Modeling and Control of Geometric and Thermal Properties in Arc Welding

by
Radhouan A. Masmoudi

B.S. in Mechanical Engineering
California State University, Long Beach.
1985

M.S. in Mechanical Engineering
Massachusetts Institute of Technology
1987

Submitted to the Department of Mechanical Engineering
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy in Mechanical Engineering
at the
Massachusetts Institute of Technology
September 1992

Signature of Author_____
Department of Mechanical Engineering
September 1992

Certified by_____
Professor David E. Hardt
Thesis Supervisor

Accepted by_____
Professor Ain A. Sonin
Chairman, Graduate Office
Department of Mechanical Engineering

Modeling and Control of Geometric and Thermal Properties in Arc Welding

by

Radhouan A. Masmoudi

Submitted to the Department of Mechanical Engineering
in September 1992 in partial fulfillment of the requirements
for the degree of
Doctor of Philosophy in Mechanical Engineering

On-line quality control of arc welding requires the use of robust multivariable nonlinear control schemes. The design, simulation, and implementation of such schemes is greatly facilitated by the availability of accurate models that capture the inherently nonlinear dynamic properties of arc welding. Real multivariable control of current welding processes is not possible due to the high coupling between the process outputs. It is therefore necessary to modify the Arc welding processes in order to increase their controllability. One way to modify the process and increase its controllability is to weave the torch at high frequencies as it moves along the weld path.

In this thesis, the effects of high-frequency torch weaving are investigated both analytically and experimentally. A nonlinear dynamic model for heat transfer in arc welding was developed. The model includes inherent nonlinearities such as latent heat of fusion and temperature-dependent thermal coefficients, and can be used for control system design and simulation. Both simulations and experiments show that high-frequency weaving leads to significant decoupling of the outputs, in this case the pool geometry (width) and the thermal properties (as represented by the Heat-Affected Zone width).

Finally, a nonlinear controller, based on Time Delay Control (TDC), was designed and simulated for independent control of geometric and thermal properties of Gas Tungsten Arc Welding (GTAW). Existing hardware limitations prevented the implementation of TDC which requires a small sampling time. However, PID Control was implemented using a vision system for pool width measurement and a scanning Infra-Red pyrometer for temperature measurement. Simulation results show that TDC has the properties of fast response and robustness to model uncertainties and disturbances.

Thesis Supervisor: Dr. David E. Hardt

Title: Professor of Mechanical Engineering

Director, Lab. for Manufacturing and Productivity

Acknowledgments

This thesis is dedicated to my wife Leila, my daughter Mariem on her 3rd birthday, and my newborn baby, Youssef. With my infinite gratitude and appreciation for my parents who have taught me the virtues of hard work, patience, and the pursuit of excellence.

I would like to thank Prof. David Hardt for his support and encouragement throughout my studies at MIT. My work with him has been both rewarding and challenging. Prof. Thomas Eagar and Prof. Kamal Youcef-Toumi provided necessary guidance and helpful suggestions throughout my work on this thesis. My appreciation also goes to my colleagues at the MIT Lab. for Manufacturing and productivity; Pierre Jalkh, Dan Walczyk, Andrew Parris, and Jae-Bok Song. They all made my studies at MIT more fun and rewarding. Also thanks to Eric Brown, a UROP student, for helping me get the final closed loop control experiments done on time and to Fred Cote for his assistance in making the experimental hardware used for this thesis.

Finally, thanks to my Lord, the God of knowledge and creativity who provided me with energy and sustenance throughout my studies and my life.

This project was supported by the U.S. Department of Energy under contract number DE-FG02-85ER-13331.

Table of Contents

List of Figures

Chapter 1

Chapter 2

Chapter 3

Chapter 4

Chapter 5

Chapter 1

Introduction

Despite a long history of research on the application of control methods to arc welding, the development of a fully autonomous welding process, that can consistently maintain high quality welding, has not been achieved. This failure is linked to inadequate sensors and to basic process limitations that control alone cannot overcome. The high coupling between the arc welding process outputs severely limits the range of operations and makes parameter disturbance rejection almost impossible. For the case of manual welding, this coupling may in fact be advantageous, since it reduces the demands on the operator. However, for truly improved automatic process control, it is necessary to change the basic design of the process in order to increase its controllability and hence its robustness to disturbances.

