Documentation for Single Board Heater System

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

IIT Bombay
November 19, 2014

Contents

List of Scilab Code **3**

1 Block diagram explanation of Single Board Heater System **4**
1.1 Microcontroller 4
1.1.1 PWM for heat and speed control 5
1.1.2 Analog to Digital conversion 7
1.2 Instrumentation amplifier 7
1.3 Communication 8
1.3.1 Serial port communication 9
1.3.2 Using USB for Communication 9
1.4 Display and Resetting the setup 9

2 Performing a Local Experiment on Single Board Heater System **13**
2.1 Using SBHS on a Windows OS 14
2.1.1 Installing Drivers and Configuring COM Port 14
2.1.2 Steps to Perform a Local Experiment 15
2.2 Using Single Board Heater System on a Linux System 18
2.3 Summary of procedure to perform a local experiment 21
2.4 Scilab Code under common_files 22

3 Using Single Board Heater System, Virtually! **28**
3.1 Introduction to Virtual Labs at IIT Bombay 28
3.2 Evolution of SBHS virtual labs 29
3.3 Current Hardware Architecture 32
3.4 Current Software Architecture 33
3.5 Conducting experiments using the Virtual lab 33
3.5.1 Registration, Login and Slot Booking 35

3.5.2 Configuring proxy settings and executing python based client . . . 36
3.5.3 Executing scilab code . . . 37
3.6 Summary . . . 40

4 Identification of Transfer Function of a Single Board Heater System through Step Response Experiment 42
4.1 Conducting Step Test on SBHS locally . . . 44
4.2 Conducting Step Test on SBHS, virtually . . . 45
4.3 Identifying First Order and Second Order Transfer Functions . . . 45
4.3.1 Determination of First Order Transfer Function . . . 45
4.3.2 Procedure . . . 47
4.4 Determination of Second Order Transfer Function . . . 48
4.4.1 Procedure . . . 49
4.5 Discussion . . . 50
4.6 Scilab Code . . . 50

5 Identification of Transfer Function of a Single Board Heater System through Ramp Response Experiment 58
5.1 Conducting Ramp Test on SBHS locally . . . 61
5.2 Conducting Ramp Test on SBHS, virtually . . . 61
5.3 Identifying First Order Transfer Function . . . 61
5.3.1 Procedure . . . 63
5.4 Discussion . . . 64
5.5 Scilab Code . . . 64

List of Scilab Code

2.1 comm.sci 22
2.2 init.sci 24
2.3 plotting.sci 25
4.1 label.sci 50
4.2 costf_1.sci 51
4.3 firstorder.sce 51
4.4 costf_2.sci 53
4.5 order_2_heater.sci 53
4.6 secondorder.sce 54
4.7 ser_init.sce 55
4.8 step_test.sci 56
4.9 stepc.sce 56
4.10 steptest.sci 56
5.1 ramp_test.sci 64
5.2 label.sci 65
5.3 cost.sci 65
5.4 cost_approx.sci 66
5.5 ramptest.sci 66
5.6 ramptest.sce 66
5.7 ramp_virtual.sce 67

Chapter 1

Block diagram explanation of Single Board Heater System

Figure1.1 shows the block diagram of 'Single Board Heater System'(SBHS). Microcontroller ATmega16 is used at the heart of the setup. The microcontroller can be programmed with the help of an In-system programmer port(ISP) available on the board. The setup can be connected to a computer via two serial communication ports namely RS232 and USB. 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 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 feed to the μC through an Instrumentation Amplifier. Some required parameter values are also 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)

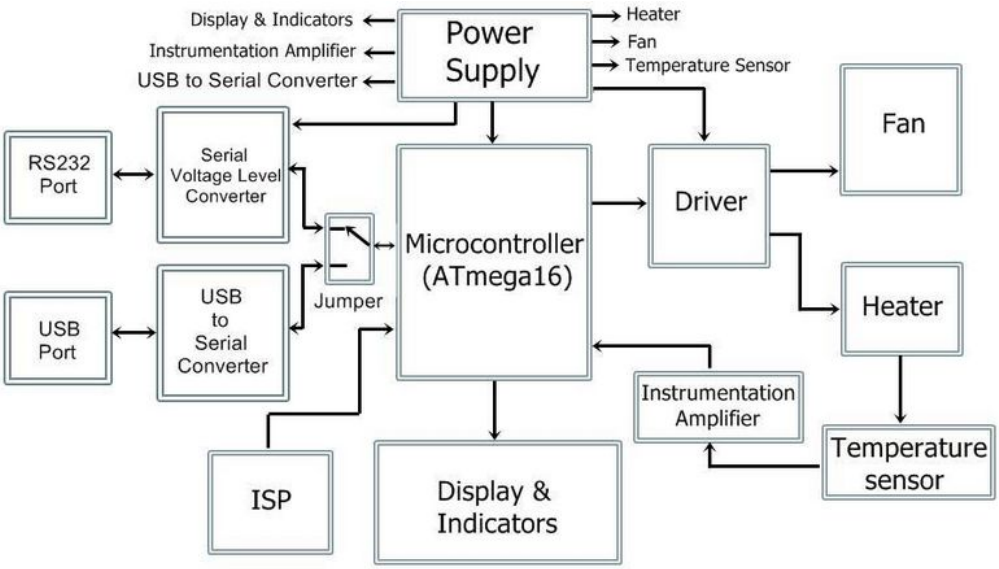

Figure 1.1: Block Diagram

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 contains a Heater coil and a Fan. The heater assembly consists of an iron plate placed at a distance of about 3.5mm from a

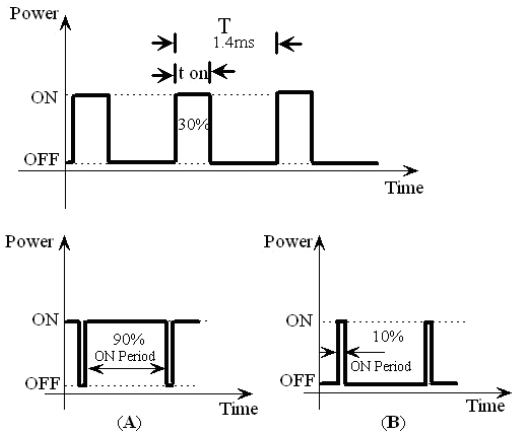

Figure 1.2: Pulse Width Modulation (A): On time is 90% of the total time period, (B): ON time is 10% of total time period

nichrome coil. When current passes through the coil it gets heated and in turn raises the temperature of the iron plate. Altering the heat generated by the coil and also the speed at which the fan is operated, are the objectives of our prime interest. The amount of power delivered to the Fan and Heater can be controlled in various ways. The technique used here is called as PWM (abbreviation of Pulse Width Modulation)technique. PWM is a process in which the duty cycle of the square wave is modulated.

$$\text{Duty cycle} = \frac{T_{ON}}{T} \tag{1.1}$$

Where T_{ON} is the ON time of the wave corresponding to the HIGH state of logic 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 set in the internal timer. The pulse width of the waveform can be varied accordingly by varying this count value. Thus, 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 to control the power delivered to each of 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. The temperature of this plate is measured by using a temperature sensor AD590.
Some of the salient features of AD590 include:

1. Linear current output: 1μA/K
2. Wide range: -55°C to +150°C
3. Sensor isolation from the case
4. Low cost

The output of AD590 is then fed to the microcontroller through an Instrumentation Amplifier. The signal obtained at the output of the Instrumentation Amplifier is in analog form. It should be converted in to digital form before feeding as an input to the microcontroller. ATmega16 features an internal 8-channel , 10 bit successive approximation ADC (analog to digital converter) with 0-Vcc(0 to Vcc) input voltage range, which is used for converting the output of Instrumentation Amplifier. An interrupt is generated on completion of analog to digital conversion. Here, ADC is initialize to have 206 μs of conversion time . Digital data thus obtained is sent to the computer via serial port as well as for further processing required for the on-board display.

1.2 Instrumentation amplifier

Instrumentation Amplifiers are often used in temperature measurement circuits in order to boost the output of the temperature sensors. A typical three Op-Amp Instrumentation amplifier is shown in the figure 1.3. The Instrumentation Amplifiers (IAs) are mostly preferred, where the sensor is located at a remote place and therefore is susceptible to signal attenuation, due to their very low DC offsets, high input impedance, very high Common mode rejection ratio (CMRR). The IAs 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

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

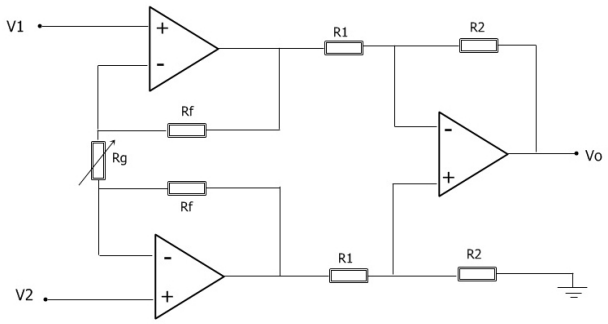

Figure 1.3: 3 Op-Amp Instrumentation Amplifier

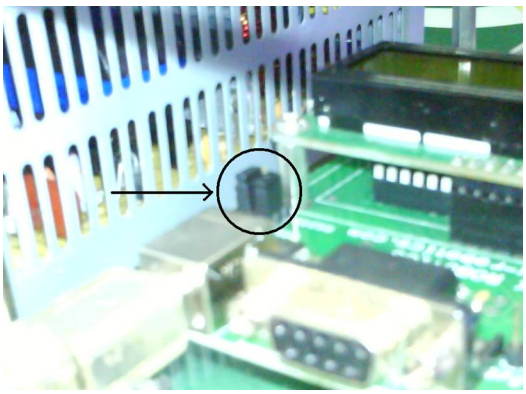

Figure 1.4: Jumper arrangement

The value of R_g is kept variable to change the overall gain of the amplifier. The signal generated by AD590 is in μA/°K. It is converted to mV/°K by taking it across a 1 KΩ resistor. The °K to°C conversion is done by subtracting 273 from the °K result. One input of the IA is fed with the mV/°K reading and the other with 273 mV. The resulting output is now in mV/°C. The output of the IA is fed to the microcontroller for further processing.

1.3 Communication

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

Figure 1.5: RS232 cable

transistor logic). However, according to RS232 standard voltage level below -5V is treated as logic 1 and voltage level above +5V is treated as logic 0. This convention is used to ensure error free transmission over long distances. For solving this compatibility issue between RS232 and TTL, an external hardware interface IC MAX202 is used. IC MAX202 is a +5V RS232 transreceiver.

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 supports 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 in the figure 1.8. To make the setup USB compatible, USB to serial conversion is carried out using IC FT232R. Note that proper USB driver should be installed on the computer.

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 microcon-

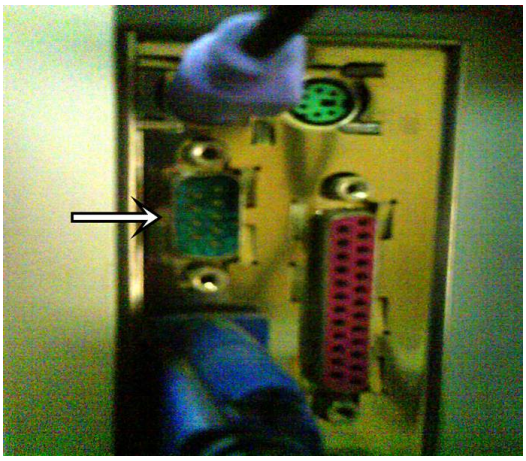

Figure 1.6: Serial port

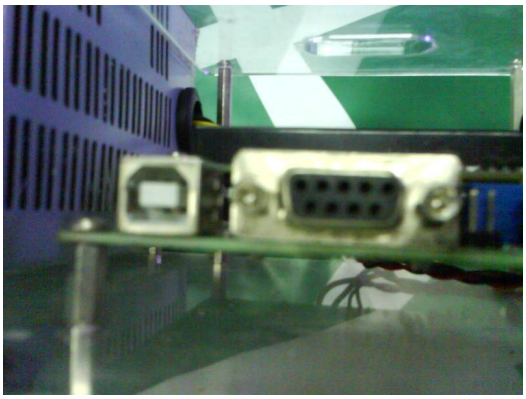

Figure 1.7: USB communication

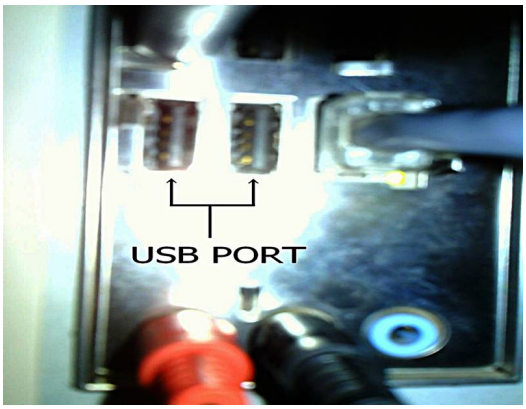

Figure 1.8: USB PORT

Figure 1.9: Display

troller. As shown in figure 1.9, numerals below TEMP indicate the actual temperature of the heater plate in °C. Numerals below HEA and FAN indicate the respective percentage values 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 using the reset button 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 is set to the maximum value. Although these reset values are not displayed on the LCD display these are preloaded to the appropriate units.

