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

Vikas Gayasen

November 29, 2010

Contents

List of Figures

List of Tables

Chapter 1

Introduction

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.

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)

1.2 Apparatus

1. Fig 1.1 shows the single board heater system on which this experiment will be performed.

2. The setup consists of a heater assembly, fan, temperature sensor, microcontroller and associated circuitry.

3. Heater assembly consists of an iron plate placed at a distance of about 3.5 mm from the nichrome coil.

4. A 12 V computer fan positioned below this heater assembly is meant for cooling the assembly.

5. The temperature sensed by the temperature sensor, AD 590, after suitable processing, is fed to the microcontroller.

6. The microcontroller ATmega16 is the heart of the setup. It provides an interface between the process and the computer.

7. The LCD display mounted above the microcontroller displays the heated plate temperature, heater and fan inputs and also the commands communicated via serial port.

8. The setup is powered by 12 V, 8 A SMPS.

Figure 1.1: Single Board Heater System

Chapter 2

Theory

2.1 Why a Self Tuning Controller?

The transfer function of SBHS is assumed as

$$\Delta T = \frac{K_p}{\tau_p s + 1}\Delta H + \frac{K_f}{\tau_f s + 1}\Delta F \tag{2.1}$$

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.

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.

2.3 Direct synthesis

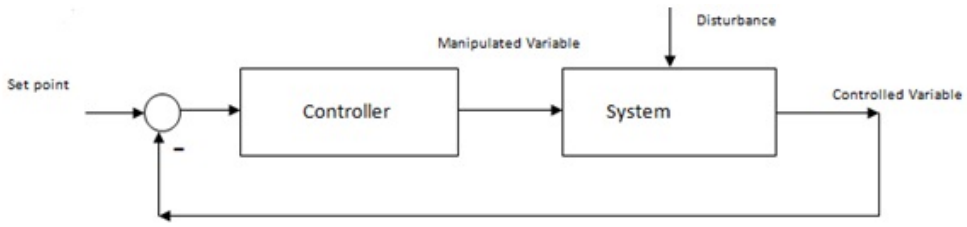

Figure 2.1: Closed Loop Circuit

$$V(s) = \frac{G_c(s)G(s)}{1 + G_c(s)G(s)} \tag{2.2}$$

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

$$V(s) = \frac{1}{(\tau_{cl} s + 1)} \tag{2.3}$$

$$G(s) = \frac{K_p}{(\tau_p s + 1)} \tag{2.4}$$

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

$$G(c) = K_c(1 + \frac{1}{\tau s}) \tag{2.5}$$

Where,
$K_c = 1\,\mathrm{K}_p\ (\tau_p\ /\ \tau_{cl}\ )$
$\tau_i = \tau_p$
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.

2.4 Ziegler Nichols Tuning

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

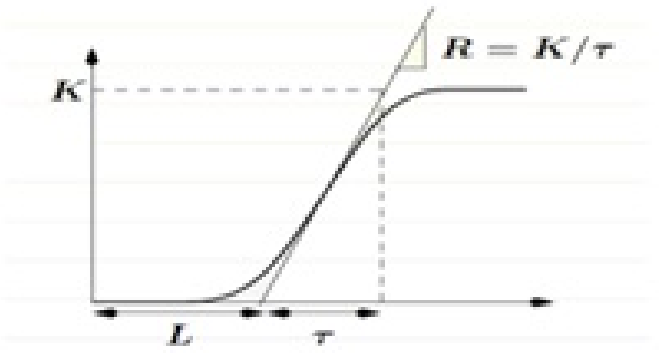

Figure 2.2: Tangent Approach to Ziegler Nichols Tuning

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

Table 2.1: Ziegler Nichols PID Settings

Table 2.1 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
- $\tau_p \approx \tau$

Chapter 3

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.

