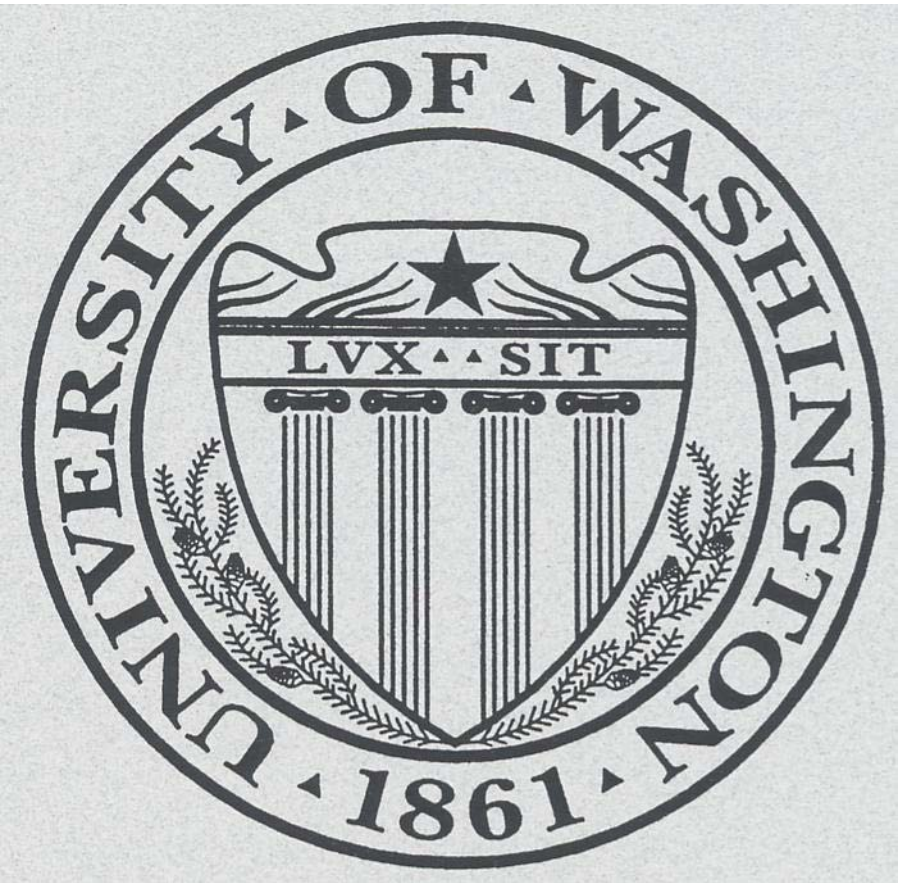
UNIVERSITY OF WASHINGTON
Department of Aeronautics and Astronautics

Modeling and Design of a Temperature Control System



February 6, 2003

Christopher Lum
Travis Reisner
Amanda Stephens
Brian Hass

LAB EXPERIMENT I

Modeling and Design of a Temperature Control System

by

Group C-I

February 6, 2003

Brian Hass **:** Experimental apparatus, controls system design.

Contribution: %

Brian Hass

Christopher Lum **:** Evaluate the accuracy of the model with simulation, assemble and perform experiments to determine the model parameters, develop MatLab code and Simulink model, perform root locus control system design.

Contribution: %

Christopher Lum

Travis Reisner **:** Procedure, evaluate experimental PID performance, conclusion, develop MatLab code.

Contribution: %

Travis Reisner

Amanda Stephens **:** Objectives, control system design, appendix.

Contribution: %

Amanda Stephens

Estimated total time spent: 60+ hours

I. Experiment Objectives

The objective of this experiment was to gain experience with the concept of feedback control and how it can be used in controlling the temperature of a block of aluminum to within a fraction of a degree Fahrenheit. This control capability was demonstrated in a two-part laboratory experiment which first included making a model of the thermal mechanical system and secondly included designing a feedback control system utilizing PID control. Secondary objectives of this laboratory were to become familiar with MatLab and Simulink which can then be used to construct models of the real world system for analysis.

II. Experiment Apparatus

For this lab, the entire physical apparatus was simply a square block of aluminum, 25mm per side and 3 mm thick, with an OHMITE 5W power resistor of 25 Ω +/-1%, and a LM35 precision centigrade temperature sensor attached. This setup can be seen below in Figure 1.

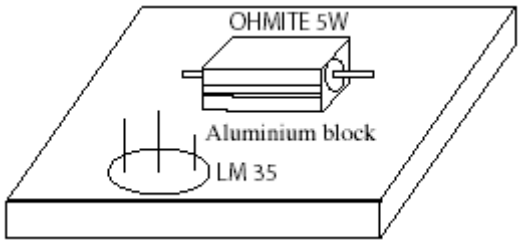


Figure 1: Illustration of Aluminum block setup

In addition, the electrical circuit was set up as seen below in Figure 2.

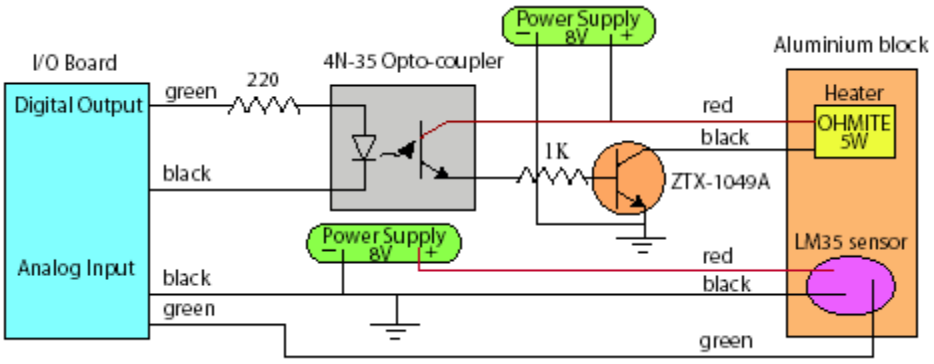


Figure 2: Complete apparatus

The I/O Board is the portion connected to the computer and LabView, the 4N-35 Opto-coupler transfers the input state (ON/OFF) into an output state (also ON/OFF) using light, and the ZTX-1049A high gain transistor provides the current necessary to heat the resistor on the block. Also, 2 resistors, 220 Ω and 1 kΩ, were used along with 2 separate 8V power supplies, one for each the heater and the sensor.

III. List of Symbols

A list of symbols used in this report are shown below in Table 1.

Table 1: List of symbols

ρ	Density of block (kg/m^3)
h	Height of block (m)
w	Width of block (m)
L	Length of block (m)
c	Specific heat of block (J/kgK)
V_{ref}	Voltage across heater (volts)
R	Resistance of heater (ohms)
T	Temperature at heater (F)
T_{sensor}	Temperature at sensor (F)
ΔT_{ss}	Change in steady state temperature (F)
E	Energy (J)
κ	Convective Heat Transfer Coefficient ($watts/m^2K$)
S	Block surface area (m^3)
u(t)	Duty cycle as a function of time (%)
u_o	Constant duty cycle (%)
K_p	Proportional gain value
K_i	Integral gain value
K_d	Derivative gain value
K_c	Proportional gain value where system is neutrally stable (with $K_i = K_d = 0$)
P_c	Period of oscillation with $K_p = K_c$ and $K_i = K_d = 0$ (sec)

IV. System Modeling

Theory

As stated in the lab sheet, the change in energy stored in the block is directly related to the change in temperature of the block as shown below.

$$\rho(hwL)c\frac{dT}{dt} = \frac{dE_{in}}{dt} - \frac{dE_{out}}{dt} \qquad \textbf{(Eq.1)}$$

The energy added into the system is directly from the power dissipated by the heater (resistive heating). However, the resistive heating is not always taking place, this is controlled by pulse width modulation which controls the duty cycle, u(t), therefore, the resistive heating as a function of duty cycle is given by

$$\frac{dE_{in}}{dt} = \frac{V_{ref}{}^2}{R}u(t) \qquad \textbf{(Eq.2)}$$

Since the system is placed in a shielded box, the heat lost from the system is mainly through conduction. By ignoring convection and radiation, the energy lost from the system is given by

$$\frac{dE_{out}}{dt} = \kappa S(T - T_{ambient}) \qquad \textbf{(Eq.3)}$$

We can define the quantity $\Delta T = (T - T_{ambient})$ by using the ambient temperature as the ambient. Substituting Eq.2 and Eq.3 into Eq.1 and rearranging terms yields

$$\frac{d\Delta T}{dt} = k_1\frac{V_{ref}{}^2}{R}u(t) - k_2\Delta T \qquad \textbf{(Eq.4)}$$

The temperature on the block at the heater is defined as T. However, the sensor temperature, which is the recorded quantity, is not located at the same place and there is a time delay. This is represented by

$$T_{sensor}(t) = \Delta T(t-\tau) + T_{ambient} \qquad \textbf{(Eq.5)}$$

Therefore, combining Eq.4 and Eq.5 and taking the Laplace transform yields

$$\Delta T_{ss}(s) = \frac{k_1}{s+k_2}\frac{V_{ref}^{\ 2}}{R}e^{-\tau s}u(s) \qquad \textbf{(Eq.6)}$$

The block diagram representation of Eq.6 is shown below in Figure 3.

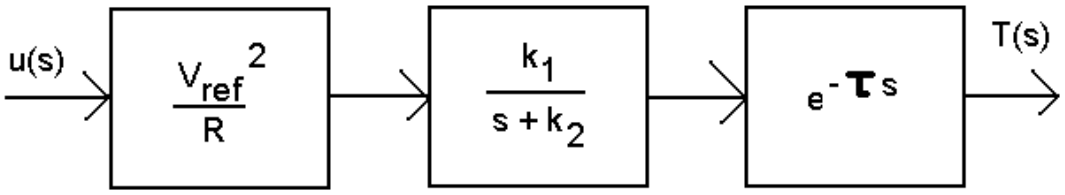


Figure 3: Block diagram of system

The system now has three unknown factors, k_1, k_2, and τ. These can be determined from the gathered data in several ways. The final change in steady state temperature due to a step input can be determined by applying the final value theorem to Eq.6 as shown below.

$$\lim_{t\to\infty}\Delta T(t) = \lim_{s\to 0} s\Delta T_{ss}(s)\frac{u_o}{s}$$

$$= \lim_{s\to 0}\frac{k_1}{s+k_2}\frac{V_{ref}^{\ 2}}{R}e^{-\tau s}u_o$$

$$\Delta T_{ss} = \frac{k_1}{k_2}\frac{V_{ref}^{\ 2}}{R}u_o \qquad \textbf{(Eq.7)}$$

Eq.7 describes a line with the x-axis of u_o and the y-axis as the change in steady state temperature (ΔT_{ss}). The slope of the line is the factor $(k_1 V_{ref}^{\ 2})/(k_2 R)$.

The value of τ may be solved by looking at the response of the system to a step change in duty cycle as shown below in Figure 4.

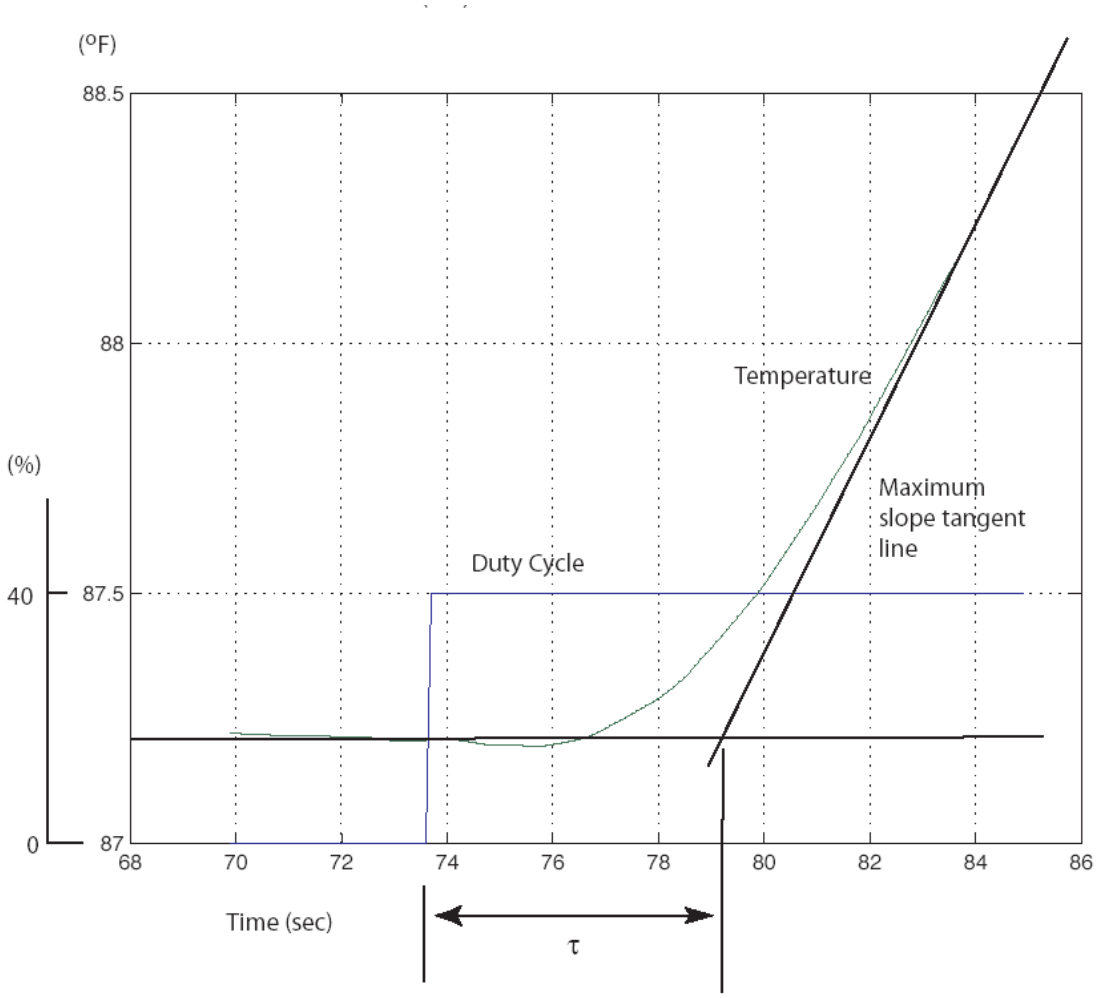


Figure 4: Theoretical temperature response to a step change in duty cycle[1]

The procedure to find τ is to obtain a plot of T_{sensor} vs. time and then to draw a tangent line with the maximum slope on the graph. τ is the time difference from where the duty cycle step change occurs and where the intersection of the maximum tangent line is to the previous steady state value.

The value of k_2 may be solved for two ways. The first method employs a graphical technique as shown below in Figure 6.

[1] Taken from lab sheet http://www.aa.washington.edu/courses/aa448/lab1/LabI.pdf

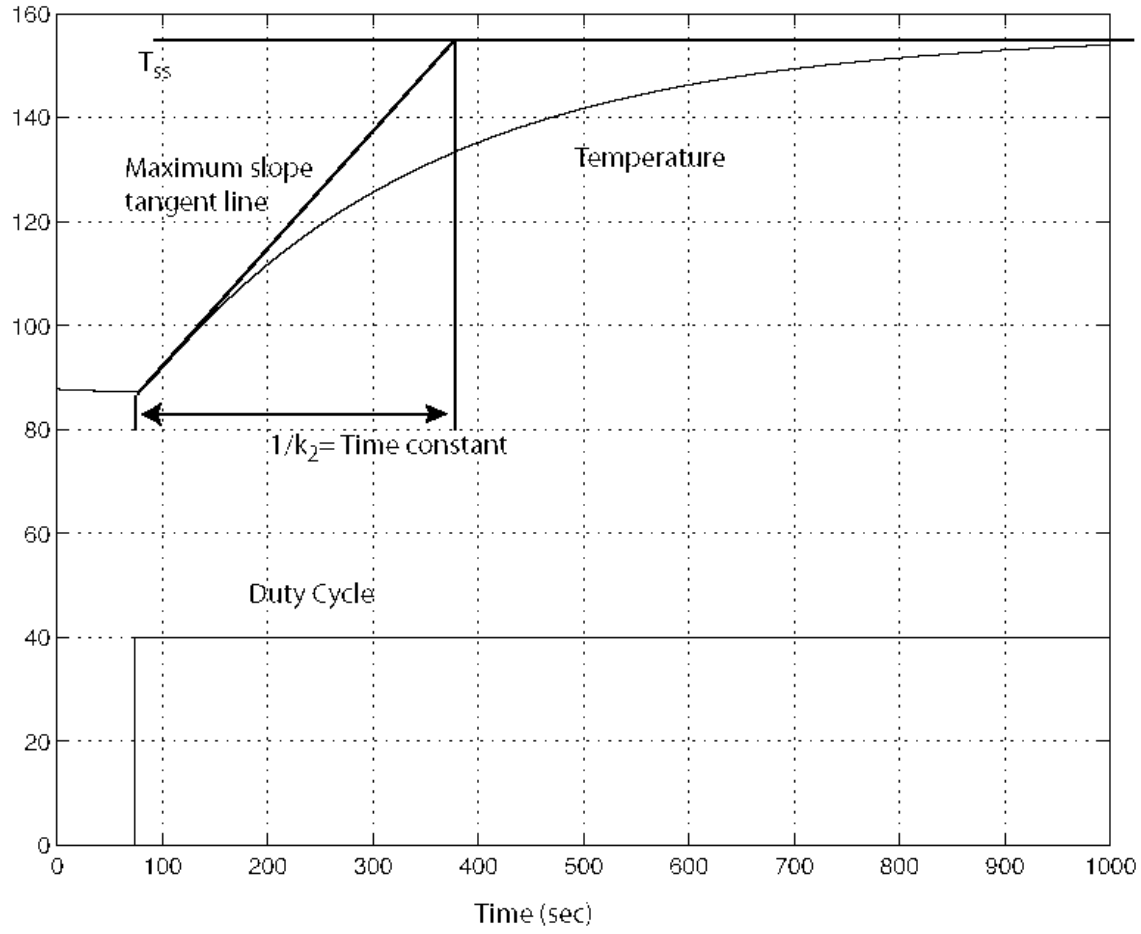


Figure 6: Theoretical temperature response to a step change in duty cycle[2]

The value of $1/k_2$ is the time difference from when the temperature begins to rise (not counting the transport delay). to where the maximum tangent line intersects the final steady state temperature.

k_2 can also be solved for mathematically. Ignoring the transport delay, and only looking at the graph of T_{sensor} vs. time <u>after</u> the temperature begins to rise, the transfer function for this system (with no time delay) is given by

$$G(s) = \frac{k_1}{s + k_2} \frac{V_{ref}^{\ 2}}{R} \qquad \textbf{(Eq.8)}$$

This can be rewritten as

$$G(s) = \frac{K_o}{s + k_2} \qquad \textbf{(Eq.9)}$$

[2] Taken from lab sheet http://www.aa.washington.edu/courses/aa448/lab1/LabI.pdf

When subjected to a step input of magnitude u_o, the change in temperature in the Laplace domain is given by

$$\Delta T_{ss}(s) = \frac{K_o}{s + k_2}\frac{u_o}{s} \qquad \textbf{(Eq.10)}$$

Taking the inverse Laplace transform yields

$$\Delta T_{ss}(t) = \frac{K_o u_o}{k_2}\left(1 - e^{-k_2 t}\right) \qquad \textbf{(Eq.11)}$$

We know that at t goes to infinity, the change in temperature must reach ΔT_{ss}. Therefore, the coefficient $K_o u_o / k_2$ must be equal to ΔT_{ss}. Eq.11 can now be written as

$$\Delta T_{ss}(t) = \Delta T_{ss}\left(1 - e^{-k_2 t}\right) \qquad \textbf{(Eq.12)}$$

Eq.12 represents the line of $T_{sensor} - T_{ambient}$ vs. time after the temperature begins to rise. At time $t = 1/k_2$, the change in steady state temperature will be given by

$$\Delta T_{ss}\left(\frac{1}{k_2}\right) = \Delta T_{ss}\left(1 - \frac{1}{e}\right) \approx 0.632\Delta T_{ss} \qquad \textbf{(Eq.13)}$$

Since ΔT_{ss} is known, one can simply find out at what time $0.632\Delta T_{ss}$ occurs. The value of $1/k_2$ is given by the difference between this point and the time where the temperature begins to rise.

Modeling Results

In order to obtain actual values of k_1, k_2, and τ, actual data must be obtained. Following the procedure for step 2.b and 2.c[3], the data for duty cycle and resulting steady temperature for three different voltages can be obtained. MatLab can then be used to calculate a linear trendline that fits the data. The data and appropriate plots are shown below.

[3] See Experimental Procedure section

Vref = 8.000 volts

Table 2: Steady temperature and duty cycle for Vref = 8.000 volts

Duty Cycle (%)	Steady Temperature (F)
35.2	140.01
40.0	147.20
48.4	160.01

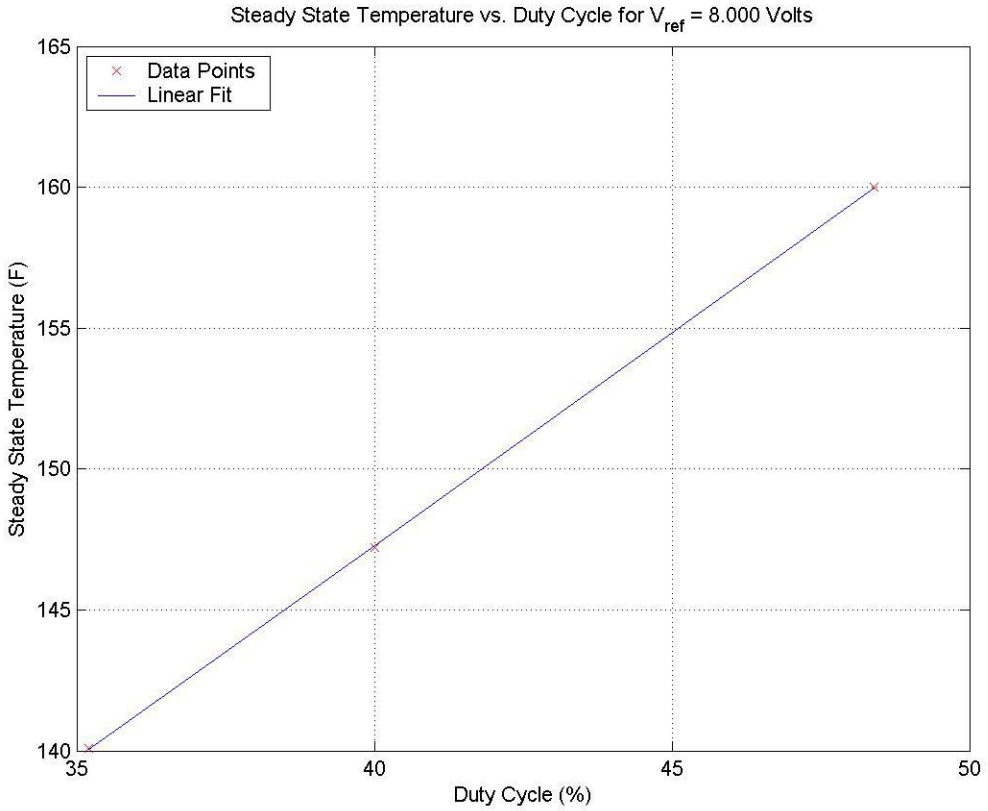


Figure 7: Steady state temperature vs. duty cycle for V_{ref} = 8.000 volts

Using MatLab to calculate slope of offset of the linear fit line yields.

Slope = 1.5102 F/%
Offset = 86.88 F

$T_{ss} = 1.5102u_o + 86.88$

Vref = 9.074 volts

Table 3: Steady temperature and duty cycle for Vref = 9.074 volts

Duty Cycle (%)	Steady Temperature (F)
26.5	140.00
31.5	150.00
37.5	160.01

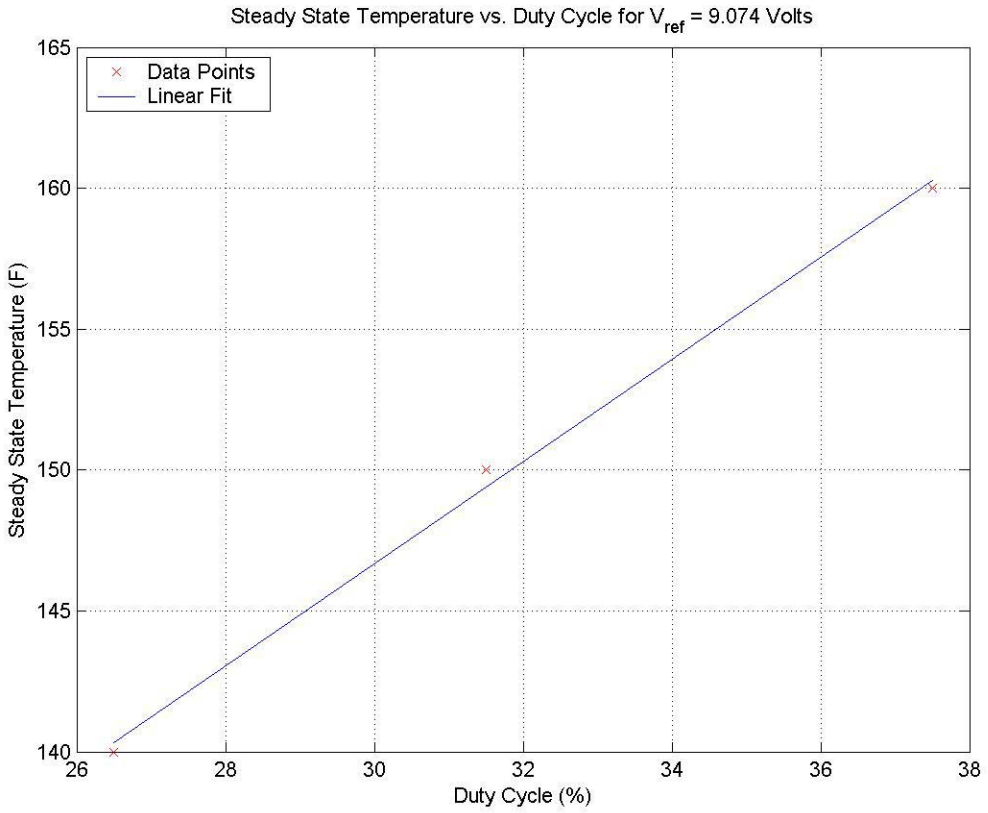


Figure 8: Steady state temperature vs. duty cycle for V_{ref} = 9.074 volts

Using MatLab to calculate slope of offset of the linear fit line yields.

Slope = 1.8141 F/%
Offset = 92.2538 F

$T_{ss} = 1.8141u_o + 92.2538$

Vref = 10.036 volts

Table 4: Steady temperature and duty cycle for Vref = 10.036 volts

Duty Cycle (%)	Steady Temperature (F)
22.0	140.00
26.1	150.00
30.2	160.00

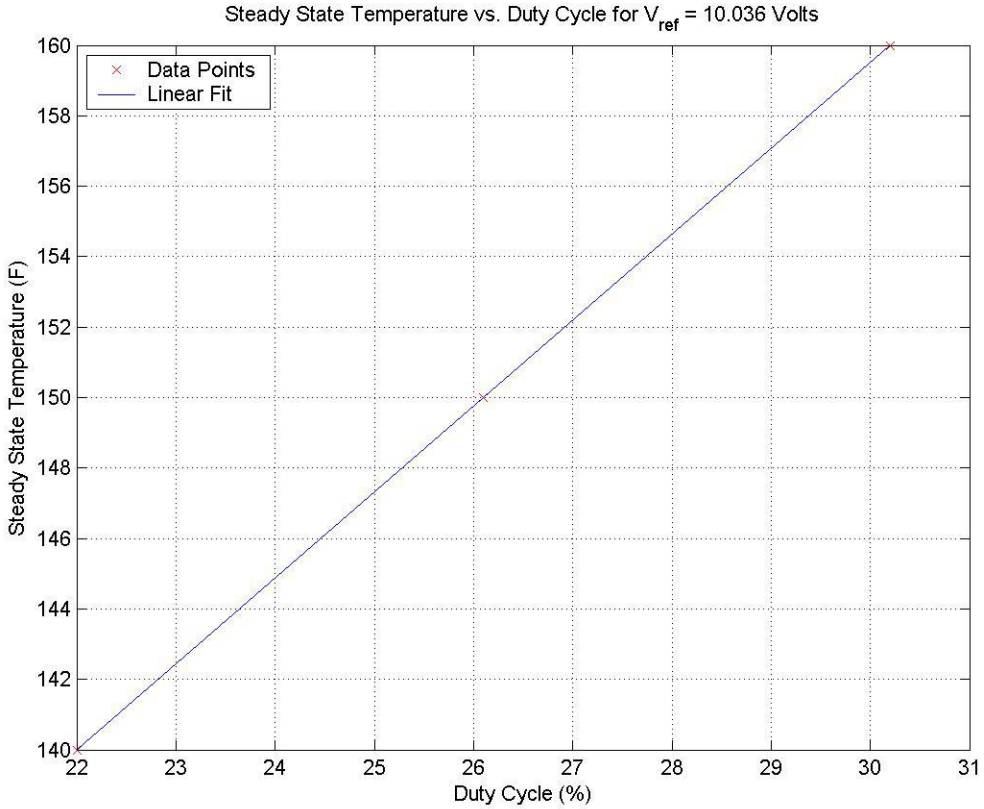


Figure 9: Steady state temperature vs. duty cycle for V_{ref} = 10.036 volts

Using MatLab to calculate slope of offset of the linear fit line yields.

Slope = 2.4390 F/%
Offset = 86.3415 F

$T_{ss} = 2.4390u_o + 86.3415$

From the above three analyses, we see that that offset of all of these are roughly constant. This makes sense since the offset corresponds to the ambient temperature (ie steady state temperature when duty cycle is 0). This shows that

the ambient temperature is roughly 86 degrees F. However, the slope seems to be very dependent on V_{ref}. This makes sense by considering Eq.7, which states that the slope should increase with the square of the reference voltage. To test this theory, a second order curve fit can be applied.

