Documentation for Single Board Heater System

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

IIT Bombay
June 13, 2014

Contents

List of Scilab Code

Chapter 1

Block diagram explanation of Single Board Heater System

Figure1.1 shows the block diagram of 'Single Board Heater System'(SBHS). Microcontroller ATmega16 is used at the heart of the setup. The microcontroller can be programmed with the help of an In-system programmer port(ISP) available on the board. The setup can be connected to a computer via two serial communication ports namely RS232 and USB. A particular port can be selected by setting the jumper to its appropriate place. The communication between PC and setup takes place via a serial to TTL interface. The μC operates the Heater and Fan with the help of separate drivers. The driver comprises of a power MOSFET. A temperature sensor is used to sense the temperature and feed to the μC through an Instrumentation Amplifier. Some required parameter values are also displayed along with some LED indications.

1.1 Microcontroller

Some salient features of ATmega16 are listed below:

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

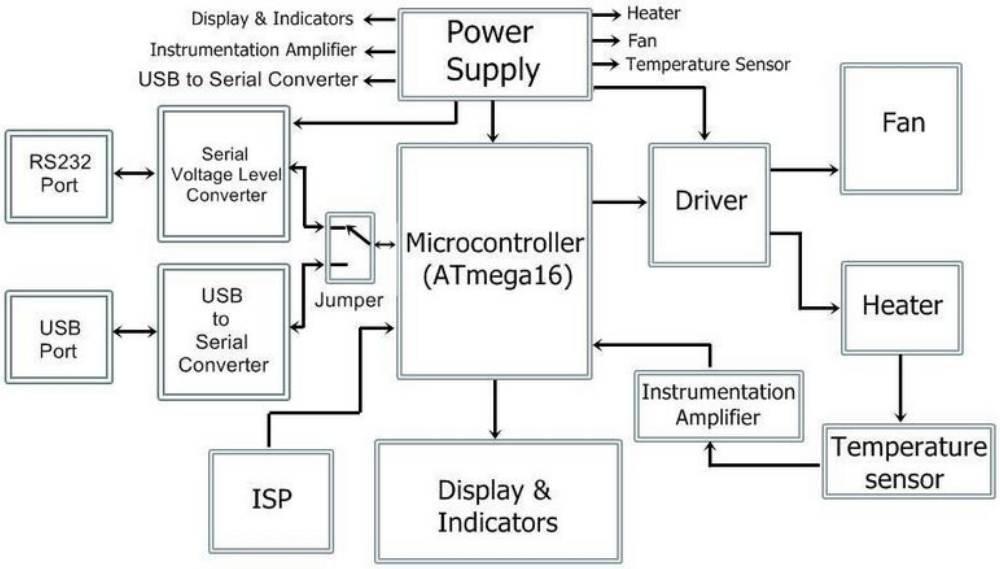

Figure 1.1: Block Diagram

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

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

1.1.1 PWM for heat and speed control

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

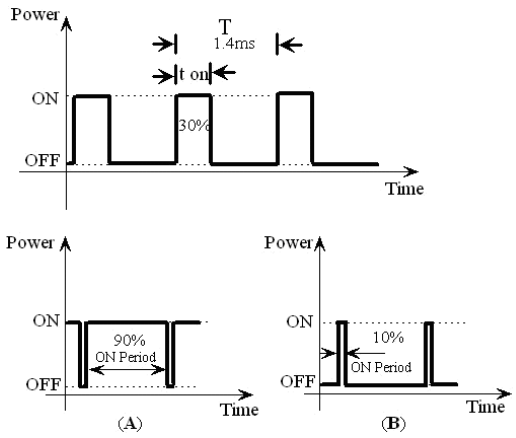

Figure 1.2: Pulse Width Modulation (A): On time is 90% of the total time period, (B): ON time is 10% of total time period

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

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

Where T_{ON} is the ON time of the wave corresponding to the HIGH state of logic and T is the total time period of the wave. Power delivered to the load is proportional to T_{ON} time of the signal. This is used to control the current flowing through the heating element and also speed of the fan. An internal timer of the microcontroller is used to generate a square wave. The ON time of the square wave depends on a count value set in the internal timer. The pulse width of the waveform can be varied accordingly by varying this count value. Thus, PWM waveform is generated at the appropriate pin of the microcontroller. This generated PWM waveform is used to control the power delivered to the load (Fan and Heater).
A MOSFET is used to switch at the PWM frequency which indirectly controls the power delivered to the load. A separate MOSFET is used to control the power delivered to each of the two loads. The timer is operated at 244Hz.

1.1.2 Analog to Digital conversion

As explained earlier, the heat generated by the heater coil is passed to the iron plate through convection. The temperature of this plate is measured by using a temperature sensor AD590.
Some of the salient features of AD590 include:

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

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

1.2 Instrumentation amplifier

Instrumentation Amplifiers are often used in temperature measurement circuits in order to boost the output of the temperature sensors. A typical three Op-Amp Instrumentation amplifier is shown in the figure 1.3. The Instrumentation Amplifiers (IAs) are mostly preferred, where the sensor is located at a remote place and therefore is susceptible to signal attenuation, due to their very low DC offsets, high input impedance, very high Common mode rejection ratio (CMRR). The IAs have a very high input impedance and hence do not load the input signal source. IC LM348 is used to construct a 3 Op-Amp IA. IC LM348 contains a set of four Op-Amps. Gain of the amplifier is given by equation 1.2

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

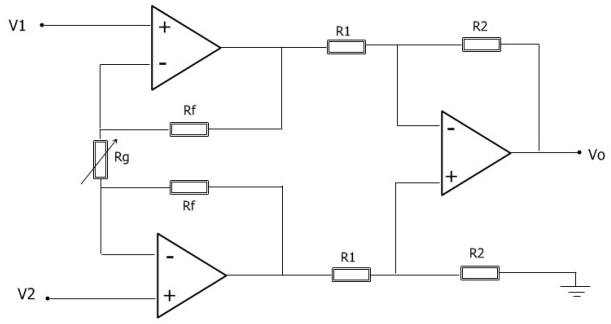

Figure 1.3: 3 Op-Amp Instrumentation Amplifier

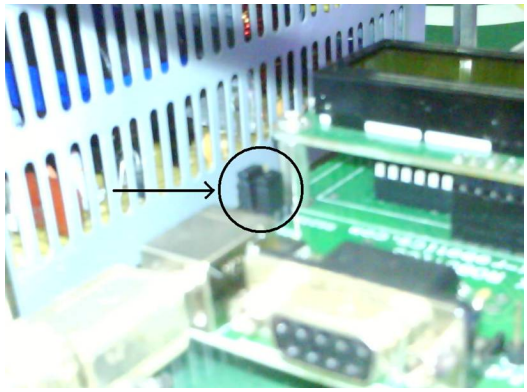

Figure 1.4: Jumper arrangement

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

1.3 Communication

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

Figure 1.5: RS232 cable

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

1.3.1 Serial port communication

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

1.3.2 Using USB for Communication

After setting the jumper to USB mode connect the set up to the computer using a USB cable at appropriate ports as shown in the figure 1.8. To make the setup USB compatible, USB to serial conversion is carried out using IC FT232R. Note that proper USB driver should be installed on the computer.

1.4 Display and Resetting the setup

The temperature of the plate, percentage values of Heat and Fan and the machine identification number (MID) are displayed on LCD connected to the microcon-

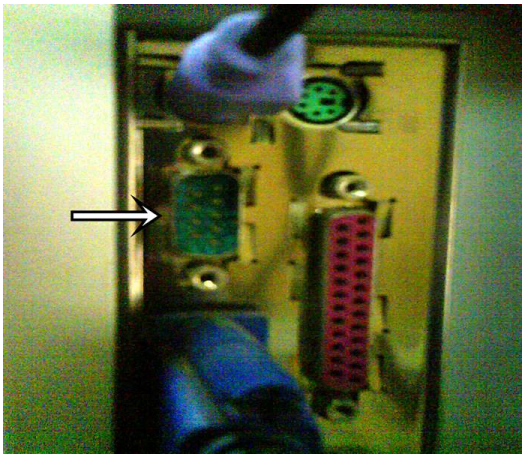

Figure 1.6: Serial port

Figure 1.7: USB communication

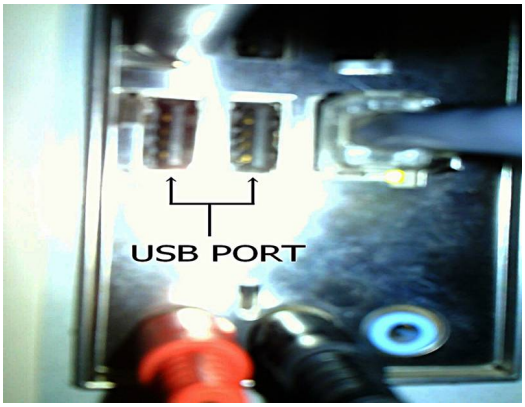

Figure 1.8: USB PORT

Figure 1.9: Display

troller. As shown in figure 1.9, numerals below TEMP indicate the actual temperature of the heater plate in °C. Numerals below HEA and FAN indicate the respective percentage values at which heater and fan are being operated. Numerals below MID corresponds to the device identification number. The set up could be reset at any time using the reset button shown in figure 1.10. Resetting the setup takes it to the standby mode where the heater current is forced to be zero and fan speed is set to the maximum value. Although these reset values are not displayed on the LCD display these are preloaded to the appropriate units.

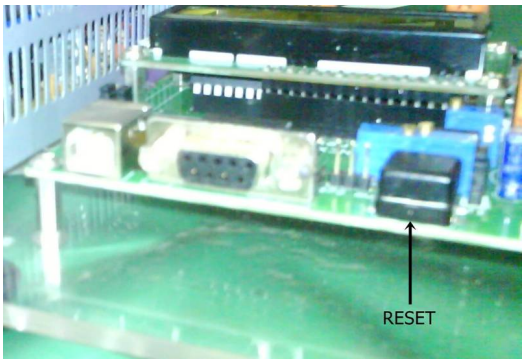

Figure 1.10: Reset

Chapter 2

Performing a Local Experiment on Single Board Heater System

This chapter explains the procedure to use Single Board Heater System locally with Scilab i.e. when you are physically accessing SBHS using your computer. An open loop experiment, step test is used for demonstrating this procedure. The process however remains the same for performing any other experiment explained in this document, unless specified otherwise.

Hardware and Software Requirements

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

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

2.1 Using SBHS on a Windows OS

This section deals with the procedure to use SBHS on a Windows Operating System. The Operating System used for this document is Windows 7, 32-bit OS. If you are using some other Operating System or the steps explained in section 2.1.1 are not sufficient to understand, refer to the official document available on the main ftdi website at `www.ftdichip.com`. On the left hand side panel, click on 'Drivers'. In the drop-down menu, choose 'VCP Drivers'. Then on the web page page, click on 'Installation Guides' link. Choose the required OS document. We would now begin with the procedure.

2.1.1 Installing Drivers and Configuring COM Port

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

Now, the communication port number assigned to the computer port to which the Single Board Heater System is connected, via an RS232 or USB cable should be identified. For identifying this port number, right click on `My Computer` and click on `Properties`. Then, select the `Hardware` tab and click on `Device Manager`. The list of hardware devices will be displayed. Locate the Ports(COM & LPT) option and click on it. The various communication ports used by the computer will be displayed. If the SBHS is connected via RS232 cable, then look for Communications Port(COM1) else look for USB Serial Port. For RS232 connection, the port number mostly remains COM1. For USB connection it may change to some other number. Note the appropriate COM number. This process is illustrated in figure 2.1

Sometimes the COM port number associated with the USB port after connecting a USB cable may be greater than 9. Since the serial tool box can handle only single digit port number (upto 9), it is necessary to change this COM port number. Following is the procedure to change the COM port number. Double click on the name of the particular port. Click on `Port Settings` tab and then click on

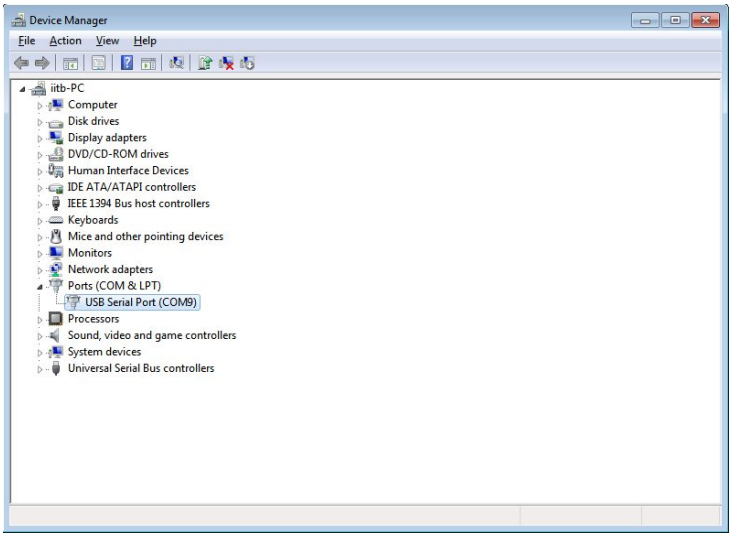

Figure 2.1: Checking Communication Port number

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

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

2.1.2 Steps to Perform a Local Experiment

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

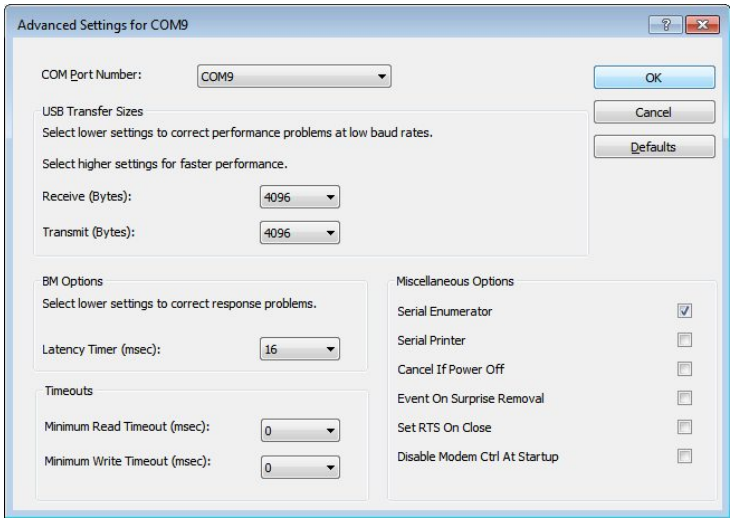

Figure 2.2: Changing Com port number

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

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

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

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

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

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

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

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

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

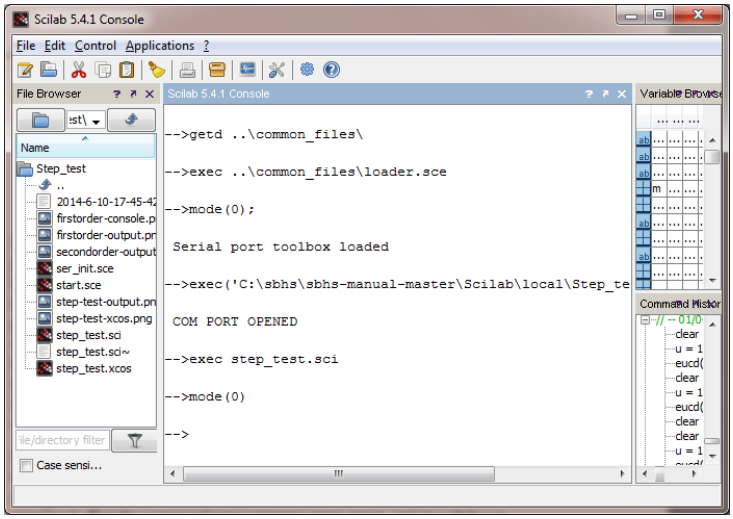

Figure 2.3: Expected responses seen on the console

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

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

2.2 Using Single Board Heater System on a Linux System

This section deals with the procedure to use SBHS on a Linux Operating System. The Operating System used for this document is Ubuntu 12.04. For Linux users, the instructions given in section 2.1 hold true with a few changes as below:

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

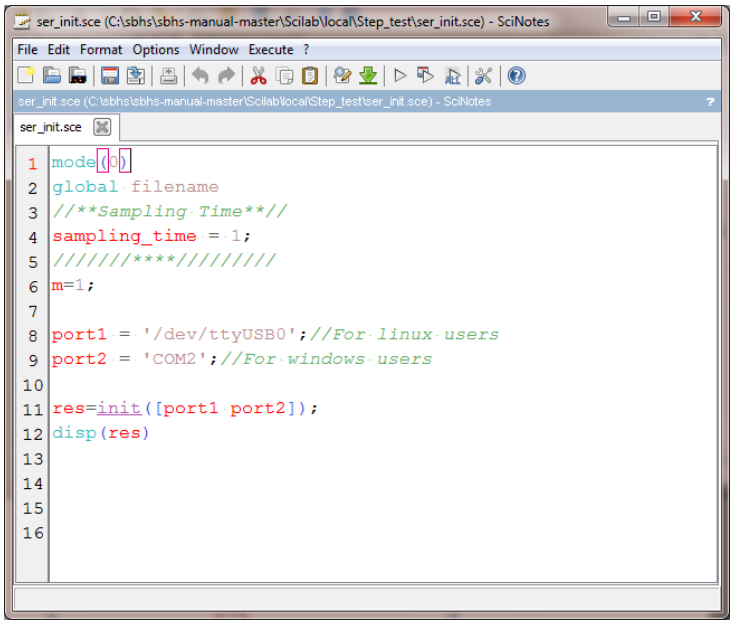

Figure 2.4: Executing script files

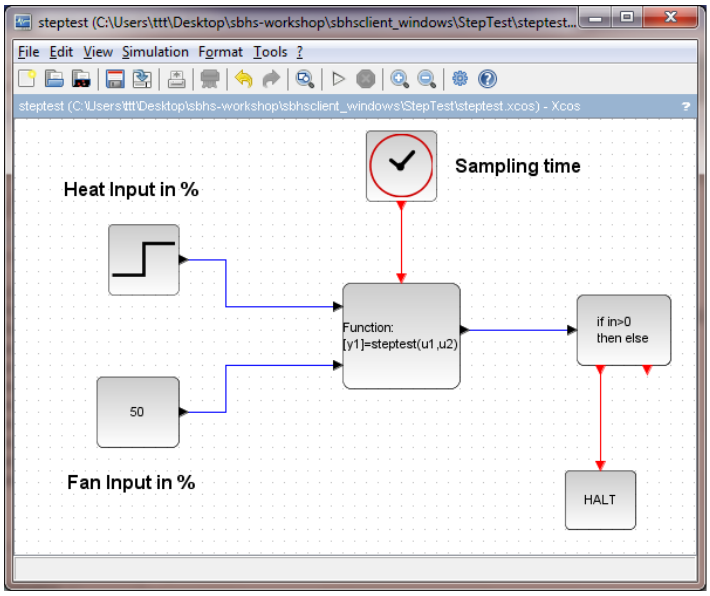

Figure 2.5: Xcos Interface

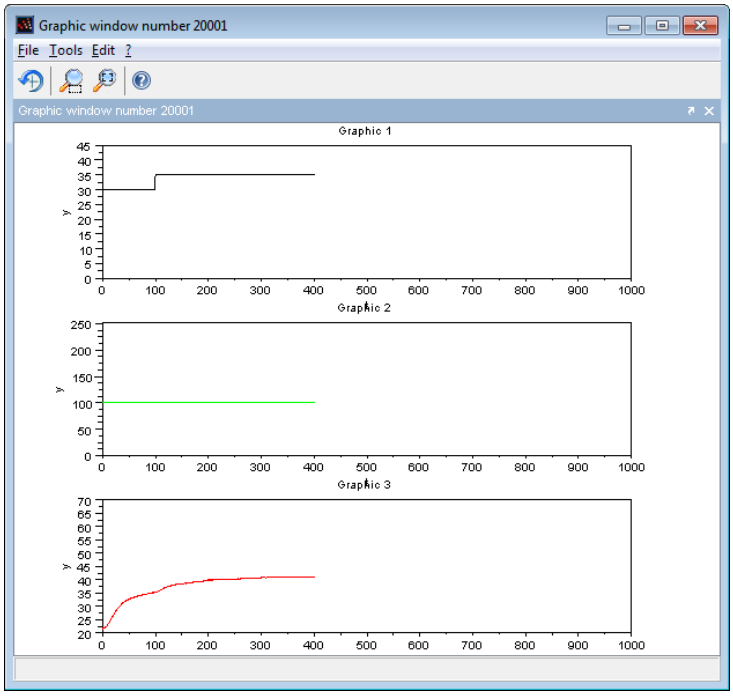

Figure 2.6: Plot obtained after executing step_test.xcos

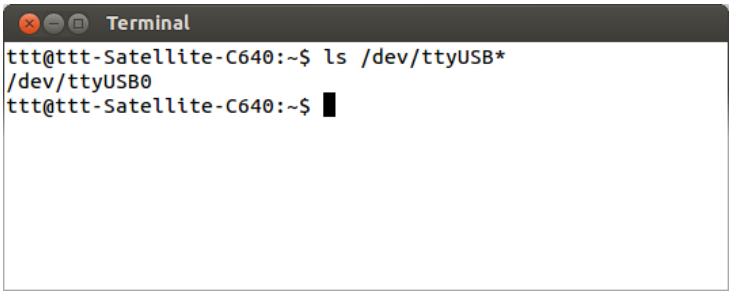

Figure 2.7: Checking the port number in linux (1)

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

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

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

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

Chapter 3

Using Single Board Heater System, Virtually!

3.1 Introduction to Virtual Labs at IIT Bombay

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

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

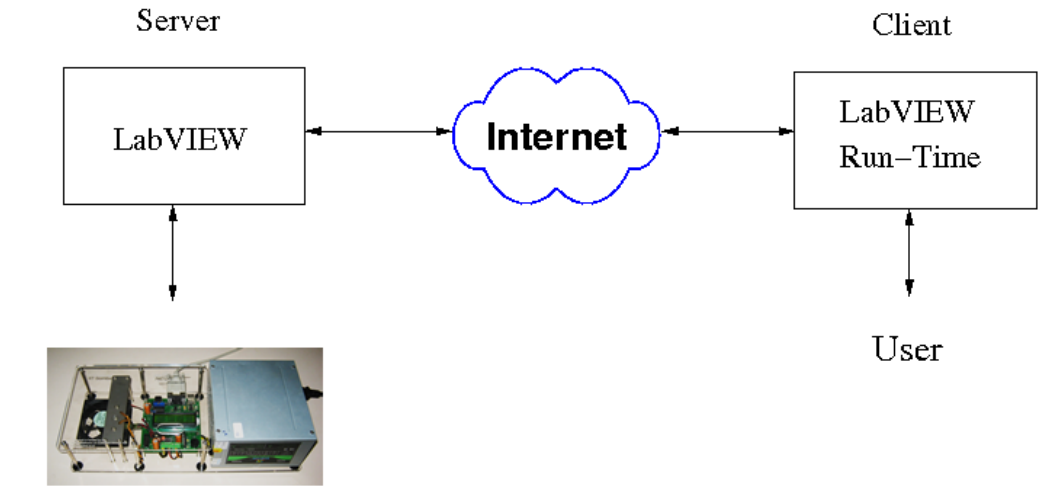

Figure 3.1: SBHS virtual laboratory with remote access using LabVIEW

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

3.2 Evolution of SBHS virtual labs

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

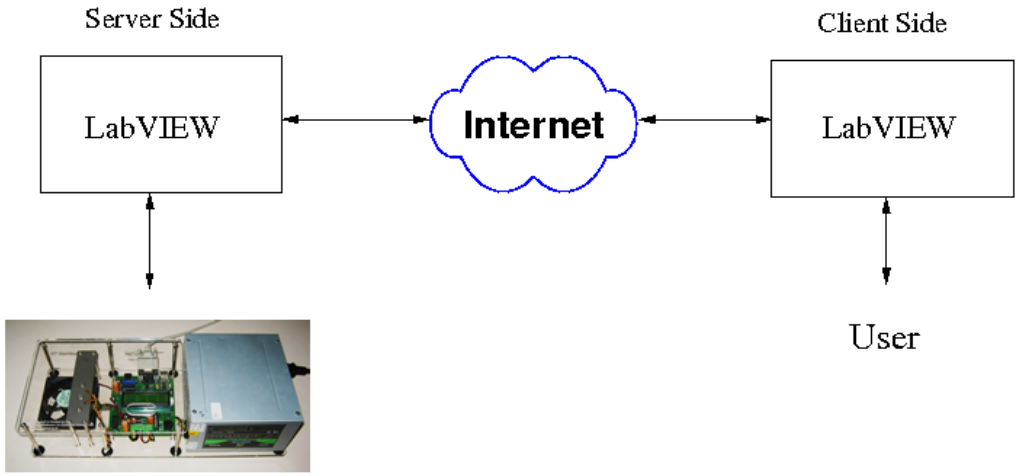

Figure 3.2: SBHS virtual laboratory with remote access and live data sharing using LabVIEW