3.1 Step Test Experiments

3.1.1 Step Change in Heater Reading from 10 to 15

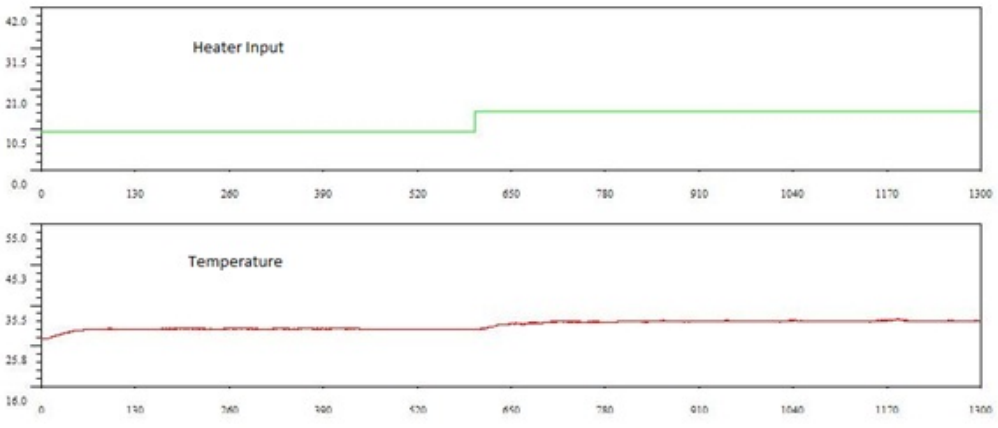

Figure 3.1: Step Responce for Heater Reading 10 to 15

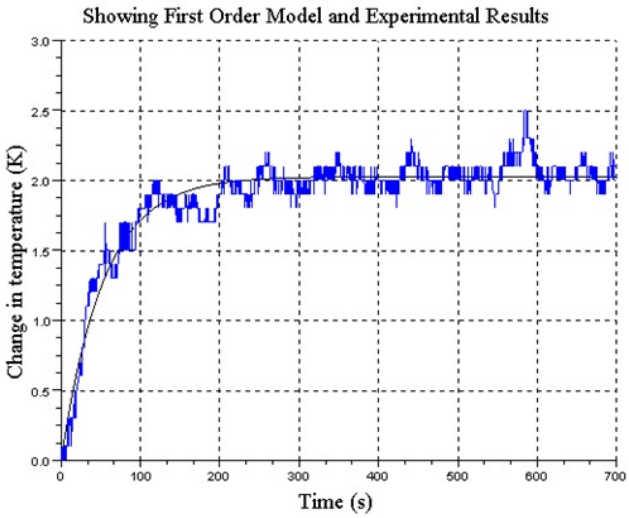

Figure 3.2: Step Response for Heater reading 10 to 15 in terms of Deviation

3.1.2 Step Change in Heater Reading from 20 to 25

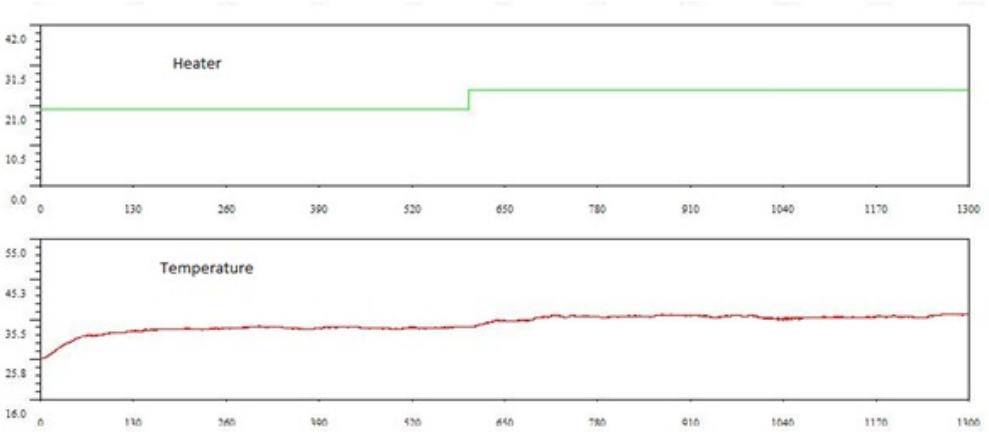

Figure 3.3: Step Responce for Heater Reading 20 to 25

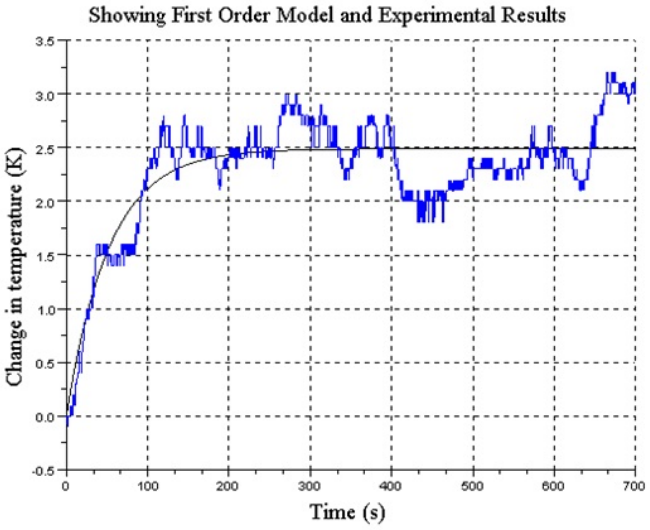

Figure 3.4: Step Response for Heater reading 20 to 25 in terms of Deviation

3.1.3 Step Change in Heater Reading from 30 to 35

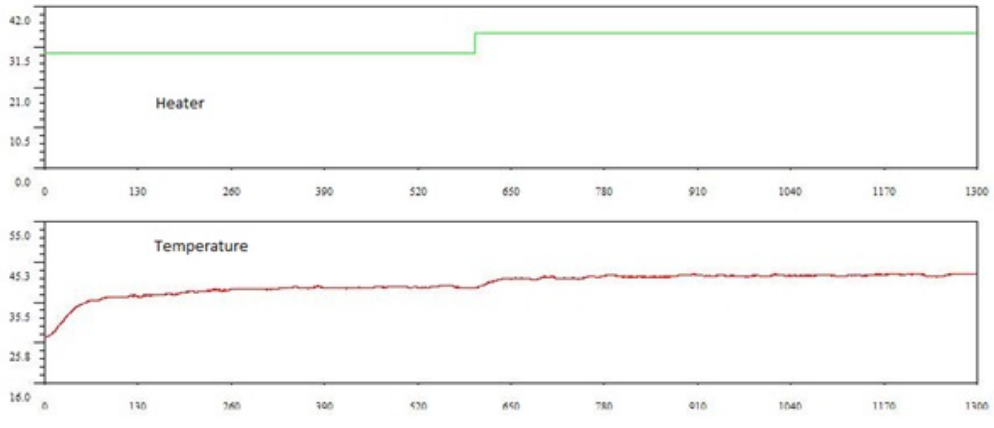

Figure 3.5: Step Responce for Heater Reading 30 to 35

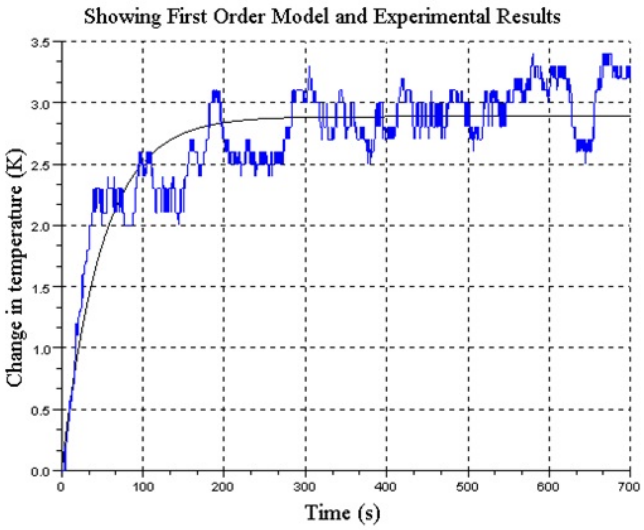

Figure 3.6: Step Response for Heater reading 30 to 35 in terms of Deviation

3.1.4 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 3.1: Open Loop Parameters

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

3.2 Conventional Controller Design

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

- Kc = 26.327
- $\tau_i = 12$
- $\tau_d = 3$

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

3.3 Self Tuning Controller Design

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

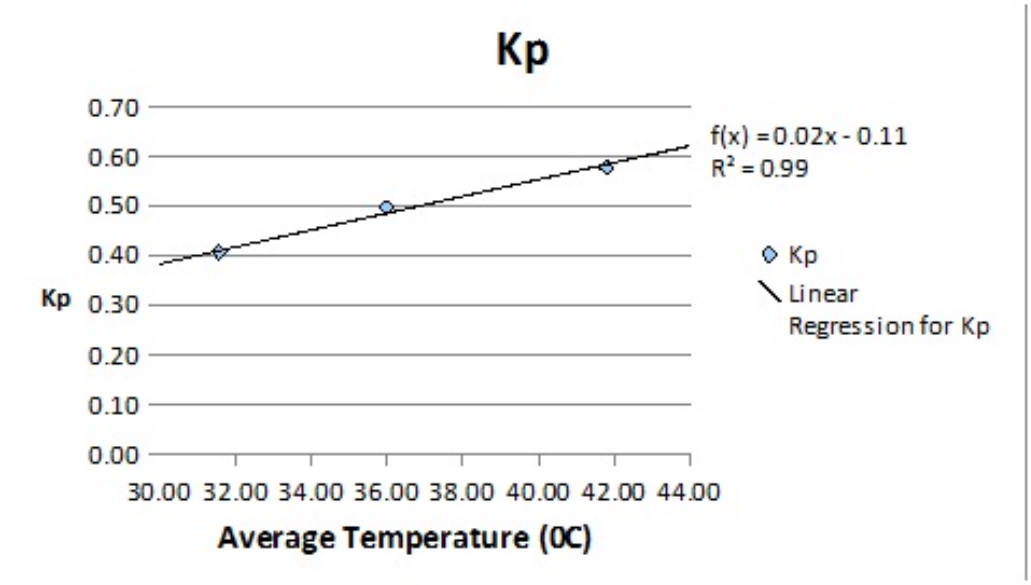

Figure 3.7: Variation of K_p with temperature

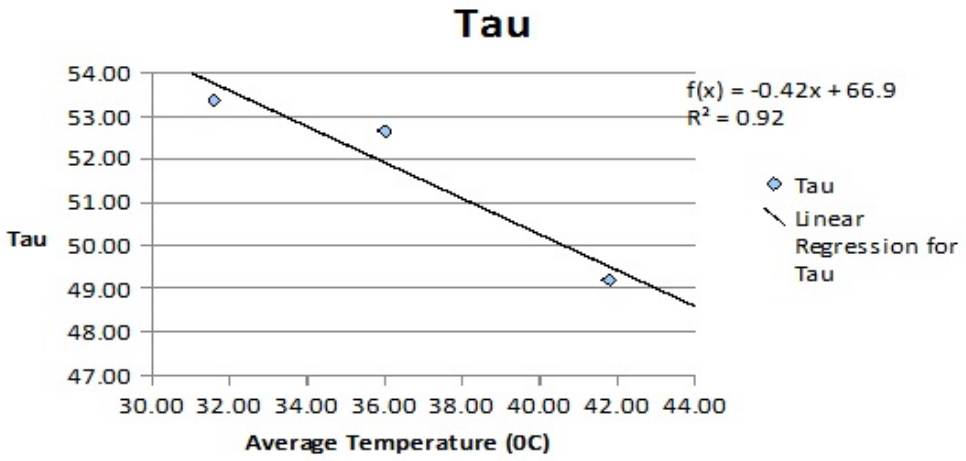

Figure 3.8: Variation of τ_p with temperature

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)
   $\tau_i = 2 \times 6 = 12$
   $\tau_d = 0.5 \times 6 = 3$

