Modeling and Nonlinear Control of a Hot-Water-to-Air Heat Exchanger

by

David Underwood

The control of Heating, Ventilation, and Air Conditioning (HVAC) systems is difficult due to the nonlinear (open- or closed-loop) nature of the components, the wide range of operating conditions under which they must operate, and the many interactions between them. Accurate models of these component loops can be a great help in evaluating the performance of controllers and control laws, strategies, and tuning techniques. While many models already exist, their accuracy under closed-loop control is very often more limited than under open-loop control, due to the change in variables during a closed-loop test.

The heat exchanger is such a nonlinear HVAC component. The practice of using linear control on heat exchangers results in sluggish control. This work developed a model for a hot-water-to-air heat exchanger, accurate over a wide range of conditions for both open- and closed-loop simulations. The model was used to develop two nonlinear control laws, which showed better results than fixed linear controllers.

Approved for public release; distribution is unlimited.
The contents of this report are not to be used for advertising, publication, or promotional purposes. Citation of trade names does not constitute an official endorsement or approval of the use of such commercial products. The findings of this report are not to be construed as an official Department of the Army position, unless so designated by other authorized documents.

DESTROY THIS REPORT WHEN IT IS NO LONGER NEEDED

DO NOT RETURN IT TO THE ORIGINATOR
The control of Heating, Ventilation, and Air conditioning (HVAC) systems is difficult due to the nonlinear (open- or closed-loop) nature of the components, the wide range of operating conditions under which they must operate, and the many interactions between them. Accurate models of these component loops can be a great help in evaluating the performance of controllers and control laws, strategies, and tuning techniques. While many models already exist, their accuracy under closed-loop control is very often more limited than under open loop control, due to the change in variables during a closed-loop test.

The heat exchanger is such a nonlinear HVAC component. The practice of using linear control on heat exchangers results in sluggish control. This work developed a model for a hot-water-to-air heat exchanger, accurate over a wide range of conditions for both open- and closed-loop simulations. The model was used to develop two nonlinear controls laws, which showed better results than fixed linear controllers.
FOREWORD

This research was funded under Project 4A161102AT23, "Basic Research in Military Construction"; Work Unit EB-EB9, "Control For HVAC Applications."

This manuscript was submitted in partial fulfillment of the requirements for the degree of Master of Science in Mechanical Engineering in the Graduate College of the University of Illinois at Urbana-Champaign. The University of Illinois advisor for this thesis was Dr. Roy Crawford. Technical guidance and support were provided by Dale L. Herron, Energy and Utility Systems Division (ES), USACERL. Dr. Gilbert R. Williamson is Chief, USACERL–ES. The USACERL technical editor was Mr. William J. Wolfe, Information Management Office.

COL Everett R. Thomas is Commander and Director of USACERL, and Dr. L.R. Shaffer is Technical Director.
# CONTENTS

FOREWORD .......................... 2
LIST OF FIGURES AND TABLES ....... 5

1 INTRODUCTION ......................... 11
  1.1 Background ......................... 11
  1.2 Previous Coil Models............... 11
  1.3 New Work .......................... 12

2 EXPERIMENTAL HARDWARE .......... 13
  2.1 Description of Experimental Setup ... 13
  2.2 Airflow Rate Measurement ......... 13
  2.3 Water Flow Rate Measurement ...... 21
  2.4 Air and Water Temperature Measurement ...... 21
  2.5 Water Flow Rate Control .......... 21

3 MODELS ............................. 24
  3.1 Nomenclature ........................ 24
  3.2 Coil Model .......................... 25
      3.2.1 Derivation .................... 25
      3.2.2 Open- and Closed-Loop Tests ... 27
      3.2.3 Coefficient Revisions .......... 29
      3.2.4 Revision of Model Form .......... 31
  3.3 Valve Model ......................... 35
      3.3.1 Valve Hysteresis Check ......... 35
      3.3.2 Polynomial Fit .................. 38
      3.3.3 Step Response ................... 40
      3.3.4 Control Signal and Water Flow Rate Correlation ................. 40
  3.4 Complete Loop Model: Controller, Valve, Coil; Closed-Loop Simulation ...... 48

4 A LINEARIZED SYSTEM .............. 48
  4.1 Linearization of the Coil Model ... 48
  4.2 Linearized Valve ................... 50
  4.3 Root Locus .......................... 50
  4.4 Design Criteria .................... 51

5 PROPORTIONAL-ONLY NONLINEAR CONTROL .......................... 57
  5.1 Nomenclature ........................ 57
  5.2 Calculation of $K_p$ vs Water Flow Rate for 25 Percent Overshoot .............. 57
  5.3 Proportional Gain as a Function of Water Flow Rate .......................... 62
  5.4 Proportional Gain as a Function of Setpoint and Inlet Air Temperature ...... 71
  5.5 Implementation of the Setpoint-Dependent Proportional Gain Nonlinear Controller on the Test Facility .............. 82
CONTENTS (Cont'd)

6 SUMMARY AND CONCLUSIONS ........................................... 95
APPENDIX A: Software Description ...................................... 96
APPENDIX B: Program Listings ........................................... 99

Accession For
NTIS GRA&I
DTIC TAB
Unannounced
Justification

By
Distribution/
Availability Codes

Dist
Avail and/or
Special

A-1
<table>
<thead>
<tr>
<th>FIGURES</th>
<th>Number</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Test Facility</td>
<td>14</td>
</tr>
<tr>
<td>2.2</td>
<td>Venturi Circuit Diagram</td>
<td>16</td>
</tr>
<tr>
<td>2.3</td>
<td>Venturi and Hot Wire Anemometer Associated Ductwork</td>
<td>17</td>
</tr>
<tr>
<td>2.4</td>
<td>Hot Wire Anemometer and Venturi Measurements vs. Time</td>
<td>19</td>
</tr>
<tr>
<td>2.5</td>
<td>Hot Wire Anemometer Measurement With Anemometer Upstream and Downstream of Coil vs. Time</td>
<td>20</td>
</tr>
<tr>
<td>2.6</td>
<td>Thermocouple Array Arrangement Diagram</td>
<td>22</td>
</tr>
<tr>
<td>2.7</td>
<td>Water Flow Rate Control and Measurement Diagram</td>
<td>23</td>
</tr>
<tr>
<td>3.1</td>
<td>Capacitance Representations of Two Different Coil Models</td>
<td>26</td>
</tr>
<tr>
<td>3.2</td>
<td>Prediction of Discharge Air Temperature for the Original Model During an Open-Loop Test</td>
<td>28</td>
</tr>
<tr>
<td>3.3</td>
<td>Discharge Air Temperature Measured and Predicted During a Closed-Loop Test</td>
<td>30</td>
</tr>
<tr>
<td>3.4</td>
<td>Full Mixing of Air Passing Through the Coil</td>
<td>32</td>
</tr>
<tr>
<td>3.5</td>
<td>Partial Mixing of Air Passing Through the Coil</td>
<td>33</td>
</tr>
<tr>
<td>3.6</td>
<td>Revised and Original Model Discharge Air Temperature Prediction Comparison</td>
<td>36</td>
</tr>
<tr>
<td>3.7</td>
<td>Water Flowmeter Signal vs. Control Signal</td>
<td>37</td>
</tr>
<tr>
<td>3.8</td>
<td>Polynomial Fit of Water Flowmeter Signal and Control Signal</td>
<td>39</td>
</tr>
<tr>
<td>3.9</td>
<td>Water Flow Rate vs. Control Signal Step</td>
<td>41</td>
</tr>
<tr>
<td>3.10</td>
<td>Measured and Predicted Water Flow Rate During Closed-Loop Control</td>
<td>42</td>
</tr>
<tr>
<td>3.11</td>
<td>Block Diagram of the Digitally Controlled Hot Water Coil</td>
<td>43</td>
</tr>
</tbody>
</table>
### FIGURES (Cont’d)

<table>
<thead>
<tr>
<th>Number</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.12</td>
<td>Simulation of the Proportional-Only Controlled Hot Water Coil; $K_p = 100 \degree C/\degree C$; $T_{sp} = 50 \degree C$</td>
</tr>
<tr>
<td>3.13</td>
<td>Simulation of the Proportional-Only Controlled Hot Water Coil; $K_p = 205 \degree C/\degree C$; $T_{sp} = 50 \degree C$</td>
</tr>
<tr>
<td>3.14</td>
<td>Simulation of the Proportional-Only Controlled Hot Water Coil; $K_p = 410 \degree C/\degree C$; $T_{sp} = 50 \degree C$</td>
</tr>
<tr>
<td>4.1</td>
<td>Root Locus of Linearized Coil and Valve Models at Water Flow Rate of 0.063 L/s</td>
</tr>
<tr>
<td>4.2</td>
<td>Root Locus of Linearized Coil and Valve Models at Water Flow Rate of 0.126 L/s</td>
</tr>
<tr>
<td>4.3</td>
<td>Root Locus of Linearized Coil and Valve Models at Water Flow Rate of 0.189 L/s</td>
</tr>
<tr>
<td>4.4</td>
<td>Root Locus of Linearized Coil and Valve Models at Water Flow Rate of 0.252 L/s</td>
</tr>
<tr>
<td>4.5</td>
<td>Root Locus of Linearized Coil and Valve Models at Water Flow Rate of 0.316 L/s</td>
</tr>
<tr>
<td>5.1</td>
<td>Simulated Discharge Air Temperature Using the Nonlinear Coil Model Producing a 25 Percent Overshoot at a Nominal Water Flow Rate of 0.063 L/s</td>
</tr>
<tr>
<td>5.2</td>
<td>Proportional Gain vs. Water Flow Rate for 25 Percent Overshoot</td>
</tr>
<tr>
<td>5.3</td>
<td>Steady State Coil Gain vs. Water Flow Rate for the Four Cases of Table 5.1</td>
</tr>
<tr>
<td>5.4</td>
<td>Curve Fit of the Base Case $K_p K_v$ vs. Water Flow Rate for 25 Percent Overshoot</td>
</tr>
<tr>
<td>5.5</td>
<td>Simulated Discharge Air Temperature Using Linear and Nonlinear Control; $SP_b = 45 \degree C$; $\delta SP = 5 \degree C$; Base Case of Table 5.1</td>
</tr>
<tr>
<td>5.6</td>
<td>Water Flow Rate Resulting From the Simulation of Linear and Nonlinear Control; With $K_{pl} = 104 \degree C/\degree C$ and $K_{pl}$ From Equation 5.4.</td>
</tr>
<tr>
<td>Number</td>
<td>Figures Description</td>
</tr>
<tr>
<td>--------</td>
<td>-------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>5.7</td>
<td>Simulated Discharge Air Temperature Using Linear and Nonlinear Control; SP(_b) = 45 °C; (\delta SP) = 5 °C; Base Case of Table 5.1; Controller Bias = 1591</td>
</tr>
<tr>
<td>5.8</td>
<td>Simulated Discharge Air Temperature Using Linear and Nonlinear Control; SP(_b) = 45 °C; (\delta SP) = 5 °C; Base Case of Table 5.1</td>
</tr>
<tr>
<td>5.9</td>
<td>Simulated Discharge Air Temperature Using Linear and Nonlinear Control; SP(_b) = 45 °C; (\delta SP) = 5 °C; Base Case of Table 5.1; (K_p) = 105 (CO/\circ C)</td>
</tr>
<tr>
<td>5.10</td>
<td>Simulated Discharge Air Temperature Using Linear and Nonlinear Control; SP(_b) = 45 °C; (\delta SP) = 5 °C; Base Case of Table 5.1; (K_p) = 105 (CO/\circ C)</td>
</tr>
<tr>
<td>5.11</td>
<td>Simulated Discharge Air Temperature Using Linear and Nonlinear Control; SP(_b) = 35.71 °C; (\delta SP) = 1.09 °C; Base Case of Table 5.1; (K_p) = 349 (CO/\circ C)</td>
</tr>
<tr>
<td>5.12</td>
<td>Simulated Discharge Air Temperature Using Linear Control for a Setpoint Upset From 32 °C to 52 °C; Base Case of Table 5.1; (K_p) = 690 (CO/\circ C)</td>
</tr>
<tr>
<td>5.13</td>
<td>Simulated Discharge Air Temperature Using Linear Control for a Setpoint Upset From 52 °C to 32 °C; Base Case of Table 5.1; (K_p) = 310 (CO/\circ C)</td>
</tr>
<tr>
<td>5.14</td>
<td>Multiplicative Curve Fit for Two Proportional Gains and Setpoint Minus Inlet Air Temperature</td>
</tr>
<tr>
<td>5.15</td>
<td>Simulated Discharge Air Temperature Using Linear and Nonlinear Control; SP(_b) = 32 °C; (\delta SP) = 20 °C; Base Case of Table 5.1</td>
</tr>
<tr>
<td>5.16</td>
<td>Simulated Discharge Air Temperature Using Linear and Nonlinear Control; SP(_b) = 32 °C; (\delta SP) = 20 °C; Base Case of Table 5.1</td>
</tr>
<tr>
<td>5.17</td>
<td>Simulated Discharge Air Temperature Using Linear and Nonlinear Control; SP(_b) = 50 °C; (\delta SP) = 5 °C; Base Case of Table 5.1</td>
</tr>
<tr>
<td>5.18</td>
<td>Simulated Discharge Air Temperature Using Linear and Nonlinear Control; SP(_b) = 40 °C; (\delta SP) = 5 °C; Base Case of Table 5.1</td>
</tr>
</tbody>
</table>
### FIGURES (Cont'd)

<table>
<thead>
<tr>
<th>Number</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.19</td>
<td>Measured Discharge Air Temperature for a Control Signal Step From 1700 to 400</td>
<td>85</td>
</tr>
<tr>
<td>5.20</td>
<td>Measured Discharge Air Temperature for Linear Control on Test Facility; $SP_b = 35 , ^\circ C$; $\delta SP = 15 , ^\circ C$; $K_{p1} = 300 , CO/\circ C$</td>
<td>86</td>
</tr>
<tr>
<td>5.21</td>
<td>Measured Discharge Air Temperature for Linear Control on Test Facility; $SP_b = 40 , ^\circ C$; $\delta SP = 10 , ^\circ C$; $K_{p1} = 600 , CO/\circ C$</td>
<td>87</td>
</tr>
<tr>
<td>5.22</td>
<td>Measured Discharge Air Temperature for Linear Control on Test Facility; $SP_b = 40 , ^\circ C$; $\delta SP = 10 , ^\circ C$; $K_{p1} = 500 , CO/\circ C$</td>
<td>88</td>
</tr>
<tr>
<td>5.23</td>
<td>Measured Discharge Air Temperature for Linear Control on Test Facility; $SP_b = 45 , ^\circ C$; $\delta SP = 5 , ^\circ C$; $K_{p1} = 350 , CO/\circ C$</td>
<td>89</td>
</tr>
<tr>
<td>5.24</td>
<td>Measured Discharge Air Temperature for Nonlinear Control on Test Facility; $SP_b = 45 , ^\circ C$; $\delta SP = 5 , ^\circ C$; $K_{pNL} = \text{Eq. 5.5}$</td>
<td>91</td>
</tr>
<tr>
<td>5.25</td>
<td>Measured Discharge Air Temperature for Nonlinear Control on Test Facility; $SP_b = 35 , ^\circ C$; $\delta SP = 15 , ^\circ C$; $K_{pNL} = \text{Eq. 5.5}$</td>
<td>92</td>
</tr>
<tr>
<td>5.26</td>
<td>Measured Discharge Air Temperature for Linear and Nonlinear Control on Test Facility; $SP_b = 35 , ^\circ C$; $\delta SP = 5 , ^\circ C$; $K_{p1} = 310 , CO/\circ C$</td>
<td>93</td>
</tr>
<tr>
<td>A1</td>
<td>Flow Diagram of Matlab Program DSTEST.M.</td>
<td>98</td>
</tr>
</tbody>
</table>
### TABLES

<table>
<thead>
<tr>
<th>Number</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Test Facility Equipment List</td>
<td>15</td>
</tr>
<tr>
<td>5.1</td>
<td>Four Steady-State Conditions Used for Calculation of Nonlinear Control Law</td>
<td>58</td>
</tr>
<tr>
<td>5.2</td>
<td>Proportional Gain for 25 %OS for the Base Case</td>
<td>59</td>
</tr>
</tbody>
</table>
MODELING AND NONLINEAR CONTROL OF A
HOT-WATER-TO-AIR HEAT EXCHANGER

1. INTRODUCTION

1.1 Background

The control of Heating, Ventilation, and Air Conditioning (HVAC) systems is difficult due to the nonlinear nature of the components, the wide range of operating conditions under which they must operate, and the many interactions between them. Accurate models of these loops can be a great tool in evaluating the performance of controllers and in particular control laws, strategies, and tuning techniques. While many models already exist, their accuracy under closed-loop control is very often limited as compared to open-loop accuracy due to the change in variables during a closed-loop test.

One component in particular, the heat exchanger, is nonlinear in nature. The general practice of using linear control on this nonlinear component results in sluggish control. The goal of the work presented here was to accurately model a hot-water-to-air heat exchanger loop and to use this model to develop a nonlinear control law with a minimum number of tuning parameters.

1.2 Previous Coil Models

Previous work\(^1\) has been done at the University of Illinois on modeling hot-water-to-air heat exchangers. Nesler modeled the dynamic response of the heating coil as a nonlinear (coefficients as a function of operating point) first-order differential equation plus dead-time lag equation. The steady-state response was derived from experimental tests performed about a fixed operating point (inlet water temperature, inlet air temperature, and airflow rate fixed). This resulted in an expression for the dynamics of the discharge air temperature about a single operating point which was dependent solely upon the water flow rate.

While Nesler’s model provided insights to stability limits about a single operating point, it lacked accuracy and the ability to explore dynamics at other operating points. Rohrer refined Nesler’s model to achieve more accuracy by defining the effect of the operating point and water flow rate on the time constant and

dead-time of the coil. He also investigated an additional operating point having an airflow rate 60 percent of that of the original.

Ghassan developed an analytical model for a one-row, two-pass cross flow hot water coil and compared it with experimental data. The comparison consisted of open-loop upsets of water flow rate, inlet water temperature, and air flow rate step changes. This work provided additional information on the open-loop dynamic response over a wider range of operating conditions, but did not explore closed-loop response.

1.3 New Work

While the work by Nesler, Rohrer, and Ghassan provided insights to the characteristics of hot water to air heat exchangers, the model’s accuracy under closed-loop control was not thoroughly investigated. Nesler and Rohrer investigated some aspects of closed-loop stability about two operating points, but their work cannot be easily extended to other conditions. While Ghassan’s model is more general and applicable to a wider range of operating conditions, his model’s accuracy under closed-loop control was never verified. Although some work was done by Rohrer regarding sampling time and PI control, very little has been done concerning the tuning process. This work addresses both closed-loop accuracy and the tuning process. First a model accurate for both open and closed-loop conditions was developed and then a nonlinear control law with a single tuning parameter was developed. This involved the following steps:

1. Study the noise characteristics of the sensors used, and where appropriate, filter the analog signal or install a new sensor.

2. Develop software for data collection, analysis, and simulations.

3. Produce an accurate model of the heat exchanger for both open and closed-loop simulations for inlet air temperature changes, air flow rate changes, inlet water temperature changes, and water flow rate changes.

4. Develop a nonlinear control law with one tuning parameter.
2. **EXPERIMENTAL HARDWARE AND SOFTWARE**

2.1 Description of Experimental Setup

Figure 2.1 shows the experimental setup used for data acquisition and control of the hot water temperature to air heat exchanger. Water flow rate was controlled by computer through a Metrabyte DAS-16 D/A board, EXP-16 multiplexer and filtering board, a Honeywell RP7517B1016-1 E/P transducer, and a Honeywell model V5011A two way valve with a pilot positioner. The heat exchanger is a McQuay, four-row hot-water-to-air heat exchanger. The hot water, supplied by a steam-to-hot-water heat exchanger, was controlled independently of the rest of the test facility. The combination of electric heaters in the inlet duct, a valve to mix unheated city water with heated water, inlet air dampers, and the control valve, all interfaced to the PIO-12 relay board, allowed computer control of the four variables affecting the process variable (outlet air temperature). Table 2.1 lists the major components of the test facility along with a short description.

2.2 Airflow Rate Measurement

The airflow rate was initially measured with a Sierra model 432, hot-wire anemometer (HWA) placed at a point representative of the average velocity. Initial measurements with the HWA were noisy and varied with air temperature, so the circuit of Figure 2.2 was constructed to accommodate airflow rate measurement with a venturi through the computer. A detail of the HWA, venturi, and the associated ductwork is seen in Figure 2.3. The empirical correlation between the HWA’s output voltage and the air velocity is given as Equation 2.1.

\[
V_{hwa} = -0.1 + 1.32 E_h - 0.8534 E_h^2 + 1.27049 E_h^3 \quad (2.1)
\]

The analytical correlation for the venturi, obtained from Bernoulli’s Equation, after scaling to account for different duct cross-sectional areas is given as Equation 2.2, where the air was assumed to be an incompressible fluid and its density a function of temperature.

13
Table 2.1
Test Facility Equipment List

<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pump</td>
<td>Bell and Gosset (1/3 hp), 1725 rpm</td>
</tr>
<tr>
<td>Fan</td>
<td>10&quot; outside radius centrifugal</td>
</tr>
<tr>
<td>Duct</td>
<td>21&quot; x 15&quot; rectangular</td>
</tr>
<tr>
<td>hot Water Coil</td>
<td>McQuay Model G, 21&quot; x 15&quot;, 4 row</td>
</tr>
<tr>
<td>Control Valve</td>
<td>Honeywell, Model V5011A11061 8432</td>
</tr>
<tr>
<td>Electric Heater</td>
<td>5 and 10 Kw</td>
</tr>
<tr>
<td>A/D board</td>
<td>Metrabyte DAS-16, 12 bit, -5v to +5v input</td>
</tr>
<tr>
<td>Computer</td>
<td>IBM PS/2 Model 30</td>
</tr>
<tr>
<td>Venturi Pressure Transducer</td>
<td>Setra Model 432, 0 to 1&quot;wc</td>
</tr>
<tr>
<td>E/P Transducer</td>
<td>Honeywell, Model RP7517B1016-1</td>
</tr>
<tr>
<td>Hot Wire Anemometer</td>
<td>Sierra, Model 432</td>
</tr>
<tr>
<td>Thermocouples</td>
<td>Type T, copper-constantan</td>
</tr>
<tr>
<td>Venturi</td>
<td>10&quot; to 8&quot; diameter</td>
</tr>
<tr>
<td>Water Flow Meter</td>
<td>Flow-Tech turbine Model FM-AC</td>
</tr>
<tr>
<td>Hot Water Valve</td>
<td>MP 953E 1319 8536, normally open</td>
</tr>
</tbody>
</table>

\[ V_{vnt} = 0.072 \left[ \frac{T_{ao}E_v}{E_h} \right]^{0.5} \]  \hspace{1cm} (2.2)

where

- \( V_{hwa} \) = Air velocity (meters per second) at section 1 of Figure 2.3 measured by hot wire anemometer
- \( V_{vnt} \) = Air velocity (meters per second) at section 1 of Figure 2.3 measured by venturi
- \( E_h \) = Voltage (volts) measured from HWA
- \( E_v \) = Voltage (volts) measured from venturi pressure transducer
- \( Tao \) = Air temperature (R)

The HWA correlation was derived from the principle that the convective heat transfer rate of a heated element exposed to a moving airflow is dependent upon, among other things, that airflow’s velocity. A fixed amount of electrical energy is supplied to an element exposed to the fluid whose velocity is to be measured. The temperatures of this element and the temperature of the unheated fluid are measured. For a correlation between the temperature differential and fluid velocity to be accurate, the other effects such as density, humidity, and fluid temperature on
Figure 2.2. Venturi circuit diagram.
Figure 2.3. Venturi and hot wire anemometer associated ductwork.
the convective heat transfer rate must be taken into account, making the instrument's accuracy susceptible to changes in any of these parameters. In order to assess the HWA's ability to compensate for changes in some of these factors, airflow rate measurements were recorded at 5-second intervals for both the HWA and a venturi while air velocity, inlet air temperature, and inlet water temperature were varied. The airflow rate was first stepped up and then down at 2 and 10 min respectively as labeled in Figure 2.4 by "Dampers Close" and "Dampers Open." Next the inlet air temperature was first stepped up and then down by turning electric heaters in the air duct on and off at 18 and 27 min respectively as labeled in Figure 2.4 by "15kW On" and "15kW Off." The inlet water temperature was stepped down and then stepped up at 35 and 43 minutes respectively, as labeled by "CW Open" and "CW Closed," referring to the opening and closing of the city water valve allowing unheated water into the water supplied to the hot water coil as shown in Figure 2.1. Although it is reasonable to expect the capacity of the fan to be partially dependent on air temperature, the changes measured by the HWA during the inlet air temperature upset at 18 and 27 min seemed excessive. The venturi reading remains relatively stable as compared to the HWA reading. Both instruments contain noise, but the venturi's noise level is much smaller than that of the HWA. While the venturi measurement does reflect a change in airflow when the inlet air temperature changes at 18 and 27 min, the reading reaches a new value and remains near that value. The HWA however, overshoots the new value, eventually manages to compensate, and also reaches a new value.

