

Documentation for Single Board Heater System

Rakhi R
Rupak Rokade
Inderpreet Arora
Kannan M. Moudgalya
Kaushik Venkata Belusonti



IIT Bombay
May 2, 2014

Contents

List of Scilab Code	2
1 Block diagram explanation of Single Board Heater System	3
1.1 Microcontroller	3
1.1.1 PWM for heat and speed control	4
1.1.2 Analog to Digital conversion	6
1.2 Instrumentation amplifier	6
1.3 Communication	7
1.3.1 Serial port communication	8
1.3.2 Using USB for Communication	8
1.4 Display and Resetting the setup	10
2 Using Scilab with Single Board Heater System	12
2.1 Accessing Single Board Heater system on a Windows System .	13
2.1.1 Installing necessary driver and COM Port Settings . . .	13
2.1.2 Configuring Scilab	14
2.2 Accessing Single Board Heater system on a Linux System . . .	15
3 PRBS modelling and implementation of pole-placement controller	22
3.1 Issues with step testing and alternate approach	24
3.2 Step by step procedure to do PRBS testing	25
3.3 Determination of discrete time transfer function	26
3.3.1 Performing PRBS testing on SBHS, virtually	27
3.4 Scilab Code	27

List of Scilab Code

3.1	ser_init.sce	27
3.2	imc.sci	28
3.3	imc_virtual.sce	32
3.4	imc_virtual.sci	32

Chapter 1

Block diagram explanation of Single Board Heater System

Figure 1.1 shows the block diagram of 'Single Board Heater System'. Microcontroller ATmega16 is used and is the heart of the setup. Two serial communication ports namely RS232 and USB can be used to connect the setup to a computer. A particular port can be selected by setting the jumper to its appropriate place. The communication between PC and setup takes place via a serial to TTL interface. The microcontroller can be programmed with the help of an In-system programmer port (ISP) available on the board. The μC operates the Heater and Fan with the help of separate drivers. The driver comprises of a power MOSFET. A temperature sensor is used to sense the temperature and fed to the μC through an Instrumentation amplifier. Some required parameter values are displayed along with some LED indications.

1.1 Microcontroller

Some salient features of ATmega16 are listed below

1. 32 x 8 general purpose registers.
2. 16K Bytes of In-System Self-Programmable flash memory
3. 512 Bytes of EEPROM
4. 1K Bytes of internal Static RAM (SRAM)

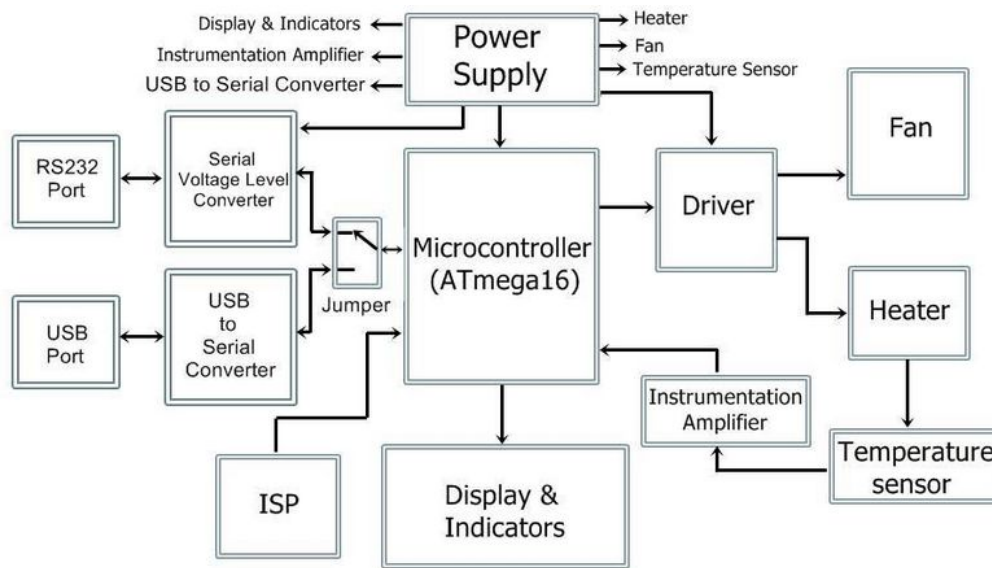


Figure 1.1: Block Diagram

5. Two 8-bit Timer/Counters
6. One 16-bit Timer/Counter
7. Four PWM channels
8. 8-channel,10-bit ADC
9. Programmable Serial USART
10. Up to 16 MIPS throughput at 16 MHz

Microcontroller plays a very important role. It controls every single hardware present on the board, directly or indirectly. It executes various tasks like, setting up communication between PC and the equipment, controlling the amount of current passing through the heater coil, controlling the fan speed, reading the temperature, displaying some relevant parameter values and various other necessary operations.

1.1.1 PWM for heat and speed control

The Single Board Heater System has a Heater coil and a Fan. The heater assembly consists of an iron plate placed at a distance of about 3.5mm from a nichrome

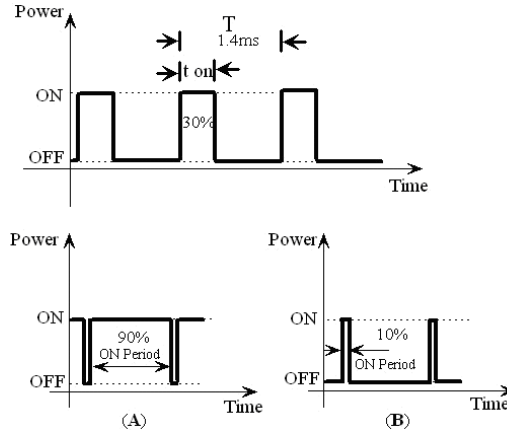


Figure 1.2: Pulse Width Modulation (A): On time is 90% of the total time period, (B): ON time is 10% of total time period

coil. When current passes through the coil it gets heated and in turn raises the temperature of the iron plate. We are interested to alter the heat generated by the coil and also the speed at which the fan is operated. There are many ways to control the amount of power delivered to the Fan and Heater. We are using the PWM technique. PWM or pulse width modulation is a process in which the duty cycle of the square wave is modulated.

$$\text{Duty cycle} = \frac{T_{ON}}{T} \quad (1.1)$$

Where T_{ON} is the ON time of the wave and T is the total time period of the wave. Power delivered to the load is proportional to $T - ON$ time of the signal. This is used to control the current flowing through the heating element and also speed of the fan. An internal timer of the microcontroller is used to generate a square wave. The ON time of the square wave depends on a count value. Hence, by varying this count value one can vary the width of the waveform. Therefore, by using this technique, PWM waveform is generated at the appropriate pin of the microcontroller. This generated PWM waveform is used to control the power delivered to the load (Fan and Heater). A MOSFET is used to switch at the PWM frequency which indirectly controls the power delivered to the load. A separate MOSFET is used for the two loads. The timer is operated at 244Hz.