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

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

Chapter 4

Implementation

4.1 PI Controller

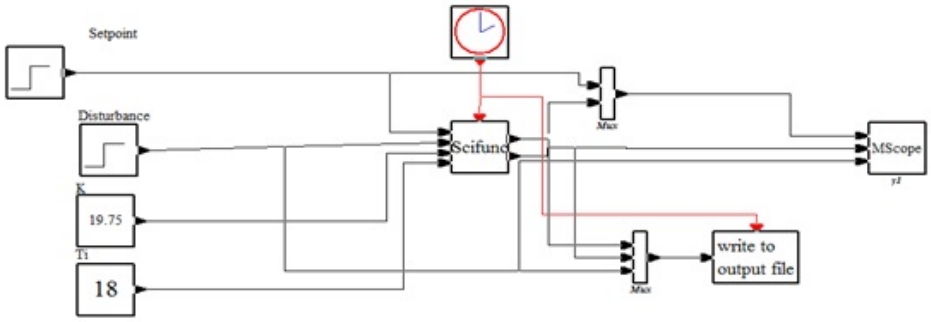

Figure 4.1: Scicos Diagram for PI Controller

The PI Controller in Continuous Time is given by:

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

On taking Laplace Transform, we obtain:

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

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

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

On Cross Multiplication, we obtain:

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

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

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

The PI Controller is usually written as:

$$u(n) = u(n-1) + s_0 e(n) + s_1 e(n-1) \tag{4.1}$$

Where,

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

4.2 PID Controller

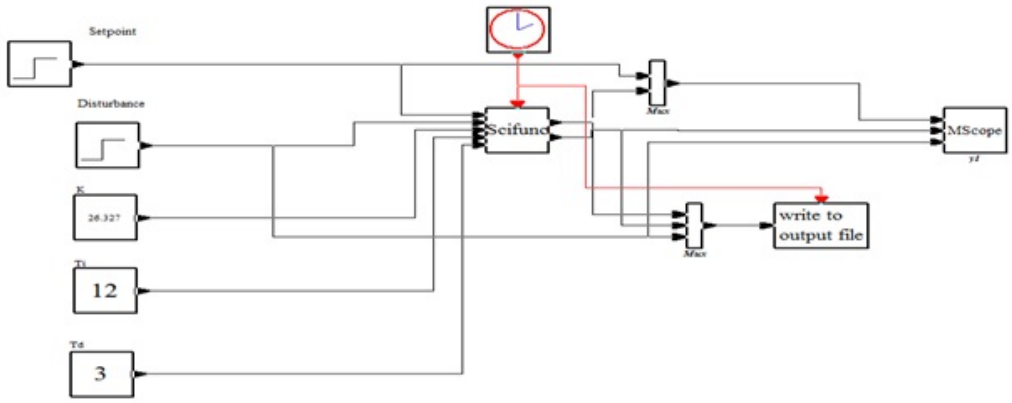

Figure 4.2: Scicos Diagram for PID Controller

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:

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

The PID Controller is usually written as:

$$u(n) = u(n-1) + s_0 e(n) + s_1 e(n-1) + s_2 e(n-2) \tag{4.2}$$

Where,

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

4.3 Self Tuning Controller

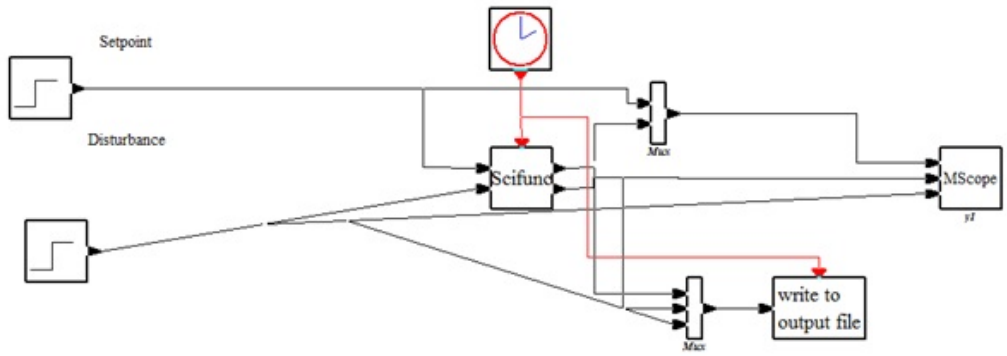

Figure 4.3: Scicos Diagram for Self Tuning Controller

The parameters of the Controller are determined dynamically using the temperature values during every sampling time. For this, the formulae derived in section 3.3 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.

Chapter 5

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 5.1 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 5.1: Set Point Changes in experiments conducted for Set Point Tracking

5.1 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.

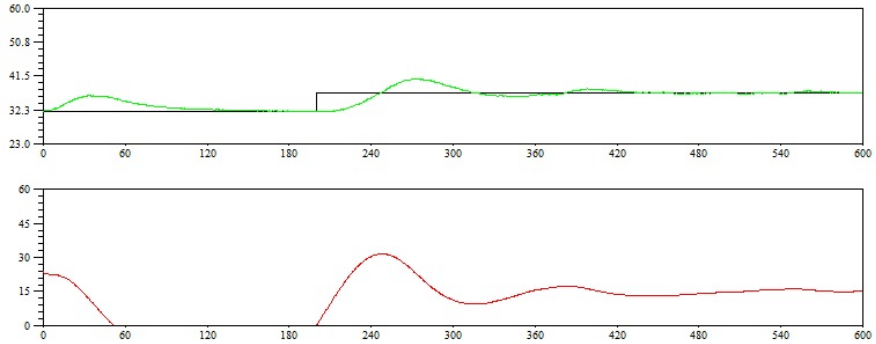

Figure 5.1: Result for Self Tuning Controller designed using Direct Synthesis for Set Point going from 32^0C to 37^0C

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.

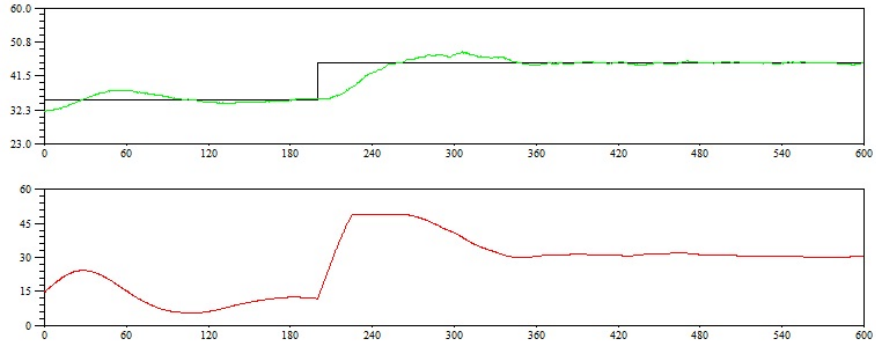

Figure 5.2: Result for Self Tuning Controller designed using Direct Synthesis for Set Point going from 35^0C to 45^0C

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.

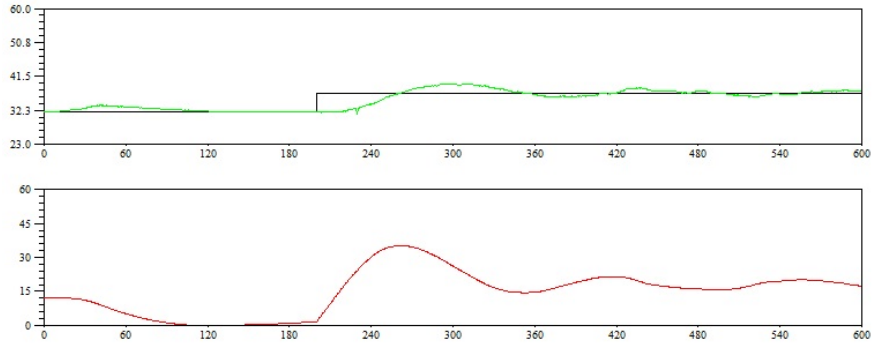

Figure 5.3: Result for Conventional Controller designed using Direct Synthesis for Set Point going from 32^0C to 37^0C

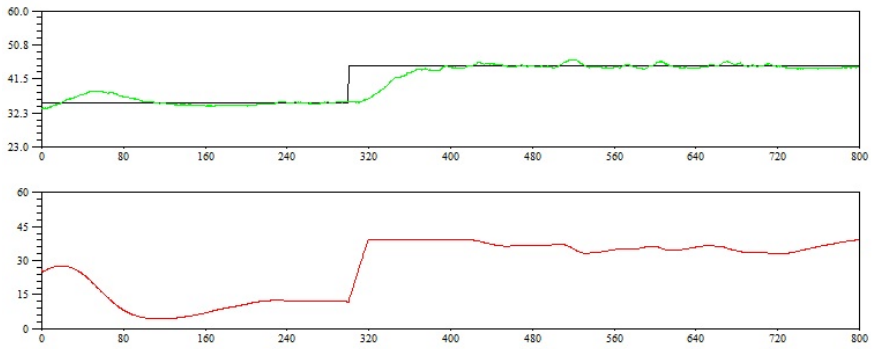

Figure 5.4: Result for Conventional Controller designed using Direct Synthesis for Set Point going from 35^0C to 45^0C

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

5.2 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.