This variation in the airflow velocity measurement could be from either turbulence in the airstream or electronic noise. Either source of variation was undesirable. Although both the HWA and venturi signals were filtered with a passive, low-pass RC filter with a time constant of 0.8 s, a significant noise level persisted for the HWA. At first, it was thought that moving the sensor upstream of the coil would reduce the variation of airflow measurement for changes in inlet water temperature and water flow rate changes, so measurements were taken with the HWA upstream of the coil, again at 5 s intervals. Figure 2.5 shows the HWA measurement both upstream and downstream of the coil. Not only is the magnitude changed significantly, but the signal to noise ratio is smaller downstream of the coil. There are at least two possible explanations. First, the coil may decrease the turbulence. Second, it may act as a sort of buffer of temperature change in air temperature. That is, the temperature of the air changes more slowly downstream of the coil than it does upstream.

The most critical aspect of the air velocity measurement was the dynamic changes. Since the noise of the HWA measurement was much larger than that of the venturi, these tests prompted the use of the venturi for all subsequent tests.
Figure 2.4. Hot wire anemometer and venturi measurements vs. time.
Figure 2.5. Hot wire anemometer measurement with anemometer upstream and downstream of coal vs. time.
2.3 Water Flow Rate Measurement

The water flow rate was measured with a Flow-Tech Model FM-AC turbine flowmeter with a rating of 0.54 L/s to 1.13 L/s (1.5 gpm to 18 gpm). This signal, like the airflow rate measurement, had a significant amount of high frequency noise and was filtered with a passive, low-pass, RC filter with a time constant of 0.8 s.

2.4 Air and Water Temperature Measurement

The air temperature measurements were performed with type-T thermocouple arrays wired in parallel, arranged in the duct as shown in Figure 2.6. This signal, which was first processed by the EXP-16 board, did not have a significant amount of noise.

2.5 Water Flow Rate Control

The water flow rate was controlled by computer through a 12 bit D/A converter (-5V to +5V analog range) and voltage to pressure (E/P) transducer as shown in Figure 2.7.
Figure 2.6. Thermocouple array arrangement diagram.
Figure 2.7. Water flow rate control and measurement diagram.
3. MODELS

3.1 Nomenclature

The following nomenclature was used in the development of the models:

- \( C_{pa} \) = Constant pressure specific heat of air
- \( C_{pw} \) = Constant pressure specific heat of water
- \( C_{pc} \) = Constant pressure specific heat of coil
- \( m_{cw} \) = Mass of coil assumed at an average temperature equal to outlet water temperature
- \( m_{ca} \) = Mass of coil assumed at an average temperature equal to outlet air temperature
- \( T_{ai} \) = Inlet air temperature
- \( T_{ao} \) = Discharge air temperature
- \( T_{wi} \) = Inlet water temperature
- \( T_{wo} \) = Outlet water temperature
- \( \bar{T}_w \) = \([T_{wi} + T_{wo}] / 2\)
- \( m_a \) = Air mass flow rate
- \( m_w \) = Water mass flow rate
- \( C_a \) = \( C_{pc}m_{cw} \)
- \( C_o \) = \( C_{pc}m_{ca} \)
- \( UA \) = Overall heat transfer coefficient
- \( CO \) = Control signal in terms of 12 bits (range 0 - 4095)
- \( T_{sp} \) = Setpoint temperature of controller
- \( K_p \) = Proportional gain of controller
3.2 Coil Model

3.2.1 Derivation

In this work, a coil model similar to the one Ghassan developed was proposed. The main difference between Ghassan’s model and this model was the representation of the coil capacitance. As seen in Figure 3.1, while Ghassan assumed that the derivative of the average coil temperature was equal to that of the outlet water temperature, here the capacitance was broken into two terms as stated in assumptions 5 and 6 listed below. The remaining assumptions were identical to Ghassan’s.

The assumptions made for this model were:

1. The tube, water, and air have constant specific heats throughout the tube length and over the ranges of temperature encountered.
2. Densities are constant.
3. There is negligible heat conduction in the axial direction.
4. The effective air temperature to be considered for convective heat transfer purposes is the inlet air temperature at any cross section.
5. The derivative of the temperature change of \( m_{ca} \) is equal to that of the outlet air temperature.
6. The derivative of the temperature change of \( m_{cw} \) is equal to that of the outlet water temperature.
7. The mean temperature difference between the two fluids driving the overall heat transfer \( UA \) is the difference of the inlet air temperature and the average water temperature.

Energy balance equations were formed using the above assumptions. The water side energy balance results in Equation 3.1 and the air side energy balance gives Equation 3.2.

\[
\begin{align*}
    m_w(t)C_{pw}[T_{w1}(t) - T_{w0}(t)] + UA(t)[T_{a1}(t) - T_v(t)] &= C_w \frac{dT_{w0}(t)}{dt} \\
    m_a(t)C_{pa}[T_{a1}(t) - T_{a0}(t)] + UA(t)[T_v(t) - T_{a1}(t)] &= C_a \frac{dT_{a0}(t)}{dt}
\end{align*}
\]
GHASSAN'S MODEL:
\[
\frac{d}{dt} [T_{\text{coil}}] = \frac{d}{dt} [T_{\text{wo}}]
\]

New model:
\[
\frac{d}{dt} [T_{m_{\text{ca}}} ] = \frac{d}{dt} [T_{\text{ao}}]
\]
\[
\frac{d}{dt} [T_{m_{\text{cw}}} ] = \frac{d}{dt} [T_{\text{wo}}]
\]

Figure 3.1. Capacitance representations of two different coil models.
Approximating the overall heat transfer as in Equation 3.3 and using the forward rectangular rule for discrete approximation to obtain a finite difference equation from Equations 3.1, 3.2, and 3.3, results in Equations 3.4 and 3.5.

\[ UA(t) = a + b\dot{m}_w(t) + d\dot{m}_a(t) \]  

(3.3)

\[ T_{wo}(k) = T_{wo}(k-1) + A\dot{m}_w(k-1)[T_{w1}(k-1) - T_{wo}(k-1)] \]
\[ + [B + Cm_w(k-1) + Dm_a(k-1)] [T_{a1}(k-1) - \bar{T}_w(k-1)] \]  

(3.4)

\[ T_{ao}(k) = T_{ao}(k-1) \]
\[ + E\dot{m}_a(k-1)[T_{a1}(k-1) - T_{ao}(k-1)] \]
\[ + [F + Gm_w(k-1) + H\dot{m}_a(k-1)] [\bar{T}_w(k-1) - T_{a1}(k-1)] \]  

(3.5)

where

\[ A = C_{pw}\Delta t/C_w \]
\[ B = a\Delta t/C_w \]
\[ C = b\Delta t/C_w \]
\[ D = d\Delta t/C_w \]
\[ E = C_{ps}\Delta t/C_a \]
\[ F = a\Delta t/C_a \]
\[ G = b\Delta t/C_a \]
\[ H = d\Delta t/C_a \]
\[ \Delta t = \text{sampling interval} \]

3.2.2 Open- and Closed-Loop Tests

The first test for model adequacy was an open loop-test. Here, an open-loop test was defined as a test in which the input variables \( T_{w1}, T_{a1}, m_a \) were ramped from an initial value to a second value as quickly as the hardware allowed and water flow rate remained constant. The ramping of the airflow rate and inlet air temperature were quick as compared to the 5-second sampling interval, while the ramp of the inlet water temperature was somewhat slower. The measured values of inlet water temperature, inlet air temperature, water flow rate, and airflow rate were then used in computer simulations using the difference Equations 3.4 and 3.5 to predict \( T_{wo} \) and \( T_{ao} \). These predictions were then plotted along with the measured \( T_{wo} \) and \( T_{ao} \) in Figure 3.2. Here, the airflow rate was first stepped up and then down at 2 and 10 min respectively as labeled in Figure 3.2 by "Dampers Close" and
Figure 3.2. Prediction of discharge air temperature for the original model during an open-loop test.
"Dampers Open." Next the inlet air temperature was first stepped up and then down by turning electric heaters in the air duct on and off at 17 and 27 min respectively, as labeled in Figure 3.2 by "15kW On" and "15kW Off." The inlet water temperature was stepped down and then up at 35 and 43 min respectively, as labeled by "CW Open" and "CW Closed," referring to the opening and closing of the CW valve allowing unheated water into the water supplied to the hot water coil (Figure 2.1). The coil model predicts the effects of inlet water temperature, inlet air temperature, and airflow rate on the outlet air temperature very well when only one variable changes at a time.

The next step to test the coil model was a closed-loop test similar to that of the open-loop test. However, during this test proportional-only control was used to modulate the water flow rate according to Equation 3.6.

\[
\text{CO}(k) = K_p[T_{sp}(k) - T_{ao}(k)] + \text{bias} \tag{3.6}
\]

Additionally, a setpoint upset was introduced in addition to those of the open-loop test. The setpoint was stepped from 40 °C to 45 °C (104 °F to 113 °F) at 52 min and back to 40 °C (113 °F) at 58 min as labeled by "setpoint +5" and "setpoint - 5" in Figure 3.3. The closed-loop data were predicted less accurately than the open-loop test. The prediction of Tao for the open loop response shown in Figure 3.2 had an average absolute error in predicting the outlet air temperature of 0.16 °C (0.29 °F). The closed-loop test, shown in Figure 3.3, had an average absolute error in predicting outlet air temperature of 0.82 °C (1.48 °F). These disappointing results were observed for several other open- and closed-loop test data and found to be repeatable.

3.2.3 Coefficient Revisions

Because of these deficiencies in closed-loop prediction of $T_{ao}$, several revised models were proposed and fit to data. The initial revisions involved attempts to more accurately represent the model coefficients. First the coil capacitance coefficients $(C_w$ and $C_a)$ were studied. Figure 3.1 seems to indicate that it would be reasonable to expect that $m_{cw}$, the portion of the coil at an average temperature equal to the outlet water temperature, would be larger for high water flow rates and lower for low water flow rates. In other words, the dashed line in that figure would increase in diameter as water flow rate increased. Similarly, as airflow rate increased, the diameter of the dashed line would decrease the second revision explored was the effect of the continuous to discrete approximations. Use of the backward rectangular rule approximation was compared with the forward rectangular rule. The results indicated that the original model was more accurate than the revised models.
Figure 3.3. Discharge air temperature measured and predicted during a closed-loop test.
3.2.4 Revision of Model Form

Because the revisions to the original model mentioned in section 3.2.3 failed to improve its accuracy, it was felt that some basic characteristic of the coil was missing. This necessitated a change in the approach, to improve the model. While the previous revisions of section 3.2.3 were attempts to more accurately represent the coefficients of the model, the model form remained the same. The absence of a basic characteristic may require a different model form, with extra or different terms. Several versions of a model using the effectiveness vs. Number of Transfer Units (NTU) relationship were derived, but like previous revisions, were found to be no better than the original model. The upset which the original model of section 3.3 had the greatest difficulty modeling (Figure 3.3), was an inlet air temperature change. The model reacted much too slowly to the step change in the inlet air temperature, $T_{ai}$. When $T_{ai}$ changes quickly, the measured outlet air temperature changes nearly instantly, and continues to change gradually. Although the original model prediction, described by Equations 3.4 and 3.5, changes gradually upon inlet air temperature disturbances, it does not have the immediate initial response seen in the experiments. This observation led to the addition of a term to the original model on the premise that there existed a cross-sectional area of the coil in which the passing air is not directly affected by the forced convection heat transfer. This extra term would therefore not be dependent upon the coil dynamics, but would be a feed-forward term. The original model assumed the situation depicted in Figure 3.4, which misses the true dynamics depicted in Figure 3.5. While most of the air flowing past the coil participates in the convection heat transfer (airstream 2), a small percentage of it (airstream 1) does not get heated until after it gets downstream of the coil and mixed with airstream 1. Assuming that the two thermocouple arrays have the same dead-time and time constant, the change sensed at the outlet thermocouple array due to a change in inlet is equal to the change in inlet air temperature delayed by the time required for the airstream to travel the duct length. At the lowest velocity of 2.0 m/s (6.7 ft/s) encountered on the test facility, the 0.3 m (1 ft) of duct length from the inlet to outlet thermocouple arrays is traveled in 0.15 s, small compared to the 5-second sampling interval. An energy balance of the unmixed and mixed airstreams, assuming the zero step delay, yields Equation 3.7,

$$m_{a1}C_{pa}(T_1 - T_{a1}) + m_{a2}C_{pa}(T_2 - T_{a2}) = [m_{a1} + m_{a2}]C_{pa}(T_{ao} - T_{a1})$$

(3.7)

and canceling $C_{pa}$ and rearranging results in Equation 3.8:

$$T_{ao} = \frac{m_{a1}}{m_a}T_1 + \frac{m_{a2}}{m_a}T_2$$

$$= R_1T_1 + R_2T_2$$

(3.8)
Figure 3.4. Full mixing of air passing through the coil.
Figure 3.5 Partial mixing of air passing through the foil.
Writing this in the finite difference form for $T_{ao}(k)$ and $T_{ao}(k-1)$ gives:

$$T_{ao}(k) - T_{ao}(k-1) = R_1[T_1(k) - T_1(k-1)] + R_2[T_2(k) - T_2(k-1)]$$

(3.9)

where

$m_{a1} = \text{air mass flow rate associated with airstream i of Figure 3.5 downstream of coil}$

$T_i = \text{temperature of air associated with airstream i of Figure 3.5 downstream of coil}$

$R_1 = \frac{m_{a1}}{m_a}$

$R_2 = \frac{m_{a2}}{m_a}$

$m_a = m_{a1} + m_{a2}$

Noting that $T_{ao}$ of Equation 3.5 represents $T_2$ of Equation 3.8 and $m_a$ is $m_{a2}$, Equation 3.5 can be rewritten as Equation 3.10.

$$T_2(k) = T_2(k-1) + \frac{E m_{a2}(k-1)[T_{a1}(k-1) - T_2(k-1)]}{m_a} + [F + Gm_u(k-1)]T_u(k-1) - T_{a1}(k-1)]$$

Substituting Equation 3.10 into 3.9 and rearranging results in Equation 3.11.

$$T_{ao}(k) = T_{ao}(k-1) + \frac{E' m_a(k-1)[T_{a1}(k-1) - T_{ao}(k-1)]}{m_a} + [F' + G'm_u(k-1)]T_u(k-1) - T_{a1}(k-1)] + I'[T_{a1}(k) - T_{a1}(k-1)]$$

Substituting the correct air mass flow rate into 3.4 results in Equation 3.12.

$$T_{wo}(k) = T_{wo}(k-1) + \frac{A m_u(k-1)[T_{w1}(k-1) - T_{wo}(k-1)]}{m_a} + [B + Cm_u(k-1) + D'm_a(k-1)][T_{a1}(k-1) - T_u(k-1)]$$

34
where

\[ T_{ao} = \text{air temperature sensed at the averaging outlet} \]

\[ \text{thermocouple array} \]

\[ D' = R_2 \quad G' = R_1G \]

\[ E' = R_1E \quad H' = R_1H \]

\[ F' = R_1F \quad I' = R_2 \]

and A, B, C, D, E, F, G, and H are as they were for Equations 3.4 and 3.5.

Equations 3.11 and 3.12 were used to simulate closed-loop control, which was compared with the measured response. Inlet air temperature, inlet water temperature, airflow rate, and water flow rate as measured on the facility, were used in the simulation whose results are shown in Figure 3.6. Here, the inlet air damper closed at 2 min and opened at 5 min and the 15 kW heaters in the inlet air duct were turned on at 10 min and off at 15 min. The added term significantly improved the accuracy of the model for inlet air temperature changes.

3.3 Valve Model

3.3.1 Valve Hysteresis Check

A test was run on the D/A converter, E/P transducer, valve, and pilot positioner as a unit to check that the pilot positioner (a device often installed on pneumatic actuators which uses position feedback to match actuator stroke with the pneumatic control signal) did in fact decrease hysteresis to a negligible value. Figure 3.7 shows the signal received from the water flow meter vs. the signal sent from the computer to the E/P transducer through a 12-bit D/A convertor. The test began with a (control signal) CO of 500, which was increased in increments of 10 until it reached 1800, with a 5-second delay between each increment. CO was then similarly decreased in increments of 10 until CO reached 500 again. The thin line represents the increasing control signal and the thicker line represents a decreasing control signal. The voltage measured from the flowmeter was nearly identical for both the opening and closing stroke of the valve, indicating that virtually no hysteresis existed.
Figure 3.6. Revised and original model discharge air temperature prediction comparison.
Figure 3.7: Water flowmeter signal vs. control signal.
3.3.2 Polynomial Fit

The water flowmeter used has an analog needle output reading in GPM, which was read for several steady state conditions and used to correlate the voltage read from the water flowmeter ($E_w$) with water volumetric flow rate, $Q_w$ (L/s) as in Equation 3.13.

$$Q_w = 0.0082 + 0.00707E_w$$  (3.13)

A third order polynomial, Equation 3.14, was fit to the data of Figure 3.7 in the range CO 600 to 1400, corresponding to a water volumetric flow rate range 0.341 L/s to 0.095 L/s (5.4 gpm to 1.5 gpm), to correlate CO and $E_w$, measured in millivolts.

$$E_w = a_0 + a_1[CO] + a_2[CO]^2 + a_3[CO]^3$$  (3.14)

The accuracy of this polynomial fit, as shown in Figure 3.8, was very good. Because the flowmeter was rated only down to 0.095 L/s (1.5 gpm), measurements below that cannot be considered accurate. Equations 3.13 and 3.14 were combined and the resulting mass flow rate, as a function of control signal and assuming a constant density, is given as Equation 3.15.

$$\dot{m}_w = R_3 + R_4E_w$$  (3.15)

where

- $E_w$ = Voltage (millivolts) read from water flowmeter
- $Q_w$ = Water volumetric flow rate (L/s)
- $\dot{m}_w$ = Water mass flow rate (kg/s)
- $a_0 = -41.29$
- $a_1 = 0.30932$
- $a_2 = -3.2681 \times 10^{-4}$
- $a_3 = 9.56 \times 10^{-8}$
- $R_3 = 0.008 \text{ kg/s}$
- $R_4 = 0.00703 \text{ kg/s/mv}$

38
Figure 3.8. Polynomial fit of water flowmeter signal and control signal.
3.3.3 Step Response

Figure 3.9 shows the time response of the measured water flow rate, sampled once a second, to two control signal (CO) step signals in opposite directions. At 10 s, CO is stepped up from 650 to 1250, and at 20 s CO is down from 1250 to 650. Since the previous simulations using 5-second sampling intervals showed good prediction of outlet air temperature, and the step response of water flow rate showed a time constant of less than 5 s, the valve response can be reasonably approximated as a pure time delay of one time step for analysis and simulation purposes.

3.3.4 Control Signal and Water Flow Rate Correlation

The measured and predicted water flow rate using Equation 3.14 and a one-step delay is shown in Figure 3.10. Here closed-loop control with step airflow rate changes and inlet air temperature were recorded at 5-second intervals with step disturbances in inlet airflow rate and inlet air temperature.

\[
Q_w(k) = 0.008 + 0.0071E_w[CO(k-1)] \quad 600 \leq CO(k-1) \leq 1440
\]

(3.16)

3.4 Complete Loop Model: Controller, Valve, Coil; Closed-Loop Simulation

A discrete control equation was combined with Equations 3.10, 3.11 and 3.15 to obtain a computer generated simulation of the entire closed-loop system. Figure 3.11 shows the entire loop in block diagram form. The proportional-only control law which resides in the digital computer, was chosen for its simplicity. The proportional-only control law in the positional form was given in Equation 3.6. While Equation 3.6 cannot be implemented exactly because of computation time, it is very closely approximated with a personal computer since the sampling interval was 5 s and the time required for sampling data and computing the control equation was less than 0.005 s. Proportional-only control was performed on the facility and data recorded for proportional values of 100, 205, and 410 °C. The tests lasted 32 min, 32 min, and 15 min respectively, and contained step disturbances of airflow rate, inlet air temperature, and inlet water temperature. The setpoint for all tests was 50 °C (122 °F). The measured and predicted outlet air temperatures are shown in Figures 3.12 through 3.14. The dynamics of the proportionally only closed-loop control were predicted very well. These plots of actual and predicted outlet air temperature reveal at least two significant details. First, the response of the coil, valve, and E/P transducer were all
Figure 3.9. Water flow rate vs. control signal step.
Figure 3.10. Measured and predicted water flow rate during closed-loop control.
Figure 3.11. Block diagram of the digitally controlled hot water coil.
Figure 3.12. Simulation of the proportional-only controlled hot water coil; $K_p = 100$ °C/°C.
Figure 3.13. Simulation of the proportional-only controlled hot water coil; $K_p = 205 \text{ CO/ C; } T_{sp} = ^\circ\text{C}.$
Figure 3.14. Simulation of the proportional-only controlled hot water coil; $K_p = 410$ Co/$^\circ$C; $T_sp = 50$ $^\circ$C.
predicted very well. This is true of not only closed-loop dynamic response, but also the steady-state values. Secondly, the final value of the process variable (Tao) is significantly different from its setpoint, a result of proportional-only control. As the proportional value is increased, the magnitude of this steady-state offset decreases; however the system becomes less stable. This presents a tradeoff in the tuning process. Tighter control is obtained at the expense of stability. Furthermore, the nonlinear system requires that tuning be performed at the point of highest system gain (ratio of change in discharge air temperature to change in control signal) to ensure stability over the complete range of operating conditions, resulting in sluggish control for the majority of the time. Thus, the motivation for developing a nonlinear control law can obtain superior dynamic response without sacrificing stability. This was done by first linearizing the system about several operating points, observing simulated dynamic response to a linear feedback control, and finally using those results in developing a nonlinear control law.
4. A LINEARIZED SYSTEM

4.1 Linearization of the Coil Model

Although nearly all systems are in reality nonlinear, as are heating coils, very often systems are analyzed linearly to facilitate analysis and design. Once preliminary design and/or analysis has been completed, the adequacy of the linearized model must be evaluated.

The partial derivative of Equations 3.11 and 3.12, with respect to the other variables, linearizes the model about an equilibrium point, resulting in Equations 4.1 and 4.2.

Let \( \delta X = \bar{X} - X \)

\( \bar{X} = \) Variable \( X \) at equilibrium point about which to linearize

\( X = \) Variable \( X \) at arbitrary point

\[
\begin{align*}
\delta T_{wo}(k) &= F_{11}\delta T_{wo}(k-1) + G_{11}\delta T_{sl}(k-1) + G_{12}\delta T_{vl}(k-1) + G_{13}\delta \hat{m}_a(k-1) \\
&\quad + G_{14}\delta \hat{m}_u(k-1) \tag{4.1} \\
\delta T_{ao}(k+1) &= F_{21}\delta T_{wo}(k) + F_{22}\delta T_{ao}(k) + G_{21}\delta T_{sl}(k) + G_{22}\delta T_{vl}(k) \\
&\quad + G_{23}\delta m_a(k) + G_{24}\delta m_u(k) \tag{4.2}
\end{align*}
\]

where

\( F_{11} = \frac{\delta (T_{wo}(k+1))}{\delta (T_{wo}(k))} \)

\( = 1 - A\hat{m}_u - 0.5[B + Cm_u + D\hat{m}_a] \)

\( F_{12} = \frac{\delta (T_{wo}(k+1))}{\delta (T_{ao}(k))} \)

\( = 0 \)

\( F_{21} = \frac{\delta (T_{ao}(k+1))}{\delta (T_{wo}(k))} \)

\( = 0.5[F + Cm_u + H\hat{m}_a] \)

\( F_{22} = \frac{\delta (T_{ao}(k+1))}{\delta (T_{wo}(k))} \)

\( = 1 - E\hat{m}_a \)

48
\[ G_{11} = \delta (T_{ao}(k+1))/ (T_{ao}(k)) \]
\[ = B + C\dot{m}_w + D\dot{m}_a \]
\[ G_{12} = \delta (T_{ao}(k+1))/ (T_{wo}(k)) \]
\[ = A\dot{m}_w - 0.5(B + C\dot{m}_w + D\dot{m}_a) \]
\[ G_{13} = \delta (T_{ao}(k+1))/ (\dot{m}_a(k)) \]
\[ = D[T_{a1} - 0.5(T_{a1} + T_{wo})] \]
\[ G_{14} = \delta (T_{ao}(k+1))/\delta (\dot{m}_w(k)) \]
\[ = A[T_{w1} - T_{wo}] + C[T_{a1} - T_w] \]
\[ G_{21} = \delta (T_{ao}(k+1))/\delta (T_{a1}(k)) \]
\[ = E\dot{m}_a - F - G\dot{m}_w - H\dot{m}_a \]
\[ G_{22} = \delta (T_{ao}(k+1))/\delta (T_{w1}(k)) \]
\[ = 0.5[F + G\dot{m}_w + H\dot{m}_a] \]
\[ G_{23} = \delta (T_{ao}(k+1))/\delta (\dot{m}_a(k)) \]
\[ = E[T_{a1} - T_{ao}] + H[0.5(T_{a1} + T_{wo}) - T_{a1}] \]
\[ G_{24} = \delta (T_{ao}(k+1))/\delta (\dot{m}_w(k)) \]
\[ = G[0.5[T_{w1} + T_{wo}] - T_{a1}] \]

All variables are evaluated at the equilibrium point.