This thesis contains analytical and experimental research on modeling and control of the Arc welding process aimed at improving the controllability of the process. The thesis consists of three main parts: The first part presents a nonlinear heat conduction model for dynamic simulation and analysis of the arc welding process. The model includes inherent nonlinearities such as latent heat of fusion and temperature-dependent thermal coefficients, and is shown to accurately predict the nonlinear dynamic response of arc welding. The second part of the thesis deals with the effects of high-frequency torch weaving on the temperature distribution in arc welding. The goal is to investigate, both analytically and experimentally, whether high-frequency weaving of the torch can decouple the process outputs, weld pool width and heat-affected zone width, and thus allow real multivariable control. The third section consists of the design and implementation of a nonlinear control scheme (called Time Delay Control, or TDC) for effective and robust multivariable control of the geometric and thermal properties in arc welding. A PID Controller is implemented using a vision system for pool width measurement and an Infra-Red sensor for temperature measurement.

1.1 Background:

Welding processes are the dominant method for joining parts in most industrial applications: e.g. automotive, ship building, pressure vessels, chemical plants, pipelines, structures, etc.. The 1980's have witnessed an exponential growth in the automatization and robotization of the welding process. Continued success of this trend requires deeper knowledge of the process dynamics and physics, as well as improved automatic control systems.

Arc welding processes are inherently nonlinear and complex, and involve several physical phenomena (heat transfer, fluid flow, metallurgical transformations, etc.). Yet, modern welding applications require automatic machinery which must be continuously regulated to insure consistent weld quality. This in-process regulation includes both geometric and thermal/metallurgical properties, both of which are three dimensional distributed-parameter phenomena.

Full automation of arc welding poses several challenges to the process control engineer. First, the welding process is extremely complex and difficult to model. Second, the process parameters are highly nonlinear functions of exogenous inputs so disturbance rejection is particularly important. Third, the geometry and the metallurgical properties of the weld are highly coupled so independent multi-input multi-output regulation is impossible.

There are two aspects to robotic welding. First, there is ***machine control*** which consists of controlling the process parameters (heat input, voltage, travel speed, and location of the arc). Seam tracking system (vision and non-vision based) is one example of machine control which has witnessed extensive research work. Even though, there is a continuous desire to improve on current standards, machine control is the easy side of robotic welding and much progress has been achieved in this area during the last decade. The second aspect of robotic welding is ***process control***. Process control requires on-line monitoring and control of the process. One or more process outputs are measured, or estimated, and feedback control is used to keep the desired outputs within desired specifications. Process control is more complicated because it requires a good understanding of the fundamental physics of the process. This understanding of the process dynamics must be captured in a model that can be used for design and analysis of the control system. In addition, process control requires accurate measurements of the states or the outputs of the process.

Manufacturing processes in general, and arc welding processes in particular, are subject to numerous process and parameter disturbances, necessitating the use of feedback to maintain a consistent quality. In most cases, this feedback is manual and applied in terms of off-line process robustness studies, or quality control methods. However, the accuracy and flexibility of many processes could be greatly enhanced by using real-time process control, that is concentrating on control of the actual part characteristics during production. The goal of process control includes the rejection or elimination of disturbances which typically arise from parameter variations, such uncertainty and variation in material properties. Uncertainties also arise from the inability to precisely model the detailed physics of complex manufacturing processes.

Among the outputs of welding process that are of interest to the control engineer are: the geometry of the pool and the temperature distribution during and after welding. Geometric properties include molten pool height, depth and width. Thermal properties are represented by the heat-affected zone width and the cooling rates at various locations. The cooling rate is extremely important because it effects the final microstructure of the weldment. Researchers at the MIT Laboratory for Manufacturing and Productivity (LMP) have worked independently on these two aspects (Hale 1990, Doumanidis 1989), but it is our goal in this thesis to develop combined thermal/ geometry control.

1.2 Research Goals:

Much research has been conducted to model and control the geometric or the thermal properties of welding. However, it is known that geometric and thermal properties are coupled, which precludes effective use of multivariable control methods. Therefore, it is necessary to look at combined thermal/geometric modeling and control.

In order to achieve multivariable control of geometric and thermal properties in arc welding, it is necessary to decouple geometric and thermal properties through some process modifications. In this thesis, this is achieved by using high-frequency torch weaving, which produces a variable heat input distribution, and it is shown that this greatly increases the process reachability [1], thereby making true multivariable control possible.