1.1.2 Analog to Digital conversion

As explained earlier, the heat generated by the heater coil is passed to the iron plate through convection. We would now want to measure the temperature of this plate. For this purpose a temperature sensor AD590 is used. Some of the salient features of AD590 include

1. Linear current output: $1\mu\text{A/K}$
2. Wide range: -55°C to $+150^\circ\text{C}$
3. Sensor isolation from case
4. Low cost

The output of AD590 is then fed to the microcontroller through an Instrumentation amplifier. The signal so obtained would be in analog form. It has to be converted into digital form to make it understandable for the microcontroller. An ADC would be an obvious solution. ATmega16 features an 8-channel, 10 bit successive approximation ADC with $0\text{--}V_{cc}$ (0 to V_{cc}) input voltage range. An interrupt is generated on completion of analog to digital conversion. Here, ADC is initialized to have $206\mu\text{sec}$ of conversion time. Digital data so obtained is sent to computer via serial port as well as for further processing for on board display.

1.2 Instrumentation amplifier

Whenever there is a temperature measurement task at hand, instrumentation amplifiers are often used. A typical three Op-Amp Instrumentation amplifier is shown in the figure 1.3. The instrumentation amplifiers (IA) have the advantage of very low DC offsets, high input impedance, very high Common mode rejection ratio (CMRR) etc. They are mostly preferred where the sensor is located at a remote place, susceptible to signal attenuation. The IA's have a very high input impedance and hence do not load the input signal source. IC LM348 is used to construct a 3 Op-Amp IA. IC LM348 contains a set of four Op-Amps. Gain of the amplifier is given by equation 1.2

$$\frac{V_o}{V_2 - V_1} = \left\{ 1 + \frac{2R_f}{R_g} \right\} \frac{R_2}{R_1} \quad (1.2)$$

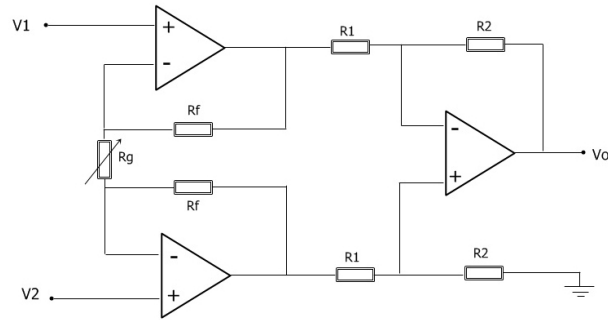


Figure 1.3: 3 Op-Amp Instrumentation Amplifier

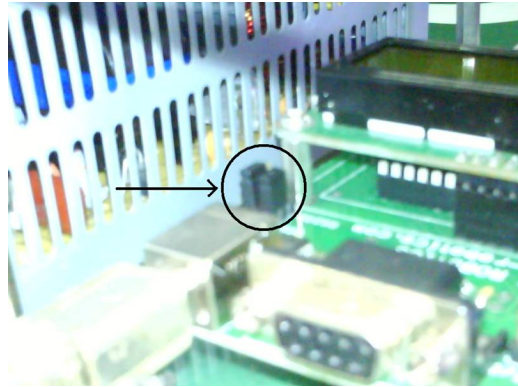


Figure 1.4: Jumper arrangement

The value of R_g is kept variable to change the overall gain of the amplifier. The signal generated by AD590 is in $\mu\text{A}/^\circ\text{K}$. It is converted to $\text{mV}/^\circ\text{K}$ by taking it across a $1\text{K}\Omega$ resistor. The $^\circ\text{K}$ to $^\circ\text{C}$ conversion is done by subtracting 237 from the $^\circ\text{K}$ result. One input of the IA is fed with the $\text{mV}/^\circ\text{K}$ reading and the other with 273mV . The resulting output is now in $\text{mV}/^\circ\text{C}$. The output of the IA is fed to microcontroller for further processing.

1.3 Communication

The set up has the facility to use either USB or RS232 for communication between set up and computer. A jumper is been provided to switch between USB and RS232. The voltages available at the TXD terminal of microcontroller are in TTL



Figure 1.5: RS232 cable

logic. The RS232 standard uses a different terminology. Voltage level below -5V is treated as logic 1 and voltage level above +5V is treated as logic 0. There are many reasons for RS232 using such terminology. One reason is to ensure error free transmission over long distances. For solving this compatibility issue an external hardware interface is required. IC MAX202 serves the purpose. IC MAX202 is a +5V RS232 transceiver.

1.3.1 Serial port communication

Serial port is a full duplex device i.e. it can transmit and receive data at the same time. ATmega16 support a programmable Universal Synchronous and Asynchronous serial receiver and transmitter(USART) . Its baud rate is fixed at 9600 bps with character size set to 8 bits and parity bits disabled.

1.3.2 Using USB for Communication

After setting the jumper to USB mode connect the set up to the computer using a USB cable at appropriate ports as shown. To make the setup USB compatible USB to serial conversion is necessary. IC FT232R is used for this purpose. You need to have proper USB driver being installed on your computer.

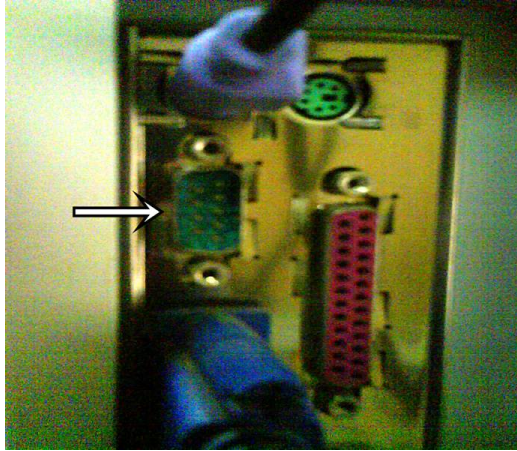


Figure 1.6: Serial port

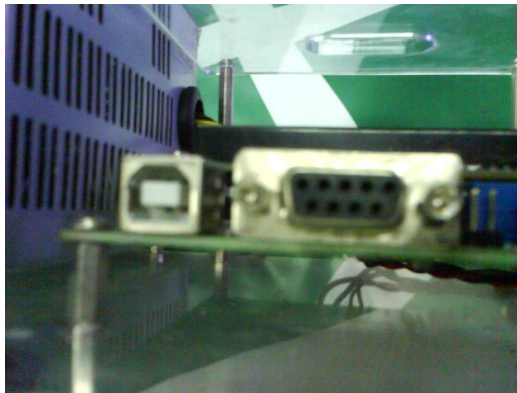


Figure 1.7: USB communication

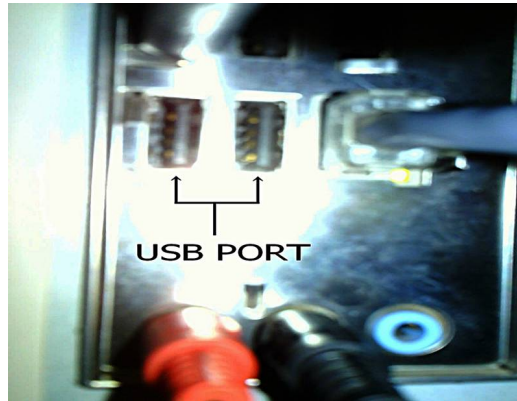


Figure 1.8: USB PORT



Figure 1.9: Display

1.4 Display and Resetting the setup

The temperature of the plate, percentage values of Heat and Fan and the machine identification number (MID) are displayed on LCD connected to the microcontroller. As shown in figure, numerals below TEMP, indicate the actual temperature of the heater plate. Numeral below HEA and FAN indicate the respective percentage value at which heater and fan are being operated. Numerals below MID corresponds to the Device identification number.