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

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

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

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

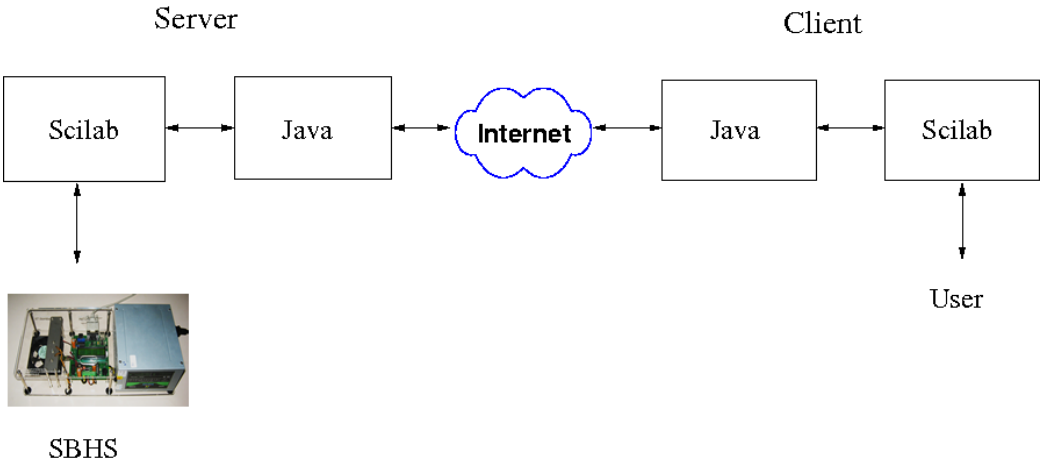

Figure 3.3: SBHS virtual laboratory using open source software

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

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

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

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

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

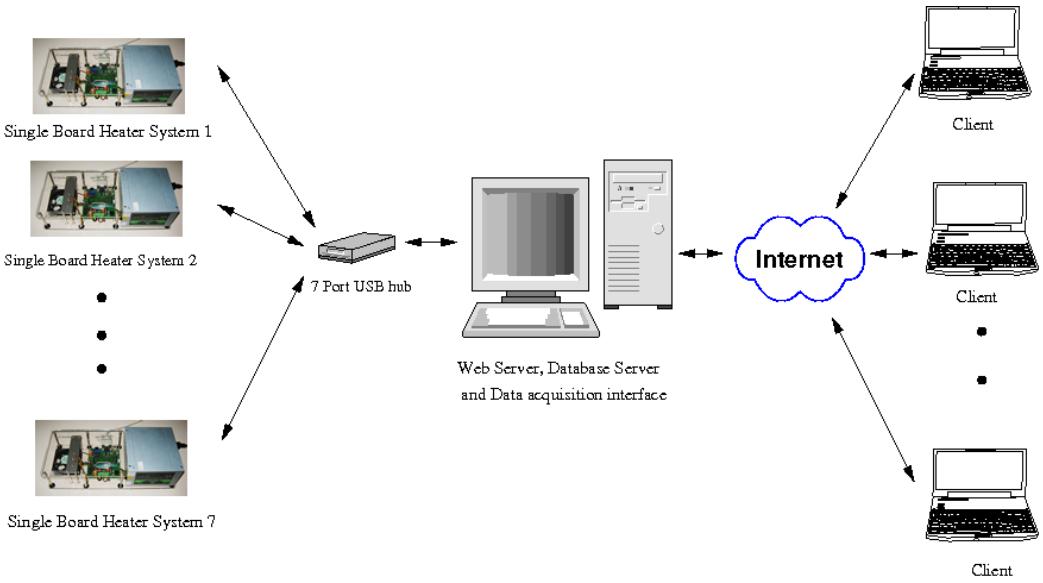

Figure 3.4: Virtual control lab hardware architecture

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

3.3 Current Hardware Architecture

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

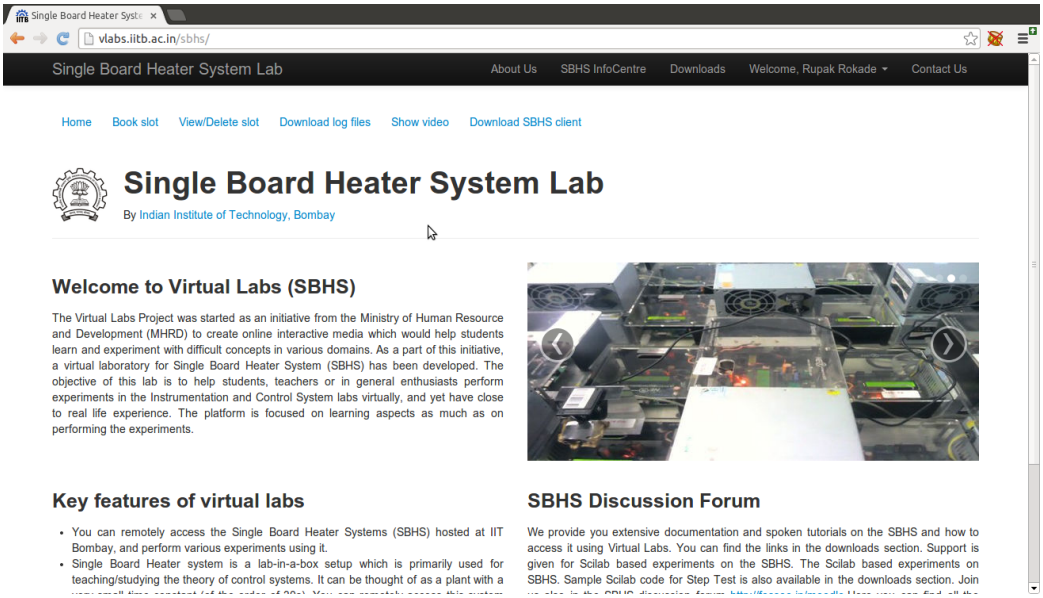

Figure 3.5: Home page of SBHS V Labs

3.4 Current Software Architecture

The current software architecture of this virtual SBHS control lab is shown in Figure 3.6. The server computer runs Ubuntu Linux 12.04.2 OS. It hosts a Apache-MySQL server. The SBHS server is based on Python-Django framework and is linked to Apache server using Apache's WSGI module. The MySQL database server has the details of all the registered users, their slot details, authentication keys to allow remote access, etc. As shown in Figure **??**, the Python-Django server has pages for registration, login, slot booking etc. [9]. On the client end, control algorithms are running in Scilab and a python based client application communicates with virtual labs server over the Internet.

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

3.5 Conducting experiments using the Virtual lab

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

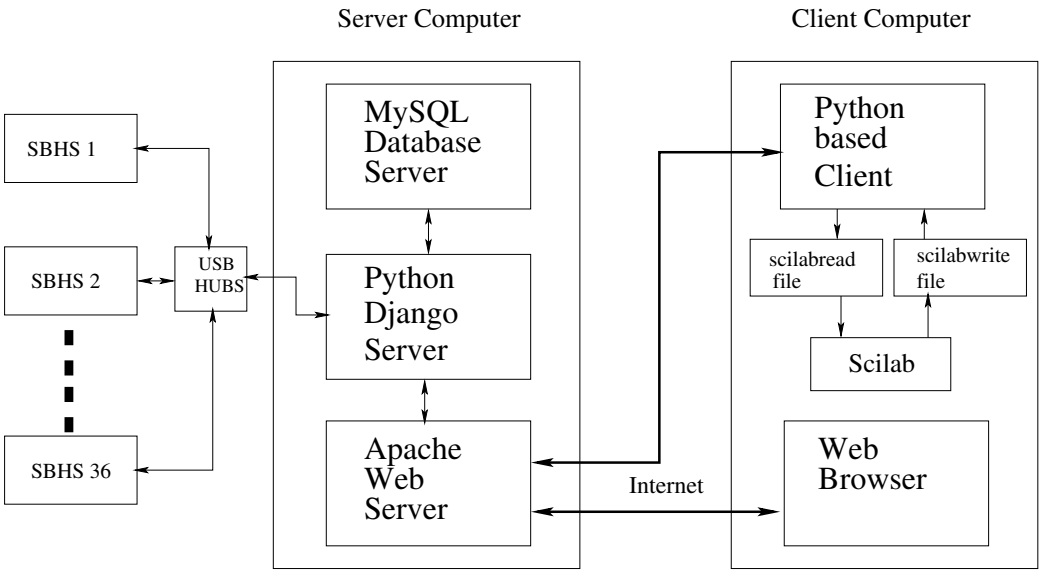

Figure 3.6: Current Architecture of SBHS Virtual Labs

the virtual labs platform. An open loop experiment, step test is used for demonstrating this procedure. The process however remains the same for performing any other experiment explained in this section, unless specified otherwise. Let us first see the required files to be downloaded and installations to be done. Scilab is required to be installed on your computer. Please refer to Section 2.1.1 and Section 2.2 for the procedure to install scilab on Windows and Linux system, respectively.

SBHS scilab code for your OS, under the title `SBHS Virtual Code`, must be downloaded from `http://sbhs.os-hardware.in/downloads`. The code downloaded will be in zip format. After the zip is unpacked, you will see the folder `scilab_codes_windows` or `scilab_codes_linux_32` or scilab_codes_linux_64 depending on which one you download. All these downloads will have scilab experiment folders such as `Step_test`, `Ramp_Test`, `pid_controller` etc. We will be using the Step_test folder. Do not alter the directory structure. If you want to copy or move an experiment outside the directory then make sure you also copy the `common_files` folder. The `common_files` folder must always be one directory outside the experiment folder. Now given that you have scilab installed and working and the required scilab code downloaded, let us see the step-by-step procedure to do a remote experiment.

Figure 3.7: Show Video

3.5.1 Registration, Login and Slot Booking

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

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

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

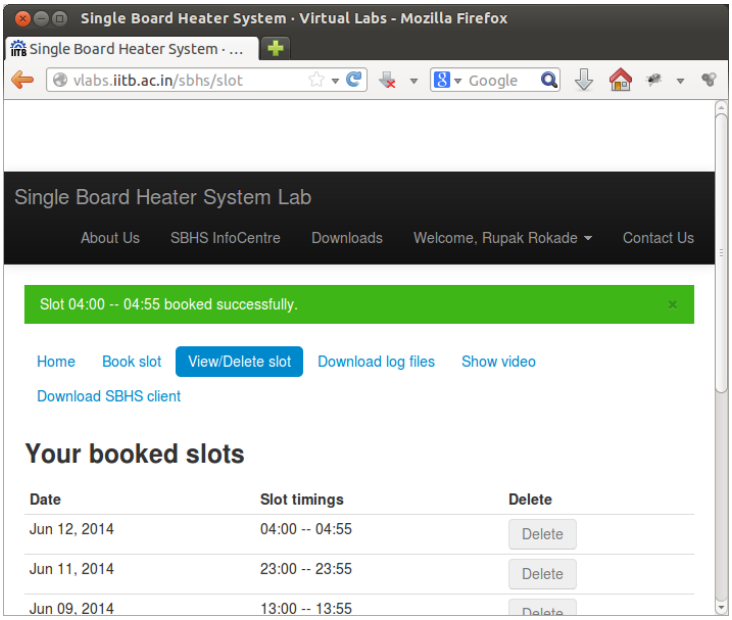

Figure 3.8: Slot booking

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

3.5.2 Configuring proxy settings and executing python based client

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

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

Do not change the contents of `config` file if

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

such as at home OR using a mobile internet

Change the contents of `config` file if

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

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

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

3.5.3 Executing scilab code

Inside the `StepTest` folder, if on a windows system, double click on the file `stepc.sce`. This should automatically launch scilab and also open the `stepc.sce` in the scilab editor. It will also automatically change the scilabs working directory. On a linux system, launch scilab manually. Then change the scilab working directory to the folder `StepTest`. This can be done by clicking on `File` menu and then selecting `change current directory`. Open the file `stepc.sce` using the `Open` option inside `File` menu. The file is shown in Fig. 3.10

The experiment sampling time can be set inside the `stepc.sce` file. You may want to change it to a higher value if your network is slow. The default value of 1 second works fine in most cases. On the menu bar, click on `Execute` option

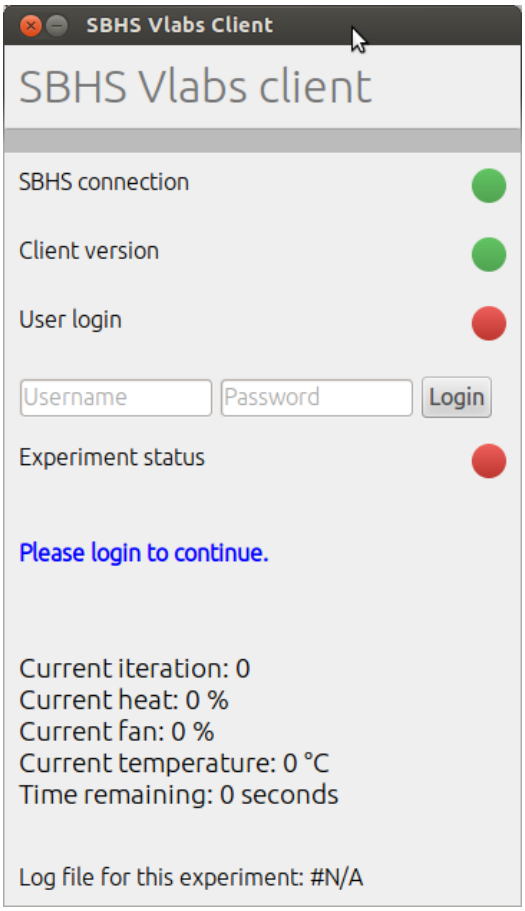

Figure 3.9: Python Client

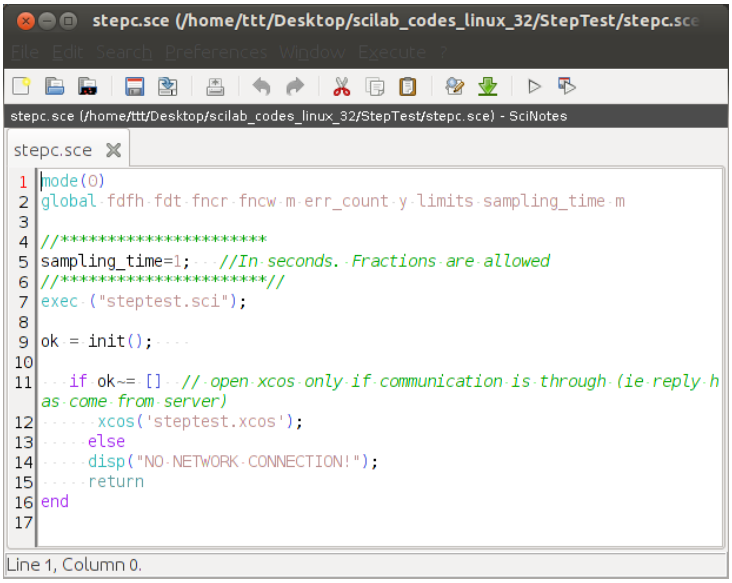

Figure 3.10: stepc.sce file

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

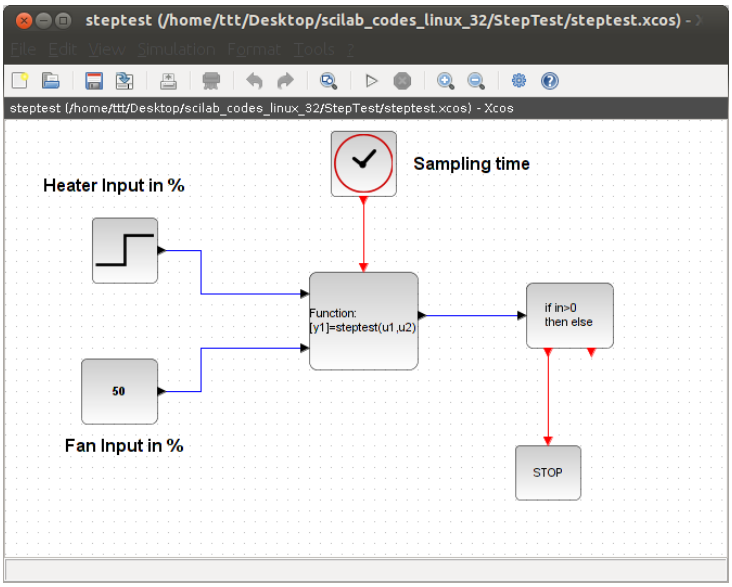

Figure 3.11: Xcos for step test

Chapter 4

PRBS modelling and implementation of pole-placement controller

The aim of this chapter is to do PRBS testing on Single Board Heater System by the application of PRBS signal and also desing a pole-placement controller. The target group is anyone who has basic knowledge of Control Engineering. We have used Scilab with Xcos as an interface for sending and receiving data. This interface is shown in figure 4.1. Heater current and Fan speed are the two inputs to the system. The Heater current is varied with a PRBS signal. A provision is made to set the parameters like PRBS amplitude and offset value. A provision is also made to time the occurance of the PRBS input, using a step block. The value of step time in the step block has to be choosen carefully. Sufficient amount of time should be given to allow the temperature to reach a steady-state before the PRBS signal is applied. In this experiment we are keeping the Fan speed constant at 50%. The temperature profile thus obtained is the output.

