

Documentation for Single Board Heater System

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



IIT Bombay
April 10, 2012

Contents

List of Scilab Code	7
1 Block diagram explanation of Single Board Heater System	10
1.1 Microcontroller	10
1.1.1 PWM for heat and speed control	11
1.1.2 Analog to Digital conversion	13
1.2 Instrumentation amplifier	13
1.3 Communication	14
1.3.1 Serial port communication	15
1.3.2 Using USB for Communication	15
1.4 Display and Resetting the setup	17
2 Using Scilab with Single Board Heater System	19
2.1 Accessing Single Board Heater system on a Windows System	20
2.1.1 Installing necessary driver and COM Port Settings	20
2.1.2 Configuring Scilab	21
2.2 Accessing Single Board Heater system on a Linux System	22
3 Using Single Board Heater System, Virtually!	29
3.1 Introduction to Virtual Labs at IIT Bombay	29
3.2 Evolution of SBHS virtual labs	30
3.3 Current Architecture	33
3.3.1 Hardware	33
3.3.2 Software	34
3.3.3 Other Implementation Issues	39
3.3.4 Support	39
3.4 Conducting experiments using the Virtual lab	40

4 Identification of transfer function of a Single Board Heater System through step response experiments	44
4.1 Step by step procedure to perform Step Test	44
4.2 Determination of First order transfer function	47
4.3 Determination of second order transfer function	50
4.4 Discussion	52
4.5 Conducting Step Test on SBHS, virtually	52
4.6 Scilab Code	54
5 Identification of transfer function of a Single Board Heater System through Ramp response experiments	65
5.1 About this Experiment	65
5.2 Theory	65
5.3 Step by step procedure to perform Ramp Test	67
5.4 Ramp Analysis	69
5.5 Discussion	69
5.6 Conducting Ramp Test on SBHS, virtually	70
5.7 Scilab Code	70
6 Frequency Response Analysis of a Single Board Heater System by the application of Sine Wave	77
6.1 Theory	77
6.2 Step by step procedure to perform Sine Test	80
6.3 Conducting Sine Test on SBHS, virtually	87
6.4 Scilab Code	87
7 Controlling Single Board Heater System by PID controller	96
7.1 Theory	96
7.1.1 Proportional Control Action	97
7.1.2 Integral Control Action	98
7.1.3 Derivative Control Action	98
7.2 Ziegler-Nichols Rule for Tuning PID Controllers	99
7.2.1 First Method	100
7.2.2 Second Method	101
7.3 Implementing PI controller using Trapezoidal Approximation	104
7.3.1 Implementing PI controller using Trapezoidal Approximation on SBHS, virtually	105

7.4	Implementing PI controller using Backward Difference Approximation	106
7.4.1	Implementing PI controller using Backward Difference Approximation on SBHS, virtually	108
7.5	Implementing PI controller using Forward Difference Approximation	108
7.5.1	Implementing PI controller using forward Difference Approximation on SBHS, virtually	109
7.6	Implementing PID controller using Backward difference approximation	110
7.6.1	Implementing PID controller using backward Difference Approximation on SBHS, virtually	112
7.7	Implementing PID controller using trapezoidal approximation for integral mode and Backward difference approximation for the derivative mode	112
7.7.1	Implementing PID controller using trapezoidal approximation for integral mode and Backward difference approximation for the derivative mode on SBHS, virtually	114
7.8	Implementing PID controller with filtering using Backward difference approximation	115
7.8.1	Implementing PID controller with filtering using Backward difference approximation on SBHS, virtually	117
7.9	Scilab Code	117
7.9.1	Scilab code for serial communication	117
7.9.2	Scilab code for PI controller	118
7.9.3	Scilab code for PID controller	122
8	Implementing ‘Two Degrees of Freedom’Controller for First order systems on a Single Board Heater System	130
8.1	Theory	130
8.2	Designing 2-DOF controller using pole placement control approach	134
8.3	Step by step procedure to design and implement a 2-DOF controller	137
8.3.1	Implementing 2dof controller on SBHS, virtually	140
8.4	Scilab Code	140
9	Implementing Internal Model Controller for first order systems on a Single Board Heater System	165
9.1	IMC Design for Single Board Heater System	165

9.2	Step for designing IMC for stable plant	167
9.3	Experimental Results	169
9.3.1	Implementing IMC controller on SBHS, virtually	171
9.4	Scilab Code	171
10	Design and Implementation of Self Tuning PI and PID Controllers on Single Board Heater System	176
10.1	Introduction	176
10.1.1	Ojective	176
10.1.2	Apparatus	177
10.2	Theory	177
10.2.1	Why a Self Tuning Controller?	177
10.2.2	The Approach Followed	179
10.2.3	Direct synthesis	179
10.3	Ziegler Nichols Tuning	181
10.4	Step Test Experiments and Parmeter Estimation	182
10.4.1	Step Test Experiments	182
10.4.2	Conventional Controller Design	185
10.4.3	Self Tuning Controller Design	186
10.5	Implementation	188
10.5.1	PI Controller	188
10.5.2	PID Controller	189
10.5.3	Self Tuning Controller	191
10.6	Set Point Tracking	192
10.6.1	PI Controller designed by Direct Synthesis	193
10.6.2	PI Controller using Ziegler Nichols Tuning	195
10.6.3	PID Controller using Ziegler Nichols Tuning	198
10.6.4	Conclusion	199
10.7	Disturbance Rejection	200
10.7.1	PI Controller designed by Direct Synthesis	200
10.7.2	PI Controller using Ziegler Nichols Tuning	204
10.7.3	PID Controller using Ziegler Nichols Tuning	207
10.7.4	Conclusion	209
10.8	Reproducing the Results	209
10.8.1	Implementing Self Tuning controller on SBHS, virtually	210
10.8.2	Serial Communication	210
10.8.3	Conventional Controller, local	211
10.8.4	Fan Disturbance in PI Controller	211

10.8.5 Self Tuning Controller, local	218
10.8.6 Conventional Controller, virtual	224
10.8.7 Fan Disturbance in PI Controller	224
10.8.8 Self Tuning Controller, local	232
11 Model Predictive Control in Single Board Heater System using SCILAB	242
11.1 Objective	242
11.2 Single Board Heater System	242
11.3 Model Predictive Control	243
11.4 Implementing MPC	244
11.5 Working of codes	244
11.6 Procedure to implement MPC on SBHS	244
11.7 Description of xcos	245
11.8 Code for MPC (<i>mpc_run.sci</i>)	246
11.9 Other codes used	249
11.10 Experiments conducted to implement MPC	250
11.11 Sample run to implement MPC	251
11.12 Positive Step Change to Set Point and Fan	251
11.13 Negative Step Change to Set Point and Fan	254
11.14 Effect of Tuning parameters: Weighting factors, We and Wu	257
11.15 For same factor of We and Wu	258
11.15.1 Positive Step Change and (We, Wu)=(1,1) (Expt 1.1)	258
11.15.2 Positive Step Change and (We, Wu)=(10,10) (Expt 1.2)	261
11.15.3 Positive Step Change and (We, Wu)=(40,40) (Expt 1.3)	264
11.15.4 Negative Step Change and (We, Wu)=(1,1) (Expt 2.1)	266
11.15.5 Negative Step Change and (We, Wu)=(10,10) (Expt 2.2)	268
11.15.6 Negative Step Change and (We, Wu)=(40,40) (Expt 2.3)	270
11.16 For different We and Wu factors	271
11.16.1 We =100 and Wu = 2 (Expt 5.1)	272
11.16.2 We =2 and Wu = 100 (Expt 5.2)	274
11.16.3 We =10 and Wu = 100 (Expt 5.3)	276
11.16.4 We =100 and Wu = 10 (Expt 5.4)	278
11.17 Conclusion on Weighting factor experiments	280
11.18 Effect of Control Horizon Parameter, q	280
11.19 For positive step change in Set point and Fan speed	282
11.19.1 For q =2 (Expt 3.1)	282
11.19.2 For q =3 (Expt 3.2)	284

11.19.3 For $q = 4$ (Expt 3.3)	286
11.20 For negative step change in Set point and Fan speed	288
11.20.1 For $q = 2$ (Expt 4.1)	288
11.20.2 For $q = 3$ (Expt 4.2)	290
11.20.3 For $q = 4$ (Expt 4.3)	292
11.21 Conclusion on the effect of Control Horizon parameter	293
11.22 Implementing Model Predictive controller on SBHS, locally . . .	294
11.23 Conclusion for MPC project	294
11.24 Acknowledgement	295
11.25 Appendix	296
11.26 Appendix 1: General Information on Experiments for this Project	296
11.27 Appendix 2: Values of State Space matrices	296
11.28 Appendix 3: Attachments and Contact Information	298
11.29 Scilab Code	298

List of Scilab Code

4.1	label.sci	54
4.2	costf_1.sci	54
4.3	firstorder.sce	55
4.4	costf_2.sci	56
4.5	order_2_heater.sci	57
4.6	secondorder.sce	57
4.7	ser_init.sce	58
4.8	step_test.sci	59
4.9	stepc.sce	60
4.10	steptest.sci	61
4.11	firstorder_virtual.sce	62
4.12	secondorder_virtual.sce	63
5.1	ramp_test.sci	70
5.2	label.sci	71
5.3	cost.sci	72
5.4	cost_approx.sci	72
5.5	ramptest.sci	72
5.6	ramptest.sce	74
5.7	ramp_virtual.sce	75
6.1	sine_test.sci	87
6.2	sinetest.sce	88
6.3	sinetest.sci	88
6.4	sine2.sce	90
6.5	lable.sci	91
6.6	bodeplot.sce	92
6.7	labelbode.sci	92
6.8	TFbode.sce	93
6.9	comparison.sce	93

6.10	sine2_virtual.sce	94
7.1	ser_init.sci	117
7.2	pi_ta.sci	118
7.3	pi_bda.sci	119
7.4	pi_fda.sci	121
7.5	pid_bda.sci	122
7.6	pid_ta_bda.sci	123
7.7	pid_filter.sci	125
7.8	pid_bda_virtual.sce	126
7.9	pid_bda_virtual.sci	127
8.1	c2d.sce	140
8.2	2-DOF_para.sce	143
8.3	2dof.sci	144
8.4	cindep.sci	146
8.5	clcoef.sci	147
8.6	clcoef.sci	148
8.7	cosfil_ip.sci	149
8.8	desired.sci	149
8.9	indep.sci	149
8.10	left_prm.sci	150
8.11	makezero.sci	153
8.12	move_sci.sci	154
8.13	polmul.sci	155
8.14	polsize.sci	156
8.15	polsplit3.sci	156
8.16	polyno.sci	157
8.17	pp_im.sci	158
8.18	rowjoin.sci	159
8.19	seshft.sci	160
8.20	t1calc.sci	161
8.21	xdync.sci	162
8.22	zpowk.sci	163
9.1	ser_init.sce	171
9.2	imc.sci	171
9.3	imc_virtual.sce	173
9.4	imc_virtual.sci	174
10.1	ser_init.sce	210
10.2	pi_bda_dist.sci	211

10.3	pi_bda.sci	213
10.4	pid_bda_dist.sci	215
10.5	pid_bda.sci	216
10.6	pi_bda_tuned_dist.sci	218
10.7	pi_bda_tuned.sci	219
10.8	pid_bda_tuned_dist.sci	221
10.9	pid_bda_tuned.sci	223
10.10	pi_bda_dist.sci	224
10.11	pi_bda.sci	226
10.12	pid_bda_dist.sci	228
10.13	pid_bda.sci	230
10.14	pi_bda_tuned_dist.sci	232
10.15	pi_bda_tuned.sci	234
10.16	pid_bda_tuned_dist.sci	237
10.17	pid_bda_tuned.sci	239
11.1	mpc_init.sce	298
11.2	mpc.sci	300
11.3	mpc_init_local.sce	301
11.4	mpc_local.sci	301

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 μ C 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 μ C 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)

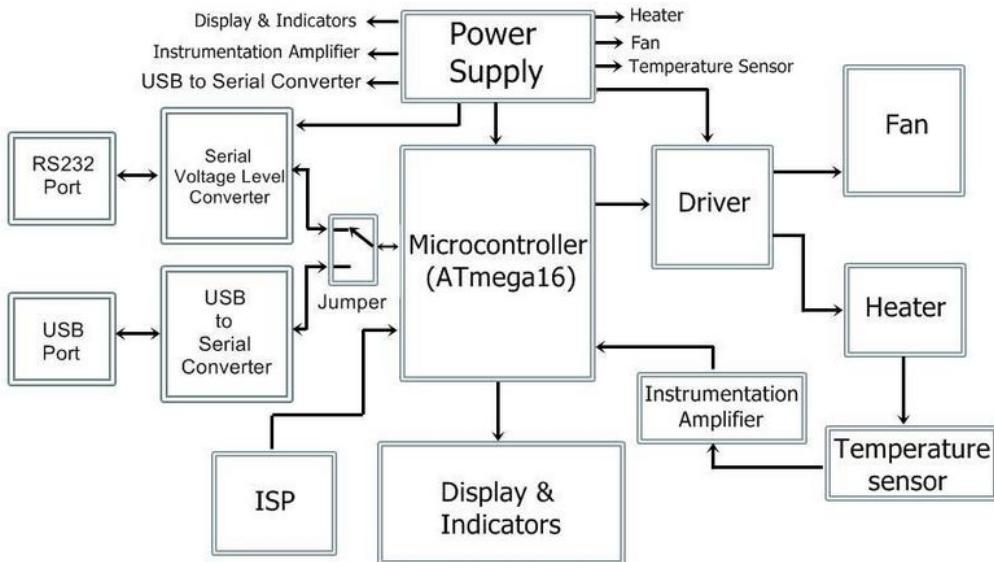


Figure 1.1: Block Diagram

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

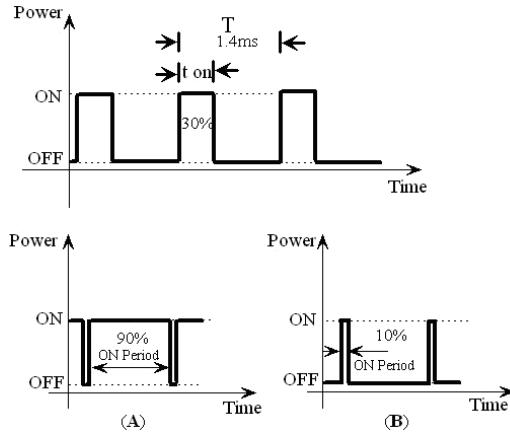


Figure 1.2: Pulse Width Modulation (A): On time is 90% of the total time period, (B): ON time is 10% of total time period

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: $1\mu\text{A}/\text{K}$
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 in to 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 $0\text{-Vcc}(0$ to Vcc) 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 does 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

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

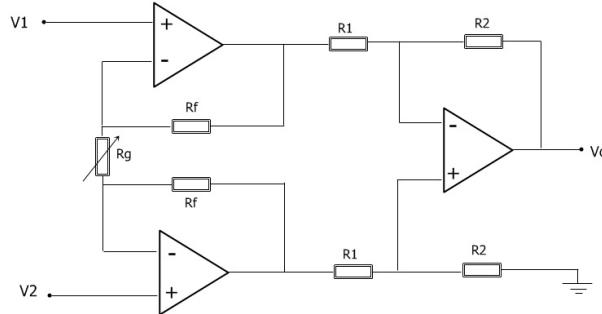


Figure 1.3: 3 Op-Amp Instrumentation Amplifier

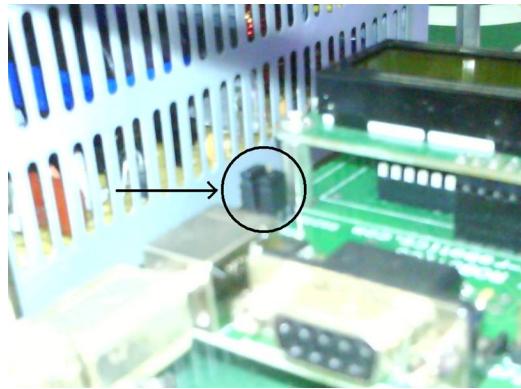


Figure 1.4: Jumper arrangement

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.

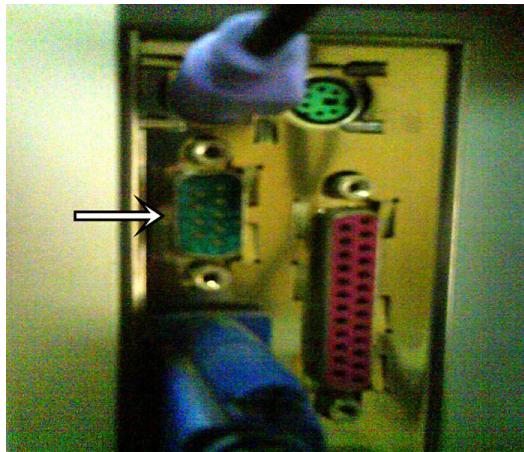


Figure 1.6: Serial port



Figure 1.7: USB communication

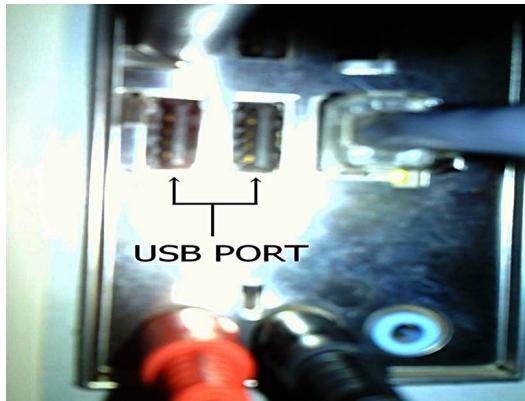


Figure 1.8: USB PORT



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. Reseting 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

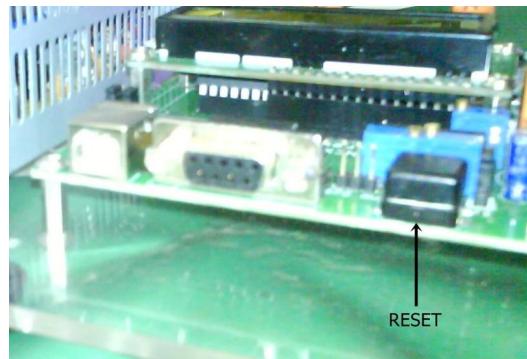


Figure 1.10: Reset

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

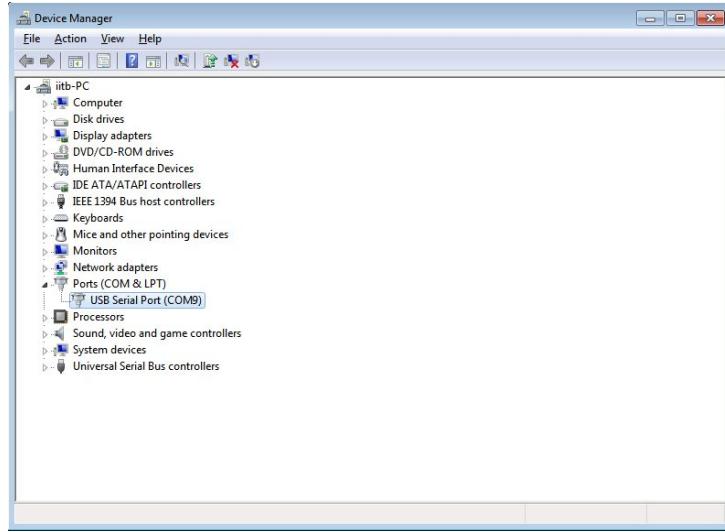


Figure 2.1: Checking Communication Port number

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 if the files are available. Here the directory is changed to the place where the relevant files for performing 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,

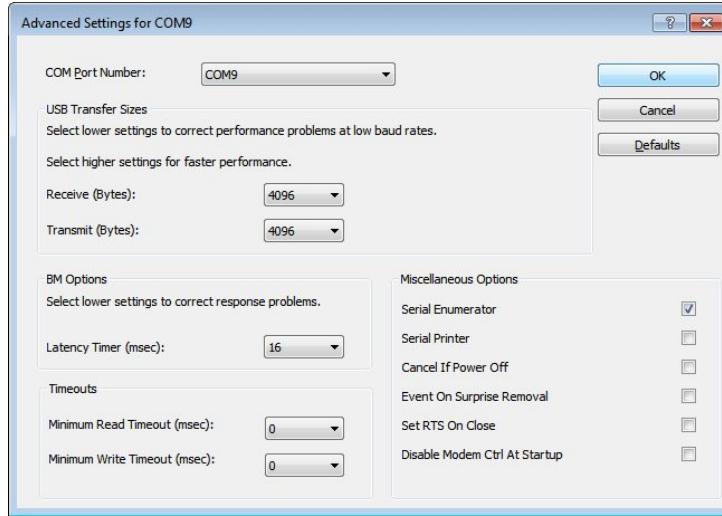


Figure 2.2: Changing Com port number

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:

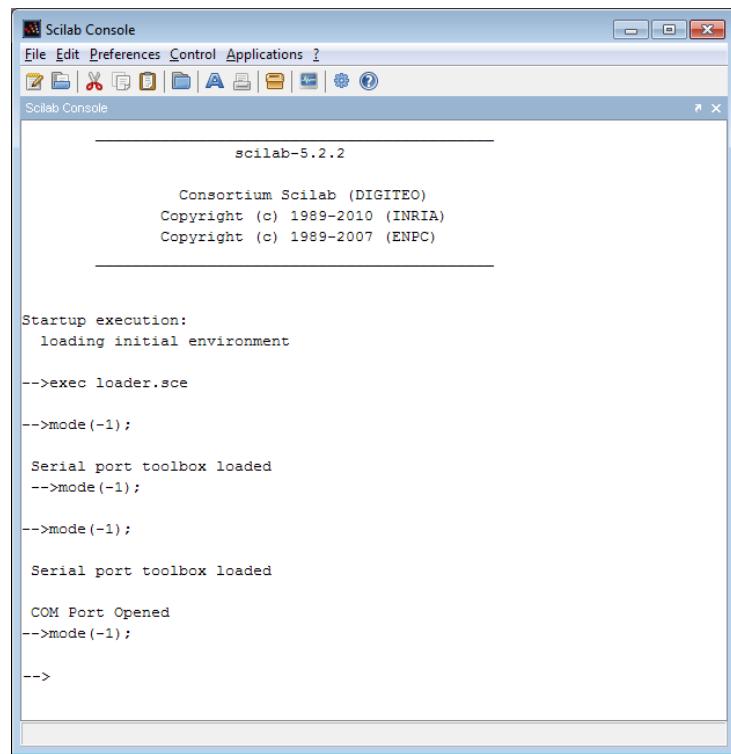
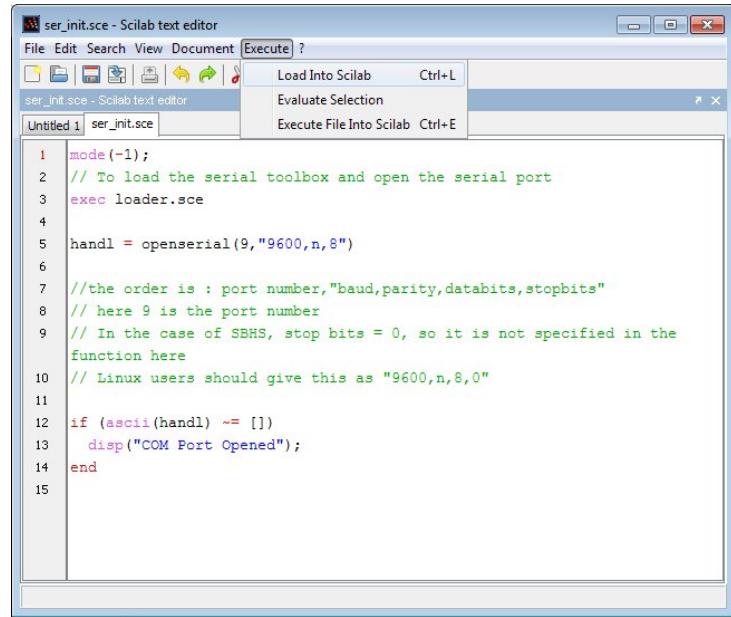


Figure 2.3: Expected responses seen on the console



The screenshot shows the Scilab text editor window titled "ser_init.sce - Scilab text editor". The menu bar includes File, Edit, Search, View, Document, Execute, and Help. The toolbar has icons for file operations like Open, Save, and Run. The main area contains the following Scilab script:

```
1 mode(-1);
2 // To load the serial toolbox and open the serial port
3 exec loader.sce
4
5 handl = openserial(9,"9600,n,8")
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
12 if (ascii(handl) ~= [])
13   disp("COM Port Opened");
14 end
15
```

Figure 2.4: Executing script files

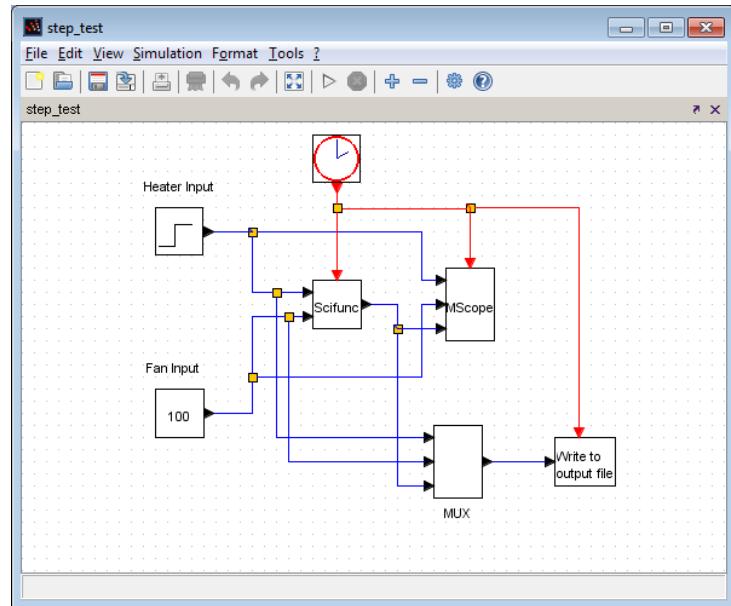


Figure 2.5: Xcos Interface

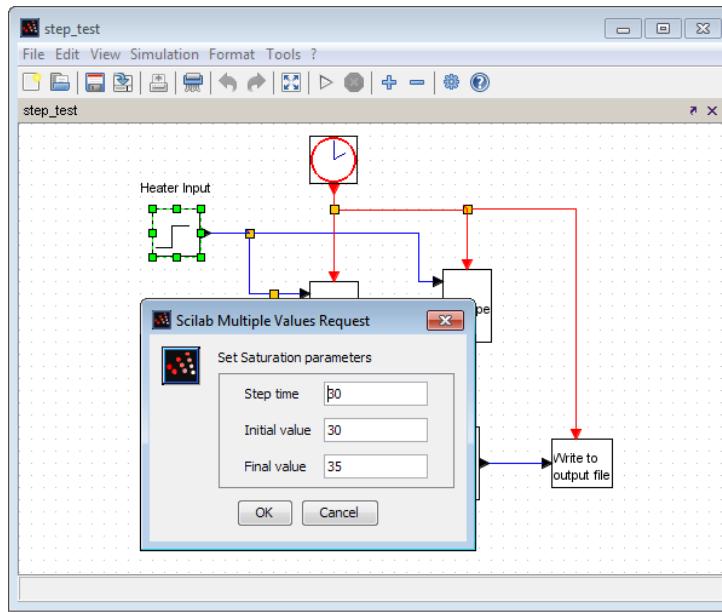


Figure 2.6: Setting Block Parameters

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 `openerserial()` function) in the `ser_init.sce` file with it. The second argument of the `openerserial()` 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

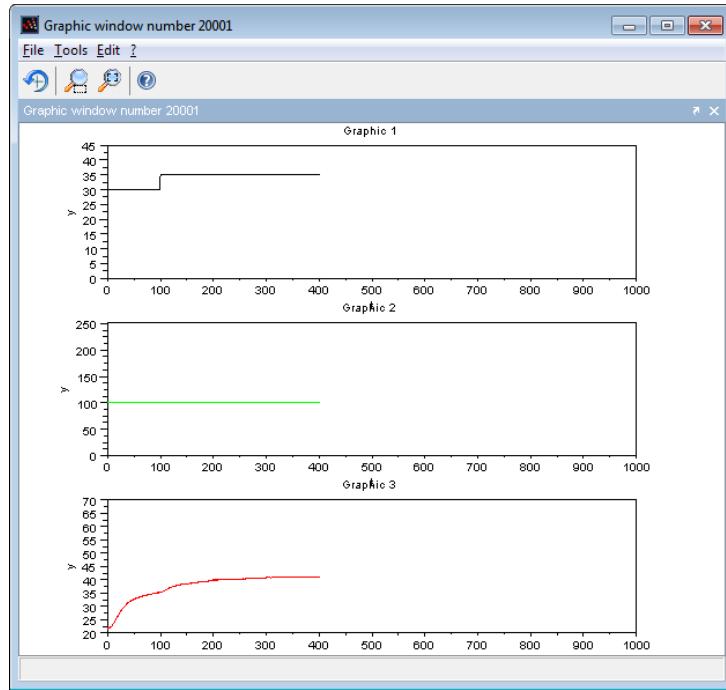


Figure 2.7: Plot obtained after executing step_test.xcos

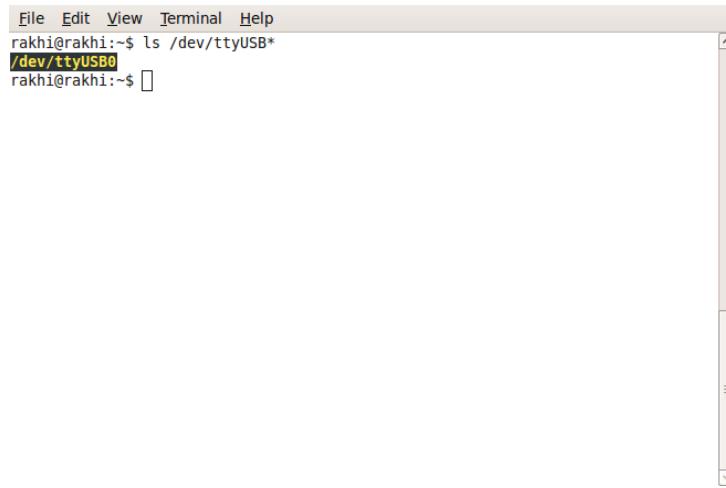
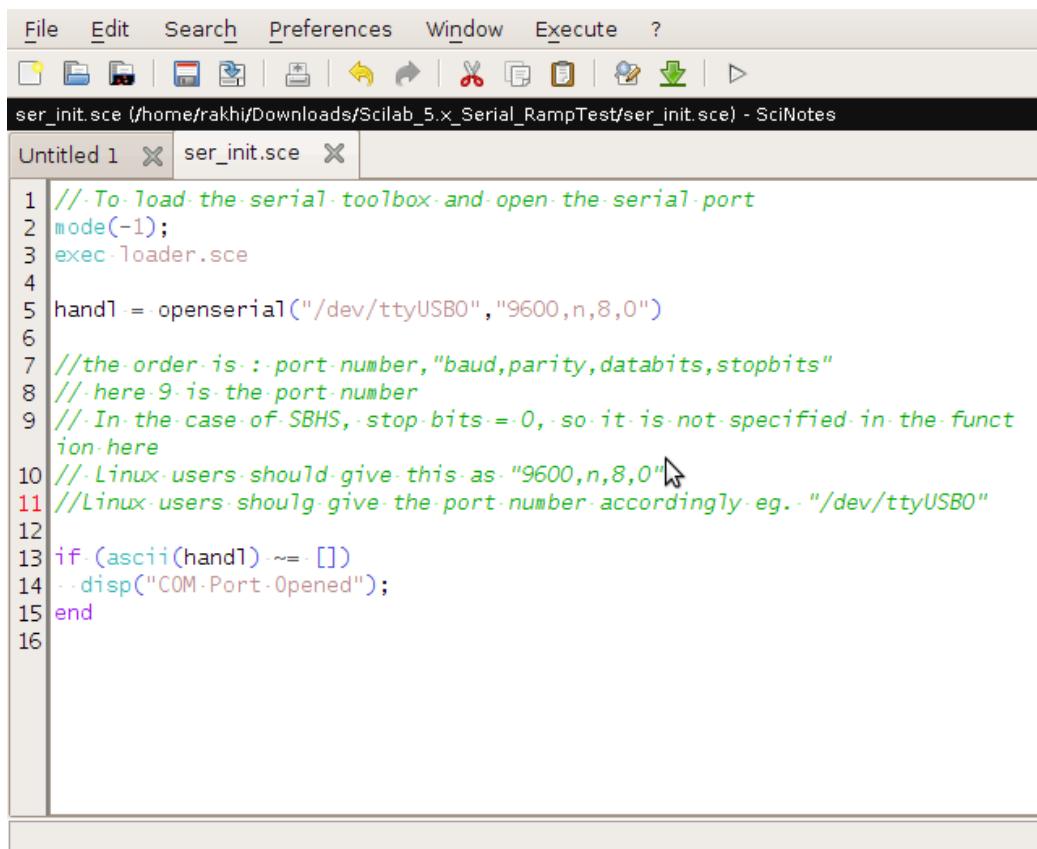


Figure 2.8: Checking the port number in linux (1)



```
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 ser_init.sce
1 // To load the serial toolbox and open the serial port
2 mode(-1);
3 exec-loader.sce
4
5 hand1 = openserial("/dev/ttyUSBO", "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/ttyUSBO"
12
13 if (ascii(hand1) ~= [])
14 disp("COM-Port-Opened");
15 end
16
```

Figure 2.10: Configuring port number and other parameters

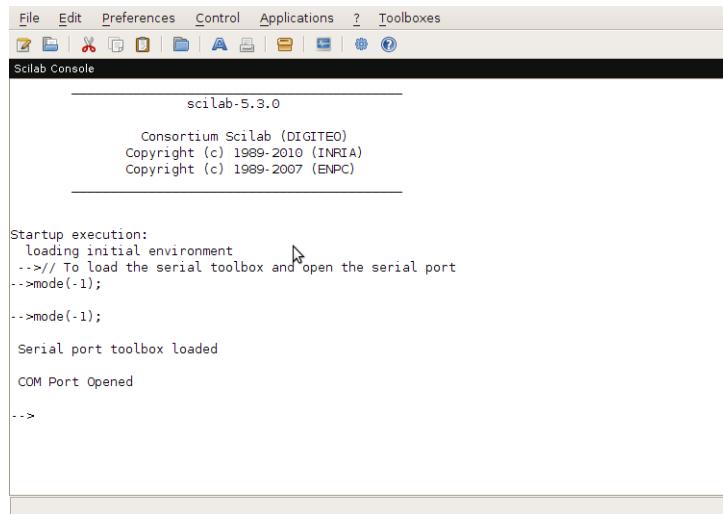


Figure 2.11: Scilab Console Message after Opening Serial Port

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.

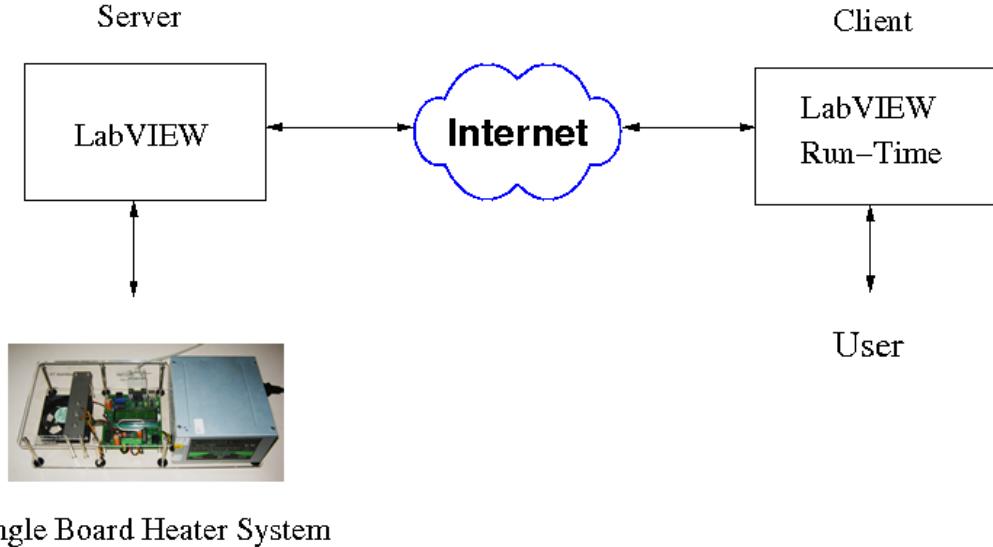
Chapter 3

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: http://www.co-learn.in/web_sbhs/. A set of 15 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 15 SBHS connected with an hours slot for 24 hrs a day, we have 360 one hour slots a day. This means that 360 individual users can access the SBHS in a



Single Board Heater System

Figure 3.1: SBHS virtual laboratory with remote access using LabVIEW

day for an hour. This also means that up to 2520 users can use the SBHS for an hour in a week and more than 10,000 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. The client is also suppose to run a java client on his computer. 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 [4], 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

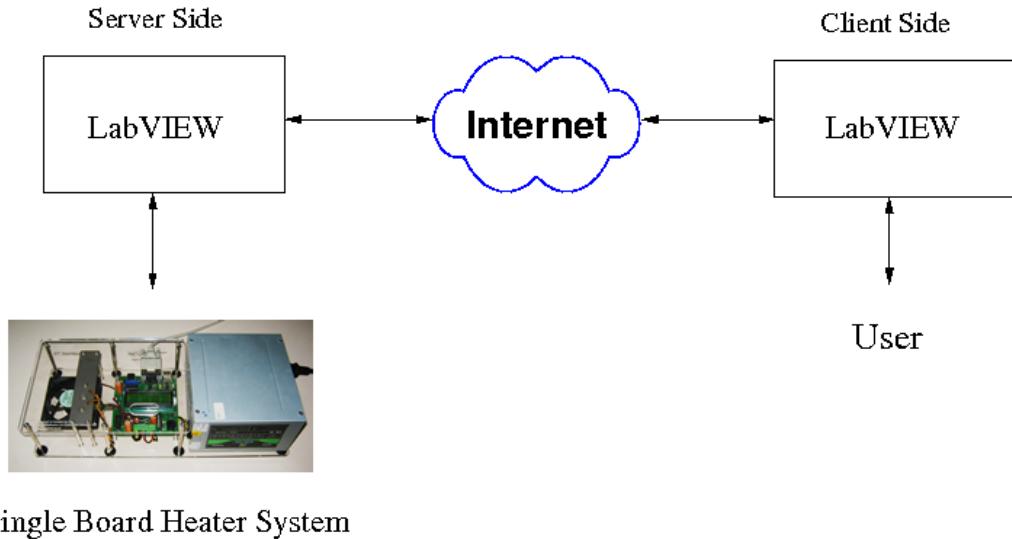


Figure 3.2: SBHS virtual laboratory with remote access and live data sharing using LabVIEW

on the server. The client accesses these algorithm/experiment over the Internet using a web browser by entering appropriate parameters.

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

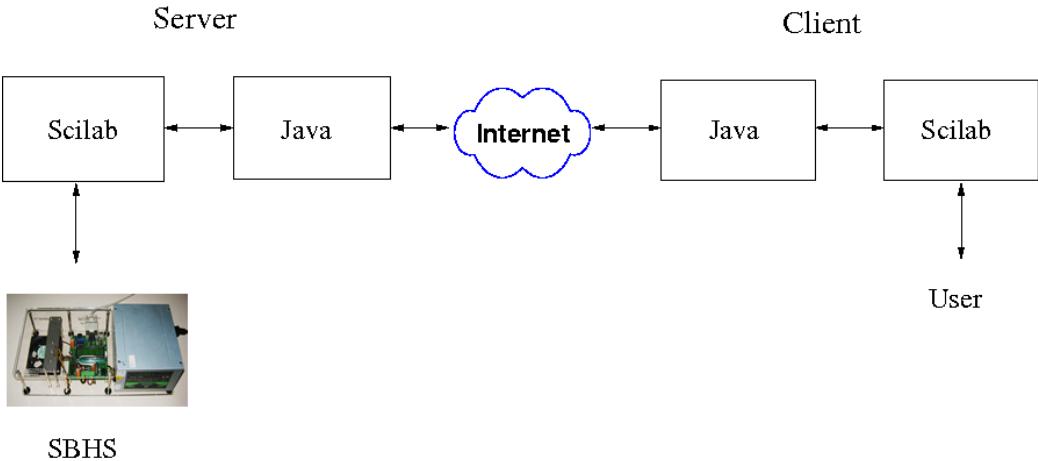


Figure 3.3: SBHS virtual laboratory using open source software

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.

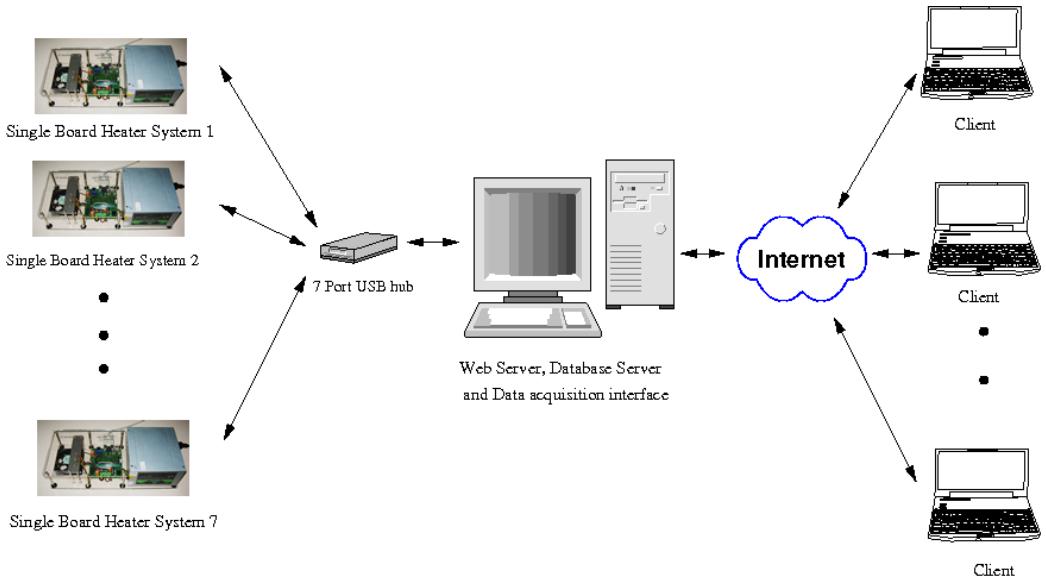


Figure 3.4: Virtual control lab hardware architecture

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 Architecture

3.3.1 Hardware

The architecture of the virtual single-board heater system lab as shown in Figure 3.4 involves 7 single-board heater systems connected to the server via a 7-port USB hub. The server computer is connected to a high speed internetwork and has enough processing capability to host data acquisition, database, and web servers. The internetwork connected client computer needs only the Java runtime engine and Scilab installation.

A similar architecture but replacing Java with python and with 15 units connected to the server via two 7-port hubs has been successfully tested on intranet 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 facility will be extended to all the

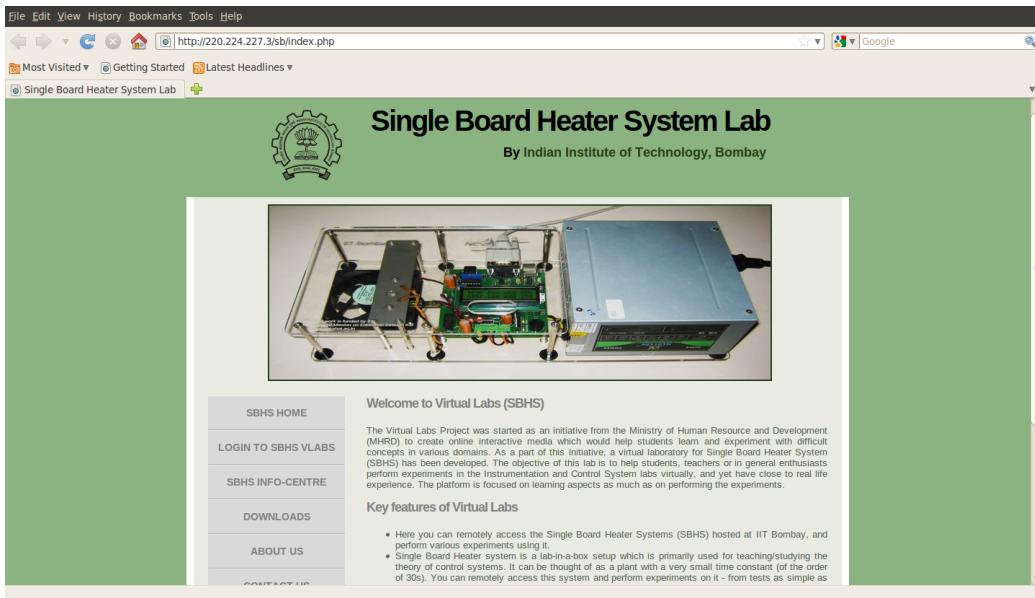


Figure 3.5: Home page of SBHS V Labs

units in future. This gives the user a feel of remote hands-on. Work on this camera facility is in progress.

3.3.2 Software

The current software architecture of this virtual control lab is shown in Figure 3.6. The server computer runs on Ubuntu Linux 10.04 OS. It hosts a LAMP (Linux-Apache-MySQL-PHP) server. The MySQL-based database server has the details of all the registered users, their slot details, authentication keys to allow remote access, etc. It also hosts a PHP based web server shown in Figure 3.5 that has pages for registration, login and slot booking [9]. For communication with the device, Django - a Python based web framework is used on the Server side. On the client end has control algorithms running in Scilab and communicates over the Internet through the python client.

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

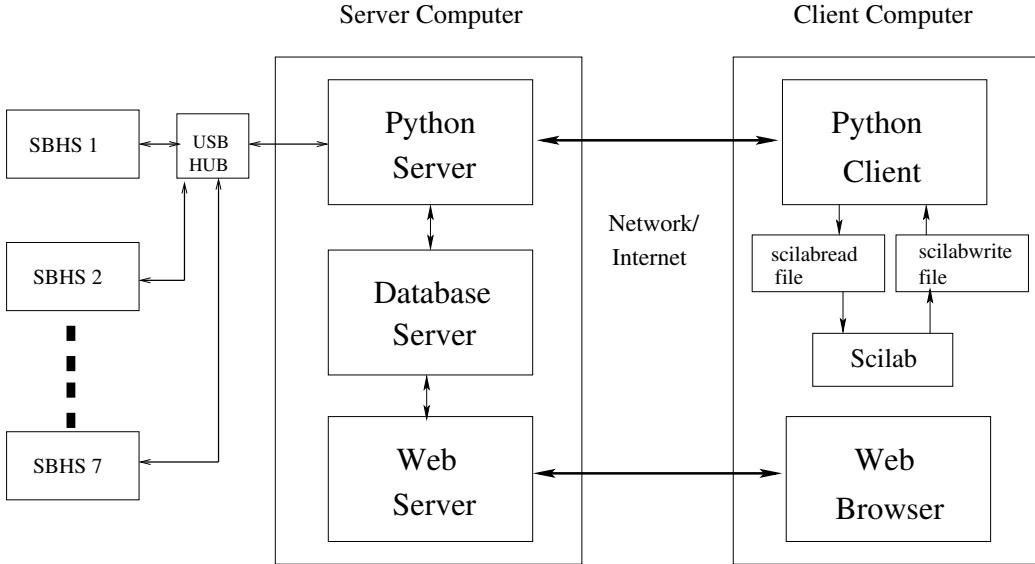


Figure 3.6: Current Architecture of SBHS Virtual Labs

3.3.2.1 Registration

A client willing to perform the experiments needs to register with us by entering the personal details on the Registration page. This sends an account activation link to the client's mailbox. Upon clicking the link, the account gets activated.

3.3.2.2 Slot booking

The client can now login and book slots to perform the experiments. Each slot lasts for an hour with 55 minutes for experimentation and 5 mins for resetting the setup. A client can book up to 2 slots, per day, in advance. Besides this, if the current slot is empty, it can be booked as a free slot. For each slot being booked, the client gets a port number and an access key. The interface depicting this is shown in Figure 3.7.

3.3.2.3 Port number

In order to maintain consistency in performing the experiments, each client is allotted the same setup every time s/he requests for the slot. To facilitate this, each setup has been allotted a machine I.D. For example, if the machine I.D. of the unit

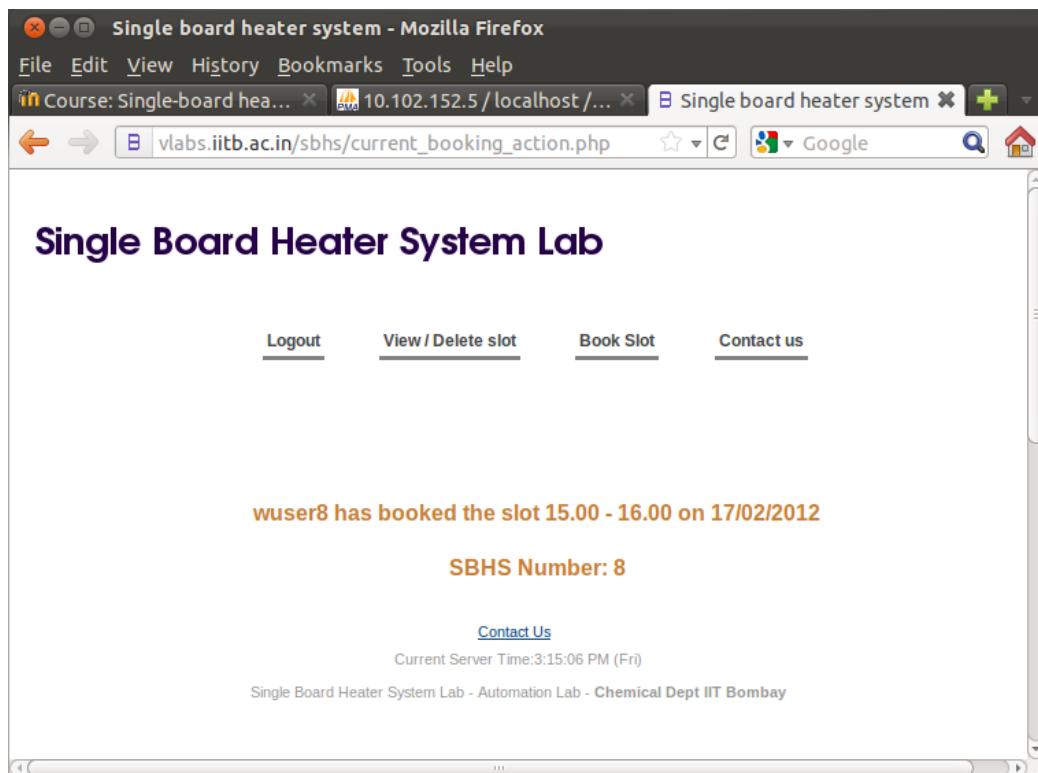


Figure 3.7: Screenshot of Slot Booking Page

is 5, it is displayed as SBHS Number: 5. This eliminates time and effort spent in plant modelling, each time, due to change in the setup.

3.3.2.4 Starting the Python client

On windows, after installing python, it will not be readily recognized by the OS. To make windows interpret python as a command, do the following steps.

1. Right click on My Computer and click on properties
2. Click on the Advanced tab. Now click on the Environment Variables button
In the system variable section, locate and click on “path”.
3. Click the edit button below it. You should see a “Edit system variable” window.
4. In the “Variable value” field, take the cursor to the last. Type a semicolon and type C:\Python27

The experiment folder has a run.bat file, settings.txt file and a sbhsclient.py file. Open the settings.txt file and make the required changes. Beware not to make any unnecessary changes to this file. The example settings are anyways given in the end of this file. Linux users should bypass the terminal proxy settings, if any, by executing the command `export http_proxy=''`. Be informed that this is just a temporary disable of proxy and it should be done everytime you open a fresh terminal. Windows users simply double click on the run.bat file. It will open up a command prompt. Linux users instead of executing the run.bat file will type the following command on the terminal `python sbhsclient.py`. This activity will ask you for the username and password. If the settings are not proper in the settings.txt file, you will get a `Connection error` message. Contact the system administrator of your network to get the correct values of the parameters. Once you are done typing your username and password, you will get a `Login Successfull` message. In case if you are logging in at a time other than your booked slot, it will tell you that `No slot is found`. Else, it will notify that you have booked a slot and can now run the scilab code. It will also create `scilabread.sce`, `scilabwrite.sce` and a log file in the experiment folder. The log file name will be in the format `date_month_year_hours_mins_seconds.txt`. This unique file name will help you track the log file of your choice in the future.



Figure 3.8: Screenshot of Video Stream

3.3.2.5 Video streaming

At present, the facility to provide video streaming for all users is in progress. The video streaming of the capture display will be something as shown in Figure 3.8. The clients with lesser bandwidth will have an option to refrain from video streaming. This facility will be available for logged in users on the website.

3.3.2.6 Executing the Scilab code

The client needs to download the scilab code from the SBHS home page. This is available under [Downloads](#). Various other experiments are also available on [fossee.in/moodle](#). The client can modify this code to implement his/her own algorithm. S/he should not edit the part of the code which is used for communicating with python. This part is clearly marked in the code. Refer the chapter specific to which experiment you want to run. It will explain you which codes and in what order they should be executed.

3.3.2.7 File handling

This subsection explains the significance of scilabread and scilabwrite files. During the experiment, the client writes heater and fan inputs to the scilabwrite.sce

file. These values are read by the Python client. It writes iteration and client departure time stamp and send it over the network. Python server at the other end reads these values, puts a server arrival time stamp and gives them to the SBHS through its data acquisition interface. After feeding the values, it reads the temperature of the heated plate, puts a server departure time stamp and sends it to the Python client. The Python client puts a client arrival time stamp and then writes all these values i.e. heater, fan and temperature along with the timestamps to the scilabread.sce file. Scilab client reads the latest temperature value and does the calculation of heater and fan inputs in accordance to the logic developed for the experiment. The live data streaming involves heater and fan inputs being sent by the client and temperature response in turn being sent by the server. The timestamp accompanying the data could be used for real-time control of the setup.

3.3.2.8 Concluding the experiment

Once the client is done with the experiment, s/he can use the log file created in the experiment folder for analysis purpose. If you happen to lose the log file, you can download it from the website, after you login, by clicking on Download link.

3.3.3 Other Implementation Issues

3.3.3.1 Mapping of machine id with the USB port number

Whenever a SBHS unit is plugged in to a USB port, the dev id (/dev/ttyUSB*) assigned to it by OS is different. A script is run which creates a table of the mapping between the machine I.D. and the USB dev id. This is done to ensure that the client gets the same machine I.D. i.e. the same SBHS each time.

3.3.3.2 Auto log off problem

In some ISPs, the network gets disconnected if it is inactive for some time. To handle this situation, we are running a shell script which checks for internet connectivity by pingging google periodically. If the network is down then it will try to reconnect.

3.3.4 Support

The SBHS support activities are being funded by National Mission on Education through Information and Communication Technology [3]. The major support ac-

tivities include conducting workshops in several colleges across the country, active discussion through Fossee moodle and spoken tutorials for self-learning.

3.3.4.1 Workshops

The SBHS team has been conducting workshops to popularise the utility of SBHS. The course content is around local and remote access of the SBHS. More than 50 college teachers from several Engineering colleges of the country have attended the workshops and have started using the setup for the relevant courses running in their colleges.

3.3.4.2 Fossee moodle

Fossee moodle is a web interface that allows discussions, post queries, upload files, upload grades, etc. Also, there is a facility to provide e-mail notification whenever there is a post on the course website. Thus, such an interface allows to address common problems of many users at the same time. It also provides a platform for the users at different locations to share their experiences during the experiments. The codes and manuals for about 8 experiments are available at [1].

3.3.4.3 Spoken tutorials

A Spoken tutorial is a screen capture along with the audio that could be used to teach any computer application. Currently, the spoken tutorials for various Free and Open source softwares are available in different Indian languages at [2]. The scripts and tutorials for SBHS are in pipeline.

3.4 Conducting experiments using the Virtual lab

In this section we will see the steps involved in conducting experiments using virtual lab. This section assumes that the necessary files and software are available on the users computer. Files can be downloaded from the v labs home page http://www.co-learn.in/web_sbhs/ as shown in fig 3.5

1. The user goes to the v labs home page and if he is a first time visitor, he registers himself and follows the information provided there, else he directly logs in.
2. After logging in, the user will book a slot.

The screenshot shows a Mozilla Firefox browser window with the title "Course: Single-board heater system - Mozilla Firefox". The URL in the address bar is <http://fossee.in/moodle/course/view.php?id=3>. The page content is as follows:

Single-board heater system

You are logged in as Inderpreet Arora (Logout)

moodle > single-board

Switch role to... Turn editing on

Activities

- Forums
- Resources
- Schedulers

People

- Participants

Search Forums

Go! Advanced search

Administration

Done

Topic outline

- News forum

1 Manuals

- Discussion on manuals
- Configuring through Scilab
- Hardware Details of SBHS
- Schematic Diagram with description
- FTDI CDM USB driver download
- FTDI CDM USB driver installation guide
- Using WINAVR for microprocessor programming

Latest News

Add a new topic...
(No news has been posted yet)

Upcoming Events

There are no upcoming events
Go to calendar...
New Event...

Recent Activity

Activity since Thursday, 5 May 2011, 03:26 PM
Full report of recent

Figure 3.9: Fossee moodle webpage to support SBHS activities

3. In the experiment folder the settings.txt file will be edited as per requirement. The client should take care not to make any unnecessary changes to this file. The example settings are anyways given in the end of this file.
4. The user will now start the python client. On windows OS, double clicking on the run.bat file will open the command prompt. On linux OS, one has to first go to the directory where the experiment files (.sce and .py) are kept. This can be done by typing `cd<space>directory name` on the terminal. Linux users should also bypass the terminal proxy settings, if any, by executing the command `export http proxy=''` on the terminal. Be informed that this is just a temporary disable of terminal proxy and it should be done everytime you open a fresh terminal. Linux user should now type the command `python sbhsclient.py` on the linux terminal.
5. The python client will first do some network checks. If it finds the network to be NOT ok for experimentation, it will put the message "No network connection". Else, if it finds the network to be ok for experimentation, it will communicate with the server and will authenticate the user if he is authorised to access the particular SBHS at the booked time. If the user is not authorised it he/she will get the message "Authentication failed. Please check your username and password". If the user is authenticate but has not booked the slot, he/she will get the message "No valid slot found. Please book a slot before starting the experiment". Also if the settings.txt file is not properly configured, it will give error `Invalid settings in the "settings.txt" file`. If every thing is fine, the client will get the "Login Successfull message." It will also display the log file name and the time left for experimentation.
6. The user now launches scilab, changes the directory to the folder where the necessary .sce, .sci and .xcos files of an experiment of interest resides. The user will execute the scilab code (.sce file) pertaining to the experiment. If the network connection is fine, it will automatically open the corresponding .xcos file. Else it will output a message on the scilab console `No network connection`.
7. The user will run the xcos file. It will open a plot of the various experimental parameters. This process will continue until the experiment is stopped or the simulation time is lapsed. The simulation time can be changed by changing

the `final integration time` parameter available in the `set up` option in the `simulation` menu on the Xcos window.

8. The slot is made to last for 55 minutes. The last 5 minutes of the slot are used to reset the SBHS so that the next user will get the SBHS at a normal operating condition. The python client ceases connection automatically as soon as 55 minutes are over. Please note that there will be no pop up warning and the experiment will be stopped automatically.
9. A log of the experimental data with time stamp is maintained on the client side. It is also available to download using the "Download" link once you login on the sbhs vlabs website.

Chapter 4

Identification of transfer function of a Single Board Heater System through step response experiments

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 the setting the properties of input block's properties in Xcos. The plots of their amplitude versus no. 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.