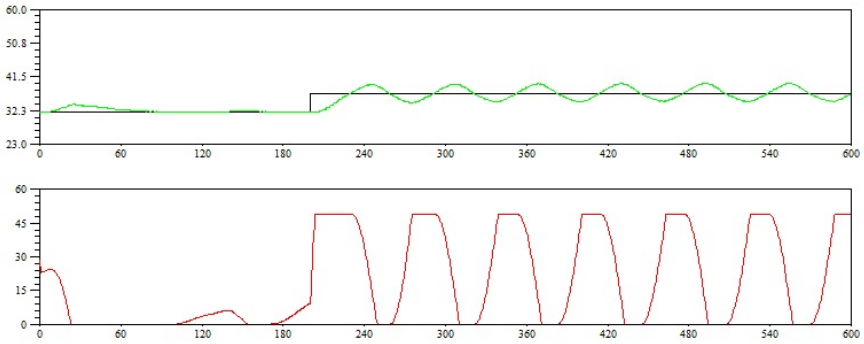

Figure 5.5: Result for Self Tuning Controller designed using Ziegler Nichols Tuning for Set Point going from 32^0C to 37^0C

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

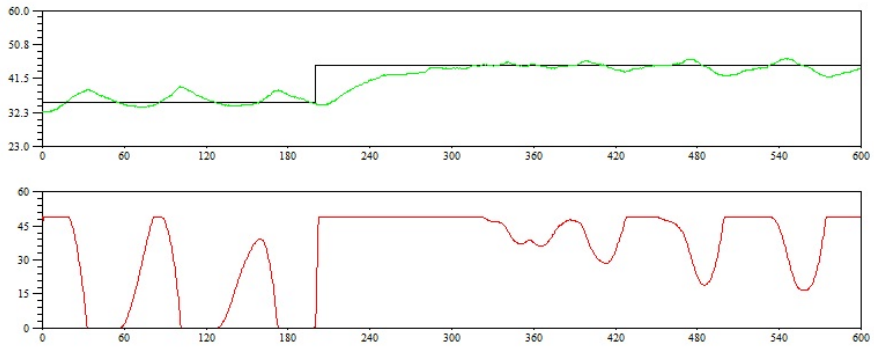

Figure 5.6: Result for Self Tuning Controller designed using Ziegler Nichols Tuning for Set Point going from 35^0C to 45^0C

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.

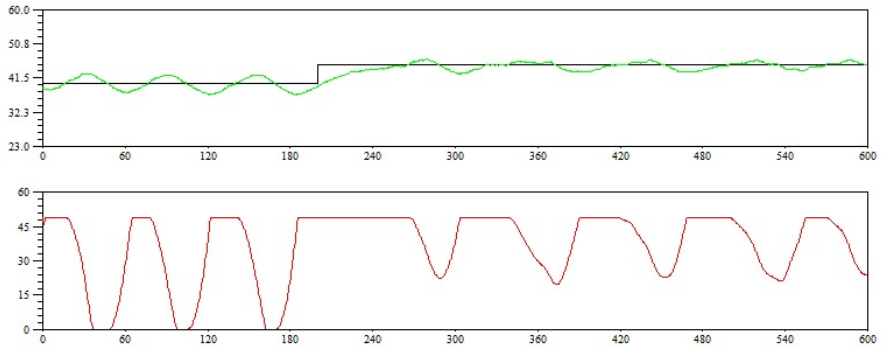

Figure 5.7: Result for Self Tuning Controller designed using Ziegler Nichols Tuning for Set Point going from 40^0C to 45^0C

In this experiment too, the controller is able to keep the temperature close to the set point and it stabilises fast.
For comparison, results of experiments done with conventional PI controller designed using the Ziegler Nichols method are also shown.

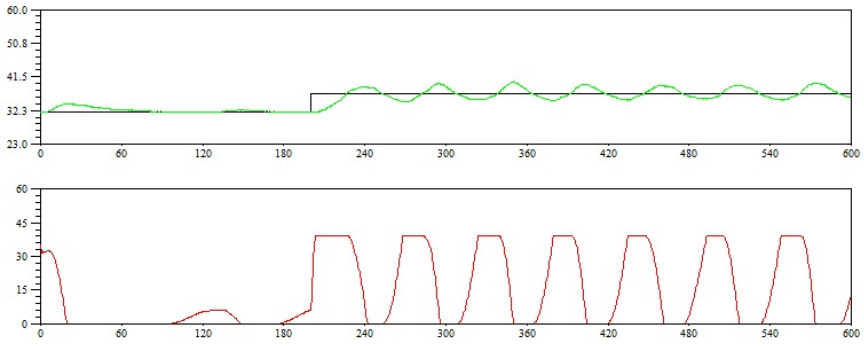

Figure 5.8: Result for Conventional Controller designed using Ziegler Nichols Tuning for Set Point going from 32^0C to 37^0C

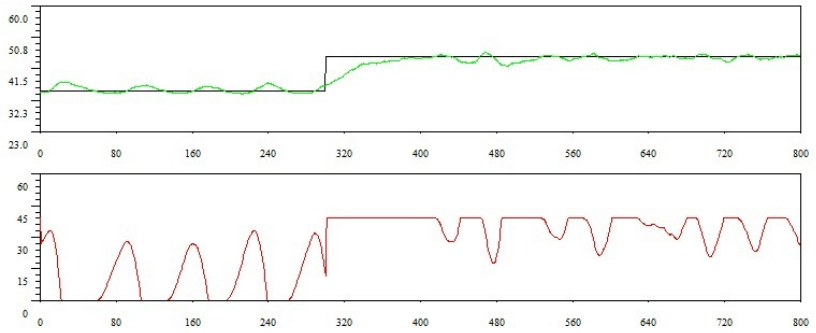

Figure 5.9: Result for Conventional Controller designed using Ziegler Nichols Tuning for Set Point going from 35^0C to 45^0C

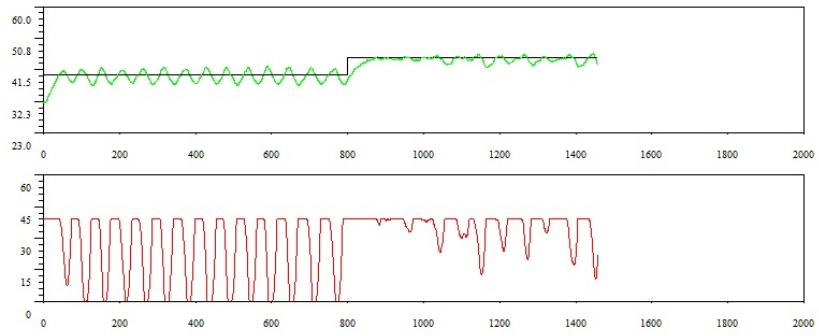

Figure 5.10: Result for Conventional Controller designed using Ziegler Nichols Tuning for Set Point going from 40^0C to 45^0C

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

5.3 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.