The set up could be reset at any time by pressing the reset button on it as shown in figure 1.10. Resetting the setup takes it to the standby mode where the heater current is forced to be zero and fan speed to be the maximum value. Though

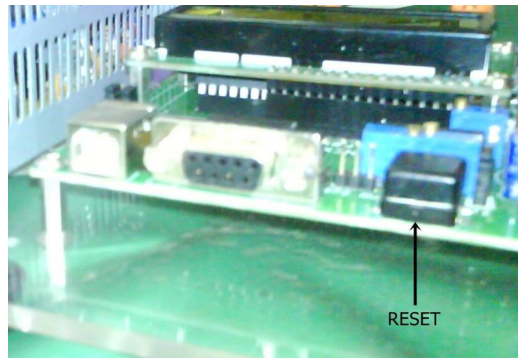


Figure 1.10: Reset

these reset values are not displayed on the LCD display but they are loaded to the appropriate units.

Chapter 2

Using Scilab with Single Board Heater System

This section explains the procedure to use Single Board Heater System with Scilab. An open loop experiment, step test is used for demonstrating this procedure. The process however remains the same for performing any other experiment explained in this document, unless specified.

Hardware and Software Requirements

For working with the Single Board Heater system, you would require the following:

1. SBHS with USB cable and power cable.
2. PC/Laptop with Scilab software installed (preferably the latest version). You may download it from:
<http://www.scilab.org/products/scilab/download>
3. FTDI Virtual Com Port driver corresponding to the OS on your PC. (You may download it from: <http://www.ftdichip.com/Drivers/VCP.htm>)
4. Example StepTest provided along with this document.

2.1 Accessing Single Board Heater system on a Windows System

This section deals with the procedure to use SBHS on a windows Operating System. The Operating System used for this document is Windows 7, 64-bit OS. In case if you are using some other Operating System or the steps explained here are not sufficient to understand, you can refer to the official document available on the main ftdi website. For doing so, go to www.ftdichip.com. On the left hand side panel, click on 'Drivers'. In the dropdown menu, choose 'VCP Drivers'. Then on the web page page, click on 'Installation Guides link'. Choose the required OS document. We would now begin with the procedure.

2.1.1 Installing necessary driver and COM Port Settings

After powering ON the SBHS and plugging in the USB cable to the PC (making sure you check the jumper settings on the board) for the very first time, the Welcome to Found New Hardware Wizard dialog box pops up. You have to choose the option Install from a list or specific location. Choose Search for best driver in these locations. Check the button Include this location in the search. Click on Browse. Specify the path where you have copied the driver (item no.3) and install it by clicking Next. Once the wizard has successfully installed the driver, your SBHS is ready for use. Please note that this procedure has to be repeated twice.

Now, we would check the communication port number assigned to the computer port to which you connect the Single Board Heater System, via an RS232 or USB cable. For checking the port number, right click on My Computer and click on properties. Here, select the hardware tab and then click on Device Manager. You would see the list of hardware devices. Look for Ports(COM & LPT) option and open it. You would see the various communication port your computer is using. If you have connected RS232 cable, then look for Communications Port(COM1) else look for USB Serial Port. For RS232 connection, the port number mostly remains COM1. For USB connection it may change to some other number. Note the appropriate COM number. This process is illustrated in figure 2.1

Sometimes the COM port number you get after connecting a USB cable is more than single digit number 9. Since the serial tool box can handle only single digit port number, it is necessary to change your COM port number. Following

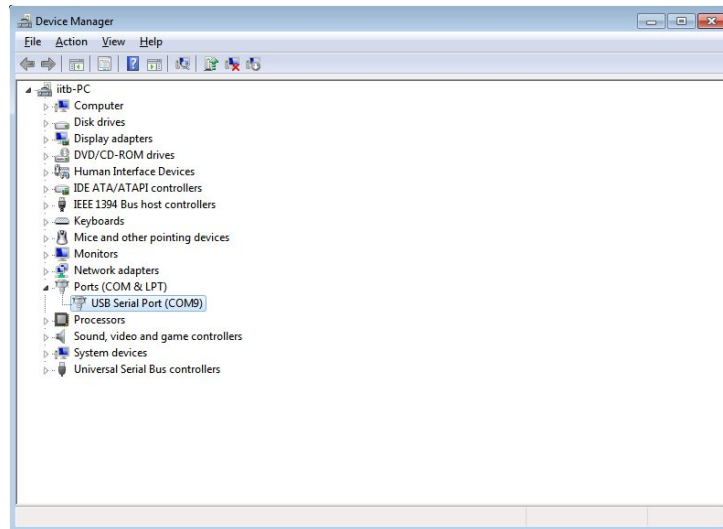


Figure 2.1: Checking Communication Port number

is the procedure to do the same. After following the procedure for knowing your com port number described above, double click that particular port. Click on Port Settings tab and then click on Advanced. In the COM port number dropdown menu, choose the port number to any other number less than 10. This procedure is illustrated in figure 2.2

2.1.2 Configuring Scilab

Launch Scilab from start menu or double click the scilab icon on the desktop. Before executing any scripts which are dependent on other files or scripts, one has to change the working directory of scilab. Doing so will tell scilab from where it has to load the other necessary files. If you have other files saved to any other directory, you have to say `getd<space>folder_path` in the scilab console. Type `ls` to see the if the files are available. Here the directory is changed to the place where the relevent files for performiong step test resides. To change the directory, click on file menu and then choose "Change directory". You can also change the directory by typing `cd<space>folder path`. Next, click on editor from the menu bar to open the scilab editor or simply type `editor` in the scilab console and open the file `ser_init.sce`. Change the port number (the first argument of the `openserial()` function) to the COM port number that you have noted down before. The second argument of the `openserial()` function requires baud rate,

