EVALUATION OF ADAPTIVE CONTROL USING A PRACTICAL MODEL OF A REHEATING FURNACE

Ralf Kiehl

A dissertation submitted to the Faculty of Engineering, University of the Witwatersrand, Johannesburg, in fulfilment of the requirements for the Degree of Master of Science in Engineering.

Johannesburg, 1990

DECLARATION

I declare that this dissertation is my own unaided work. It is being submitted for the Degree of Master of Science in Engineering at the University of the Witwatersrand, Johannesburg. It has not been submitted before for any degree or examination at any other university.

R. Kiehl

R. Kiehl

19th day of March 1990

ACKNOWLEDGEMENTS

I wish to express my thanks to:

- Prof. I.M. MacLeod, who has supervised this disserta_ion, for his support.

- Mr. A. Montini, for his assistance during the initial stages of this dissertation.

- Iscor for their continued interest and financial support of this project and for granting me enough time to finish this work.

ABSTRACT

The aim of this dissertation is to investigate improved control of the Bottom Heating Zone of a steel slab reheating furnace through the use of digital adaptive control techniques. An existing algorithm is applied and its performance is evaluated. A mathematical model of the Bottom Heating Zone is developed for simulation purposes. The model is based on an energy balance and is of such a nature that its parameters are directly related to zone characteristics. Experimental work to validate and fine tune the model is discussed. A robust PI adaptive controller design is thoroughly tested by means of simulation using the model. The results show that the controller performs well under all the test conditions and it is concluded that this design could be used with a fair amount of confidence on the real furnace.

CONTENTS

PAGE

LIST OF FIGURES

CHAPTER 1 Introduction

1.1 Background

The reheating furnace at the Iscor Vanderbijlpark Steel Works is part of the Hot Mills North. It is called Furnace Two. Figure 1.1 shows a block diagram of the Hotmill. There are three reheating furnaces in the Hot Mills North that supply the Roughing Mill with heated slabs. Here the slabs are reduced in thickness and width. From the roughing mill the partially rolled steel enters the 6-Stand which reduces the thickness of the steel to a predefined setting in six stages. The thickness usually being less than one centimeter. Once the steel has gone through the 6-Stand it is coiled by one of the two coilers.

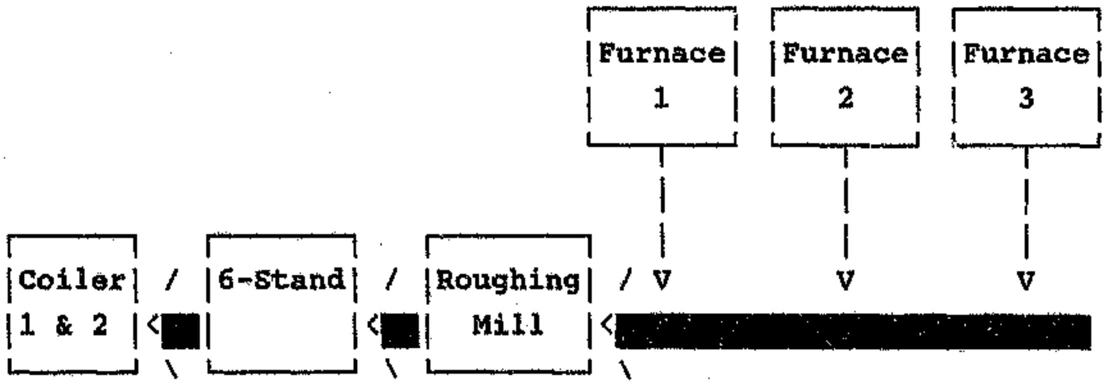


Fig 1.1 Block Diagram of the Hotmill

The furnace consists of five temperature zones which are controlled separately. Each temperature zone has between two and four controllers. Two controllers always control the air and gas flow of each zone. The top zones have another two controllers which have a protective function. They influence the system once the roof or slab temperatures become too high. There are three more controllers that are responsible for controlling the main gas line pressure, the main air line pressure and the inside furnace pressure. The soak zone is split into two zones from a control point of view. The furnace therefore has approximately twenty three controllers.

Once all the contiollers are tuned properly the furnace operates well. However, the furnace's parameters change with time and the temperature of the various zones in the furnace start oscillating. This occurs within a week and typically the technicians have to retune various control loops twice a week. Because of this unsatisfactory state of affairs, an investigation into the role that adaptive control techniques might play in alleviating the problem with the existing control system is of interest.

This project therefore attempts to improve the control of the furnace. Although many different approaches could be tried, it was decided to investigate digital adaptive control because:

(i) a seminar on adaptive control was presented at Iscor which claimed promising results.

(ii) the control problems displayed by the furnace are most probably due to changes in its characteristics which would make adaptive control inherently suitable.

(iii) I wanted to get a deeper insight into adaptive control.

The furnace operates for months before a shutdown is necessary. During this time some characteristics of the furnace change e.g. piston, valve, transmitter or burner characteristics. After monitoring the furnace for a few weeks it was found that the Bottom Heating Zone was the one whose controller had to be retuned more often than any other zone. It was therefore decided that the Bottom Heating Zone was the critical one. This research therefore concentrates on the Bottom Heat Zone.

The aim of this investigation is not to develop low-level adaptive controller theory but rather to apply a suitable existing algorithm and evaluate its performance. In order to evaluate the controller design a good model of the Bottom Heating Zone is required. A large part of this thesis therefore concentrates on the development of such a model.

1.2 Overview of the Dissertation

Chapter Two provides a statement of the problem and describes the control system in detail.

The development of a model of the furnace is covered in Chapter Three. In order to model the furnace, real data had to be logged. An IBM PC compatible machine with a sixteen channel Analogue to Digital (A/D) converter card was used two achieve this. The relevant software for the logging and graphical representation of the data was also developed. Various input tests were performed so that the furnace could be characterized. Finally an energy balance in conjunction with test results was used to derive the model.

Chapter Four covers the adaptive controller. It describes the derivation of the algorithm in the review section. The actual implementation of this algorithm on an IBM AT compatible computer using a Turbo Pascal program is presented in the last part of this chapter.

Finally the designed adaptive Controller was tested on the model via a computer simulation. Chapter Five covers this aspect of the project.

Chapter Six rounds off the dissertation with a conclusion. Here all the results of the thesis are briefly discussed.

CHAPTER 2 Statement of the Problem

This chapter describes the reheating furnace and its control system in detail. The problems experienced during operation are discussed and scope and purpose of the research project are defined.

2.1 Description of the Furnace

The reheating furnace is situated in the North Works at the Iscor Vanderbijlpark Steel Works. It is part of the Hotmill North. There are three reheating furnaces in total, but this investigation is confined to Furnace Two.

It is not normally possible to achieve one hundred percent direct hot rolling of steel slabs. In order to detect and remove surface defects the slab has to be cooled down. The temperature of the slab on leaving the continuous casting process is also too low for immediate hot rolling. A reheating furnace is therefore necessary in a Hot Strip Mill.

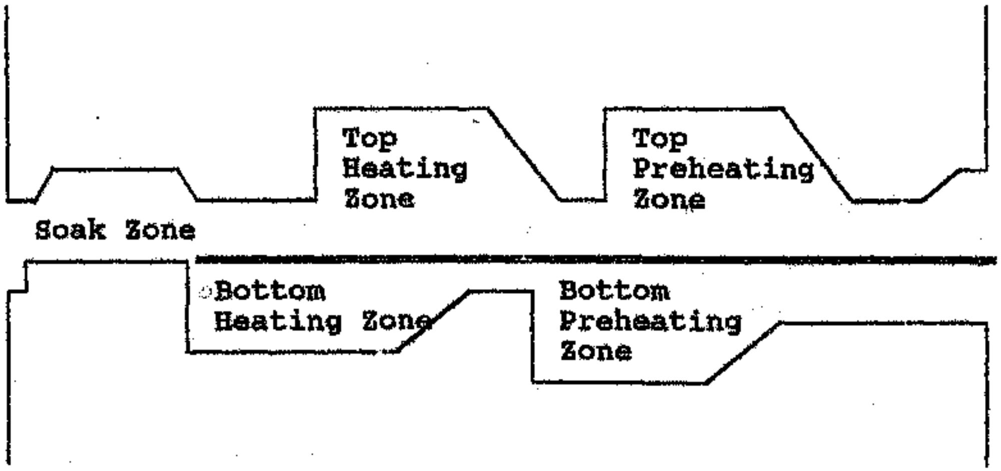


Fig. 2.1 Diagram of Reheating Furnace

Furnace Two (see figure 2.1) is of the pusher type. The slabs are put side by side on a skid and then pushed through via a mechanical pusher. The furnace is therefore charged by pushing in one slab after the other.

It consists of five zones namely: Bottom Preheat (BP), Top Preheat (TP), Bottom Heating (BH), Top Heating (TH) and the Soak Zone (SZ). For control purposes the Soak Zone is divided into two; Soak Zone West and Soak Zone East.

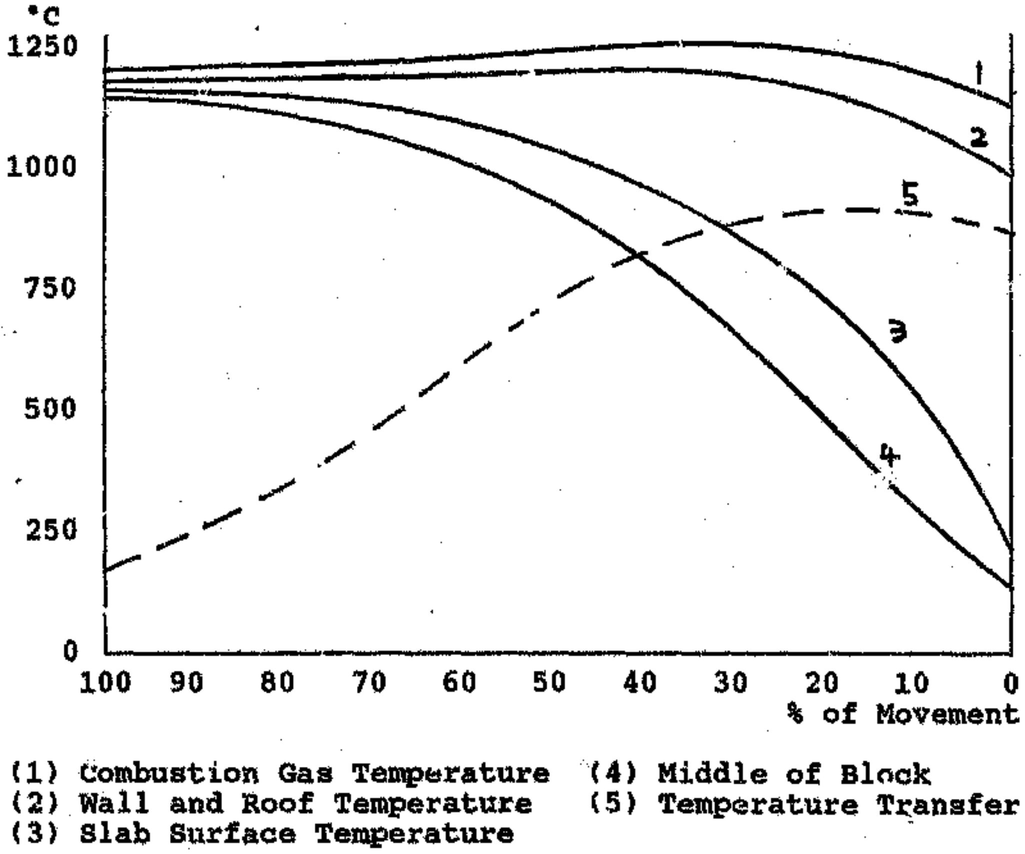


Fig. 2.2 Temperature Distributions within the Furnace

Heat transfer within the furnace chamber occurs as radiation and convection to the slab surface and conduction within the slab. Most energy is consumed by the preheating zones where the slabs are heated up from ambient temperature to ±900°C. The next largest consumers of energy are the heating zones which heat up the slab to ±1150°C. The soak zones ensure an even temperature of ±1200°C throughout the slab. Fahien (1983) describes the various temperatures within the furnace and the temperature distribution within the slabs throughout the furnace. Figure 2.2 and 2.3 show these temperature distributions.

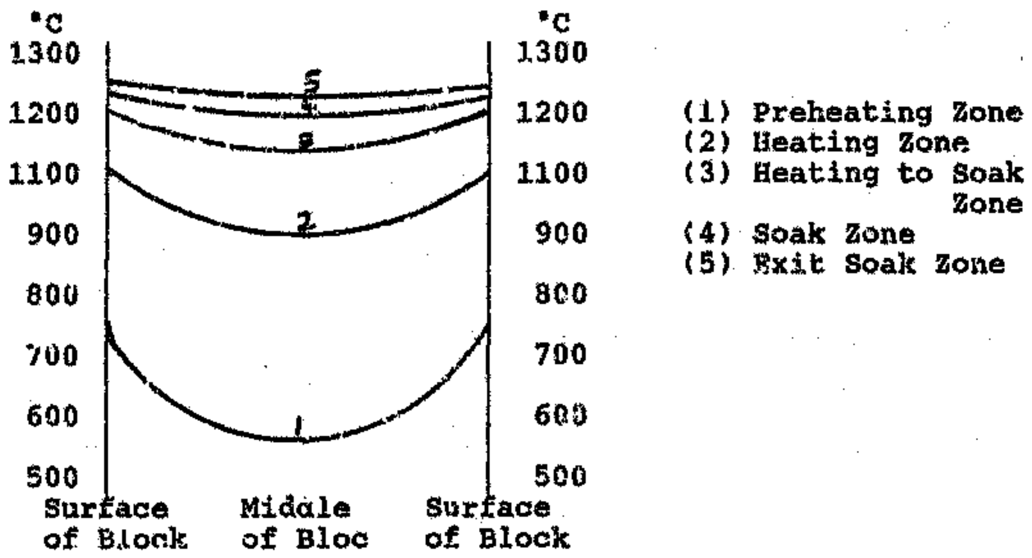


Fig. 2.3 Temperature Distribution within the Slab

Within the furnace the primary source of heat are the burners. They are responsible for the combustion of the air and the coke oven gas. The exhaust gases are sent through a heat recuperator to preheat combustion air before they leave the furnace via the stack.

2.2 The existing Control System

A detailed diagram of the control system of the furnace is given in Appendix A. The original design of the furnace also allowed the burning of oil and steam. However, this option has become redundant.

The SZ, TH and TP zones measure three temperatures each, namely zone temperature, roof temperature and slab temperature. Thermocouples are used to measure the zone and roof temperatures. For backup purposes two thermocouples are installed for each measurement. The operator can switch from one thermocouple to the other if he suspects a fault and thereby compare them against each other.

The roof temperature is monitored for protection of the furnace. The zone temperature is the measured variable for the temperature controller whose output is fed into a low signal selector. This selector also receives the signals of the roof and slab temperature controllers. Should either the roof or the slab temperature sensor detect a dangerously high temperature, this selector will switch over to the correct controller which is set in such a way that it will react immediately by cutting the air and gas flows to prevent any damage.

The slab temperature is measured by a pyrometer. This temperature measurement is not accurate because of reflections within the furnace and can thus only be used for protection purposes. The BH and BP zones only measure the zone temperatures.

It was previously mentioned that the exhaust gases are passed through a recuperator before leaving the system via the stack. The recuperator has to be monitored so that it does not become too hot. Its temperature is monitored using several thermocouples. Whenever the temperature becomes too high, cold air is blown into the exhaust gas in order to cool it down. The exhaust gas temperature is measured inside the stack so that the efficiency of the recuperator can be estimated. The recuperator heats up the air for the burners located in the TP, BP, TH and BH zones to about 400°C. This is done to conserve energy. The air pressure inside the pipes feeding the above mentioned burners is also controlled.

The soak zones are fed with air at ambient temperature. The pressure in this system is controlled separately. The gas pressure in the main gas line is also controlled by yet another control loop. In order to prevent the cold outside air from entering the furnace the internal pressure is controlled at slightly above atmospheric pressure.

The Bottom Heating zone was chosen as critical for this project because it presented the most difficulties. According to the plant technicians the BH zone temperature cycled more often than any of the other zones and that its controllers were the most difficult to tune. It was thus decided to work on this zone first. Research findings and experience gained could then be applied to the other zones.

Appendix B shows a detailed control diagram of the BH zone. As previously mentioned, steam and oil are no longer used and can be ignored. Figure 2.4 shows a simplified diagram of the control setup. The temperature controller can receive its setpoint from either the mainframe coordinating computer or from the operator via the front panel of the controller. The measured variable is sent to the controller via one of the two thermocouples. The reason for having two thermocouples is that the one backs up the other. The operator can switch between the two. The output of the controller is then sent to a current to pressure converter. The pressure signal then operates on the butterfly valve in the gas line. The piston together with the valve displays a linear response.

The second control loop regulates the air flow. This

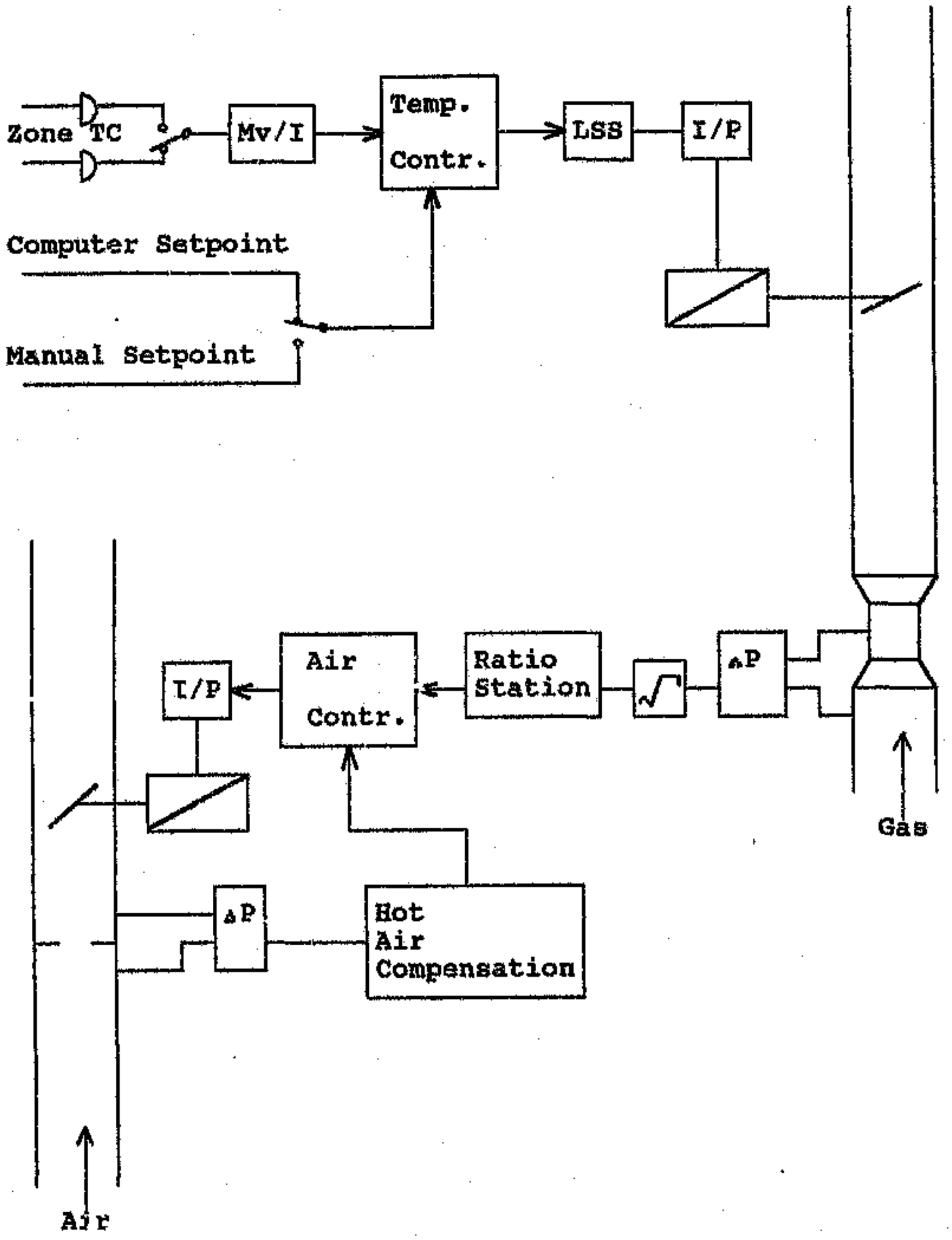


Fig 2.4 Control Diagram of Bottom Heating Zone

controller receives its setpoint from the ratio station which is set to a fixed ratio. The ratio station in turn receives the gas flow signal via the flow sensor. This sensor consists of a venturi tube together with a

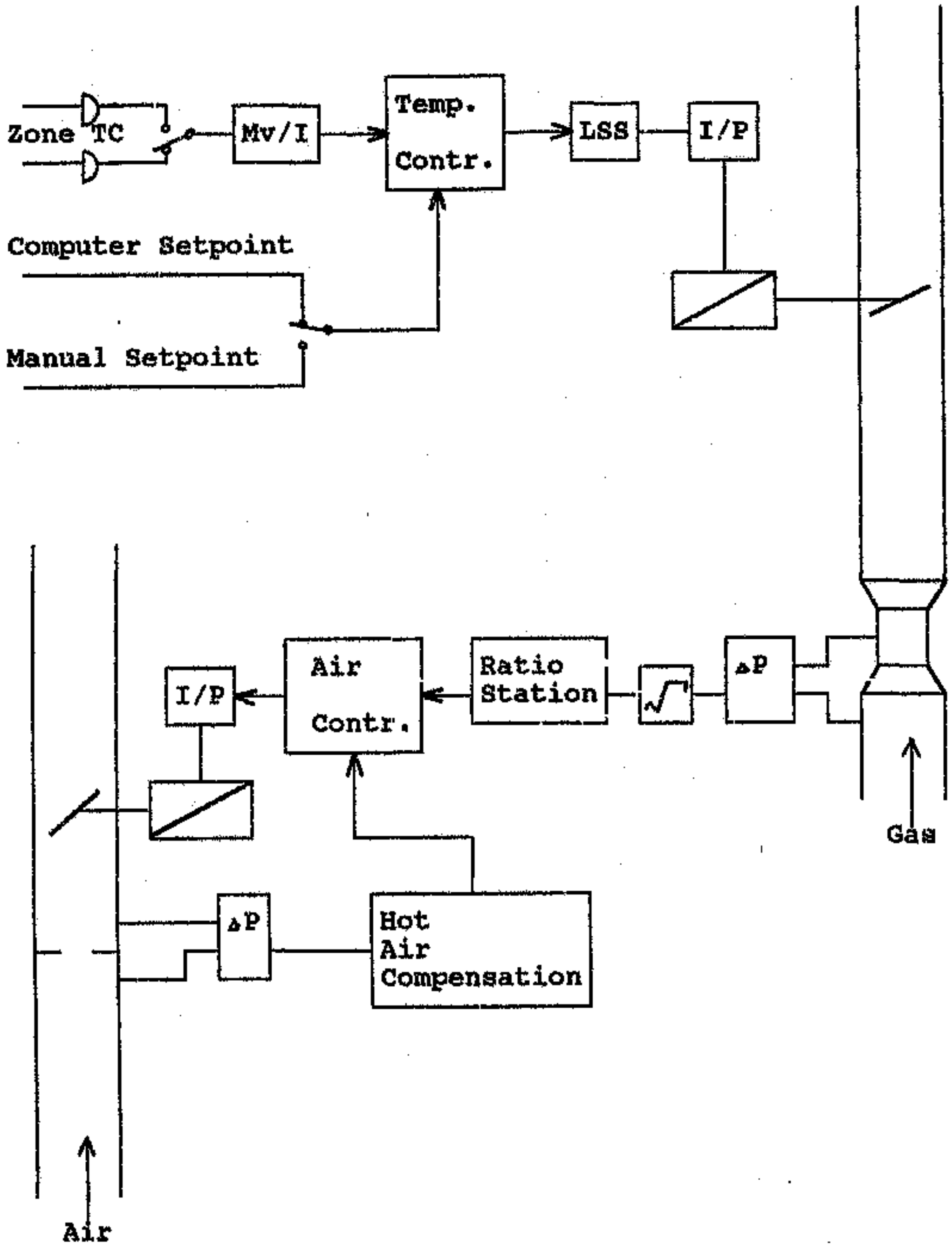


Fig 2.4 Control Diagram of Bottom Heating Zone

cortroller receives its setpoint from the ratio station which is set to a fixed ratio. The ratio station in turn receives the gas flow signal via the flow sensor. This sensor consists of a venturi tube together with a

Differential Pressure (DP) cell. Root extraction is performed on the signal before it is fed to the ratio station. The measured variable in this loop is the air flow. The controller receives this measurement from the mass flow computer which compensates for the hct air temperature. The air flow sensor in this case is an orifice plate in conjunction with a DP cell. The output of the controller is then again sent to the butterfly valve in the air pipe via a current to pressure converter. This time the characteristic of the valve together with its piston is logarithmic. The original linear response was changed to logarithmic because the response of the air flow was very slow and resulted in inefficient burning.

CHAPTER 3 Modelling the Furnace

This chapter discusses the steps followed in developing a dynamic model for the furnace. The resulting model is then executed and evaluated using a digital computer simulation package.

3.1 Logging Data

The first step in the modelling of the furnace was to log real data. As no suitable software was available that would function as desired with the A/D converter plug-in card for the IBM PC computer, it was necessary to develop the necessary routines. A routine was required that allowed the logging of all sixteen channels for several days. The data had to be stored on hard disk in a format that could be read by the later developed graphics routine. The logging routine also had to allow the setting of the logging interval to a precise value. Once this software had been completed the logging of real data was started.

3.1.1 Development of the Logging Software

A sixteen channel A/D converter plug-in card with four D/A channels was purchased. This card was plugged into a portable IBM XT compatible computer. The software that was delivered with this card was unfortunately not suitable. It was therefore necessary to develop the required routines.

After looking at recorded data of the furnace it was seen that the rise time of the system was about two minutes. A sampling period of a tenth of the rise time is usually sufficient. This would result in a twelve second sampling period. It was decided to log at five second intervals which easily satisfied this condition. The Nyquist criteria states that the sampling frequency must be at least twice that of the highest frequency signal in the system. The furnace is much slower than ten seconds and the five second sampling rate therefore comfortably satisfies the Nyquist criteria.

A program thus had to be written that would log sixteen channels at a sampling rate of five seconds. This program allows the user to start the logging process after any key is depressed. It then displays the time every five seconds so that the operators can see that logging is in progress and therefore do not interfere. Every hour all the data is automatically saved in one file on the hard disk and logging is continued. Data can thus be logged over twenty four hour periods and more. When any key is depressed on the keyboard the data logged until that instant is saved and the program halts. The portable keyboard can be locked so that nobody without the key can stop the logging. This program was written in Turbo Pascal and is given in Appendix C.

The logged data was then saved on disk. It was, however, necessary to develop another routine that would display and print the data. This program was much more involved than the first. Figure 3.1 shows the main menu available and gives an indication of all its features.

The user first has to enter the number of hours over which he has logged data and the number of samples taken per hour before he reaches the main menu. Here he has the option of displaying all channels one at a time for all the logged hours. Alternatively he can choose the channels he wants to display and whether he wants them all on one graph or separately. The time period that

```
<1> Display All Hours Maximum Range

<2> Display All Hours Autoscaled

<3> Display All Hours at Given Range

<4> Display Selected Hours Maximum Range

<5> Display Selected Hours Autoscaled

<6> Display Selected Hours at Given Range


<C>hannel Selection

<O>verlapping    Or    <S>eparate Graphs

<A>ctivate       Or    <D>eactivate Printing Option?


<X> Exit Program
```

Fig 3.1 Main Menu of Graphics Routine

needs to be displayed can also be chosen.

Another feature of the program is that the user can manipulate the Y-axis range. He has three options. The first called maximum range selects the range 0 to 5000 (i.e. 0mV to 5000mV). The Autoscale option first searches through the selected channels and finds the minimum and maximum value. If the difference between these two is larger than 1000 these two limits become the Y-axis minima and maxima respectively. If this difference is smaller than 1000 the program increases it to 1000 by adding to the maximum a calculated amount and subtracting the same amount from the minimum. The last option allows the user to enter the Y-axis minima and maxima. Here the program will automatically ignore any value exceeding these limits.

This program automatically senses the type of screen and allows the user to print the graphs in either draft or high resolution quality. A separate procedure that would read the screen pixel by pixel and then fire the printer needles directly had to be developed because the DOS graphics program only allows the graphics screen printing on a CGA screen. However, a Hercules graphics card with a monochrome screen was used for this project.

When all channels are selected to be displayed separately the program generates the correct headings. However, in all other cases the user can choose the heading. The program generates its own file names unless the user specifies that one hundred hours were logged in which case it will prompt for the data filename.

The furnace characteristics such as rise time, disturbances etc. had to be determined from the graphs generated from the logged data and it was thus necessary to program a routine with all the above features. A printout of this program is given in Appendix D.