The slopes and reference voltages can be displayed in Table 5. A data point of (0,0) must also be included. This means that the slope will be zero if the reference voltage is zero.

Table 5: Slopes and reference voltages

Slope (F/%)	V_{ref} (Volts)	Comment
0.0000	0.000	Added datapoint
1.5102	8.000	Actual datapoint
1.8141	9.074	Actual datapoint
2.4390	10.036	Actual datapoint

Plotting these points and a second order curve fit in MatLab yields Figure 10.

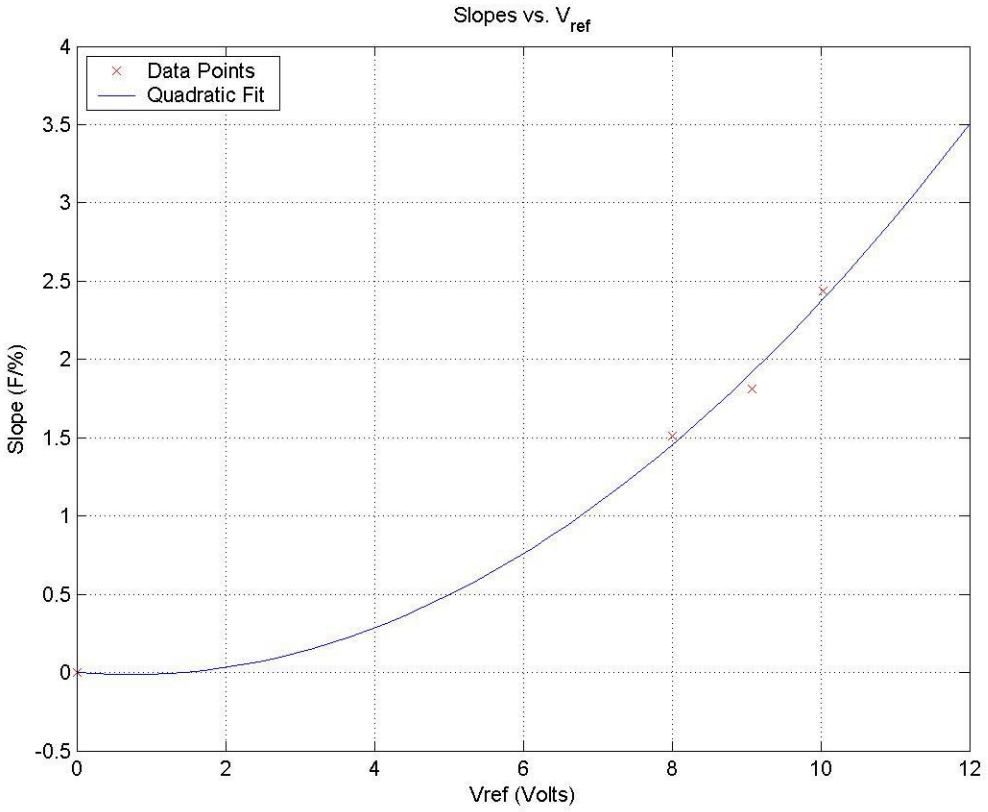


Figure 10: slope vs. V_{ref} including (0,0) datapoint

As can be seen, the second order curve fit matches the data very nicely. This shows that the slope of the T_{ss} vs. Duty cycle is dependent on the square of V_{ref} and Eq.7 is valid.

For the remainder of the lab, the reference voltage is set to roughly 8 volts as stated in the experimental procedure section. Section 2.e and 2.f are performed with $V_{ref} = 8.007$ volts.

These two sections concern obtaining the response of the system when subjected to a step input. The conditions for the step input are as follows

$u_{o\ initial} = 37\%$
$u_{o\ final} = 45\%$

Step magnitude = 8%

A plot of sensor temperature vs. time is shown below if Figure 11.

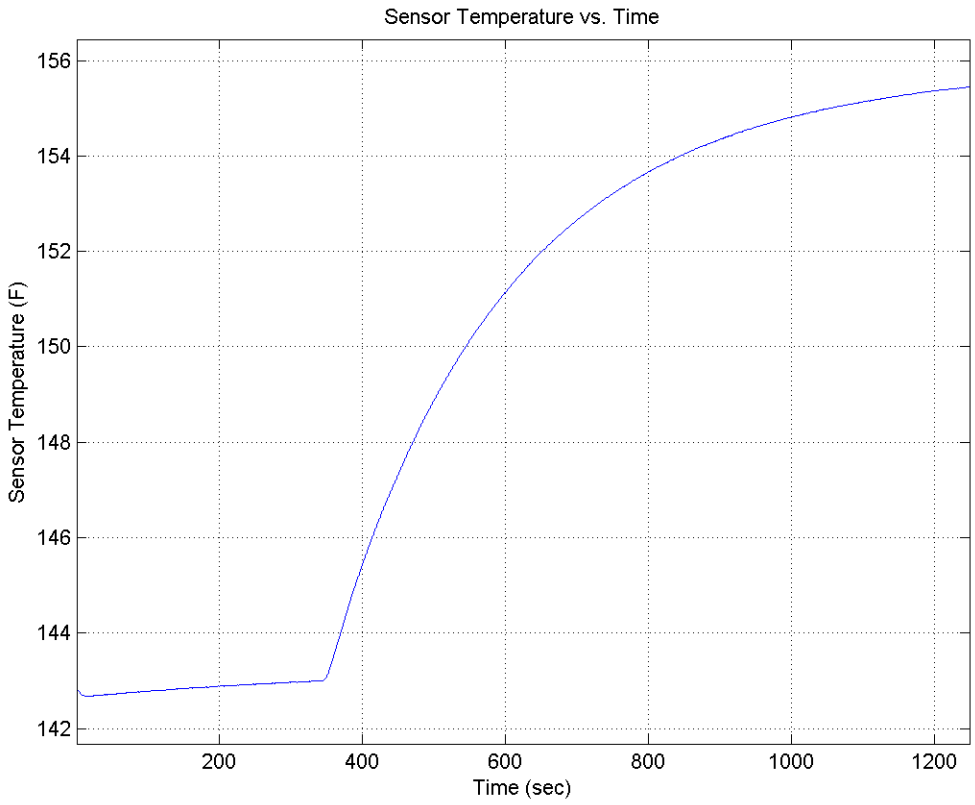


Figure 11: T_{sensor} vs. time response to a duty cycle step change of magnitude 8%

With an initial duty cycle of 37%, the block was allowed to reach a steady state temperature of roughly 143 F. Then, at t = 343.300 seconds, a step of

magnitude 8% was introduced. This caused the system to leave the initial steady state temperature and reach a new steady state temperature of 155.44 F.

As shown above in Figure 5 and Figure 6, a maximum tangent line must be drawn. In order to do this, the slope at each time must be calculated. The slope vs. time of the system after the temperature starts to rise is shown below in Figure 12.

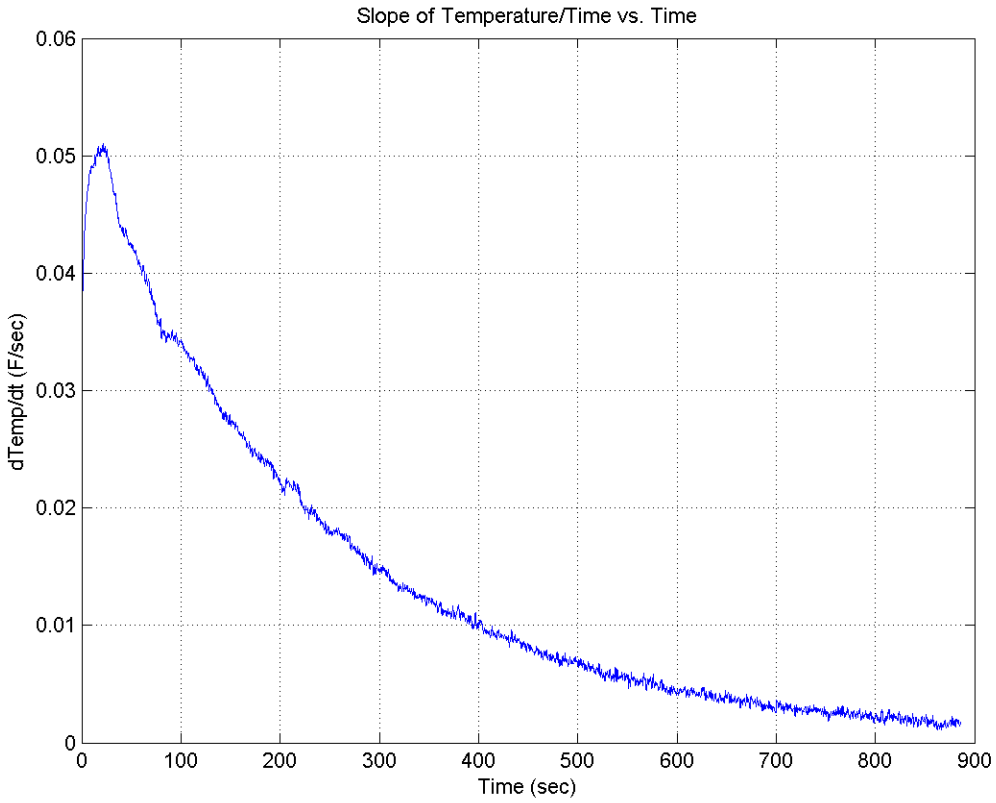


Figure 12: Slope vs. time for T_{sensor}. On this plot, t = 0 corresponds to when the temperature begins to rise.

As can be seen, there is some noise in Figure 12. Since this is due to the fact that the derivative is calculated numerically. Since the time step is so small, then the smallest noise in the temperature signal will cause large jumps in the numerical calculation of the derivative. In order to combat this, Figure 12 was generated from two temperature readings that were 15 seconds apart.

The data from Figure xx can then be used to construct a maximum tangent line. This yields Figure 13 (Figure 6 using real data).

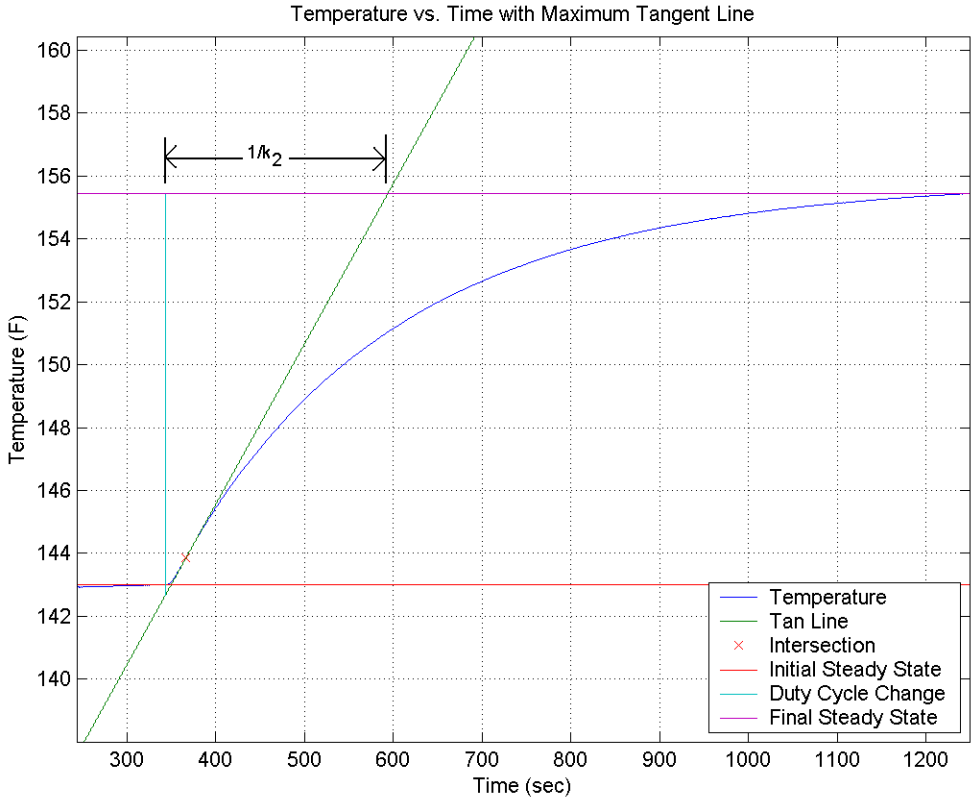


Figure 13: Temperature vs. time including maximum tangent line.

From this graph, a value of k_2 may be obtained using either the graphical method as shown in Figure A or the mathematical method described in Eq.13.

$k_2 = 0.004098$ (obtained graphically)
$k_2 = 0.004161$ (obtained mathematically)

As can seen, both of the methods provide very similar results. Both of these methods have advantages and disadvantages. The mathematical model assumes that the system will respond in a way that can be modeled with a first order system. If this is not the case, then the accuracy will suffer. Likewise, the graphical method is highly dependent on the maximum slope that is calculated. The derivative is calculated numerically and thus, the actual slope is difficult to obtain. This could also lead to inaccuracy in the calculation of k_2. However, the error using the mathematical method is probably less than the graphical technique, so its value of k_2 will be used in the rest of the report.

A value of τ can be calculated by zooming in Figure 13 where the temperature begins to rise. This yields Figure 14.

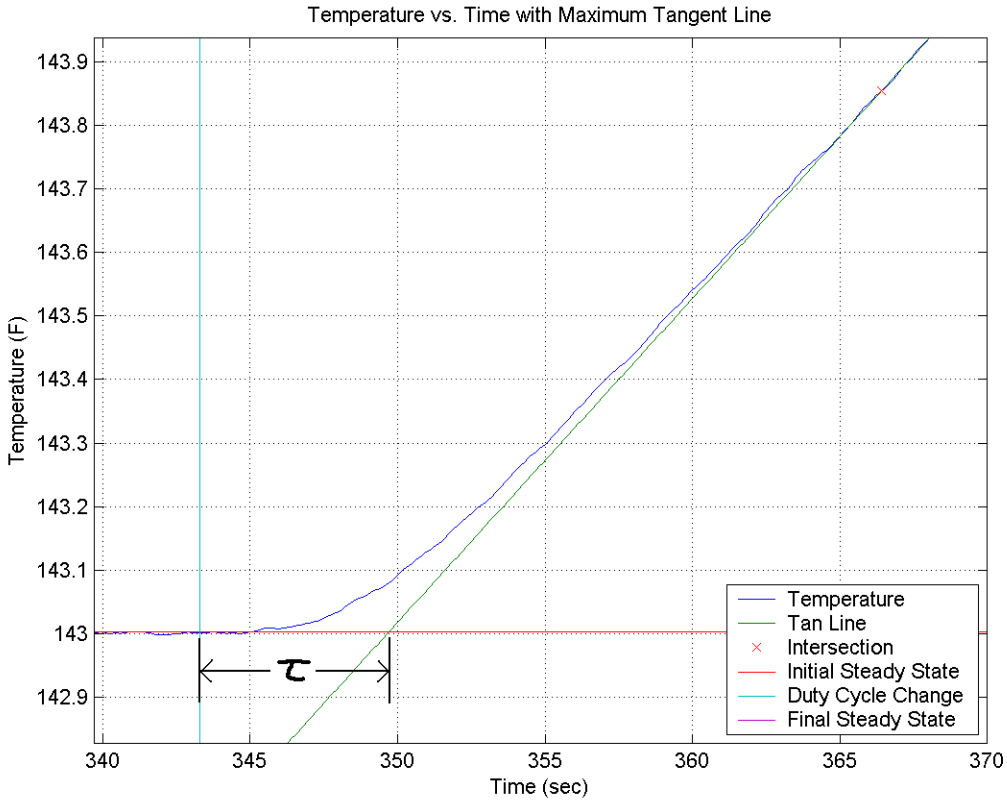


Figure 14: Close up view of T_{sensor} vs. time when temperature begins to rise

A value of τ can easily be determined from Figure 14.

τ = 6.3988 seconds

A value of k_1 can be obtained from Figure 7 and Eq.7. Although the V_{ref} used to generate Figure 7 and Figure 14 was not the same, the difference was very small. Therefore, the equation for the slope can be solved for a value of k_1 since k_2 , V_{ref}, and R (R = 25 ohms) are known. This yields

k_1 = 0.002455

All of the system parameters are now derived for the model. Naturally, these will not yield a perfect fit to the data. Another way to obtain values of k_1, k_2, and τ would be to use least squared optimization. MatLab can be used to simulate the response of the system shown in Figure xx. (a 4^{th} order Pade approximation is used for the transport delay). The parameters k_1, k_2, and τ can be varied systematically and the least squared error can be calculated each set of values of k_1, k_2, and τ. The set of values that yields the least error can be used for the simulations.

This procedure yields the following values for k_1, k_2, and τ.

k_1 = 0.002494
k_2 = 0.003964
τ = 4.4792 seconds

The response of the system using the two sets of parameters can be plotted using the step function in MatLab (with a 4th order Pade approximation for the transport delay). This yields Figure15.

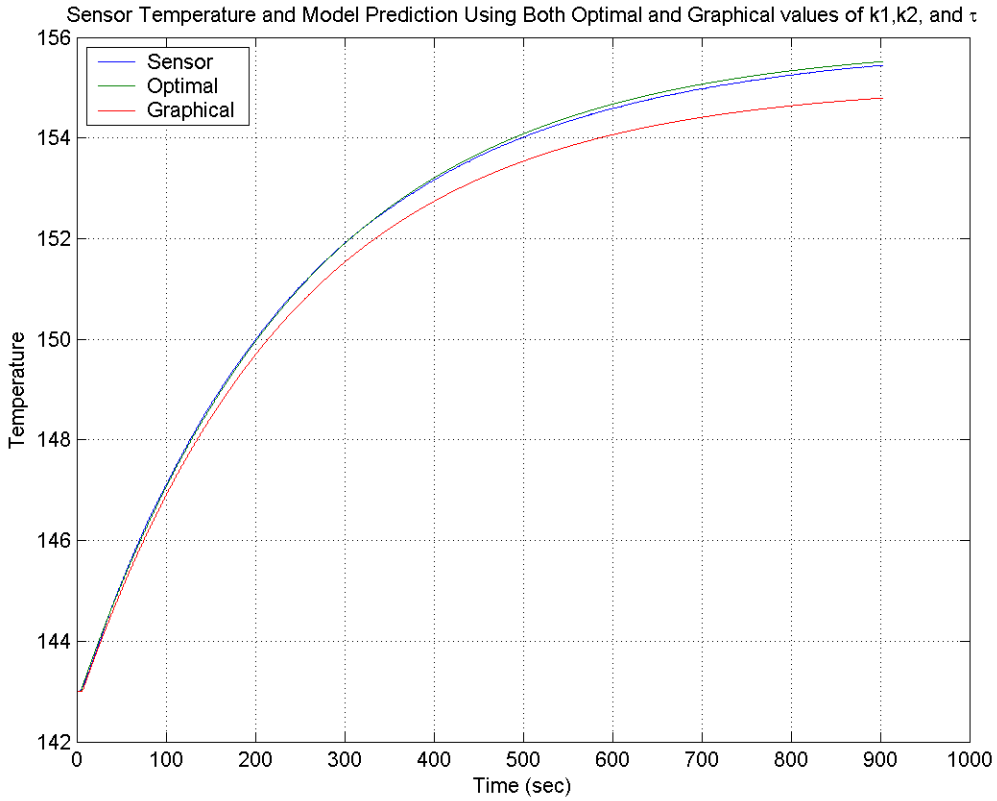


Figure 15: T_{sensor} vs. time and model using different sets of k_1, k_2, and τ

As can be seen from Figure 15, the model of our system using both sets of parameters both follow the actual sensor temperature fairly well. At a first glance, it appears that the values obtained using least squared optimization follows the actual temperature better. However, it turns out that the parameters derived using graphical techniques is actually superior. The reason why Figure 15 may be misleading is obvious when the response of the system during the time lag is focused on as shown below in Figure 16.

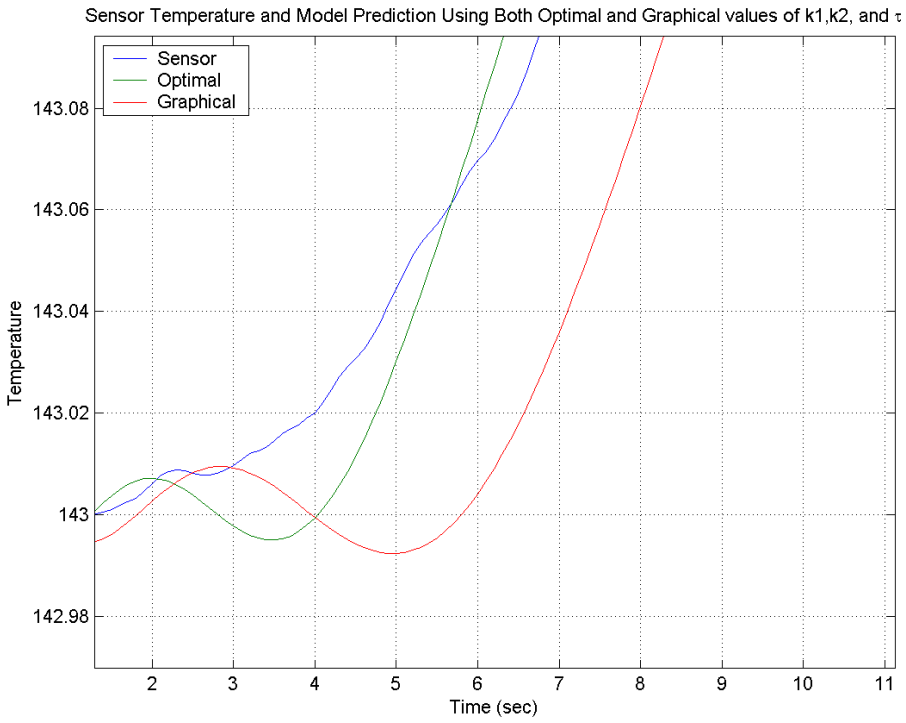


Figure 16: Closeup view of Pade approximation for time delay

As can be seen, when using MatLab (not Simulink) to calculate the response of the system due to a step change in duty cycle, a 4th order Pade approximation is used for the time delay. This type of delay is a series of sin waves that keeps the average value at roughly 0 for the duration of the delay. However, in the real system and in Simulink, a pure time delay is used, not an approximation. Therefore, in order to determine which set of parameters are better, they must be used with Simulink with a step response in duty cycle and see which matches better with the actual response. The parameters are summarized below in Table 5

Table 5: Steady temperature and duty cycle for Vref = 10.036 volts

	Graphical Derivation	**Least-Squared Optimization**	**Percent Difference**
$\mathbf{k_1}$	0.002455	0.002494	-1.589 %
$\mathbf{k_2}$	0.004161	0.003964	4.734 %
$\boldsymbol{\tau}$	6.3988 sec	4.4792 sec	30.000 %

As can be seen, both methods provide roughly the same k_1 and k_2 value (with in 5% of each other). However, the value of τ is different. As will be seen later in the Laboratory Discussion section, the parameters derived using graphical means matches the response better, so these values will be used.

V. Control System Design

The procedure of picking appropriate K_p, K_i, and K_d values is fairly involved. Since there are three variables that may be simultaneously varied, the problem requires logical procedures to pick these parameters. For this lab, three different methods of choosing combinations of gains were used.

1. Ziegler Nichols Oscillation Method (designed online)
2. Ziegler Nichols Model-Based Method (designed offline)
3. Root Locus Method (designed offline)

1. Ziegler Nichols Oscillation Method (designed online)

The Ziegler Nichols Oscillation method uses the controller critical gain (K_c) and the period of oscillation (P_c) to determine the PID control gains. Initially, the proportional was set at a low value of 10. The proportional gain was then increased until the system became neutrally stable. The proportional gain and period of oscillation were recorded and are shown below

K_c = 73.5
P_c = 22 seconds

Using these two values, two additional variables were defined at this time and are shown below.

$$T_r = \frac{P_c}{2} \qquad \textit{(Eq.14)}$$

$$T_d = \frac{P_c}{8} \qquad \textit{(Eq.15)}$$

The PID gains were adjusted according to the following relations.

$$K_p = 0.6K_c \qquad \textit{(Eq.16)}$$

$$K_i = \frac{K_p}{T_r} \qquad \textit{(Eq.17)}$$

$$K_D = K_p T_d \qquad \textit{(Eq.18)}$$

The values calculated from these relationships are shown below

K_p = 44.1
K_i = 4.01
K_d = 121.28

2. Ziegler Nichols Model-Based (designed offline)

The Ziegler Nichols Model-Based method uses the physical model parameters (k_1, k_2, and τ) obtained from part 1 (first week) of the lab. This method defines two variables which determine the PID gains required.

$$K_o = \frac{k_1}{k_2}\frac{V_{ref}^{\ 2}}{R} \qquad \textit{(Eq.19)}$$

$$\upsilon_o = \frac{1}{k_2} \qquad \textit{(Eq.20)}$$

These constants are then used to choose the PID gains using the following relations.

$$K_p = \frac{1.2\upsilon_o}{K_o\tau} \qquad \textit{(Eq.21)}$$

$$K_i = \frac{K_p}{2\tau} \qquad \textit{(Eq.22)}$$

$$K_d = \frac{K_p\tau}{2} \qquad \textit{(Eq.23)}$$

The values calculated from these relationships are shown below[4]

K_p = 29.84
K_i = 2.33
K_d = 95.47

[4] These are the correct gains to use. However, in lab, the gain values were derived using slightly inaccurate values of k_1, k_2, and τ. The data was acquired using K_p = 23.61, K_i = 1.86, and K_d = 74.76.

3. Root Locus Method (designed offline)

The Root-Locus method uses the system dynamic model obtained previously in the form:

$$\Delta Tss(s) = \frac{k1}{s+k2}\frac{Vref^2}{(s+k2)s}e^{-\varpi}u(s) \qquad \textbf{(Eq.24)}$$

In conjunction with a PID controller in the form

$$C(s) = K_p + sK_d + \frac{K_i}{s} \qquad \textbf{(Eq.25)}$$

With the PID controller and plant model (dynamic model) the closed-loop characteristic equation takes the form:

$$1 + k1(\frac{Vref^2}{R}\frac{Kp*s + Ki + Kd*s^2}{(s+k2)s})e^{-\varpi} = 0 \qquad \textbf{(Eq.26)}$$

This design method changes the location of the roots of the characteristic equation by varying K_p, K_d, and K_i values in order to achieve good stability and performance. It is very difficult to solve all 3 gains simultaneously, and so MatLab's SISO tool was used to design the PID gains. This requires building a Simulink model of the system (see Model validation section for Simulink Model).

First, switch 4 is set to the ground position in order to ensure that the only system between the input and output points are only the plant. Then, the LTI viewer is used to obtain the linearized system (using a 4th order Pade approximation for the time delay). This is achieved from the Simulink model

Tools
Linear Analysis

The LTI viewer can then be used to introduce a small perturbation at the input point (duty cycle) and measure the response at the output point (temperature) using

Simulink
Get Linear System

This yields the following figure.

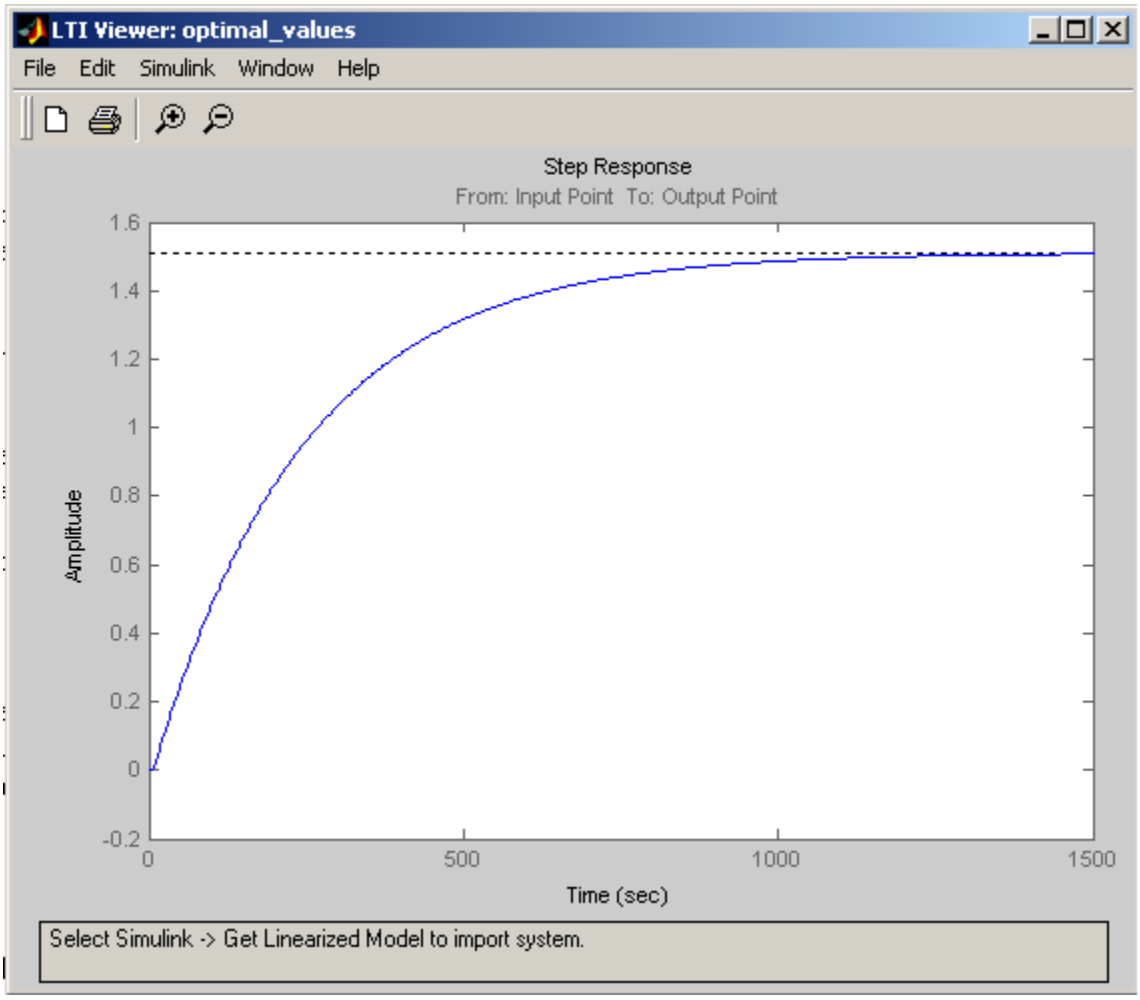


Figure 17: Linear system of only plant using LTI viewer

This system matches very well with Figure xx where T_{sensor} vs. time was plotted for a step change in duty cycle. This shows that the LTI view is actually applying a perturbation in duty cycle. Another interesting observation to make is that the DC gain of Eq.7 to a unit step input (ie $u_o = 1$) is 1.5102. This corresponds exactly with the final value shown in Figure xx.