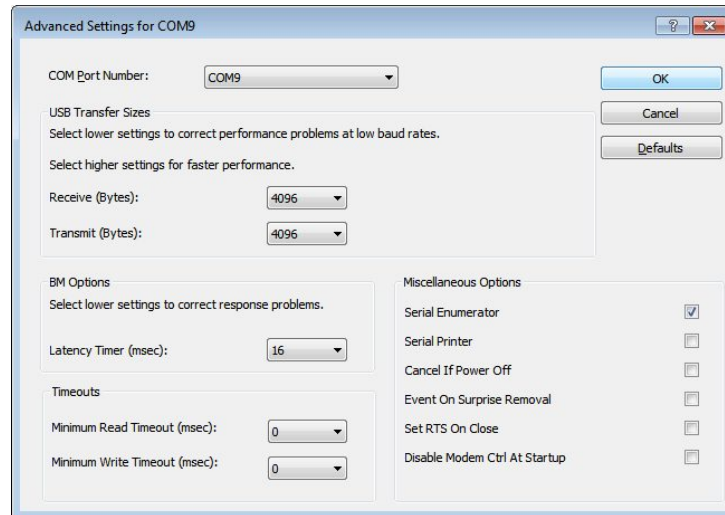


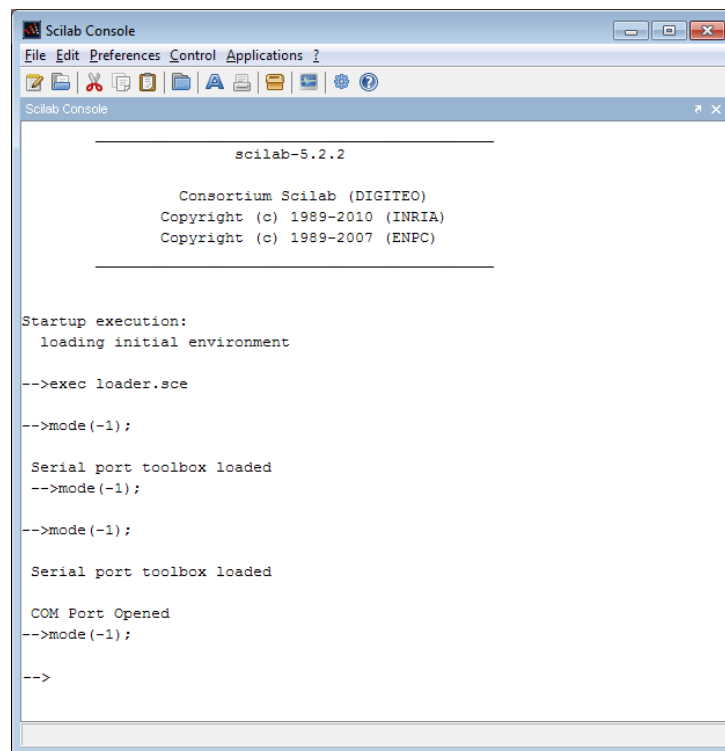
Figure 2.2: Changing Com port number

parity, data bits and stop bits as a string. You should give it as "9600,n,8". Since stop bit is zero in our case, omit the parameter from the string to indicate it as zero. Execute this .sce file. You will get a message COM Port Opened. If it complains, reconnecting the USB cable and/or restarting Scilab may help. Now execute the step_test.sci file. The results are illustrated in figure 2.3.

Type Xcos on the scilab console or click on Applications and choose Xcos to open Xcos environment. Load the step_test.xcos file from the File menu. The Xcos interface that will open is as shown in figure 2.5. You can set the block parameters by double clicking on the block as shown in figure 2.6. To run the code click on Simulation menu and choose start. After running Xcos successfully you would see the plots as shown in figure 2.7. See that the values of fan and heater you input to the xcos file is getting reflected on the board display. To stop the experiment click on the stop option on the menu bar of the Xcos environment.

2.2 Accessing Single Board Heater system on a Linux System

This section deals with the procedure to use SBHS on a Linux Operating System. The Operating System used for this document is Ubuntu 10.04. For Linux users, the instructions given in section 2.1 hold true with a few changes as below:



The image shows a screenshot of the Scilab Console window. The window has a title bar 'Scilab Console' and a menu bar with 'File', 'Edit', 'Preferences', 'Control', and 'Applications ?'. Below the menu bar is a toolbar with various icons. The main area of the console displays the following text:

```
-----  
scilab-5.2.2  
  
Consortium Scilab (DIGITEO)  
Copyright (c) 1989-2010 (INRIA)  
Copyright (c) 1989-2007 (ENPC)  
-----  
  
Startup execution:  
loading initial environment  
  
-->exec loader.sce  
  
-->mode(-1);  
  
Serial port toolbox loaded  
-->mode(-1);  
  
-->mode(-1);  
  
Serial port toolbox loaded  
  
COM Port Opened  
-->mode(-1);  
  
-->
```