3.1.2 The Data Logging Experiments

It was necessary to check the accuracy of the instrumentation of the whole furnace to ensure that the logged readings were reliable. The furnace had been on a shutdown for maintenance recently and everything was therefore calibrated. In order to use the available sixteen channels most effectively it was decided to log the following signals:

0 - Soak Zone West Setpoint
1 - Soak Zone West Temperature
2 - Soak Zone East Setpoint
3 - Soak Zone East Temperature
4 - Top Heating Zone Setpoint
5 - Top Heating Zone Temperature
6 - Bottom Heating Zone Setpoint
7 - Bottom Heating Zone Temperature
8 - Top Preheating Zone Setpoint
9 - Top Preheating Zone Temperature
10 - Bottom Preheating Zone Setpoint
11 - Bottom Preheating Zone Temperature
12 - Furnace Pressure
13 - Main Line Coke Oven Gas Pressure
14 - Bottom Heating Zone Air Flow
15 - Bottom Heating Zone Gas Flow

A problem occurred when connecting the A/D card to the instrumentation equipment. Although the DC voltage levels between the instrumentation equipment grounds and computer ground were checked to be zero the A/D card was damaged. The fault was later found to be a 110V AC difference between the two grounds. The computer's earth had been disconnected. Re-connecting the earth solved the problem. After the card was repaired the logging could be continued.

Long logging sessions were run first to see if any useful data could be logged in such a way. This method least affected plant production. It was, however, found that not much could be done with this data. Only small steps took place during this time which were too small in relation to the disturbances. A single step in only one zone did not occur in any of the long run sessions and the effects of the zones on each other could therefore not be determined.

Specific tests had to be run. These, however, affect production and it was therefore necessary to wait for a time when these tests had the least adverse effect on production. For this purpose the furnace had to be on manual setpoint. Sometimes months went by before any tests could be performed.

The duration of a typical test was in the region of one hour. During this period the conditions had to be ideal i.e. no interferences. This was however seldom the case and the tests therefore had to be repeated numerou times before satisfactory results were obtained. The bigger the step into the system the better the resulting data would be. This is due to the fact that usually the amplitude of the noise and disturbances stays constant and a bigger step therefore reduces the modelling error. However, when stepping temperature setpoints great care had to be taken to ensure that the temperature would not become too high thus damaging the furnace.

It was observed that the Bottom Heating Zone was not affected by temperature variations in the other zones. The push rate of the slabs through the furnace affects all zones. A dip in temperature could usually be noticed in all zones whenever a new cold slab was pushed into the furnace. Enough useful data was eventually logged to model the Bottom Heating Zone of the furnace.

3.2 Discussion of the Results

The first set of tests are used to establish whether a step in one of the other furnace zones has an effect on the Bottom Heating Zone. Subsequent tests are then performed to characterize the Bottom Heating Zone.

3.2.1 Stepping the Soak Zones

The Bottom Heating Zone is switched to manual control by pushing the manual button on the temperature controller. This effectively puts it into an open loop configuration. Next both Soaking Zones are stepped by 40°C by increasing the temperature controller's setpoint by that amount. The resulting graph is given in figure 3.2.

It is believed that the peak in the temperature of the Top Heating Zone was caused by the stepping of the Soak Zones since they are adjacent to each other. The doors of the furnace were opened for a short period shortly after the step but could not have caused such a large effect on the Top Heating Zone. However the zone we are actually interested in, the Bottom Heating Zone, was not at all affected by this step. A slab was taken out fifteen minutes after the start of the test. A small dip in the Bottom Heating Zone Temperature could be seen. The two preheating zones showed an even larger dip. Fourteen minutes later another slab was taken out and large dips in temperatures were noted in all zones accept the Bottom Heating Zone which is not affected. It can therefore be concluded that the Soak Zones have a negligible effect on the Bottom Heating Zone.

3.2.2 Stepping the Top Preheating Zone

It was possible to step this zone by a 100°C because the steady state temperature at that point was quite low. The resulting graph is shown in Figure 3.3. All other zones show a slight dip in temperature which might be due to the pressure drop in the main gas line caused by this very large step. The drop in temperature in the Bottom Heating Zone, however, is so small that it can be

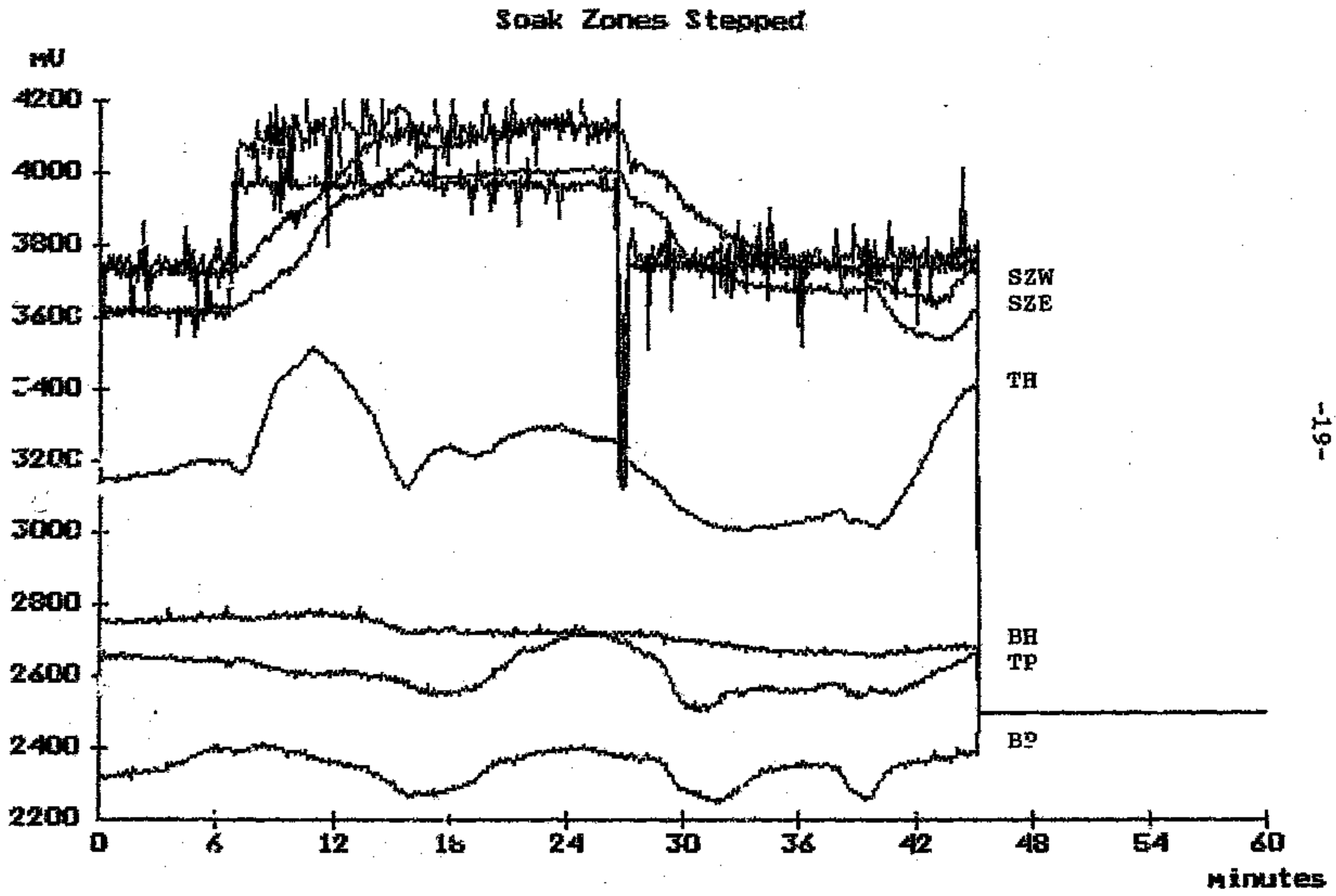


Fig 3.2 Soak Zones Stepped

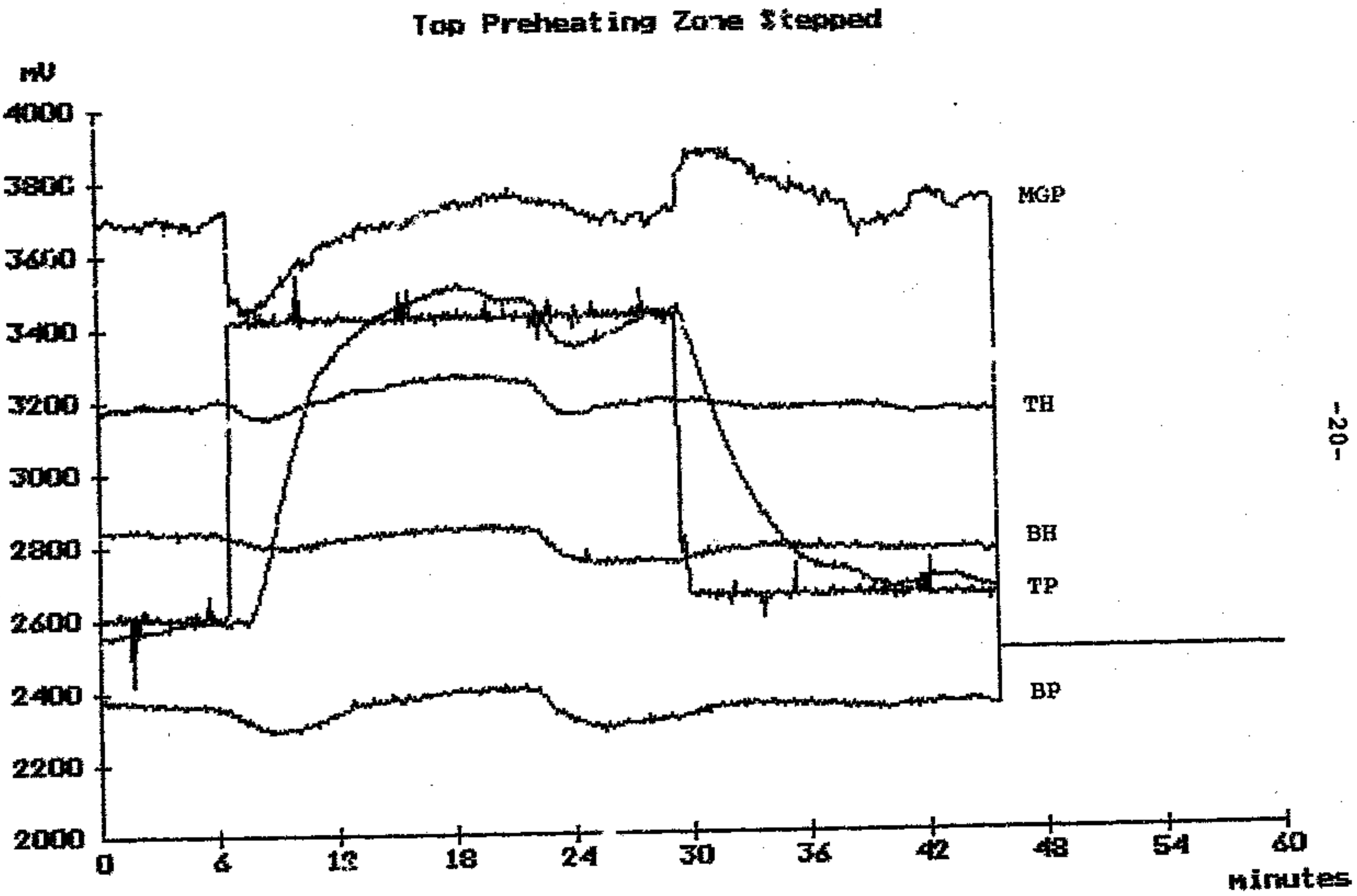


Fig 3.3 Top Preheating Zone Stepped

neglected. Twenty two minutes after the start of the test there is another drop in temperature, larger than the first. This drop is caused by a cold slab that is pushed into the furnace at that time. From this test it can again be concluded that the Top Preheating Zone has a negligible effect on the Bottom Heating Zone.

3.2.3 Stepping the Botton Preheating Zone

The Bottom Preheating Zone is adjacent to the Bottom Heating Zone. Again it was possible to use a large 100°C step. This time no temperature drop is noted in the other zones although the same pressure drop is noted in the main gas line. This suggests that the drops in temperatures after the Top Preheating Zone was stepped were not caused by the main gas line pressure drop. The Top Heating Zone again displays a peak after the step input to the Bottom Preheating Zones. At the end of this test, at 39 minutes, a slab is taken out and this results in small dips in zone temperatures. However, this test again shows that the Bottom Preheating Zone is unaffected by the Bottom Heating Zone. Figure 3.4 shows the results of this test.

3.2.4 Stepping the Top Heating Zone

If any of the Zones would affect the Bottom Heating Zone one would expect it to be the Top Heating Zone, because it is directly above. From the previous tests it can be seen that this zone is the one that is affected most by all other zones. Should the Top Heating Zone affect the Bottom Heating one would expect all these interferences to filter through to the Bottom Heating Zone. It is therefore important to determine the interaction between these two zones.

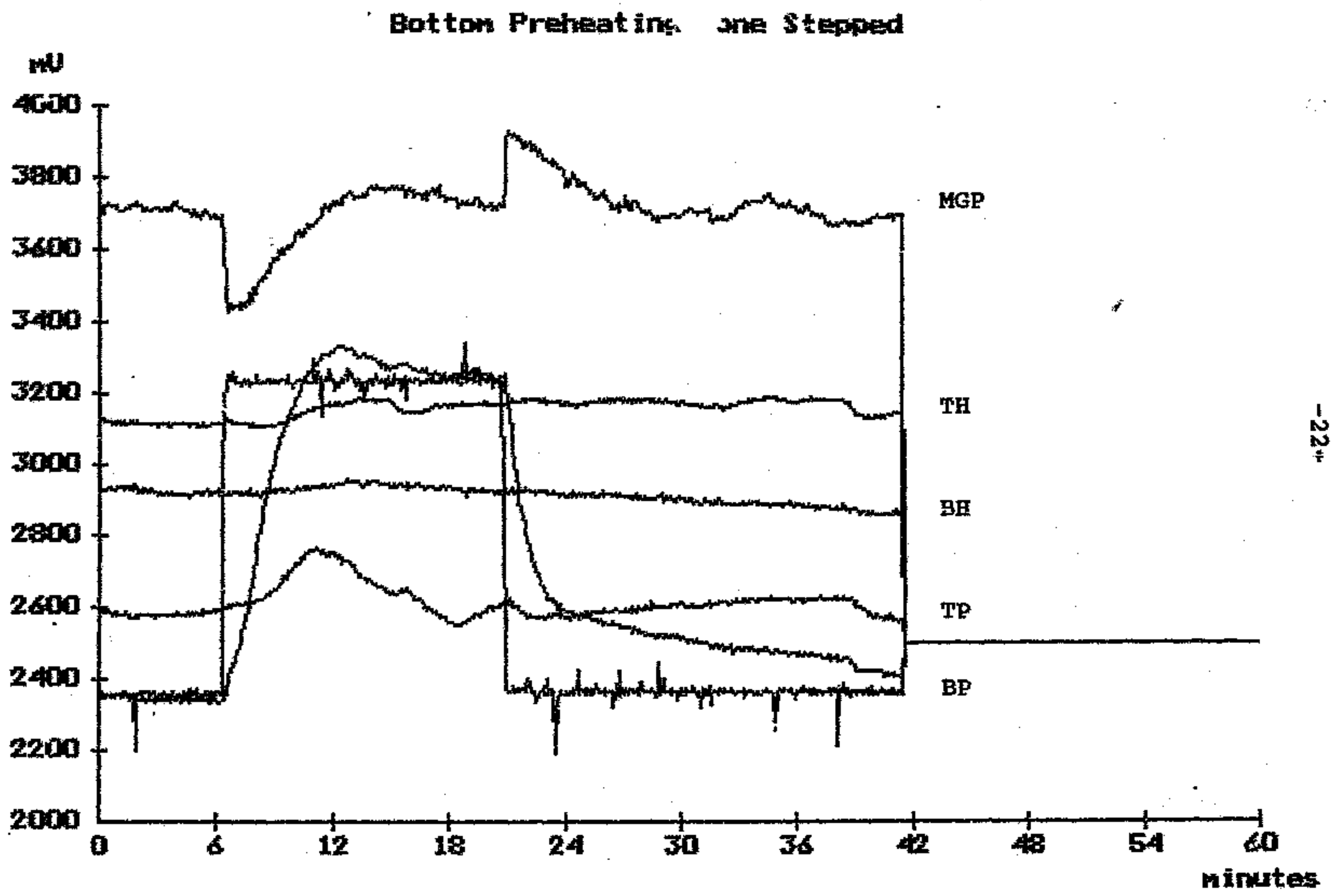


Fig 3.4 Bottom Preheating Zone Stepped

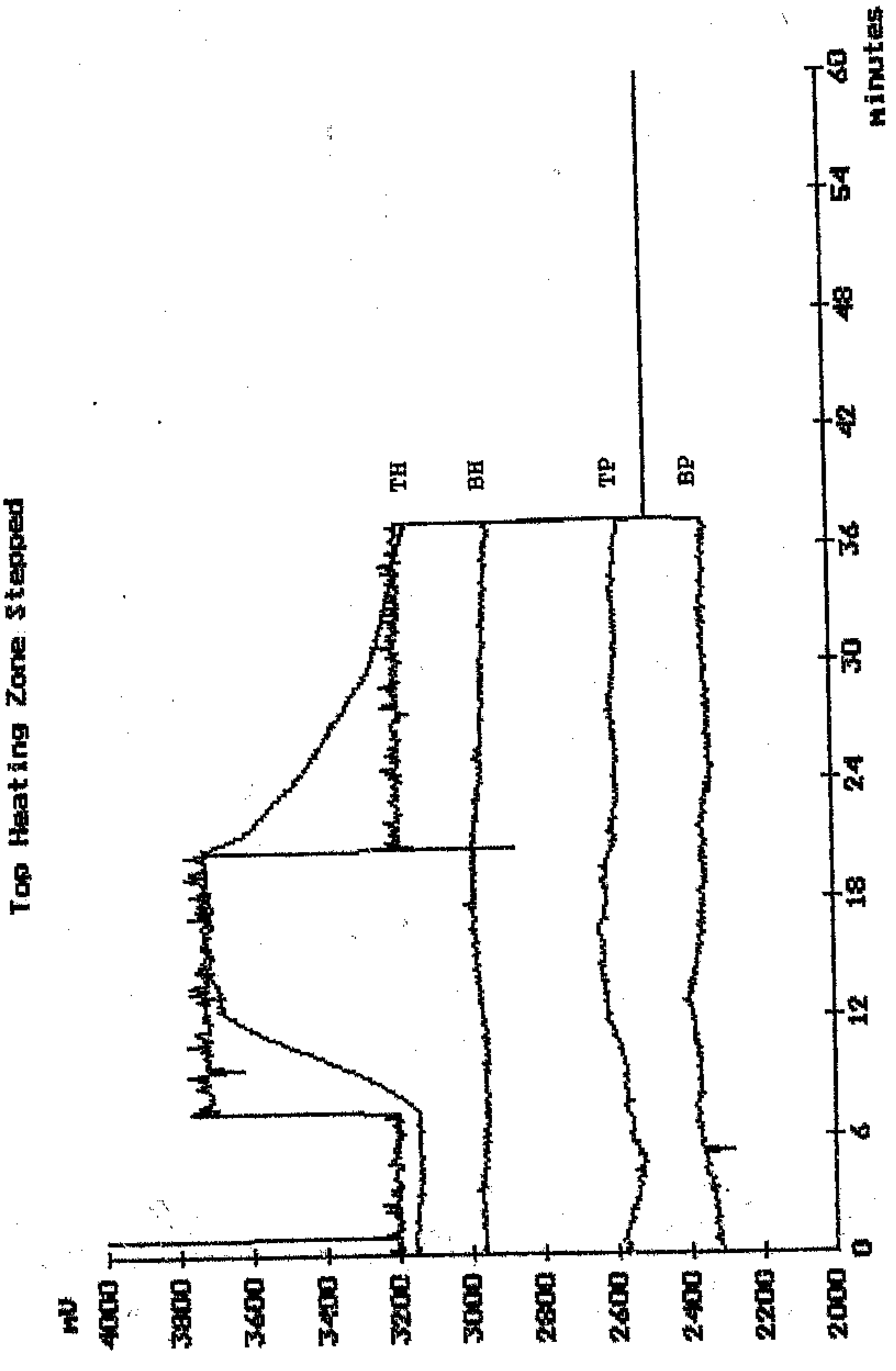


Fig. 3.5 Top Heating Zone Stepped

Figure 3.5 shows the result of a 60°C step into the Top Heating Zone. No slabs were taken in or out during this test and it can bo seen that the Bottom Heating Zone stays at a constant temperature.

After all these tests it can therefore be concluded that the Bottom Heating Zone is not noticeably effected by any other zone.

3.2.5 Performing a Closed Loop Test on the Bottom Heating Zone

The results of this test can be seen in Figure 3.6. During the test the settings for the temperature controller were as follows:

P : 80%
I : 120sec
D : disabled

while the combustion air controller was set as follows:

P : 250%
I : 32sec
D : disabled

A 100°C step was used as the test signal. The results show that the gas flow increases much faster than the air flow although they are set at a fixed ratio. The access gas causes a black flame which is not desirable since it indicates inefficient burning.

The slabs inside the furnace are at their desired temperature throughout the test. No cold slabs are inserted during the test and therefore no heat can be absorbed by a cold slab as is the case under normal

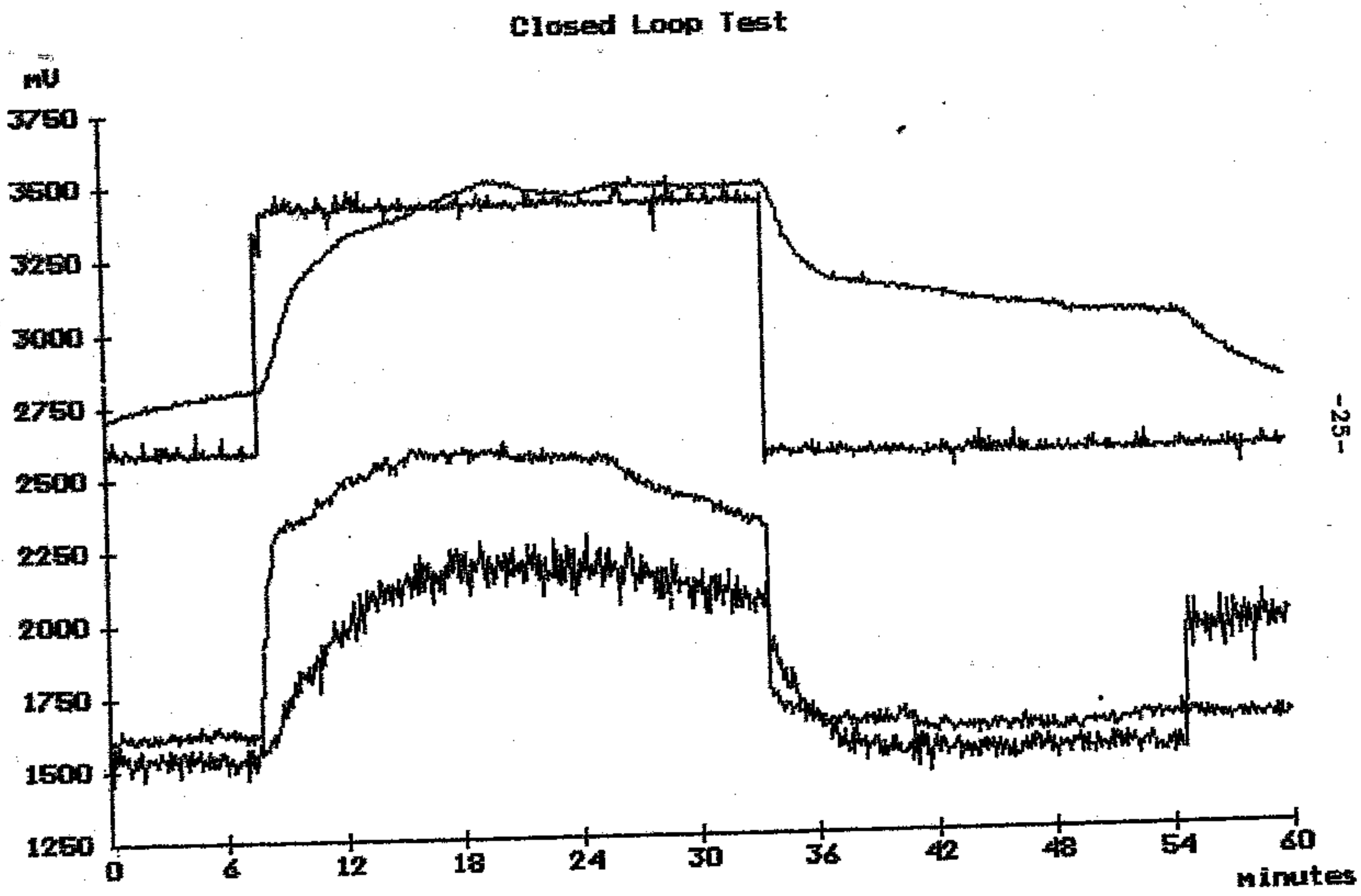


Fig. 3.6 Closed Loop Test

running conditions. This explains why the zone looses temperature very slowly after it is stepped down again. The pilot flames are set too high and the gas and air therefore stay at around 15% instead of the required 4% which furthermore decreases the rate at which the Bottom Heating Zone can loose its heat. This test is not good but displays the characteristics of the Bottom Heating Zone. The open loop test that follows will be more useful.

3.2.6 Performing an Open Loop Test on the Bottom Heating Zone

In order to perform an open loop test the temperature controller is switched into manual by pushing the manual button on the controller. The butterfly valve controlling the coke oven gas flow can now be operated directly from the controller faceplate. The combustion air controller kept the same PID settings as it had in the closed loop test.

The gas was then stepped from about 15% to 30% of maximum flow. There were difficulties in obtaining a steady 30% because the temperature controller is of the old type requiring a dial to be turned in order to change the manipulated variable. After the dial is turned the result is checked on the chart recorder. This is more accurate than the analog display on the controller. This is a time consuming procedure which resulted in a few readjustments because the gas flow was too high after the very first setting and would have caused the furnace to overheat. A small peak followed by a dip in the gas flow graph is therefore seen.

The resulting graph is given in figure 3.7. It is interesting to see that the combustion air flow is much slower than the gas. During the step-up stage the air to

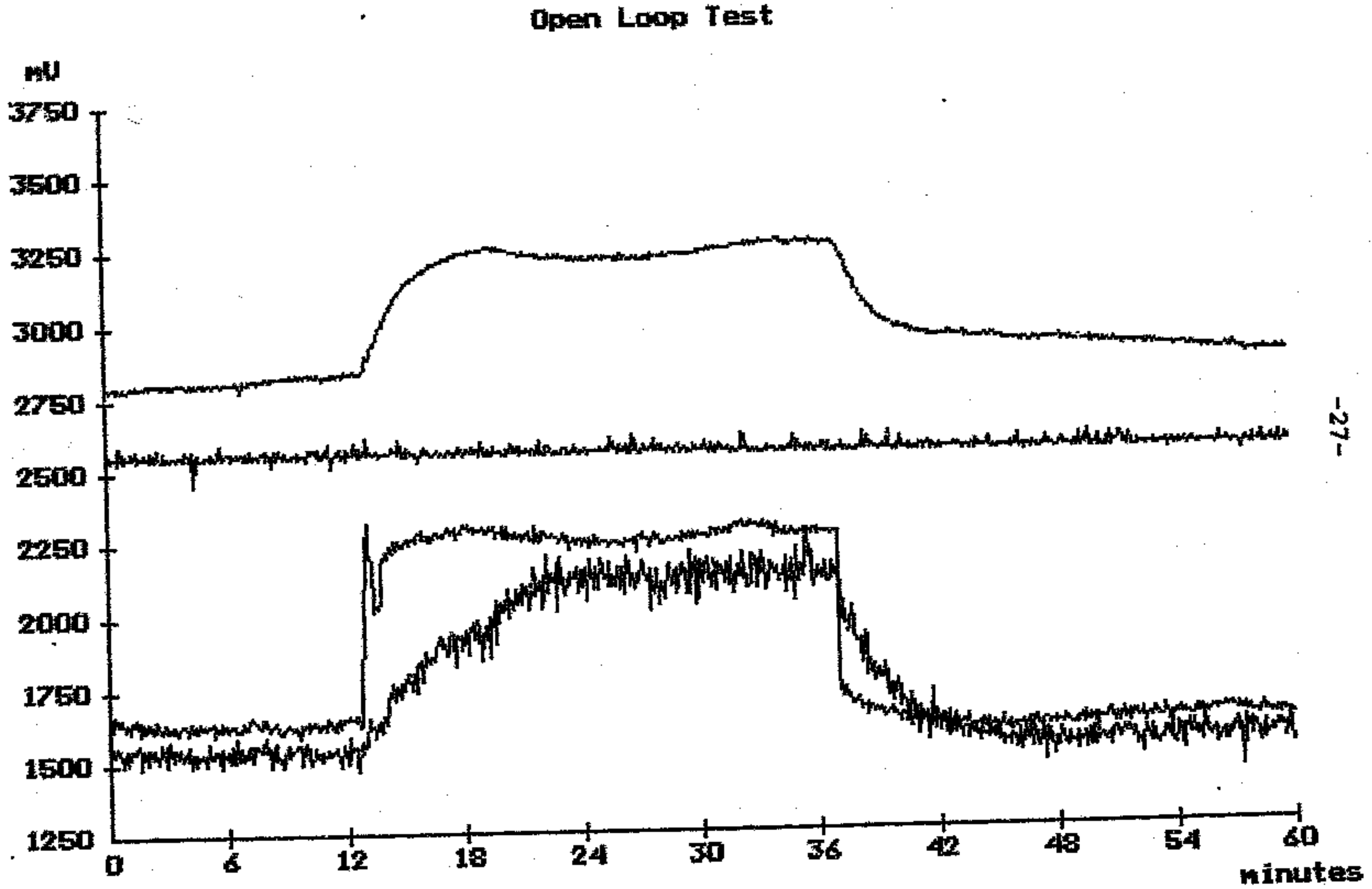


Fig. 3.7 Open Loop Test

gas ratio becomes so small that there is no enough air to burn all the available gas. The air flow therefore determines the amount of heat generated by the burners during this phase. When the gas is stepped down the air to gas ratio exceeds one. This results in more heat leaving the furnace via the exhaust gases than is necessary via the unused combustion air.