This linear model of the system can then be exported to the workspace using

File
Export (pick system)
To Workspace

The SISO design tool (single input, single output) can then be used to view the root loci for this system by typing "sisotool" in the MatLab command line. The SISOtool has several feedback configurations. The configuration used for this analysis is shown in Figure 18.

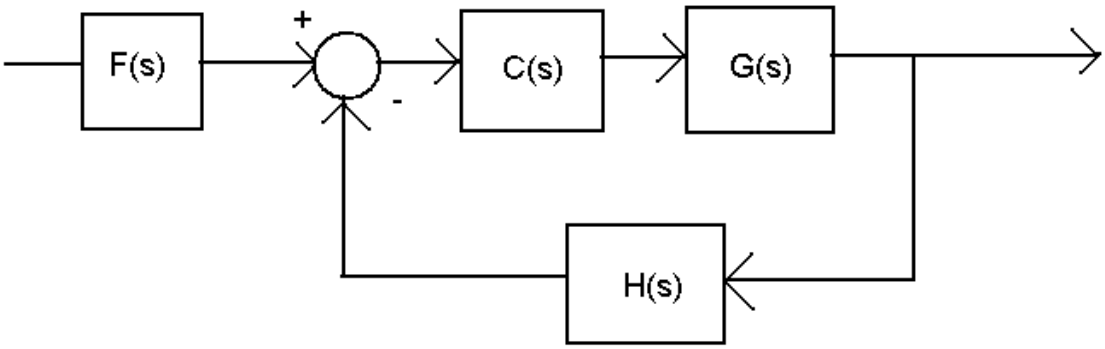


Figure 18: General block diagram configuration for SISOtool

The linear system of just the plant can be imported into the SISOtool in the G(s) location. The rest of the parameters, C, H, and F are all set to 1. This corresponds to the following block diagram.

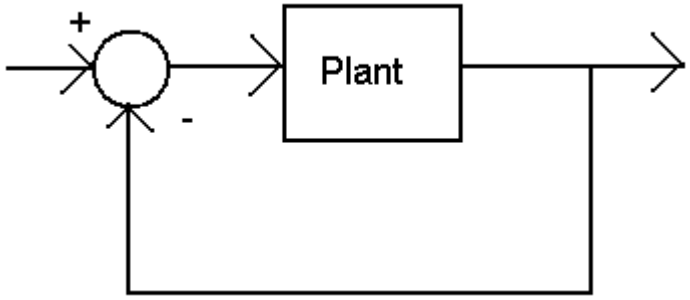


Figure 19: Block diagram of system being analyzed by SISOtool. With C(s) = H(s) = F(s) = 1

Where the plant is comprised of three blocks in series

1. A constant gain of magnitude $Vref^2/R$
2. A continuous transfer function given by $k_1/(s+k_2)$
3. A 4th order Pade approximation time delay

The root loci for this system can be derived using the SISOtool and is shown below in Figure 20.

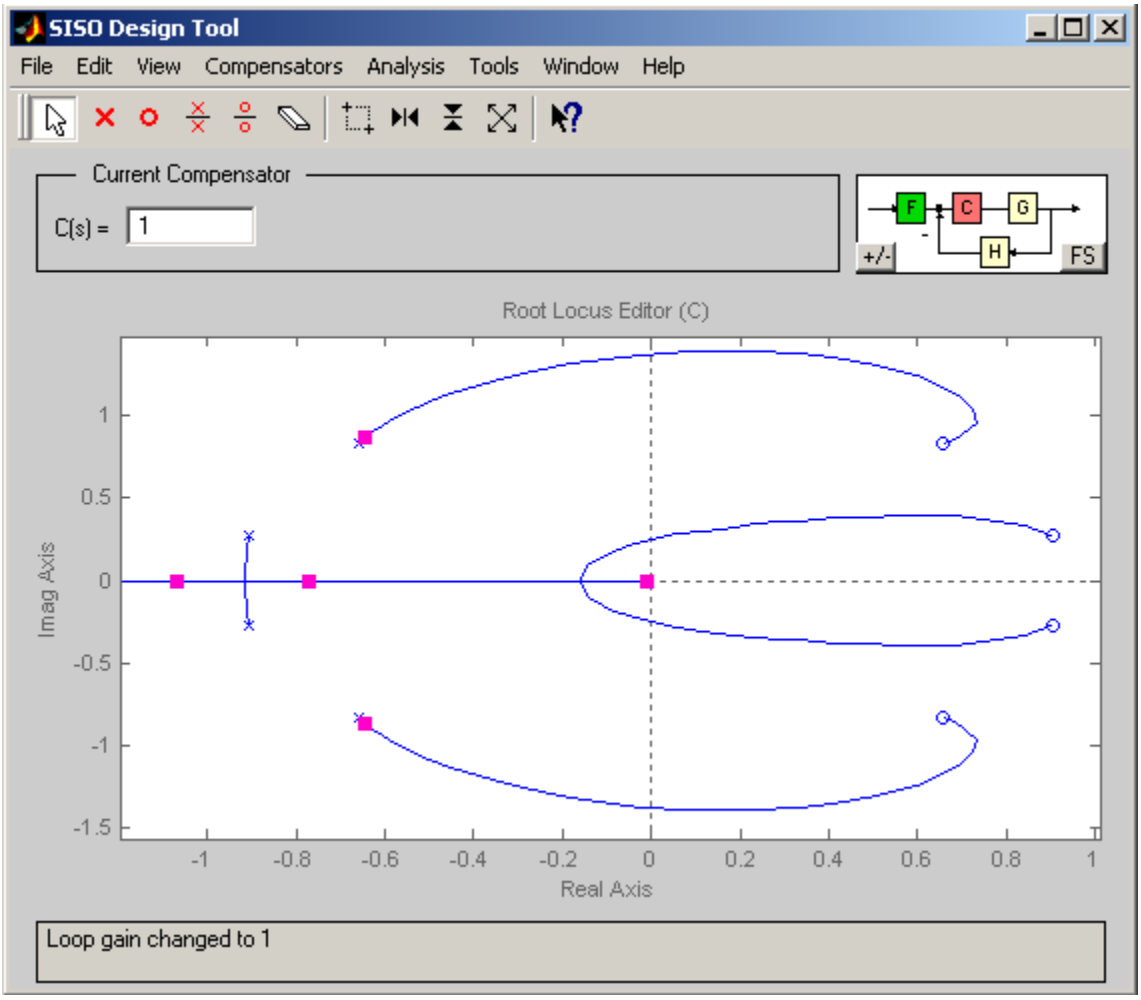


Figure 20: Root loci for block diagram shown in Figure xx

As can be seen from the above diagram, there are 5 poles and 4 zeros. Since a 4th order Pade approximation was used for the time delay, there are 4 poles and 4 zeros from the time delay. Also, the plant has a pole at $s = -k_2$, which accounts for the one pole very close to the origin.

Changing the value of C(s) in the upper left to anything other than 1 corresponds to adding a constant gain in front of the plant (G). This corresponds to a pure proportional controller and the value of C(s) is effectively K_p. In order to also simulate the derivative term, a zero on the real axis is added. This changes the root loci as shown below in Figure 21.

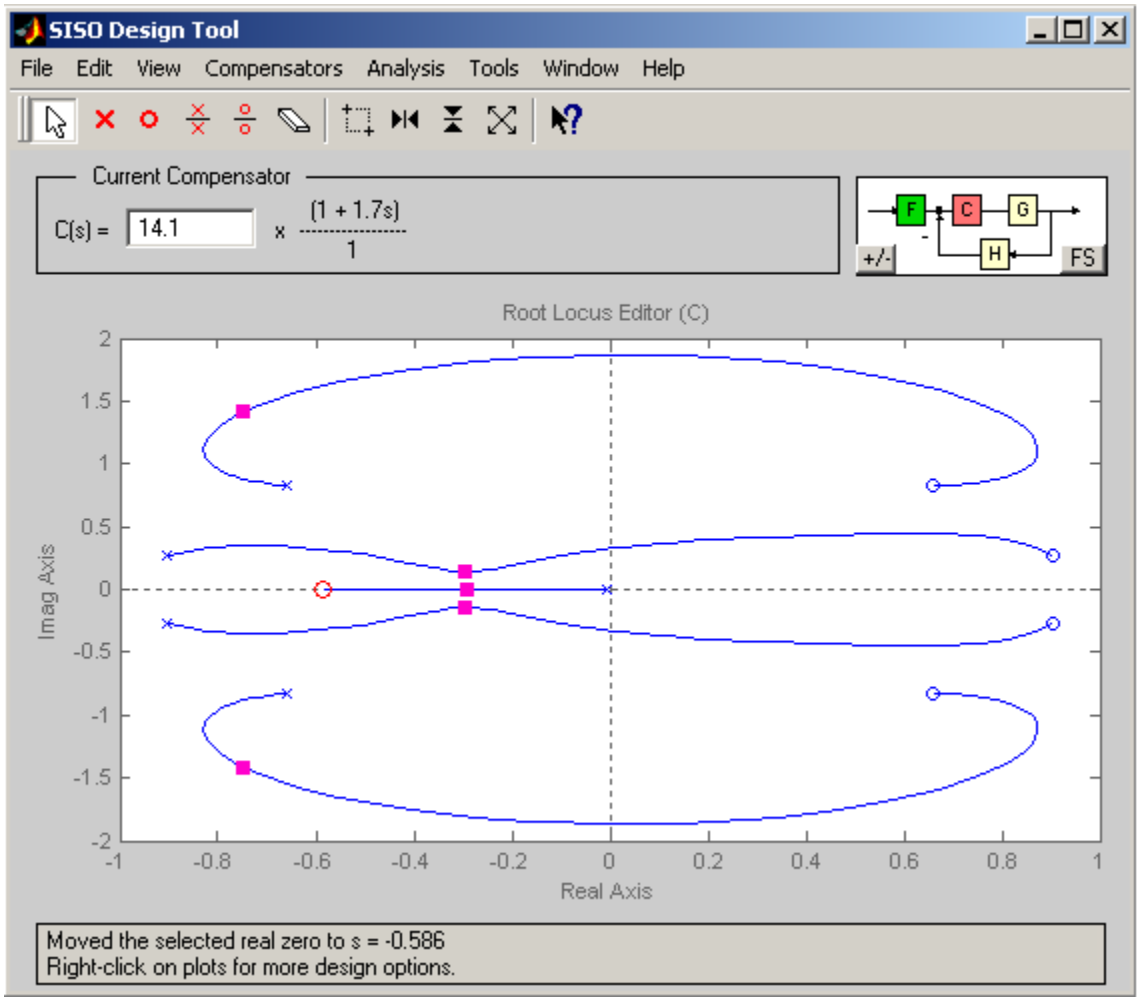


Figure 21: Root loci of plant with proportional and derivative compensator

As can be seen, the addition of the zero changes the root loci. The location of the roots can be changed by varying the parameters. The compensator C(s) is shown in the upper left corner and is given by

$$C(s) = K_p(1 + \alpha s) \qquad \textit{(Eq.27)}$$

This is the form of a PD controller where $K_d = K_p \alpha$. The system performance can be changed by simultaneously varying K_p and K_d. Varying the gain value in the white box varies Kp where as moving the location of the zero varies α (which varies K_d).

The root loci shown above corresponds to the following block diagram.

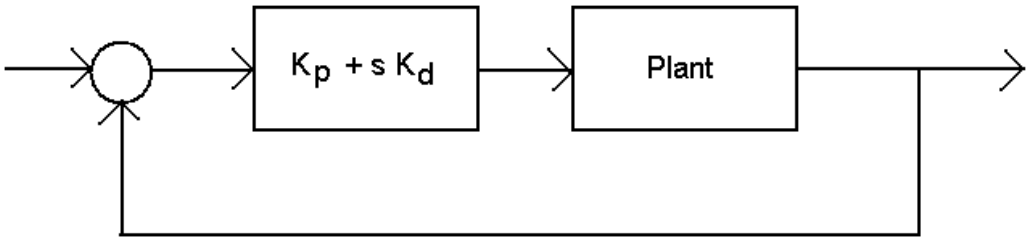


Figure 22: Block diagram corresponding to root loci

In order to obtain the best response, the roots of the system should be placed as far away from the imaginary axis as possible. This corresponds to a faster system response. The best possible values are shown above in Figure 21. This yields the following K_p and K_d values.

K_p = 14.1
K_d = 23.97

In order to incorporate the integral portion of the controller, the values of K_p and K_d calculated above are entered into the Simulink model (with $K_i = 0$). Then, Switch 4 is set to the up position in order to incorporate the proportional and derivative control. Then, the system is linearized again. This yields Figure 23.

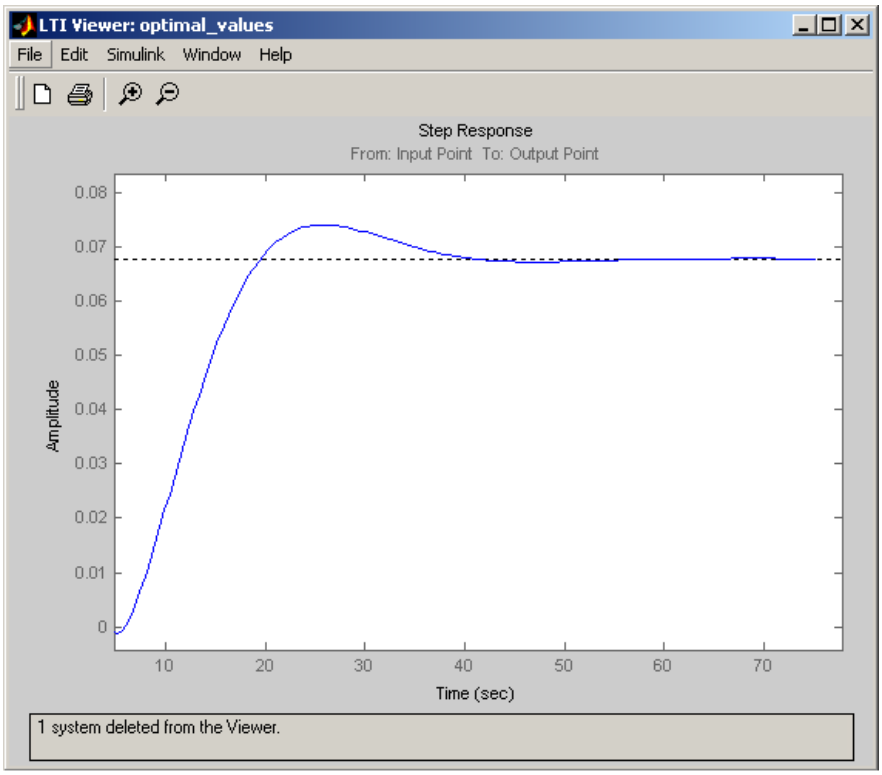


Figure 23: Step response of linearized system with proportional and derivative control

As can be seen, this is significantly different from Figure 17 since there is now proportional and derivative control. Once again, this can be imported into the SISOtool into the G(s) position. This corresponds to the following block diagram.

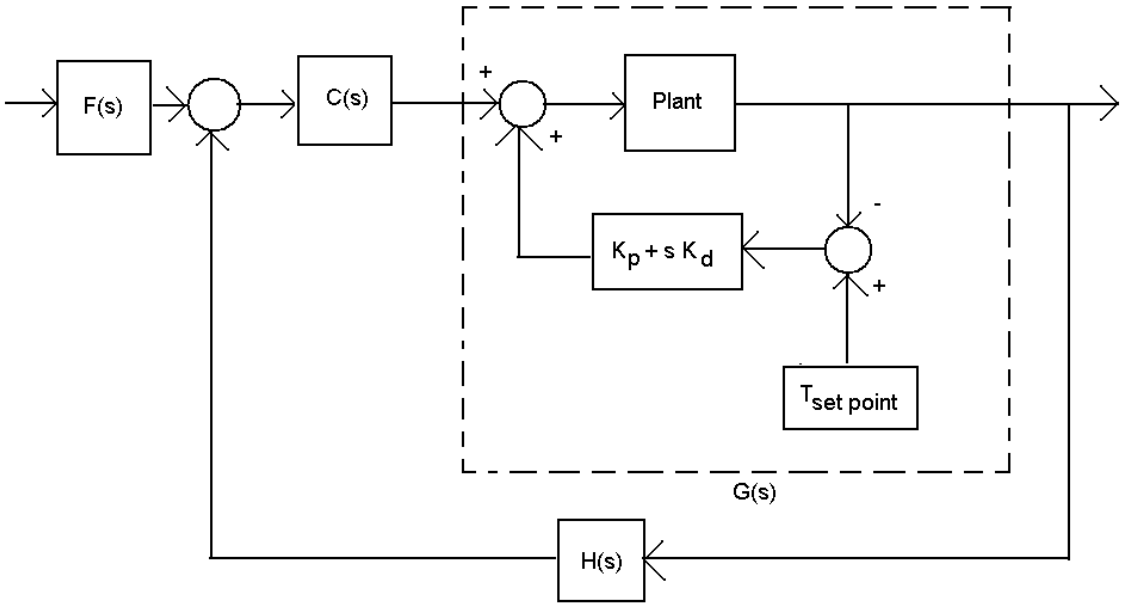


Figure 24: Block diagram of linearized system including proportional and derivative controller

Like before, the prefilter (F(s)) and the feedback gain (H(s)) are set to 1. Therefore, in order to simulate a proportional, derivative, and integral controller, the compensator C(s) needs to take the form of K_i/s. This can be achieved by placing a pole at the origin in the SISOtool design to modify the root locus. This yields Figure 25.

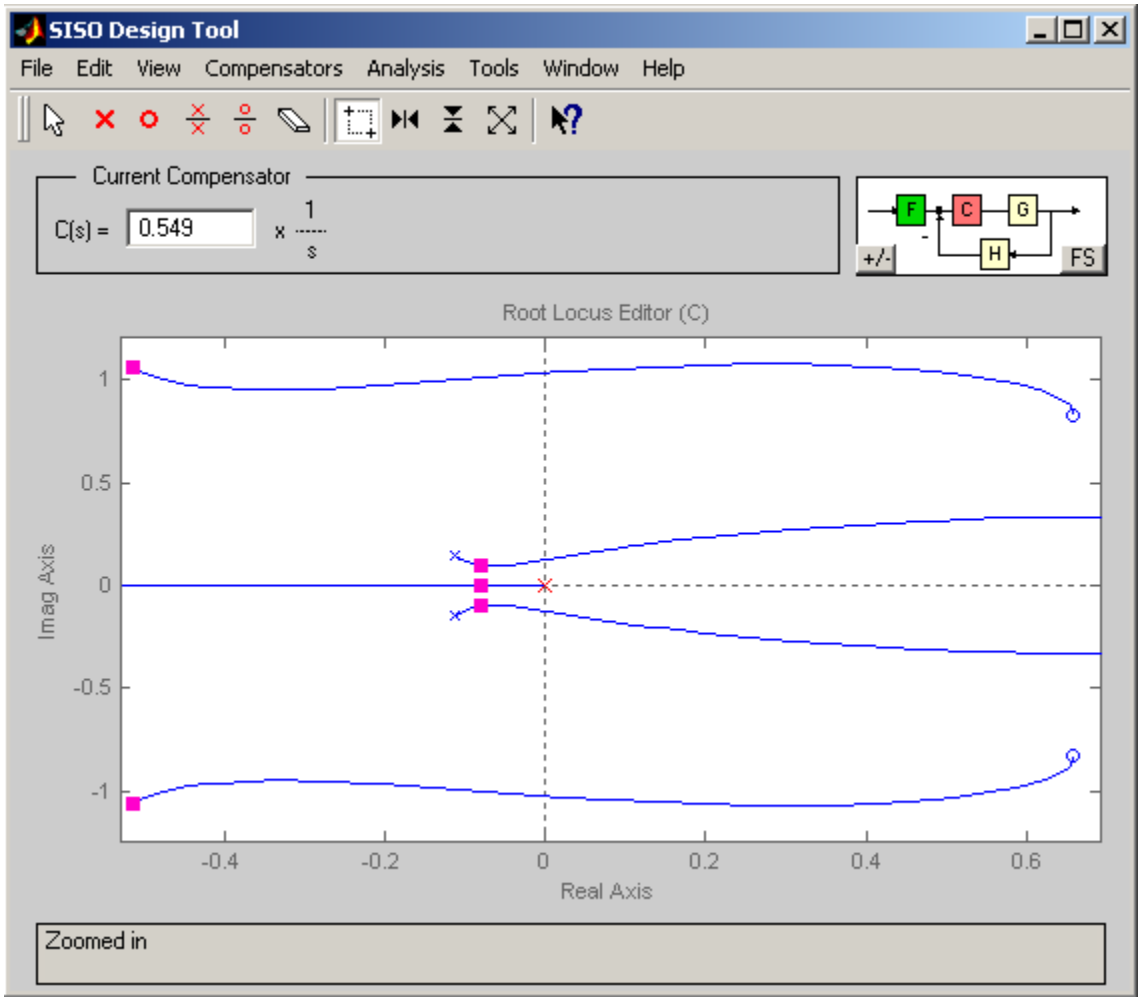


Figure 25: Root loci of plant with proportional, derivative, and integral compensator (zoomed in)

Once again, the value of K_i is then changed in order to move the poles to a desired position (ie far away from the imaginary axis or with a required damping ratio) as shown above in Figure 25. From this we see that the appropriate value of K_i to use is

$K_i = 0.549$

Therefore, the set of gains derived using the root locus method are[5]

$K_p = 14.1$
$K_d = 23.97$
$K_i = 0.549$

[5] These are the correct gains to use. However, in lab, the Simulink model used to perform the above procedure used slightly inaccurate values of k_1, k_2, and τ. Therefore, when using the root locus method, slightly different values of K_p, K_d, and K_i were derived. The data was acquired using $K_p = 13.1$. $K_d = 14.4$, and $K_i = 0.603$.

VI. Experimental Procedure

1) Equipment Set-Up

Prior to beginning the execution of this experiment, the circuit shown in Figure 26 was assembled.

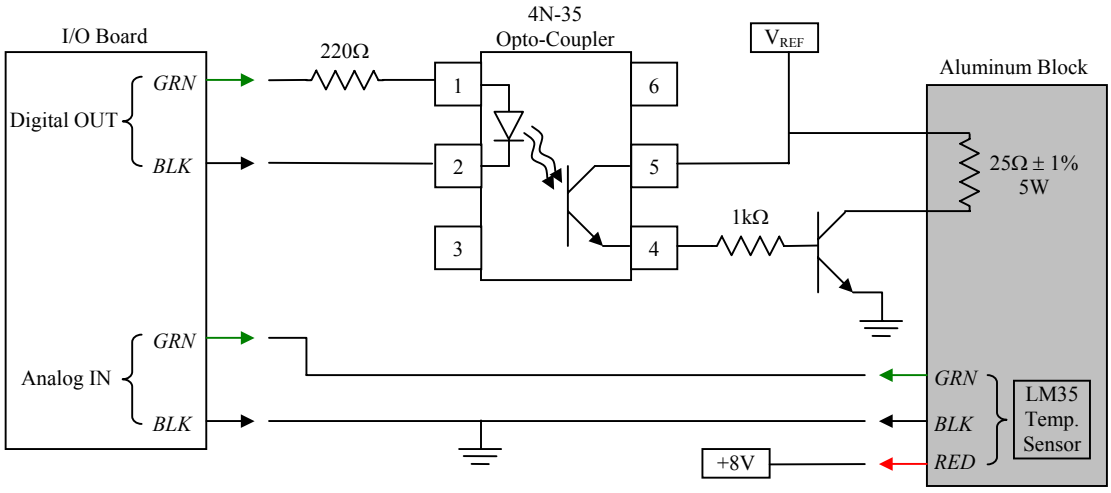


Figure 26 - Heater/Sensor Circuit Diagram

This circuit was constructed on a standard breadboard and powered using a dual source DC power supply. Except where noted, the resistors used are accurate ±5%. The provided *Temperature Control Lab* LabView virtual instrument (VI) provided the software control for the system. Lastly, the aluminum block was placed in a cardboard box to minimize the cooling effects of air currents in the laboratory.

2) System Modeling

A) Duty Cycle to Steady-State Temperature Calibration

The first portion of the experiment was conducted to obtain the information necessary to be able to construct a model the system. With the LabView VI running in the ON-OFF mode, the temperature set point was set to 60°F. This temperature was chosen arbitrarily. Any value at or below room temperature would suffice to force the controller to operate at the Lower Duty Cycle. Using a varying combination of reference voltage, V_{REF}, and Lower Duty Cycle, the circuit and controller were then turned on allowed to reach a steady-state temperature. The combinations of V_{REF}, Lower Duty Cycles, and steady state temperatures have already been displayed in Tables 2 through 4. These combinations were chosen to provide a reasonable range of steady state temperatures from 140°F to 160°F.

After the first two Lower Duty Cycle settings, the controller was placed in PID control mode with gains provided as $K_p = 60$, $K_I = 2$, $K_D = 180$. Then, with specified temperatures, the system was allowed to reach a steady-state temperature. The resulting Integral component was used as the Duty Cycle required maintaining that steady-state temperature. This was done to reduce the time required to reach steady state.

B) Temperature Response to a Duty Cycle Step Input

With the system at a steady-state temperature and duty cycle and the controller in On-Off mode, the software's data logging was enabled. This data logging feature records 11 different parameters of the controller and system. Relevant to this portion of the experiment is the time stamp, measured temperature, and the control duty cycle. A step change in the controller's duty cycle was then commanded by changing the duty cycle with the controller running.

The initial duty cycle was chosen to provide a steady-state temperature of 140 – 145°F and the stepped duty cycle chosen to yield a steady-state temperature of 155 – 160°F Once the system had reached its new steady-state temperature, the controller was stopped and the data written to file.

3) On-Off (Bang-Bang) Controller

The second portion of the experiment empirically evaluated the effectiveness and performance characteristics of an On-Off, or Bang-Bang controller. Using the same VI as before, the temperature set point was set to 150°F. Upper and lower duty cycles were set to 100% and 0%, respectively.

The controller was allowed to run in two different variations. The first variation set the hysteresis band to 1°F. This corresponded to a lower temperature limit of 149.5°F and an upper temperature limit of 150.5°F. The second variation opened the hysteresis band to 10°F yielding upper and lower temperature limits of 145°F and 155°F, respectively. The system response to these controller settings was recorded using the data logging previously mentioned.

4) PID Control

A) PID Controller Characterization

For the PID Controller Characterization, the previously used LabView VI was placed in PID mode. The objective of this portion of the experiment was to obtain an understanding for the contributions and behaviors of the different components of a PID controller.

The controller was initialized with a proportional gain, K_P, of 10, and zero integral and differential gain (K_I and K_D, respectively). Additionally, the Loop Frequency was set to 10Hz and the PWM frequency to 100Hz. Lastly, the temperature set point was set to 150°F.

The system's response to purely proportional controller was observed as K_P was increased from 10 to 70 in steps of 20. Special attention was paid to the average steady-state error and the average duty cycle. The upper limit of K_P was determined when the system was noticed to begin oscillation.

The proportional gain was then restored to 60 and an integral gain, K_I of 2 was added to the controller. Again the system was observed for its average steady-state error and average duty cycle. Once this was determined, K_I was increased to 10 and the observations repeated. K_I was then restored to a value of 2.

Lastly, the differential contribution to the control was examined by setting K_D to a value of 180. The controller was then allowed to reach a steady-state with the average steady-state error and average duty cycle being observed.

With the pieces of the controller characterized, the performance of the controller was observed by causing a temperature setting step change. This was accomplished by having the system at a temperature setting of 150°F. The system was then commanded to maintain a temperature setting of 151°F. During this step change, the controller's response and error settling behavior was observed. The process was then repeated by stepping back down to a temperature setting of 150°F.

The final characterization was performed with the system at a steady-state corresponding to a temperature setting of 150°F. The power supplied to the heater, V_{REF}, was then increased from 8V to 10V, and the system response observed. Of key interest was the behavior of the steady-state duty cycle and its comparison to the results obtained during part 2.A of the experiment.

B) Testing of the PID Controller Designed with the Ziegler-Nichols Oscillation Method

Using the Ziegler-Nichols Oscillation Method outlined in the Control System Design section gains for the PID controller were determined. These values were then input into the LabView VI's PID controller.

The system's performance was then observed through the use of the 1°F step change process described previously. With V_{REF} restored to 8V, and the PID controller running with Loop and PWM frequencies of 10Hz and 100Hz, respectively, the controller was allowed to reach a steady-state corresponding to a 150°F temperature set point. The system was then commanded to maintain a temperature setting of 151°F. The transient response and steady-state behavior were observed for average steady-state error, average duty cycle, overshoot, settling time and overall stability. The same observations were then repeated when the system was commanded back down to a temperature setting of 150°F.

C) Testing of the PID Controller Designed with the Ziegler-Nichols Model Based Method

Similar to the previous portion of the experiment, the PID controller was provided with gains calculated using the Ziegler-Nichols Model Based Method. The system's behavior was then observed as previously described when commanded with a +1°F step change from 150°F temperature setting and then a -1°F step change from 151°F temperature setting.

D) Testing of the PID Controller Designed with the Root-Locus Method

PID controller gains calculated using the Root-Locus Method were provided to the LabView VI's PID controller. As was done in the previous two portions of the experiment, the system's response to +1°F step change and then a -1°F step change was observed.