Assuming the external disturbances to be zero (inlet air temperature, inlet water temperature and airflow rate constant at the equilibrium value), the dynamic response of the discharge air temperature can be calculated as Equation 4.3. Using \( z^{-1} \) as the one-step delay in the z domain, Equation 4.3 becomes the transfer function of Equation 4.4.

\[ \delta T_{ao}(k+1) = F_{22} \delta T_{ao}(k) + [G_{24} + G_{25}G_{14}] \dot{\delta m}_w(k+1) \]
\[ + G_{25}F_{11} \delta T_{wo}(k) \]  
\[ (4.3) \]

\[ \frac{\delta (T_{ao})}{\delta (\dot{m}_w)} = \frac{F_{21}G_{14} + G_{24}[z - F_{11}]}{[z - F_{11}][z - F_{22}]} \]  
\[ (4.4) \]
4.2 Linearized Valve

Linearization of Equation 3.14 gives Equation 4.5.

\[
\delta E_w = [a_1 + 2a_2 \overline{CO} + 3a_3 \overline{CO}^2] \delta CO
\]

\[
= \overline{E}_w \delta CO \quad (4.5)
\]

Equation 3.15 yields Equation 4.6 and combining this with Equation 4.5 yields Equation 4.7

\[
\delta \dot{m}_w(k) = R_4 \delta E_w \quad (4.6)
\]

\[
\delta \dot{m}_w / \delta CO = R_4 \overline{E}_w = K_w \quad (4.7)
\]

4.3 Root Locus

The linearized coil transfer function (Equation 4.4) was combined with the linearized valve transfer function to completely describe the open loop transfer function. This linearized system was next analyzed using the root locus. Since Equations 3.10 and 3.11 contain six variables, the four constant (at equilibrium) inputs can be selected arbitrarily, and then the output variables \((T_{wo} \text{ and } T_{ao})\) solved for as in Equations 4.8 and 4.9.

\[
\overline{T}_{wo} = \frac{[B + C \dot{m}_w + D \dot{m}_a][T_{a1} - 0.5 T_{w1} + \dot{A} \dot{m}_w T_{w1} + 0.5[B + C \dot{m}_w + D \dot{m}_a]}}{\dot{A} \dot{m}_w + 1.5[B + C \dot{m}_w + D \dot{m}_a]} \quad (4.8)
\]

\[
\overline{T}_{ao} = T_{a1} + \left[\left[F/Em_{a}\right] + \left[Gm_{w}/Em_{a}\right]\left[T_{wl} + T_{w0}/2 - T_{a1}\right]\right] + \left[H/Em_{a}\right]\left[T_{wl} + T_{w0}/2 - T_{wl}\right] \quad (4.9)
\]

The equilibrium value for the external disturbance variables (not controlled) were chosen as inlet air temperature of 30 °C (86 °F), inlet water temperature of 74 °C (165 °F), and airflow rate of 0.78 kg/s (1.72 lbm/s). The fourth input variable, controlled water flow rate, was varied from 0.063 L/s to 0.32 L/s (1.0 gpm to 5.0 gpm). Equations 4.8 and 4.9 were then used to calculate the corresponding equilibrium values of outlet water temperature and outlet air temperature at each water flow rate. Next, the transfer function parameters \(F_{21}, G_{14}, G_{24}, F_{11}, \text{ and } F_{22}\) were computed for each water flow rate. The coil-open loop transfer function was then computed from Equation 4.4 for each water flow rate. Program "FGFRR3.PAS" automated the procedure. Program descriptions are included in Appendix A; source code is in Appendix B. Root locus plots were then made for the five resulting transfer functions.
using a constant valve gain of 0.000442 kg/s/ΔCO, and are shown in Figures 4.1 through 4.5.

4.4 Design Criteria

As expected, the root locus revealed that as water flow rate increases, the proportional gain allowable for stable operation also increases. The desired result is a control law which varies the proportional gain (K_p) with water flow rate. One good design for HVAC control might be a critically damped system, since the primary objective is simply to get to a set point within a reasonable time period without excessive control action. In terms of root locus, this would be equivalent to the point at which the roots break from the real axis. However, the relative positions of the poles and zeros of the linearized model prevent this strategy from being practical. As Figure 4.3 shows, at certain combinations of the variables, two poles lie at the same position so that the root locus breaks away from the real axis immediately.

Alternatively, a less stringent dynamic response characteristic had to be chosen. The most common dynamic response characteristics specified in a design are percent overshoot (%OS), settling time, and rise time. For a second order system, these can be expressed as simple algebraic expressions dependent upon roots of the characteristic equation. Percent overshoot, for example, can be expressed as a function of damping ratio.²

\[
\%OS = 100\{1 - DR(0.06)\}
\]  

(4.10)

where

\[
DR = \text{Damping ratio}
\]

\[
\%OS = 100\{\text{Max value} - \text{Final value}\}/\text{[Final value} - \text{Initial value]}
\]

In terms of the root locus, Equation 4.10 represents a logarithmic spiral as outlined in Figures 4.1 through 4.5. With this choice, instead of being concerned with eliminating small oscillations entirely, which are acceptable, small overshoots are permitted and the final value is reached more quickly. This corresponds to finding the value of \( K_p \) which results in the root locations intersecting the logarithmic spiral, marked in Figure 4.2 by two squares.

Figure 4.2. Root locus of linearized coil and valve models at water flow rate of 0.126 L/s.
Figure 4.3. Root locus of linearized coil and valve models at water flow rate of 0.189 L/s.
Figure 4.5. Root locus of linearized coil and valve models at water flow rate of 0.316 L/s.
5. **PROPORTIONAL-ONLY NONLINEAR CONTROL**

5.1 **Nomenclature**

The following nomenclature is used in the discussion of the development and implementation of a nonlinear control law for the hot water coil:

- \( SP_b \) = The value of the setpoint (\( T_{sp} \)) at which steady state is obtained before an experimental test or simulation begins
- \( \delta SP \) = The amount by which \( T_{sp} \) is increased or decreased during an experimental test or simulation
- \( K_{pn} \) = Proportional gain of a nonlinear control law
- \( K_p \) = Proportional gain of a linear control law
- \( \text{error} = T_{sp} - T_{ao} \)

5.2 **Calculation of \( K_p \) Vs. Water Flow Rate for 25 percent Overshoot**

As mentioned in Chapter 4, percent overshoot (\%OS) can be calculated as a function of the closed-loop poles for a second order system. Under certain conditions, \%OS can be estimated for higher-order systems via less simple algebraic equations.\(^3\) For the general case however, simulations can be more easily iterated for the desired result. Two sets of programs, one of which used the linearized coil model while the second used the nonlinear model, were written to simulate the closed-loop step response of the coil using proportional-only control.

The first set of programs used the linearized coil transfer function of Equation 4.4. A flow chart giving the sequence of calculations and program flow is given in Appendix A as Figure A.1. First a compiled PASCAL program, "CLTF.EXE", computed the closed-loop transfer function. Given the value of the three variables of Table 5.1 along with the control variable, water flow rate, Equations 4.8 and 4.9 were used to calculate the value of the two remaining variables \( T_{wo} \) and \( T_{ao} \). Substitution of the results into Equation 4.4 then gave the coil transfer function. "CLTF.EXE" then calculated the closed-loop transfer function parameters for a proportional-only controller, and saved it in a format readable by the commercially available software package, Matlab. The Matlab program "DSTEST.M" then computed the closed-loop step response to a unit step input, automatically calculated \%OS, and iterated the

Table 5.1
Four Steady-State Conditions Used for Calculation of Nonlinear Control Law

<table>
<thead>
<tr>
<th>Case</th>
<th>$T_{at}$ (°C)</th>
<th>$T_{wt}$ (°C)</th>
<th>$\bar{m}_b$ (kg/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base</td>
<td>30</td>
<td>74</td>
<td>0.85</td>
</tr>
<tr>
<td>1</td>
<td>47</td>
<td>74</td>
<td>0.85</td>
</tr>
<tr>
<td>2</td>
<td>30</td>
<td>74</td>
<td>0.49</td>
</tr>
<tr>
<td>3</td>
<td>30</td>
<td>45</td>
<td>0.85</td>
</tr>
</tbody>
</table>

proportional gain until a %OS of 25 was found. Simulations were run for water flow rates ranging from 0.03 L/s to 0.32 L/s (0.5 gpm to 5.0 gpm) for four different cases, as listed in Table 5.1.

A second simulation program, "PSSPSELF.PAS", used the nonlinear coil model and the proportional-only control law of Equation 3.6. "PSSPSELF.PAS" performed setpoint disturbance simulations (equivalent to a step input to the transfer function of a linear system) to verify the results of the linear analysis. Simulations for conditions identical to the transfer function simulations performed using "DSTEST.M" were run, and the results verified that the linearized coil model adequately describes the behavior of the nonlinear coil for small setpoint disturbances, as seen in Table 5.2. The discharge air temperature of one such simulation calculated by "PSSPSELF.PAS", the base case at 0.063 L/s (1.0 gpm), is shown in Figure 5.1. Here $SP_b = 38.73 \, ^{оС}(101.71 \, ^{OF})$ and $\delta SP = 0.59 \, ^{оС}(1.06 \, ^{OF})$. This small setpoint disturbance was necessary to obtain nearly equal %OS for both an increase and a decrease in setpoint. Table 5.2 lists the water flow rate, $SP_b$, and $K_p$ resulting in a 25 %OS calculated by both "PSSPSELF.PAS" and "DSTEST.M". Since the transfer function is by definition linear, a step response has the same %OS for a step disturbance regardless of its magnitude or sign. However, the nonlinear simulation performed step setpoint upsets in both directions since %OS is dependent upon both the sign and magnitude of the step input. Because of this, an average is given for the %OS value calculated by "PSSPSELF.PAS". $SP_b$ of Table 5.2 represents the setpoint required for a proportional-only controller to achieve steady state control with the values of proportional gain and water flow rate listed in the table. It is important to note here that a linearized gain of 0.000442 kg/s/°С (calculated at 0.19 L/s [3 gpm], from Equation 4.5) was used for the valve in both simulation programs. Figure 5.2 shows the relation of the combined controller, transducer, and
Table 5.2
Proportional Gain for 25 %OS for the Base Case

<table>
<thead>
<tr>
<th>( Q_w ) (L/s)</th>
<th>( K_v ) (CO/°C)</th>
<th>SP(_b) (°C)</th>
<th>%OS (avg)</th>
<th>( K_p ) (CO/°C)</th>
<th>%OS</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.032</td>
<td>125</td>
<td>28.0</td>
<td>24.8</td>
<td>105</td>
<td>24.9</td>
</tr>
<tr>
<td>0.064</td>
<td>260</td>
<td>38.2</td>
<td>24.4</td>
<td>283</td>
<td>25.2</td>
</tr>
<tr>
<td>0.095</td>
<td>340</td>
<td>42.1</td>
<td>24.3</td>
<td>367</td>
<td>25.0</td>
</tr>
<tr>
<td>0.126</td>
<td>350</td>
<td>44.1</td>
<td>24.5</td>
<td>369</td>
<td>24.5</td>
</tr>
<tr>
<td>0.158</td>
<td>350</td>
<td>45.8</td>
<td>24.7</td>
<td>369</td>
<td>25.2</td>
</tr>
<tr>
<td>0.189</td>
<td>350</td>
<td>47.2</td>
<td>24.0</td>
<td>367</td>
<td>25.4</td>
</tr>
<tr>
<td>0.221</td>
<td>340</td>
<td>48.5</td>
<td>25.3</td>
<td>364</td>
<td>25.5</td>
</tr>
<tr>
<td>0.252</td>
<td>340</td>
<td>49.8</td>
<td>25.5</td>
<td>360</td>
<td>25.3</td>
</tr>
<tr>
<td>0.316</td>
<td>330</td>
<td>50.9</td>
<td>23.3</td>
<td>355</td>
<td>25.0</td>
</tr>
</tbody>
</table>

Valve gain \((K_v K_r)\) as a function of water flow rate for each case of Table 5.1 which allows for a 25 percent overshoot. These curves, obtained from the "DSTEST.M" simulation, represent the nonlinear coil gain and water flow rate relationship from which a nonlinear control law was developed. The shape of these curves was as expected. As water flow rate decreases, the magnitude of its effect on discharge air temperature increases. This can be plainly seen by computing the steady-state gain of Equation 4.4, the linearized coil transfer function according to the final value theorem, assuming that for all poles of \((1-z)P(z)\), the system lies within the unit circle.

\[
G_{coil} = \lim_{z \to 1} \frac{(z-1)}{(z-1)} P(z) \quad (5.1)
\]

where \( z/(z-1) \) = the unit step function

\( P(z) \) = the coil transfer function

\( G_{coil} \) = steady state gain of the coil
Figure 5.1. Simulated discharge air temperature using the nonlinear coil model producing a 25 percent overshoot at a nominal water flow rate of 0.063 L/s.
Figure 5.2. Proportional gain vs. water flow rate for 25 percent overshoot.
The steady-state coil gain was computed for the four cases of Table 5.1 and water flow rates from 0.0 L/s to 0.347 L/s (0.0 to 5.5 gpm). The results, plotted in Figure 5.3, are consistent with Figure 5.2. As water flow rate decreases, coil gain increases, and the proportional gain produces 25 percent overshoot.

5.3 Proportional Gain as a Function of Water Flow Rate

The goal of this work was to obtain a control law such that the controller would not have to be tuned at one particular operating point, specifically at the least stable point, in order to assure stability for the system's entire range, and thus sacrifice controller performance. That is the problem inherent in a fixed, linear control strategy.

The curves of Figure 5.2 can be described by an equation having the form:

\[ K_pK_v = a_1 \left( \frac{m_w}{m_{w_{\text{max}}}} \right)^{b_1} \]  

(5.2)

where

- \( m_{w_{\text{max}}} \) = The maximum water flow rate
- \( a_1 \) = Proportional constant \( \text{CO}/\text{C} \)
- \( b_1 \) = Dimensionless parameter

This form of curve fit was chosen because of its relatively simple form and its ability to take the shape required to fit the data. The water flow rate here is divided by the maximum possible (corresponding to a fully open valve) so that the constant \( b_1 \) would not be dependent upon the units used for water flow rate. The curves of Figure 5.2 are very similar in their shape. In order to use one of these relationships for a control law that provides for 25 percent overshoot or less for any of the four cases, the curve corresponding to the lowest \( K_p \) must be used. Therefore, the curve of Figure 5.2 corresponding to the base case of Table 5.1 was fit to the curve described by Equation 5.2. The predictions by Equation 5.3 were plotted in Figure 5.4 along with the values in Table 5.2.

\[ K_pK_v = 0.1901 \left( \frac{m_w}{m_{w_{\text{max}}}} \right)^{0.3796} \]  

(5.3)
Figure 5.3. Steady state coil gain vs. water flow rate for the four cases of Table 5.1.
Figure 5.4. Curve fit of base case KpKv vs. water flow rate for 25 percent overshoot.
The relationship of Equation 5.3, was combined with the positional form of proportional-only control (Equation 3.6) and a valve gain of 0.000442 kg/s/CO, to yield Equation 5.4.

\[ \text{CO}(k) = a_1 \left( \frac{\text{m}_w(k-1)}{0.35} \right) [T_{sp}(k) - T_{ao}(k)] + \text{bias} \] (5.4)

where

\[ \text{m}_w = \text{Water mass flow rate (kg/s)} \]
\[ \text{bias} = \text{Control signal for zero error} \]
\[ a_1 = 430 \text{ CO/°C} \]
\[ b_1 = 0.3796 \]

Since the performance analysis of a controller is always somewhat qualitative by nature, it is natural to assess the performance of a control law by comparing it to an already accepted control law. Thus, the nonlinear proportional-only controller was compared with a linear proportional-only controller.

For the hot water coil of this study, low water flow rates correspond to the highest system gains, and are therefore the least stable. A linear controller would therefore require tuning at an operating point corresponding to the lowest water flow rate for which the tuning technique does not saturate (completely close) the valve. For the system considered here, a water flow rate of 0.032 L/s (0.5 gpm) is reasonable for this system. The curves of Figure 5.2, which assume a fixed valve gain of 0.000442 kg/s/°CO, indicate that for a 25 percent overshoot at an operating point corresponding to the base case of Table 5.1 with a water mass flow rate of 0.3 kg/s (0.14 lbm/s), a \( K_p \) value of 104 CO/°C would be required. A fixed linear controller would have to be left at this value for all operating conditions in order to assure stability. The relatively small gain would result in a larger steady state offset and sluggish response at operating points requiring higher water flow rates. A nonlinear controller, developed from the relationship of Figure 5.2, however, could vary the controller gain base upon the control signal and valve characteristics to compensate for the nonlinear coil.

Several simulations of closed-loop performance were conducted for both a fixed linear controller and a nonlinear controller using Equations 3.6 and 5.4. The simulation sets a base setpoint (\( SP_b \)) and waits 75 s for steady state. The setpoint is decreased by \( \delta SP \) at 75 s, and at time 450 s, the setpoint returned to its original value. Figure 5.5 is a plot of the simulated discharge air temperature for both a linear (\( K_{pl} = 104 \text{ CO/°C} \)) and a nonlinear controller. The operating conditions for the test were those of
Figure 5.5. Simulated discharge air temperature using linear and nonlinear control; $S_{P_b} = 45 \, ^\circ\text{C}$; $\delta S_P = 5 \, ^\circ\text{C}$; base case of Table 5.1.
the base case of Table 5.1, \( SP_b = 45 \, ^\circ C \) (113 \, ^\circ F), \( \delta SP = 5 \, ^\circ C \) (9 \, ^\circ F), \( K_{pl} \) 104 \, CO/\circ C, and bias of 1207. This is not at all the desired result. The nonlinear control not only oscillates, but it also has a larger offset. The problem lies in the feedback of the varying proportional gain. When the setpoint was decreased to 40 \, ^\circ C (104 \, ^\circ F), this resulted in a steady state water flow rate of 0.05 L/s (0.84 gpm) for the linear control. The water flow rate of the nonlinear control however, oscillated between 0.0 L/s (0.0 gpm) and 0.17 L/s (2.7 gpm) as shown in Figure 5.6. When conditions change such that the valve must be closed or nearly closed, \( K_p \) approaches zero, resulting in the control signal value approaching the bias value. Since the bias value was set corresponding to a water flow rate in the middle of the flow range for zero error, the water flow rate cycled between zero and this midrange value. Two possible solutions to this were explored. One possible solution would be to change the bias value. A second possibility would be to restrict \( K_p \) to a certain range. A third possibility would be to filter the feedback variable that adjusts the proportional gain (water flow rate). The first two possibilities were considered the only viable alternatives since the third would involve adding additional constants to be chosen.

Altering the bias value may work for specific operating conditions, but is unlikely to work under varying conditions. Setting the bias such that the valve closes at \( K_p = 0 \) would provide for a zero water flow rate when \( K_p \) is zero, which is consistent with the correlation of Equation 5.2. However, this is not desirable. This would imply an infinite coil gain at zero water flow, which is not the case as evidenced by Figure 5.2. Although coil gain is certainly larger at lower water flow rates than at higher water flow rates, it does not go to infinity. Figure 5.7 shows the effect of setting the bias to close the valve at zero error. Here \( SP_b = 45 \, ^\circ C \) (113 \, ^\circ F), \( \delta SP = 5 \, ^\circ C \) (9 \, ^\circ F), bias = 1591, and the conditions of the base case in Table 5.1 exist. Notice too that this results in a final value below the setpoint. The remaining simulations use a bias value of 1207.

Therefore, Equation 5.1 is not quite the right form. To fit an equation of a more correct form, proportional gains for 25 percent overshoot of operating conditions with a water flow rate below 0.032 L/s (0.5 gpm) would be required. This could be done using simulations, but might prove to be difficult if it were required for tuning an actual system. Additionally, this would ultimately require that another controller constant be found, since a curve fit to a more complicated form would involve additional parameters. One alternative would be to assume \( K_p \) constant for water flow rates below some value. For the system used here, since it was assumed that a linear controller would have to be tuned at 0.032 L/s 0.5 gpm, the curve fit of Figure 5.4 was assumed to be flat below water flow rates of 0.032 L/s (0.5 gpm). Figure 5.8 shows the result of using the nonlinear control law of Equation 5.4 with an added condition that \( K_p \geq 105 \, CO/\circ C \). Here \( SP_b = 45 \, ^\circ C \), \( \delta SP \)
Figure 5.6. Water flow rate resulting from the simulation of linear and nonlinear control with $K_p = 104 \, \text{CO}/\text{C}$ and $K_{pnl}$ from Equation 5.4.
Figure 5.7. Simulated discharge air temperature using linear and nonlinear control; SPb = 45 °C; δSP = 5 °C; base case of Table 5.1; controller bias = 1591.
Figure 5.8. Simulated discharge air temperature using linear and nonlinear control; $SP_b = 45 \, ^\circ C$; $\delta SP = 5 \, ^\circ C$; base case of Table 5.1.
= 5 °C (9 °F), and base conditions of Table 5.1 exist. The nonlinear control decreased the steady state offset as compared to the linear control, but a small oscillation persists for the decrease in setpoint.

Since Equation 5.1 has two parameters, it allows a simple manipulation of the parameters \( a_1 \) and \( b_1 \) to make the curve fit pass through points on the low and high end. Changing the value of \( a_1 \) shifts the entire curve up and down the y axis, and changing \( b_1 \) alters the \( K_p \) value at higher water flow rates. Since setpoint decreases result in lower water flow rates, this means that \( a_1 \) could be tuned for setpoint decreases. Conversely, \( b_1 \) could be tuned for setpoint increases. The value of 104 °C/°C for \( K_p \) at a water flow rate of 0.03 L/s (0.5 gpm) was used to solve for \( a_1 \) to be 262 °C/°C, assuming \( b_1 \) to remain at 0.3796. This value for \( a_1 \) successfully eliminated oscillation for this simulation as shown in Figure 5.9, which is the same simulation as Figure 5.8 with the new value of \( a_1 \). Thus the nonlinear control law, with \( K_v = 0.000442 \) kg/s/°C, was found to be:

\[
K_p = 262 \frac{m_w}{\dot{m}_{\text{max}}}^{0.3796} \tag{5.4}
\]

Equation 5.4 was used in several simulations for the base case. Small disturbances, \( \delta SP = 5 °C \) (9 °F) were used because large disturbances tended to completely open or close the valve, regardless of the control law. Figure 5.10 shows a simulation result for the base case with \( SP_b \) 55 °C (131 °F) and \( \delta SP \) of 5 °C (9 °F). Again, the base conditions of Table 5.1 apply. This controller does provide tighter control than possible with a fixed controller. Although a fixed controller could be tuned with a greater \( K_p \), it would be unstable at lower water flow rates. For instance, if a fixed controller were tuned to give the response of the nonlinear controller in Figure 5.8, its response would be very nearly the same as the nonlinear controllers near those operating conditions. The linear controller response at a lower setpoint however, corresponding to a lower flow rate, would be unstable as evidenced by the simulation result shown in Figure 5.11.

5.4 Proportional Gain as a Function of Setpoint and Inlet Air Temperature

Since the nonlinear control law of section 5.2 did not have a greatly improved control, a second form of nonlinear control was sought. One variable which affects the process variable and is always known is the setpoint. This can be used in a feed-forward manner to compensate for setpoint disturbances. In many instances, the inlet air temperature to a coil is also known. Most air handlers in HVAC systems control the temperature supplied to a coil by mixing outside intake and return air. If control of a heating coil and this mixed air were performed by the same multi-loop
Figure 5.9. Simulated discharge air temperature using linear and nonlinear control; \( SP_b = 45 \, ^\circ C \); \( \delta SP = 5 \, ^\circ C \); base case of Table 5.1; \( K_{pl} = 105 \, CO/\circ C \).
Figure 5.10. Simulated discharge air temperature using linear and nonlinear control; SPb = 55 °C; δSP = 5 °C; base case of Table 5.1; Kpl = 105 CO/°C.
Figure 5.11. Simulated discharge air temperature using linear and nonlinear control; SPb = 35.71 °C; δSP = 109 °C; base case of Table 5.1; Kpl = 349 CO/°C.
controller, as in a direct digital control (DDC) panel, the measured mixed air temperature could be used in this control law.

The difference between setpoint and inlet air temperature is a strong indicator of the load required of the heating coil. The form of Equation 5.1, with the difference of setpoint and inlet air temperature as the independent variable makes sense as a control law. As the setpoint approaches the inlet air temperature, the water flow rate should also approach zero.

The design criteria, or the acceptable response, was chosen as one which results in a slight decaying oscillation for both the high and low system gain conditions. Two setpoints corresponding to a nearly open and nearly closed valve for the base conditions of Table 5.1 were chosen to develop a control law. Simulations were run with the setpoint varying between these setpoints, using a fixed linear controller with a bias of 120\(^\circ\)F, which corresponds to a water flow rate at half of the full range, and \(K_p\) iterated until a slight oscillation of \(T_{so}\) was observed. An increase of setpoint from 32\(^\circ\)C to 52\(^\circ\)C (90\(^\circ\)F to 126\(^\circ\)F) resulted in a slight oscillation when a value of 690\(^\circ\)C/\(^\circ\)C was used as shown in Figure 5.12. This value will hereafter be referred to as \(K_{phigh}\). The decrease in setpoint from 52\(^\circ\)C to 32\(^\circ\)C resulted in a slight oscillation when a value for \(K_p\) of 310\(^\circ\)C/\(^\circ\)C was used as shown in Figure 5.13. This value will hereafter be referred to as \(K_{plow}\). The values of \(T_{sp}, T_{a1}\), and \(K_p\) were then used to fit Equation 5.5.

\[
K_p = c_1 (T_{sp} - T_{a1})
\]

(5.5)

where

\[
c_1 = 246
\]

\[
d_1 = 0.3337
\]

Figure 5.14 shows the resulting control gain. By fitting a multiplicative curve through these two points, superior control by a nonlinear controller using this as a gain equation is assured. This is also very appealing for implementation concerns, as it would simply require observing system response at two setpoints.