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

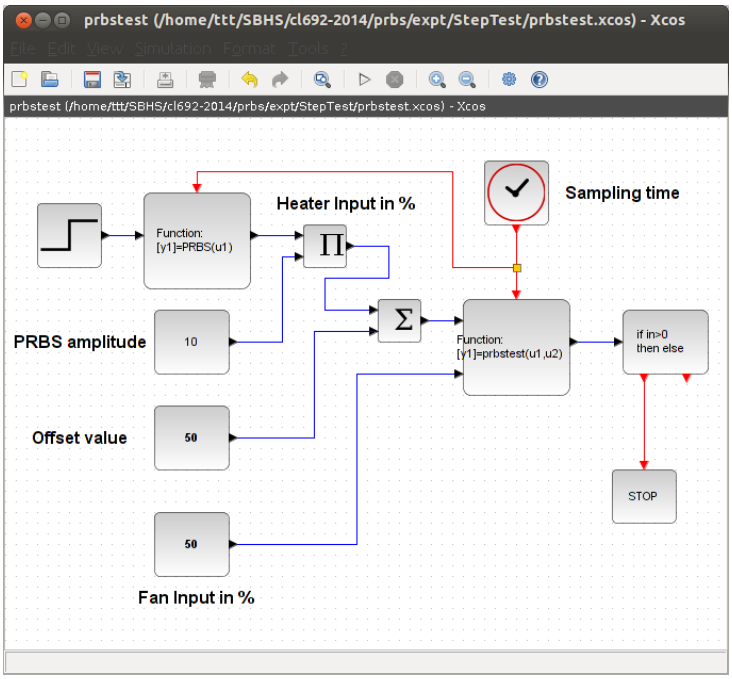

Figure 4.1: Xcos for PRBS testing experiment

4.1 Issues with step testing and alternate approach

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

There is an alternate approach which is widely used in industry. The input given to the heater for identification is not step, but a **pseudo-random binary sequence (PRBS)**. The concept behind PRBS is that the input is perturbated in such a way that the time average of the input is the value at which it is being operated currently. Thus, some positive and some negative steps can be given. This results some positive and some negative changes in the temperature which leads to the time average of the performance of the plant remaining the same. Thus, PRBS testing can be done in a working plant without affecting the plant performance unlike step testing. A typical PRBS and corresponding plant output looks like as shown in figure 4.2

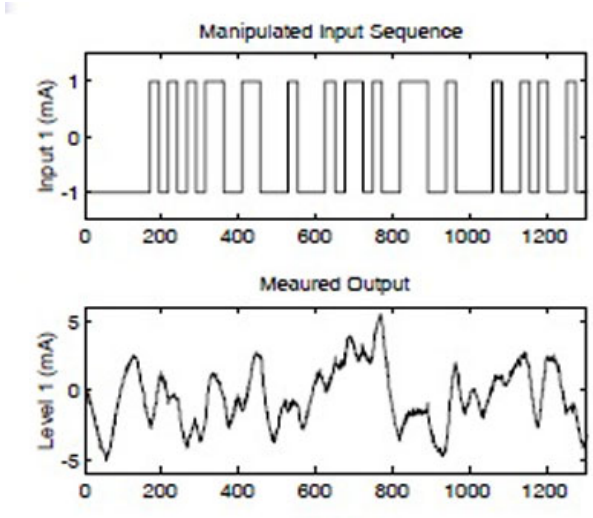

Figure 4.2: PRBS testing input and output *[Image source: CL 686 Advanced Process Control, Spring 2013-14 lecture slides. Prof. S. C. Patwardhan, IIT Bombay]*

4.2 Step by step procedure to do PRBS testing

Similar to Chapter **??** and **??**, in this section we will find the transfer function model of SBHS. But there are two major differences. First difference is that we will give a Pseudo Random Binary Sequence to the heater input of SBHS and second difference is that we will find the discrete time transfer function. A Pseudo Random Binary Sequence is nothing but a signal whose amplitude varies between two limits randomly at any given time. An illustration of the same is given in figure 4.3. A PRBS signal can be easily generated using the *rand*() function in scilab. Scilab code to generate the PRBS signal is given at the end of this chapter. Figure 4.1 shows the xcos diagram for PRBS testing. The PRBS amplitude and offset value to the input can be adjusted using the relevant blocks.

The steps to be followed to conduct PRBS test experiment remains same as explained in section **??** for linux users and section 2.1.2 for windows users. The only difference is that the prbs experiment function is written in `prbstest.sci` file and the xcos is saved in file `prbstest.xcos`. These files should be used for doing prbs experiment. The PRBS response obtained after executing the experiment successfully is shown in figure 4.4

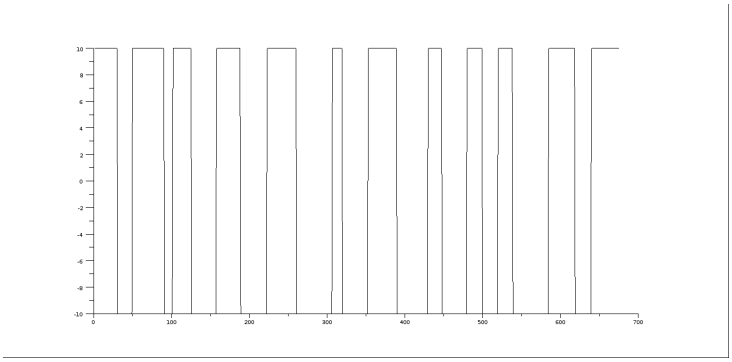

Figure 4.3: A Pseudo Random Binary Sequence

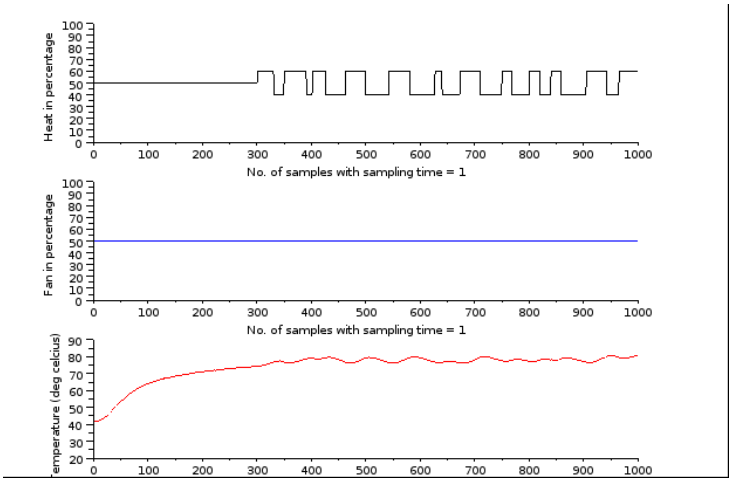

Figure 4.4: PRBS testing response

4.3 Determination of discrete time transfer function

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

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

$$G(z) = \frac{b_1 + b_2 z^{-1}}{1 + a_1 z^{-1} + a_2 z^{-2}} z^{-d} \tag{4.1}$$

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

$$(\hat{b_1}, \hat{b_2}, \hat{a_1}, \hat{a_2}) = \underset{b_1, b_2, a_1, a_2}{argmin} \sum_{i=0}^{N} (y(k) - \hat{y}(k))^2 \tag{4.2}$$

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

$$\hat{y}(k) = -a_1 \hat{y}(k-1) - a_2 \hat{y}(k-2) + b_1 u(k-d) + b_2 u(k-1-d) \tag{4.3}$$

The optimization is performed using the optimization routine "optim" of Scilab. Copy the data file to the folder `prbs_analysis`. Change the scilab working directory to `prbs_analysis` folder. Open the file `optimize.sce` and put the name of the data file (with 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`,

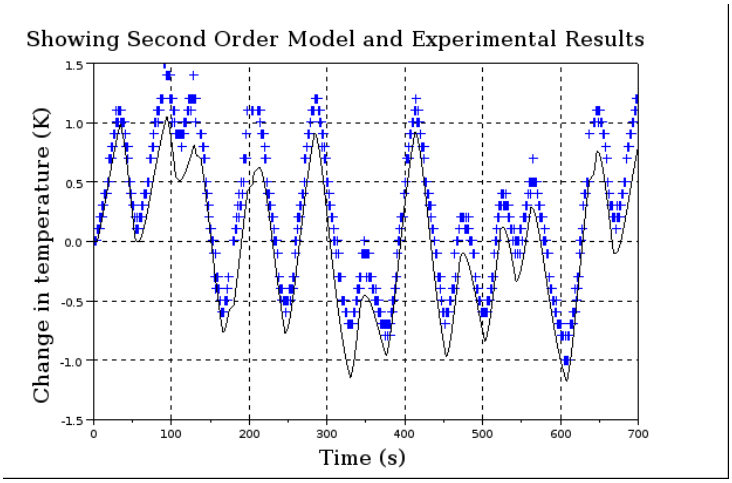

Figure 4.5: PRBS testing response

`costfunction.sci` and `second_order.sci`. This code will give optimized values for a_1, a_2, b_1, b_2 which can be used to define a second order discrete time transfer function as given in equation 4.1. The results generated after executing optimization routine over the data file obtained earlier is shown in figure 4.5 and figure 4.6. The initial values were choosen to be [0.5 0.5 0.5 0.5]. The value of *err* along with the fit obtained has to be used to change the initial values and the value of *delay* The transfer function thus obtained is

$$G(z) = \frac{0.0022266 - 0.0017656z^{-1}}{1 - 1.912876z^{-1} + 0.9162750z^{-2}} z^{-d} \tag{4.4}$$

4.4 Performing PRBS testing on SBHS, virtually

The step by step procedure for conducting an experiment virtually is explained in section 3.5. The required .sce file is `prbstest.sce`. You will find this file in the `prbs` directory under `virtual` folder. The necessary codes are listed in the section 4.6

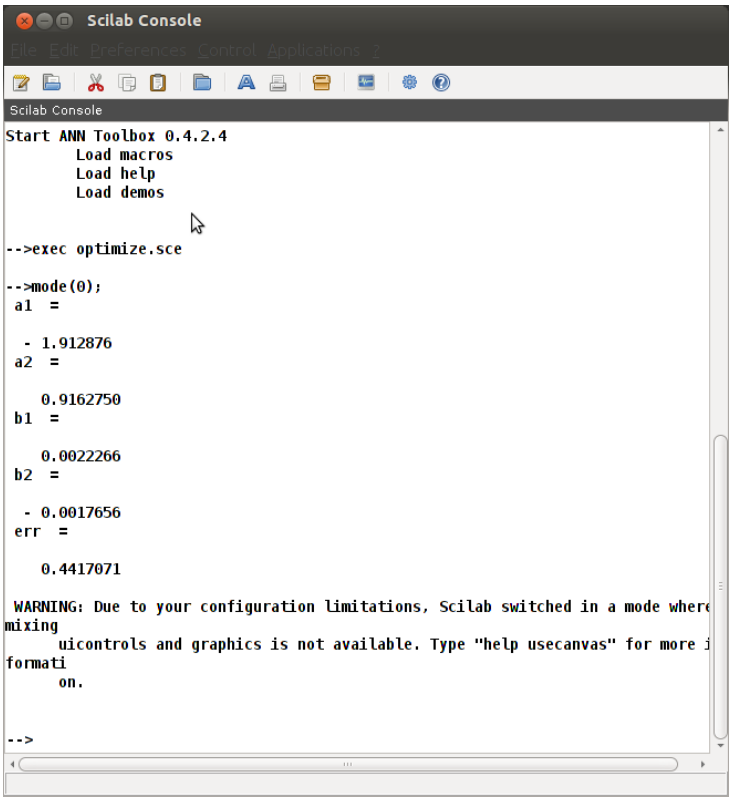

Figure 4.6: PRBS testing response

4.5 Implementing 2DOF pole-placement controller using PRBS model

For deriving the Two degrees of freedom control law, please refer to the chapter **??**

We identified the PRBS model in section 4.6 as shown in equation 4.3. The parameters for the 2-DOF pole-placement controller obtained for this model are shown here

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

These paramenters are computed by the file `twodof_para.sce`.

4.6 Scilab Code

Scilab Code 4.1 `ser_init.sce`

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

Scilab Code 4.2 imc.sci

```
1  mode(0)
2  function [temp] = imc(setpoint,fan,alpha)
3  global temp heat_in fan_in C0 u_old u_new e_old e_new
       e_old_old
4
5  global heatdisp fandisp tempdisp setpointdisp
       sampling_time m name
6
7  e_new = setpoint - temp;
8  b=((1-alpha)/0.01163);
9
10 u_new = u_old + b*(e_new - (0.9723*e_old));
11
12
13 u_old = u_new;
14 e_old = e_new;
15
16
17 heat = u_new;
18
19      temp = comm(heat,fan);
20
21      plotting([heat fan temp setpoint],[0 0 20 0],[100
           100 40 1000])
22
23     m=m+1;
24 endfunction
```

Scilab Code 4.3 imc_virtual.sce

```
1  mode(0);
2  // For scilab 5.1.1 or lower version users, use scicos
       command to open scicos diagrams instead of xcos
3
4  global fdfh fdt fncr fncw m err_count y
5
```

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

Scilab Code 4.4 imc_virtual.sci

```
1  mode(0)
2
3  global fan
4  function [temp,heat,e_new,stop] = imc_virtual(setpoint
       ,fan,alpha)
5  global temp heat et SP u_new u_old u_new e_old e_new
       fdfh fdt fncr fncw m err_count stop
6
7  fncr = 'clientread.sce';            // file to be read -
       temperature
8  fncw = 'clientwrite.sce';          // file to be written
       - heater, fan
9
```

```
10 a = mgetl(fdt,1);
11 b = evstr(a);
12 byte = mtell(fdt);
13 mseek(byte,fdt,'set');
14
15 if a~= []
16     temp = b(1,$); heats = b(1,$-2);
17     fans = b(1,$-1); y = temp;
18
19 e_new = setpoint - temp;
20
21 b=((1-alpha)/0.01163);
22 u_new = u_old + b*(e_new - (0.9723*e_old));
23
24 if u_new> 39
25   u_new = 39;
26 end;
27
28 if u_new< 0
29   u_new = 0;
30 end;
31
32 heat=u_new;
33 u_old = u_new;
34 e_old = e_new;
35
36
37
38 A = [m,m,heat,fan];
39   fdfh = file('open','clientwrite.sce','unknown');
40   file('last', fdfh)
41   write(fdfh,A,'(7(e11.5,1x))');
42   file('close', fdfh);
43   m = m+1;
44
45   else
46     y = 0;
47     err_count = err_count + 1; // counts the no of
```

```
          times network error occurs
48      if err_count > 300
49        disp("NO NETWORK COMMUNICATION!");
50       stop = 1;   // status set for stopping simulation
51   end
52  //  disp(stop)
53    end
54
55  return
56  endfunction
```

Chapter 5

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

5.1 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.

5.1.1 Ojective

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).

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

5.2 Theory

5.2.1 Why a Self Tuning Controller?

The transfer function of SBHS is assumed as

$$\Delta T = \frac{K_p}{\tau_p s + 1}\Delta H + \frac{K_f}{\tau_f s + 1}\Delta F \tag{5.1}$$

ΔT: Temperature Change
ΔF: Fan Input Change
ΔH: Heater Input Change

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 of 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.