VII. Modeling Validation

In order to validate the values of k_1, k_2, and τ, a Simulink model of the temperature control system is constructed as shown below in Figure 27.

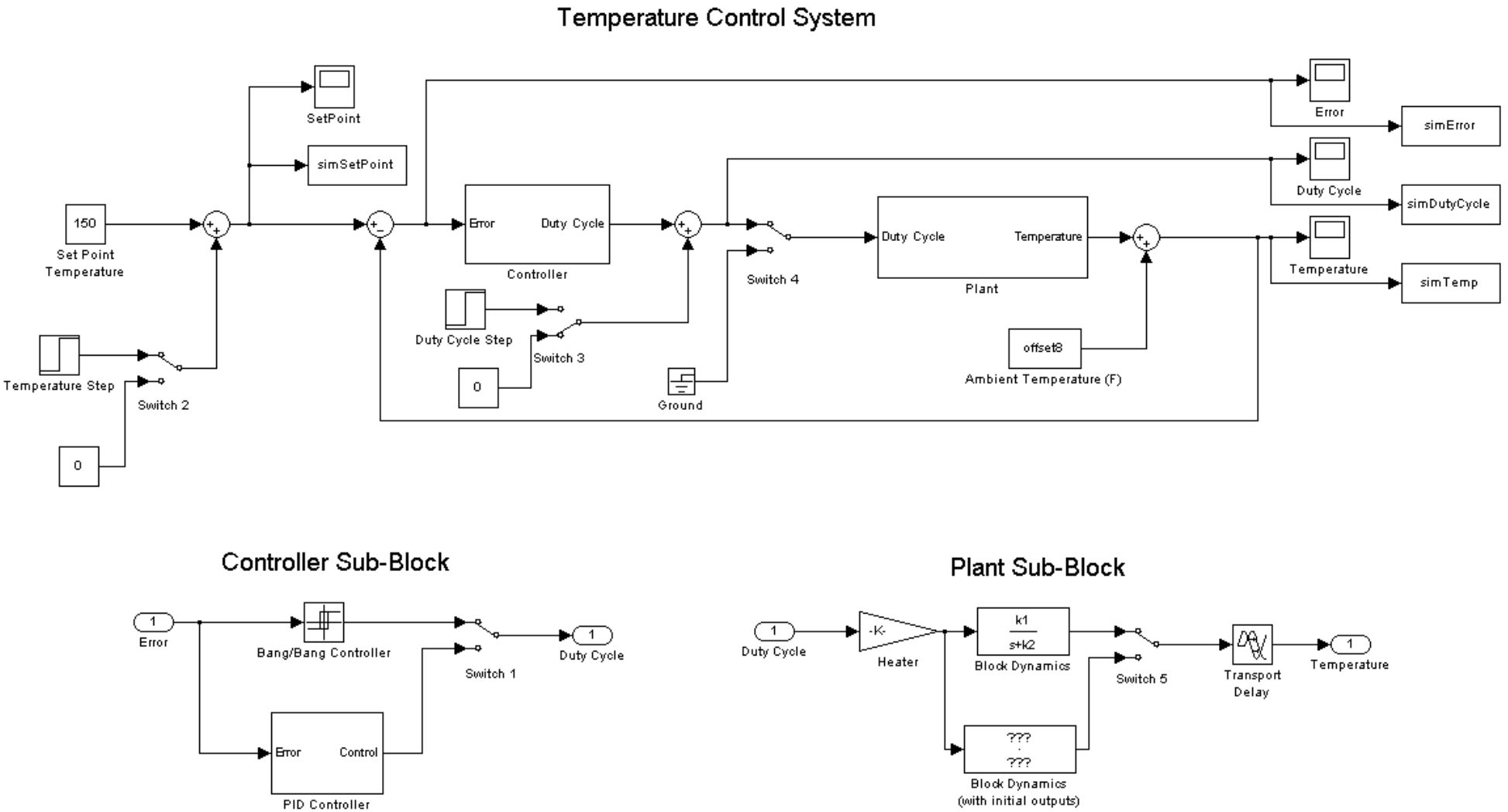


Figure 27: Simulink model of temperature control system.

This system is designed to maintain the set point temperature using either an ON/OFF (bang/bang) controller or and proportional, differential, and integral controller (PID). The PID Controller sub-system is shown below in Figure 28

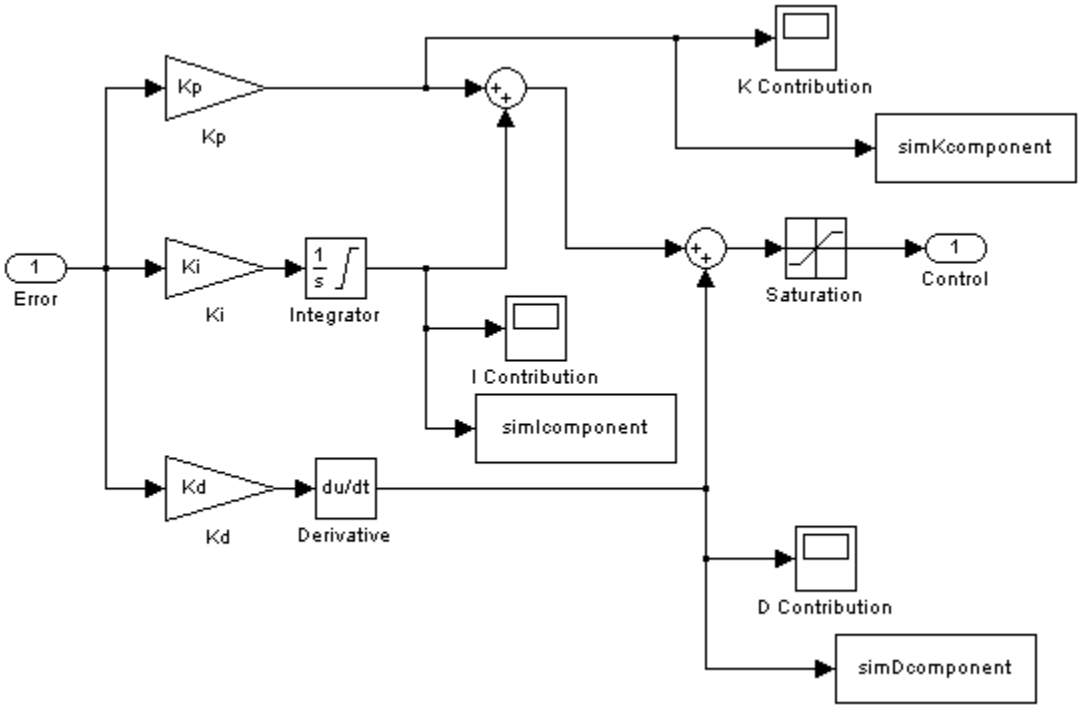


Figure 28: PID sub-system

Each block in the above figures are described below.

- Set Point Temperature - This block is a constant source with value of the desired temperature.
- Temperature Step - This block provides a 1 degree step increase in temperature at a desired time (see later section for more information).
- Duty Cycle Step - This block provides an 8% set in duty cycle at a desired time (see section 2 for more information).
- Controller - This block is the controller that outputs a duty cycle.
- Ground - This block is used to completely open the loop for linear analysis.
- Plant - This block contains a 1st order transfer function that models the dynamics of the aluminum block and heater.
- Ambient temperature - This block is a constant source with a value equal to the ambient temperature (ie steady state temperature when duty cycle is 0).
- simTemp, simDutyCycle, simSetPoint, etc. - These are output blocks which export the data to the workspace. The save format is set to StructureWithTime with a sample time of 0.1 seconds.
- Input Point/Output Point - This blocks are used for linear analysis with the SISOtool.

- Switch 1 - This determines if a bang/bang or PID controller is used.
- Switch 2 - This determines if a temperature step is introduced.
- Switch 3 - This determines if a duty cycle step is introduced.
- Switch 4 - This is used for linear analysis of the model.
- Switch 5 – This is used to change the plant to have initial conditions
- Bang/Bang Controller - This is a relay that simulates the ON/OFF controller. It has a switch on value of 0.5, and a switch off value of 0.5, which simulates a 1-degree hysteresis band. The output when on is 100 and output when off is 0.
- Heater - This is a constant gain with a value of V_{ref}^2 / R which is used to simulate the heater.
- Block Dynamics - This is the transfer function that simulates the block of aluminum dynamics. It has a value of $k_1/(s + k_2)$.
- Block Dynamics (with initial outputs) – This is the same as the Block Dynamics block except that it has an initial output temperature.
- Transport Delay - This is a block that delays the signal by a value of τ. For linearization, it has a 4th order Pade approximation.
- Integrator - This is a simple transfer function with form 1/s. This also has a saturation limit with limits of 0 to 100[6]. This is used to prevent integrator windup (ie when the error is initially large, there is a potential for the integrator to accumulate a lot of error. This in turn would make the output signal very large. Therefore a saturation limit is necessary).
- Derivative - This is a numerical derivative of the signal.
- Kp, Kd, Ki - These are all gains on their respective components.
  Saturation - This is a saturation limit with bounds of 0 to 100. This is used to simulate the fact that the duty cycle may only vary between 0 and 100 percent.

This model can now be used to simulate the system. In lab, the actual system is run using a simple ON/OFF controller. This has a set point temperature of 150 F with a 1-degree hysteresis band. This yields the following data as shown below in Figure 29.

[6] Normally, the saturation limits would be +/-100. However, after speaking with Greg Lipski, the engineer who programmed the LabView controller, it was discovered that that LabView controller had saturation limits form 0 to 100. Therefore, in order to make the Simulink model as close to the LabView controller as possible, the limits are set from 0 to 100.

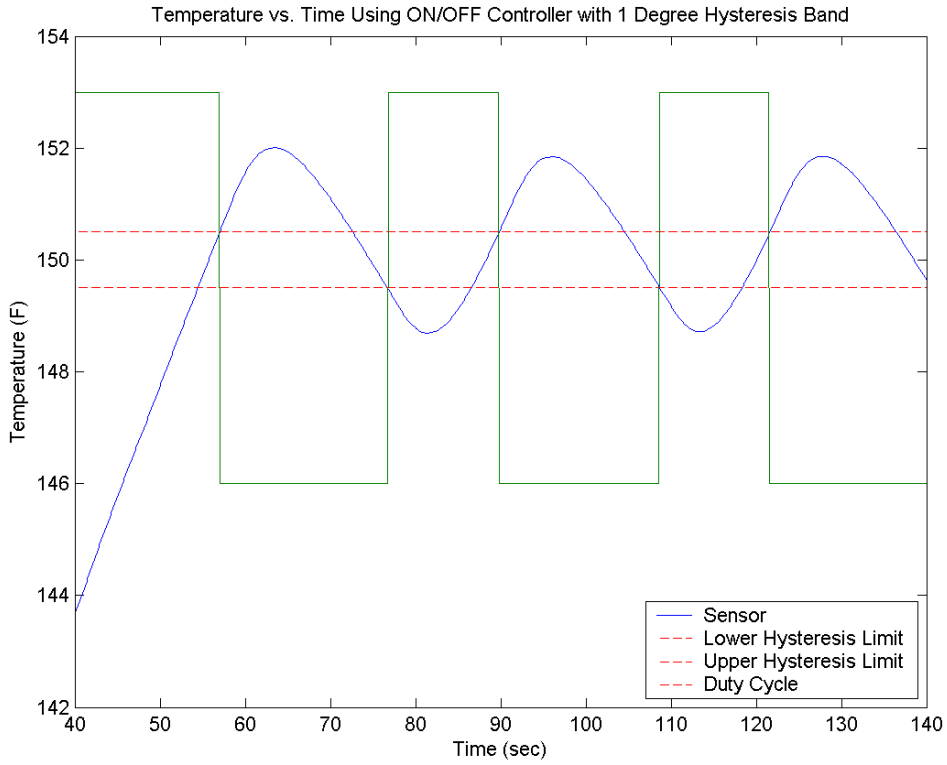


Figure 29: Actual T_{sensor} vs. time with a set point of 150 F and a 1-degree hysteresis band

As can be seen, the temperature and duty cycle behave as expected. The duty cycle remains on until the temperature exceeds the upper hysteresis limit. At this point the duty cycles goes to zero until the temperature drops below the lower hysteresis limit. The response of the system can now be compared with the output of the simulation. In a similar fashion, the simulation output is shown below in Figure 30.

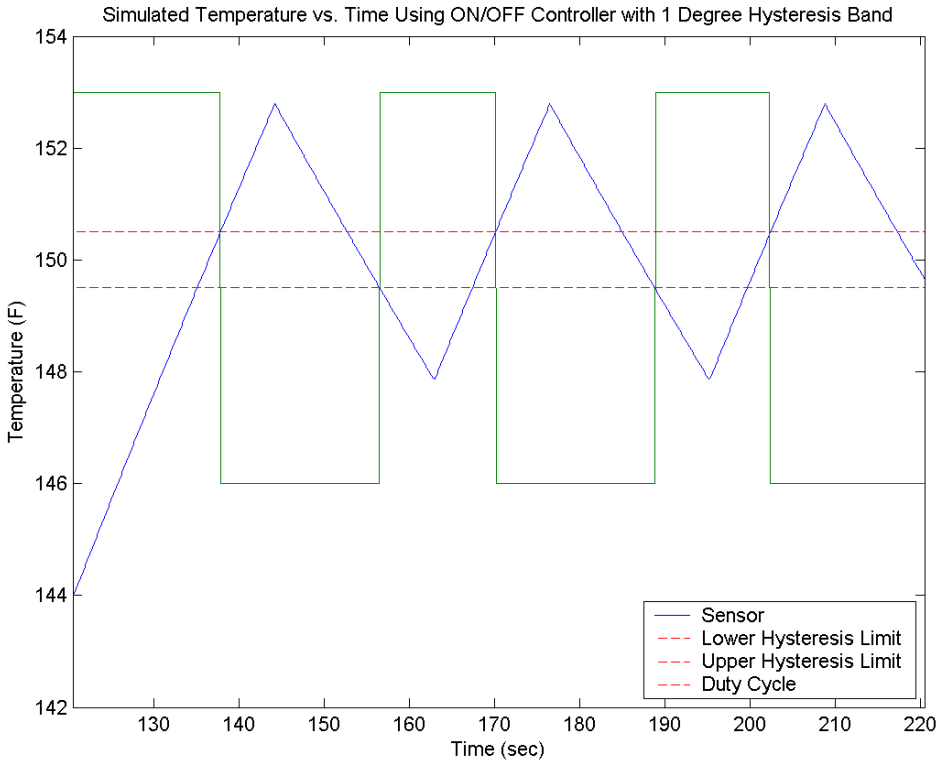


Figure 30: T_{sensor} vs. time with a set point of 150 F and a 1-degree hysteresis band from Simulink

As can be seen by comparing Figures 29 and 30, the simulation produces a very similar output to the actual system. This shows that the values of k_1, k_2, and τ are fairly accurate. The similarities can be better viewed if the temperature response from the experiment and from the simulation are shown on the same figure as shown below in Figure 31.

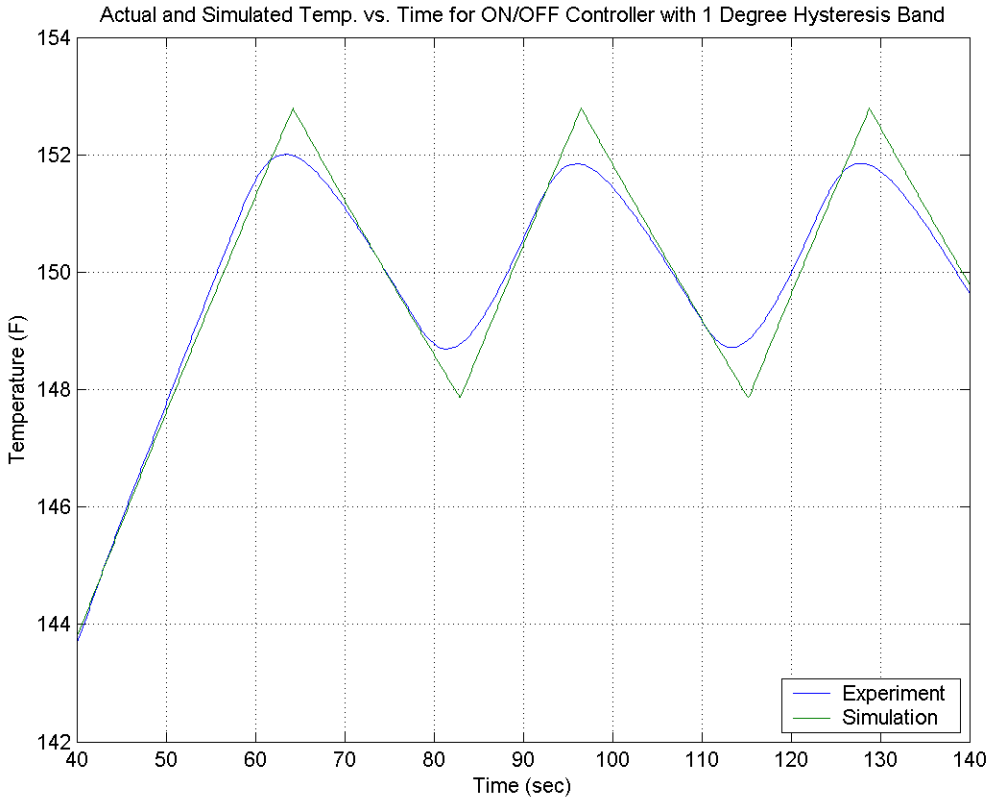


Figure 31: T_{sensor} vs. time with a set point of 150 F and a 1-degree hysteresis band from Simulink and experiement

As can be seen, the general trend and period of oscillation is similar. However, there are several noticeable differences between the two outputs. First, the response from the simulation has sharp changes in slope whereas the real system has a continuous T_{sensor}. This can be explained since the simulation is approximated with a 1st order system. Also, since the duty cycles is maxed out at 100 and 0, the response in non-linear. The other differences are shown below in Table 6.

Table 6: Difference between simulation and real experiment.

	Period (sec)	**Positive Max (F)**	**Negative Min (F)**
Experiment	34.0	152.0	148.4
Simulation	32.3	152.8	147.9

As can be seen from Table 6. The simulation matches up very well with the actual system. This shows that the parameters derived using graphical techniques work well for the bang/bang controller.

In addition to comparing the simulation with the experiment for the bang/bang controller, the simulation can also be run with a PID controller and the results compared with laboratory results. Table 7 shows that gain values used for each method.

Table 7: Gain values used to acquire data[7]

Design Method	K_P	K_I	K_D
Ziegler-Nichols Oscillation	44.100	4.009	121.275
Ziegler-Nichols Model	23.61	1.860	74.760
Root-Locus	13.100	0.603	14.400

The output of the simulation can now be compared with the output of the actual experiment. In order to do this, Switch 2 is changed to the up position. This allows a 1-degree step in temperature to be introduced after the system has reached steady state.

[7] Once again, the values used to acquire data are different from the values derived in the Controls System Design section because during lab, slightly less accurate values of k_1, k_2, and τ were available.

Gains from Root-Locus method

Starting with the values obtained with the Root-Locus method, the step response of the actual system vs. time and the step response of the simulation vs. time can be plotted on the same plot to yields Figure 32.

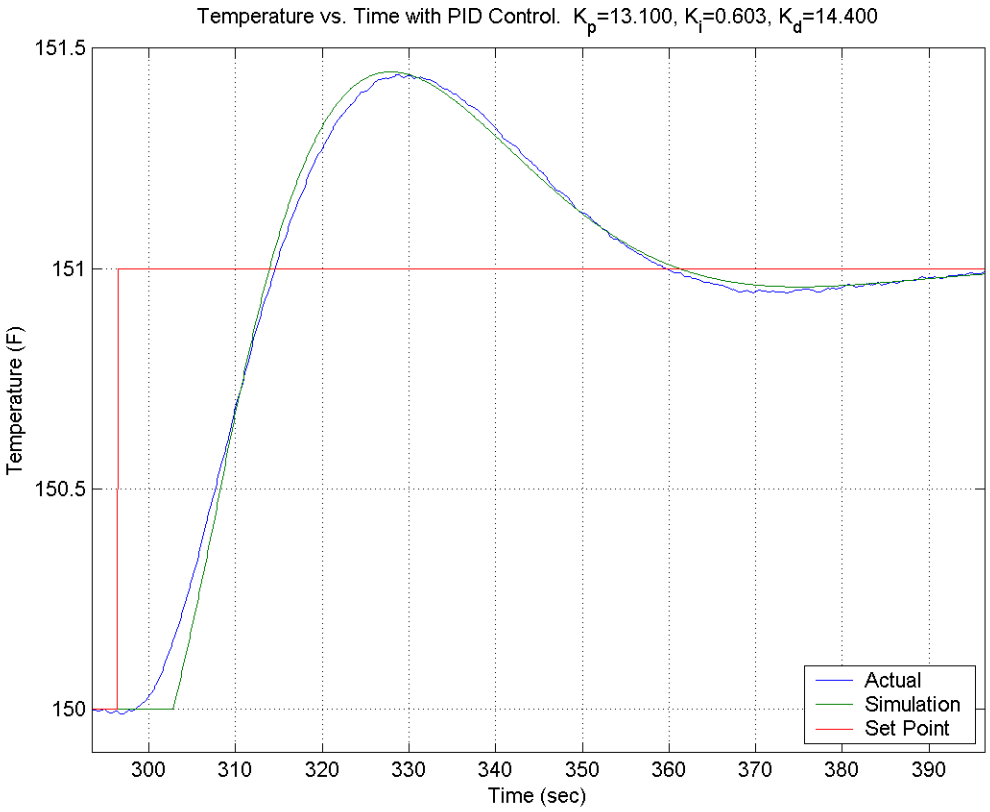


Figure 32: Temperature vs. time response to a 1-degree step input.

As can be seen from the above figure, the simulation and the actual experiment match up very nicely. This goes to further reinforce that the values of k_1, k_2, and τ were choose accurately. The process can be repeated with a different set of gains.

Gains from Ziegler-Nichols Model method

The simulation is rerun using gains derived using the Ziegler-Nichols Model based methods. The gains are shown on the top of Figure 33.

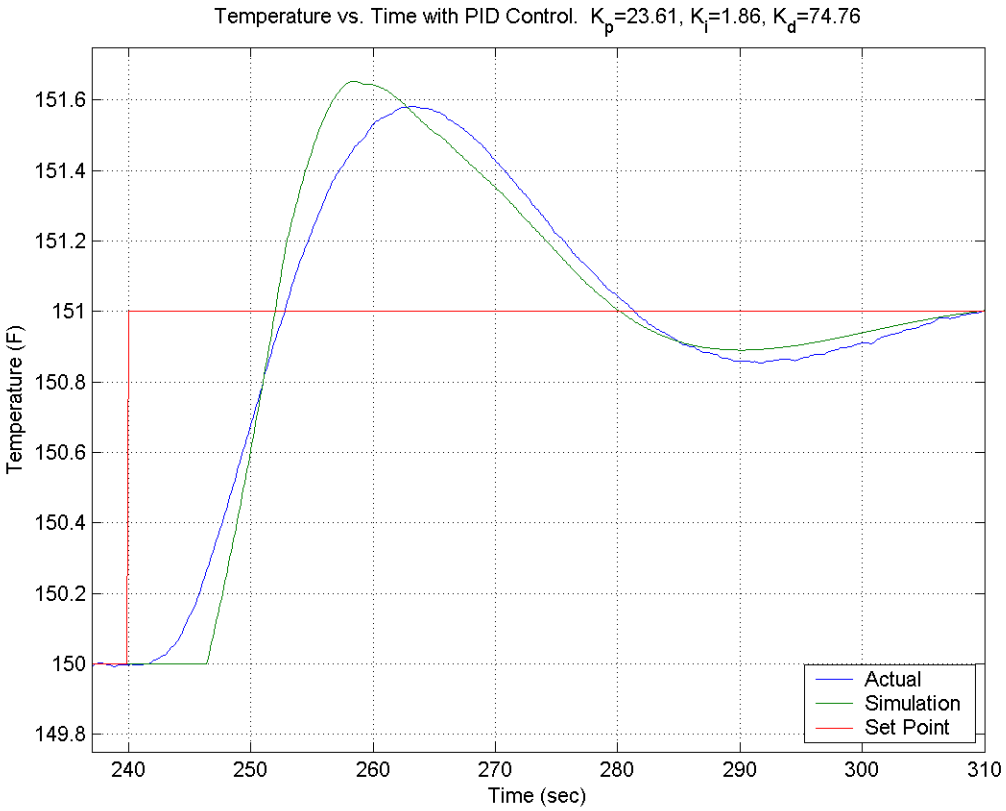


Figure 33: Temperature vs. time response to a 1-degree step input.

It appears that the model still simulations the actual system fairly well. There appears to be slightly more error using these set of gains than before. The main difference between the two sets of gains is that the derivative gain is much higher. This will probably cause problems as can be seen in the next example.

Gains from Ziegler-Nichols Oscillation method

The simulation is rerun using gains derived using the Ziegler-Nichols Oscillation based methods. The gains are shown on the top of Figure 34.

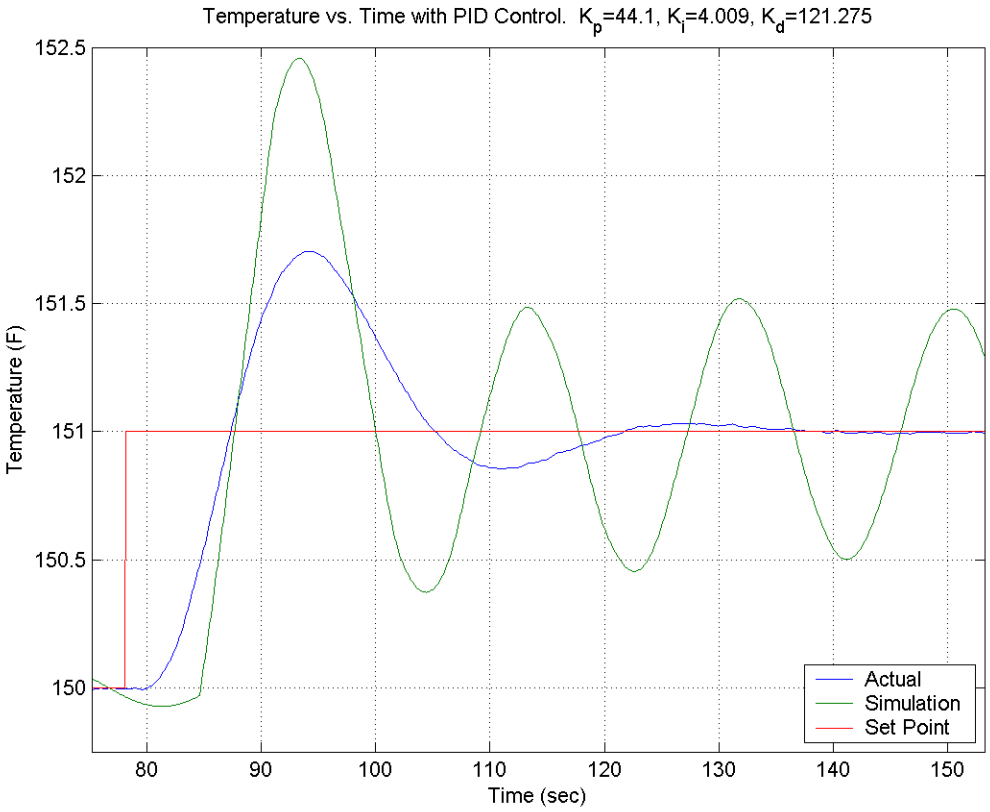


Figure 34: Temperature vs. time response to a 1-degree step input.

As can be seen from the above figure, in this situation, the simulation does not seem to follow the experiment very well. It appears that the general trend seems to follow the experiment, but there are severe oscillations in the simulation. Obviously, this is not what is happening to the actual system.

There are several possible reasons for the discrepancy. The first thing to notice is that these set of gains are much more aggressive than the previous sets. All of them are at least double the values used in the previous example. This can lead one to believe that the calculation of the derivative is a problem. Since the derivative is calculated numerically with a small Δt, this could yield large fluxuations in the derivative portion of the signal. Another difference between the simulation and the lab was that the controller in the lab contained a second order, low-pass Butterworth filter with a cut off frequency of 0.5 Hz. This was placed inside the loop on the signal for temperature. This effectively filtered out temperature fluctuations with a period of greater than 2 seconds. As can be seen

from the simulation, the frequency of oscillation of the temperature is roughly 22 seconds. This means that the filter in lab would have stopped these oscillations, however this filter was not included in lab due to the time constrains.

In general, the values of k_1, k_2, and τ derived using the graphical techniques provided a very decent model. The Simulink model is accurate for as long as the derivative gains on the PID controller are not increased beyond the values stipulated in Figure 33.

VIII. Control Design Performance

The three PID control design methods were implemented in order to develop three distinct sets of proportional, integral, and differential gains. These gains were then input into the controller and the system evaluated for a small positive and then a small negative step response.

Of interest was the system's behavior to the step inputs. For comparison four parameters were calculated. The first of these parameters is the percent overshoot, PO. This value describes amount the system overshoots the desired setting and is calculated as a percentage of the desired setting.

Secondly, the rise or fall time, t_r, is calculated as the time required for the system to first reach the new set point. However, in an oscillatory response, the system will often rise or fall past this set point one or more times.

The third parameter useful in comparing the performance of the system controllers is the settling time, t_s. The time is defined as the time it takes the system to settle to a value within a certain percentage of the desired value. For this experiment, the settling time is calculated as the time the system takes to reach and maintain the desired temperature setting within ±0.02%.

Lastly the steady state error is calculated for comparison. This parameter is defined as the remaining difference between the system temperature and the desired temperature setting once the system has reached a steady state. For the three sets of PID control gains, no steady-state error was observed with this system..

Below, Table 8 provides an overview of the PID control gains used and the system's performance when controlled with those gains.