Figure 2.3: Expected responses seen on the console

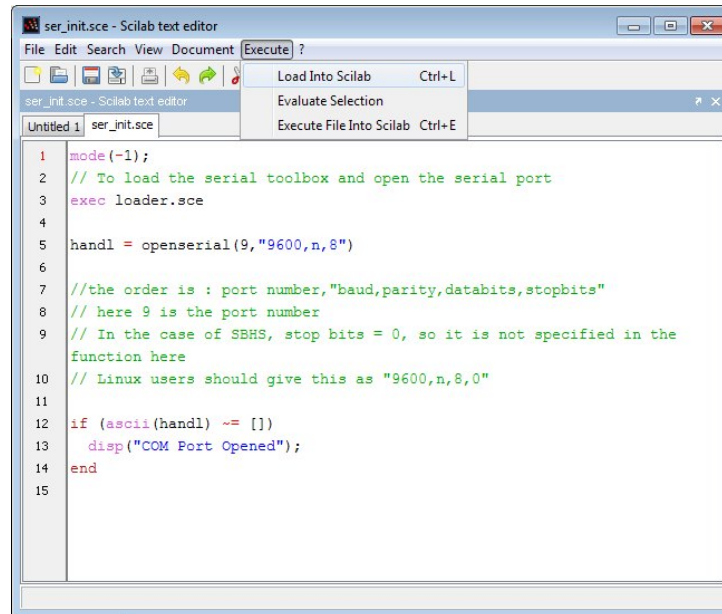


Figure 2.4: Executing script files

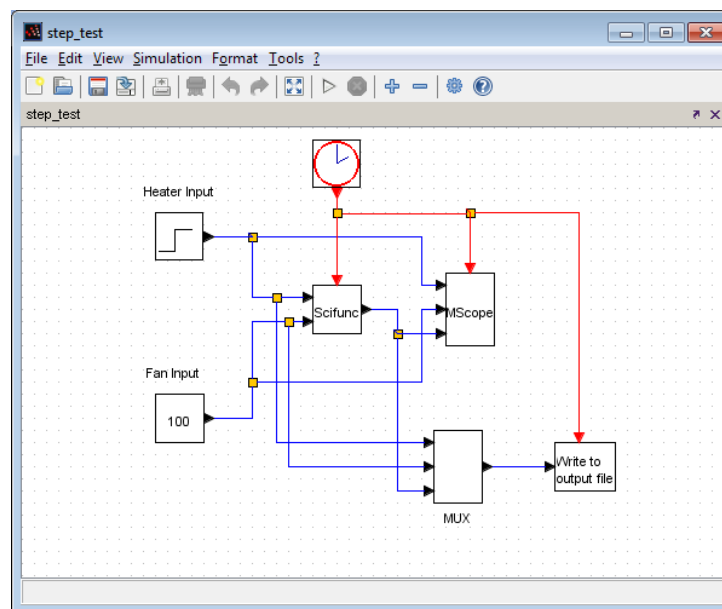


Figure 2.5: Xcos Interface

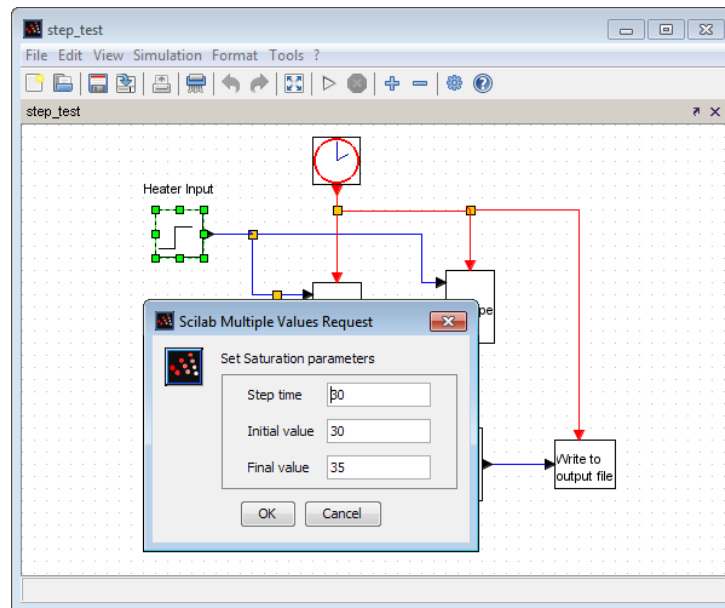


Figure 2.6: Setting Block Parameters

You do not require FTDI COM port drivers for connecting your SBHS to the PC. After plugging in the USB cable to your PC, check the serial port number by typing `ls /dev/ttyUSB*` on the terminal, refer Fig.2.8. Usually, the highest numbered one will be your device port number. eg:- `/dev/ttyUSB0`. If you want to connect more than one USB device, then type `tail -f /var/log/messages | grep ttyUSB` on the linux terminal just before plugging in the individual USB cable, refer Fig.2.9. The USB number will then be shown on the screen. Press `Ctrl+ c` to abort the command.

Note this number and change the port number (the first argument of the `openserial()` function) in the `ser_init.sce` file with it. The second argument of the `openserial()` function should be `"9800,n,8,0"`, refer Fig.2.10. This defines baud rate, parity, data bits and stop bits in that order. It has been found that if we omit the last parameter i.e., stop bits instead of specifying it as zero, scilab gives an error. Execute this file. Once the serial port initialisation is successfully done, you get a message as shown in Fig.2.11. If it complains, reconnecting the USB cable and/or restarting Scilab may help.

Now execute the `step_test.sci` file. The results are illustrated in figure 2.3. Type `Xcos` on the scilab console or click on Applications and choose `Xcos` to open Xcos environment. Load the `step_test.xcos` file from the File menu. The Xcos

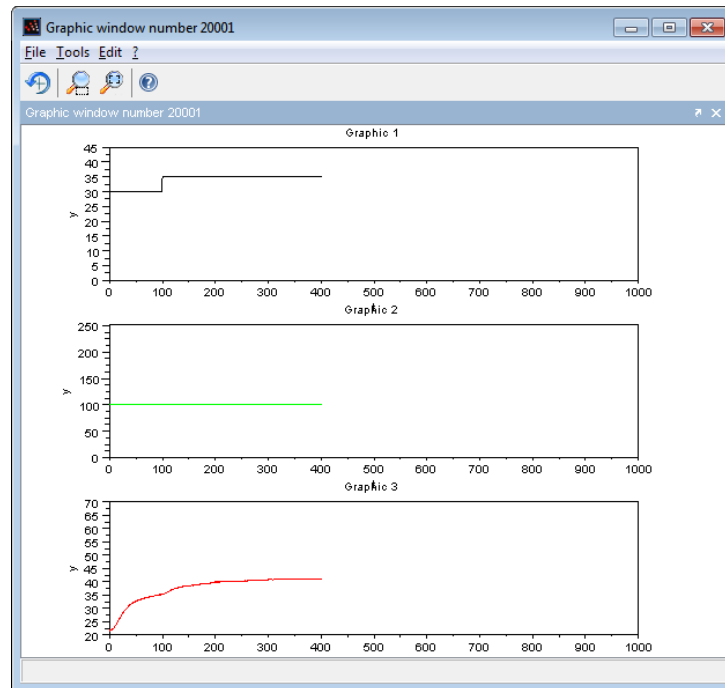
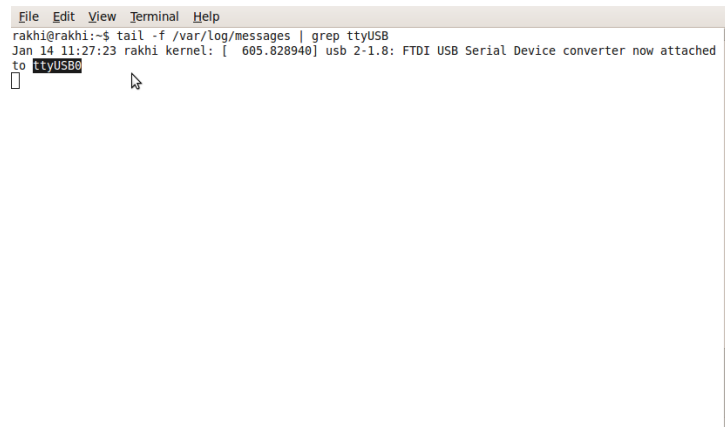


Figure 2.7: Plot obtained after executing step_test.xcos

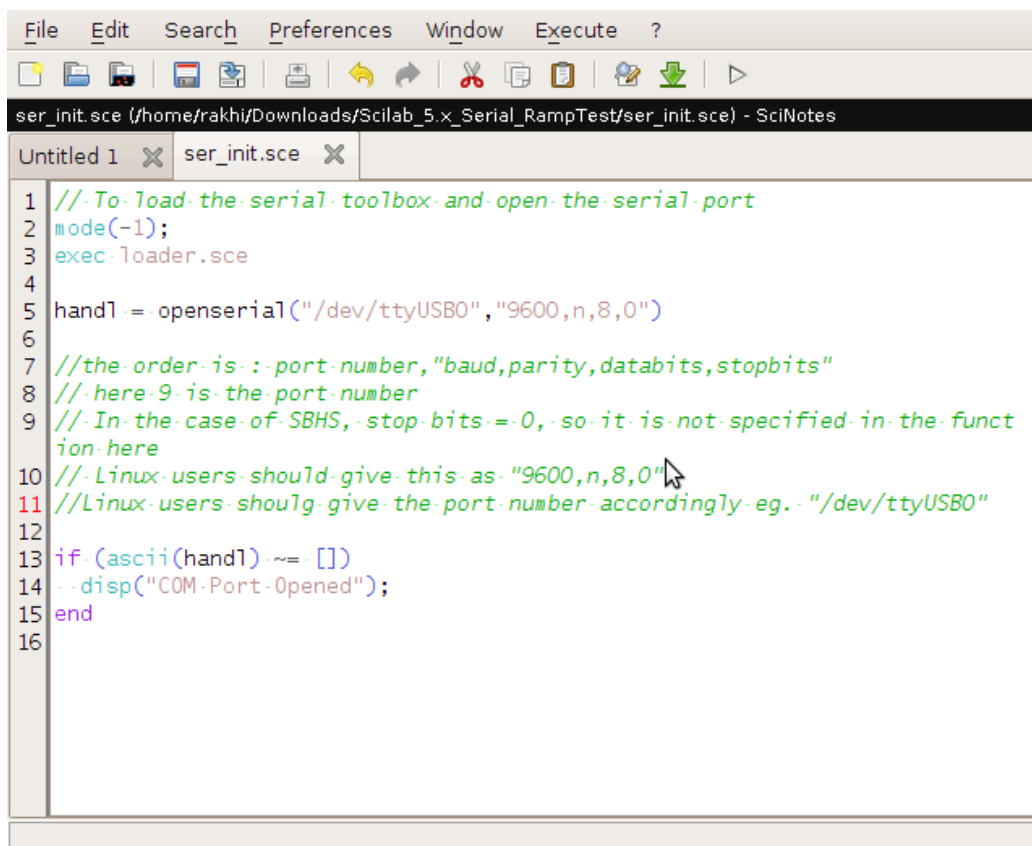
```
File Edit View Terminal Help
rakhi@rakhi:~$ ls /dev/ttyUSB*
/dev/ttyUSB0
rakhi@rakhi:~$
```