5.2.2 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.

5.2.3 Direct synthesis

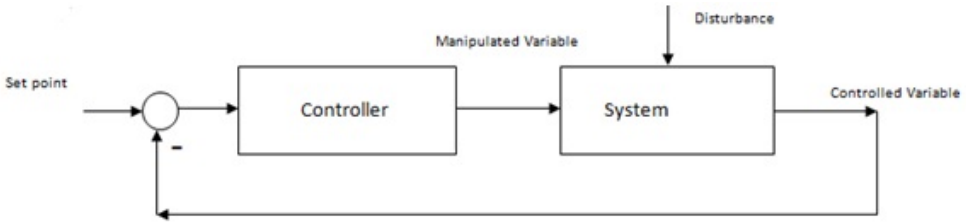

Figure 5.1: Closed Loop Circuit

$$V(s) = \frac{G_c(s)G(s)}{1 + G_c(s)G(s)} \tag{5.2}$$

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

$$V(s) = \frac{1}{(\tau_{cl} s + 1)} \tag{5.3}$$

$$G(s) = \frac{K_p}{(\tau\_ps + 1)} \tag{5.4}$$

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

$$G(c) = K_c(1 + \frac{1}{\tau s}) \tag{5.5}$$

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.

5.3 Ziegler Nichols Tuning

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

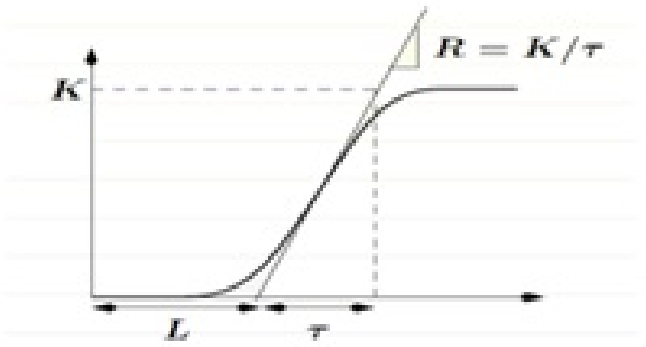

Figure 5.2: Tangent Approach to Ziegler Nichols Tuning

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

Table 5.1: Ziegler Nichols PID Settings

Table 5.1 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
- $\tau_p \approx \tau$

5.4 Step Test Experiments and Parmeter 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.

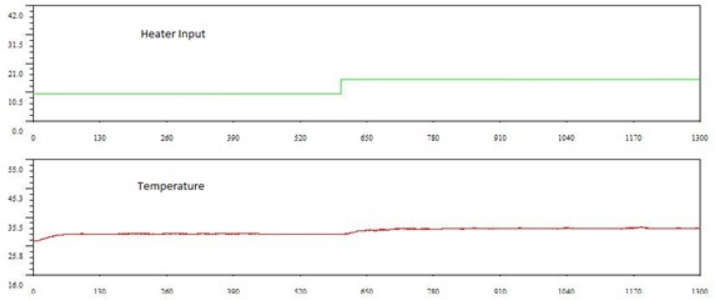

Figure 5.3: Step Responce for Heater Reading 10 to 15

5.4.1 Step Test Experiments

5.4.1.1 Step Change in Heater Reading from 10 to 15

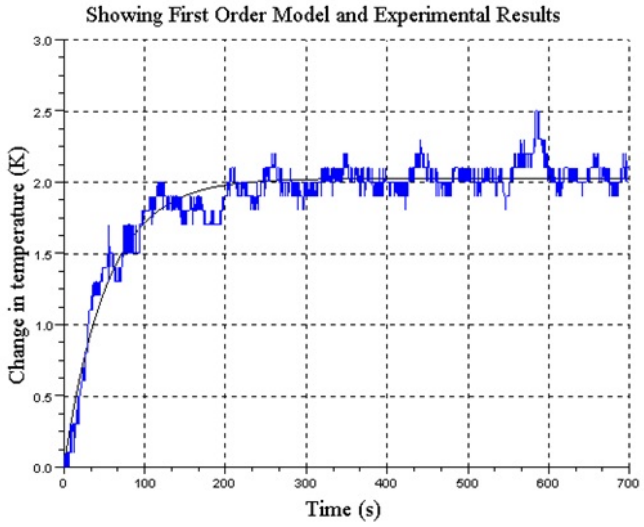

Figure 5.4: Step Response for Heater reading 10 to 15 in terms of Deviation

5.4.1.2 Step Change in Heater Reading from 20 to 25

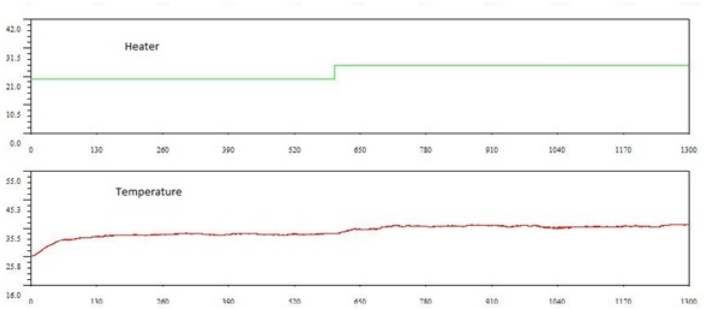

Figure 5.5: Step Responce for Heater Reading 20 to 25

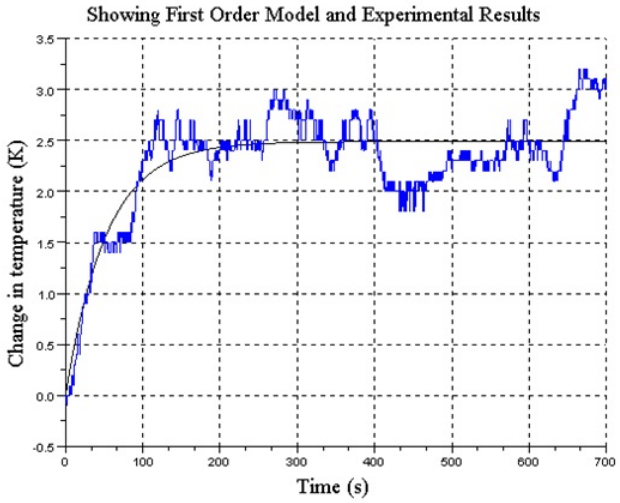

Figure 5.6: Step Response for Heater reading 20 to 25 in terms of Deviation

5.4.1.3 Step Change in Heater Reading from 30 to 35

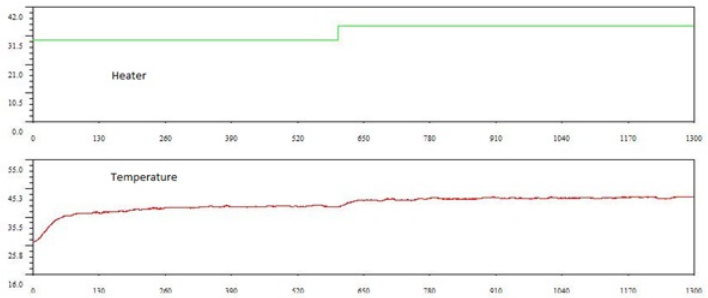

Figure 5.7: Step Responce for Heater Reading 30 to 35

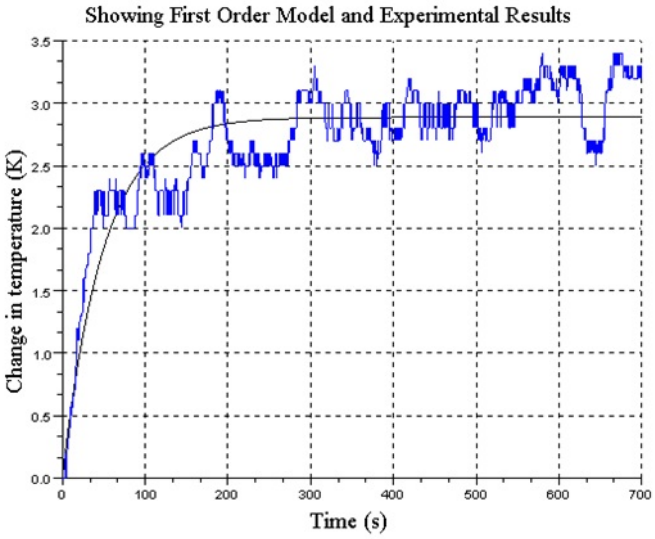

Figure 5.8: Step Response for Heater reading 30 to 35 in terms of Deviation

5.4.1.4 The Open Loop Parameters

Initial Heater Reading	Final Heater Reading	Average Temperature(^{0}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 5.2: Open Loop Parameters

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

5.4.2 Conventional Controller Design

1. PI Controller using Ziegler Nichols Tuning with the results of the first step test experiment:
   - Kc = 19.75
   - τ_i = 18
2. PID Controller using Ziegler Nichols Tuning with the results of the first step test experiment:
   - Kc = 26.327
   - τ_i = 12
   - τ_d = 3
3. PI Controller Using Direct Synthesis on the results of the second step test experiment (τ_{cl} is taken as τ_p/2):
   - Kc = 4.02
   - τ_i = 52.645

5.4.3 Self Tuning Controller Design

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

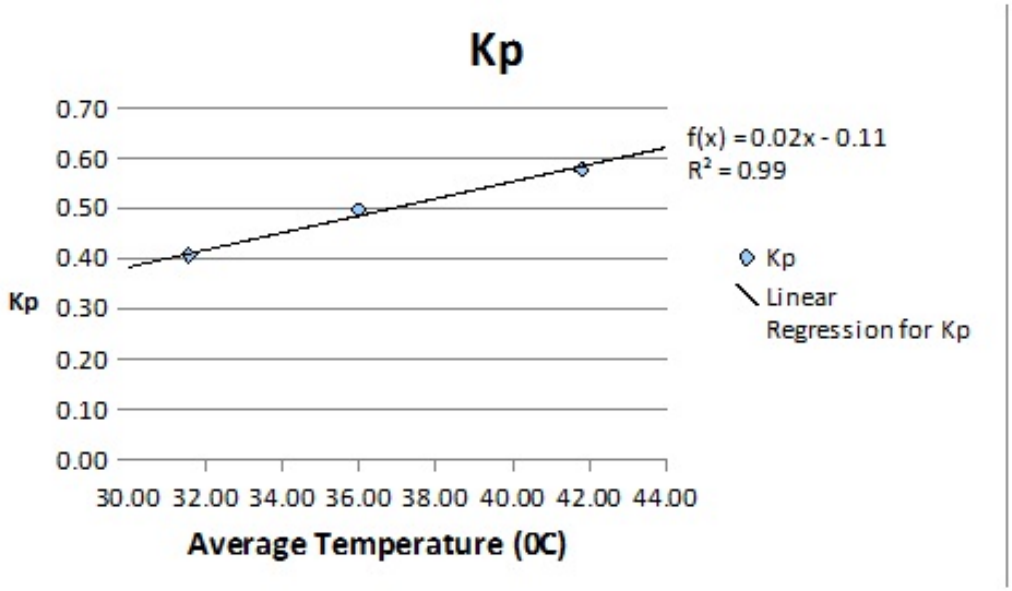

Figure 5.9: Variation of K_p with temperature

1. **PI Controller using Ziegler Nichols Tuning:**

   L = 6
   R = (0.016×T-0.114)/(66.90-0.415×T) where T is the temperature
   Kc = 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 × T-0.114)/(66.90-0.415 × T) where T is the temperature
   K = 1.2(66.90-0.415T)/6(0.016T-0.114)
   = (80.28 - 0.498T)/(0.096T - 0.684)
   $\tau_i = 2 \times 6 = 12$
   $\tau_d = 0.5 \times 6 = 3$

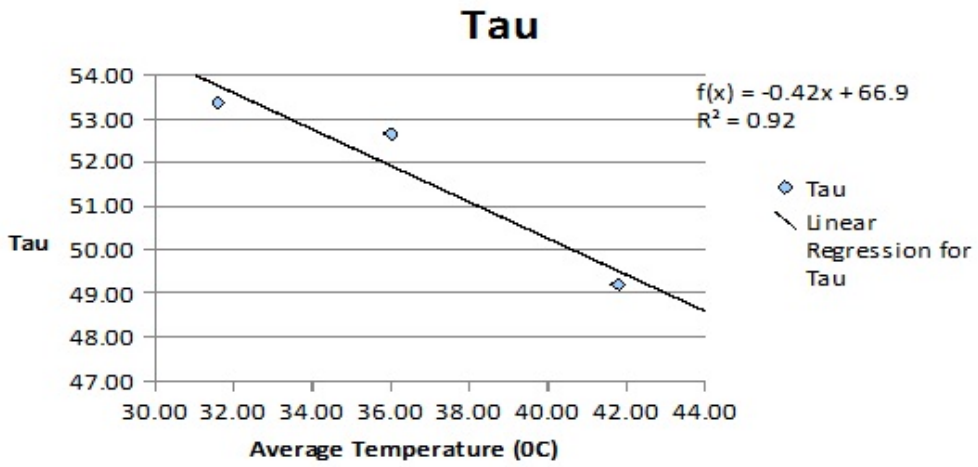

Figure 5.10: Variation of τ_p with temperature

3. **PI Controller using Direct Synthesis (τ_{cl} is taken as τ_p/2):**

   K = 2/(0.016×T-0.114)
   τ_i = (66.90-0.415×T) where T is the temperature

5.5 Implementation

5.5.1 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:

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

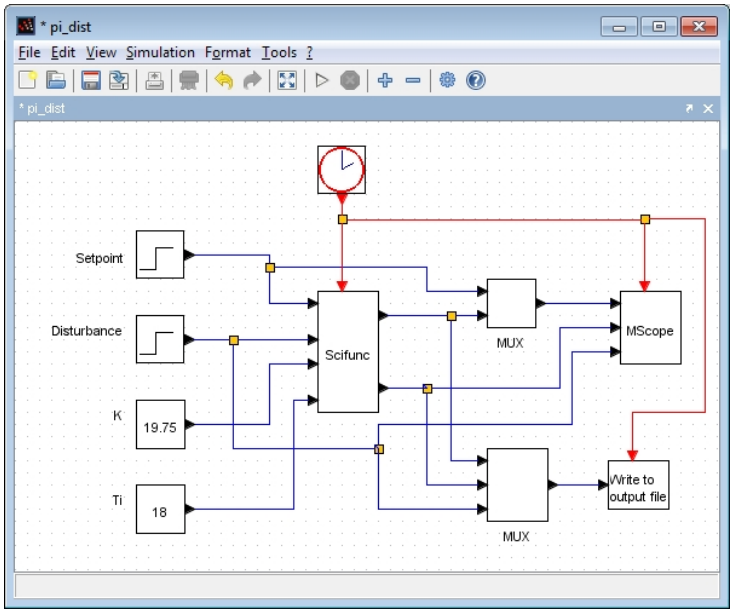

Figure 5.11: Xcos Diagram for PI Controller

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:

$$u(n) = u(n-1) + s_0 e(n) + s_1 e(n-1) \tag{5.6}$$

Where,

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

5.5.2 PID Controller

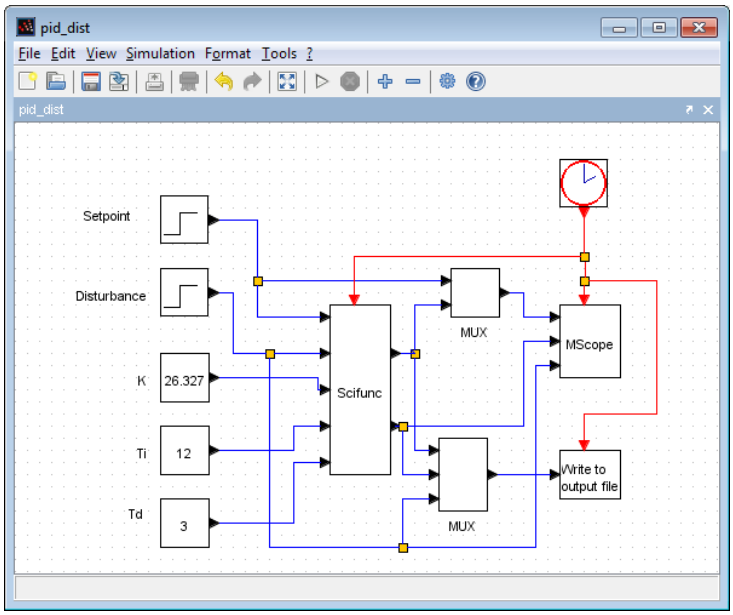

Figure 5.12: Xcos Diagram for PID Controller

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

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

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:

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

The PID Controller is usually written as:

$$u(n) = u(n-1) + s_0 e(n) + s_1 e(n-1) + s_2 e(n-2) \tag{5.7}$$

Where,

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

5.5.3 Self Tuning Controller

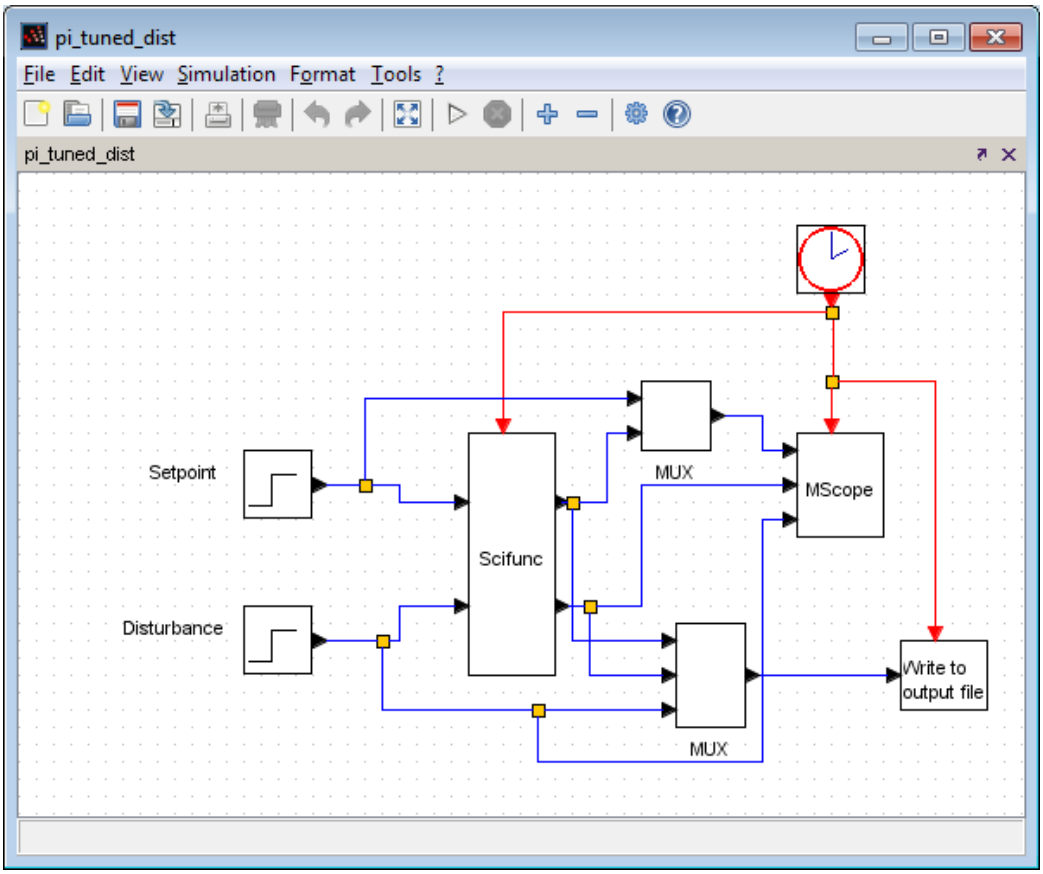

Figure 5.13: Xcos Diagram for Self Tuning Controller

The parameters of the Controller are determined dynamically using the temperature values during every sampling time. For this, the formulae derived in section 5.4.3 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.

5.6 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 5.3 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	32^0C to 37^0C 35^0C to 45^0C	32^0C to 37^0C 35^0C to 45^0C
Ziegler Nichols PI	32^0C to 37^0C 35^0C to 45^0C 40^0C to 45^0C	32^0C to 37^0C 35^0C to 45^0C 35^0C to 45^0C
Ziegler Nichols PID	31^0C to 45^0C 32^0C to 37^0C	32^0C to 46^0C 32^0C to 37^0C

Table 5.3: Set Point Changes in experiments conducted for Set Point Tracking

5.6.1 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.

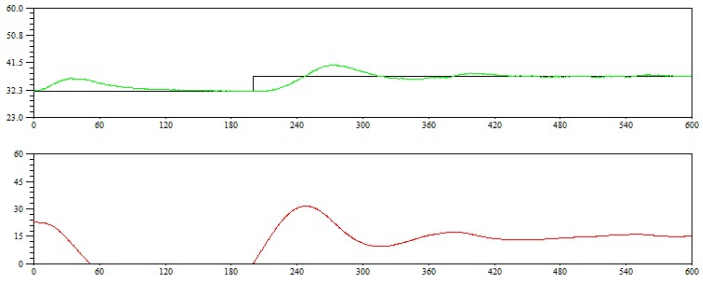

Figure 5.14: Result for Self Tuning Controller designed using Direct Synthesis for Set Point going from 32^0C to 37^0C

Although there is a small overshoot, the controller is able to make the actual temperature follow the set point temperature quiet closely. Looking at higher values of set point changes, the result for set point change going from 35^0C to 45^0C is shown.

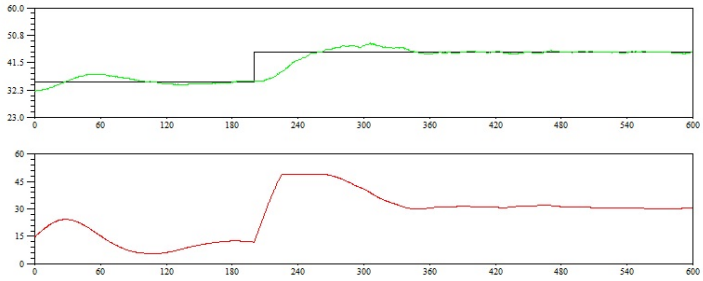

Figure 5.15: Result for Self Tuning Controller designed using Direct Synthesis for Set Point going from 35^0C to 45^0C

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.

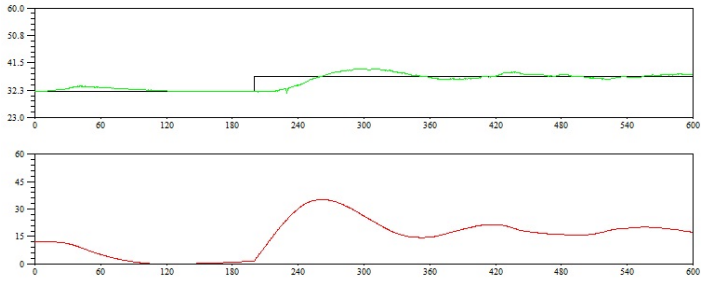

Figure 5.16: Result for Conventional Controller designed using Direct Synthesis for Set Point going from 32^0C to 37^0C

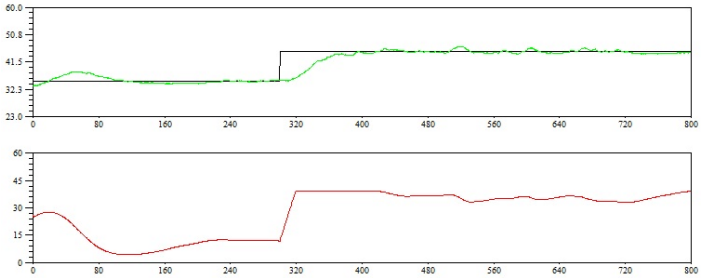

Figure 5.17: Result for Conventional Controller designed using Direct Synthesis for Set Point going from 35^0C to 45^0C

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

5.6.2 PI Controller using Ziegler Nichols Tuning

The results of the 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.

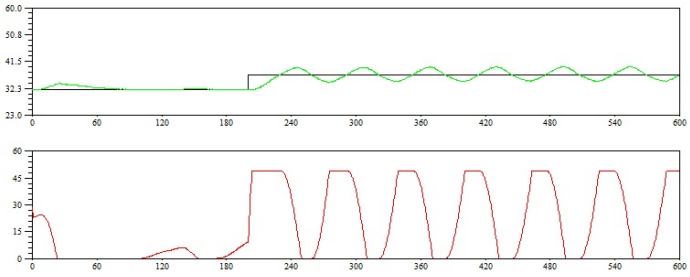

Figure 5.18: Result for Self Tuning Controller designed using Ziegler Nichols Tuning for Set Point going from 32^0C to 37^0C

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

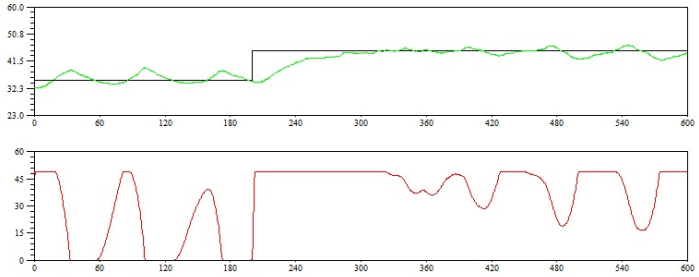

Figure 5.19: Result for Self Tuning Controller designed using Ziegler Nichols Tuning for Set Point going from 35^0C to 45^0C

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.

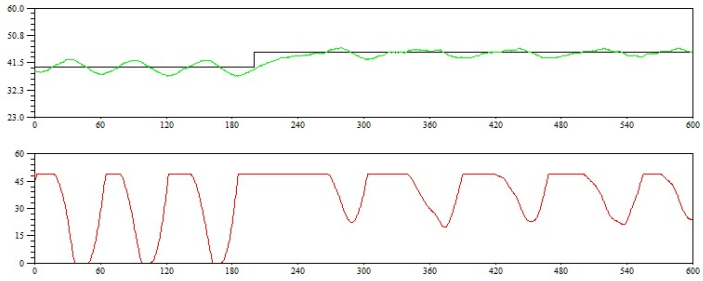

Figure 5.20: Result for Self Tuning Controller designed using Ziegler Nichols Tuning for Set Point going from 40^0C to 45^0C

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

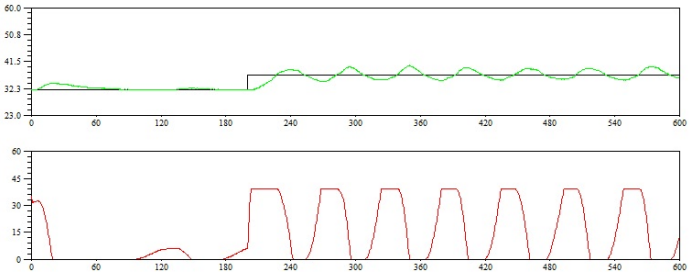

Figure 5.21: Result for Conventional Controller designed using Ziegler Nichols Tuning for Set Point going from 32^0C to 37^0C

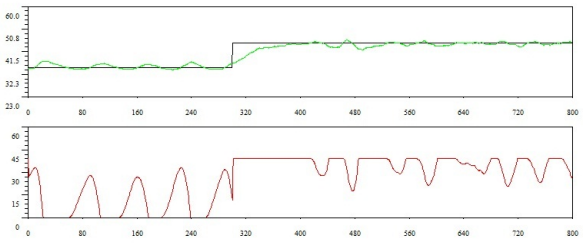

Figure 5.22: Result for Conventional Controller designed using Ziegler Nichols Tuning for Set Point going from 35^0C to 45^0C

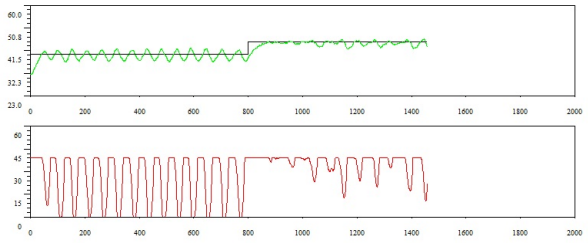

Figure 5.23: Result for Conventional Controller designed using Ziegler Nichols Tuning for Set Point going from 40^0C to 45^0C

For set point change from 40^0C to 45^0C, the self tuning controller showed small oscillations, but the conventional controller shows bigger oscillations.

5.6.3 PID Controller using Ziegler Nichols Tuning

The results of the 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.