[1] In control theory, the term reachability is used to refer to the entire range on the output space that can be reached with the available input range.

The effects of high-frequency torch weaving on the temperature distribution and the process outputs are investigated both analytically and experimentally. A nonlinear heat conduction model is used to simulate the welding process and to perform dynamic open-loop and closed-loop simulations. This model was developed to study the effects of high-frequency weaving and to aid in the design and analysis of control systems.

For inherently nonlinear systems, control techniques have been successfully developed and implemented for different applications. Nonlinear control methods have found increasing applications in the last decade with the advent of inexpensive and programmable digital processors as well as new and improved nonlinear control methods. In this thesis, a nonlinear control method, called Time Delay Control, is shown through simulations to provide a simple and effective method of controlling arc welding. Simulation results show that TDC controller is robust to model uncertainties and disturbances. Closed-loop PID control was implemented using a vision system for pool width measurement and Infra-Red temperature sensor for HAZ width measurement.

1.3 Modeling of Arc Welding:

A dynamic model of the process is useful for dynamic simulations and control system design. Modeling of the arc welding process is a difficult undertaking because of the complex and nonlinear nature of the process. There are different types of numerical, analytical, and empirical models that have been used to model various aspects of the arc welding process. In this thesis, an analytical model is validated with experimental data to improve the accuracy of the model. The use of empirical data makes the model less general but greatly increases its accuracy.

Arc welding is a very complex process. During welding, heat flow occurs by conduction, convection, and radiation. Fluid flow in the pool is highly turbulent and unpredictable. The material properties change with temperature, and at elevated temperature (more than 2000 °K) cannot be accurately predicted. Latent heat of fusion also plays an important role at the solid/liquid interface. Chemical reactions sometimes take place between filler and parent material, and sometimes between the melted metal and the atmosphere (an inert gas is usually used to shield the weldment from the atmosphere). Phase transformations occur at various locations and temperatures, and

usually result in a complex and mixed microstructure. Under certain conditions some porosity is created inside the weld joint which causes embrittlement. This porosity may be caused by steam formation or by the inclusion of hydrogen molecules inside the molten pool. A high cooling rate results in cracking due to the high residual stresses and the presence of brittle martensite structure. Pre-heating, post-weld heat treatments, and multiple passes are sometimes used to improve the properties of the weld.

A complete analytical model that includes all the physical phenomena in the welding process is difficult to obtain and to solve. In fact, the physics of many of these phenomena are still not clearly understood and are the subject of ongoing research. However, a good simulation model is very useful for understanding the process dynamics and for control system design and simulation.

There are basically three different types of models of the welding process; analytical, numerical, and empirical models. Rosenthal (1947) was among the first researchers to develop a simple analytical model based on conduction heat transfer for predicting the shape of the weld pool for two and three-dimensional welds. However, Rosenthal's model does not accurately describe weld pool heat transfer , and does not account for non-linear (temperature dependent) material properties at elevated temperatures.

To overcome these limitations, numerical heat transfer models have been developed. These models utilize either finite difference or finite element techniques to represent more complex local and transient behavior and workpiece geometry. This increases the model flexibility and versatility, but also the computation time and may obscure the causal relationships between the inputs and the outputs. A finite difference approach for numerically integrating the unsteady conduction equation was developed by Doumanidis (1988). This model includes temperature-dependent material properties and multiple distributed heat sources. The model deals with the convection in the molten pool by designating an "enhanced conduction region" and determining the conduction coefficients empirically. The major shortcoming of this model is that metal transfer and fluid flow effects are not included. Numerical models are useful in obtaining a more accurate representation of the process, but cannot be used for control design purposes. To develop a control strategy, Doumanidis and Hardt (1989), used the numerical model to derive local transfer functions.

The third modeling approach used by Hale (1989) is to empirically develop transfer functions using a series of step response tests. It was found that a second order model

with constant damping ratio and velocity-dependent natural frequency is appropriate. Based on prior work (Hardt et al (1986), Doumanidis and Hardt (1989) and Suzuki and Hardt (1990)) it was found that the structure of such transfer functions remains constant, but the parameters are highly variable.