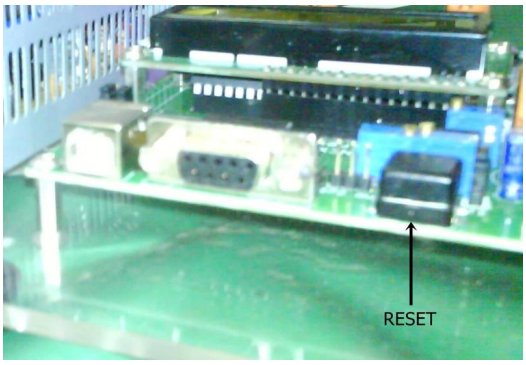

Figure 1.10: Reset

Chapter 2

Performing a Local Experiment on Single Board Heater System

This chapter explains the procedure to use Single Board Heater System locally with Scilab i.e. when you are physically accessing SBHS using your computer. 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 otherwise.

Hardware and Software Requirements

For working with the Single Board Heater system, following components are required:

1. SBHS with USB cable and power cable.
2. PC/Laptop with Scilab software installed. Scilab can be downloaded from: `http://www.scilab.org`
3. FTDI Virtual Com Port driver corresponding to the OS on your PC. Linux users do not need this. The driver can be downloaded from: `http://www.ftdichip.com/Drivers/VCP.htm`

2.1 Using SBHS on a Windows OS

This section deals with the procedure to use SBHS on a Windows Operating System. The Operating System used for this document is Windows 7, 32-bit OS. If you are using some other Operating System or the steps explained in section 2.1.1 are not sufficient to understand, refer to the official document available on the main ftdi website at `www.ftdichip.com`. On the left hand side panel, click on 'Drivers'. In the drop-down 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 Drivers and Configuring COM Port

After powering ON the SBHS and plugging in the USB cable to the PC (check the jumper settings on the board are set to USB communication) for the very first time, the `Welcome to Found New Hardware Wizard` dialog box will pop up. Select the option `Install from a list or specific location`. Choose `Search for best driver in these locations`. Check the box `Include this location in the search`. Click on `Browse`. Specify the path where the driver is copied as explained earlier (item no.3) and install the driver by clicking `Next`. Once the wizard has successfully installed the driver, the SBHS is ready for use. Please note that this procedure should be repeated twice.

Now, the communication port number assigned to the computer port to which the Single Board Heater System is connected, via an RS232 or USB cable should be identified. For identifying this port number, right click on `My Computer` and click on `Properties`. Then, select the `Hardware` tab and click on `Device Manager`. The list of hardware devices will be displayed. Locate the Ports(COM & LPT) option and click on it. The various communication ports used by the computer will be displayed. If the SBHS is connected via 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 associated with the USB port after connecting a USB cable may be greater than 9. Since the serial tool box can handle only single digit port number (upto 9), it is necessary to change this COM port number. Following is the procedure to change the COM port number. Double click on the name of the particular port. Click on `Port Settings` tab and then click on

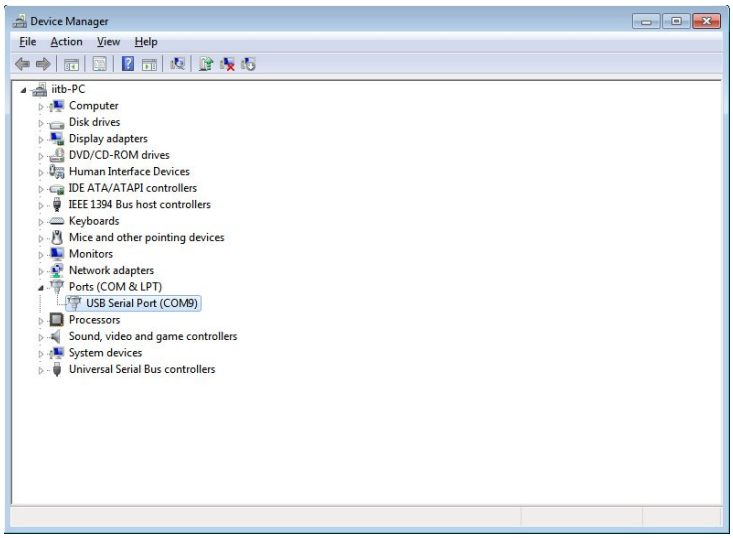

Figure 2.1: Checking Communication Port number

`Advanced`. In the COM port number drop-down menu, choose the port number to any other number less than 10. This procedure is illustrated in figure 2.2. After following the procedure the COM port number can be verified as described earlier.

Scilab must be installed on your computer. We recommend the use of scilab-5.3.3. This is because all the codes are created and tested using scilab-5.3.3. These codes may very well work in higher versions of scilab but one cannot use the same codes back again in scilab-5.3.3. This is because a software is always backward compatible, never forward compatible. Scilab for windows or linux can be downloaded from `scilab.org`. However, if scilab-5.3.3 for your OS is not available on `scilab.org` then one can download it from `sbhs.os-hardware.in/downloads`. Installation of scilab on windows is very straight-forward. After you download the .exe file one has to double click on it and proceed with the instructions given by the installer. All default options will work. However, note that scilab on windows requires internet connection during installation.

2.1.2 Steps to Perform a Local Experiment

Go to `sbhs.os-hardware.in/downloads`. Let us take a look at the downloads page. There are two versions of the scilab code. One which can be used with SBHS locally i.e. when you are physically accessing SBHS using your computer and another to be used for accessing SBHS virtually. This section expects you to

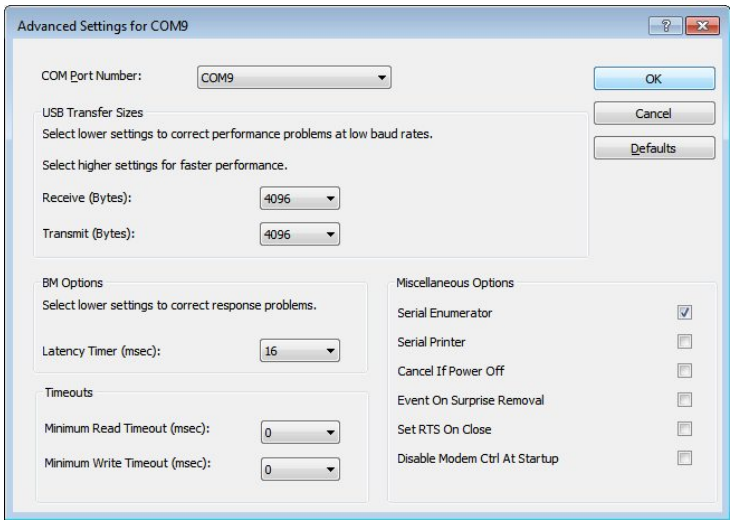

Figure 2.2: Changing Com port number

download the local version. On extracting the file that you will download, you will get a folder `local`. This folder will contain many folders named after the experiment. You will also find a directory named `common-files`. We are going to use the folder named `Step_test`.

1. Launch Scilab from start menu or double click the Scilab icon on the desktop (if any). Before executing any scripts which are dependent on other files or scripts, one needs to change the working directory of Scilab. This will set the directory path in Scilab from where the other necessary files should be loaded. To change the directory, click on file menu and then choose "Change directory". This can also be performed by typing `cd<space>folder path`. Change the directory to the folder `Step_test`. There is another quicker way to make sure you are in the required working directory. Open the experiment folder. Double click on the scilab file you want to execute. Doing so will automatically launch scilab and also automatically change the working directory. To know your working directory at any time, execute the command `pwd` in the scilab console.

2. Next, we have to load the content of `common-files` directory. Notice that this directory is just outside the `Step_test` directory. The `common-files` directory has several functions written in .sci files. These functions are required for executing any experiment. To load these functions type

getd<space>folder path. The folder path argument will be the complete path to common-files directory. Since this directory is just outside our Step_test directory, the command can be modified to getd<space>..\common_files So now we have all functions loaded.

3. Next we have to load the serial communication toolbox. For doing so we have to execute the loader.sce file present in the common-files directory. To do so execute the command exec<space>..\common_files\loader.sce or exec<space>folder path\loader.sce.

4. Next, click on editor from the menu bar to open the Scilab editor or simply type editor on the Scilab console and open the file ser_init.sce. Change the value of the variable port2 to the COM number identified for the connected SBHS. For example, one may enter 'COM5' as the value for port2. Notice that there is no space between COM and 5 and COM5 is in single quotes. Keep all other parameters untouched. Execute this .sce file by clicking on the execute button available on the menu bar of scilab editor window. The message COM Port Opened is displayed on successful implementation. If there are any errors, reconnecting the USB cable and/or restarting Scilab may help.

5. Next we have to load the function for the step test experiment. This function is written in step_test.sci file. Since we do not have to make any changes in this file we can directly execute it from scilab console without opening it. Run the command exec<space>step_test.sci in scilab console. The results are illustrated in figure 2.3.

6. Next, type Xcos on the Scilab console or click on Applications and select Xcos to open Xcos environment. Load the step_test.xcos file from the File menu. The Xcos interface is shown in figure 2.5. The block parameters can be set by double clicking on the block. To run the code click on Simulation menu and click on Start. After executing the code in Xcos successfully the plots as shown in figure 2.6 will be generated. Note that the values of fan and heater given as input to the Xcos file are reflected on the board display.

7. To stop the experiment click on the Stop option on the menu bar of the Xcos environment.

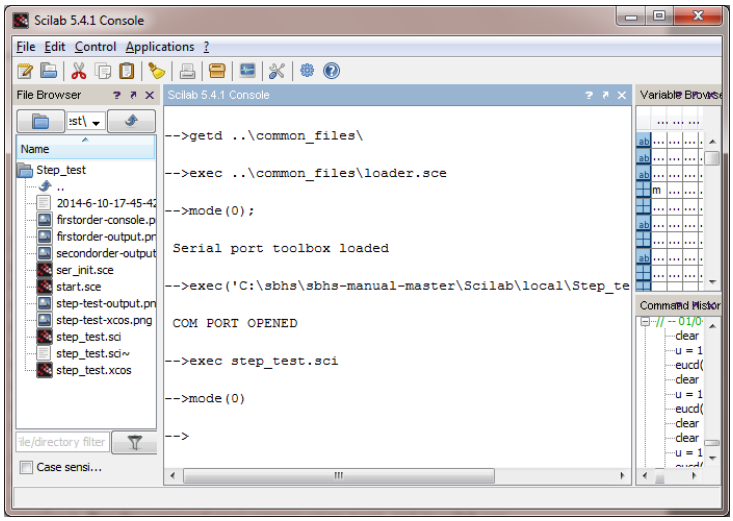

Figure 2.3: Expected responses seen on the console

All of the activities mentioned above, from `getd<space>..\common_files` untill starting the xcos simulation, are coded in a file named `start.sce`. Executing this file will do all necessary things automatically with just click of a button. This file however assumes three things. These are

1. The location of `common-files` directory is not changed
2. The current working directory is correct
3. The port number mentioned in `ser_init.sce` is correct

2.2 Using 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 12.04. For Linux users, the instructions given in section 2.1 hold true with a few changes as below:

On a linux system, Scilab-5.3.3 can be either installed from available package manager (synaptic in case of Ubuntu) or its portable version can be downloaded from scilab.org or `http://sbhs.os-hardware.in/downloads`. If in-

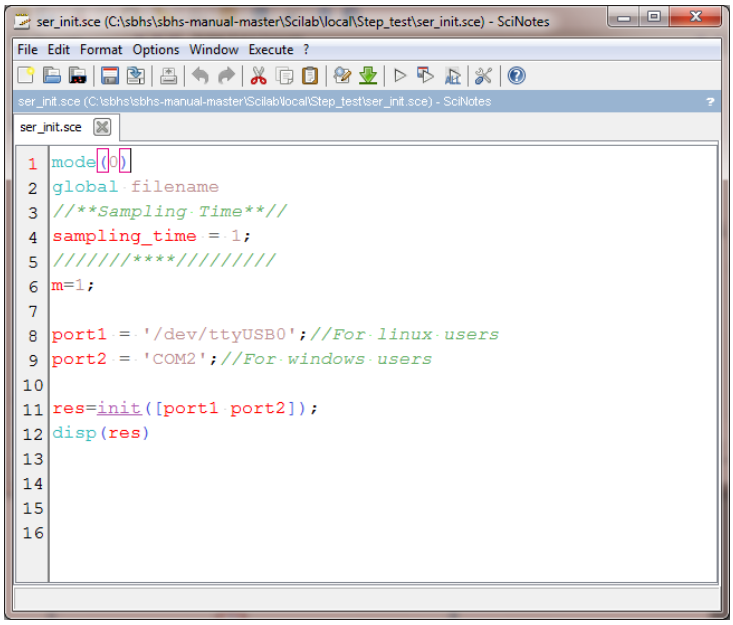

Figure 2.4: Executing script files

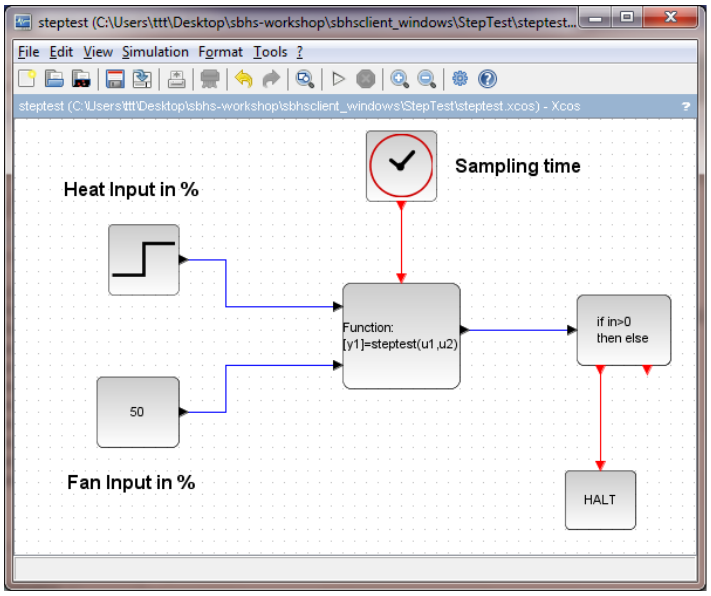

Figure 2.5: Xcos Interface

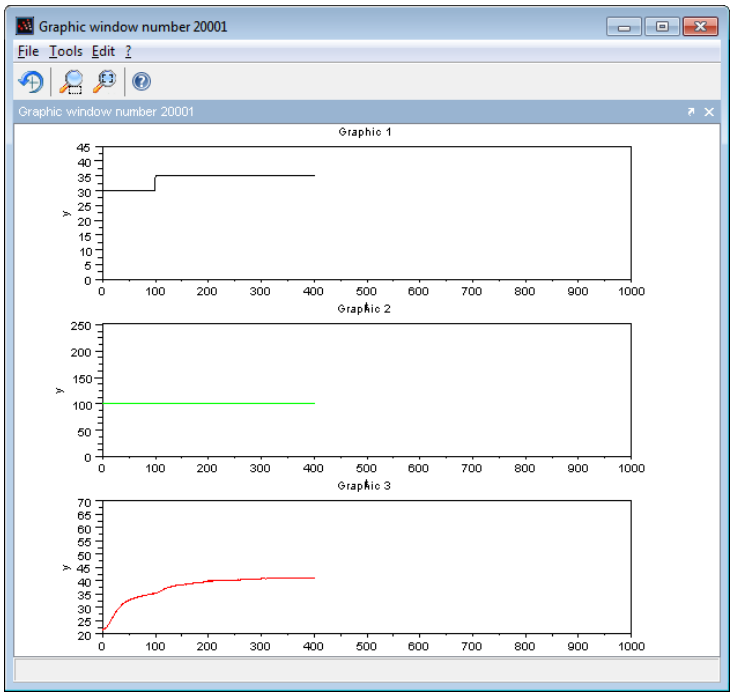

Figure 2.6: Plot obtained after executing step_test.xcos

Figure 2.7: Checking the port number in linux (1)