Figure 5.15 shows the results of a simulation using the nonlinear control law of Equation 5.5 and the linear control law of Equation 3.6 with \(K_{pl} = 690\)\(^\circ\)C/\(^\circ\)C. The base conditions of Table 5.1 were used. The linear controller matches the performance of the nonlinear for the setpoint increase, but it is oscillatory at the lower setpoint. The simulation was run again but this time with \(K_{pl} = 310\)\(^\circ\)C/\(^\circ\)C and the resulting \(T_{so}\) plotted in Figure 5.16. The linear controller nearly matches the nonlinear controller at the lower setpoint in terms of both steady state error and dynamic response, but it cannot provide the steady state accuracy of the
a nonlinear controller using this as a gain equation is assured. This is also very appealing for implementation concerns, as it would simply require observing system response at two setpoints.

Figure 5.15 shows the results of a simulation using the nonlinear control law of Equation 5.5 and the linear control law of Equation 3.6 with $K_{p1} = 690 \text{ CO}^\circ/\text{C}$. The base conditions of Table 5.1 were used. The linear controller matches the performance of the nonlinear for the setpoint increase, but it is oscillatory at the lower setpoint. The simulation was run again but this time with $K_{p1} = 310 \text{ CO}^\circ/\text{C}$ and the resulting $T_\infty$ plotted in Figure 5.16. The linear controller nearly matches the nonlinear controller at the lower setpoint in terms of both steady state error and dynamic response, but it cannot provide the steady state accuracy of the
Figure 5.12. Simulated discharge air temperature using linear control for a setpoint upset from 32 °C; to 52 °C; base case of Table 5.1; $K_p = 690$ CO/°C.
Figure 5.13. Simulated discharge air temperature using linear control for a setpoint upset from 52 °C to 32 °C; base case of Table 2; Kp1 = 310 CO/°C.
Figure 5.14. Multiplicative curve fit for two proportional gains and setpoint minus inlet air temperature.
Figure 5.15. Simulated discharge air temperature using linear and nonlinear control; $SP_b = 32 \, ^\circ C$; $\Delta SP = 20 \, ^\circ C$; base case of Table 5.1.
Figure 5.16. Simulated discharge air temperature using linear and nonlinear control; \( \text{SPb} = 32 \, ^\circ\text{C} \); \( \delta \text{SP} = 20 \, ^\circ\text{C} \); base case of Table 5.1.
nonlinear controller at higher setpoints. The operating conditions in between were also simulated and the nonlinear controller performed well here also (Figures 5.17 and 5.18).

5.5 Implementation of the Setpoint-Dependent Proportional-Gain Nonlinear Controller on the Test Facility

The simulations showed that the nonlinear controller can outperform the linear controller only slightly. The simulations however, did not take into account the effects of a nonlinear valve. The nonlinear effects of the valve were expected to enhance the performance advantage of a nonlinear control law.

Implementation of the nonlinear controller involves selecting the constants for a system for which the gain characteristics are not known. Thus, Equation 5.5 was tuned on the test facility as if no knowledge of its dynamics were available. One very common method for tuning linear proportional controllers is a trial-and-error approach very similar to those employed in determining the constants of Equation 5.5 in the simulations. The general idea then is to find an acceptable gain for two operating points and to derive a nonlinear gain equation from those gains and some third variable that changes and is always known (the setpoint).

The design criteria, or the acceptable response, was chosen to be one that results in a slight decaying oscillation for both the high and low system gain conditions. The first step for tuning the control Equation 5.5 was to determine the maximum and minimum possible values for the discharge air temperature. Figure 5.19 shows a step in CO from a fully closed to fully open valve. The maximum obtainable temperature is 55 °C (131 °F) and the minimum 32 °C (90 °F). Although the upper value seems quite reasonable, the minimum value should theoretically be equal to the inlet air temperature which was 26 °C. The reality is that the valve never completely shuts off the flow unless the pump is turned off. The next step was to conduct closed-loop step setpoint response tests. Since the maximum and minimum values for $T_o$ are 55 °C and 32 °C, a guess that setpoints of 50 °C and 35 °C would nearly fully open and close the valve were used in conducting tests. A test which did not exercise the valve through its full range was conducted first in order to obtain a first guess for $K_{p\text{high}}$ and $k_{p\text{low}}$. (Figure 20)

The next test performed, whose discharge air temperature is plotted in Figure 5.21, used $SP_b = 40 ^\circ C$, $\Delta SP = 10 ^\circ C$, and $K_p = 600 CO/\circ C$. Here continued oscillation provided the information that $K_{p\text{high}} \leq 600$ and $K_{p\text{low}} < 600 CO/\circ C$. Next, $K_p$ was decreased to 500 CO/°C and the response of Figure 5.22 observed. This meant that $K_{p\text{high}} \leq 500 CO/\circ C$ depending upon the allowable amount of oscillation. Since the loop would likely require additional fine tuning, this was assumed to be the desired value for the time being. Figure 5.23 shows the observed result of decreasing $K_p$ to 350 CO/°C and
Figure 5.17. Simulated discharge air temperature using linear and nonlinear control; SPb = 50 °C; δSP = 5 °C; base case of Table 5.1.
Figure 5.18. Simulated discharge air temperature using linear and nonlinear control; SPb = 40 °C; \( \delta SP = 5 \) °C; base case of Table 5.1.
Figure 5.19. Measured discharge air temperature for a control signal step from 1700 to 400.
Figure 5.20. Measured discharge air temperature for linear control on test facility; 
SPb = 35 °C; δSP = 15 °C; Kpl = 300 CO/°C.
Figure 5.21. Measured discharge air temperature for linear control on test facility; SPb = 40 °C; δSP = 10 °C; Kpl = 600 CO/°C.
Figure 5.22. Measured discharge air temperature for linear control on test facility; SPb = 40 °C; δSP = 10 °C; Kpl = 500 CO/°C.
Figure 5.23. Measured discharge air temperature for linear control on test facility; SPb = 45 °C; δSP = 5 °C; Kpl = 350 CO/°C.
observing the step response with $SP_b = 45 \, ^\circ C$ and $\delta SP = 5 \, ^\circ C$. Thus, $K_{plow}$ was assumed to be 300 CO/$^\circ C$.

Using the values obtained from the tuning tests ($K_{p_{high}} = 500$ CO/$^\circ C$, $K_{plow} = 300$ CO/$^\circ C$), the control law of Equation 5.6 was obtained by solving for $a_i$ and $b_i$ using the value of $T_{ai}$ (26 $^\circ C$), which was measured during the tests.

$$K_p = 95.3[T_{sp} - T_{ai}]^{0.5208} \, \text{CO/}^\circ C$$

The results of implementing Equation 5.6 in the positional control law of Equation 3.6 is shown in Figure 5.24. Here, $SP_b = 35 \, ^\circ C$ (95 $^\circ F$), $\delta SP = 5 \, ^\circ C$ (9 $^\circ F$). The response is oscillatory at both high and low setpoints, suggesting that the multiplicative constant, 95.3 is too high. For the inlet air temperature present during this test, Equation 5.6 gave a gain of 530 CO/$^\circ C$, higher than the maximum found in the earlier tests. A value of 475 CO/$^\circ C$ was assumed for $K_p$ and the control law of Equation 5.7 was calculated assuming the power constant, 0.5208.

$$K_p = 85[T_{sp} - T_{ai}]^{0.5208} \, \text{CO/}^\circ C$$

The disturbance with $SP_b = 35 \, ^\circ C$, $\delta SP = 15 \, ^\circ C$ was again performed using Equation 5.6 An acceptable response was observed, so a linear controller was used for the same test for comparison. A proportional gain corresponding to a stable value at high setpoints (low water flow rates) was used, $K_{pl} = 310$ CO/$^\circ C$. The results plotted in Figure 5.25 were very similar.

It is possible that this system simply does not have enough nonlinearities to take advantage of the nonlinear control law. Although it was shown that the coil studied had extremely nonlinear steady state gain between 0.0 L/s and 0.06 L/s in which the full flow rate is 0.34 L/s, beyond 0.06 L/s, the coil gain was linear. Because of this, the nonlinear control law could only outperform the linear control law if setpoints corresponding to this very narrow range of operation were used. Setpoints corresponding to that operating range physically made no sense. Because the proportional-only controller always has an offset, the temperature achieved, depending on the bias value used, is always either greater than or less than the setpoint. The usual practice is to pick the bias value such that when the error is zero, the bias drives the actuator to provide a flow rate in the midrange of the valve. In that case, for setpoints near that of the inlet air temperature, the resulting discharge air temperature is greater than the setpoint. The result is that, in order to achieve water flow rates in the range that result in highly nonlinear coil gain, setpoints below that of the inlet air temperature were required. Figure 5.26 shows the resulting measured discharge air temperature.
Figure 5.24. Measured discharge air temperature for nonlinear control on test facility; SPb = 45 °C; δSP = 15 °C; Kpnl = Equation 5.5.
Figure 5.25. Measured discharge air temperature for nonlinear control on test facility; SPb = 35 °C; δSP = 15 °C; Kpnl = Equation 5.5.
Figure 5.26. Measured discharge air temperature for linear and nonlinear control on test facility; \( SP_b = 45 \, ^oC \); \( \delta SP = 5 \, ^oC \); \( K_{pl} = 310 \, \text{CO} / ^oC \).
of two separate tests, the first of which used the linear control law with $K_{pl} = 310 \, \text{C0/°C}$ and the second, which used Equation 5.6. Here the setpoint was decreased from $35 \, ^\circ\text{C}$ to $29 \, ^\circ\text{C}$ in both tests. As the simulations predicted, the nonlinear controller provided stable control while the linear controller oscillated. Although this represents an artificially low setpoint, the same results could be expected for other conditions with realistic setpoints requiring low water flow rates such as lower airflow rates, higher inlet water temperatures, and higher inlet air temperatures.
6. SUMMARY AND CONCLUSIONS

Much of the preliminary work performed for this study concerned the data acquisition hardware and software. Thermocouples for temperature measurement performed very well. Hot wire anemometers for airflow rate measurement should be avoided if possible. The software developed for data acquisition and analysis would greatly simplify the task of any future work performed on the test facility or on the models created.

The goals of this work were to develop an accurate dynamic model for analysis of closed-loop control. A model was developed to be accurate over a wide range of conditions, including a closed-loop test with several simultaneously changing variables. The major difference between this model and previous models is its verified accuracy under closed-loop conditions. One reason for this is a feed-forward term involving the inlet air temperature.

Two nonlinear proportional-only control laws were developed and experimentally verified to work slightly better than a proportional-only controller. The first nonlinear control law, which used water flow rate to vary the proportional gain, was unstable for many operating conditions. This was solved by preventing the proportional gain from assuming values below that at which a linear controller would have been tuned. The difference between this resulting controller and the fixed linear controller were still not impressive. Experimental validation of this controller’s performance was not performed.

A second nonlinear controller was designed so that the proportional gain was a function of the setpoint and inlet air temperature. This controller was also better than the fixed linear controller, but the difference was not impressive. This controller was implemented on the test facility and compared to a fixed linear proportional-only controller. The experimental results were found to give very similar results to the simulations.
APPENDIX A. SOFTWARE DESCRIPTION

A.1 "PIDSIMM.PAS"

To automate the data acquisition, modeling, analysis, and control of the hot-water-to-air heat exchanger, an integrated set of programs were written using the TURBO PASCAL programming language. The 4.0 version of TURBO PASCAL utilizes units, a collection of compiled procedures and functions. One advantage of units lies in the ability to compile, test, and debug them separately from a main program. Once a unit is completely debugged and compiled, it need never be recompiled. The main or calling program has the name scheme of *.PAS while the compiled unit called by the main program has the name scheme of *.TPU where the * can be any eight or fewer combination of alphanumeric characters.

The main program for data acquisition, modeling, and simulation is "PIDSIMM.PAS". PIDSIMM.PAS calls on units "FACILITY.TPU", "FROMFILE.TPU", "DRAW.TPU", and "MODELFIT.TPU" as shown in the flow chart Figure A.1.

A.2 "FACILITY.TPU" and "FROMFILE.TPU"

Data acquisition is accomplished by the "FACILITY.TPU" unit. Packaged routines for accessing the analog-to-digital and digital-to-analog interfaces of the Metrabyte DAS-16 and EXP-16 boards were used to acquire system data inlet and outlet air temperature, inlet and outlet water temperature, airflow rate, and water flow rate.

As seen in the flowchart of psimm.drw, if data already acquired is to be analyzed, the compiled unit "FROMFILE.TPU" is run. First the user is asked the number of data points to be taken, whether upsets are to be automatically initiated, and what control algorithm to use. Choosing algorithm 0 results in an open-loop test.

A.3 "FITMODEL.TPU"

Models were fit by "FITMODEL.TPU" using a batch form of multiple regression least squares. This allowed a model to be fit immediately after data acquisition. Once parameters were calculated, they were saved on disk to the file Param.dat.

A.4 "DSTEST.M"

This is an M file, a set of Matlab commands executed automatically in the Matlab environment. "DSTEST.M" first calls compiled TURBO PASCAL program "FGFRR3.EXE," which computes the linearized closed-loop transfer function for proportional-only control. Once the transfer function is computed, it is saved to a disk file which is read by Matlab. Next the step response is
simulated and a $\%OS$ is computed. Depending on the value of $\%OS$, the proportional gain is increased or decreased until a $\%OS$ greater than 24 and less than 25 is found, terminating the program. The flow chart for "DSTEST.M" is shown in Figure A.1.

A.5 "READMAT.PAS"

This is a TURBO PASCAL program to read a matrix saved in the Matlab format and save it in ASCII format. This was used for re-plotting the root loci plots.

A.6 "PSSPSELF.PAS"

This is a PASCAL program to simulate the response of the nonlinear coil model to a setpoint step using both a linear and a nonlinear control law.
User inputs:
Kp start
Kp increment

Old parameter file erased

New parameter file saved
F_GFRR3 calculates
closed loop TF and saves
to MATLAB format file

Read TF from file
Calculate step response
Calculate % overshoot

24.5 < %OS < 25.5

%OS > 25
and increment > 0

0 < δ%OS ≤ 5
and %OS < 20

Increment = -Increment/2

Increment = -2*Increment

Increment MdotW

MdotW > 5.5

Yes
Stop

Figure A1. Flow diagram of Matlab program DSTEST.M.

98
APPENDIX B: PROGRAM LISTINGS

"PIDSIMM.PAS"

{$R+}  (Range checking off)
{$B+}  (Boolean complete evaluation on)
{$S+}  (Stack checking on)
{$I+}  (I/O checking on)
{$M 65500,16384,655360}  (Turbo 3 default stack and heap)

program PIDSIMM(input,output);  (7-19-89)

Uses Dos, Crt, Draw, Fitmodel, Facility, Fromfile, Pictures;

type  Data_array = array[0..820,1..7] of real;  (maximum 800
datapoints:

  1: Inlet air temperature
  2: Outlet air temperature
  3: Inlet water temperature
  4: Outlet water temperature
  5: Water flow
  6: Air flow
  7: Control signal)

type  Parameters = record  (model parameters)
        date: string;
        a: real; b: real; c: real; d: real; e: real;
        f: real; g: real; h: real; j: real;
    end;

type  Information = record  (miscellaneous data)
        Filename: string;
        Datatype: integer;  {0: facility, 1: from file}
        Datapoints: integer;  {number of datapoints}
        Textfile: integer;  {write to text file ? 0: no, 1: yes}
        Transducer: integer;  {0: venturi, 1: pitot tube, 2: hot wire
                      anemometer}
        PID: record  (PID algorithm, Ki, Kp, Td)
               alg:integer; Ki: real; Kp: real; Td: real; end;
               (PID alg  0: open loop
                1: interacting rectangular
                2: non-interacting velocity
                3: constant
                4: step
                5: self tune)
        Newparam: integer;  {calculate new model parameters ? 0: no,
                      1: yes}
        Setpoint: real;
        Steadystate: integer;  {find steady state before beginning
                      test?}
                     0: no, 1: yes)
        Upsets: record
               Run: integer;  {run upsets ? 0: no, 1: yes}

99
Double: integer; (adjust water flow after upset)
Damperon: integer; (close damper)
Damperoff: integer; (open damper)
Heateron: integer; (turn 15kw heaters on)
Heatersoff: integer; (turn 15kw heaters off)
Coldwateron: integer; (add cold water to system)
Coldwateroff: integer; (turn cold water off)
Setpointon: integer; (raise setpoint)
Setpointoff: integer; (lower setpoint)
Setpointsize: real; (size of setpoint error)
Wtrflowhigh: integer; (in PID alg 4, increase water flow)
Wtrflowlow: integer; (in PID alg 4, decrease water flow)

end;
Paramselect: integer; (parameter set used in test)
Error_model_measured: real; (average error between measured outlet air temperature and calculated outlet air temperature)
Error_setpoint_measured: real; (average error between setpoint and measured outlet air temperature)
Error_setpoint_model: real; (average error between setpoint and calculated outlet air temperature)
end;

Const max_buffer = 1000;
GPM_to_Kgps = 0.06296382; (conversion for water flow, gallons per minute to kilograms per second)
FPM_to_kgps = 0.001221041; (conversion for air flow, feet per second to kilograms per second)

var J, K, Code: integer;
X: char;
Year, Month, Day, Dayofweek: word;
Monthstr, Daystr: string;
P: Parameters;
I: Information;
Paramfile: file of Parameters;
Infofile: file of Information;
Filename: string;
Numberofrecords: integer;
stopt: integer; (variable returned from facility or from file to stop program)
Paramselecttemp: integer;

function getvalI(default: integer): integer;
(read string from screen and convert it to integer)
var Z: string;
Z_val, code: integer;

100
begin
    readln(Z); if Z <> '' then
    begin
        val(Z,Z_val,code);
        getvalI := Z_val;
    end
    else
    getvalI := default;
end;

function getvalR(default: real): real;
    (read string from screen and convert it to real)
var Y: string;
    Y_val: real;
    code: integer;
begin
    readln(Y); if Y <> '' then
    begin
        val(Y,Y_val,code);
        getvalR := Y_val;
    end
    else
    getvalR := default;
end;

procedure findfile(filename: string; var fileexists: integer);
    (uses "findfirst" and "findnext to determine if file requested exists)
var K : integer;
    fileinfo : searchrec;
    lastfile : string;
begin
    fileexists := 1;
    findfirst('\THESIS\DATA\*.*',anyfile,fileinfo);
    if fileinfo.name <> filename then begin
    repeat
        findnext(fileinfo);
        if lastfile = fileinfo.name then
            fileexists := 0;
        lastfile := fileinfo.name;
        until (fileinfo.name = filename) or (fileexists = 0);
    end;
end;

procedure readinfo;
    (determines which information file to use and reads it)
var datatypetemp: integer;
    Fileexists: integer;
begin
    .  101
datatypetemp := I.datatype;
if I.datatype = 0 then
    assign(Infofile,'\THESIS\DATA\INFO.DAT') {reserved information
    file with standard data}
else begin
    findfile(I.Filename+'.IN2',Fileexists);
    if Fileexists = 1 then
        assign(Infofile,'\THESIS\DATA\'+I.Filename+'.IN2')
    else begin
        findfile(I.Filename+'.IN1',Fileexists);
        if Fileexists = 1 then
            assign(Infofile,'\THESIS\DATA\'+I.Filename+'.IN1')
        else
            assign(Infofile,'\THESIS\DATA\'+I.Filename+'.INF');
    end;
end;
reset(Infofile); {open information file}
read(Infofile,I);
close(Infofile);
I.Datatype := datatypetemp;
end;

procedure getupsets;
begin
    window(1,1,80,25); clrscr;
    writeln;
    writeln('Select Upsets -1);
    drawbox(1,3,80,20); window(2,4,80,20);
    write(' Enter 1 for water flow step change(l) - '); repeat
        I.Upsets.double := getvalI(1);
        until I.Upsets.double in [0..1];
    if I.PID.alg > 0 then
        begin
            write(' The setpoint is (',I.Setpoint:2:0,') - ');
            I.Setpoint := getvalR(I.Setpoint);
        end;
    write(' Time to close damper(',I.Upsets.Damperon:4,') - ');
    I.Upsets.Damperon := getvalI(I.Upsets.damperon);
    write(' Time to open damper(',I.Upsets.Damperoff:4,') - ');
    I.Upsets.Damperoff := getvalI(I.Upsets.damperoff);
    write(' Time to turn heaters on(',I.Upsets.Heaterson:4,') - ');
    I.Upsets.Heaterson := getvalI(I.Upsets.heaterson);
    if I.Datatype = 0 then begin
        write(' Time to turn heaters off(',I.Upsets.Heatersoff:4,') - ');
    end else
    I.Upsets.Heatersoff := I.Upsets.Heaterson + 505; {if upsets are
    added to data from file, the heaters must remain on for 505
    seconds} write(' Time to turn cold water

102
on(',I.Upsets.Coldwateron:4,') - ');
I.Upsets.Coldwateron := getvalI(I.Upsets.coldwateron);
if I.Datatype = 0 then begin write(' Time to turn cold water
off(',I.Upsets.Coldwateroff:4,') - ');
I.Upsets.coldwateroff := getvalI(I.Upsets.coldwateroff)
end
else
I.Upsets.Coldwateroff := I.Upsets.Coldwateron + 500; (if upsets
are added to data from file, coldwater must remain on for at
least 500 seconds) if I.PID.alg > 0 then begin write(' Time to
increase setpoint(',I.Upsets.Setpointon:4,') - ');
I.Upsets.Setpointon := getvalI(I.Upsets.setpointon);
write(' Time to decrease setpoint(',I.Upsets.Setpointoff:4,')
- ');
I.Upsets.Setpointoff := getvalI(I.Upsets.setpointoff);
write(' Size of setpoint upset(',I.Upsets.Setpointsize:2:0',
- ');
I.Upsets.Setpointsize := getvalR(I.Upsets.setpointsize);
end
else
begin
I.Upsets.Setpointon := 0;
I.Upsets.Setpointoff := 0;
end;
end;

procedure correctinfo;
var Left, Top, Right, Bottom : integer;
fileinfo          : searchrec;
Filename,Filename_ : string;
Fileexists,Code   : integer;
begin
window(1,1,80,25);
writeln(' For Data from the facility enter 0,');
write (' For Data from a file enter the filename - ');
readln(I.Filename);
Filename_ := '';
for K := 1 to length(I.filename) do
  Filename_ := Filename_ + upcase(I.Filename[K]);
I.Filename := Filename_;
if I.Filename = '0' then val(I.Filename,I.Datatype,code)
else begin
  I.Datatype := 1;
  findfile(I.Filename+'.DAT',fileexists);
  if fileexists <> 1 then begin
    textcolor(white+blink);
    writeln; write(' File does not exist, press ctrl-break');
    write(chr(7));
    X := readkey;
  end;
end;
drawbox(1,3,80,23);
Filename_ := I.Filename;
readinfo;
I.Filename := Filename_; 
drawbox(1,3,80,5); window(2,4,80,5);
if I.Datatype = 0 then begin
  write(' Number of datapoints - '); 
  readln(I.Datapoints);
  for K := 32 to 60 do write(' ');
  highvideo; write('Run time = ',I.Datapoints*5); lowvideo;
end
else begin
  for K := 2 to 60 do write(' ');
  highvideo; write('File: ',I.Filename); lowvideo;
end;
window(1,1,80,25); drawbox(1,5,80,9); window(2,6,80,9);
if I.Datatype = 0 then begin
  writeln(' Enter 0 for air flow Data from Venturi,');
  writeln(' enter 1 for air flow Data from Pitot tube,');
  writeln(' enter 2 for air flow Data from Hot Wire Anemometer(0)
  - ');
  repeat
    I.Transducer := getvalI(0);
  until I.Transducer in [0..2];
end
else begin
  write(' Enter 1 to write to an ASCII file(0) - '); 
  repeat
    I.Textfile := getvalI(0);
  until I.Textfile in [0..1];
  write(' Enter 1 to calculate new parameters(0) - ');
  repeat
    I.Newparam := getvalI(0);
  until I.Newparam in [0..1];
end;
window(1,1,80,25); drawbox(1,9,80,20); window(2,10,80,20);
if (I.Newparam = 0) or (I.datatype = 0) then begin
  writeln;
  writeln(' 0 = open loop');
  writeln(' 1 = interacting positional PID rectangular approx
  integral');
  writeln(' 2 = non-interacting velocity PID');
  writeln(' 3 = Constant');
  writeln(' 4 = Step ');
gotoxy(1,1); write(' Choose pid alg(',I.PID.alg,') - ');
  repeat
    I.PID.alg := getvalI(I.PID.alg);
  until I.PID.alg in [0..4];
  if(I.PID.alg in [1..2,4]) then begin
    window(1,1,80,25);
case I.PID.alg of
  1..2 : begin
    write(' Ki(',I.PID.Ki:2:2,') - ');
    I.PID.Ki := getvalR(I.PID.Ki);
    write(' Kp(',I.PID.Kp:2:2,') - ');
    I.PID.Kp := getvalR(I.PID.Kp);
    write(' Td(',I.PID.Td:2:2,') - ');
    I.PID.Td := getvalR(I.PID.Td);
    I.Upsets.wtrflowhigh := 0;
    I.Upsets.wtrflowlow := 0;
  end;
  4 : begin
    write(' Time to set water flow hi(')
    I.Upsets.wtrflowhigh := getvalI(0);
    write(' Time to set water flow low(')
    I.Upsets.wtrflowlow := getvalI(0);
  end;
end;

procedure readparameters;
var Left, Top, Right, Bottom: integer;
empty: char;
begin
  window(1,1,80,25); clrscr;
  drawbox(1,1,80,24);
  window(1,1,80,25); drawbox(1,20,80,23);
  if I.Datatype = 0 then begin
    write(' Enter 1 to check for steady state(1) - ');
    repeat
      I.Steadystate := getvalI(1);
      until I.Steadystate in [0..1];
    write(' Enter 1 to run upsets(1) - ');
    repeat
      I.Upsets.run := getvalI(1);
      until I.Upsets.run in [0..1];
  end
  else if I.Newparam = 0 then begin
    write(' Enter 1 to add upsets to file(0) - ');
    repeat
      I.Upsets.run := getvalI(0);
      until I.Upsets.run in [0..1];
  end;
  if I.Upsets.run = 1 then getupsets;
end;
assign(Paramfile,'\THEESIS\DATA\PARAMF.DAT');
reset(Paramfile);
Numberofrecords := filesize(Paramfile);
J := 0; K := 0;
repeat
  seek(Paramfile,J);
  read(Paramfile,P);
  gotoxy(2,(K*3)+2);
  write(J:2,' Parameters from: ',P.date);
  gotoxy(2, (K*3)+3);
  gotoxy(2, (K*3)+4);
  J := J + 1; K := K + 1;
  if K/7 = 1 then
    begin
      gotoxy(2,25); write('Press any key for more...');
      X := readkey;
      clrscr;
      drawbox(1,1,80,24);
      K := 0;
    end;
  until EOF(Paramfile);
  gotoxy(2,25);
  if I.Datatype = 0 then
    write('Select parameter set(',Numberofrecords - 1,' - ') 
  else
    write('Select parameter set(',I.Paramselect,') - '); 
  while keypressed do empty := readkey;
  if I.Datatype = 0 then
    I.Paramselect := getvalI(Numberofrecords - 1)
  else
    I.Paramselect := getvalI(I.Paramselect);
  seek(Paramfile,I.Paramselect);
  read(Paramfile,P);
  close(Paramfile);
  Paramselecttemp := I.Paramselect;
end;

Begin
clrscr;
randomize;
textcolor(random(6) + 1);
getdate(Year,Month,Day,Dayofweek);
str(Month,Monthstr);
str(Day,Daystr);
I.Filename := Monthstr+'-'+Daystr;
correctinfo;
if (I.datatype = 1) and (I.Newparam = 0) then readparameters;
if I.Datatype = 0 then
begin
  if I.PID.alg = 0 then I.Filename := Monthstr+'-'+Daystr+'OL';
  if I.PID.alg > 0 then I.Filename := Monthstr+'-'+Daystr+'CL';
end;
if I.datatype = 0 then begin
  assign(Infofile,'\THESIS\DATA\INFO.DAT');
  rewrite(Infofile);
  write(Infofile,I);
  close(Infofile);
  assign(Infofile,'\THESIS\DATA\'+I.Filename+' .IN1');
  rewrite(Infofile);
  write(Infofile,I);
  close(Infofile);
  Datafromfacility(I.Filename,stop);
end
else begin
  assign(Infofile,'\THESIS\DATA\'+I.Filename+' .IN2');
  rewrite(Infofile);
  write(Infofile,I);
  close(Infofile);
  if I.Newparam = 1 then Findparameters(I.Filename)
  else begin
    Datafromfile(I.Filename,stop);
    if stop <> 0 then plotdata(I.Paramselect,I.Filename);
  end;
end;
End.