In chapter 2, we present a model based on conduction heat transfer, using numerical integration of a transient analytical solution of conduction heat transfer with a moving gaussian-distribution heat source in a semi-infinite plate. The model is versatile and includes several nonlinearities such as latent heat of fusion and temperature-dependent coefficients. This model was used to simulate heat transfer during welding and to investigate the effects of high-frequency torch weaving on the pool width and the HAZ width. These theoretical predictions were verified with experiments that implement high-frequency weaving as an additional control input in GTA welding.

The model's implementation as a Matlab function make it very useful for process modeling and simulation of arc welding. Simulation results show good correlation with experimental data. The main advantage of this model are its simplicity, accuracy, and ease of implementation in comparison with large and complicated finite-element or finite difference models. The model calculates the instantaneous temperature at a fixed point in a moving frame by creating a simple line mesh consisting of 20 elements along the direction of motion of the torch. It then solves a 3-D heat conduction equation at each of those points by using the temperature of the preceding point as initial conditions. This is an efficient method of calculating the transient temperature at a point without having to solve a large finite-element model. Using this small mesh, allows us to calculate the transient temperatures at the point of interest for each time step. Otherwise, if temperature at the same point and previous time step is used, temperature will keep increasing even when the heat input is decreased.

The model has been shown to accurately predict the transient behavior of the temperature distribution in arc welding accurately, and therefore is extremely useful in performing dynamic simulations of closed-loop control methods. The alternative method consists of using empirically-developed transfer functions which approximate the observed dynamic response of the process. However, the dynamic response of arc welding processes is highly nonlinear and depends on the operating conditions and parameters. Linear transfer functions are valid only in a small neighborhood of the operating point, and for changing conditions or inputs, the transfer functions must be changed accordingly. This makes the process of modeling very difficult and time

consuming. In contrast, the dynamic model developed in this thesis is very simple, accurate, captures most of the important dynamics of the process, and is valid throughout the entire operating range.

The dynamic model can also be used to select the appropriate inputs in a two-input-two-output (TITO) control system. The two outputs of interest are the molten pool width and the heat-affected zone (HAZ) width. Both of these outputs capture the geometric as well as thermal properties of the weld, and therefore give a good indication of the overall quality and strength of the weld joint. The Heat-affected zone (HAZ) is important because it contains different microstructures and is characterized by high temperature gradients. The width of the HAZ zone is a function of the temperature gradients inside HAZ, which determines the extent of residual stresses due to heat treatments of the material close to the molten pool region. Most of the fractures in weld joints occur inside the HAZ region. It is therefore essential that closed-loop control system provides tight regulation of the HAZ width. Geometric properties such as pool width, height, and depth do not capture enough information about the metallurgical transformations and the mechanical strength of the weldment.

Chapter 4 describes the design and implementation of a multivariable control system for arc welding. The goal is to achieve real-time independent control of weld pool width and heat-affected zone (HAZ) width. Two control systems are developed; one based on the linear PID method and the other on a new nonlinear control method called Time Delay Control (TDC). It is shown through simulations that the TDC method offers much better transient response as well as robustness to model errors and disturbances.

A straightforward and popular control method for many types of processes is the linear Proportional-Integral-Derivative (PID) method. Proportional control is very simple and provides a control action that is proportional to the error between the desired output and the actual output. PID control has been shown to work fairly well in many applications and is widely used in the industry due to its simplicity and ease of use. However, PID control requires an accurate model of the process in order to design the control system. When the process parameters change over time, it is necessary to tune the PID controller accordingly. Fine tuning of PID controllers is therefore time consuming and often needed to guarantee that the performance of the controller does not deteriorate over time. Another limitations of PID controllers is that it is a linear technique. When the process is inherently nonlinear, as is the case with

welding and most manufacturing processes, the model must be linearized around some operating conditions. In order to utilize PID control with a nonlinear process over a wide range of operation, 'gain scheduling' can be used to divide the operating range into different sections were the process can be linearized and different gains are used accordingly. Gain scheduling is time consuming and does not provide any stability guarantee since the appropriate schedule may not be a stationary function.

1.4 Welding Process Outputs:

The purpose of this research work is to develop and implement independent control of geometric and thermal properties in arc welding. For simplicity, we select one output that represents the geometry of the weld pool, namely the weld pool width, and one output that represents the thermal distribution in the weldment, namely the heat-affected zone (HAZ) width (see Fig. 1.1). Both of these outputs are important to the overall strength and quality of the final weld joint.