4.1 Step by step procedure to perform Step Test

The procedure to perform a step test is exactly the same as explained in Chapter 2. In the `step_test.xcos` file, a block named `write to output file` is used to make a log of the data generated during the experiment. Open the block's parameters to change the name of the file generated. Apply a step change of say 5 percent to the heater at operating point of 30 percent of heater after 300 seconds. This means that the block parameters of the step input block will have

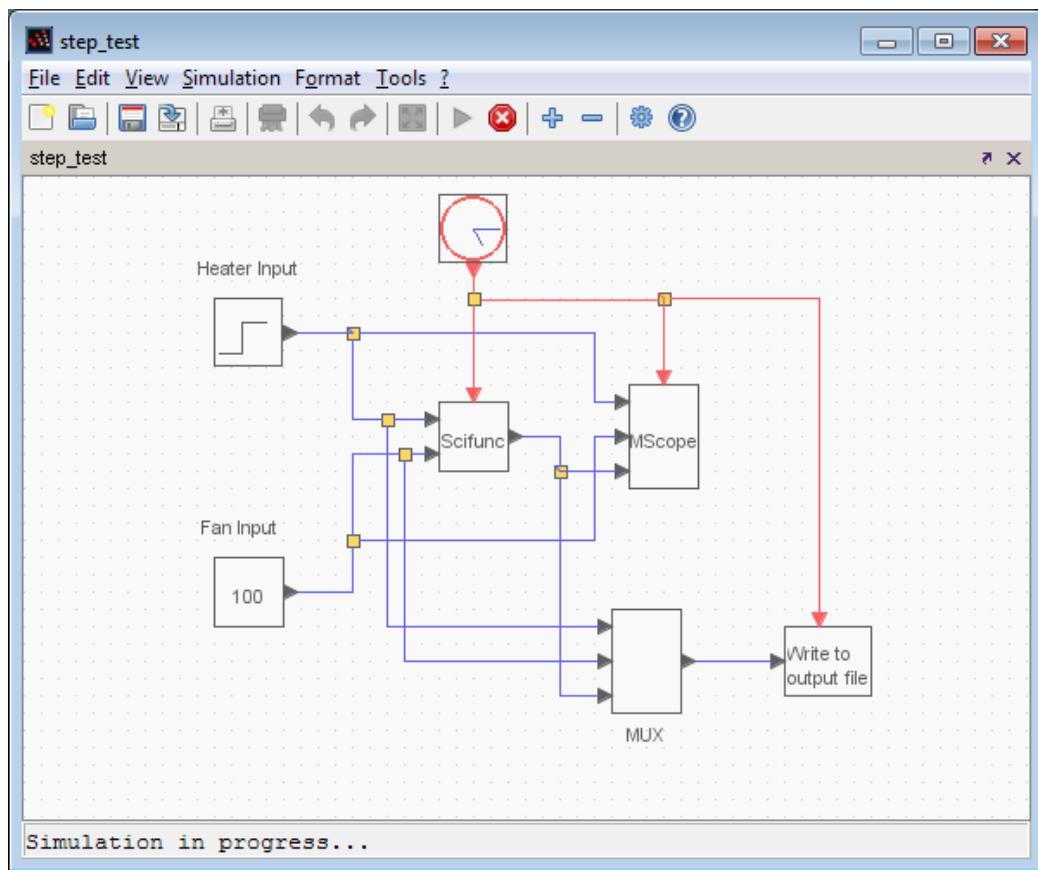


Figure 4.1: Xcos for this experiment

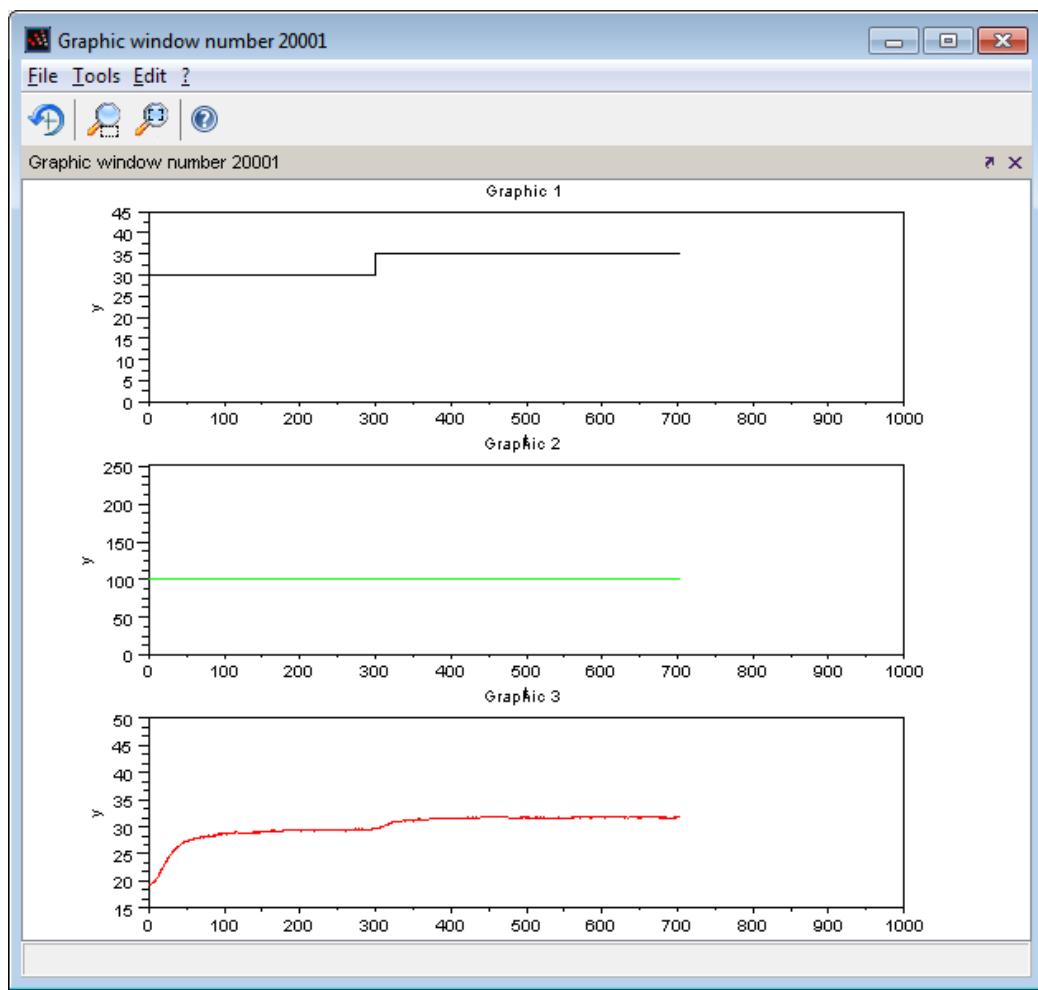


Figure 4.2: Graph shows Heater current , Fan speed and Output Temperature

0.100E+00	0.300E+02	0.100E+03	0.195E+02
0.200E+00	0.300E+02	0.100E+03	0.195E+02
.	.	.	.
0.704E+03	0.350E+02	0.100E+03	0.318E+02
0.704E+03	0.350E+02	0.100E+03	0.318E+02

Table 4.1: Step data obtained after performing the Step Test

Step time = 300, Initial value = 30 and Final value = 35. Keep the fan input constant at 50 percent. Start the experiment and let it continue until you see the temperature almost reaching the steady state. Referring to data file thus obtained as shown in Tabel 4.1, the first column in this table denotes time. The second column in this table denotes heater current. It start at 30 and increases with a step size of 5 units. The third column denotes the fan speed. It has been held constant at 50 percent. The fourth column refers to the value of temperature.

4.2 Determination of First order transfer function

Identification of the transfer function of a system is quite important since 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

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

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

Rewriting the Equation, we get

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

A step is given as an 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

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

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

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

Taking the Inverse Laplace transform of equation 4.5, we get

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

from the above equation it is clear that for $t=0$ the value of $c(t)$ is zero. For $t=\infty$, $c(t)$ approaches unity. Also as the value of ‘t’ becomes equal to τ , the value of $c(t)$ becomes 0.632. The τ is called as the time constant and represents the speed of response of the system. But it should be noted that, the smaller the time constant, the 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

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

to 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 . Suppose that the current changes by a step of size u . Then, we obtain the following relation between the current and the temperature.

$$y(s) = G(s)u(s) \quad (4.8)$$

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

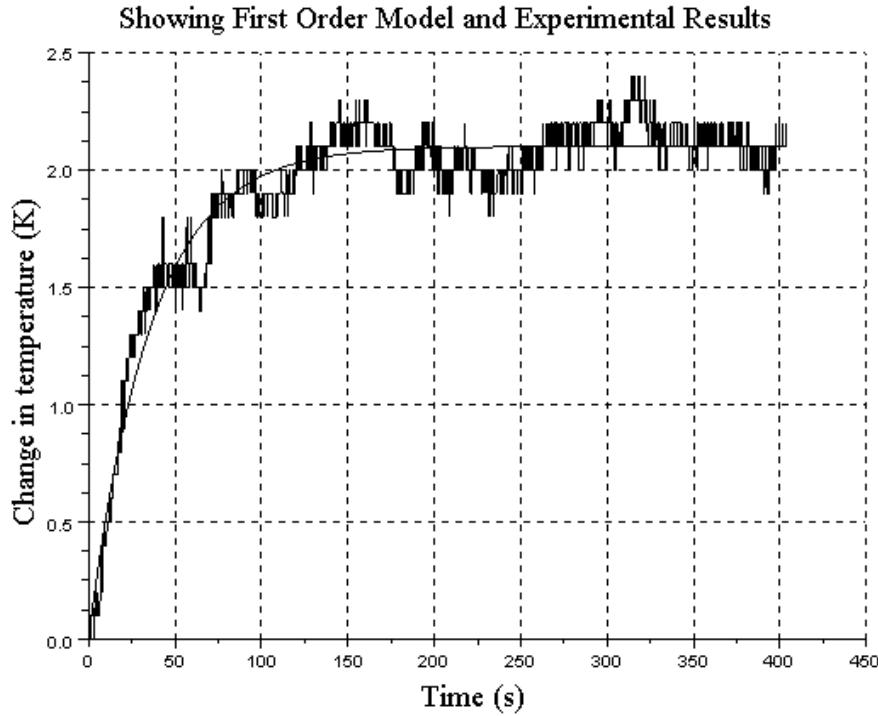


Figure 4.3: Output of the scilab code `firstorder.sce`

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

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

Copy the step test data file to the folder `Kp-tau-order1`. Change the scilab working directory to `Kp-tau-order1` folder under `Step_Analysis` folder. Open the file `firstorder.sce` and put the name of the data file (with extention) 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` and `costf_1.sci`

The plot thus obtained is reasonably good. See the Scilab console to get the values of τ and K . The figure 4.4 shows a screen shot of the same. We obtain $\tau = 35.61087$, $K = 0.4201$. The transfer function obtained here is at the operating point of 30 pwm 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

$$G_s(s) = \frac{0.4201}{35.61s + 1} \quad (4.11)$$

4.3 Determination of second order transfer function

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

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

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

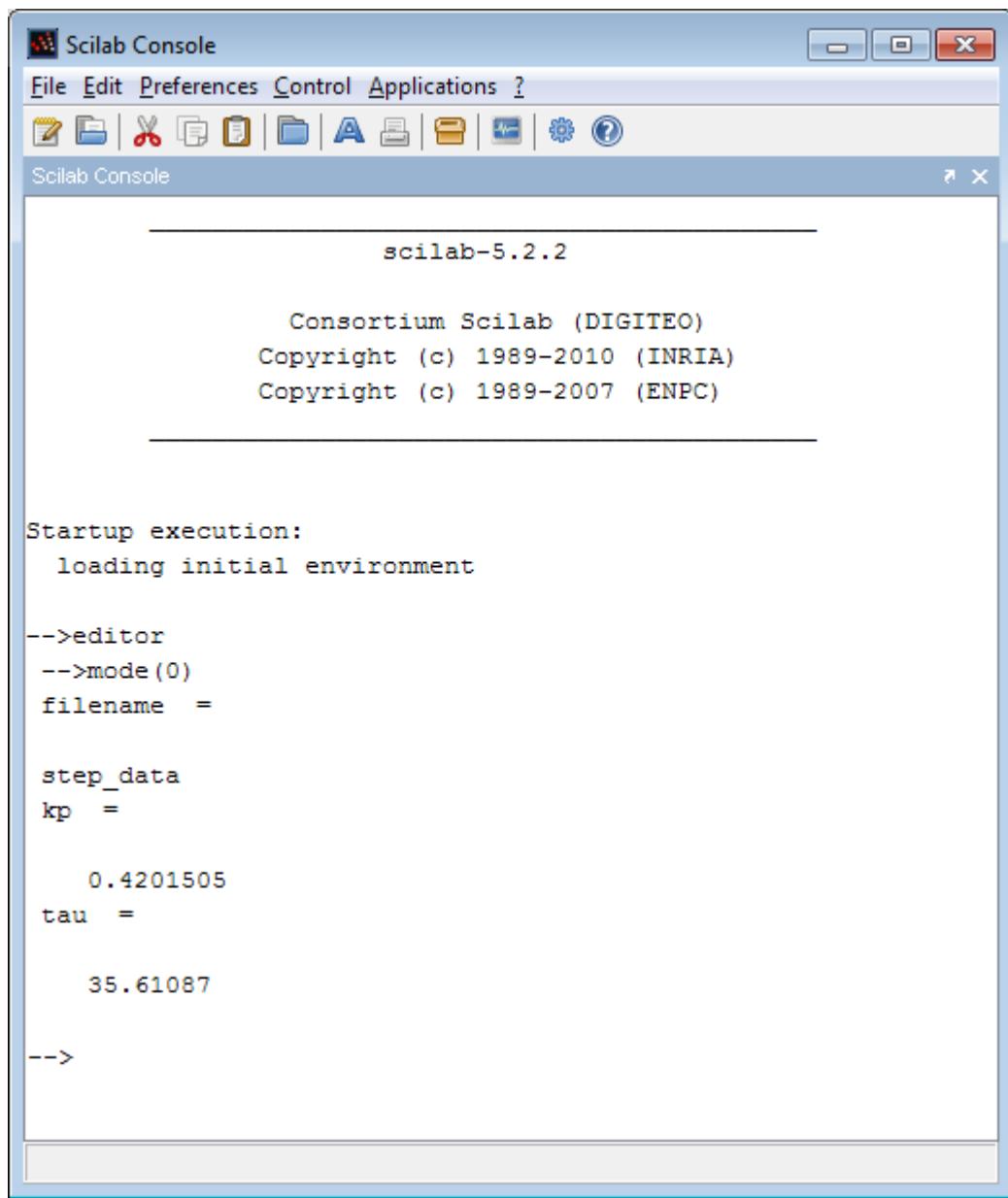
$$y(s) = \frac{K}{(\tau_1 s + 1)(\tau_2 s + 1)} \frac{\Delta u}{s} \quad (4.13)$$

Splitting this 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:

$$\begin{aligned} A &= K \\ B &= -\frac{K\tau_1}{\tau_1 - \tau_2} \\ C &= \frac{K\tau_2}{\tau_1 - \tau_2} \end{aligned}$$



The image shows a screenshot of the Scilab Console window. The window title is "Scilab Console". The menu bar includes "File", "Edit", "Preferences", "Control", "Applications", and "?". Below the menu is a toolbar with various icons. The main console area 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

-->editor
-->mode(0)
filename =
step_data
kp =
0.4201505
tau =
35.61087
-->
```

Figure 4.4: The value of time constant and gain as shown on the console by firstorder.sce

On substitution and inversion, we obtain

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

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` does these calculations and outputs a the gain and two time constants. Again change the scilab working directory to the folder `Kp-tau-order2`. Copy the step test data file in this folder. Run the code `secondorder.sce` with the appropriate data file name. The plot shown in 4.5 is obtained. It corresponds to the following transfer function with the parameters written at the top of the plot.

$$G_s(s) = \frac{0.420}{(34.4s + 1)(1s + 1)} \quad (4.15)$$

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

4.4 Discussion

We summarize our findings now. For the firstorder analysis the gain is 0.4201 and the time constant τ is 35.61 sec. For the second order analysis, the initial inflexion is well captured with the two time constants $\tau_1=34.4$, $\tau_2= 1$ and gain = 0.420. Negative steps can also be introduced to make the experiment more informative. One can even need not keep a particular input constant. By varying both the inputs, one can imagine it to be like a step varying disturbance signal.

4.5 Conducting Step Test on SBHS, virtually

The step by step procedure for conducting an experiment virtually is explained in section 3.4. The required .sce file is `stepc.sce`. You will find this file in the `StepTest` directory under `virtual` folder. Please note that the analysis code of step test data obtained by a virtual experiment is slightly different. The procedure to use the analysis code however remains the same as explained earlier. To do a first order analysis, one has to use the file `firstorder_virtual.sce`. Similarly, `secondorder_virtual.sce` for second order analysis. You will find this file in

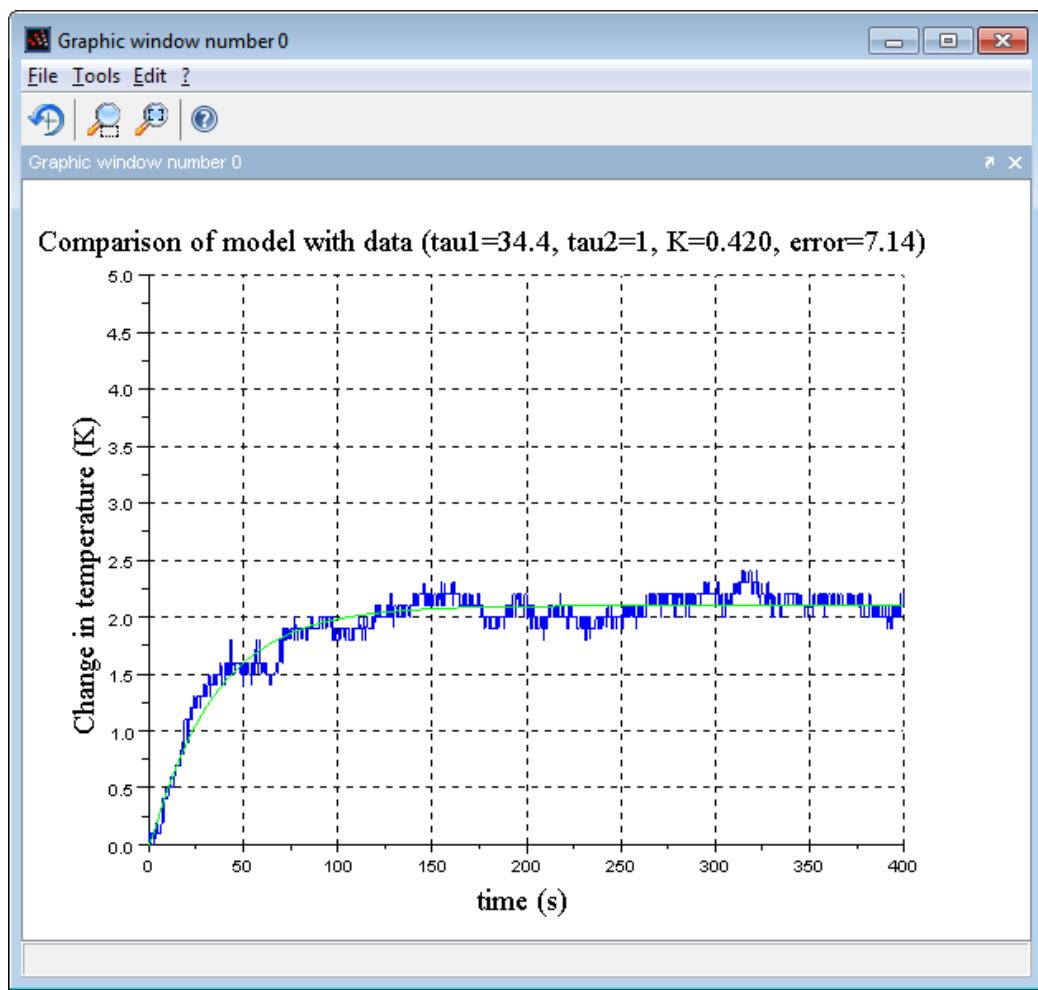


Figure 4.5: Output of the scilab code secondorder.sce

the StepTest directory under virtual folder. The necessary codes are listed in the section 4.6.

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
8 t = a.title;
9 t.font_size = tfont; // Title font size
10 t.font_style = 2; // Title font style
11 t.text = tname;
12
13 u = a.xlabel;
14 u.font_size = xyfont; // Label font size
15 u.font_style = 2; // Label font style
16
17 v = a.ylabel;
18 v.font_size = xyfont; // Label font size
19 v.font_style = 2; // Label font style
20
21 // a.label_font_size = 3;
22
23 endfunction;
```

Scilab Code 4.2 costf_1.sci

```
1 function [f,g,ind] = costf_1(x,ind)
2 kp = x(1); tau = x(2);
3 y_prediction = kp * ( 1 - exp(-t/tau) );
```

```

4 f = (norm(y-y_prediction,2))^2;
5 g = numdiff(func_1,x);
6 endfunction
7
8 function f = func_1(x)
9 kp = x(1); tau = x(2);
10 y_prediction = kp * (1 - exp(-t/tau));
11 f = (norm(y-y_prediction,2))^2;
12 endfunction

```

Scilab Code 4.3 firstorder.sce

```

1 mode(0)
2 funcprot(0)
3 filename = "step_data"
4 clf
5 exec('costf_1.sci');
6 exec('label.sci');
7 data = fscanfMat(filename);
8 time = data(:, 1);
9 heater = int(data(:, 2));
10 fan = int(data(:, 3));
11 temp = data(:, 4);
12
13
14 len = length(heater);
15
16 time2 = 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 = time(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 kp = xopt(1);
41 tau = xopt(2);
42 y_prediction = kp * (1 - exp(-t/tau));
43 plot2d(t, y_prediction);
44 plot2d(t, y);
45 label('Showing First Order Model and Experimental
   Results', 4, 'Time (s)', 'Change in temperature (K)'
   , 4);
46 kp = kp/delta_u
47 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 lster = 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); lster =
   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 = [H T y_prediction]; x = [t t t];
14 // xbase();
15 plot2d(x,ord ,rect=limits), xgrid();
16 title = 'Comparison of model with data (tau1='
17 title = title+string(tau1)+', tau2='+string(tau2)
18 title = title+', K='+string(kp)
19 title = title+', error='+string(lster)+')'
20 label(title ,4 , 'time (s)', 'Change in temperature (K)'
   ,4);
21 endfunction;

```

Scilab Code 4.6 secondorder.sce

```

1 mode(0)
2 funcprot(0)

```

```

3  filename = "step_data"
4  clf
5  exec('costf_2.sci');
6  exec('label.sci');
7  exec ('order_2_heater.sci');

8
9
10 data = fscanfMat(filename);
11 time = data(:, 1);
12 heater = int(data(:, 2));
13 fan = int(data(:, 3));
14 temp = data(:, 4);

15 // times =[ time(1) ; time(1:$-1) ];
16 time2 = 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 = time(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 = [0,0,1000,10]; no=10000; // first step
36 // limits = [400,0,900,26]; no = 5000; // second step
37 lterr = order_2(t,H,T,limits,no)

```

Scilab Code 4.7 ser_init.sce

```

1 // To load the serial toolbox and open the serial port
2 exec loader.sce
3
4
5
6 handl = openserial("/dev/ttyUSB0","9600,n,8,0")
7
8 // the order is : port number , baud , parity , databits ,
9 // stop bits "
10 // here 9 is the port number
11 // In the case of SBHS , stop bits = 0 , so it is not
12 // specified in the function here
13 // Linux users should give this as ("/", "9600 , n , 8 , 0")
14
15 if (ascii(handl) ~= [])
16   disp("COM Port Opened");
17 end

```

Scilab Code 4.8 step-test.sci

```

1 global temp heat fan
2
3 function temp = step_test(heat,fan)
4
5 writeserial(handl,ascii(254)); // Input Heater ,
6 writeserial accepts
7 convert 254 into its
8 strings ; so
9 equivalent
10 writeserial(handl,ascii(heat));
11 writeserial(handl,ascii(253)); // Input Fan
12 writeserial(handl,ascii(fan));
13 writeserial(handl,ascii(255)); // To read Temp
14 sleep(100);
15
16 temp = ascii(readserial(handl)); // Read serial
17 returns a string , so

```

```

    convert it to
its integer (ascii)
equivalent
13 temp = temp(1) + 0.1*temp(2); // convert to temp with
decimal points
e.g.: 40.7
14
15 endfunction

```

Scilab Code 4.9 stepc.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 fncrefnetworkcounty
5
6 fncrefnetworkcounty = 'scilabread.sce';
7 fdt = mopen(fncrefnetworkcounty);
8 mseek(0);
9
10 err_count = 0; // initialising error count for network
   error
11 m = 1;
12 exec ("steptest.sci");
13 A = [0.1, 0, 100, m];
14 fdfh = file('open', 'scilabwrite.sce', 'unknown');
15 write(fdfh, A, '(7e11.5, 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)
   xcos('steptest.xcos');
21 else
22   disp("NO NETWORK CONNECTION!");
23   return

```

25 **end**

Scilab Code 4.10 steptest.sci

```
1 mode(0);
2 function [y,stop] = steptest(heat,fan)
3 global fdfh fdt fncre fncrew m err_count stop
4
5
6 fncre = 'scilabread.sce'; // file to be read -
    temperature
7 fncrew = 'scilabwrite.sce'; // file to be written
    - heater , fan
8
9 a = mgetl(fdt,1);
10 b = evstr(a);
11 byte = mtell(fdt);
12 mseek(byte,fdt,'set');
13
14 if a~= []
15     temps = b(1,4); heats = b(1,2);
16     fans = b(1,3); y = temps;
17
18     if heat>100
19         heat = 100;
20     elseif heat<0
21         heat = 0;
22     end;
23
24     if fan>100
25         fan = 100;
26     elseif fan<0
27         fan = 0;
28     end;
29
30 A = [m,heat,fan,m];
31 fdfh = file('open','scilabwrite.sce','unknown');
32 file('last',fdfh)
```

```

33   write(fdः,A,'(e11.5,1x))');
34   file('close', fdः);
35   m = m+1;
36
37   else
38     y = 0;
39     err_count = err_count + 1; // counts the no. of
      times network error occurs
40   if err_count > 200
41     disp("NO NETWORK COMMUNICATION!");
42     stop = 1; // status set for stopping simulation
43   end
44 end
45
46 return
47 endfunction

```

Scilab Code 4.11 firstorder_virtual.sce

```

1 mode(0)
2 filename = "27Jan2012_20_13_32.txt"
3 clf
4 exec('costf_1.sci');
5 exec('label.sci');
6 data = fscanfMat(filename);
7 time = data(:, 5);
8 heater = int(data(:, 2));
9 fan = int(data(:, 3));
10 temp = data(:, 4);
11
12
13 len = length(heater);
14
15 time1 = time - time(1);
16 time2 = time1/1000;
17
18 len = length(heater);
19 heaters1 = [heater(1); heater(1:len-1)];

```

```

20 del_heat = heater - heaters1;
21 ind = find(del_heat > 1);
22
23 step_instant = ind($)-1;
24
25 t = time2(step_instant:len);
26 t = t - t(1);
27 H = heater(step_instant:len);
28 F = fan(step_instant:len);
29 T = temp(step_instant:len);
30 T = T - T(1);
31 delta_u = heater(step_instant + 1) - heater(
    step_instant);
32
33 // finding Kp and Tau between Heater (H) and
   Temperature (T)
34 y = T; // temperature
35 global('y','t');
36 x0 = [.3 40];
37 // [f, xopt, gopt] = optim(costf_1, 'b', [0.1 0.1], [5 100],
   x0, 'ar')
38 [f, xopt] = optim(costf_1, x0);
39 kp = xopt(1);
40 tau = xopt(2);
41 y_prediction = kp * (1 - exp(-t/tau));
42 plot2d(t, y_prediction);
43 plot2d(t, y);
44 label('Showing First Order Model and Experimental
   Results', 4, 'Time (s)', 'Change in temperature (K)',
   4);
45 kp = kp/delta_u
46 tau

```

Scilab Code 4.12 secondorder_virtual.sce

```

1 mode(0)
2 filename = "27Jan2012_20_13_32.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 lterr = order_2(t,H,T,limits,no)

```

Chapter 5

Identification of transfer function of a Single Board Heater System through Ramp response experiments

The Aim of this experiment is to perform Ramp test on the Single Board Heater System and to identify the system transfer function using Ramp response data. The target group is anyone who has basic knowledge of Control Engineering.

5.1 About this Experiment

We have used Scilab-5.2.2 and Xcos for sending and receiving data. This interface is shown in Fig.5.1. Heater current and fan speed are the two inputs to the SBHS system. They are given in PWM units. These inputs can be varied through the Xcos interface by setting the properties of the input blocks in Xcos. The data acquired in the process is stored in the local drive using the "Write to output file" block and is available to the user for further analysis.

5.2 Theory

Identification of the transfer function of a system is quite important since it helps us to model the physical system mathematically. Once the transfer function is obtained one can find out the response of the system, to various inputs, without actually applying them to the system. Consider the standard first order transfer

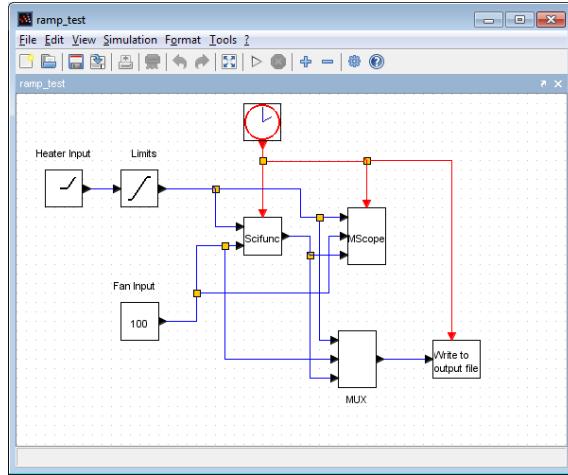


Figure 5.1: Xcos for Ramp Test experiment

function given below

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

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

Rewriting the equation by substituting equation 4.2 in equation 4.1, we get

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

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

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

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

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

$$C(s) = Kv \left\{ \frac{1}{s^2} - \frac{\tau}{s} + \frac{\tau^2}{\tau s + 1} \right\} \quad (5.6)$$

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

$$c(t) = Kv \left\{ t - \tau + \tau e^{-\frac{t}{\tau}} \right\} \quad (5.7)$$

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

$$e(t) = r(t) - c(t) \quad (5.8)$$

$$e(t) = Kv t - Kv t + Kv \tau - Kv \tau e^{-\frac{t}{\tau}} \quad (5.9)$$

$$e(t) = Kv \tau (1 - e^{-\frac{t}{\tau}}) \quad (5.10)$$

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

$$e(t) = \tau \quad (5.11)$$

This means that the error in following the ramp input is equal to τ for large value of t [7]. Hence, the smaller the time constant τ , the smaller is the steady state error.

5.3 Step by step procedure to perform Ramp Test

Change the scilab working directory to `Ramp_Test` folder. Execute the code `ser_init.sce` and `ramp_test.sci`. Open the Xcos code `ramp_test.xcos`. 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 labeled as "Heater input". Put the following values in the respective fields. Slope = 0.1, start time = 300, initial output = 20. Keep the fan constant at 100. Note that the value of heater current will not exceed 40 PWM units due to the use of limit blocks. Running the xcos code may warn you with a message saying "No continuous-time states. Thresholds are ignored". Ignore the message by clicking on OK button. The data thus obtained is stored using "Write to output file" Xcos block as shown in Fig.5.1. The first column of Table 5.1 denotes time in seconds. The second column denotes heater current. The third column denotes the fan speed. It has been held constant at 100 units. The last column denotes the plate temperature.

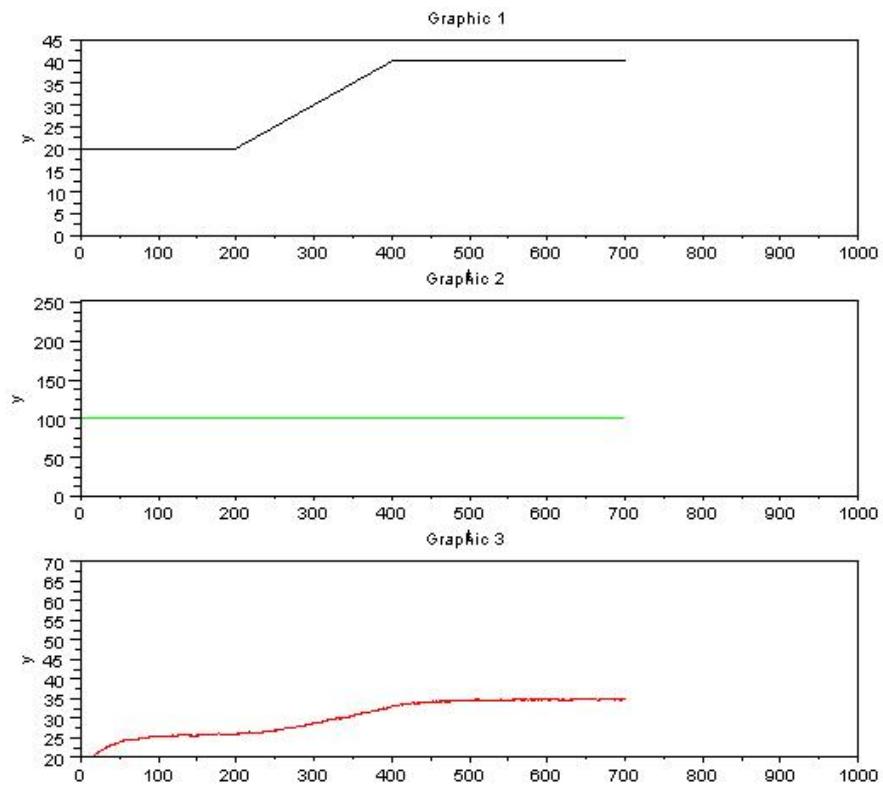


Figure 5.2: Screen shot of Ramp Test Experiment

```

0.000E+00  0.100E+02  0.100E+03  0.216E+02
0.100E+00  0.100E+02  0.100E+03  0.216E+02
.
.
.
0.251E+03  0.300E+02  0.100E+03  0.291E+02
0.251E+03  0.300E+02  0.100E+03  0.291E+02

```

Table 5.1: Ramp data obtained after performing the Ramp Test

5.4 Ramp Analysis

After completing the ramp test experiment, let us do the analysis. Change the directory to `Ramp Analysis`. Execute the file `ramp.sce`. On executing this file, you get the values of K_p , τ and K_p approx and τ approx on the Scilab Console Window. You will also get a plot of the ramp response calculated using the equation 5.7 for K_p and τ values.

Showing First Order Model and Experimental Results for k_p and τ

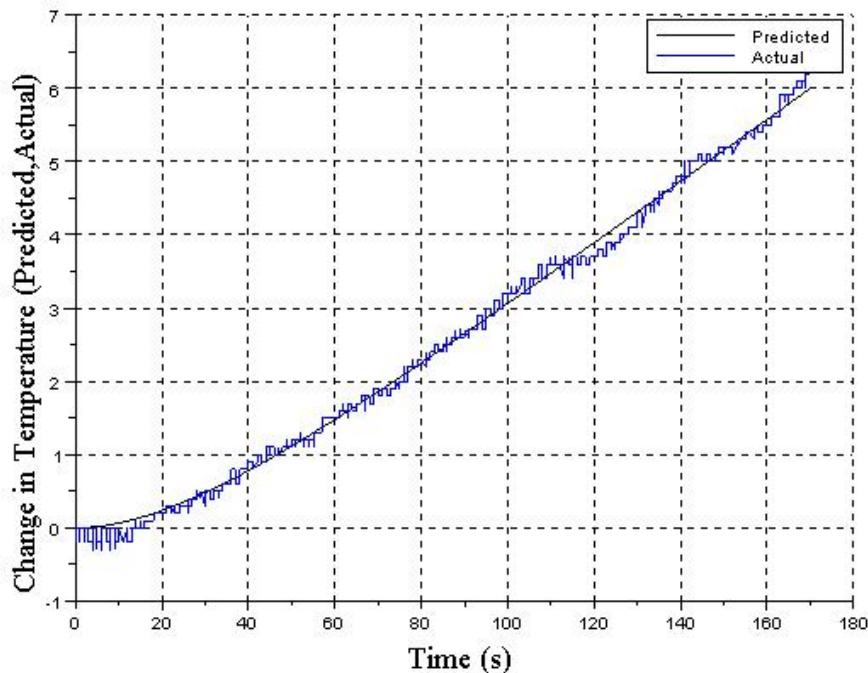


Figure 5.3: Ramp response for K_p and τ

5.5 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 try out the experiment using different combinations of fan speed

and heater current. Negative ramp can also be tried out 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 since it increases the accuracy of the acquired transfer function.[6]

5.6 Conducting Ramp Test on SBHS, virtually

The step by step procedure for conducting an experiment virtually is explained in section 3.4. The required .sce file is `ramptest.sce`. You will find this file in the `RampTest` directory under `virtual` folder. Please note that the analysis code of ramp test data obtained by a virtual experiment is slightly different. The procedure to use the analysis code however remains the same as explained earlier. To do a first order analysis, one has to use the file `firstorder_virtual.sce`. These files are available in the `Ramp Analysis` folder under the `virtual` folder. The necessary codes are listed in the section 5.7.

5.7 Scilab Code

Scilab Code 5.1 ramp_test.sci

```
1 mode(-1);
2 global temp heat fan
3
4 function temp = ramp_test(heat,fan)
5
6 writeserial(handl,ascii(254)); // Input Heater,
    writeserial accepts strings; so convert 254 into
    its
    equivalent
7 writeserial(handl,ascii(heat));
8 writeserial(handl,ascii(253)); // Input Fan
9 writeserial(handl,ascii(fan));
10 writeserial(handl,ascii(255)); // To read Temp
11 sleep(100);
12
```

```

13 temp = ascii(readserial(handle)); // Read serial
      returns a string , so
                           convert it to
      its integer (ascii) equivalent
14 temp = temp(1) + 0.1*temp(2); // convert to temp with
      decimal points
      e.g : 40.7
15
16 endfunction

```

Scilab Code 5.2 label.sci

```

1 mode(-1);
2 // Updated (9-12-06) , written by Inderpreet Arora
3 // Input arguments : title , xlabel , ylabel and their
   font sizes
4
5 function label(tname , tfont , labelx , labely , xyfont)
6 a = get("current_axes")
7 xtitle(tname , labelx , labely)
8 xgrid
9 t = a.title;
10 t.font_size = tfont; // Title font size
11 t.font_style = 2; // Title font style
12 t.text = tname;
13
14 u = a.xlabel;
15 u.font_size = xyfont; // Label font size
16 u.font_style = 2; // Label font style
17
18 v = a.ylabel;
19 v.font_size = xyfont; // Label font size
20 v.font_style = 2; // Label font style
21
22 // a.label_font_size = 3;
23
24 endfunction;

```

Scilab Code 5.3 cost.sci

```
1 function f = func_1(x)
2     k = x(1);
3     tau = x(2);
4     y_prediction = k*(t + tau*(exp(-t/tau) - 1));
5     f = (norm(y - y_prediction,2))^2;
6 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 mode(0);
2 function [y,stop] = ramptest(heat,fan)
```

```

3 global fdfh fdt fncre fncre m err_count stop
4
5
6 fncre = 'scilabread.sce'; // file to be read -
    temperature
7 fncre = 'scilabwrite.sce'; // file to be written
    - heater , fan
8
9 a = mgetl(fdt,1);
10 b = evstr(a);
11 byte = mtell(fdt);
12 mseek(byte,fdt,'set');
13
14 if a~= []
15     temps = b(1,4); heats = b(1,2);
16     fans = b(1,3); y = temps;
17
18 if heat>100
19     heat = 100;
20 elseif heat<0
21     heat = 0;
22 end;
23
24 if fan>100
25     fan = 100;
26 elseif fan<0
27     fan = 0;
28 end;
29
30 A = [m,heat,fan,m];
31 fdfh = file('open','scilabwrite.sce','unknown');
32 file('last',fdfh)
33 write(fdfh,A,'(7e11.5,1x)'));
34 file('close',fdfh);
35 m = m+1;
36
37 else
38     y = 0;

```

```

39     err_count = err_count + 1; // counts the no of
        times network error occurs
40 if err_count > 300
    disp("NO NETWORK COMMUNICATION!");
42 stop = 1; // status set for stopping simulation
43 end
44 end
45
46 return
47 endfunction

```

Scilab Code 5.6 rampptest.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 fncre fncre m err_count y
5
6 fncre = 'scilabread.sce';
7 fdt = mopen(fncre);
8 mseek(0);
9
10 err_count = 0; // initialising error count for network
    error
11 m = 1;
12 exec ("rampptest.sci");
13 A = [0.1,m,0,100];
14 fdfh = file('open','scilabwrite.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)
    xcos('rampptest.xcos');
21 else

```

```

23     disp ("NO NETWORK CONNECTION!");  

24     return  

25 end

```

Scilab Code 5.7 ramp_virtual.sce

```

1 mode(-1);  

2  

3 filename = "16Feb2012_12_18_19.txt"; // complete path  

   of the saved data file  

4 slope = 0.1; // change this to the slope that you have  

   used in the experiment  

5 ind1=3;  

6 // Ramp Analysis  

7 exec('cost_approx.sci');  

8 exec('cost.sci');  

9 exec('label.sci');  

10  

11 data = fscanfMat(filename);  

12 time = data(:, 5);  

13 heater = int(data(:, 2));  

14 fan = int(data(:, 3));  

15 temp = data(:, 4);  

16  

17  

18 len = length(heater);  

19 heaters1 = [heater(1); heater(1:$-1)];  

20 del_heat = abs(heater - heaters1);  

21 ind = find(del_heat >.5);  

22  

23 t = time(ind(2):ind($-1));  

24 H = heater(ind(2):ind($-1));  

25 T = temp(ind(2):ind($-1));  

26  

27 t = t - t(1);  

28 T = T - T(1);  

29  

30 y = T;

```

```

31 x0 = [.5 100]
32 global('y','t');
33
34 [f, xopt] = optim(cost,x0);
35 kp = xopt(1)/slope
36 tau = xopt(2)
37
38 len = length(t);
39 halfway = ceil(len/2);
40
41 t_approx = t(halfway:len);
42 y_approx = y(halfway:len);
43 global('y_approx','t_approx');

44
45 [f_approx,xopt_approx] = optim(cost_approx,x0);
46 kp_approx = xopt_approx(1)/slope;
47 tau_approx = xopt_approx(2);

48 // Display and Plot
49 disp('kp = ');
50 disp(kp);
51 disp('tau = ');
52 disp(tau);
53 disp('kp_approx = ');
54 disp(kp_approx);
55 disp('tau_approx = ');
56 disp(tau_approx);

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

```

Chapter 6

Frequency Response Analysis of a Single Board Heater System by the application of Sine Wave

The aim of this experiment is to do a Frequency Response Analysis of a Single Board Heater System by the application of Sine Wave. 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 Fig.6.1. Heater current and Fan speed are the two inputs to the system. The Heater current is varied sinusoidally. A provision is made to set the parameters related to it, like Frequency, Amplitude and Offset. The temperature profile thus obtained is the output. In this experiment we are applying a sine change in the heater current by keeping the Fan speed constant. After application of sine change, wait for sufficient amount of time to allow the temperature to reach a steady-state.

6.1 Theory

Frequency Response of a system means its steady-state response to a sinusoidal input. For obtaining a Frequency Response of a system, we vary the frequency of the input signal over a spectrum of interest. The analysis is actually quite useful and also simple because it can be carried out with the available signal generators and measuring devices. Consider a sinusoidal input

$$U(t) = A \sin \omega t \quad (6.1)$$

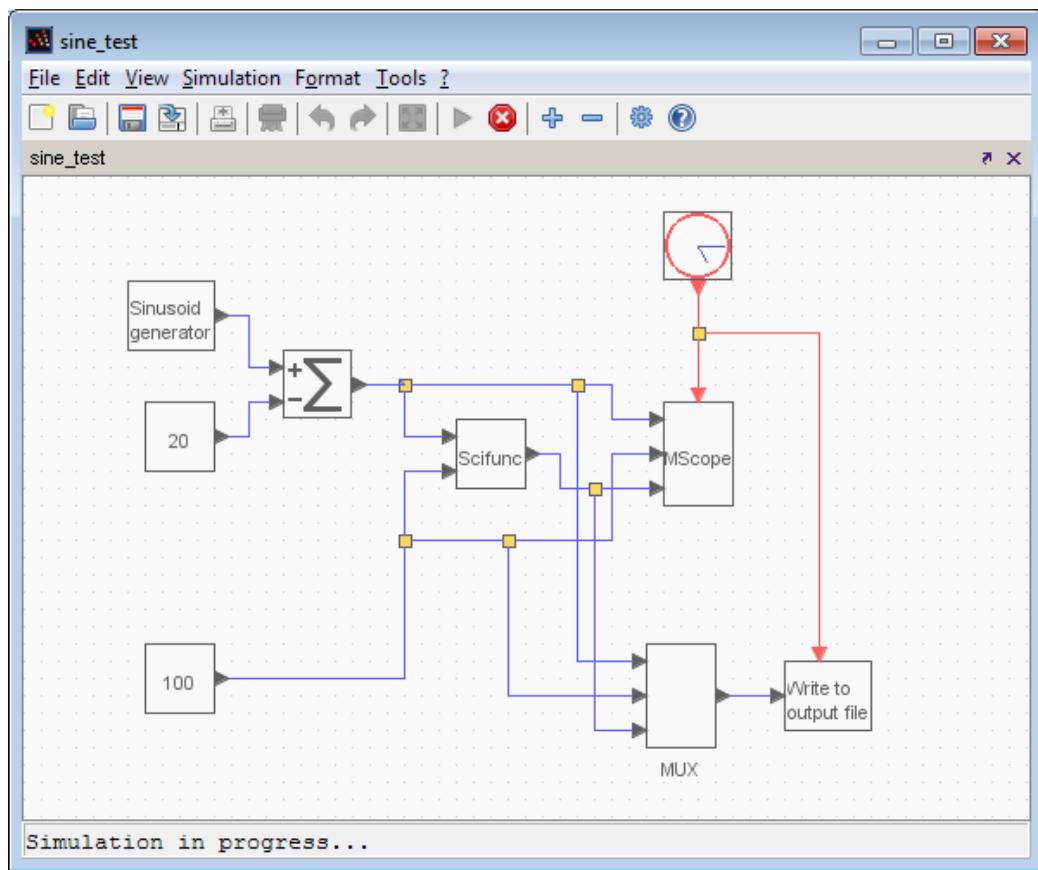


Figure 6.1: Xcos for this experiment

The Laplace Transform of the above equation yields

$$U(s) = \frac{A\omega}{s^2 + \omega^2} \quad (6.2)$$

Consider the standard first order transfer function given below

$$G(s) = \frac{Y(s)}{U(s)} = \frac{K}{s + 1} \quad (6.3)$$

Putting the value of $U(s)$ from equation, we get

$$Y(s) = \frac{KA\omega}{(\tau s + 1)(s^2 + \omega^2)} \quad (6.4)$$

$$= \frac{KA}{\omega^2\tau^2 + 1} \left[\frac{\omega\tau^2}{\tau s + 1} - \frac{\tau s\omega}{s^2 + \omega^2} + \frac{\omega}{s^2 + \omega^2} \right] \quad (6.5)$$

Taking Laplace Inverse, we get

$$y(t) = \left[\frac{KA}{\omega^2\tau^2 + 1} \right] \left[\omega\tau e^{\frac{-t}{\tau}} - \omega\tau \cos(\omega t) + \sin(\omega t) \right] \quad (6.6)$$

The above equation has an exponential term $e^{\frac{-t}{\tau}}$. Hence, for large value of time, its value will approach to zero and the equation will yield a pure sine wave. One can also use trigonometric identities to make the equation look more simple.

$$y(t) = \left[\frac{KA}{\sqrt{\omega^2\tau^2 + 1}} \right] [\sin(\omega t) + \phi] \quad (6.7)$$

where,

$$\phi = -\tan^{-1}(\omega\tau) \quad (6.8)$$

By observing the above equation one can easily make out that for a sinusoidal input the output is also sinusoidal but has some phase difference. Also, the amplitude of the output signal, \hat{A} , has become a function of the input signal frequency, ω .

$$\hat{A} = \frac{KA}{\sqrt{\omega^2\tau^2 + 1}} \quad (6.9)$$

The amplitude ratio (AR) can be calculated by dividing both sides by the input signal amplitude A.

$$AR = \frac{\hat{A}}{A} = \frac{K}{\sqrt{\omega^2\tau^2 + 1}} \quad (6.10)$$

Dividing the above equation by the process gain K yields the normalized amplitude ratio (AR_n)

$$AR_n = \frac{AR}{K} = \frac{1}{\sqrt{\omega^2\tau^2 + 1}} \quad (6.11)$$

Because the process steady state gain is constant, the normalized amplitude ratio often is used for frequency response analysis.[8]

(6.12)

6.2 Step by step procedure to perform Sine Test

Change the current working directory of scilab to the folder **Sine_Test**. Execute the code **ser_init.sce** and **sine_test.sci**. Open the Xcos code **sine_test.xcos**. Initiate a sine input to the system by setting Sinusoid generator block properties with some value of the frequency (here 0.007Hz) and amplitude(hear 10). Note that at high frequencies the plant output is not sinusoidal,which is not of any use. Hence,avoid choosing frequencies above 0.04Hz .

Refering to table 6.1 the first column represents time. The second column represents heater current. Here, it is sinusoidally varied. The third column represents fan speed. Note that its value is 100 throughout the experiment. The fourth column represents the output temperature. It should be taken in to consideration that all the values mentioned in the data file are in PWM (Pulse Width Modulation) units, except for the temperature which is in $^{\circ}\text{C}$.

Now let us see the step for calculating Amplitude Ratio and Phase Difference. Change the current working directory of scilab to the folder **Sine_Analysis**. Copy the data files generated after the completion of the experiment in to the **Sine_Analysis** folder. Input the arguments **f** and **filename** in the scilab code **sine2.sce** for the calculation of the above said parameters and execute it. Here **f** means input frequency. It could be seen from figure 6.3 that the Amplitude

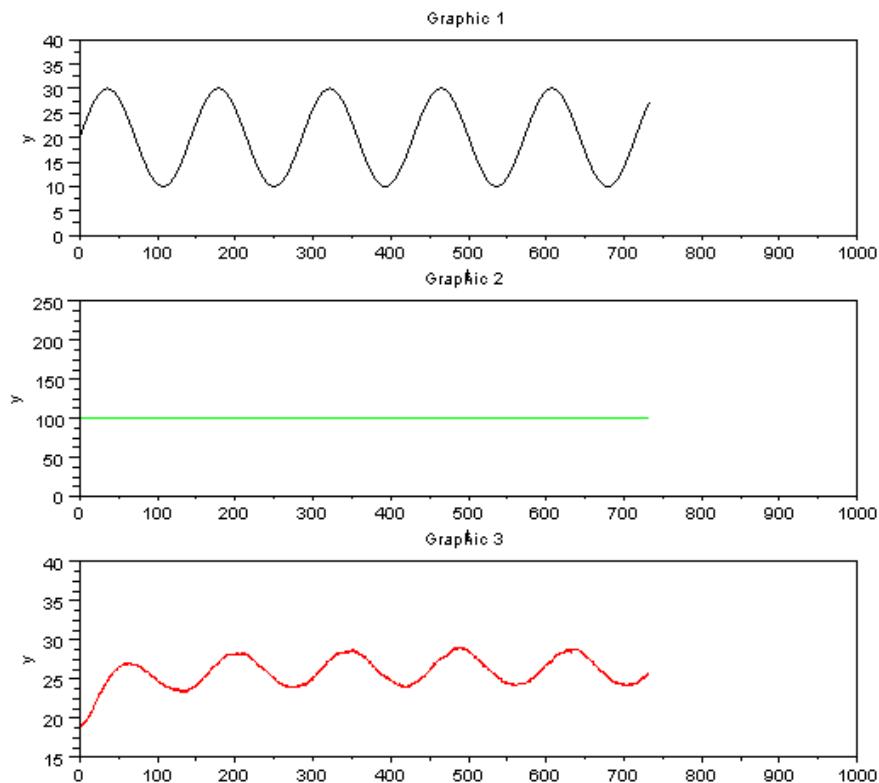


Figure 6.2: Plot for sine input 0.007Hz

```

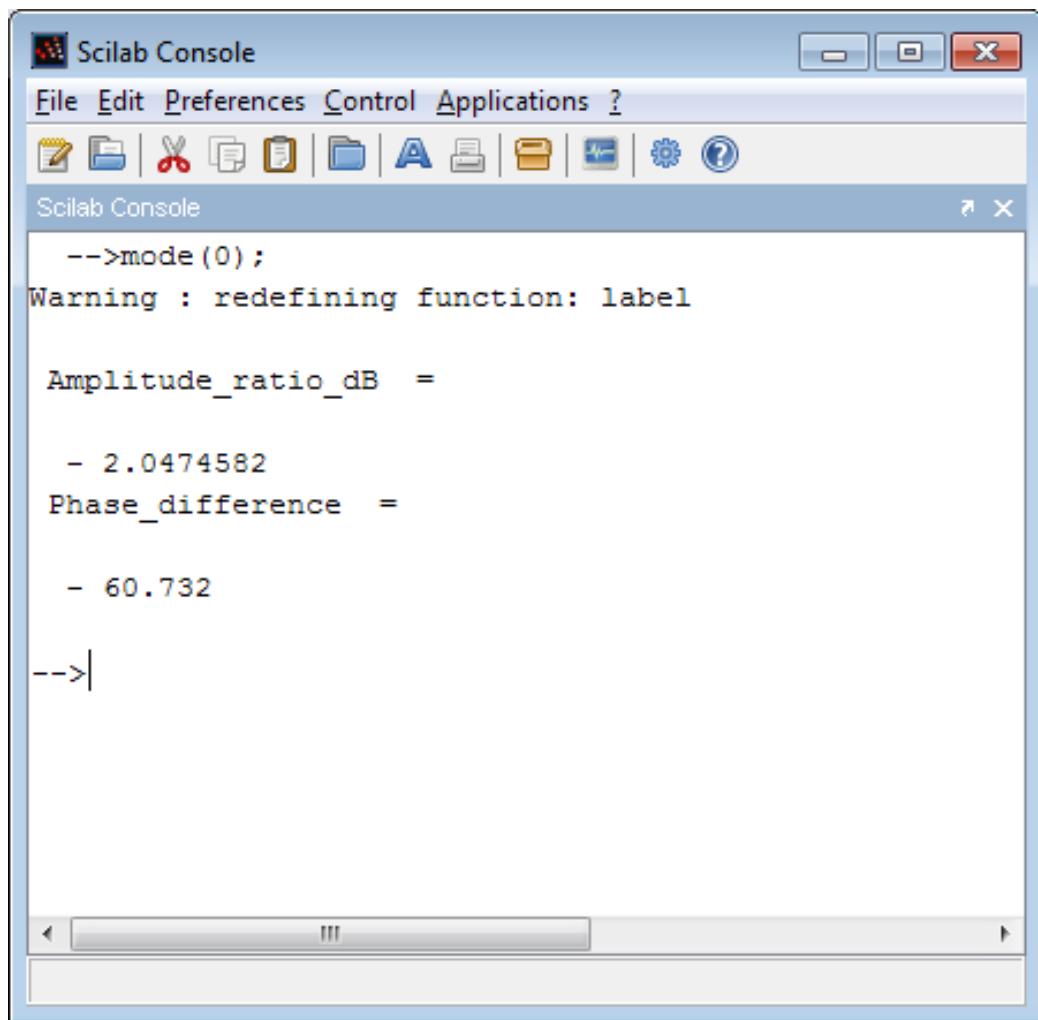
0.100E+00  0.200E+02  0.100E+03  0.239E+02
0.200E+00  0.201E+02  0.100E+03  0.238E+02
0.300E+00  0.201E+02  0.100E+03  0.238E+02

.
.

0.749E+03  0.300E+02  0.100E+03  0.301E+02
0.749E+03  0.300E+02  0.100E+03  0.302E+02
0.749E+03  0.300E+02  0.100E+03  0.302E+02

```

Table 6.1: Data obtained after application of sine input of 0.04Hz



The image shows a screenshot of the Scilab Console window. The title bar reads "Scilab Console". The menu bar includes "File", "Edit", "Preferences", "Control", "Applications", and "?". The toolbar below the menu contains icons for file operations like Open, Save, Print, and others. The main console area displays the following text:

```
-->mode(0);
Warning : redefining function: label

Amplitude_ratio_dB =
- 2.0474582
Phase_difference =
- 60.732
-->|
```

Figure 6.3: Scilab Output

Plot of sine input in heater and the corresponding temperature profile

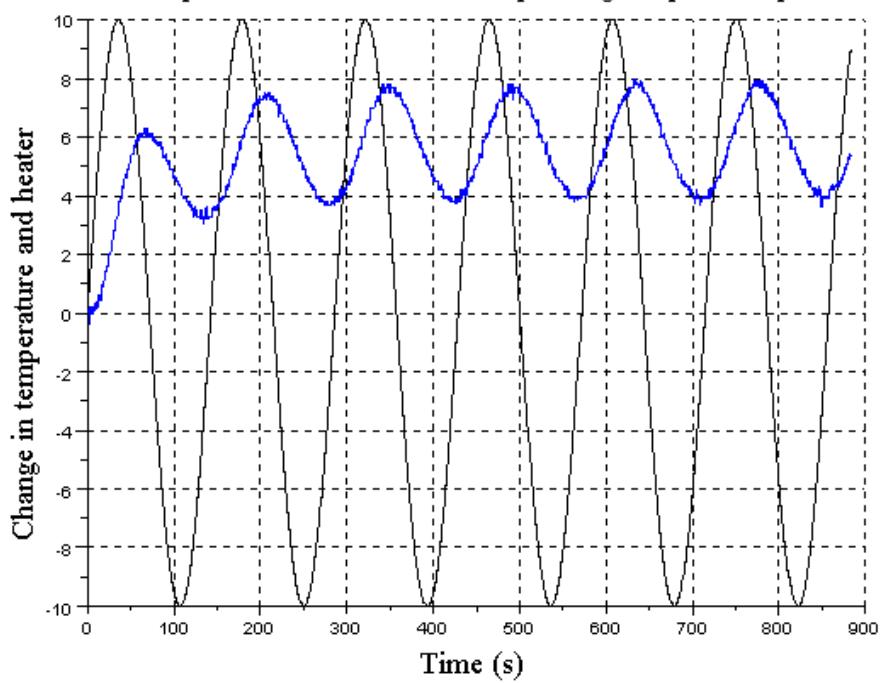


Figure 6.4: Plot of Input and Output vs time

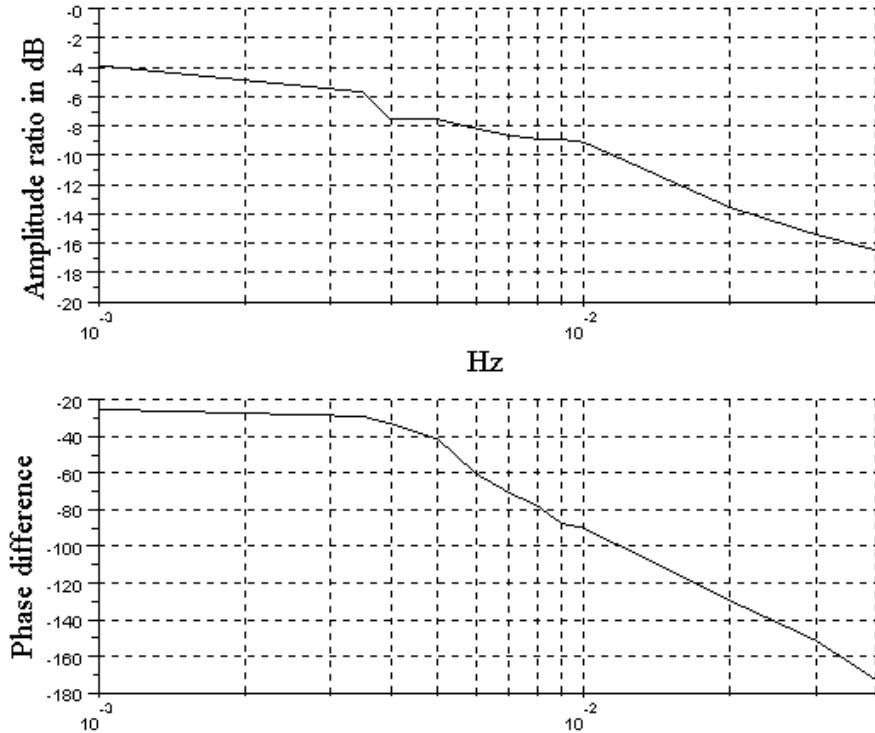


Figure 6.5: Bode Plot obtained from the plant

Ratio turns out to be -2.047dB and phase difference to be -60.732° . The plot thus obtained is shown in figure 6.4

Repeat this calculation over a range of frequencies and note down the values of Amplitude Ratio in dB and Phase Difference. Input these values for the appropriate frequencies in to the Scilab code `BodePlot.sce` and execute it to get a Bode Plot of the plant which is illustrated in figure 6.5.

Bode Plot can be obtained directly from the plants Secon order Transfer function [6] with the help of scilab code `TFbode.sce`, as shown in figure 6.6. A visual comparison of the two bode plots can be done to validate the bode diagram obtained from the plant.