107
 UNIT Facility;

 Interface
 Uses dos,crt,tp4dl6,tp4misc,tp4tclin,stdhdr,sumstats,pictures;

 procedure Datafromfacility(Filename: string; var Stop:integer);

 type Data_array = array[0..820,1..7] of real; (maximum 800 datapoints:
  1: Inlet air temperature
  2: Outlet air temperature
  3: Inlet water temperature
  4: Outlet water temperature
  5: Water flow
  6: Air flow
  7: Control signal)

 type Parameters = record  (model parameters)
     date: string;
     a: real; b: real; c: real; d: real; e: real; f: real; g: real;
     h: real;
   end;

 type Information = record  (miscellaneous data)
     Filename: string;
     Datatype: integer; (0: facility, 1: from file)
     Datapoints: integer; (number of datapoints)
     Textfile: integer; (write to text file ? 0: no, 1: yes)
     Transducer: integer; (0: venturi, 1: pitot tube, 2: hot wire anemometer)
     PID: record (PID algorithm, Ki, Kp, Td)
         alg:integer; Ki: real; Kp: real; Td: real; end;
         (PID alg 0: open loop
         1: interacting rectangular
         2: non-interacting velocity
         3: constant
         4: step
         5: self tune)
     Newparam: integer; (calculate new model parameters ? 0: no,
                         1: yes)
     Setpoint: real;
     Steadystate: integer; (find steady state before beginning
     108
test? 0: no, 1: yes)
Upsets: record
  Run: integer; (run upsets? 0: no, 1: yes)
  Double: integer; (adjust water flow after upset)
  Damperon: integer; (close damper)
  Damperoff: integer; (open damper)
  Heaterson: integer; (turn 15kw heaters on)
  Heatersoff: integer; (turn 15kw heaters off)
  Coldwateron: integer; (add cold water to system)
  Coldwateroff: integer; (turn cold water off)
  Setpointon: integer; (raise setpoint)
  Setpointoff: integer; (lower setpoint)
  Setpointsizes: real; (size of setpoint error)
  Wtrflowhigh: integer; (in PID alg 4, increase water flow)
  Wtrflowlow: integer; (in PID alg 4, decrease water flow)
end;
Paramselect: integer; (parameter set used in test)
Error_model_measured: real; (average error between measured outlet air temperature and calculated outlet air temperature)
Error_setpoint_measured: real; (average error between setpoint and measured outlet air temperature)
Error_setpoint_model: real; (average error between setpoint and calculated outlet air temperature)
end;

{global declarations}
Const max_buffer = 1000; (pascal requirement)
  GPM_to_Kgps = 0.06296382; (conversion for water flow, gallons per minute to kilograms per second)
  FPM_to_kgps = 0.001221041; (conversion for air flow, feet per second to kilograms per second)
  Dubup = 1050; (size of change in control signal for adjustment after upset)

var I: Information;
P: Parameters;
Infofile: file of Information;
K: integer;
  Sum_error: real; (error between setpoint and outlet air temperature, used in PID procedure)
aot_count_real, aot_count_real_old: real; (control signal in bytes declared as real)
Filename: string;
Ks,Kr,Kc:real; (PID values for PID alg 5)
pid5flag: integer; (boolean for using PID alg 5)

Implementation

procedure Init_boards; (initialize dash16 and PIO12 boards)
var board_num, int_level, dma_level, base adr, 
dig_cntrl_code, port_num, err_code: integer;

begin 
board_num := 0; int_level := 7; dma_level := 1; base_adr := 768; 
d16_init(board_num, base_adr, int_level, dma_level, err_code);

board_num := 1; int_level := 7; dma_level := 1; base_adr := 816; 
dig_cntrl_code := 128;
pio12_init(board_num, base_adr, dig_cntrl_code, err_code);
end;

procedure pio12_(dot_out: integer); {signal to control upsets} 
var board_num, err_code, port_num: integer;

begin 
board_num := 1; port_num := 0;
pio12_bous(board_num, port_num, dot_out, err_code);
if err_code <> 0 then 
d16_print_error(err_code)
end;

procedure d16ains_(chanlo: integer; var dataval_ains:integer); 
var board_num, err_code: integer; {to read data}

begin 
board_num := 0;
d16_ains(board_num, chanlo, dataval_ains, err_code);
if err_code <> 0 then 
d16_print_error(err_code)
end;

procedure d16bous_(var dataval_bous:integer); 
var board_num, err_code: integer; {set exp16 channel to read}

begin 
board_num := 0;
d16_bous(board_num, dataval_bous, err_code);
if err_code <> 0 then 
d16_print_error(err_code)
end;

(*****************************************************************************************)
procedure Datafromfacility;
var Data : Data_array;
   CO: integer; {control signal in bytes}
   New_error, Old_error, Old_old_error: real; {error between

110
setpoint and outlet air temperature, used in PID procedure)  
*******************************************************************************

procedure upsets(Filename: string; var stop: integer; initialflag: integer);

var Y: string;
Y_val, code, err_code: integer;
dot_out: byte; (output to control upsets, passed to PI012_procedure)
infofile: file of information;
Z: string;
Z_val: integer; (convert string to integer)
empty: char;
begin
if initialflag = 1 then begin (in first pass through program read
information file, and initialize
data)
assign(Infofile, '\thesis\data\'+Filename+' .ini');
reset(Infofile); (open information file)
read(Infofile,I);
close(Infofile);
(write all information to screen)
clrscr;
drawbox(3,9,80,24);
gotoxy(5,10);
write('The upsets from the file \thesis\data\'+I.Filename+' .inf
are - ');
gotoxy(5,12); write('Setpoint is: ','I.Setpoint:2:0, '
degrees');
gotoxy(5,13); write('Steady state: ');
if I.Steadystate = 1 then write('yes') else
write('no');
gotoxy(5,14); write('Filename: \thesis\data\',I.Filename,'.*');
gotoxy(5,15); write('Datatype: ');
if I.Datatype = 1 then write('from file') else
write('from facility');
write(' Datapoints: ','I.Datapoints,' Time: ',I.datapoints*5,' Textfile: ');
if I.Textfile = 1 then write('yes') else
write('no');
gotoxy(5,16); write('PID alg: ');
case I.PID.alg of
0: write('open loop');
1: write('interacting rectangular');
2: write('non-interacting velocity');
3: write('constant');
4: write('step');
5: write('self tune');
end;
write(' Ki: ',I.PID.Ki:2:2,
111
Kp: ',I.PID.Kp:2:2,' Td: ',I.PID.Td:2:2);
gotoxy(5,17); write('New parameters: ');
  if I.Newparam = 1 then write('yes') else
    write('no');
gotoxy(5,18); write('Run upsets: ');
  if I.Upsets.run = 1 then write('yes') else
    write('no');
  write(' Valve upset: ');
  if I.Upsets.double = 1 then write('yes') else
    write('no');
gotoxy(5,19); write('Damperon: ',I.Upsets.Damperon,
    ' Damperoff: ',I.Upsets.Damperoff);
gotoxy(5,20); write('Heaters on: ',I.Upsets.Heaterson,
    ' Heaters off: ',I.Upsets.Heatersoff);
gotoxy(5,21); write('Cold wtr on: ',I.Upsets.Coldwateron,
    ' Cold wtr off: ',I.Upsets.Coldwateroff);
gotoxy(5,22); write('Setpoint up: ',I.Upsets.Setpointon,
    ' Setpoint down: ',I.Upsets.Setpointoff,
    ' Size: ',I.Upsets.Setpointsize:2:2,' degrees');
gotoxy(5,23); write('Wtr flow high: ',I.Upsets.Wtrflowhigh,
    ' Wtr flow low: ',I.Upsets.Wtrflowlow);
gotoxy(5,25); write('Enter 0 if this is incorrect(1) - ');

while keypressed do empty := readkey; (empty keyboard buffer)
  (if any information is incorrect the user can start over)
readln(Z); if Z <> '' then
  begin
    val(Z,Z_val,code); (convert string to integer)
    stop := Z_val;
  end
  else stop := 1; (passed to PIDSIMM)
  if stop = 0 then exit; (do not run rest of unit)
if I.PID.alg = 0 then d16_aous(0,0,1050,err_code);
end; (initialization)
(run every pass)
if initialflag = 0 then begin
  if K < 5 then dot_out := 0; piol2_(dot_out); (turn all upsets off)
  if (K * 3 > I.Upsets.Damperon - 4) and
    (K * 3 < I.Upsets.Damperon + 4) then begin (check time for upset)
    dot_out := dot_out or $08; piol2_(dot_out); (send signal for upset to PIO12_ procedure)
    writeln('closing damper');
    If I.Upsets.Double = 1 then begin (adjust waterf low if asked for)
      aot_count_real := aot_count_real + Dubup;
      CO := round(aot_count_real);
      d16_aous(0,0,CO,err_code); (output to E/P)
end;
end;

if (K * 3 > I.Upsets.Damperoff - 4) and
   (K * 3 < I.Upsets.Damperoff + 4) then begin
   dot_out := dot_out and $F7; piol2_(dot_out);
   writeln('opening damper');
   if I.Upsets.Double = 1 then begin
      aot_count_real := aot_count_real - Dubup;
      CO := round(aot_count_real);
      d16_aous(0,0,CO,err_code);
   end;
end;

if (K * 3 > I.Upsets.Heaterson - 4) and
   (K * 3 < I.Upsets.Heaterson + 4) then begin
   dot_out := dot_out or $06; piol2_(dot_out);
   writeln('Turning Heaters On');
   if I.Upsets.Double = 1 then begin
      aot_count_real := aot_count_real + Dubup;
      CO := round(aot_count_real);
      d16_aous(0,0,CO,err_code); {output to E/P}
   end;
end;

if (K * 3 > I.Upsets.Heatersoff - 4) and
   (K * 3 < I.Upsets.Heatersoff + 4) then begin
   dot_out := dot_out or $01; piol2_(dot_out);
   writeln('Turning Cold Water On');
   if I.Upsets.Double = 1 then begin
      aot_count_real := aot_count_real - Dubup;
      CO := round(aot_count_real);
      d16_aous(0,0,CO,err_code); {output to E/P}
   end;
end;

if (K * 3 > I.Upsets.Coldwateron - 4) and
   (K * 3 < I.Upsets.Coldwateron + 4) then begin
   dot_out := dot_out or $01; piol2_(dot_out);
   writeln('Turning Cold Water On');
   if I.Upsets.Double = 1 then begin
      aot_count_real := aot_count_real - Dubup;
      CO := round(aot_count_real);
      d16_aous(0,0,CO,err_code); {output to E/P}
   end;
end;

if (K * 3 > I.Upsets.Coldwateroff - 4) and
   (K * 3 < I.Upsets.Coldwateroff + 4) then begin
dot_out := dot_out and $FE; piol2_(dot_out);
writeln('Turning Cold Water Off');
If I.Upsets.Double = 1 then begin
  aot_count_real := aot_count_real + dubup;
  CO := round(aot_count_real);
  d16_aous(0,0,CO,err_code); (output to E/P)
end;
end;
end;
if (K * 3 > I.Upsets.Setpointon - 4) and
  (K * 3 < I.Upsets.Setpointon + 4) then begin
I.Setpoint := I.Setpoint + I.Upsets.Setpointsize;
writeln('Setpoint increased by 5');
end;
if (K * 3 > I.Upsets.Setpointoff - 4) and
  (K * 3 < I.Upsets.Setpointoff + 4) then begin
writeln('Setpoint decrease by 5');
end;
end;
end;

procedure calc_pid; (of data_from_facility)
var
  scan_min: real; (scans per minute)
  PID_calc: real; (solution to algorithm)
  aot_count: integer; (control signal in bytes)
begin
  scan_min := 5/60;
  old_old_error := old_error; (save setpoint – measured outlet
  air temperature at time = 2)
  old_error := new_error; (save setpoint – measured outlet air
  temperature at time = 1)
  If (CO < 1496) and (CO > 610) then     (anti-windup)
    sum_error := sum_error + old_error;
  (anti-windup, actual range 1400 to 3400 for a 0-5V 12 bit
  A/D)
  new_error := -I.Setpoint + Data[K,2]; (reverse acting)
  Case I.PID.alg of

  1: begin
    pid_calc := new_error * I.PID.Kp * (1 + I.PID.Td / scan_min) –
                   (old_error * I.PID.Kp * I.PID.Td / scan_min) +
                   (sum_error * I.PID.Ki * I.PID.Kp * scan_min);
    aot_count_real := 4095 * pid_calc/5; (control signal in

  114
2: begin
    pid_calc := I.PID.Kp*(new_error - old_error) +
                I.PID.Ki*(5) * new_error +
                (I.PID.Td/(5)) *
               (new_error - 2*(old_error) + old_old_error);
    aot_count_real := 4095 * PID_calc/5;
    aot_count_real_old := aot_count_real;
end;

3: aot_count_real := 1350.0;

4: begin
    if (K * 3 > I.Upsets.Wtrflowhigh - 4) and
       (K * 3 < I.Upsets.Wtrflowlow - 4) then aot_count_real := 1600;
    if (K * 3 > I.Upsets.Wtrflowlow - 4) then aot_count_real := 200;
end;

5: begin
    pid_calc := Kc*new_error;
    aot_count_real := PID_calc;
    if (aot_count_real > 610) and (aot_count_real < 1496) then
        aot_count_real := aot_count_real_old + aot_count_real;
    if pid5flag = 1 then
        aot_count_real := 1150;
        aot_count_real_old := aot_count_real;
end;
end; {Case}

Data[K,7] := aot_count_real;          (0 - 100%)
if aot_count_real >= 1496 then aot_count_real := 1496
                               {valve never closes}
                               {min MdotW = 1.3gpm}
else
    if aot_count_real < 610 then aot_count_real := 610;
    aot_count := round(aot_count_real);  { declared in this
procedure }
    CO := aot_count;                    {CO declared globally}             (0 - 4095)
end;

procedure PID;  {of data_from_facility}
const max_buffer = 1000;
var last_volts, volts: array[0..1] of real;
lsv,hsv,llim,hlim,up: real;
valout : integer;
err_code,board_num,chanlo: integer;
begin
if K >= 2 then
begin
    New_error := -(I.Setpoint - Data[K,2]);
    Old_error := -(I.Setpoint - Data[K-1,2]);
    Old_old_error := -(I.Setpoint - Data[K-2,2]);
    lsv := 0; hsv := 5; llim := 0; hlim := 5;
    board_num := 0; chanlo := 0;
    up := 5;
    calc_pid;
    d16_aous(board_num,chanlo,CO,err_code); (output to E/P)
end
else
    Data[K,7] := 0;
end; (procedure PID)

procedure get_data;  {of data from facility}
var Temp: array[0..3] of real;  {temporary data storage}
    0: Inlet air temperature
    1: Outlet air temperature
    2: Inlet water temperature
    3: Outlet water temperature
J: integer;  {counter}
GPMin, FPMin, FPMtemp, V: real;  {temporary data storage}
cjc_bin: integer;  {dash16 requirments}
dataval_ains: integer;

begin
    init_boards;
d16ains_(7,dataval_ains);  {get cold junction temperature}
cjc_bin := dataval_ains;
for K := 0 to I.Datapoints do begin
    delay(3000);
    if I.Upsets.Run = 1 then upsets(Filename,stop,0);
    for J := 0 to 3 do begin  {get temperatures}
        d16bous_(J);  {set exp16 channel}
        d16ains_(0,dataval_ains);  {read data}
        Temp[J] := exp16_tc_lin(dataval_ains,1000,cjc_bin,'T');
        {linearize thermocouple data}
    end;
    Temp[0] := Temp[0] - 0.182;
    Temp[2] := Temp[2] + 0.296;  {calculate for offsets}
    {calaculated from tests of 1/24/89}
    d16bous_(4);  {set exp16 channel to read water flow}
    d16ains_(0,dataval_ains);
    if (dataval_ains * (50/2048)) > 48.22 then GPMin := 5.40
    else if CO > 1496 then GPMin := 0.00
else GPMin := 0.01267 + (0.11173 * (dataval_ains * (50/2048)));
FPMtemp := 0;
for J:= 1 to 4 do begin {average air flow to reduce noise}
  if I.transducer = 1 then d16ains_3 (dataval_ains)
  else d16ains_5 (dataval_ains);
  V := dataval_ains * 5 / 2048;
  if I.Transducer = 2 then { Hot Wire Anemometer }
    FPM := -16.9066 + 259.0772 * V - 167.9884 * V * V + 250.9507 * V * V * V;
  if I.transducer = 1 then { Pitot-Tube }
    FPM := 15.746*sqrt(((Temp[1] * 9/5) + 492) * abs(V));
  if I.Transducer = 0 then { Venturi }
    FPM := 14.262*sqrt(((Temp[1] * 9/5) + 492) * abs(V));
FPMtemp := FPMtemp + FPM;
end;
FPM := FPMtemp/4;
Data[K,1] := Temp[0];
Data[K,2] := Temp[1];
Data[K,3] := Temp[2];
Data[K,4] := Temp[3];
Data[K,5] := GPMin; if Data[K,5] < 0 then Data[K,5] := 0;
PID;
writeln((K*5):2,' ',',Data[K,1]:2:2, ',',Data[K,2]:2:2,
  ',',Data[K,3]:2:2, ',',Data[K,4]:2:2,
  ',',Data[K,5]:2:2,
  ',',Data[K,6]:2:2, ',',Data[K,7]:2:3(',
  ',',Data[K,8]:2:2,
  ',',Data[K,9]:2:2));
end;
end; (procedure get_data)

procedure writedata;
var Writefile: file of real;
begin
  write('Writing to '+Filename);
  assign(writefile,Filename);
  rewrite(writefile);
  for K := 0 to I.Datapoints do begin
    write(writefile,Data[K,1],Data[K,2],Data[K,3],Data[K,4],
      Data[K,5],Data[K,6],Data[K,7],Data[K,8],Data[K,9]));
    if (K/40 - INT(K/40) < 0.10) and (K/40 - INT(K/40) > -0.10) then write('.');
  end;
  close(writefile);
  writeln;
end; (procedure write_file)

procedure Steady_state(condition: integer);
var Datapoints_stored: integer; \{temporary storage\}
Runupsets_stored: integer; \{temporary storage\}
Min_air_temp, Max_air_temp: real;

begin
Datapoints_stored := I.Datapoints;
Runupsets_stored := I.Upsets.Run;
I.Upsets.Run := 0; \{do not run any upsets\}
I.Datapoints := 24; \{steady state for 24 counts\}
repeat
get_data;
Min_air_temp := Data[0,2]; \{initialize\}
Max_air_temp := Data[0,2];
for K := 1 to I.Datapoints do begin
  if Data[K,2] > Max_air_temp then Max_air_temp := Data[K,2];
  if Data[K,2] < Min_air_temp then Min_air_temp := Data[K,2];
end;
writeln('Temp difference = ',ABS(Min_air_temp - Max_air_temp):2:2);
until ABS(Min_air_temp - Max_air_temp) < condition;
Filename := '\thesis\data\'+I.Filename+'\ss';
writedata(o);
I.Datapoints := Datapoints_stored;
I.Upsets.Run := Runupsets_stored;
end;

procedure Integral_Only_Tuning(var Ks,Kr,Kc:real);
\{PID alg 5\}
const numobs = 50;
numcol = 1;
Var
dataset : recmat;
dataset2 : array[0..500] of real;
Noise_Tao : real;
err_code : integer;
Nloop,Td : integer;
minima,maxima,range,sumxx,mean,varience,stddev,semean,mode:
ShortVector;
cjc_bin: integer;
dataval_ains,dataval_bous: integer;
Counter : integer;
Min_air_temp,Max_air_temp: real;
Tao_fs,Tao_ss: real;
Tc,Tc_calc,Tc_low,Tc_high: real;
S: real;
X :char;
steadystatepass: integer;
statfile: text;
procedure Small_data(writetofile : integer);
begin
init_boards;
d16ains_(7,dataval_ains);
cjc_bin := dataval_ains;
For K := 0 to (Numobs-1) do begin
delay(500);
d16bous_(1);
d16ains_(0,dataval_ains);

dataset [ K , 0 ] :=
exp16_tc_lin(dataval_ains,tc_bin_in,1000,cjc_bin,'T');
dataset2[steadystatepass * 50 + K] := dataset[K,0];
writeln('(',K:2,')',dataset[K,0]:5:2);
if writetofile = 1 then
writeln(statfile,'(',K:3,')',dataset[K,0]:5:2);
end;
end; {procedure Small_Data}

Begin
assign(statfile, '\thesis\data\sumstat.dat');
d16_aous(0,0,1150,err_code);
pid5flag := 1;
steady_state(1);
pid5flag := 0;
steadystatepass := 0;
rewrite(statfile);
repeat
small_data(1);
Min_air_temp := Dataset[0,0];
Max_air_temp := Dataset[0,0];
for K := 0 to (numobs-1) do begin
if Dataset[K,0] > Max_air_temp then Max_air_temp :=
Dataset[K,0];
if Dataset[K,0] < Min_air_temp then Min_air_temp :=
Dataset[K,0];
end;
writeln('Pass ',steadystatepass:2,
'Temp difference = ',ABS(Min_air_temp -
Max_air_temp):2:2);
steadystatepass := steadystatepass + 1;
until ABS(Min_air_temp - Max_air_temp) < 0.5;

Tao_ss := (Min_air_temp + max_air_temp)/2;

SummaryStats(dataset,numobs,numcol,minima,maxima,range,sumxx,mean
, variance,
stddev,semean,mode);
Noise_Tao := 3.0*stddev[0];
d16_aous(0,0,CO-200,err_code); {Step output to E/P}
steadystatepass := 0;
repeat
small_data(1);
Min_air_temp := Dataset[0,0];
Max_air_temp := Dataset[0,0];
for counter := 1 to (numobs-1) do
begin
  if Dataset[counter,0] > Max_air_temp then Max_air_temp := Dataset[counter,0];
  if Dataset[counter,0] < Min_air_temp then Min_air_temp := Dataset[counter,0];
end;
writeln('Pass ',steadystatepass:2,
  ' Temp difference = ',ABS(Min_air_temp - Max_air_temp):2:2);
steadystatepass := steadystatepass + 1;
until ABS(Min_air_temp - Max_air_temp) < 0.5;
Td := 0;
Repeat
  Td := Td + 1;
Until (dataset2[Td] - mean[0] > Noise_Tao*1.25)
or (Td > steadystatepass * 50);
Tao_fs := (Min_air_temp + max_air_temp)/2;