This graph was used extensively in the modelling stage. The step response displays the same characteristics as a first order system and we therefore generate a first order model for the Bottom Heating Zone. All the steady state values and the rise time were determined from this graph and will be discussed in the following section.

3.3 Derivation of the Mathematical Model

In order to model the furnace the energy balance had to be determined. Figure 3.8 shows the different energy gains and losses of the Bottom Heat Zone.

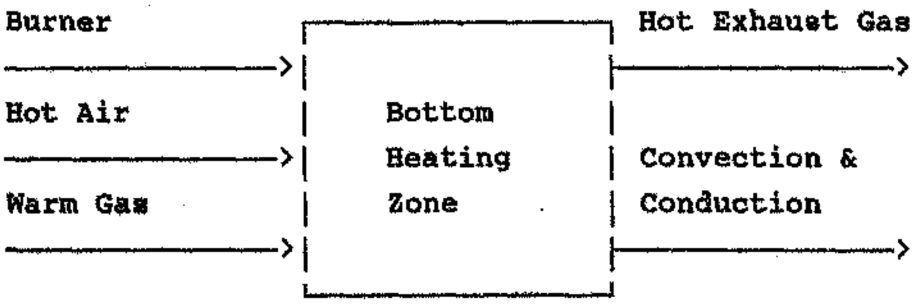


Fig. 3.8 Energies in the Bottom Heating Zone

Due to the open loop test characteristics we assume a first order system and get the following equation:

$$
\begin{aligned}
C*dT/dt = {} & f(a,g) - k1 * T - k2 * e * Te + k3 * g * Tg \\
& + k4 * a * Ta \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (1)
\end{aligned}
$$

where:

C = total heat capacity
T = zone temperature
f(a,g) = energy generated by the burners
a = combustion air flow
g = coke oven gas flow
e = exhaust gas flow
k1 = heat capacity of furnace walls and slabs
k2 = specific heat of the exhaust gas
k3 = specific heat of the coke oven gas
k4 = anecific heat of the combustion air
Te = exhaust gas temperature
Tg = coke oven gas temperature
Ta = combustion air temperature

The objective now is to determine all the constants so that an equation involving three variables namely, a, g and T can be obtained.

The three temperatures Te, Tg and Ta are assumed to be constant because they only change slightly and therefore affect the furnace negligibly. They are as follows:

Te = 900°C
Tg = 35°C
Ta = 400°C

Tn order to determine the constants k2, k3 and K4 we have to look at the chemical reactions that takes place whan the combustion air combines with the coke oven gas to generate heat.

Composition of Coke Oven Gas:

H_2 : 55.6%
CH_4 : 24.1%

CO	: 9.4%
N_2	: 2.8%
CO_2	: 2.7%
C_2H_4	: 3.0%
C_2H_6	: 0.8%
C_3H_8	: 0.3%
O_2	: 0.1%

Composition of Combustion Air:

N_2	: 79%
O_2	: 21%

Therefore the main heat generating reactions are:

$$H_2 + \tfrac{1}{2}O_2 \longrightarrow H_2O(g)$$
$$CH_4 + 2O_2 \longrightarrow CO_2 + 2H_2O(g)$$
$$CO + \tfrac{1}{2}O_2 \longrightarrow CO_2$$

From the above we determine that

$0.56 * 1 + 0.24 * 1 + 0.09 * 1 = 0.890$ moles of coke oven gas

combine with

$0.56 * \tfrac{1}{2} + 0.24 * 2 + 0.09 * \tfrac{1}{2} = 0.805$ moles of oxygen

to form

$0.56 * 1 + 0.24 * (1 + 2) + 0.09 * (1) = 1.37$ moles of product

Now for each mole of oxygen we have 79/21 = 3.762 moles of nitrogen. In the above reaction we use 0.805 moles of oxygen and we must therefore add 0.805 * 3.762 = 3.03 moles of nitrogen. This nitrogen does not react and we therefore find it on both sides of the reaction. This results in:

1.37 product + 3.03 nitrogen = 4.40 moles of exhaust.

The composition of the exhaust gas is therefore as follows:

(0.56 + 2 * 0.24) / 4.40 * 100 = 24% H_2O
(0.24 + 0.09) / 4.40 *100 = 8% CO_2
3.03 / 4.40 * 100 = 69% N_2

We can now determine k2, k3 and k4. The specific heats for the various compounds and elements are given in Appendix E.

0.24 * 0.403 + 0.08 * 0.525 + 0.69 * 0.331 = 0.367
——> k2 = 0.367 kcal/m^3°C

0.56 * 0.306 + 0.24 * 0.372 + 0.09 * 0.311 + 0.03 * 0.311 + 0.03 * 0.387 + 0.03 + 0.444 + 0.01 * 0.532 = 0.328

——> k3 = 0.328 kcal/m^3°C

And k4 = 0.318 kcal/m^3°C

The next step is to express the exhaust gas flow e in terms of a and g. From previous calculations we know:
0.890 moles gas + 0.805 moles oxygen ——> 1.37 moles product

——> 1.695 moles in result in 1.37 moles out
——> Ratio = 1.37 / 1695 = 0.808

——> e = 0.808 * (g + 0.805 / 0.890 * g)
+ (a - 0.805 . 0.890 * g)

⟶ $\underline{e = 0.634 * g + a}$

We now have to find a function that describes the heat generation of the burners as a function of a and g. The calorific value of the coke oven gas is 4395 kcal/m^3. Assuming a surplus of oxygen we get:

$$f(a,g) = 4395 * g$$

If there is not enough oxygen the combustion air flow will determine how much of the gas is burned. In this case we get:

$$f(a,g) = 4395 * (0.21 * a * 0.890 / 0.805) = 1020 * a$$

The critical air to gas ratio at which there is just enough air to burn all the available gas is as follows:

$$R_{crit} = 0.805 / 0.890 / 0.21 = 4.307$$

⟶ $\underline{\text{for } (a / g) \geq 4.307 \quad f(a,g) = 4395 * g}$
& $\underline{\text{for } (a / g) < 4.307 \quad f(a,g) = 1020 * a}$

Substituting the above constants we can now rewrite equation (1) as follows:

$$C * dT/dt = f(a,g) - k1 * T - 0.367 * (0.634 * g + a) * 900 + 0.328 * g * 35 + 0.318 * a * 400 \qquad \ldots\ldots\ldots\ldots\ldots(2)$$

The above equation would hold if the furnace displayed linear characteristics from 0°C to 1400°C. This is, however, not the case. The furnace displays linear characteristics in its normal operating range from about 1100°C to about 1250°C. We therefore add an offset of K_{off} to the temperature. Therefore actual zone temperature is $K_{off} + T$. We can determine two steady state conditions from the open loop test graph. The first is at the 24 minutes interval. At that instant we obtain the following values:

$T_{1ss} = 1179°C$
$g_{1ss} = 0.521\ m^3/s$
$a_{1ss} = 2.719\ m^3/s$

The other steady state that we will use is at the 6 minute mark on the same graph. The values are as follows:

$T_{2ss} = 1129°C$
$g_{2ss} = 0.264\ m^3/s$
$a_{2ss} = 1.228\ m^3/s$

When the system is in steady state condition the dT/dt term is equal to zero.

$$\Longrightarrow k1 * T_1 = f_{ss}(a_{ss}, g_{ss}) - 0.367 * (0.634 * g_{ss} + a_{ss}) * 900 + 0.328 * g_{ss} * 35 + 0.318 * a_{ss} * 400$$

Now solving for the first steady state condition:

$a_{1ss}/g_{1ss} = 5.219$ which is greater than 4.307

$\Longrightarrow f_{ss} = 4395 * 0.521 = 2289.80$ kcal/°Cs

$\Longrightarrow k1 = 1634.44 / T_1$ kcal/s

Now because of T_{oss} we have:

$T_{1ss} = T_1 + T_{oss} \Longrightarrow T_1 = T_{1ss} - T_{oss}$

$\Longrightarrow k1 = 1634.44 / (1179 - T_{oss})$ kcal/s ....(3)

Similarly for the second steady state we get:

k1 = 858.62 / (1129 - T_{oss}) kcal/s ....(4)

Equating (3) and (4) and solving for T_{oss}:

——> T_{oss} = (1634.44*1129 - 858.62*1179) / (1634.44 - 858.62)

——> T_{oss} = 1073.66°C

We therefore choose T_{oss} = 1075°C which is a "nice" number.

Using equation (3) we get:

<u>k1 = 15.716 kcal/s</u>

The last constant that still has to be determined is C. We can simplify our equation as follows:

C/k1 dT/dt + T = constant

——> C/k1 = τ where τ is the time constant

τ can be read off the open loop test graph as 92 seconds.

——> C = 92 * 15.716

——> <u>C = 1445.87 kcal</u>

The resulting model of the bottom heating zone is thus as follows:

T_{zone} = T + 1075 °C

where T is solved by

$$dT/dt = \frac{(f(a,g) - 15.716 * T - 197.93 * g - 203.1 * a)}{1445.87}$$

where

$$f(a,g) = \begin{cases} 4395 * g & \text{for } (a / g) \geq 4.307 \\ 1020 * a & \text{for } (a / g) < 4.307 \end{cases}$$

3.4 Evaluating the Derived Model

In order to evaluate the model a program is required that will run the model, generate graphs and print the results. The University of the Witwatersrand supplied me with some short pascal routines that supplied the basic building blocks for the simulation. A slightly changed version of the graphics routine developed for the logging part of the thesis was incorporated into the simulation routine. A printout of the simulation routines can be found in Appendix F.

Figure 3.9 shows the real plant response to a step against the model generated output to the same magnitude step. The graphs showing perfectly straight lines are the model graphs and the ones with fuzzy lines are the plant graphs. The bottom two graphs are the air and gas flow and the top graph is the zone temperature.

The plant graph displays a slight overshoot before it settles to the same value as the calculated model output. Since our assumption was that the furnace can be modelled adequately by a first order system, we do not see any overshoot in the model's response. However, the model displays the same rise time of 92 seconds.

During the step down phase the model is faster at the

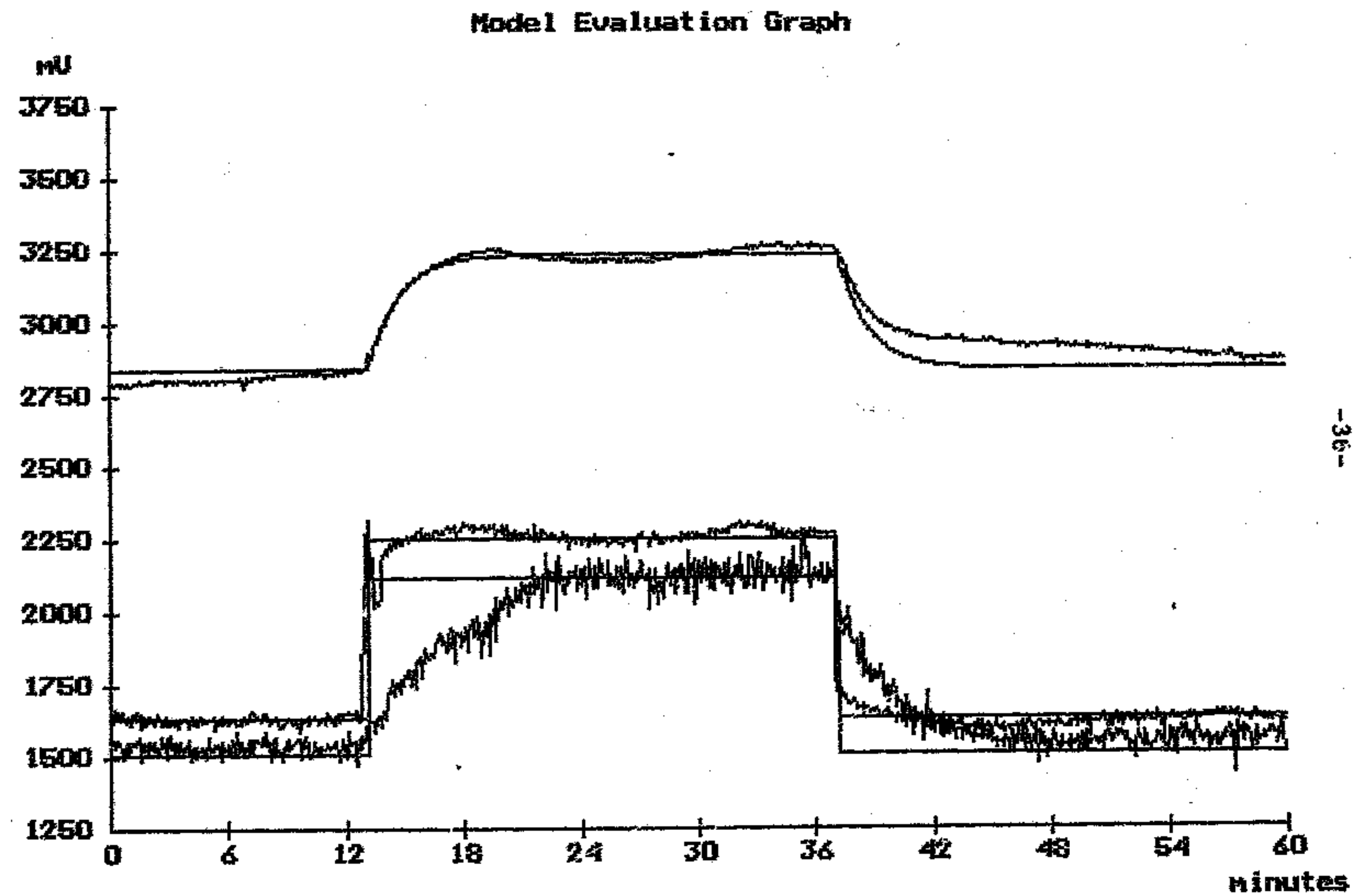


Fig. 3.9 Model Evaluation Graph with Air and Gas Stepped

end of the step. In the section on the open loop test it was explained that the pilot flames were incorrectly set and therefore the furnace only cools very slowly at the end of the step. In addition it was found that the furnace does not display the same rise times when stepped up and down. We can therefore conclude that the furnace has some non linear characteristics. Taking everything into account until this point we can resolve that the above model is a good one.

Figure 3.9 shows that the air flow curve of the model does not fit the one from the plant at all. The reason for this is that one could only step the gas flow on the plant while the air flow had to stay in automatic mode. When deriving the mathematical model it was assumed that both gas and air were stepped. The graphs in figure 3.9 prove that in such a case the model would be excellent. However, we are working here on a real furnace whose input air flow curve resembles a ramp rather than a step. Figure 3.10 shows the result when the derived model is subjected to air and gas flow curves similar to those of the furnace during the logging stage.

Comparing the two graphs we see three important differences. The rise time of the new response is 166 seconds which is considerably larger than 92 seconds. The reason for the larger rise time is that in the beginning of the step the air to gas ratio is below the required value of 4.307. There is excess gas that cannot combust due to an oxygen deficiency. The heat generation is therefore governed by the air flow input curve until this ratio is above 4.307 again. The air flow curve resembles a ramp rather than a step and thus results in a longer rise time.

The second important feature is that the step down part

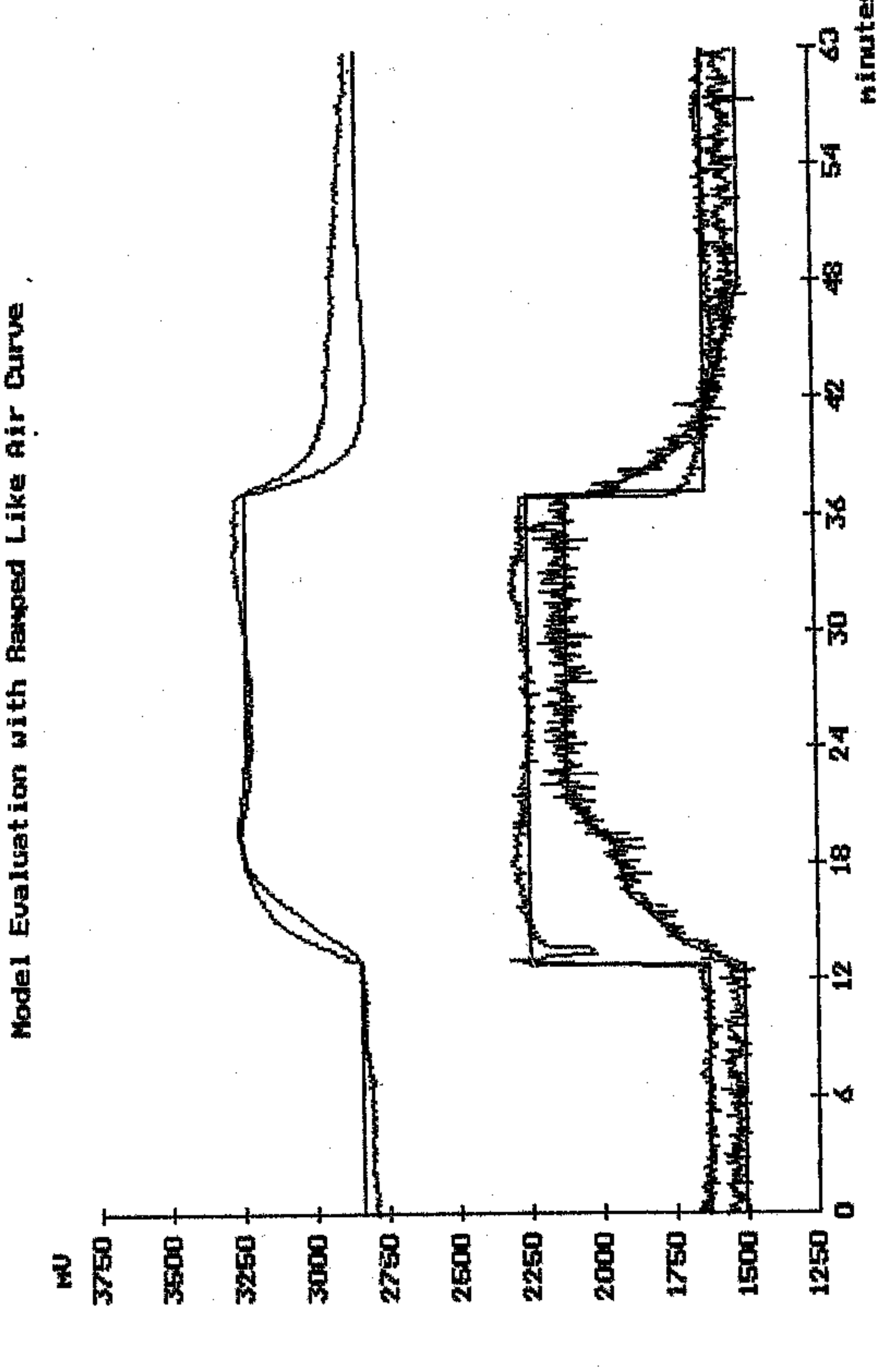


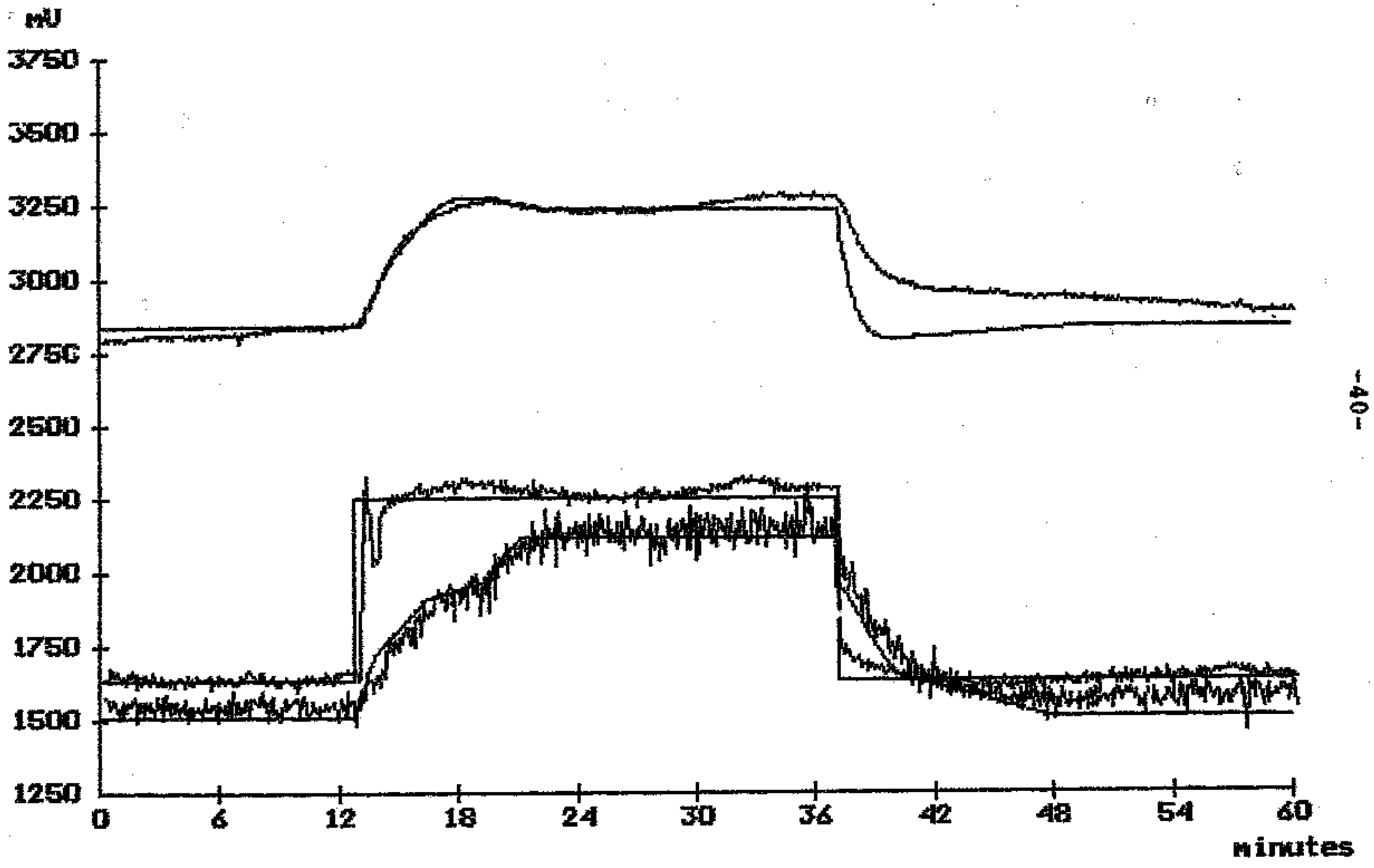
Fig. 3.10 Model Evaluation Graph with Ramped like Air Flow Curve

of the response looks even worse than before. This again is due to the air flow. Again a large section of the step down curve of the air flow is a ramp. Here the air to gas ratio becomes extremely large which results in a large amount of air being heated up to 900°C and then leaving the Bottom Heating Zone. The air is therefore cooling the furnace and results in a faster cooling curve for the model.

The third feature is that the model temperature curve displays a slight overshoot. This suggests that we are introducing second order characteristics by treating the air and gas flows separately. However, when inspecting the model more closely, this overshoot can be explained. The air to gas ratio at a point after the step when all curves have stabilized is 5.22. Since the gas has long stabilized and the air is still busy ramping the burners actually exceed the ideal air to gas ratio of 4.307. At that point the Bottom Heat Zone will reach its highest temperature. The Zone is, however, controlled at a "safer" ratio to ensure that the least amount of gas is wasted. The temperature must therefor decrease again as the air to gas ratio reaches the value of 5.22.

If we want to improve the model's step up response we will have to reduce the rise time to something closer to the initial 92 seconds. The heat capacity C of the Bottom Heating Zone is directly related to the rise time. In figure 3.11 we can see the response to our input curves where the C constant of the model has been changed to 800 Kcal. The resultant rise time is 110 seconds which is much closer to the furnaces 92 seconds.

However, this improvement of the rise time is at the cost of an even greater overshoot and an even faster dropping temparature in the step down phase of the test. The slope of the model's step down curve is now



Model Evaluation with C = 800 Kcal

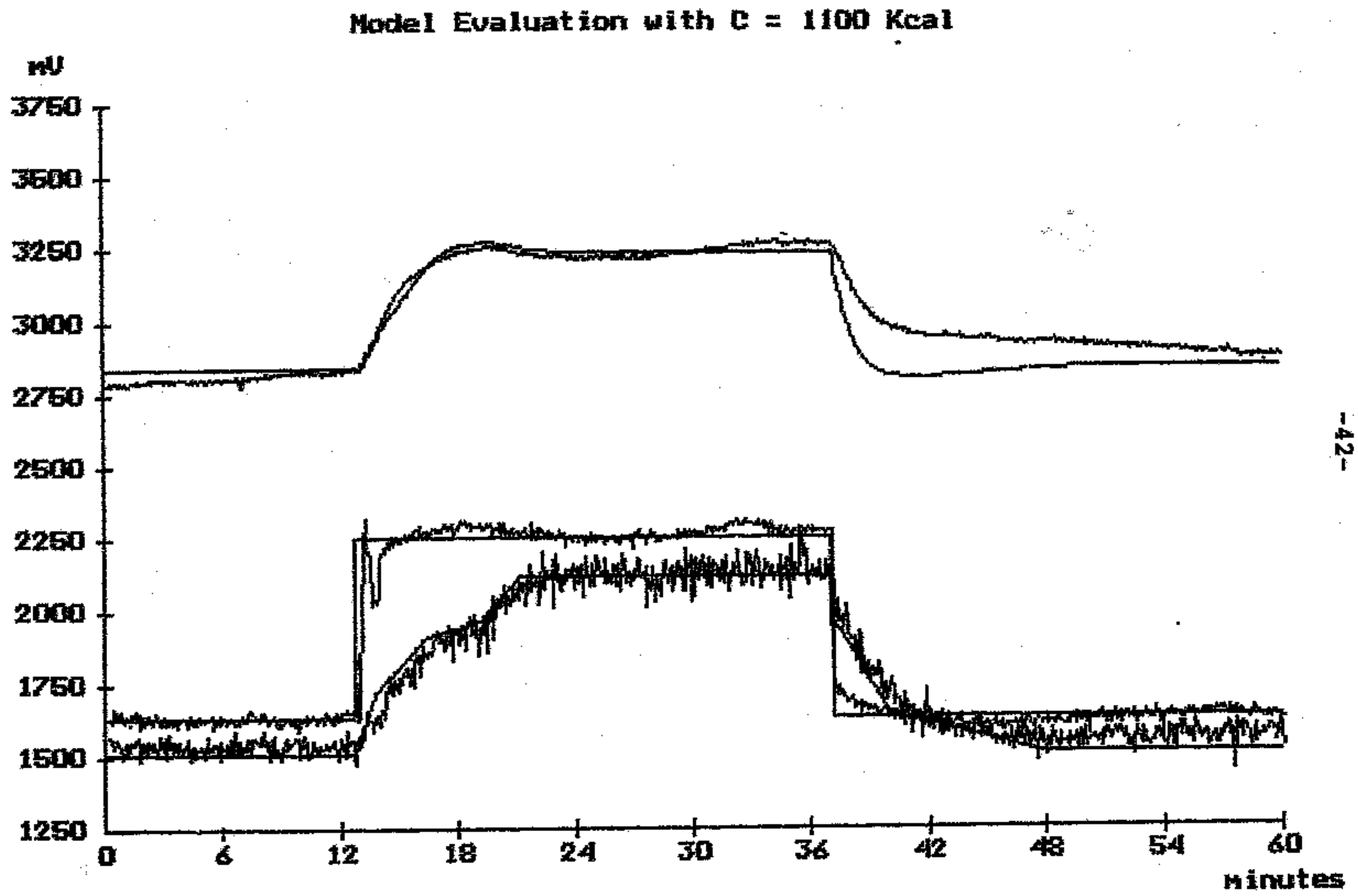
Fig 3.11 Model Evaluation Graph with C = 800 Kcal

considerably larger even in the beginning of the down step.

A compromise between the above two models had to be found. Figure 3.12 shows the response of a model with C equal to 1100 Kcal to the same input gas and air curves as previously. The rise time of the calculated temperature curve is now 129 seconds. The overshoot is smaller again and the step down characteristics are closer to the actual furnace's than the ones in figure 3.11.

We therefore conclude this chapter by noting that our model only models the Bottom Heating Zone approximately. However, the model is still useful since it displays very similar characteristics to the real furnace. What makes this model special is that all variables and constants in equation (2) have a physical meaning. This is important for the later part of this dissertation when disturbances and changes to the furnace will have to be simulated in order to test the adaptive controller.

Fig. 3.12 Model Evaluation Graph with C = 1100 Kcal



CHAPTER 4 The Adaptive Controller

This chapter describes the adaptive controller that is used in this dissertation. It explains the theory behind its design and implementation on an IBM PC compatible using Turbo Pascal.

4.1 Review of Theory

The adaptive controller described in this chapter has been developed by MacLeod and Bergesen (1988). Further modifications by MacLeod resulted in the final version for this dissertation.