For comparing above two plots we are plotting it on same graph as shown in figure 6.7

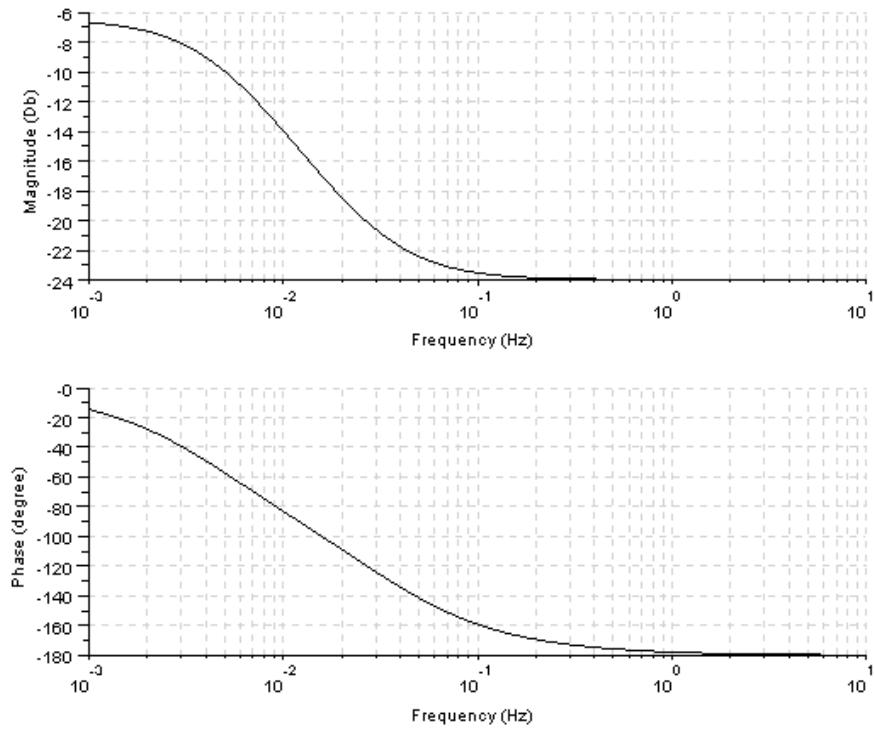


Figure 6.6: Bode Plot obtained through plants Transfer function

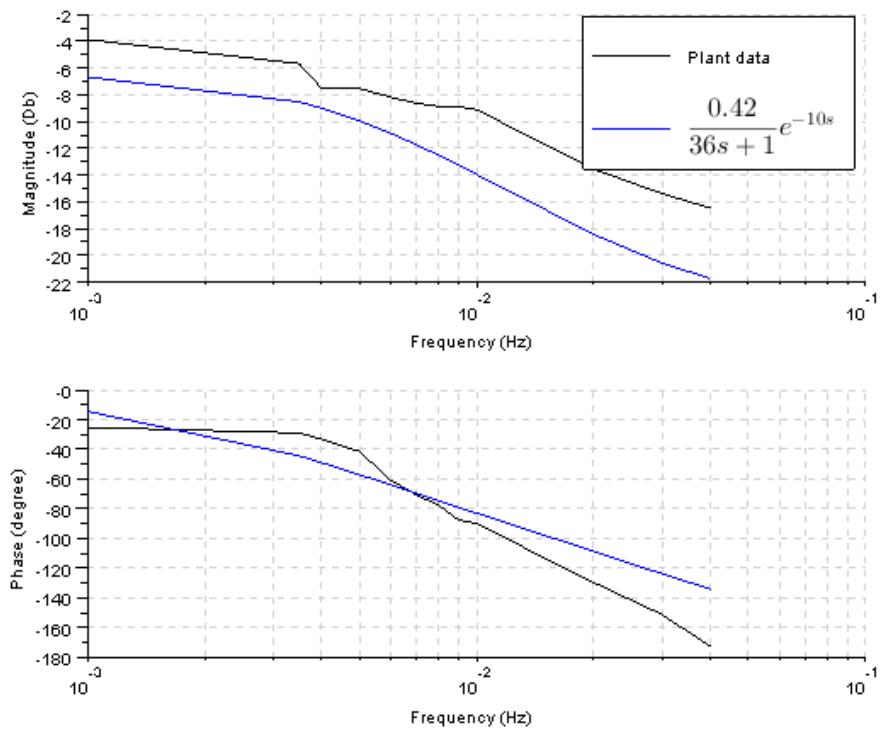


Figure 6.7: Comparison of Bode Plots

6.3 Conducting Sine Test on SBHS, virtually

The step by step procedure for conducting an experiment virtually is explained in section 3.4. The required .sce file is `sinetest.sce`. You will find this file in the `SineTest` directory under `virtual` folder. Please note that the analysis code of sine test data obtained by a virtual experiment is slightly different. The procedure to use the analysis code however remains the same as explained earlier. To calculate the **Amplitude Ratio** and **Phase Difference**, one has to use the file `sine2_virtual.sce`. These files are available in the `Sine Analysis` folder under the `virtual` folder. The necessary codes are listed in the section 6.4.

6.4 Scilab Code

Scilab Code 6.1 `sine_test.sci`

```
1 global temp heat fan
2
3 function temp = sine_test(heat,fan)
4
5 writeserial(handl,ascii(254)); // Input Heater ,
6           writeserial accepts
7           strings ; so
8           convert 254 into its
9           string
10          equivalent
11
12 writeserial(handl,ascii(heat));
13 writeserial(handl,ascii(253)); // Input Fan
14 writeserial(handl,ascii(fan));
15 writeserial(handl,ascii(255)); // To read Temp
16 sleep(100);
17
18 temp = ascii(readserial(handl)); // Read serial
19           returns a string , so
20           convert it to
21           its integer (ascii)
22           equivalent
```

```

13 temp = temp(1) + 0.1*temp(2); // convert to temp with
      decimal points
      e.g.: 40.7
14
15 endfunction

```

Scilab Code 6.2 sinetest.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 fncrefnetworkcount y
5
6 fncrefnetworkcount = 'scilabread.sce';
7 fdt = mopen(fncrefnetworkcount);
8 mseek(0);
9
10 err_count = 0; // initialising error count for network
    error
11 m=1;
12 exec ("sinetest.sci");
13 A = [0.1,m,0,100];
14 fdfh = file('open','scilabwrite.sce','unknown');
15 write(fdfh,A,'(7e11.5,1x)');
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)
   xcos('sinetest.xcos');
21 else
22   disp("NO NETWORK CONNECTION!");
23   return
24
25 end

```

Scilab Code 6.3 sinetest.sci

```

1 mode(0);
2 function [y,stop] = sinetest(heat,fan)
3 global fdfh fdt fncrefn fncw m err_count stop
4
5
6 fncrefn = 'scilabread.sce'; // file to be read -
    temperature
7 fncw = 'scilabwrite.sce'; // file to be written
    - heater , fan
8
9 a = mgetl(fdt,1);
10 b = evstr(a);
11 byte = mtell(fdt);
12 mseek(byte,fdt,'set');
13
14 if a~= []
15     temps = b(1,4); heats = b(1,2);
16     fans = b(1,3); y = temps;
17
18 if heat>100
19     heat = 100;
20 elseif heat<0
21     heat = 0;
22 end;
23
24 if fan>100
25     fan = 100;
26 elseif fan<0
27     fan = 0;
28 end;
29
30 A = [m,heat,fan,m];
31 fdfh = file('open','scilabwrite.sce','unknown');
32 file('last',fdfh)
33 write(fdfh,A,(7e11.5,1x));
34 file('close',fdfh);
35 m = m+1;
36
```

```

37     else
38         y = 0;
39         err_count = err_count + 1; // counts the no of
             times network error occurs
40         if err_count > 300
41             disp("NO NETWORK COMMUNICATION!");
42             stop = 1; // status set for stopping simulation
43         end
44     end
45
46 return
47 endfunction

```

Scilab Code 6.4 sine2.sce

```

1 mode(0);
2 filename= 'sine001'; // Enter the data file name in
           single quotes
3 f=0.001; // Enter the frequency
4 data7=fscanfMat(filename);
5 exec('labelbode.sci');
6 T = data7(:,1); fan = data7(:,3); // T is time , fan is
           fan speed
7 u = data7(:,2)-data7(1,2); y = data7(:,4)-data7(1,4);
           // u is current , y is temperature
8 period=ceil(1/f);
9 p=length(u);
10 sampling = T(3)-T(2); // sampling time
11 index = round((period)/sampling); // calculating the
           duration of last cycle of waveform
12 times=T($-index:$);
13 temp = y($-index:$); // output for last cycle
14 heater = u($-index:$); // input for last cycle
15 [max_heater , pointer1] = max(heater); // determining max
           amplitude and index for last cycle of input (index
           is relative to last cycle)
16 [max_temp , pointer2] = max(temp); // determining max
           amplitude and index for last cycle of input (index

```

```

        is relative to last cycle)
17 pointer1 = pointer1 + (p-index); // conversion of index
    for input in terms of complete data period
18 pointer2 = pointer2 + (p-index); // conversion of index
    for output in terms of complete data period
19 Amplitude_ratio_dB = 20*log10(y(pointer2)/u(pointer1))
    // To find gain in dB
20 Phase_difference = 360*f*(pointer1-pointer2)*sampling
    // phase difference in degrees
21 // Phase_difference = -((pointer1 - pointer2) / (1/f)) * 360
22
23 plot2d(T,[u y]);
24 label('Plot of sine input in heater and the
    corresponding temperature profile',4,'Time (s)','
    Change in temperature and heater',4);
25 // legend(['Heater';'Temperature']);

```

Scilab Code 6.5 label.sci

```

1 // Updated (9-12-06), written by Inderpreet Arora
2 // Input arguments : title , xlabel , ylabel and their
    font sizes
3 function label(tname,tfont,labelx,labely,xyfont)
4 a = get("current_axes")
5 xtitle(tname,labelx,labely)
6 xgrid
7 t = a.title;
8 t.font_size = tfont; // Title font size
9 t.font_style = 2; // Title font style
10 t.text = tname;
11 u = a.xlabel;
12 u.font_size = xyfont; // Label font size
13 u.font_style = 2; // Label font style
14 v = a.ylabel;
15 v.font_size = xyfont; // Label font size
16 v.font_style = 2; // Label font style
17 // a.label_font_size = 3;
18 endfunction;

```

Scilab Code 6.6 bodeplot.sce

```
1 // bodeplot
2 exec('labelbode.sci');
3 x=[0.001,0.0035,0.004,0.005,0.006,0.007,...  
4 0.008,0.009,0.01,0.02,0.03,0.04]; // Input frequency (Hz)
5 y=[-3.87,-5.67,-7.53,-7.53,-8.17,-8.64,...  
6 -8.87,-8.90,-9.11,-13.55,-15.39,-16.47]; // Amplitude  
ratio (dB)
7 subplot(2,1,1);
8 plot2d(x,y,rect=[0.001,-20,0.04,0],logflag="ln");
9 xgrid();
10 y=[-25.2,-28.98,-33.11,-41.4,-60.48,-70.56,...  
11 -77.76,-87.48,-90,-129.6,-151.2,-172.8]; // Phase  
difference (degree)
12 title = '';
13 label(title,4,'Hz','Amplitude ratio in dB ',4);
14 subplot(2,1,2);
15 plot2d(x,y,rect=[0.001,-180,0.04,-20],logflag="ln");
16 label(title,4,'','Phase difference ',4);
17 subplot(2,1,2);
18 xgrid();
19
20 // s = poly(0,'s')
21 // h = syslin('c',(0.475/(124.827*s^2+57.26*s+1)));
22 // bode(h,0.001,0.04);
```

Scilab Code 6.7 labelbode.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
8 t = a.title;
9 t.font_size = tfont; // Title font size
10 t.font_style = 2; // Title font style
11 t.text = tname;
12 u = a.x_label;
13 u.font_size = xyfont; // Label font size
14 u.font_style = 2; // Label font style
15 v = a.y_label;
16 v.font_size = xyfont; // Label font size
17 v.font_style = 2; // Label font style
18 // a.label_font_size = 3;
19 endfunction;

```

Scilab Code 6.8 TFbode.sce

```

1 s=poly(0,'s')
2 dt=10; // delay time
3 // h= syslin ('c',((0.510/(65.49*s+1)))); // transfer
   function using first order pade' approximation
4 tf=((0.475/(36*s+1))*((-dt/2)*s+1/(dt/2)*s+1));
5 bode(h,0.001,10);

```

Scilab Code 6.9 comparison.sce

```

1 s=poly(0,'s');
2 frq = [0.001,0.0035,0.004,0.005,0.006,0.007,...
3 0.008,0.009,0.01,0.02,0.03,0.04]; // Input frequency (Hz)
4 dt=10; // delay time
5
6 tf=((0.475/(36*s+1))*((-dt/2)*s+1/(dt/2)*s+1)); // transfer
   function using pade' approximation
7 h=syslin('c',tf);
8
9 [frq1,rep]=repfreq(h,frq);
10 [dB1,phi1]=dbphi(rep);
11 title = 'From actual plant data';
12 dB = [-3.87,-5.67,-7.53,-7.53,-8.17,-8.64,...]

```

```

13 -8.87,-8.90,-9.11,-13.55,-15.39,-16.47]; // Amplitude
      ratio ( dB )
14 phi = [-25.2,-28.98,-33.11,-41.4,-60.48,...
15 -70.56,-77.76,-87.48,-90,-129.6,-151.2,-172.8]; // Phase
      difference ( degree )
16 bode([ freq ],[ dB;dB1 ],[ phi ;phi1 ])
17 legend(['Plant data';'$\frac{0.42}{36s+1}e^{-10s}$'])
18
19 // transfer function using pade approximation

```

Scilab Code 6.10 sine2_virtual.sce

```

1 mode(0);
2 filename= '16Feb2012_17_28_57.txt'; // Enter the data
      file name in single quotes
3 f=0.01; // Enter the frequency
4 data6=fscanfMat(filename);
5 data7=data6(2:$,:);
6 exec('labelbode.sci');
7 T = data7(:,5); fan = data7(:,3); // T is time , fan is
      fan speed
8 u = data7(:,2)-data7(1,2); y = data7(:,4)-data7(1,4);
      // u is current , y is temperature
9 period=ceil(1/f);
10 p=length(u);
11 sampling = T(3)-T(2); // sampling time
12 sampling = sampling/1000;
13 index = round((period)/sampling); // calculating the
      duration of last cycle of waveform
14 times=T($-index:$);
15 temp = y($-index:$); // output for last cycle
16 heater = u($-index:$); // input for last cycle
17 [max_heater ,pointer1] = max(heater); // determining max
      amplitude and index for last cycle of input (index
      is relative to last cycle)
18 [max_temp ,pointer2] = max(temp); // determining max
      amplitude and index for last cycle of input (index
      is relative to last cycle)

```

```

19 pointer1 = pointer1 + (p-index); // conversion of index
   for input in terms of complete data period
20 pointer2 = pointer2 + (p-index); // conversion of index
   for output in terms of complete data period
21 Amplitude_ratio_dB = 20*log10(y(pointer2)/u(pointer1))
   // To find gain in dB
22 Phase_difference = 360*f*(pointer1-pointer2)*sampling
   // phase difference in degrees
23 // Phase_difference = -((pointer1 - pointer2) / (1/f)) * 360
24
25 plot2d(T,[u y]);
26 label('Plot of sine input in heater and the
   corresponding temperature profile',4,'Time (s)','
   Change in temperature and heater',4);
27 // legend(['Heater';'Temperature']);

```

Chapter 7

Controlling Single Board Heater System by PID controller

The aim of this experiment is to apply a PID controller to the single board heater system. 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 Fig.7.1. Heater current and fan speed are the two inputs for this system. They are given in PWM units. A provision is made to set the parameters related to PID controller (K, τ_i, τ_d) in Xcos. In this experiment we keep the Fan speed constant. The output temperature profile, 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.

7.1 Theory

A PID controller is one which tries to minimize the error between measured variable and the set point by calculating the error and then putting a suitable corrective action. Note that the output of interest of a process is called the measured variable or process variable, the difference between the set point and the measured variable is called the error and the control action taken to adjust the process is called the manipulated variable. A PID controller does not simply add or subtract the control action but instead it manipulates it using three distinct control features, namely, Proportional, Integral and Derivative. Thus, a PID controller has three separate parameters.

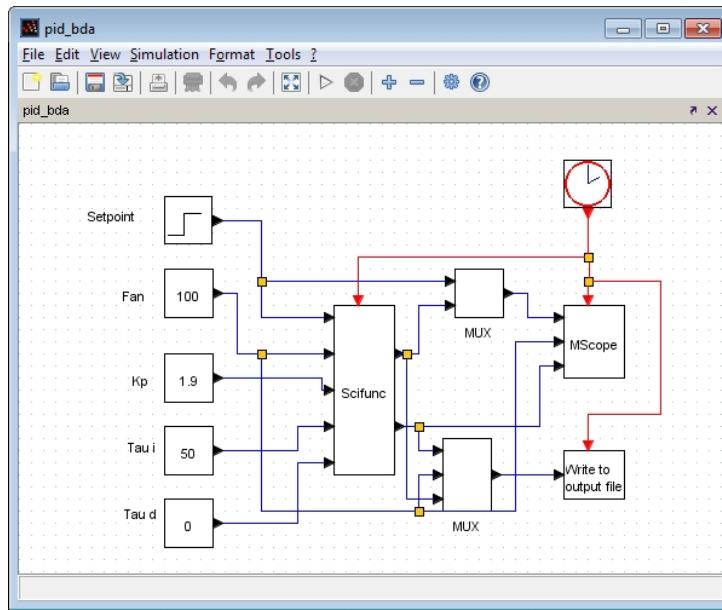


Figure 7.1: Xcos interface for this experiment

7.1.1 Proportional Control Action

This parameter generates a control action based on the current value of the error. In a more simplified sense, if the error is +2, the control action is -2. The proportional action can be generated by multiplying the error with a Proportional constant K_p . Mathematical representation of the same is given below,

$$P = K_p e(t) \quad (7.1)$$

where,

P is the proportional output

K_p is the proportional gain

$e(t)$ is the error signal

The value of K_p is very important. A large value of K_p may lead to instability of the system. In contrast, a smaller value of K_p may decrease the controller's sensitivity towards error. The problem involved in using only Proportional action is that, the control action will never settle down to its target value and will always retain a steady-state error.

7.1.2 Integral Control Action

This parameter generates a control action depending on the history of errors. It means that the action is based on the sum of the recent errors. It is proportional to both the magnitude as well as duration of the error. The summation of the error over a period of time gives a value of the offset that should have been corrected previously. The integral action can thus be generated by multiplying this accumulated error with an integral gain K_i . Mathematical representation of the same is given below.

$$I = K_i \int_0^t e(t)dt \quad (7.2)$$

where,

I is the integral output

K_i is the integral gain ($K_i = K_p/\tau_i$, where, τ_i is the integral time)

The integral action tends to accelerate the control action. However, since it looks only at the past values of the error, there is always a possibility of it causing the present values to overshoot the set point values.

7.1.3 Derivative Control Action

As the name suggests, a derivative parameter generates a control action by calculating the rate of change of error. A derivative action is thus generated by multiplying the value of rate of change of error with a derivative gain K_d . Mathematical representation of the same is given below.

$$D = K_d \frac{d}{dt} e(t) \quad (7.3)$$

where,

D is the derivative output

K_d is the derivative gain ($K_d = K_p/\tau_d$, where, τ_d is the derivative time)

The derivative action slows down the rate of change of the controller output. A derivative controller is quite useful when the error is continuously changing with time. One should, however, avoid using it alone. This is because there is no output when the error is zero and when the rate of change of error is constant.

When all the above control actions are summed up and used together, the final

equation becomes

$$PID = K_p e(t) + K_i \int_0^t e(t) dt + K_d \frac{d}{dt} e(t) \quad (7.4)$$

The above equation represents an ideal form of PID controller. This means that the integral controller was used independently. However, it is not a good decision since, the integral action begins only after the error exists for some amount of time. The proportional controller however begins as soon as the error starts existing. Hence, the integral controller is often used in conjunction with a proportional controller. This is popularly known as PI controller and the equation for Proportional Integral action becomes,

$$PI = K_p e(t) + \left(K_p / \tau_i \right) \int_0^t e(t) dt \quad (7.5)$$

$$= K_p \left\{ e(t) + (1 / \tau_i) \int_0^t e(t) dt \right\} \quad (7.6)$$

Similarly, as discussed before, independent use of derivative controller is also not desirable. Moreover, if the process contains high frequency noise then the derivative action will tend to amplify the noise. Hence, derivative controller is also used in conjunction with Proportional or Proportional Integral controller popularly known as PD or PID, respectively. Therefore the equation for Proportional Derivative action becomes,

$$PD = K_p e(t) + K_p \tau_d \frac{d}{dt} e(t) \quad (7.7)$$

$$= K_p \left\{ e(t) + \tau_d \frac{d}{dt} e(t) \right\} \quad (7.8)$$

Finally, writing the equation for PID controller,

$$PID = K \left\{ e(t) + \frac{1}{\tau_i} \int_0^t e(t) dt + \tau_d \frac{d}{dt} e(t) \right\} \quad (7.9)$$

7.2 Ziegler-Nichols Rule for Tuning PID Controllers

There are many rules to tune a PID controller. We shall see the two popular methods suggested by Ziegler-Nichols.

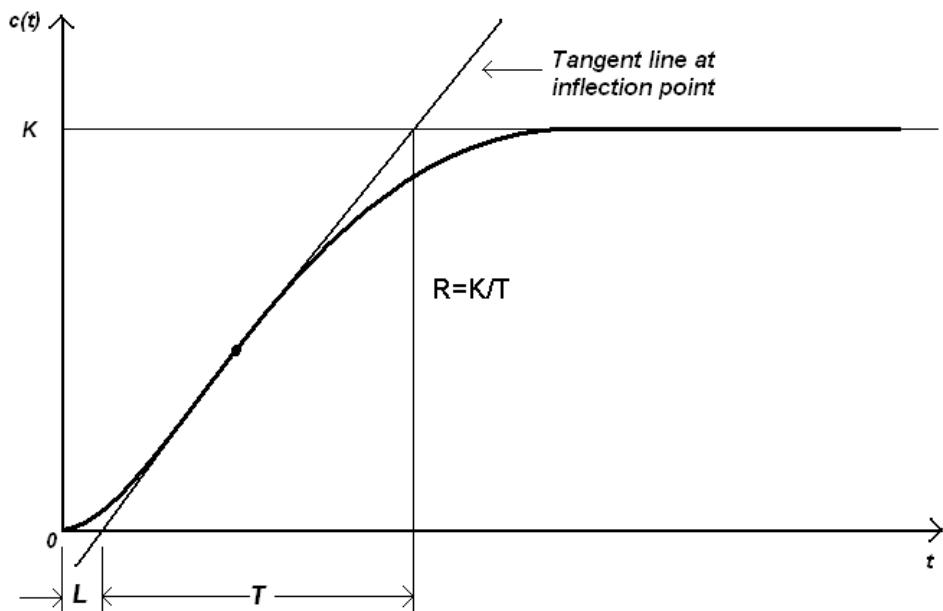


Figure 7.2: Reaction Curve[5]

7.2.1 First Method

Ziegler-Nichols give rule for determining the values of gain K , integral time τ_i and derivative time τ_d based on the step response characteristics of a given plant. In this method one can experimentally obtain the response of a plant to a step input, as shown in figure 7.2. This method is applicable only when the response to the step input exhibits S-shaped curve.[7]

As shown in figure 7.2, by drawing the tangent line at the inflection point and determining the intersection of the tangent line with the time axis and the line $c(t) = K$, we get two constants, namely, delay time L and time constant T .

Ziegler and Nichols suggested to set the values of K, τ_i, τ_d according the formula shown in table 7.1. Notice that the PID controller tuned by the Ziegler-

Type of controller	K	τ_i	τ_d
P	$\frac{1}{RL}$	∞	0
PI	$\frac{0.9}{RL}$	$3L$	0
PID	$\frac{1.2}{RL}$	$2L$	$0.5L$

Table 7.1: Ziegler-Nichols tuning rule based on step response of plant

Nichols rule gives,

$$G_c(s) = K_p \left(1 + \frac{1}{T_i s} + T_d s \right) \quad (7.10)$$

$$= 1.2 \frac{T}{L} \left(1 + \frac{1}{2Ls} + 0.5Ls \right) \quad (7.11)$$

$$= 0.6T \frac{\left(s + \frac{1}{L} \right)^2}{s} \quad (7.12)$$

Thus the PID controller has a pole at the origin and double zeros at $s = -1/L$.

7.2.2 Second Method

The second method is also known as ‘instability method’[5]. This is a closed loop method in which the Integral and Derivative gains of the PID controller are made zero with a unity value for proportional gain. A setpoint change is made and the temperature profile is observed for some time. The temperature would most likely maintain a steady-state with some offset. The gain is increased to a next distinct value (say 2) with a change in the setpoint. The procedure is repeated until the temperature first varies with sustained oscillations. It is necessary that the output (temperature) should have neither under damped nor over damped oscillations. At this particular frequency of sustained oscillations, the corresponding value of K_p is noted and is called as the critical gain K_{cr} . The corresponding period of oscillation is known as P_{cr} . Refer Fig. 7.3.

The various P, PI and PID parameters are then calculated with the help of table 7.2.

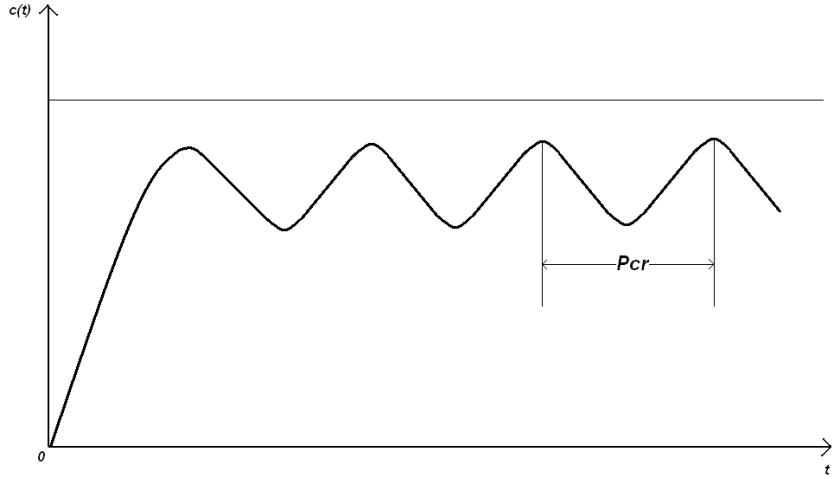


Figure 7.3: Ziegler-Nichols instability tuning method

Type of controller	K	τ_i	τ_d
P	$0.5K_u$	∞	0
PI	$0.45K_u$	$\frac{1}{0.2}P_u$	0
PID	$0.6K_u$	$0.5P_u$	$0.125P_u$

Table 7.2: Ziegler-Nichols tuning rule for instability tuning method

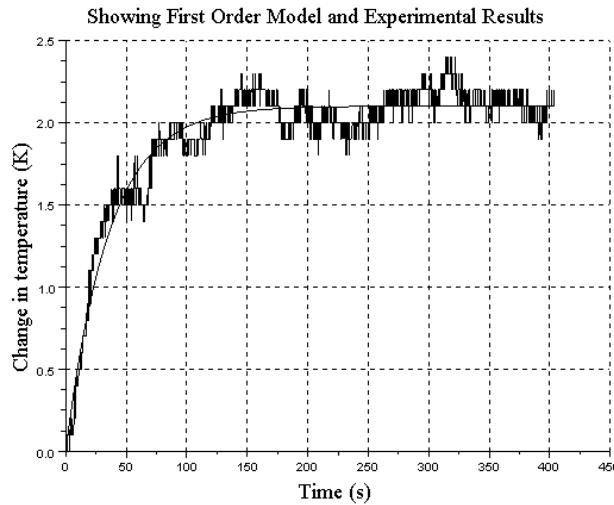


Figure 7.4: Refer ‘Step Test’ experiment[6]

Using the Ziegler-Nichols first method explained earlier, the following values were obtained. Refer figure 7.4.

$$L = 6 \text{ s}$$

$$T = 193 \text{ s}$$

For PI

$$K = 6.031$$

$$\tau_i = 18$$

For PID

$$K = 8$$

$$\tau_i = 12$$

$$\tau_d = 3$$

While performing the experiment fine tuning of K, τ_i, τ_d may be required.

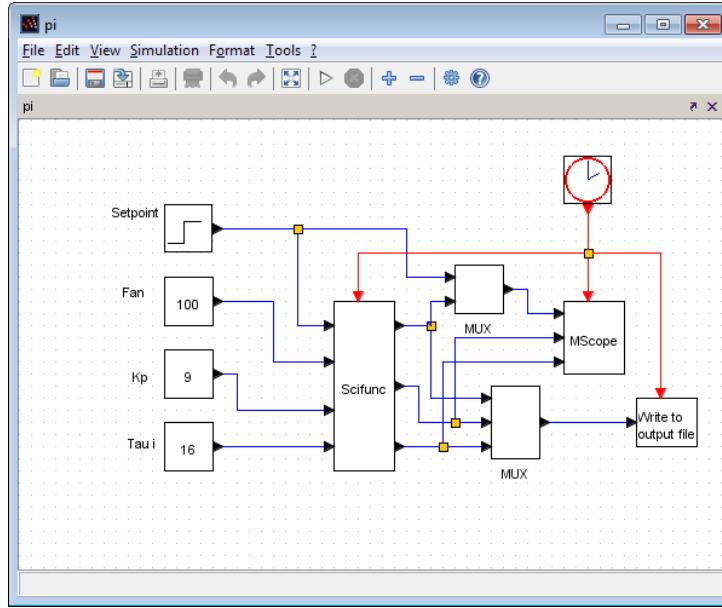


Figure 7.5: Xcos for PI controller available as `pi.xcos`

7.3 Implementing PI controller using Trapezoidal Approximation

Fig.7.5 shows Xcos for implementing PI controller. The PI controller in continuous time is given by

$$u(t) = K \left\{ e(t) + \frac{1}{\tau_i} \int_0^t e(t) dt \right\} \quad (7.13)$$

On taking the Laplace transforms,we obtain

$$u(t) = K \left\{ 1 + \frac{1}{\tau_i s} \right\} e(t) \quad (7.14)$$

By mapping controller given in Eq.7.14 to the discrete time domain using trapezoidal approximation

$$u(n) = K \left\{ 1 + \frac{T_s}{2\tau_i} \frac{z+1}{z-1} \right\} e(n) \quad (7.15)$$

On cross multiplying, we obtain

$$(z - 1)u(n) = K \left\{ (z - 1) + \frac{T_s}{2\tau_i}(z + 1) \right\} e(n) \quad (7.16)$$

We divide by z and then by using shifting theorem we obtain

$$u(n) - u(n - 1) = K \left\{ e(n) - e(n - 1) + \frac{T_s}{2\tau_i}e(n) + \frac{T_s}{2\tau_i}e(n - 1) \right\} \quad (7.17)$$

The PI controller is usually written as

$$u(n) = u(n - 1) + s_0 e(n) + s_1 e(n - 1) \quad (7.18)$$

Where

$$s_0 = K \left(1 + \frac{T_s}{2\tau_i} \right) \quad (7.19)$$

$$s_1 = K \left(-1 + \frac{T_s}{2\tau_i} \right) \quad (7.20)$$

For implementing above PI controller, scilab code is given in `pi_ta.sci` file, listed at the end of this document. Change the current working directory to the folder `pid_controller`. Execute the file `ser_init.sce` with the appropriate com port number and then execute the file `pi_ta.sci` for loading the function. Run the xcos file `pi_ta.xcos`. Output of Xcos is shown in below fig.7.6. Figure shows three plots. First subplot shows Setpoint and output temperature profile. Second sub plot shows control effort and third subplot shows error between set-point and plant output.

7.3.1 Implementing PI controller using Trapezoidal Approximation on SBHS, virtually

The step by step procedure for conducting an experiment virtually is explained in section 3.4. The required .sce file is `pi_ta_virtual.sce`. You will find this file in the `pid_controller` directory under `virtual` folder. The necessary codes are listed in the section 7.9

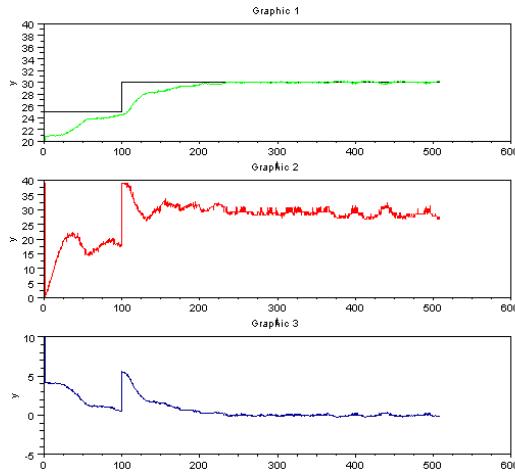


Figure 7.6: PI controller (Trapezoidal Approximation) output

7.4 Implementing PI controller using Backward Difference Approximation

The PI controller in continuous time is given by

$$u(t) = K \left\{ e(t) + \frac{1}{\tau_i} \int_0^t e(t) dt \right\} \quad (7.21)$$

On taking the Laplace transform, we obtain

$$U(s) = K \left\{ 1 + \frac{1}{\tau_i s} \right\} E(s) \quad (7.22)$$

By mapping controller given in Eq.7.22 to the discrete time domain using Backward difference approximation :

$$u(n) = K \left\{ 1 + \frac{T_s}{\tau_i} \frac{z}{z-1} \right\} e(n) \quad (7.23)$$

On cross multiplying, we obtain

$$(z-1)u(n) = K \left\{ (z-1) + \frac{T_s}{\tau_i} (z) \right\} e(n) \quad (7.24)$$

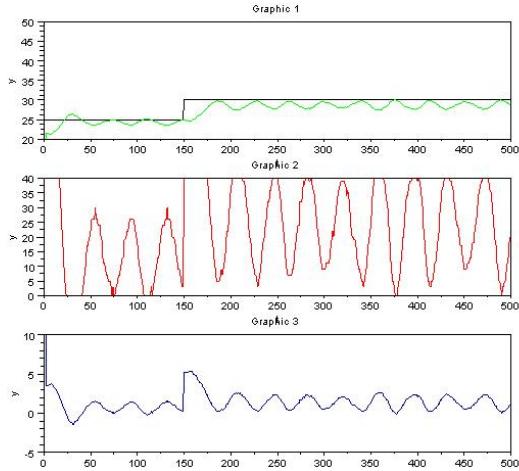


Figure 7.7: PI controller (Backward Difference Approximation) output

We divide by z and then by using shifting theorem we obtain

$$u(n) - u(n-1) = K \left\{ e(n) - e(n-1) + \frac{T_s}{\tau_i} e(n) \right\} \quad (7.25)$$

The PI controller is usually written as

$$u(n) = u(n-1) + s_0 e(n) + s_1 e(n-1) \quad (7.26)$$

Where

$$s_0 = K \left(1 + \frac{T_s}{\tau_i} \right) \quad (7.27)$$

$$s_1 = -K \quad (7.28)$$

For implementing above PI controller scilab code is given in `pi_bda.sci` file, listed at the end of this document. Change the current working directory to the folder `pid_controller`. Execute the file `ser_init.sce` with the appropriate com port number and then execute the file `pi_bda.sci` for loading the function. Run the xcos file `pi_bda.xcos`. Output of Xcos is shown in below fig.7.7 Figure shows three plots. First subplot shows Setpoint and output temperature profile. Second sub plot shows control effort and third subplot shows error between set-point and plant output.

7.4.1 Implementing PI controller using Backward Difference Approximation on SBHS, virtually

The step by step procedure for conducting an experiment virtually is explained in section 3.4. The required .sce file is `pi_bda_virtual.sce`. You will find this file in the `pid_controller` directory under `virtual` folder. The necessary codes are listed in the section 7.9

7.5 Implementing PI controller using Forward Difference Approximation

The PI controller in continuous time is given by

$$u(t) = K \left\{ e(t) + \frac{1}{\tau_i} \int_0^t e(t) dt \right\} \quad (7.29)$$

On taking the Laplace transforms, we obtain

$$u(t) = K \left\{ 1 + \frac{1}{\tau_i s} \right\} e(t) \quad (7.30)$$

By mapping controller given in Eq.7.30 to the discrete time domain using forward difference formula :

$$u(n) = K \left\{ 1 + \frac{T_s}{\tau_i} \frac{1}{z - 1} \right\} e(n) \quad (7.31)$$

On cross multiplying, we obtain

$$(z - 1)u(n) = K \left\{ (z - 1) + \frac{T_s}{\tau_i} \right\} e(n) \quad (7.32)$$

We divide by z and then by using shifting theorem we obtain

$$u(n) - u(n - 1) = K \left\{ e(n) - e(n - 1) + \frac{T_s}{\tau_i} e(n - 1) \right\} \quad (7.33)$$

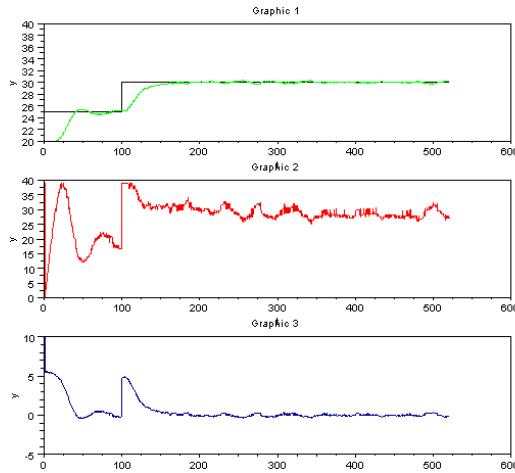


Figure 7.8: PI controller implementation (forward difference approximation)

The PI controller is usually written as

$$u(n) = u(n - 1) + s_0 e(n) + s_1 e(n - 1) \quad (7.34)$$

Where

$$s_0 = K \quad (7.35)$$

$$s_1 = K \left(-1 + \frac{T_s}{\tau_i} \right) \quad (7.36)$$

For implementing above PI controller scilab code is given in `pi_fda.sci` file, listed at the end of this document. Change the current working directory to the folder `pid_controller`. Execute the file `ser_init.sce` with the appropriate com port number and then execute the file `pi_fda.sci` for loading the function. Run the xcos file `pi_fda.xcos`. Output of Xcos is shown in below fig.7.8 Figure shows three plots. First subplot shows Setpoint and output temperature profile. Second sub plot shows control effort and third subplot shows error between set-point and plant output.

7.5.1 Implementing PI controller using forward Difference Approximation on SBHS, virtually

The step by step procedure for conducting an experiment virtually is explained in section 3.4. The required .sce file is `pi_fda_virtual.sce`. You will find this file

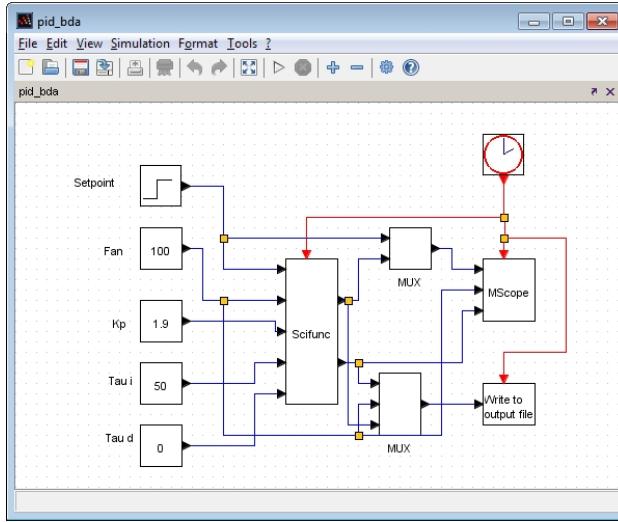


Figure 7.9: Xcos for PID controller available as `pid_bda.xcos`

in the `pid_controller` directory under `virtual` folder. The necessary codes are listed in the section 7.9

7.6 Implementing PID controller using Backward difference approximation

Fig.7.9 shows Xcos for implementing PID controller . The PID controller in continuous time is given by

$$u(t) = K \left\{ e(t) + \frac{1}{\tau_i} \int_0^t e(t) dt + \tau_d \frac{de(t)}{dt} \right\} \quad (7.37)$$

On taking the Laplace transforms,we obtain

$$u(t) = K \left\{ 1 + \frac{1}{\tau_i s} + \tau_d s \right\} e(t) \quad (7.38)$$

By mapping controller given in Eq.7.38 to the discrete time domain using backward difference formula :

$$u(n) = K \left\{ 1 + \frac{T_s}{\tau_i} \frac{z}{z-1} + \frac{\tau_d}{T_s} \frac{z-1}{z} \right\} e(n) \quad (7.39)$$

On cross multiplying, we obtain

$$(z^2 - z)u(n) = K \left\{ (z^2 - z) + \frac{T_s}{\tau_i} z^2 + \frac{\tau_d}{T_s} (z-1)^2 \right\} e(n) \quad (7.40)$$

We divide by z^2 and by using shifting theorem we obtain

$$\begin{aligned} u(n) - u(n-1) &= K \left\{ e(n) - e(n-1) + \frac{T_s}{\tau_i} e(n) \right. \\ &\quad \left. + \frac{\tau_d}{T_s} [e(n) - 2e(n-1) + e(n-2)] \right\} \end{aligned} \quad (7.41)$$

The PID controller is usually written as

$$u(n) = u(n-1) + s_0 e(n) + s_1 e(n-1) + s_2 e(n-2) \quad (7.42)$$

Where

$$s_0 = K \left[1 + \frac{T_s}{\tau_i} + \frac{\tau_d}{T_s} \right] \quad (7.43)$$

$$s_1 = K \left[-1 - 2 \frac{\tau_d}{T_s} \right] \quad (7.44)$$

$$s_2 = K \left[\frac{\tau_d}{T_s} \right] \quad (7.45)$$

For implementing above PID controller scilab code is given in `pid_bda.sci` file, listed at the end of this document. Change the current working directory to the folder `pid_controller`. Execute the file `ser_init.sce` with the appropriate com port number and then execute the file `pid_bda.sci` for loading the function. Run the xcos file `pid_dda.xcos`. Output of Xcos is shown in below fig.7.10. Figure shows three plots. First subplot shows Setpoint and output temperature profile. Second sub plot shows control effort and third subplot shows error between setpoint and plant output.

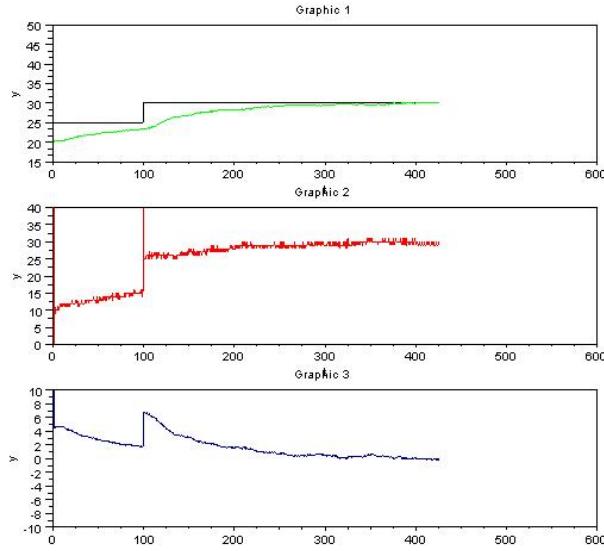


Figure 7.10: PID controller (Backward Difference Approximation) Output

7.6.1 Implementing PID controller using backward Difference Approximation on SBHS, virtually

The step by step procedure for conducting an experiment virtually is explained in section 3.4. The required .sce file is `pid_bda_virtual.sce`. You will find this file in the `pid_controller` directory under `virtual` folder. The necessary codes are listed in the section 7.9

7.7 Implementing PID controller using trapezoidal approximation for integral mode and Backward difference approximation for the derivative mode

The PID controller in continuous time is given by

$$u(t) = K \left\{ e(t) + \frac{1}{\tau_i} \int_0^t e(t) dt + \tau_d \frac{de(t)}{dt} \right\} \quad (7.46)$$

On taking the Laplace transforms, we obtain

$$u(t) = K \left\{ 1 + \frac{1}{\tau_i s} + \tau_d s \right\} e(t) \quad (7.47)$$

By mapping controller given in Eq.7.47 to the discrete time domain using trapezoidal approximation for integral mode and Backward difference approximation for the derivative mode

$$u(n) = K \left\{ 1 + \frac{T_s}{2\tau_i} \frac{z+1}{z-1} + \frac{\tau_d}{T_s} \frac{z-1}{z} \right\} e(n) \quad (7.48)$$

On cross multiplying, we obtain

$$(z^2 - z)u(n) = K \left\{ (z^2 - z) + \frac{T_s}{2\tau_i} (z^2 + z) \frac{\tau_d}{T_s} (z-1)^2 \right\} e(n) \quad (7.49)$$

We divide by z^2 and then by using shifting theorem we obtain

$$\begin{aligned} u(n) - u(n-1) &= K \left\{ e(n) - e(n-1) + \frac{T_s}{2\tau_i} e(n) + e(n-1) \right. \\ &\quad \left. + \frac{\tau_d}{T_s} [e(n) - 2e(n-1) + e(n-2)] \right\} \end{aligned} \quad (7.50)$$

The PID controller is usually written as

$$u(n) = u(n-1) = s_0 e(n) + s_1 e(n-1) + s_2 e(n-2) \quad (7.51)$$

Where

$$s_0 = K \left[1 + \frac{T_s}{2\tau_i} + \frac{\tau_d}{T_s} \right] \quad (7.52)$$

$$s_1 = K \left[-1 + \frac{T_s}{2\tau_i} - 2 \frac{\tau_d}{T_s} \right] \quad (7.53)$$

$$s_2 = K \frac{\tau_d}{T_s} \quad (7.54)$$

For implementing above PID controller scilab code is given in `pid_ta_bda.sci` file, listed at the end of this document. Change the current working directory to

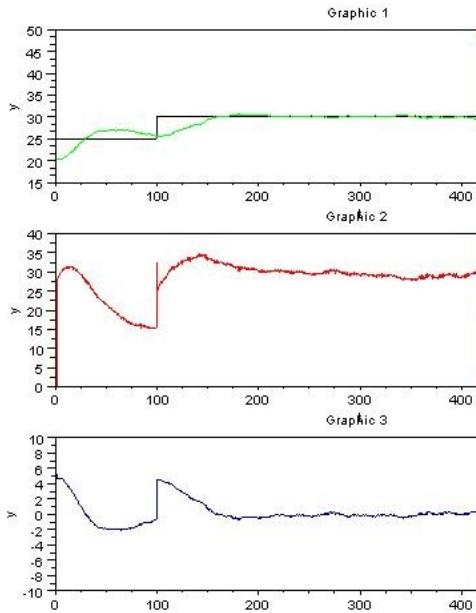


Figure 7.11: PID controller (TA - BDA) implementation

the folder `pid_controller`. Execute the file `ser_init.sce` with the appropriate com port number and then execute the file `pid_ta_bda.sci` for loading the function. Run the xcos file `pid_ta_bda.xcos`. Output of Xcos is shown in below fig.7.11 Figure shows three plots. First subplot shows Setpoint and output temperature profile. Second sub plot shows control effort and third subplot shows error between setpoint plant output i.e. temperature.

7.7.1 Implementing PID controller using trapezoidal approximation for integral mode and Backward difference approximation for the derivative mode on SBHS, virtually

The step by step procedure for conducting an experiment virtually is explained in section 3.4. The required .sce file is `pid_ta_bda_virtual.sce`. You will find this file in the `pid_controller` directory under `virtual` folder. The necessary codes are listed in the section 7.9

Due to the introduction of derivative action control effort shows lots of fluctuations. By using filtered form of PID we can make derivative mode implementable.

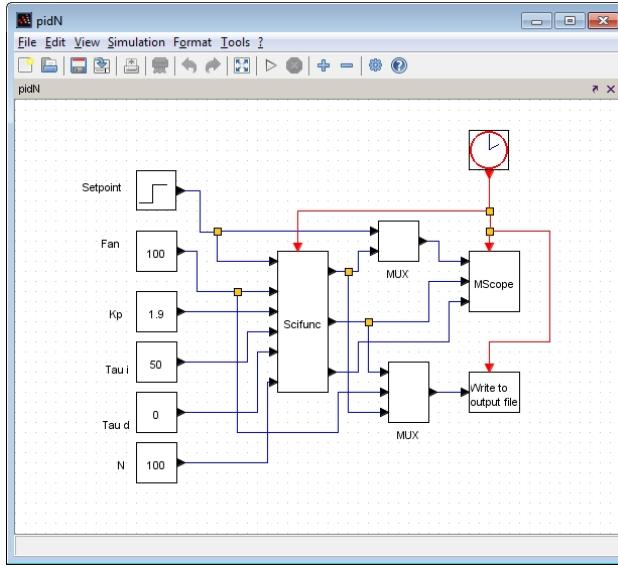


Figure 7.12: Xcos for PID controller with filtering available as pidN.xcos

7.8 Implementing PID controller with filtering using Backward difference approximation

Fig.7.12 shows Xcos for implementing PID controller with filtering

PID filtered form is given by

$$u(t) = K \left\{ 1 + \frac{1}{\tau_i s} + \frac{\tau_d s}{1 + \frac{\tau_d s}{N}} \right\} e(t) \quad (7.55)$$

Where N is large number, of the order of 100. By mapping controller given in Eq.7.55 to the discrete time domain using backward difference formula :

$$u(n) = K \left(1 + \frac{T_s}{\tau_i} \frac{1}{1 - z^{-1}} + \frac{\tau_d (1 - z^{-1})}{1 + \frac{\tau_d (1 - z^{-1})}{N}} \right) e(n) \quad (7.56)$$

$$u(n) = K \left(1 + \frac{T_s}{\tau_i} \frac{1}{1 - z^{-1}} + \frac{Nr_1 (1 - z^{-1})}{1 + r_1 z^{-1}} \right) e(n) \quad (7.57)$$

Where

$$r_1 = -\frac{\frac{\tau_d}{N}}{\frac{\tau_d}{N} + T_s} \quad (7.58)$$

On cross multiplying, we obtain

$$\begin{aligned} (1 - z^{-1})(1 + r_1 z^{-1})u(n) &= K[(1 - z^{-1})(1 + r_1 z^{-1}) \\ &\quad + \frac{T_s}{\tau_i}(1 + r_1 z^{-1}) + \frac{\tau_d}{T_s}(1 - z^{-1})^2]e(n) \end{aligned} \quad (7.59)$$

Simplifying and then by using shifting theorem we obtain

$$\begin{aligned} u(n) + (r_1 - 1)u(n - 1) \\ -r_1 u(n - 2) &= K \left[1 + \frac{T_s}{\tau_i} - Nr_1 \right] e(n) \\ &\quad + K \left[r_1 \left(1 + \frac{T_s}{\tau_i} + 2N \right) - 1 \right] e(n - 1) \\ &\quad - K [r_1(1 + N)] e(n - 2) \end{aligned} \quad (7.60)$$

hence

$$\begin{aligned} u(n) &= r_1 u(n - 2) - (r_1 - 1)u(n - 1) \\ &\quad + s_0 e(n) + s_1 e(n - 1) + s_2 e(n - 2) \end{aligned} \quad (7.61)$$

Where

$$s_0 = K \left[1 + \frac{T_s}{\tau_i} - Nr_1 \right] \quad (7.62)$$

$$s_1 = K \left[r_1 \left(1 + \frac{T_s}{\tau_i} + 2N \right) - 1 \right] \quad (7.63)$$

$$s_2 = -K [r_1(1 + N)] \quad (7.64)$$

For implementing above PID controller scilab code is given in `pid_filter.sci` file, listed at the end of this document. Change the current working directory to the folder `pid_controller`. Execute the file `ser_init.sce` with the appropriate com port number and then execute the file `pid_filter.sci` for loading the function. Run the xcos file `pidN.xcos`. Output of Xcos is shown in below fig.7.13

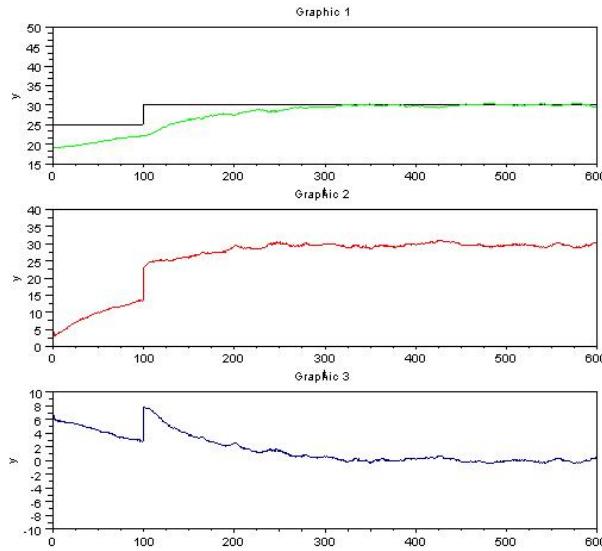


Figure 7.13: PID controller (with filtering) implementation

Figure shows three plots. First subplot shows Setpoint and output temperature profile. Second sub plot shows control effort and third subplot shows error between setpoint and plant output. By comparing fig.7.10 and fig.7.13 it is clear that introduction of filtered form of PID reduces fluctuations in control effort.

