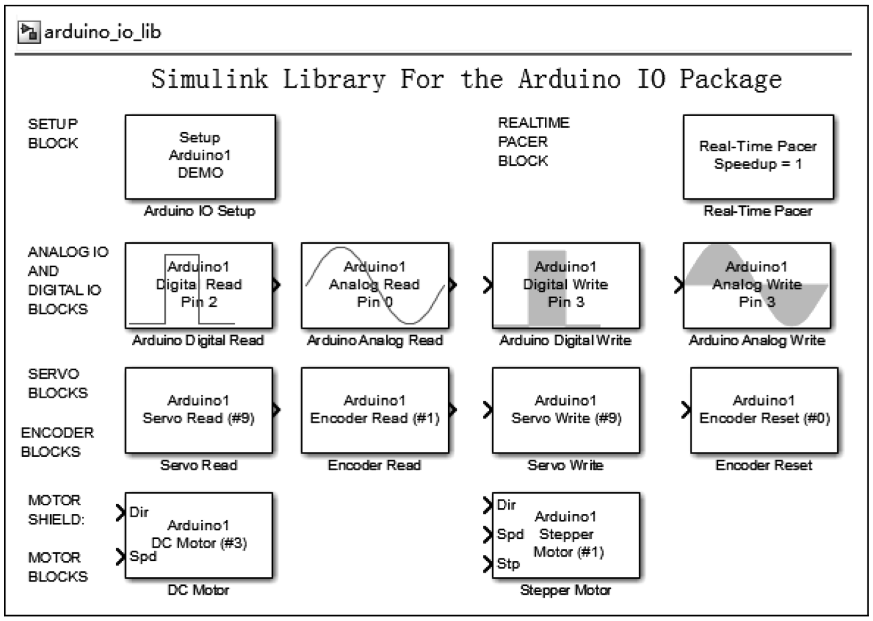
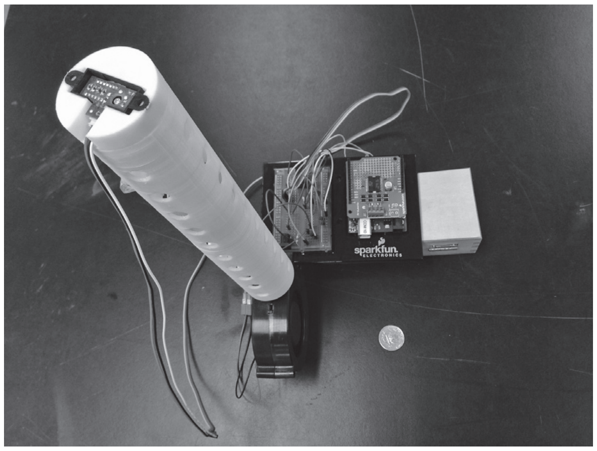


**Figure 9.41** The blocks in the Arduino support package.

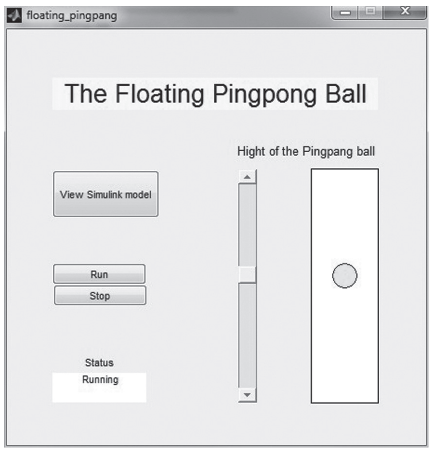
**Example 9.5** *This example shows a simple closed-loop control system developed on the Simulink Arduino support package. The hardware set-up is shown in Fig. 9.42 (a). The control objective is to make the ball track the height set-point and, after settling down, maintain its height against disturbances.*

*A MATLAB GUI, shown in Fig. 9.42 (b), is designed to visualize the control and response. Users can adjust the height set-point by sliding the bar in the GUI. The Simulink model is displayed in Fig. 9.43, in which the* ***Arduino analog read Pin 0*** *is connected to an infrared (IR) sensor on top of the tube, and* ***Arduino analog write Pin 3*** *is connected to a motor-driven fan to blow the ball. The IR sensor is used to detect the height of the ball within the tube.*

*It is worth mentioning that, during the running of such an application, the serial communication between the host PC and the Arduino target cannot be interrupted because the signal processing is in fact being carried out on the PC and only the sensing and actuation is performed on the Arduino.*



(a) experimental setup



(b) control GUI

**Figure 9.42** The ping-pong ball floating experiment.

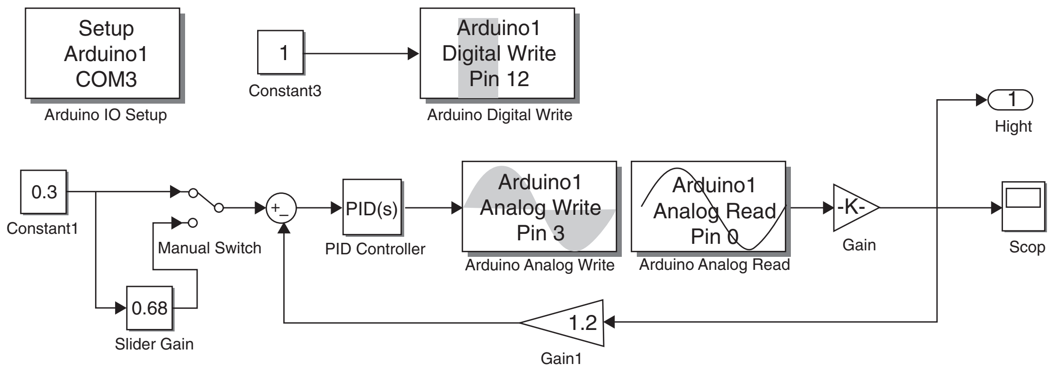


**Figure 9.43** The Simulink model (model name: pingpang_sim).

## 9.6.3 *The MESABox*

An educational experimental toolbox, named MESABox™, based on the Arduino Uno core and peripheral hardware components, was developed in the MESA (Mechatronics, Embedded Systems and Automation) Laboratory at University of California, Merced, for the purpose of making the traditionally cumbersome mechatronics laboratories into a small-sized portable handset [9]. The cost of the key components in MESABox is approximately $180. The assembled box is shown in Fig. 9.44.