stalled from a package manager then scilab can be launched by opening a terminal (Alt+Ctrl+T) and executing the command `sudo scilab`. If one downloads the portable version then first the file has to unpacked. This can be done by right clicking on it and choosing `Extract here`. Then one has to open the terminal and change the directory to `scilab/bin`. Then the command `sudo ./scilab` must be executed to launch scilab. Note that scilab must always be launched with sudo permisions to be able to communicate with the SBHS.

FTDI COM port drivers are not required for connecting the SBHS to the PC. After plugging in the USB cable to the PC, check the serial port number by typing `ls /dev/ttyUSB*` on the terminal, refer Fig.2.7.

Note down this number and change the value of the variable `port1` inside the ser_init.sce file, refer Fig.2.4.

Except for these changes rest all of the steps mentioned in Section 2.1.2 can be followed.

2.3 Summary of procedure to perform a local experiment

This section sumarrizes and only lists the required commands/activities to be done in the given order to do a local experiment. It doesnot explain the expected results etc. The procedure is common for both windows as well as linux users. However, make sure you have reffered to the earlier sections of this chapter for clarity.

1. Step1: Change/ensure scilab working directory to `Step_test`. Command `pwd` can be used to know the present working directory of scilab

2. Step2: Load the functions available in `common_files` directory by executing the command `getd<space>..\common_files`

3. Step3: Load the serial communication toolbox by executing the command `exec<space>..\common_files\loader.sce`

4. Step4: Ensure correct communication port number in the `ser_init.sce` file and execute it. Execution can be done using the command `exec<space>ser_init.sce`

5. Step5: Load step test function by executing command `exec<space>step_test.sci`

6. Step6: Launch Xcos code for step test and execute it. This can be done using the command `xcos<space>step_test.xcos`

7. Step7: Execute this xcos code by clicking on the start button available on the menubar of xcos window. Let the execution run for sufficient time, until the output reaches a steadstate or untill sufficient data is collected.

Note that advance users can always make use to `start.sce` file for quickly performing the experiments.

2.4 Scilab Code under common_files

Scilab Code 2.1 `comm.sci`

```
1  m=1;
2  function [temp] = comm(heat,fan)
3      global heatdisp fandisp tempdisp sampling_time m
          name handl filename
4
5  if heat<0
6      heat=0
7  end
8  if heat>100
9      heat=100
10 end
11 if fan<0
```

```
12      fan=0
13  end
14  if fan>100
15      fan=100
16  end
17
18  writeserial(handl,ascii(254)); //Input Heater,
        writeserial accepts
                                                strings; so
        convert 254 into its
                                                 string
        equivalent
19  writeserial(handl,ascii(heat));
20  writeserial(handl,ascii(253)); //Input Fan
21  writeserial(handl,ascii(fan));
22  writeserial(handl,ascii(255)); //To read Temp
23  sleep(100);
24
25  temp = ascii(readserial(handl)); // Read serial
        returns a string, so
                                                convert it to
        its integer(ascii)
                                               equivalent
26  temp = temp(1) + 0.1*temp(2); // convert to temp with
        decimal points
        eg: 40.7
27  epoch=getdate('s');
28  dt=getdate();
29  ms=dt(10);
30  epoch=(epoch*1000)+ms;
31
32  A = [m,heat,fan,temp,epoch];
33
34  fdfh = file('open',filename,'unknown');
35
36  file('last', fdfh)
37
38  write(fdfh,A,'(7(f15.1,3x))');
```

```
39
40 file('close', fdfh);
41
42 endfunction
```

Scilab Code 2.2 `init.sci`

```
1  global filename m
2  function status = init(port)
3      global handl filename
4
5      OS = getos();
6
7      if OS == string('Linux')
8          port_num = port(1);
9          handl = openserial(port_num,"9600,n,8,0")
10     else
11         port_num = port(2);
12         handl = openserial(port_num,"9600,n,8")
13     end
14
15 if (ascii(handl) ~= [])
16   status = string('COM PORT OPENED')
17 else
18     status = string('ERROR: Check port number or USB
         connection')
19 end
20
21 m = 1;
22
23 dt = getdate();
24 year = dt(1);
25 month = dt(2);
26 day = dt(6);
27 hour = dt(7);
28 minutes = dt(8);
29 seconds = dt(9);
30
```

```
31
32 file1 = strcat(string([year month day hour minutes
       seconds]),'-');
33 string txt;
34 filename = strcat([file1, "txt"],'.');
35
36 endfunction
```

Scilab Code 2.3 `plotting.sci`

```
1  function [] = plotting(var,low_lim,high_lim)
2
3      global heatdisp fandisp tempdisp setpointdisp
          sampling_time m
4
5      timeTitle = "No. of samples with sampling time = "
          +string(sampling_time)
6      if low_lim~=[] & high_lim~=[]
7          heat_min = low_lim(1)
8          fan_min = low_lim(2)
9          temp_min = low_lim(3)
10         time_min = low_lim(4)
11
12         heat_max = high_lim(1)
13         fan_max = high_lim(2)
14         temp_max = high_lim(3)
15         time_max = high_lim(4)
16
17     else
18         heat_min = 0
19         fan_min = 0
20         temp_min = 20
21         time_min = 0
22
23         heat_max = 100
24         fan_max = 100
25         temp_max = 100
26         time_max = 1000
```

```
27      end
28
29
30
31      if length(var)==3
32          heat = var(1);
33          fan = var(2);
34          temp = var(3);
35
36          heatdisp=[heatdisp;heat];
37          subplot(311);
38          xtitle("",timeTitle,"Heat in percentage")
39          plot2d(heatdisp,rect=[time_min,heat_min,
                time_max,heat_max],style=1)
40
41          fandisp=[fandisp;fan];
42          subplot(312);
43          xtitle("",timeTitle,"Fan in percentage")
44          plot2d(fandisp,rect=[time_min,fan_min,
                time_max,fan_max],style=2)
45
46          tempdisp=[tempdisp;temp];
47          subplot(313)
48          xtitle("",timeTitle,"Temperature (deg
                celcius)")
49          plot2d(tempdisp,rect=[time_min,temp_min,
                time_max,temp_max],style=5)
50
51
52        elseif length(var) == 4
53
54          heat = var(1);
55          fan = var(2);
56          temp = var(3);
57          setpoint = var(4);
58
59          heatdisp=[heatdisp;heat];
60          subplot(311);
```

```
61              xtitle("",timeTitle,"Heat in percentage")
62              plot2d(heatdisp,rect=[time_min,heat_min,
                    time_max,heat_max],style=1)
63
64              fandisp=[fandisp;fan];
65              subplot(312);
66              xtitle("",timeTitle,"Fan in percentage")
67              plot2d(fandisp,rect=[time_min,fan_min,
                    time_max,fan_max],style=2)
68
69              tempdisp=[tempdisp;temp];
70              setpointdisp=[setpointdisp;setpoint]
71              subplot(313)
72              xtitle("",timeTitle,"Temperature (deg
                    celcius)")
73              plot2d(tempdisp,rect=[time_min,temp_min,
                    time_max,temp_max],style=5)
74              plot2d(setpointdisp,rect=[time_min,temp_min,
                    time_max,temp_max],style=1)
75
76      end
77  endfunction
```

Chapter 3

Using Single Board Heater System, Virtually!

3.1 Introduction to Virtual Labs at IIT Bombay

The concept of virtual laboratory is a brilliant step towards strengthening the education system of an university/college, a metropolitan area or even an entire nation. The idea is to use the ICT i.e. Information and Communications Technology, mainly the Internet for imparting education or exchange of educational information. Virtual Laboratory mainly focuses on providing the laboratory facility, virtually. Various experimental set-ups are hooked up to the internet and made available to use for the external world. Hence, anybody can connect to that equipment over the internet and carry out various experiments pertaining to it. The beauty of this idea is that a college who cannot afford to have some experimental equipments can still provide laboratory support to their students through virtual lab, and all that will cost it is a fair Internet connection! Moreover, the laboratory work does not ends with the college hours, one can always use the virtual lab at any time and at any place assuming the availability of an internet connection.

A virtual laboratory for SBHS is launched at IIT Bombay. Here is the url to access it: **vlabs.iitb.ac.in/sbhs/**. A set of 36 SBHS are made available to use over the internet 24× 7. These individual kits are made available to the users on hourly basis. We have a slot booking mechanism to achieve this. Since there are 36 SBHS connected with an hours slot for 24 hrs a day, we have 864 one hour slots a day. This means that 864 individual users can access the SBHS in a day for an hour.

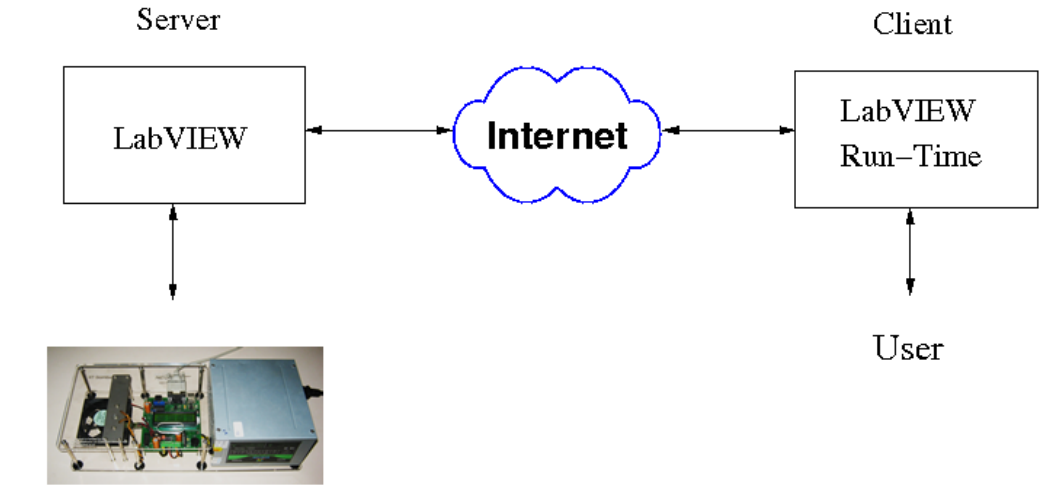

Figure 3.1: SBHS virtual laboratory with remote access using LabVIEW

This also means that up to 6048 users can use the SBHS for an hour in a week and 181440 in a month! A web page is hosted which is the first interface to the user. The user registers/logs in himself/herself here. The user is also supposed to book a slot for accessing the SBHS. A database server maintains a record of the data generated through the web interface. A python script is hosted on the server side and it helps in connecting the user with the corresponding SBHS placed remotely. A free and open source scientific computing Software, Scilab, is used by the user for implementing the experiment on SBHS, in terms of simple Scilab coding.

3.2 Evolution of SBHS virtual labs

In [**?**], the control algorithm is implemented at the server end and the remote student just keys in the parameters, as shown in Figure 3.1. LabVIEW was used for the implementation of the same. The server end consisted of a computer connected with an SBHS with a full blown copy of LabVIEW installed on it. The client has a LabVIEW run time engine available for free download from the National Instruments website. A few LabVIEW algorithms/experiments were hosted on the server. The client accesses these algorithm/experiment over the Internet using a web browser by entering appropriate parameters.