7.8.1 Implementing PID controller with filtering using Backward difference approximation on SBHS, virtually

The step by step procedure for conducting an experiment virtually is explained in section 3.4. The required .sce file is `pid_filter_virtual.sce`. You will find this file in the `pid_controller` directory under `virtual` folder. The necessary codes are listed in the section 7.9

7.9 Scilab Code

7.9.1 Scilab code for serial communication

Scilab Code 7.1 `ser_init.sci` used for serial communication

```

1 // To load the serial toolbox and open the serial port
2 exec loader.sce
3
4 handl = openserial(6,"9600,n,8")
5
6 // the order is : port number , "baud , parity , databits ,
7 // 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 if (ascii(handl) ~= [])
12   disp("COM Port Opened");
13 end

```

7.9.2 Scilab code for PI controller

Scilab Code 7.2 pi_ta.sci

```

1 mode(0);
2 // PI Controller using trapezoidal approximation .
3 // Heater input is passed as input argument to
   introduce control effort u(n)
4 // Fan input is passed as input argument which is kept
   at constant level
5 // Range of Fan input : 20 to 252
6 // Temperature is read
7
8 function [temp,C0,e_new] = pi_ta(setpoint,disturbance,
   K,Ti)
9
10 global temp heat_in fan_in C0 u_old u_new e_old e_new
11
12 e_new = setpoint - temp;
13
14 Ts=0.4;
15 S0=K(1+Ts/(2*Ti));

```

```

16 S1=K*(-1+(Ts/(2*Ti)));
17 u_new = u_old+(S0*e_new)+(S1*e_old);
18
19
20 if u_new> 39
21     u_new = 39;
22 end;
23
24 if u_new< 0
25     u_new = 0;
26 end;
27
28 C0=u_new;
29 heat_in = C0;
30 fan_in = disturbance;
31 u_old = u_new;
32 e_old = e_new;
33
34 writeserial(handl,ascii(254)); // heater
35 writeserial(handl,ascii(heat_in));
36 writeserial(handl,ascii(253));
37 writeserial(handl,ascii(fan_in));
38 writeserial(handl,ascii(255));
39 sleep(1);
40 temp = ascii(readserial(handl,2));
41 temp = temp(1) + 0.1*temp(2);
42
43 endfunction;

```

Scilab Code 7.3 pi_bda.sci

```

1 mode(0);
2 // PI Controller using backward difference formula
3 // Heater input is passed as input argument to
   introduce control effort u(n)
4 // Fan input is passed as input argument which is kept
   at constant level
5 // Range of Fan input : 20 to 252

```

```

6 // Temperature is read
7
8 function [temp,heat_in,e_new] = pi_bda(setpoint,
9 disturbance,K,Ti)
10
11 global temp heat_in fan_in C0 u_old u_new e_old e_new
12
13 e_new = setpoint - temp;
14
15 Ts=0.5;
16 S0=K(1+(Ts/Ti));
17 S1=-K;
18
19
20
21 u_new = u_old+ S0*e_new+ S1*e_old;
22
23
24 u_old = u_new;
25 e_old = e_new;
26
27 heat_in = u_new;
28 fan_in = disturbance;
29
30 if heat_in >100
31     heat_in = 100;
32 elseif heat_in < 0
33     heat_in = 0;
34 end;
35
36 if fan_in >100
37     fan_in = 100;
38 elseif fan_in < 0
39     fan_in = 0;
40 end;
41
42 writeserial(handl,ascii(254)); // heater

```

```

43 writeserial(handl, ascii(heat_in));
44 writeserial(handl, ascii(253));
45 writeserial(handl, ascii(fan_in));
46 writeserial(handl, ascii(255));
47 sleep(1);
48 temp = ascii(readserial(handl,2));
49 temp = temp(1) + 0.1*temp(2);
50
51 endfunction;

```

Scilab Code 7.4 pi_fda.sci

```

1 mode(0);
2 // PI Controller using forward difference formula
3 // Heater input is passed as input argument to
   introduce control effort u(n)
4 // Fan input is passed as input argument which is kept
   at constant level
5 // Range of Fan input : 20 to 252
6 // Temperature is read
7
8 function [temp,C0,e_new] = pi_fda(setpoint,disturbance
   ,K,Ti)
9
10 global temp heat_in fan_in C0 u_old u_new e_old e_new
11
12 e_new = setpoint - temp;
13
14 Ts=0.4;
15 S0=K*(1+((Ts/Ti)));
16 S1=-K;
17 u_new = u_old+(S0*e_new)+(S1*e_old);
18
19
20 if u_new> 39;
21   u_new = 39;
22 end;
23

```

```

24 if u_new < 0;
25   u_new = 0;
26 end;
27
28 C0=u_new;
29 heat_in = C0;
30 fan_in = disturbance;
31 u_old = u_new;
32 e_old = e_new;
33
34 writeserial(handl,ascii(254)); // heater
35 writeserial(handl,ascii(heat_in));
36 writeserial(handl,ascii(253));
37 writeserial(handl,ascii(fan_in));
38 writeserial(handl,ascii(255));
39 sleep(1);
40 temp = ascii(readserial(handl,2));
41 temp = temp(1) + 0.1*temp(2);
42
43 endfunction;

```

7.9.3 Scilab code for PID controller

Scilab Code 7.5 pid_bda.sci

```

1 mode(-1);
2 function [temp,heat,et] = pid_bda(setpoint,fan,K,Ti,Td
3 )
3 global temp heat_in fan_in C0 u_old u_new e_old e_new
4   e_old_old
4
5
6 e_new = setpoint - temp;
7
8 Ts=0.5;
9
10 S0=K*(1+(Ts/Ti)+(Td/Ts));
11 S1=K*(-1-((2*Td)/Ts));

```

```

12 S2=K*(Td/Ts);
13
14 u_new = u_old + S0*e_new + S1*e_old + S2*e_old_old;
15
16 u_old = u_new;
17 e_old_old = e_old;
18 e_old = e_new;
19
20
21 heat = u_new;
22
23 if heat >40
24     heat = 40;
25 elseif heat < 0
26     heat = 0;
27 end;
28
29 writeserial(handl,ascii(254)); // heater
30 writeserial(handl,ascii(heat));
31 writeserial(handl,ascii(253));
32 writeserial(handl,ascii(fan));
33 writeserial(handl,ascii(255));
34 sleep(1);
35 temp = ascii(readserial(handl,2));
36 temp = temp(1) + 0.1*temp(2);
37 endfunction;

```

Scilab Code 7.6 pid_ta_bda.sci

```

1 mode(0);
2 // PID Controller using trapezoidal approximation for
   the integral mode and
3 // backward difference formula for derivative mode .
4 // Heater input is passed as input argument to
   introduce control effort u_new .
5 // Fan input is passed as input argument which is kept
   at constant level
6 // Range of Fan input : 60 to 252

```

```

7 // Temperature is read
8
9 function [temp,CO,et] = pid_ta_bda(setpoint,
10 disturbance,K,Ti,Td)
11 global temp heat_in fan_in et SP CO eti u_old u_new
12 e_old e_new e_old_old
13
14 Ts=1.4;
15 S0=K*(1+(Ts/(2*Ti))+(Td/Ts));
16 S1=K*(-1+(Ts/(2*Ti))-(2*Td/Ts));
17 S2=(K*Td/Ts);
18 u_new = u_old + S0*e_new + S1*e_old + S2*e_old_old;
19 et = setpoint - temp;
20 CO = u_new;
21
22 if CO>39
23     CO = 39;
24 end;
25
26 if CO<0
27     CO = 0;
28 end;
29
30 u_new = CO;
31
32 u_old = u_new;
33 e_old_old = e_old;
34 e_old = e_new;
35
36
37 heat_in = CO;
38 fan_in = disturbance;
39
40 writeserial(handl,ascii(254)); // heater
41 writeserial(handl,ascii(heat_in));
42 writeserial(handl,ascii(253));

```

```

43 writeserial(handl,ascii(fan_in));
44 writeserial(handl,ascii(255));
45 sleep(1);
46 temp = ascii(readserial(handl,2));
47 temp = temp(1) + 0.1*temp(2);
48 endfunction;

```

Scilab Code 7.7 pid_filter.sci

```

1 mode(0);
2 // PID Controller (with filtering)
3 // Heater input is passed as input argument to
   introduce control effort u_new.
4 // Fan input is passed as input argument which is kept
   at constant level
5 // Range of Fan input : 60 to 252
6 // Temperature is read
7
8 function [temp,CO,et] = pid(setpoint,disturbance,K,Ti,
   Td,N)
9 global temp heat_in fan_in et SP CO eti u_old
   u_old_old u_new e_old e_new e_old_old
10
11 e_new = setpoint - temp;
12
13 Ts=0.5;
14
15 r1=-(Td/N)/((Td/N)+Ts));
16
17 S0=K*(1+(Ts/Ti)-(N*r1));
18 S1=K*((r1*(1+(Ts/Ti)+(2*N))-1));
19 S2=-K*r1*(1+N);
20
21 u_new = r1*u_old_old -(r1-1)*u_old + S0*e_new + S1*
   e_old + S2*e_old_old;
22 et = setpoint - temp;
23 CO = u_new;
24

```

```

25     if CO>39
26         CO = 39;
27     end ;
28
29     if CO<0
30         CO = 0;
31     end ;
32
33 u_new = CO;
34
35 u_old = u_new;
36 e_old_old = e_old ;
37 e_old = e_new ;
38
39
40 heat_in = CO;
41 fan_in = disturbance ;
42
43 writeserial(handl , ascii(254)); // heater
44 writeserial(handl , ascii(heat_in));
45 writeserial(handl , ascii(253));
46 writeserial(handl , ascii(fan_in));
47 writeserial(handl , ascii(255));
48 sleep(1);
49 temp = ascii(readserial(handl ,2));
50 temp = temp(1) + 0.1*temp(2);
51 endfunction ;

```

Scilab Code 7.8 pid_bda_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 fncrefnfcw m err_count y
5
6 fncre = 'scilabread.sce';
7 fdt = mopen(fncre);

```

```

8 mseek(0);
9
10 err_count = 0; // initialising error count for network
11 error
12 m = 1;
13 exec ("pid_bda_virtual.sci");
14 A = [0.1,m,0,100];
15 fdfh = file ('open','scilabwrite.sce','unknown');
16 write(fdfh,A,'(7e11.5,1x)');
17 file ('close',fdfh);
18 sleep(2000);
19 a = mgetl(fdt,1);
20 mseek(0);
21 if a~= [] // open xcos only if communication is
22 through (ie reply has come from server)
23 xcos('pid_bda_virtual.xcos');
24 else
25 disp("NO NETWORK CONNECTION!");
26 return
27 end

```

Scilab Code 7.9 pid_bda_virtual.sci

```

1 mode(0);
2 // PI Controller using trapezoidal approximation .
3 // Heater input is passed as input argument to
   introduce control effort u(n)
4 // Fan input is passed as input argument which is kept
   at constant level
5 // Range of Fan input : 20 to 252
6 // Temperature is read
7
8 function [temp,heat,e_new,stop] = pid_bda_virtual(
   setpoint,disturbance,K,Ti,Td)
9
10 global temp heat fan C0 u_old u_new e_old e_new
    e_old_old fdfh fdt fncc fnccw m err_count stop
11

```

```

12 fncre = 'scilabread.sce'; // file to be read -
   temperature
13 fncrew = 'scilabwrite.sce'; // file to be written
   - heater , fan
14
15 a = mgetl(fdt,1);
16 b = evstr(a);
17 byte = mtell(fdt);
18 mseek(byte,fdt,'set');
19
20 if a~= []
21     temp = b(1,4); heats = b(1,2);
22     fans = b(1,3); y = temp;
23
24 e_new = setpoint - temp;
25
26
27 Ts=1;
28 S0=K*(1+(Ts/Ti)+(Td/Ts));
29 S1=K*(-1-((2*Td)/Ts));
30 S2=K*(Td/Ts);
31
32 u_new = u_old + S0*e_new + S1*e_old + S2*e_old_old;
33
34
35
36 if u_new> 100
37     u_new = 100;
38 end;
39
40 if u_new< 0
41     u_new = 0;
42 end;
43
44
45 heat=u_new;
46 fan = disturbance;
47

```

```

48 if fan> 100
49     fan = 100;
50 end ;
51
52 if fan< 0
53     fan = 0;
54 end ;
55
56 u_old = u_new;
57 e_old_old = e_old ;
58 e_old = e_new ;
59
60 A = [m,heat ,fan ,m];
61 fdfh = file('open','scilabwrite.sce','unknown');
62 file('last',fdfh)
63 write(fdfh ,A,'(7(e11.5,1x))');
64 file('close',fdfh);
65 m = m+1;
66
67 else
68     y = 0;
69     err_count = err_count + 1; // counts the no of
       times network error occurs
70     if err_count > 300
71         disp("NO NETWORK COMMUNICATION!");
72         stop = 1; // status set for stopping simulation
73     end
74 end
75
76 return
77 endfunction

```

Chapter 8

Implementing ‘Two Degrees of Freedom’Controller for First order systems on a Single Board Heater System

The aim of this experiment is to implement a 2DOF controller on a single board heater system. 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 Fig.8.1. Fan speed and Heater current are the two inputs to the system. For this experiment, the heater current is used as a control effort generated by inputting the various 2-DOF controller parameters like R_c , S_c , T_c and gamma. The fan input could be thought of as an external disturbance.

8.1 Theory

Degree of freedom as far as the control theory is concerned is the number of parameters on which the plant is no more dependent or the number of parameters that are free to vary. This means that a higher degree of freedom controller makes the plant less susceptible to disturbances. Controllers are broadly classified as feedback and feed forward controllers. Feedback controllers are further classified as One Degree of Freedom controller and Two Degree of Freedom controller. Feed forward controllers are those who take the control action before a disturbance dis-

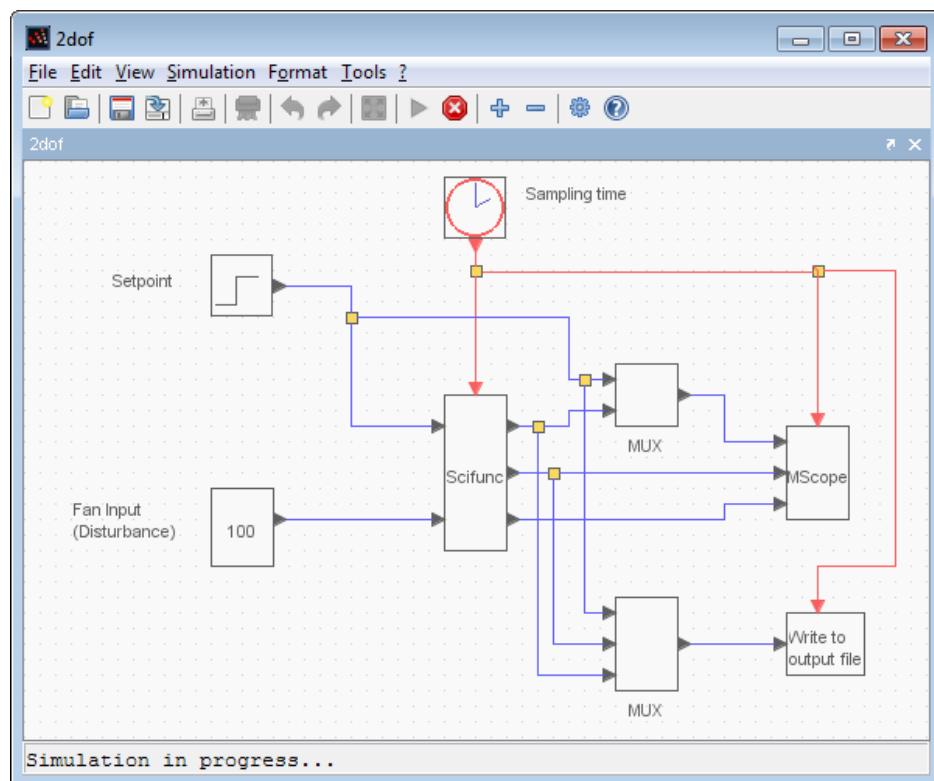


Figure 8.1: Xcos interface for this experiment

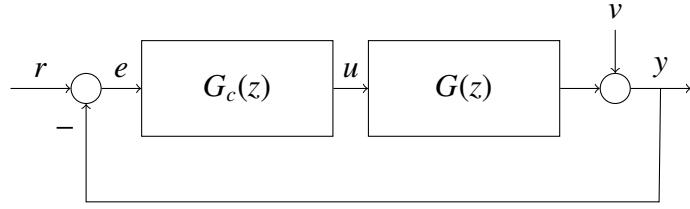


Figure 8.2: Feed back control strategy

turbs the plant. But this implies an ability to sense the disturbance. Moreover, exact knowledge about the plant is also needed. Nevertheless, due to these restrictions, it is rarely used alone. A feedback control strategy is as shown in figure 8.2. The reference and the output is continuously compared to generate error which is fed to the controller to take the appropriate control action. Here, exact knowledge about the plant, $G(z)$ and the disturbance, v is not necessary. Solving for $y(n)$, we get

$$y(n) = \frac{G(z)G_c(z)}{1 + G(z)G_c(z)}r(n) + \frac{1}{1 + G(z)G_c(z)}v(n) \quad (8.1)$$

let,

$$T(z) = \frac{G(z)G_c(z)}{1 + G(z)G_c(z)} \quad (8.2)$$

$$S(z) = \frac{1}{1 + G(z)G_c(z)} \quad (8.3)$$

this implies

$$y(n) = T(z)r(n) + S(z)v(n) \quad (8.4)$$

Here it could be seen that the controller has to track the reference input as well as eliminate the effect of external disturbance. But, however from the above equation it could be seen that

$$S + T = 1 \quad (8.5)$$

Hence it is not possible to achieve both of the requirements, simultaneously in this particular control arrangement. This control arrangement is called One Degree

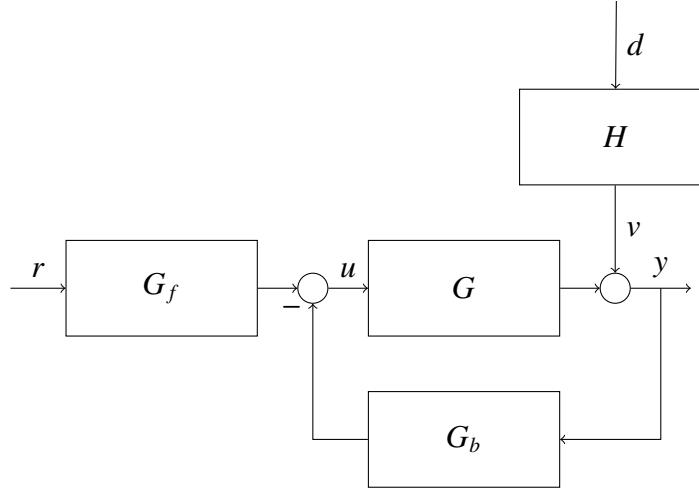


Figure 8.3: 2DOF Feed back control strategy

of Freedom, abbreviated as 1-DOF. A Two Degrees of Freedom strategy is as shown in figure 8.3. Here, G_b and G_f together constitute the controller. G_b is in the feedback path and is used to eliminate the effect of disturbances, whereas, G_f is in the feed forward path and is used to help the output track reference input. We need a control law something of the form,

$$R_c(z)u(n) = T_c(z)r(n) - S_c(z)y(n) \quad (8.6)$$

The terms R_c , S_c and T_c are all in polynomials of z^{-1} .

It could be seen that,

$$G_b = \frac{S_c}{R_c} \quad (8.7)$$

and

$$G_f = \frac{T_c}{R_c} \quad (8.8)$$

Consider a plant with model

$$A(z)y(n) = z^{-k}B(z)u(n) + v(n) \quad (8.9)$$

Substituting equation 8.6 in equation 8.9, we get

$$Ay(n) = z^{-k} \frac{B}{R_c} [T_c r(n) - S_c y(n)] + v(n) \quad (8.10)$$

solving for $y(n)$, we get

$$\left(\frac{R_c A + z^{-k} B S_c}{R_c} \right) y(n) = z^{-k} \frac{B T_c}{R_c} r(n) + v(n) \quad (8.11)$$

This can also be written as

$$y(n) = z^{-k} \frac{B T_c}{\phi_{cl}} r(n) + \frac{R_c}{\phi_{cl}} v(n) \quad (8.12)$$

where

$$\phi_{cl} = R_c(z)A(z) + z^{-k}B(z)S_c(z) \quad (8.13)$$

and is known as the closed-loop characteristic polynomial.

Now, we want the following conditions to be satisfied.

1. The zeros of ϕ_{cl} should be inside the unit circle, so that the closed-loop system becomes stable.
2. The value of $z^{-k} \frac{B T_c}{\phi_{cl}}$ must be close to unity so that reference tracking is achieved
3. The value of $\frac{R_c}{\phi_{cl}}$ must be as small as possible to achieve disturbance rejection

We would now see the pole placement controller approach to design a 2DOF controller.[5]

8.2 Designing 2-DOF controller using pole placement control approach

A 2DOF pole placement controller is as shown in the figure 8.4 It should be noted that the effect of external disturbance will not be considered for this section. We

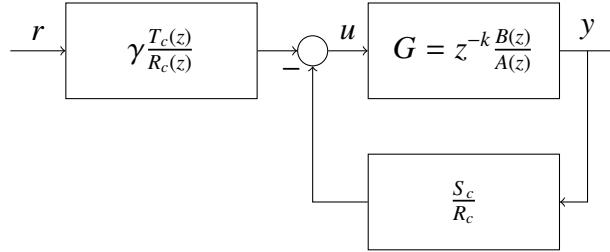


Figure 8.4: 2-DOF pole placement controller

want the closed loop transfer function to behave in such a way so that the output y is related to the setpoint r in the following manner

$$Y_m(z) = \gamma z^{-k} \frac{B_r}{\phi_{cl}} R(z) \quad (8.14)$$

Here, $Y_m(z)$ means the model output. ϕ_{cl} is nothing but the closed loop characteristic polynomial obtained by the desired location analysis.

The value of gamma is chosen in such a way so that at steady-state the output of the model is equal to the setpoint.

$$\gamma = \frac{\phi_{cl(1)}}{B_r(1)} \quad (8.15)$$

Simplifying the block diagram shown in figure 8.4 yields

$$Y = \gamma z^{-k} \frac{BT_c}{AR_c + z^{-k} BS_c} R \quad (8.16)$$

Here we have dropped the argument of z for convenience

On comparing equation 8.14 and 8.16 we can see that

$$\frac{BT_c}{AR_c + z^{-k} BS_c} = \frac{B_r}{\phi_{cl}} \quad (8.17)$$

Here after factorization of the LHS we can expect some cancellations between the numerator and the denominator thereby making the $\deg B_r < \deg B$. But the cancellations ,if any, must be between *stable* poles and zeros. One should avoid the cancellation of an unstable pole with a zero.

Hence, we differentiate the factors as *good* and *bad* factors. Therefore we write A and B as

$$A = A^g A^b \quad (8.18)$$

$$B = B^g B^b \quad (8.19)$$

We also split R_c, S_c and T_c as shown

$$R_c = B^g R_1 \quad (8.20)$$

$$S_c = A^g S_1 \quad (8.21)$$

$$T_c = A^g T_1 \quad (8.22)$$

Hence, the equation 8.17 becomes

$$\frac{B^g B^b A^g T_1}{A^g A^b B^g R_1 + z^{-k} B^g B^b A^g S_1} = \frac{B_r}{\phi_{cl}} \quad (8.23)$$

After appropriate cancellations, we obtain

$$\frac{B^b T_1}{A^b R_1 + z^{-k} B^b S_1} = \frac{B_r}{\phi_{cl}} \quad (8.24)$$

Equating the LHS and RHS of equation 8.24 we obtain

$$B^b T_1 = B_r \quad (8.25)$$

$$A^b R_1 + z^{-k} B^b S_1 = \phi_{cl} \quad (8.26)$$

Equation 8.26 is known as the aryabhatta's identity and can be used to solve for R_1 and S_1 . There are many options to choose for the value of T_1 . By choosing T_1 to be equal to S_1 the 2-DOF controller is reduced to 1-DOF controller. We usually choose $T_1=1$.

Equation 8.25 becomes

$$B^b = B_r \quad (8.27)$$

hence the expression of gamma is now changed to

$$\gamma = \frac{\phi_{cl(1)}}{B^b(1)} \quad (8.28)$$

and the desired closed loop transfer function now becomes

$$Y_m(z) = \gamma z^{-k} \frac{B^b}{\phi_{cl}} R(z) \quad (8.29)$$

This implies that the open loop model imposes two limitations on the closed loop model.

- The bad portion of the open loop model cannot be canceled out and it appears in the closed loop model.
- The open loop plant delay cannot be removed or minimized,i.e. the closed loop model cannot be made faster then the open loop model.

8.3 Step by step procedure to design and implement a 2-DOF controller

We obtain a first order transfer function of the plant using the step test approach.The model so obtained is

$$G(s) = \frac{0.42}{35.61s + 1} \quad (8.30)$$

with time constant $\tau = 35.6sec$ and gain $K = 0.42$

After discretization with sampling time = 1 second, we obtain

$$G(z) = \frac{0.0116304}{z - 0.9723086} \quad (8.31)$$

$$= \frac{0.0116304z^{-1}}{1 - 0.9723086z^{-1}} \quad (8.32)$$

We would now define good and bad terms

$$\begin{aligned} A^g &= 1 - 0.9723086z^{-1} \\ A^b &= 1 \\ B^g &= 0.0116304 \\ B^b &= 1 \end{aligned}$$

Let us now define the transient specifications. We choose,

$$\text{Rise time} = 100 \text{ seconds}$$

No. of samples per rise time (N_r) is calculated as

$$\begin{aligned} N_r &\leq \frac{\text{Rise time}}{\text{Sampling time}} \\ &= 100 \end{aligned}$$

next

$$\begin{aligned} \omega &= \frac{\pi}{2N_r} \\ &= 0.015708 \end{aligned}$$

We choose,

$$\begin{aligned} \text{Overshoot}(\epsilon) &= 0.05 \dots \text{i.e } 5\% \\ \rho &\leq \epsilon^{\omega/\pi} \\ &= 0.860 \end{aligned}$$

Let us now calculate 2DOF Controller parameters. The closed loop characteristic polynomial is given by

$$\begin{aligned} \phi_{cl} &= 1 - z^{-1}2\rho\cos\omega + \rho^2z^{-2} \\ &= 1 - 1.7198065z^{-1} + 0.7396z^{-2} \end{aligned}$$

But according to equation 8.26

$$A^b R_1 + z^{-k} B^b S_1 = \phi_{cl}$$

Recall that we had not considered external disturbance in the block diagram shown in figure8.4. However, we can still, up to some extent, take care of the disturbances. This is achieved by using the internal model principle. If a model of step is present inside the loop, step disturbances can be rejected. We can apply this by forcing R_c to have this term. A step model is given by

$$1(z) = \frac{1}{1 - z^{-1}}$$

Let the denominator of the step model be denoted as Δ

$$\Delta = 1 - z^{-1}$$

Therefore,

$$R_c = B^g \Delta R_1$$

Δ has a root which lies on the unit circle. Hence it has to be treated as a bad part and should not be canceled out. Hence, we should make sure that all of the occurrences of R_1 have this term.

Therefore,

$$\phi_{cl} = A^b \Delta R_1 + z^{-k} B^b S_1 \quad (8.33)$$

Hence,

$$A^b \Delta R_1 + z^{-k} B^b S_1 = 1 - 1.7198065z^{-1} + 0.7396z^{-2}$$

The expression is known as the Aryabhatta Identity and is solved using rigorous Matrix calculations. The explanation of this operation is not considered here. You may refer to the book "Digital Control" by Prof. Kannan Moudgalya [5]

$$\begin{aligned} R_c &= R_{c1} + R_{c2}z^{-1} + R_{c3}z^{-2} \\ &= 0.0116304 - 0.0229175z^{-1} + 0.0112871z^{-2} \\ S_c &= S_{c1} + S_{c2}z^{-1} \\ &= 0.0004641 - 0.0004512z^{-1} \\ T_c &= T_{c1} + T_{c2}z^{-1} \\ &= 1 - 0.9723z^{-1} \\ \gamma &= 0.0004641 \end{aligned}$$

Scilab code `twodof_para.sce` does these calculations. This code utilizes various other scilab codes provided at the end of this document. Execute this scilab code with the first order transfer function for your SBHS. You would obtain a Z-Transformed transfer function for the continuous time transfer function you input. You would also obtain the various parameters of 2dof controller as shown in figure 8.5¹ After execution of `twodof_para.sce`, run the Xcos code `twodof.xcos` with required setpoint value and observe the temperature profile. The performance of the controller is shown in figure 8.6 Make sure that you input the sampling time(Clock period) same as the one you used for discretization of the continuous time plant transfer function. It could be seen that the output (temperature) tracks the setpoint irrespective of the step changes in the fan speed. We can see that the Over shoot turns out to be 6% and rise time turns out to be 60 seconds, which is acceptable.

To implement a second order transfer function, input the correct second order transfer function in `twodof_para.sce`. Also, make sure you comment the first order control law equation and uncomment the second order control law equation in `twodof.sci` file.

8.3.1 Implementing 2dof controller on SBHS, virtually

The step by step procedure for conducting an experiment virtually is explained in section 3.4. The required .sce file is `twodof.sce`. You will find this file in the `2dof_controller` directory under virtual folder. The necessary code is listed in the section 8.4

8.4 Scilab Code

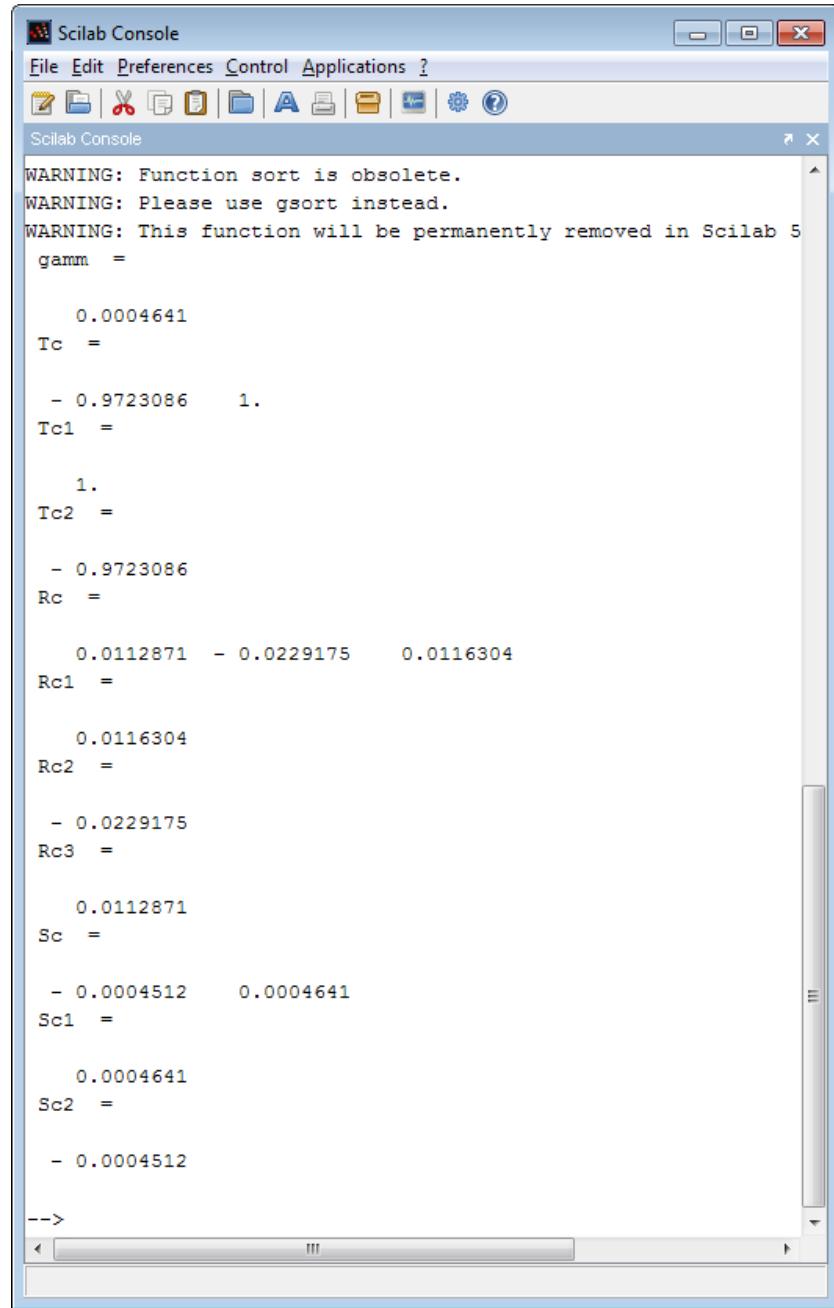
Scilab Code 8.1 c2d.sce

```

1 mode(0)
2 global NUM chi
3 s=poly(0,'s'); // Defines s to be a polynomial variable
4 TFcont = syslin('c',[0.532/((30.09*s+1)*(6.971*s+1))])
           // Creating cont-time transfer function
5 SScont = tf2ss(TFcont); // Converting cont-time
                           transfer function to state-space model

```

¹NOTE:- The scilab codes are given at the end of this document.



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

```
WARNING: Function sort is obsolete.  
WARNING: Please use gsort instead.  
WARNING: This function will be permanently removed in Scilab 5  
gamm =  
  
0.0004641  
Tc =  
  
- 0.9723086 1.  
Tc1 =  
  
1.  
Tc2 =  
  
- 0.9723086  
Rc =  
  
0.0112871 - 0.0229175 0.0116304  
Rc1 =  
  
0.0116304  
Rc2 =  
  
- 0.0229175  
Rc3 =  
  
0.0112871  
Sc =  
  
- 0.0004512 0.0004641  
Sc1 =  
  
0.0004641  
Sc2 =  
  
- 0.0004512  
-->
```

Figure 8.5: Scilab output for 2DOF_para.sce

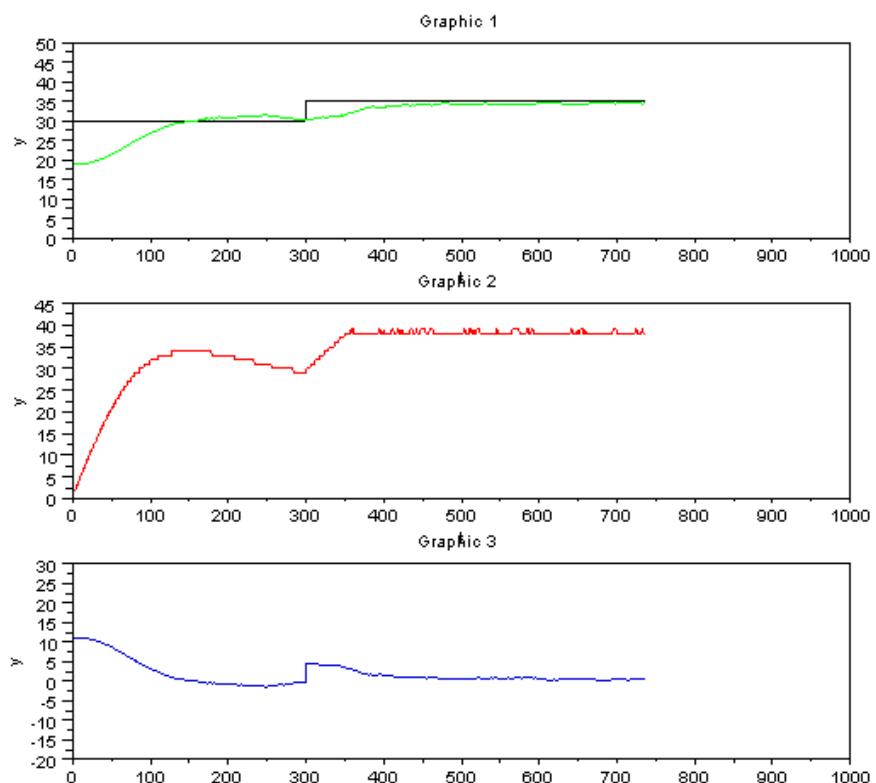


Figure 8.6: Implementation of 2DOF controller

```

6 Ts=0.5; // Sampling time
7 SSdisc=dscr(SScont,Ts); // Discretizing cont-time state
   - space model
8 TFDisc=ss2tf(SSdisc) // Converting discr-time ss model
   to tf
9 [Ds,NUM,chi]=ss2tf(SSdisc)

```

Scilab Code 8.2 2-DOF_para.sce

```

1 mode(0)
2 funcprot(0)
3 global temp NUM chi heat_in fan_in CO u_new u_old
   u_old_old r_old y_old Rc1 Rc2 Rc3 Sc1 Sc2 Tc1 Tc2
   gamm A B
4
5 exec cinddep.sci;
6 exec clcoef.sci;
7 exec colsplit.sci;
8 exec cosfil_ip.sci;
9 exec desired.sci;
10 exec indep.sci;
11 exec left_prm.sci;
12 exec makezero.sci;
13 exec move_sci.sci;
14 exec polmul.sci;
15 exec polsize.sci;
16 exec polsplit3.sci;
17 exec polyno.sci;
18 exec pp_im.sci;
19 exec rowjoin.sci;
20 exec seshft.sci;
21 exec t1calc.sci;
22 exec xdync.sci;
23 exec zpowk.sci;
24
25
26 // Transfer function
27 tf=coeff(chi);

```

```

28 B = coeff(NUM);
29 A = [ tf(2) tf(1)]; k=1;
30
31 // Transient specifications
32 rise = 5;
33 epsilon = 0.1;
34 Ts = 0.5;
35 phi = desired(Ts, rise, epsilon);
36
37 // Controller design
38 Delta = [1 -1]; // internal model of step used
39 [Rc, Sc, Tc, gamm] = pp_im(B, A, k, phi, Delta);
40
41 // parameters for twodof.cos
42 gamm
43 [Tcp1, Tcp2] = cosfil_ip(Tc, 1); // Tc / 1
44 Tc=coeff(Tcp1)
45 Tc1=Tc(1,2)
46 Tc2=Tc(1,1)
47 [Rcp1, Rcp2] = cosfil_ip(1, Rc); // 1 / Rc
48 Rc=coeff(Rcp2)
49 Rc1=Rc(1,3)
50 Rc2=Rc(1,2)
51 Rc3=Rc(1,1)
52 [Scp1, Scp2] = cosfil_ip(Sc, 1); // Sc / 1
53 Sc=coeff(Scp1)
54 Sc1=Sc(1,2)
55 Sc2=Sc(1,1)
56
57 // exec 2dof.sci ;

```

Scilab Code 8.3 2dof.sci

```

1 mode(0)
2 // 2 DOF Controller
3 // Heater input is passed as input argument to
   introduce control effort 'CO'

```

```

4 // Fan input is passed as input argument which is kept
   at constant level( disturbance )
5 // Range of Fan input :60 to 252
6 // Temperature is read
7
8 function [temp ,heat ,e_new] = twodof( setpoint ,fan)
9 global temp CO u_new u_old u_old_old r_old y_old Rc1
   Rc2 Rc3 Sc1 Sc2 Tc1 Tc2 gamm
10
11 e_new = setpoint - temp ;
12 r_new = setpoint ;
13 y_new = temp ;
14
15
16
17 u_new = (1/Rc1)*(gamm*Tc1*r_new + gamm*Tc2*r_old -Sc1*
   y_new -Sc2*y_old -Rc2*u_old - Rc3*u_old_old );
18
19 CO = u_new ;
20
21 if CO>39
22   CO = 39;
23 end ;
24
25 if CO<0
26   CO =0;
27 end ;
28
29 u_new = CO;
30
31 u_old_old = u_old ;
32 u_old = u_new ;
33 r_old = r_new ;
34 y_old = y_new ;
35
36 heat = u_new ;
37

```

```

38 writeserial(handl,ascii(254)); // Input Heater ,
    writeserial accepts strings ; so convert 254 into
    its string equivalent
39 writeserial(handl,ascii(heat));
40 writeserial(handl,ascii(253)); // Input Fan
41 writeserial(handl,ascii(fan));
42 writeserial(handl,ascii(255)); // To read Temp
43 sleep(100);

44
45 temp = ascii(readserial(handl)); // Read serial
    returns a string , so convert it to its integer (
    ascii ) equivalent
46 temp = temp(1) + 0.1*temp(2); // convert to temp with
    decimal points eg : 40.7
47
48 endfunction;

```

Scilab Code 8.4 cindep.sci

```

1 // Updated ----No change
2 // function b = cindep( S , gap )
3 // used in XD + YN = C . all rows except the last of
    are assumed to
4 // be independent . The aim is to check if the last
    row is dependent on the
5 // rest and if so how . The coefficients of dependence
    is sent in b
6 function b = cindep( S , gap )
7
8 if argn(2) == 1
9     gap = 1.0e8 ;
10 end
11 eps = 2.2204e-016;
12 [rows , cols ] = size(S);
13 if rows > cols
14     ind = 0;
15 else
16     sigma = svd(S);

```

```

17 len = length(sigma);
18 if (sigma(len)/sigma(1) <= (eps*max(i,cols)))
19     ind = 0;                                // not independent
20 else
21     if or(sigma(1:len-1) ./ sigma(2:len)>=gap)
22         ind = 0;                            // not dependent
23     else
24         ind = 1;                            // independent
25     end
26 end
27 if ind
28     b = [];
29 else
30     b = S(rows,:)/S(1:rows-1,:);
31     b = makezero(b,gap);
32 end
33
34 endfunction

```

Scilab Code 8.5 clcoef.sci

```

1 // Updated ----- No change
2 // H. Kwakernaak, July , 1990
3 // Modified by Kannan Moudgalya in Nov. 1992
4
5 function [P,degP] = clcoef(Q,degQ)
6
7 [rQ,cQ] = polysize(Q,degQ);
8
9 if and(and(Q==0))
10    P = zeros(rQ,cQ);
11    degP = 0;
12 else
13    P = Q; degP = degQ; rP = rQ; cP = cQ;
14    j = degP+1;
15    while j >= 0
16    X = P(:,(j-1)*cP+1:j*cP)
17    if max(sum(abs(X'))) < (1e-8)*max(sum(abs(P)))

```

```

18      P = P(:,1:(j-1)*cP);
19      degP = degP-1;
20  else
21      j = 0;
22  end
23      j = j-1;
24  end
25 end
26 endfunction

```

Scilab Code 8.6 clcoef.sci

```

1 // Updated ----- No change
2 // H. Kwakernaak, July , 1990
3 // Modified by Kannan Moudgalaya in Nov. 1992
4
5 function [P,degP] = clcoef(Q,degQ)
6
7 [rQ,cQ] = polsize(Q,degQ);
8
9 if and(and(Q==0))
10    P = zeros(rQ,cQ);
11    degP = 0;
12 else
13    P = Q; degP = degQ; rP = rQ; cP = cQ;
14    j = degP+1;
15    while j >= 0
16      X = P(:,(j-1)*cP+1:j*cP)
17      if max(sum(abs(X'))) < (1e-8)*max(sum(abs(P)))
18        P = P(:,1:(j-1)*cP);
19        degP = degP-1;
20      else
21        j = 0;
22      end
23      j = j-1;
24    end
25 end
26 endfunction

```

Scilab Code 8.7 cosfil_ip.sci

```
1 // Updated (31 - 7 - 07)
2 // Input arguments are numerator and denominator
3 // polynomials' coefficients in ascending
4 // powers of z^-1
5
6 // Scicos blocks need input polynomials
7 // with positive powers of z
8
9 function [nume,deno] = cosfil_ip(num,den)
10
11 [Nn,Nd] = polyno(num,'z');
12 [Dn,Dd] = polyno(den,'z');
13 nume = Nn*Dd;
14 deno = Nd*Dn;
15
16 endfunction;
```

Scilab Code 8.8 desired.sci

```
1 // Updated (26 - 7 - 07)
2 // 9.4
3 function [phi,dphi] = desired(Ts,rise,epsilon)
4
5 Nr = rise/Ts; omega = %pi/2/Nr; rho = epsilon^(omega/
%pi);
6 phi = [1 -2*rho*cos(omega) rho^2]; dphi = length(phi)
-1;
7 endfunction;
```

Scilab Code 8.9 indep.sci

```
1 // Updated ---- No change
2 // function b = indep(S,gap)
3 // determines the first row that is dependent on the
previous rows of S.
```

```

4 // The coefficients of dependence is returned in b
5 function b = indep( S,gap)
6
7 if argn(2) == 1
8     gap = 1.0e8;
9     end
10 [rows ,cols ] = size(S);
11 ind = 1;
12 i = 2;
13 eps = 2.2204e-016;
14 while ind & i <= rows
15     sigma = svd(S(1:i,:));
16     len = length(sigma);
17     if( sigma(len)/sigma(1) < (eps*max(i ,cols )))
18         ind =0;
19     else
20         shsig = [ sigma(2:len);sigma(len)];
21         if or( (sigma ./ shsig) > gap)
22             ind = 0;
23         else
24             ind = 1;
25             i = i +1;
26         end
27     end
28
29 end
30 if ind
31     b =[ ];
32
33 else
34     c = S(i,:) /S(1:i-1,:);
35     c = makezero(c ,gap);
36     b = [-c 1];
37 end
endfunction

```

Scilab Code 8.10 left_prm.sci

```

1 // function [B, degB, A, degA, Y, degY, X, degX] = ...
2 // left prm (N, degN, D, degD, job, gap)
3 //
4 // does three different things according to integers
5 // that 'job' takes
6 // job = 1.
7 // this is the default. It is always done for all
8 // jobs.
9 // -1
10 // Given ND , returns coprime B and A where ND = A
11 // B
12 // It is enough if one sends the first four input
13 // arguments
14 // If gap is required to be sent, then one can send
15 // either 1 or a null
16 // entry for job
17 // job = 2.
18 // first solve for job = 1 and then solve XA + YB = I
19 // job = 3.
20 // used in solving XD + YN = C
21 // after finding coprime factorization, data are
22 // returned
23 // convention: the variable with prefix deg stand for
24 // degrees
25 // of the corresponding polynomial matrices
26 //
27 // input:
28 // N: right fraction numerator polynomial matrix
29 // D: right fraction denominator polynomial matrix
30 // N and D are not necessarily coprime
31 // gap: variable used to zero entries; default value
32 // is 1.0e+8
33 //
34 // output
35 // b and A are left coprime num. and den. polynomial
36 // matrices

```

```

29 // X and Y are solutions to Aryabhatta identity , only
   for job = 2
30
31 function [B,degB,A,degA,Y,degY,X,degX] = left_prm(N,
   degN,D,degD,job,gap)
32 if argn(2) == 4 | argn(2) == 5
33   gap = 1.0e8 ;
34 end
35 // pause
36 if argn(2) == 4,
37   job = 1; end
38 [F,degF] = rowjoin(D,degD,N,degN);
39 [Frows,Fbcols] = polysize(F,degF);           // Fbcols =
   block columns
40 Fcols = Fbcols * (degF+1);                   // actual
   columns of F
41 T1 = []; pr = []; degT1 = 0; T1rows = 0; shft = 0;
42 S=F; sel = ones(Frows,1); T1bcols = 1;
43 abar = (Fbcols + 1):Frows;                  // a superbar
   of B-C. Chang
44 while isempty(T1) | T1rows < Frows - Fbcols
45   Srows = Frows*T1bcols; // max actual columns of
   result
46   [T1,T1rows,sel,pr] = ...
47     t1calc(S,Srows,T1,T1rows,sel,pr,Frows,
   Fbcols,abar,gap);
48   [T1rows,T1cols] = size(T1);
49   if T1rows < Frows - Fbcols
50     T1 = [T1 zeros(T1rows,Frows)];
51     T1bcols = T1bcols + 1;           // max. block
   columns of result
52     degT1 = degT1 + 1;             // degree of
   result
53     shft = shft +Fbcols;
54     S = seshft(S,F,shft);
55     sel = [ sel; sel(Srows-Frows+1:Srows) ];
56     rowvec = (T1bcols-1)*Frows+(Fbcols+1):T1bcols
   * Frows;

```

```

57         abar = [abar rowvec];           // A_s u p e r _ b a r
58         o f   B - C . c h a n g
59     end
60
61 [B,degB,A,degA] = colsplit(T1,degT1,Fbcols,Frows-
62 Fbcols);
62 [B,degB] = clcoef(B,degB);
63 B = -B;
64 [A,degA] = clcoef(A,degA);
65 // pause
66 if job == 2
67     S = S(m1b_logical(sel),:);
68             // columns
68 [redSrows,Scols] = size(S);
69 C = [eye(Fbcols,Fbcols) zeros(Fbcols,Scols-
70 Fbcols)];    // append with zeros
70 T2 = C/S;
71 T2 = makezero(T2,gap);
72 T2 = move_sci(T2,find(sel),Srows);
73 [X,degX,Y,degY] = colsplit(T2,degT1,Fbcols,Frows-
74 Fbcols);
74 [X,degX] = clcoef(X,degX);
75 [Y,degY] = clcoef(Y,degY);
76 elseif job == 3
77     Y = S;
78     degY = sel;
79     X = degT1;
80     degX = Fbcols;
81 else
82     if job ~= 1
83         error('Message from left prm: no legal job
84             number specified')
85     end
85 end
86 endfunction

```

Scilab Code 8.11 makezero.sci

```

1 // Updated
2 // function B = makezero(B, gap)
3 // where B is a vector and gap acts as a tolerance
4
5 function B = makezero(B, gap)
6
7 if argn(2) == 1
8     gap = 1.0e8;
9 end
10 temp = B(find(B));           // non zero entries of B
11 temp = -gsort(-abs(temp));   // absolute values sorted
12 len = length(temp);
13 ratio = temp(1:len-1) ./ temp(2:len); // each ratio >1
14 min_ind = min(find(ratio>gap));
15 if ~isempty(min_ind)
16     our_eps = temp(min_ind+1);
17     zeroind = find(abs(B)<=our_eps);
18     B(zeroind) = zeros(1,length(zeroind));
19 end
20 endfunction

```

Scilab Code 8.12 move_sci.sci

```

1 // function result = move_sci(b, nonred, max_sci)
2 // Moves matrix b to matrix result with the
3 // information on where to move,
4 // decided by the indices of nonred.
5 // The matrix result will have as many rows as b has
6 // and max number of columns.
7 // b is augmented with zeros to have nonred number of
8 // columns;
9 // The columns of b put into those of result as
10 // decided by nonred.
11
12 function result = move_sci(b, nonred, max_sci)
13 [brows, bcols] = size(b);
14 b = [b zeros(brows, length(nonred)-bcols)];

```

```

11 result = zeros(brows,max_sci);
12 result(:,nonred') = b;
13 endfunction

```

Scilab Code 8.13 polmul.sci

```

1 // Updated ----- No change
2 // polmul
3 // The command
4 // [ C , degA ] = polmul ( A , degA , B , degB )
5 // produces the polynomial matrix C that equals the
// product A*B of the
6 // polynomial matrices A and B .
7 //
8 // H. Kwakernaak , July , 1990
9
10
11 function [C,degC] = polmul(A,degA,B,degB)
12 [rA,cA] = polysize(A,degA);
13 [rB,cB] = polysize(B,degB);
14 if cA ~= rB
15   error('polmul: Inconsistent dimensions of input
matrices');
16 end
17
18 degC = degA+degB;
19 C = [];
20 for k = 0:degA+degB
21   mi = 0;
22   if k-degB > mi
23     mi = k-degB;
24   end
25   ma = degA;
26   if k < ma
27     ma = k;
28   end
29   Ck = zeros(rA,cB);
30   for i = mi:ma

```

```

31      Ck = Ck + A(:, i*cA+1:(i+1)*cA)*B(:,(k-i)*cB
32          +1:(k-i+1)*cB);
33
34 end
35 C = [C Ck];
end
endfunction

```

Scilab Code 8.14 polsize.sci

```

1 // Updated ----- No change
2 // function [rQ , cQ ] = polsize ( Q , degQ )
3 // FUNCTION polsize TO DETERMINE THE DIMENSIONS
4 // OF A POLYNOMIAL MATRIX
5 //
6 // H. Kwakernaak , August , 1990
7
8 function [rQ , cQ ] = polsize ( Q , degQ )
9
10 [rQ , cQ ] = size ( Q ) ; cQ = cQ / ( degQ + 1 ) ;
11 if abs ( round ( cQ ) - cQ ) > 1e-6
12     error ( ' polsize : Degree of input inconsistent with
13             number of columns ' );
14 else
15     cQ = round ( cQ );
16 end
endfunction

```

Scilab Code 8.15 polsplit3.sci

```

1 // Updated (18 -7 -07)
2 // 9.11
3 // function [ goodpoly , badpoly ] = polsplit3 ( fac , a )
4 // Splits a scalar polynomial of z ^{ -1 } into good and
5 // bad
6 // factors . Input is a polynomial in increasing degree
6 // of
6 // z ^{ -1 } . Optional input is a , where a <= 1 .

```

```

7 // Factors that have roots outside a circle of radius
8 // a or
9 // with negative roots will be called bad and the rest
10 // good. If a is not specified, it will be assumed as
11 // 1.
12
13 function [goodpoly,badpoly] = polsplit3(fac,a)
14 if argn(2) == 1, a = 1; end
15 if a>1 error('good polynomial also is unstable'); end
16 fac1 = poly(fac(length(fac):-1:1), 'z', 'coeff');
17 rts = roots(fac1);
18 rts = rts(length(rts):-1:1);
19
20 // extract good and bad roots
21 badindex = mtlb_find((abs(rts)>=a-1.0e-5)|(real(rts)
22 <-0.05));
23 badpoly = coeff(poly(rts(badindex), 'z'));
24 goodindex = mtlb_find((abs(rts)<a-1.0e-5)&(real(rts)
25 >=-0.05));
26 goodpoly = coeff(poly(rts(goodindex), 'z'));
27
28 // scale by equating the largest terms
29 [m,index] = max(abs(fac));
30 goodbad = convol(goodpoly,badpoly);
31 goodbad = goodbad(length(goodbad):-1:1);
32 factor1 = fac(index)/goodbad(index);
33 goodpoly = goodpoly(length(goodpoly):-1:1);
34 badpoly = badpoly(length(badpoly):-1:1);
35 endfunction;

```

Scilab Code 8.16 polyno.sci

```

1 // Updated (1 - 8 - 07)
2 // Operations :
3 // Polynomial definition
4 // Flipping of coefficients

```

```

5 // Variable ----- passed as input argument (either ,
6 // s' or 'z')
7 // Both num and den are used mostly used in scicos
8 // files ,
9 // to get rid of negative powers of z
10 // Polynomials with powers of s need to
11 // be flipped only
12 function [polynu,polyde] = polyno(zc,a)
13 zc = clean(zc);
14 polynu = poly(zc(length(zc):-1:1),a,'coeff');
15 if a == 'z'
16 polyde = %z^(length(zc) - 1);
17 else
18 polyde = 1;
19 end
20
21 // Scicos (4.1) Filter block shouldn't have constant /
22 // constant
23 if type(polynu)==1 & type(polyde)==1
24 if a == 'z'
25 polynu = %z; polyde = %z;
26 else
27 polynu = %s; polyde = %s;
28 end;
29 end;
30 endfunction

```

Scilab Code 8.17 pp_im.sci

```

1 // Updated (27-7-07)
2 // 9.8
3 // function [Rc , Sc , Tc , gamma , phit] = pp_im (B , A , k , phi ,
4 // Delta )
5 // Calculates 2-DOF pole placement controller .
5 // -----

```

```

6
7 function [Rc , Sc , Tc , gamm] = pp_im(B , A , k , phi , Delta)
8
9 // Setting up and solving Aryabhatta identity
10 [Ag , Ab] = polsplit3(A); dAb = length(Ab) - 1;
11 [Bg , Bb] = polsplit3(B); dBb = length(Bb) - 1;
12
13 [zk , dzk] = zpowk(k);
14
15 [N , dN] = polmul(Bb , dBb , zk , dzk);
16 dDelta = length(Delta) - 1;
17 [D , dD] = polmul(AB , dAb , Delta , dDelta);
18 dphi = length(phi) - 1;
19
20 [S1 , dS1 , R1 , dR1] = xdync(N , dN , D , dD , phi , dphi);
21
22 // Determination of control law
23 Rc = convol(Bg , convol(R1 , Delta)); Sc = convol(Ag , S1);
24 Tc = Ag; gamm = sum(phi)/sum(Bb);
25 endfunction;

```

Scilab Code 8.18 rowjoin.sci

```

1 // Updated -----No change
2 // function [P , degP] = rowjoin(P1 , degP1 , P2 , degP2)
3 // MATLAB FUNCTION rowjoin TO SUPERPOSE TWO POLYNOMIAL
4 // MATRICES
5
6 // H. Kwakernaak , July , 1990
7
8 function [P , degP] = rowjoin(P1 , degP1 , P2 , degP2)
9
10 [rP1 , cP1] = polsize(P1 , degP1);
11 [rP2 , cP2] = polsize(P2 , degP2);
12 if cP1 ~= cP2
13     error('rowjoin: Inconsistent numbers of columns');
14 end
15

```

```

16 rP = rP1+rP2; cP = cP1;
17 if degP1 >= degP2
18     degP = degP1;
19 else
20     degP = degP2;
21 end
22
23 if isempty(P1)
24     P = P2;
25 elseif isempty(P2)
26     P = P1;
27 else
28     P = zeros(rP,(degP+1)*cP);
29     P(1:rP1,1:(degP1+1)*cP1) = P1;
30     P(rP1+1:rP,1:(degP2+1)*cP2) = P2;
31 end
32 endfunction

```

Scilab Code 8.19 seshft.sci

```

1 // Updated ----- No change
2 // function C = seshft(A,B,N)
3 // given A and B matrices, returns C = [ <-A-> 0
4 //                                     0 <-B-> ] with B
5 // shifted east by N cols
6
6 function C = seshft(A,B,N)
7 [Arows , Acols] = size(A);
8 [Brows , Bcols] = size(B);
9 if N >= 0
10    B = [zeros(Brows ,N) B];
11    Bcols = Bcols + N;
12 elseif N < 0
13    A = [zeros(Arows ,abs(N)) A];
14    Acols = Acols +abs(N);
15 end
16 if Acols < Bcols
17    A = [A zeros(Arows ,Bcols-Acols )];

```

```

18    elseif Acols > Bcols
19        B = [B zeros(Brows ,Acols-Bcols )];
20    end
21    C = [A
22        B];
23 endfunction

```

Scilab Code 8.20 t1calc.sci

```

1 // Updated
2 // function [ T1 , T1rows , sel , pr ] = ...
3 // t1calc (S , Srows ,T1 , T1rows , sel , pr , Frows , Fbcols , abar ,
4 // gap )
5 // calculates the coefficient matrix T1
6 // redundant row information is kept in sel : redundant
// rows are marked
7 // with zeros . The undeleted rows are marked with
8 // ones .
9
10
11 function [T1,T1rows ,sel ,pr] = t1calc (S ,Srows ,T1 ,T1rows
12 , sel ,pr ,Frows ,Fbcols ,abar ,gap)
13 b = 1; // vector of
14 primary red. rows
15
16 while (T1rows < Frows - Fbcols) & or(sel==1) & ~
17 isempty(b)
18     S = clean(S);
19     b = indep(S( m1b_logical(sel) ,:) ,gap); // send
// selected rows of S
20     if ~isempty(b)
21         b = clean(b);
22         b = move_sci(b ,find(sel) ,Srows );
23         j = length(b);
24         while ~(b(j) & or(abar==j)) // pick largest
// non zero entry
25             j = j-1; // of coeff .
26             b belonging to abar
27         if ~j

```

```

21      fprintf( '\nMessage from t1calc ,  

22          called from left_prm\n\n' )  

23      error( 'Denominator is noninvertible  

24          ' )  

25      end  

26      if ~or(j<pr & pmodulo( pr , Frows ) == pmodulo( j ,  

27          Frows )) // pr(2), pr(1)  

28          T1 = [T1; b]; // condition  

29          is not violated  

30          T1rows = T1rows +1; // accept this  

31          vector  

32      end // else don't  

33          accept  

34          pr = [pr; j]; // update  

35          prime red row info  

36          while j <= Srows  

37              sel(j) = 0;  

38              j = j + Frows;  

39          end  

40      end  

41  endfunction

```

Scilab Code 8.21 xdync.sci

```

1 // Updated ----No change
2 // function [Y, degY, X, degX, B, degB, A, degA] = xdync(N,
3 // degN, D, degD, C, degC, gap)
4 // given coefficient matrix in T1, primary redundant
5 // row information sel,
6 // solves XD + YN = C
7 // calling order changed on 16 April 2005. Old order:
8 // function [B, degB, A, degA, Y, degY, X, degX] = xdync(N,
degN, D, degD, C, degC, gap)

```

```

9  function [Y,degY,X,degX,B,degB,A,degA] = xdync(N,degN,
10   D,degD,C,degC,gap)
11   if argn(2) == 6
12     gap = 1.0e+8;
13   end
14
15
16   [F,degF] = rowjoin(D,degD,N,degN);
17
18   [Frows,Fbcols] = polysize(F,degF); // Fbcols = block
19   columns
20
21   [B,degB,A,degA,S,sel,degT1,Fbcols] = left_prm(N,degN,D
22   ,degD,3,gap);
23
24   // if issoln(D,degD,C,degC,B,degB,A,degA)
25   [Crows,Ccols] = size(C);
26   [Srows,Scols] = size(S);
27   S = clean(S);
28   S = S(mtlb_logical(sel),:);
29   T2 = [];
30
31   for i = 1:Crows,
32     Saug = seshft(S,C(i,:),0);
33     b = cinddep(Saug);
34     b = move_sci(b,find(sel),Srows);
35     T2 =[T2; b];
36   end
37
38   [X,degX,Y,degY] = colsplit(T2,degT1,Fbcols,Frows-
39   Fbcols);
40
41   [X,degX] = clcoef(X,degX);
42   [Y,degY] = clcoef(Y,degY);
43   Y = clean(Y); X = clean(X);
44
45 endfunction

```

Scilab Code 8.22 zpowk.sci

1 // Updated (26-7-07)