Table 8: PID control gain sets and performance parameter summary

Design Method	Step	K_P	K_I	K_D	PO	t_r [s]	t_s [s]	Fig. #
Ziegler-Nichols Oscillation	+1°F	44.100	4.009	121.275	70.22%	9.200	50.600	35
	–1°F				71.96%	9.801	52.101	36
Ziegler-Nichols Model	+1°F	36.390	3.640	93.440	78.22%	9.780	73.681	37
	–1°F				78.22%	10.000	75.800	38
Root-Locus	+1°F	13.100	0.603	14.400	44.06%	18.200	90.301	39
	–1°F				43.62%	18.401	90.301	40

Ultimately, the overall performance of a controller is a subjective matter depending on the importance of the various performance parameters. The three sets of PID gains developed in this experiment demonstrate a trade-off in

performance parameters. A “fast” system response can be obtained at the price of a greater overshoot.

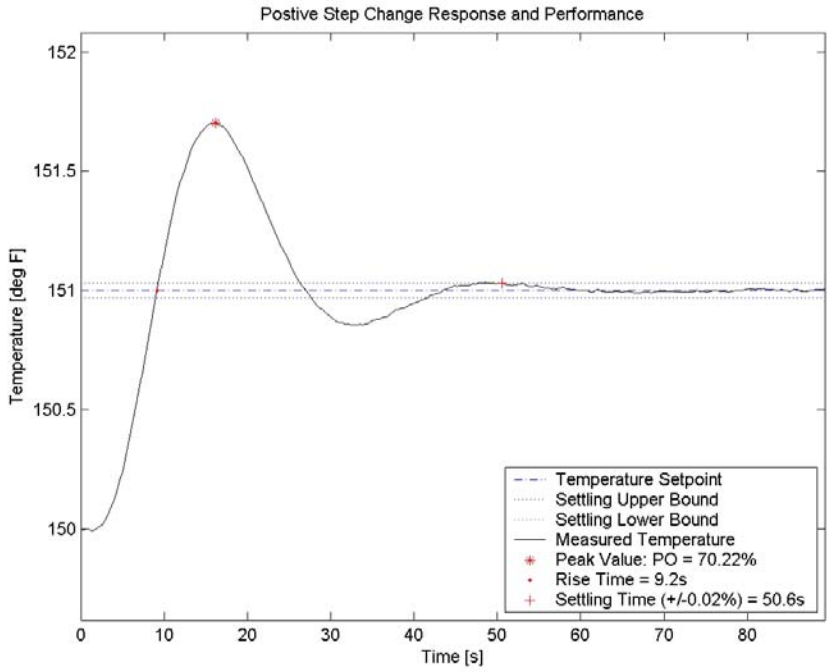


Figure 35 - Ziegler-Nichols Oscillation Method – Positive Step Change

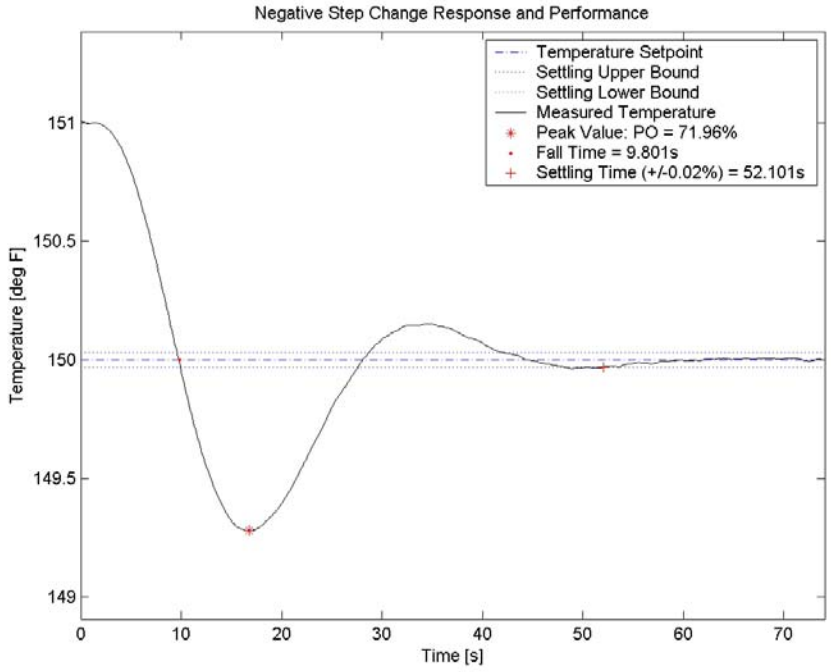


Figure 36 - Ziegler-Nichols Oscillation Method – Negative Step Change

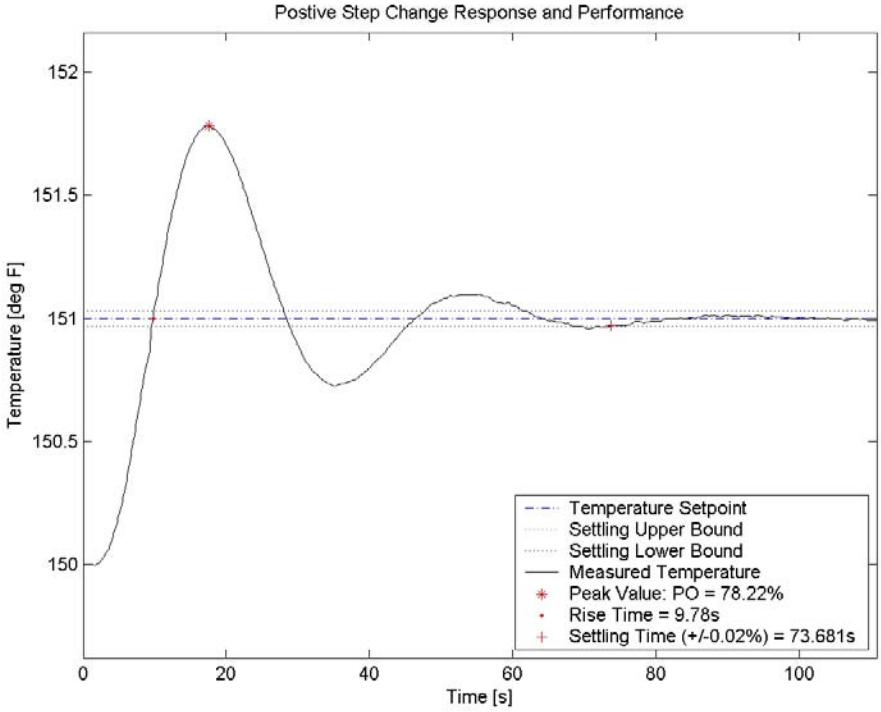


Figure 17 - Ziegler-Nichols Model Based Method – Positive Step Change

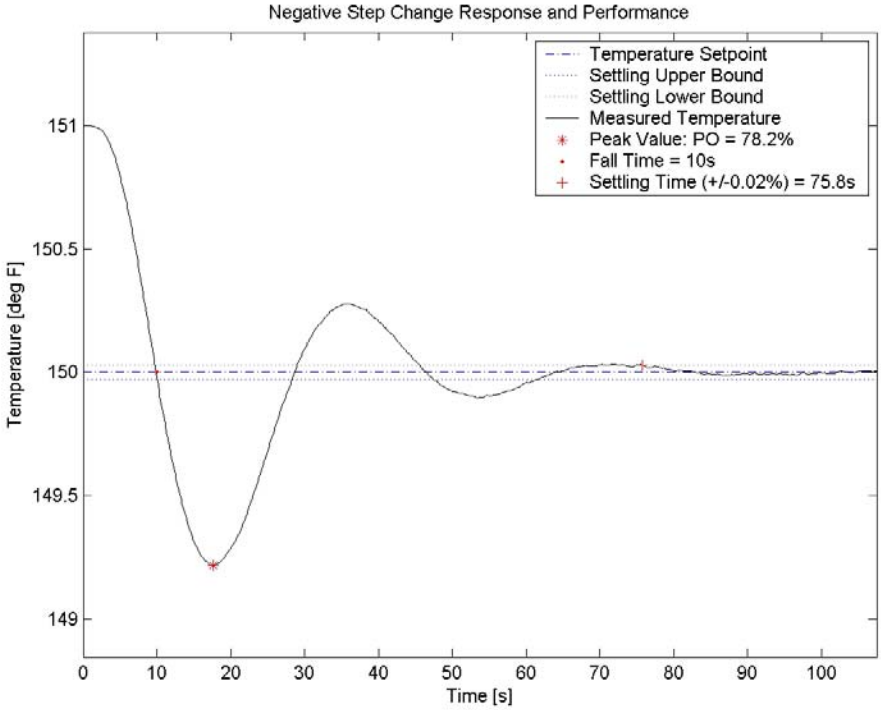


Figure 38 - Ziegler-Nichols Model Based Method – Negative Step Change

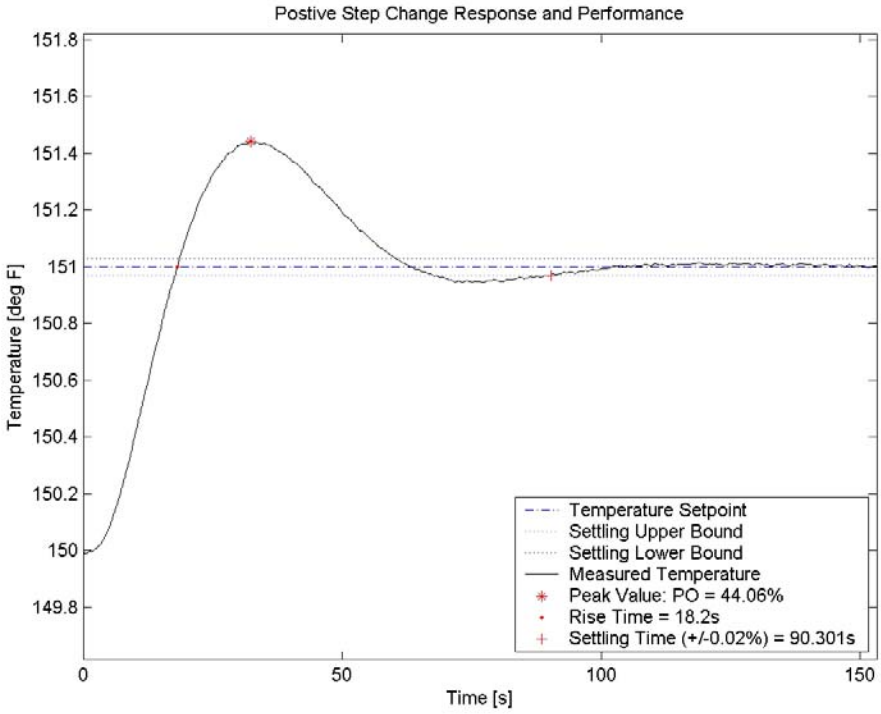


Figure 38 - Root Locus Method – Positive Step Change

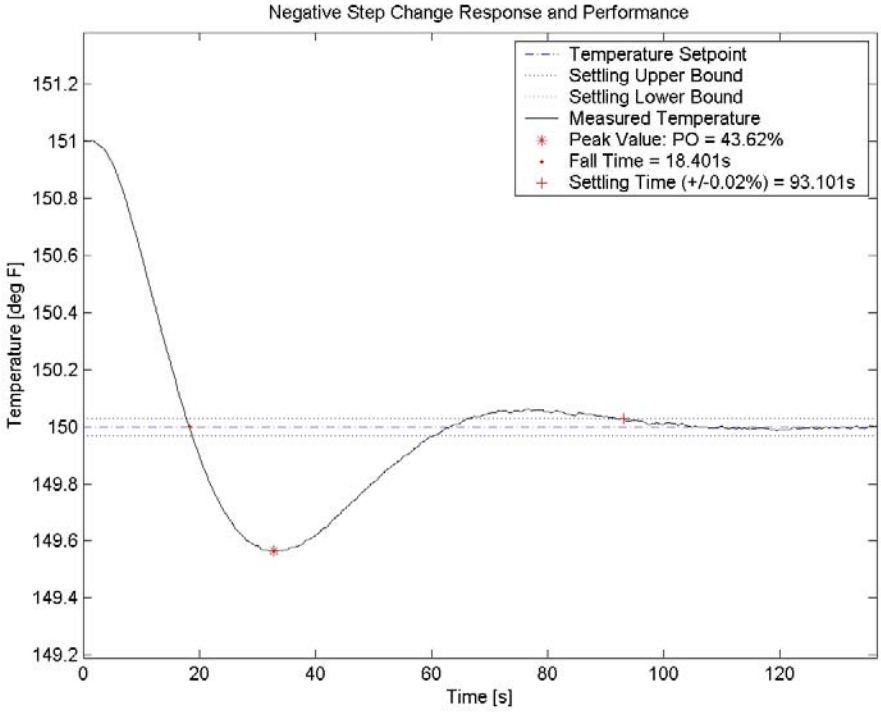


Figure 40 - Root Locus Method – Negative Step Change

Of the three sets of PID gains tested, the set developed through the Ziegler-Nichols Model method appears to be the least desirable. It produces the greatest overshoot, and rise and settling times midway between the Ziegler-Nichols Oscillation Method designed controller and the Root Locus Designed controller.

The slowest responding controller is the controller designed through the Root-Locus method. This is likely do to the linearization of the model that is required for this method. When linearizing the model, the pure exponential time delay is approximated as a fourth order function. This leads to inaccuracies in the Root-Locus diagram used for pole and zero placements. However, this controller provides the benefit of producing the lowest overshoot of the controllers tested.

The controller designed through the Ziegler-Nichols Oscillation Method produced the fastest responding controller. This is a result of the control design procedure that produces as high a proportional gain as possible. This proportional gain increases the speed of the controller's response yet also increases the amount of overshoot in the controller.

The result of comparing these three PID control design method's performance serves to illustrate the trade-offs made in control design. For a system that needs a PID controller developed quickly, that is likely to respond quickly, the Ziegler-Nichols Oscillation Method yields suitable results. This design approach is quick and although requires on-line development, is easy to execute.

In contrast, if more time is available, the Root-Locus design is a desirable method. Through this method, the optimal gains can be developed through mathematical placement of the controller's poles and zeros. However, much of the success of this method relies on the accuracy of the model.

IX. Conclusions

The experiment performed yielded results suitable for the system. While some margin of uncertainty exists in the measurements, the magnitude is negligible when compared to the scale of the system. This uncertainty exists due to the graphical methods for determining certain parameters and the limitations of the data acquisition system. However, since the system under investigation is such a relatively slow system, minor uncertainties in time values are insignificant.

In the objectives for this experiment it was stated that the goals of this experiment were to gain insight into closed-loop feedback controllers and two different methods of control. Additionally the design of a PID controller was covered. The results of this experiment successfully produce the desired objectives.

The difference in closed and open loop control has been clearly illustrated. Attempting to control a system in an open loop requires significant a priori knowledge of the plant, desired output state and the system's environment. By closing the loop the system can become "self controlling." The actions taken by the system based upon this feedback provide the difference in control methods.

One method investigated was that of the On-Off controller. While this controller is not very complex, nor highly accurate, it can yield results suitable for many applications. In the specific case of temperature control, an environment tolerant of a few degrees difference from the desired temperature is an ideal candidate for this control methodology. Such applications exist in typical HVAC (Heating, Ventilation, and Air Conditioning) systems employed in most buildings. Due to its inherent simplicity in design and implementation, the Bang-Bang controller is typically a low cost method. However if precise control was desired, such as needed in certain laboratory or medical environments, Bang-Bang control is unsuitable.

The other method investigated was that of Proportional-Integral-Differential control. As stated above, there was some mixed success with this controller. For low gain values, the simulation mirrors the actual lab results very well. However, as the gains increased, the simulation became unstable. The probable cause for this is the absence of the 2^{nd} order Butterworth filter in the simulation. Given more time, a filter would be added to increase the robustness of the system.

The PID controller is a very common controller in many applications where a system needs to accurately track command changes and maintain zero steady-state error. Many methods, both on- and off-line, exist for designing PID controllers. The time requirements of these different methods vary as well as their overall design cost. Additionally, most PID controllers will ultimately require some on-line tuning for optimum performance due to modeling inaccuracies.

Despite the increased accuracy and precision available with the PID controller, due to their added complexity, PID controllers are typically more costly than more simplistic control mechanisms.

In summary, the stated qualitative objectives of this experiment were met with great success. Additionally, the qualitative portions of the experiment were performed and produced results of accuracy suitable to the equipment being used and the system being investigated.

Laboratory Discussion Items

1. See Model Validation Section for Simulink Model.

2. In order to plot the step response of the open loop system, the several modifications to the Simulink model must be made. The four switches are positioned as follows

- Switch 1 is set to the Bang/Bang controller.
- Switch 2 is set to zero (no step change in setpoint)
- Switch 3 is set to Duty Cycle Step (step change duty cycle)
- Switch 4 is set to the up position (closed loop)
- Switch 5 is set to the block dynamics (with initial outputs)

In lab, a step of 8% (from 37% to 45%) in duty cycle was used. In order to simulate an open loop system, the relay (Bang/Bang Controller) lower limit (output when off) is set to 37. Furthermore, the Set Point Temperature is set to a value lower than the ambient temperature. This ensures that the Bang/Bang controller constantly puts out a duty cycle of 37%. Lastly, the Duty Cycle Step block is set to introduce a step of magnitude 8 at time where the temperature of the system is roughly 143 degrees F (in lab, the 8% step in duty cycle was applied when the temperature was 143 degrees F). Performing these operations yields Figure 41.

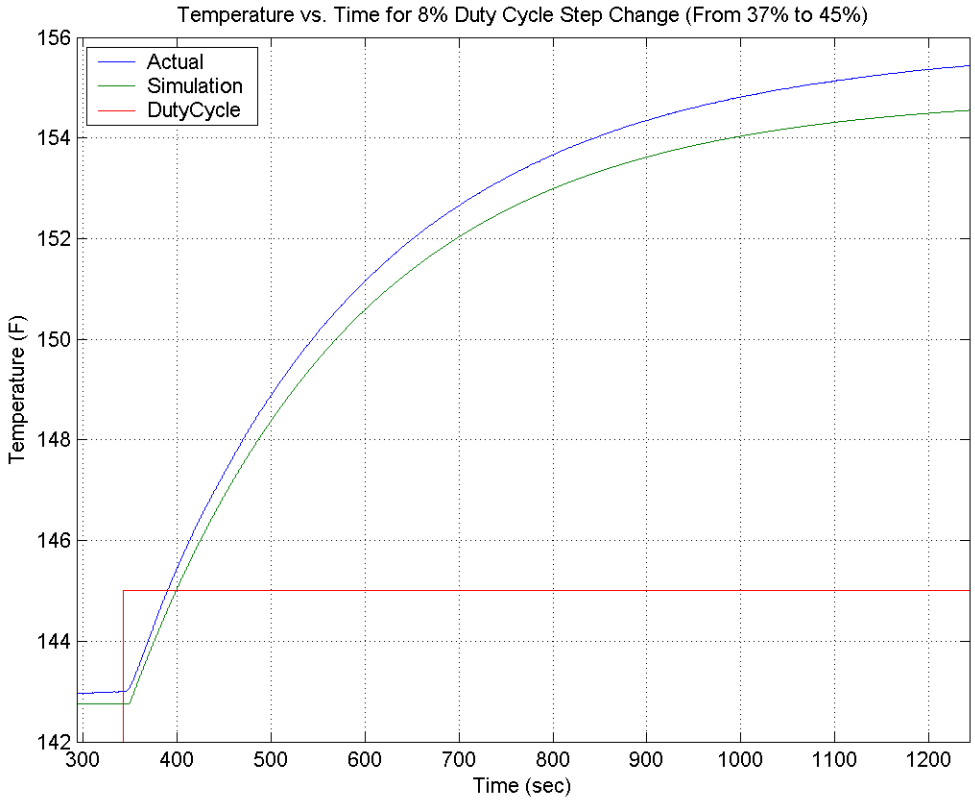


Figure 41: Step response comparison between simulation and actual data

As can be seen, the response if very close, but not exact. Once again, there are several possible reasons for error. First of all, the parameters were derived using graphical methods and there may be some error there. The presence of error in the values of k_1, k_2, and τ is shown by looking at the final value of the system with a duty cycle of 37%. Using Eq. 10 yields a final value of 142.76 F. This shows that the simulation cannot achieve the temperature achieved in lab.

3. The step responses of the three PID controllers have already been compared with actual step responses in the Model Validation Section.

4. The gain and phase margins of the system can now be evaluated. Recall that the block diagram for the system is given by Figure 42

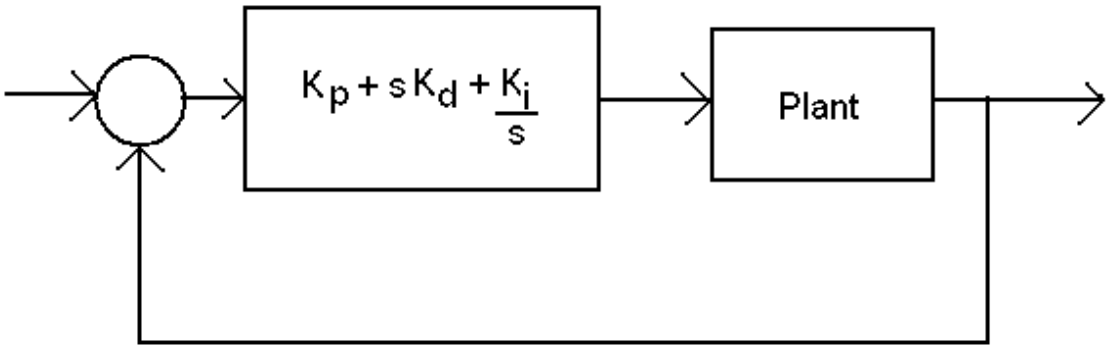


Figure 42: Block diagram of PID controlled system

The input and output points can first be placed around the plant and a linearized model the plant can be obtained. Then the input and output points can be placed around the PID controller with the correct gains to obtain the compensator's linearized model. The linearized plant and compensator models can then be imported into SISOtool in the G and C position, respectively. The open-loop bode diagram is then plotted using the SISOtool.[8] Performing this operation for the system with the PID controller designed using the Ziegler-Nichols oscillation method yields Figure 43.

[8] The open loop bode diagram is more useful than the closed loop diagram because the signal is fed back to the plant.

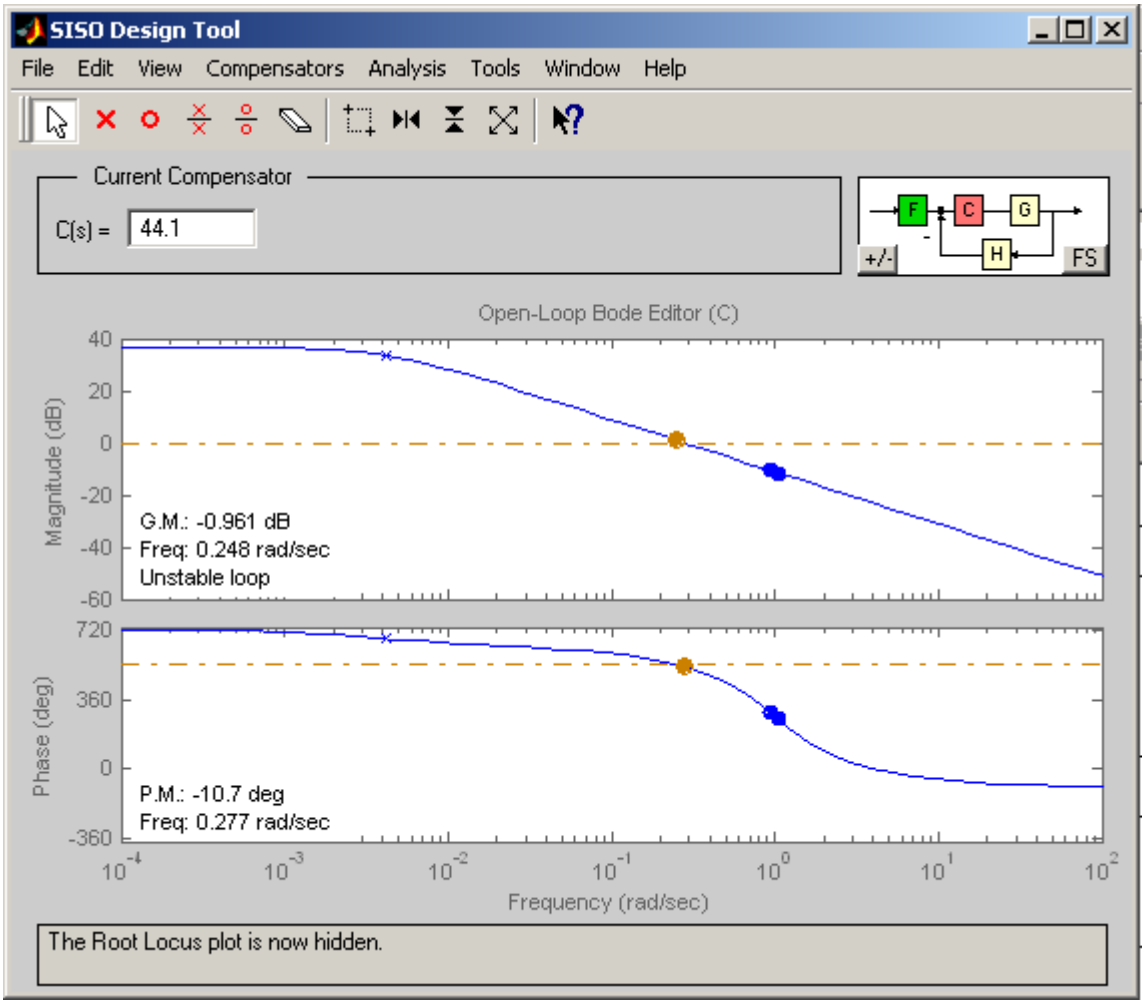


Figure 43: Open loop bode diagram for system with K_p=44.1, K_d=121.275, K_i = 4.009

As can be seen, this system is unstable according to the simulation. This is obvious since both the gain and phase margin are negative. Furthermore, using the SISOtool root locus feature, there are a pair of complex closed loop poles with a positive real portion. This guarantees that the system is unstable at high frequencies. In general, this controller is not very robust according to the simulation. However, as shown before, the controller the best in the laboratory environment with the actual system. Once again, this discrepancy here is likely due to the fact that the controller used in lab is different from the controller used in Simulink. Repeating this process for the controller designed using the Ziegler-Nichols model based method yields Figure 44.

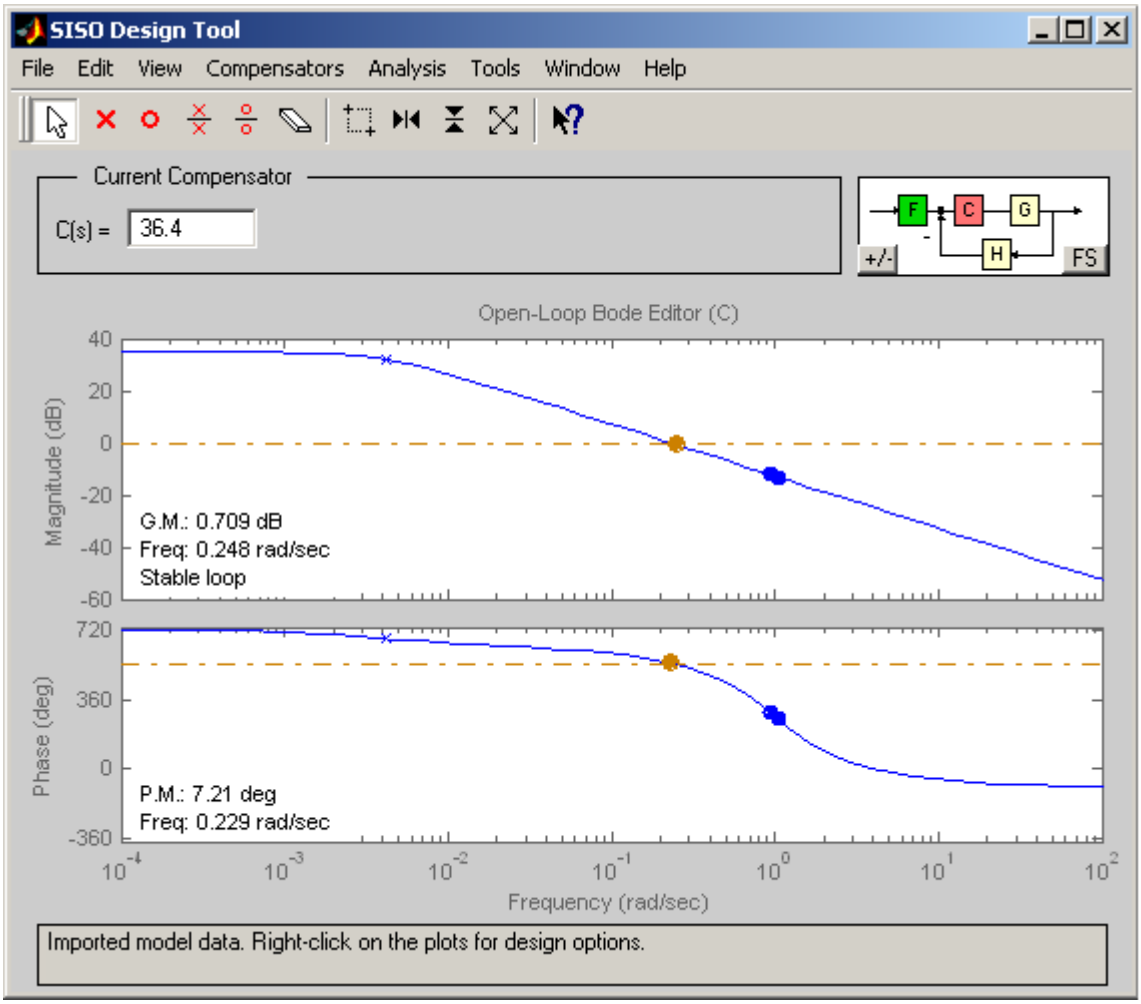


Figure 44: Open loop bode diagram for system with K_p=36.39, K_d=93.44, K_i = 3.64

As can be seen, this controller appears to be stable. This agrees with the results obtained using Simulink. The simulation parameters seemed to match the actual data. However, the phase margin is not very large, only 7.21 degrees. This does not allow much room for error due to unmodeled dynamics. Lastly, this can be repeated for the PID control designed using the Root Locus method to yield Figure 45.