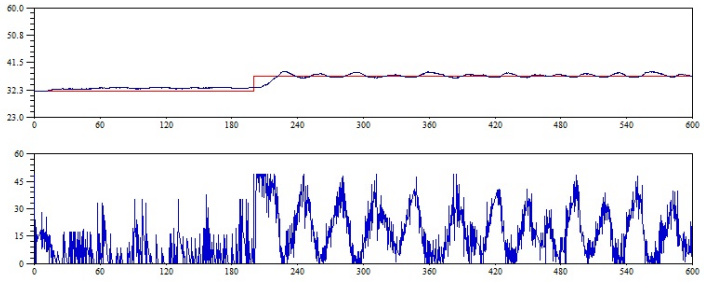

Figure 5.24: Result for Self Tuning PID Controller designed using Ziegler Nichols Tuning for Set Point going from 32^0C to 37^0C

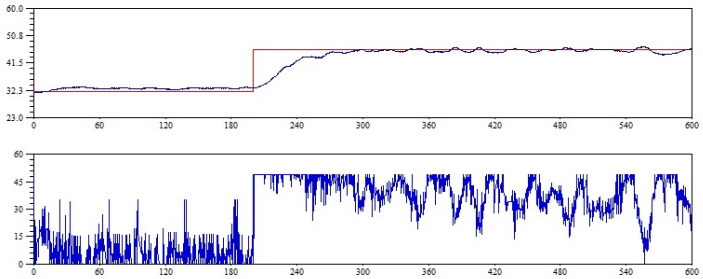

Figure 5.25: Result for Self Tuning PID Controller designed using Ziegler Nichols Tuning for Set Point going from 32^0C to 46^0C

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

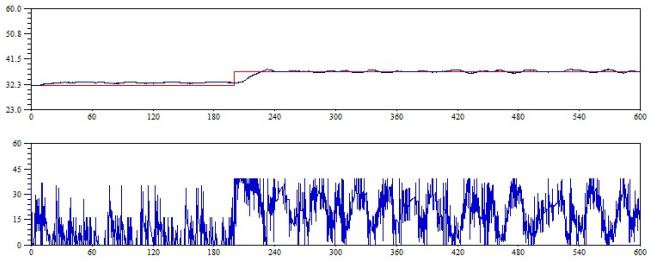

Figure 5.26: Result for Conventional PID Controller designed using Ziegler Nichols Tuning for Set Point going from 32^0C to 37^0C

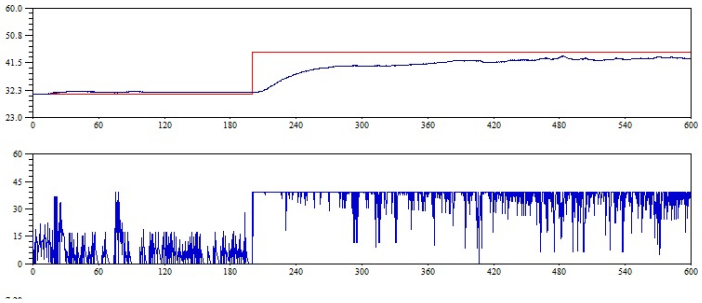

Figure 5.27: Result for Conventional PID Controller designed using Ziegler Nichols Tuning for Set Point going from 31^0C to 45^0C

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

5.6.4 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 5.6.3.

5.7 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 vraibles 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 5.4 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 5.4: Fan Input Changes in experiments conducted for Disturbance Rejection

5.7.1 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.

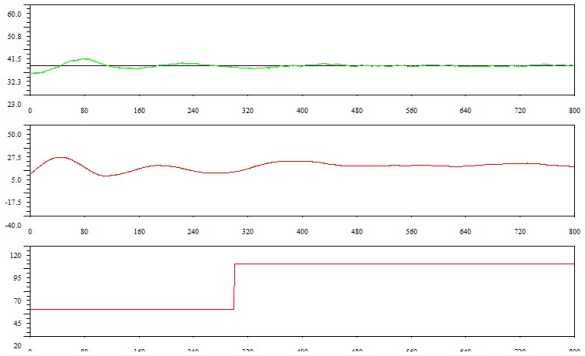

Figure 5.28: Results for Fan Input Change from 50 to 100 for Self Tuning PI Controller designed using Direct Synthesis

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.

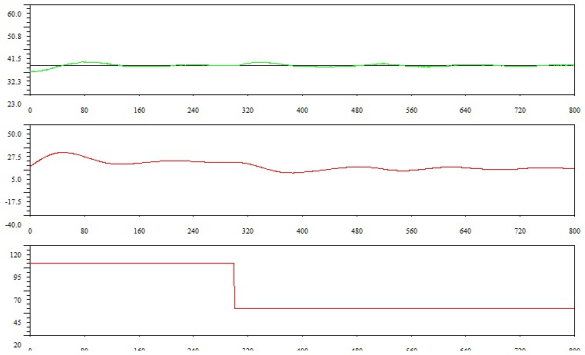

Figure 5.29: Results for Fan Input Change from 100 to 50 for Self Tuning PI Controller designed using Direct Synthesis

In this figure also, the temperature clearly increses 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 syntheis has successfully rejected the disturbance.

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

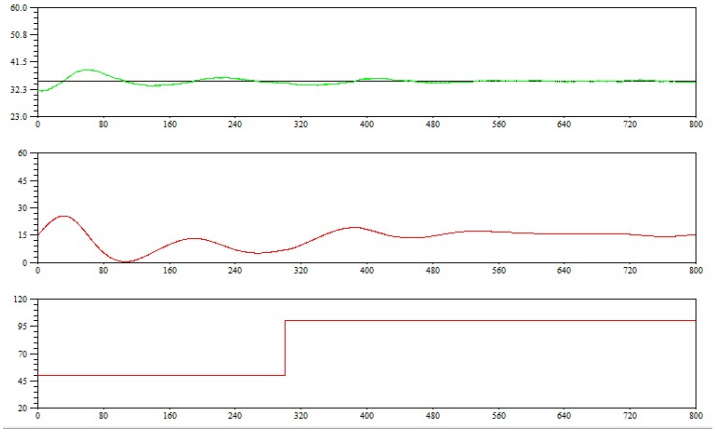

Figure 5.30: Results for the Fan input change from 50 to 100 to Conventional PI Controller designed using Direct Synthesis

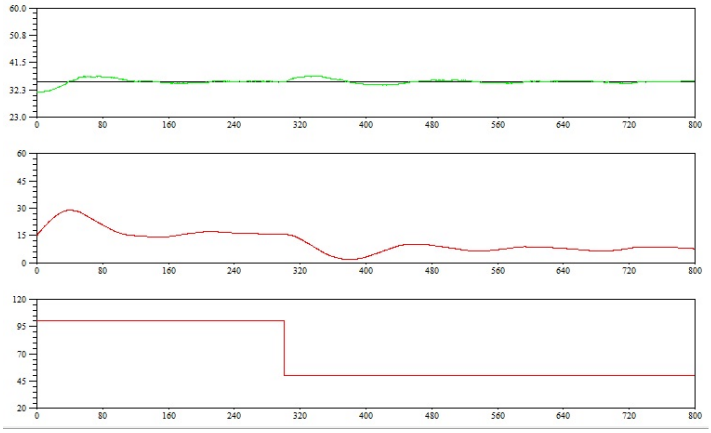

Figure 5.31: Results for the Fan input change from 100 to 50 to Conventional PI Controller designed using Direct Synthesis

5.7.2 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.

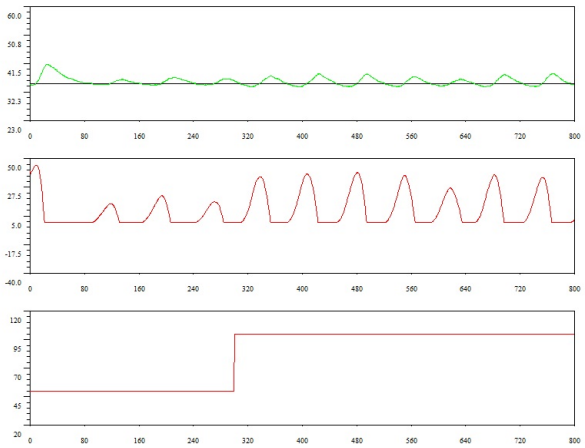

Figure 5.32: Results for Fan Input change from 50 to 100 given to Self Tuning PI Controller designed using Ziegler Nichols Method

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 keept the temperature close to the set point.
From the above two results, it is clear that the self tuning controller designed with direct syntheis has successfully rejected the disturbance.

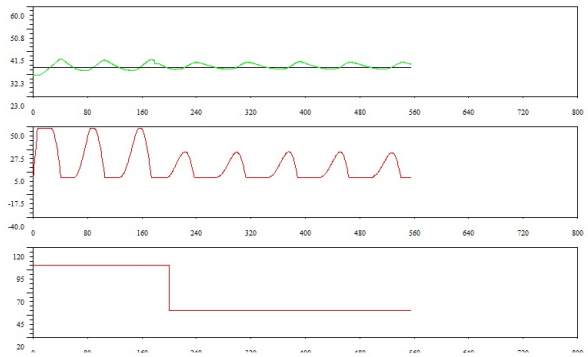

Figure 5.33: Results for Fan Input change from 100 to 50 given to Self Tuning PI Controller designed using Ziegler Nichols Method

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

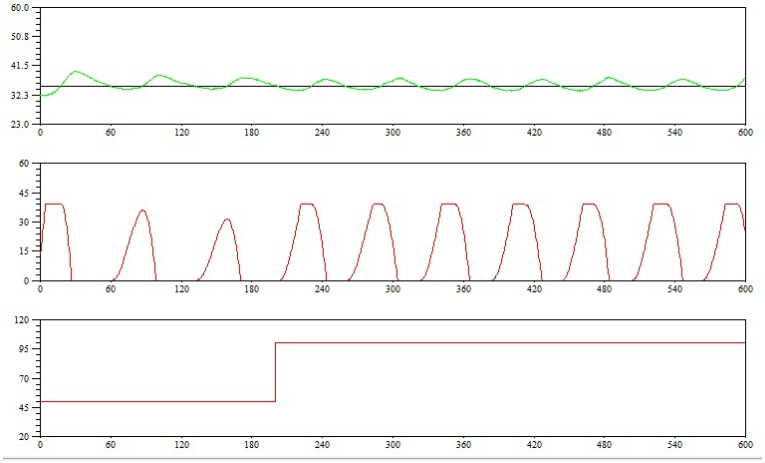

Figure 5.34: Results for the Fan input change from 50 to 100 to Conventional PI Controller designed using Ziegler Nichols Tuning

5.7.3 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.

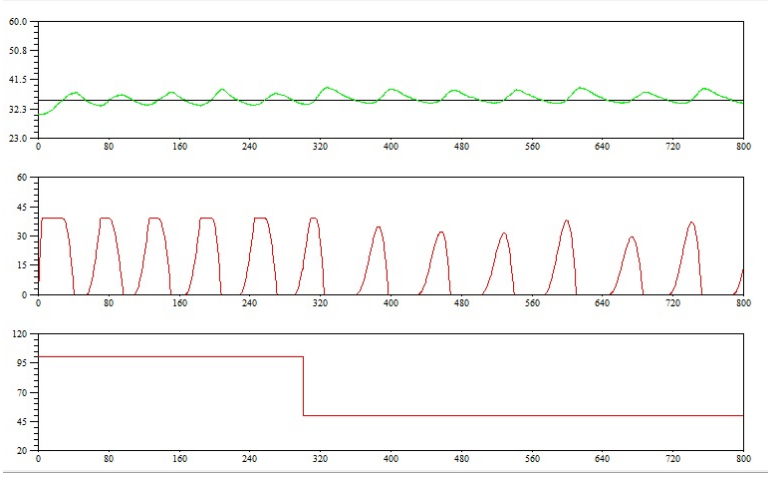

Figure 5.35: Results for the Fan input change from 100 to 50 to Conventional PI Controller designed using Ziegler Nichols Tuning

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

In this figure also, the temperature clearly increses 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.

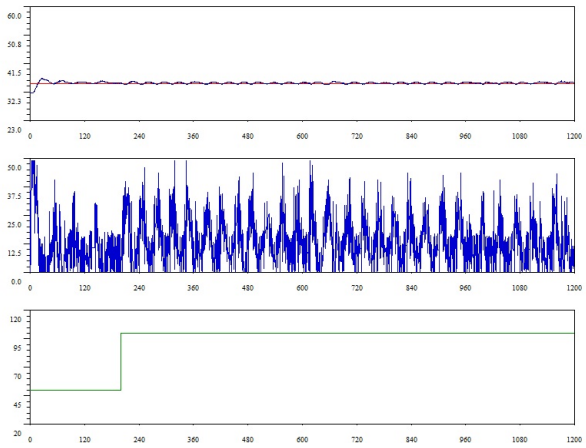

Figure 5.36: Results for Fan Input change from 50 to 100 given to Self Tuning PID Controller designed using Ziegler Nichols Method

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

5.7.4 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.