```
2    / /    9 . 6
3    / /    - - - - -
4
5 function [zk ,dzk ] = zpowk(k)
6 zk = zeros(1 ,k+1); zk(1 ,k+1) = 1;
7 dzk = k ;
8 endfunction
```

Chapter 9

Implementing Internal Model Controller for first order systems on a Single Board Heater System

The Aim of this experiment is to implement an Internal Model Controller for first order systems on a single board heater system. 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 Fig.9.1. Fan speed and Heater current are the two inputs to the system. For this experiment, the heater current is used as a control effort. The fan input could be thought of as an external disturbance.

9.1 IMC Design for Single Board Heater System

Internal Model Controller contain explicit model of plant as its part, hence it is name as Internal Model Controller [6]. With stable open loop transfer function and stable controller, the closed loop system can be stabled. The IMC has been used mainly for stable plants. Now, transfer function of the stable plant be denoted by $G_p(z)$ and it's model is denoted by $G(z)$,hence

$$y(n) = G(z)u(n) + \xi(n) \quad (9.1)$$

where;

y(n)=plant output;

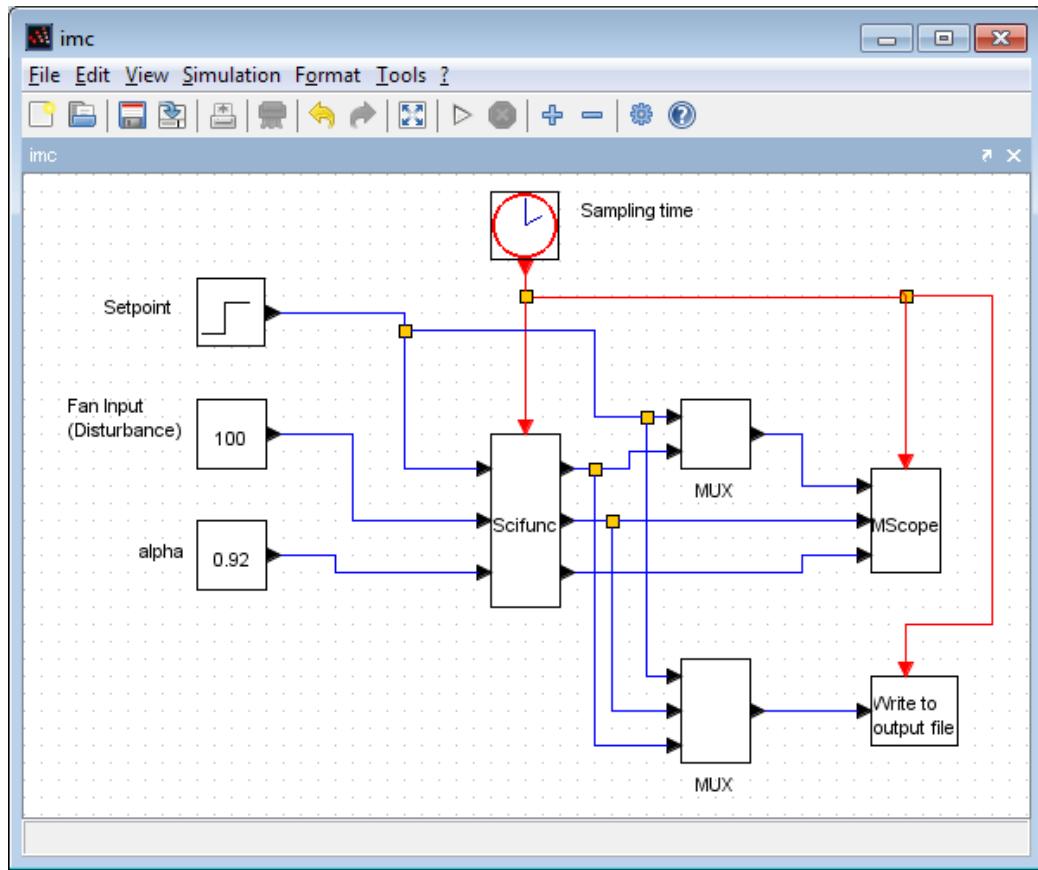


Figure 9.1: Xcos interface for this experiment

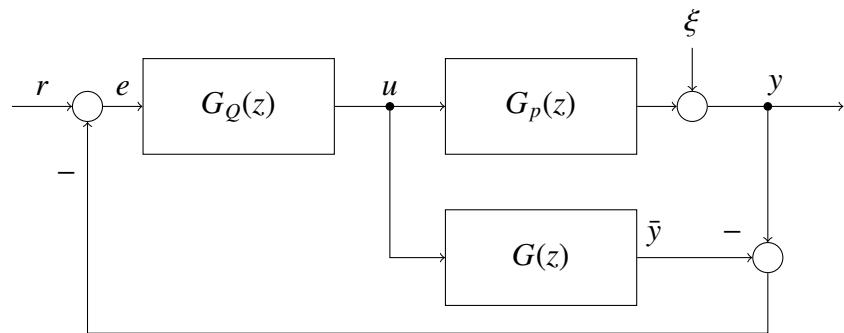


Figure 9.2: IMC feedback configuration

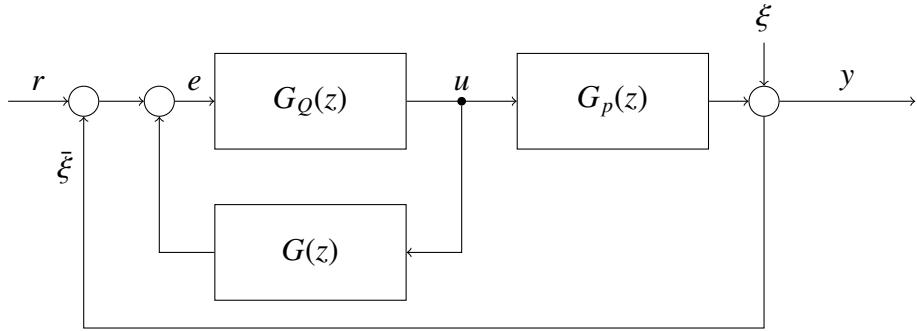


Figure 9.3: IMC feedback configuration

$u(n)$ =plant input;

$\xi(n)$ =noise.

Fig.9.2 shows method to control stable plant by using internal model control. For noise rejection with $y=0$ we required $G_Q = G_p^{-1}$ and $G = G_p$ i.e. for stable G_Q we required an approximate inverse of G .Also, for internal stability transfer function between any two points in the feedback loop must be stable.[5]

9.2 Step for designing IMC for stable plant

Invert the delay free plant model so that G_Q is realizable. For non-minimum phase part of the plant, reciprocal polynomial is used for stable controller. Negative real part of the plant should replace with the steady state equivalent of that part to avoid oscillatory nature of control effort. Low pass filter must be used to avoid the high frequency components because of model mismatch. IMC design means obtaining a realizable G_Q that is stable and approximately inverse of G . Invert the delay free plant model so that G_Q is realizable. We have model of SBHS ,which is given by,

$$G = Z^{-1} \frac{0.01163}{1 - 0.9723Z^{-1}} \quad (9.2)$$

Inverting delay free plant, We get

$$\frac{A}{B} = \frac{1 - 0.9723Z^{-1}}{0.01163} \quad (9.3)$$

Now, Comparing plant model with equation,

$$G = Z^{-1} \frac{B^g B^- B^{nm+}}{A} \quad (9.4)$$

$$B^g = 0.01163 \quad (9.5)$$

$$B^- = 1 \quad (9.6)$$

$$B^{nm+} = 1 \quad (9.7)$$

$$A = 1 - 0.9723Z^{-1} \quad (9.8)$$

For the stable system internal model controller is give by

$$G_Q = \frac{A}{B^g B_s^- B_r^{nm+}} G_f \quad (9.9)$$

$$G_Q = \frac{1 - 0.9723Z^{-1}}{0.01163} \frac{1 - \alpha}{1 - \alpha Z^{-1}} \quad (9.10)$$

Now,

$$G_c = \frac{G_Q}{1 - GG_Q} \quad (9.11)$$

$$\frac{u}{e} = \frac{\frac{1 - 0.9723Z^{-1}}{0.01163} \frac{1 - \alpha}{1 - \alpha Z^{-1}}}{1 - Z^{-1} \frac{0.01163}{1 - 0.9723Z^{-1}} \frac{1 - 0.9723Z^{-1}}{0.01163} \frac{1 - \alpha}{1 - \alpha Z^{-1}}} \quad (9.12)$$

After simplifying, We get

$$\frac{u}{e} = \frac{1 - \alpha}{0.01163} \frac{1 - 0.9723Z^{-1}}{1 - Z^{-1}} \quad (9.13)$$

$$\frac{u}{e} = b \frac{1 - 0.9723Z^{-1}}{1 - Z^{-1}} \quad (9.14)$$

Where,

$$(9.15)$$

$$b = \frac{1 - \alpha}{0.01163} \quad (9.16)$$

Hence,

$$u(n) = u(n - 1) + b[e(n) - 0.9723e(n - 1)] \quad (9.17)$$

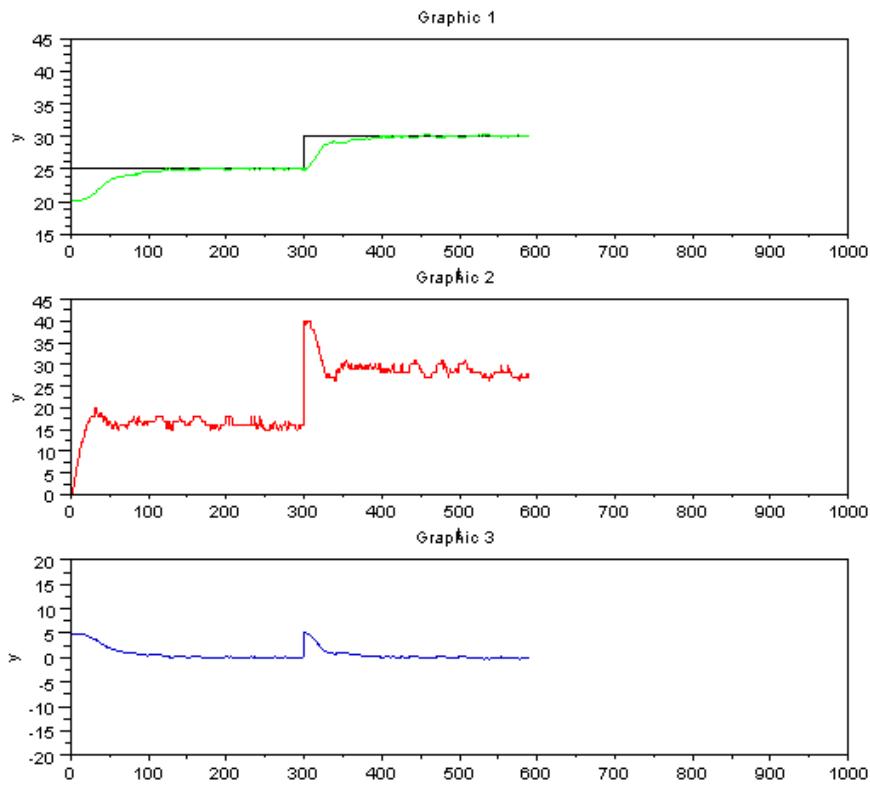


Figure 9.4: Experimental Results with IMC for $\alpha = 0.92$

For implementing above IMC controller, scilab code is given in `imc.sci` file, listed at the end of this document. Change the current working directory to the folder `imc_controller`. Execute the file `ser_init.sce` with the appropriate com port number and then execute the file `imc.sci` for loading the function. Run the xcos file `imc.xcos`. Output of Xcos is shown in below fig.9.4. Figure shows three plots. First subplot shows Setpoint and output temperature profile. Second subplot shows control effort and third subplot shows error between setpoint and plant output.

9.3 Experimental Results

By comparing above two graph we can say that for $\alpha = 0.92$ the response of the controller is sluggish. For $\alpha = 0.85$ the controller starts responding quickly and

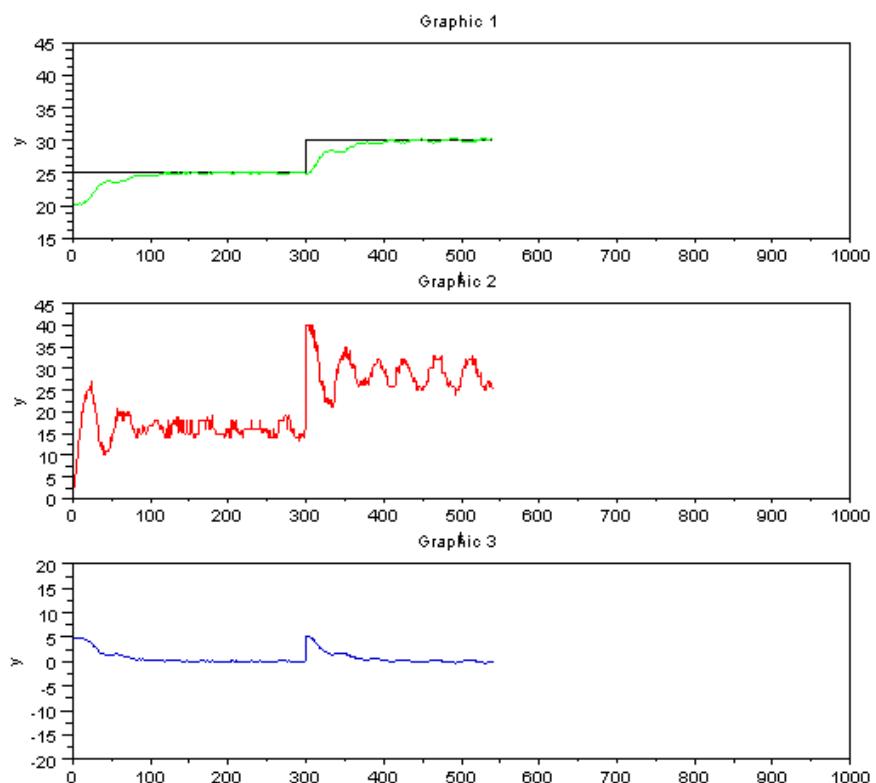


Figure 9.5: Experimental Results with IMC for $\alpha = 0.85$

no overshoots are seen in the temperature profile.

9.3.1 Implementing IMC controller on SBHS, virtually

The step by step procedure for conducting an experiment virtually is explained in section 3.4. The required .sce file is `imc_virtual.sce`. You will find this file in the `imc_controller` directory under `virtual` folder. The necessary codes are listed in the section 9.4

9.4 Scilab Code

Scilab Code 9.1 ser_init.sce

```
1 // To load the serial toolbox and open the serial port
2 exec loader.sce
3
4 handl = openserial(6,"9600,n,8")
5
6 // the order is : port number ,” baud , parity , databits ,
7 // stop bits ”
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 if (ascii(handl) ~= [])
12     disp("COM Port Opened");
13 end
```

Scilab Code 9.2 imc.sci

```
1 mode(0)
2 function [temp,heat,e_new] = imc(setpoint,fan,alpha)
3 global temp heat_in fan_in et SP u_new u_old u_new
4 e_old e_new
5 e_new = setpoint - temp;
```

```

6   b=((1-alpha)/0.01163);
7   u_new = u_old + b*(e_new - (0.9723*e_old));
8
9
10  if u_new>100
11      u_new = 100;
12  end;
13
14  if u_new<0
15      u_new = 0
16      ;
17  end;
18
19
20  if fan>100
21      fan = 100;
22  end;
23
24  if fan<0
25      fan = 0
26      ;
27  end;
28
29
30 heat = u_new;
31 u_old = u_new;
32 e_old = e_new;
33
34
35 writeserial(handl,ascii(254)); // Input Heater,
36           writeserial accepts strings; so convert 254 into
37           its string equivalent
36 writeserial(handl,ascii(heat));
37 writeserial(handl,ascii(253)); // Input Fan
38 writeserial(handl,ascii(fan));
39 writeserial(handl,ascii(255)); // To read Temp
40 sleep(100);
41

```

```

42 temp = ascii(readserial(handl)); // Read serial
   returns a string , so convert it to its integer (
   ascii ) equivalent
43 temp = temp(1) + 0.1*temp(2); // convert to temp with
   decimal points eg : 40.7
44
45 endfunction;

```

Scilab Code 9.3 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 fncre fncre m err_count y
5
6 fncre = 'clientread.sce';
7 fdt = mopen(fncre);
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 )
   xcos('imc.xcos');
21 else
22 disp("NO NETWORK CONNECTION!");
23 return
24
25 end

```

Scilab Code 9.4 imc_virtual.sci

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

```

Chapter 10

Design and Implementation of Self Tuning PI and PID Controllers on Single Board Heater System

10.1 Introduction

This chapter presents Design and Implementation of Self Tuning PI and PID Controllers on Single Board Heater System done by Mr. Vikas Gayasen.¹ When a plant is wired in a close loop with a PID controller, the parameters, K_c , τ_i and τ_d determine the variation of the manipulated input that is given by the controller. This, in turn, determines the variation of the controlled variable, when a set point is given. Suitable values of these parameters can be found out when plant transfer function is known. However, with large changes in the controlled variable, there may be appreciable changes in the plant transfer function itself. Therefore, it is needed to dynamically update the controller parameters according to the transfer function.

10.1.1 Objective

The objective of the present study was to design and implement an algorithm that would dynamically update the values of the controller parameters that are used to control the temperature in the Single Board Heater System (SBHS).

¹Copyright: Mr. Vikas Gayasen, student of Prof. Kannan Moudgalya, IIT Bombay for process control course, 2010

10.1.2 Apparatus

1. Fig 10.1 shows the single board heater system on which this experiment will be performed.
2. The setup consists of a heater assembly, fan, temperature sensor, microcontroller and associated circuitry.
3. Heater assembly consists of an iron plate placed at a distance of about 3.5 mm from the nichrome coil.
4. A 12 V computer fan positioned below this heater assembly is meant for cooling the assembly.
5. The temperature sensed by the temperature sensor, AD 590, after suitable processing, is fed to the microcontroller.
6. The microcontroller ATmega16 is the heart of the setup. It provides an interface between the process and the computer.
7. The LCD display mounted above the microcontroller displays the heated plate temperature, heater and fan inputs and also the commands communicated via serial port.
8. The setup is powered by 12 V, 8 A SMPS.

10.2 Theory

10.2.1 Why a Self Tuning Controller?

The transfer function of SBHS is assumed as

$$\Delta T = \frac{K_p}{\tau_p s + 1} \Delta H + \frac{K_f}{\tau_f s + 1} \Delta F \quad (10.1)$$

ΔT : Temperature Change

ΔF : Fan Input Change

ΔH : Heater Input Change



Figure 10.1: Single Board Heater System

The values of K_p , K_f , τ_s and τ_f can be found by conducting step test experiments. Using these values, the parameters (K_c , τ_i and τ_d) of the PID controller can be defined using methods like Direct Synthesis or Ziegler Nichols Tuning. However, when the apparatus is used in over a large range of temperature, the values of the plant parameters (K_p , K_f , τ_s and τ_f) may change. The new values would give new values of PID controller parameters. However, in a conventional PID controlled system, the parameters K_c , τ_i and τ_d are defined beforehand and are not changed when the system is working. Therefore, we might have a situation in which the PID controller is working with unsuitable values that may not give the desired performance. Therefore, it becomes necessary to change/update the values of the PID parameters so that the plant gives the optimum performance.

10.2.2 The Approach Followed

Following is the Variable Discription for this project:

- Manipulated Variable: Heater Input
- Disturbance Variable: Fan Input
- Controlled Variable: Temperature

Several open loop step test experiments were performed (giving step changes in the heater input) and the values of K_p and τ_p were found from the results for each experiment by fitting the inverse laplace transform of the assumed transfer function with the experimental data. These values were plotted with respect to the corresponding average temperatures. From these plots, correlations were found for both K_p and τ_p as functions of temperature. From correlations of K_p and τ_p , the PID parameters could be found as functions of temperature. Thus, in the new PID controller, the values of K_c , τ_i and τ_d are calculated using the temperature of the system. For the calculation of PID settings, two approaches: Direct Synthesis and Ziegler-Nichols Tuning are followed.

10.2.3 Direct synthesis

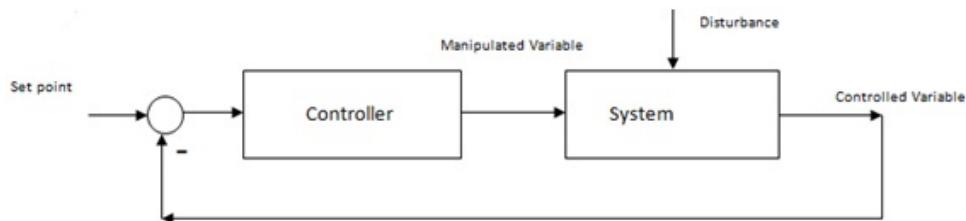


Figure 10.2: Closed Loop Circuit

$$V(s) = \frac{G_c(s)G(s)}{1 + G_c(s)G(s)} \quad (10.2)$$

Where

$V(s)$: Overall closed-loop transfer function

$G_c(s)$: Controller transfer function

$G(s)$: System transfer function.

Therefore,

$$G_c(s) = \frac{1}{G(s)} \frac{V(s)}{1 - V(s)}$$

Let the desired closed loop transfer function be of form

$$V(s) = \frac{1}{(\tau_{cl}s + 1)} \quad (10.3)$$

$$G(s) = \frac{K_p}{(\tau_ps + 1)} \quad (10.4)$$

By using the equations for $G(s)$ and $V(s)$, we get:

$$G(c) = K_c(1 + \frac{1}{\tau s}) \quad (10.5)$$

Where,

$$K_c = \frac{1}{K_p} (\tau_p / \tau_{cl})$$

$$\tau_i = \tau_p$$

When K_p and τ_p are known as a function of time, the values of K_c and τ_i can be found as function of temperature as well.

10.3 Ziegler Nichols Tuning

For the Ziegler Nichols Tuning, we use the step response of the open loop experiment.

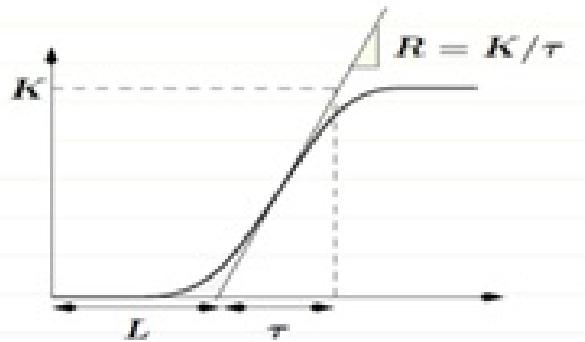


Figure 10.3: Tangent Approach to Ziegler Nichols Tuning

	K_c	τ_i	τ_d
P	$1/RL$		
PI	$0.9/RL$	$3L$	
PID	$1.2/RL$	$2L$	$.5L$

Table 10.1: Ziegler Nichols PID Settings

Table 10.1 gives the PID settings. In this approach too, for every open step test, K and τ are found and correlated as function of average temperature and PID settings are then found as functions of temperature.

Note: For a First Order transfer function that we are assuming,

- $K_p \approx K$
- $\tau_p \approx \tau$

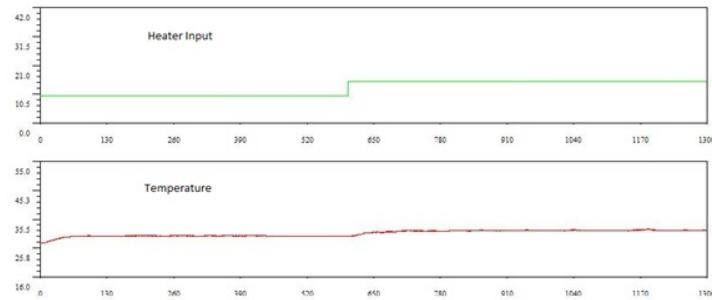


Figure 10.4: Step Response for Heater Reading 10 to 15

10.4 Step Test Experiments and Parameter Estimation

Several Open Loop step test experiments were carried out and the values of the open loop parameters were found by curve fitting. The results are shown.

10.4.1 Step Test Experiments

10.4.1.1 Step Change in Heater Reading from 10 to 15

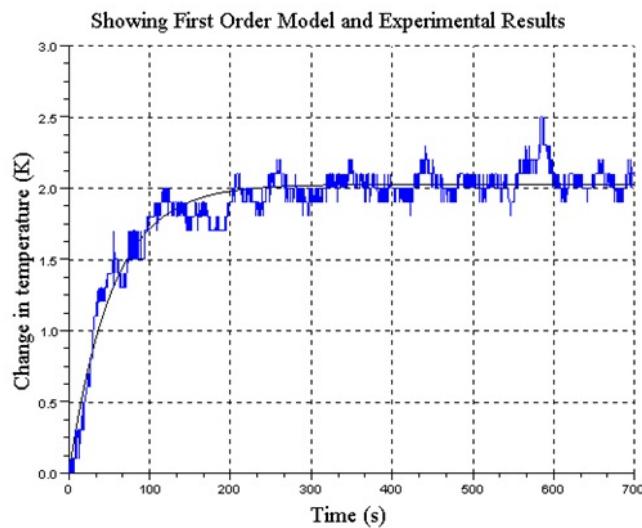


Figure 10.5: Step Response for Heater reading 10 to 15 in terms of Deviation

10.4.1.2 Step Change in Heater Reading from 20 to 25

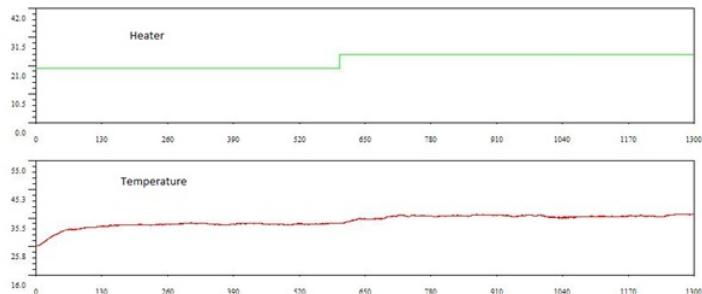


Figure 10.6: Step Responce for Heater Reading 20 to 25

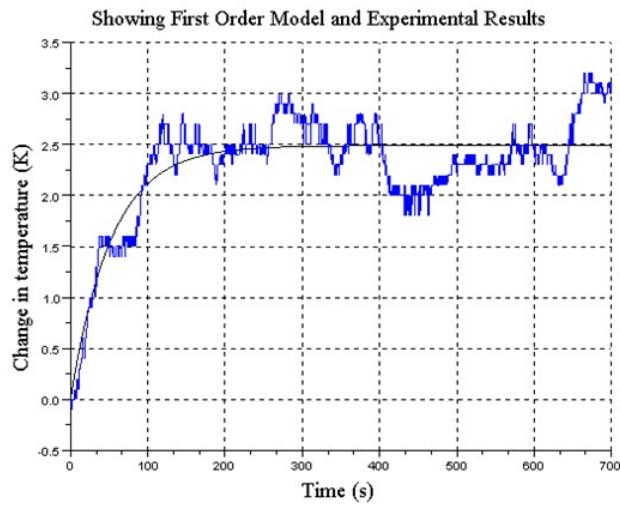


Figure 10.7: Step Response for Heater reading 20 to 25 in terms of Deviation

10.4.1.3 Step Change in Heater Reading from 30 to 35

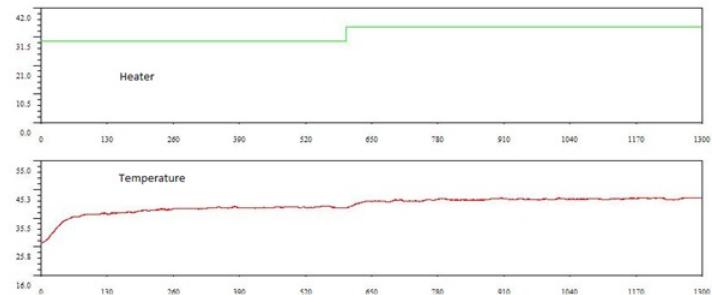


Figure 10.8: Step Responce for Heater Reading 30 to 35

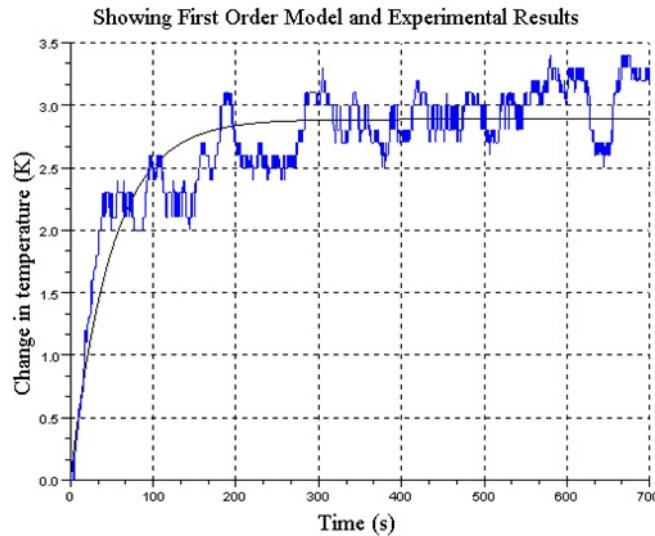


Figure 10.9: Step Response for Heater reading 30 to 35 in terms of Deviation

10.4.1.4 The Open Loop Parameters

Initial Heater Reading	Final Heater Reading	Average Temperature($^{\circ}\text{C}$)	K_p	τ_p
10	15	31.57	0.41	53.37
20	25	36.00	0.50	52.64
30	35	41.79	0.58	49.21

Table 10.2: Open Loop Parameters

It can be seen from the graphs that there is a lag of approximately 6 seconds in each experiment.

10.4.2 Conventional Controller Design

1. PI Controller using Ziegler Nichols Tuning with the results of the first step test experiment:
 - $K_c = 19.75$
 - $\tau_i = 18$

2. PID Controller using Ziegler Nichols Tuning with the results of the first step test experiment:
 - $K_c = 26.327$
 - $\tau_i = 12$
 - $\tau_d = 3$

3. PI Controller Using Direct Synthesis on the results of the second step test experiment (τ_{cl} is taken as $\tau_p/2$):
 - $K_c = 4.02$
 - $\tau_i = 52.645$

10.4.3 Self Tuning Controller Design

The graphs showing the variation of K_p and τ_p are shown below:

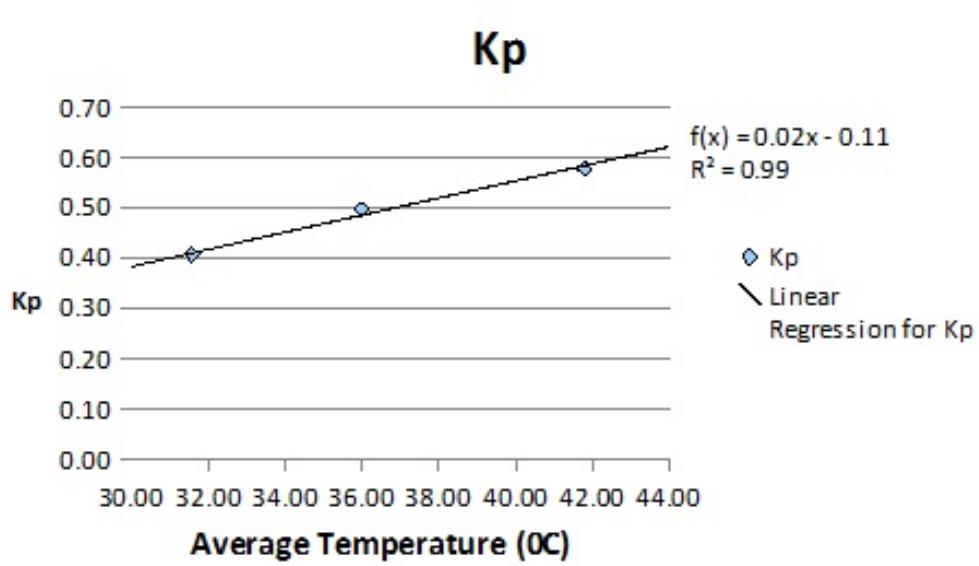


Figure 10.10: Variation of K_p with temperature

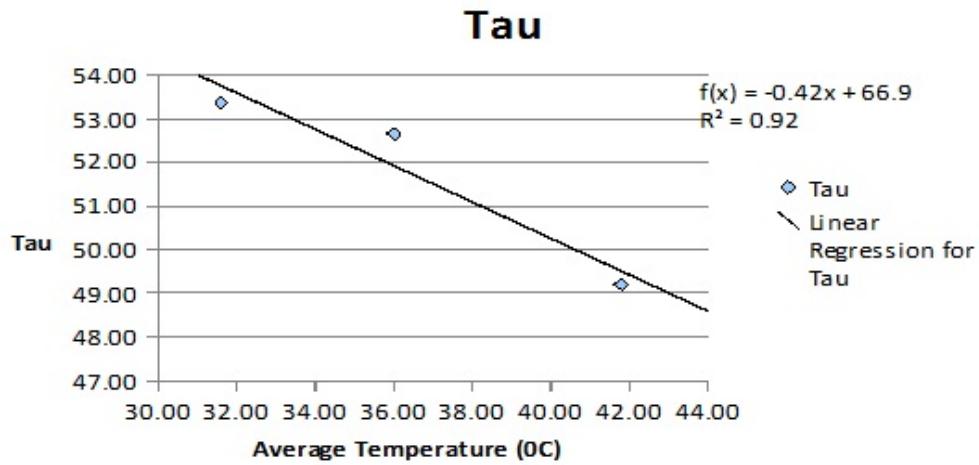


Figure 10.11: Variation of τ_p with temperature

1. PI Controller using Ziegler Nichols Tuning:

$$L = 6$$

$$R = (0.016 \times T - 0.114) / (66.90 - 0.415 \times T) \text{ where } T \text{ is the temperature}$$

$$K_c = 0.9(66.90 - 0.415T) / 6(0.016T - 0.114)$$

$$= (60.21 - 0.3735T) / (0.096T - 0.684)$$

$$\tau_i = 3 \times 6 = 18$$

2. PID Controller using Ziegler Nichols Tuning:

$$L = 6$$

$$R = (0.016 \times T - 0.114) / (66.90 - 0.415 \times T) \text{ where } T \text{ is the temperature}$$

$$K = 1.2(66.90 - 0.415T) / 6(0.016T - 0.114)$$

$$= (80.28 - 0.498T) / (0.096T - 0.684)$$

$$\tau_i = 2 \times 6 = 12$$

$$\tau_d = 0.5 \times 6 = 3$$

3. PI Controller using Direct Synthesis (τ_{cl} is taken as $\tau_p/2$):

$$K = 2/(0.016 \times T - 0.114)$$

$$\tau_i = (66.90 - 0.415 \times T) \text{ where } T \text{ is the temperature}$$

10.5 Implementation

10.5.1 PI Controller

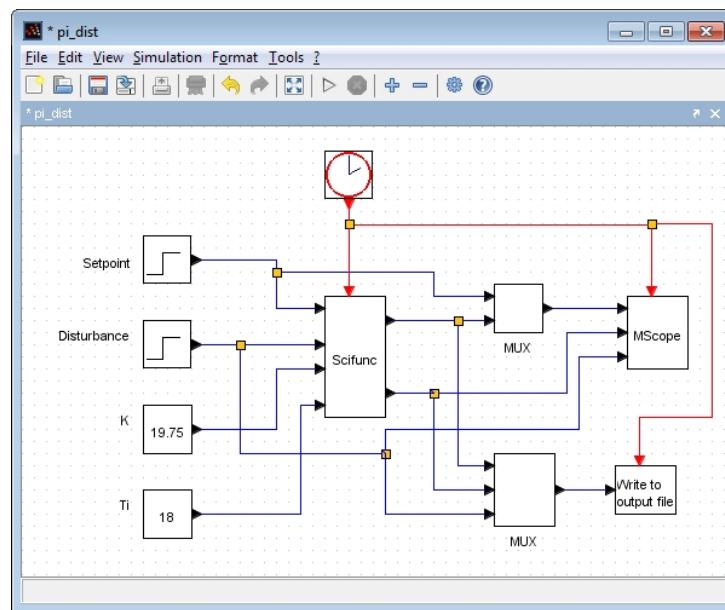


Figure 10.12: Xcos Diagram for PI Controller

The PI Controller in Continuous Time is given by:

$$u(t) = K \left[e(t) + \frac{1}{\tau_i} \int_0^t e(t) dt \right]$$

On taking Laplace Transform, we obtain:

$$u(s) = K \left[1 + \frac{1}{\tau_i s} \right] e(s)$$

By mapping the above to discrete time interval using Backward Difference Approximation

$$u(n) = K \left[1 + \frac{T_s}{\tau_i} \frac{z}{z-1} \right] e(n)$$

On Cross Multiplication, we obtain:

$$(z-1) \times u(n) = K \left[(z-1) + \frac{T_s}{\tau_i} (z) \right] e(n)$$

We devide by z, and using the shifting theorem, we obtain:

$$u(n) - u(n-1) = K \left[e(n) - e(n-1) + \frac{T_s}{\tau_i} e(n) \right]$$

The PI Controller is usually written as:

$$u(n) = u(n-1) + s_0 e(n) + s_1 e(n-1) \quad (10.6)$$

Where,

$$\begin{aligned} s_0 &= K \left(1 + \frac{T_s}{\tau_i} \right) \\ s_1 &= -K \end{aligned}$$

10.5.2 PID Controller

The PID Controller in Continuous Time is given by:

$$u(t) = K \left[e(t) + \frac{1}{\tau_i} \int_0^t e(t) dt + \tau_d \frac{de(t)}{dt} \right]$$

On taking Laplace Transform, we obtain:

$$u(s) = K \left[1 + \frac{1}{\tau_i s} + \tau_d s \right] e(s)$$

By mapping the above to discrete time interval by using the Trapezoidal Approximation for integral mode and Backward Difference Approximation for Derivative mode

$$u(n) = K \left[1 + \frac{T_s}{\tau_i} \frac{z}{z-1} + \frac{\tau_d}{T_s} \frac{z-1}{z} \right] e(n)$$

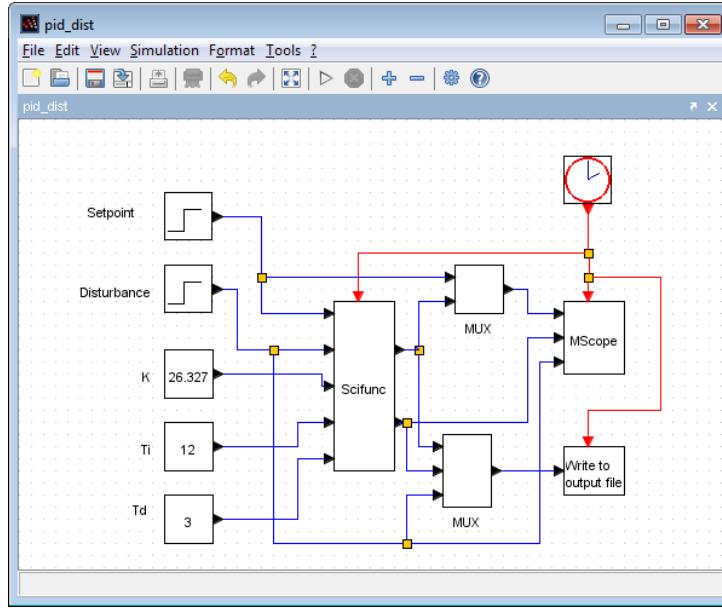


Figure 10.13: Xcos Diagram for PID Controller

On Cross Multiplication, we obtain:

$$(z^2 - z) \times u(n) = K \left[(z^2 - z) + \frac{T_s}{\tau_i} (z^2) + \frac{\tau_d}{T_s} (z - 1)^2 \right] e(n)$$

We devide by z, and using the shifting theorem, we obtain:

$$u(n) - u(n - 1) = K \left[e(n) - e(n - 1) + \frac{T_s}{\tau_i} e(n) + \frac{\tau_d}{T_s} \{e(n) - 2e(n - 1) + e(n - 2)\} \right]$$

The PID Controller is usually written as:

$$u(n) = u(n - 1) + s_0 e(n) + s_1 e(n - 1) + s_2 e(n - 2) \quad (10.7)$$

Where,

$$\begin{aligned} s_0 &= K \left(1 + \frac{T_s}{\tau_i} + \frac{\tau_d}{T_s} \right) \\ s_1 &= K \left[-1 - 2 \frac{\tau_d}{T_s} \right] \\ s_2 &= K \left[\frac{\tau_d}{T_s} \right] \end{aligned}$$

10.5.3 Self Tuning Controller

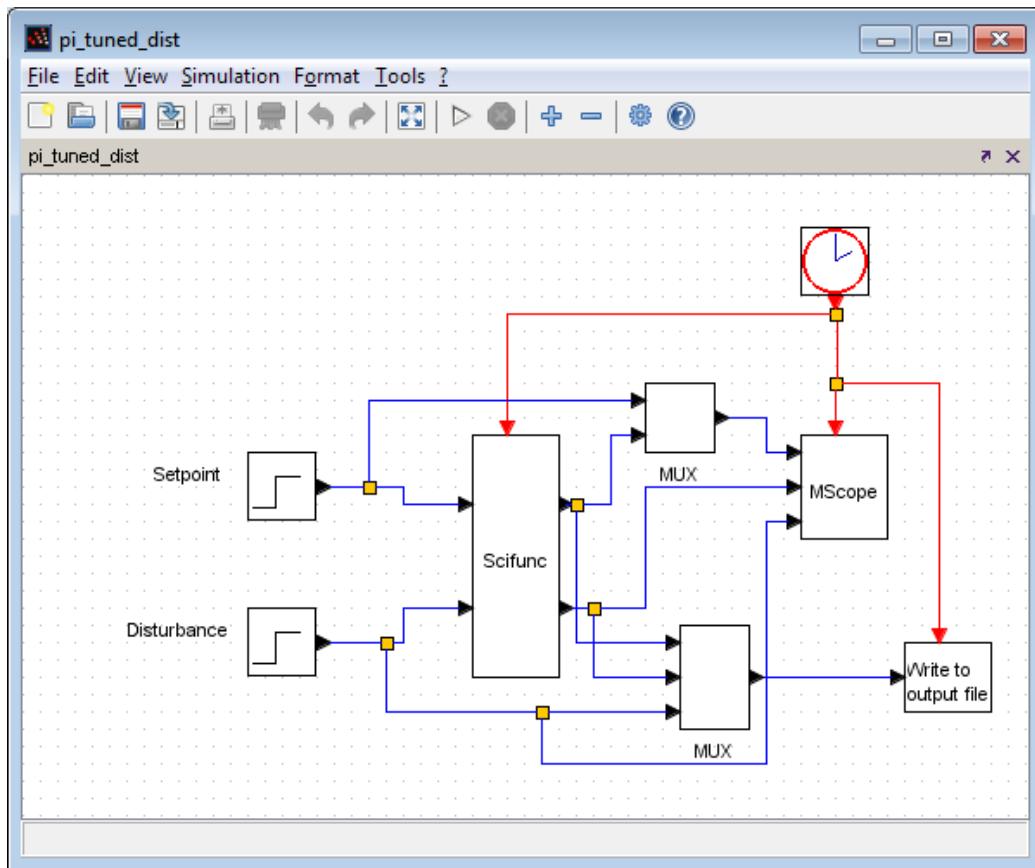


Figure 10.14: Xcos Diagram for Self Tuning Controller

The parameters of the Controller are determined dynamically using the temperature values during every sampling time. For this, the formulae derived in section 10.4.3 are used. The formulae for the control effort are same as the conventional PI and PID controllers. So the PI/PID settings are calculated for every sampling time and the control effort is calculated thereafter using the formulae derived for conventional controllers.

10.6 Set Point Tracking

The main aim of the controller is to track the set point and to reject disturbances. When the set point of the controlled variable (temperature in this case) is changed, the controller should work in such a manner that the actual temperature follows the set point as close as possible.

In this project, several experiments were conducted with the self tuning and conventional PI/PID Controllers. Table 10.3 shows the set point changes given during the various experiments that were conducted with conventional and self tuning controllers designed using several methods.

	Conventional Controller	Self Tuning Controller
Direct Synthesis PI	32°C to 37°C 35°C to 45°C	32°C to 37°C 35°C to 45°C
Ziegler Nichols PI	32°C to 37°C 35°C to 45°C 40°C to 45°C	32°C to 37°C 35°C to 45°C 35°C to 45°C
Ziegler Nichols PID	31°C to 45°C 32°C to 37°C	32°C to 46°C 32°C to 37°C

Table 10.3: Set Point Changes in experiments conducted for Set Point Tracking

10.6.1 PI Controller designed by Direct Synthesis

The results of the experiments carried out for the self tuning PI controller using direct synthesis method are shown. The upper plot shows the variations of the set point temperature (the black line) and the actual temperature (the green line) in the SBHS. The lower plot shows the control effort.

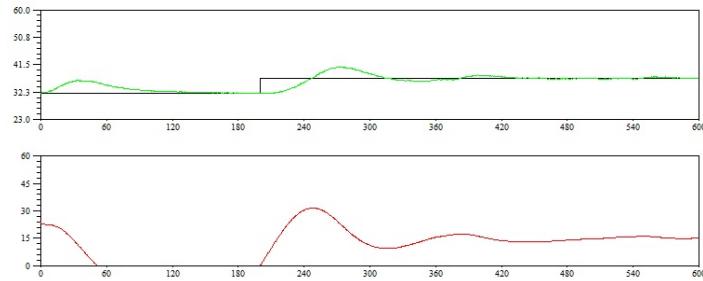


Figure 10.15: Result for Self Tuning Controller designed using Direct Synthesis for Set Point going from 32°C to 37°C

Although there is a small overshoot, the controller is able to make the actual temperature follow the set point temperature quite closely. Looking at higher values of set point changes, the result for set point change going from 35°C to 45°C is shown.

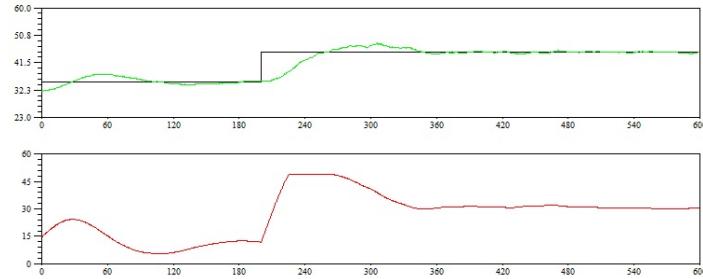


Figure 10.16: Result for Self Tuning Controller designed using Direct Synthesis for Set Point going from 35°C to 45°C

For a higher set point change also, the controller is able to make the temperature follow the set point closely. Notice the abrupt change in the control effort as soon as the step change in the set point is encountered.

For comparison, results of experiments done with conventional PI controller designed using the Direct Synthesis method are also shown.

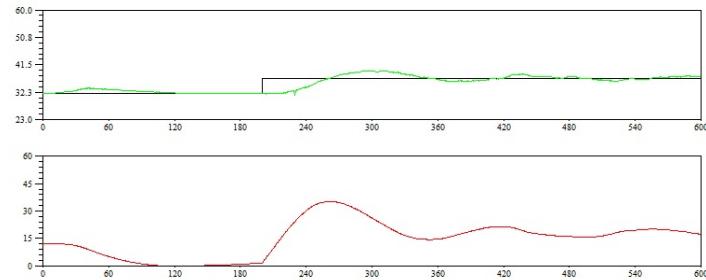


Figure 10.17: Result for Conventional Controller designed using Direct Synthesis for Set Point going from 32°C to 37°C

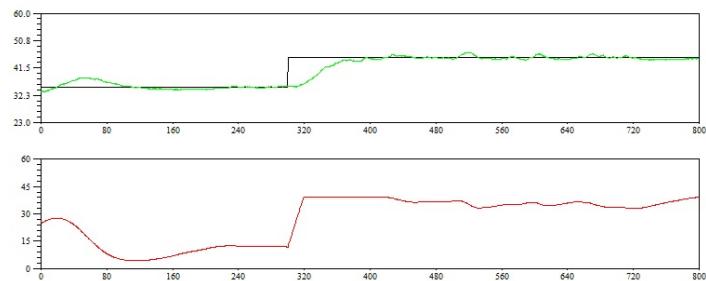


Figure 10.18: Result for Conventional Controller designed using Direct Synthesis for Set Point going from 35°C to 45°C

As can been seen from the graph, the self tuning controller stabilised the temperature faster.

10.6.2 PI Controller using Ziegler Nichols Tuning

The results of the experiments carried out for the self tuning PI controller using Ziegler Nichols tuning method are shown. The upper plot shows the variations of the set point temperature (the black line) and the actual temperature (the green line) in the SBHS. The lower plot shows the control effort.

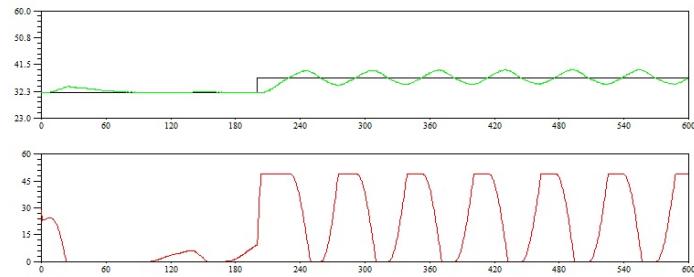


Figure 10.19: Result for Self Tuning Controller designed using Ziegler Nichols Tuning for Set Point going from 32°C to 37°C

Although there are oscillations, the temperature remains near the set point. The result for a higher value of set point change is also shown.

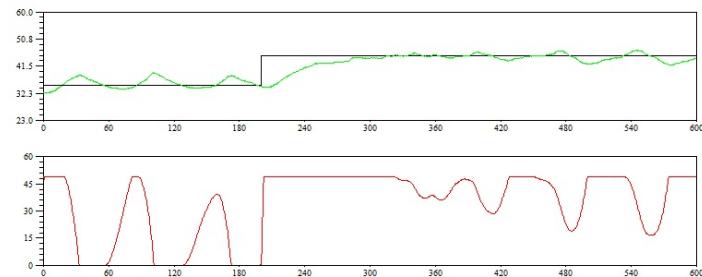


Figure 10.20: Result for Self Tuning Controller designed using Ziegler Nichols Tuning for Set Point going from 35°C to 45°C

For this experiment, the controller is able to make the temperature follow the set point closely. The fluctuations may be due to noises and the surrounding conditions. The plot for result of an experiment with another value of set point change is also shown.

In this experiment too, the controller is able to keep the temperature close to the set point and it stabilises fast.

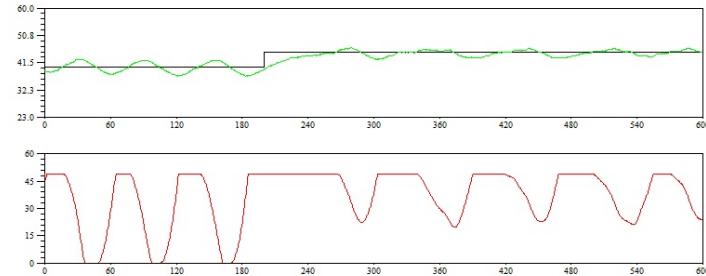


Figure 10.21: Result for Self Tuning Controller designed using Ziegler Nichols Tuning for Set Point going from 40°C to 45°C

For comparison, results of experiments done with conventional PI controller designed using the Ziegler Nichols method are also shown.

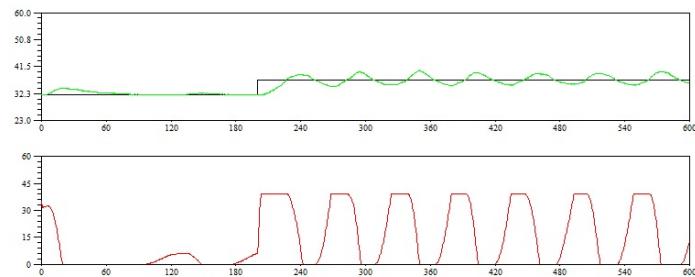


Figure 10.22: Result for Conventional Controller designed using Ziegler Nichols Tuning for Set Point going from 32°C to 37°C

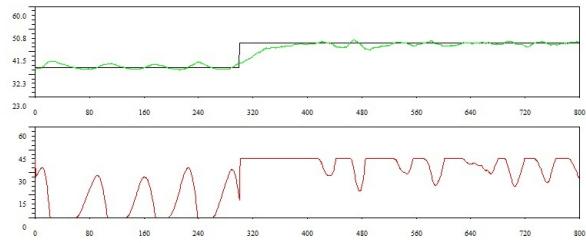


Figure 10.23: Result for Conventional Controller designed using Ziegler Nichols Tuning for Set Point going from 35°C to 45°C

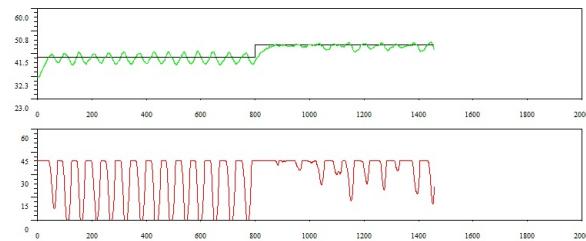


Figure 10.24: Result for Conventional Controller designed using Ziegler Nichols Tuning for Set Point going from 40°C to 45°C

For set point change from 40°C to 45°C , the self tuning controller showed small oscillations, but the conventional controller shows bigger oscillations.

10.6.3 PID Controller using Ziegler Nichols Tuning

The results of the experiments carried out for the self tuning PID controller using Ziegler Nichols tuning method are shown. The upper plot shows the variations of the set point temperature (the black line) and the actual temperature (the purple line) in the SBHS. The lower plot shows the control effort.

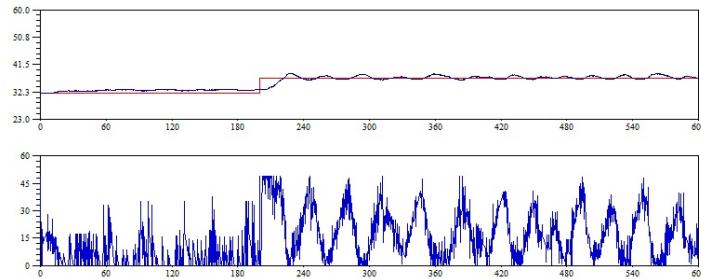


Figure 10.25: Result for Self Tuning PID Controller designed using Ziegler Nichols Tuning for Set Point going from 32°C to 37°C

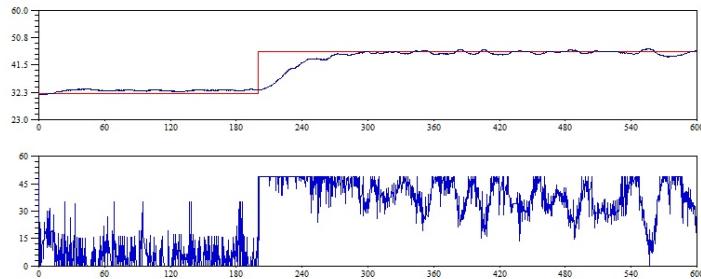


Figure 10.26: Result for Self Tuning PID Controller designed using Ziegler Nichols Tuning for Set Point going from 32°C to 46°C

From the graph it can be seen that for both the above experiments, the self tuning PID controller is able to keep the temperature close to the set point and the stabilisation is also fast. For comparison, plots for experiments conducted with conventional PID controller designed using Ziegler Nichols method are also shown.

From the above graph we can see that the conventional PID controller is not able to make the temperature close to the set point when the set point value is

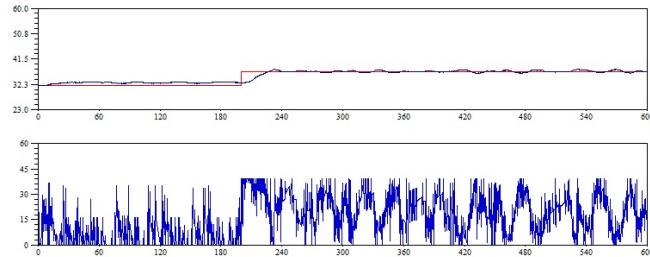


Figure 10.27: Result for Conventional PID Controller designed using Ziegler Nichols Tuning for Set Point going from 32^0C to 37^0C

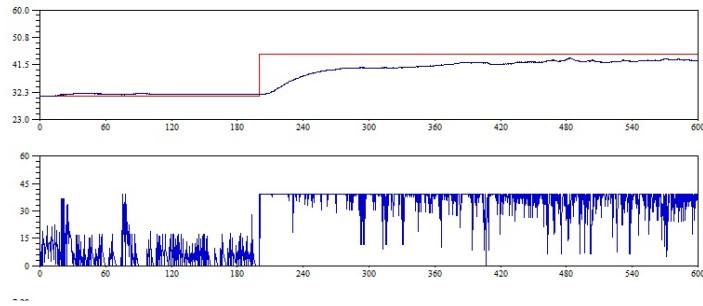


Figure 10.28: Result for Conventional PID Controller designed using Ziegler Nichols Tuning for Set Point going from 31^0C to 45^0C

45^0C . The self tuning PID controller had successfully brought the temperature to 45^0C .

10.6.4 Conclusion

The self tuning PI controller is able to accomplish the aim of keeping the temperature as close as possible to the set point. Although it may show some initial overshoot or oscillation for some values of set point change, the time needed for stabilisation is low. There may also some cases where the conventional controller shows bigger oscillations than the self tuning controller.

The PI controllers, both conventional and self tuning, show oscillations for some values of set point change. To eliminate the oscillations, when we use the PID Controller, the self tuning design definately seems to be a better option be-

cause for higher values of set point change, the self tuning PID controller shows a better performance than the conventional controller as seen in section 10.6.3.

10.7 Disturbance Rejection

Apart from tracking the set point, the system should also be able to reject disturbances. There may be several factors influencing the controlled variable and not all of them can be manipulated. Therefore, it becomes necessary for the controller not to let the changes in the non-manipulated variables to affect the controlled variable. This is called Disturbance Rejection.

In this system, the disturbance variable is the fan input. Therefore, the controller has to work in such a way that changes in the fan input doesn't affect the temperature in the SBHS.

In this project, several experiments were conducted with the self tuning and conventional PI/PID Controllers. Table 10.4 shows the fan input changes given during the various experiments that were conducted with conventional and self tuning controllers designed using several methods.

	Conventional Controller	Self Tuning Controller
Direct Synthesis PI	50 to 100 100 to 50	50 to 100 100 to 50
Ziegler Nichols PI	50 to 100 100 to 50	50 to 100 100 to 50
Ziegler Nichols PID	50 to 100 100 to 50	50 to 100 100 to 50

Table 10.4: Fan Input Changes in experiments conducted for Disturbance Rejection

10.7.1 PI Controller designed by Direct Synthesis

The results of the experiments carried out for the self tuning PI controller using direct synthesis method are shown. The upper plot shows the variations of the set point temperature (the black line) and the actual temperature (the green line) in

the SBHS. The second plot shows the control effort and the third shows the fan input.

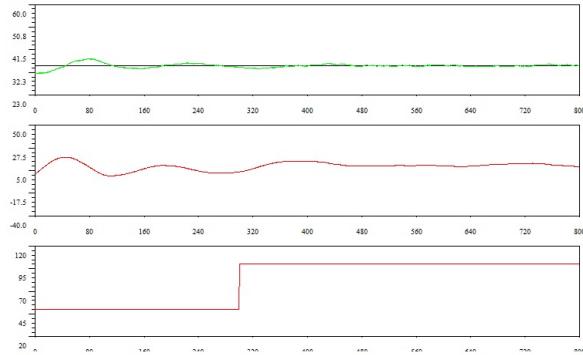


Figure 10.29: Results for Fan Input Change from 50 to 100 for Self Tuning PI Controller designed using Direct Synthesis

The change in the fan input introduces a small dent in the temperature. However, the controller brings the temperature back to the set point. Notice the slight change in the controller behaviour on encountering the fan input change. The time taken for stabilising back is also low.

Here, results for fan input change from 100 to 50 are also shown.

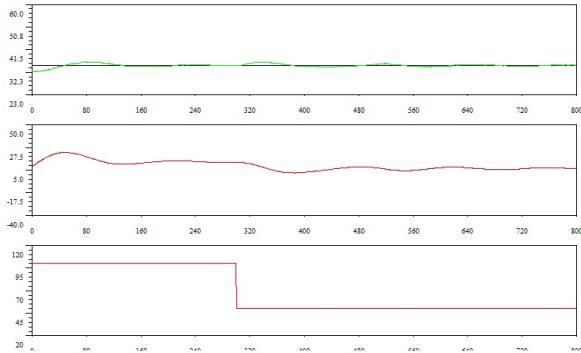


Figure 10.30: Results for Fan Input Change from 100 to 50 for Self Tuning PI Controller designed using Direct Synthesis

In this figure also, the temperature clearly increases a bit when the step change in the fan input is encountered. However, it quickly stabilises back and continues to be close to the set point.

From the above two results, it is clear that the self tuning controller designed with direct synthesis has successfully rejected the disturbance.

For comparison, results of the disturbance change for conventional PI Controller designed with direct synthesis are also shown.

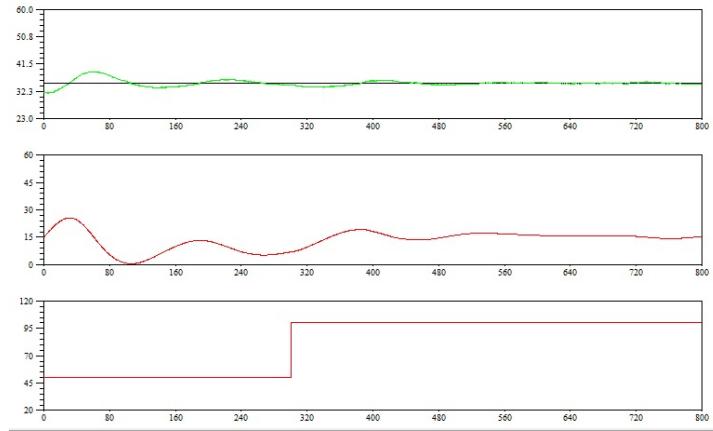


Figure 10.31: Results for the Fan input change from 50 to 100 to Conventional PI Controller designed using Direct Synthesis

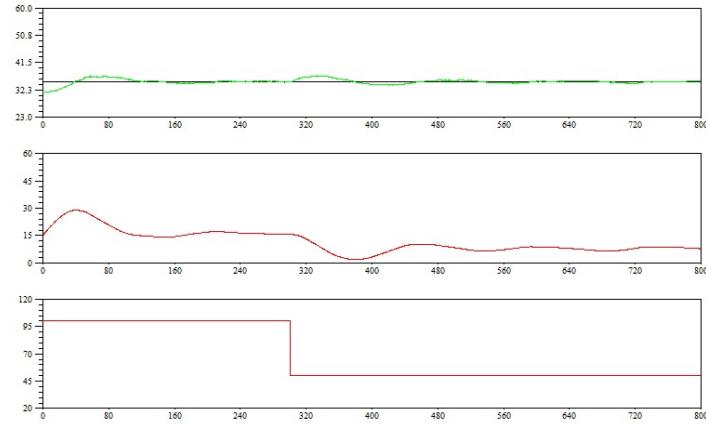


Figure 10.32: Results for the Fan input change from 100 to 50 to Conventional PI Controller designed using Direct Synthesis

10.7.2 PI Controller using Ziegler Nichols Tuning

The results of the experiments carried out for the self tuning PI controller using Ziegler Nichols method are shown. The upper plot shows the variations of the set point temperature (the black line) and the actual temperature (the green line) in the SBHS. The second plot shows the control effort and the third shows the fan input.

Even on encountering the fan input change, the temperature remains close to the set point. Notice the change in the controller behaviour on encountering the fan input change.

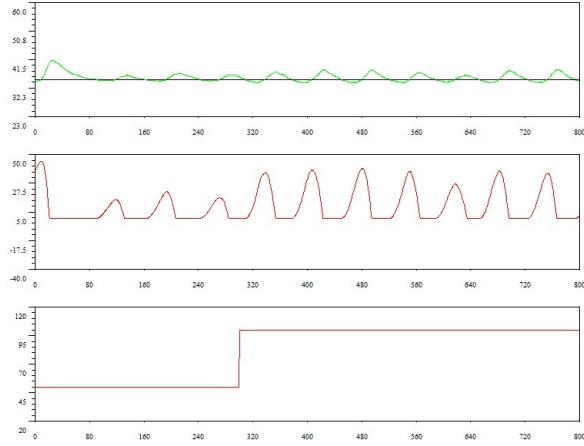


Figure 10.33: Results for Fan Input change from 50 to 100 given to Self Tuning PI Controller designed using Ziegler Nichols Method

Here, result for the fan input going from 100 to 50 is also shown.

Here, a change in the control effort can be noticed. This change has been brought by the PI Controller to keep the temperature close to the set point. From the above two results, it is clear that the self tuning controller designed with direct synthesis has successfully rejected the disturbance.

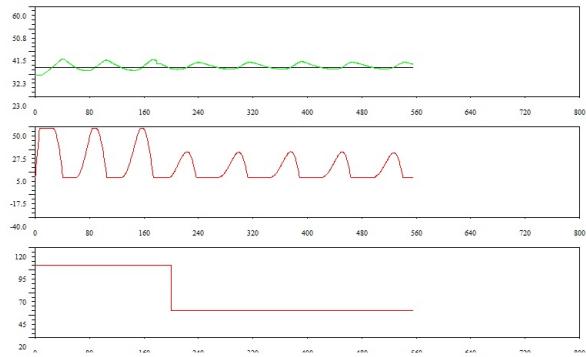


Figure 10.34: Results for Fan Input change from 100 to 50 given to Self Tuning PI Controller designed using Ziegler Nichols Method

For comparison, corresponding results are also shown for Conventional PI Controllers designed using ziegler nichols tuning.

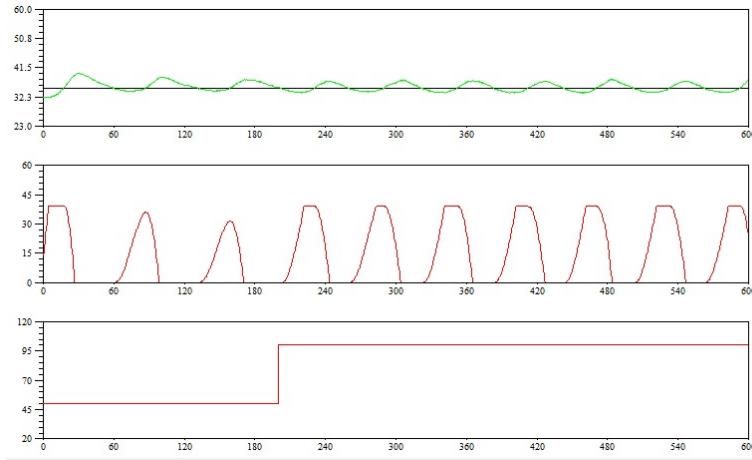


Figure 10.35: Results for the Fan input change from 50 to 100 to Conventional PI Controller designed using Ziegler Nichols Tuning

10.7.3 PID Controller using Ziegler Nichols Tuning

The results of the experiments carried out for the self tuning PID controller using Ziegler Nichols method are shown. The upper plot shows the variations of the set point temperature (the black line) and the actual temperature (the purple line) in the SBHS. The second plot shows the control effort and the third shows the fan input.

In this system also, on encountering the fan input change, the temperature remains close to the set point. Notice the change in the control effort profile when the change in the fan input is given.

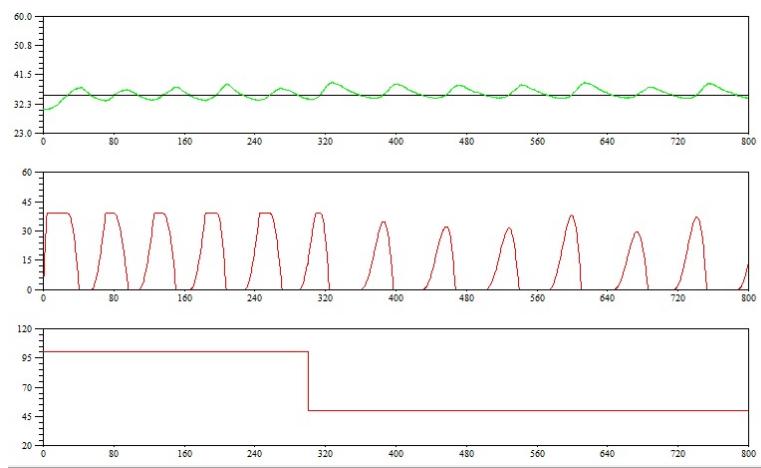


Figure 10.36: Results for the Fan input change from 100 to 50 to Conventional PI Controller designed using Ziegler Nichols Tuning

Here, result for the fan input going from 100 to 50 is also shown.

In this figure also, the temperature clearly increases a bit when the step change in the fan input is encountered. However, it quickly stabilizes back and continues to be close to the set point.

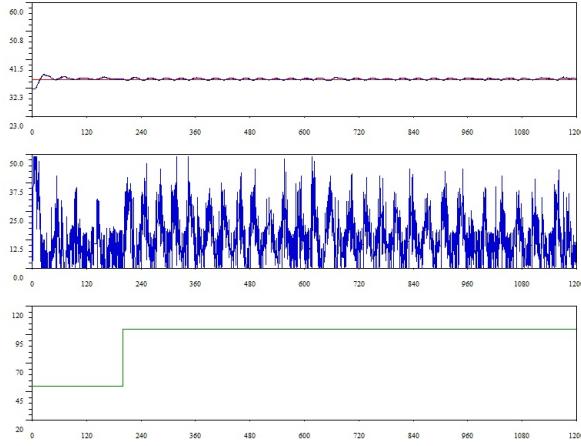


Figure 10.37: Results for Fan Input change from 50 to 100 given to Self Tuning PID Controller designed using Ziegler Nichols Method

For comparison, corresponding results are also shown for Conventional PID Controllers designed using ziegler nichols tuning.

10.7.4 Conclusion

We see that the self tuning controller manages to keep the temperature close to the set point temperature, even when the change in fan input is encountered. This shows that it can reject the disturbances quite nicely.

10.8 Reproducing the Results

The following steps can be used for conducting the experiments:

1. Open and run the program for serial communication in scilab. This opens the comm port.
2. Open and execute the sci file corresponding to the experiment that is being done.
3. Load the scicos diagram, ensure that the parameters are correct and run the experiment.

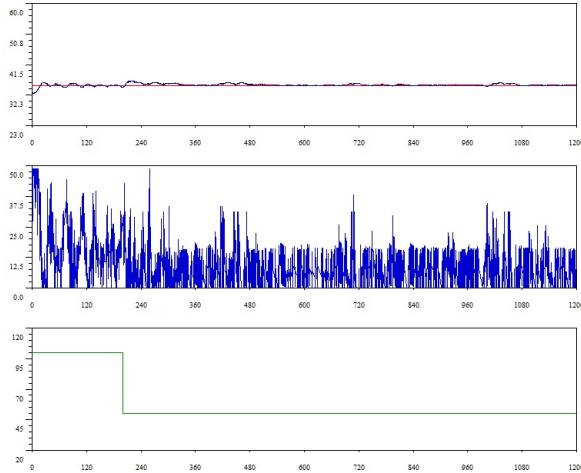


Figure 10.38: Results for Fan Input change from 100 to 50 given to Self Tuning PID Controller designed using Ziegler Nichols Method

The various codes used for conventional and self tuning controllers are shown in Section 10.8.3 and 10.8.5

10.8.1 Implementing Self Tuning controller on SBHS, virtually

The step by step procedure for conducting an experiment virtually is explained in section 3.4. The required .sce file is `pi_bda_tuned_dist_virtual.sce` for example if you want to run the PI Controller Fan disturbance experiment. Under the virtual folder there are two folders `Self_tuning_controller` and `SelfTuning_Vikas`. You will find this file in the `PIControllerFandisturbance` directory. The necessary code is listed in the section 10.8.6 and 10.8.8

10.8.2 Serial Communication

Scilab Code 10.1 `ser_init.sce`