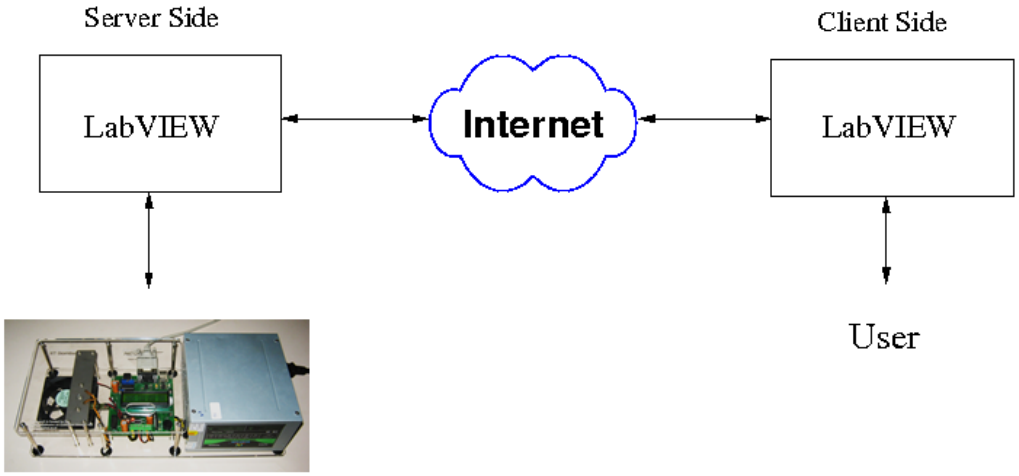

Figure 3.2: SBHS virtual laboratory with remote access and live data sharing using LabVIEW

It was realized that the learning experience is not complete for this structure. This is because the server hosts some pre-built LabVIEW algorithms and a user can only access these few algorithms. The user can in no way change the program and can only input experimental parameters. Hence, we came up with a new architecture as shown in the Figure 3.2 that used full blown copies of LabVIEW at both server and client ends.

This idea uses the DataSocket technology of LabVIEW. Since now the client is having a complete LabVIEW installation on his/her computer she can now implement her own algorithms. Thus this architecture did provide a complete learning experience to the students. There are some shortcomings as well:

- LabVIEW is expensive and students may not be able to afford to buy it. It is also prohibitively expensive for the Government to distribute it.
- We used the LabVIEW version 8.04, which had restricted scripting language. It was tedious to create new control algorithms in it.

This made us shift to free and open source (FOSS) software. We replaced LabVIEW with Java and Scilab as shown in Figure 3.3. Scilab at the server end is

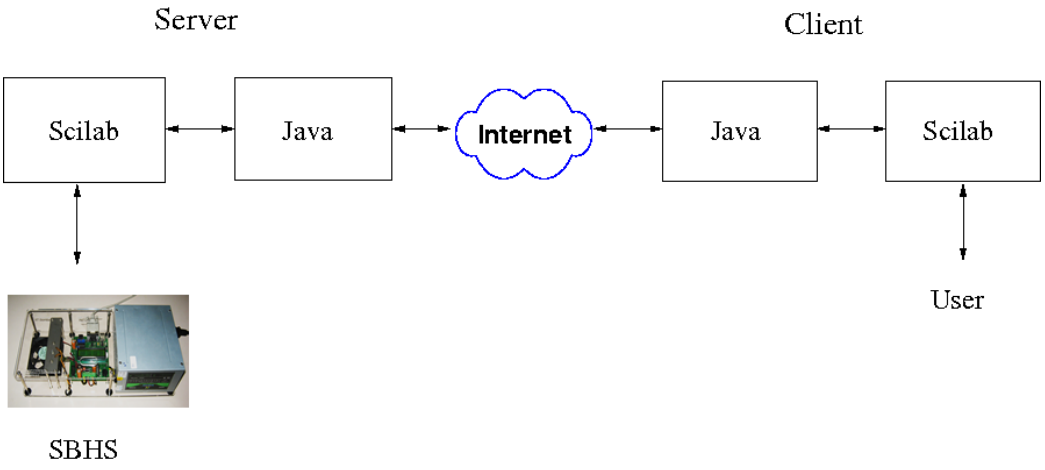

Figure 3.3: SBHS virtual laboratory using open source software

used for communicating with SBHS. Scilab at the client end is used for implementing the algorithms. Java is used at both the server as well as client end for communication over the Internet thereby connecting the client with the server.

For the above solution, we need a dedicated copy of scilab running at the server end for every SBHS. One way to do this is to host it on multiple computers with unique IPs. Hence the number of SBHS we want to host requires as many computer's and public IPs thereby making it expensive. Moreover, it also limits its scalability. The other way to do this is to host multiple java and scilab servers on the same computer. Hosting many copies of Scilab simultaneously requires a powerful computer for the server.

For these reasons we decided to take scilab off the server computer and to use java alone to communicate with the SBHS directly. Java also communicates with the client computer. We connected seven SBHS systems to a USB port through a serial port hub. This architecture was implemented on a Windows Operating System. We faced the following difficulties in this solution.

- When we connected more than one serial hub to a PC, the port ID could not be retrieved correctly. Port ID information is required if we want a student to use the same SBHS for all their experiments during different sessions.

- The experiments required time stamping of the data communicated to and from the server. But this time stamping was not linear and suffered instability.

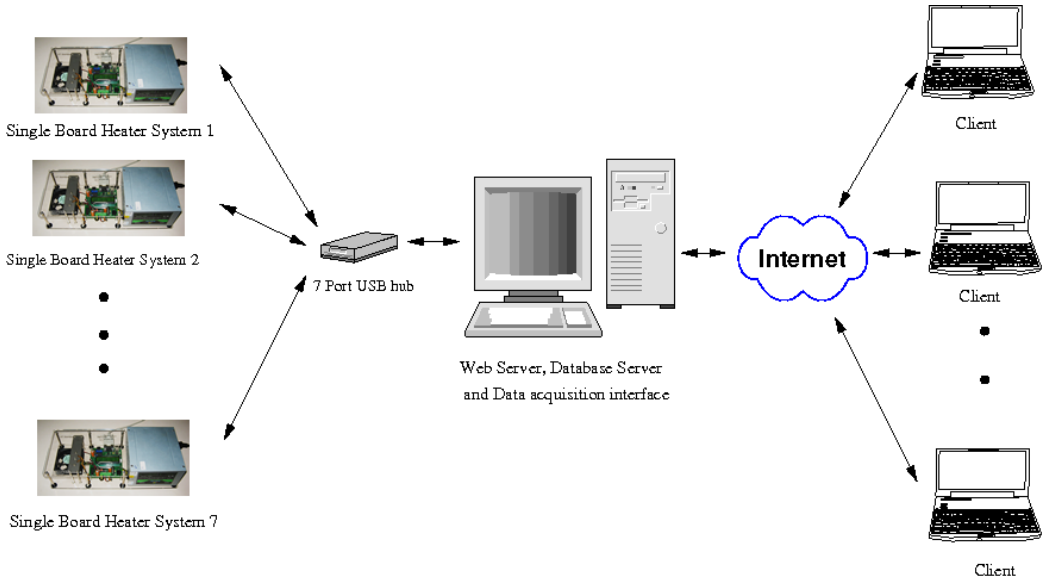

Figure 3.4: Virtual control lab hardware architecture

This made us to completely switch to FOSS with Ubuntu Linux as the OS and is the current structure of the Virtual lab as shown in Figure 3.6

3.3 Current Hardware Architecture

The current hardware architecture of the virtual single-board heater system lab involves 36 single-board heater systems connected to the server via multiple 7 and 10-port USB hubs. The server computer is connected to a high speed internetwork and has enough processing capability to host data acquisition, database, and web servers. It has been successfully tested for the undergraduate Process Control course and the graduate Digital Control and Embedded systems courses conducted at IIT Bombay as well as few workshops over the internet. Currently, this architecture is integrated with a cameras on each SBHS to facilitate live video streaming. This gives the user a feel of remote hands-on.

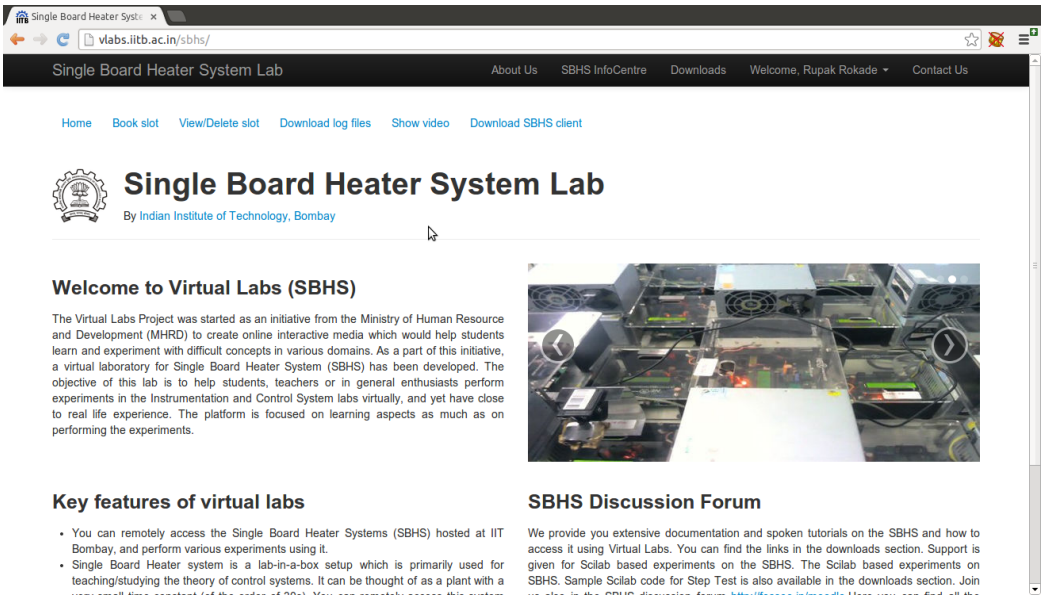

Figure 3.5: Home page of SBHS V Labs

3.4 Current Software Architecture

The current software architecture of this virtual SBHS control lab is shown in Figure 3.6. The server computer runs Ubuntu Linux 12.04.2 OS. It hosts a Apache-MySQL server. The SBHS server is based on Python-Django framework and is linked to Apache server using Apache's WSGI module. The MySQL database server has the details of all the registered users, their slot details, authentication keys to allow remote access, etc. As shown in Figure **??**, the Python-Django server has pages for registration, login, slot booking etc. [**?**]. On the client end, control algorithms are running in Scilab and a python based client application communicates with virtual labs server over the Internet.

The steps to be performed before and during each experiment are explained next.

3.5 Conducting experiments using the Virtual lab

This section explains the procedure to use Single Board Heater System remotely using Scilab i.e. when you are accessing SBHS remotly using your computer over

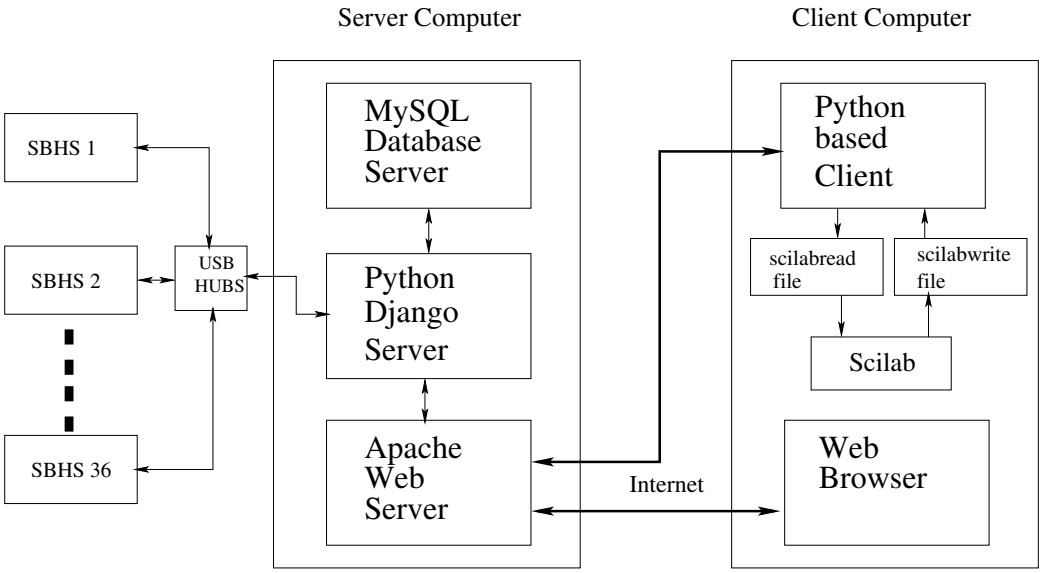

Figure 3.6: Current Architecture of SBHS Virtual Labs

the virtual labs platform. 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 section, unless specified otherwise. Let us first see the required files to be downloaded and installations to be done. Scilab is required to be installed on your computer. Please refer to Section 2.1.1 and Section 2.2 for the procedure to install scilab on Windows and Linux system, respectively.

SBHS scilab code for your OS, under the section `SBHS Virtual Code`, must be downloaded from `http://sbhs.os-hardware.in/downloads`. The code downloaded will be in zip format. After the zip is unpacked, you will see scilab experiment folders such as `Step_test`, `Ramp_Test`, `pid_controller` etc. We will be using the Step_test folder. Do not alter the directory structure. If you want to copy or move an experiment outside the directory then make sure you also copy the `common_files` folder. The `common_files` folder must always be one directory outside the experiment folder. Now given that you have scilab installed and working and the required scilab code downloaded, let us see the step-by-step procedure to do a remote experiment.

Figure 3.7: Show Video

3.5.1 Registration, Login and Slot Booking

Go to the website `sbhs.os-hardware.in` and click on the `Virtual labs` link available on the left hand side. The home page of Virtual labs is illustrated in Fig 3.5. If you are a first time user, click on the link `Login/Register`. Fill out the registration form and submit it. If the registration form is submitted successfully, you will receive an activation link on your registered Email id. Use this link to complete the registration process. If you skip this step you will not be able to login. Registration is a one time process and need not be repeated more than once. After completing registration login with your username and password. You should now get the options to Book Slot, Delete Slot etc.