5.7.5 Implementing Self Tuning controller on SBHS, virtually

The step by step procedure for conducting an experiment virtually is explained in section 3.5. 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 5.7.10 and 5.7.12

5.7.6 Serial Communication

Scilab Code 5.1 `ser_init.sce`

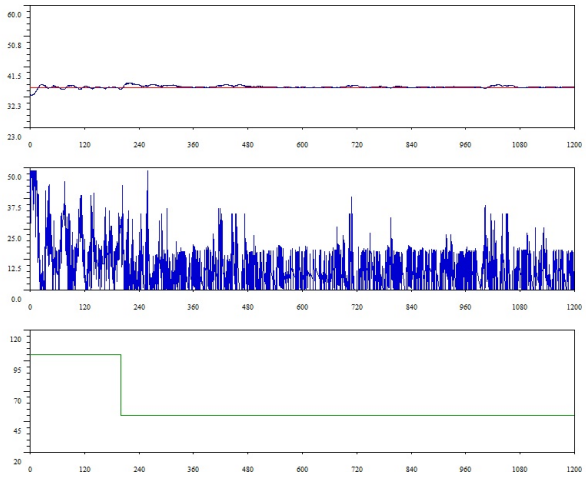

Figure 5.37: Results for Fan Input change from 100 to 50 given to Self Tuning PID Controller designed using Ziegler Nichols Method

```
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")
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
```

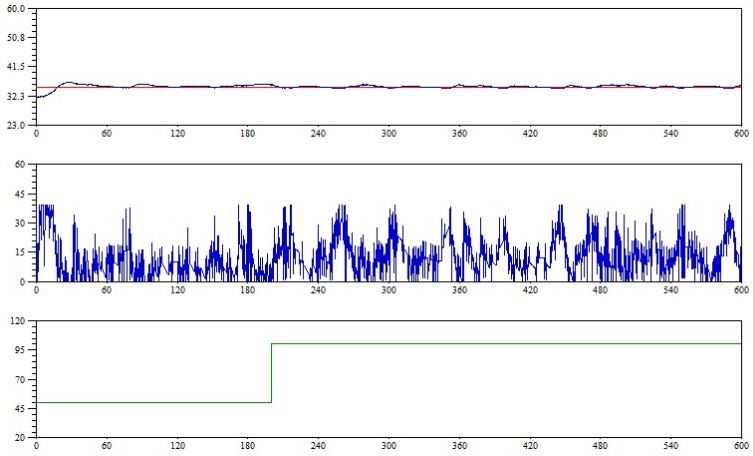

Figure 5.38: Results for the Fan input change from 50 to 100 to Conventional PID Controller designed using Ziegler Nichols Tuning

5.7.7 Conventional Controller, local

5.7.8 Fan Disturbance in PI Controller

Scilab Code 5.2 `pi_bda_dist.sci`

```
1  mode(0)
2  // global temp heat fan sampling_time m heatdisp
       fandisp tempdisp x name
3  function temp = pi_bda_dist(setpoint,fan,K,Ti)
4  global heatdisp fandisp tempdisp setpointdisp
       sampling_time m name temp heat_in fan_in C0 u_old
       u_new e_old e_new
5
6  Ts = sampling_time;
7  e_new = setpoint - temp;
8
9  S0=K*(1+((Ts/Ti)));
10 S1=-K;
11 u_new = u_old + S0*e_new + S1*e_old;
12
13
```

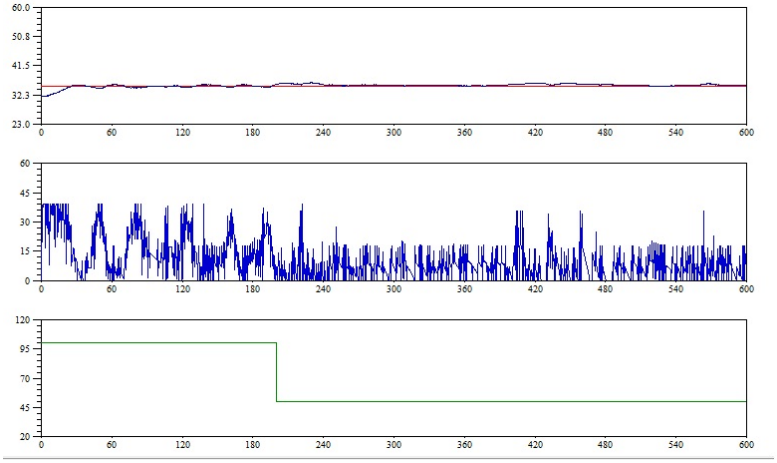

Figure 5.39: Results for the Fan input change from 100 to 50 to Conventional PID Controller designed using Ziegler Nichols Tuning

```
14 u_old = u_new;
15 e_old = e_new;
16
17 heat = u_new;
18     temp = comm(heat,fan);
19
20     plotting([heat fan temp setpoint],[0 0 20 0],[100
           100 40 1000])
21
22     m=m+1;
23 endfunction
```

5.7.8.1 Set Point Change in PI Controller

Scilab Code 5.3 `pi_bda.sci`

```
1 mode(0)
2 function temp = pi_bda(setpoint,fan,K,Ti)
3 global heatdisp fandisp tempdisp setpointdisp
    sampling_time m name temp heat_in fan_in C0 u_old
    u_new e_old e_new
```

```
4
5  Ts = sampling_time;
6  e_new = setpoint - temp;
7
8  S0=K*(1+((Ts/Ti)));
9  S1=-K;
10 u_new = u_old+(S0*e_new)+(S1*e_old);
11
12
13 u_old = u_new;
14 e_old = e_new;
15
16 heat = u_new;
17      temp = comm(heat,fan);
18
19      plotting([heat fan temp setpoint],[0 0 20 0],[100
           100 40 1000])
20
21     m=m+1;
22 endfunction
```

5.7.8.2 Fan Disturbance to PID Controller

Scilab Code 5.4 `pid_bda_dist.sci`

```
1  mode(0)
2  function [temp] = pid_bda_dist(setpoint,fan,K,Ti,Td)
3  global temp heat_in fan_in C0 u_old u_new e_old e_new
      e_old_old
4
5  global heatdisp fandisp tempdisp setpointdisp
      sampling_time m name
6
7  e_new = setpoint - temp;
8
9  Ts=sampling_time;
10
11 S0=K*(1+(Ts/Ti)+(Td/Ts));
```

```
S1=K*(-1-((2*Td)/Ts));
S2=K*(Td/Ts);

u_new = u_old + S0*e_new + S1*e_old + S2*e_old_old;

u_old = u_new;
e_old_old = e_old;
e_old = e_new;


heat = u_new;
    temp = comm(heat,fan);

    plotting([heat fan temp setpoint],[0 0 20 0],[100
       100 40 1000])

    m=m+1;
endfunction
```

5.7.8.3 Set Point Change in PID Controller

Scilab Code 5.5 `pid_bda.sci`

```
mode(0)
function [temp] = pid_bda(setpoint,fan,K,Ti,Td)
global temp heat_in fan_in C0 u_old u_new e_old e_new
   e_old_old

global heatdisp fandisp tempdisp setpointdisp
   sampling_time m name

e_new = setpoint - temp;

Ts=sampling_time;

S0=K*(1+(Ts/Ti)+(Td/Ts));
S1=K*(-1-((2*Td)/Ts));
S2=K*(Td/Ts);
```

```
14
15 u_new = u_old + S0*e_new + S1*e_old + S2*e_old_old;
16
17 u_old = u_new;
18 e_old_old = e_old;
19 e_old = e_new;
20
21
22 heat = u_new;
23      temp = comm(heat,fan);
24
25      plotting([heat fan temp setpoint],[0 0 20 0],[100
          100 40 1000])
26
27     m=m+1;
28 endfunction
```

5.7.9 Self Tuning Controller, local

5.7.9.1 Fan Discturbance to PI Controller

Scilab Code 5.6 pi_bda_tuned_dist.sci

```
1 mode(0)
2 // global temp heat fan sampling_time m heatdisp
      fandisp tempdisp x name
3 function temp = pi_bda_tuned_dist(setpoint,fan)
4 global heatdisp fandisp tempdisp setpointdisp
     sampling_time m name temp heat_in fan_in C0 u_old
     u_new e_old e_new
5
6 Ts = sampling_time;
7 //L = 6;
8 //R = (0.016*temp -0.114)/(66.90 -0.415*temp);
9 //K = 0.9/(R*L);
10 //Ti = 3*L;
11
12       //the above is the ziegler nichols part
```

```
13
14
15 K = 2/(0.016*temp-0.114);
16 Ti = (66.90-0.415*temp);
17
18 // the above is the direct synthesis part
19
20 e_new = setpoint - temp;
21 S0=K*(1+((Ts/Ti)));
22 S1=-K;
23 u_new = u_old+(S0*e_new)+(S1*e_old);
24
25
26 u_old = u_new;
27 e_old = e_new;
28
29 heat = u_new;
30
31      temp = comm(heat,fan);
32
33      plotting([heat fan temp setpoint],[0 0 20 0],[100
           100 40 1000])
34
35     m=m+1;
36 endfunction
```

5.7.9.2 Set Point Change to PI Controller

Scilab Code 5.7 pi_bda_tuned.sci

```
1 mode(0)
2 function temp = pi_bda_tuned(setpoint,fan)
3 global heatdisp fandisp tempdisp setpointdisp
     sampling_time m name temp heat_in fan_in C0 u_old
     u_new e_old e_new
4
5 Ts =sampling_time;
6
```

```
7  //L = 6;
8  //R = (0.016*temp -0.114)/(66.90 -0.415*temp);
9  //K = 0.9/(R*L);
10 //Ti = 3*L;
11
12
13 //The above is the Ziegler nichols part.
14
15 K = 2/(0.016*temp-0.114);
16 Ti = (66.90-0.415*temp);
17 //The above is the direct synthesis part
18
19 e_new = setpoint - temp;
20
21 S0=K*(1+((Ts/Ti)));
22 S1=-K;
23 u_new = u_old+(S0*e_new)+(S1*e_old);
24
25 u_old = u_new;
26 e_old = e_new;
27
28 heat = u_new;
29      temp = comm(heat,fan);
30
31      plotting([heat fan temp setpoint],[0 0 20 0],[100
           100 40 1000])
32
33     m=m+1;
34 endfunction
```

5.7.9.3 Fan Disturbance to PID Controller

Scilab Code 5.8 `pid_bda_tuned_dist.sci`

```
1 mode(0)
2
3 function [temp] = pid_bda_tuned_dist(setpoint,fan)
```

```
4  global temp heat_in fan_in C0 u_old u_new e_old e_new
       e_old_old
5
6  global heatdisp fandisp tempdisp setpointdisp
       sampling_time m name
7
8  L = 6;
9  R = (0.016*temp-0.114)/(66.90-0.415*temp);
10 K = 1.2/(R*L)
11 Ti = 2*L;
12 Td = 0.5*L;
13
14
15 e_new = setpoint - temp;
16
17 Ts=sampling_time;
18
19 S0=K*(1+(Ts/Ti)+(Td/Ts));
20 S1=K*(-1-((2*Td)/Ts));
21 S2=K*(Td/Ts);
22
23 u_new = u_old + S0*e_new + S1*e_old + S2*e_old_old;
24
25 u_old = u_new;
26 e_old_old = e_old;
27 e_old = e_new;
28
29
30 heat = u_new;
31      temp = comm(heat,fan);
32
33      plotting([heat fan temp setpoint],[0 0 20 0],[100
           100 40 1000])
34
35     m=m+1;
36 endfunction
```

5.7.9.4 Set Point Change to PID Controller

Scilab Code 5.9 pid_bda_tuned.sci

```
1  mode(0)
2  function [temp] = pid_bda_tuned(setpoint,fan)
3  global temp heat_in fan_in C0 u_old u_new e_old e_new
       e_old_old
4
5  global heatdisp fandisp tempdisp setpointdisp
       sampling_time m name
6
7  L = 6;
8  R = (0.016*temp-0.114)/(66.90-0.415*temp);
9  K = 1.2/(R*L)
10 Ti = 2*L;
11 Td = 0.5*L;
12
13
14 e_new = setpoint - temp;
15
16 Ts=sampling_time;
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
24 u_old = u_new;
25 e_old_old = e_old;
26 e_old = e_new;
27
28
29 heat = u_new;
30      temp = comm(heat,fan);
31
```

```
32      plotting([heat fan temp setpoint],[0 0 20 0],[100
          100 40 1000])
33
34      m=m+1;
35  endfunction
```

5.7.10 Conventional Controller, virtual

5.7.11 Fan Disturbance in PI Controller

Scilab Code 5.10 `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
       at constant level
5  //Range of Fan input :20 to 252
6  //Temperature is read
7
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 fncr fncw m err_count stop
11
12
13 fncr = 'clientread.sce';              //file to be read -
       temperature
14 fncw = '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
59        disp("NO NETWORK COMMUNICATION!");
60       stop = 1;  // status set for stopping simulation
61       end
62    end
63
64  return
65  endfunction
```

5.7.11.1 Set Point Change in PI Controller

Scilab Code 5.11 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 fncr fncw m err_count stop
10
11
12 fncr = '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
```

5.7.11.2 Fan Disturbance to PID Controller

Scilab Code 5.12 `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 fncr 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 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
```

5.7.11.3 Set Point Change in PID Controller

Scilab Code 5.13 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 fncr 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
57        disp("NO NETWORK COMMUNICATION!");
58       stop = 1;  // status set for stopping simulation
59       end
60    end
61
```

```
62 return
63 endfunction
```

5.7.12 Self Tuning Controller, local

5.7.12.1 Fan Discturbance to PI Controller

Scilab Code 5.14 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
       at constant level
5  //Range of Fan input :20 to 252
6  //Temperature is read
7
8  function [temp,heat,stop] = pi_bda_tuned_dist(setpoint
       ,disturbance)
9  global temp heat fan C0 u_old u_new e_old e_new fdfh
       fdt fncr fncw m err_count stop
10
11 K = 2/(0.016*temp-0.114);
12 Ti = (66.90-0.415*temp);
13
14
15 fncr = 'clientread.sce';              //file to be read -
       temperature
16 fncw = 'clientwrite.sce';             //file to be written
       - heater, fan
17
18 a = mgetl(fdt,1);
19 b = evstr(a);
20 byte = mtell(fdt);
21 mseek(byte,fdt,'set');
22
23 if a~= []
```

```
24      temp = b(1,$); heats = b(1,$-2);
25      fans = b(1,$-1); y = temp;
26
27  e_new = setpoint - temp;
28
29  Ts=1;
30  S0=K*(1+((Ts/Ti)));
31  S1=-K;
32  u_new = u_old+(S0*e_new)+(S1*e_old);
33
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
```

5.7.12.2 Set Point Change to PI Controller

Scilab Code 5.15 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 fncr 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 fncr = '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
```

5.7.12.3 Fan Disturbance to PID Controller

Scilab Code 5.16 `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 fncr fncw 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 fncr = 'clientread.sce';              // file to be read -
       temperature
14 fncw = '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
```

5.7.12.4 Set Point Change to PID Controller

Scilab Code 5.17 `pid_bda_tuned.sci`

```
1 mode(0);
2 //PID using Backward difference approximation for I
      and D
3
4 function [temp,heat,et,stop] = pid_bda_tuned(setpoint,
      disturbance)
5 global temp heat fan et SP CO eti u_old u_new e_old
      e_new e_old_old fdfh fdt fncr fncw 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 //Kc and tau_i calculated
11 Td = 0.5*L;
12
13
14 fncr = 'clientread.sce';              // file to be read -
       temperature
15 fncw = 'clientwrite.sce';             // file to be written
       - heater, fan
16
17 a = mgetl(fdt,1);
18 b = evstr(a);
19 byte = mtell(fdt);
20 mseek(byte,fdt,'set');
21
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
```

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.