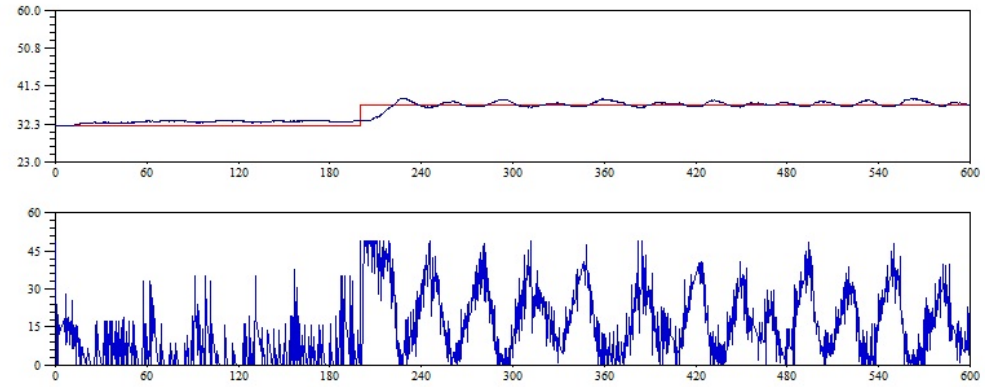

Figure 5.11: Result for Self Tuning PID Controller designed using Ziegler Nichols Tuning for Set Point going from 32^0C to 37^0C

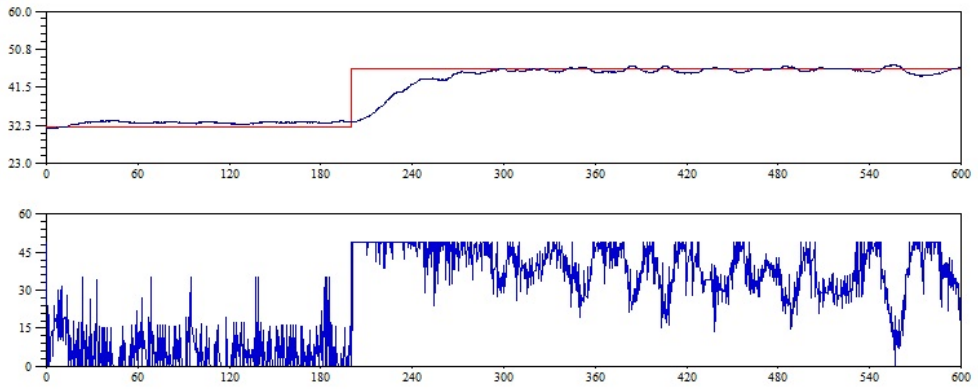

Figure 5.12: Result for Self Tuning PID Controller designed using Ziegler Nichols Tuning for Set Point going from 32^0C to 46^0C

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.

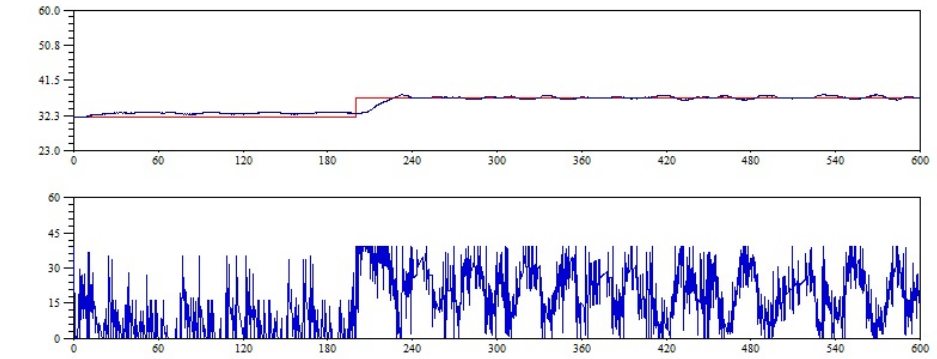

Figure 5.13: Result for Conventional PID Controller designed using Ziegler Nichols Tuning for Set Point going from 32^0C to 37^0C

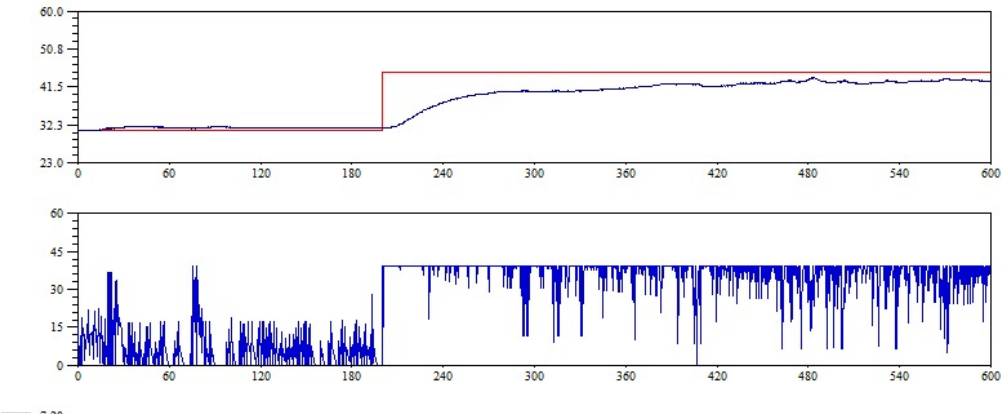

Figure 5.14: Result for Conventional PID Controller designed using Ziegler Nichols Tuning for Set Point going from 31^0C to 45^0C

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

5.4 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 because for higher values of set point change, the self tuning PID controller shows a better performance than the conventional controller as seen in section 5.3.

Chapter 6

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 6.1 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 6.1: Fan Input Changes in experiments conducted for Disturbance Rejection

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

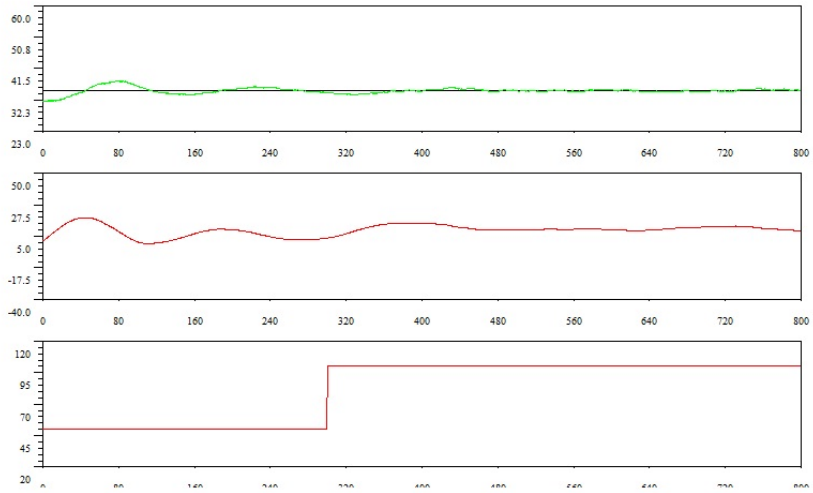

Figure 6.1: Results for Fan Input Change from 50 to 100 for Self Tuning PI Controller designed using Direct Synthesis

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.

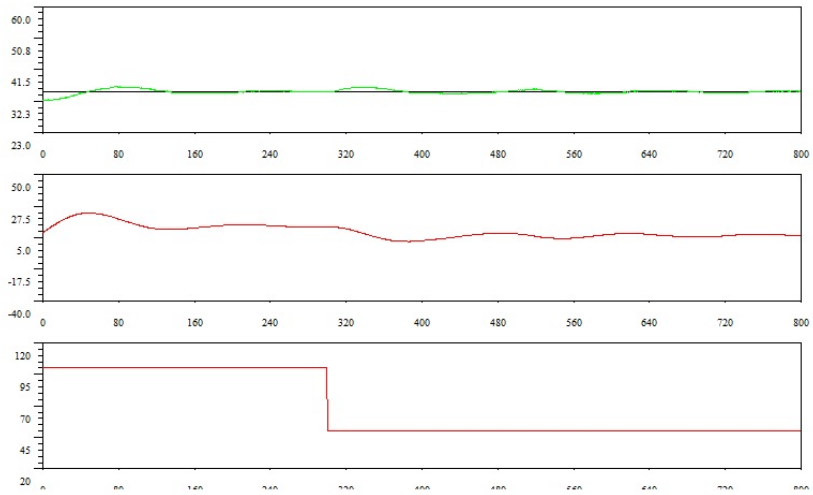

Figure 6.2: Results for Fan Input Change from 100 to 50 for Self Tuning PI Controller designed using Direct Synthesis

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.