In MATLAB R2012b, varieties of “Apps” such as the “Floating ball” and “Fan-and-Plate” have been developed for the MESABox to support its educational usage in the MESA LAB. They are seamlessly integrated into MATLAB as a Toolbox/App.

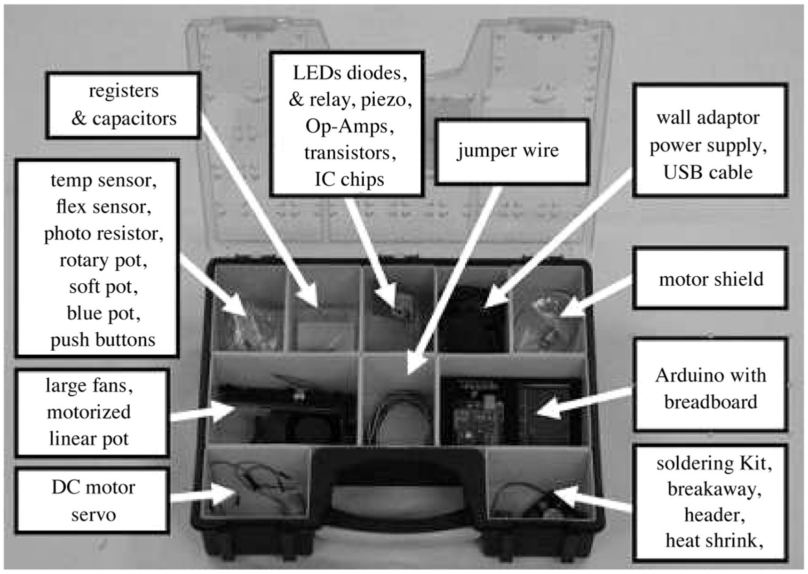

registers & capacitors
LEDs diodes, & relay, piezo, Op-Amps, transistors, IC chips
jumper wire
wall adaptor power supply, USB cable
temp sensor, flex sensor, photo resistor, rotary pot, soft pot, blue pot, push buttons
motor shield
large fans, motorized linear pot
Arduino with breadboard
DC motor servo
soldering Kit, breakaway, header, heat shrink,

**Figure 9.44** MESABox for take-home mechatronics and control labs.

## Exercises

**9.1** To help understand the concepts and applications of different simulation modes in Simulink with practical examples, the modes **Normal**, **Accelerator** and **External** should be tested and suitable applications of different modes explored.

**9.2** Consider the four-bar mechanism studied in Chapter 7, with the model name c7mmech4.mdl. Convert the model into a standalone executable file, and run the program in the DOS command box environment and compare the simulation results.

**9.3** Select one of the Simulink models created earlier. Use two methods to simulate the model in real time. The executable simulation program can be executed on host and target computers. Also try to control the target computer through internet and complete the simulation process.

**9.4** Design a virtual instrumentation interface that acquires a signal from an actual A/D converter, and display the signal with gauges. Use the tracker-differentiator to display the original signal and its derivative. Select suitable parameters for the tracker-differentiator and compare the results. Note that in real-time simulation, S-functions written in MATLAB are not supported. An S-function written in C should be used instead.

**9.5** If the reader has access to tools such as Quanser, dSPACE, NIAT or Arduino Uno, try to construct a nonlinear plant model, and design a controller for it.

**9.6** In practical control systems, the derivatives of some signals cannot be measured easily. Alternative indirect methods should be used instead, such as the filters $50s/(s+50)$, $80s/(s+80)$. Of course, the tracker-differentiator discussed in Section 6.3.2 can be used. Construct a simulation model for the Pendubot system with the tracker-differentiator and find suitable parameters.

**9.7** Consider the mathematical model of the Pendubot system. Try to design a switching controller such that, starting at an initial downward position, it will swing up the Pendubot, and when the arms enter the equilibrium point, the controller switches to an optimum linear quadratic controller.

## References

[1] The MathWorks Inc. Real-time windows target user's guide, 2010
[2] dSPACE Inc. DS1104 R&D controller board installation and configuration guide, 2001
[3] Quanser Inc. SRV02 – Series rotary experiment # 1: Position control, 2002
[4] Quanser Inc. SRV02 – Series rotary experiment # 3: Ball & beam, 2002
[5] dSPACE Inc. Control Desk – experiment guide, Release 3.4, 2002
[6] Mechatronic Systems Inc. Pendubot model P-2 user's manual, 1998
[7] MathWorks. Hardware for project-based learning. `http://www.mathworks.com/academia/hardware-resources/index.html`, 2012
[8] MathWorks Classroom Resources Team. MATLAB Support Package for Arduino (aka ArduinoIO Package). MATLAB Central # 32374, 2011
[9] B Stark, Z Li, B Smith, and Y Q Chen. Take-home mechatronics control labs: a low-cost personal solution and educational assessment, Proceedings of the ASME 2013 International Design Engineering Technical Conferences & Computers and Information in Engineering Conference, Portland, Oregon, USA, 2013