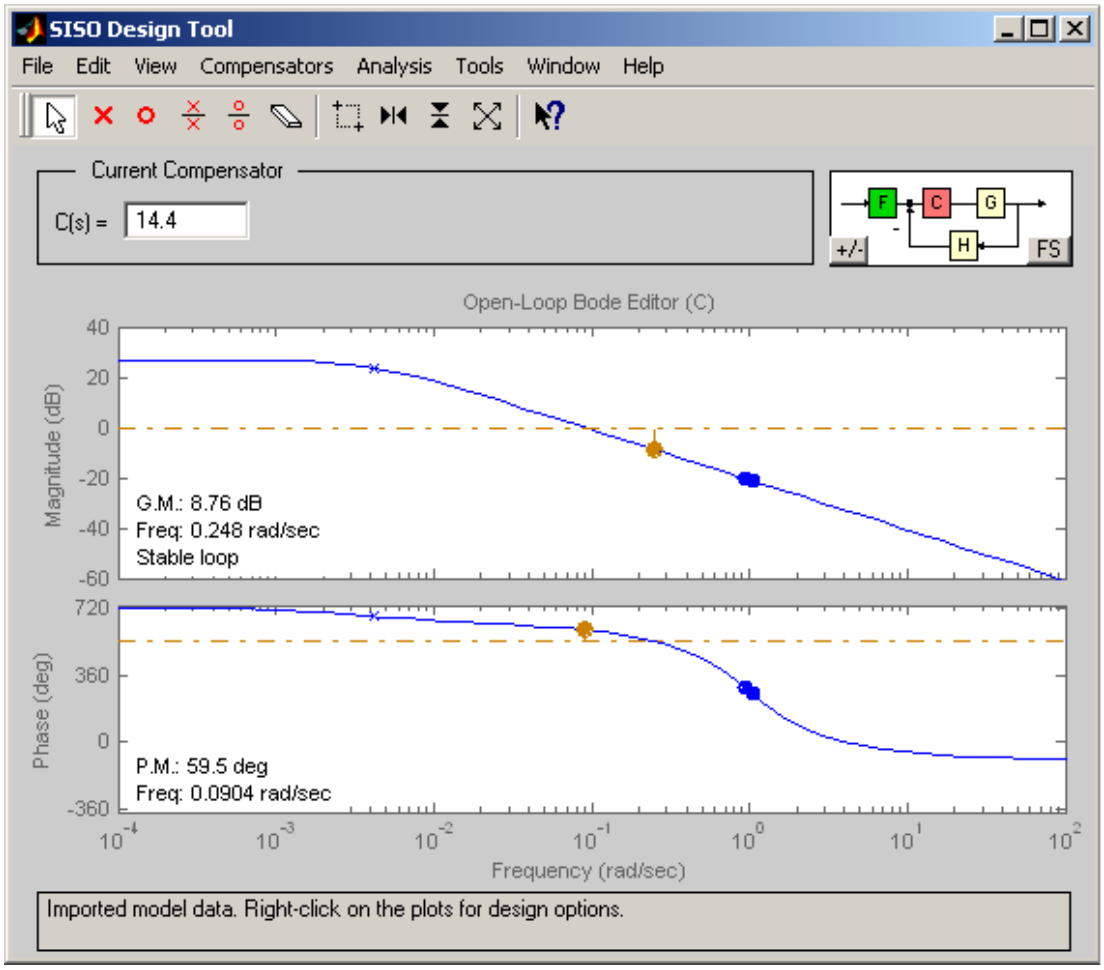


Figure 45: Open loop bode diagram for system with K_p=14.4, K_d=13.1, K_i =0.603

As can be seen here, this appears to be the most robust controller with a gain and phase margin of 8.76 dB and 59.5 deg, respectively. This makes sense considering that the gains are the least aggressive of the three designs. The gain and phase margins and system bandwidth for the three designs are summarized below in Table 9. The band width in this situation is defined as the frequency where the magnitude drops 3 dB from the low frequency value.

Table 9: Gain and phase margins for the three designs

Method	Kp	Kd	Ki	Phase Margin (deg)	Gain Margin (dB)	Bandwidth (Hz)
Ziegler Nichols Oscillation	44.1	121.275	4.009	-10.7	-0.961	N/A Unstable
Ziegler Nichols Model	36.9	93.44	3.64	7.21	0.709	0.0047

Root Locus	14.4	13.1	0.603	59.5	8.76	0.0050

As can be seen, it appears that as the gains increase, the gain and phase margin decrease. This makes sense. It means that the system has better response at the cost of stability.

5. The duty cycle for the three PID control designs can be easily plotted by running the simulation three different times. This yields Figure 46.

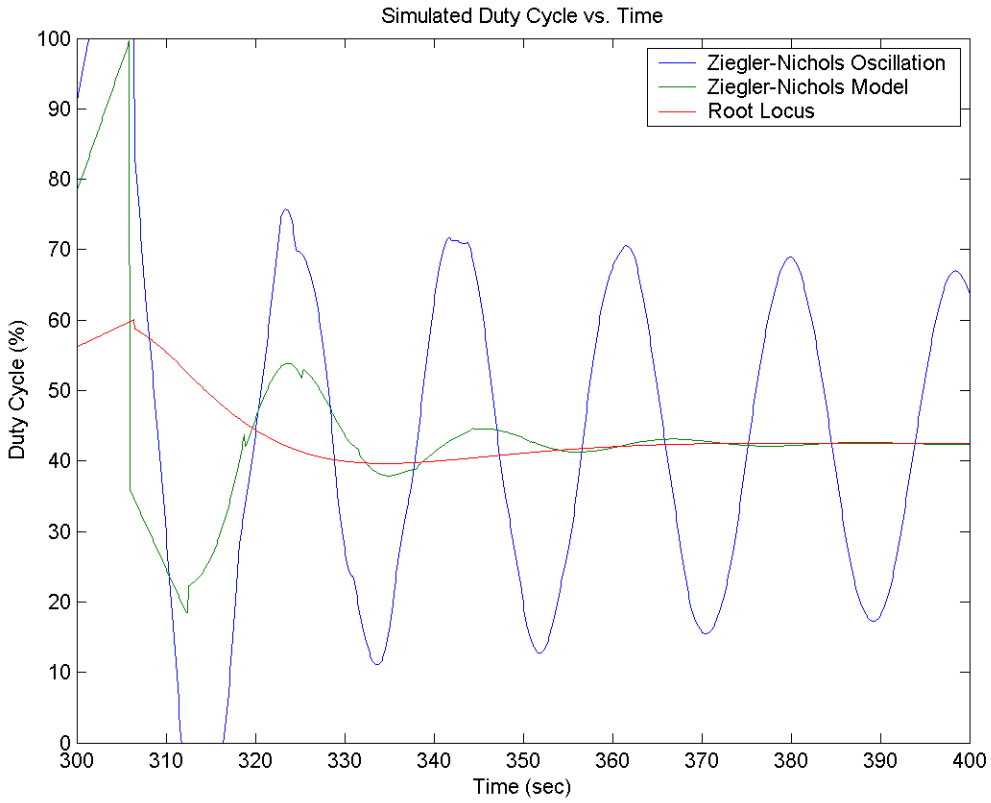


Figure 46: Duty cycle vs. time response to a 1 degree step input

In the figure above, the 1 degree step change is introduced at t = 300 seconds. This shows that initially, the duty cycle is perturbed due to the sudden presence of error. As expected, as the gains become more aggressive, the controller becomes less stable. The reason why this occurs can be make more obvious by observing the proportional, derivative, and integral terms as shown below in Figure 47, 48, and 49, respectively.

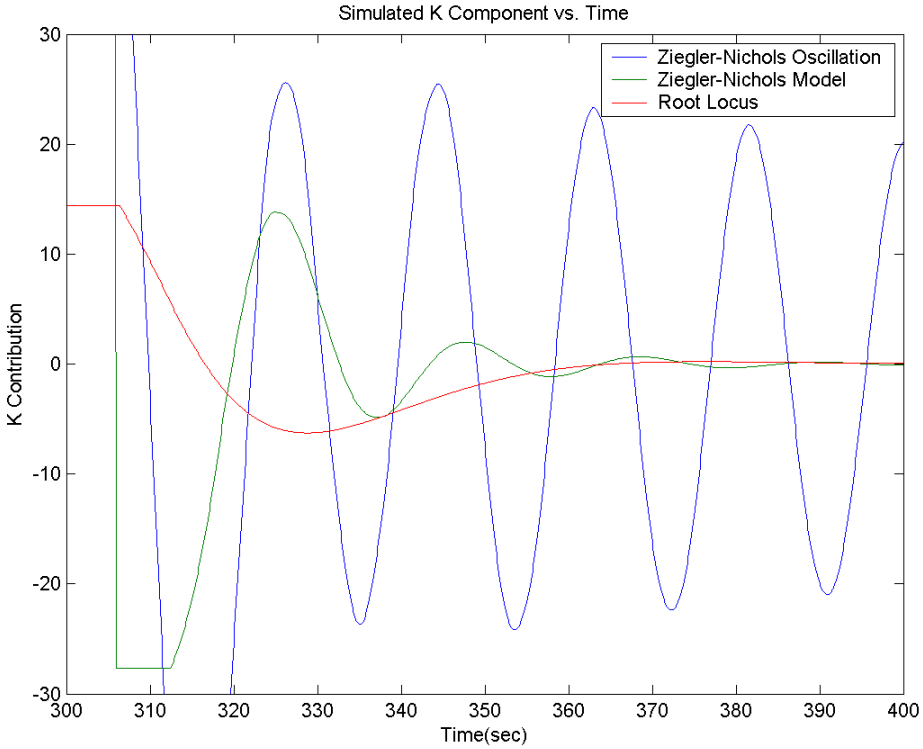


Figure 47: Proportional component vs. time response to a 1 degree step input

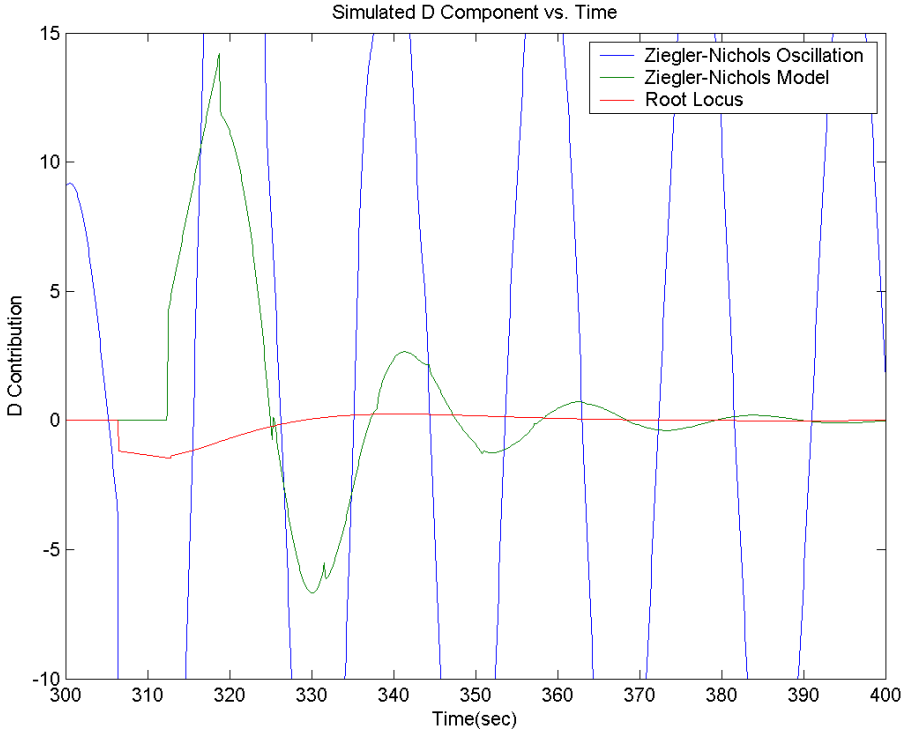


Figure 48: Derivative component vs. time response to a 1 degree step input

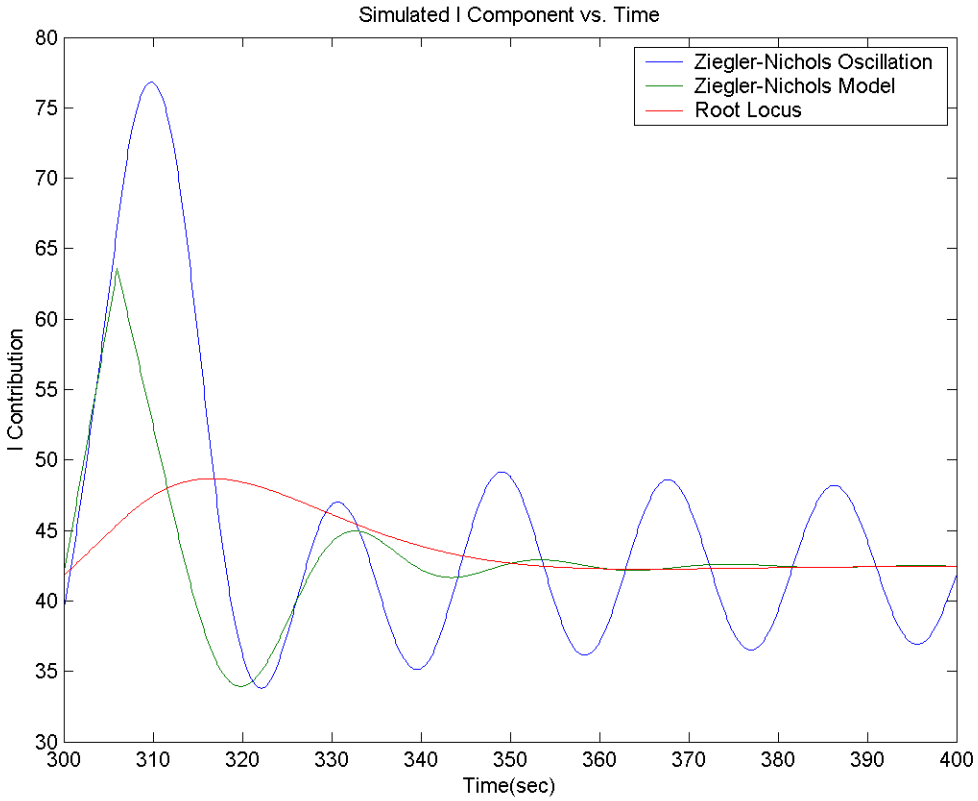


Figure 49: Integral component vs. time response to a 1 degree step input

As can be seen above, the reason why the duty cycle is so noisy at higher gains is because the proportional and derivative terms become excessively noisy and oscillatory at higher gains. The other interesting thing to notice is that for the two stable responses (Ziegler-Nichols Model based method and the Root Locus Method), both the proportional and derivative terms become zero as time goes on. However the integral term remains constant. Furthermore, after a long time, the integral term is the only one contributing to the duty cycle. This is because the integral term is used to remove the steady state error. This shows what duty cycle is required to sustain a setpoint temperature of 151 degrees.

6. The steps involved in deriving the Root Locus design and accompanying plots have already been presented in Control System Design Section.

X. Appendix

MatLab Code

Derive k_1, k_2, and τ

```
%Christopher Lum
%Amanda Stephens
%Travis Reisner
%Brian Hass

%AA448 Ly
%Jan.24, 2003

%This is a second attempt at the step input.  we need
to try and get a
%better steady state value before the step change.

clear
clc
close all

%---------LAB 1 - MODELING AND DESIGN OF A TEMPERATURE
CONTROL---------------

%PART 2
%Here we would like to construct a linear line from
the data points.  We
%used three different reference voltages

%-------------------------Vref = 8 volts-------------
---------------------

%What is the actual reference voltage across the
heater?
Vref8 = 8.000;          %Volts

%Enter in the duty cycle and steady state temperatures
for 8 volts
DutyCycle8 = [35.2 40 48.4];
Tss8 = [140.1 147.2 160.01];

%Try a linear fit to the data (order 1)
coefficients8 = polyfit(DutyCycle8,Tss8,1);

%Extract the slope and offset of the linear fit line
slope8 = coefficients8(1,1);
offset8 = coefficients8(1,2);

%Get a set of y values for plotting the linear fit
curve
Tss8Fit = (slope8*DutyCycle8) + offset8;

%Plot the datapoints and the linear fit
figure
plot(DutyCycle8,Tss8,'rx',DutyCycle8,Tss8Fit)
title('Steady State Temperature vs. Duty Cycle for
V_r_e_f = 8.000 Volts')
xlabel('Duty Cycle (%)')
ylabel('Steady State Temperature (F)')
legend('Data Points','Linear Fit',2)
grid


%-------------------------Vref = 9 volts-------------
---------------------

%What is the actual reference voltage across the
heater?
Vref9 = 9.074;          %Volts

%Enter in the duty cycle and steady state temperatures
for 9 volts
DutyCycle9 = [26.5 31.5 37.5];
Tss9 = [140.00 150.00 160.01];

%Try a linear fit to the data (order 1)
coefficients9 = polyfit(DutyCycle9,Tss9,1);

%Extract the slope and offset of the linear fit line
slope9 = coefficients9(1,1);
offset9 = coefficients9(1,2);

%Get a set of y values for plotting the linear fit
curve
Tss9Fit = (slope9*DutyCycle9) + offset9;

%Plot the datapoints and the linear fit
figure
plot(DutyCycle9,Tss9,'rx',DutyCycle9,Tss9Fit)
title('Steady State Temperature vs. Duty Cycle for
V_r_e_f = 9.074 Volts')
xlabel('Duty Cycle (%)')
ylabel('Steady State Temperature (F)')
legend('Data Points','Linear Fit',2)
grid



%-------------------------Vref = 10 volts------------
----------------------

%What is the actual reference voltage across the
heater?
Vref10 = 10.036;         %Volts

%Enter in the duty cycle and steady state temperatures
for 10 volts
DutyCycle10 = [22.0 26.1 30.2];
Tss10 = [140.00 150.00 160.00];

%Try a linear fit to the data (order 1)
coefficients10 = polyfit(DutyCycle10,Tss10,1);

%Extract the slope and offset of the linear fit line
slope10 = coefficients10(1,1);
offset10 = coefficients10(1,2);

%Get a set of y values for plotting the linear fit
curve
Tss10Fit = (slope10*DutyCycle10) + offset10;

%Plot the datapoints and the linear fit
figure
plot(DutyCycle10,Tss10,'rx',DutyCycle10,Tss10Fit)
title('Steady State Temperature vs. Duty Cycle for
V_r_e_f = 10.036 Volts')
xlabel('Duty Cycle (%)')
ylabel('Steady State Temperature (F)')
legend('Data Points','Linear Fit',2)
grid

%In order to explore the effect of Vref on the slope
of the linear fit
%lines, we should plot slope vs. Vref.

%create an array of slopes.  We can add a value of 0
in the beginning since
%the slope will obviously be zero if Vref is 0.
slopes = [0 slope8 slope9 slope10];

%create an array of the Vref values. Likewise, we add
the point (0,0)
Vrefs = [0 Vref8 Vref9 Vref10];

%The response should be second order, so try a second
order fit
coefficients = polyfit(Vrefs,slopes,2);

%Get a set of y values for plotting the quadratic fit
curve.  We should
%create an array of Vref that is easier to plot
VrefsPlotting = linspace(0,12,50);

YFit = (coefficients(1)*VrefsPlotting.^2) +
(coefficients(2)*VrefsPlotting) +
(coefficients(3)*ones(1,length(VrefsPlotting)));

figure
plot(Vrefs,slopes,'rx',VrefsPlotting,YFit)
title('Slopes vs. V_r_e_f')
xlabel('Vref (Volts)')
ylabel('Slope (F/%)')
legend('Data Points','Quadratic Fit',2)
grid

%We now need to solve for a value of k2.  We need to
first import the data
load step_change_second

time =          tempdata(:,1)/1000;     %Extract time
and convert to seconds
SetPoint =      tempdata(:,2);          %Extract set
point temperature
Tsensor =       tempdata(:,3);          %Extract
sensor temperature
Terror =        tempdata(:,4);          %Extract
temperature error
DutyCycle =     tempdata(:,5);          %Extract duty
cycle
Kp =            tempdata(:,6);          %Extract
proportional gain value
Ki =            tempdata(:,7);          %Extract
integral gain value
Kd =            tempdata(:,8);          %Extract
derivative gain value
Proportional =  tempdata(:,9);          %Extract
proportional contribution
Integral =      tempdata(:,10);         %Extract
integral contribution
Derivative =    tempdata(:,11);         %Extract
derivative contribution


%Plot the duty cycle
figure
plot(time,DutyCycle)
title('Duty Cycle vs. Time')
xlabel('Time (seconds)')
ylabel('Duty Cycle (%)')
```

```
grid
axis([min(time) max(time) min(DutyCycle)-2
max(DutyCycle)+2])


%What is the initial duty cycle?
InitialDutyCycle = DutyCycle(1,1);

%We need to first solve for when the step input is
applied (ie when the
%duty cycle is changed). We will cycle through the
entire array and see
%when the array changes value.

%We also need to account for the fact that the duty
has a small change in
%it.  We are not interested in this, but rather the
large jump in step
%input.
for counter=1:length(DutyCycle)

    %What is the current duty cycle
    CurrentDutyCycle = DutyCycle(counter,1);


    if CurrentDutyCycle == max(DutyCycle)
        %If the duty cycle is at maximum, then break
        break
    else
        %If the duty cycle is not a maximum, increment
counter
    end
end

%What is the duty cycle after the step input?
FinalDutyCycle = DutyCycle(counter,1);

%What time did the change occur at?
StepTime = time(counter,1);


%We can plot the sensor temperature response with
respect to time.
figure
plot(time,Tsensor)
title('Sensor Temperature vs. Time')
xlabel('Time (sec)')
ylabel('Sensor Temperature (F)')
grid
axis([min(time) max(time) min(Tsensor)-1
max(Tsensor)+1])


%We can find what time the temperature increases from
visually inspecting
%the graph.  This occurs at roungly t = 345 seconds
TimeStart = 345;

%We need to find out at what index of the time array
corresponds to 345
%seconds.
difference = abs(time-TimeStart);

%Find the minimum difference
[minvalue StartTimeIndex] = min(difference);

%What is the maximum temperature achieved at steady
state after the step
%input is implemented
Tmax = max(Tsensor);

%We can now find the steady state temperature before
the step input
Tmin = Tsensor(StartTimeIndex);

%We can also calculate what the previous steady state
temperature should
%have been with a duty cycle of 37% using the slopes
and offsets from the
%first part (we can directly use slope8 and offset8
since Vref for this
%part and the other part are exactly the same).
TminCalculated = (slope8*min(DutyCycle)) + offset8;

%We can now calculate the difference in steady state
temperatures
DeltaTss = Tmax - Tmin;

%The equation of the line (assuming a first order
system subject to a step
%input) should be
%
%   y(t) = Tss(1-e^-(t/T)
%
%We can now find what the change in temperature should
be at t = T = 1/k2
DeltaCriticalTemp = DeltaTss*(1-exp(-1));

%We can find the temperature at t = T = 1/k2
CriticalTemp = Tmin + DeltaCriticalTemp;

%We would like to find the index where this
temperature occurs
difference = abs(Tsensor - CriticalTemp);

%Find the minimum difference
[minvalue CriticalTempIndex] = min(difference);

%We can now find out the time constant, T
T = time(CriticalTempIndex) - TimeStart;

%Solve for k2
k2 = 1/T;

%We can now solve for k1 as well
R = 25;         %ohms

k1 = (slope8*k2*R)/(Vref8^2);



%We now want to plot both the sensor temperature and
the duty cycle on the
%same graph
figure
plotyy(time,DutyCycle,time,Tsensor)
title('Duty Cycle and Sensor Temperature vs. Time')
xlabel('Time (sec)')
ylabel('Duty Cycle (%)')
grid


%We need to calculate the maximum tangent line in
order to find the value
%of tau.  Due to the amount of noise in the signal, we
need to calculate
%the slope between two points spaced 15 second apart
in order to filter out
%the noise.
TimeInterval = 15;           %interval in seconds

%The time array is spaced in 0.1 seconds, so the index
interval is given
%by
IndexInterval = TimeInterval/0.1;

%We only want to look for the maximum slope after the
temperature begins to
%rise after the step input (ie. After Tstart).  This
ignores the time
%delay.
counter = 1;

for IndexCounter = StartTimeIndex:length(time)

    %We need to check that we aren't at the end of the
array (can't add
    %IndexInterval to the end of the array
    if IndexCounter > length(Tsensor) - IndexInterval
        break
    else
    end

    %Calculate the slope from this point to a point 1
second later
    slope(:,counter) = (Tsensor(IndexCounter +
IndexInterval) - Tsensor(IndexCounter))/TimeInterval;

    %Increment the slope counter by 1
    counter = counter + 1;

end

%We need to create an array of time to plot the slopes
against.  We know
%that the sampling rate is one every 0.1 seconds
endtime = length(slope)*0.1;

slopeTime = [0:.1:endtime-0.1];

%plot the slope as a function of time
figure
plot(slopeTime,slope)
title('Slope of Temperature/Time vs. Time')
xlabel('Time (sec)')
ylabel('dTemp/dt (F/sec)')
grid


%The graph looks like we were able to filter out most
of the noise, so we
%can now find the maximum slope
MaxSlope = max(slope);

%We need to find out where this slope occurs.
MaxSlopeIndex = find(slope==MaxSlope);

%In case max slope occurs more than once
MaxSlopeIndex = MaxSlopeIndex(1);

%Find the time where the max slope occurs
TimeAtMaxSlope = slopeTime(MaxSlopeIndex) + TimeStart;

%Find the temperature at this time
TimeDifference = abs(time - TimeAtMaxSlope);
minTimeDifference = min(TimeDifference);

TimeAtMaxSlopeIndex =
find(TimeDifference==minTimeDifference);

%find the temperature when the maximum slope occurs
TempAtMaxSlope = Tsensor(TimeAtMaxSlopeIndex);

%Create and array for the temperatures for the linear
line that passes
```

```
%through the point where the max slope occcurs using
point-slope format.
TempForLinearLine = MaxSlope*(time - TimeAtMaxSlope) +
TempAtMaxSlope;

%Find the final steady state temperature
Tfinal = max(Tsensor);

%plot the tangent line along with the temperature and
the steady state
%value and a vertical line showing where the duty
cycle starts and a
%horizontal line showing the final steady state value
figure
plot(time,Tsensor,time,TempForLinearLine,TimeAtMaxSlop
e,TempAtMaxSlope,'rx',time,Tmin*ones(1,length(time)),S
tepTime*[1 1],[min(Tsensor)
max(Tsensor)],time,Tfinal*ones(1,length(time)))
title('Temperature vs. Time with Maximum Tangent
Line')
xlabel('Time (sec)')
ylabel('Temperature (F)')
legend('Temperature','Tan
Line','Intersection','Initial Steady State','Duty
Cycle Change','Final Steady State',4)
axis([(StepTime-100) max(time) (Tmin-5) (Tmax+5)])
grid

%Now we need to find out when the line intersects the
steady state
%temparature
IntersectionTime = ((Tmin -
TempAtMaxSlope)/MaxSlope)+TimeAtMaxSlope;

%We can now calculate tau, which is the difference
from where the tangent
%line intersects the steady state temperature line and
the time when the
%step was introduced.
Tau = IntersectionTime - StepTime;


%Print out the appropriate values on the screen
sprintf('Parameters Derived Using Graphical Means for
k1 and Tau and Mathematical for k2\n  T = %3.2f\n\n
k1 = %1.6f\n  k2 = %1.6f\n  tau = %1.4f',T,k1,k2,Tau)


%We can also find the value of k2 by looking to see
where the maximum
%tangent line intersects the next steady state value.
FinalDifference = abs(TempForLinearLine - Tfinal);
minFinalDifference = min(FinalDifference);
FinalIndex =
find(FinalDifference==minFinalDifference);

%At what time does this intersection with the final
steady state occur
FinalIntersectionTime = time(FinalIndex);

%Solve for 1/k2
k2graphical = 1/(FinalIntersectionTime -
IntersectionTime);

sprintf('Parameters Derived Using Graphical Means for
k1, k2, and Tau\n  T = %3.2f\n\n  k1 = %1.6f\n  k2 =
%1.6f\n  tau = %1.4f',T,k1,k2graphical,Tau)

%--------------------------OPTIMIZATION--------------
-----------------
%We can try and find different constants k1, k2, and
tau that better fit
%the data acquired.  We need to create an array of
only the data points
%that we are interested in (namely, Tsensor after the
step change is
%introduced).  We need to find the index number where
the step change
%occurs
StepIndex = find(time==StepTime);

%We now need to cut off all values of Tsensor that are
before this index.
ModifiedCounter = StepIndex;
NormalCounter = 1;

while ModifiedCounter <= length(Tsensor)
    TsensorModified(1,NormalCounter) =
Tsensor(ModifiedCounter);

    NormalCounter = NormalCounter + 1;
    ModifiedCounter = ModifiedCounter + 1;
end

%We also need to create a modified time scale which
starts at time = 0 when
%the step input is introduced.  We know that the step
size is roughly 0.1
%seconds.  We need to calculate what the range of
times will be.
timeInterval = 0.1;                 %seconds
endtime = timeInterval*length(TsensorModified);
timeModified = [0:timeInterval:endtime -
timeInterval];

%We now would like to make a range of values of k1,
k2, and Tau based on
%the values that we calculated earlier.
PercentDeviation = .30;        %Percent to vary each
parameter +/-.  ie .2 = 20%
N = 20;                        %number of different
values to try


k1Range = linspace(k1-
PercentDeviation*k1,k1+PercentDeviation*k1,N);
k2Range = linspace(k2-
PercentDeviation*k2,k2+PercentDeviation*k2,N);
TauRange = linspace(Tau-
PercentDeviation*Tau,Tau+PercentDeviation*Tau,N);

%We can now enter in the transfer function with
varying values of k1, k2,
%and Tau and calculate the squared error
k1Counter = 1;
k2Counter = 1;
TauCounter = 1;

Npade = 4;                      %order of Pade
approximation for transport delay
MagnitudeofStep = 8;            %since step command
assumues unit step, what is the necessary
multiplicative factor
n = 1;                          %Counter that shows
how many times the loop has been repeated.  Need to
initialize with n = 1

%We need to fill the first row of our data spreadsheet
with zeros since
%there will be no data.  The spreadsheet will be in
the form
%
%   1st column = k1
%   2nd column = k2
%   3rd column = Tau
%   4th column = Error
Data = [0 0 0 0];

%Start a clock
tic
for k1Counter=1:length(k1Range)

    for k2Counter=1:length(k2Range)

        for TauCounter = 1:length(TauRange)
            %Increment the counter
            n = n + 1;

            %Calculate the transfer function for the
combination of the
            %first two blocks (ie the constant gain of
Vref^2/R and the
            %plant)
            firstTF =
tf([(Vref8^2/R)*k1Range(k1Counter)],[1
k2Range(k2Counter)]);

            %Calculate the transfer function for the
second block (ie
            %the tranport delay using a Nth order Pade
approximation
            [num,den] =
pade(TauRange(TauCounter),Npade);
            secondTF = tf(num,den);

            %Calculate the overall trasnfer fucntion
            overallTF = firstTF*secondTF;

            %Calculate the response of the system to a
step input using the
            %modified time range
            [TemperatureFromStep TimeFromStep] =
step(overallTF,timeModified);

            %The step input assumes a unit step, so we
need to multply by
            %the appropriate factor and also add the
offset

            TemperatureFromStep =
MagnitudeofStep*TemperatureFromStep;
            TemperatureFromStep = TemperatureFromStep
+ Tsensor(StepIndex);

            %Calculate the squared error in a vector.
We need to transpose
            %one of the vectors to make the dimensions
match
            SquaredError = (TemperatureFromStep' -
TsensorModified).^2;

            %sum up all of the error
            SumError = sum(SquaredError);

            %Store this in the proper place in the
data spreadsheet
            Data(n,:) = [k1Range(k1Counter)
k2Range(k2Counter) TauRange(TauCounter) SumError];
        end
    end
end

%See how long it took to run the whole process
toc
```

```
%We now need to look for what values of k1, k2, and
Tau yield the smallest
%error.  We need to find the index of the row where
the minimum error
%occurs.  This means searching the 4th column of the
Data spreadsheet.  We
%need to fill the first row with the maximum error or
else the min function
%will always return the index for the first row.
maxValue = max(Data(:,4));
Data(1,:) = maxValue*ones(1,4);

%We can now find out the minimum error and the row
where it occurs
[minError minIndex] = min(Data(:,4));

%We now have values for the optimal values of k1, k2,
and Tau
k1Optimal = Data(minIndex,1);
k2Optimal = Data(minIndex,2);
TauOptimal = Data(minIndex,3);

TOptimal = 1/k2Optimal;

sprintf('Parameters Derived Using Least Squares
Optimization\n  T Optimal = %3.2f\n\n  k1 Optimal =
%1.6f\n  k2 Optimal = %1.6f\n  Tau Optimal = %1.4f\n\n
Total Error =
%4.2f',TOptimal,k1Optimal,k2Optimal,TauOptimal,minErro
r)

%Lets plot the actual temperature from the sensor
versus the model using
%the optimal values
FirstOptimal = tf([(Vref8^2/R)*k1Optimal],[1
k2Optimal]);

[num,den] = pade(TauOptimal,Npade);
SecondOptimal = tf(num,den);

[TemperatureFromStepOptimal TimeFromStepOptimal] =
step(FirstOptimal*SecondOptimal,timeModified);

TemperatureFromStepOptimal =
MagnitudeofStep*TemperatureFromStepOptimal;
TemperatureFromStepOptimal =
TemperatureFromStepOptimal + Tsensor(StepIndex);

%We can also calculate the response of the system
using the values
%of k1, k2, and tau derived using graphical means
FirstGraphical = tf([(Vref8^2/R)*k1],[1 k2]);

[num,den] = pade(Tau,Npade);
SecondGraphical = tf(num,den);

[TemperatureFromStepGraphical TimeFromStepGraphical] =
step(FirstGraphical*SecondGraphical,timeModified);

TemperatureFromStepGraphical =
MagnitudeofStep*TemperatureFromStepGraphical;
TemperatureFromStepGraphical =
TemperatureFromStepGraphical + Tsensor(StepIndex);


%Plot the actual sensor temperature, the model using
values obtained
%graphically, and the model using values obtained from
optimization
figure
plot(timeModified,TsensorModified,timeModified,Tempera
tureFromStepOptimal,timeModified,TemperatureFromStepGr
aphical)
title('Sensor Temperature and Model Prediction Using
Both Optimal and Graphical values of k1,k2, and \tau')
xlabel('Time (sec)')
ylabel('Temperature')
legend('Sensor','Optimal','Graphical',2)
grid


%---------------------------PART 4B-----------------
-----------------
%We can compute values of Kp, Ki, and Kd using the
Ziegler Nichols
%Model-Based Method.

%--------------------Using Measured k1, k2, and Tau---
-----------------
%We can easily calculate Ko
Ko = (k1/k2)*((Vref8^2)/R);

%We already know T so we can calculate appropriate
values of Kp, Ki, and Kd
Kp = (1.2*T)/(Ko*Tau);
Ki = Kp/(2*Tau);
Kd = Kp*0.5*Tau;

sprintf('Ziegler Nicholas Model-Based Method Using
Measured k1, k2, and Tau\n  Kp = %2.2f\n  Ki = %2.2f\n
Kd = %2.2f',Kp,Ki,Kd)


%-------------------Using Optimal k1, k2, and Tau-----
-----------------
KoOptimal = (k1Optimal/k2Optimal)*((Vref8^2)/R);

%We can calculate an optimal T

TOptimal = 1/k2Optimal;

KpOptimal = (1.2*TOptimal)/(KoOptimal*TauOptimal);
KiOptimal = KpOptimal/(2*TauOptimal);
KdOptimal = KpOptimal*0.5*TauOptimal;

sprintf('Ziegler Nicholas Model-Based Method Using
Optimal k1, k2, and Tau\n  Kp = %2.2f\n  Ki = %2.2f\n
Kd = %2.2f',KpOptimal,KiOptimal,KdOptimal)

%Display the pade order approximation
sprintf('Pade order approximation = %d',Npade)
```