View/Delete slot option allows you to delete your booked slots. This option however will work only for slots booked for the future. You cannot delete a past or the current slot. Download log files option gives you the facility to download your experiment log files. Clicking on it will give you a list of all of the experiments you had performed. Show video option can be used to see the live video feed of your SBHS. Web cameras are mounted on every SBHS. You can see the display of your SBHS as shown in Fig. 3.7.

Clicking on the Book slot option will allow you to book an experiment time slot. Slots are of 55 minutes duration. Click on the Book slot option. If the current slot is free, Book now option will appear. Click on it. Else you have to book an advance slot for the next hour or any other future time using the calender that appears on this page. There is a limit to how many slots one can book in a day. We are allowing only two non-consecutive slots, per user, to be booked in a day. However, there is no limit to how many current slots you book and use. Book an

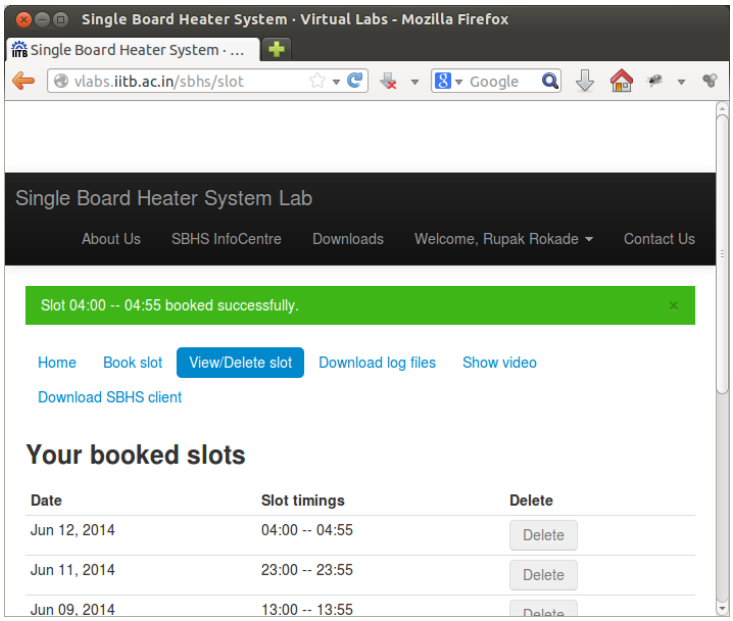

Figure 3.8: Slot booking

experiment slot. Once you successfully book a slot a **Slot booked successfully** message highlighted in green color will appear on the top side. This is shown in Fig. 3.8. It will automatically take you to the View/Delete slot page.

3.5.2 Configuring proxy settings and executing python based client

After booking a slot, the web activity is over. You may close the web browser unless you need it open to see live video feed of your SBHS. The next step is to establish the communication link between the server and your computer. A python based application is created which handles the network communication.

Let us first see how to do the proxy settings if you are behind a proxy network. Open the folder `common_files`. Open the file `config`. This files contains various arguments whose values must be eneterd to configure proxy.

Do not change the contents of `config` file if

- You are accessing from inside IIT Bombay OR
- You are accessing from outside IIT Bombay and using an open network

such as at home OR using a mobile internet

Change the contents of `config` file if

- You are outside IIT Bombay and using a proxy network such as at an institute, office etc.

If you have to put the proxy details, first change the argument `use_proxy` = Yes (Y should be capital in Yes and N should be capital in No). Fill in the other details as per your proxy network. If your proxy network allows un-authenticated login then make the argument `proxy_username` and `proxy_password` blank. This proxy setting has to done only once.

Open the `Step_test` folder. Double click on the file `run`. This will open the client application as shown in Fig. 3.9. Note that for first time execution, it will take a minute to open the client application. It will show various parameters related to the experiment such as SBHS connection, Client version, User login and Experiment status. The green indicators show that the corresponding activity is correct or functional. Here it says that the Client application is been able to connect to the server and the client version being used is the latest. The User login and Experiment status is showing red and will turn green after a registered username and password is entered. If the SBHS is offline or there are some other issues, the corresponding error will be displayed and the respective indicator will turn red. Enter your registered username and password and press login. You should get the message `Ready to execute scilab code`. The application also shows the value of iteration, heat, fan, temperature and time remaining for experimentation. It also shows the name of log file created for the experiment.

3.5.3 Executing scilab code

Inside the `StepTest` folder, if on a windows system, double click on the file `stepc.sce`. This should automatically launch scilab and also open the `stepc.sce` in the scilab editor. It will also automatically change the scilabs working directory. On a linux system, launch scilab manually. Then change the scilab working directory to the folder `StepTest`. This can be done by clicking on `File` menu and then selecting `change current directory`. Next, execute the command `getd ../common_files`. Scilab command `getd` is used to load all functions defined in all .sci files inside a specified folder. Here we have some important function files inside the `../common_files` directory. Executing this command will load all of

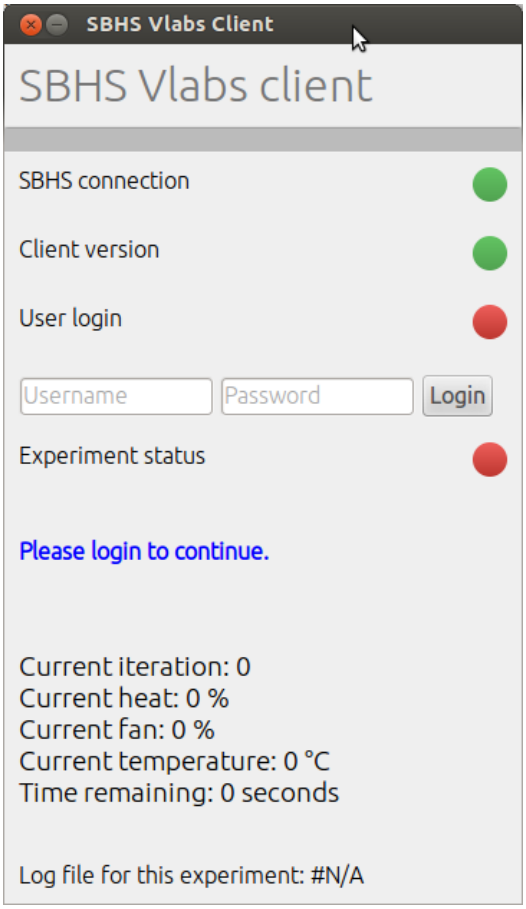

Figure 3.9: Python Client

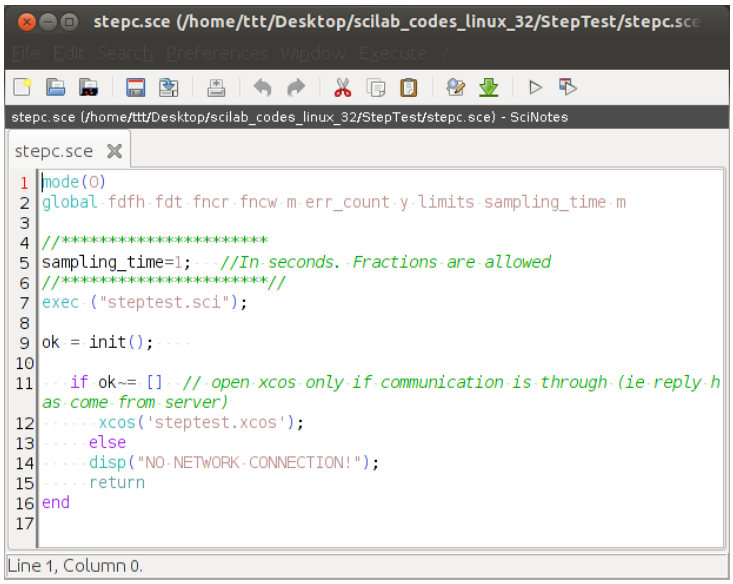

Figure 3.10: stepc.sce file

the functions that the experiment needs. Open the file `stepc.sce` using the `Open` option inside `File` menu. The file is shown in Fig. 3.10

The experiment sampling time can be set inside the `stepc.sce` file. You may want to change it to a higher value if your network is slow. The default value of 1 second works fine in most cases. On the menu bar, click on `Execute` option and choose option `file with echo`. This will execute the scilab code. If the network is working fine, an xcos diagram will open automatically. If it doesnt open then see the scilab console for error messages. If you get a `No network connection` error message then try executing the scilab code again. The xcos diagram is for the step test experiment as shown in Fig. 3.11. You can set the value of the heat and fan. Keep the default values. On the menu bar of the xcos window, click on `start` button. This will execute the xcos diagram. If there is no error, you will get a graphic window with three plots. It will show the value of Heat in % Fan in % and ..temperature in degree celcius as shown in Fig. 3.12. After sufficient time of experimentation click on the stop button to stop the experiment. Go to the `StepTest` folder. Here you will find a `logs` folder. This folder will have another folder named after your username. It will have the log file for your experiment. Read the log file name as
YearMonthDate_hours_minutes_seconds.txt. This log file contains all the values

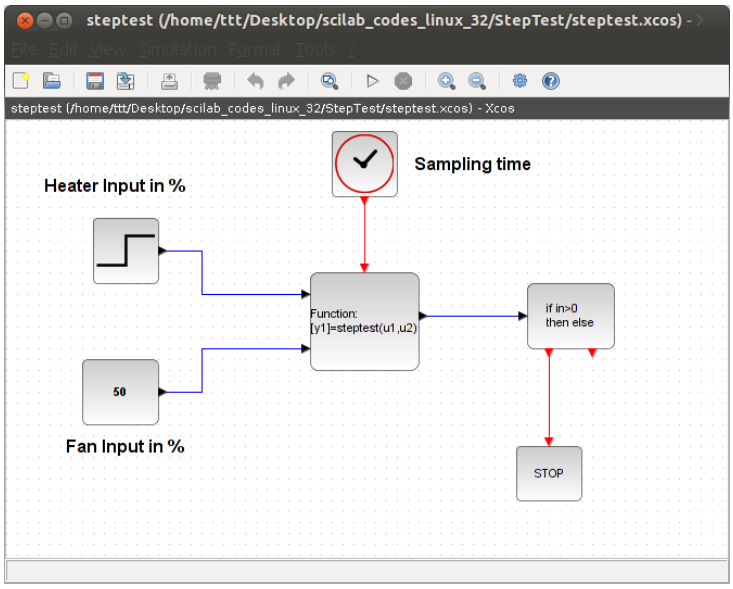

Figure 3.11: Xcos for step test

of heat fan and temperature. It can be used for further analysis.

3.6 Summary

This section summarizes the process to perform an experiment on SBHS using virtual lab interface. This section assumes that the user has already created an account and booked a slot as explained in section 3.5.1. It also assumes that the proxy settings are already done as explained in section 3.5.2. The user should follow these steps within the booked slot time.

- Step1: Open the StepTest experiment directory
- Step2: Double-click on the file `run`. Expect the SBHS cient application to open.
- Step3: Enter the username and password inside the SBHS cient application and press `login` button. Expect the message `Ready to execute scilab code`
- Step4: Switch to the StepTest experiment directory and double-click on the file `stepc.sce`. This will launch scilab and also open the file `stepc.sce`

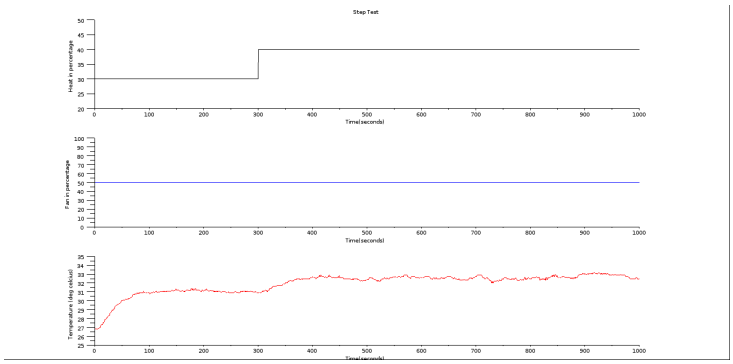

Figure 3.12: Output of Step Test

in the scilab editor. Linux users will have to launch scilab manually. They also have to change the working directory to StepTest and then open the `stepc.sce` file in the scilab editor.

- Switch to the scilab console and execute the command `getd ../common_files`
- Step5: Execute the file `stepc.sce`. Expect the step test xcos diagram to open automatically. If this doesnt happen, check the scilab console for error message.
- Step6: Execute the step test xcos daiagram. You may change the input parameters, if required, before executing. Expect a plot window to open automatically showing three graphs.
- Step7: Stop the Xcos simulation after the experiment is completed properly.

Chapter 4

Identification of Transfer Function of a Single Board Heater System through Step Response Experiment

The aim of this experiment is to perform step test on a Single Board Heater System and to identify system transfer function using step response data. The target group is anyone who has basic knowledge of control engineering.

We have used Scilab and Xcos as an interface for sending and receiving data. Xcos diagram is shown in figure 4.1. Heater current and fan speed are the two inputs for this system. They are given in percentage of maximum. These inputs can be varied by setting the properties of the input block's properties in Xcos. The plots of their amplitude versus number of collected samples are also available on the scope windows. The output temperature profile, as read by the sensor, is also plotted. The data acquired in the process is stored on the local drive and is available to the user for further calculations.

In the `step_test.xcos` file, open the heater block's parameters to apply a step change of say 10 percent to the heater at operating point of 30 percent of heater after 250 seconds. The block parameters of the step input block will have `Step time = 250`, `Initial value = 30` and `Final value = 40`. Keep the fan input constant at 50 percent. Start the experiment and let it continue until you see the temperature reach the steady state.