Figure 2.8: Checking the port number in linux (1)



```
File Edit View Terminal Help
rakhi@rakhi:~$ tail -f /var/log/messages | grep ttyUSB
Jan 14 11:27:23 rakhi kernel: [ 605.828940] usb 2-1.8: FTDI USB Serial Device converter now attached
to ttyUSB0
```

Figure 2.9: Checking the port number in linux (2)



```
File Edit Search Preferences Window Execute ?
ser_init.sce (/home/rakhi/Downloads/Scilab_5.x_Serial_RampTest/ser_init.sce) - SciNotes
Untitled 1 x ser_init.sce x
1 // To load the serial toolbox and open the serial port
2 mode(-1);
3 exec loader.sce
4
5 handle = openserial("/dev/ttyUSB0", "9600,n,8,0")
6
7 // the order is : port number, "baud, parity, databits, stopbits"
8 // here 9 is the port number
9 // In the case of SBHS, stop bits = 0, so it is not specified in the function here
10 // Linux users should give this as "9600,n,8,0"
11 // Linux users should give the port number accordingly eg. "/dev/ttyUSB0"
12
13 if (ascii(handle) ~= [])
14     disp("COM Port Opened");
15 end
16
```

Figure 2.10: Configuring port number and other parameters

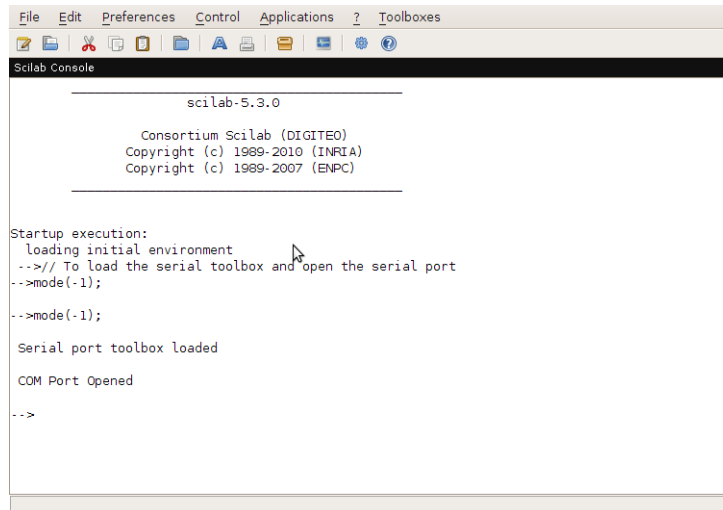


Figure 2.11: Scilab Console Message after Opening Serial Port

interface that will open is as shown in figure 2.5. You can set the block parameters by double clicking on the block as shown in figure 2.6. To run the code click on Simulation menu and choose start. After running Xcos successfully you would see the plots as shown in figure 2.7. See that the values of fan and heater you input to the xcos file is getting reflected on the board display. To stop the experiment click on the stop option on the menu bar of the Xcos environment.

Chapter 3

PRBS modelling and implementation of pole-placement controller

The aim of this experiment is to do PRBS testing on Single Board Heater System by the application of PRBS signal. The target group is anyone who has basic knowledge of Control Engineering. We have used Scilab with Xcos as an interface for sending and receiving data. This interface is shown in figure 3.1. Heater current and Fan speed are the two inputs to the system. The Heater current is varied with a PRBS signal. A provision is made to set the parameters like PRBS amplitude and offset value. A provision is also made to time the occurrence of the PRBS input, using a step block. The value of step time in the step block has to be chosen carefully. Sufficient amount of time should be given to allow the temperature to reach a steady-state before the PRBS signal is applied. In this experiment we are keeping the Fan speed constant at 50%. The temperature profile thus obtained is the output.

The first half of this chapter is dedicated to do system identification of the SBHS system using the response obtained for a PRBS (Pseudo Random Binary Sequence) input. In the second half, a pole-placement controller is designed using this model and implemented on SBHS.

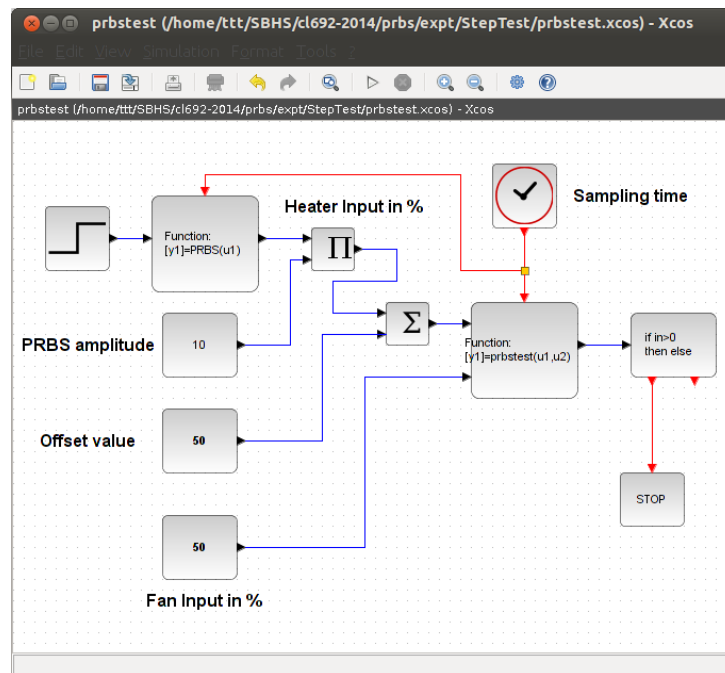


Figure 3.1: Xcos for PRBS testing experiment

3.1 Issues with step testing and alternate approach

SBHS is an example of a heater. Suppose you are working in a full scale plant. Current control system designed to control one of the heaters of the plant is lousy and your supervisor asked you to design a new controller from scratch. The first step you need to do is identification of the heater transfer function. The catch is, plant is currently operational. You can't shut the plant down to identify the heater transfer function. You have to do it while the heater is operating in the plant. You might think of giving the heater a positive step and measuring the response in the controlled temperature. This will increase the temperature of the component being heated for the period of time step is applied. However, if the process is sensitive to temperature of the component (distillation, for example), it will go off the desired course and the output of the whole plant will be affected and will be undesirable.