Ks := (Tao_fs - Tao_ss)/200;
Tc_calc := Tao_ss + (Tao_fs - Tao_ss) * 0.63;
K := 0;
repeat
  if (K > 0) and (K < steadystatepass * 50) then begin
    Tc_low := Dataset2[K-1];
    Tc_high := Dataset2[K];
  end;
  K := K + 1;
until (Tc_calc > Tc_low) and (Tc_calc < Tc_high);
writeln;
Tc := K - 1.5;

S := ((Td/Tc)*(Td/Tc) + 4 -(Td/Tc + 2))/(2*Td);
Kr := EXP(Td*S) * ((1/Tc) + S) * S;
Kc := Kr * (Tc/Ks);
writeln(minima[0]:2:2,' ',maxima[0]:2:2,' ',range[0]:2:2,' ',
  sumxx[0]:2:2,' ',mean[0]:2:2,' ',varience[0]:2:2,' ',
  stddev[0]:2:2,' ',smean[0]:2:2);
writeln(Tao_fs:2:2,' ',Tao_ss:2:2,' ',Td:2,' ',Tc:2:2,' ',
  noise_tao:2:5,' ',S:2:2,' ',Ks:2:5,' ',Kc:2:5);
writeln(statfile,minima[0]:2:2,' ',maxima[0]:2:2,' ',
  range[0]:2:2,' ',
  sumxx[0]:2:2,' ',mean[0]:2:2,' ',varience[0]:2:2,' ',
  stddev[0]:2:2,' ',smean[0]:2:2);
writeln(statfile,Tao_fs:2:2,' ',Tao_ss:2:2,' ',Td:2,' ',
  Tc:2:2,' ',

120
noise_tao:2:5,' ',S:2:2,' ',Ks:2:5,' ',Kr:2:5,' ',Kc:2:5);

close(statfile);
d16_aous(0,0,1150,err_code);
end; (procedure Integral_Only_Tuning)

begin  (procedure data_from_facility)
  window(1,1,80,25);
aot_count_real := 1050;
aot_count_real_old := aot_count_real;
K := 0;
Init_boards;
sum_error := 0;
upsets(Filename,stop, l);
if stop <> 0 then
  begin
    clrscr;
    writeln;
    if I.PID.alg = 5 then begin
      writeln('Calculating PID values');
      integral_only_tuning(Ks,Kr,Kc);
    end;
    if I.Steady_state = 1 then
      begin
        writeln('Waiting for steady state');
        steady_state(l);
      end;
    writeln;
    writeln('Data from facility');
    get_data;
    Filename := 'thesis\data\'+I.Filename+'.dat';
    writedata(0);
  end;
end;  (procedure data_from_facility)

End.
UNIT Fitmodel;

Interface
Uses Dos, Crt, StdHdr, MatMath;
procedure Findparameters(Filename: string);

type Parameters = record
  date: string;
  a: real; b: real; c: real; d: real;
  e: real; f: real; g: real; h: real; j: real;
end;

type Information = record
  Filename: string;
  Datatype: integer;
  Datapoints: integer;
  Textfile: integer;
  Transducer: integer;
  PID: record
    alg: integer; Ki: real; Kp: real; Td: real; end;
  Newparam: integer;
  Setpoint: real;
  Steadystate: integer;
  Upsets: record
    Run: integer;
    Double: integer;
    Damperon: integer;
    Damperoff: integer;
    Heaterson: integer;
    Heatersoff: integer;
    Coldwateron: integer;
    Coldwateroff: integer;
    Setpointon: integer;
    Setpointoff: integer;
    Setpointsize: real;
    Wtrflowhigh: integer;
    Wtrflowlow: integer;
  end;
  Paramselect: integer;
end;

var I: information;
P: Parameters;
Infofile: file of Information;
Implementation

procedure findfile(filename: string; var fileexists: integer);
var K : integer;
   fileinfo : searchrec;
   lastfile : string;
begin
   fileexists := 1;
   findfirst('THESIS\DATA\*.\*,anyfile,fileinfo);
   if fileinfo.name <> filename then begin
      repeat
         findnext(fileinfo);
         if lastfile = fileinfo.name then
            fileexists := 0;
         lastfile := fileinfo.name;
      until (fileinfo.name = filename) or (fileexists = 0);
   end;
end;

Procedure Param;
var
   readfile, write_file: file of real;
   Air_temp_in,Air_temp_out,Water_temp_in,Water_temp_out, Water_flow,Air_flow,Control_signal, WaterSS, AirSS,
   Air_temp_in_1,Air_temp_out_1,Water_temp_in_1,Water_temp_out_1, Water_flow_1,Air_flow_1: real;
   K: integer;
   Twbar, DTA, Dtw, A, B, C, D, E, F, G, H, J: real;
   Filename: string;

Begin
   Filename := '\thesis\data\'+I.Filename+'\'.dat';
   Writeln('Reading from '+I.Filename+'\');
   assign(readfile,Filename);
   Filename := '\thesis\data\'+I.Filename+'\'.par';
   Writeln('Writing to '+Filename);
   assign(write_file,Filename);
   reset(readfile);
   rewrite(write_file);
   K := 0;
   repeat
      read(readfile,Air_temp_in,Air_temp_out,Water_temp_in,
         Water_temp_out, Water_flow,Air_flow,Control_signal{,WaterSS,AirSS});
      K := K + 1;
      if K > 1 then ...
Twbar := (Water_temp_in_1 + Water_temp_out_1)/2;
DTw := (Water_temp_out - Water_temp_out_1);
A := Water_flow_1 * 0.06296382 * (Water_temp_in_1 - Water_temp_out_1);
B := Air_temp_in_1 - Twbar;
C := Water_flow_1 * 0.06296382 * (Air_temp_in_1 - Twbar);
D := Air_flow_1 * 0.001221041 * (Air_temp_in_1 - Twbar);
Twbar := (Water_temp_in_1 + Water_temp_out_1)/2;
DTA := -(Air_temp_out_1 - Air_temp_out);
E := Air_flow_1 * 0.001221041 * (Air_temp_in_1 - Air_temp_out_1);
F := Twbar - Air_temp_in_1;
G := Water_flow_1 * 0.06296382 * (Twbar - Air_temp_in_1);
H := Air_flow_1 * 0.001221041 * (Twbar - Air_temp_in_1);
J := Air_temp_in - Air_temp_in_1;

if (K/40 - INT(K/40) > -0.10) and (K/40 - INT(K/40) < 0.10) then
  write('.');
  write(write_file,Dtw,A,B,C,D,DTA,E,F,G,H,J);
end;
Air_temp_in_1 := Air_temp_in;
Air_temp_out_1 := Air_temp_out;
Water_temp_in_1 := Water_temp_in;
Water_temp_out_1 := Water_temp_out;
Water_flow_1 := Water_flow;
Air_flow_1 := Air_flow;
until EOF(readfile);
I.Datapoints := K;
close(readfile);
close(write_file);
writeln;
end;

procedure Findparameters;
type
  square4x4 = Array[0..3,0..3] of real;
  Vector = Array[0..4] of real;
var
  A, Ainv, AA, AAinv, B, BB, C, CC : recmat;
  X, XX : Vector;
  Count, M : integer;
  shit, Y, YY, Det, RSSW, RSA : Real;
  K, N, H : Integer;
  DataFile, DataOutFile : file of real;
  Inlable, Outlable : string[10];
  Fileanal : string;
  FirstLine : string[50];
readfile : text;
Infofilename : string;
Z : char;
fileexists : integer;

Begin
  window(1,1,80,25);
  gotoxy(1,24);
  findfileCfilename+.IN2',fileexists);
  if fileexists = 1 then begin
    assign(Infofile,'\thesis\data\'+Filename+'.in2');
  end
  else begin
    findfile(filename+'.IN1',fileexists);
    if fileexists = 1 then begin
      assign(infofile,'\thesis\data\'+filename+'.ini');
    end
    else begin
      assign(infofile,'\thesis\data\'+filename+'.inf');
    end;
  end;
  reset(Infofile);
  read(Infofile,I);
  close(Infofile);
  FillChar(A,Sizeof(A),0);
  FillChar(AA,Sizeof(AA),0);
  FillChar(B,Sizeof(B),0);
  FillChar(BB,Sizeof(BB),0);
  FillChar(C,Sizeof(C),0);
  FillChar(CC,Sizeof(CC),0);
  FillChar(X,Sizeof(X),0);
  FillChar(XX,Sizeof(XX),0);
  FillChar(Ainv,Sizeof(Ainv),0);
  FillChar(AAinv,Sizeof(AAinv),0);
  Count := 0;
  Fileanal := '\thesis\data\'+I.Filename+'.par';
  writeln('Reading from '+Fileanal);
  Assign(DataFile,Fileanal);
  Reset(DataFile);

  Read(DataFile,Y,X[0],X[1],X[2],X[3],YY,XX[0],XX[1],XX[2],XX[3],XX[4]);
  (dont use first data points, sometime get bad readings)
  N := I.Datapoints - 1;
  For M := 2 to N Do
    begin
      Read(DataFile,Y,X[0],X[1],X[2],X[3],YY,XX[0],XX[1],XX[2],XX[3],XX[4]);
      if (M/40 - INT(M/40) > -0.10) and (M/40 - INT(M/40) < 0.10) then write('.');
For $K := 0$ to 3 do begin  
  For $H := 0$ to 3 do begin  
  end;  
  $C[K,0] := C[K,0] + (X[K] \times Y);$
end;

for $K := 0$ to 4 do begin  
  for $H := 0$ to 4 do begin  
  end;  
  $C[K,0] := C[K,0] + (XX[K] \times YY);$
end;

end;  
{X'X
$C[K,0] := C[K,0] + (X[K] \times Y);$

end;

end;  
{X'Y

for $K := 0$ to 4 do begin
  for $H := 0$ to 4 do begin
    $AA[K,H] := AA[K,H] + (XX[K] \times XX[H]);$
  end;
  $CC[K,0] := CC[K,0] + (XX[K] \times YY);$
end;

end;  
{X'Y

Close(DataFile);
writein;
MatInvert(A,4,\text{det},Ainv);
MatInvert(AA,5,\text{det},AAinv);
MatProd(Ainv,C,4,4,4,B);
MatProd(AAinv,CC,5,5,5,BB);
WriteIn('A B C D');
WriteIn(B[0,0]:2:6, ' ',B[1,0]:2:6, ' ',B[2,0]:2:6, ' ',B[3,0]:2:6);
writein;
writeIn('E F G H J');
WriteIn(BB[0,0]:2:6, ' ',BB[1,0]:2:6, ' ',BB[2,0]:2:6, ' ',BB[3,0]:2:6, ' ',BB[4,0]:2:6);
assign(Paramfile,'\thesis\data\Paramf.dat');
reset(Paramfile);
Numberofrecords := filesize(Paramfile);
for $K := 0$ to Numberofrecords - 1 do begin
  read(Paramfile,P);
  if P.date = I.Filename then begin
    writeln('Parameter set named ',I.Filename,' already exists');
    writeln(' choose a different name,');
    writeln(' or enter 0 to ignore - ');
    readln(I.Filename);
  end;
end;
P.Date := I.Filename;
P.a := B[0,0]; P.b := B[1,0]; P.c := B[2,0]; P.d := B[3,0];
P.e := BB[0,0]; P.f := BB[1,0];
P.g := BB[2,0]; P.h := BB[3,0]; P.j := BB[4,0];
seek(Paramfile,Numberofrecords);
if I.Filename <> '0' then write(Paramfile,P);
close(Paramfile);
End.
"FROMFILE.TPU"

{R+} (Range checking off)
{B+} {Boolean complete evaluation on}
{S+} (Stack checking on)
{I+} (I/O checking on)
{M 65500,16384,655360} (Turbo 3 default stack and heap)

UNIT Fromfile;

Interface
Uses dos,crt,pictures;

type Dataarray = array[0..820,1..7] of real; {maximum 800 datapoints:
1: Inlet air temperature
2: Outlet air temperature
3: Inlet water temperature
4: Outlet water temperature
5: Water flow
6: Air flow
7: Control signal}

type Parameters = record {model parameters}
  date: string;
  a: real; b: real; c: real; d: real; e: real;
  f: real; g: real; h: real; j: real;
end;

type Information = record {miscellaneous data}
  Filename: string;
  Datatype: integer; {0: facility, 1: from file}
  Datapoints: integer; {number of datapoints}
  Textfile: integer; {write to text file ? 0: no, 1: yes}
  Transducer: integer; {0: venturi, 1: pitot tube, 2: hot wire anemometer}
  PID: record {PID algorithm, Ki, Kp, Td}
    alg:integer; Ki: real; Kp: real; Td: real; end;
    {PID alg 0: open loop
    1: interacting rectangular
    2: non-interacting velocity
    3: constant
    4: step
    5: self tune}
  Newparam: integer; {calculate new model parameters ? 0: no, 1: yes}
  Setpoint: real;
  Steadystate: integer; {find steady state before beginning test? 0: no, 1: yes}
  Upsets: record
    Run: integer; {run upsets ? 0: no, 1: yes}

128
Double: integer; (adjust water flow after upset)
Damperon: integer; (close damper)
Damperoff: integer; (open damper)
Heaterson: integer; (turn 15kw heaters on)
Heatersoff: integer; (turn 15kw heaters off)
Coldwateron: integer; (add cold water to system)
Coldwateroff: integer; (turn cold water off)
Setpointon: integer; (raise setpoint)
Setpointoff: integer; (lower setpoint)
Setpointsize: real; (size of setpoint error)
Wtrflowhigh: integer; (in PID alg 4, increase water flow)
Wtrflowlow: integer; (in PID alg 4, decrease water flow)
end;

Paramselect: integer; (parameter set used in test)
Error_model_measured: real; (average error between measured outlet air temperature and calculated outlet air temperature)
Error_setpoint_measured: real; (average error between setpoint and measured outlet air temperature)
Error_setpoint_model: real; (average error between setpoint and calculated outlet air temperature)
end;

procedure Datafromfile(Filename:string; var Stop: integer);

Const max_buffer = 1000;
GPM_to_Kgps = 0.06296382; (conversion for water flow, gallons per minute to kilograms per second)
FPM_to_kgps = 0.001221041; (conversion for air flow, feet per second to kilograms per second)

var P: Parameters;
I: Information;
K,J: integer; (counters)
Sum_error: real; (setpoint - measured outlet air temperature)
Kc,Kr,Ks: real; (PID constants from PID alg 5)
Filename: string;
CO, CO_1: real; (control signal, last control signal)
New_error, Old_error,
Old_old_error: real; (setpoint - measured outlet air temperature,

Implementation 129
function XtoY(X,Y:real):real; {raise x to a power y}
var  Z: real;
begin
  if X = 0 then XtoY := 0
  else begin
    Z := Y * ln(X);
    XtoY := exp(Z);
  end;
end;

procedure findfile(filename: string; var fileexists: integer);
{uses "findfirst" and "findnext" to determine if file exists}
var  K : integer;
    fileinfo : searchrec;
    lastfile : string;
begin
  fileexists := 1; {set boolean to affirmative}
  findfirst('\THESIS\DATA\*.*',anyfile,fileinfo);
  if fileinfo.name <> filename then begin {if first file in
    directory is not
      looked for file,}
    continue
    repeat
      findnext(fileinfo); {check each file for filename}
      if lastfile = fileinfo.name then
        fileexists := 0; {if file found set boolean to negative}
        lastfile := fileinfo.name;
      until (fileinfo.name = filename) or (fileexists = 0);
    end;
  end; {findfile}

procedure equals(Upsettime,Upsettimeoff: integer;
      var equal:boolean; var Upseton,Upsetoff:
      integer);
  {is time within +- delay of upset time}
begin
  if (K * delay > Upsettime - (delay + 1))
      and (K * delay < Upsettime + (delay - 1)) then
    equal := true
  else
    equal := false;
  Upseton := ROUND(Upsettime/delay);
  Upsetoff := ROUND(Upsettimeoff/delay);
end;
procedure Datafromfile;  {this procedure does everything}
    var  Data : Data_array;
         mv : real;

procedure PID;
    const max_buffer = 1000;
    var last_vols, volts: array[0..1] of real;
       lsv,hsv,llim,hlim,up: real;
       err_code: integer;

procedure calc_pid;
    const alpha1 = 0.0165; alpha2 = -0.1755; alpha3 = 0.5757;
    alpha4 = -0.1778;
    var aot_count_real, aot_count_real_old, PID_calc: real;
       aot_count: integer; {control signal in bytes}
       Mdotwbar : real;
       Mwcount : byte;
       alpha1, alpha2, alpha3, alpha4 : real;
       Numwavg : integer;
       Kpmult : real;

begin
    if (CO < 1496) and (CO > 610) then {anti-windup}
        sum_error := sum_error + old_error;  {deg C}
    Case I.PID.alg of
        1: begin  { degC   * Kp(5V/degC) }
            pid_calc := 610 + (I.PID.Kp*409.5*(Data[K,2] - I.Setpoint));
            aot_count_real := pid_calc;
            end;
        2: begin
            Numwavg := round(I.PID.Ki);
            Kpmult := I.PID.Td;
            Mdotwbar := 0;
            If K > numwavg then
                begin
                    For mwcount := 1 to numwavg do mdotwbar := mdotwbar
                    + (Data[K-mwcount,5]);
                    mdotwbar := mdotwbar/numwavg;
                end
            Else Mdotwbar := Data[K-1,5];
            I.PID.Kp := (alpha1*mdotwbar*mdotwbar*mdotwbar)
            + (alpha2*mdotwbar*mdotwbar)
            + (alpha3*mdotwbar)
            131
\[ I.PID.Kp := I.PID.Kp \times Kpmult; \]
\[ \text{writeln}(I.PID.Kp:2:2); \]
\[ \text{pid}_\text{calc} := 610 + (409.5 \times I.PID.Kp \times \text{new}\_\text{error}); \]
\[ \text{aot}\_\text{count}\_\text{real} := \text{pid}\_\text{calc}; \]
\[ \text{end}; \]

3: \ (constant\ output) \n\[ \text{aot}\_\text{count}\_\text{real} := 1350.0; \]
(PID alg 4 is in the upset procedure)

5:
\[ \text{begin} \]
\[ \text{pid}\_\text{calc} := \text{Kc}\times\text{new}\_\text{error}; \]
\[ \text{aot}\_\text{count}\_\text{real} := \text{PID}_\text{calc}; \]
\[ \text{if } \{\text{aot}\_\text{count}\_\text{real} > 610}\ \text{and}\ \{\text{aot}\_\text{count}\_\text{real} < 1496\} \]
\[ \text{then} \]
\[ \text{aot}\_\text{count}\_\text{real} := \text{aot}\_\text{count}\_\text{real}\_\text{old} + \text{aot}\_\text{count}\_\text{real}; \]
\[ \{\text{declared in this procedure}\} \]
\[ \text{aot}\_\text{count}\_\text{real}\_\text{old} := \text{aot}\_\text{count}\_\text{real}; \]
\[ \text{end}; \]
\[ \text{end}; \ (\text{Case}) \]

if \{\text{aot}\_\text{count}\_\text{real} > 4095\} \text{or} \{\text{aot}\_\text{count}\_\text{real} < 0\} \text{then} \begin{align*}
\text{if } \{\text{aot}\_\text{count}\_\text{real} > 4095\} \text{then } \text{aot}\_\text{count}\_\text{real} := 4095; \\
\text{if } \{\text{aot}\_\text{count}\_\text{real} < 0\} \text{then } \text{aot}\_\text{count}\_\text{real} := 0; \\
\text{if } \{\text{aot}\_\text{count}\_\text{real}\_\text{old} > 4095\} \text{then } \text{aot}\_\text{count}\_\text{real}\_\text{old} := 4095; \\
\text{if } \{\text{aot}\_\text{count}\_\text{real}\_\text{old} < 0\} \text{then } \text{aot}\_\text{count}\_\text{real}\_\text{old} := 0; \\
\text{end}; \\
\text{aot}\_\text{count} := \text{round}(\text{aot}\_\text{count}\_\text{real}); \quad \{\text{uses PID calculation, not CO of actual test}\} \\
\text{end}; \]

begin
\[ \text{New}\_\text{error} := -(\text{I.Setpoint} - \text{Data}[K,2]); \]
\[ \text{Old}\_\text{error} := -(\text{I.Setpoint} - \text{Data}[K-1,2]); \]
\[ \text{Old}\_\text{old}\_\text{error} := -(\text{I.Setpoint} - \text{Data}[K-2,2]); \]
\[ \text{lsv} := 0; \quad \text{hsv} := 5; \quad \text{llim} := 0; \quad \text{hlim} := 5; \]
\[ \text{up} := 5; \]
\[ \text{calc}\_\text{pid}; \]
\[ \text{If } \text{CO}_1 < 660 \text{then } \text{CO}_1 := 660; \]
\[ \text{If } \text{CO}_1 > 1440 \text{then } \text{CO}_1 := 1440; \]
\[ \text{mv} := (-4.1290589864E+01) + ((3.0932439193E-01) \times \text{CO}_1) + \]
\[ (-3.2681341813E-04) \times \text{XtoY}(	ext{CO}_1,2) \]
\[ + ((9.5607316380E-08) \times \text{XtoY}(	ext{CO}_1,3)); \]
\[ \text{Data}[K,5] := 0.01267 + (0.11173 \times \text{mv}); \quad \{\text{gpm}\} \]

132
if Data[K,5] > 5.37 then Data[K,5] := 5.37;
end; (procedure PID)

procedure upsets(Filename:string; J:integer; var stop:
integer);

var Z: string;
Z_val, code: integer;
Upseton, Upsetoff: integer;
Infofilename: string;
Infofile: file of Information;
Paramfile: file of Parameters;
Upsetfile: file of integer;
Datachange: Integer; (size of change in data during
upsets)
M: integer;
empty: char;
equal: boolean;
fileexists: integer;
extension: string;

begin
  if J = 1 then begin (initialize data only on the first pass)
    findfile(filename+'.IN2',fileexists); (determine which
information file to open)
    if fileexists = 1 then begin
      assign(Infofile,'thesis\data\'+Filename+'.in2');
      extension := '.IN2';
    end
    else begin
      findfile(filename+'.IN1',fileexists);
      if fileexists = 1 then begin
        assign(Infofile,'thesis\data\'+Filename+'.in1');
        extension := '.IN1';
      end
      else begin
        assign(Infofile,'thesis\data\'+Filename+'.inf');
        extension := '.INF';
      end;
    end;
    reset(Infofile); (open information file)
    read(Infofile,I);
    close(Infofile);
    assign(Paramfile,'\thesis\data\paramf.dat');
    reset(Paramfile); (open parameters file)
    if I.Newparam = 1 then seek(Paramfilefilesize(Paramfile) -
1)
      else seek(Paramfile,I.Paramselect); (read parameters
selected in pidsimm)
    read(Paramfile,P);
    close(Paramfile);
    {write selections to screen}
The parameters selected for this test are:


if I.Newparam = 0 then
begin
  begin
    case I.Transducer of
      0: write('Venturi');
      1: write('Pitot tube');
      2: write('Hot wire anemometer');
    end;
    gotoxy(5,13); write('Setpoint is: ',I.Setpoint:2:0, ' degrees');
    gotoxy(5,14); write('Steady state: '); if I.Steadystate = 1 then write('yes') else write('no');
    gotoxy(5,15); write('Datatype: '); if I.Datatype = 1 then write('from file') else write('from facility');
    write(' Datapoints: ',I.Datapoints,' Textfile: '); if I.Textfile = 1 then write('yes') else write('no');
  end;
  gotoxy(5,16); write('PID alg: '); if I.PID.alg = 0 then write('open loop');
    case I.PID.alg of
      0: write('open loop');
      1: write('interacting rectangular');
      2: write('non-interacting velocity');
      3: write('constant');
      4: write('step');
      5: write('self tune');
      end;
  end;
end;

if I.Newparam = 1 then write('yes') else write('no');

if I.Upsets.run = 1 then write('yes') else write('no');

if I.Upsets.double = 1 then write('yes') else write('no');

if I.Upsets.Damperon = 1 then write('yes') else write('no');

if I.Upsets.Damperoff = 1 then write('yes') else write('no');
gotoxy(5,20); write('Heaters on: ',I.Upsets.Heaterson, 'Heaters off: ',I.Upsets.Heatersoff);
gotoxy(5,21); write('Cold wtr on: ',I.Upsets.Coldwateron, 'Cold wtr off: ',I.Upset.Coldwateroff);
gotoxy(5,23); write('Wtr flowhigh: ',I.Upsets.Wtrflowhigh, 'Wtr flowlow: ',I.Upsets.Wtrflowlow);
end;
gotoxy(5,25); write('Enter 0 if this is incorrect(1) - '); (allow user to quit and correct inputs)
while keypressed do empty := readkey;
readln(Z); if Z <> '' then
begin
  val(Z,Z_val,code);
  stop := Z_val;
end
else stop := 1;
if stop = 0 then exit;
end; (first pass only)
equals(I.Upsets.Damperon,I.Upsets.Damperoff,equal,Upseton,Upset off);
if equal = true then
begin
  for M := Upseton to Upsetoff do
    Data[M,6] := Data[M,6] - 297; (reduce air flow to simulate closed damper)
end;
equals(I.Upsets.Heaterson,I.Upsets.Heatersoff,equal,Upseton,Upset off);
if equal = true then
begin
  for M := Upseton to Upsetoff + 101 do begin
    read(Upsetfile,Datachange); (read the 101 values for increasing inlet air temperature)
    Data[M,1] := Data[M,1] + Datachange;
  end;
  for M := Upseton + 102 to Upsetoff - 99 do
    Data[M,1] := Data[M,1] + 15; (add constant change between increase and decrease inlet air temperature)
  repeat
    read(Upsetfile,Datachange); (read the 99 values for decreasing inlet air temperature)
    Data[M,1] := Data[M,1] + Datachange;
  M := M + 1;
end;
until EOF(Upsetfile);
close(Upsetfile);
end;
equals(I.Upsets.Coldwateron, I.Upsets.Coldwateroff, equal, Upseton, Upsetoff);
if equal = true then
begin
Upseton := ROUND(I.Upsets.Coldwateron/5);
Upsetoff := ROUND(I.Upsets.Coldwateroff/5);
assign(Upsetfile, '\thesis\data\Coldwtr.ups');
reset(Upsetfile);
for M := Upseton to Upseton + 100 do begin
read(Upsetfile, Datachange); {read the 100 values for decreasing inlet water temperature}
Data[M, 3] := Data[M, 3] + Datachange;
end;
for M := Upseton + 101 to Upsetoff - 100 do
Data[M, 3] := Data[M, 3] - 22; {fill constant change in inlet water temperature between decrease and increase}
repeat
read(Upsetfile, Datachange); {read the 100 values for increasing inlet water temperature}
Data[M, 3] := Data[M, 3] + Datachange;
M := M + 1;
until EOF(Upsetfile);
close(Upsetfile);
end;
equals(I.Upsets.Setpointon, I.Upsets.Setpointoff, equal, Upseton, Upsetoff);
if equal = true then {change setpoint}
I.Setpoint := I.Setpoint + I.Upsets.Setpointsize;
equals(I.Upsets.Setpointoff, I.Upsets.Setpointon, equal, Upseton, Upsetoff);
if equal = true then
end;

procedure Read_datafile;
var Readfile: file of real; {fill data array}
m: integer;
begin
write('Reading from ' + Filename);
assign(readfile, Filename);
reset(readfile);
K := 0;
while ((not EOF(readfile)) and (K<820)) do
begin
read(readfile, Data[K,1], Data[K,2], Data[K,3], Data[K,4], Data[K,5], Data[K,6], Data[K,7]);
if K > 0 then begin {check for and replace bad data}
if (Data[K,5] < 0.01) and (Data[K,2] < 20) then begin

end;
for m := 1 to 7 do Data[K,m] := Data[K-1,m];
end;