A new operator called the Delta or p operator is used. The reason for this is that the q operator and its z transform do not handle fast sampling very well. Under fast sampling unstable zeros are generated using that method. However, introducing the p operator solves this problem. It is defined as follows:

$$p = \frac{q - 1}{T}$$

where q is the usual shift operator and T is the sampling interval.

Another advantage of the delta operator is that at high sampling rates it gives a close correlation between exact continuous and exact discrete transfer function plant models.

The transform corresponding to the delta operator is called the Γ-transform. Figure 4.1 shows the stability region in the Γ-transform. We should note that as T approaches zero i.e. very fast sampling, the left hand

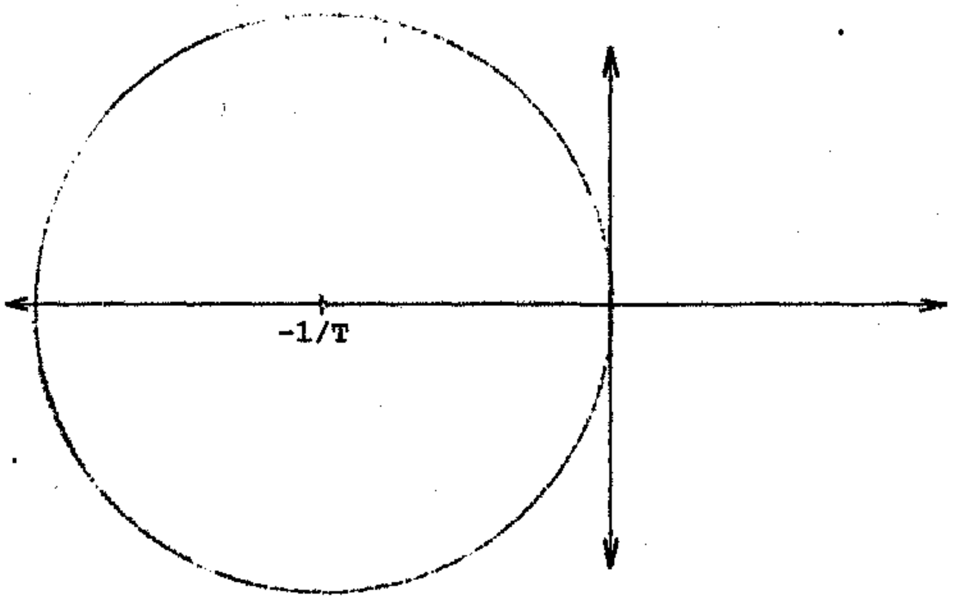


Fig. 4.1 Stability Region in the Γ-Transform

plane becomes the stability zone which is then the same as in the Laplace transform.

The first step in designing the adaptive controller is the development of an estimator. A robust estimator structure is used. We start of with the general equation

$$A(p)y(kt) = B(p)u(kt) + V(kt)$$

For convenience we drop the arguments

$$\Longrightarrow Ay = Bu + V$$

In order to include measurable deterministic disturbances and measurable random disturbances we get

$$Ay = Bu + Fz + d + V \text{ ....} (5)$$

where

$$A = a_0 + a_1p + a_2p^2 + \text{ .... } + p^n$$
$$B = b_0 + b_1p + b_2p^2 + \text{ .... } + p^m$$

$F = f_o + f_1p + f_2p^2 + \ldots\ldots\ldots\ldots + p^n$

z = measurable random disturbance signal

d = measurable deterministic disturbance signal

V = modelling error and noise

The measured disturbance signal z is used to develop an adaptive feed forward block and the measurable deterministic disturbance signal is used to model known waveforms.

The deterministic disturbance d is described by

$$Dd = 0$$

where D is called the deterministic disturbance nulling polynomial and is represented by

$$D = d_o + d_1p + d_2p^2 + \ldots\ldots\ldots\ldots + p^n$$

In order to eliminate the deterministic disturbance d we multiply equation (5) by D.

$$\Longrightarrow ADy = BDu + FDz + DV$$

but D has roots on the stability boundary and woulc thus result in amplifying the noise V at high frequencies and we therefore use

$$D' = p + \epsilon$$

Now dividing through by D' results in

$$\frac{AD}{D'}y = \frac{BD}{D'}u + \frac{FD}{D'}z + \frac{D}{D'}V \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots(6)$$

Note $\frac{D}{D'} = \frac{p}{p + \epsilon}$ which is effectively a High Pass Filter

We also band limit the estimated model by introducing the Low Pass Filter

$$E = e_0 + e_1 p + e_2 p^2 + \ldots\ldots\ldots\ldots + p^n$$

This filter is used to filter the u, y, z and [illegible] signals to eliminate the high frequency components. Equation (6) thus becomes

$$\frac{AD}{D'E}\, y = \frac{BD}{D'E}\, u + \frac{FD}{D'E}\, z + \frac{D}{D\ E}\, V$$

We now define the output of the high pass filter as y'.

$$\Longrightarrow y' = D/L'\ y$$

And reuefi ing the low pass filtered signals as follows

$$y_* = D/(D'E)\ y$$
$$u_* = D/(D'E)\ u$$
$$z_* = D/(D'E)\ z$$
$$v_* = D/(D'E)\ v$$

we get the filtered model

$$Ay_* = Bu_* + Fz_* + v_* \ldots\ldots\ldots\ldots\ldots\ldots\ldots (7)$$

Adding Ey_* to both sides gives

$$Ey_* = Ey_* - Ay_* + Bu_* + Fz_* + \text{[illegible]}$$

Since $y' = Ey_*$

$$\Longrightarrow y' = (E - A)y_* + Bu_* + Fz_* + v_*$$

Ignoring the noise term v_* we can generate a standard regression for y'(k) as follows:

$$y'(k) = \phi(k-1)^T \cdot \tau(k)$$

where

$$\phi(k-1)^T = [y_f(k), \ldots p^{n-1}y_f(k), u_f(k), \ldots p^m u_f(k), z_f(k), \ldots p^r z_f(k)]$$
$$\tau(k) \quad = [e_0-a_0, e_1-a_1, \ldots, e_{n-1}-a_{n-1}, b_0 \ldots b_m, f_0 \ldots f]$$

The above equation can be realized using a recursive least squares algorithm.

Good recursive estimation relies heavily on sufficient excitation of u. As this is not always possible a relative deadzone function is used that ensures that parameters are only updated if u has sufficiently changed. The function is as follows:

$$f(g,e) = \begin{cases} e-g & \text{if } e > g \\ 0 & \text{if } |e| \leq g \\ e+g & \text{if } e \leq -g \end{cases}$$

The relative deadzone is now deveioped to vary the size of g. The function m(k) is defined as follows:

$$m(k) = p_0 m(k-1) + \Omega_0 + \Omega_1|u'(k-1)| + \Omega_2|y'(k-1)| + \Omega_3 z'(k-1)|$$

where

$p_0' \in (0,1)$
$\Omega_0 \geq 0;$
$\Omega_1 \geq 0;$
$\Omega_2 \geq 0;$
$\Omega_3 \geq 0;$
$m_0 \geq 0;$

and choose $p_0 \in (p_0',1)$ and $m(0) = m_0$

so that

$$|n_z(k)| \leq m(k) \text{ for all } k$$

m(k) therefore low pass filters the process input and output signals. It must be larger than the noise term $n_z(k)$.

For a unity gain deadzone filter m(k) becomes

$$m(k) = p_o m(k-1) + (1-p_o)|\Omega_o + \Omega_1|u'(k-1)| + \Omega_2|y'(k-1)| \\ + \Omega_3|z'(k-1)||$$

The deadzone can then be implemented as follows:

$$\beta = \sqrt{\Omega_4} + 1/(1-p)$$

where $\Omega_4 > 0$ and $p \in (0,1)$

We therefore define the magnitude function a(k) as follows:

$$a(k) = \begin{cases} 0 & \text{if } |e(k) < \beta m(k) \\ p \cdot f(e(k);\beta m(k))/e(k) & \text{otherwise} \end{cases}$$

We can now write down the RLS algorithm in two stages in recursive form as follows:

$$\Phi(k) = \Phi(k-1) + a(k) \cdot \frac{P(k-2)\Phi(k-1)}{\tau(k-1)^T P(k-2)\Phi(k-1) + 1} \cdot e(k) \tag{8}$$

$$P(k-1) = P(k-2) - a(k) \cdot \frac{P(k-2)\Phi(k-1)\Phi(k-1)^T P(k-2)}{\Phi(k-1)^T P(k-2)\Phi(k-1) + 1} \tag{9}$$

We have now defined the robust estimator by equations (8) and (9). The above RLS algorithm is numerically implemented by using the UDU^T factorization proposed by Biermann which is more robust.

The robust controller design must now be performed. An explicit closed loop pole assignment algorithm is used. The design is made robust by:

(i) not cancelling the open loop zeros.

(ii) providing for the nulling of deterministic disturbances as well as deterministic setpoint tracking.

(iii) checking whether any of the estimated plant parameters are zero which would not allow a solution of the pole placement equations under normal conditions.

We start of by assuming that setpoint y* satisfies a model of the form

$$Sy^* = 0$$

where S is a monic polynomial of degree p_m and has roots on the stability boundary.

By the Internal Model Principle the control law is

$$LDS.U = Py^* - Gy$$

where

$L = l_0 + l_1p + \dots + p^l$

D = deterministic nulling polynomial as defined previously

$P = p_0 + p_1p + \dots + p^m$

$G = g_0 + g_1p + \dots + p^m$

Now, set $L' = LDS$ and $P = G$

$$\Longrightarrow L'U = Pe \quad \dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots (10)$$

where $e = (y^* - y)$

Furthermore

$$y/y^* = G_{cL}(p)$$

where

$$G_{cL} = \frac{PB/L'A}{1 + PB/L'A} = \frac{PB}{L'A + PB}$$

from the above equation we can see that the closed loop poles are given by

$$L'A + PB = A^* \;\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots(11)$$

where A^* is the desired closed loop characteristic polynomial. The appropriate controller is therefore calculated by solving equation (11) for a given A^*.

The next step in the design is to include a feed forward component in the control signal.

$$\Longrightarrow u = u' - Hz \;\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots(12)$$

where

H is some desired feed forward transfer function
Z is the measured plant disturbance
u' is the control input signal derived from feedback

Now using

$$ADy = BDu + FDz + ADV$$

and multiplying through by LS

$$\Longrightarrow ADLSy = BDLSu + FDLSz + ADLSV$$

Now, since $ADLS = A^* - BP$ (using equation (10))
and $DLSu = P(y^*-y)$ (using equation (11))

we get the following by substituting into equation (12)

$$A^*y = BPy^* + DLS(F - BH)z + ADLSV$$

From the above equation we can see that in order to null the disturbance signal through feed forward we have to choose

$$H = \frac{F}{BD_1}$$

where D_1 is some stable polynomial to make H proper.

The above statement would constrain the polynomial B(p) to be stable. We therefore factorize B as follows

$$B = B^+B^-$$

where B^+ represents all the zeros inside the stability zone and B^- represents all zeros outside the stability zone or zeros corresponding to poorly damped or oscillatory modes. We can therefore design the feed forward transfer function by

$$H = \frac{F}{B^+D_1} \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots(13)$$

which will ensure a stable control signal response. It should be noted that this design will not display perfect disturbance rejection when $B^-<>0$.

4.2 Presentation of Algorithm

The above mentioned algorithm is implemented on an IBM AT compatible computer using a procedure written in Turbo Pascal. A listing of this procedurc is given in Appendix G. The main procedure is called "adaptive controller". In its parameter list it expects the manipulated variable. The adaptive controller routine is called by the simulation program that was described in the previous chapter.

The adaptive controller procedure is divided into three main sub-procedures. They are called apcon1, apcon2 and PIDalg. At this point it should be noted that the procedure apcon0 is also part of the adaptive controller. It is only called once to initialize all the controller variables and therefore is called by the simulation program and not by procedure "adaptive controller".

Apcon1 implements the time critical part of the adaptive controller design. It uses the results from the previous apcon2 calculation. Once apcon1 has finished its calculations, it returns the three PID constants to procedure adaptive___controller. Internally it uses procedure dead to calculate the dead zone non-linearity. The results are then used by RLS1 which uses the previously calculated Kalman gains to compute the new parameter estimates. In the last part of the procedure the actual PID parameter calculations are done.

Apcon2 executes the non-time-critical part of the controller diapason algorithm. The bulk of the parameter estimation work is handled by this procedure. It first high pass filters the control input and the feed forward measurement before calculating the dead-zone width. In the last step of apcon2 procedure RLS2 is called. RLS2 is responsible for the calculation of the Kalman gains and the updating of the Recursive Least Squares parameter estimation. A four step method is used to compute the Kalman gains.

In order to use the results of apconl a PID controller is required. The procedure PIDalg implements such a controller digitally. It will calculate the controller output from the given inputs i.e. setpoint, measured variable and PID parameters and forward it to the simulation program. The above routines will be used extensively in chapter 5 where we test the design using the developed furnace model.

CHAPTER 5 Simulation and Testing

This Chapter describes how the alaptive controller algorithm is integrated with the furnace model and with simulation software to allow the execution of various tests. All the tests are then described in detail.

5.1 Integration of Adaptive Controller into the System

As explained in the modelling section we know that both the gas and air flows determine the amount of heat generated by the burners. Both flows are controlled by a controller. It was decided to use the adaptive controller in place of the existing PID gas controller. [illegible] air flow controller receives its setpoint via a ratio station from the gas flow measurement. Because of the simple nature of the air flow controller and the observation that it works well in practice, this controller can be adequately approximated by the function

air = K * gas   where K is the set ratio

This simplifies the whole system. However, it takes time before the air flow controller receives its setpoint signal from the gas flow controller via the ratio station and we therefore delay the air flow response by one sample interval.

It was decided to disable the feed forward function of the adaptive controller. After all the initial parameters had been set up, a test run was performed. The result was that the controller did not stabilize. The calculated P, I and D values kept on oscillating and resulted in an uncontrolled zone temperature. After looking at this problem carefully it was found that the second order adaptive controller was trying to identify and control a plant that behaves essentially like a first order system. There are therefore too many degrees of freedom for such a system. A decision was then made to use a PI adaptive controller instead. The source code was modified accordingly.

The adaptive controller parameters were chosen to give a closed loop response with half the rise time of the open loop response. The resultant PI adaptive controller proved to be successful as is shown by the tests that follow.

5.2 Fixed Controller Zero versus Calculated Controller Zero

Normally when setting up the adaptive controller one would first decide on the type of response required from the closed loop system. One suggestion is to choose the closed loop response so that the resultant rise time is half that of the open loop system. The closed loop poles are then calculated so as to give this specified performance. However, a first order system has one pole and the zero contributed by the PI controller can therefore be placed so as to cancel it. This is easy to do if the time constant of the plant is known. The closed loop response is then dominated by the remaining poles which can be specified at will. For our system the open loop time constant is approximately 100 seconds.

Figure 5.1 shows the results of such a test run. In all the following temperature graphs the top two curves are the zone temperature and its setpoint and the bottom two are the gas and air flows. On the PI graphs the curve initially at the top is the I constant in seconds while the bottom curve is the proportional band P.

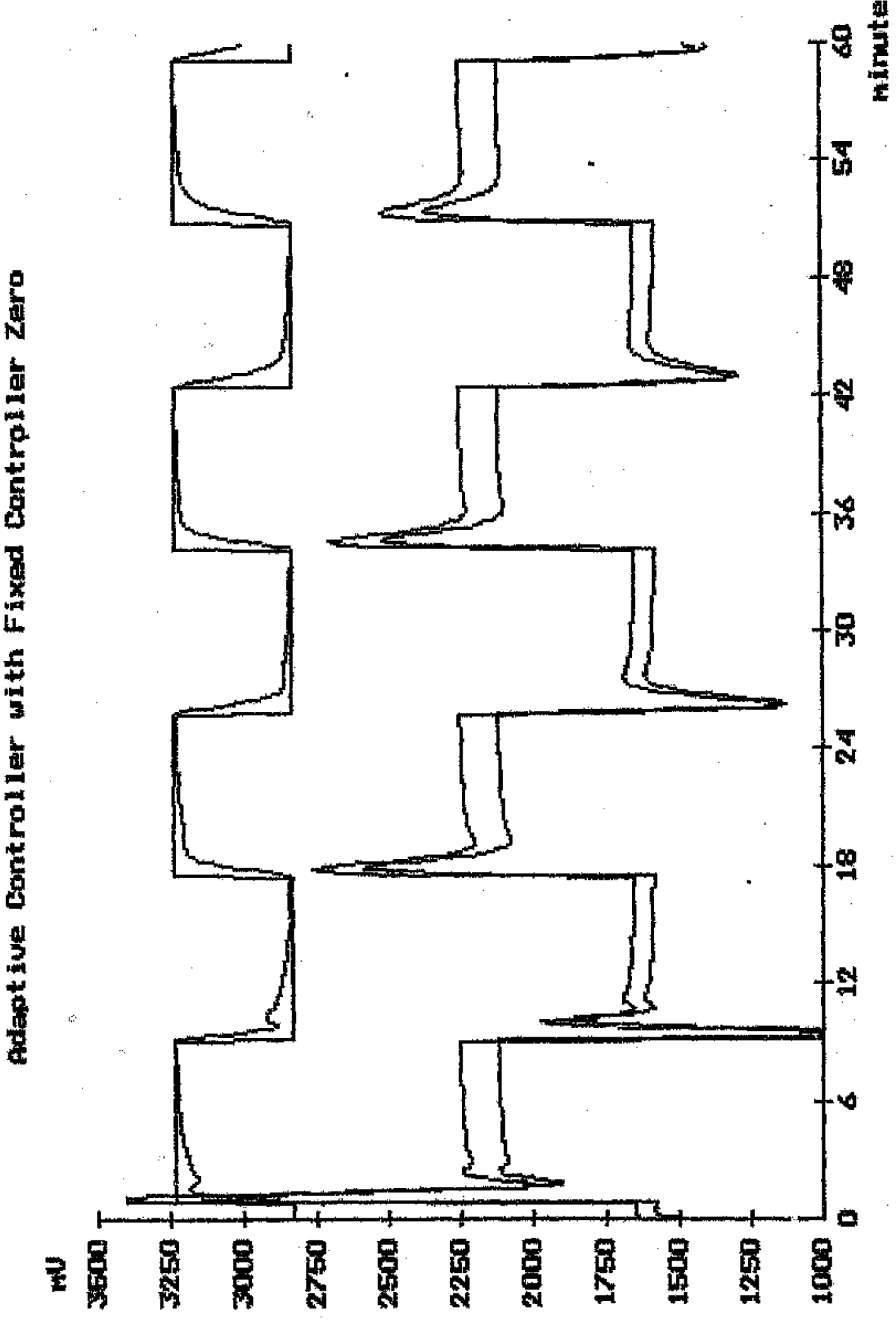


Fig. 5.1 Zone Temperature Graph for Adaptive Controller with Fixed Controller Zero

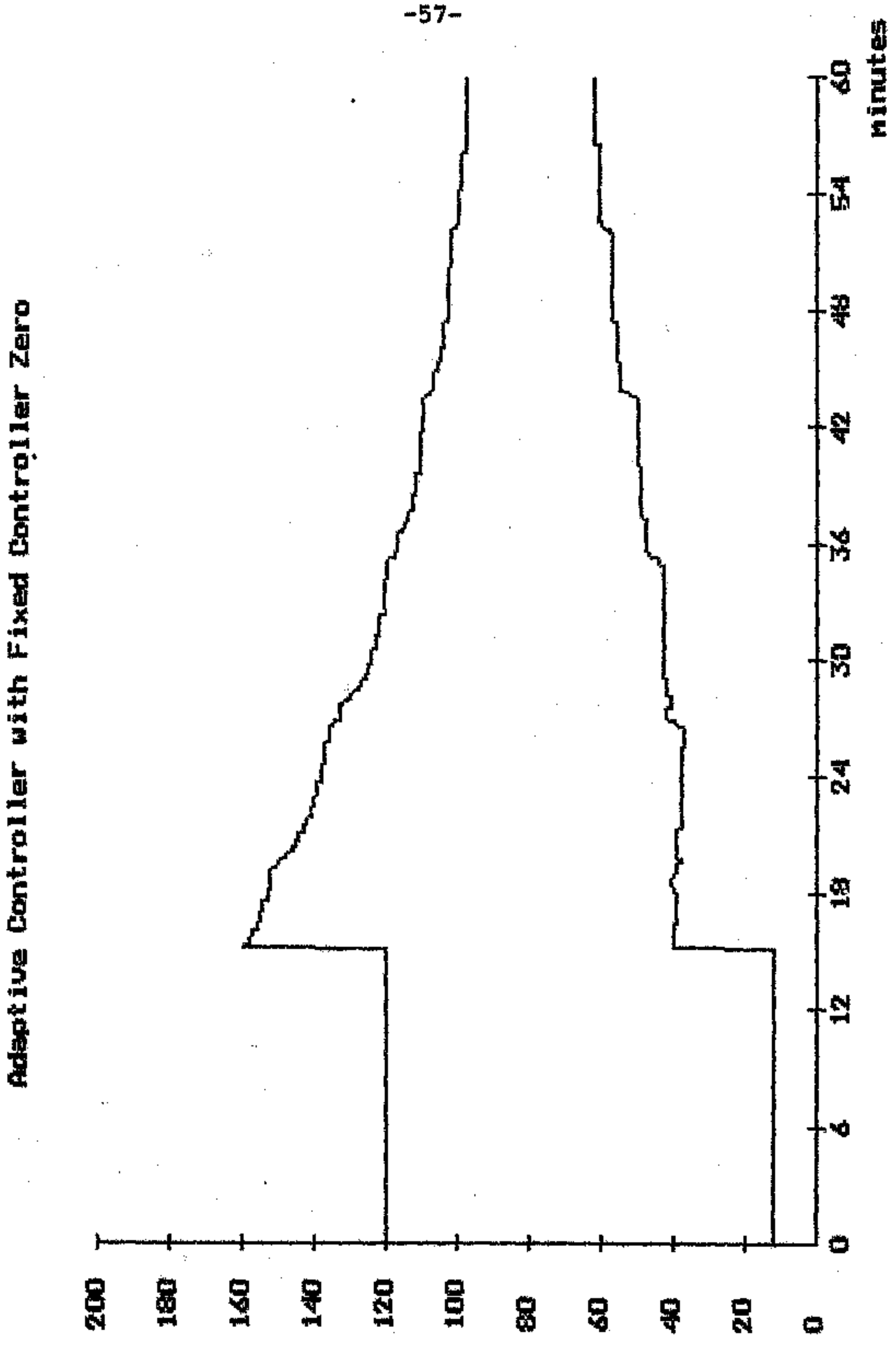


Fig. 5.2 P and I Constants Graph for Adaptive Controller with Fixed Controller Zero

It can be seen in figure 5.2 that the first P and I constants calculations occur at the 15 minute mark when the adaptive controller design is activated. At the start of the test the P and I values are chosen to give a poor response as can be seen in figure 5.1. The adaptive controller improves the response immediately once it is activated. It keeps on improving the response until it reaches a rise time of 55 seconds for the last step which is very close to the specified 50 seconds. The resultant P and I values are:

P = 65% and I = 97 seconds

These are comparable with the controller settings used on the actual plant, i.e.

P = 80% and I = 120 seconds

Another way of setting up this controller is to use the system identification estimates to place the controller zero. As the system is better identified the controller design would improve. The advantage of this would be that the user could specify the rise time that he requires. The controller would take care of the rest and achieve this goal if the system is capable of attaining the specified rise time.

Figures 5.3 and 5.4 show the resulting graphs. The setpoint curve is the same as for the previous test. A rise time of 50 seconds is specified. The first step after the adaptive controller is activated shows a very bad response. The Temperature does not reach th required setpoint throughout the step. Furthermore the PI graph shows a negative initial I value which is cut off at -1000 seconds due to an IF statement in the controller algorithm. Three minutes later a I value of +1000 seconds is calculated which effectively disables

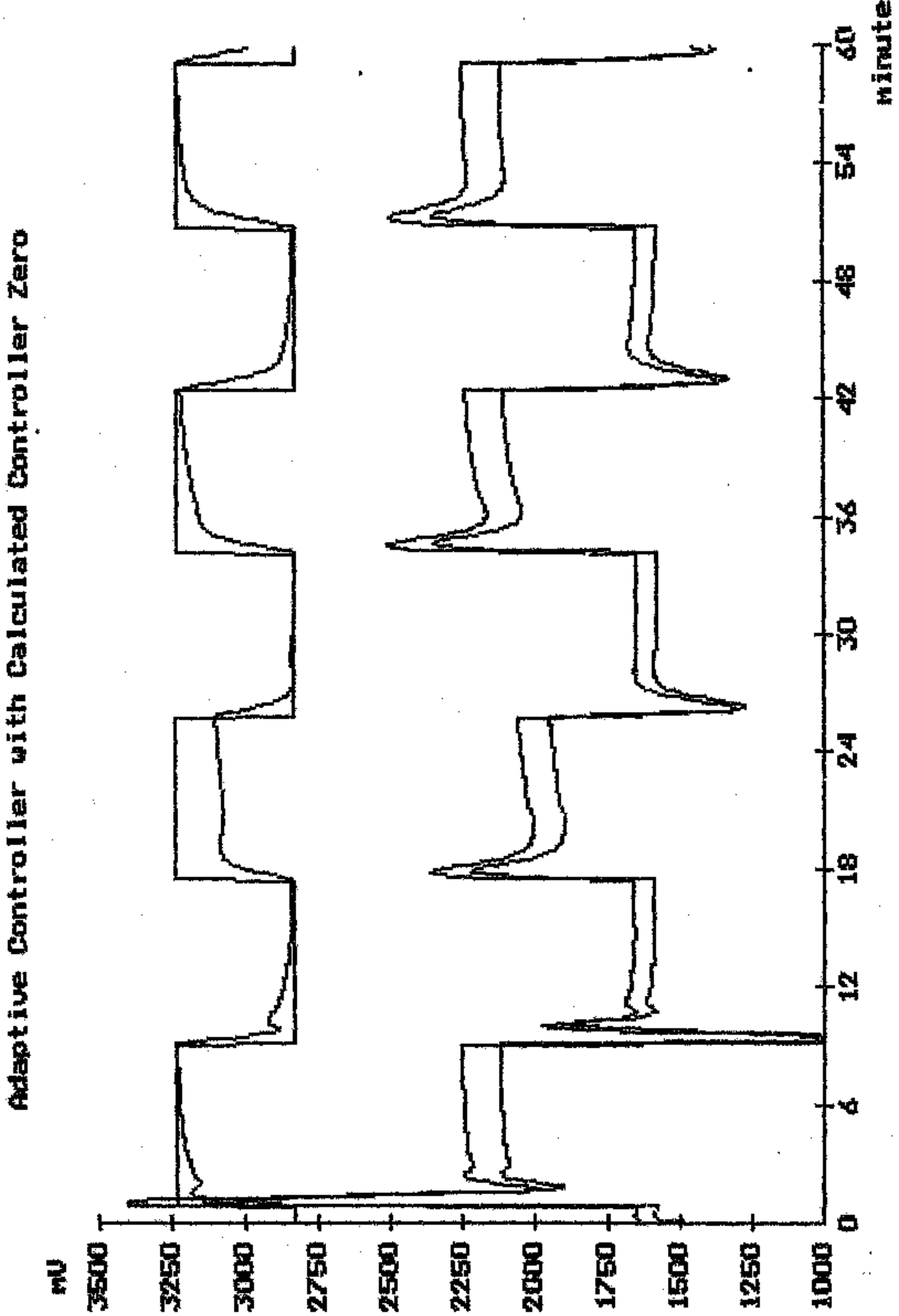


Fig. 5.3 Zone Temperature Graph for Adaptive Controller with Calculated Controller Zero

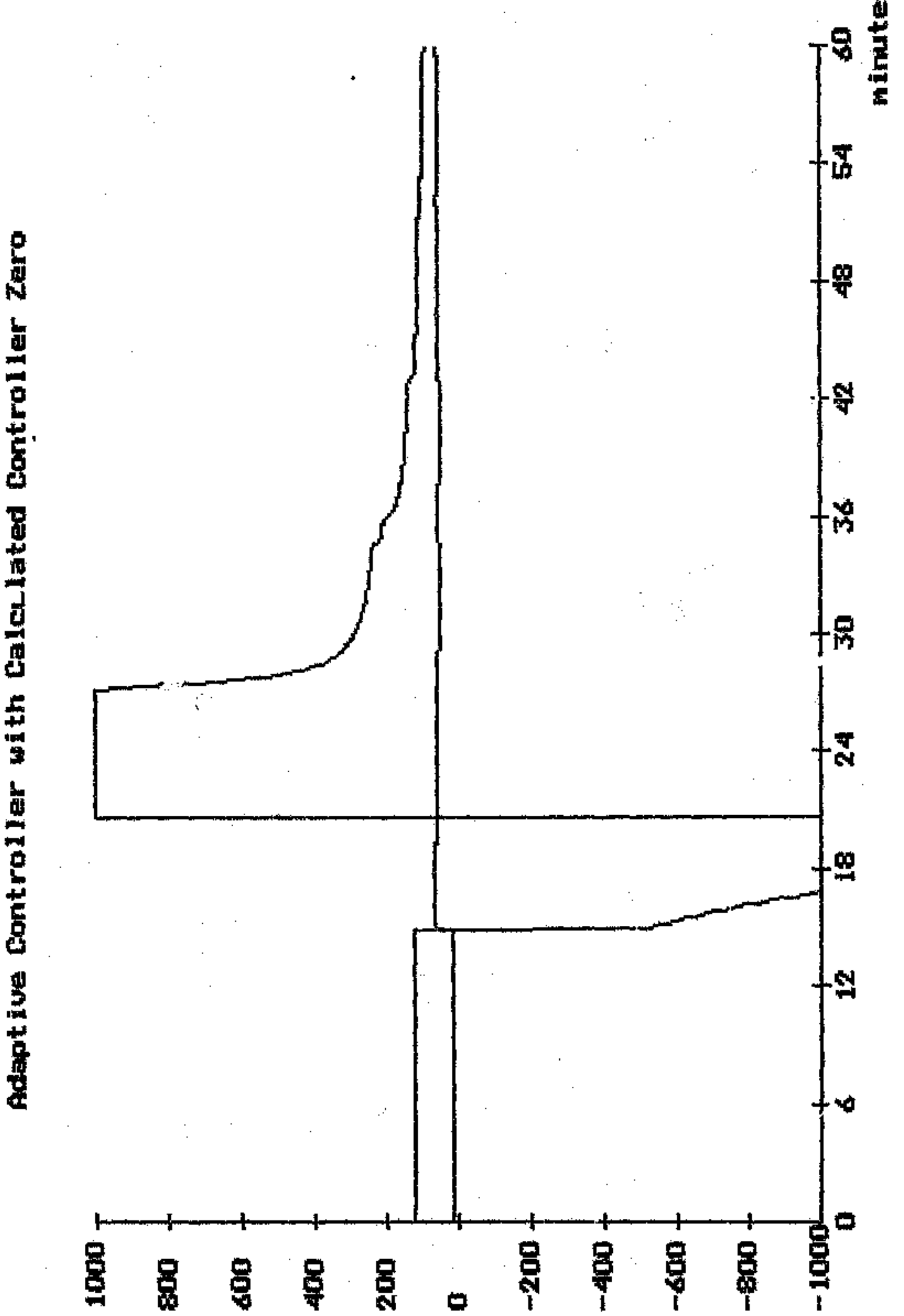


Fig. 5.4 P and I Constants Graph for Adaptive Controller with Calculated Controller Zero

the integral action. It is thus understandable that under these conditions the setpoint cannot be reached. The proportional band, however, is quite stable during this period. At the end of the test we can see that the controller has recovered completely. The final P and I values are:

P = 63% and I = 95 seconds

The above two values are very similar to those obtained in the previous test. The user specified rise time of 50 seconds is achieved. It can therefore be concluded that this controller setup performs as expected after the initial transient.