ON/OFF Control

```
%--------------------Part (3) ON/OFF Control----------
--------------------
%The purpose of this section is to compare the
Simulink model with the data
%recorded using the on/off controller.

%clear the command window and close graphs, but do not
clear variables
%since we need the simout variable from Simulink.
clc
close all


%1 DEGREE HYSTERESIS BAND
%We need to first load the data from the on/off
controller with a 1 degree
%hysteresis band
load Hysterisis1

time_on_off_1 =          tempdata(:,1)/1000;
%Extract time and convert to seconds
SetPoint_on_off_1 =      tempdata(:,2);
%Extract set point temperature
Tsensor_on_off_1 =       tempdata(:,3);
%Extract sensor temperature
Terror_on_off_1 =        tempdata(:,4);
%Extract temperature error
DutyCycle_on_off_1 =     tempdata(:,5);
%Extract duty cycle
Kp_on_off_1 =            tempdata(:,6);
%Extract proportional gain value
Ki_on_off_1 =            tempdata(:,7);
%Extract integral gain value
Kd_on_off_1 =            tempdata(:,8);
%Extract derivative gain value
Proportional_on_off_1 =  tempdata(:,9);
%Extract proportional contribution
Integral_on_off_1 =      tempdata(:,10);
%Extract integral contribution
Derivative_on_off_1 =    tempdata(:,11);
%Extract derivative contribution

%With a 1 degree hysteresis band and a setpoint
temperature of 150, the heater should turn on when the
%temperature drops below 149.5.  Likewise, the heater
should turn off when
%the temperature rises above 150.5.

hysteresis_band = 1;            %degree F

%for plotting, we need two horizontal lines that
represent the upper and
%lower bounds of the hysteresis band

on_temp = (SetPoint_on_off_1 -
(hysteresis_band/2))*ones(1,2);
off_temp = (SetPoint_on_off_1 +
(hysteresis_band/2))*ones(1,2);

%Lets plot the duty cycle and temperature, and the two
horizontal lines
%showing the hysteresis band

figure
plot(time_on_off_1,Tsensor_on_off_1,[min(time_on_off_1
) max(time_on_off_1)],on_temp,'r--
',[min(time_on_off_1) max(time_on_off_1)],off_temp,'r-
-',time_on_off_1,(DutyCycle_on_off_1*.07)+146)
title('Temperature vs. Time Using ON/OFF Controller
with 1 Degree Hysteresis Band')
xlabel('Time (sec)')
ylabel('Temperature (F)')
legend('Sensor','Lower Hysteresis Limit','Upper
Hysteresis Limit','Duty Cycle',4)
axis([40 140 142 154])


%Do you need to load the optimal values of k1, k2, and
Tau into the
%workspace?
%
%   1 = yes
%   2 = no
selection1 = 1;
if selection1==1
    k1Optimal = 0.00249373
    k2Optimal = 0.00396434
    TauOptimal = 4.47917800

    k1 = 0.00245496
    k2 = 0.00416146
    Tau = 6.39882571

    offset8 = 86.88247312

    Vref8 = 8
    R = 25
else
end

%Make sure to run the Simulink simulation or else the
following code will

%Extract the time, duty cycle, and simulated
temperature
SimulatedTime = simDutyCycle.time;
SimulatedDutyCycle = simDutyCycle.signals.values;
SimulatedTemperature = simTemp.signals.values;
SimulatedSetPoint = simSetPoint.signals.values;

%Plot the duty cycle and temperature vs. time in a
plot similar to the
%first one

figure
plot(SimulatedTime,SimulatedTemperature,[min(Simulated
Time) max(SimulatedTime)],on_temp,'r--
',[min(SimulatedTime) max(SimulatedTime)],off_temp,'r-
-',SimulatedTime,(SimulatedDutyCycle*0.07)+146)
title('Simulated Temperature vs. Time Using ON/OFF
Controller with 1 Degree Hysteresis Band')
xlabel('Time (sec)')
ylabel('Temperature (F)')
legend('Sensor','Lower Hysteresis Limit','Upper
Hysteresis Limit','Duty Cycle',4)
axis([120.5 220.5 142 154])

%Lets plot just the temperature of the actual model
and the simulation on
%top of one another in order to see if better
figure
plot(time_on_off_1,Tsensor_on_off_1,SimulatedTime-
80,SimulatedTemperature)
title('Actual and Simulated Temp. vs. Time for ON/OFF
Controller with 1 Degree Hysteresis Band')
xlabel('Time (sec)')
ylabel('Temperature (F)')
legend('Experiment','Simulation',4)
axis([40 140 142 154])
grid
```

ON/OFF Control (similar for all three parts)

```
%--------------------Part (4.b.A) PID Control---------
---------------------
%Using optimal values of k1, k2, and tau to compute
Kp, Ki, and Kd (note:
%these are the optimal values for the frist time the
lab was run.  They
%are not the best values that depict the model.  The
optimal values that
%were derived when I repeated the lab a second time
are the ones used in
%MatLab.
%
%clear the command window and close graphs, but do not
clear variables
%since we need the simout variable from Simulink.
clc
close all

%-------4.b.A Ziegler-Nichols Based Method (designed
online)----------
%This data was acquired using the following Kp, Ki,
and Kd values
%
%       Kp = 44.1
%       Ki = 4.009
%       Kd = 121.275
%
%We can load the data
load 4_b_A

time_A =          tempdata(:,1)/1000;     %Extract
time and convert to seconds
SetPoint_A =      tempdata(:,2);          %Extract set
point temperature
Tsensor_A =       tempdata(:,3);          %Extract
sensor temperature
Terror_A =        tempdata(:,4);          %Extract
temperature error
DutyCycle_A =     tempdata(:,5);          %Extract
duty cycle
Kp_A =            tempdata(:,6);          %Extract
proportional gain value
Ki_A =            tempdata(:,7);          %Extract
integral gain value
Kd_A =            tempdata(:,8);          %Extract
derivative gain value
Proportional_A =  tempdata(:,9);          %Extract
proportional contribution
Integral_A =      tempdata(:,10);         %Extract
integral contribution
Derivative_A =    tempdata(:,11);         %Extract
derivative contribution

%We can also load the data from the simulation
%Extract the simulated time, duty cycle, temperature,
and setpoint.
SimulatedTime = simDutyCycle.time;
SimulatedDutyCycle = simDutyCycle.signals.values;
SimulatedTemperature = simTemp.signals.values;
SimulatedSetPoint = simSetPoint.signals.values;

%We need to figure out when the step change occured in
the simulation.
for SimulatedSetPointIndex=1:length(SimulatedSetPoint)
    currentSetPoint =
SimulatedSetPoint(SimulatedSetPointIndex);
    if currentSetPoint == max(SimulatedSetPoint)
        break
    else
    end
end

SimulatedSetPointStepTime =
SimulatedTime(SimulatedSetPointIndex);

%We need to figure out when the step change occurs in
the experiment
for SetPoint_A_Index=1:length(SetPoint_A)
    currentSetPoint_A = SetPoint_A(SetPoint_A_Index);
    if currentSetPoint_A == max(SetPoint_A)
        break
    else
    end
end

SetPointStepTime = time_A(SetPoint_A_Index);

sprintf('In lab, 1 degree step change was applied at t
= %3.2f\n\nIn Simulink, 1 degree step change was
applied at t =
%3.2f',SetPointStepTime,SimulatedSetPointStepTime)
sprintf('In lab, the data was acquired with the
following parameters\n\n  Kp = %3.3f\n  Ki = %3.3f\n
Kd = %3.3f',Kp_A(1),Ki_A(1),Kd_A(1))


%What is the time difference between the two steps?
TimeDifference = SimulatedSetPointStepTime -
SetPointStepTime;

%We can plot this but we need to shift the time of the
simulation such that
%the steps occur at the same time.

figure
plot(time_A,Tsensor_A,SimulatedTime -
TimeDifference,SimulatedTemperature,time_A,SetPoint_A)
title('Temperature vs. Time with PID Control.
K_p=44.1, K_i=4.009, K_d=121.275')
xlabel('Time (sec)')
ylabel('Temperature (F)')
legend('Actual','Simulation','Set Point',4)
axis([(SetPointStepTime-3) (SetPointStepTime+75)
(min(SetPoint_A)-.25) (max(SetPoint_A)+1.5)])
grid
```

PID Performance

```
disp(' ');
disp('Warning! This will clear all variables and close
all plots!');
choice = input('Do you wish to continue? (1=yes,
0=no)');
if choice ~= 1
    break;
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%
%Ziegler-Nichols Oscillation Data

close all
clear
clc
load '4_b_A.mat'
pull_vals

Ts_low = 150;
Ts_high = 151;

%Find when the two step inputs were applied
t_upstep = t(findstep(Ts, Ts_low, Ts_high));
t_downstep = t(findstep(Ts, Ts_high, Ts_low));
%plot(t,Ts,[t_upstep, t_upstep],[150,151],[t_downstep,
t_downstep],[150,151])
%legend('Ts','UpStep','DownStep')
%title('Step Change Detection Verification')

t_up_i0 = find(t==t_upstep);            %Index for
when Ts went high
t_up_if = find(t==t_downstep) - 1;      %End index for
Ts high
t_up_offset = t(t_up_i0);
t_up = t(t_up_i0:t_up_if) - t_up_offset;
%Data for Ts high
T_up = T(t_up_i0:t_up_if);
P_up = P(t_up_i0:t_up_if);
I_up = I(t_up_i0:t_up_if);
D_up = D(t_up_i0:t_up_if);
Terr_up = Terr(t_up_i0:t_up_if);
Ts_up = Ts(t_up_i0:t_up_if);


t_dn_i0 = find(t==t_downstep);          %Index for
when Ts went low
t_dn_if = length(t);                    %End index for
Ts low
t_dn_offset = t(t_dn_i0);
t_dn = t(t_dn_i0:t_dn_if) - t_dn_offset;
%Data for Ts high
T_dn = T(t_dn_i0:t_dn_if);
P_dn = P(t_dn_i0:t_dn_if);
I_dn = I(t_dn_i0:t_dn_if);
D_dn = D(t_dn_i0:t_dn_if);
Terr_dn = Terr(t_dn_i0:t_dn_if);
Ts_dn = Ts(t_dn_i0:t_dn_if);

%Positive Step Change
          tic;
    %Percent Overshoot
          [t_up_POval t_up_PO] = max(T_up);
          t_up_PO = t_up(t_up_PO);
    PO_up = ((max(T_up) - Ts_low) * (1/(Ts_high-
Ts_low)) - 1) * 100;
          %plot(t_up,Ts_up,'--g',t_up,T_up,'-
b',t_up_PO,t_up_POval,'xr')
          %axis([min(t_up) max(t_up) .9975*min(T_up)
1.0025*max(T_up)])


          %Rise Time
          t_up_rise = find(T_up>=Ts_high);
          t_up_rise = t_up(t_up_rise(1));
          t_up_riseval = Ts_high;
          %plot(t_up,Ts_up,'--g',t_up,T_up,'-
b',t_up_rise,t_up_riseval,'xr')
          %axis([min(t_up) max(t_up) .9975*min(T_up)
1.0025*max(T_up)])

          %Settling Time
          band = (1/50) * 1/100;
          t_up_ub = ones(size(t_up)) * (1 + band) *
Ts_high;
          t_up_lb = ones(size(t_up)) * (1 - band) *
Ts_high;

          inflag = 0;
          for i = 1:length(t_up)
        if (T_up(i) >= (1 - band) * Ts_high) &
(T_up(i) <= (1 + band) * Ts_high)
            if inflag == 0
                inflag = i;
            end
        else
            inflag = 0;
        end
          end
          t_up_settle = t_up(inflag);
          t_up_settleval = T_up(inflag);
%         plot(t_up,Ts_up,'--g',...
%         t_up,T_up,'-b',...
%        t_up_settle,t_up_settleval,'xr',...
%        t_up,t_up_ub,':g',...
%        t_up,t_up_lb,':g')
%         axis([min(t_up) max(t_up) .9975*min(T_up)
1.0025*max(T_up)])

    figure;
    plot(t_up,Ts_up,'-.b',...
        t_up,t_up_ub,':b',...
        t_up,t_up_lb,':b',...
        t_up, T_up,'-k',...
        t_up_PO,t_up_POval,'*r',...
        t_up_rise,t_up_riseval,'.r',...
        t_up_settle,t_up_settleval,'+r')
          axis([min(t_up) max(t_up) .9975*min(T_up)
1.0025*max(T_up)])
          legend('Temperature Setpoint',...
        'Settling Upper Bound',...
        'Settling Lower Bound',...
        'Measured Temperature',...
        ['Peak Value: PO = ' num2str(PO_up,'%0.4g')
'%'],...
        ['Rise Time = ' num2str(t_up_rise) 's'],...
        ['Settling Time (+/-' num2str(band * 100) '%)
= ' num2str(t_up_settle) 's'],4)
          xlabel('Time [s]')
          ylabel('Temperature [deg F]')
    title('Postive Step Change Response and
Performance')

    disp(' ')
          disp('Positive Step Change Performance:')
          disp([' Peak Value: PO = '
num2str(PO_up,'%0.4g') '%'])
          disp([' Rise Time = ' num2str(t_up_rise)
's'])
          disp([' Settling Time (+/-' num2str(band *
100) '%) = ' num2str(t_up_settle) 's'])
    disp([' Calc Time: ' num2str(toc) 's'])
    print -djpeg100 'Z-N Osc-Up.jpg'

%Negative Step Change
    tic;
    %Percent Overshoot
          [t_dn_POval t_dn_PO] = min(T_dn);
          t_dn_PO = t_dn(t_dn_PO);
    PO_dn = ((min(T_dn) - Ts_high) * (1/(Ts_low-
Ts_high))-1) * 100;
          %plot(t_dn,Ts_dn,'--g',t_dn,T_dn,'-
b',t_dn_PO,t_dn_POval,'xr')
          %axis([min(t_dn) max(t_dn) .9975*min(T_dn)
1.0025*max(T_dn)])


          %Fall Time
          t_dn_fall = find(T_dn<=Ts_low);
          t_dn_fall = t_dn(t_dn_fall(1));
          t_dn_fallval = Ts_low;
          %plot(t_dn,Ts_dn,'--g',t_dn,T_dn,'-
b',t_dn_fall,t_dn_fallval,'xr')
          %axis([min(t_dn) max(t_dn) .9975*min(T_dn)
1.0025*max(T_dn)])

          %Settling Time
          band = (1/50) * 1/100;
          t_dn_ub = ones(size(t_dn)) * (1 + band) *
Ts_low;
          t_dn_lb = ones(size(t_dn)) * (1 - band) *
Ts_low;

          inflag = 0;
          for i = 1:length(t_dn)
        if (T_dn(i) >= (1 - band) * Ts_low) & (T_dn(i)
<= (1 + band) * Ts_low)
            if inflag == 0
                inflag = i;
            end
        else
            inflag = 0;
        end
          end
          t_dn_settle = t_dn(inflag);
          t_dn_settleval = T_dn(inflag);
%         plot(t_dn,Ts_dn,'--g',...
%         t_dn,T_dn,'-b',...
%        t_dn_settle,t_dn_settleval,'xr',...
%        t_dn,t_dn_ub,':g',...
%        t_dn,t_dn_lb,':g')
%         axis([min(t_dn) max(t_dn) .9975*min(T_dn)
1.0025*max(T_dn)])

    figure;
    plot(t_dn,Ts_dn,'-.b',...
        t_dn,t_dn_ub,':b',...
        t_dn,t_dn_lb,':b',...
        t_dn, T_dn,'-k',...
        t_dn_PO,t_dn_POval,'*r',...
        t_dn_fall,t_dn_fallval,'.r',...
        t_dn_settle,t_dn_settleval,'+r')
          axis([min(t_dn) max(t_dn) .9975*min(T_dn)
1.0025*max(T_dn)])
          legend('Temperature Setpoint',...
        'Settling Upper Bound',...
        'Settling Lower Bound',...
        'Measured Temperature',...
        ['Peak Value: PO = ' num2str(PO_dn,'%0.4g')
'%'],...
        ['Fall Time = ' num2str(t_dn_fall) 's'],...
```

```
        ['Settling Time (+/-' num2str(band * 100) '%)
= ' num2str(t_dn_settle) 's'],1)
          xlabel('Time [s]')
          ylabel('Temperature [deg F]')
    title('Negative Step Change Response and
Performance')

    disp(' ')
          disp('Negative Step Change Performance:')
          disp([' Peak Value: PO = '
num2str(PO_dn,'%0.4g') '%'])
          disp([' Fall Time = ' num2str(t_dn_fall)
's'])
          disp([' Settling Time (+/-' num2str(band *
100) '%) = ' num2str(t_dn_settle) 's'])
    disp([' Calc Time: ' num2str(toc) 's'])
    print -djpeg100 'Z-N Osc-Dn.jpg'

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%
%Ziegler-Nichols Model Data

close all
clear
load '4_b_B_measured.mat'
pull_vals

Ts_low = 150;
Ts_high = 151;

%Find when the two step inputs were applied
t_upstep = t(findstep(Ts, Ts_low, Ts_high));
t_downstep = t(findstep(Ts, Ts_high, Ts_low));
%plot(t,Ts,[t_upstep, t_upstep],[150,151],[t_downstep,
t_downstep],[150,151])
%legend('Ts','UpStep','DownStep')
%title('Step Change Detection Verification')

t_up_i0 = find(t==t_upstep);              %Index for
when Ts went high
t_up_if = find(t==t_downstep) - 1;        %End index for
Ts high
t_up_offset = t(t_up_i0);
t_up = t(t_up_i0:t_up_if) - t_up_offset;
%Data for Ts high
T_up = T(t_up_i0:t_up_if);
P_up = P(t_up_i0:t_up_if);
I_up = I(t_up_i0:t_up_if);
D_up = D(t_up_i0:t_up_if);
Terr_up = Terr(t_up_i0:t_up_if);
Ts_up = Ts(t_up_i0:t_up_if);


t_dn_i0 = find(t==t_downstep);            %Index for
when Ts went low
t_dn_if = length(t);                      %End index for
Ts low
t_dn_offset = t(t_dn_i0);
t_dn = t(t_dn_i0:t_dn_if) - t_dn_offset;
%Data for Ts high
T_dn = T(t_dn_i0:t_dn_if);
P_dn = P(t_dn_i0:t_dn_if);
I_dn = I(t_dn_i0:t_dn_if);
D_dn = D(t_dn_i0:t_dn_if);
Terr_dn = Terr(t_dn_i0:t_dn_if);
Ts_dn = Ts(t_dn_i0:t_dn_if);

%Positive Step Change
          tic;
    %Percent Overshoot
          [t_up_POval t_up_PO] = max(T_up);
          t_up_PO = t_up(t_up_PO);
    PO_up = ((max(T_up) - Ts_low) * (1/(Ts_high-
Ts_low)) - 1) * 100;
          %plot(t_up,Ts_up,'--g',t_up,T_up,'-
b',t_up_PO,t_up_POval,'xr')
          %axis([min(t_up) max(t_up) .9975*min(T_up)
1.0025*max(T_up)])


          %Rise Time
          t_up_rise = find(T_up>=Ts_high);
          t_up_rise = t_up(t_up_rise(1));
          t_up_riseval = Ts_high;
          %plot(t_up,Ts_up,'--g',t_up,T_up,'-
b',t_up_rise,t_up_riseval,'xr')
          %axis([min(t_up) max(t_up) .9975*min(T_up)
1.0025*max(T_up)])

          %Settling Time
          band = (1/50) * 1/100;
          t_up_ub = ones(size(t_up)) * (1 + band) *
Ts_high;
          t_up_lb = ones(size(t_up)) * (1 - band) *
Ts_high;

          inflag = 0;
          for i = 1:length(t_up)
        if (T_up(i) >= (1 - band) * Ts_high) &
(T_up(i) <= (1 + band) * Ts_high)
            if inflag == 0
                inflag = i;
            end
        else
            inflag = 0;
        end
          end
          t_up_settle = t_up(inflag);
          t_up_settleval = T_up(inflag);
%         plot(t_up,Ts_up,'--g',...
%         t_up,T_up,'-b',...
%        t_up_settle,t_up_settleval,'xr',...
%        t_up,t_up_ub,':g',...
%        t_up,t_up_lb,':g')
%         axis([min(t_up) max(t_up) .9975*min(T_up)
1.0025*max(T_up)])

    figure;
    plot(t_up,Ts_up,'-.b',...
        t_up,t_up_ub,':b',...
        t_up,t_up_lb,':b',...
        t_up, T_up,'-k',...
        t_up_PO,t_up_POval,'*r',...
        t_up_rise,t_up_riseval,'.r',...
        t_up_settle,t_up_settleval,'+r')
          axis([min(t_up) max(t_up) .9975*min(T_up)
1.0025*max(T_up)])
          legend('Temperature Setpoint',...
        'Settling Upper Bound',...
        'Settling Lower Bound',...
        'Measured Temperature',...
        ['Peak Value: PO = ' num2str(PO_up,'%0.4g')
'%'],...
        ['Rise Time = ' num2str(t_up_rise) 's'],...
        ['Settling Time (+/-' num2str(band * 100) '%)
= ' num2str(t_up_settle) 's'],4)
          xlabel('Time [s]')
          ylabel('Temperature [deg F]')
    title('Postive Step Change Response and
Performance')

    disp(' ')
          disp('Positive Step Change Performance:')
          disp([' Peak Value: PO = '
num2str(PO_up,'%0.4g') '%'])
          disp([' Rise Time = ' num2str(t_up_rise)
's'])
          disp([' Settling Time (+/-' num2str(band *
100) '%) = ' num2str(t_up_settle) 's'])
    disp([' Calc Time: ' num2str(toc) 's'])
    print -djpeg100 'Z-N Mdl-Up.jpg'

%Negative Step Change
    tic;
    %Percent Overshoot
          [t_dn_POval t_dn_PO] = min(T_dn);
          t_dn_PO = t_dn(t_dn_PO);
    PO_dn = ((min(T_dn) - Ts_high) * (1/(Ts_low-
Ts_high))-1) * 100;
          %plot(t_dn,Ts_dn,'--g',t_dn,T_dn,'-
b',t_dn_PO,t_dn_POval,'xr')
          %axis([min(t_dn) max(t_dn) .9975*min(T_dn)
1.0025*max(T_dn)])


          %Fall Time
          t_dn_fall = find(T_dn<=Ts_low);
          t_dn_fall = t_dn(t_dn_fall(1));
          t_dn_fallval = Ts_low;
          %plot(t_dn,Ts_dn,'--g',t_dn,T_dn,'-
b',t_dn_fall,t_dn_fallval,'xr')
          %axis([min(t_dn) max(t_dn) .9975*min(T_dn)
1.0025*max(T_dn)])

          %Settling Time
          band = (1/50) * 1/100;
          t_dn_ub = ones(size(t_dn)) * (1 + band) *
Ts_low;
          t_dn_lb = ones(size(t_dn)) * (1 - band) *
Ts_low;

          inflag = 0;
          for i = 1:length(t_dn)
        if (T_dn(i) >= (1 - band) * Ts_low) & (T_dn(i)
<= (1 + band) * Ts_low)
            if inflag == 0
                inflag = i;
            end
        else
            inflag = 0;
        end
          end
          t_dn_settle = t_dn(inflag);
          t_dn_settleval = T_dn(inflag);
%         plot(t_dn,Ts_dn,'--g',...
%         t_dn,T_dn,'-b',...
%        t_dn_settle,t_dn_settleval,'xr',...
%        t_dn,t_dn_ub,':g',...
%        t_dn,t_dn_lb,':g')
%         axis([min(t_dn) max(t_dn) .9975*min(T_dn)
1.0025*max(T_dn)])

    figure;
    plot(t_dn,Ts_dn,'-.b',...
        t_dn,t_dn_ub,':b',...
        t_dn,t_dn_lb,':b',...
        t_dn, T_dn,'-k',...
        t_dn_PO,t_dn_POval,'*r',...
        t_dn_fall,t_dn_fallval,'.r',...
        t_dn_settle,t_dn_settleval,'+r')
          axis([min(t_dn) max(t_dn) .9975*min(T_dn)
1.0025*max(T_dn)])
          legend('Temperature Setpoint',...
        'Settling Upper Bound',...
        'Settling Lower Bound',...
        'Measured Temperature',...
```

```
        ['Peak Value: PO = ' num2str(PO_dn,'%0.4g')
'%'],...
        ['Fall Time = ' num2str(t_dn_fall) 's'],...
        ['Settling Time (+/-' num2str(band * 100) '%)
= ' num2str(t_dn_settle) 's'],1)
          xlabel('Time [s]')
          ylabel('Temperature [deg F]')
    title('Negative Step Change Response and
Performance')

    disp(' ')
          disp('Negative Step Change Performance:')
          disp([' Peak Value: PO = '
num2str(PO_dn,'%0.4g') '%'])
          disp([' Fall Time = ' num2str(t_dn_fall)
's'])
          disp([' Settling Time (+/-' num2str(band *
100) '%) = ' num2str(t_dn_settle) 's'])
    disp([' Calc Time: ' num2str(toc) 's'])
    print -djpeg100 'Z-N Mdl-Dn.jpg'

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%
%Root-Locus Data

close all
clear
load '4_b_C.mat'
pull_vals

Ts_low = 150;
Ts_high = 151;

%Find when the two step inputs were applied
t_upstep = t(findstep(Ts, Ts_low, Ts_high));
t_downstep = t(findstep(Ts, Ts_high, Ts_low));
%plot(t,Ts,[t_upstep, t_upstep],[150,151],[t_downstep,
t_downstep],[150,151])
%legend('Ts','UpStep','DownStep')
%title('Step Change Detection Verification')

t_up_i0 = find(t==t_upstep);            %Index for
when Ts went high
t_up_if = find(t==t_downstep) - 1;      %End index for
Ts high
t_up_offset = t(t_up_i0);
t_up = t(t_up_i0:t_up_if) - t_up_offset;
%Data for Ts high
T_up = T(t_up_i0:t_up_if);
P_up = P(t_up_i0:t_up_if);
I_up = I(t_up_i0:t_up_if);
D_up = D(t_up_i0:t_up_if);
Terr_up = Terr(t_up_i0:t_up_if);
Ts_up = Ts(t_up_i0:t_up_if);


t_dn_i0 = find(t==t_downstep);          %Index for
when Ts went low
t_dn_if = length(t);                    %End index for
Ts low
t_dn_offset = t(t_dn_i0);
t_dn = t(t_dn_i0:t_dn_if) - t_dn_offset;
%Data for Ts high
T_dn = T(t_dn_i0:t_dn_if);
P_dn = P(t_dn_i0:t_dn_if);
I_dn = I(t_dn_i0:t_dn_if);
D_dn = D(t_dn_i0:t_dn_if);
Terr_dn = Terr(t_dn_i0:t_dn_if);
Ts_dn = Ts(t_dn_i0:t_dn_if);

%Positive Step Change
          tic;
    %Percent Overshoot
          [t_up_POval t_up_PO] = max(T_up);
          t_up_PO = t_up(t_up_PO);
    PO_up = ((max(T_up) - Ts_low) * (1/(Ts_high-
Ts_low)) - 1) * 100;
          %plot(t_up,Ts_up,'--g',t_up,T_up,'-
b',t_up_PO,t_up_POval,'xr')
          %axis([min(t_up) max(t_up) .9975*min(T_up)
1.0025*max(T_up)])


          %Rise Time
          t_up_rise = find(T_up>=Ts_high);
          t_up_rise = t_up(t_up_rise(1));
          t_up_riseval = Ts_high;
          %plot(t_up,Ts_up,'--g',t_up,T_up,'-
b',t_up_rise,t_up_riseval,'xr')
          %axis([min(t_up) max(t_up) .9975*min(T_up)
1.0025*max(T_up)])

          %Settling Time
          band = (1/50) * 1/100;
          t_up_ub = ones(size(t_up)) * (1 + band) *
Ts_high;
          t_up_lb = ones(size(t_up)) * (1 - band) *
Ts_high;

          inflag = 0;
          for i = 1:length(t_up)
        if (T_up(i) >= (1 - band) * Ts_high) &
(T_up(i) <= (1 + band) * Ts_high)
            if inflag == 0
                inflag = i;
            end
        else
            inflag = 0;
        end
          end
          t_up_settle = t_up(inflag);
          t_up_settleval = T_up(inflag);
%         plot(t_up,Ts_up,'--g',...
%         t_up,T_up,'-b',...
%        t_up_settle,t_up_settleval,'xr',...
%        t_up,t_up_ub,':g',...
%        t_up,t_up_lb,':g')
%         axis([min(t_up) max(t_up) .9975*min(T_up)
1.0025*max(T_up)])

    figure;
    plot(t_up,Ts_up,'-.b',...
        t_up,t_up_ub,':b',...
        t_up,t_up_lb,':b',...
        t_up, T_up,'-k',...
        t_up_PO,t_up_POval,'*r',...
        t_up_rise,t_up_riseval,'.r',...
        t_up_settle,t_up_settleval,'+r')
          axis([min(t_up) max(t_up) .9975*min(T_up)
1.0025*max(T_up)])
          legend('Temperature Setpoint',...
        'Settling Upper Bound',...
        'Settling Lower Bound',...
        'Measured Temperature',...
        ['Peak Value: PO = ' num2str(PO_up,'%0.4g')
'%'],...
        ['Rise Time = ' num2str(t_up_rise) 's'],...
        ['Settling Time (+/-' num2str(band * 100) '%)
= ' num2str(t_up_settle) 's'],4)
          xlabel('Time [s]')
          ylabel('Temperature [deg F]')
    title('Postive Step Change Response and
Performance')

    disp(' ')
          disp('Positive Step Change Performance:')
          disp([' Peak Value: PO = '
num2str(PO_up,'%0.4g') '%'])
          disp([' Rise Time = ' num2str(t_up_rise)
's'])
          disp([' Settling Time (+/-' num2str(band *
100) '%) = ' num2str(t_up_settle) 's'])
    disp([' Calc Time: ' num2str(toc) 's'])
    print -djpeg100 'R-L Up.jpg'

%Negative Step Change
    tic;
    %Percent Overshoot
          [t_dn_POval t_dn_PO] = min(T_dn);
          t_dn_PO = t_dn(t_dn_PO);
    PO_dn = ((min(T_dn) - Ts_high) * (1/(Ts_low-
Ts_high))-1) * 100;
          %plot(t_dn,Ts_dn,'--g',t_dn,T_dn,'-
b',t_dn_PO,t_dn_POval,'xr')
          %axis([min(t_dn) max(t_dn) .9975*min(T_dn)
1.0025*max(T_dn)])


          %Fall Time
          t_dn_fall = find(T_dn<=Ts_low);
          t_dn_fall = t_dn(t_dn_fall(1));
          t_dn_fallval = Ts_low;
          %plot(t_dn,Ts_dn,'--g',t_dn,T_dn,'-
b',t_dn_fall,t_dn_fallval,'xr')
          %axis([min(t_dn) max(t_dn) .9975*min(T_dn)
1.0025*max(T_dn)])

          %Settling Time
          band = (1/50) * 1/100;
          t_dn_ub = ones(size(t_dn)) * (1 + band) *
Ts_low;
          t_dn_lb = ones(size(t_dn)) * (1 - band) *
Ts_low;

          inflag = 0;
          for i = 1:length(t_dn)
        if (T_dn(i) >= (1 - band) * Ts_low) & (T_dn(i)
<= (1 + band) * Ts_low)
            if inflag == 0
                inflag = i;
            end
        else
            inflag = 0;
        end
          end
          t_dn_settle = t_dn(inflag);
          t_dn_settleval = T_dn(inflag);
%         plot(t_dn,Ts_dn,'--g',...
%         t_dn,T_dn,'-b',...
%        t_dn_settle,t_dn_settleval,'xr',...
%        t_dn,t_dn_ub,':g',...
%        t_dn,t_dn_lb,':g')
%         axis([min(t_dn) max(t_dn) .9975*min(T_dn)
1.0025*max(T_dn)])

    figure;
    plot(t_dn,Ts_dn,'-.b',...
        t_dn,t_dn_ub,':b',...
        t_dn,t_dn_lb,':b',...
        t_dn, T_dn,'-k',...
        t_dn_PO,t_dn_POval,'*r',...
        t_dn_fall,t_dn_fallval,'.r',...
        t_dn_settle,t_dn_settleval,'+r')
          axis([min(t_dn) max(t_dn) .9975*min(T_dn)
1.0025*max(T_dn)])
          legend('Temperature Setpoint',...
```

```
        'Settling Upper Bound',...
        'Settling Lower Bound',...
        'Measured Temperature',...
        ['Peak Value: PO = ' num2str(PO_dn,'%0.4g')
'%'],...
        ['Fall Time = ' num2str(t_dn_fall) 's'],...
        ['Settling Time (+/-' num2str(band * 100) '%)
= ' num2str(t_dn_settle) 's'],1)
          xlabel('Time [s]')
          ylabel('Temperature [deg F]')
    title('Negative Step Change Response and
Performance')

    disp(' ')
          disp('Negative Step Change Performance:')
          disp([' Peak Value: PO = '
num2str(PO_dn,'%0.4g') '%'])
          disp([' Fall Time = ' num2str(t_dn_fall)
's'])
          disp([' Settling Time (+/-' num2str(band *
100) '%) = ' num2str(t_dn_settle) 's'])
    disp([' Calc Time: ' num2str(toc) 's'])
    print -djpeg100 'R-L Dn.jpg'



function [ index ] = FindStep( dat, val_0, val_f )

f_low = 0;

for i = 1:length(dat)
    if dat(i) == val_0
        f_low = 1;
    elseif dat(i) == val_f & f_low == 1
        index = i;
        break
    end
end



t = tempdata(:,1) / 1000;
Ts = tempdata(:,2);
T = tempdata(:,3);
Terr = tempdata(:,4);
Duty = tempdata(:,5);
Kp = tempdata(:,6);
Ki = tempdata(:,7);
Kd = tempdata(:,8);
P= tempdata(:,9);
I = tempdata(:,10);
D = tempdata(:,11);
```