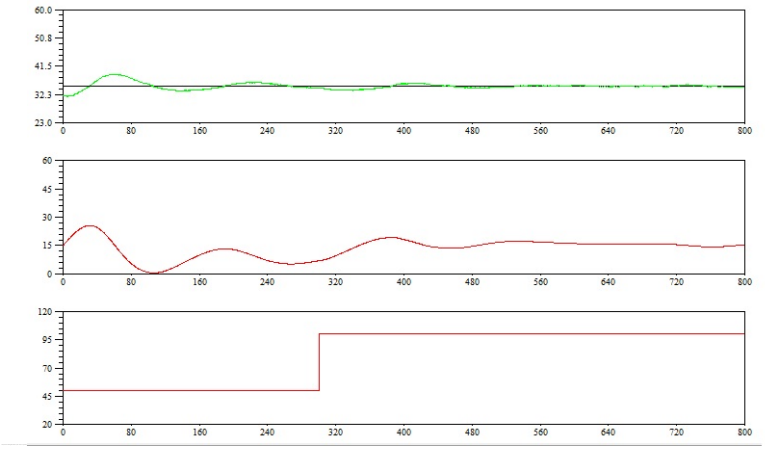

Figure 6.3: Results for the Fan input change from 50 to 100 to Conventional PI Controller designed using Direct Synthesis

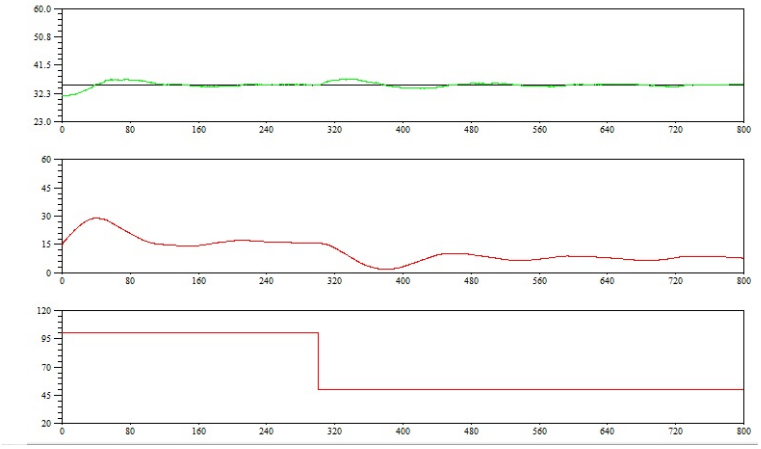

Figure 6.4: Results for the Fan input change from 100 to 50 to Conventional PI Controller designed using Direct Synthesis

6.2 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.

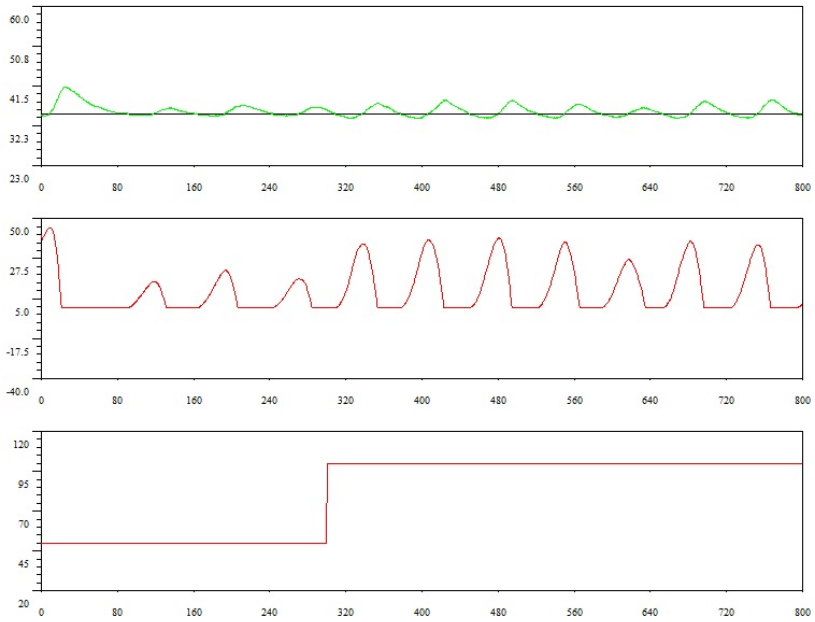

Figure 6.5: Results for Fan Input change from 50 to 100 given to Self Tuning PI Controller designed using Ziegler Nichols Method

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.

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

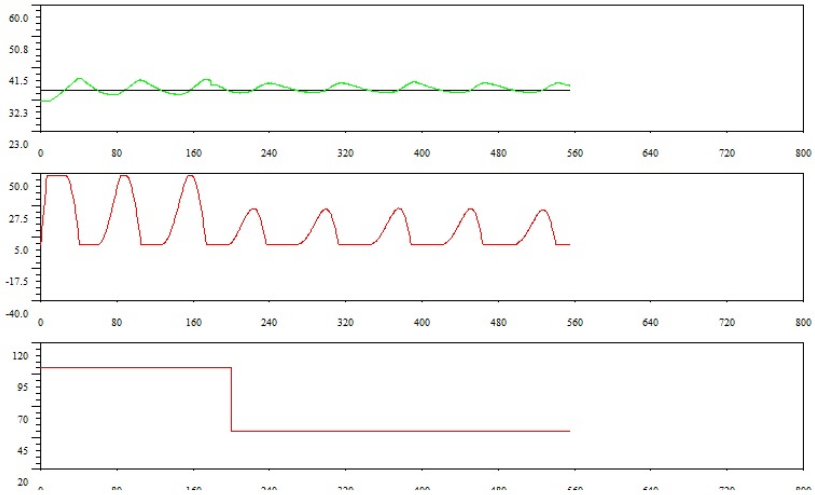

Figure 6.6: Results for Fan Input change from 100 to 50 given to Self Tuning PI Controller designed using Ziegler Nichols Method

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.

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

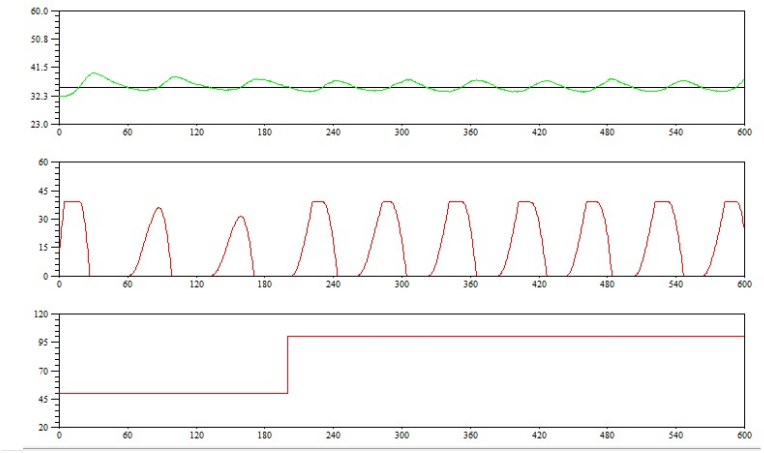

Figure 6.7: Results for the Fan input change from 50 to 100 to Conventional PI Controller designed using Ziegler Nichols Tuning

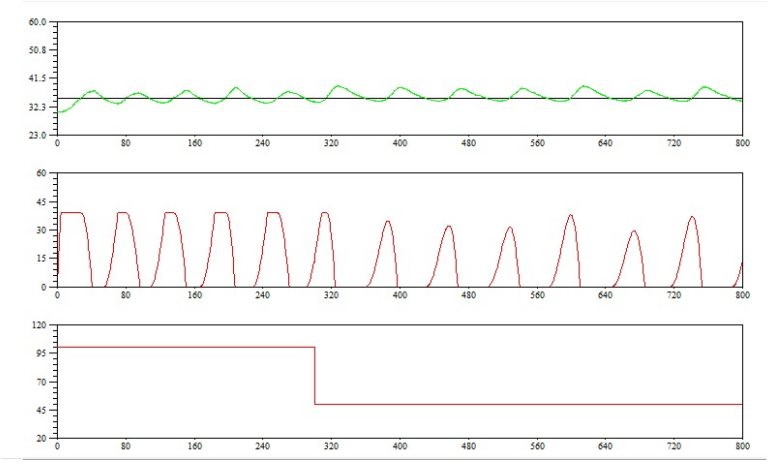

Figure 6.8: Results for the Fan input change from 100 to 50 to Conventional PI Controller designed using Ziegler Nichols Tuning

6.3 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.