The step test data file will be saved in `Step_test` folder. The name of the file will be the date and time at which the experiment was conducted. A sample data file is provided in the same folder. The sample data file is named as `step-data-local.txt` and `step-data-virtual.txt`. Refer to the one de-

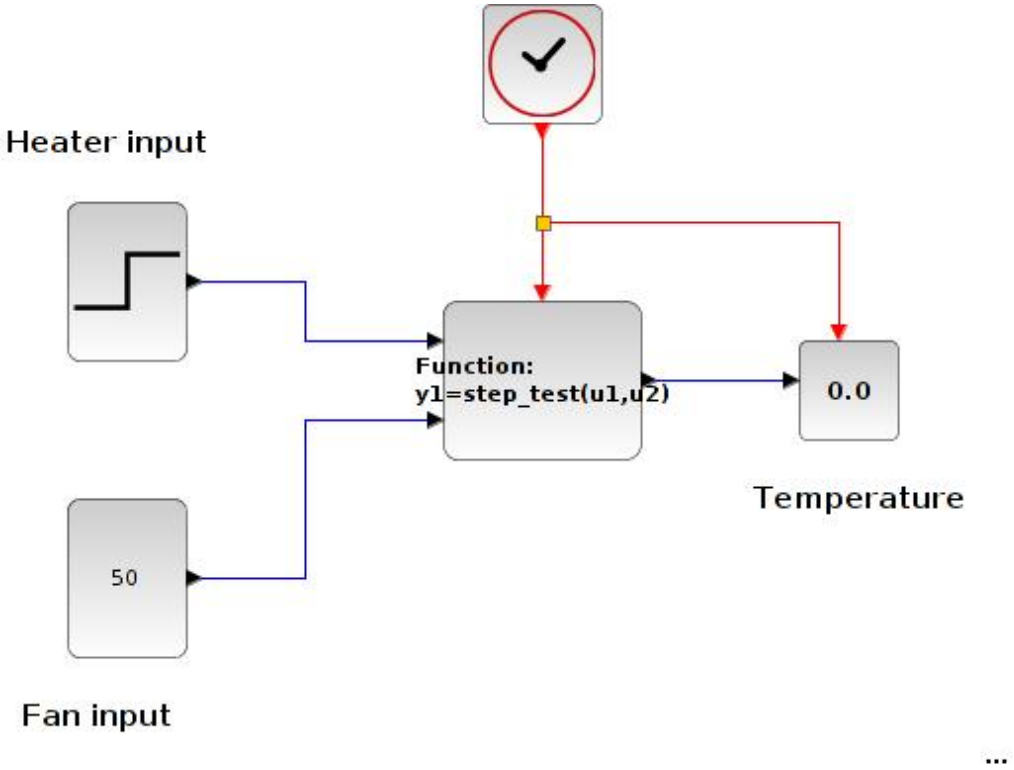

Figure 4.1: Xcos for this experiment

1.0	30.0	50.0	29.3	1412400132192.0
2.0	30.0	50.0	29.5	1412400133044.0
.				
.				
820.0	40.0	50.0	37.0	1412400950197.0
821.0	40.0	50.0	37.2	1412400951202.0

Table 4.1: Step data obtained after performing local Step Test

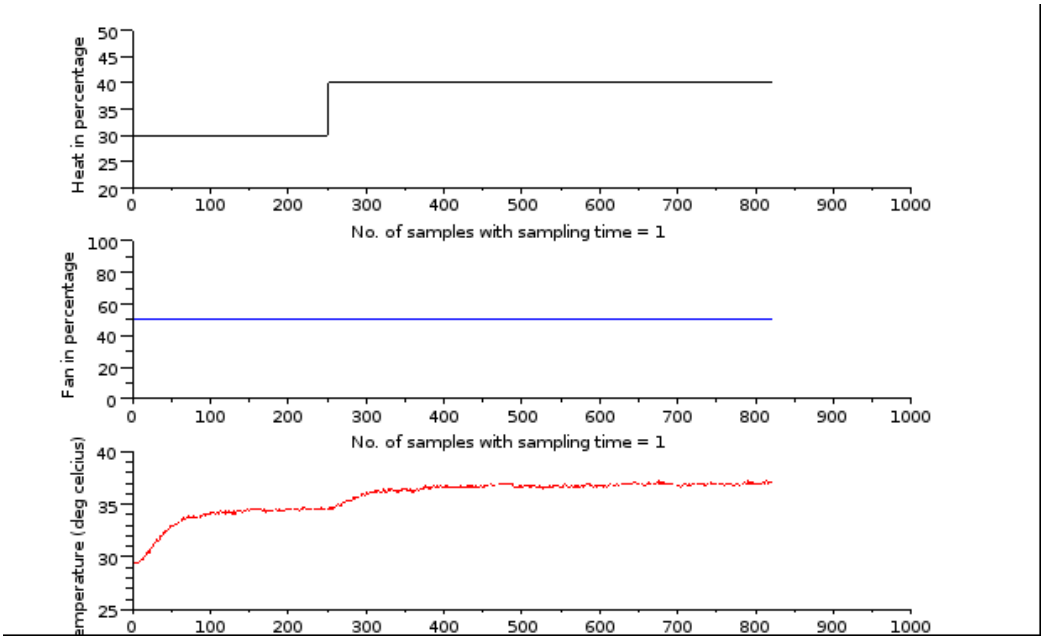

Figure 4.2: Graph shows heater current, fan speed and output temperature

pending on wheather you are performing a local or a virtual experiment. Referring to the data file thus obtained as shown in table 4.1, the first column in this table denotes samples. The second column in this table denotes heater in percentage. It starts at 30 and increases with a step size of 10 units. The third column denotes the fan in percentage. It has been held constant at 50 percent. The fourth column refers to the value of temperature. The fifth column denotes time stamp. The virtual data file will have total four time stamp columns apart from first 3 columns. These four time stamp columns are client departure, server arrival, server departure and client arrival. These can be used for advanced control algorithms. These additional time stamps exist in virtual mode because of the presense of network delay.

4.1 Conducting Step Test on SBHS locally

The detailed procedure to perform a local experiment is explained in Chapter2. A summary of the same is provided in section 2.3 It is exactly the same for this section.

4.2 Conducting Step Test on SBHS, virtually

The detailed procedure to perform a local experiment is explained in Chapter3. A summary of the same is provided in section 3.5 It is exactly the same for this section.

4.3 Identifying First Order and Second Order Transfer Functions

In this section we shall determine the first and second order transer function model using the data obtained after performing step test experiment. Please note that this procedure is common for data obtained using both local and virtual experiments.

4.3.1 Determination of First Order Transfer Function

Identification of the transfer function of a system is important as it helps us to represent the physical system mathematically. Once the transfer function is obtained, one can acquire the response of the system for various inputs without actually applying them to the system.

Consider the standard first order transfer function given below

$$G(s) = \frac{C(s)}{R(s)} \tag{4.1}$$

$$G(s) = \frac{1}{\tau s + 1} \tag{4.2}$$

Rewriting the equation, we get

$$C(s) = \frac{R(s)}{\tau s + 1} \tag{4.3}$$

A step is given as input to the first order system. The Laplace transform of a step function is $\frac{1}{s}$. Hence, substituting $R(s) = \frac{1}{s}$ in equation 4.3, we obtain

$$C(s) = \frac{1}{\tau s + 1}\frac{1}{s} \tag{4.4}$$

Solving $C(s)$ using partial fraction expansion, we get

$$C(s) = \frac{1}{s} - \frac{1}{s + \frac{1}{\tau}} \tag{4.5}$$

Taking the Inverse Laplace transform of equation 4.5, we get

$$c(t) = 1 - e^{\frac{-t}{\tau}} \tag{4.6}$$

From the above equation it is clear that for t=0, the value of c(t) is zero. For t= ∞, c(t) approaches unity. Also, as the value of 't 'becomes equal to τ, the value of c(t) becomes 0.632. τ is called the time constant and represents the speed of response of the system. But it should be noted that, smaller the time constant-faster the system response. By getting the value of τ, one can identify the transfer function of the system.

Consider the system to be first order. We try to fit a first order transfer function of the form

$$G(s) = \frac{K}{\tau s + 1} \tag{4.7}$$

to the Single Board Heater System. Because the transfer function approach uses deviation variables, $G(s)$ denotes the Laplace transform of the gain of the system between the change in heater current and the change in the system temperature. Let the change in the heater current be denoted by Δu. We denote both the time domain and the Laplace transform variable by the same lower case variable. Let the change in temperature be denoted by y. Let the current change by a step of size u. Then, we obtain the following relation between the current and the temperature.

$$y(s) = G(s)u(s) \tag{4.8}$$

$$y(s) = \frac{K}{\tau s + 1}\frac{\Delta u}{s} \tag{4.9}$$

Note that Δ u is the height of the step and hence is a constant. On inversion, we obtain

$$y(s) = K[1 - e^{\frac{-t}{\tau}}]\Delta u \tag{4.10}$$

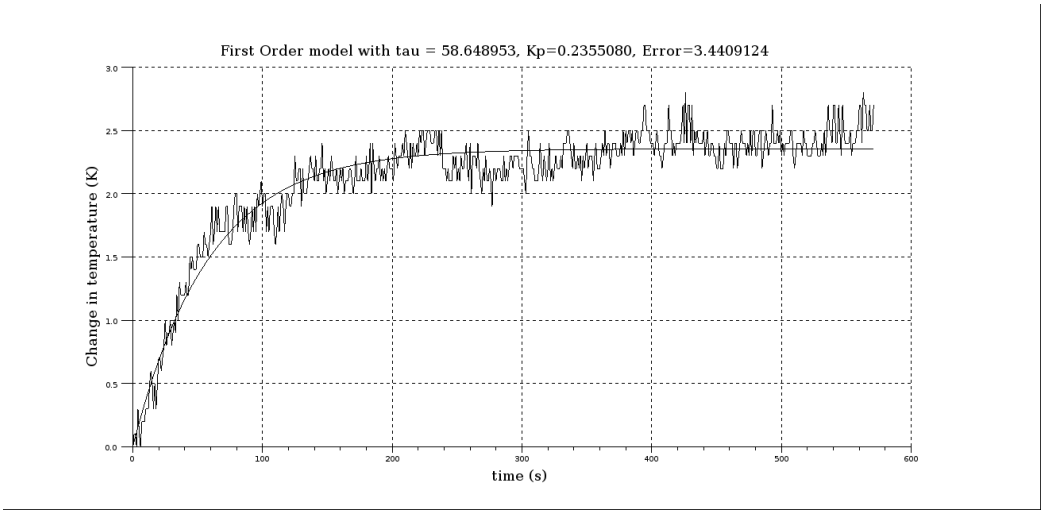

Figure 4.3: Output of the Scilab code `firstorder.sce`

4.3.2 Procedure

1. Download the Analysis folder from the sbhs website. It will be available under `downloads` section. The download will be in zip format. Extrat the downloaded zip file. You will get a folder `Analysis`.

2. Open the `Analysis` folder and then locate and open the folder `Step_Analysis`.

3. Open the `Kp-tau-order1` folder.

4. Copy the step test data file to the folder `Kp-tau-order1`.

5. Change the Scilab working directory to `Kp-tau-order1` folder under `Step_Analysis` folder.

6. Open the file `firstorder.sce` in scilab editor and enter the name of the data file (with extention) in the `filename` field.

7. Save and run this code and obtain the plot as shown in figure 4.3.

This code uses the routines `label.sci` and `costf_1.sci`

The plot thus obtained is reasonably good. See the Scilab plot to get the values of τ and K. The figure 4.3 shows a screen shot of the same. We obtain $\tau = 58.64$, K = 0.23. The transfer function obtained here is at the operating point of 30 percentage of heat. If the experiment is repeated at a different operating point, the transfer function obtained will be different. The gain will correspondingly be more at a higher operating point. This means that the plant is faster at higher temperature. Thus the transfer function of the plant varies with the operating point. Let the transfer function we obtain in this experiment be denoted as G_s. We obtain

$$G_s(s) = \frac{0.23}{58.64s + 1} \tag{4.11}$$

4.4 Determination of Second Order Transfer Function

In this section, we explore the efficacy of a second order model of the form

$$G(s) = \frac{K}{(\tau_1 s + 1)(\tau_2 s + 1)} \tag{4.12}$$

The response of the system to a step input of height Δu is given by

$$y(s) = \frac{K}{(\tau_1 s + 1)(\tau_2 s + 1)} \frac{\Delta u}{s} \tag{4.13}$$

Splitting into partial fraction expansion, we obtain

$$y(s) = \frac{K}{\tau_1 \tau_2} \frac{1}{\left(s + \frac{1}{\tau_1}\right)\left(s + \frac{1}{\tau_2}\right)} = \frac{A}{s} + \frac{B}{s + \frac{1}{\tau_1}} + \frac{C}{s + \frac{1}{\tau_2}}$$

Through Heaviside expansion method, we determine the coefficients:

$$A = K$$
$$B = -\frac{K\tau_1}{\tau_1 - \tau_2}$$
$$C = \frac{K\tau_2}{\tau_1 - \tau_2}$$

On substitution and inversion, we obtain

$$y(t) = K\left[1 - \frac{1}{\tau_1 - \tau_2}\left(\tau_1 e^{-t/\tau_1} - \tau_2 e^{-t/\tau_2}\right)\right] \tag{4.14}$$

We have to determine three parameters K, τ_1 and τ_2 through optimization. Once again, we follow a procedure identical to the first order model. The only difference is that we now have to determine three parameters. Scilab code `secondorder.sce` calculates the gain and two time constants.