The reasons for the initial unsatisfactory response were found to be the poor AC estimates of the identification algorithm. The calculated controller zero is a function of the user specified rise time and the A0 estimate which relates to the plant zero. The position of the controller zero therefore changes with the A0 estimate resulting in a varying controller specification.

In order to improve the initial response it was decided to give the system identification part of the adaptive controller good initial parameter estimates. However, this had no effect on the response. It was then decided to change the cortroller algorithm so that it would check the calculated P and I values. Any negative values would be replaced by more reasonable ones. Should there be a negative integral value the algorithm would effectively disable this function by choosing the very large value of 1000 seconds. Similarly a negative gain would result in a very small gain of 0.1 which slows down the system tremendously.

Figures 5.5 and 5.6 show the results of this test run.

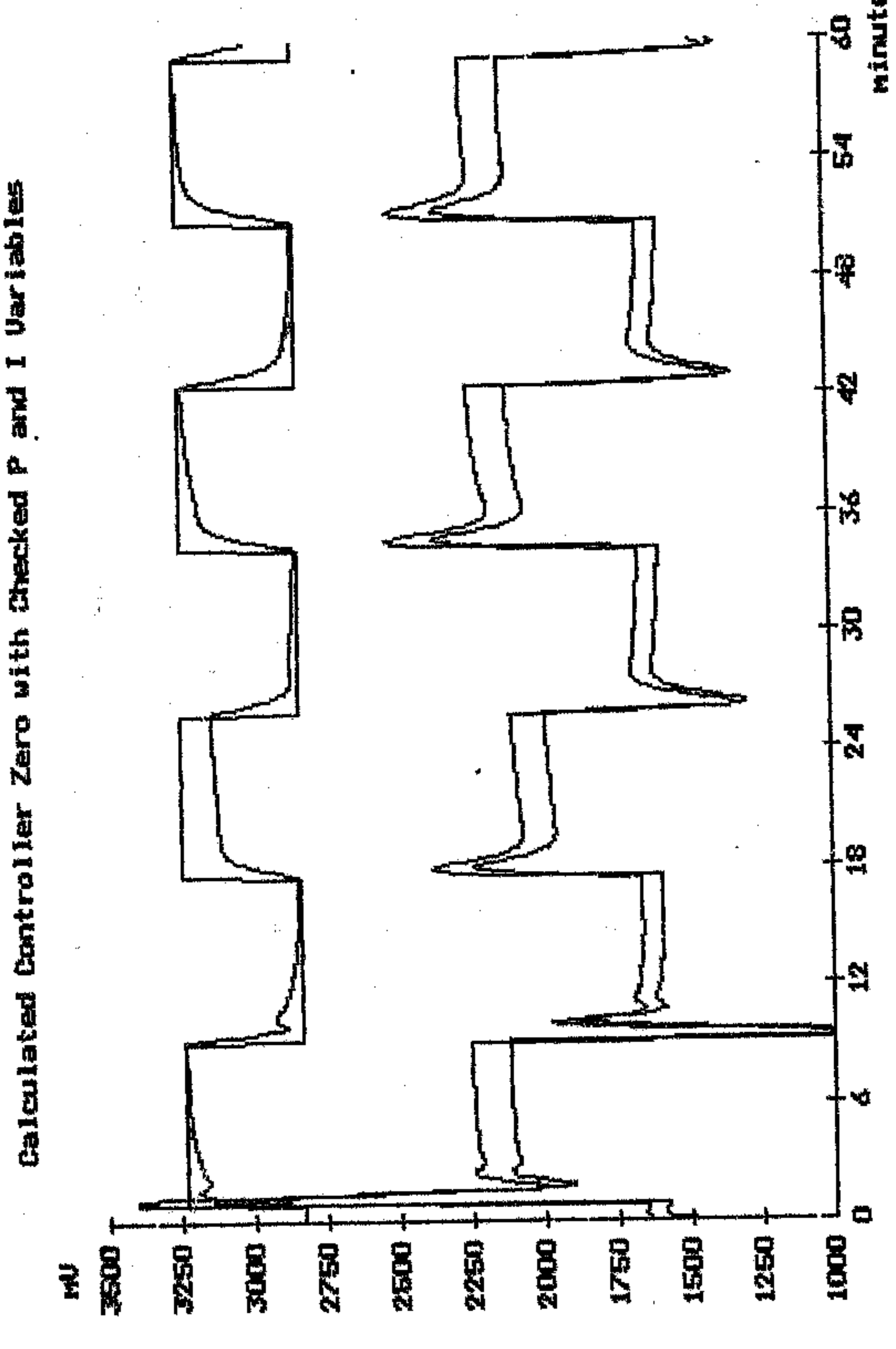


Fig. 5.5 Zone Temperature Graph for Adaptive Controller with Checked P and I Variables

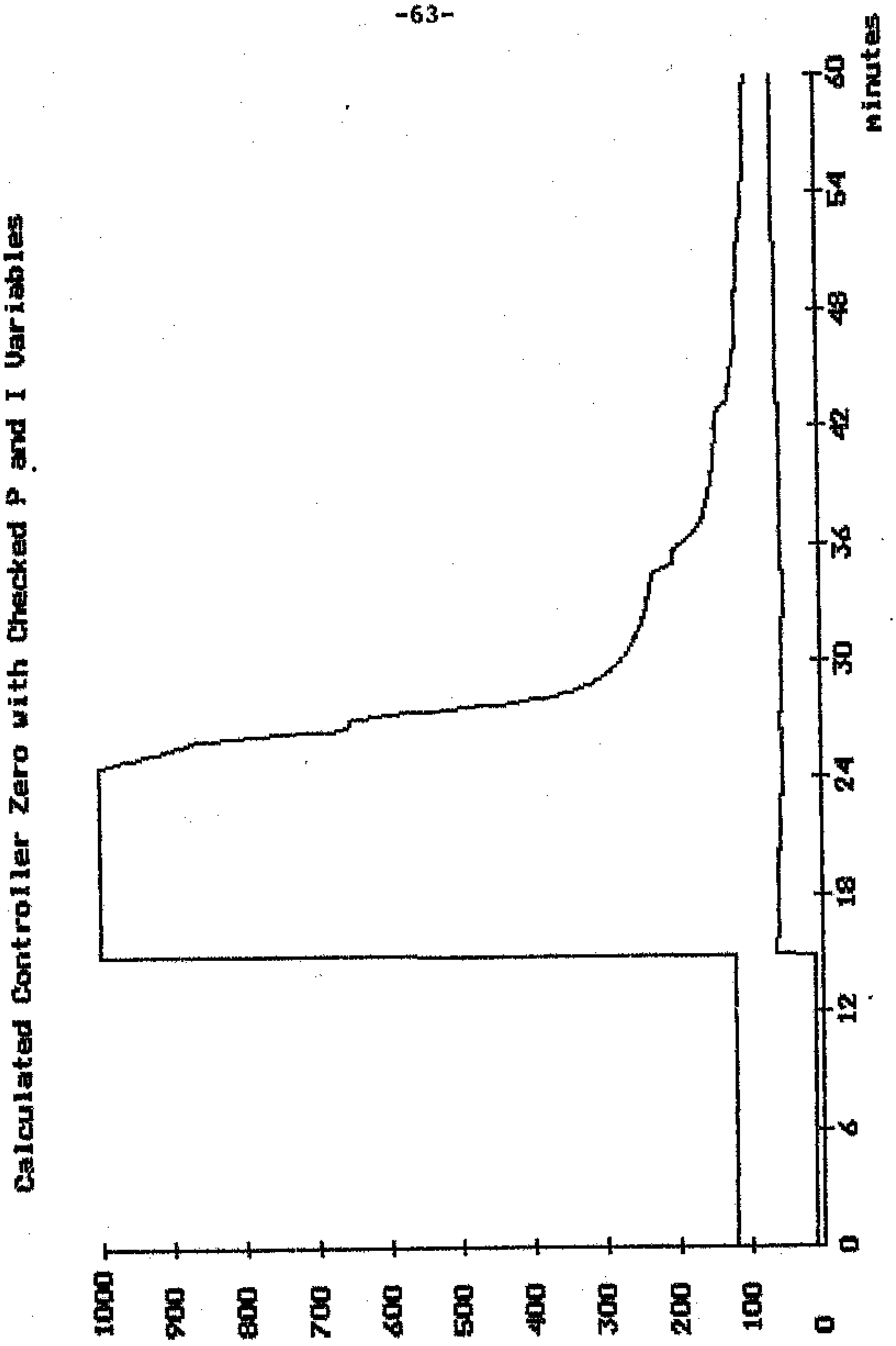


Fig. 5.6 P and I Constants Graph for Adaptive Controller with Checked P and I Variables

It can be seen that the initial response has been improved. However the setpoint is still not reached in the first step after the activation of the controller design algorithm. Ag..in the controller stabilizes and its final P and I values are identical to the previous ones.

However, the best results were obta'ned from the fixed controller zero design and we therefore use this configuration for all future tests.

5.3 Simulation of a Cold Slab Entering the Furnace

The entry of a cold slab can be simulated by increasing the constant k1 i.e. the heat capacity of the furnace wall and slabs of equation (2) in section 3.3. In figures 5.7 and 5.8 the system is subjected to a 27% step increase in k1 at the 30 minute mark.

The result is a slight dip in the furnace temperature. The gas and air flow rates increase immediately. When comparing figure 5.8 to 5.2 we see how the P and I estimates are affected by this step. The proportional band experiences a dip at the 36 minute mark. It eventually settles at 64% which is very close to the 65% of the first test. The integral time displays a very similar curve when compared to the one in figure 5.2 but eventually settles at a lower value of 89 seconds.

The overall response of this test is good since the system stabilizes and again achieves a rise time of 55 seconds. This test therefore suggests that this controller would handle a large change in the heat capacity of the furnace walls and slabs.

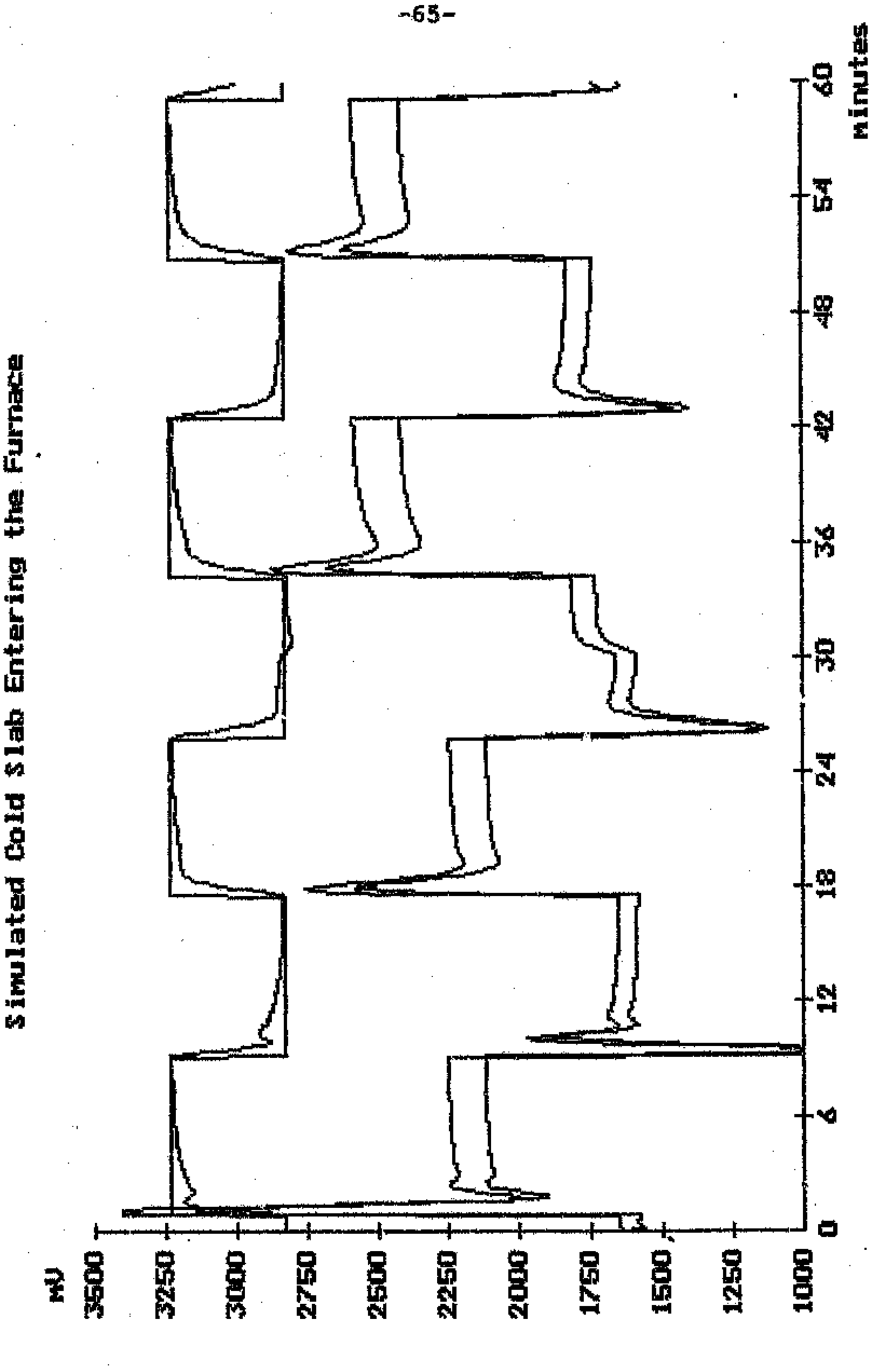


Fig. 5.7 Zone Temperature Graph for Simulated Cold Slab Entering the Furnace

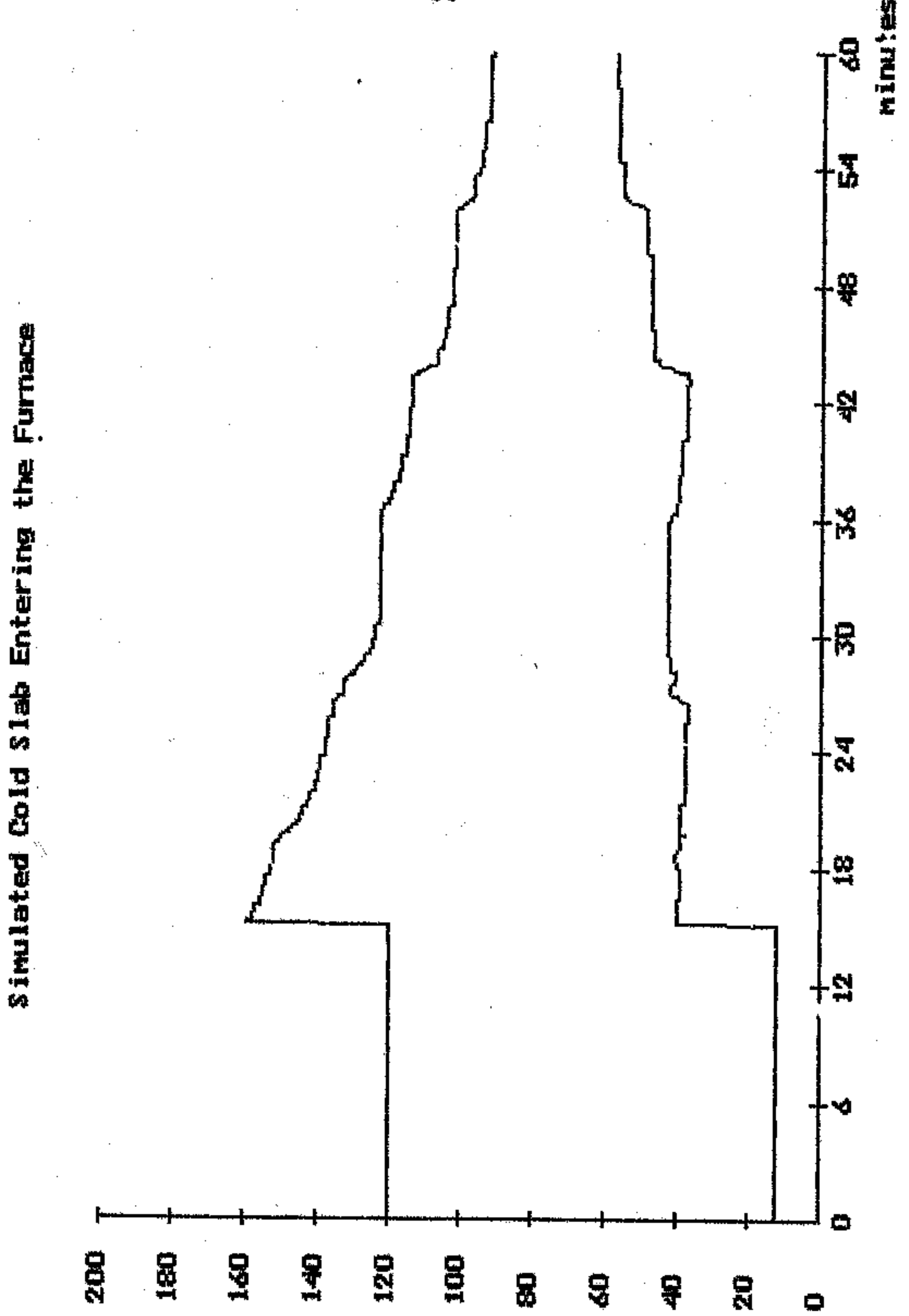


Fig. 5.8 P and I Constants Graph for Simulated Cold Slab Entering the Furnace

5.4 Simulation of a Change in the Exhaust Gas Temperature

One of the assumptions made in modelling the furnace was that the exhaust gas temperature is a constant 900°C. In this test we first step this temperature down to 500°C at the 21 minute mark and then up again to 1100°C at 38 minutes. These steps are much larger than one would expect in the real system. The controller is therefore tested under severe conditions.

Figures 5.9 and 5.10 display the results. We notice a temperature rise at 21 minutes due to the lower exhaust gas temperature which results in a smaller heat loss. The gas and air flows react by cutting both flows. At that instant the P and I curves do not display characteristics similar to those of figure 5.2. The adaptive controller recovers quickly as can be seen by the step starting at 34 minutes. However, halfway through this step the system experiences an even larger step to 1100°C in the exhaust gas temperature. A relatively large decline in the zone temperature can be seen due to the larger heat loss. This time a rise in the integral time and drop in the proportional band can be seen in figure 5.10. Again the controller recovers as can be seen by the last step. The final values for P and I are:

P = 47% and I = 105 seconds

The controller therefore chose a larger gain with less integral action. From these results we can conclude that the controller can handle large changes in exhaust gas temperature

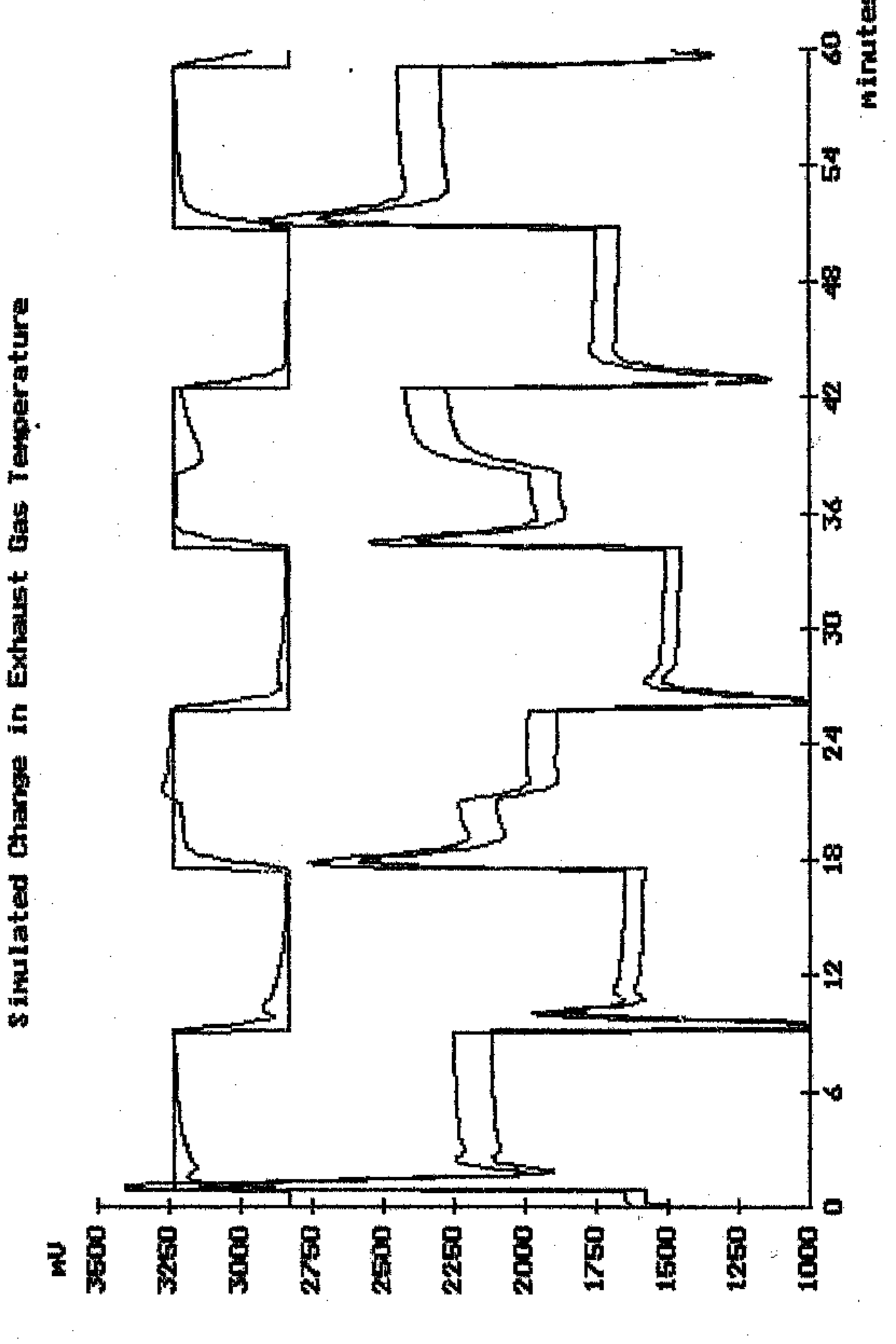


Fig. 5.9 Zone Temperature Graph for Simulated Change in Exhaust Gas Temperature

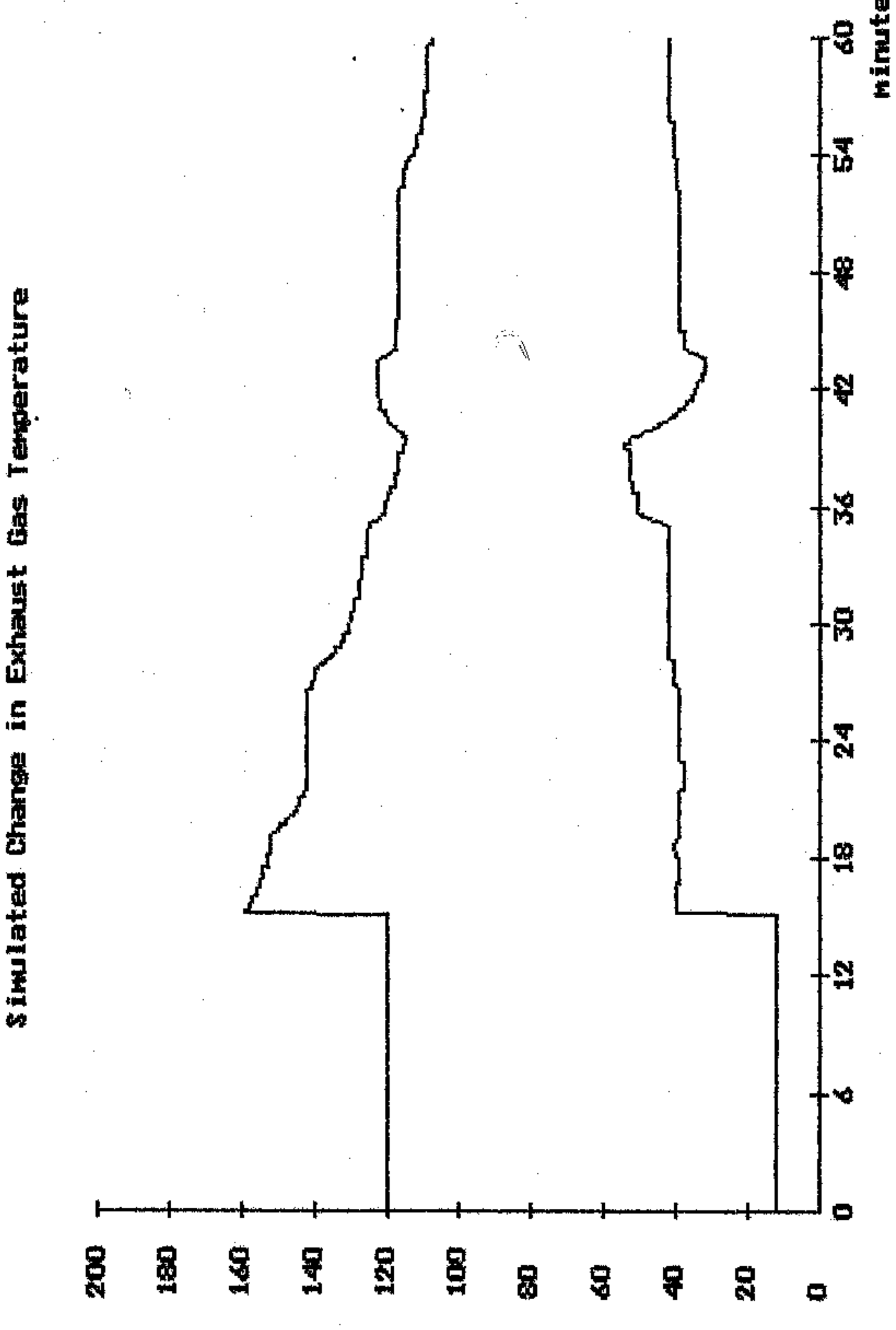


Fig. 5.10 P and I Constants Graph for Simulated Change in Exhaust Gas Temperature

5.5 Simulation of a Change in the Combustion Air Temperature

Another modelling assumption that was made was that the combustion air temperature is a constant 400°C. At 21 minutes the air temperature is stepped down to 200°C and then at 38 minutes stepped up again to 600°C. Again these steps are considerably larger than one would expect from the real plant. The result is expected to be very similar to the test that was performed in section 5.4. However, since the heated combustion air brings energy into the system unlike the exhaust gases which take energy out, we first expect a zone temperature drop. The magnitude of the resultant disturbance in zone temperature and PI values is expected to be smaller, because the air flow is much less than the exhaust gas flow.

Figures 5.11 and 5.12 confirm our expectations. The controller stabilizes at the following values for P and I:

P = 60% and I = 100 seconds

These values are again fairly close to the ones in the first test of section 5.2 which suggests that the combustion air does not influence the furnace characteristics as much as the exhaust gas. This was expected. Again the controller performs well under these conditions.