K := K + 1;
if (K/40 - INT(K/40) < 0.10) and (K/40 - INT(K/40) > -0.10) then
    write('.');
end;
if (K/40 - INT(K/40) < 0.10) and (K/40 - INT(K/40) > -0.10) then
    write('.');
end;
close(readfile);
I.Datapoints := K - 1;
writeln;
end;

procedure wriedata(Write_to_screen: integer);
var Writefile: file of real;
begin
if Write_to_screen = 0 then begin
    write('Writing to '+Filename);
    assign(writefile,Filename);
    rewrite(writefile);
    for K := 0 to I.Datapoints do
        begin
            write(writefile,Data(K,1),Data(K,2),Data[K,3],Data[K,4],
                 Data[K,5],Data[K,6],Data[K,7],Data[K,8],Data[K,9]);
            if (K/40 - INT(K/40) < 0.10) and (K/40 - INT(K/40) > -0.10) then
                write('.');
        end;
    end
else
    writeln((K*5):2,' ',Data(K,1):2:2,' ',Data[K,2]:2:2,
    ' ',Data[K,3]:2:2,' ',Data[K,4]:2:2,
    ' ',Data[K,5]:2:2,' ',Data[K,6]:2:2,
    ' ',Data[K,7]*4096:2:0,' ',Data[K,8]:2:2,
    ' ',Data[K,9]:2:2);
    if Write_to_screen = 0 then close(writefile);
    writeln;
end;  (procedure write_file)

procedure Wtr_out_calculation(a,b,c,d: real);
var MdotA, MdotW, Wtr_avg: real;
begin
    MdotA := Data[K-1,6]*FPM_to_Kgps;
    MdotW := Data[K-1,5]*GPM_to_Kgps;
    Wtr_avg := (Data[K-1,3] + Data[K-1,4]) / 2;
    Data[K,4] := Data[K-1,4] + (a * MdotW * (Data[K-1,3] - Data[K-1,4])) + (b * (Data[K-1,1] - Wtr_avg)) + (c * MdotW * (Data[K-1,1] - Wtr_avg))
procedure Air_out_calculation(e, f, g, h, j: real);
var MdotA, MdotW, Wtr_avg: real;
begin
  MdotA := Data[K, 6] * FPM_to_Kgps;
  MdotW := Data[K, 5] * GPM_to_Kgps;
  Wtr_avg := (Data[K, 3] + Data[K, 4]) / 2;
  Data[K+1, 2] := Data[K, 2] + (e * MdotA * (Data[K, 1] - Data[K, 2])) + (f * (Wtr_avg - Data[K, 1])) + (g * MdotW * (Wtr_avg - Data[K, 1])) + (h * MdotA * (Wtr_avg - Data[K, 1])) + (j * (Data[K+1, 1] - Data[K, 1]));
end;

procedure make_predictions;
begin
  Filename := '\thesis\data\'+I.Filename+' .prd';
  write ('Calculating');
  for K := 2 to I.Datapoints do
    begin
      if I.Upsets.Run = 1 then upsets(Filename, J, stop);
      if I.PID.alg <> 0 then PID;
      Wtr_out_calculation(P.a, P.b, P.c, P.d);
      Air_out_calculation(P.e, P.f, P.g, P.h, P.j);
      if (K/40 - INT(K/40) < 0.10) and (K/40 - INT(K/40) > -0.10) then
        write ('.');
      end;
  writeln;
  writeln;
  writeln;
end;

begin (procedure Data_from_file)
  J := 1; upsets(Filename, J, stop); J := 0;
  if stop <> 0 then
    begin
      Sum_error := 0;
      Filename := '\thesis\data\'+I.Filename+' .dat';
      read_datafile;
      make_predictions;
    end;
end;
End.
"DRAW.TPU"

{$R+}$  (Range checking off)
{$B+}$  (Boolean complete evaluation on)
{$S+}$  (Stack checking on)
{$I+}$  (I/O checking on)
{$M$ 65500,16384,655360}  (Turbo 3 default stack and heap)

UNIT draw;

Interface
uses dos,crt,gdriver,gkernel;

procedure plotdata(Paramselect: integer; Filename: string);

type Data_array = array[0..820,1..7] of real;  (maximum 800 datapoints:
1: Inlet air temperature
2: Outlet air temperature
3: Inlet water temperature
4: Outlet water temperature
5: Water flow
6: Air flow
7: Control signal)

type Parameters = record  (model parameters)
  date: string;
  a: real; b: real; c: real; d: real; e: real;
  f: real; g: real; h: real; j: real;
end;

type Information = record  (miscellaneous data)
  Filename: string;
  Datatype: integer;  (0: facility, 1: from file)
  Datapoints: integer;  (number of datapoints)
  Textfile: integer;  (write to text file ? 0: no, 1: yes)
  Transducer: integer;  (0: venturi, 1: pitot tube, 2: hot wire anemometer)
  PID: record (PID algorithm, Ki, Kp, Td)
    alg:integer; Ki: real; Kp: real; Td: real; end;
    (PID alg 0: open loop
      1: interacting rectangular
      2: non-interacting velocity
      3: constant
      4: step
      5: self tune)
  Newparam: integer;  (calculate new model parameters ? 0: no, 1: yes)
end;

139
Upsets: record
    Run: integer; (run upsets ? 0: no, 1: yes)
    Double: integer; (adjust water flow after upset)
    Damperon: integer; (close damper)
    Damperoff: integer; (open damper)
    Heaterson: integer; (turn 15kw heaters on)
    Heatersoff: integer; (turn 15kw heaters off)
    Coldwateron: integer; (add cold water to system)
    Coldwateroff: integer; (turn cold water off)
    Setpointon: integer; (raise setpoint)
    Setpointoff: integer; (lower setpoint)
    Setpointsize: real; (size of setpoint error)
    Wtrflowhigh: integer; (in PID alg 4, increase water flow)
    Wtrflowlow: integer; (in PID alg 4, decrease water flow)
end;

Paramselect: integer; (parameter set used in test)

Error_model_measured: real; (average error between measured outlet air temperature and calculated outlet air temperature)

Error_setpoint_measured: real; (average error between setpoint and measured outlet air temperature)

Error_setpoint_model: real; (average error between setpoint and calculated outlet air temperature)
end;

{global declarations}
var I: Information;
P: Parameters;
Infofile: file of Information;
Paramfile: file of Parameters;
Filename: string;
Maxtemp, Mintemp, Maxtime: real; (boudaries for graph)
    K, Time_limit: integer;
Data: Data_array;

Implementation

procedure findfile(filename: string; var fileexists: integer);
{uses "findfirst" and "findnext" to determine if file exists}
var K: integer;
    fileinfo : searchrec;
    lastfile : string;
begin
    fileexists := 1; (set boolean to affirmative)
    findfirst('\\THEESIS\DATA\*.*',anyfile,fileinfo);
    if fileinfo.name <> filename then begin (if first file in directory is not looked for file, continue)
repeat
end;
findnext(fileinfo); (check each file for filename)
if lastfile = fileinfo.name then
    fileexists := 0; (if file found set boolean to negative)
    lastfile := fileinfo.name;
until (fileinfo.name = filename) or (fileexists = 0);
end;
end; {findfile}

procedure totext;
{rewrite data and predicted files declared real to files in ASCII}
type Data_array = array[0..820,1..9] of real; {maximum 800 datapoints:
1: Inlet air temperature
2: Outlet air temperature
3: Inlet water temperature
4: Outlet water temperature
5: Water flow
6: Air flow
7: Control signal
8: prediction data
9: prediction data}
var readfile: file of real;
    Writefile: text;
    Z: integer;
    Data: Data_array;
    Time: real;
begin
Filename := '\thesis\data\'+I.Filename+'.dat';
write('Reading from '+Filename);
assign(Readfile,Filename);
reset(Readfile); {open data file}
Z := 0;
repeat (fill data array from data file)
    read(Readfile,Data[Z,1],Data[Z,2],Data[Z,3],Data[Z,4],
    Data[Z,5],Data[Z,6],Data[Z,7]);
    Z := Z + 1;
    if (Z/40 - INT(Z/40) > -0.10) and (Z/40 - INT(Z/40) < 0.10)
then
    write('.'); {write dots to screen}
until EOF(readfile);
close(readfile);
writeI.Datapoints:=(Z - 1); {determine number of datapoints actually in file}
Filename := '\thesis\data\'+I.Filename+'.dtx';
write('writing to '+Filename);
assign(writefile,Filename);
rewrite(writefile);  {open data text file}
{write parameters and information to data text file}
writeln(writefile,'Parameters used');
writeln(writefile,'Setpoint: ',I.Setpoint:2:O);
writeln(writefile,'Damper on: ',I.Upsets.Damperon,'  Damper off: ',I.Upsets.Damperoff);
writeln(writefile,'Heaters on: ',I.Upsets.Heaterson,'  Heaters off: ',I.Upsets.Heatersoff);
writeln(writefile,'Cold water on: ',I.Upsets.Coldwateron,'  Cold water off: ',I.upsets.Coldwateroff);
writeln(writefile,'Water flow set low at: ',I.Upsets.Wtrflowlow);
writeln(writefile,'Water flow set high at: ',I.Upsets.Wtrflowhigh);
for Z := 0 to I.Datapoints do begin {write data to data text file}
  if (Z/40 - INT(Z/40) > -0.10) and (Z/40 - INT(Z/40) < 0.10) then
  write('.');
end;
close(writefile);
writeln;

Filename ::= '\thesis\data\'+I.Filename+'.prd';
write('reading from '+Filename);
assign(readfile,Filename);
reset(readfile);  {open predictions file}
Z := 0;
repeat {fill data array from predictions file}
  read(readfile,Data[Z,1],Data[Z,2],Data[Z,3],Data[Z,4],
     Data[Z,5],Data[Z,6],Data[Z,7],Data[Z,8],Data[Z,9]);
  Z := Z + 1;
  if (Z/40 - INT(Z/40) > -0.10) and (Z/40 - INT(Z/40) < 0.10) then
  write('.');  {draw dots to screen}
until EOF(readfile);
close(readfile);
writeln;
I.Datapoints := Z - 1; (determine number of datapoints actually in file)

Filename := '\thesis\data\'+I.Filename+'.ptx';
write('writing to '+Filename);
assign(writefile,Filename);
rewrite(writefile); {open predictions text file}
{write parameters, information, and average errors to predictions text file}
writeln(writefile,'Parameters used');
writeln(writefile,'Setpoint: ',I.Setpoint:2:0);
writeln(writefile,'Damper on: ',I.Upsets.Damperon,' Damper off: ',I.Upsets.Damperoff);
writeln(writefile,'Heaters on: ',I.Upsets.Heaterson,' Heaters off: ',I.Upsets.Heatersoff);
writeln(writefile,'Cold water on: ',I.Upsets.Coldwateron,' Cold water off: ',I.upsets.Coldwateroff);
writeln(writefile,'Water flow set low at: ',I.Upsets.Wtrflowlow);
writeln(writefile,'Water flow set high at: ',I.Upsets.Wtrflowhigh);
writeln(writefile,'Sum error model measured: ',I.Error_model_measured:2:2);
writeln(writefile,'Sum error setpoint measured: ',I.Error_setpoint_measured:2:2);
writeln(writefile,'Sum error setpoint model: ',I.Error_setpoint_model:2:2);
for Z := 0 to I.Datapoints do begin {write data to predictions text file, datapoints 0 and 1 only have 7 values}
  if Z < 1 then
  else
procedure caic_error;
(var Readfile, Readfile2, writefile, writefile2: file of real;
Infofile: file of Information;
Time,Air in,Wtr in,Wtr-out,Wtr-flow,Air-flow,CS: real; (data from data file)
Time2,Air in2,Wtr in2,Wtr-out2,Wtr-flow2,Air-flow2,CS2: real; (data from predictions file)
More: array[0..820,1..8] of real; (array for variables other than outlet air temperature in data and prediction files
X, write dot: real;
error-model-measured, error-setpoint-measured,
error-setpoint-model: real; (see global type declarations for detail)
begin
clrscr;
Filename := '\thesis\data\'+I. Filename+'.dat';
writeln('Reading from '+Filename+'
assign(Readfile,Filename);
Filename := '\thesis\data\'+I. Filename+'.prd';
write('Reading from '+Filename);
assign(Readfile2,Filename);
reset(Readfile); (open data file)
reset(Readfile2); (open predictions file)
K := 0;
(Data[K,1] is outlet air temperature in data,
Data[K,2] is outlet air temperature in predictions)
read(Readfile,More[K,1],Data[K,1],More[K,2],More[K,3],
More[K,4],More[K,5],More[K,6],More[K,7],More[K,8]));
read(Readfile2,More[K,1],Data[K,2],More[K,2],More[K,3],
More[K,4],More[K,5],More[K,6],More[K,7],More[K,8]);
if Data[0,1] >= Data[0,2] then Max_temp := Data[0,1]
else Max_temp := Data[0,2]; (initialize max_temp)
if Data[0,1] <= Data[0,2] then Min_temp := Data[0,1]
else Min_temp := Data[0,2]; (initialize min_temp)
I.Error_model_measured := 0; (initialize errors)
I.Error_setpoint_measured := 0;
I.Error_setpoint_model := 0;
repeat (fill data and more arrays)
  K := K + 1;
  read(Readfile,More[K,1],Data[K,1],More[K,2],More[K,3],
  read(Readfile2,More[K,1],Data[K,2],More[K,2],More[K,3],
        More[K,4],More[K,5],More[K,6],More[K,7],More[K,8];
  if Data[K,1] > Max_temp then Max_temp := Data[K,1];
  if Data[K,2] > Max_temp then Max_temp := Data[K,2];
  if Data[K,1] < Min_temp then Min_temp := Data[K,1];
  if Data[K,2] < Min_temp then Min_temp := Data[K,2];
  Error_model_measured := ABS(Data[K,1] - Data[K,2]);
  Error_setpoint_measured := ABS(Data[K,1] - I.Setpoint);
  Error_setpoint_model := ABS(Data[K,2] - I.Setpoint);
  I.Error_model_measured :=
    I.Error_model_measured + Error_model_measured;
  I.Error_setpoint_measured :=
    I.Error_setpoint_measured + Error_setpoint_measured;
  I.Error_setpoint_model :=
    I.Error_setpoint_model + Error_setpoint_model;
  if (K/40 - INT(K/40) > -0.10) and (K/40 - INT(K/40) < 0.10)
    then write('.'); (write dots to screen)
until EOF(Readfile);
close(readfile);
close(readfile2);
I.Datapoints := K; (determine datapoints in file)
{finalize average errors}
I.Error_model_measured := I.Error_model_measured/I.Datapoints;
I.Error_setpoint_measured :=
  I.Error_setpoint_measured/I.Datapoints;
I.Error_setpoint_model :=
  I.Error_setpoint_model/I.Datapoints;
writeln;
{finalize borders for graph}
Max_time := (K * 5) + 5;
Max_temp := Max_temp + 1;
Min_temp := Min_temp - 1;
Time_limit := ROUND((Max_time - 5) / 5);
{write parameters, information and average errors to screen}
        P.g:2:6,' H: ',P.h:2:6);
        I.PID.Kp:2:2,' Td: ',I.PID.Td:2:2);
writeln('Setpoint: ',I.Setpoint:2:0);
 writeln('Damper on: ',I.Upsets.Damperon,
        ' Damper off: ',I.Upsets.Damperoff);
 writeln('Heaters on: ',I.Upsets.Heaterson,
        ' Heaters off: ',I.Upsets.Heatersoff);
 writeln('Cold water on: ',I.Upsets.Coldwateron,
        ' Cold water off: ',I.Upsets.Coldwateroff);
writeln('Water flow set low at: ', I.Upsets.Wtrflowlow);
writeln('Water flow set high at: ', I.Upsets.Wtrflowhigh);
writeln('Max time: ', Max_time:2:0, ' Max temp: ', Max_temp:2:2, ' Min temp: ', Min_temp:2:2);
writeln('Sum error model measured: ',\n'I.Error_model_measured:2:2);
writeln('Sum error setpoint measured: ',\n'I.Error_setpoint_measured:2:2);
writeln('Sum error setpoint model: ',\n'I.Error_setpoint_model:2:2);
assign(infofile, '\thesis\data\'+I.Filename+'.in2');
rewrite(infofile); (open information file)
write(infofile, I); (write average errors to information file)
close(infofile);
end; (calc_error)

procedure plot_data;
(draw graph and plot outlet air temperature in data and predictions)
var X: real;
K: integer;
S,amount_str: string;
Sa,Sb,Sc,Sd,Se,sf,Sg,Sh: string; (values converted to strings)
begin
  initgraphic;
  defineworld(1,0,Max_temp,Max_time,Min_temp);
  definewindow(1,6,0,79,150); (window 1 is graph)
  selectworld(1);
  selectwindow(1);
  drawline(0,Min_temp,Max_time,Min_temp); (draw axis)
  drawline(0,0,0,Max_temp);
  defineworld(2,0,10,Max_time,0);
  definewindow(2,6,150,79,160); (window 2 is X axis)
  selectworld(2);
  selectwindow(2);
  X := Max_time / 5;
  for K := 1 to 5 do begin (write tick marks and axis numbers)
    drawline(X*K,7,X*K,10);
    str(K*X:2:0,S);
    drawtextw((X*K)-Max_time/25,5,1,S);
  end;
  defineworld(3,0,Max_temp,10,Min_temp);
  definewindow(3,0,0,5,150); (window 3 is Y axis)
X := (Max_temp − Min_temp) / 5;
for K := 1 to 5 do begin (write tick marks and axis numbers)
   drawline(9, (X*K) + Min_temp, 11, (X*K) + Min_temp);
   str((X*K)+Min_temp:2:0,S);
   drawtextw(5, (X*K)+Min_temp-0.1,1,S);
end;
defineworid (4,0,0,100,29);
definewindow(4,6,160,79,189); (window 4 is information)
selectworld(4);
selectwindow(4);
draworder;
(write upsets, average errors, files and parameters used)
str(I.Setpoint:4:0,Sa);
drawtextw(1,2,1,'Setpoint: '+Sa);
str(I.Upsets.Damperon:4, Sa); str (I.Upsets.Damperoff:4, Sb);
drawtextw(1,8,1,'Damper on: '+Sa+' Damper off: '+Sb);
str(I.Upsets.Heaterson:4, Sa); str (I.Upsets.Heatersoff:4, Sb);
drawtextw(1,15,1,'Heaters on: '+Sa+' Heaters off: '+Sb);
str(I.Upsets.Coldwateron:4, Sa); str (I.Upsets.Coldwateroff:4, Sb);
drawtextw(40,15,1,'Cold water on: '+Sa+' Cold water off: '+Sb);
str(I.Upsets.Setpointon:4, Sa); str (I.Upsets.Setpointoff:4, Sb);
str(I.Upsets.Setpointsize:4:0,Sc);
drawtextw(25,2,1,'Setpoint on: '+Sa+' Setpoint off: '+Sb+' Setpoint size: '+Sc);
str(I.Upsets.Wtrflowlow:5, Sa); str(I.Upsets.Wtrflowhigh:5, Sb);
drawtextw(40,8,1,'Water flow set low at: '+Sa+' Water flow set high at: '+Sb);
str(I.error_model_measured:5:2,Sa);
str(I.error_setpoint_measured:5:2,Sb);
str(I.error_setpoint_model:5:2,Sc);
drawtextw(1,23,1,'Err model meas: '+Sa+' Err spt meas: '+Sb+' Err spt model: '+Sc);
drawtextw(67,23,1,'File: '+I.Filename+' Param: '+P.Date);
selectworld(1);
selectwindow(1);
(draw lines, and labels for upsets)
if I.Upsets.Damperon > 0 then
   drawline(I.Upsets.Damperon,Min_temp+1,I.Upsets.Damperon,Max_temp-1);
if I.Upsets.Damperoff > 0 then
   drawline(I.Upsets.Damperoff,Min_temp+1,I.Upsets.Damperoff,Max_temp-1);
if I.Upsets.Heaterson > 0 then

drawline(I.Upsets.Heaterson, Min_temp+1, I.Upsets.Heaterson, Max_temp-1); if I.Upsets.Heaterson > 0 then

drawline(I.Upsets.Heatersonoff, Min_temp+1, I.Upsets.Heatersonoff, Max_temp-1); if I.Upsets.Coldwateron > 0 then

drawline(I.Upsets.Coldwateron, Min_temp+1, I.Upsets.Coldwateron, Max_temp-1); if I.Upsets.Coldwateroff > 0 then

drawline(I.Upsets.Coldwateroff, Min_temp+1, I.Upsets.Coldwateroff, Max_temp-1); if I.Upsets.Setpointon > 0 then

drawline(I.Upsets.Setpointon, Min_temp+1, I.Upsets.Setpointon, Max_temp-1); if I.Upsets.Setpointoff > 0 then

drawline(I.Upsets.Setpointoff, Min_temp+1, I.Upsets.Setpointoff, Max_temp-1); if I.PID.alg = 4 then
  Begin if I.Upsets.Wtrflowlow > 0 then

  drawline(I.Upsets.Wtrflowlow, Min_temp+1, I.Upsets.Wtrflowlow, Max_temp-1); if I.Upsets.Wtrflowhigh > 0 then

  drawline(I.Upsets.Wtrflowhigh, Min_temp+1, I.Upsets.Wtrflowhigh, Max_temp-1);
  end;
  Time_limit := ROUND((Max_time - 5) / 5);
  for K := 1 to Time_limit do begin
    drawline((K*5) - 5, Data[K-1,1], (K*5), Data[K,1]); {draw line for measured outlet air temperature}
    {drawline((K*5) - 5, Data[K-1,2], (K*5), Data[K,2]);}
    drawtextw(K*5, Data[K,2], 2, chr(27)+'1'); {draw + for predicted outlet air temperature}
  end;
  savescreen('\thesis\data\'+I.Filename+'.'scr');
end; (plot_data)

procedure Plotdata;
'main program
var
  Infofilename: string;
  fileexists: integer; {boolean for findfile}
  extension: string; {extension for filename}
  
  148
Begin

window(1,1,80,25); {use whole screen}
clrscr;

{find which information file exists,
 *.IN2: predicted data exits
 *.IN1: predicted data does not exist
 *.INF: data taken before August, 1989}
findfile(filename+'.1N2',fileexists);
if fileexists = 1 then begin
    assign(Infofile,'\thesis\data\'+Filename+'.in2');
    extension := '.1N2';
end
else begin
    findfile(filename+'.IN1',fileexists);
    if fileexists = 1 then begin
        assign(Infofile,'\thesis\data\'+Filename+'.in1');
        extension := '.IN1';
    end
    else begin
        assign(Infofile,'\thesis\data\'+Filename+'.inf');
        extension := '.INF';
    end;
end;
reset(Infofile); {open information file}
read(Infofile,I);
close(Infofile);
Filename := '\thesis\data\Paramf.dat';
writeln('reading from '+Filename);
assign(Paramfile,Filename);
reset(Paramfile); {open parameter file}
seek(Paramfile,Paramselect);
read(Paramfile,P);
close(Paramfile);
if I.Datatype = 1 then Calc_error;
if I.Textfile = 1 then Totext;
Plot_data;
end; {plotdata}