4.4.1 Procedure

1. Download the Analysis folder from the sbhs website. It will be available under `downloads` section. The download will be in zip format. Extrat the downloaded zip file. You will get a folder `Analysis`.
2. Open the `Analysis` folder and then locate and open the folder `Step_Analysis`.
3. Open the `Kp-tau-order2` folder.
4. Copy the step test data file to the folder `Kp-tau-order2`.
5. Change the Scilab working directory to `Kp-tau-order2` folder under `Step_Analysis` folder.
6. Open the file `secondorder.sce` in scilab editor and enter the name of the data file (with extention) in the `filename` field.
7. Save and run this code and obtain the plot as shown in figure 4.4.

$$G_s(s) = \frac{0.235}{(57.39s + 1)(1s + 1)} \tag{4.15}$$

The fit is much better now. In particular, the initial inflexion is well captured by this second order transfer function.

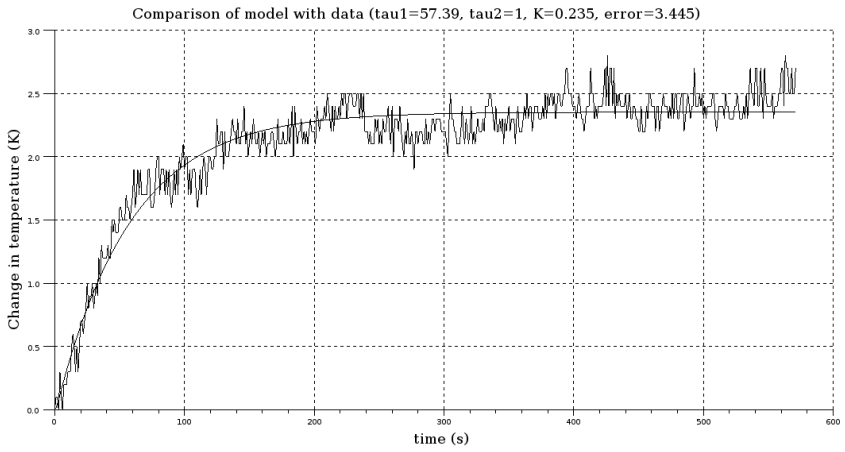

Figure 4.4: Output of the Scilab code `secondorder.sce`

4.5 Discussion

We summarize our findings now. For the first order analysis, the gain is 0.23 and the time constant τ is 58.64 seconds. For the second order analysis, the initial inflexion is well captured with the two time constants τ_1=57.39, τ_2= 1 and gain = 0.235. Negative steps can also be introduced to make the experiment more informative. One need not keep a particular input constant. By varying both the inputs, one can imagine it to be like a step varying disturbance signal.

4.6 Scilab Code

Scilab Code 4.1 `label.sci`

```
1 // Updated (9-12-06), written by Inderpreet Arora
2 // Input arguments: title, xlabel, ylabel and their
      font sizes
3
4 function label(tname,tfont,labelx,labely,xyfont)
5 a = get("current_axes")
6 xtitle(tname,labelx,labely)
7 xgrid
```

```
t = a.title;
t.font_size = tfont; // Title font size
t.font_style = 2; // Title font style
t.text = tname;

u = a.x_label;
u.font_size = xyfont; // Label font size
u.font_style = 2; // Label font style

v = a.y_label;
v.font_size = xyfont; // Label font size
v.font_style = 2; // Label font style

// a.label_font_size = 3;

endfunction;
```

Scilab Code 4.2 costf_1.sci

```
function [f,g,ind] = costf_1(x,ind)
kp = x(1); tau = x(2);
y_prediction = kp * ( 1 - exp(-t/tau) );
f = (norm(y-y_prediction,2))^2;
g = numdiff(func_1,x);
endfunction

function f = func_1(x)
kp = x(1); tau = x(2);
y_prediction = kp * ( 1 - exp(-t/tau) );
f = (norm(y-y_prediction,2))^2;
endfunction
```

Scilab Code 4.3 firstorder.sce

```
mode(0)
filename = "step-data-local.txt"

clf
```

```
5  exec('costf_1.sci');
6  exec('label.sci');
7  data = fscanfMat(filename);
8  time = data(:, 5);
9  heater = int(data(:, 2));
10 fan = int(data(:, 3));
11 temp = data(:, 4);
12
13
14 len = length(heater);
15
16 time1 = time - time(1);
17 time2 = time1/1000;
18
19 len = length(heater);
20 heaters1 = [heater(1); heater(1:len-1)];
21 del_heat = heater - heaters1;
22 ind = find(del_heat>1);
23
24 step_instant = ind($)-1;
25
26 t = time2(step_instant:len);
27 t = t - t(1);
28 H = heater(step_instant:len);
29 F = fan(step_instant:len);
30 T = temp(step_instant:len);
31 T = T - T(1);
32 delta_u = heater(step_instant + 1)- heater(
       step_instant);
33
34 // finding Kp and Tau between Heater (H) and
       Temperature (T)
35 y = T;     // temperature
36 global('y','t');
37 x0 = [.3  40];
38 //[f,xopt,gopt] = optim(costf_1,'b',[0.1  0.1],[5  100],
       x0,'ar')
39 [f, xopt] = optim(costf_1,x0);
```

```
40 lsterr=sqrt(f);
41 kp = xopt(1);
42 tau = xopt(2);
43 y_prediction = kp * ( 1 - exp(-t/tau) );
44 plot2d(t,y_prediction);
45 plot2d(t,y);
46 title = 'First Order model with tau = ';
47 title = title+string(tau);
48 title = title+', Kp='+string(kp/delta_u);
49 title = title+', Error='+string(lsterr)+'';
50 label(title,4,'time (s)','Change in temperature (K)'
   ,4);
51 kp = kp/delta_u
52 tau
```

Scilab Code 4.4 costf_2.sci

```
1 function [f,g,ind] = costf_2(x,ind)
2 kp = x(1); tau1 = x(2); tau2 = x(3);
3 y_prediction = kp * delta_u * (1 - ...
4 (tau1*exp(-(t)/tau1)-tau2*exp(-(t)/tau2)) ...
5 /(tau1-tau2));
6 f = (norm(T-y_prediction,2))^2;
7 g = numdiff(func_2,x);
8 endfunction;
9 function f = func_2(x)
10 kp = x(1); tau1 = x(2); tau2 = x(3);
11 y_prediction = kp * delta_u * (1 - ...
12 (tau1*exp(-(t)/tau1)-tau2*exp(-(t)/tau2)) ...
13 /(tau1-tau2));
14 f = (norm(T-y_prediction,2))^2;
15 endfunction;
```

Scilab Code 4.5 order_2_heater.sci

```
1 function lsterr = order_2(t,H,T,limits,no)
2 x0 = [2 200 150];
3 // delta_u = u(2) - u(1); u = u - u(1); y = y - y(1);
```

```
4
5 delta_u = H(2)-H(1);
6
7
8 [f,xopt,gopt] = optim(costf_2,'b',[0 2 1],[18 300
    350],x0,'ar',200,200)
9 kp = xopt(1); tau1 = xopt(2); tau2 = xopt(3); lsterr =
     sqrt(f);
10 y_prediction = kp * delta_u * (1 - ...
11 (tau1*exp(-(t)/tau1)-tau2*exp(-(t)/tau2)) ...
12 /(tau1-tau2));
13 format('v',6); ord = [T y_prediction]; x = [t t t];
14 // xbasc();
15  plot2d(t,T);
16
17 plot2d(t,y_prediction);
18 title = 'Comparison of model with data (tau1='
19 title = title+string(tau1)+', tau2='+string(tau2)
20 title = title+', K='+string(kp)
21 title = title+', error='+string(lsterr)+')'
22 label(title,4,'time (s)','Change in temperature (K)'
    ,4);
23 endfunction;
```

Scilab Code 4.6 secondorder.sce

```
1 mode(0)
2 filename = "step-data-local.txt";
3 clf
4 exec('costf_2.sci');
5 exec('label.sci');
6 exec ('order_2_heater.sci');
7
8
9 data = fscanfMat(filename);
10 time = data(:,5);
11 heater = int(data(:, 2));
12 fan = int(data(:, 3));
```

```
13 temp = data(:, 4);
14
15 // times=[time(1); time(1:$-1)];
16 time1 = time - time(1);
17 time2 = time1/1000;
18
19
20 // find where the step change happens
21
22 len = length(heater);
23 heaters1 = [heater(1); heater(1:len-1)];
24 del_heat = heater - heaters1;
25 ind = find(del_heat>1);
26
27 step_instant = ind($)-1;
28 t = time2(step_instant:len);
29 t = t - t(1);
30 H = heater(step_instant:len);
31 F = fan(step_instant:len);
32 T = temp(step_instant:len);
33 T = T - T(1);
34
35 // limits = [0,0,500,10]; no=10000; // first step
36 // limits = [400,0,900,26]; no=5000;// second step
37 lsterr = order_2(t,H,T,)
```

Scilab Code 4.7 ser_init.sce

```
1 mode(0)
2 global filename
3 //** Sampling Time **//
4 sampling_time = 1;
5 ////////////////////
6 m=1;
7
8 port1 = '/dev/ttyUSB0';// For linux users
9 port2 = 'COM2';// For windows users
10
```

```
11 res=init([port1 port2]);
12 disp(res)
```

Scilab Code 4.8 step_test.sci

```
1 mode(0)
2 function temp = step_test(heat,fan)
3     temp = comm(heat,fan);
4
5     plotting([heat fan temp],[20 0 25 0],[50 100 40
        1000])
6
7     m=m+1;
8 endfunction
```

Scilab Code 4.9 stepc.sce

```
1 mode(0)
2 global fdfh fdt fncr fncw m err_count y limits
    sampling_time m
3
4 //***********************
5 sampling_time=1;    //In seconds. Fractions are allowed
6 //***********************//
7 exec("steptest.sci");
8
9 ok = init();
10
11    if ok~= []  // open xcos only if communication is
        through (ie reply has come from server)
12        xcos('steptest.xcos');
13      else
14      disp("NO NETWORK CONNECTION!");
15      return
16 end
```

Scilab Code 4.10 steptest.sci

```
1 function [stop] = steptest(heat,fan)
2
3     [stop,temp] = comm(heat,fan); // Never edit this
        line
4     plotting([heat fan temp],[0 0 25 0],[100 100 50
        1000])
5
6 endfunction
```

Chapter 5

Identification of Transfer Function of a Single Board Heater System through Ramp Response Experiment

The aim of this experiment is to perform ramp test on a Single Board Heater System and to identify system transfer function using ramp response data. The target group is anyone who has basic knowledge of control engineering.

We have used Scilab and Xcos as an interface for sending and receiving data. Xcos diagram is shown in figure 5.1. Heater current and fan speed are the two inputs for this system. They are given in percentage of maximum. These inputs

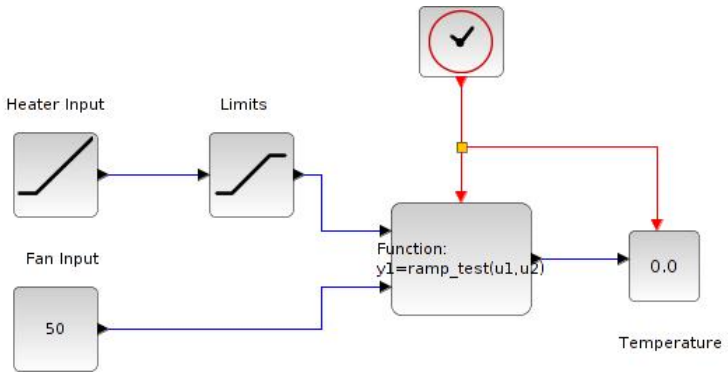

Figure 5.1: Xcos for ramp test experiment

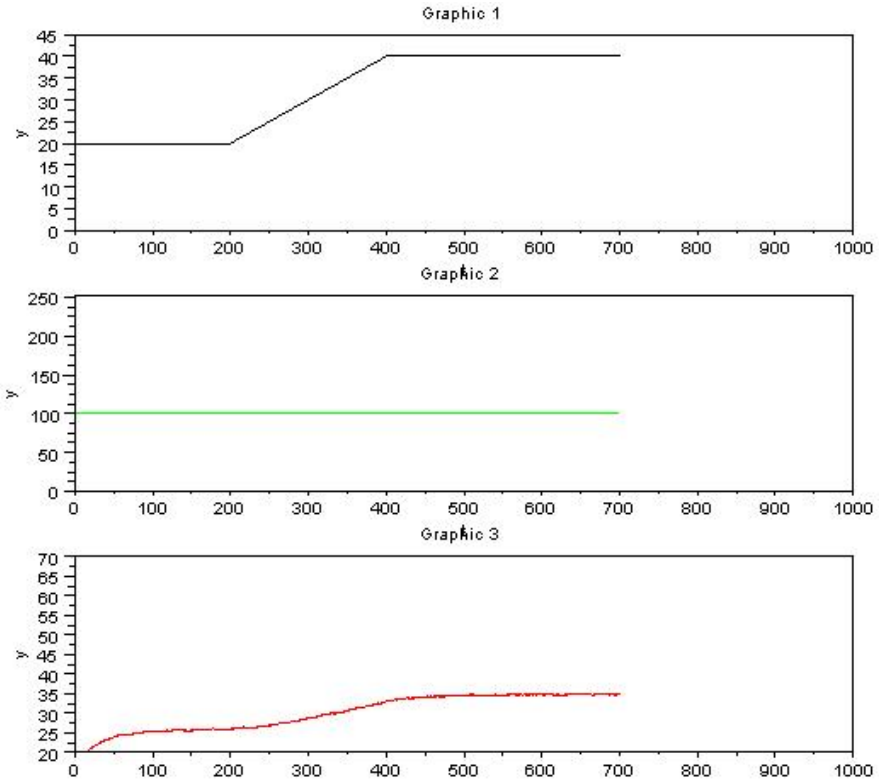

Figure 5.2: Screen shot of ramp test experiment

```
Add Ramp Local Data here
.
820.0    40.0     50.0    37.0   1412400950197.0
821.0    40.0     50.0    37.2   1412400951202.0
```

Table 5.1: Ramp data obtained after performing local Step Test

can be varied by setting the properties of the input block's properties in Xcos. The plots of their amplitude versus number of collected samples are also available on the scope windows. The output temperature profile, as read by the sensor, is also plotted. The data acquired in the process is stored on the local drive and is available to the user for further calculations.