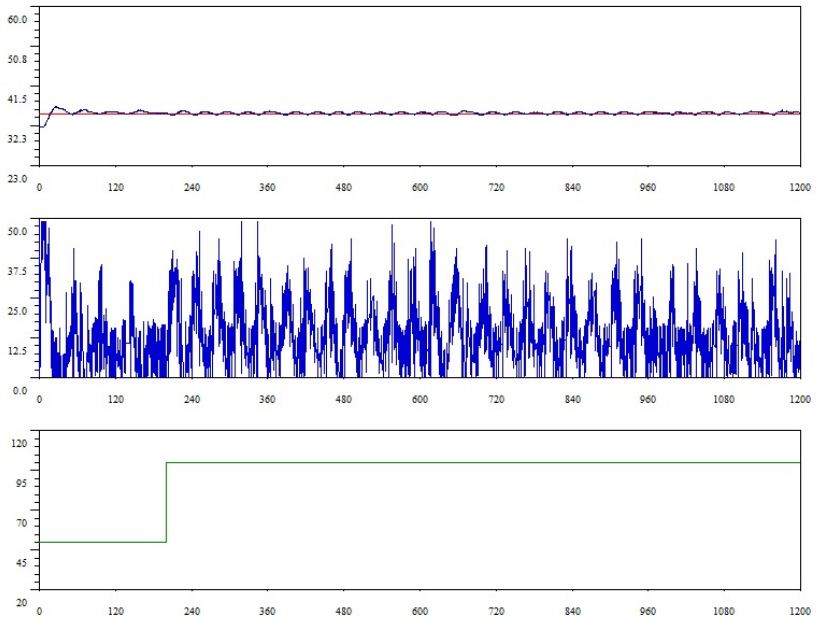

Figure 6.9: Results for Fan Input change from 50 to 100 given to Self Tuning PID Controller designed using Ziegler Nichols Method

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.

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

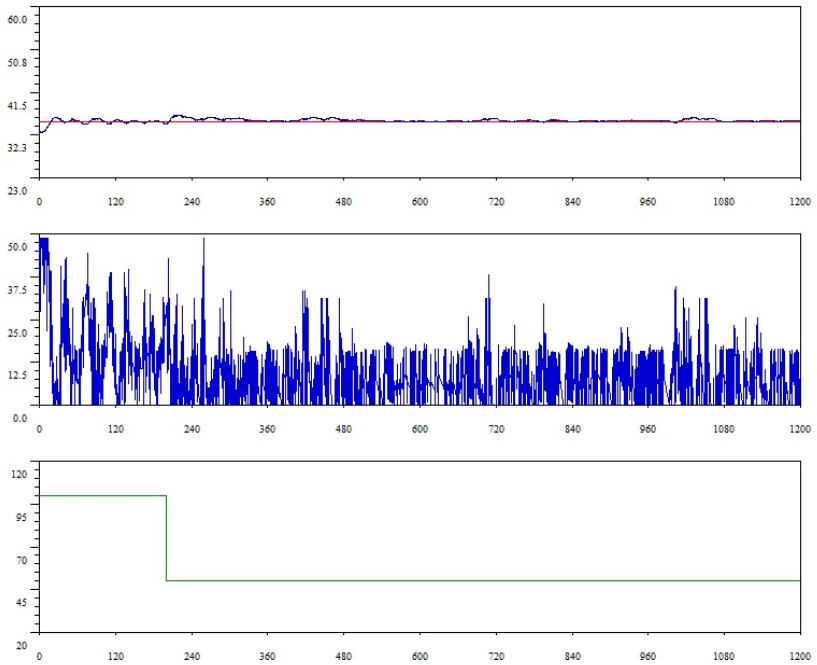

Figure 6.10: Results for Fan Input change from 100 to 50 given to Self Tuning PID Controller designed using Ziegler Nichols Method

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.

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

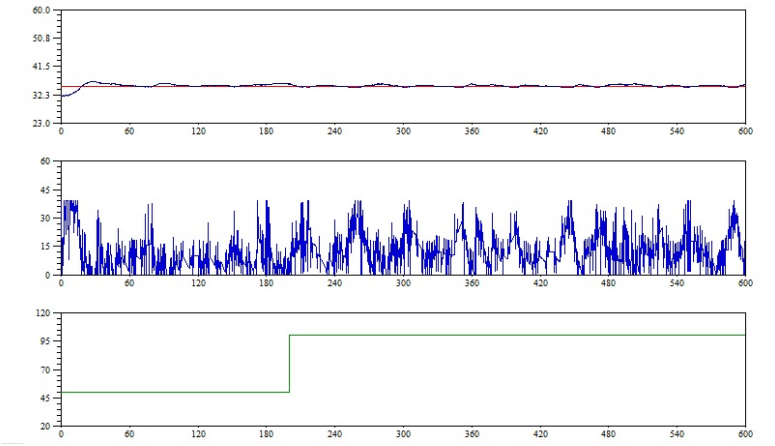

Figure 6.11: Results for the Fan input change from 50 to 100 to Conventional PID Controller designed using Ziegler Nichols Tuning

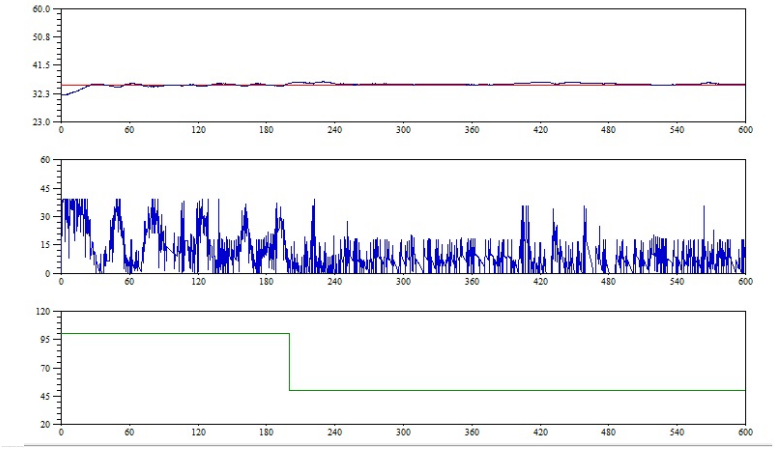

Figure 6.12: Results for the Fan input change from 100 to 50 to Conventional PID Controller designed using Ziegler Nichols Tuning

6.4 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.

Chapter 7

Reproducing the Results

The following steps can be used for conducting the experiments:

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

The various codes used for conventional and self tuning controllers are shown.