There are several types of Arc Welding processes. The most widely used are:

1- Gas Metal Arc Welding (GMAW a.k.a MIG - Metal Inert Gas):

GMAW is the most widely used fusion welding process (70% of all applications). It consists of adding heat using a consumable electrode. The filler material (electrode) is fed at high feedrate and is melted inside the arc and the fusion zone. Once it solidifies, the filler material becomes part of the pool area, so it is important that the filler material be selected properly to bond well with the base material. GMAW is easily automated because it is robust to outside disturbances such as variations in torch-to-workpiece distance. The process compensates for imperfections with metal addition. The arc is started by feeding the electrode and touching the workpiece ('Soft' arc starting). On the other side, GMAW causes too much spatter which maybe a problem in certain environments.

2- Gas Tungsten Arc Welding (GTAW a.k.a. TIG - Tungsten Inert Gas):

GTAW is a high-integrity process. It is mainly used for 'exotic' materials (e.g. Titanium, nickel-based alloys, etc..). It is very sensitive to disturbances such as torch-

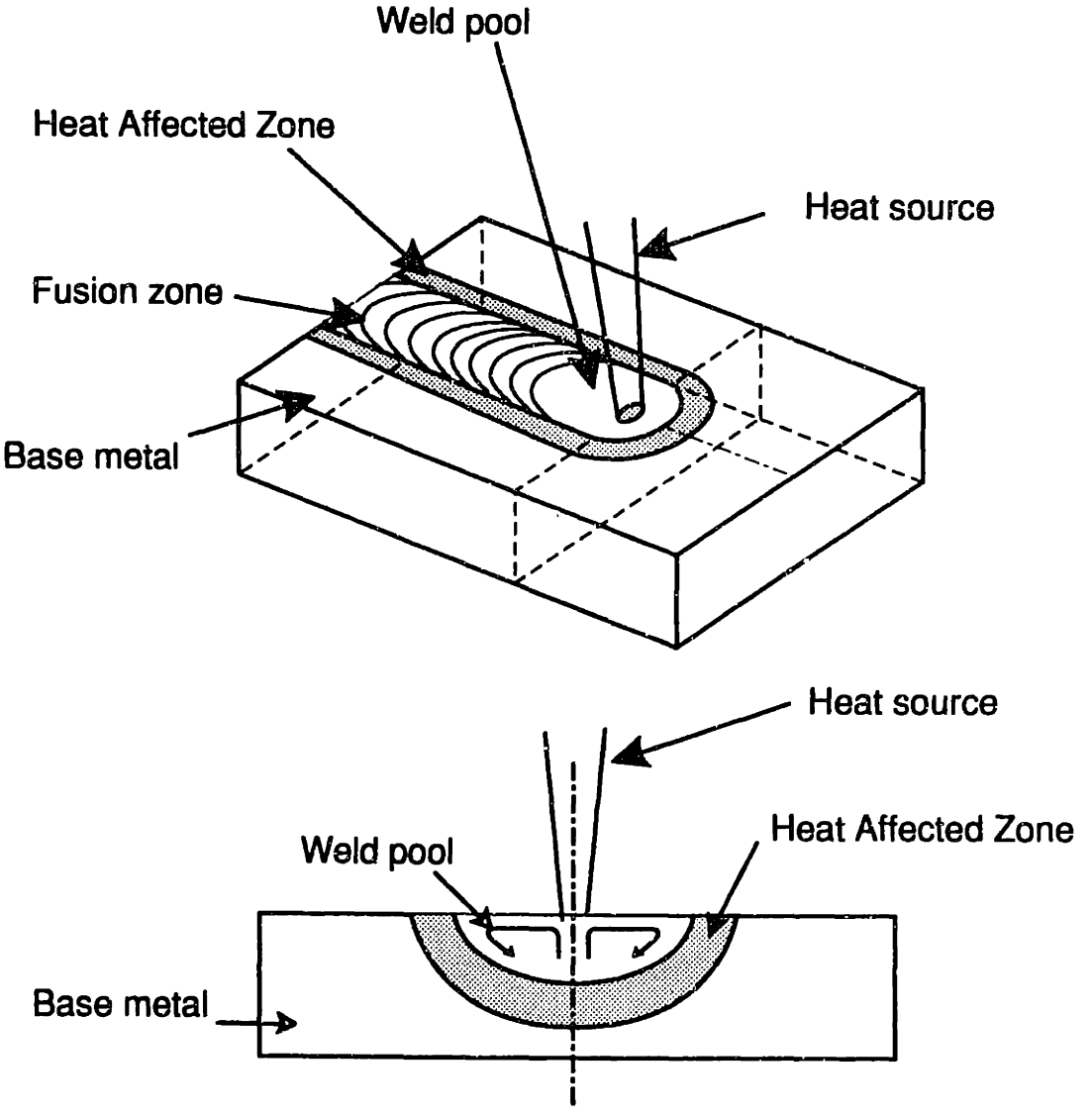


Fig. 1.1: Arc Welding Process Outputs

to-work distance, and gap between components to be welded. The arc tends to move toward the shortest distance. GTAW is a slow process due to the low heat input used.

3- Plasma Welding:

Plasma welding is a high speed process with more concentrated heat input (useful for robotic applications). Plasma welding is less sensitive to joint fit-up than GTAW, and is also used for cutting materials. It is more flexible in terms of number of inputs to control. Plasma welding results in high-quality weldments and has a high repeatability.