In the `ramp_test.xcos` file, open the heater block's parameters to give a ramp input to the system with some value for slope. For this experiment, we have chosen slope = 0.1. Double click on the ramp input block labled as `Heater input`. Change the following values in the respective fields: slope = 0.1, start time = 200, initial output = 20. Keep the fan constant at 100.

The ramp test data file will be saved in `Ramp_Test` folder. The name of the file will be the date and time at which the experiment was conducted. A sample data file is provided in the same folder. The sample data file is named as `ramp-data-local.txt` and `ramp-data-virtual.txt`. Refer to the one depending on wheather you are performing a local or a virtual experiment. Referring to the data file thus obtained as shown in table 5.1, the first column in this table denotes samples. The second column in this table denotes heater in percentage. It starts at 30 and increases with a step size of 10 units. The third column denotes the fan in percentage. It has been held constant at 50 percent. The fourth column refers to the value of temperature. The fifth column denotes time stamp. The virtual data file will have four time stamp columns apart from first 3 columns. These four time stamp columns are client departure, server arrival, server departure and client arrival. These can be used for advanced control algorithms. These additional time stamps exist in virtual mode because of the presense of network delay.

5.1 Conducting Ramp Test on SBHS locally

The detailed procedure to perform a local experiment is explained in Chapter2. A summary of the same is provided in section 2.3 It is same for this section with following changes.

1. Step1: The working directory is `Ramp_test`
2. Step2: Same
3. Step3: Same
4. Step4: Same
5. Step5: Load ramp test function by executing command
   `exec<space>ramp_test.sci`
6. Step6: Load Xcos code for ramp test using the command
   `exec<space>ramp_test.xcos`
7. Step7: Same

5.2 Conducting Ramp Test on SBHS, virtually

The detailed procedure to perform a local experiment is explained in Chapter3. A summary of the same is provided in section 3.5. It is same for this section with following changes.

5.3 Identifying First Order Transfer Function

In this section we shall determine the first order transer function model using the data obtained after performing step test experiment. Please note that this procedure is common for data obtained using both local and virtual experiments.

Identification of the transfer function of a system is important as it helps us to represent the physical system mathematically. Once the transfer function is obtained, one can acquire the response of the system for various inputs without

actually applying them to the system. Consider the standard first order transfer function given below

$$G(s) = \frac{C(s)}{R(s)} \tag{5.1}$$

$$G(s) = \frac{K}{\tau s + 1} \tag{5.2}$$

Combining the previous two equations, we get

$$C(s) = K\left\{\frac{R(s)}{\tau s + 1}\right\} \tag{5.3}$$

Let us consider the case of giving a ramp input to this first order system. The Laplace transform of a ramp function with slope = υ is $\frac{\upsilon}{s^2}$. Substituting $R(s) = \frac{\upsilon}{s^2}$ in equation 5.3, we obtain

$$C(s) = \frac{K}{\tau s + 1}\frac{\upsilon}{s^2} \tag{5.4}$$

$$= \frac{A}{s} + \frac{B}{s^2} + \frac{C}{\tau s + 1} \tag{5.5}$$

Solving $C(s)$ using Heaviside expansion approach, we get

$$C(s) = K\upsilon\left\{\frac{1}{s^2} - \frac{\tau}{s} + \frac{\tau^2}{\tau s + 1}\right\} \tag{5.6}$$

Taking the Inverse Laplace transform of the above equation, we get

$$c(t) = K\upsilon\left\{t - \tau + \tau e^{\frac{-t}{\tau}}\right\} \tag{5.7}$$

The difference between the reference and output signal is the error signal $e(t)$. Therefore,

$$e(t) = r(t) - c(t) \tag{5.8}$$

$$e(t) = K\upsilon t - K\upsilon t + K\upsilon\tau - K\upsilon\tau e^{\frac{-t}{\tau}} \tag{5.9}$$

$$e(t) = K\upsilon\tau(1 - e^{\frac{-t}{\tau}}) \tag{5.10}$$

Normalizing equation 5.10 for $t >> \tau$, we get

$$e(t) = \tau \tag{5.11}$$

This means that the error in following the ramp input is equal to τ for large value of t [**?**]. Hence, smaller the time constant τ, smaller the steady state error.

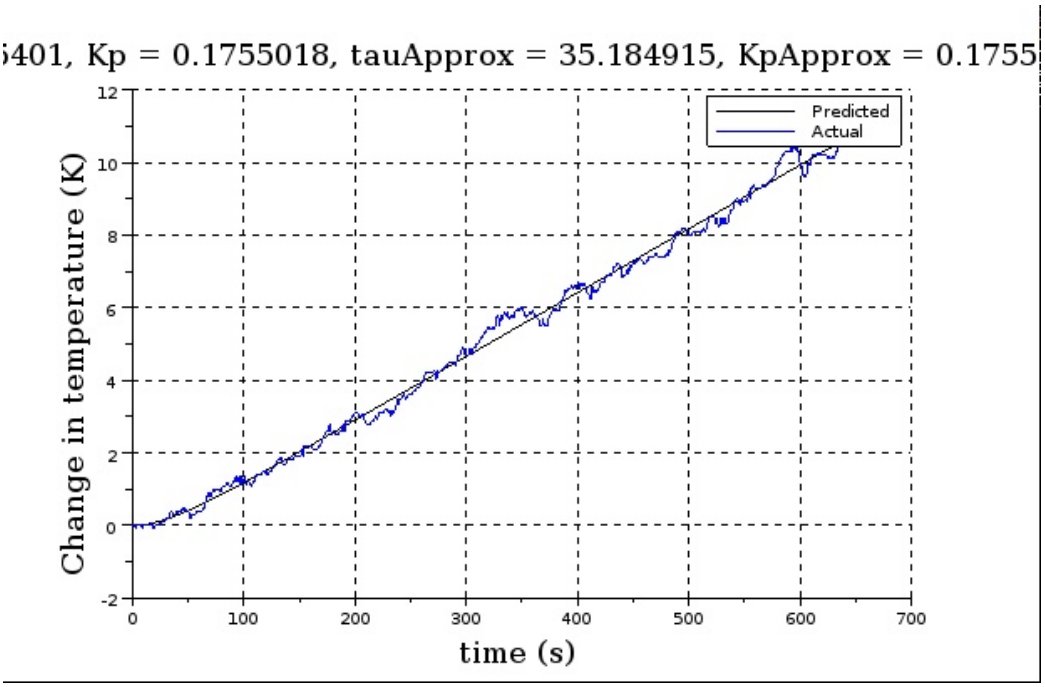

Figure 5.3: Output of the Scilab code`ramp_virtual.sce`

5.3.1 Procedure

1. Download the Analysis folder from the sbhs website. It will be available under `downloads` section. The download will be in zip format. Extrat the downloaded zip file. You will get a folder `Analysis`.
2. Open the `Analysis` folder and then locate and open the folder `Ramp_Analysis`.
3. Copy the ramp test data file to this folder.
4. Change the Scilab working directory to `Ramp_Analysis`
5. Open the file `ramp_virtual.sce` in scilab editor and enter the name of the data file (with extention) in the `filename` field.
6. Save and run this code and obtain the plot as shown in figure 5.3.

This code uses the routines `label.sci` and `costf_1.sci`

The plot thus obtained is reasonably good. See the Scilab plot to get the values of τ and K. The figure 5.3 shows a screen shot of the same. We obtain τ = enterValue, K = enterValue. The transfer function obtained here is at the operating point of enterValue percentage of heat. If the experiment is repeated at a different operating point, the transfer function obtained will be different. The gain will correspondingly be more at a higher operating point. This means that the plant is faster at higher temperature. Thus the transfer function of the plant varies with the operating point. Let the transfer function we obtain in this experiment be denoted as G_s. CHECK THE VALUES FIRST. We obtain

$$G_s(s) = \frac{0.23}{58.64s + 1} \tag{5.12}$$

5.4 Discussion

We summarize our findings now. The experiment has been performed by varying the heater current and keeping the fan speed constant. However, the user is encouraged to experiment using different combinations of fan speed and heater current. Negative ramp can also be used to make the experiment more informative. It is not necessary to keep a particular input constant. For example, you can try giving a step input to the disturbance signal, i.e., the fan input. The system can also be treated as a second order system. This consideration is necessary as it increases the accuracy of the acquired transfer function [**?**].

The necessary codes are listed in the section 5.5.

5.5 Scilab Code

Scilab Code 5.1 `ramp_test.sci`

```
1 mode(0)
2 function temp = ramp_test(heat,fan)
3     temp = comm(heat,fan);
4
5     plotting([heat fan temp],[0 0 20 0],[100 100 50
          1000])
6
7     m=m+1;
8 endfunction
```

Scilab Code 5.2 `label.sci`

```
mode(-1);
// Updated (9-12-06), written by Inderpreet Arora
// Input arguments: title, xlabel, ylabel and their
    font sizes

function label(tname,tfont,labelx,labely,xyfont)
a = get("current_axes")
xtitle(tname,labelx,labely)
xgrid
t = a.title;
t.font_size = tfont; // Title font size
t.font_style = 2; // Title font style
t.text = tname;

u = a.x_label;
u.font_size = xyfont; // Label font size
u.font_style = 2; // Label font style

v = a.y_label;
v.font_size = xyfont; // Label font size
v.font_style = 2; // Label font style

// a.label_font_size = 3;

endfunction;
```

Scilab Code 5.3 `cost.sci`

```
function f = func_1(x)
  k = x(1);
  tau = x(2);
  y_prediction = k*(t + tau*(exp(-t/tau) - 1));
  f = (norm(y - y_prediction,2))^2;
endfunction
```

```
7
8  function [f,g,ind1] = cost(x,ind1)
9    k = x(1);
10   tau = x(2);
11   y_prediction = k*(t + tau*(exp(-t/tau) - 1));
12   f = (norm(y - y_prediction,2))^2;
13   g = numdiff(func_1,x);
14 endfunction
```

Scilab Code 5.4 cost_approx.sci

```
1  function f = func_approx(x)
2    k = x(1);
3    tau = x(2);
4    y_p_approx = k*(t_approx - tau);
5    f = (norm(y_approx - y_p_approx,2))^2;
6  endfunction
7
8  function [f,g,ind] = cost_approx(x,ind)
9    k = x(1);
10   tau = x(2);
11   y_p_approx = k*(t_approx - tau);
12   f = (norm(y_approx - y_p_approx,2))^2;
13   g = numdiff(func_approx,x);
14 endfunction
```

Scilab Code 5.5 ramptest.sci

```
1  function [stop] = ramptest(heat,fan)
2
3      [stop,temp] = comm(heat,fan);// Never edit this
           line
4      plotting([heat fan temp]);
5
6  endfunction
```

Scilab Code 5.6 ramptest.sce

```
mode(0)
global fdfh fdt fncr fncw m err_count y limits
    sampling_time m

// ************************
sampling_time=1;     // In seconds. Fractions are allowed
// ************************//
exec ("ramptest.sci");

ok = init();

    if ok~= []   // open xcos only if communication is
        through (ie reply has come from server)
        xcos('ramptest.xcos');
      else
      disp("NO NETWORK CONNECTION!");
      return
end
```

Scilab Code 5.7 `ramp_virtual.sce`

```
mode(-1);

// filename = "20 Apr2012_15_10_35.txt";    // complete
    path of the saved data file
filename ="ramp-data-local.txt";
slope = 0.1; // change this to the slope that you have
     used in the experiment
ind1=3;
// Ramp Analysis
exec('cost_approx.sci');
exec('cost.sci');
exec('label.sci');

data = fscanfMat(filename);
time = data(:, 5);
heater = int(data(:, 2));
fan = int(data(:, 3));
```

```
16 temp = data(:, 4);
17
18
19 len = length(heater);
20 heaters1 = [heater(1); heater(1:$-1)];
21 del_heat = abs(heater - heaters1);
22 ind = find(del_heat >.5);
23
24 t = time(ind(2):ind($-1));
25 t=t/1000
26 H = heater(ind(2):ind($-1));
27 T = temp(ind(2):ind($-1));
28
29 t = t - t(1);
30 T = T - T(1);
31
32 y = T;
33 x0 = [.5 100]
34 global('y','t');
35
36 [f, xopt] = optim(cost,x0);
37 kp = xopt(1)/slope
38 tau = xopt(2)
39
40 len = length(t);
41 halfway = ceil(len/2);
42
43 t_approx = t(halfway:len);
44 y_approx = y(halfway:len);
45 global('y_approx','t_approx');
46
47 [f_approx,xopt_approx] = optim(cost_approx,x0);
48 kp_approx = xopt_approx(1)/slope;
49 tau_approx = xopt_approx(2);
50
51 // Display and Plot
52 disp('kp = ');
53 disp(kp);
```

```
54 disp('tau = ');
55 disp(tau);
56 disp('kp_approx = ');
57 disp(kp_approx);
58 disp('tau_approx = ');
59 disp(tau_approx);
60
61 y_p = kp*slope*(t + tau*(exp(-t/tau) - 1));
62 y_p_approx = kp_approx*slope*(t_approx - tau_approx);
63 y_p_approx = y_p_approx';
64 plot2d(t,[y_p,T]);
65 label('Showing First Order Model and Experimental
     Results for kp and tau',4,'Time (s)','Change in
     Temperature (Predicted,Actual)',4);
66 legend(['Predicted';'Actual']);
```