7.1 Serial Communication

```
dll_link = c_link('serial');
//if not loaded - load it
if ~dll_link,
  addinter('serial.dll','serial',...
  [ 'opencom';'closecom'; 'writecom';...
  'writebincom';'readbincom';'resetcom';'getcomhandles']);
end;

com_config = tlist(['Config';'port';....
'baudrate';'nbits';'parity';'stopbits';'protocol'],5,9600,8,0,0,0);
handl = opencom(com_config);
```

7.2 Conventional Controller

7.2.1 Fan Disturbance in PI Controller

```
function [temp,C0] = pi_bda_dist(setpoint,disturbance,K,Ti)

global temp heat_in fan_in C0 u_old u_new e_old e_new

Ts=1;
S0=K*(1+((Ts/Ti)));
S1=-K;
u_new = u_old+(S0*e_new)+(S1*e_old);
e_new = setpoint - temp;

if u_new> 39;
  u_new = 39;
end;

if u_new< 0;
  u_new = 0;
end;

C0=u_new;
heat_in = C0;
fan_in = disturbance;
u_old = u_new;
e_old = e_new;

ok = writebincom(handl,[254]); //heater
ok = writebincom(handl,[heat_in]);
ok = writebincom(handl,[253]); //fan
ok = writebincom(handl,[fan_in]);
ok = writebincom(handl,[255]);
sleep(10);
[temp,ok,nbytes] = readbincom(handl,2);
temp = temp(1) + 0.1*temp(2);

endfunction;
```

7.2.2 Set Point Change in PI Controller

```
function [temp,C0,e_new] = pi_bda(setpoint,disturbance,K,Ti)

global temp heat_in fan_in C0 u_old u_new e_old e_new

Ts=1;
S0=K*(1+((Ts/Ti)));
S1=-K;
u_new = u_old+(S0*e_new)+(S1*e_old);
e_new = setpoint - temp;

if u_new> 39;
  u_new = 39;
end;

if u_new< 0;
  u_new = 0;
end;

C0=u_new;
heat_in = C0;
fan_in = disturbance;
u_old = u_new;
e_old = e_new;

ok = writebincom(handl,[254]); //heater
ok = writebincom(handl,[heat_in]);
ok = writebincom(handl,[253]); //fan
ok = writebincom(handl,[fan_in]);
ok = writebincom(handl,[255]);
sleep(10);
[temp,ok,nbytes] = readbincom(handl,2);
temp = temp(1) + 0.1*temp(2);

endfunction;
```

7.2.3 Fan Disturbance to PID Controller

```
function [temp,CO] = pid_bda_dist(setpoint,disturbance,K,Ti,Td)
global temp heat_in fan_in et SP CO eti u_old u_new e_old e_new e_old_old
```

```
e_new = setpoint - temp;

Ts=1;

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;
et = setpoint - temp;
CO = u_new;

  if CO>39
    CO = 39;
  end;

  if CO<0
    CO = 0;
  end;

u_new = CO;

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


heat_in = CO;
fan_in = disturbance;

ok = writebincom(handl,[254]); //heater
ok = writebincom(handl,[heat_in]);
ok = writebincom(handl,[253]);
ok = writebincom(handl,[fan_in]);
ok = writebincom(handl,[255]);
sleep(1);
[temp,ok,nbytes] = readbincom(handl,2);
temp = temp(1) + 0.1*temp(2);
endfunction;
```

7.2.4 Set Point Change in PID Controller

```
function [temp,CO,et] = pid_bda(setpoint,disturbance,K,Ti,Td)
global temp heat_in fan_in et SP CO eti u_old u_new e_old e_new e_old_old

e_new = setpoint - temp;

Ts=1;

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;
et = setpoint - temp;
CO = u_new;

  if CO>39
    CO = 39;
  end;

  if CO<0
    CO = 0;
  end;

u_new = CO;

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


heat_in = CO;
fan_in = disturbance;

ok = writebincom(handl,[254]); //heater
ok = writebincom(handl,[heat_in]);
ok = writebincom(handl,[253]);
ok = writebincom(handl,[fan_in]);
ok = writebincom(handl,[255]);
sleep(1);
```

```
[temp,ok,nbytes] = readbincom(handl,2);
temp = temp(1) + 0.1*temp(2);
endfunction;
```

7.3 Self Tuning Controller

7.3.1 Fan Discturbance to PI Controller

```
function [temp,C0] = pi_bda_tuned_dist(setpoint,disturbance)

global temp heat_in fan_in C0 u_old u_new e_old e_new
//L = 6;
//R = (0.016*temp-0.114)/(66.90-0.415*temp);
//K = 0.9/(R*L);
//Ti = 3*L;

    //the above is the ziegler nichols part


K = 2/(0.016*temp-0.114);
Ti = (66.90-0.415*temp);

//the above is the direct synthesis part


Ts=1;
S0=K*(1+((Ts/Ti)));
S1=-K;
u_new = u_old+(S0*e_new)+(S1*e_old);
e_new = setpoint - temp;

if u_new> 49;
  u_new = 49;
end;

if u_new< 0;
  u_new = 0;
end;
```

```
C0=u_new;
heat_in = C0;
fan_in = disturbance;
u_old = u_new;
e_old = e_new;

ok = writebincom(handl,[254]); //heater
ok = writebincom(handl,[heat_in]);
ok = writebincom(handl,[253]); //fan
ok = writebincom(handl,[fan_in]);
ok = writebincom(handl,[255]);
sleep(10);
[temp,ok,nbytes] = readbincom(handl,2);
temp = temp(1) + 0.1*temp(2);

endfunction;
```

7.3.2 Set Point Change to PI Controller

```
function [temp,C0,e_new] = pi_bda_tuned(setpoint,disturbance)

global temp heat_in fan_in C0 u_old u_new e_old e_new
//L = 6;
//R = (0.016*temp-0.114)/(66.90-0.415*temp);
//K = 0.9/(R*L);
//Ti = 3*L;


//The above is the Ziegler nichols part.

K = 2/(0.016*temp-0.114);
Ti = (66.90-0.415*temp);
//The above is the direct synthesis part

Ts=1;
S0=K*(1+((Ts/Ti)));
S1=-K;
u_new = u_old+(S0*e_new)+(S1*e_old);
e_new = setpoint - temp;
```

```
if u_new> 49;
  u_new = 49;
end;

if u_new< 0;
  u_new = 0;
end;

C0=u_new;
heat_in = C0;
fan_in = disturbance;
u_old = u_new;
e_old = e_new;

ok = writebincom(handl,[254]); //heater
ok = writebincom(handl,[heat_in]);
ok = writebincom(handl,[253]); //fan
ok = writebincom(handl,[fan_in]);
ok = writebincom(handl,[255]);
sleep(10);
[temp,ok,nbytes] = readbincom(handl,2);
temp = temp(1) + 0.1*temp(2);

endfunction;
```

7.3.3 Fan Disturbance to PID Controller

```
function [temp,CO] = pid_bda_tuned_dist(setpoint,disturbance)
global temp heat_in fan_in et SP CO eti u_old u_new e_old e_new e_old_old
L = 6;
R = (0.016*temp-0.114)/(66.90-0.415*temp);
K = 1.2/(R*L)
Ti = 2*L;
Td = 0.5*L;


e_new = setpoint - temp;

Ts=1;

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;
et = setpoint - temp;
CO = u_new;

  if CO>49
    CO = 49;
  end;

  if CO<0
    CO = 0;
  end;

u_new = CO;

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


heat_in = CO;
fan_in = disturbance;

ok = writebincom(handl,[254]); //heater
ok = writebincom(handl,[heat_in]);
ok = writebincom(handl,[253]);
ok = writebincom(handl,[fan_in]);
ok = writebincom(handl,[255]);
sleep(1);
[temp,ok,nbytes] = readbincom(handl,2);
temp = temp(1) + 0.1*temp(2);
endfunction;
```

7.3.4 Set Point Change to PID Controller

```
function [temp,CO,et] = pid_bda_tuned(setpoint,disturbance)
global temp heat_in fan_in et SP CO eti u_old u_new e_old e_new e_old_old
L = 6;
R = (0.016*temp-0.114)/(66.90-0.415*temp);
```

```
K = 1.2/(R*L)
Ti = 2*L;
//Kc and tau_i calculated
Td = 0.5*L;
e_new = setpoint - temp;
Ts=1;
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;
et = setpoint - temp;
CO = u_new;

  if CO>49
    CO = 49;
  end;

  if CO<0
    CO = 0;
  end;

u_new = CO;

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

heat_in = CO;
fan_in = disturbance;

ok = writebincom(handl,[254]); //heater
ok = writebincom(handl,[heat_in]);
ok = writebincom(handl,[253]);
ok = writebincom(handl,[fan_in]);
ok = writebincom(handl,[255]);
sleep(1);
[temp,ok,nbytes] = readbincom(handl,2);
temp = temp(1) + 0.1*temp(2);
endfunction;
```