```

1 mode(-1);
2 // To load the serial toolbox and open the serial port
3 exec loader.sce
4
5 handl = openserial(7,"9600,n,8")

```

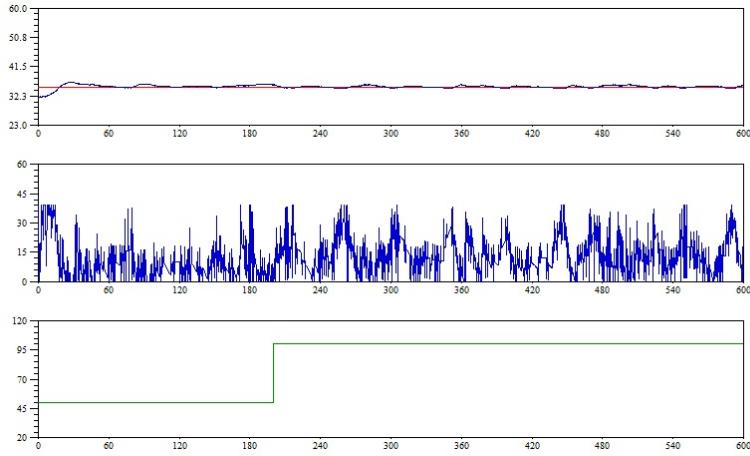


Figure 10.39: Results for the Fan input change from 50 to 100 to Conventional PID Controller designed using Ziegler Nichols Tuning

```

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
12 if (ascii(handl) ~= [])
13   disp(”COM Port Opened”);
14 end

```

10.8.3 Conventional Controller, local

10.8.4 Fan Disturbance in PI Controller

Scilab Code 10.2 pi_bda_dist.sci

```

1 mode(0);
2 // PI Controller using backward difference formula

```

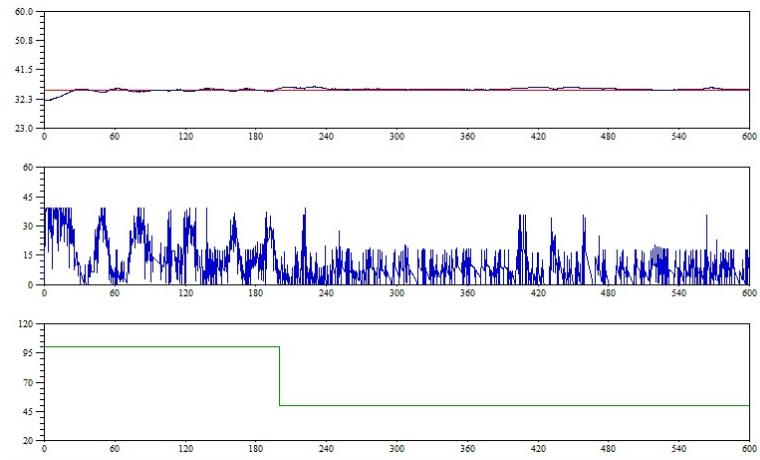


Figure 10.40: Results for the Fan input change from 100 to 50 to Conventional PID Controller designed using Ziegler Nichols Tuning

```

3 // Heater input is passed as input argument to
   introduce control effort u(n)
4 // Fan input is passed as input argument which is kept
   at constant level
5 // Range of Fan input : 20 to 252
6 // Temperature is read
7
8 // mode(-1);
9 function [temp,C0] = pi_bda_dist(setpoint,disturbance,
   K,Ti,Td)
10
11 global temp heat_in fan_in u_old u_new e_old e_new S0
   S1
12
13 e_new = setpoint - temp;
14
15 Ts=1;
16 S0=K*(1+((Ts/Ti)));
17 S1=-K;
18 u_new = u_old + S0*e_new + S1*e_old;
19

```

```

20 if u_new > 39;
21   u_new = 39;
22 end;
23
24 if u_new < 0;
25   u_new = 0;
26 end;
27
28 C0=u_new;
29 heat_in = C0;
30 fan_in = disturbance;
31 u_old = u_new;
32 e_old = e_new;
33
34 writeserial(handl,ascii(254)); // heater
35 writeserial(handl,ascii(heat_in));
36 writeserial(handl,ascii(253));
37 writeserial(handl,ascii(fan_in));
38 writeserial(handl,ascii(255));
39 sleep(10);
40 temp = ascii(readserial(handl,2));
41 temp = temp(1) + 0.1*temp(2);
42
43 endfunction;

```

10.8.4.1 Set Point Change in PI Controller

Scilab Code 10.3 pi_bda.sci

```

1 mode(0);
2 // PI Controller using backward difference formula
3 // Heater input is passed as input argument to
   introduce control effort u(n)
4 // Fan input is passed as input argument which is kept
   at constant level
5 // Range of Fan input : 20 to 252
6 // Temperature is read
7

```

```

8 function [ temp ,C0 ,e_new ] = pi_bda( setpoint ,disturbance
9   ,K, Ti )
10
11   global temp heat_in fan_in C0 u_old u_new e_old e_new
12
13   e_new = setpoint - temp ;
14
15   Ts=1;
16   S0=K*(1+((Ts/Ti)));
17   S1=-K;
18   u_new = u_old +(S0*e_new)+(S1*e_old);
19
20   if u_new> 39
21     u_new = 39;
22   end;
23
24   if u_new< 0
25     u_new = 0;
26   end;
27
28   C0=u_new;
29   heat_in = C0;
30   fan_in = disturbance;
31   u_old = u_new;
32   e_old = e_new;
33
34
35   writeserial(handl ,ascii(254)); // heater
36   writeserial(handl ,ascii(heat_in));
37   writeserial(handl ,ascii(253));
38   writeserial(handl ,ascii(fan_in));
39   writeserial(handl ,ascii(255));
40   sleep(10);
41   temp = ascii(readserial(handl ,2));
42   temp = temp(1) + 0.1*temp(2);
43
44 endfunction ;

```

10.8.4.2 Fan Disturbance to PID Controller

Scilab Code 10.4 pid_bda_dist.sci

```
1 mode(0);
2 // PID using Backward difference approximation for I
   and D
3
4 function [temp,CO] = pid_bda_dist(setpoint,disturbance
   ,K,Ti,Td)
5 global temp heat_in fan_in et SP CO eti u_old u_new
   e_old e_new e_old_old
6
7 e_new = setpoint - temp;
8
9 Ts=1;
10
11 S0=K*(1+(Ts/Ti)+(Td/Ts));
12 S1=K*(-1-((2*Td)/Ts));
13 S2=K*(Td/Ts);
14
15 u_new = u_old + S0*e_new + S1*e_old + S2*e_old_old;
16 et = setpoint - temp;
17 CO = u_new;
18
19 if CO>39
20     CO = 39;
21 end;
22
23 if CO<0
24     CO = 0;
25 end;
26
27 u_new = CO;
28
29 u_old = u_new;
```

```

30 e_old_old = e_old;
31 e_old = e_new;
32
33
34 heat_in = CO;
35 fan_in = disturbance;
36
37 writeserial(handl, ascii(254)); // heater
38 writeserial(handl, ascii(heat_in));
39 writeserial(handl, ascii(253));
40 writeserial(handl, ascii(fan_in));
41 writeserial(handl, ascii(255));
42 sleep(1);
43 temp = ascii(readserial(handl,2));
44 temp = temp(1) + 0.1*temp(2);
45
46 endfunction;

```

10.8.4.3 Set Point Change in PID Controller

Scilab Code 10.5 pid_bda.sci

```

1 mode(0);
2 // PID using Backward difference approximation for I
   and D
3
4 function [temp,CO,et] = pid_bda(setpoint,disturbance,K
   ,Ti,Td)
5 global temp heat_in fan_in et SP CO eti u_old u_new
   e_old e_new e_old_old
6
7 e_new = setpoint - temp;
8
9 Ts=1;
10
11 S0=K*(1+(Ts/Ti)+(Td/Ts));
12 S1=K*(-1-((2*Td)/Ts));
13 S2=K*(Td/Ts);

```

```

14
15 u_new = u_old + S0*e_new + S1*e_old + S2*e_old_old;
16 et = setpoint - temp;
17 CO = u_new;
18
19 if CO>39
20     CO = 39;
21 end;
22
23 if CO<0
24     CO = 0;
25 end;
26
27 u_new = CO;
28
29 u_old = u_new;
30 e_old_old = e_old;
31 e_old = e_new;
32
33
34 heat_in = CO;
35 fan_in = disturbance;
36
37 writeserial(handl, ascii(254)); // heater
38 writeserial(handl, ascii(heat_in));
39 writeserial(handl, ascii(253));
40 writeserial(handl, ascii(fan_in));
41 writeserial(handl, ascii(255));
42 sleep(1);
43 temp = ascii(readserial(handl,2));
44 temp = temp(1) + 0.1*temp(2);
45
46 endfunction;

```

10.8.5 Self Tuning Controller, local

10.8.5.1 Fan Discturbance to PI Controller

Scilab Code 10.6 pi_bda_tuned_dist.sci

```
1 mode(0);
2 // PI Controller using backward difference formula
3 // Heater input is passed as input argument to
4 // introduce control effort u(n)
5 // Fan input is passed as input argument which is kept
6 // at constant level
7 // Range of Fan input : 20 to 252
8 // Temperature is read
9
10 function [temp,C0] = pi_bda_tuned_dist(setpoint,
11 disturbance)
12
13 global temp heat_in fan_in C0 u_old u_new e_old e_new
14 // L = 6;
15 // R = (0.016 * temp - 0.114) / (66.90 - 0.415 * temp);
16 // K = 0.9 / (R * L);
17 // Ti = 3 * L;
18
19 // the above is the ziegler nichols part
20
21 K = 2/(0.016*temp-0.114);
22 Ti = (66.90-0.415*temp);
23
24 // the above is the direct synthesis part
25
26 e_new = setpoint - temp;
27 Ts=1;
28 S0=K*(1+((Ts/Ti)));
29 S1=-K;
30 u_new = u_old+(S0*e_new)+(S1*e_old);
```

```

30
31 if u_new> 100
32     u_new = 100;
33 end;
34
35 if u_new< 0
36     u_new = 0;
37 end;
38
39 if disturbance> 100
40     disturbance = 39;
41 end;
42
43 if disturbance< 0
44     disturbance = 0;
45 end;
46
47 C0=u_new;
48 heat_in = C0;
49 fan_in = disturbance;
50 u_old = u_new;
51 e_old = e_new;
52
53 writeserial(handl ,ascii(254)); // heater
54 writeserial(handl ,ascii(heat_in));
55 writeserial(handl ,ascii(253));
56 writeserial(handl ,ascii(fan_in));
57 writeserial(handl ,ascii(255));
58 sleep(10);
59 temp = ascii(readserial(handl ,2));
60 temp = temp(1) + 0.1*temp(2);
61
62 endfunction;

```

10.8.5.2 Set Point Change to PI Controller

Scilab Code 10.7 pi_bda_tuned.sci

```

1 mode(0);
2 // PI Controller using backward difference formula
3 // Heater input is passed as input argument to
4 // introduce control effort u(n)
5 // Fan input is passed as input argument which is kept
6 // at constant level
7 // Range of Fan input :20 to 252
8 // Temperature is read
9
10 function [temp,C0,e_new] = pi_bda_tuned(setpoint,
11 disturbance)
12
13 global temp heat_in fan_in C0 u_old u_new e_old e_new
14 // L = 6;
15 // R = (0.016 * temp - 0.114) / (66.90 - 0.415 * temp);
16 // K = 0.9 / (R * L);
17 // Ti = 3 * L;
18
19 // The above is the Ziegler nichols part.
20 K = 2/(0.016*temp-0.114);
21 Ti = (66.90-0.415*temp);
22 // The above is the direct synthesis part
23 e_new = setpoint - temp;
24
25 Ts=1;
26 S0=K*(1+((Ts/Ti)));
27 S1=-K;
28 u_new = u_old+(S0*e_new)+(S1*e_old);
29
30
31 if u_new> 39;
32 u_new = 39;
33 end;
34
35 if u_new< 0;

```

```

36     u_new = 0;
37 end;
38
39 C0=u_new;
40 heat_in = C0;
41 fan_in = disturbance;
42 u_old = u_new;
43 e_old = e_new;
44
45 writeserial(handl,ascii(254)); // heater
46 writeserial(handl,ascii(heat_in));
47 writeserial(handl,ascii(253));
48 writeserial(handl,ascii(fan_in));
49 writeserial(handl,ascii(255));
50 sleep(10);
51 temp = ascii(readserial(handl,2));
52 temp = temp(1) + 0.1*temp(2);
53
54 endfunction;

```

10.8.5.3 Fan Disturbance to PID Controller

Scilab Code 10.8 pid_bda_tuned_dist.sci

```

1 mode(0);
2 // PID using Backward difference approximation for I
   and D
3
4 function [temp,CO] = pid_bda_tuned_dist(setpoint,
   disturbance)
5 global temp heat_in fan_in et SP CO eti u_old u_new
   e_old e_new e_old_old
6 L = 6;
7 R = (0.016*temp - 0.114)/(66.90 - 0.415*temp);
8 K = 1.2/(R*L)
9 Ti = 2*L;
10 Td = 0.5*L;
11

```

```

12
13 e_new = setpoint - temp;
14
15 Ts=1;
16
17 S0=K*(1+(Ts/Ti)+(Td/Ts));
18 S1=K*(-1-((2*Td)/Ts));
19 S2=K*(Td/Ts);
20
21 u_new = u_old + S0*e_new + S1*e_old + S2*e_old_old;
22 et = setpoint - temp;
23 CO = u_new;
24
25 if CO>39
26     CO = 39;
27 end;
28
29 if CO<0
30     CO = 0;
31 end;
32
33 u_new = CO;
34
35 u_old = u_new;
36 e_old_old = e_old;
37 e_old = e_new;
38
39
40 heat_in = CO;
41 fan_in = disturbance;
42
43 writeserial(handl,ascii(254)); // heater
44 writeserial(handl,ascii(heat_in));
45 writeserial(handl,ascii(253));
46 writeserial(handl,ascii(fan_in));
47 writeserial(handl,ascii(255));
48 sleep(1);
49 temp = ascii(readserial(handl,2));

```

```

50 temp = temp(1) + 0.1*temp(2);
51
52 endfunction;

```

10.8.5.4 Set Point Change to PID Controller

Scilab Code 10.9 pid_bda_tuned.sci

```

1 mode(0);
2 // PID using Backward difference approximation for I
   and D
3
4 function [temp,CO,et] = pid_bda_tuned(setpoint,
   disturbance)
5 global temp heat_in fan_in et SP CO eti u_old u_new
   e_old e_new e_old_old
6 L = 6;
7 R = (0.016*temp - 0.114)/(66.90 - 0.415*temp);
8 K = 1.2/(R*L)
9 Ti = 2*L;
10 // Kc and tau_i calculated
11 Td = 0.5*L;
12
13
14 e_new = setpoint - temp;
15
16 Ts = 1;
17
18 S0 = K * (1 + (Ts / Ti) + (Td / Ts));
19 S1 = K * (-1 - ((2 * Td) / Ts));
20 S2 = K * (Td / Ts);
21
22 u_new = u_old + S0 * e_new + S1 * e_old + S2 * e_old_old;
23 et = setpoint - temp;
24 CO = u_new;
25
26 if CO > 39
27   CO = 39;

```

```

28     end ;
29
30     if CO<0
31         CO = 0;
32     end ;
33
34     u_new = CO;
35
36     u_old = u_new;
37     e_old_old = e_old ;
38     e_old = e_new ;
39
40
41     heat_in = CO;
42     fan_in = disturbance ;
43
44     writeserial(handl , ascii(254)); // heater
45     writeserial(handl , ascii(heat_in));
46     writeserial(handl , ascii(253));
47     writeserial(handl , ascii(fan_in));
48     writeserial(handl , ascii(255));
49     sleep(1);
50     temp = ascii(readserial(handl ,2));
51     temp = temp(1) + 0.1*temp(2);
52
53 endfunction ;

```

10.8.6 Conventional Controller, virtual

10.8.7 Fan Disturbance in PI Controller

Scilab Code 10.10 pi_bda_dist.sci

```

1 mode(0);
2 // PI Controller using backward difference formula
3 // Heater input is passed as input argument to
   introduce control effort u(n)

```

```

4 // Fan input is passed as input argument which is kept
5 at constant level
6 // Range of Fan input : 20 to 252
7 // Temperature is read
8 // mode( -1 );
9 function [temp,heat,stop] = pi_bda_dist(setpoint,
    disturbance,K,Ti,Td)
10 global temp heat fan u_old u_new e_old e_new S0 S1
      fdfh fdt fnrcr fnrw m_err_count stop
11
12
13 fnrcr = 'clientread.sce'; // file to be read -
    temperature
14 fnrw = 'clientwrite.sce'; // file to be written
    - heater , fan
15
16 a = mgetl(fdt,1);
17 b = evstr(a);
18 byte = mtell(fdt);
19 mseek(byte,fdt,'set');
20
21 if a~= []
22     temp = b(1,$); heats = b(1,$-2);
23     fans = b(1,$-1); y = temp;
24
25 e_new = setpoint - temp;
26
27 Ts=1;
28 S0=K*(1+((Ts/Ti)));
29 S1=-K;
30 u_new = u_old + S0*e_new + S1*e_old;
31
32 if u_new> 39
33     u_new = 39;
34 end;
35
36 if u_new< 0

```

```

37     u_new = 0;
38 end;
39
40 heat=u_new;
41 fan = disturbance;
42
43
44 u_old = u_new;
45 e_old = e_new;
46
47
48 A = [m,m,heat ,fan ];
49 fdfh = file('open','clientwrite.sce','unknown');
50 file('last',fdfh)
51 write(fdfh,A,'(7(e11.5,1x))');
52 file('close',fdfh);
53 m = m+1;
54
55 else
56 y = 0;
57 err_count = err_count + 1; // counts the no of
      times network error occurs
58 if err_count > 300
      disp("NO NETWORK COMMUNICATION!");
59 stop = 1; // status set for stopping simulation
60 end
61
62 end
63
64 return
65 endfunction

```

10.8.7.1 Set Point Change in PI Controller

Scilab Code 10.11 pi_bda.sci

```

1 mode(0);
2 // PI Controller using backward difference formula

```

```

3 // Heater input is passed as input argument to
   introduce control effort u(n)
4 // Fan input is passed as input argument which is kept
   at constant level
5 // Range of Fan input :20 to 252
6 // Temperature is read
7
8 function [temp,heat,e_new,stop] = pi_bda(setpoint,
   disturbance,K,Ti)
9 global temp heat fan C0 u_old u_new e_old e_new fdfh
   fdt fncre fncw m err_count stop
10
11
12 fncre = 'clientread.sce'; // file to be read -
   temperature
13 fncw = 'clientwrite.sce'; // file to be written
   - heater , fan
14
15 a = mgetl(fdt,1);
16 b = evstr(a);
17 byte = mtell(fdt);
18 mseek(byte,fdt,'set');

19
20 if a~= []
21     temp = b(1,$); heats = b(1,$-2);
22     fans = b(1,$-1); y = temp;
23
24 e_new = setpoint - temp;
25
26 Ts=1;
27 S0=K*(1+((Ts/Ti)));
28 S1=-K;
29 u_new = u_old+(S0*e_new)+(S1*e_old);
30
31 if u_new> 39
32     u_new = 39;
33 end;
34

```

```

35 if u_new < 0
36     u_new = 0;
37 end;
38
39 heat=u_new;
40 fan = disturbance;
41
42
43 u_old = u_new;
44 e_old = e_new;
45
46
47 A = [m,m,heat ,fan ];
48 fdfh = file('open','clientwrite.sce','unknown');
49 file('last',fdfh)
50 write(fdfh,A,'(7(e11.5,1x))');
51 file('close',fdfh);
52 m = m+1;
53
54 else
55     y = 0;
56     err_count = err_count + 1; // counts the no of
      times network error occurs
57 if err_count > 300
58     disp("NO NETWORK COMMUNICATION!");
59 stop = 1; // status set for stopping simulation
60 end
61 end
62
63 return
64 endfunction

```

10.8.7.2 Fan Disturbance to PID Controller

Scilab Code 10.12 pid_bda_dist.sci

```

1 mode(0);

```

```

2 // PID using Backward difference approximation for I
   and D
3
4 function [temp,heat,stop] = pid_bda_dist(setpoint,
   disturbance,K,Ti,Td)
5 global temp heat fan et SP CO eti u_old u_new e_old
   e_new e_old_old fdfh fdt fncl fnclw m err_count stop
6
7
8 fncl = 'clientread.sce'; // file to be read -
   temperature
9 fnclw = 'clientwrite.sce'; // file to be written
   - heater , fan
10
11 a = mgetl(fdt,1);
12 b = evstr(a);
13 byte = mtell(fdt);
14 mseek(byte,fdt,'set');
15
16 if a~= []
17   temp = b(1,$); heats = b(1,$-2);
18   fans = b(1,$-1); y = temp;
19
20 e_new = setpoint - temp;
21
22 Ts=1;
23
24 S0=K*(1+(Ts/Ti)+(Td/Ts));
25 S1=K*(-1-((2*Td)/Ts));
26 S2=K*(Td/Ts);
27
28 u_new = u_old + S0*e_new + S1*e_old + S2*e_old_old;
29 if u_new> 39
30   u_new = 39;
31 end;
32
33 if u_new< 0
34   u_new = 0;

```

```

35 end ;
36
37 heat=u_new;
38 fan = disturbance;
39
40 u_old = u_new;
41 e_old_old = e_old;
42 e_old = e_new;
43
44 A = [m,m,heat ,fan ];
45 fdfh = file('open','clientwrite.sce','unknown');
46 file('last',fdfh)
47 write(fdfh ,A,'(7(e11.5,1x))');
48 file('close',fdfh);
49 m = m+1;
50
51 else
52     y = 0;
53     err_count = err_count + 1; // counts the no of
      times network error occurs
54     if err_count > 300
55         disp("NO NETWORK COMMUNICATION!");
56         stop = 1; // status set for stopping simulation
57     end
58 end
59
60 return
61 endfunction

```

10.8.7.3 Set Point Change in PID Controller

Scilab Code 10.13 pid_bda.sci

```

1 mode(0);
2 // PID using Backward difference approximation for I
   and D
3

```

```

4  function [temp ,heat ,et ,stop] = pid_bda( setpoint ,
      disturbance ,K,Ti ,Td)
5  global temp heat fan et SP CO eti u_old u_new e_old
      e_new e_old_old fdfh fdt fnrc fncw m err_count stop
6
7
8  fncr = 'clientread.sce'; // file to be read -
      temperature
9  fncw = 'clientwrite.sce'; // file to be written
      - heater , fan
10
11 a = mgetl(fdt ,1);
12 b = evstr(a);
13 byte = mtell(fdt);
14 mseek(byte ,fdt , 'set');
15
16 if a~= []
17     temp = b(1,$); heats = b(1,$-2);
18     fans = b(1,$-1); y = temp;
19
20 e_new = setpoint - temp;
21
22 Ts=1;
23
24 S0=K*(1+(Ts/Ti)+(Td/Ts));
25 S1=K*(-1-((2*Td)/Ts));
26 S2=K*(Td/Ts);
27
28 u_new = u_old + S0*e_new + S1*e_old + S2*e_old_old;
29
30 if u_new> 39
31     u_new = 39;
32 end;
33
34 if u_new< 0
35     u_new = 0;
36 end;
37

```

```

38 heat=u_new;
39 fan = disturbance;
40
41 u_old = u_new;
42 e_old_old = e_old;
43 e_old = e_new;
44
45
46 A = [m,m,heat ,fan ];
47 fdfh = file('open','clientwrite.sce','unknown');
48 file('last',fdfh)
49 write(fdfh,A,'(7(e11.5,1x))');
50 file('close',fdfh);
51 m = m+1;
52
53 else
54 y = 0;
55 err_count = err_count + 1; // counts the no of
      times network error occurs
56 if err_count > 300
      disp("NO NETWORK COMMUNICATION!");
57 stop = 1; // status set for stopping simulation
58 end
59
60 end
61
62 return
63 endfunction

```

10.8.8 Self Tuning Controller, local

10.8.8.1 Fan Discturbance to PI Controller

Scilab Code 10.14 pi_bda_tuned_dist.sci

```

1 mode(0);
2 // PI Controller using backward difference formula
3 // Heater input is passed as input argument to
   introduce control effort u(n)

```

```

4 // Fan input is passed as input argument which is kept
5 at constant level
6 // Range of Fan input : 20 to 252
7 // Temperature is read
8
9 function [temp,heat,stop] = pi_bda_tuned_dist(setpoint
10 ,disturbance)
11 global temp heat fan C0 u_old u_new e_old e_new fdfh
12 fdt fnrcr fnrw m err_count stop
13
14
15 fnrcr = 'clientread.sce'; // file to be read -
16 fnrw = 'clientwrite.sce'; // file to be written
17 - heater, fan
18
19 a = mgetl(fdt,1);
20 b = evstr(a);
21 byte = mtell(fdt);
22 mseek(byte,fdt,'set');
23
24 if a~= []
25     temp = b(1,$); heats = b(1,$-2);
26     fans = b(1,$-1); y = temp;
27
28 e_new = setpoint - temp;
29
30 Ts=1;
31 S0=K*(1+((Ts/Ti)));
32 S1=-K;
33 u_new = u_old+(S0*e_new)+(S1*e_old);
34
35 if u_new> 39
36     u_new = 39;

```

```

37 end ;
38
39 if u_new < 0
40     u_new = 0;
41 end ;
42
43 heat=u_new;
44 fan = disturbance;
45
46 u_old = u_new;
47 e_old = e_new;
48
49 A = [m,m,heat ,fan ];
50 fdfh = file('open','clientwrite.sce','unknown');
51 file('last',fdfh)
52 write(fdfh,A,'(7(e11.5,1x))');
53 file('close',fdfh);
54 m = m+1;
55
56 else
57     y = 0;
58     err_count = err_count + 1; // counts the no of
      times network error occurs
59     if err_count > 300
60         disp("NO NETWORK COMMUNICATION!");
61         stop = 1; // status set for stopping simulation
62     end
63 end
64
65 return
66 endfunction

```

10.8.8.2 Set Point Change to PI Controller

Scilab Code 10.15 pi_bda_tuned.sci

```

1 mode(0);
2 // PI Controller using backward difference formula

```

```

3 // Heater input is passed as input argument to
   introduce control effort u(n)
4 // Fan input is passed as input argument which is kept
   at constant level
5 // Range of Fan input :20 to 252
6 // Temperature is read
7
8 function [temp,heat,e_new,stop] = pi_bda_tuned(
   setpoint,disturbance)
9 global temp heat fan C0 u_old u_new e_old e_new fdfh
   fdt fnr fncw m err_count stop
10
11 // L = 6;
12 // R = (0.016 * temp - 0.114) / (66.90 - 0.415 * temp);
13 // K = 0.9 / (R * L);
14 // Ti = 3 * L;
15
16
17 // The above is the Ziegler nichols part.
18
19 K = 2/(0.016*temp-0.114);
20 Ti = (66.90-0.415*temp);
21 // The above is the direct synthesis part
22
23
24 fnr = 'clientread.sce'; // file to be read -
   temperature
25 fncw = 'clientwrite.sce'; // file to be written
   - heater , fan
26
27 a = mgetl(fdt,1);
28 b = evstr(a);
29 byte = mtell(fdt);
30 mseek(byte,fdt,'set');
31
32 if a~= []
33   temp = b(1,$); heats = b(1,$-2);
34   fans = b(1,$-1); y = temp;

```

```

35
36 e_new = setpoint - temp;
37
38 Ts=1;
39 S0=K*(1+((Ts/Ti)));
40 S1=-K;
41 u_new = u_old+(S0*e_new)+(S1*e_old);
42
43
44 if u_new> 39
45 u_new = 39;
46 end;
47
48 if u_new< 0
49 u_new = 0;
50 end;
51
52 heat=u_new;
53 fan = disturbance;
54
55 u_old = u_new;
56 e_old = e_new;
57
58 A = [m,m,heat ,fan ];
59 fdfh = file('open','clientwrite.sce','unknown');
60 file('last',fdfh)
61 write(fdfh,A,'(7(e11.5,1x))');
62 file('close',fdfh);
63 m = m+1;
64
65 else
66 y = 0;
67 err_count = err_count + 1; // counts the no of
times network error occurs
68 if err_count > 300
69 disp("NO NETWORK COMMUNICATION!");
70 stop = 1; // status set for stopping simulation
71 end

```

```

72     end
73
74 return
75 endfunction
```

10.8.8.3 Fan Disturbance to PID Controller

Scilab Code 10.16 pid_bda_tuned_dist.sci

```

1 mode(0);
2 // PID using Backward difference approximation for I
   and D
3
4 function [temp,heat,stop] = pid_bda_tuned_dist(
   setpoint,disturbance)
5 global temp heat fan et SP CO eti u_old u_new e_old
   e_new e_old_old fdfh fdt fncre fncre m err_count stop
6 L = 6;
7 R = (0.016*temp - 0.114)/(66.90 - 0.415*temp);
8 K = 1.2/(R*L)
9 Ti = 2*L;
10 Td = 0.5*L;
11
12
13 fncre = 'clientread.sce'; // file to be read -
   temperature
14 fncre = 'clientwrite.sce'; // file to be written
   - heater , fan
15
16 a = mgetl(fdt,1);
17 b = evstr(a);
18 byte = mtell(fdt);
19 mseek(byte,fdt,'set');
20
21 if a~= []
22   temp = b(1,$); heats = b(1,$-2);
23   fans = b(1,$-1); y = temp;
24
```

```

25 e_new = setpoint - temp;
26
27 Ts=1;
28
29 S0=K*(1+(Ts/Ti)+(Td/Ts));
30 S1=K*(-1-((2*Td)/Ts));
31 S2=K*(Td/Ts);
32
33 u_new = u_old + S0*e_new + S1*e_old + S2*e_old_old;
34
35
36 if u_new> 39
37     u_new = 39;
38 end;
39
40 if u_new< 0
41     u_new = 0;
42 end;
43
44 heat=u_new;
45 fan = disturbance;
46
47 u_old = u_new;
48 e_old_old = e_old;
49 e_old = e_new;
50
51 A = [m,m,heat ,fan ];
52 fdfh = file('open','clientwrite.sce','unknown');
53 file('last',fdfh)
54 write(fdfh,A,'(7(e11.5,1x))');
55 file('close',fdfh);
56 m = m+1;
57
58 else
59     y = 0;
60     err_count = err_count + 1; // counts the no of
       times network error occurs
61 if err_count > 300

```

```

62 disp ("NO NETWORK COMMUNICATION!") ;
63 stop = 1; // status set for stopping simulation
64 end
65 end
66
67 return
68 endfunction

```

10.8.8.4 Set Point Change to PID Controller

Scilab Code 10.17 pid_bda_tuned.sci

```

1 mode(0);
2 // PID using Backward difference approximation for I
3 and D
4
5 function [temp,heat,et,stop] = pid_bda_tuned(setpoint,
6 disturbance)
7 global temp heat fan et SP CO eti u_old u_new e_old
8 e_new e_old_old fdfh fdt fncrefnew m err_count stop
9 L = 6;
10 R = (0.016*temp -0.114)/(66.90 -0.415*temp);
11 K = 1.2/(R*L);
12 Ti = 2*L;
13
14 // Kc and tau_i calculated
15 Td = 0.5*L;
16
17 fncread = 'clientread.sce'; // file to be read -
18 temperature
19 fncrewrite = 'clientwrite.sce'; // file to be written
20 heater, fan
21
22 a = mgetl(fdt,1);
23 b = evstr(a);
24 byte = mtell(fdt);
25 mseek(byte,fdt,'set');

```

```

22  if a~= []
23      temp = b(1,$); heats = b(1,$-2);
24      fans = b(1,$-1); y = temp;
25
26 e_new = setpoint - temp;
27
28 Ts=1;
29
30 S0=K*(1+(Ts/Ti)+(Td/Ts));
31 S1=K*(-1-((2*Td)/Ts));
32 S2=K*(Td/Ts);
33
34 u_new = u_old + S0*e_new + S1*e_old + S2*e_old_old;
35
36
37 if u_new> 39
38     u_new = 39;
39 end;
40
41 if u_new< 0
42     u_new = 0;
43 end;
44
45 heat=u_new;
46 fan = disturbance;
47
48 u_old = u_new;
49 e_old_old = e_old;
50 e_old = e_new;
51
52 A = [m,m,heat ,fan ];
53 fdfh = file('open','clientwrite.sce','unknown');
54 file('last',fdfh)
55 write(fdfh,A,'(7(e11.5,1x))');
56 file('close',fdfh);
57 m = m+1;
58
59 else

```

```
60     y = 0;
61     err_count = err_count + 1; // counts the no of
       times network error occurs
62     if err_count > 300
63         disp("NO NETWORK COMMUNICATION!");
64         stop = 1; // status set for stopping simulation
65     end
66 end
67
68 return
69 endfunction
```

Chapter 11

Model Predictive Control in Single Board Heater System using SCILAB

This chapter presents Model Predictive Control in Single Board Heater System done by Mr. Pratik Behera.¹

11.1 Objective

- To implement Model Predictive Control (MPC) in Single Board Heater System using Scilab and perform experiments using it
- To perform experiments for various values of tuning parameters and study its effect on the system

11.2 Single Board Heater System

It is a single heater system, where in, 5cm x 2cm stainless steel blade acts as a plant, which is heated by a heating coil and cooled by a fan.

For Single Board Heater System (SBHS):

¹Copyright: Mr.Pratik Behera

- Control variable: temperature
- Manipulated variable: heater
- Disturbance variable: fan

The heater element consists of Nichrome wire - of 0.7mm diameter, wound with 20 equally spaced helical turns into a coil of 5mm x 11mm. The heater element is kept at a distance of 3.5 mm from the steel blade.

Cooling is done by a computer fan, which is placed below the stainless steel blade.

11.3 Model Predictive Control

An equivalent quadratic programming (QP) formulation for constrained DMC (as given in LQG_MPC_notes by Prof Sachin Patwardhan) is given as follows

$$\min_{U_f} \frac{1}{2} U_f(k)^T H U_f(k) + F^T U_f(k) \quad (11.1)$$

Subject to

$$A U_f(k) \leq b \quad (11.2)$$

where

$$(11.3)$$

$$A = \begin{bmatrix} I_{qm} \\ -I_{qm} \end{bmatrix}$$

$$b = \begin{bmatrix} U^H \\ -U_L \end{bmatrix}$$

Also, we have outputs and manipulated variables related to state variables by

$$x(k + 1) = \Phi x(k) + \Gamma(k) + w(k) \quad (11.4)$$

$$y(k) = Cx(k) + v(k) \quad (11.5)$$

$$(11.6)$$

ϕ is represented by matrix A in the code, Γ is represented as matrix B and C is represented as C matrix in the code.

11.4 Implementing MPC

As mentioned earlier, MPC experiments were performed on SBHS 12 remotely. For this, the scilab codes uploaded on Moodle for Process Control SBHS assignments were used. The folder containing the codes, which was used to perform MPC experiments, have been included in the attached zip file in a folder named *codes*.

11.5 Working of codes

There are three main codes, which are being used for this experiment. *mpc_init.sce* is the code which opens the xcos window, wherein, we have step block for the set-point for temperature and the fan speed. Once the values have been entered into the xcos window and the simulation is started, the *scifunc* block of xcos calls the function *mpc.sci* after every sampling time. The *mpc.sci* in turn calls *mpc_run.sci* every time it is called by *scifunc* block. The *mpc_run.sci* code optimizes manipulated variable (heater) over control horizon and returns only the first manipulated variable (heater) value. This new heater value is then sent to the heater of the SBHS to control the temperature at the set point.

11.6 Procedure to implement MPC on SBHS

1. Open the folder named *Client-Java-latest10032011* and open VirtualLab-Client.
2. After entering the details and connecting to the allotted SBHS, open Scilab 5.3.1

3. Change the current directory to this folder, where mpc codes are present.
4. Open *mpc_init.sce* and *load in scilab*
5. The xcos window opens. Description of xcos window is mentioned below in the next section.
6. Enter the required step change values, if any, in the Temperature set point and/or fan block.
7. Enter the sampling time in the clock block as 1 second
8. Start the simulation from the xcos
9. After the experiment is over, the data files can be downloaded from the Java client.

11.7 Description of xcos

When *mpc_init.sce* is executed in scilab, an xcos window opens up. The xcos window has two step input blocks. The first step input block on the left side, is for the Temperature set point and the second step input block is for the fan (disturbance variable). Also the sampling time can be entered via clock block present on the xcos.

For all the experiments done for this project, sampling time of 1 second was used (entered via clock block of xcos).

Refer to the figure below for a clear picture of the xcos.

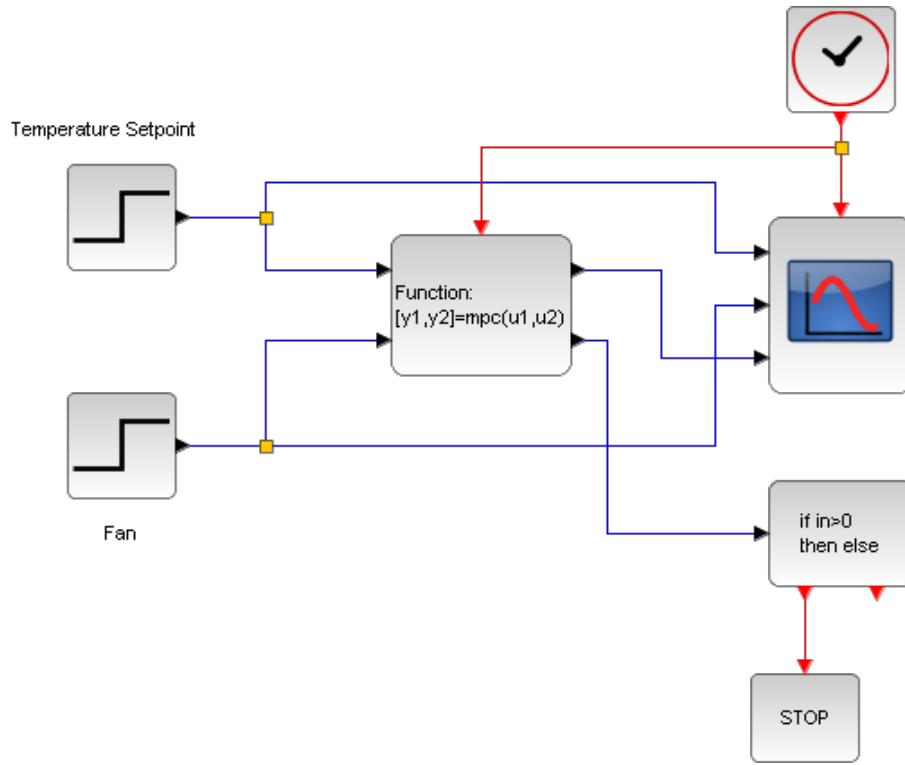


Figure 11.1: Screenshot of the xcos window with step input blocks labeled

After entering the values in the input step block, the simulation can be started. This opens up a graph, which shows the values of Temperature-set-point, fan and the actual temperature at each time instant during the simulation.

11.8 Code for MPC (*mpc_run.sci*)

The MPC code, implemented by me has been mentioned below.

```

1 function [ u_new ] = mpc_run(T, u_prev , Tset)
2 global p q xk_old
3
4 A = [0.9780 0 0 0 0 0 0 0;
5           1 0 0 0 0 0 0 0;
6           0 1 0 0 0 0 0 0;
```

```

7      0    0 1 0 0 0 0 0;
8      0    0 0 1 0 0 0 0;
9      0    0 0 0 1 0 0 0;
10     0    0 0 0 0 1 0 0;
11     0    0 0 0 0 0 1 0];
12 B = [1; 0; 0; 0; 0; 0; 0; 0];
13 C = [0 0 0 0 0 0 0.0079];
14
15 Tmax = 70;           // Maximum Temperature
16 We = 100*eye(p,p);   // Error Weighting Matrix , We
17 Wu = 10*eye(q,q);   // Control Weighting Matrix ,
Wu
18 xk=A*xk_old+B*u_prev;
19
20 // Formation of Su Matrix for Quadratic term of
optimization
21 for i = 1:1:p
22     for j = 1:1:q
23         if i <= q
24             if (i-j) >= 0
25                 Su(i,j)= C*A^(i-j)*B;
26             else
27                 Su(i,j)= 0;
28             end
29         else
30             if j < q
31                 Su(i,j)= C*A^(i-j)*B;
32             else
33                 Su(i,j) = Su(i-1,j) + C*A^(i-j)*B;
34             end
35         end
36     end
37 end
38
39 du_matrix=ones(q,q);
40
41 // Lambda Matrix for Quadratic term of optimization
42 for i = 1:1:q

```

```

43   for j = 1:1:q
44     if i == j
45       du_matrix(i,j) = 1;
46     if i >1
47       du_matrix(i,j-1) = -1;
48     end
49   else
50     du_matrix(i,j) = 0;
51   end
52 end
53
54
55 du_matrix_0 = eye(1,q); // Declaration of Lambda_0
56   vector
57
58 // Formation of Sx Matrix for Linear term of
59 // optimization
60 for i = 1:1:p
61   Sx(i,:) = C * A^i;
62 end
63
64 // Declaration of S_eta Matrix
65 S_eta = ones(1,p);
66
67 // Declaration of Set Point Vector
68 R = ones(1,p)*Tset;
69
70 // Temperature Prediction using information till
71 // previous instant
72 T_pred = C*xk;
73
74 // Measurement Error
75 eta = T - T_pred;
76
77 // Quadratic Term for Optimization
78 Su_t=Su';
79 du_matrix_t=du_matrix';
80 Q=2*((Su_t*We*Su)+(du_matrix_t*Wu*du_matrix));

```