4- Heat Affected Zone (HAZ)

The Heat-Affected zone (HAZ) is an unmixed zone where the base metal composition is subjected to the weld cooling cycle. The HAZ is crucial to the strength of the weldment since most cracking and fractures occur inside the HAZ region. HAZ is defined as the region between the melting point temperature (T_m) and the AC1 temperature (T_{c1}). It is usually a small region characterized by very high temperature gradients (more than 300 °K per mm. for mild steel). As a result, HAZ contains a variety of micro-structures some of which have weak toughness and strength properties.

As shown in Fig. 1.2, the metallurgical composition of the heat-affected zone (HAZ) region for mild steel is very diverse and contains some martensite. For mild steel, the width of the HAZ is of the order of 2-4 mm. It is therefore essential to maintain tight control over the thermal properties of this critical region.

The Heat-Affected Zone is a site for potential defects such as grain growth (austenitic) close to the pool. Aluminum is sometimes added to steel to inhibit grain growth. Hydrogen embrittlement is also a concern in the HAZ region because it causes increased hardness and reduced toughness due to high heat input. Residual stresses cause stress corrosion cracking.

1.5 Process modification for Enhanced Controllability

A major obstacle to the successful implementation of automatic control schemes in the welding process has been the strong coupling of the outputs, particularly the