End.
"PICTURES.TPU"

UNIT pictures;

interface
uses dos,crt;
procedure drawbox(Left,Top,Right,Bottom:integer);

implementation

procedure drawbox;
var  J: integer;
   X: integer;
begin
  textcolor(random(6) + 1);  (random colors)
  gotoxy(Left,Top);
  case mem[$B800 : 160*(Top-1) + 2*(Left-1)] of (check for corners use correct character to keep corner)
    0..178,181 : write(chr(218));
    179,180,192,195 : write(chr(195));
    191,194,196 : write(chr(194));
    193,197,217 : write(chr(197));
  end;
  for J := Left + 1 to Right - 1 do begin
    if (mem[$B800 : 160*(Top-1) + 2*(J-1)] = 196)
      or (mem[$B800 : 160*(Top-1) + 2*(J-1)] < 179)
    then write(chr(196))
    else write(chr(mem[$B800 : 160*(Top-1) + 2*(J-1)]));
  end;
  case mem[$B800 : 160*(Top-1) + 2*(Right-1)] of
    0..178,191 : write(chr(191));
    179,180,217 : write(chr(180));
    192,193,195,197 : write(chr(197));
    194,196,218 : write(chr(194));
  end;
  for J := Top + 1 to Bottom - 1 do begin
    gotoxy(Left,J);
    if (mem[$B800 : 160*(J-1) + 2*(Left-1)] <= 179)
      then write(chr(179))
    else write(chr(mem[$B800 : 160*(J-1) + 2*(Left-1)]));
    gotoxy(Right,J);
    if (mem[$B800 : 160*(J-1) + 2*(Right-1)] <= 179)
      then write(chr(179))
    else write(chr(mem[$B800 : 160*(J-1) + 2*(Left-1)]));
  end;
  gotoxy(Left,Bottom);
  case mem[$B800 : 160*(Bottom-1) + 2*(Left-1)] of
    0..178,192 : write(chr(192));
    179,195,218 : write(chr(195));
    180,191,194,197 : write(chr(197));
    193,196,217 : write(chr(193));
end;
end;
for J := Left + 1 to Right - 1 do begin
  if (mem[ $B800 : 160*(Bottom-1) + 2*(J-1) ] = 196)
    or (mem[ $B800 : 160*(Bottom-1) + 2*(J-1) ] < 179)
  then write(chr(196))
    else write(chr(mem[ $B800 : 160*(Bottom-1) + 2*(J-1) ]));
end;
case mem[ $B800 : 160*(Bottom-1) + 2*(Right-1) ] of
  0..178,217    : write(chr(217));
  179,180,191   : write(chr(180));
  192,193,196   : write(chr(193));
  194,195,197,218 : write(chr(197));
end;
end;

End.
"PSSPSELF.PAS"

{SR+} (Range checking off)
{SB+} (Boolean complete evaluation on)
{SS+} (Stack checking on)
{SI+} (I/O checking on)
{SM 65500,16384,655360} (Turbo 3 default stack and heap)
{ stack heapmin heapmax }

program Pssp(input,output);  (11-28-89)
Uses Dos, Crt, gdriver, gkernel, Pictures;

type Parameters = record
  date: string;
  a: real; b: real; c: real; d: real;
  e: real; f: real; g: real; h: real; j: real;
end;

arr3x3 = array[1..3,1..3] of real;
Data_array = array[0..200,1..7] of real;

Const
GPM_to_Kgps = 0.06296382;
FPM_to_kgps = 0.001221041;

var K, iterations, stop, Datapoints, bias, Time_limit: integer;
P: Parameters;
over_shoot : arr3x3;
kos : array[1..10,1..3] of real;
Data : Data_array;
Kosfile, writefile : text;
Filename: string;
Sa,Sb,Sc,Sd,Se,Sf,Sg,Sh: string;
Kpstart, Kpend, Kpstep, setpointbase, setpointstep: real;
MdotWmiddle, Delta_water, Kp, Setpoint : real;
Max_temp, Min_temp, Max_time: real;
PercOS, Watflow, K_step, Sum_error: real;
Kc, Kr, Ks, CO, CO_1, MdotWplus, MdotWminus: real;
x1, x2, setpointplus, setpointminus, Twssbase, Taossbase,
Twssminus, Taossminus, temp1, temp2, Kv, setpointupper : real;

function XtoY(X,Y:real):real;
var Z: real;
beg
  if X = 0 then XtoY := 0
  else begin
    Z := Y * ln(X);
  end

152
\[ XtoY := \exp(Z); \]
end;
end;

procedure PID;

procedure calc_pid;

var aot_count_real, aot_count_real_old, PID_calc: real;
aot_count: integer;
Mdotwbar : real;
Mwcount : byte;

begin
  pid_calc := Kp*(Data[K,2] - Setpoint);
aot_count_real := pid_calc;
aot_count := round(aot_count_real);  \{ declared in this procedure \}
end;

begin
  CO_1 := CO;
  CO_1 := aot_count;  \{ uses PID calculation, not CO of actual test \}
end;

begin
  calc_pid;
  Data[K,5] := 3 + (Kv/0.06296382)*(CO_1 - 1164);
  if Data[K,5] < 0.0 then Data[K,5] := 0.0;
  if Data[K,5] > 5.37 then Data[K,5] := 5.37;
end;  \{ procedure PID \}

procedure Set_data;
var
  m: integer;
begin
  for m := 0 to 200 do
  begin
    Data[m,1] := 30;  Data[m,2] := 50;  Data[m,3] := 74;
    Data[m,4] := 56;
    Data[m,5] := 3;  Data[m,6] := 700;  \{ base case \}
    if (m/40 - INT(m/40) < 0.10) and (m/40 - INT(m/40) > -0.10)
    then
      write('.');
    writeln;
  end;
end;

procedure Wtr_out_calculation(a,b,c,d: real);
var MdotA, MdotW, Wtr_avg: real;
begin
  MdotA := Data[K-1,6]*FPM_to_Kgps;
  MdotW := Data[K-1,5]*GPM_to_Kgps;
  Wtr_avg := (Data[K-1,3] + Data[K-1,4]) / 2;
  Data[K,4] := Data[K-1,4]
\[ + (a \times M_{dotW} \times (Data[K-1,3] - Data[K-1,4])) \\
+ (b \times (Data[K-1,1] - Wtr_{avg})) \\
+ (c \times M_{dotW} \times (Data[K-1,1] - Wtr_{avg})) \\
+ (d \times M_{dotA} \times (Data[K-1,1] - Wtr_{avg})) \]

end;

procedure Air_out_calculation(e,f,g,h,j: real);
var MdotA, MdotW, Wtr_avg : real;
begin
MdotA := Data[K,6]*FPM_{to Kgps};
MdotW := Data[K,5]*GPM_{to Kgps};
Wtr_avg := (Data[K,3] + Data[K,4]) / 2;
Data[K+1,2] := Data[K,2] \\
+ (e \times M_{dotA} \times (Data[K,1] - Data[K,2])) \\
+ (f \times (Wtr_{avg} - Data[K,1])) \\
+ (g \times M_{dotW} \times (Wtr_{avg} - Data[K,1])) \\
+ (h \times M_{dotA} \times (Wtr_{avg} - Data[K,1])) \\
+ (j \times (Data[K+1,1]-Data[K,1]));
gotoxy(1,25); \\
textcolor(Random(6) + 1);
write('Tao pred = ',Data[K+1,2]:2:2);
end;

procedure make_predictions;
begin
window(1,1,80,25); \\
write ('Calculating'); \\
K := 2; \\
datapoints := 200; \\
Data[0,7] := setpoint_base; \\
Data[1,7] := setpoint_base; \\
data[2,7] := setpoint_base; \\
repeat \\
If K <= 75 then setpoint := setpoint_base; \\
If (K > 75) and (K <= 150) then setpoint := setpoint_plus - setpoint_step; \\
If K > 150 then setpoint := setpoint_base; \\
Data[K,7] := setpoint; \\
PID; \\
Wtr_out_calculation(P.a,P.b,P.c,P.d); \\
Air_out_calculation(P.e,P.f,P.g,P.h,P.j); \\
K := K + 1; \\
Until K = datapoints - 1; \\
writeln;
end;

procedure Datafromfile;
begin \\
writeln('Kp = ',Kp:2:2); \\
MdotW_{plus} := (MdotW_{middle} + Delta_{water})*0.06296382; \\
154
\[ M_{\text{dot}W\text{minus}} := (M_{\text{dot}W\text{middle}} - \Delta \text{water}) \ast 0.06296382; \]

\[ x_1 := ((P.b + (P.c \ast M_{\text{dot}W\text{plus}}) + (P.d \ast \text{data}[1,6] \ast 0.001221041)) \ast (\text{data}[1,1] - (0.5 \ast \text{data}[1,3])) + (P.a \ast M_{\text{dot}W\text{plus}} \ast \text{data}[1,3])) / \\
((P.a \ast M_{\text{dot}W\text{plus}}) + (0.5 \ast (P.b + (P.c \ast M_{\text{dot}W\text{plus}}) + (P.d \ast \text{data}[1,6] \ast 0.001221041)))\}; \]

\[ \text{temp1} := (\text{data}[1,3] + x_1) / 2; \]

\[ \text{temp2} := \text{temp1} - \text{data}[1,1]; \]

\[ x_2 := \text{data}[1,1] + \\
((P.f / (P.e \ast \text{data}[1,6] \ast 0.001221041)) \ast \text{temp2}) + \\
((P.g \ast M_{\text{dot}W\text{plus}}) / (P.e \ast \text{data}[1,6] \ast 0.001221041)) \ast \text{temp2} + \\
(P.h / P.e) \ast \text{temp2}; \]

\[ \text{Twssbase} := \text{temp1}; \quad \text{Taossbase} := x_2; \]

\[ \text{setpointplus} := x_2 - (((M_{\text{dot}W\text{plus}} / 0.0629) - 3) / -0.00702) + \\
1164 - \text{bias}) / K_p; \]

\[ x_1 := ((P.b + (P.c \ast M_{\text{dot}W\text{minus}}) + (P.d \ast \text{data}[1,6] \ast 0.001221041)) \ast (\text{data}[1,1] - (0.5 \ast \text{data}[1,3])) + (P.a \ast M_{\text{dot}W\text{minus}} \ast \text{data}[1,3])) / \\
((P.a \ast M_{\text{dot}W\text{minus}}) + (0.5 \ast (P.b + (P.c \ast M_{\text{dot}W\text{minus}}) + (P.d \ast \text{data}[1,6] \ast 0.001221041)))\}; \]

\[ \text{temp1} := (\text{data}[1,3] + x_1) / 2; \]

\[ \text{temp2} := \text{temp1} - \text{data}[1,1]; \]

\[ x_2 := \text{data}[1,1] + \\
((P.f / (P.e \ast \text{data}[1,6] \ast 0.001221041)) \ast \text{temp2}) + \\
((P.g \ast M_{\text{dot}W\text{minus}}) / (P.e \ast \text{data}[1,6] \ast 0.001221041)) \ast \text{temp2} + \\
(P.h / P.e) \ast \text{temp2}; \]

\[ \text{Twssminus} := x_1; \quad \text{Taossminus} := x_2; \]

\[ \text{setpointminus} := x_2 - (((M_{\text{dot}W\text{minus}} / 0.0629) - 3) / -0.00702) + \\
1164 - \text{bias}) / K_p; \]

\[ \text{setpointbase} := \text{setpointplus}; \]

\[ \text{setpointstep} := \text{setpointplus} - \text{setpointminus}; \]

\[ \text{writeln}(\text{spbase}', \text{setpointbase}:2:2, ', \text{spstep}' , \text{setpointstep}:2:2); \]

\[ \text{writeln}('\text{Twbar} = ', \text{Twssbase}:2:2, ', ', \text{Taobar} = '
, \text{Taossbase}:2:2); \]

\[ \text{Delay}(1000); \]

\[ \text{make_predictions}; \]

end; \quad \text{(procedure Data_from_file)}

procedure calc_error(var stop: integer);

var
    \[ \text{Time, Air\_in, Wtr\_in, Wtr\_out, Wtr\_flow, Air\_flow, CS: real}; \quad \text{real}; \]
    \[ \text{Time2, Air\_in2, Wtr\_in2, Wtr\_out2, Wtr\_flow2, Air\_flow2, CS2: real}; \]
    \[ \text{X, write\_dot: real}; \]
    \[ \text{Max\_temp\_pos1, Min\_temp\_pos1, Max\_temp\_pos2, Min\_temp\_pos2: real}; \]
    \[ \text{PercCos : real}; \]

begin
clrscr;
K := 0;
Max_temp_pos1 := 0;
Min_temp_pos1 := 100;
if Data[0,1] >= Data[0,2] then Max_temp := Data[0,1]
else Max_temp := Data[0,2];
if Data[0,1] <= Data[0,2] then Min_temp := Data[0,1]
else Min_temp := Data[0,2];
repeat
K := K + 1;
if Data[K,2] > Max_temp then Max_temp := Data[K,2];
if (Data[K,2] > Max_temp_pos1) and (K >= 75) then
Max_temp_pos1 := Data[K,2];
if Data[K,2] < Min_temp then Min_temp := Data[K,2];
if (Data[K,2] < Min_temp_pos1) and (K >= 75) then Min_temp_pos1 := Data[K,2];
if (K/40 - INT(K/40) > -0.10) and (K/40 - INT(K/40) < 0.10) then
write('.');
until K = 150;
writeln;
writeln('Min = ',Min_temp_pos1:2:2);
writeln('Max = ',Max_temp_pos1:2:2);
writeln('final = ',Data[K,2]:2:2);
If abs(Data[K,2] - Max_temp_pos1) > 0.001 then
PercOS := 100*(Min_temp_pos1 - Data[K,2]) / (Data[K,2] - Data[75,2])
else PercOS := 0;
if (PercOS > 24) and (PercOS < 26) then stop := 1;
writeln(PercOS);
str(PercOS:5:2,Sa);
if (PercOS > 24.5) and (PercOS < 25.5) then stop := 1;
append(KOSfile);
write(KOSfile,PercOS:12:2);
close(KOSfile);
over_shoot[1,1] := over_shoot[2,1];
over_shoot[2,1] := Percos;
over_shoot[3,1] := over_shoot[2,1] - over_shoot[1,1];
Max_temp_pos2 := Data[150,2];
Min_temp_pos2 := Data[150,2];
repeat
K := K + 1;
if Data[K,2] > Max_temp then Max_temp := Data[K,2];
if Data[K,2] > Max_temp_pos2 then Max_temp_pos2 := Data[K,2];
if Data[K,2] < Min_temp then Min_temp := Data[K,2];
if Data[K,2] < Min_temp_pos2 then Min_temp_pos2 := Data[K,2];
if (K/40 - INT(K/40) > -0.10) and (K/40 - INT(K/40) < 0.10) then
write('.');
until K=198;
write;
writeln('Min = ',Min_temp_pos2:2:2);
writeln('Max = ',Max_temp_pos2:2:2);
writeln('final = ',Data[K,2]:2:2);
If (Data[K,2] - Min_temp_pos2) > 0.001 then
PercOS := 100*(Max_temp_pos2 - Data[K,2])/(Data[K,2] - Data[150,2])
else PercOS := 0;
writeln(PercOS);
if (PercOS > 24) and (PercOS < 26) then stop := 1;
str(PercOS:5:2,Sb);
append(KOSfile);
over_shoot[1,2] := over_shoot[2,2];
over_shoot[2,2] := PercOS;
over_shoot[3,2] := over_shoot[2,2] - over_shoot[1,2];
writeln(KOSfile,PercOS:12:2,over_shoot[3,1]:8:2,over_shoot[3,2]:8:2);
close(KOSfile);
writeln;
Max_time := (K * 5) + 5;
Max_temp := Max_temp + 1;
Min_temp := Min_temp - 1;
Time_limit := ROUND((Max_time - 5) / 5);
Watflow := Data[75,3];
If (over_shoot[2,1] > 25) and (over_shoot[3,1] > 0) then
over_shoot[1,3] := - over_shoot[1,3]/2;
If (0 < over_shoot[3,1]) and (over_shoot[3,1] < 20) then
over_shoot[1,3] := 2*over_shoot[1,3];
If stop = 1 then begin write(char(7)); delay(2000); end;
end; (procedure calc error)

procedure plot_data;
var X: real;
K: integer;
S,amount_str: string;
begin
initgraphic;
defineworld(1,0,Max_temp,Max_time,Min_temp);
defineworld(1,6,0,79,150);
selectworld(1);
selectwindow(1);
drawline(0,Min_temp,Max_time,Min_temp);
drawline(0,0,0,Max_temp);
defineworld(2,0,10,Max_time,0);
defineworld(2,6,150,79,160);
selectworld(2);
selectwindow(2);
X := Max_time / 5;
for K := 1 to 5 do
begin
    drawline(X * K,7,X * K,10);
    str(K*X:2:0,S);
    drawtextw((X*K)-Max_time/25,5,1,S);
end;
defineworld(3,0,Max_temp,10,Min_temp);
definewindow(3,0,0,5,150);
selectworld(3);
selectwindow(3);
X := (Max_temp - Min_temp) / 5;
for K := 1 to 5 do
    begin
        drawline(9,(X*K)+Min_temp,11,(X*K)+Min_temp);
        str((X*K)+Min_temp:2:0,S);
        drawtextw(5,(X*K)+Min_temp-0.1,1,S);
    end;
defineworld(4,0,0,100,29);
definewindow(4,6,160,79,189);
selectworld(4);
selectwindow(4);
drawborder;
drawtextw(1,23,1,'Percent OS down '+Sa+
    ' Percent OS up: '+Sb);
str(Watflow:2:2,Sc);
drawtextw(1,10,1,'Water flow: '+Sc);
str(Kp:2:2,Sc);
drawtextw(30,10,1,'Kp: '+Sc);
selectworld(1);
selectwindow(1);
drawline(375,Min_temp+l,375,Max_temp-i);
drawline(750,Min_temp+l,750,Max_temp-i);
Time_limit := ROUND((Max_time - 5) / 5);
for K := 1 to Time_limit do 
    begin 
        drawline((K*5) - 5,Data[K-1,3]*10,(K*5),Data[K,3]*10);
        drawtextw(K*5,Data[K,2],2,chr(27)+'1');
    end;
end;

procedure Plotdata;
var Infofilename: string;
    fileexists: integer;
    extension: string;
Begin
    stop := 0;
    window(1,1,80,25);
    clrscr;
    assign(KOSfile,'\thesis\data\KOS.dat');
    append(KOSfile);
    write(KOSfile,Kp:8:2);
    close(KOSfile);
Calc_error(stop);
Plot_data;
delay(500);
if stop <> 1 then leavegraphic;
end;

Begin {main program}
P.a := 0.498213; P.b := 0.019103; P.c := 0.064465; P.d :=
0.022166;
P.e := 0.152283; P.f := 0.022538; P.g := 0.078853; P.h :=
0.045770;
P.j := 0.210209; (from fit of 8-15cl1)
{write('setpoint upper?');
readln(setpointupper);}
Kv := -0.000442;
 writeln('bias');
 readln(bias);
 clrscr;
gotoxy(3,6); write('Kp start = '); readln(Kpstart);
gotoxy(3,7); write('Kp end = '); readln(Kpend);
gotoxy(3,8); write('Kp step = '); readln(Kpstep);
gotoxy(3,9); write('Mw step (gpm) = '); readln(Delta_water);
}
Delta_water := 0.05;
gotoxy(45,6); write('Mdowt middle (GPM) = ');
readln(MdotWmiddle);
stop := 0;
iterations := 0;
Kp := Kpstart;
assign(KOSfile, '\thesis\data\KOS.dat');
 rewrite(KOSfile);
 writeln(KOSfile, 'Kp':8, '% OS down':12, '% OS up':12, 'Water flow':13, 'MdotWmiddle:6:2');
close(KOSfile);
 fillchar(over_shoot,sizeof(over_shoot),0);
 fillchar(Data,sizeof(Data),0);
While (Kp <= Kpend) and (stop = 0) do begin
 randomize;
 textcolor(random(6) + 1);
gotoxy(3,6); write('Kp start = ',Kpstart:2:2);
gotoxy(3,7); write('Kp end = ',Kpend:2:2);
gotoxy(3,8); write('Kp step = ',Kpstep:2:2);
gotoxy(45,6); write('Waterflow = ',MdotWmiddle:2:2,' ');
over_shoot[1,3] := Kpstep;
 Datafromfile;
 plotdata;
 Kpstep := over_shoot[1,3];
 Kp := Kp + Kpstep;
 Delay(10000);
end; {while}
leavegraphic;
clrscr;
Kp := Kp - Kpstep;
writeln('Kp = ',Kp:2:2);
writeln('SPb = ',setpointbase:2:2);
writeln('%OS down = ',over_shoot[2,1]:2:2);
writeln('%OS up = ',over_shoot[2,2]:2:2);
writeln('del SP = ',setpointbase-setpointminus:2:2);

(Filename := '\thesis\data\simexl.dtx';
assign (writefile, filename);
rewrite (writefile);
writeln(writefile,Kp:2:2,' ',over_shoot[2,1]:2:2,' 
',over_shoot[2,2]:2:2);
For K := 0 to datapoints do writeln(writefile, K*5/60:2:2,' 
',Data[K,2]:2:2,' 
',Data[K,7]:2:2);
close (writefile);)
End.
program Readmat;
($N+$)
var matrixfile : file;
mattype,matrows,matcols, matimag,matnamelength : longint;
matname : byte;
matrix : array[1..10,1..500] of double;
I,J,K : integer;
matrixtext : text;

begin
assign(matrixfile,'\thesis\data\3gpmrl.mat'); {file root locus saved in}
reset(matrixfile,1);
for K := 1 to 1 do begin
  blockread(matrixfile,mattype,4); writeln(mattype:5);
  blockread(matrixfile,matrows,4); writeln(matrows:5);
  blockread(matrixfile,matcols,4); writeln(matcols:5);
  blockread(matrixfile,matimag,4); writeln(matimag:5);
  blockread(matrixfile,matnamelength,4);
  for I := 0 to matnamelength - 1 do begin
    blockread(matrixfile,matname,1);
    write(chr(matname));
  end;
  writeln;
  for I := 1 to matcols do begin
    for J := 1 to matrows do begin
      blockread(matrixfile,matrix[I,J],8);
      write(matrix[I,J]:10:4);
    end;
    writeln;
  end;
  if matimag = 1 then begin
    for I := matcols + 1 to matcols * 2 do begin
      for J := 1 to matrows do begin
        blockread(matrixfile,matrix[I,J],8);
        write(matrix[I,J]:10:4);
      end;
    end;
    writeln;
  end;
end;
close(matrixfile);
assign(matrixtext,'\thesis\data\3gpm.sy'); {ascii file}
rewrite(matrixtext);
for J := 1 to matrows do begin
  write(matrix[1,J]:10:4);
  if matrix[4,J] > 0 then write(' +') else
  if matrix[4,J] = 0 then write('  ') else
  write(matrix[4,J]:10:4);
write(' -');
if matrix[4,J] = 0 then write(' :7) else
  write(ABS(matrix[4,J]):7:4);
if matrix[4,J] = 0 then write(' ') else
  write(' i');
write(matrix[2,J]:10:4);
if matrix[5,J] > 0 then write(' +') else
  if matrix[5,J] = 0 then write(' ') else
    write(' -');
  if matrix[5,J] = 0 then write(' :7) else
    write(ABS(matrix[5,J]):7:4);
if matrix[5,J] = 0 then write(' ') else
  write(' i');
write(matrix[3,J]:10:4);
if matrix[6,J] > 0 then write(' +') else
  if matrix[6,J] = 0 then write(' ') else
    write(' -');
  if matrix[6,J] = 0 then write(' :7) else
    write(ABS(matrix[6,J]):7:4);
if matrix[6,J] = 0 then writeln(' ') else
  writeln(' i');
writeln(matrixxtext,matrix[1,J]:9:4,matrix[4,J]:9:4,matrix[2,J]:9:4,
end;
close(matrixxtext);
end.
"DSTEST.M"

% An M-file to plot calculate closed loop transfer function
% and plot step response for the FRR3 model
clear
stopit = 0
mw = 0.5
kp = input('Kp to start with')
while (stopit ~= 1)
    clear kos y os
    j = 1
    os(1) = 0
    delos = 10
    incr = input('increment value')
    while abs(os(j) - 25) > 0.5
        !erase \matlab.313\under\param.tmp
        fprintf('\matlab.313\under\param.tmp', '%9.3g %9.3g', kp, mw)
        !test3
        load \matlab.313\under\f_gfrr3.mat
        ymax = max(y)
        ymin = min(y)
        os(j+1) = 100*(ymax-y(99))/(y(99)-ymin)
        kos(j+1,1) = kp
        kos(j+1,2) = os(j+1)
        if (os(j+1) > 25 & (incr>0), incr=-incr/2, end
        if (j>3), delos=os(j+1)-os(j), end
        if ((delos<5) & (os(j+1)<20) & (j>3)), incr=2*incr, end
        if ((delos<0) & (incr<0) & (os(j+1)<24)), incr=-incr/2, end
        if abs(incre)<0.001, incr=2*incr, end
        pause(1)
        plot(y,'*')
        pause(1)
        kp = kp + incr
        j = j + 1
    end
!beep
mw kos
stopit = input('do another 0 stop 1 ')
mw = mw + 0.5
end
end