```

78
79 // Linear Term in Optimization
80 R_t=R';
81 S_eta_t=S_eta';
82 du_matrix_0_t=du_matrix_0';
83 F_term1_t=(R_t-(Sx*xk)-(S_eta_t*eta))';
84 F=-2*((F_term1_t*We*Su)+((du_matrix_0_t*u_prev)'*Wu*
     du_matrix));
85
86 // Inequality Matrices and Vectors
87 A_ineq = [eye(q,q); -1*eye(q,q)];
88
89 b_ineq_term1_1=-Sx*xk-S_eta';
90 b_ineq_term1_2=(Tmax*ones(1,p))';
91 b_ineq_term1=(b_ineq_term1_1*eta+b_ineq_term1_2)';
92 b_ineq_term2_1=-Sx*xk-S_eta'*eta;
93 b_ineq_term2_2=(Tmax*ones(1,p))';
94 b_ineq_term2=-1*(b_ineq_term2_1+b_ineq_term2_2)';
95 b_ineq = [40*ones(1,q) zeros(1,q)];
96
97 me=0;
98 ci=zeros(q,1);
99 cs=40*ones(q,1);
100
101 [x, iact, iter, f]=qpsolve(Q,F,A_ineq,b_ineq');
102
103 u_new=x(1);
104 xk_old=xk;
105 endfunction

```

11.9 Other codes used

Other codes that were used for conducting this experiment are *mpc_init.sce* and *mpc.sci*. Both these codes were originally taken from Moodle (Process controls course for SBHS assignment). Please note that, both these codes were slightly modified to work with our MPC.

The only changes done in the original codes are:

- addition of global variables p,q and xk_old (p is the prediction horizon, q is the control horizon, xk_old represents the last value of an internal state)
- initialization of p, q and xk_old
- removal of some unnecessary lines (ie, lines not relevant to MPC implementation)

11.10 Experiments conducted to implement MPC

Experiments were performed as shown in table above for implementation of MPC. We carried out experiments in which both positive and negative step changes were given to Set point and Fan (disturbance variable) and the output response was obtained by application of MPC. We also have performed several experiments to study the effect of change in the values of q (control horizon) and tuning parameters - error and manipulated variable weighting factors.

The details of the experiments mentioned in this report has been tabulated in the table given in the next page. The first column of the table represents the experiment version (or number). For all the outputs and their figures, we have mentioned only their experiment version (or number) to tag them. Also note that the data files for these experiments are also named as per their experiment version number.

p and q mentioned in the table represents the prediction and control horizon respectively.

Please note: For all the above experiments and graphs, we adhered to:

- Scilab Version: 5.2.2
- SBHS number: 12 (remotely accessed)
- Sampling time: 1 second

For graphs: Until and unless mentioned, Graphic 1 represents the Temperature set point, Graphic 2 represents the Fan and Graphic 3 represents the Temperature.

Also, please note that there are two types of graphs. The first graph, containing Graphic 1, Graphic 2 and Graphic 3 were directly obtained via mscope of xcos.

The graph following this in all the experiments is the temperature and heater value graphs, which were obtained from the data (from the text file downloaded from the server after each experiment).

Expt No	Temperature Set point			Fan			(p,q)	Weighing factor (We, Wu)
	T_initial (°C)	T_final (°C)	Time (s)	F_initial	F_final	Time (s)		
1.1	35	40	250	100	150	500	(40,4)	1,1
1.2	35	40	250	100	150	500	(40,4)	10,10
1.3	35	40	250	100	150	500	(40,4)	40,40
2.1	42	37	250	150	100	500	(40,4)	1,1
2.2	42	37	250	150	100	500	(40,4)	10,10
2.3	42	37	250	150	100	500	(40,4)	40,40
3.1	35	40	250	100	150	500	(40,2)	10,10
3.2	35	40	250	100	150	500	(40,3)	10,10
3.3	35	40	250	100	150	500	(40,4)	10,10
4.1	42	37	250	150	100	500	(40,2)	10,10
4.2	42	37	250	150	100	500	(40,3)	10,10
4.3	42	37	250	150	100	500	(40,4)	10,10
5.1	35	40	250	100	150	500	(40,4)	100,2
5.2	35	40	250	100	150	500	(40,4)	2,100
5.3	35	40	250	100	150	500	(40,4)	10,100
5.1	35	40	250	100	150	500	(40,4)	100,10

Figure 11.2: Experiments performed

All the experiments mentioned in this report has been labeled as shown in this table. This table is just a summary of all the parameters that was used for the corresponding experiment. Details on the inputs and a description of the output observed for each case has been mentioned in the corresponding section of each experiment.

11.11 Sample run to implement MPC

11.12 Positive Step Change to Set Point and Fan

Let us consider experiment 1.1, wherein, a positive step change of 5° C (from 35° C to 40° C) was provided to set point at time t=250 s and a step change to fan was provided at t = 500 s, from 100 to 150.

The graph obtained has been attached below:

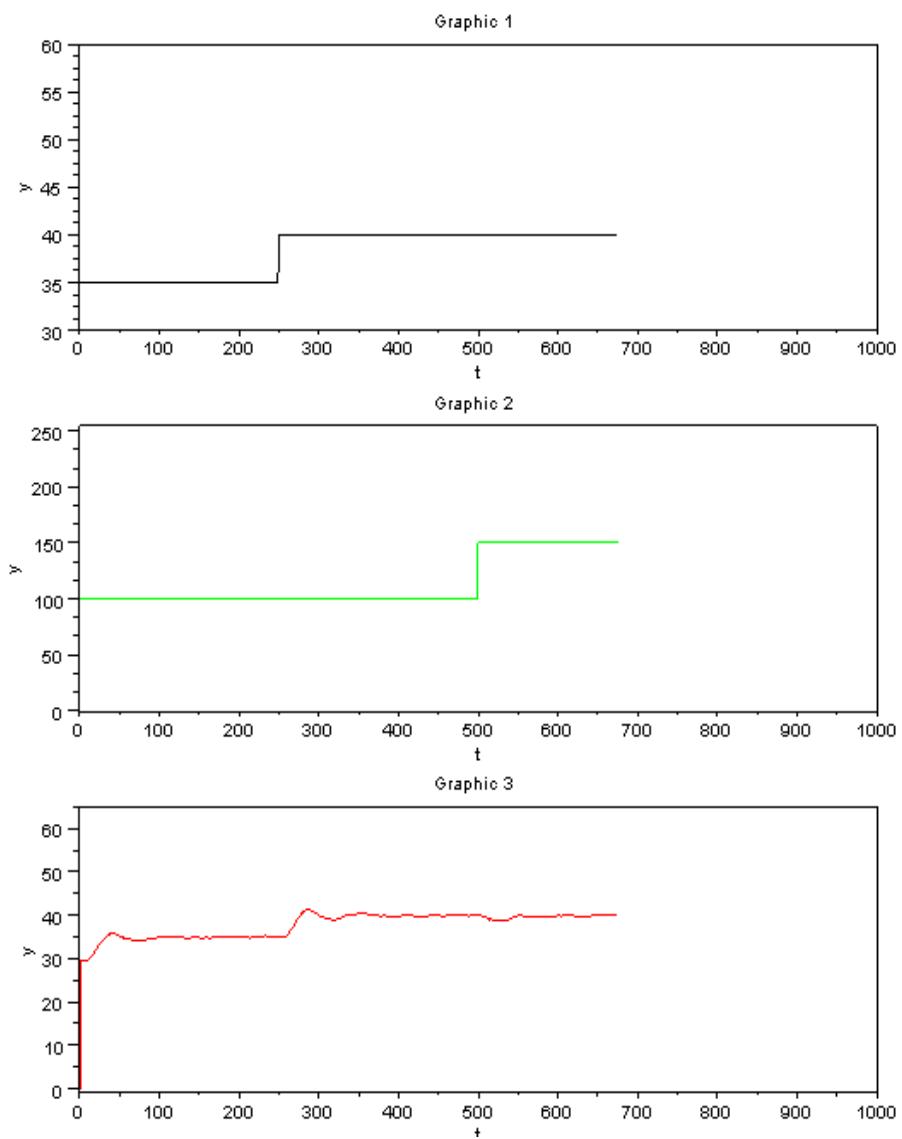


Figure 11.3: Expt 1.1

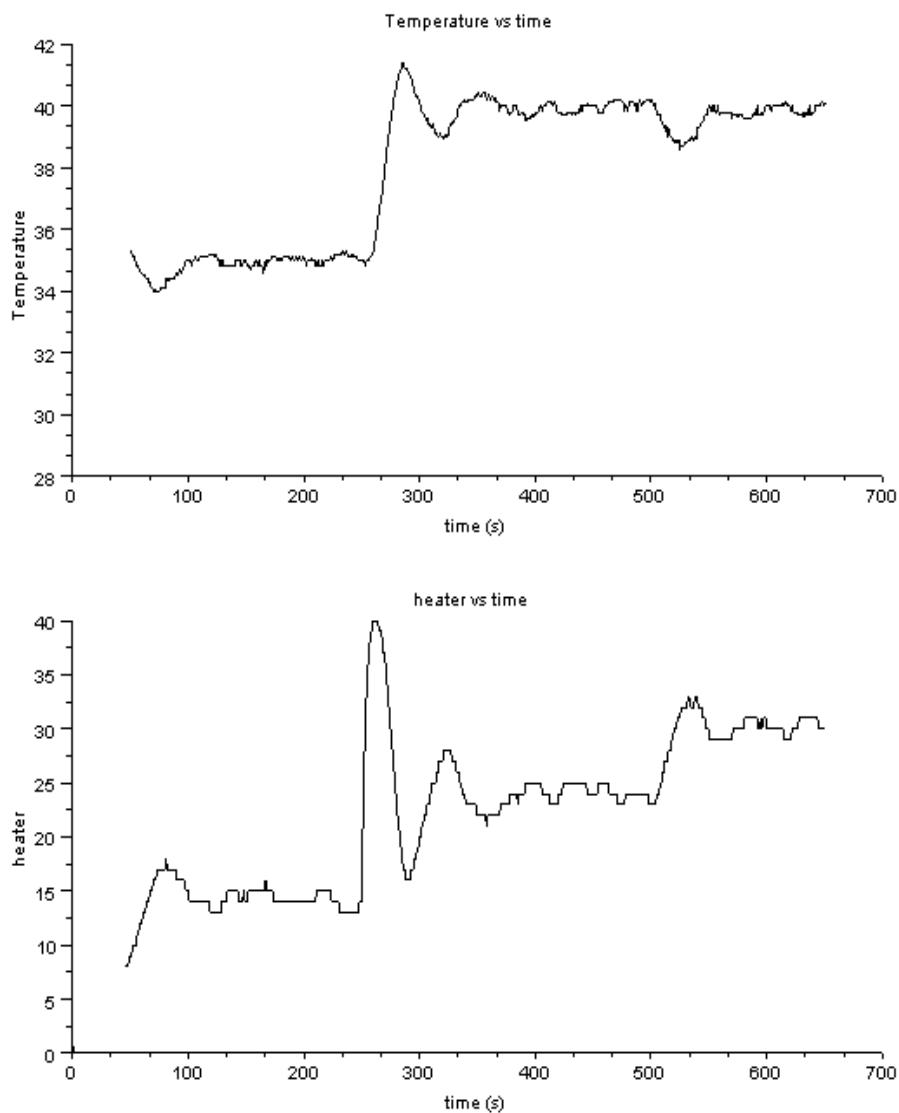


Figure 11.4: Expt 1.1

As can, be seen above, when, the temperature set point is raised to 40 from 30, at $t=250$ s, the value of the heater increases, so that it can heat up the plant upto the required set point. Similarly, when the fan speed is increased at $t=500$ s, the heater value increases yet again to maintain the same constant temperature of the SBHS blade.

11.13 Negative Step Change to Set Point and Fan

Let us consider experiment 2.1, wherein, a negative step change of 5°C (from 42°C to 37°C) was provided to set point at time t=250 s and a step change to fan was provided at t = 500 s, from 150 to 100.

The graph obtained has been attached below:

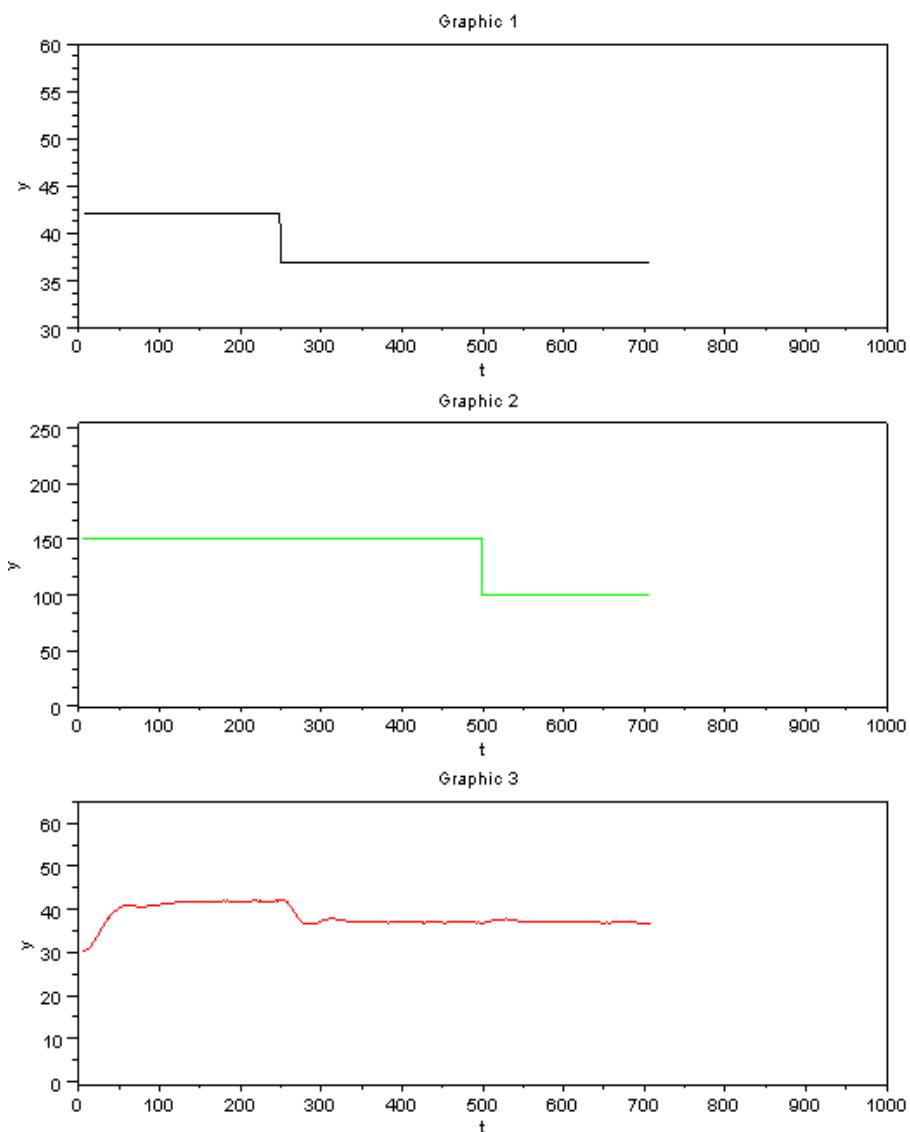


Figure 11.5: Expt 2.1

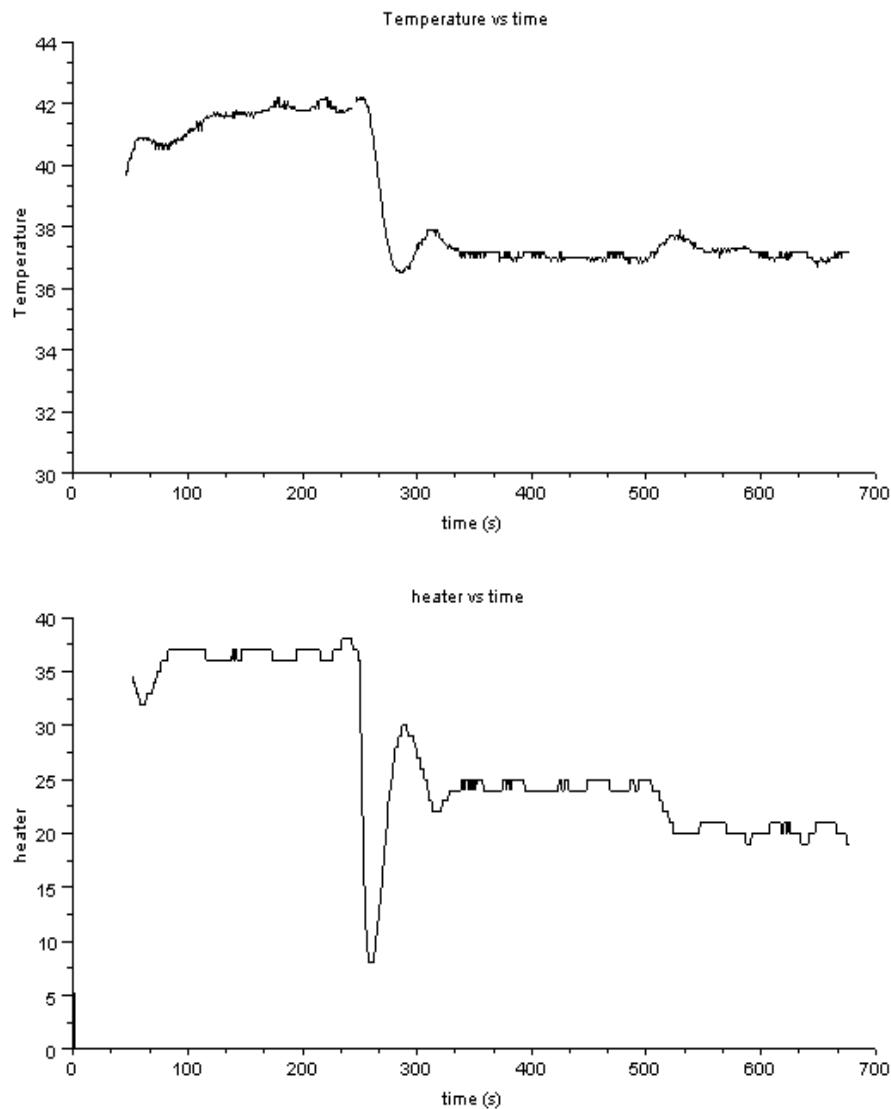


Figure 11.6: Expt 2.1

As can be seen from the graphs above, when the temperature set point drops at $t=250$ s, the value of the heater too falls, so that the plant (SBHS blade) can cool down to the required set point. Similarly, when the fan speed was decreased at $t=500$ s, the heater value decreased yet again to maintain the same constant temperature of the SBHS blade.

11.14 Effect of Tuning parameters: Weighting factors, We and Wu

We also, conducted several experiments in order the study the effect of the value of Weighting factors (both error,We and manipulated variable,Wu). We used weighting factors to be 1, 10 and 40 for both positive and negative step changes to both set point and fan (as has been summarized in Table 1). Also, experiments were done for different values of We and Wu. The results have been shown in the following graph.

11.15 For same factor of We and Wu

11.15.1 Positive Step Change and $(We, Wu) = (1,1)$ (Expt 1.1)

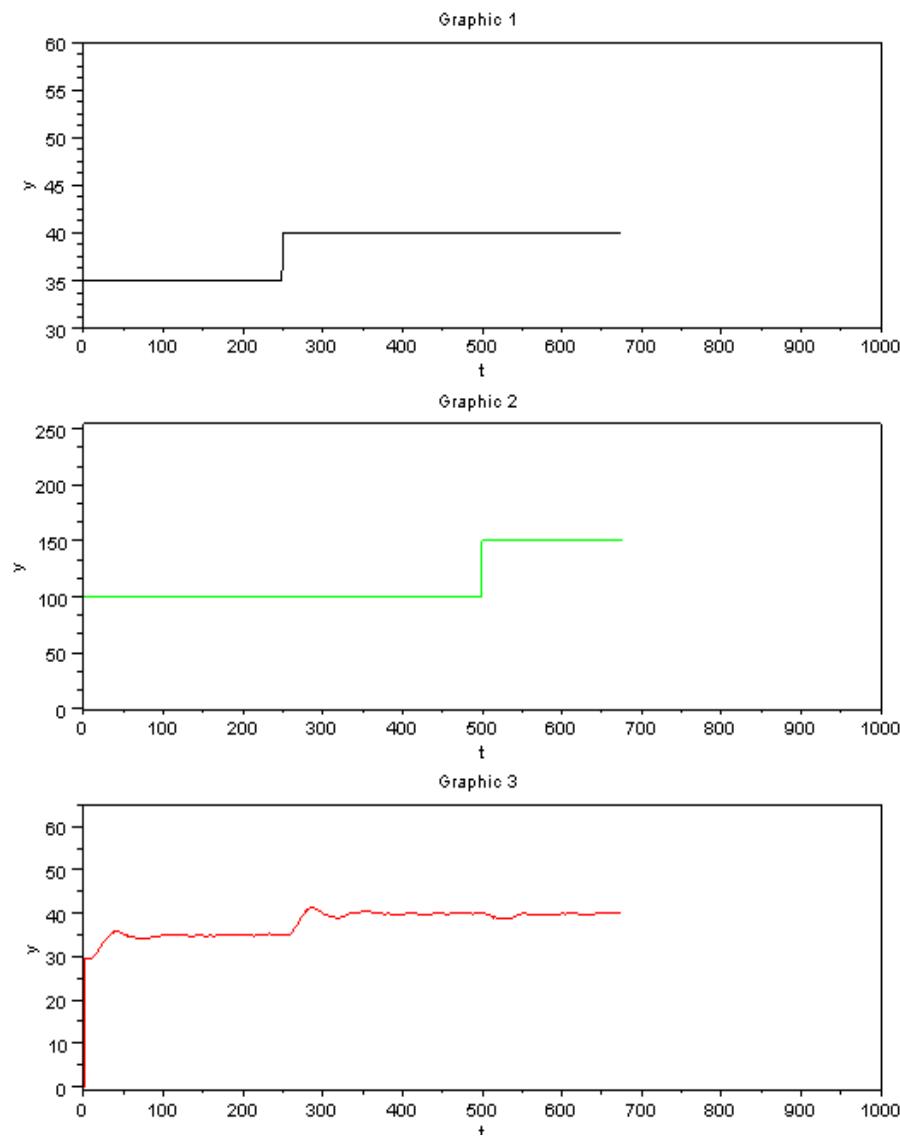


Figure 11.7: Expt 1.1

Here we can clearly see the expected output. Providing a positive step to temperature set point at 250 seconds, increased heater value as per the control effort put in by MPC. A positive step in fan at 500 seconds, decreased the temperature below its set point and hence heater value increased to take the temperature close to its setpoint.

This will be clear from the heater graph attached next.

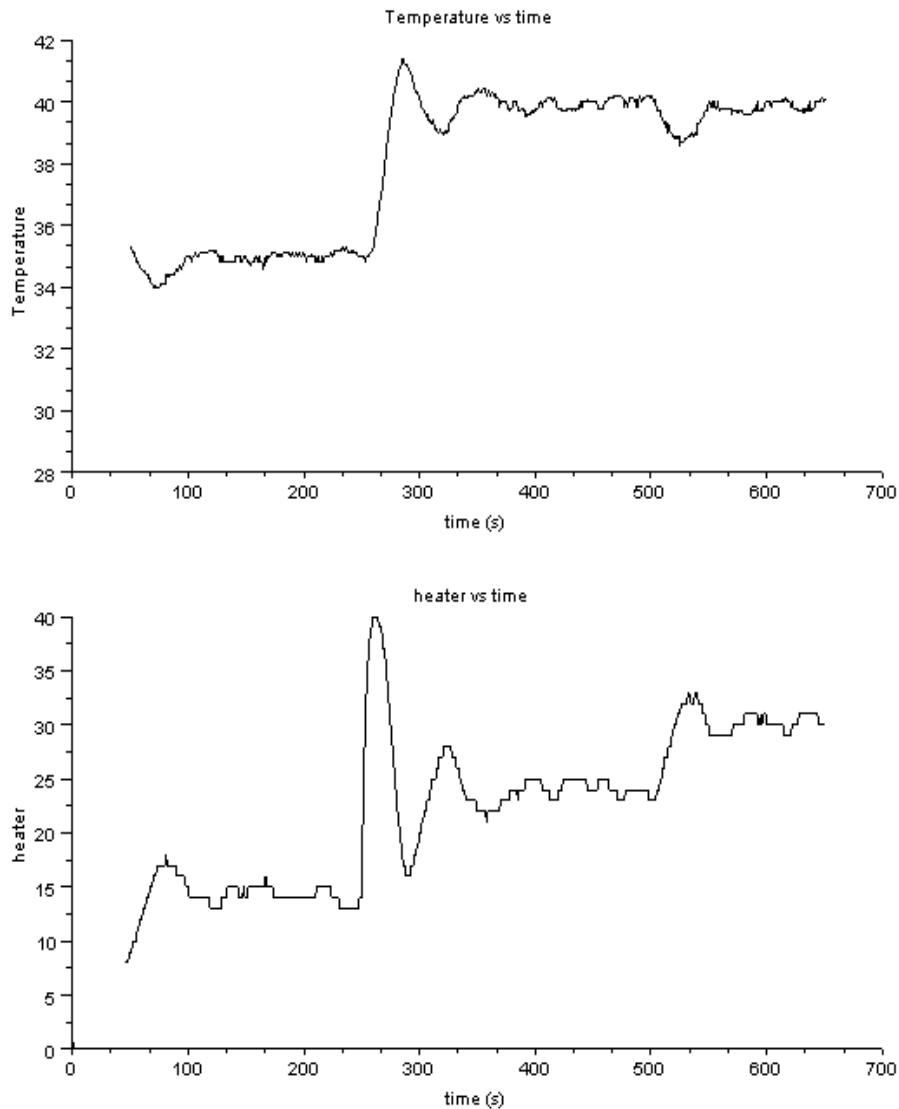


Figure 11.8: Expt 1.1

As can be clearly seen, the heater graph follows the expected trend that we talked of in the last page. Also, note that the temperature variation can be clearly seen from this graph.

This graph shows the result for the case, where we had same weighting factors for both error and manipulated variables (We and Wu). We will now see if changing

both of these is going to have any effect on the control behavior.
So, we now try an experiment with both We and Wu increased to 10.

11.15.2 Positive Step Change and $(We, Wu)=(10,10)$ (Expt 1.2)

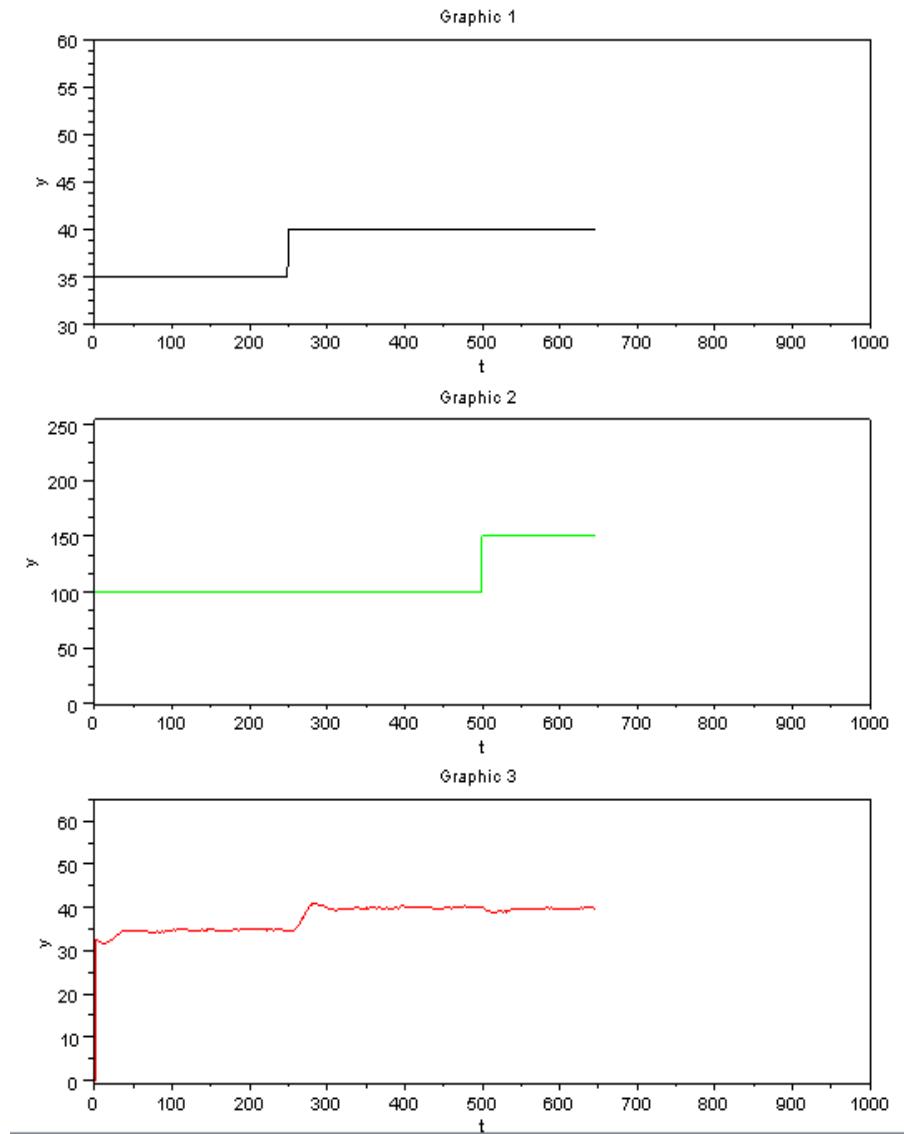


Figure 11.9: Expt 1.2

Using the same logic as has been explained in the last section, we expected to see similar temperature and heater value profiles for the positive step change in temperature set point and the fan. (Heater graph is shown in the next page along with the temperature on an expanded scale).

In this experiment, we increased We and Wu both to 10 from 1 and wish to observe if this changes the response of the plant.

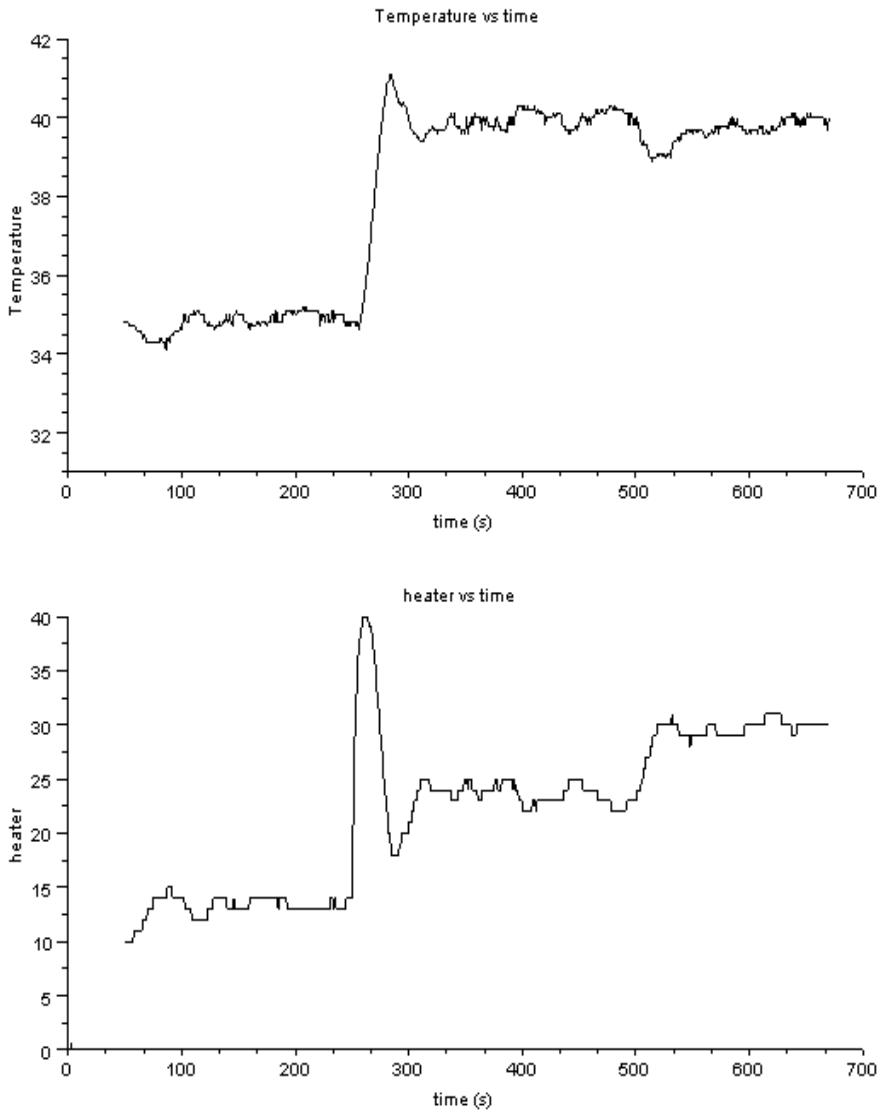


Figure 11.10: Expt 1.2

The results here are almost the same as that mentioned in the last section (where We and Wu both were 1). So, we can for the time being keep in mind that We and Wu isn't actually much affected the output. We now will carry out the experiment for even higher We and Wu (say 40) and see if it really does affect the output much.

11.15.3 Positive Step Change and $(W_e, W_u) = (40, 40)$ (Expt 1.3)

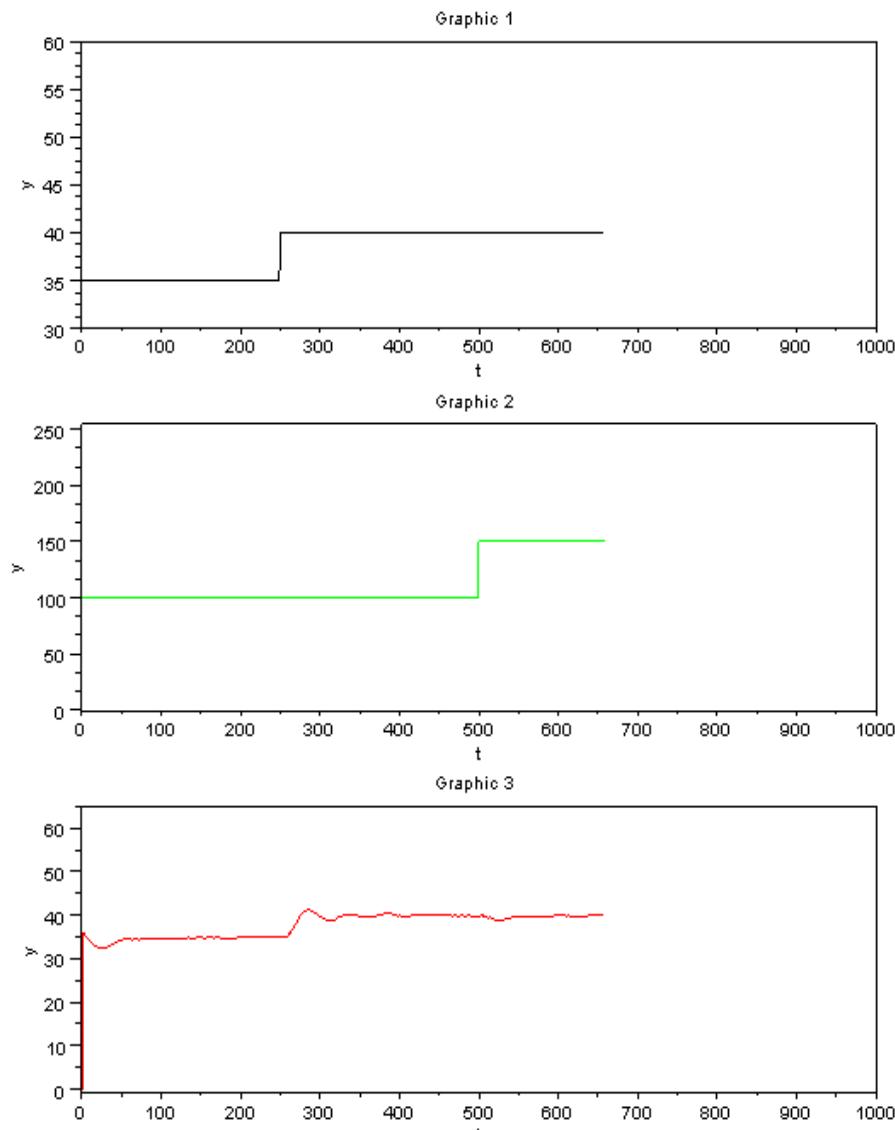


Figure 11.11: Expt 1.3

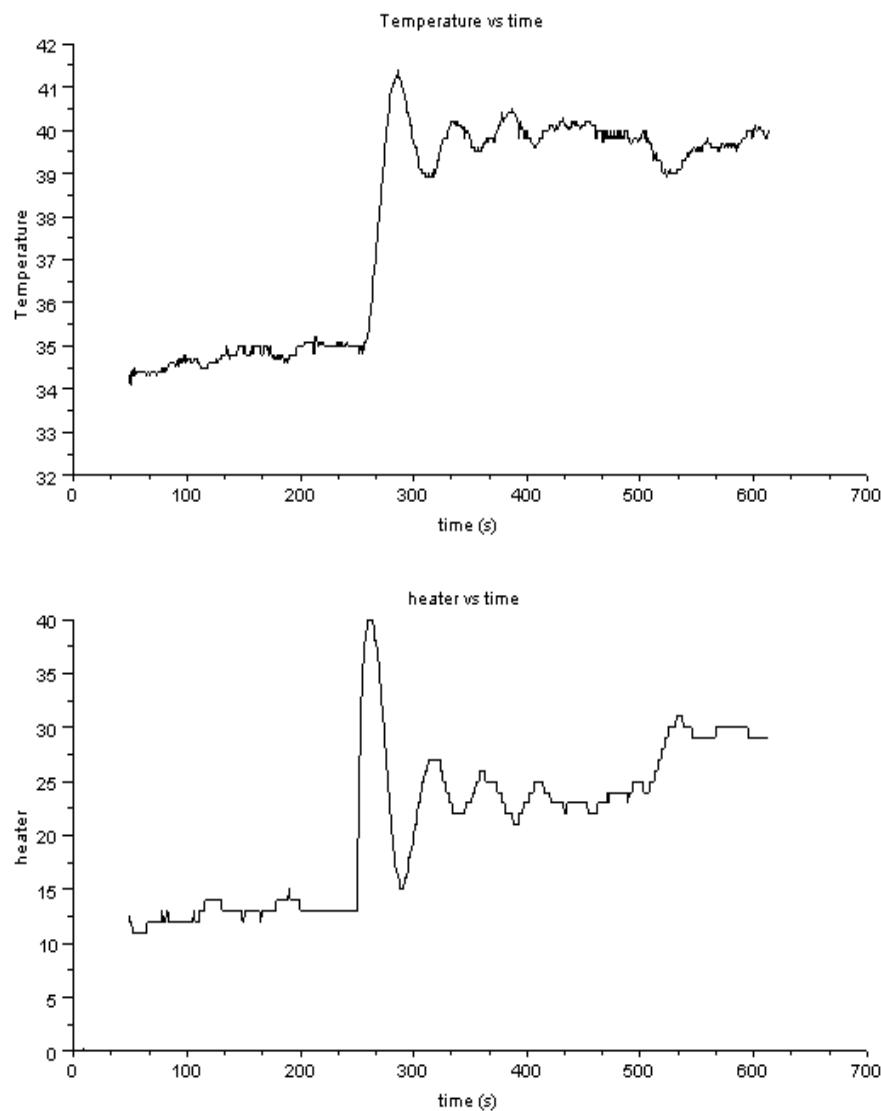


Figure 11.12: Expt 1.3

Even the results with We and Wu as 40 doesn't show much difference. They are more or less similar looking as the last two experiment's results.

11.15.4 Negative Step Change and $(W_e, W_u) = (1,1)$ (Expt 2.1)

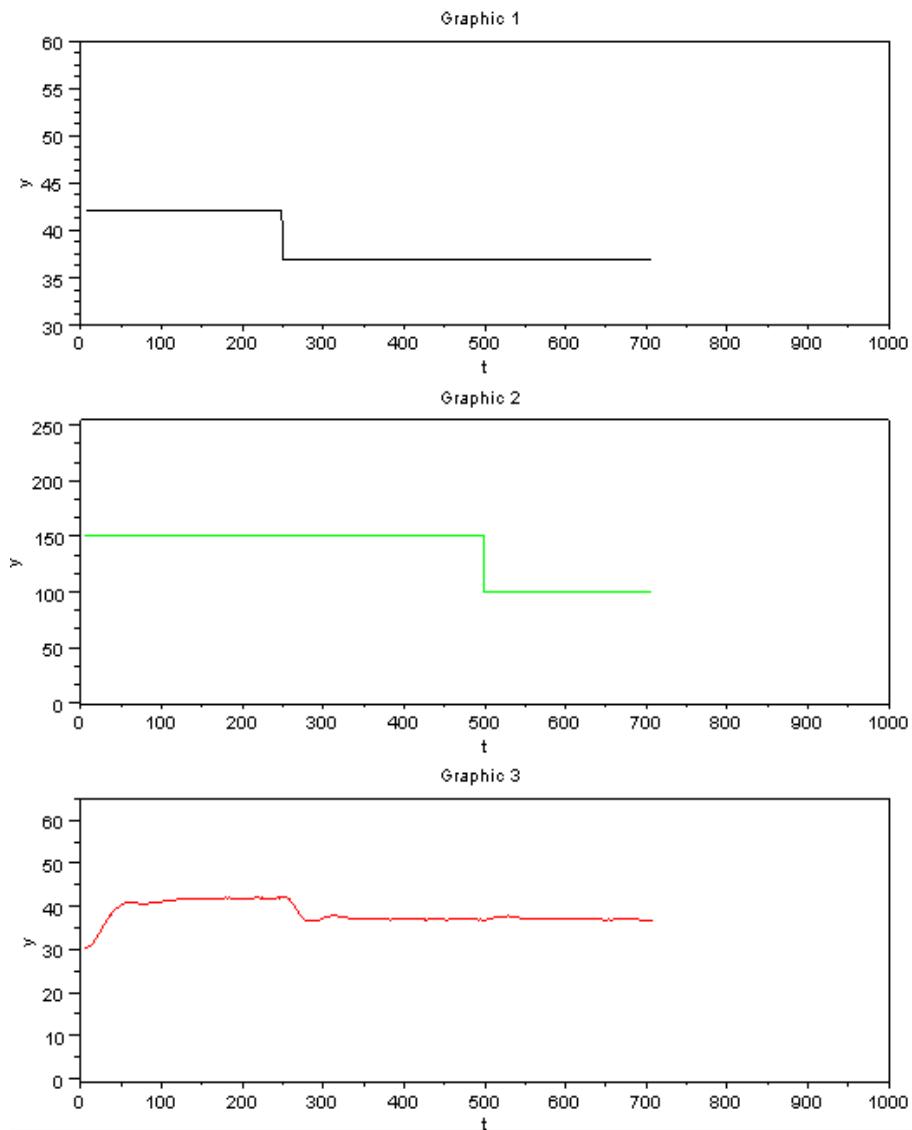


Figure 11.13: Expt 2.1

Here we expect somewhat similar results as was the case with positive step in temperature set point.

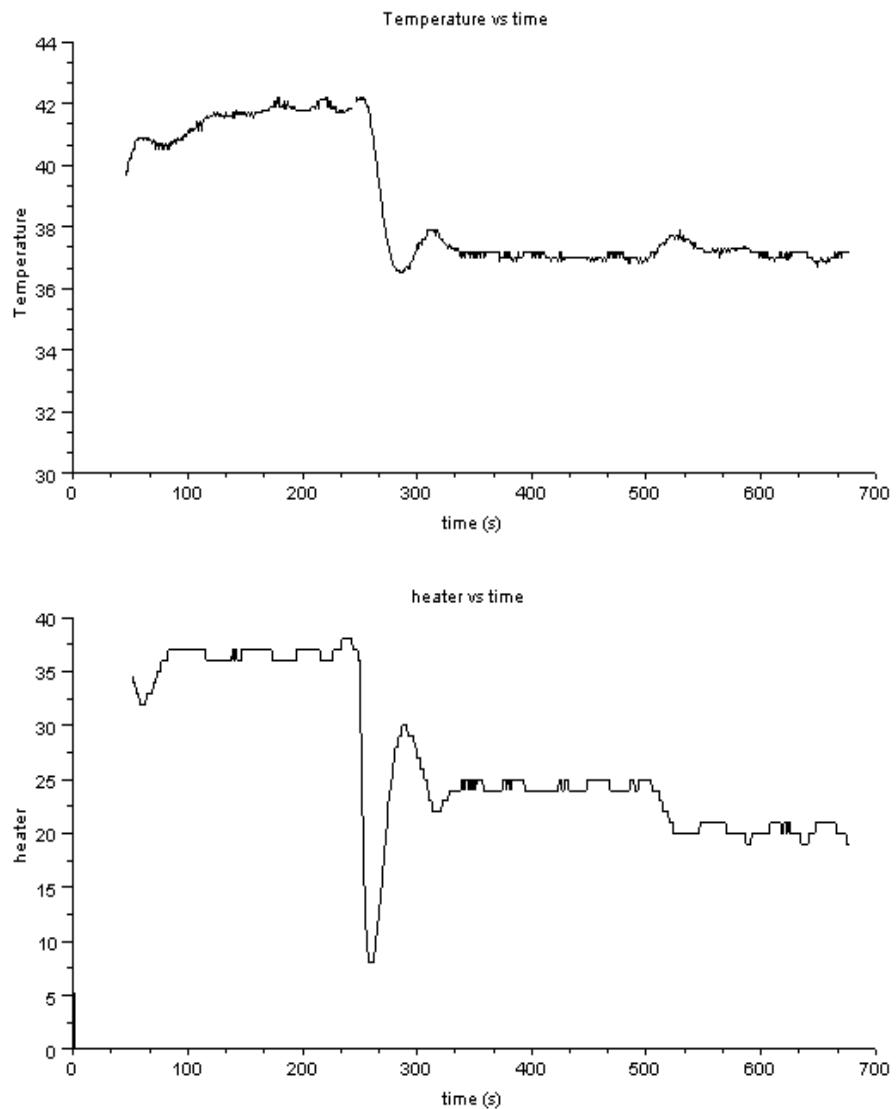


Figure 11.14: Expt 2.1

We can very clearly make out that the results follow the trends as was explained for the negative step input in the section 5.2

11.15.5 Negative Step Change and $(We, Wu)=(10,10)$ (Expt 2.2)

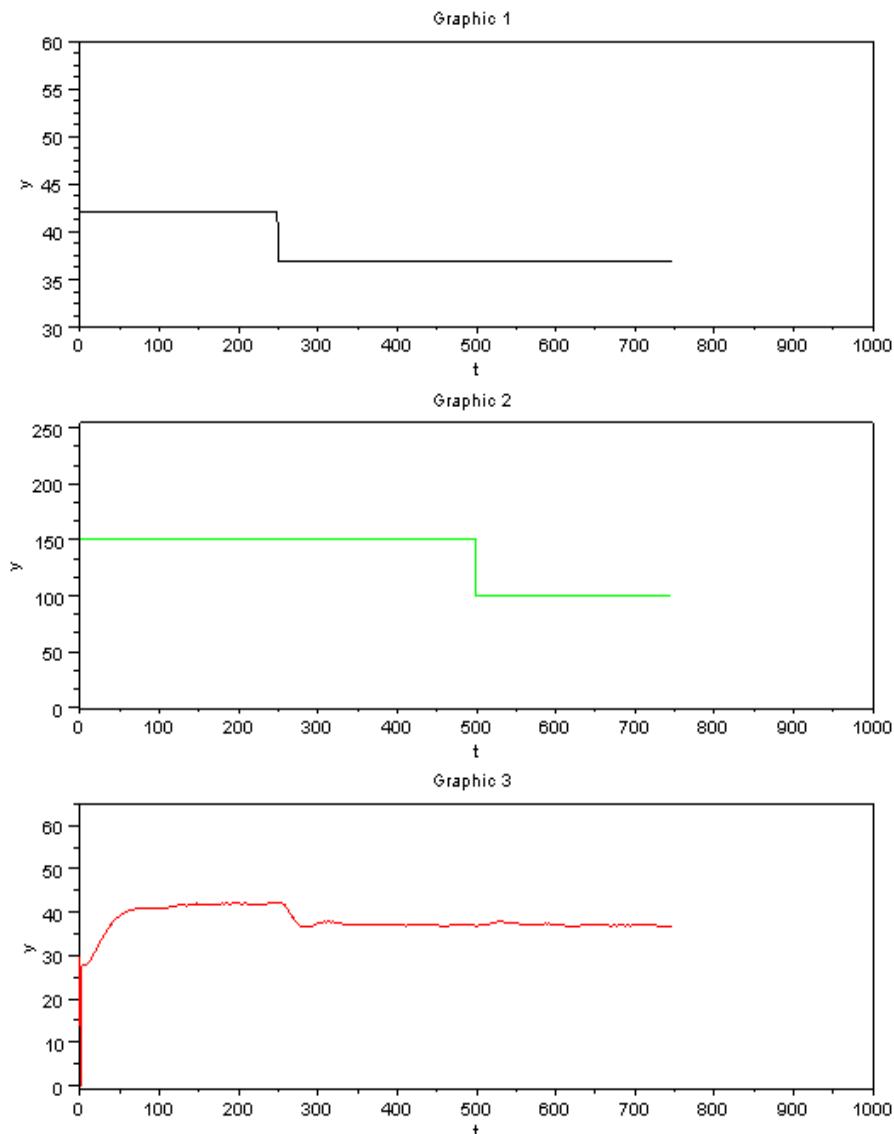


Figure 11.15: Expt 2.2

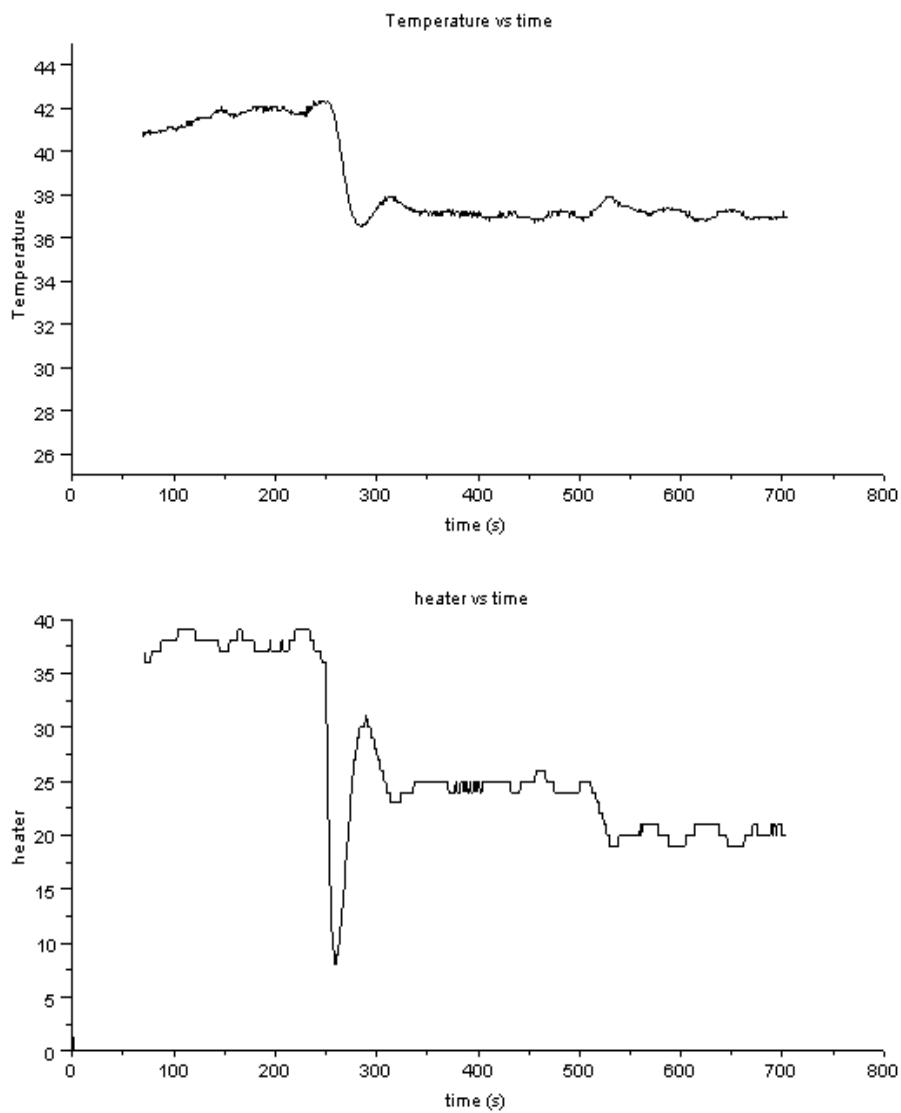


Figure 11.16: Expt 2.2

11.15.6 Negative Step Change and $(W_e, W_u) = (40, 40)$ (Expt 2.3)

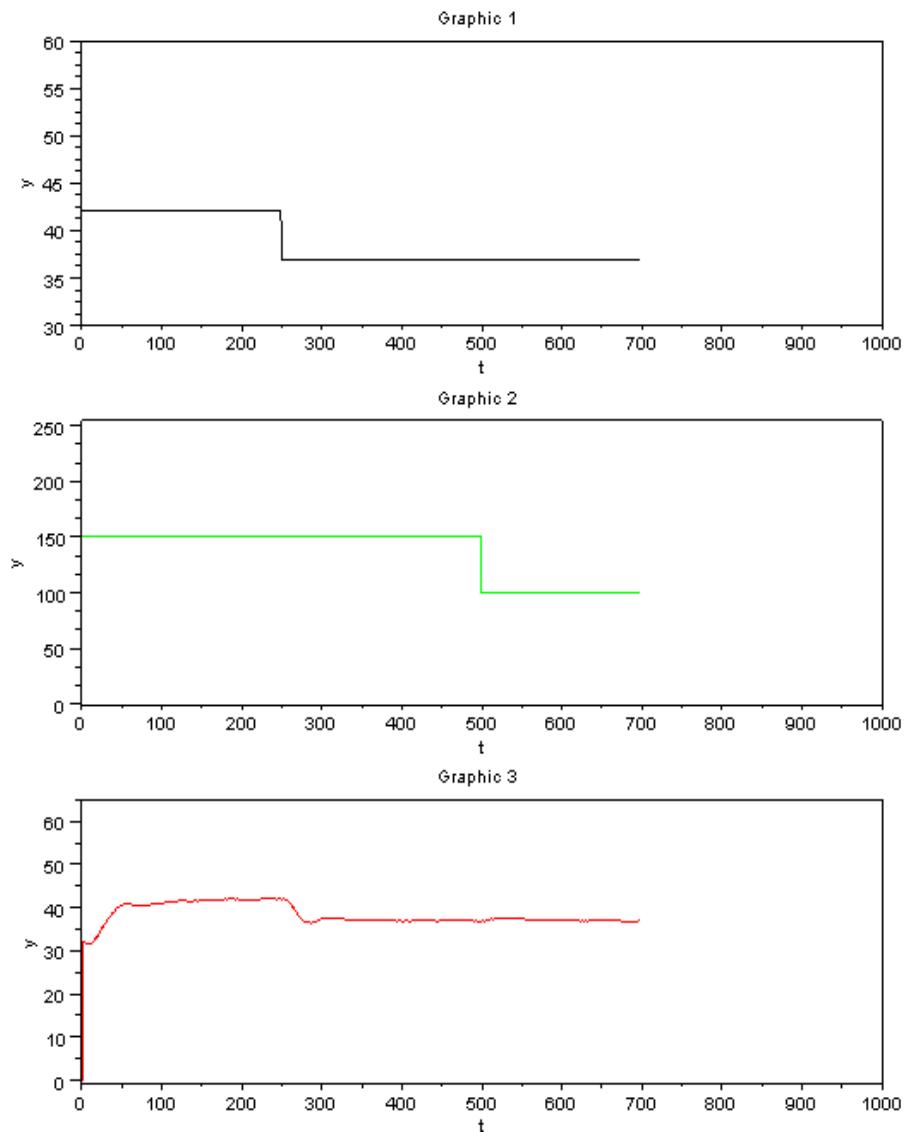


Figure 11.17: Expt 2.3

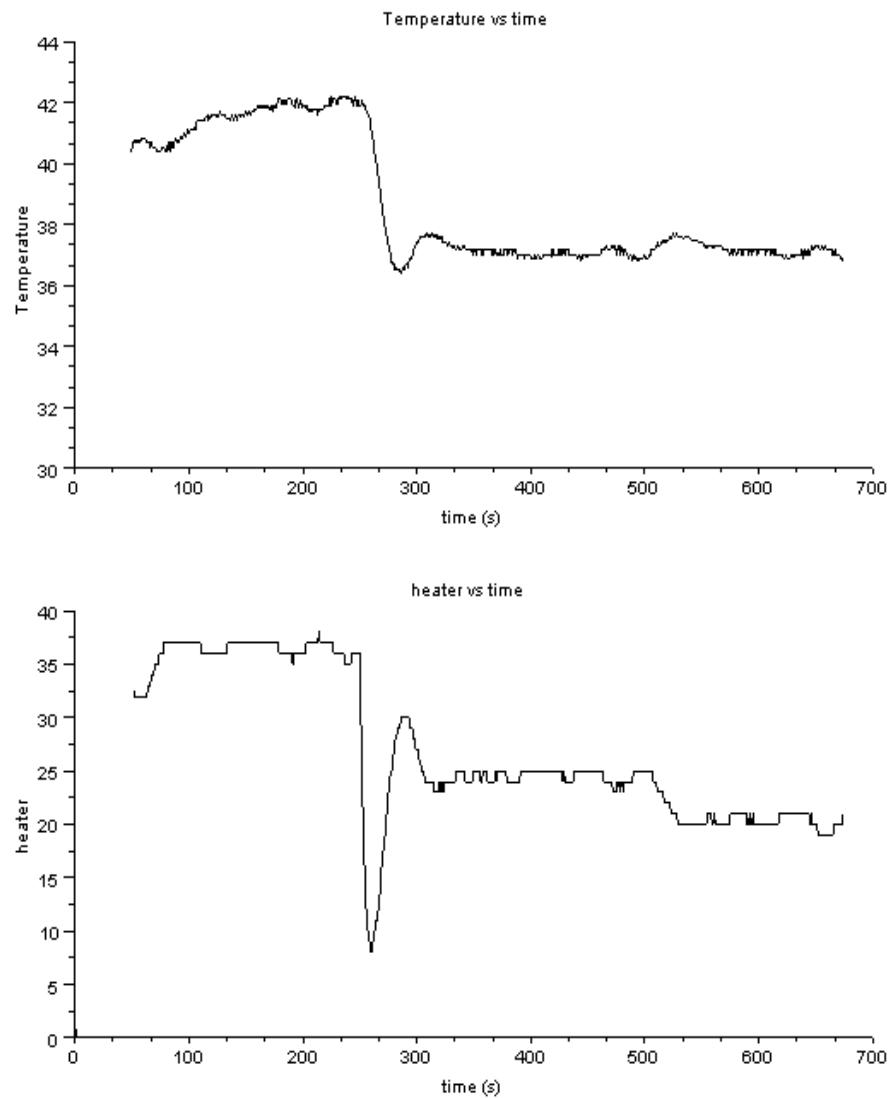


Figure 11.18: Expt 2.3

11.16 For different We and Wu factors

We very clearly see that using the same values of We and Wu is not making much difference in the control response. So, will now be trying different values for We and Wu.

11.16.1 We =100 and Wu = 2 (Expt 5.1)

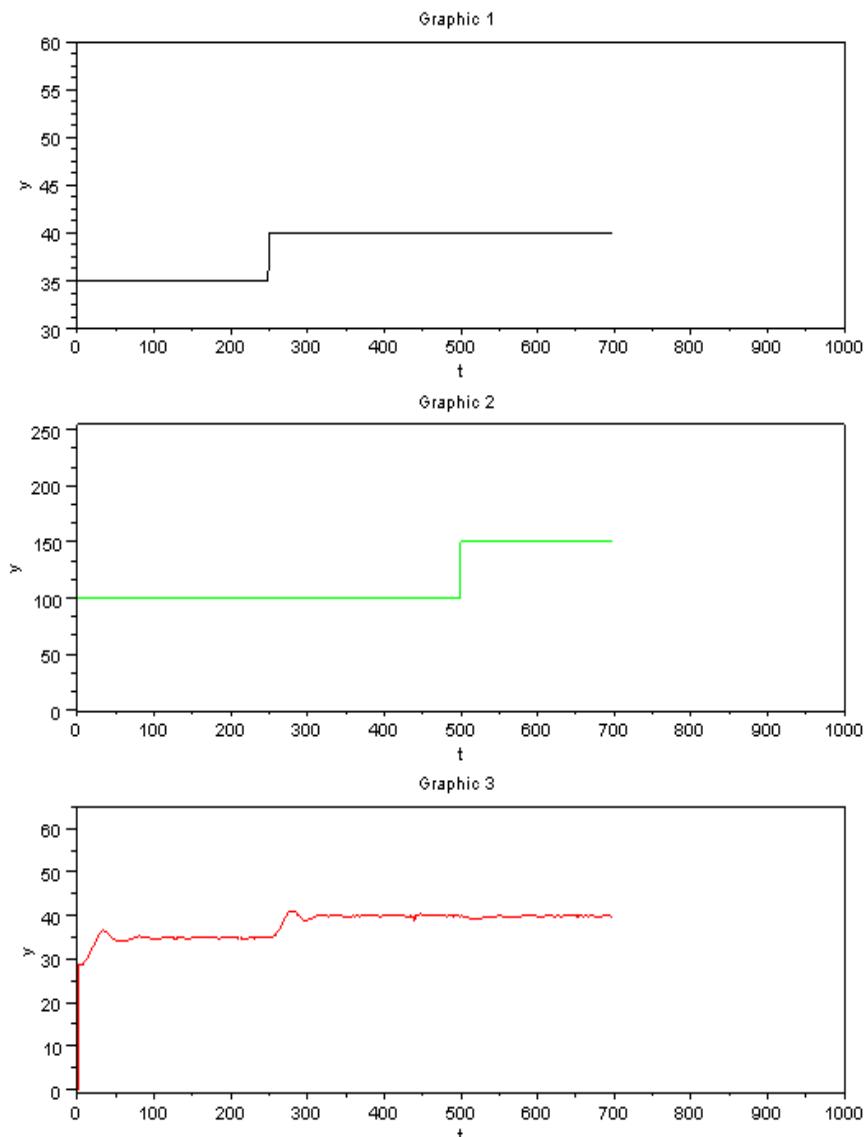


Figure 11.19: Expt 5.1

Here, we have used We as 100 and Wu as 2. The response after the positive step in temperature set point is slightly oscillatory. The temperature very well stabilizes at the required setpoint. The settling time observed is fairly low.

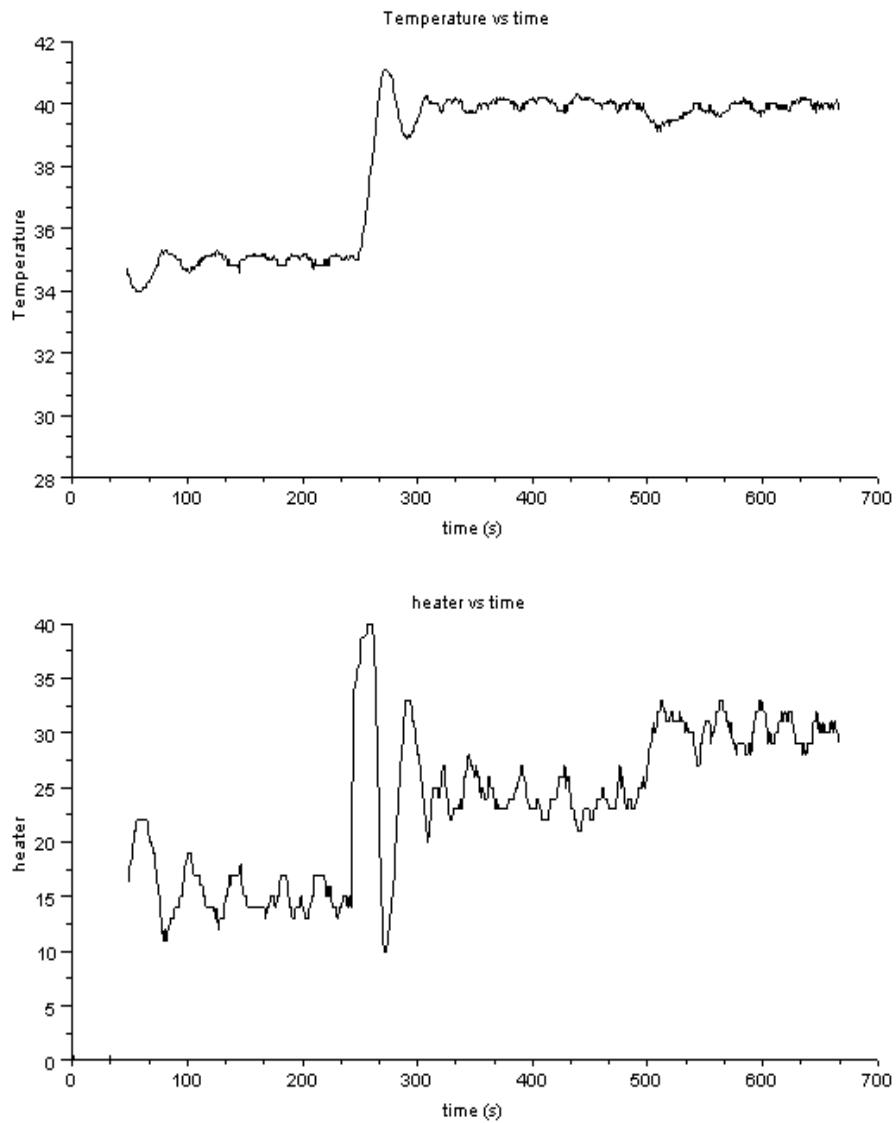


Figure 11.20: Expt 5.1

Now having seen the results of this experiment, we would like to check the possible effect of reversing the values of We and Wu . So, we conduct the next experiment, in which we have We as 2 and Wu as 100.

11.16.2 We =2 and Wu = 100 (Expt 5.2)

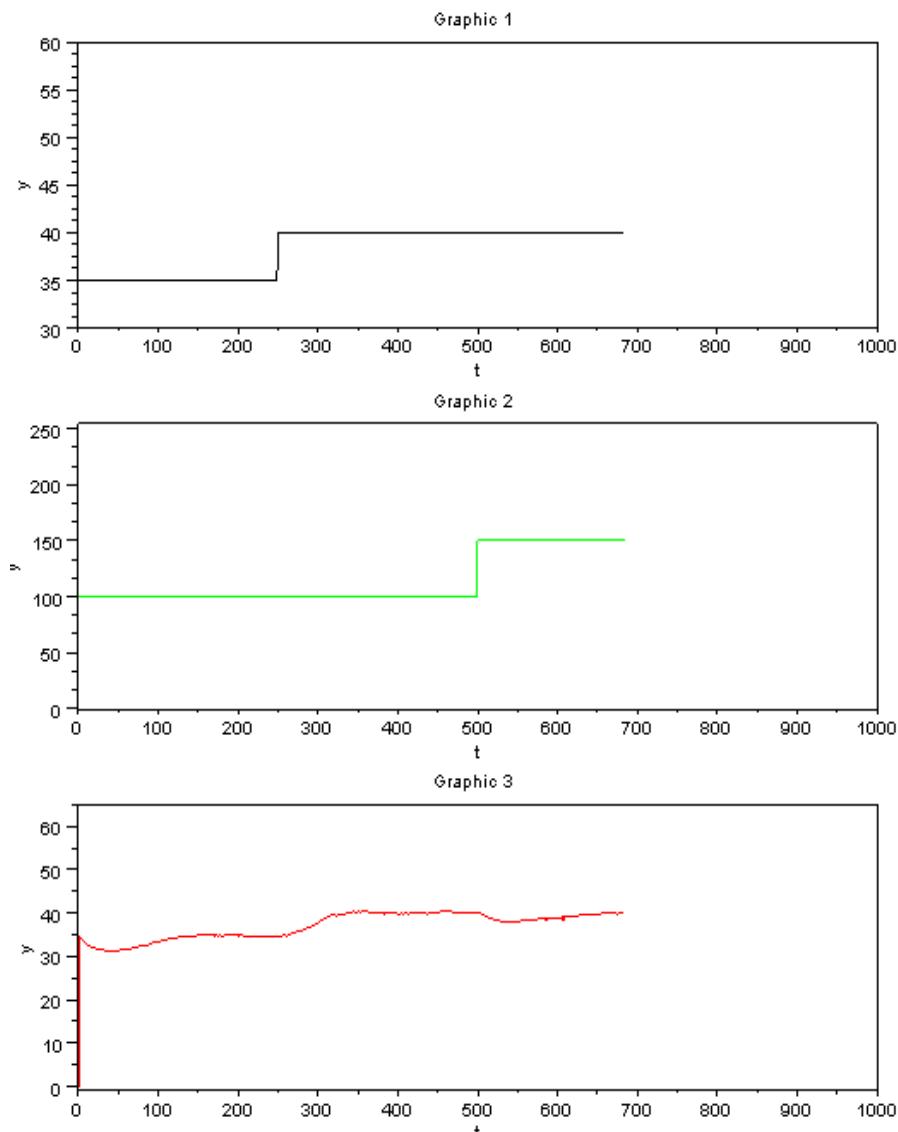


Figure 11.21: Expt 5.2

With increase in W_u , we observe that the temperature stabilizes at the required set-point, but the settling time for reaching that setpoint increases. Also, the response is not oscillatory. This result can be very clearly seen in the following temperature

and heater graph.

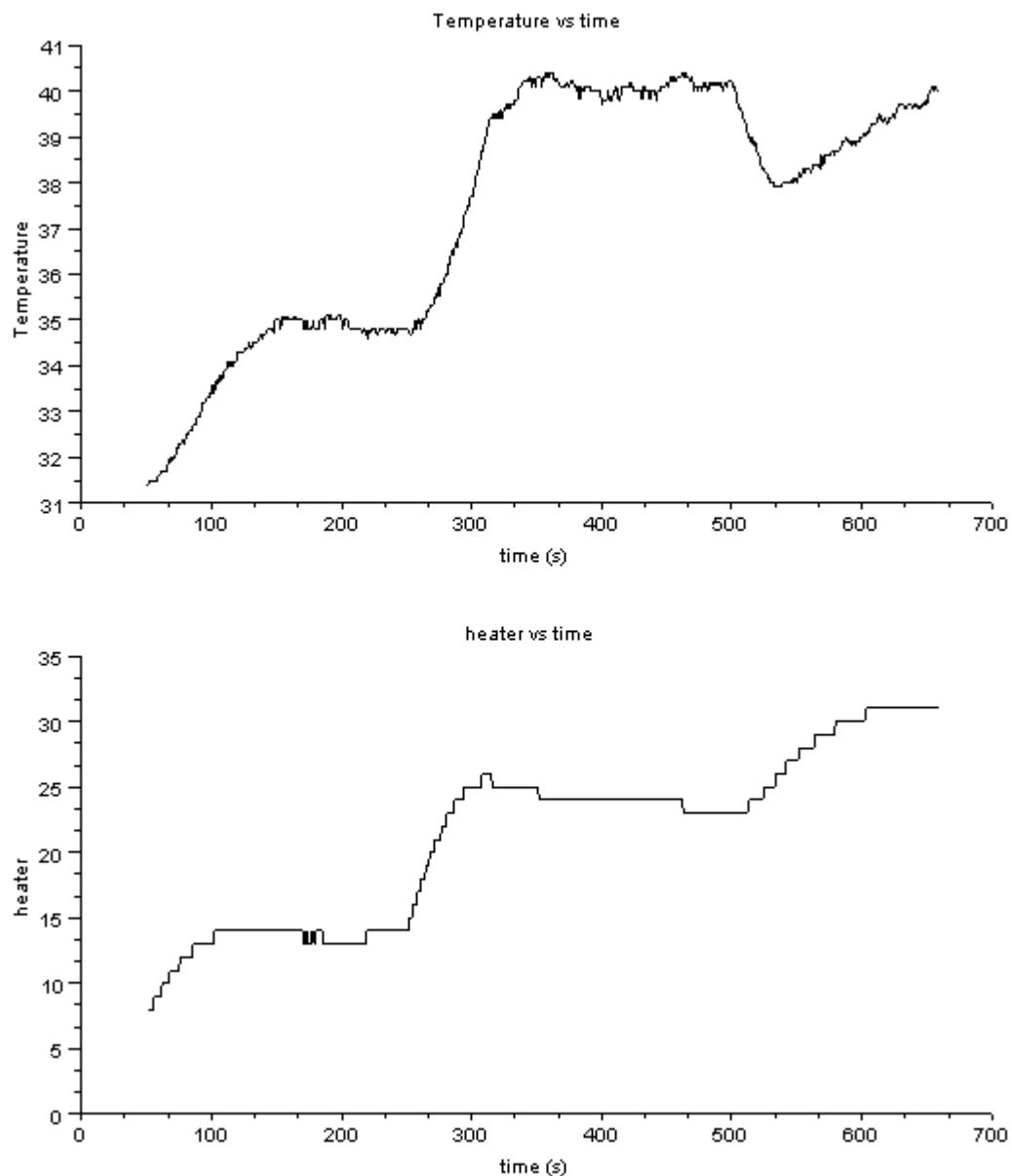


Figure 11.22: Expt 5.2

11.16.3 We =10 and Wu = 100 (Expt 5.3)

Having seen the effect of low We and high Wu (in the last section), we would like to see what happens if We is slightly increased keeping Wu the same. For this we increase the value of We to 10 and keep Wu at constant 100.

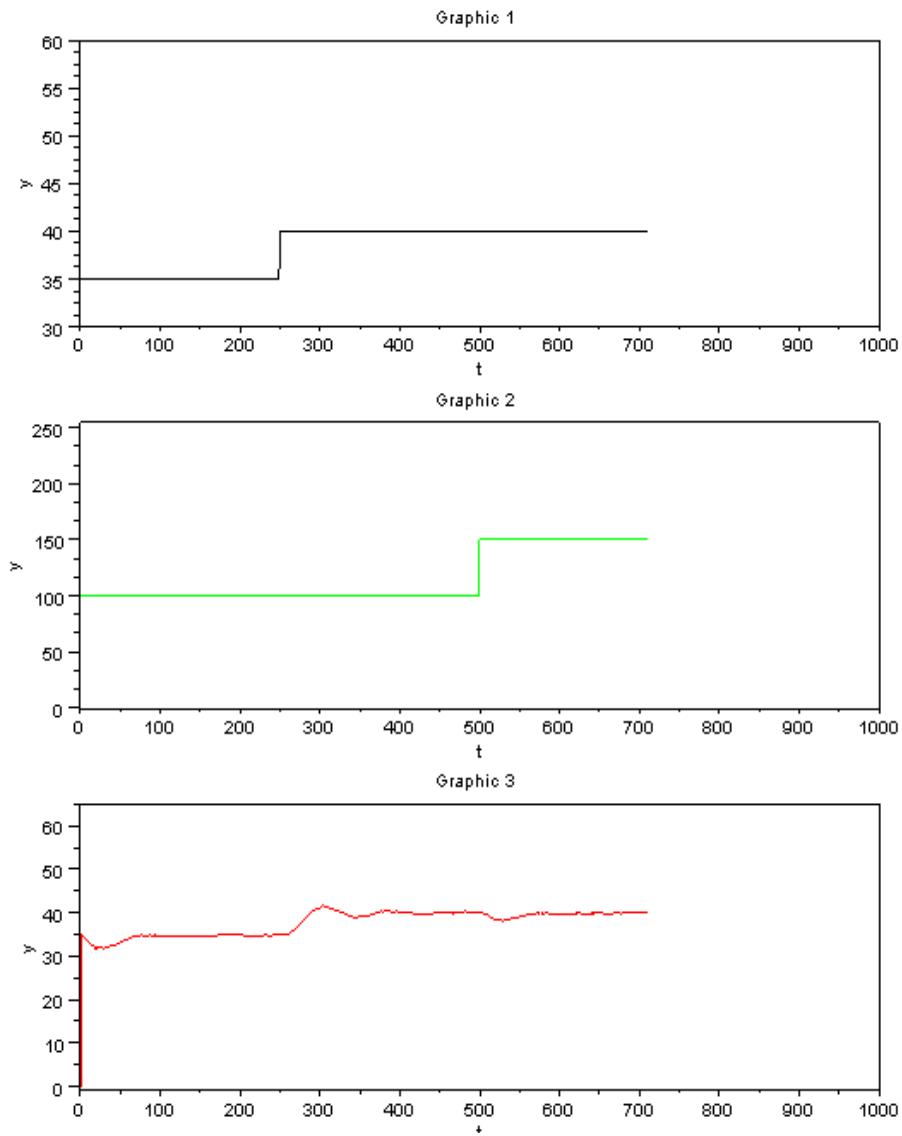


Figure 11.23: Expt 5.3

We observe that this experiments performs better than in the last section (where We was 2). It is slightly oscillatory and also, the settling time decreased much as compared to last experiment.

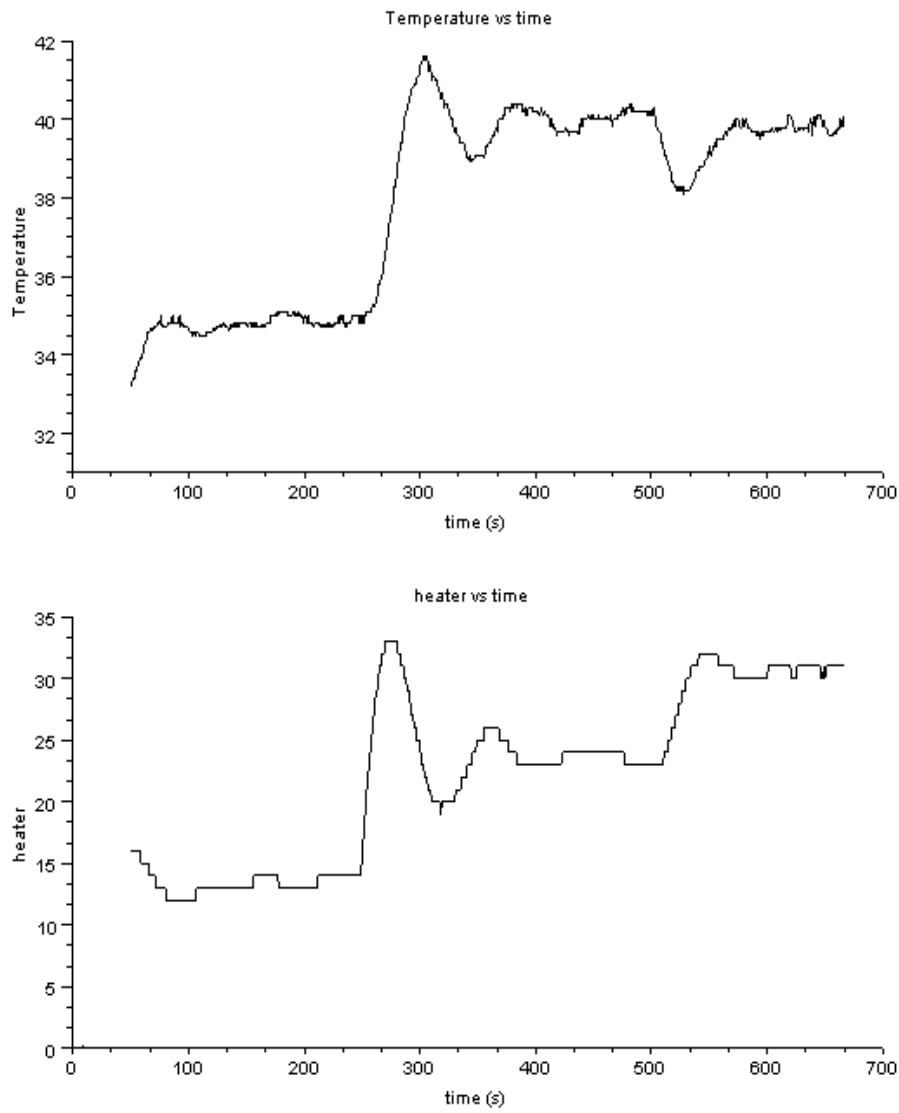


Figure 11.24: Expt 5.3

11.16.4 We =100 and Wu = 10 (Expt 5.4)

We now do a similar study for the case of Wu. We increase the value of Wu to 10, keeping We constant at 100.

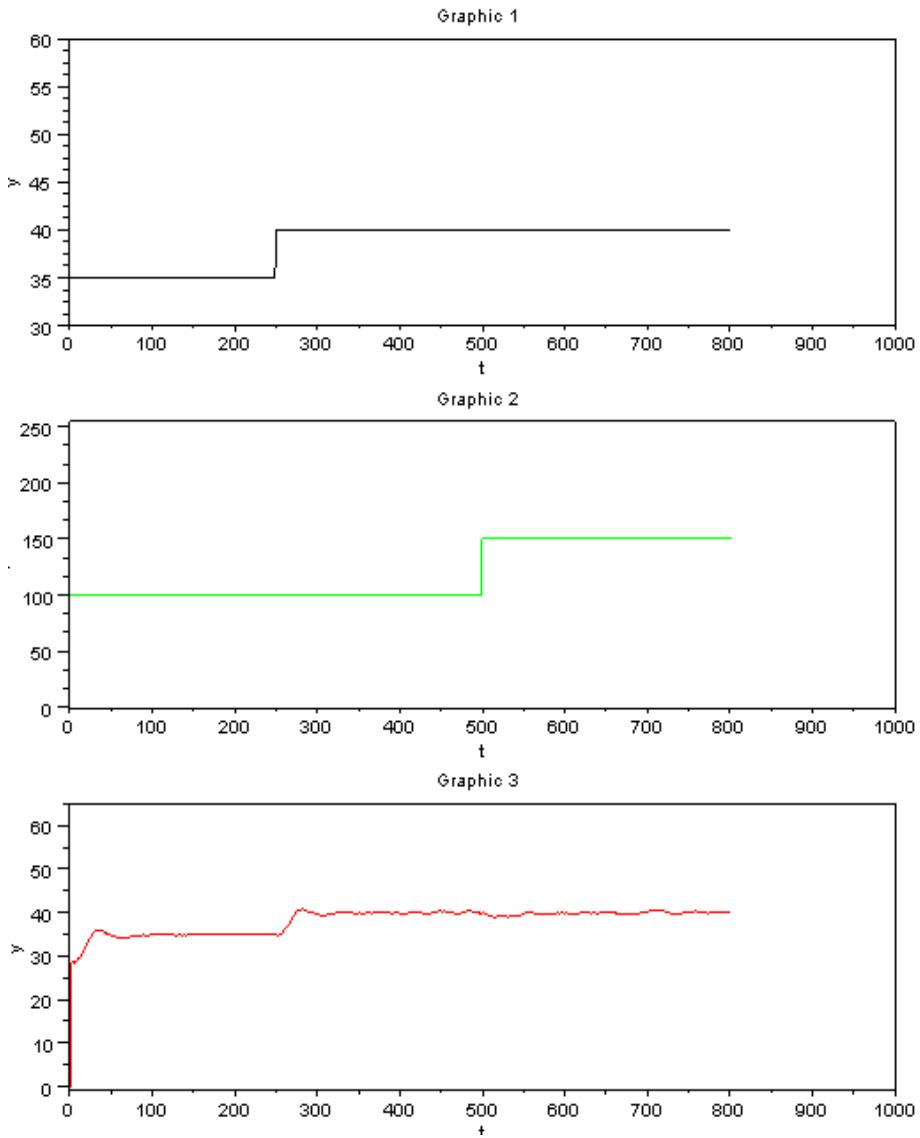


Figure 11.25: Expt 5.4

As is clear from the figure, we see slightly lesser oscillations compared to the

case when We was 100 and Wu was 2. Settling time more or less remained the same.

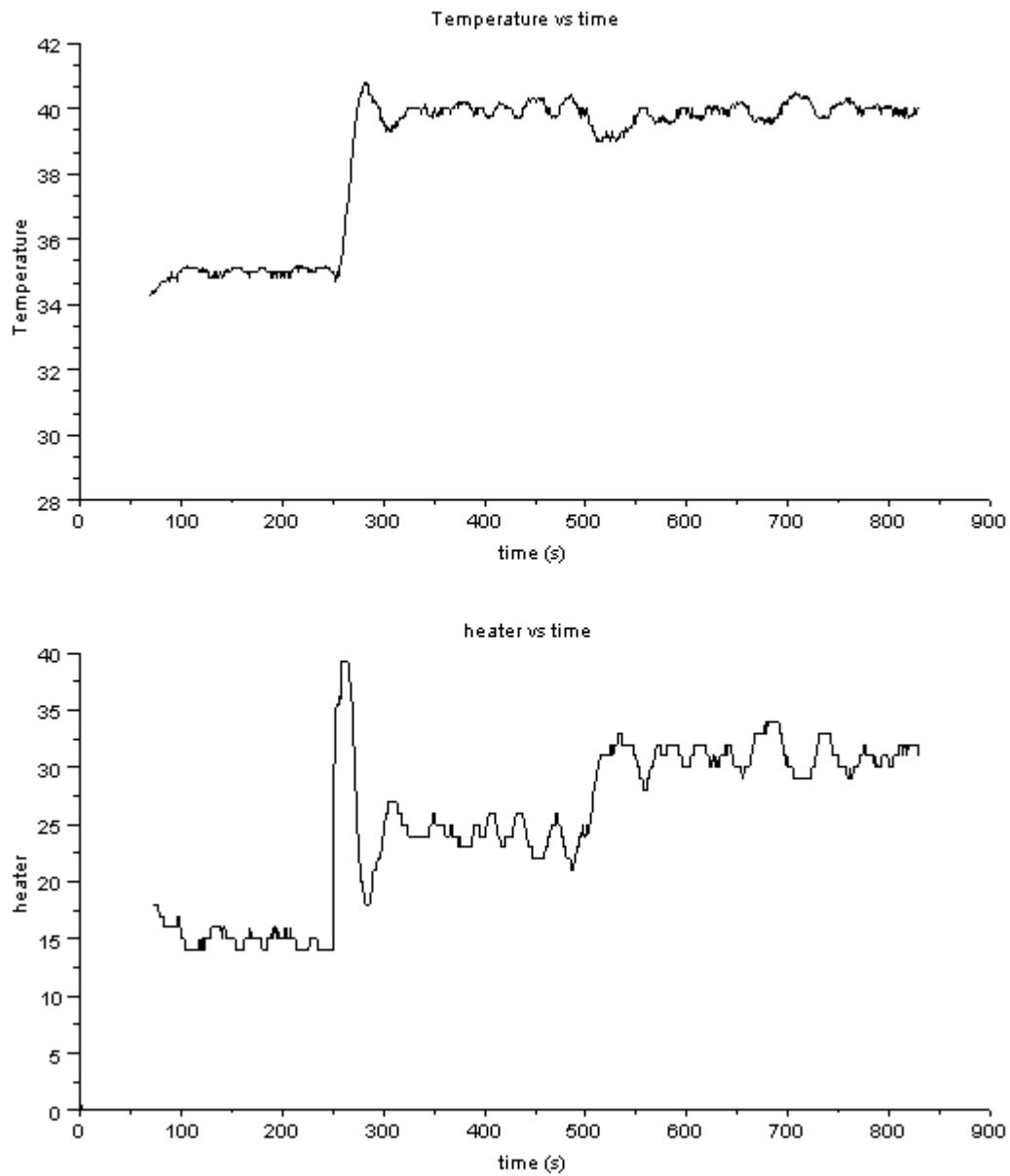


Figure 11.26: Expt 5.4

11.17 Conclusion on Weighting factor experiments

For experiments with same values of We and Wu:

- Not much difference was seen in heater value trends and temperature value trends for all the experiments performed above.
- Reason for this will be clear from the discussion on the trends mentioned below (for experiments with different values of We and Wu)

For experiments with different values of We and Wu:

- Keeping We as large (around 100) and Wu as small (2) shows better performance as compared to the case when the values are kept the other way around.
- With We very small (say around 1-2), oscillations are less, and the settling time observed was found to be more.
- With increase in We, the oscillations were observed to increase and the settling time was found to reduce and hence, better control was observed.
- So, with increase in We, any error is quickly dealt with, because with increase in We, we are actually increasing the significance of change of temperature in deciding the control action.
- With increase in Wu, oscillations reduced and the settling time was found to increase and hence, less preferred.

So, the best performance is obtained for the cases with high We and low Wu.

11.18 Effect of Control Horizon Paramter, q

We also tried to study the effect of change of control horizon (q) on the response of the SBHS to step change in Setpoint and disturbance variable. Generally the value of q (control parameter) is taken somewhere between 2 to 5. So, we performed our SBHS experiment for values of q as 2, 3 and 4 (as suggested by Mr Prashant Gupta).

Both positive and negative step change experiments for temperature set point and

disturbance variable (fan) was performed for the sake of completeness. The results obtained thereby has been mentioned in form of graphs in this section. The overall conclusion over these experiments has been mentioned in the conclusion of this part.

11.19 For positive step change in Set point and Fan speed

11.19.1 For q =2 (Expt 3.1)

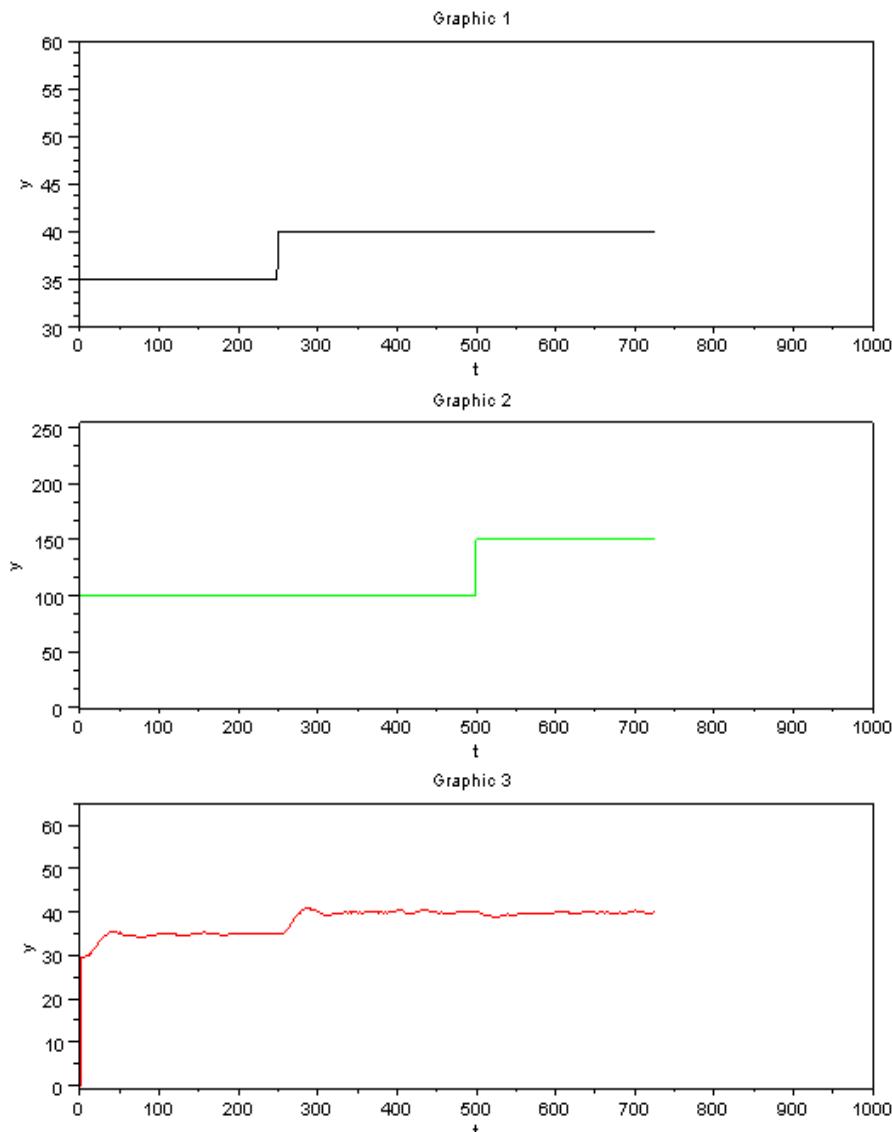


Figure 11.27: Expt 3.1

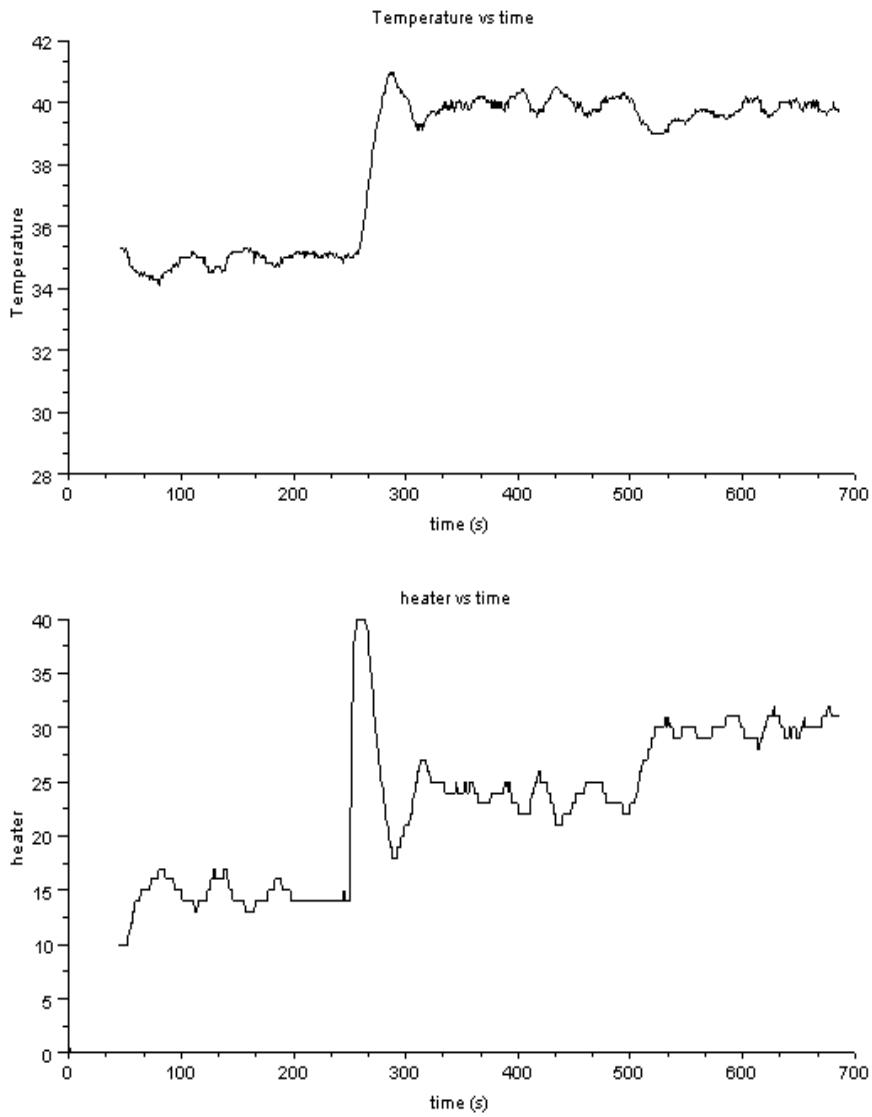


Figure 11.28: Expt 3.1

11.19.2 For $q = 3$ (Expt 3.2)

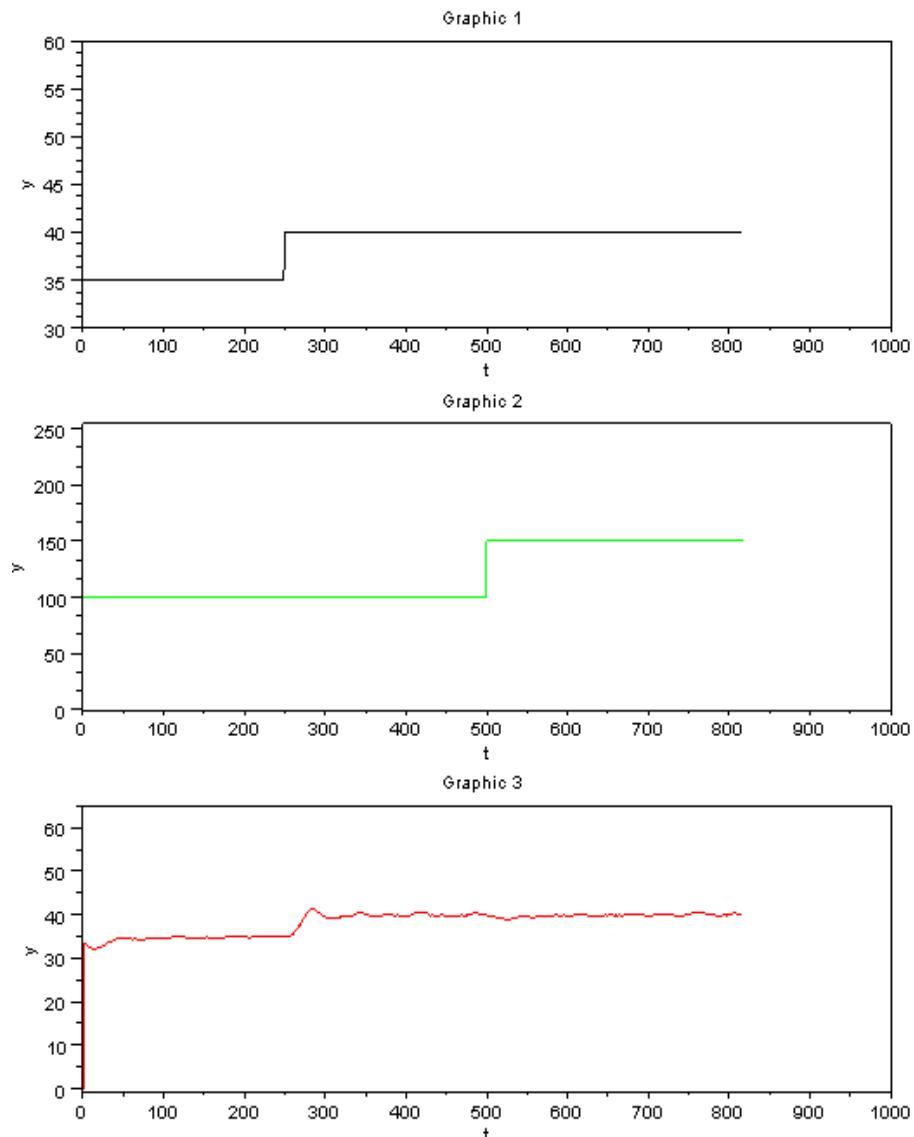


Figure 11.29: Expt 3.2

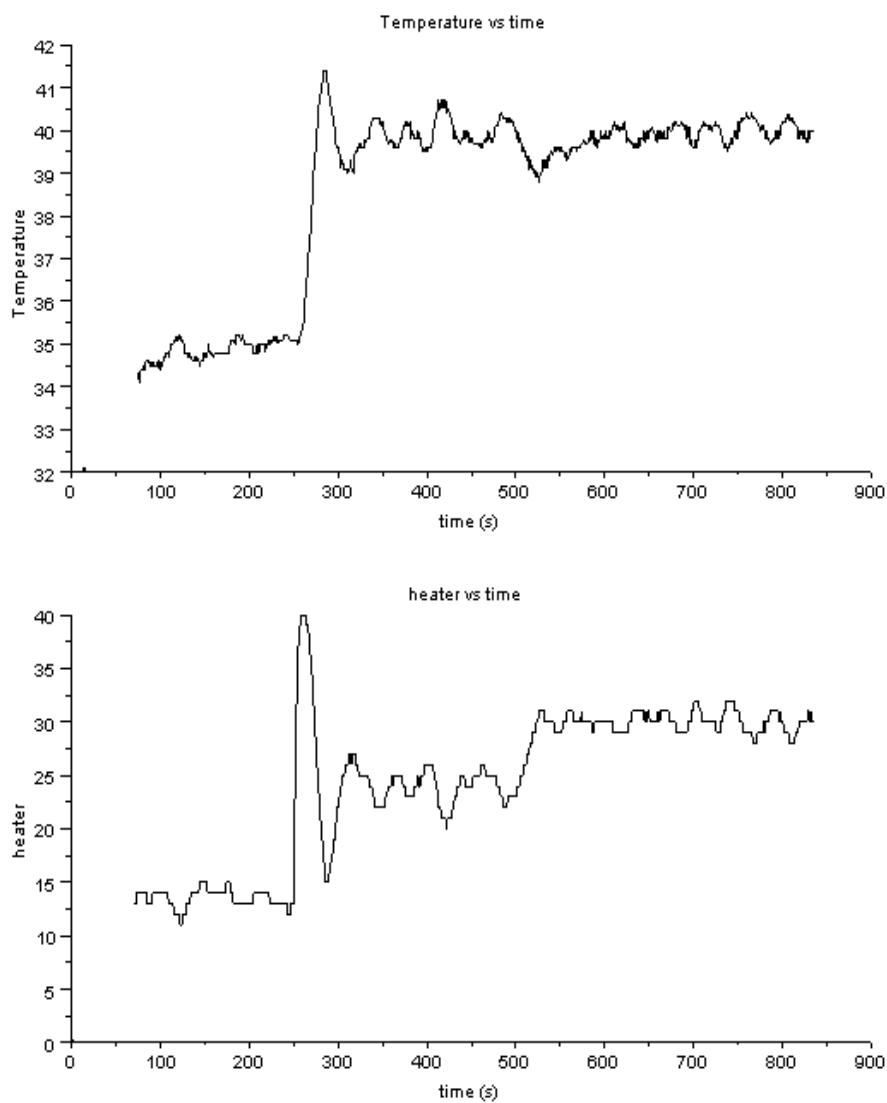


Figure 11.30: Expt 3.2

11.19.3 For q = 4 (Expt 3.3)

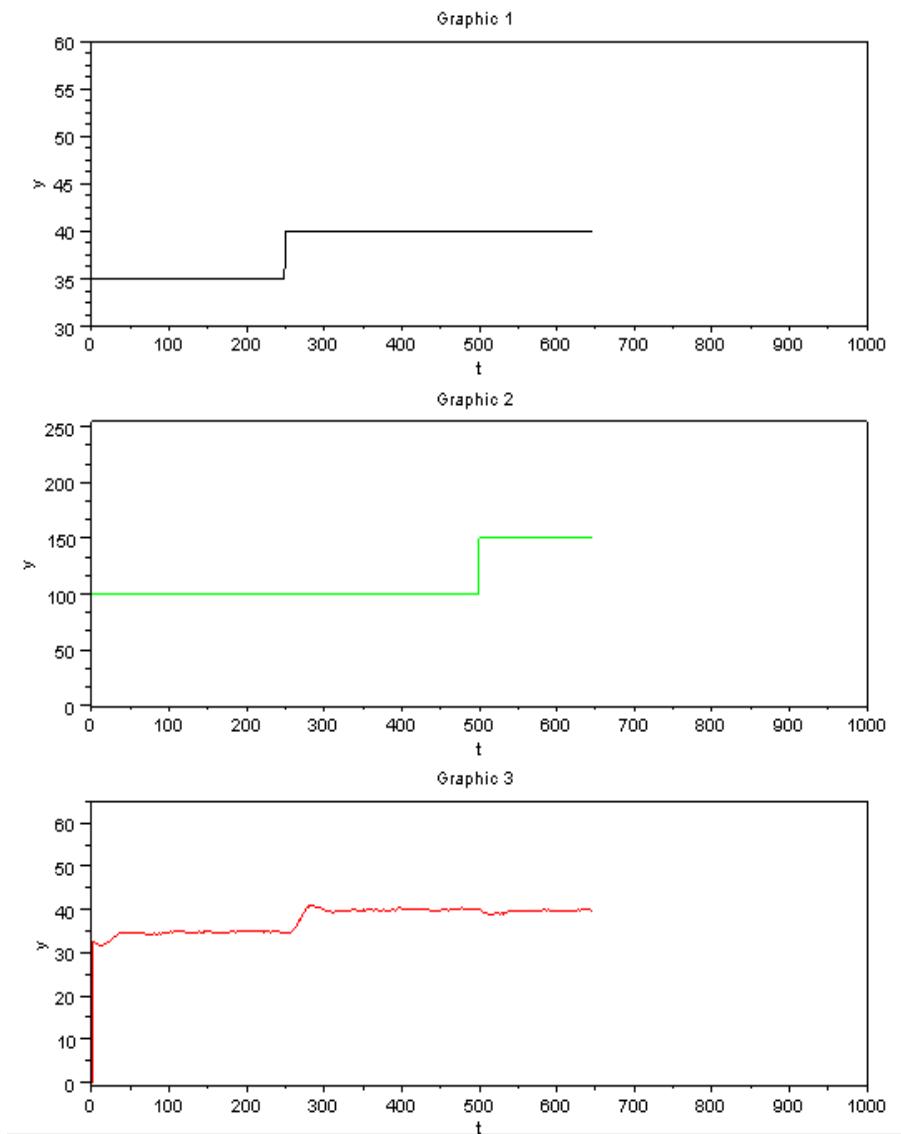


Figure 11.31: Expt 3.3

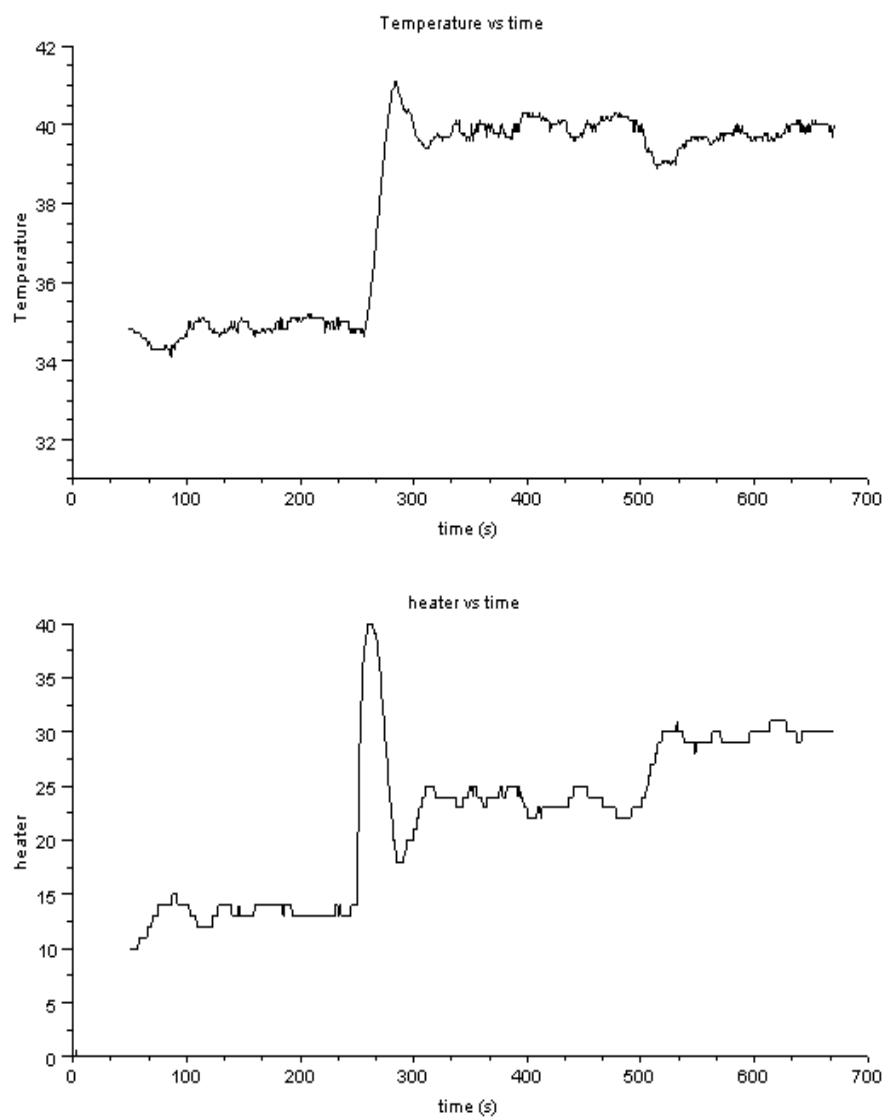


Figure 11.32: Expt 3.3

11.20 For negative step change in Set point and Fan speed

11.20.1 For q =2 (Expt 4.1)

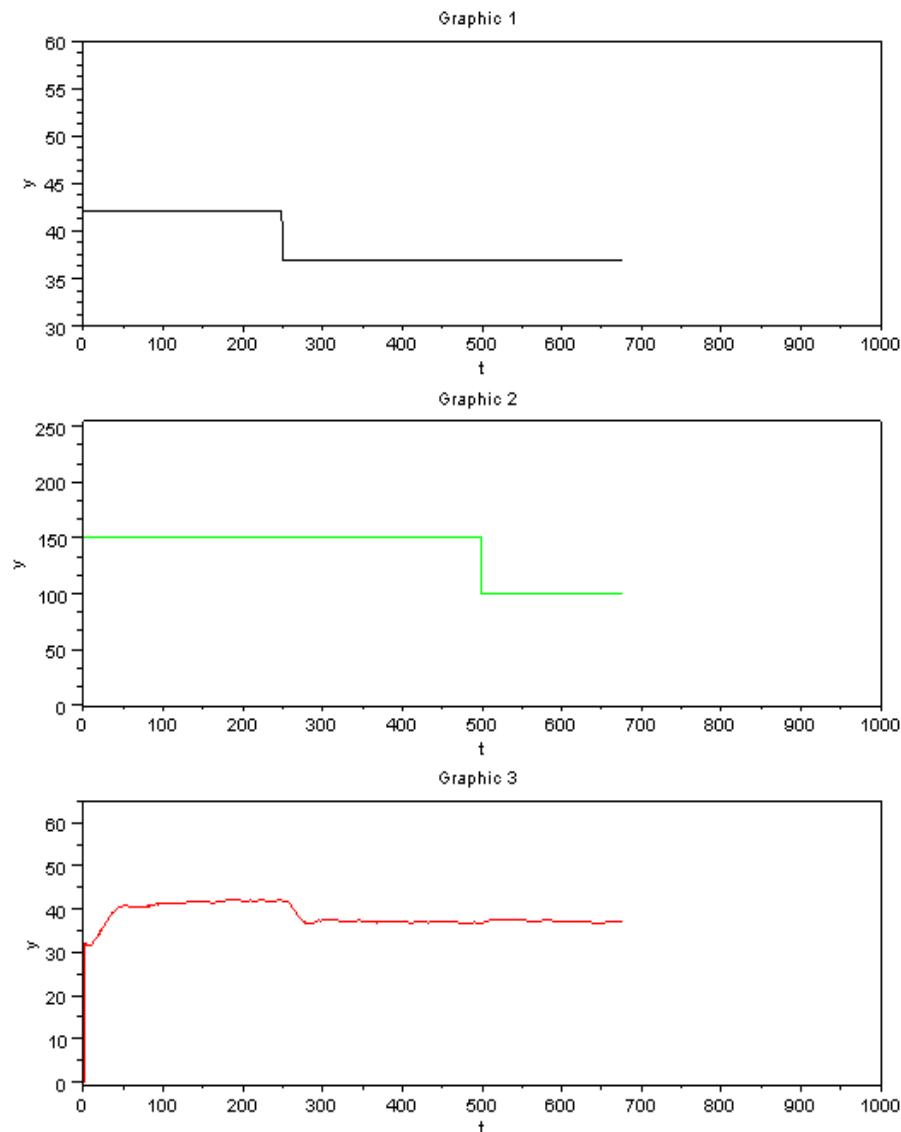


Figure 11.33: Expt 4.1

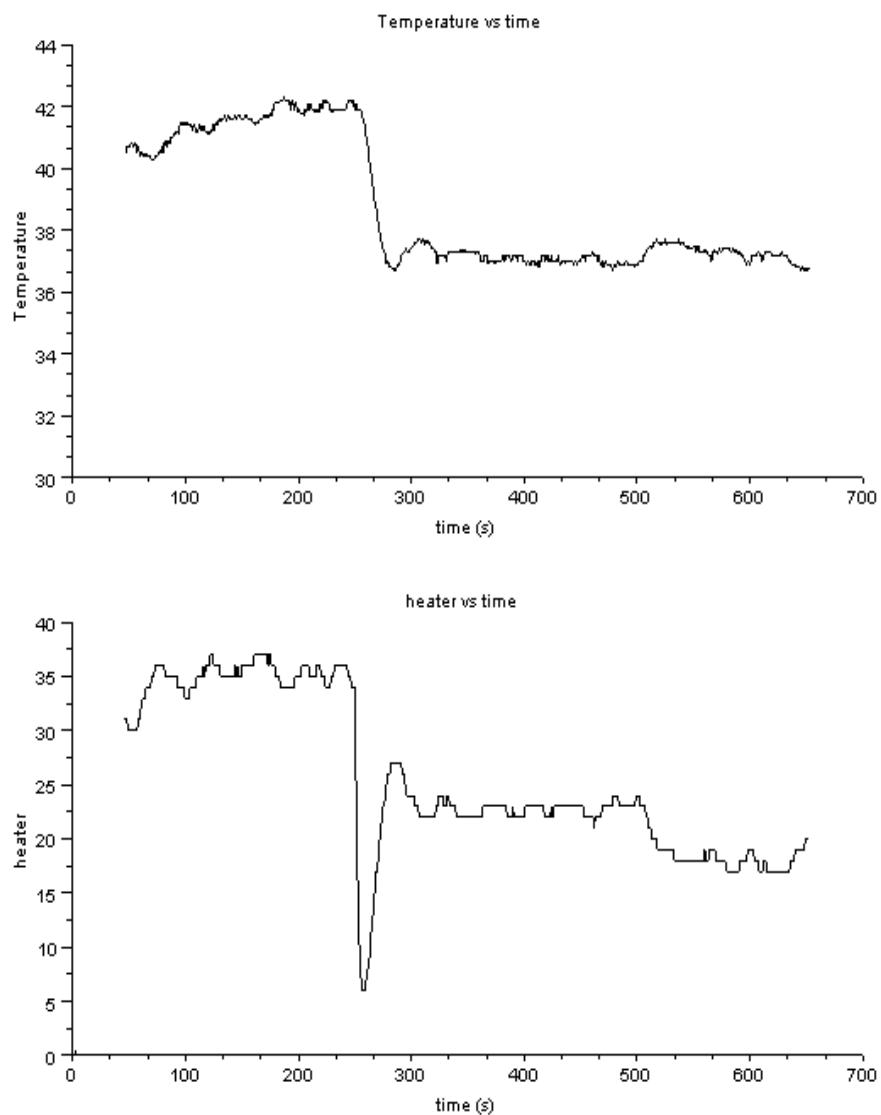


Figure 11.34: Expt 4.1

11.20.2 For $q = 3$ (Expt 4.2)

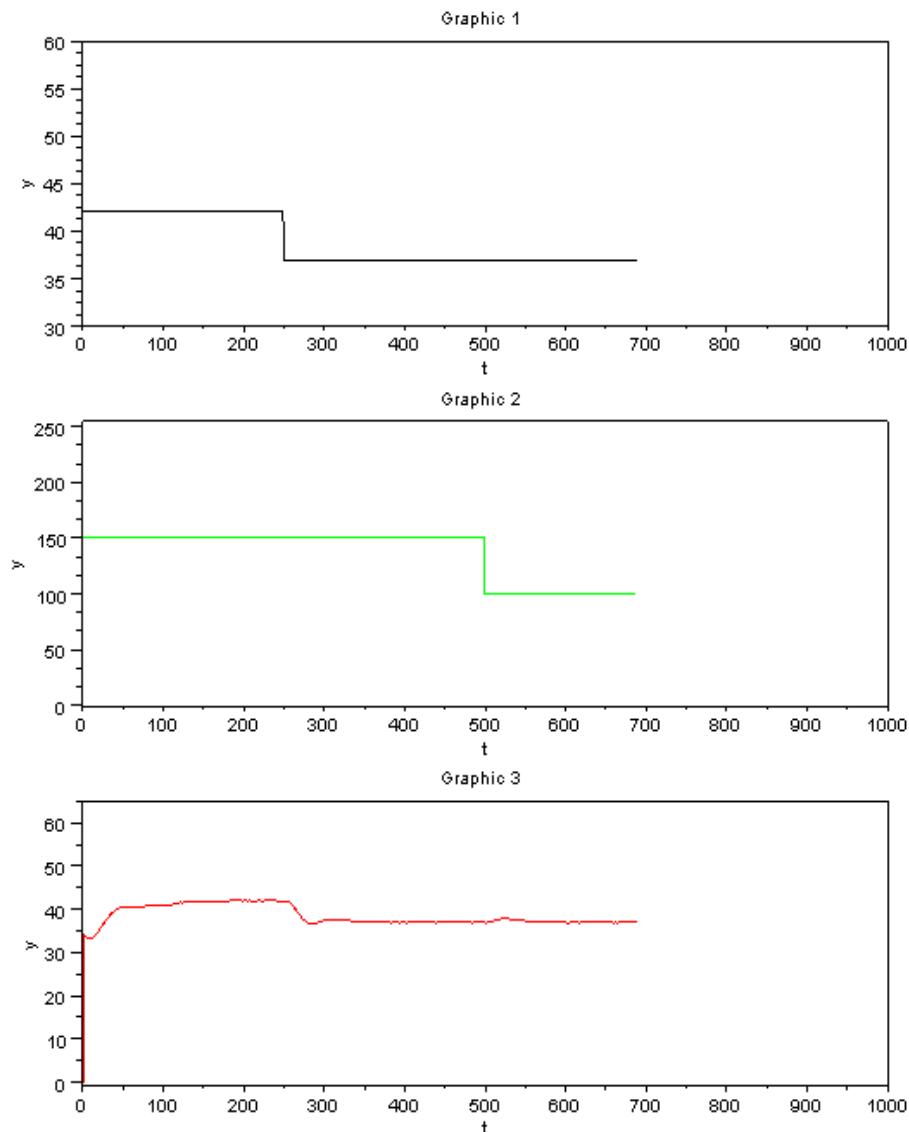


Figure 11.35: Expt 4.2

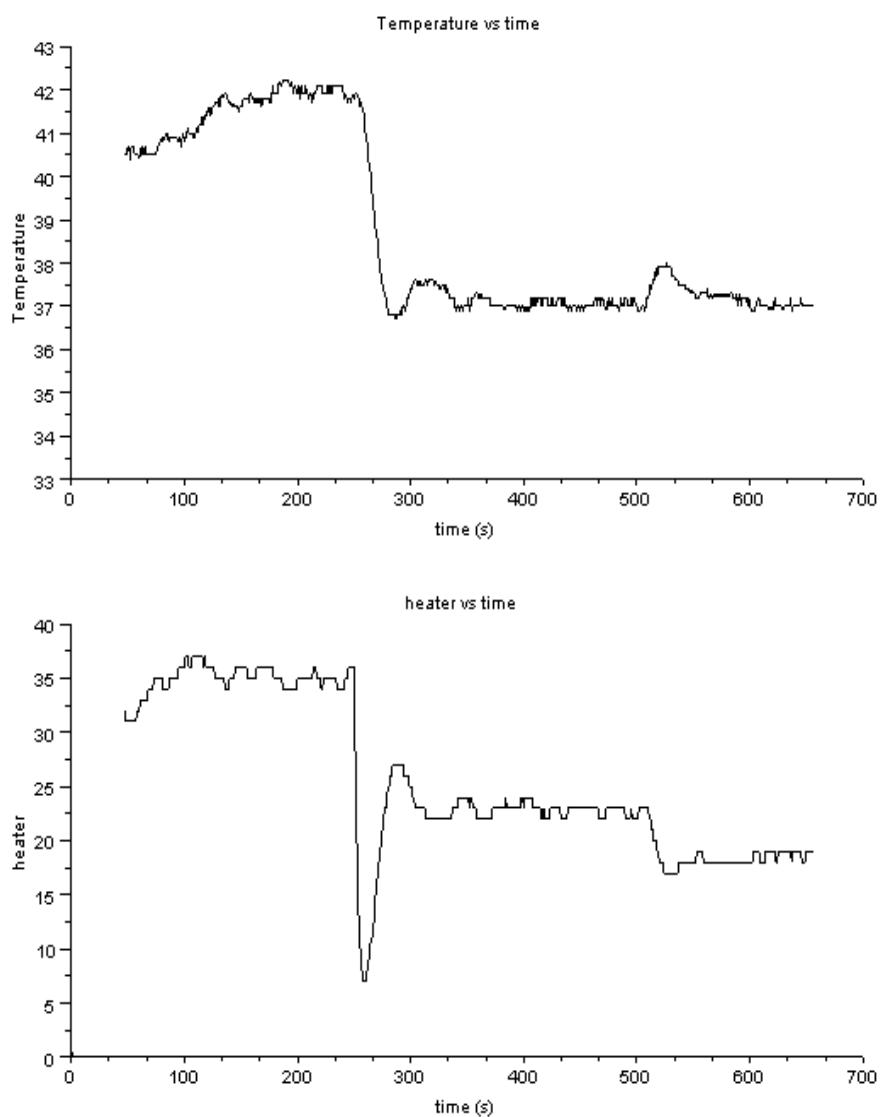


Figure 11.36: Expt 4.2

11.20.3 For $q = 4$ (Expt 4.3)

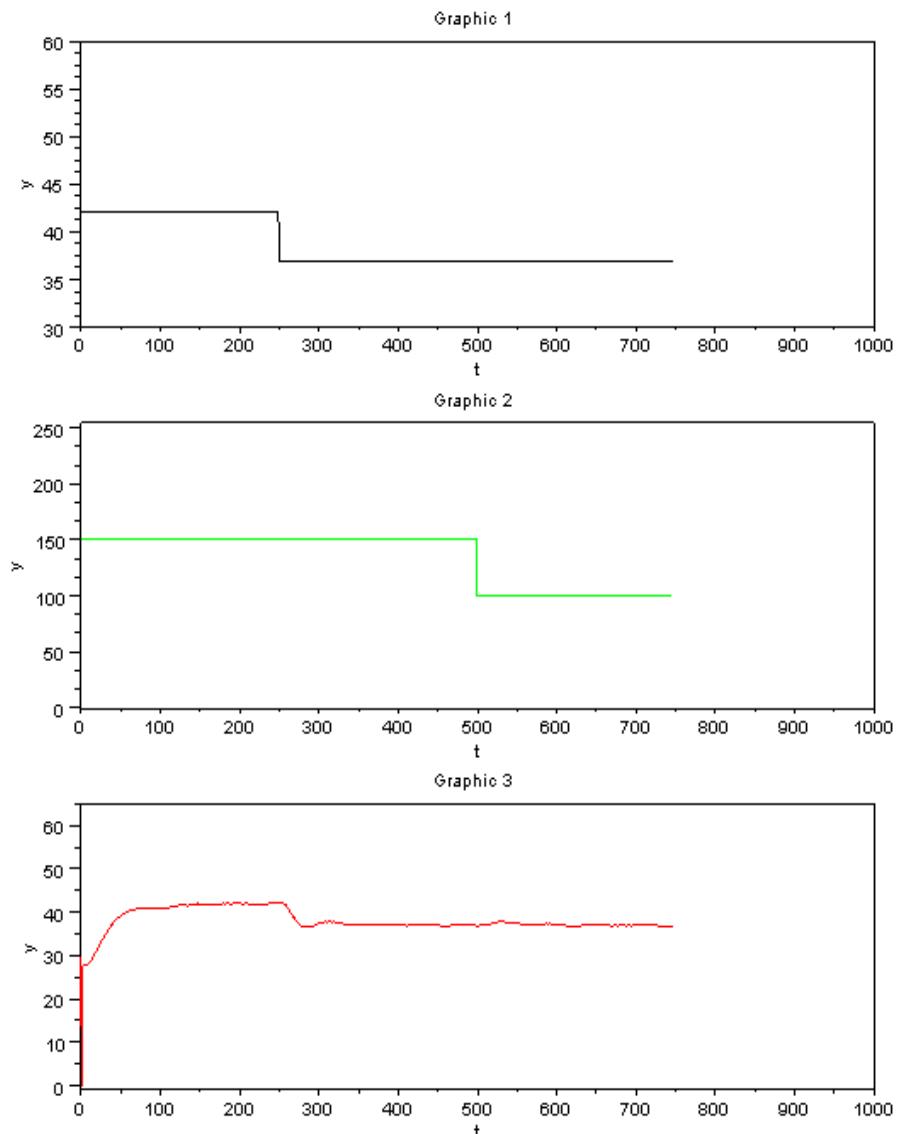


Figure 11.37: Expt 4.3

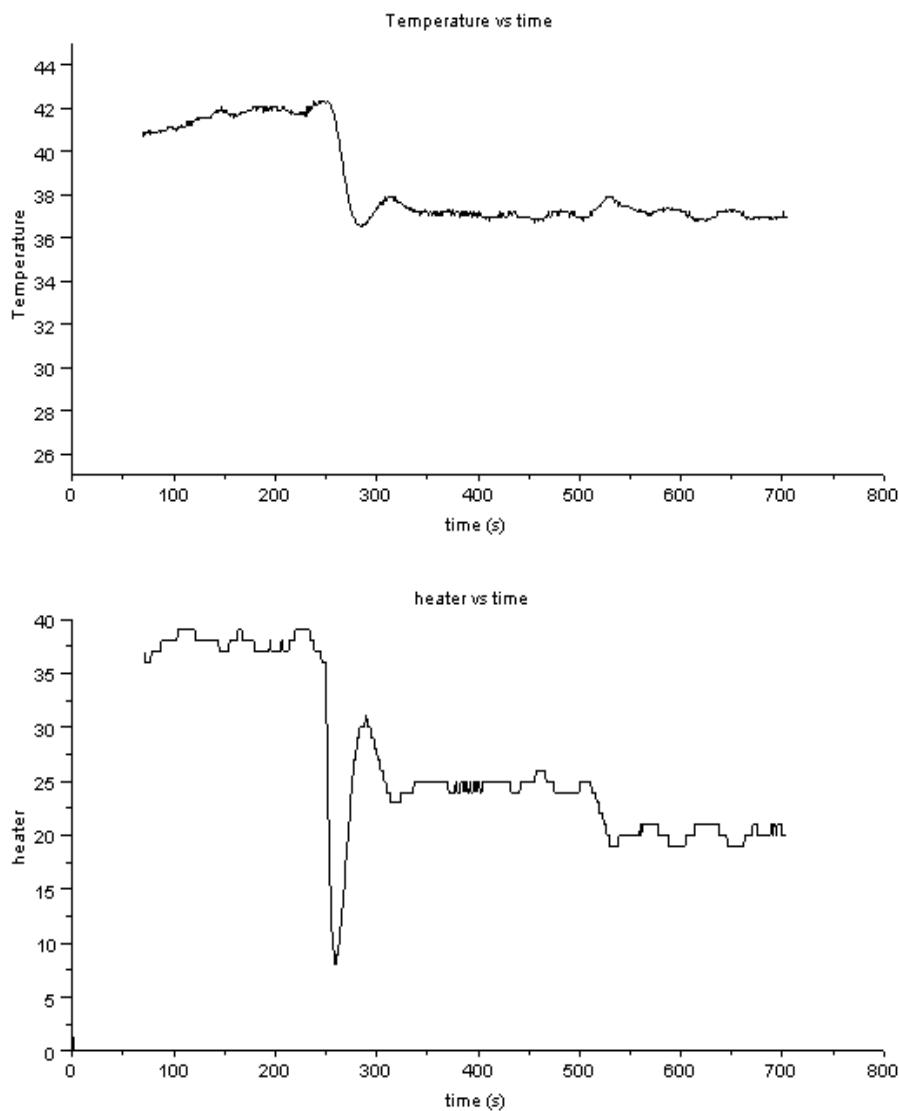


Figure 11.38: Expt 4.3

11.21 Conclusion on the effect of Control Horizon parameter

- The effect of change in q isn't very distinct in the experiments performed.

- While we are calculating the optimized value of manipulated variable at a time, the number of manipulated input moves is increasing as we are increasing the q value.
- But, only the first value of the optimized manipulated variable vector is used for control.
- Increase in q is only increasing the length of the manipulated variable vector which is to be optimized.
- Since, only the first value of manipulated variable vector is used, which itself lies in some specified range, the effect of changing q isn't very significant for SBHS.
- Also, SBHS system is a simple system with very few variables (as compared to real life industrial systems).
- Ideally, the value of q is to be maintained at 3 or 4.

11.22 Implementing Model Predictive controller on SBHS, locally

Change the working directory of scilab to the folder where the local mpc code are kept. Run the `ser_init.sce` file with the appropriate com port number. Then execute the file `mpc_init`. Launch xcos and execute the file `mpc.xcos`. The code is listed in Section 11.29

11.23 Conclusion for MPC project

The objective of this project, ie, implementing Model Predictive Control in Single Board Heater System using Scilab was successfully achieved. Several experiments were successfully performed using the developed SCILAB MPC algorithm for both positive and negative step changes in both temperature-set-point and the disturbance variable (fan).

In addition to the above objective, we also tried studying the effect of weighting factors (tuning parameter) and control horizon parameter. We observed and

concluded that increase in values of We (error weighting factor), increases oscillations and decreases settling time, while decrease in We leads to opposite effect. Wu (manipulated variable weighting factor), on the other hand has an opposite effect. It decreases oscillations and increases settling time with increase in its value. Hence, better control is obtained for high value of We and low value of Wu .

Thus, with this project, we were able to implement MPC successfully and also were able to comment on the general preferred tuning parameters (weighting factors for error and manipulated variable).

11.24 Acknowledgement

Firstly, I would like to thank Prof Moudgalya Kannan, for giving me this opportunity to undertake MPC project on SBHS. This project, which involved implementing Model Predictive Control in SBHS using SCILAB, was very interesting and provided an excellent learning opportunity. For developing the MPC algorithm, lecture notes on Model Predictive Control by Prof Sachin Patwardhan too were extremely helpful. Also, I got to learn a lot from the speaking tutorials of SCILAB and LaTeX, which had to be referred to for the completion of this project. Over and above this, it was very encouraging to see the experiments working perfectly with the developed Model Predictive Control algorithm.

I would also like to sincerely thank Mr Prashant Gupta, without whom, this project would not have been splendidly completed. I would like to thank him for the time he spent explaining the concepts, clearing the doubts and suggestions for the experiments to implement MPC.

11.25 Appendix

11.26 Appendix 1: General Information on Experiments for this Project

All the experiments for this project was performed remotely on SBHS 12, using a sampling time of 1 second. Basic codes (mpc_init.sce and mpc.sci) was taken from moodle for this course. Code for implementing MPC was written in scilab and has been mentioned in the report.

Scilab Version used: 5.2.2

SBHS number: 12 (remotely used)

Sampling time: 1 second

For graphs: Until and unless mentioned, Graphic 1 represents the Temperature set point, Graphic 2 represents the Fan and Graphic 3 represents the Temperature.

11.27 Appendix 2: Values of State Space matrices

Initially, open loop experiment was performed, and Plant Transfer function was obtained. For the open loop experiment, a step change in heater from 15 to 25 units at $t = 200$ seconds was provided (sampling time 1s). The response data was fitted to a first order transfer function with a time delay and the following was observed:

$K_p=0.37$, time constant = 45s and delay = 7s.

Using the above, we obtained the plant transfer function:

$$G_p = \frac{0.37}{1 + 45s} e^{-7s} \quad (11.7)$$

Scilab Method to calculate State Space matrices

State space matrices for a transfer function can be calculated as follows using Scilab:

```

1 s=poly(0,'s');
2 TFcont=syslin('c',[kp*(1-0.5*D)/(tau*s+1)/(1+0.5*D)]);
3 SScont=tf2ss(TFcont);

```

SScont (in the last line above), has the value of the required State Space matrices.
 (Please note: Time delays can not be directly handled in Scilab. So, for systems with delays, we will have to use alternate approach. Pade's approximation for time delay being one of the approach.)

The transfer function which we derived for our SBHS was very close to the transfer function derived by Mr Prashant Gupta. So, using the values of A, B and C which were already calculated by him previously, we obtain the following exact values:

$$A = \begin{bmatrix} 0.9780 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

$$B = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

$$C = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.0079 \end{bmatrix}$$

11.28 Appendix 3: Attachments and Contact Information

Attachments

- Folder named *codes*, which contains all the codes used for the experiments.
The codes in this folder should be used to reproduce MPC control experiments mentioned in this report.
- Folder named *data files*, which contains data files for all the experiments performed
- MPC Report

Contact Information

My details:

Pratik Behera (07002054)
pratik_behera@iitb.ac.in
+91 99871 54061

11.29 Scilab Code

Scilab Code 11.1 mpc_init.sce

```
1 // For scilab 5.1.1 or lower version users ,
2 // use scicos command to open scicos diagrams instead
3 // of xcos
4 global fdfh fdt fnctr fnctr m err_count y p q xk_old
5
6 p = 40; // prediction horizon
7 q = 4; // control horizon
8 xk_old = zeros(8,1);
9
10 fnctr = 'clientread.sce';
11 fdt = mopen(fnctr);
12 mseek(0);
13
```

```

14 err_count = 0; // initialising error count for network
15   error
16 m =1;
17 exec ("mpc.sci");
18 exec("mpc_run.sci");
19
20 fdfh = mopen('clientwrite.sce');
21 mseek(0);
22 b = mgetl(fdfh,1);
23 a = mgetl(fdt,1);
24 mclose(fdfh);
25
26 if a ~= []
27   if b~= []
28     disp ("ERROR!EMPTY THE CLIENTREAD AND CLIENTWRITE
29       FILES");
30   return
31 else
32   disp ("ERROR!EMPTY THE CLIENTREAD FILE");
33   return
34 end
35 else
36   if b~= []
37     disp ("ERROR!EMPTY THE CLIENTWRITE FILE");
38   return
39 end;
40 A = [0.1,m,0,251];
41 fdfh = file('open','clientwrite.sce','unknown');
42 write(fdfh,A,'(7(e11.5,1x))');
43 file('close',fdfh);
44 sleep(1000);
45 a = mgetl(fdt,1);
46 mseek(0);
47 if a~= [] // open xcos only if communication is
48   through
49   xcos('mpc.xcos');
50 else
51   disp ("NO NETWORK CONNECTION!");

```

```
49     return  
50 end
```

Scilab Code 11.2 mpc.sci

```
1 function [y,stop] = mpc(Tsp,fan)  
2 global fdfh fdt fncre fncrew m err_count stop p q xk_old  
3  
4 fncre = 'clientread.sce';  
5 fncrew = 'clientwrite.sce';  
6  
7 a = mgetl(fdt,1);  
8 b = evstr(a);  
9 byte = mtell(fdt);  
10 mseek(byte,fdt,'set');  
11  
12 if a~= []  
13     temps = b(1,$); heats = b(1,$-2);  
14     fans = b(1,$-1); y = temps;  
15  
16     heat = mpc_run(y,heats,Tsp);  
17  
18     if heat>40  
19         heat = 40;  
20     elseif heat<0  
21         heat = 0;  
22     end;  
23  
24 A = [m,m,heat,fan];  
25 fdfh = file('open','clientwrite.sce','unknown');  
26 file('last',fdfh)  
27 write(fdfh,A,'(7(e11.5,1x))');  
28 file('close',fdfh);  
29 m = m+1;  
30  
31 else  
32     y = 0;
```

```

33     err_count = err_count + 1; // counts the no of
34         times network error occurs
35 if err_count > 300
36     disp("NO NETWORK COMMUNICATION!");
37     stop = 1; // status set for stopping simulation
38 end
39
40 return
41 endfunction

```

Scilab Code 11.3 mpc_init_local.sce

```

1 // For scilab 5.1.1 or lower version users ,
2 // use scicos command to open scicos diagrams instead
3 // of xcos
4
5 global err_count y p q xk_old Tsp heats fan temp heat
6 p = 40; // prediction horizon
7 q = 4; // control horizon
8 xk_old = zeros(8,1);
9 Tsp=1;
10 heats=1;
11 fan=1;
12 temp=1;
13
14 exec ("mpc_local.sci");
15 exec ("mpc_run.sci");

```

Scilab Code 11.4 mpc_local.sci

```

1
2 global temp heat heats et SP u_new u_old u_new e_old
3     e_new err_count stop p q xk_old
4 function [temp] = mpc(Tsp,fan)
5

```

```

6 u_new = mpc_run(temp,heats,Tsp);
7
8
9 if u_new>40
10    u_new = 40;
11 end;
12
13 if u_new<0
14    u_new = 0
15    ;
16 end;
17
18 heat=u_new;
19
20 heats=heat;
21
22
23 writeserial(handl,ascii(254)); // Input Heater,
   writeserial accepts strings; so convert 254 into
   its string equivalent
24 writeserial(handl,ascii(heat));
25 writeserial(handl,ascii(253)); // Input Fan
26 writeserial(handl,ascii(fan));
27 writeserial(handl,ascii(255)); // To read Temp
28 sleep(100);
29
30 temp = ascii(readserial(handl)); // Read serial
   returns a string, so convert it to its integer(
   ascii) equivalent
31 temp = temp(1) + 0.1*temp(2); // convert to temp with
   decimal points eg: 40.7
32
33 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.