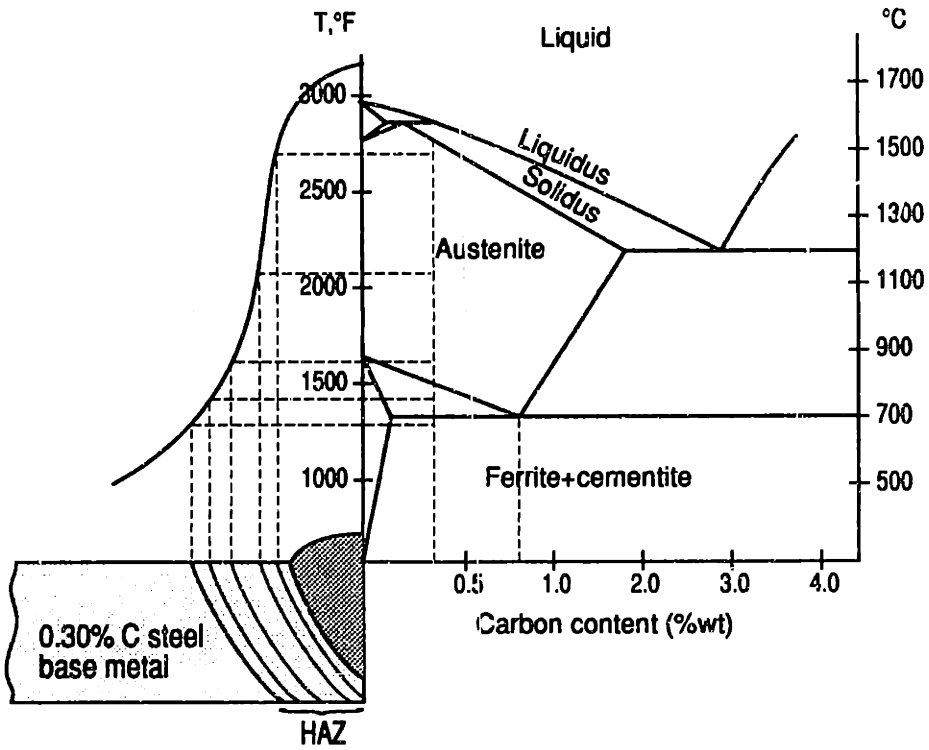


Fig. 1.2: Metallurgical composition of the heat-affected zone (HAZ) (Source: American Welding Society)

geometry of the pool and the thermal properties of the weld. In most welding applications, it is desirable to independently control the pool width and the heat affected zone width in order to obtain the desired combination of thermal and geometric properties. Yet these properties are inherently coupled because they are both determined by the shape of the temperature isotherms.

Previous experiments (Hale and Hardt) using feedback control show that for a given width, the HAZ can be modulated at most about 0.5 mm and in many cases much less. Hale (1990) showed that there is a high coupling between pool width and HAZ width (see Fig. 1.3) which precludes real multivariable control. Furthermore, he showed that this coupling makes the control system not robust to disturbances by shifting the reachability range when using heat input and travel speed as control inputs. This is because, even though these two inputs have an important effect on the process, they are both related by the heat-input-per-unit-length (Q/v). Thus, it is clear that real multiva-

riable control will require process re-design to enlarge the range of reachability and thus make the outputs more decoupled.

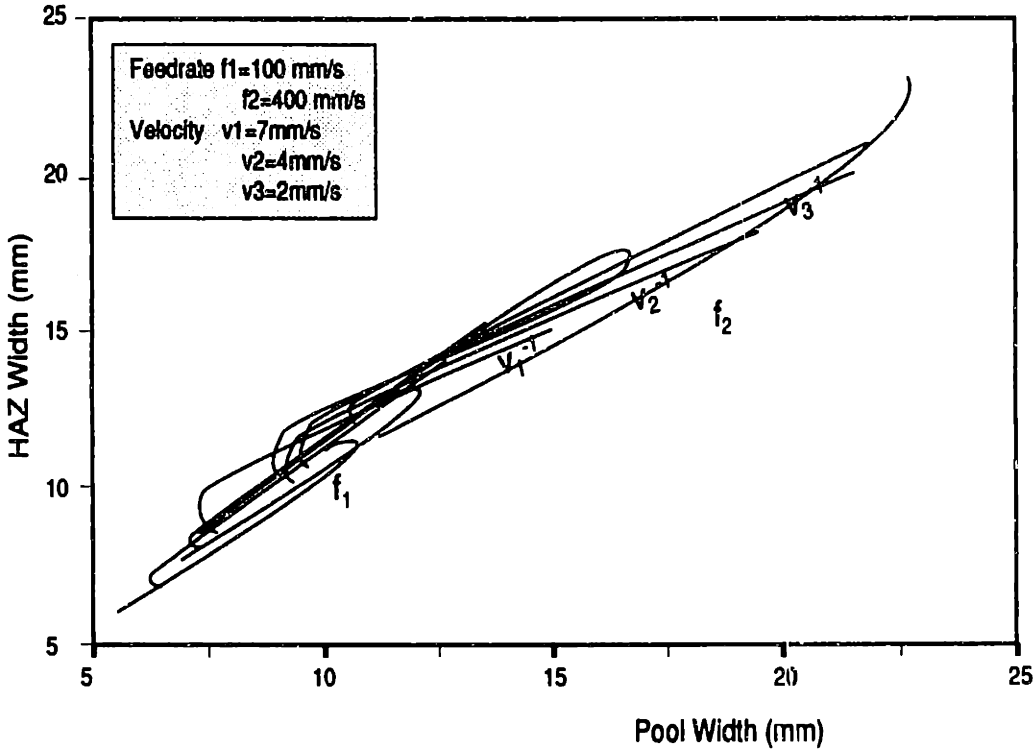


Fig. 1.3: Output map using heat input and travel speed as control inputs (From Hale (1990))

To achieve decoupling between geometric and metallurgical phenomena, the heat input distribution must be modified in order to exert more control over the amount and location of heat input. The most intuitive method to achieve this is by applying the heat in a mechanically controlled fashion by moving the torch transverse to the weld via high frequency weaving, and controlling the HAZ width by modulating current (i.e., heat input) or torch travel speed.