There is an alternate approach which is widely used in industry. The input given to the heater for identification is not step, but a **pseudo-random binary sequence (PRBS)**. The concept behind PRBS is that the heat input is perturbed in such a way that the time average of the heat input is the value at which the heater is being operated currently. Thus, some positive and some negative steps can be given. This results some positive and some negative changes in the temperature which leads to the time average of the performance of the plant remaining the same. Thus, PRBS testing can be done in a working plant without affecting the plant performance unlike step testing.

A typical PRBS and corresponding plant output looks like

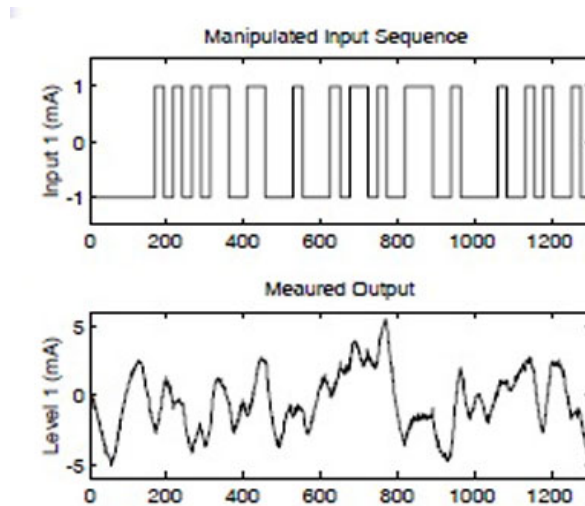


Figure 3.2: PRBS testing input and output [Image source: CL 686 Advanced Process Control, Spring 2013-14 lecture slides. Prof. S. C. Patwardhan, IIT Bombay]

3.2 Step by step procedure to do PRBS testing

Similar to Chapter ?? and ??, in this section we will find the transfer function model of SBHS. But there are two major differences. First difference is that we will give a Pseudo Random Binary Sequence to the heater input of SBHS and second difference is that we will find the discrete time transfer function. A Pseudo Random Binary Sequence is nothing but a signal whose amplitude varies between two limits randomly at any given time. An illustration of the same is given in figure 3.3. A PRBS signal can be easily generated using the `rand()` function in scilab. Scilab code to generate the PRBS signal is given at the end of this chapter. Figure 3.1 shows the xcos diagram for PRBS testing. The PRBS amplitude and offset value to the input can be adjusted using the relevant blocks.

The steps to be followed to conduct PRBS test experiment remains same as explained in section 2.2 for linux users and section 2.1.2 for windows users. The only difference is that the `prbs` function is written in `prbstest.sci` file and the xcos is saved in file `prbstest.xcos`. These files should be used for doing prbs experiment.

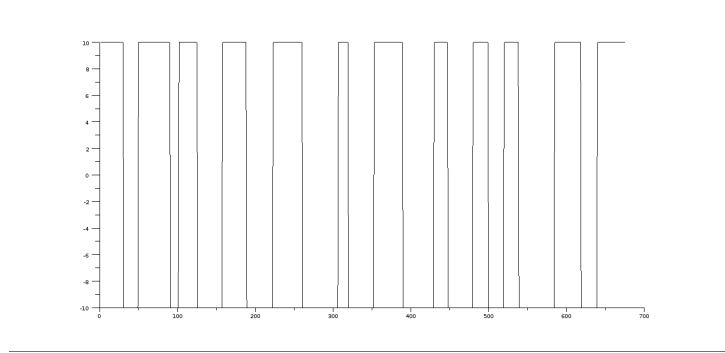


Figure 3.3: A Pseudo Random Binary Sequence

3.3 Determination of discrete time transfer function

System identification is carried out to identify the transfer function between the input signal to the system and output from the system. Firstly a transfer function with unknown parameters is assumed. The system is given a known input and its response is obtained and then the values of the unknown parameters is chosen such that the sum of squares of the errors is minimized. Here, the error is the difference between the actual output and the output predicted by the transfer function model assumed.

For the given SBHS system, we assume a second order transfer function:

$$G(z) = \frac{b_1 + b_2 z^{-1}}{1 + a_1 z^{-1} + a_2 z^{-2}} z^{-d} \quad (3.1)$$

The unknown parameters a_1, a_2, b_1, b_2 and d are to be obtained through the response of the system to the known inputs. a_1, a_2, b_1, b_2 are real numbers and d is the plant delay which is an integer. For these model parameters estimation, we used a pseudo random binary sequence (PRBS) input. Since the optimization over discrete variables (d in this case) is a very difficult routine for computers, we assume a value of d and then optimize over a_1, a_2, b_1, b_2 . The optimization problem, then, becomes:

$$(\hat{b}_1, \hat{b}_2, \hat{a}_1, \hat{a}_2) = \underset{b_1, b_2, a_1, a_2}{argmin} \sum_{i=0}^N (y(k) - \hat{y}(k))^2 \quad (3.2)$$

Here, $y(k)$ is the output obtained from the system, so it is known, $\hat{y}(k)$ is the estimated output using y the model assumed, which can be written as a difference equation:

$$\hat{y}(k) = -a_1\hat{y}(k-1) - a_2\hat{y}(k-2) + b_1u(k-d) + b_2u(k-1-d) \quad (3.3)$$

The optimization is performed using the optimization routine “optim” of Scilab. Copy the data file to the folder `prbs_analysis`. Change the scilab working directory to `prbs_analysis` folder. Open the file `optimize.sce` and put the name of the data file (with extension) in the filename field. Save and run this code and obtain the plot as shown in figure 4.3. This code uses the routines `label.sci`, `costfunction.sci` and `second_order.sci`. This code will give optimized values for a_1, a_2, b_1, b_2 which can be used to define a second order discrete time transfer function as given in equation 3.1

3.3.1 Performing PRBS testing on SBHS, virtually

The step by step procedure for conducting an experiment virtually is explained in section ???. The required .sce file is `prbstest.sce`. You will find this file in the `prbs` directory under `virtual` folder. The necessary codes are listed in the section 3.4

3.4 Scilab Code

Scilab Code 3.1 `ser_init.sce`

```

1 // To load the serial toolbox and open the serial port
2 exec loader.sce
3
4
5 // **** Linux Users **** //
6 handl = openserial("/dev/ttyUSB0", "9600,n,8,0") // Linux
      users should use this line and comment the below
      windows line
7
8 // **** Windows Users **** //
```

```

9 // handl = openserial(2,"9600,n,8") // Windows users
    should use this line and comment the above linux
    line
10
11
12 // the order is : port number , " baud , parity , databits ,
    stopbits "
13 // here 9 is the port number
14 // In the case of SBHS , stop bits = 0 , so it is not
    specified in the function here
15 // Linux users should give this as ("/" ,"9600,n,8,0")
16
17 if (ascii(handl) ~= [])
18     disp("COM Port Opened");
19 end

```

Scilab Code 3.2 imc.sci

```

1 mode(0)
2 global temp heat fan sampling_time m
3 // heatdisp fandisp tempdisp setpointdisp x name
4
5 clear heatdisp fandisp tempdisp setpointdisp
6
7
8 // ** Sampling Time **//
9 sampling_time = 1;
10 // ///// * * * * //
11
12 // m = 1;
13
14 dt = getdate();
15 year = dt(1);
16 month = dt(2);
17 day = dt(6);
18 hour = dt(7);
19 minutes = dt(8);
20 seconds = dt(9);