5.6 Simulation of a Change in the Burner Characteristics

The characteristics of the burner would be most affected by a change in the calorific value of the coke oven gas. We therefore step the calorific value from 4395

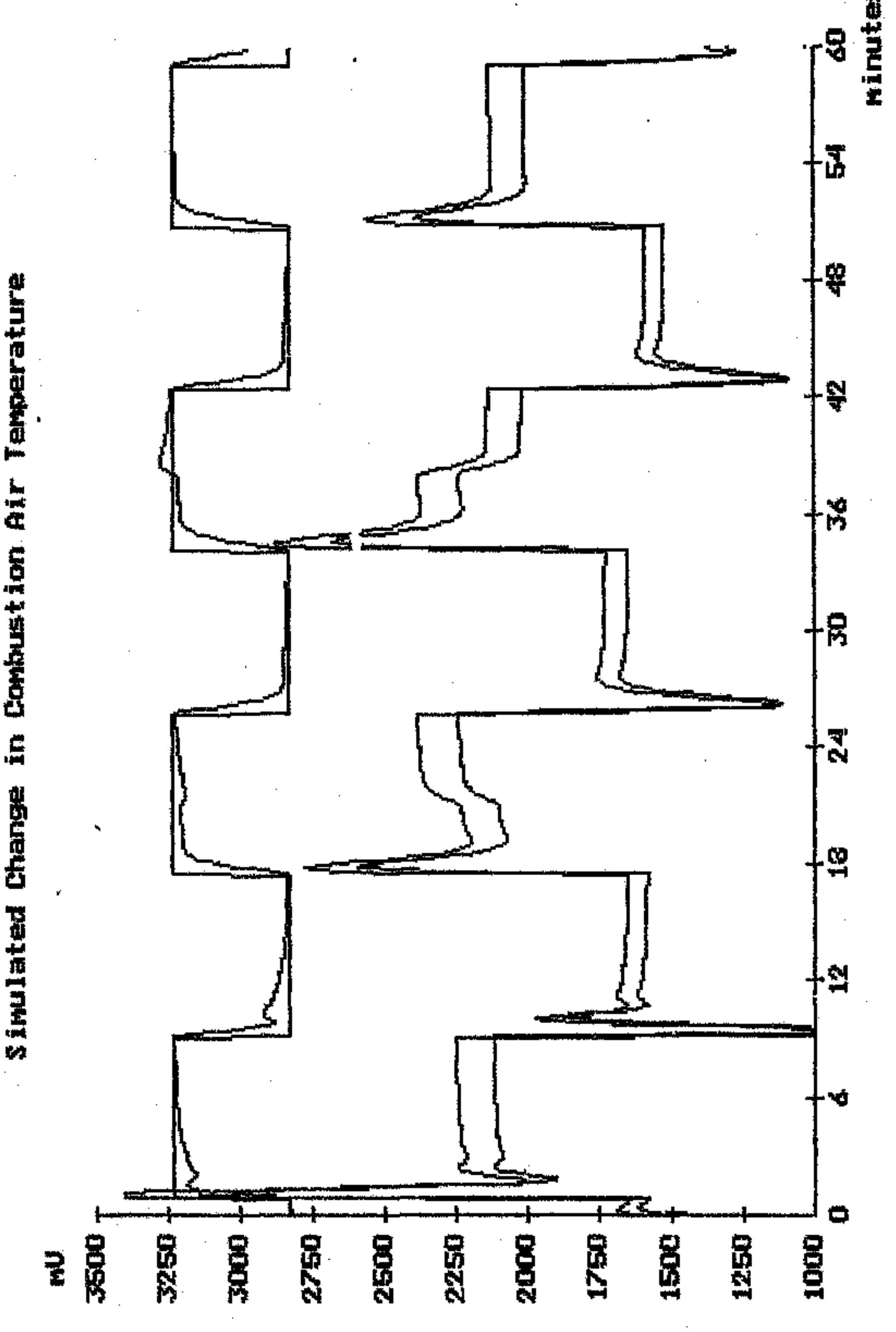


Fig. 5.11 Zone Temperature Graph for Simulated Change in Combustion Air Temperature

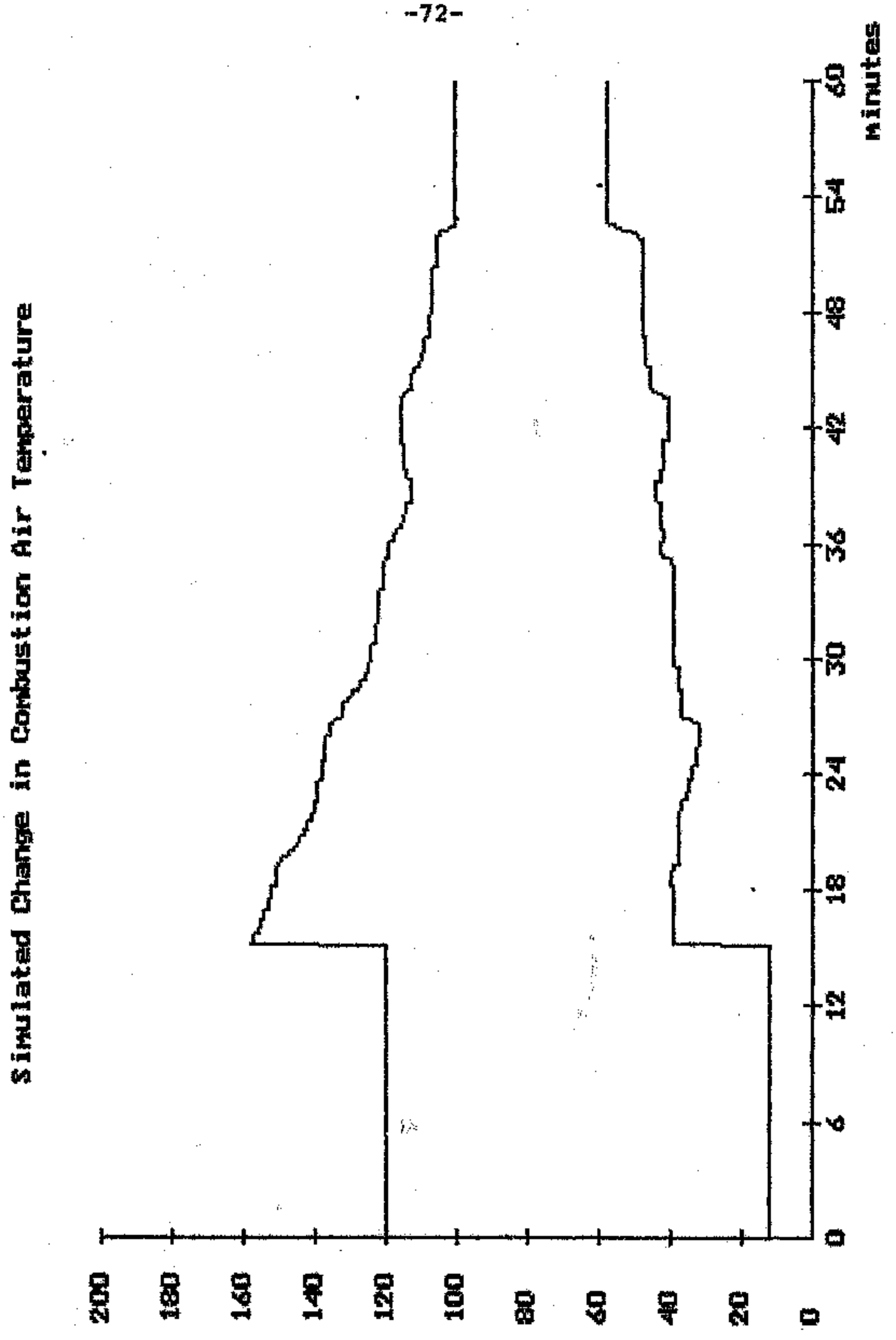


Fig. 5.12 P and I Constants Graph for Simulated Change Combustion Air Temperature

Kcal/m^3 to 5000 Kcal/m^3 at the 21 minute mark. At 38 minutes we step it down to 4000 Kcal/m^3. The resulting response is shown by figures 5.13 and 5.14.

These figures display a very similar response to the one in figures 5.9 and 5.10. By increasing the calorific value of the coke oven gas we incre: . the energy output per cubic meter which results in the small zone temperature rise while the lower calorific value results in a larger zone temperature drop. After the controller has stabilized the P and I values are as follows

P = 50% and I = 103 seconds

These values are very similar to the ones in the exhaust gas temperature tests. We can once again conclude that the controller performs well in this test.

5.7 The Addition of Random Noise to the Zone Temperature

The final test for the adaptive controller is the addition of a random noise signal to the zone temperature. The amplitude of the noise signal is chosen to be ±5% of the input step amplitude.

Figures 5.15 and 5.16 show that the controller performs reliably under these conditions. When compared to the initial test in section 5.2 which has no noise or disturbance added to the system we find that the PI graphs are very similar. This confirms that the controller design is not affected by the noise and suggests that the signal filtering used in the adaptive controller algorithm is effective. The final P and I variables are:

P = 66% and I = 96 seconds

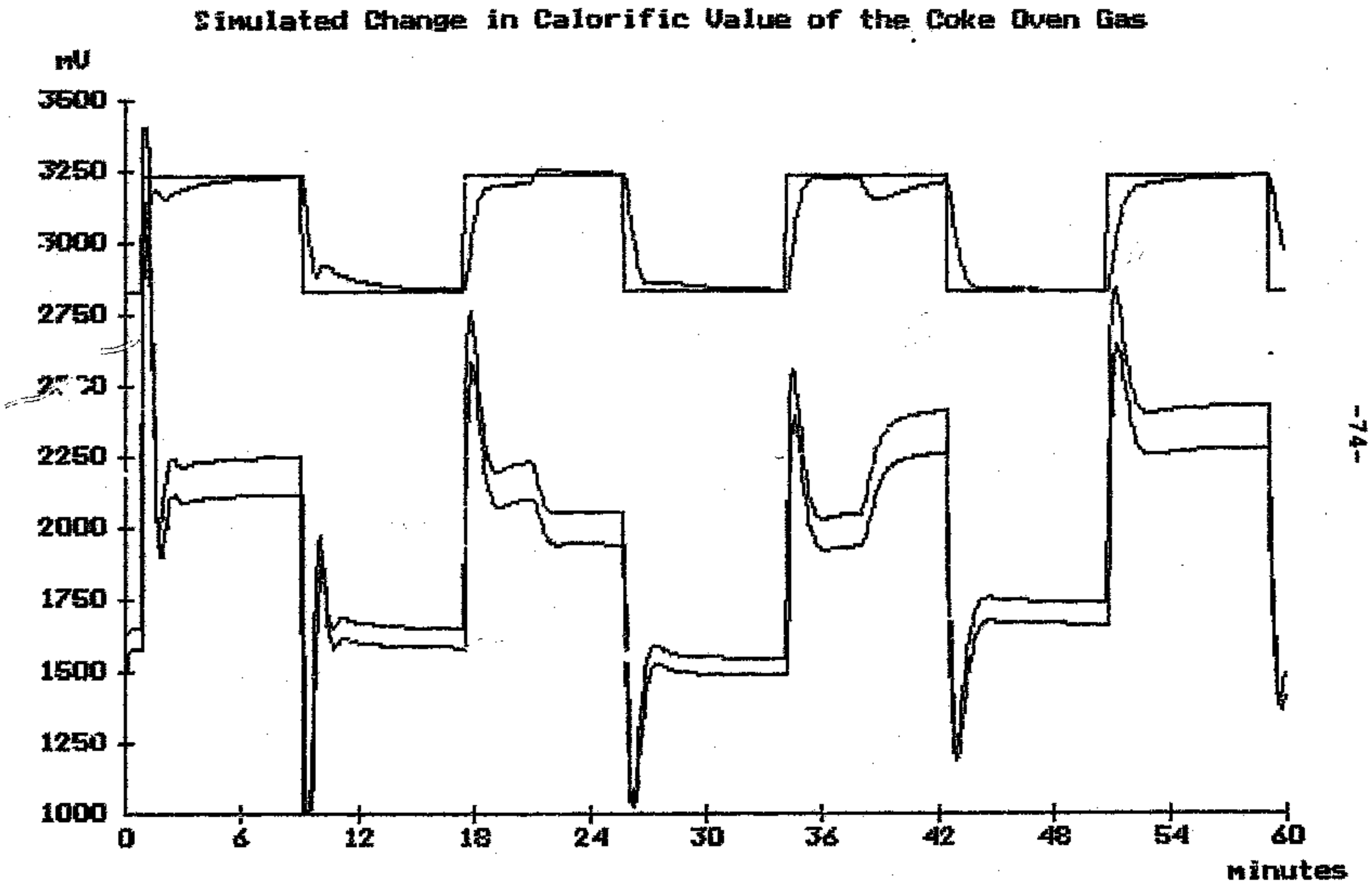


Fig. 5.13 zone Temperature Graph for Simulated Change in Burner Characteristics

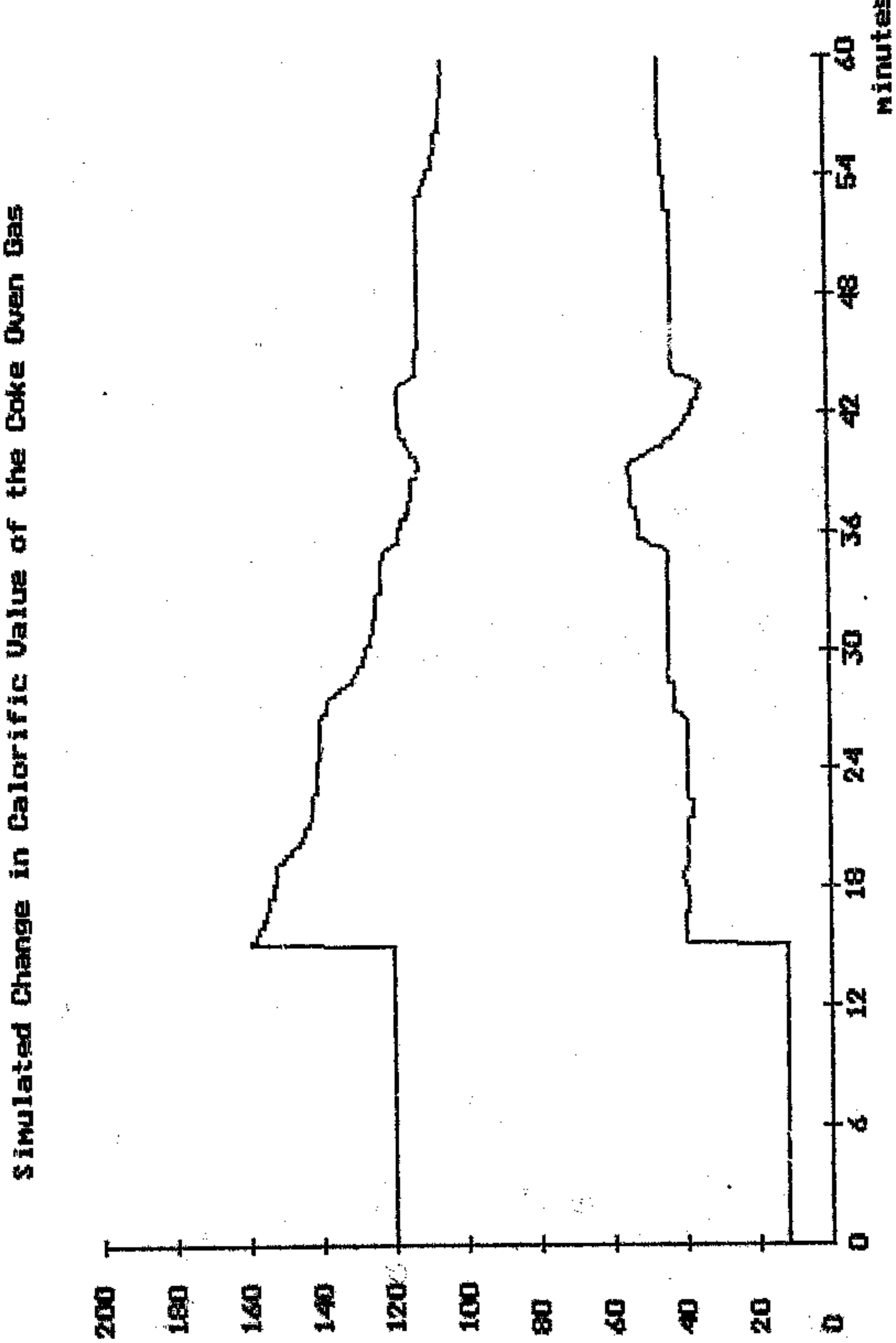


Fig. 5.14 P and I Constants Graph for Simulated Change in Burner Characteristics

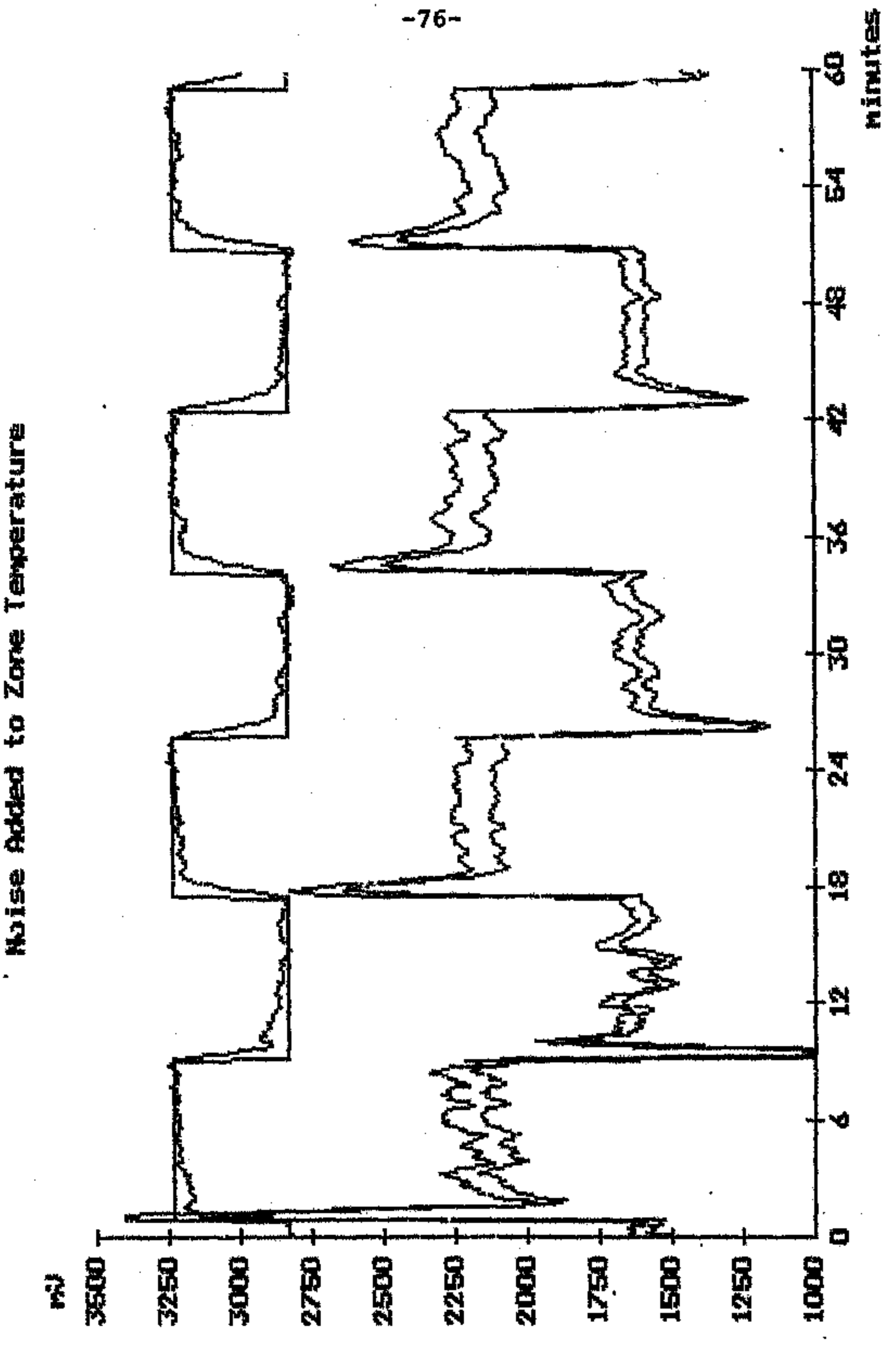


Fig. 5.15 Zone Temperature Graph with Noise added to the Zone Temperature Signal

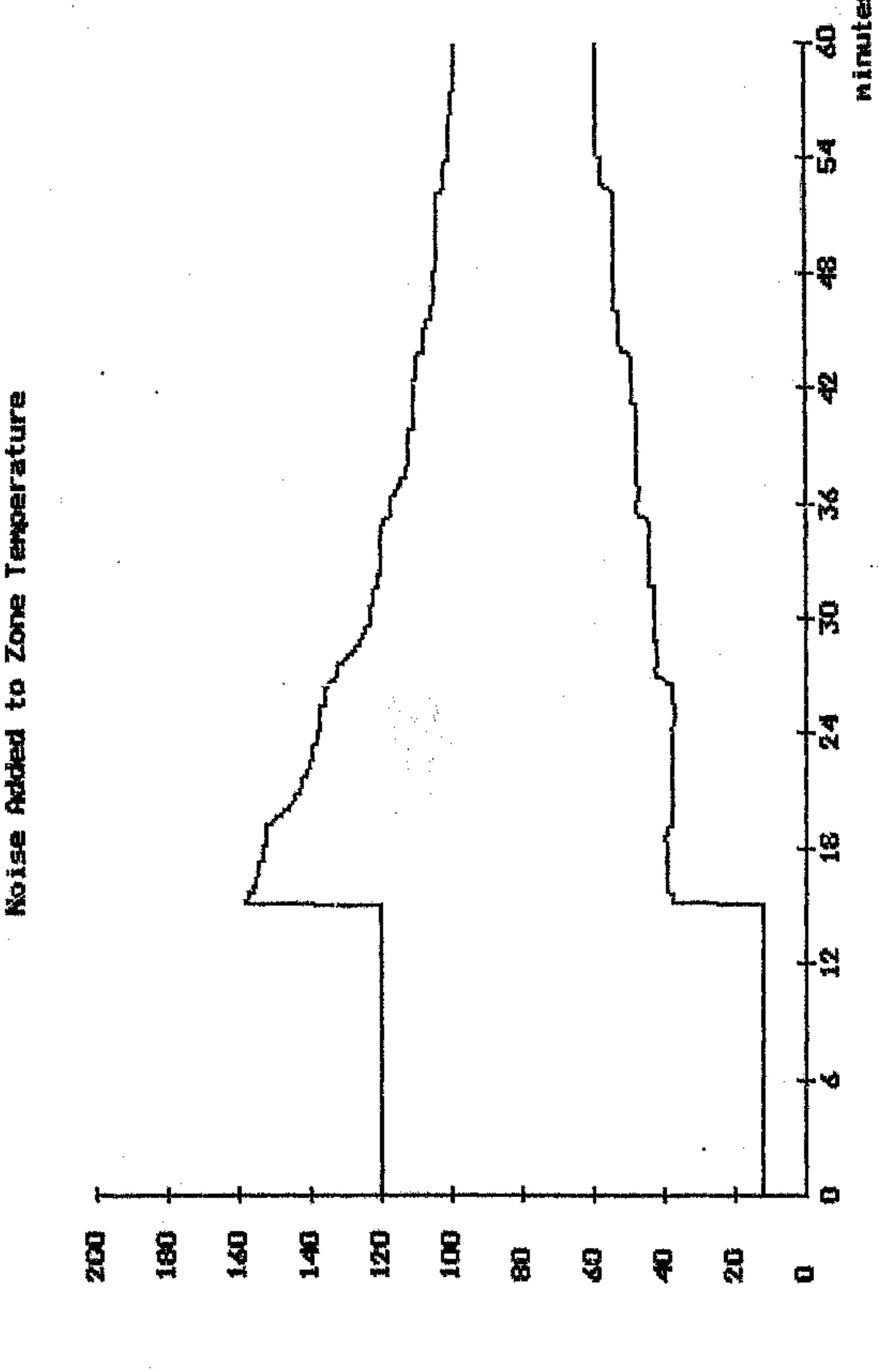


Fig. 5.16 P and I Constants Graph with Noise added to the Zone Temperature Signal

5.8 Evaluation of Adaptive Controller Performance

In the first part of this chapter an interesting idea is investigated. The position of the controller zero is made dependent on the identification parameters and therefore allows the user to specify a rise time. It was found that the controller attains this rise time once it has stabilized. This is only true for the case where the system characteristics allow it. However, the controller requires some time before it stabilizes during which the zone temperature is not kept at setpoint. It was therefore decided to use the fixed zero approach. Future research would probably improve the stability of the calculated controller zero algorithm.

Various physical changes in the furnace's characteristics were simulated and the effects on the controller were investigated. The controller performed reliable under these conditions. The addition of noise to the measured variable i.e. the zone temperature had no negative effect on the controller's ability to control the zone temperature. These results are encouraging and one could try this adaptive controller design with a fair amount of confidence on the real system.

CHAPTER 6 Conclusions

In the first part of this dissertation we examine the control system of a reheating furnace situated at the Iscor Vanderbijlpark steel works. It is shown that the Bottom Heating Zone is the most difficult to control and this project therefore examines a solution to these difficulties.

In order to test potential controller designs it was necessary to develop a mathematical model of the Bottom Heating Zone. Simulation techniques can then be used to evaluate closed loop performance.

The first step in modelling was to obtain furnace input output data. This was achieved with the use of a portable IBM XT compatible computer and a sixteen channel A/D converter plug-in card. The necessary logging and graphics routines were developed using Turbo Pascal. Once the furnace was understood and its instrumentation equipment calibrated the necessary data was logged using a sampling period of five seconds.

Various tests were performed to determine what effect the other zones have on the Bottom Heating Zone. These effects were found to be negligible. An open loop test was performed which was used extensively in the modelling phase. It was found that the Bottom Heating Zone displayed a rise time of 92 seconds.

The goal of the modelling process was to develop a model whose parameters would have a physical meaning related to the furnace. The model was thus developed using an energy balance as its base. Three chemical reactions were identified as the main energy producers.

This model was then evaluated against the real plant data. An important assumption that was made was that both the gas and the air flow were stepped simultaneously during the open loop test. This was, however, not physically possible. The air flow curve displayed more ramp-like characteristics. The model therefore had to be "fine tuned" by using different values for C the total heat capacity of the furnace. In the end an acceptable model was developed that displayed very similar characteristics to the real furnace.

An adaptive controller developed by MacLeod and Bergesen is chosen. The reasons for this choice are that it is a robust design developed for digital applications. It allows fast sampling if necessary and employs a relative dead zone to ensure that no parameter updating occurs if the input signal excitation is insufficient. The chosen algorithm uses a robust controller design and provides for the nulling of deterministic disturbances as well as deterministic setpoint tracking.

The above algorithm was simplified to an adaptive PI controller because the original adaptive PID controller performed unsatisfactorily. The reasons for the poor performance were that the second order PID adaptive controller was trying to identify our plant model which is essentially first order. This resulted in too many degrees of freedom.

The PI adaptive controller was then tested thoroughly against the model. The first test used a fixed controller zero design and the results were found to be good. Both the P and the I values calculated by the adaptive controller were similar to those used on the plant. This furthermore suggested that our Bottom Heating Zone model was a good one.

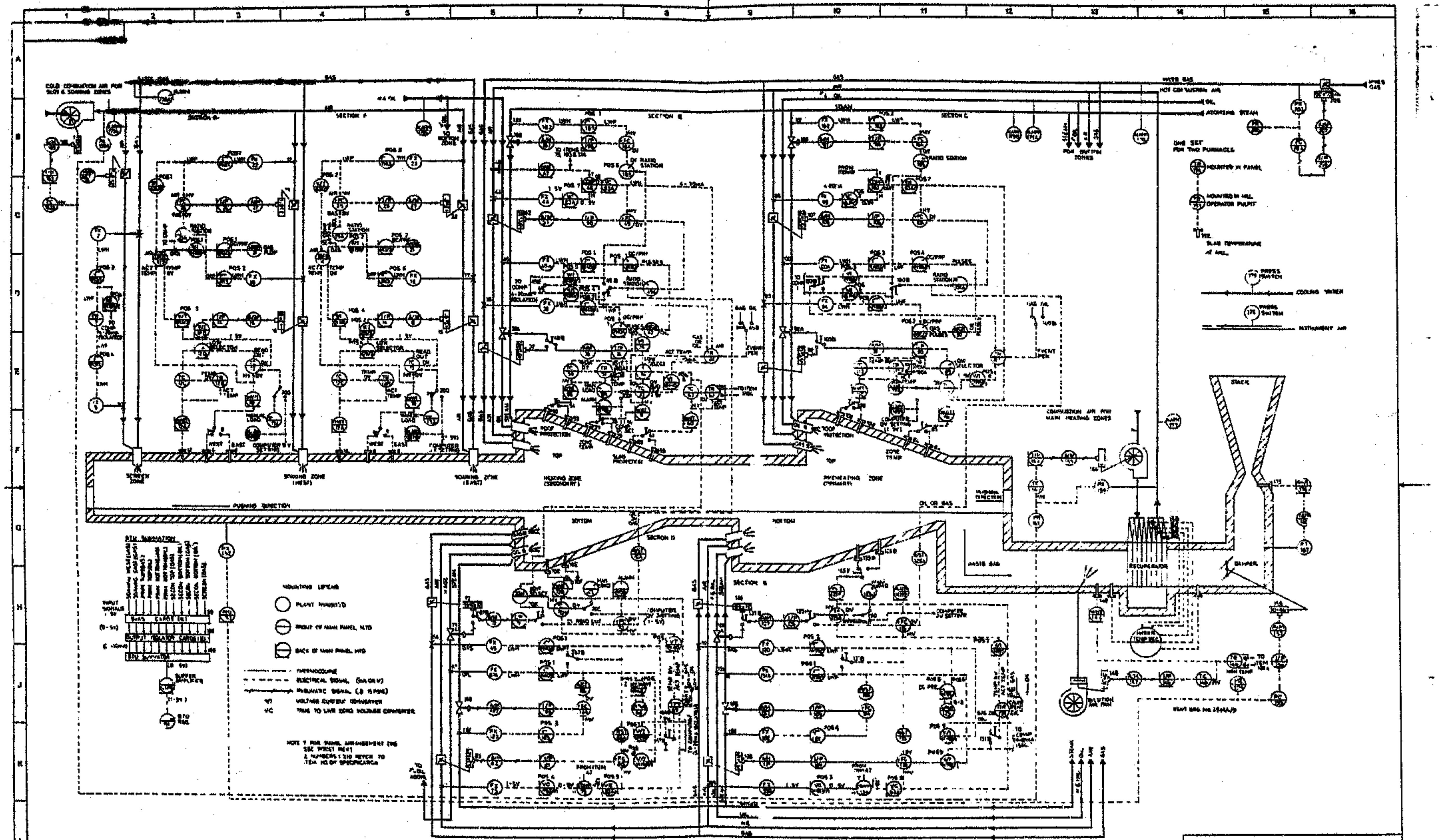
An interesting idea that was investigated was to employ a controller design algorithm that would recalculate the controller zero position based on the system identification parameters. This would allow the user to specify the closed loop rise time. The results proved that this algorithm worked well once the controller had stabilized. However, it displays poor control during the initial stages of the test. The fixed controller zero design was therefore chosen for the other tests. Future research into this field could be very interesting and might resolve this problem.