Lab Discussion Part 2

```
%-------------------LABORATORY DISCUSSION ITEMS-------
-------------------
%clear the command window and close graphs, but do not
clear variables
%since we need the simout variable from Simulink.
clc
close all

%(2) Plot the open-loop response to a step in duty
cycle of magnitude 8.

%We need to first load the data from the step response
from the lab
load step_change_second

time_step =          tempdata(:,1)/1000;     %Extract
time and convert to seconds
SetPoint_step =      tempdata(:,2);          %Extract
set point temperature
Tsensor_step =       tempdata(:,3);          %Extract
sensor temperature
Terror_step =        tempdata(:,4);          %Extract
temperature error
DutyCycle_step =     tempdata(:,5);          %Extract
duty cycle
Kp_step =            tempdata(:,6);          %Extract
proportional gain value
Ki_step =            tempdata(:,7);          %Extract
integral gain value
Kd_step =            tempdata(:,8);          %Extract
derivative gain value
Proportional_step =  tempdata(:,9);          %Extract
proportional contribution
Integral_step =      tempdata(:,10);         %Extract
integral contribution
Derivative_step =    tempdata(:,11);         %Extract
derivative contribution

%We can also load the data from the simulation
%Extract the simulated time, duty cycle, temperature,
and setpoint.
SimulatedTime = simDutyCycle.time;
SimulatedDutyCycle = simDutyCycle.signals.values;
SimulatedTemperature = simTemp.signals.values;
SimulatedSetPoint = simSetPoint.signals.values;

%We need to figure out when the step change occured in
the simulation.
for
SimulatedDutyCycleIndex=1:length(SimulatedDutyCycle)
    currentDutyCycle =
SimulatedDutyCycle(SimulatedDutyCycleIndex);
    if currentDutyCycle == max(SimulatedDutyCycle)
        break
    else
    end
end

SimulatedDutyCycleStepTime =
SimulatedTime(SimulatedDutyCycleIndex);

%We need to figure out when the step change occurs in
the experiment
for DutyCycle_step_Index=1:length(DutyCycle_step)
    currentDutyCycle =
DutyCycle_step(DutyCycle_step_Index);
    if currentDutyCycle == max(DutyCycle_step)
        break
    else
    end
end

DutyCycleStepTime = time_step(DutyCycle_step_Index);

%We would like to find out what the temperature was
when the step change
%was applied
TempStep = Tsensor_step(DutyCycle_step_Index);

sprintf('In lab, duty cycle step change was applied at
t = %3.2f\n\nIn Simulink, duty cycle step change was
applied at t = %3.2f\n\n  The temperature in lab when
the step was applied was
%3.2f',DutyCycleStepTime,SimulatedDutyCycleStepTime,Te
mpStep)

%What is the time difference between the two steps?
TimeDifference = SimulatedDutyCycleStepTime -
DutyCycleStepTime;

%We can plot this but we need to shift the time of the
simulation such that
%the steps occur at the same time.
figure
plot(time_step,Tsensor_step,SimulatedTime -
TimeDifference,SimulatedTemperature,time_step,DutyCycl
e_step+100)
title('Temperature vs. Time for 8% Duty Cycle Step
Change (From 37% to 45%)')
xlabel('Time (sec)')
ylabel('Temperature (F)')
legend('Actual','Simulation','DutyCycle',2)
axis([(DutyCycleStepTime-50) (DutyCycleStepTime+900)
142 156])
grid

%We should find out why the simulation does not reach
a steady state value
%of TempStep.  We need to find the final value with a
duty cycle of 37%
uo = 37;
FinalValue = ((k1/k2)*((Vref8^2)/R)*uo)+offset8;

sprintf('The final value of the system with a duty
cycle of %2.2f is %3.3f',uo,FinalValue)
```

Lab Discussion Part 4

```
%-------------------LABORATORY DISCUSSION ITEMS-------
-------------------
%clear the command window and close graphs, but do not
clear variables
%since we need the simout variable from Simulink.
clc
close all

%(4)  Evaluate the robutness of the three PID control
esigns in Part 4.b in
%terms of gain and phase margin.
%
%We need to first input model paramters
k1 = 0.00245496;
k2 = 0.00416146;
Tau = 6.39882571;

offset8 = 86.88247312

Vref8 = 8;
R = 25;

%We now need to input the gain values that were used
in part 4.b

%Ziegler-Nichols Oscillation
KpA = 44.1;
KdA = 121.275;
KiA = 4.009;

%Ziegler-Nichols Model Based
KpB = 36.39;
KdB = 93.44;
KiB = 3.64;
```

```
%Root Locus Method
KpC = 14.4;
KdC = 13.1;
KiC = 0.603;

%We can now calculate the transfer function of the
plant using a 4th order
%pade approximation for the time delay
PadeApproximationOrder = 4;

[num,den] = pade(Tau,PadeApproximationOrder);
delay = tf(num,den);

%Now calculate the transfer fucntion of the block
dynamics
block = tf([k1],[1 k2]);

%the transfer function of the plant is the block
dynamics, and delay, and
%the constant gain
plant = block*delay*((Vref8^2)/R);

%We can now calculate the transfer function for the
PID controller.  Recall
%that the controller given by
%
%   Kp + s*Kd + Ki/s
%
%can be rewritten as
%
%   (Kd*s^2 + Kp*s + Ki)/s)

PID_A = tf([KdA KpA KiA],[1 0]);
PID_B = tf([KdB KpB KiB],[1 0]);
PID_C = tf([KdC KpC KiC],[1 0]);

%We can now calculate multiply the transfer fuctions
of the plant and the
%PID controllers together to obtain the open loop
transfer function
G_A = plant*PID_A;
G_B = plant*PID_B;
G_C = plant*PID_C;

%recall that the bode plots are most often drawn for
open loop transfer
%functions since the signal is fed back.

%We can now plot the bode diagrams for the systems.
Note that these may
%not be the actual bode diagrams of the system because
the saturation is
%not taken into account.
figure
bode(G_A)
title('Open Loop Bode Diagram For System with
K_p=44.1, K_d=121.275, and K_i=4.009')

figure
bode(G_B)
title('Open Loop Bode Diagram For System with
K_p=36.39, K_d=93.44, and K_i=3.64')

figure
bode(G_C)
title('Open Loop Bode Diagram For System with
K_p=14.4, K_d=13.1, and K_i=0.603')
```

Lab Discussion Part 5

```
%-------------------LABORATORY DISCUSSION ITEMS-------
-------------------
%clear the command window and close graphs, but do not
clear variables
%since we need the simout variable from Simulink.
clc
close all

%(5)  We can plot the duty cycle and all of the
contributing components in
%response to a step.

%The gains derived are shown below
%Ziegler-Nichols Oscillation
KpA = 44.1;
KdA = 121.275;
KiA = 4.009;

%Ziegler-Nichols Model Based
KpB = 36.39;
KdB = 93.44;
KiB = 3.64;

%Root Locus Method
KpC = 14.4;
KdC = 13.1;
KiC = 0.603;

%-------------------------Ziegler Nichols Oscillation-
------------
%We can run the simulation using the first set of
gains
Kp = KpA;
Kd = KdA;
Ki = KiA;

sim('optimal_values')

%We need to load the simulated time, duty cycle, and
components from the
%workspace.
SimulatedTimeA = simDutyCycle.time;
SimulatedSetPointA = simSetPoint.signals.values;
SimulatedDutyCycleA = simDutyCycle.signals.values;
SimulatedKcomponentA = simKcomponent.signals.values;
SimulatedDcomponentA = simDcomponent.signals.values;
SimulatedIcomponentA = simIcomponent.signals.values;

%We need to figure out when the 1 degree step change
occured in the simulation.
for SimulatedSetPointIndex=1:length(SimulatedSetPoint)
    currentSetPoint =
SimulatedSetPoint(SimulatedSetPointIndex);
    if currentSetPoint == max(SimulatedSetPoint)
        break
    else
    end
end

SimulatedSetPointStepTime =
SimulatedTime(SimulatedSetPointIndex);

%-------------------------Ziegler Nichols Model-Based-
------------
%We can run the simulation using the second set of
gains
Kp = KpB;
Kd = KdB;
Ki = KiB;

sim('optimal_values')

%We need to load the simulated time, duty cycle, and
components from the
%workspace.
SimulatedTimeB = simDutyCycle.time;
SimulatedSetPointB = simSetPoint.signals.values;
SimulatedDutyCycleB = simDutyCycle.signals.values;
SimulatedKcomponentB = simKcomponent.signals.values;
SimulatedDcomponentB = simDcomponent.signals.values;
SimulatedIcomponentB = simIcomponent.signals.values;


%---------------------Root Locus----------------------
----------
%We can run the simulation using the third set of
gains
Kp = KpC;
Kd = KdC;
Ki = KiC;

sim('optimal_values')

%We need to load the simulated time, duty cycle, and
components from the
%workspace.
SimulatedTimeC = simDutyCycle.time;
SimulatedSetPointC = simSetPoint.signals.values;
SimulatedDutyCycleC = simDutyCycle.signals.values;
SimulatedKcomponentC = simKcomponent.signals.values;
SimulatedDcomponentC = simDcomponent.signals.values;
SimulatedIcomponentC = simIcomponent.signals.values;



%We can easily plot the simulated duty cycle
plotting_time = 100;        %how many seconds after
step do you want to plot?

figure
plot(SimulatedTimeA,SimulatedDutyCycleA,SimulatedTimeB
,SimulatedDutyCycleB,SimulatedTimeC,SimulatedDutyCycle
C)
title('Simulated Duty Cycle vs. Time')
xlabel('Time (sec)')
ylabel('Duty Cycle (%)')
axis([SimulatedSetPointStepTime
SimulatedSetPointStepTime+plotting_time 0 100])
legend('Ziegler-Nichols Oscillation','Ziegler-Nichols
Model','Root Locus')

%We can also plot the K, D, and I contributions
figure
plot(SimulatedTimeA,SimulatedKcomponentA,SimulatedTime
B,SimulatedKcomponentB,SimulatedTimeC,SimulatedKcompon
entC)
title('Simulated K Component vs. Time')
xlabel('Time(sec)')
ylabel('K Contribution')
axis([SimulatedSetPointStepTime
SimulatedSetPointStepTime+plotting_time -30 30])
legend('Ziegler-Nichols Oscillation','Ziegler-Nichols
Model','Root Locus')

figure
plot(SimulatedTimeA,SimulatedDcomponentA,SimulatedTime
B,SimulatedDcomponentB,SimulatedTimeC,SimulatedDcompon
entC)
title('Simulated D Component vs. Time')
xlabel('Time(sec)')
ylabel('D Contribution')
```

```
axis([SimulatedSetPointStepTime
SimulatedSetPointStepTime+plotting_time -10 15])
legend('Ziegler-Nichols Oscillation','Ziegler-Nichols
Model','Root Locus')

figure
plot(SimulatedTimeA,SimulatedIcomponentA,SimulatedTime
B,SimulatedIcomponentB,SimulatedTimeC,SimulatedIcompon
entC)
title('Simulated I Component vs. Time')
xlabel('Time(sec)')
ylabel('I Contribution')
axis([SimulatedSetPointStepTime
SimulatedSetPointStepTime+plotting_time 30 80])
legend('Ziegler-Nichols Oscillation','Ziegler-Nichols
Model','Root Locus')
```

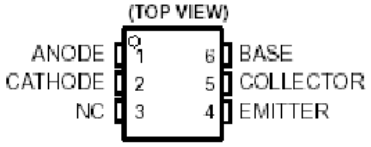
4N35:

4N35 Datasheet [Complete 4N35 Datasheet]

COMPATIBLE WITH STANDARD TTL INTEGRATED CIRCUITS

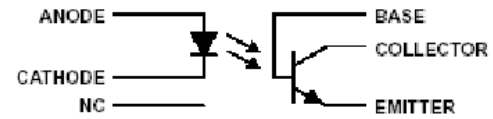
- Gallium-Arsenide-Diode Infrared Source Optically Coupled to a Silicon npn Phototransistor
- High Direct-Current Transfer Ratio
- High-Voltage Electrical Isolation
  1.5-kV, 2.5-kV, or 3.55-kV Rating
- High-Speed Switching
  t_r = 7 µs, t_f = 7 µs Typical
- Typical Applications Include Remote Terminal Isolation, SCR and Triac Triggers, Mechanical Relays and Pulse Transformers
- Safety Regulatory Approval
  UL/CUL, File No. E65085

DCJ† OR 6-TERMINAL DUAL-IN-LINE PACKAGE
(TOP VIEW)



†4N35 only
NC – No internal connection

schematic



LM35:

Temperature Sensor Datasheet [Complete LM35 Datasheet]

TO-92
Plastic Package

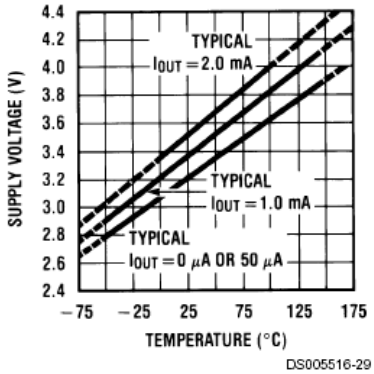
BOTTOM VIEW
DS005516-2

Order Number LM35CZ,
LM35CAZ or LM35DZ
See NS Package Number Z03A

Features

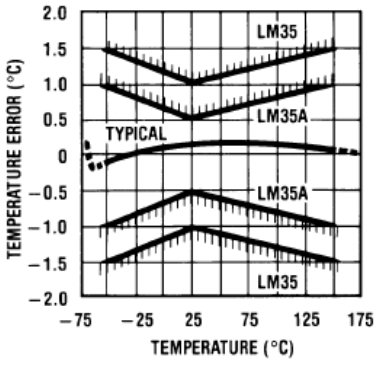
- Calibrated directly in ° Celsius (Centigrade)
- Linear + 10.0 mV/°C scale factor
- 0.5°C accuracy guaranteeable (at +25°C)
- Rated for full −55° to +150°C range
- Suitable for remote applications
- Low cost due to wafer-level trimming
- Operates from 4 to 30 volts
- Less than 60 µA current drain
- Low self-heating, 0.08°C in still air
- Nonlinearity only ±¼°C typical
- Low impedance output, 0.1 Ω for 1 mA load

Minimum Supply Voltage vs. Temperature



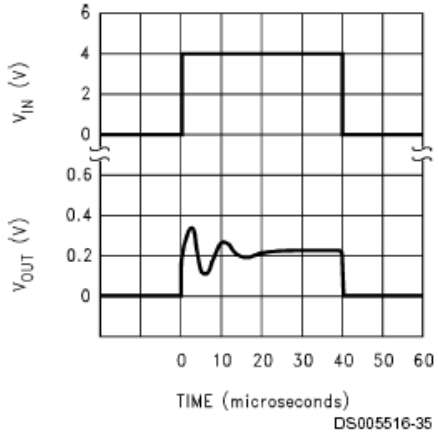
DS005516-29

Accuracy vs. Temperature (Guaranteed)



DS005516-32

Start-Up Response



ZTX1049A:

NPN SILICON PLANAR MEDIUM POWER HIGH GAIN TRANSISTOR

ZTX1049A

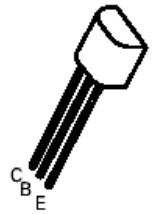
ISSUE 1 – JUNE 1995

FEATURES

* V_{CEV} = 80V
* Very low saturation voltages
* High Gain
* 20 Amps pulse current

APPLICATIONS

* LCD Backlight converters
* Emergency lighting
* DC-DC converters

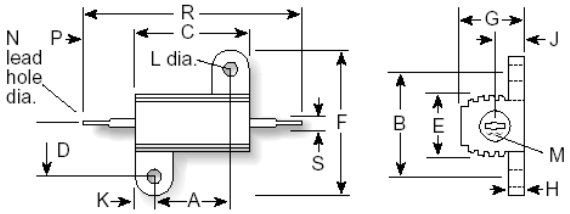


E-Line
TO92 Compatible

OHMITE-5W:

89 Series

Metal-Mite® MIL-R-18546 Approved
Aluminum Housed Axial Lead
Wirewound Resistors 1% Tolerance



The 89 Series are high-performance axial-lead type resistors. These molded-construction metal-housed resistors are available in higher power ratings than standard axial-lead resistors and are better suited to withstanding vibration, shock and harsh environmental conditions.

The 89 Series Metal-Mite® resistors are aluminum housed to maintain high stability during operation and to permit secure mounting to chassis surfaces.

The metal housing also provides heat-sinking capabilities, allowing the units to exceed the power ratings set by MIL specifications. Use the 89 Series resistors with the confidence that they meet or exceed MIL-R-18546 specifications.

Power rating: Rating is based on chassis mounting area and temperature stability. Proper heat sink as follows: 5W and 10W units, 4" x 6" x 2" x .040" Aluminum chassis; 25W units, 5" x 7" x 2" x .040" Aluminum chassis; 50W units, 12" x 12" x .059" Aluminum panel.

Maximum ohmic values: See chart.

Overload: 5 times rated wattage for 5 seconds.

Temperature coefficient:
Under 1 ohm: ±90 ppm/°C
1 to 9.99 ohms: ±50 ppm/°C
10 ohms and over: ±20 ppm/°C.

Dielectric withstanding voltage:
5W and 10W rating,1000 VAC;
25 and 50W ratings, 2250 VAC.

Series	Wattage	Ohms	Dimensions (in. / *mm*) Length	Height	Width	Voltage
805 (RE60G)	5	0.10-25K	1.125 / *28.58*	0.320 / *8.13*	0.646 / *16.41*	210
810 (RE65G)	10	0.10-50K	1.375 / *34.93*	0.390 / *9.91*	0.800 / *20.32*	320
825 (RE70G)	25	0.005-75K	1.938 / *49.23*	0.546 / *13.87*	1.080 / *27.43*	520
850 (RE75G)	50	0.005-100K	2.781 / *70.64*	0.610 / *15.49*	1.140 / *28.96*	1170

Non-Inductive versions available. Insert "N" before tolerance code. ***Example*** - 850NF560

NI-PCI-6023E:

Dynamic Characteristics

Settling time for full-scale step.....10 µs to ±0.5 LSB accuracy

Slew rate .....10 V/µs

Noise .....200 μV_{rms}, DC to 1 MHz

Midscale transition glitch

- Magnitude.....±45 mV
- Duration ..... 2.0 µs

Stability

Offset temperature coefficient .....±50 µV/°C

Gain temperature coefficient .....±25 ppm/°C

Calibration

Recommended warm-up time ..... 15 min

Interval ..... 1 year

External calibration reference ..... > 6 and < 10 V

Onboard calibration reference

- Level ..... 5.000 V (±3.5 mV) (actual value stored in EEPROM)
- Temperature coefficient..... ±5 ppm/°C max
- Long-term stability ..... ±15 ppm/$\sqrt{1,000\ h}$

Power Requirement

+5 VDC (±5%)..... 0.7 A

Note Excludes power consumed through V_{cc} available at the I/O connector.

Power available at I/O connector..... +4.65 to +5.25 VDC at 1 A

Physical

Dimensions (not including connectors)

- PCI devices ..... 17.5 by 10.6 cm (6.9 by 4.2 in.)
- PXI devices..... 16.0 by 10.0 cm (6.3 by 3.9 in.)

I/O connector

6023E/6024E.....68-pin male SCSI-II type

6025E.....100-pin female 0.05D type

Operating Environment

Ambient temperature .....0 to 55 °C

Relative humidity .....10 to 90% noncondensing

- PXI-6025E only

Functional shock.....MIL-T-28800 E Class 3 (per Section 4.5.5.4.1) Half-sine shock pulse, 11 ms duration, 30 g peak, 30 shocks per face

Operational random vibration.....5 to 500 Hz, 0.31 g_{rms}, 3 axes

Storage Environment

Ambient temperature .....–20 to 70 °C

Relative humidity .....5% to 95% noncondensing

- PXI-6025E only

Non-operational random vibration .....5 to 500 Hz, 2.5 g_{rms}, 3 axes