```

```

21
22 file1 = strcat(string([year month day hour minutes
    seconds]),'-');
23 string txt;
24 filename = strcat([file1 , "txt"],'.');
25
26
27 function [temp] = imc(setpoint ,fan ,alpha)
28 global temp heat_in fan_in C0 u_old u_new e_old e_new
    e_old_old
29
30 global heatdisp fandisp tempdisp setpointdisp
    sampling_time m name
31
32 e_new = setpoint - temp;
33 b=((1-alpha)/0.01163);
34
35 u_new = u_old + b*(e_new - (0.9723*e_old));
36
37
38 u_old = u_new;
39 e_old = e_new;
40
41
42 heat = u_new;
43
44     if heat >100
45         heat = 100;
46     elseif heat < 0
47         heat = 0;
48     end;
49
50
51 writeserial(handl ,ascii(254)); // Input Heater ,
    writeserial accepts
                                     strings ; so
    convert 254 into its
                                     string

```

```

    equivalent
52  writeshdrial(handl ,ascii(heat));
53  writeshdrial(handl ,ascii(253)); // Input Fan
54  writeshdrial(handl ,ascii(fan));
55  writeshdrial(handl ,ascii(255)); // To read Temp
56  sleep(100);
57
58  temp = ascii(readserial(handl)); // Read serial
    returns a string , so
                                convert it to
    its integer (ascii)
                                equivalent
59  temp = temp(1) + 0.1*temp(2); // convert to temp with
    decimal points
    eg : 40.7
60
61  A = [m, heat , fan , temp ];
62
63  fdh = file( 'open' , filename , 'unknown' );
64
65  file( 'last' , fdh)
66
67  write(fdh ,A, '(7(e11.5,1x))');
68
69  file( 'close' , fdh);
70
71
72  x=ceil(1/sampling_time);
73
74      if (modulo(m,x) == 1|sampling_time >= 1)
75
76
77          heatdisp=[heatdisp;heat];
78
79          subplot(311);
80
81          xtitle("PID Controller","Time(seconds)",
                "Heat in percentage")

```

```

82
83         plot2d( heatdisp , rect =[0 ,0 ,1000 ,100] ,
               style =1)
84
85
86
87         fandisp =[ fandisp ; fan ] ;
88
89         subplot(3 1 2) ;
90
91         xtitle ( "" , "Time( seconds )" , "Fan in
               percentage" )
92
93         plot2d( fandisp , rect =[0 ,0 ,1000 ,100] , style
               =2)
94
95
96
97         tempdisp =[ tempdisp ; temp ] ;
98         setpointdisp =[ setpointdisp ; setpoint ] ;
99
100        subplot(3 1 3)
101
102        xtitle ( "" , "Time( seconds )" , "Temperature (
               deg celcius )" )
103
104        plot2d( tempdisp , rect =[0 ,30 ,1000 ,40] ,
               style =5)
105        plot2d( setpointdisp , rect =[0 ,30 ,1000 ,40] ,
               style =1)
106
107        m=m+1 ;
108
109        else
110            m = m+1 ;
111    end
112
113

```


114 **endfunction**

Scilab Code 3.3 imc_virtual.sce

```
1 mode(0);
2 // For scilab 5.1.1 or lower version users , use scicos
   command to open scicos diagrams instead of xcos
3
4 global fdfh fdt fncr fncw m err_count y
5
6 fncr = 'clientread.sce';
7 fdt = mopen(fncr);
8 mseek(0);
9
10 err_count = 0; // initialising error count for network
   error
11 m =1;
12 exec ("imc_virtual.sci");
13 A = [0.1,m,0,251];
14 fdfh = file('open','clientwrite.sce','unknown');
15 write(fdfh,A,'(7(e11.5,1x))');
16 file('close', fdfh);
17 sleep(2000);
18 a = mgetl(fdt,1);
19 mseek(0);
20 if a~= [] // open xcos only if communication is
   through (ie reply has come from server)
21 xcos('imc.xcos');
22 else
23 disp ("NO NETWORK CONNECTION!");
24 return
25 end
```

Scilab Code 3.4 imc_virtual.sci

```
1 mode(0)
2
3 global fan
```

```

4  function [temp,heat,e_new,stop] = imc_virtual(setpoint
    ,fan,alpha)
5  global temp heat et SP u_new u_old u_new e_old e_new
    fdfh fdt fncr fncw m err_count stop
6
7  fncr = 'clientread.sce';           // file to be read -
    temperature
8  fncw = 'clientwrite.sce';         // file to be written
    - heater , fan
9
10 a = mgetl(fdt,1);
11 b = evstr(a);
12 byte = mtell(fdt);
13 mseek(byte,fdt,'set');
14
15 if a~= []
16     temp = b(1,$); heats = b(1,$-2);
17     fans = b(1,$-1); y = temp;
18
19 e_new = setpoint - temp;
20
21 b=((1-alpha)/0.01163);
22 u_new = u_old + b*(e_new - (0.9723*e_old));
23
24 if u_new> 39
25     u_new = 39;
26 end;
27
28 if u_new< 0
29     u_new = 0;
30 end;
31
32 heat=u_new;
33 u_old = u_new;
34 e_old = e_new;
35
36
37

```

```

38 A = [m,m,heat,fan];
39     fdfh = file('open','clientwrite.sce','unknown');
40     file('last', fdfh)
41     write(fdfh,A,'(7(e11.5,1x))');
42     file('close', fdfh);
43     m = m+1;
44
45     else
46         y = 0;
47         err_count = err_count + 1; // counts the no of
            times network error occurs
48         if err_count > 300
49             disp("NO NETWORK COMMUNICATION!");
50             stop = 1; // status set for stopping simulation
51         end
52         // disp(stop)
53     end
54
55 return
56 endfunction

```

Bibliography

- [1] Fossee moodle. <http://www.fossee.in/moodle/>. Seen on 10 May 2011.
- [2] Spoken tutorials. http://spoken-tutorial.org/Study_Plans_Scilab. Seen on 10 May 2011.
- [3] K. M. Moudgalya. Introducing National Mission on Education through ICT. <http://www.spoken-tutorial.org/NMEICT-Intro>, 2010.
- [4] K. M. Moudgalya and Inderpreet Arora. A Virtual Laboratory for Distance Education. In *Proceedings of 2nd Int. conf. on Technology for Education, T4E*, IIT Bombay, India, 1–3 July 2010. IEEE.
- [5] Kannan M. Moudgalya. *Digital Control*. John Wiley and Sons, 2009.
- [6] Kannan M. Moudgalya. *Identification of transfer function of a single board heater system through step response experiments*. 2009.
- [7] Katsuhiko Ogata. *Modern Control Engineering*. Prentice-Hall of India, 2005.
- [8] Dale E. Seborg, Thomas F. Edgar, and Duncan A. Mellichamp. *Process Dynamics and Control*. John Wiley and Sons, 2nd edition, 2004.
- [9] Virtual labs project. Single board heater system. http://www.co-learn.in/web_sbhs. Seen on 11 May 2011.