During the simulated severe furnace characteristic changes the adaptive PI controller performed very well. It handled all situations without becoming unstable. The noise test furthermore proved that the adaptive controller algorithm is robust to measurement noise.

We can therefore conclude that this controller could be used on the real plant with a fair amount of confidence. We should, however, remember that the controller was only tested via simulation against a simplified model. The real plant could display characteristics not displayed by the model.

This project covered a wide range of topics and was of great educational value to the author. At the same time it was thoroughly enjoyed.

APPENDIX A



LOFTUS ENGINEERING CORPORATION
ONE OLIVER PLAZA
PITTSBURGH, PA. 15222

DWG. No. 7538-382

ISCOR YSKOR – VANDERBIJLPARK

REV. 2/10/74 JHL
DWG. NO. WAS 71063 REV 2

2050mm SEMI-CONTINUOUS HOT STRIP MILL

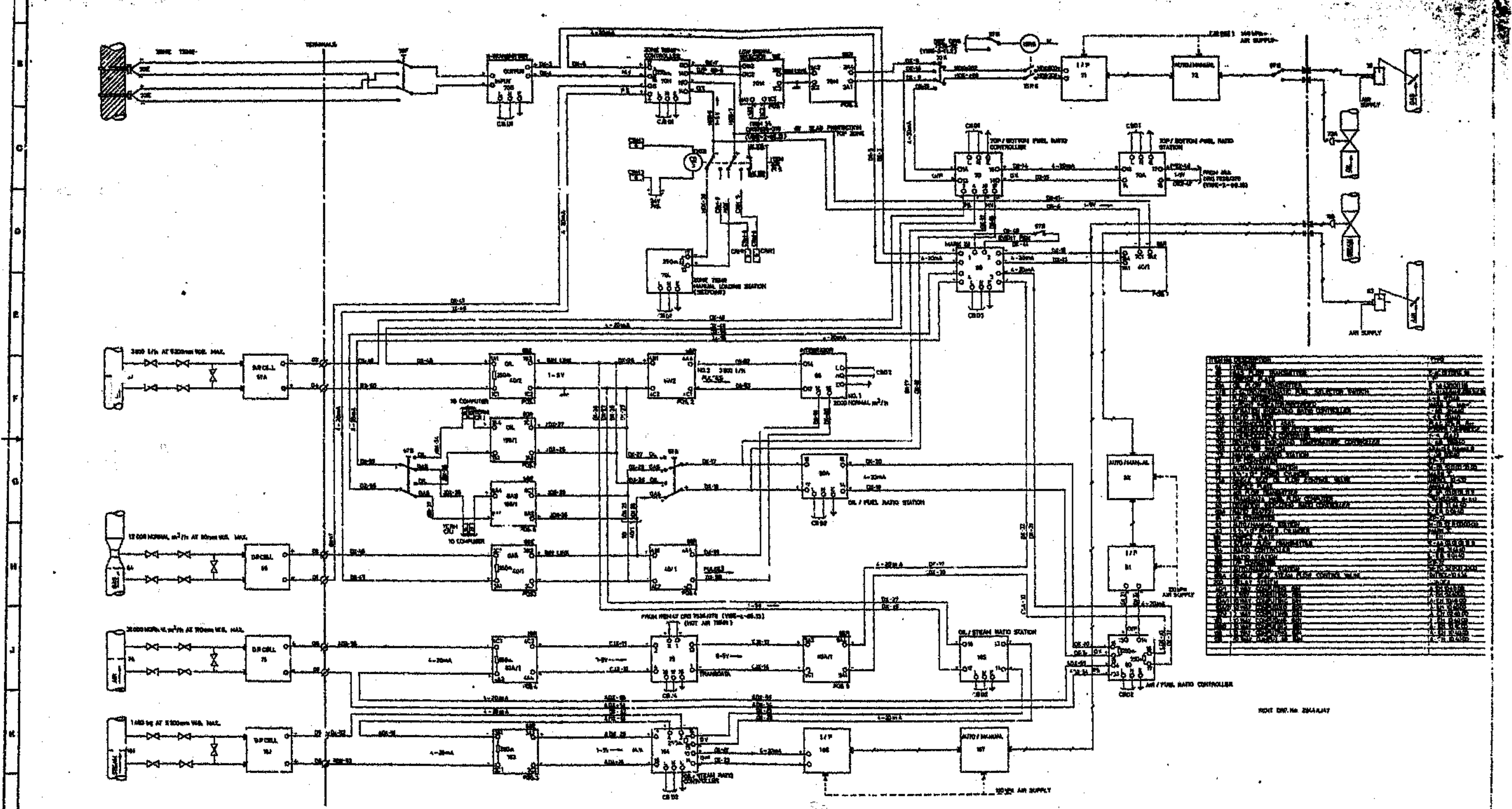
TWO(2) FIVE ZONE REHEAT PUSHER F'CES CONTROL & MEASURING SCHEMATIC FOR SLAB REHEATING FURNACES AT ISCOR VBP

SCALE: DO NOT SCALE

V15C-2-H-52 R

A0

APPENDIX B



NOTES: 1. ON ALL GIR COMPUTING [illegible] CARDS PINS [illegible] ARE LINKED
2. TERM. CR[illegible] PINS [illegible] 1-5V FROM COMPUTER FOR MAIN ZONE TEMP CONTR'L SET POINTS
3. TERM. CR[illegible] PINS [illegible] CONTACTS AT COMPUTER FOR SWITCHING ZONE TEMP CONTROLLERS FROM LOCAL SET TO COMPUTER SET POINT
4. TERM. CRJ PINS 13,14,15 & 16 4-20mA D.C. FUEL FLOW SIGNALS

REV △ 8-3-74 JML
WIRE No ADDED

LOFTUS ENGINEERING CORPORATION
ONE OLIVER PLAZA
PITTSBURGH, PA. 15222
DWG. No. 7538-456

ISCOR YSKOR - VANDERBIJLPARK

2080MM SEMI-CONTINUOUS HOT STRIP MILL
TWO(3)FIVE ZONE REHEAT PUSHER TYPE(S)
SLAB REHEATING FURNACE SECONDARY
HEATING BOTTOM ZONE

V15¢-2-c5.2F

APPENDIX C

Listing of LOG.PAS, page 1 at 11:55am 03/09/90

```
PROGRAM  LOG (input,output);
{###########################}

USES
  crt,
  dos;

CONST
  max = 1500;

TYPE
  datarecord  = RECORD
                  samp0,
                  samp1,
                  samp2,
                  samp3,
                  samp4,
                  samp5,
                  samp6,
                  samp7,
                  samp8,
                  samp9,
                  samp10,
                  samp11,
                  samp12,
                  samp13,
                  samp14,
                  samp15 : integer;
                END;
   samplearray = array[1..max,0..15] of integer;

 VAR
   tfile   : text;
   st      : string;
   sample  : samplearray;
   samples,
   hour_no : integer;

 {******************************************************************* Showtime *}

 FUNCTION time : string;
 {***********}

 VAR
   h,
   m,
   s,
   d       : string[2];
   hour,
   minute,
   second,
   digits : word;


 BEGIN
   gettime(hour,minute,second,digits);
   str(hour,h);
  str(minute,m);
  str(second,s);
  str(digits,d);
  time := h + 'H' + m   ':' + s + '.' + d + '   ';
END;


{******************************************************************* filedata *

PROCEDURE filedata (filename : string);
{****************}

VAR
  no      : integer;
  datrec  : datarecord;
  xydat   : file OF datarecord;

BEGIN
  filename := filename + '.DAT';
  assign(xydat,filename);
  rewrite(xydat);
  FOR no := 1 TO samples DO
    BEGIN
      WITH datrec DO
        BEGIN
          samp0  := sample[no,0];
          samp1  := sample[no,1];
          samp2  := sample[no,2];
          samp3  := sample[no,3];
          samp4  := sample[no,4];
          samp5  := sample[no,5];
          samp6  := sample[no,6];
          samp7  := sample[no,7];
          samp8     sample[no,8];
          samp9  := sample[no,9];
          samp10 := sample[no,10];
          samp11 := sample[no,11];
          samp12 := sample[no,12];
          samp13 := sample[no,13];
          samp14 := sample[no,14];
          samp15 := sample[no,15];
        END;
      write(xydat,datrec);
    END;
    close(xydat);
END;


{*************************************************************** samplesystem *

Procedure Samplesystem (hour_no,
{********************}  samples : integer);

VAR
  channel,
  i,j,d    : integer;
  s        : string;
```

```
 stop    : boolean;
 ch      : char;

 {---------------------------------------------------------------------- irit -

 PROCEDURE init (VAR sample : samplearray);
 {-----------}

 CONST
   value = 2500;

 VAR
   no,nno : integer;

 BEGIN
   FOR no := 1 TO max DO
     FOR nnc := 0 TO 15 DO
       sample[no,nno] := value;
 END;

{************************************************************* samplesystem *

BEGIN
  stop := false;
  i    := 1;
  WHILE (i <= hour_no) AND NOT stop DO
    BEGIN
      j := 1;
      init(sample);
      WHILE (j <= samples) AND NOT keypressed DO
        BEGIN
          FOR channel := 0 TO 15 DO
            BEGIN
              port[$702] := (channel SHL 4) + 02; {channel select and software.
                                                  {clock bit clear
              port[$702] := (channel SHL 4) + 03; {channel select and software
                                                  {clock bit set
              FOR d := 0 TO 4 DO
                BEGIN                             {loop until end of conversic
                END;
              sample[j,channel] := ((port[$701] AND $0F) * 256) + port[$700];
              sample[j,channel] := round((sample[j,channel] / 4095) * 10000);
            END;
          FOR d := 1 TO 5 DO
            delay(1000);
          gotoxy(1,10);
          write('Current Time is    : ',time);
          inc(j);
        END;
      IF (j <> (samples + 1)) THEN BEGIN
                                   stop := true;
                                   ch   := readkey;
                                  END;
    str(i,s);
    filedata('DATFIL' + s);
```

Listing of LOG.PAS, page 4 at 11:55am 03/09/90

```
    inc(i);
  END;
END;

{################################################################ main program #}

BEGIN                                                               {initialize PPI}
  port[$703] := $92;
  clrscr;
  gotoxy(10,12);
  write('Enter Number of Hours            : ');
  readln(hour_no);
  gotoxy(10,14);
  write('Enter Number of Samples per Hour : ');
  readln(samples);
  clrscr;
  gotoxy(1,8);
  write('Started Logging at : ',time);
  gotoxy(1,10);
  assign(tfile,'TIME.DAT');
  rewrite(tfile);
  writeln(tfile,time);
  close(tfile);
  samplesystem(hour_no,samples);
  append(tfile);
  writeln(tfile,time);
  close(tfile);
END.
```

APPENDIX D

Listing of GRAPHICS.PAS, page 1 at 12:08pm 03/09/90

```
PROGRAM Graphics (input,output);
{###############################}

{$N-}
{$M 65500,0,15000}

USES
  crt,
  dos,
  graph,
  printer;

CONST
  arraymax  = 1200;

TYPE
  grapharrays   = array[0..arraymax] OF integer;
  dataarrays    = array[0..15] OF grapharrays;
  channelarrays = array[0..16] OF byte;

VAR
  regs          : registers;
  all,
  draft,
  overlap,
  autoscale,
  printeron     : boolean;
  cch,ch        : char;
  st,
  header,
  hournumber    : string;
  pos,
  maxX,
  maxY,
  minX,
  minY,
  Xsize,
  Ysize,
  first,
  last,
  minXlimit,
  maxXlimit,
  minYlimit,
  maxYlimit,
  maxsample,
  maxhour,
  channelpos    : integer;
  values        : grapharrays;
  dataarray     : dataarrays;
  channelarray  : channelarrays;


{########################################################################## initial }

PROCEDURE initial (VAR first,last,minXlimit,
{###############}       maxXlimit,minYlimit,maxYlimit : integer;
                   VAR printeron,autoscale,all        : boolean;
                   VAR channelarray                  : channelarrays);
VAR
  no : integer;

BEGIN
  printeron := false;
  draft     := true;
  overlap   := false;
  first     := 1;
  last      := 1;
  minXlimit := 0;
  maxXlimit := 60;
  minYlimit := 0;
  maxYlimit := 5000;
  autoscale := false;
  all       := false;
  FOR no := 0 TO 15 DO
    channelarray[no] := no;
  channelarray[16] := 255;
END;


{################################################################ enterspecs #}

PROCEDURE enterspecs (VAR maxhour,maxSample : integer);
{##################}

BEGIN
  clrscr;
  gotoXY(5,10);
  write('Enter the Number of Hours Data was Logged : ');
  readln(maxhour);
  gotoXY(5,12);
  write('Enter Number of Samples per Hour          : ');
  readln(maxSample);
END;


{##################################################################### hours #

PROCEDURE hours (VAR first,last : integer;
{#############}  VAR stop       : boolean);

BEGIN
  clrscr;
  gotoXY(5,10);
  write('Enter First Hour to be Displayed : ');
  readln(first);
  gotoXY(5,12);
  write('Enter Last Hour to be Displayed  : ');
  readln(last);
  IF (first > last) THEN stop := false;
END;


{############################################################### SelectRange #
```

```
PROCEDURE SelectRange (VAR minYlimit,maxYlimit : integer;
{####################} VAR stop                : boolean);

BEGIN
  gotoXY(5,14);
  write('Enter Minimum Y Value              : ');
  readln(minYlimit);
  gotoXY(5,16);
  write('Enter Maximun Y Value              : ');
  readln(maxYlimit);
  IF (minYlimit > maxYlimit) THEN stop := false;
END;

{################################################################ cursor_on #}

PROCEDURE cursor_on (turnon : boolean);
{#################}

{Switches the cursor on and off.}

VAR
  regs :registers;

BEGIN
  IF turnon THEN
    IF mem[0:$449] = 7 THEN regs.CX := $0C0D
      ELSE regs.CX := $0607
  ELSE regs.CX := $2000;
  regs.AX := $0100;
  intr($10,Dos.Registers(regs));
END;

{###################################################################### menu #}

PROCEDURE menu (VAR ch : char);
{############}

BEGIN
  clrscr;
  cursor_on(false);
  gotoXY(17,3);
  write('<1> Display All Hours Maximum Range');
  gotoXY(17,5);
  write('<2> Display All Hours Autoscaled');
  gotoXY(17,7);
  write('<3> Display All Hours at Given Range');
  gotoXY(17,9);
  write('<4> Display Selected Hours Maximum Range');
  gotoXY(17,11);
  write('<5> Display Selected Hours Autoscaled');
  gotoXY(17,13);
  write('<6> Display Selected Hours at Given Range');
  gotoXY(17,17);
  write('<C>hannel Selection');
  gotoXY(17,19);
  write('<O>verlapping    Or     <S>eparate Graphs');
  gotoXY(17,21);
  write('<A>ctivate       Or     <D>eactivate Printing Option? ');
  gotoXY(17,25);
  write('<X> Exit Program');
  ch := readkey;
  cursor_on(true);
END;


{####################################################### channelselection #}

PROCEDURE channelselection (VAR channelarray : channelarrays);
{#######################}

VAR
  no,
  pos,
  number,
  code   : integer;
  answer : string;
  stop   : boolean;

BEGIN
  FOR no := 0 TO 16 DO
    channelarray[no] := 255;
  pos := 0;
  REPEAT
    clrscr;
    gotoXY(7,10);
    write('Enter <A> for all Channels OR the Channel Number: ');
    readln(answer);
    IF (answer = 'A') OR (answer = 'a')
      THEN FOR no := 0 TO 15 DO
             channelarray[no] := no
      ELSE IF (answer <> '') THEN BEGIN
                                     val(answer,number,code);
                                     channelarray[pos] := number;
                                   END;
    inc(pos);
  UNTIL  (answer = 'A') OR (answer = 'a') OR (answer = '') OR (pos > 15);
  clrscr;
  gotoXY(27,2);
  write('Channels Selected');
  gotoXY(27,3);
  write('*****************');
  no := 0;
  WHILE channelarray[no] <> 255 DO
    BEGIN
      gotoxy(27,5 + no);
      write('Channel : ',channelarray[no]);
      inc(no);
    END;
  cursor_on(false);
  answer := readkey;
  cursor_on(true);
END;
```

```
{################################################################## setprint #}

PROCEDURE setprint (VAR printeron,draft : boolean);
{################}

VAR
  answer : char;
  stop   : boolean;

BEGIN
  printeron := true;
  REPEAT
    stop := true;
    clrscr;
    gotoXY(7,10);
    write('<D>raft or <H>igh Resolution? ');
    answer := readkey;
    CASE upcase(answer) OF
      'D' : draft := true;
      'H' : draft := false;
      ELSE stop := false;
    END;
  UNTIL stop;
  stop := false;
END;


{################################################################# userspecs #

PROCEDURE userspecs (VAR autoscale,all,printeron,draft,overlap  : boolean;
{#################}  VAR first,last,minYlimit,maxYlimit          : integer;
                     VAR ch                                      : char   );

VAR
  answer : char;
  stop   : boolean;

BEGIN
  REPEAT
    first := 1;
    stop  := false;
    menu(ch);
    CASE upcase(ch) OF
      '1' : BEGIN
              minYlimit := 0;
              maxYlimit := 5000;
              stop      := true;
              autoscale := false;
              all       := true;
            END;
      '2' : BEGIN
              stop      := true;
              autoscale := true;
              all       := true;
            END;
```

```
        '3' : BEGIN
                stop      := true;
                all       := true;
                autoscale := false;
                clrscr;
                SelectRange(minYlimit,maxYlimit,stop);
              END;
        '4' : BEGIN
                minYlimit := 0;
                maxYlimit := 5000;
                stop      := true;
                all       := false;
                autoscale := false;
                hours(first,last,stop);
              END;
        '5' : BEGIN
                stop      := true;
                all       := false;
                autoscale := true;
                hours(first,last,stop);
              END;
        '6' : BEGIN
                stop      := true;
                all       := false;
                autoscale := false;
                hours(first,last,stop);
                IF stop THEN SelectRange(minYlimit,maxYlimit,stop);
              END;
        'C' : channelselection(channelarray);
        'A' : setprint(printeron,draft);
        'D' : printeron := false;
        'O' : overlap := true;
        'S' : overlap := false;
        'X' : stop:= true;
       ELSE stop := false;
     END;
   UNTIL stop;
 END;


{################################################################## readvalues #

PROCEDURE readvalues (VAR dataarray : dataarrays;
{###################}  maxSample     : integer;
                      stno          : string    );

TYPE
  datarecords = RECORD
                  channel0,
                  channel1,
                  channel2,
                  channel3,
                  channel4,
                  channel5,
                  channel6,
                  channel7,
                  channel8,
                  channel9,
                  channel10,
                  channel11,
                  channel12,
                  channel13,
                  channel14,
                  channel15 : integer;
                END;

VAR
  name       : string;
  no,nno     : integer;
  datafile   : FILE OF datarecords;
  datarecord : datarecords;

BEGIN
  FOR no := 0 TO 15 DO
    FOR nno := 0 TO arraymax DO
      dataarray[no,nno] := 0;
  IF (st = '100')
    THEN BEGIN
           clrscr;
           gotoxy(10,12);
           write('Enter the Name of the Datafile : ');
           readln(name);
           assign(datafile, name + '.dat');
         END
    ELSE assign(datafile,'datfil' + stno + '.dat');
  reset(datafile);
  FOR no := 0 TO (maxSample - 1) DO
    BEGIN
      read(datafile,datarecord);
      WITH datarecord DO
        BEGIN
          dataarray[0,no]  := channel0;
          dataarray[1,no]  := channel1;
          dataarray[2,no]  := channel2;
          dataarray[3,no]  := channel3;
          dataarray[4,no]  := channel4;
          dataarray[5,no]  := channel5;
          dataarray[6,no]  := channel6;
          dataarray[7,no]  := channel7;
          dataarray[8,no]  := channel8;
          dataarray[9,no]  := channel9;
          dataarray[10,no] := channel10;
          dataarray[11,no] := channel11;
          dataarray[12,no] := channel12;
          dataarray[13,no] := channel13;
          dataarray[14,no] := channel14;
          dataarray[15,no] := channel15;
        END;
    END;
  close(datafile);
END;


{################################################################## getname #
```

```
PROCEDURE getname (channel     : integer;
{###############}  VAR header : string);

BEGIN
  CASE channel OF
    0  : header := 'Soak Zone West Setpoint';
    1  : header := 'Soak Zone West Temperature';
    2  : header := 'Soak Zone East Setpoint';
    3  : header := 'Soak Zone East Temperature';
    4  : header := 'Top Heating Zone Setpoint';
    5  : header := 'Top Heating Zone Temperature';
    6  : header := 'Bottom Heating Zone Setpoint';
    7  : header := 'Bottom Heating Zone Temerature';
    8  : header := 'Top Preheating Zone Setpoint';
    9  : header := 'Top Preheating Zone Temperature';
    10 : header := 'Bottom Preheating Zone Setpoint';
    11 : header := 'Bottom Preheating Zone Temperature';
    12 : header := 'Furnace Pressure';
    13 : header := 'Main Line Mixed Gas Pressure';
    14 : header := 'Bottom Heating Zone Air Flow';
    15 : header := 'Bottom Heating Zone Gas Flow';
    ELSE BEGIN
           clrscr;
           writeln('ERROR in getname Procedure!');
           halt;
         END;
  END;
END;


{################################################################## findmax #

PROCEDURE findmax (values       : grapharrays;
{###############}  VAR max,min : integer;
                   maxSample   : integer;
                   overlap,
                   last        : boolean      );

VAR
  no,
  factor : integer;

BEGIN
  IF NOT overlap THEN BEGIN
                        max := 0;
                        min := values[0];
                      END;
  FOR no := 0 TO (maxSample - 1) DO
    BEGIN
      IF (values[no] > max)
        THEN max := values[no]
        ELSE IF (values[no] < min) THEN min := values[no];
    END;
  IF last AND ((max - min) < 1000)
    THEN BEGIN
           factor := round((1000 - (max - min)) / 2);
           min    := min - factor;
           max    := max + factor;
         END;
END;


{################################################################# GraphInit #}

PROCEDURE GraphInit(VAR maxX,maxY,minX,minY,Xsize,Ysize : integer);
{#################}

VAR
  GraphDriver,
  GraphMode    : integer;


BEGIN
  GraphDriver := detect;
  DetectGraph(GraphDriver,GraphMode);
  CASE Graphdriver OF
    1,2    : BEGIN
               Xsize := 600 {320};
               Ysize := 200 {200};
             END;
    3,4,5 : BEGIN
               Xsize := 640;
               Ysize := 350;
             END;
    7      : BEGIN
               Xsize := 720;
               Ysize := 348;
             END;
    ELSE BEGIN
           clrscr;
           writeln('Cannot determine what Graphics Card is used!');
           halt;
         END;
  END;
  maxX := round(0.95 * Xsize);
  maxY := round(0.10 * Ysize);
  minX    := round(0.10 * Xsize);
  minY    := round(0.90 * Ysize);
  InitGraph(GraphDriver,GraphMode,'');
  SetViewPort(0,0,GetMaxX,GetMaxY,true);
END;


{################################################################## graphics #

PROCEDURE graphics (minXval,maxXval,minYval,maxYval,
{#################}  maxSample,maxX,maxY,minX,minY,Xsize,Ysize : integer;
                    values                                    : grapharrays;
                    st,hournumber                             : string);

 {********************************************************************* axis *
 PROCEDURE axis (minX,minY,maxX,maxY,maxXval,maxYval,minXval,minYval : integer
{************}  Xunit,Yunit : string                                    );

VAR
  no           : integer;
  number       : string;
  interval,
  Xdivision,
  Ydivision    : real;

BEGIN
  line(minX,maxY,minX,minY);
  line(minX,minY,maxX,minY);
  Xdivision := (maxX - minX) / 10;
  FOR no := 1 to 10 DO
    line(round(Xdivision*no)+minX,minY-3,round(Xdivision*no)+minX,minY+3);
  Ydivision := (minY - maxY) / 10;
  FOR no := 1 to 10 DO
    line(minX-3,minY-round(Ydivision*no),minX+3,minY-round(Ydivision * no));
  SetTextJustify(1,2);
  interval := (maxXval - minXval) / 10;
  FOR no := 0 TO 10 DO
    BEGIN
      str(round(minXval + no * interval),number);
      MoveTo(round(minX + no * Xdivision),minY + 6);
      OutText(number);
    END;
  MoveTo(maxX,minY + 20);
  OutText(Yunit);
  SetTextJustify(0,1);
  interval := (maxYval - minYval) / 10;
  FOR no := 0 TO 10 DO
    BEGIN
      str(round(minYval + no * interval),number);
      MoveTo(minX - 40,round(minY - no * Ydivision));
      OutText(number);
    END;
  SetTextJustify(1,2);
  MoveTo(minX - 25,maxY - 20);
  OutText(Xunit);
END;


{******************************************************************** header *
PROCEDURE header (st,hournumber : string;
{**************}  Xsize,Ysize   : integer);

BEGIN
  SetTextJustify(1,2);
  SetTextStyle(0,0,1);
  MoveTo(round(Xsize/2),0);
  OutText(st);
  SetTextStyle(0,0,1);
  MoveTo(Xsize - (round(TextWidth(hournumber)/2) + 1),0);
  OutText(hournumber);
END;
```

```
 {************************************************************** PlotValues *}
 PROCEDURE PlotValues (values : grapharrays;
 {******************}  maxX,maxY,minX,minY : integer);

 VAR
   xstep,
   ystep : real;
   no    : integer;

 BEGIN
   xstep := (maxX - minX) / maxSample;
   ystep := (minY - maxY) / (maxYval - minYval);
   MoveTo(minX,round(minY - (values[0] - minYval) * ystep));
   FOR no := 1 TO (maxSample - 1) DO
     BEGIN
       IF (values[no] > maxYval) THEN MoveTo(round((minX + no * xstep)),maxY)
         ELSE IF (values[no] < minYval) THEN MoveTo(round(minX + no * xstep),mi
Y)
                 ELSE LineTo(round(minX + no * xstep),round(minY - (values[no]
 m
     END;
 END;


{################################################################ graphics #}

BEGIN
  axis(minX,minY,maxX,maxY,maxXval,maxYval,minXval,minYval,'mV','minutes');
  header(st,hournumber,Xsize,Ysize);
  PlotValues(values,maxX,maxY,minX,minY);
END;


{##.  ############################################################ printgraph #}

PROCEDURE printgraph (Xsize,Ysize : integer;
{##################}  draft       : boolean );

VAR
  x,
  y,
  n1,
  n2,
  no,
  dots,
  line,
  point,
  value,
  column  : integer;
  needles : byte;

BEGIN
  x := 0;
  write(lst,chr(27),chr(51),chr(24));
  IF draft THEN BEGIN
                  value := 75;
                  dots  := Ysize;
                END
           ELSE BEGIN
```

```
                  value := 90;
                  dots  := 4 * Ysize;
                END;
  n1 := dots MOD 256;
  n2 := dots DIV 256;
  FOR line := 1 to round(Xsize / 8) DO
    BEGIN
      write(lst,chr(27),chr(value),chr(n1),chr(n2));
      FOR column := (Ysize - 1) DOWNTO 0 DO
        BEGIN
          y       := column;
          needles := 0;
          FOR no := 0 TO 7 DO
            BEGIN
              point   := GetPixel(x + no,y);
              needles := needles * 2;
              IF (point <> 0) THEN needles := needles + 1;
            END;
          IF (needles = 26) THEN needles := 10;
          write(lst,chr(needles));
          IF NOT draft THEN BEGIN
                              write(lst,chr(needles));
                              write(lst,chr(needles));
                              write(lst,chr(needles));
                            END;
        END;
        x := line * 8;
        writeln(lst);
     END;
   write(lst,chr(27),chr(5));
   writeln(lst,chr(12));
END;


{################################################################# askheader #}

PROCEDURE askheader (VAR header : string);
{#################}

BEGIN
  clrscr;
  gotoXY(5,10);
  write('Enter the Graph Heading : ');
  readln(header);
END;


{############################################################## main program #
BEGIN
  enterspecs(maxhour,maxsample);
  initial(first,last,minXlimit,maxXlimit,minYlimit,maxYlimit,printeron,
          autoscale,all,channelarray);
  REPEAT
    userspecs(autoscale,all,printeron,draft,overlap,first,last,minYlimit,
              maxYlimit,ch);
    IF overlap AND (upcase(ch) <> 'X') THEN askheader(header);
    IF all THEN last := maxhour;
    IF (maxhour = 100) THEN first := 100;
    cch := ' ';
    WHILE (upcase(ch) <> 'X') AND (upcase(cch) <> 'X') AND (first <= last) DO
      BEGIN
        str(first,st);
        hournumber := 'Hour ' + st;
        readvalues(dataarray,maxsample,st);
        channelpos := 0;
        GraphInit(maxX,maxY,minX,minY,Xsize,Ysize);
        cch := ' ';
        IF autoscale AND overlap
          THEN BEGIN
                 pos       := 0;
                 maxYlimit := 0;
                 minYlimit := dataarray[channelarray[0]][0];
                 WHILE (channelarray[pos] <> 255) DO
                   BEGIN
                     findmax(dataarray[channelarray[pos]],maxYlimit,
                             minYlimit,maxSample,true,false);
                     inc(pos);
                   END;
                 END;
        WHILE (upcase(cch) <> 'X') AND (channelarray[channelpos] <> 255) DO
          BEGIN
            IF NOT overlap THEN getname(channelarray[channelpos],header);
            IF autoscale AND NOT overlap
              THEN findmax(dataarray[channelarray[channelpos]],maxYlimit,
                           minYlimit,maxSample,false,true);
            graphics(minXlimit,maxXlimit,minYlimit,maxYlimit,maxsample,maxX,
                     maxY,minX,minY,Xsize,Ysize,dataarray[channelarray[channel
s]
                     header,hournumber);
            IF NOT overlap
              THEN BEGIN
                     IF printeron THEN printgraph(Xsize,Ysize,draft)
                                ELSE cch := readkey;
                     ClearViewport;
                   END;
            inc(channelpos);
          END;
        IF overlap
          THEN IF printeron THEN printgraph(Xsize,Ysize,draft)
                            ELSE cch := readkey;
        inc(first);
        CloseGraph;
        RestoreCrtMode;
      END;
  UNTIL (upcase(ch) = 'X');
END.
```

APPENDIX E

°C	CO_2	H_2O	N_2	O_2	H_2	CO	CH_4	SKW	C_2H_4	C_2H_6	Air
0	0,387	0,356	0.311	0,313	0.306	0,311	0,372	0,471	0,444	0,532	0,311
100	0,414	0,359	0,311	0,315	0,308	0,312	0,393	0,530	0,500	0,600	0,312
200	0,422	0,363	0.312	0,319	0,310	0,313	0,420	0,589	0,556	0,668	0,313
300	0,450	0,368	0.313	0,324	0,310	0,315	0.452	0,645	0.607	0,733	0,315
400	0,467	0,373	0,315	0.329	0,311	0,318	0.482	0,695	0.653	0,794	0,318
500	0,482	0,379	0,318	0,334	0,312	0,321	0,512	0,744	0,696	0,852	0,321
600	0,494	0,385	0,321	0,339	0,313	0,325	0,541	0,789	0,735	0,913	0,324
700	0,506	0,391	0,325	0,343	0,314	0,328	0,570	0,828	0,769	-	0,328
800	0,516	0,397	0,328	0,347	0,315	0,332	0,595	0,865	0,802	-	0,331
900	0,525	0,403	0,331	0,351	0,317	0,335	0,620	-	0,833	-	0,334
1000	0,533	0,410	0,334	0,354	0,318	0,338	0,642	-	0,861	-	0,337
1200	0,547	0,422	0,340	0,350	0,321	0,344	0,684	-	0,914	-	0,343
1400	0,558	0,434	0,344	0,364	0,325	0,349	0,720	-	-	-	0,348
1600	0,568	0,445	0,349	0,368	0,320	0,353	-	-	-	-	0,352
1800	0,576	0,455	0,353	0,372	0,333	0,357	-	-	-	-	0,356
2000	0,583	0,465	0,356	0,376	0,336	0,360	-	-	-	-	0,359

APPENDIX F

Listing of SIM2.PAS, page 1 at 12:31pm 03/09/90

```
{-----------------------------------------------------------------------}
{                                                                       }
{ This is a skeleton time-domain simulation program along similar       }
{ lines to ACSL or CSMP. However there is no translation, everything    }
{ is directly in PASCAL-2.                                              }
{ Only very limited portions of this program have to be changed to      }
{ simulate a particular system - usually only procedures DYNAMICS,      }
{ INITIALIZATION, MODEL, PLOT and ENDING.                               }
{ PLOT has to be  taylored to achieve the required output.              }
{                                                                       }
{    Written by .RC De Almeida                         20/03/1986       }
{                                                                       }
{-----------------------------------------------------------------------}

program sim(input, output);
{$N-}
{$M 65500,0,15000}

 USES
   crt,
   dos,
   graph,
   printer;

 {$I simvar2.pas}
> { this file is included with the simulation file and user files          }
> { so that both may have access to the following global variables:        }
>
> {      NSTP - number of DELTA intervals per logging interval (user set) }
>
> {      NSTATE - number of states (user set)                              }
>
> {      METHOD - to define type of numerical technique (user set)         }
> {                    * 'E' - EULER                                       }
> {                    * 'R' - RUNGE-KUTTA                                  }
>
> {      T - independent variable (initial value is user set)              }
>
> {      LOG - time span between consecutive logging (user set)            }
>
> {      TSTOP - how long should the simulation run for (user set)         }
>
> {      X - array of states (initial values are user set)                 }
>
> {      D - array of state derivatives (used by user to define model)     }
>
> {      Q - array of inputs to the system (user set)                      }
>
> {      STOP - flag to define when simulation should finish (may be user set)
>
> TYPE
>   datarecords = RECORD
>                     channel0,
>                     channel1,
>                     channel2,
>                     channel3,
>                     channel4,
```

Listing of SIM2.PAS-include file SIMVAR2.PAS, page 2 at 12:31pm 03/09/90

```
>                         channel5,
>                         channel6,
>                         channel7,
>                         channel8,
>                         channel9,
>                         channel10,
>                         channel11,
>                         channel12,
>                         channel13,
>                         channel14,
>                         channel15 : integer;
>                       END;
>
> var
>   NSTP, NSTATE: integer;
>
>   STOP: boolean;
>
>   METHOD: char;
>
>   T, LOG, TSTOP: real;
>
>   D, X, Q: array [1..20] of real;
>
>   { variables used by sim.pas }
>
>   DFLT: real;
>
>   { variables by user in user.pas }
>
>   { user list }
>   burner,
>   exhaust,
>   k2,
>   k3,
>   k4,
>   air,
>   gas         : real;
>   fname       : string;
>   datafile    : FILE OF datarecords;
>   datarecord  : datarecords;

  {$I user2.pas}
>
> {------------------------------------------------------------}
> {                                                            }
> { This is the user-dependent part of the simulation program. }
> { There are four procedures that are available for the user  }
> { to define his system, viz. INITIALIZATION, DYNAMIC, MODEL  }
> { PLOT and ENDING.                                           }
> {                                                            }
> { Written by ARC De Almeida                    20/03/1986    }
> {                                                            }
> {------------------------------------------------------------}
>
>
```

SIM2.PAS-include file USER2.PAS page 2

```
> {--------------------------------------------------------------------}
> {                                                                    }
> { In this procedure the user initialises any data that he might need }
> { at a later stage.                                                  }
> { The following parameters should be initialised to his needs.       }
> {         T, TSTOP, LOG, NSTATE, NSTP and METHOD.                    }
> { The initial values for the states should also be given here.       }
> {                                                                    }
> {--------------------------------------------------------------------}
>
>
> procedure INITIALIZATION;
>
>   begin
>     { user initialization in PASCAL statements }
>     clrscr;
>     gotoxy(10,12);
>     fname := 'ttest';
>     assign(datafile, fname + '.dat');
>     rewrite(datafile);
>     X[1]   := 55;
>     T      := 0;
>     Tstop  := 3600.0;
>     Nstate := 1;
>     log    := 5;
>     nstp   := 1;
>     method := 'R';
>     WITH datarecord DO
>         BEGIN
>           channel0 := 0;
>           channel1 := 0;
>           channel2 := 0;
>           channel3 := 0;
>           channel4 := 0;
>           channel5 := 0;
>           channel6 := 0;
>           channel7 := 0;
>           channel8 := 0;
>           channel9 := 0;
>           channel10 := 0;
>           channel11 := 0;
>           channel12 := 0;
>           channel13 := 0;
>           channel14 := 0;
>           channel15 := 0;
>         END;
>   end;
>
> {--------------------------------------------------------------------}
> {                                                                    }
> { This procedure holds the code to be executed every communication   }
> { interval. It is useful for calculating values derived from state   }
> { variables for printout purposes or to simulate digital control     }
> { systems.                                                           }
> { It can also be used to define your input signal if this is a step. }
> {                                                                    }
> {--------------------------------------------------------------------}
```

```
>
>
> procedure DYNAMICS; { User program in PASCAL code }
>
>
>   begin
>     IF ((T >= 768) AND (T<=2225))
>       THEN BEGIN
>               IF (air < 1.715)
>                 THEN air := air + 0.04
>                 ELSE IF (air < 2.202)
>                        THEN air := air + 0.015
>                        ELSE IF (air < 2.331)
>                               THEN air := air + 0.00379
>                               ELSE IF (air < 2.719)
>                                      THEN air := air + 0.015
>                                      ELSE air := 2.719;
>               gas := 0.521;
>             END
>       ELSE BEGIN
>               IF (T > 2225)
>                 THEN IF (air > 2.35)
>                        THEN air := 2.35
>                        ELSE IF (air > 1.6)
>                               THEN air := air - 0.0228
>                               ELSE IF (air > 1.228)
>                                      THEN air := air - 0.00379
>                                      ELSE air := 1.228
>                 ELSE air := 1.228;
>               gas := 0.264;
>             END;
>   end;
>
> {------------------------------------------------------------------}
> {                                                                  }
> { This procedure is where the dynamic model of the plant should be. }
> { The form of the equation should be:                              }
> {                                                                  }
> {   D = f ( T, Q, X)                                               }
> {                                                                  }
> { where D is the time-derivative-of-the-states vector              }
> {       X is the state vector                                      }
> {       Q is the input vector                                      }
> {   and T is the independent variable (time).                      }
> {                                                                  }
> {------------------------------------------------------------------}
>
>
> procedure MODEL;
>
>   begin
>     IF (gas <> 0)
>       THEN IF (air / gas >= 4.307) THEN burner := 4395 * gas
>                                    ELSE burner := 1020 * air
>       ELSE burner := 0;
>
>     D[1] := (burner - 15.7163 * X[1] - 0.367 * (0.634 * gas + air) * 900
>               + 0.328 * gas * 35 + 0.318 * air * 400) / 1100;
>   end;
>
> {------------------------------------------ --------------------}
> {                                                                }
> { This is the printing routine and can be used to print the results }
> { either on the screen or into files.                            }
> {                                                                }
> {----------------------------------------------------------------}
>
>
> procedure PLOT;
>
>   begin
>     { User program in PASCAL code }
>
>     datarecord.channel0 := trunc((X[1] + 175) * 8 + 1000);
>     datarecord.channel1 := trunc(air * 411.43 + 1000);
>     datarecord.channel2 := trunc(gas * 2400 + 1000);
>     write(datafile,datarecord);
>
>   end;
>
> {----------------------------------------------------------------}
> {                                                                }
> { This is the final procedure that is run once at end of the program }
> {                                                                }
> {----------------------------------------------------------------}
>
> -- Illegal nested include of file SGRAPHICS.PAS --
>
> procedure ENDING;
>
>   begin
>
>     { User program in PASCAL code }
>
>     close(datafile);
>
>     display;
>
>   end;

 { This procedure uses the 1st order EULER method to integrate }
 { over the calculation interval (DELT).                       }


  procedure EULER;

    var
      I: integer;


    begin
      MODEL;
      for I := 1 to NSTATE do
      X[I] := X[I] + DELT * D[I];

    T := T + DELT;
  end;

{ This procedure uses the 4th order Runge-Kutta method to integrate }
{ over the calculation interval (DELT).                             }
{ Ref: C. Froberg, Introduction to Numerical Analysis,              }
{      Addison Wesley, 1973, p268.                                  }


procedure RKUTTA;

  var
    I: integer;
    XSTART, K1, K2, K3: array [1..20] of real;


  begin

    MODEL;

    for I := 1 to NSTATE do
      begin
      XSTART[I] := X[I];
      K1[I] := D[I] * DELT;
      X[I] := XSTART[I] + K1[I] / 2.0;
      end;

    T := T + DELT / 2.0;

    MODEL;

    for I := 1 to NSTATE do
      begin;
      K2[I] := D[I] * DELT;
      X[I] := XSTART[I] + K2[I] / 2.0;
      end;

    MODEL;

    for I := 1 to NSTATE do
      begin
      K3[I] := D[I] * DELT;
      X[I] := XSTART[I] + K3[I];
      end;

    T := T + DELT / 2.0;
    MODEL;

    for I := 1 to NSTATE do
      begin;
      X[I] := XSTART[I] + (K1[I] + K2[I] * 2.0 + K3[I] * 2.0 + D[I] *
              DELT) / 6.0;
      end;
  end;
```

```
{ This procedure performs the integration over one communication }
{ interval. It uses the RUNGE-KUTTA or the EULER routines to do  }
{ this in 'NSTP' calculation steps.                              }
{ At a later stage other integration routines might be made      }
{ available for selection at this point.                         }

procedure INTEGRATION;

  label leave;

  var
    I: integer;

  begin

    DELT := LOG / NSTP; { giving a value for DELT }

    for I := 1 to NSTP do
      begin
      case METHOD of

        'E', 'e':
          EULER; { calling the Euler method}
        'R', 'r':
          RKUTTA; { calling the Runge-Kutta method}

        end;

      if T >= TSTOP then
        begin
          stop := true;
          goto leave;
        end;

      end;

  leave:

  end;

{ Main program }

begin

  clrscr;

  STOP := false;

  { Execute the initialization given by the user }

  INITIALIZATION;

  { Main integration loop, done until STOP flag is set true }
  while (not STOP) do
    begin
   { Execution of user defined dynamic block (done every logging interval) }
    DYNAMICS;
    { Execution of user defined logging procedure }
    PLOT;
    { Actual integration over one logging interval }
    INTEGRATION;
    end;
  PLOT;
  ENDING;
end.
```

APPENDIX G

Listing of CONTROL.PAS, page 1 at 12:47pm 03/09/90

```
{############################################################################
PROCEDURE adaptive_controller(VAR gas : real);
{###########################}

VAR
  synth  : byte;


{******************************************************************** apcon0
PROCEDURE apcon0 (VAR A     : apcrec;
{**************}  Uini,
                  Yini,
                  Zini : real  );

{Performs all initialization required before the first iteration of the advanc
 process controller algorithm.}


{---------------------------------------------------------------------- RLS0
PROCEDURE RLS0 (VAR A : apcrec);
{------------}

{Performs initialization for the Recursive Least Squares parameter estimation}

VAR
  E0,
  E1 : real;
  j  : integer;

BEGIN
  A.BJ  := 1.0;
  A.A00 := 0.01;
  A.A10 := 0;
  A.B00 := 0.01;
  A.B10 := 0;
  FOR j := 1 TO npar DO
    BEGIN
      A.theta[j] := 0.0;
      A.phik1[j] := 0.0;
      A.K[j]     := 0.0;
      A.D[j]     := A.ct2;
    END;
  A.D[1] := 0.0;
  A.D[3] := 0.0;
  A.D[5] := 0.0;
  A.D[6] := 0.0;
  For j := 1 TO NUvec DO
    A.Uvec[j] := 0.0;
  E1 := 2.0 * 0.71 * A.WE;
  E0 := sqr(A.WE);
  WITH A DO
    BEGIN
      eavec[1] := -E1;
      eavec[2] := -E0;
      gam[1]   := 0.0;
      gam[2]   := E0;
      gam[3]   := 0.0;
      gam[4]   := E0;
      gam[5]   := gamma * E0;
      gam[6]   := E0;
      theta[1] := 0.0;
      theta[2] := -A00 / gam[2];
      theta[3] := 0.0;
      theta[4] := B00 / gam[4];
      theta[5] := 0.0;
      theta[6] := 0.0;
    END;
  randomize;
END;


{********************************************************************** apcon0 *
BEGIN
  A.FFenb    := 0;
  A.delta    := 5;
  A.WE       := 0.05;
  A.EPSHP    := 0.0025;
  A.GAMMA    := 100;


  A.Astar[1]:= 1;
  A.Astar[2]:= 0.03;
  A.Astar[3]:= 2E-4;


  A.alpha    := 0.4;
  A.beta     := 1.3;
  A.sigma    := 0.98;
  A.eps0     := 0.05;
  A.eps1     := 1E-4;
  A.eps2     := 0.0;
  A.ct2      := 100.0;
  A.reg      := 1000;
  A.ffac     := 0.985;

  A.Ybar   := 0;
  A.Xhp1   := Yini / A.epshp;
  A.Xhp2   := Uini / A.epshp;
  A.Xhp3   := Zini / A.epshp;
  A.YbarP  := 0.0;
  A.sigi   := 1.0 - A.sigma;    {initializing dead-zone variables}
  A.MZ0    := 0.1 * A.eps0;
  RLS0(A);                      {initialize recursive least square aspects}
END;

{********************************************************************** apcon1 '
PROCEDURE apcon1 (VAR A         : apcrec;
{**************}  VAR Y         : real;
                  VAR synth     : byte;
                  VAR K1,K2,K3  : real    );

{This Procedure implements the time-critical part of the controller adaption
 algorithm. The amount of computation from the time the measured variables are
 read to the time the controller settings are updated is kept to a minimum.}

VAR
  A1, A0, B1, B0, F1, F0,            {Plant parameter estimates}
  E,                                 {Prediction Error}
  BMZ,                               {Dead zone width}
  CP0, CP1, CP2,                     {Controller coefficients}
  X,                                 {CL observer pole position}
  XB01, XB10, CA, CB       : real;

{-------------------------------------------------------------------------- RLS1 -
PROCEDURE RLS1 (VAR A                         : apcrec;
{------------}  VAR E,A1,A0,B1,B0,F1,F0 : real    );

{Use Kalman gains previously calculated by RLS2 to calculate the new parameter
 estimates.}

VAR
  EE : real;
  j  : integer;

BEGIN
  WITH A DO
    BEGIN
      EE := E * Azone / BJ;
      FOR j := 1 TO npar DO
        theta[j] := theta[j] + EE * K[j];
      A1 := -theta[1] * gam[1];
      A0 := -theta[2] * gam[2];
      B1 := theta[3] * gam[3];
      B0 := theta[4] * gam[4];
      F1 := theta[5] * gam[5];
      F0 := theta[6] * gam[6];
    END;
END;

{-------------------------------------------------------------------------- dead -
FUNCTION dead (limit, inp : real) : real;
{-----------}

{This Function generates a dead-zone non-linearity. Between the dead band
 limits, output is zero. Outside these limits, the output is a linear function
 of the input, offset by the closer deadband limit.

       y = x + lim           if x < -lim
       y = 0                 if |x| <= lim
       y = x - lim           if x > lim

VAR
  outp : real;

BEGIN
  IF (inp < -limit)
    THEN outp := inp + limit
    ELSE IF (inp > limit)
```

```
          THEN outp := inp - limit
          ELSE outp := 0.0;
  dead := outp;
END;


{*********************************************************************** apconl }
BEGIN
  WITH A DO
    BEGIN
      {High Pass Filter the measured variable Y}
      Ybar := -epshp * Xhp1 + Y;
      Xhp1 := Xhp1 + delta * Ybar;

      {Calc the estimator prediction error}
      E    := phik1[1] - YbarP;

      {Implement the relative dead zone}
      BMZ := beta * MZ0;
      IF (abs(E) >= BMZ) OR (synth = 2)
        THEN BEGIN
               Azone := alpha / E * dead(BMZ,E);
               RLS1(A,E,A1,A0,B1,B0,F1,F0);
               Zflag := 1;
               IF (synth <> 0)        {Synthesise the controller poly coeffs}
                 THEN BEGIN
                        {Astar[2] := A0 + 1 / Tuser;
                         Astar[3] := A0 / Tuser;}

                         CP0      := Astar[3] / B0;
                         CP1      := (Astar[2] - A0) / B0;
                         {Convert to PID controller settings}
                         K1 := CP1;               {Gain}
                         IF (K1 > 100)
                           THEN K1 := 100
                           ELSE IF (K1 < 0) THEN K1 := 0.1;
                         K2 := CP1 / CP0;         {Ti  }
                         IF (K2 > 1000)
                           THEN K2 := 1000
                           ELSE IF (K2 < 0) THEN K2 := 1000;
                         K3 := 0.0;               {Td  }
                         IF (FFenb <> 0)
                           THEN BEGIN
                                  IF (XB10 >= 0.5/WE)
                                    THEN TFF2 := XB10  {Plant zero signif.}
                                    ELSE TFF2 := FFTAU;
                                  IF (abs(F0) > 1.0E-10)
                                    THEN TFF1 := F1 / F0
                                    ELSE TFF1 := 0.0;
                                  KFF := F0 / B0;
                                END;
                      END;
             END
        ELSE Zflag := 0;
    END
END;
```

```
{*********************************************************************** apcon2 *
PROCEDURE apcon2 (VAR A          : apcrec;
{**************}  U,Y,Z          : real    );

{This Procedure executes the non-time-critical section of the controller
 adaption algorithm. This includes the bulk of the parameter estimation work.}

VAR
  Ubar,
  Zbar : real;


{---------------------------------------------------------------------------- RLS2 -
PROCEDURE RLS2 (VAR A          : apcrec;
{------------}  VAR UB,YB,ZB : real    );

{Implements a 4 step procedure to calculate the Kalman gains and update the
 regression vector for the Recursive Least Squares parameter estimation.}

TYPE
  phiks = ARRAY[1..npar] OF real;

VAR
  phik : phiks;
  FJ,
  GJ,
  BJ1,
  MUJ,
  W,
  PY,
  PU,
  PZ   : real;
  i,
  j,
  LF,
  LU   : integer;

BEGIN
  WITH A DO
    BEGIN
      {STEP 1}
      {Update phik1 for next time using only phik1 itself, not phik}
      PY := YB;
      PU := UB;
      PZ := ZB;
      FOR j := 1 TO NA DO
        BEGIN
          PY := PY + EAvec[j] * phik1[j];
          PU := PU + EAvec[j] * phik1[NA + j];
          PZ := PZ + EAvec[j] * phik1[NA + NA + j];
        END;
      PY := phik1[1] + delta * PY;
      PU := phik1[NA + 1] + delta * PU;
      PZ := phik1[NA + NA + 1] + delta * PZ;
      FOR j := npar DOWNTO 2 DO
        phik1[j] := phik1[j] + delta * phik1[j-1];
  phik1[1]           := PY;
  phik1[NA + 1]      := PU;
  phik1[NA + NA + 1] := PZ;

  {STEP 2}
  {Get the current phik and apply the Gamma factors}
  FOR j := 1 TO npar DO
    phik[j] := phik1[j] * gam[j];

  {STEP 3}
  {Calc YbarP for next time using phik}
  YbarP := 0.0;
  FOR j := 1 TO npar DO
    YbarP := YbarP + phik[j] * theta[j];

  {STEP 4}
  {Recursive Least Squares algorithm}
  {It uses phik to calc the estimation gain vector K}
  {Bierman's U' * D * U covariance update algorithm is used}
  IF (Zflag <> 0)
    THEN BEGIN
           FJ   := phik[1];
           GJ   := D[1] * FJ;
           K[1] := GJ;
           BJ   := 1.0 + GJ * FJ;
           D[1] := D[1] / BJ / FFAC;

           {From here on we assume that npar is always > 1}
           LF := 0;
           LU := 0;
           FOR j := 2 TO npar DO
             BEGIN
               FJ := PHIK[j];
               FOR i := 1 TO (j-1) DO
                 BEGIN
                   inc(LF);
                   FJ := FJ + phik[i] * Uvec[LF];
                 END;
               GJ   := FJ * D[j];
               K[j] := GJ;
               BJ1  := BJ;
               BJ   := BJ + GJ * FJ;
               D[j] := D[j] * BJ1 / BJ /FFAC;
               IF (D[j] > reg) THEN D[J] := reg; {Regularization necessary
               MUJ  := -FJ * Azone / BJ1;
               FOR i := 1 TO (j-1) DO
                 BEGIN
                   inc(LU);
                   W        := Uvec[LU] + K[i] * MUJ;
                   K[i]     := K[i] + Uvec[LU] * GJ;
                   Uvec[LU] := W;
                 END;
             END;
         END
    ELSE FOR j := 1 TO npar DO
           K[j] := 0.0;
END;
```

```
END;

{*************************** ********************************************** apcon2 *
BEGIN
  WITH A DO
    BEGIN

      {High-pass filter the control input U}
      Ubar := -epshp * Xhp2 + U;
      Xhp2 := Xhp2 + delta * Ubar;

      {High-pass filter the feed forward measurement Z}
      Zbar := -epshp * Xhp3 + Z;
      Xhp3 := Xhp3 + delta * Zbar;

      {Calculate the dead zone width}
      MZ0  := sigma * MZ0 + sigi * (eps0 + eps1 * abs(Ubar) + eps2 * abs(Y));

      {Do parameter estimation if the prediction error was large enough}
      RLS2(A,Ubar,Ybar,Zbar);
    END;
END;


{*********************************************************************** PIDalg *
PROCEDURE PIDalg (VAR Yout,CEF,CDF1 : real;
{**************}  setp,measured,Cfac,K1,K2,K3,ddelta : real);

{An incremental PID algorithm with a 1st order L.P. filter on all terms. A
 backward difference approximation to the continuous-time equation and highly
 optimized (frac) arithmetic is used.}


CONST
  Umin = 0;
  Umax = 1;

VAR
  Uout,
  CE,
  CDF,
  res : real;


BEGIN
  CE   := setp - measured;
  CDF  := Cfac * (CE - CEF);                           {filtered (1-q**-1)*CE}
  Uout := Yout + K1 * (CDF + Ddelta/K2*CEF + K3/Ddelta * (CDF-CDF1));
  CEF  := CEF + CDF;                                   {filtered CE}
  CDF1 := CDF;
  IF (Uout < Umin)
    THEN Yout := Umin
    ELSE IF (Uout > Umax)
           THEN Yout := Umax
           ELSE Yout := Uout;
END;
```

```
{############################################################ main procedure ;
BEGIN
  If (T=0)
    THEN BEGIN
           apcon0(A,0.26,1152/2000,0);
           Tuser := 50;
           treff := 54/200;
           Yout  := 0.264;
           CEF   := 0;
           CDF1  := 0;
           k1    := 8;
           k2    := 120;
           k3    := 0;
         END;

  Yplant := X[1] / 200;

  IF ((trunc(T)+450) MOD 500 = 0)
    THEN IF (treff = 104 / 200)
           THEN treff := 54 / 200
           ELSE treff := 104 / 200;
  IF (T > 900)
    THEN synth := 1
    ELSE IF (T = 900)
           THEN synth := 2
           ELSE synth := 0;
  apcon1(A,Yplant,synth,K1,K2,K3);
  PIDalg(Yout,CEF,CDF1,treff,Yplant,0.3,K1,K2,K3,A.delta);
  apcon2(A,Yout,Yplant,0);
  gas := Yout;

END;
{###########################################################################}
```

REFERENCES

Bergesen, M.L. (1988) The Implementation of a Generalised Robust Adaptive Controller, University of the Witwatersrand Master Dissertation

Fahien, W. (1983) Fundamentals of Transport Phenomena, Mc Graw-Hill

BIBLIOGRAPHY

Carpenter, D.G. (August 1987) Temperature Control and Optimization of a Reheat Furnace Using a Distributed Control System, Iron and Steel Engineer, pp. 44 - 49

Dorf, R.C. (1980) Modern Control Systems 3rd Edition, Addison Wesley Publishing Company

Harris, C.J. Billings, S.A. (1981) Self-Tuning and Adaptive Control Theory and Applications, Peter Peregrinus

Millamon, P. (1986) Micro-Computer Controlled Operation of the Steel Reheating Furnaces, World Steel & Metalworking Vol.7, pp. 145 - 148

Moore, J.W. Davies, W.G. Collins, R.W. (1978) Chemistry International Edition, Mc Graw-Hill

Ono, M. Yokoi, T. Makino, T. (May 1987), Mathematical Model and Control System of Heating Furnace and Heat Treatment Furnace, The Sumitomo Search No.34, pp. 70 - 78

Perry, R. Chirton, C. (1983) Chemical Engineers Handbook 2nd Edition, Mc Graw-Hill

Zongyu Li, P.V. Barr, J.K. (1988) Computer Simulation of the Slab Reheating Furnace, University Pittsburgh, PA. 15213-3890 U.S.A.

BUCYRUS ERIE 1570W DRAGLINE - ANALYSIS OF THE UPPER MAIN SUSPENSION SYSTEM

Neil Brown

A project report submitted to the Faculty of Engineering, University of the Witwatersrand, Johannesburg, in partial fulfilment of the requirement for the degree of Master of Science in Engineering

JOHANNESBURG 1989

[redacted]