In this thesis, high-frequency weaving is investigated as a method to enhance the controllability of the welding process. A new control input, weave amplitude, may be used to decouple the geometric and thermal properties.

The dynamic heat transfer model, presented in chapter 2, was used to simulate the arc welding process, and to investigate the effects of High Frequency weaving on the 3-D temperature distribution in the workpiece. The effects of high-frequency weaving were also investigated experimentally by using a servo-motor driven weaving mechanism capable of high frequency weaving (up to 10 Hz.) and 0.05 mm. accuracy. More than 300 open-loop experiments, using Gas Tungsten Arc Welding (GTAW), were conducted to investigate the effects of High Frequency weaving on the process outputs, namely the molten pool width and the Heat-Affected zone (HAZ) width. Both simulations and experiments show that High Frequency weaving decouples the outputs and increases the reachability range.

Chapter 2

Welding Process Model for Thermal/Geometry Control

The heat supplied by a welding arc produces complex thermal cycles in the metal which cause changes in the microstructure of the heat-affected zone and result in residual stress and distortions in the final weld joint. In order to fully understand these effects, we must first analyze the heat flow in the weldment.

In this chapter, an analytical model of heat flow in welding is presented, and validated with experimental data. A simple heat conduction model using numerical integration is used to calculate the 3-D temperature distribution in a semi-infinite plate with a moving heat source. The model will be used in Chapter 3 to simulate the effects of high-frequency torch weaving on the welding process by looking at two process outputs; namely the molten pool width and the heat affected zone (HAZ) width.

The process model presented here is based on heat conduction, numerically integrating the Green's function solution to heat diffusion from a moving heat source to a semi-infinite plate. The model is versatile and can easily include nonlinearities such as temperature-dependent parameters and heat losses due to radiation and convection. Heat losses, as well as heat stored during melting, are modelled as polynomial functions that are empirically derived.

Information about the temperature distribution and history during welding allows the prediction of final weldment strength and properties. For instance, the weld pool width and the heat-affected zone (HAZ) width can be calculated from the peak

temperature reached at a given point. If this temperature is larger than the solidification temperature (A_{c1}) then that point is inside the heat-affected zone, and if it is greater than the melting point temperature (T_m) of the base metal, then it lies inside the molten pool region. Similarly, the temperature history determines the heat treatment of the metal and therefore its metallurgical properties. Residual stresses in the weld is a concern and can be investigated by looking at the cooling rates in the areas adjacent to the weld joint. These phenomena, and many other properties of the weld, can be studied or controlled by simply looking at the heat flow in the vicinity of the weld joint.

Rosenthal (1941) was among the first researchers to develop a simple analytical model based on conduction heat transfer for predicting the shape of the weld pool for two and three-dimensional welds. However, Rosenthal's model suffers from the inability to describe weld pool heat transfer accurately, and does not account for non-linear (temperature dependent) conduction at elevated temperatures. There has been many refinements of Rosenthal's model to improve its accuracy over the past forty years.

2.1 Fundamentals of Heat Conduction in Welding:

In general, the rate of heat transfer through conduction is given by the Fourier equation:

$$\frac{dQ}{dt} = \dot{Q} = -kA\frac{dT}{dx} \qquad (1)$$

where t is time in seconds, T is temperature in °K, A is the area perpendicular to heat flow in mm^2, x is the direction of heat flow in mm., and k is the thermal conductivity of the material. When the heat flux Q is in W/m^2 and the temperature gradient in °K/m, the thermal conductivity k has units W/m °K. Let us take a small element of the material in which heat is conducted (figure 2.1). The net heat input through the left hand surface can be written as (from eq. 1):

$$\dot{Q}_1 = -\left(k\frac{dT}{dx}\right)\Big|_x \Delta y \Delta z \qquad (2)$$

Similarly, the heat output through the right hand surface can be written as:

$$\dot{Q}_2 = -\left(k\frac{dT}{dx}\right)\Big|_{x+\Delta x}\Delta y\Delta z \qquad (3)$$

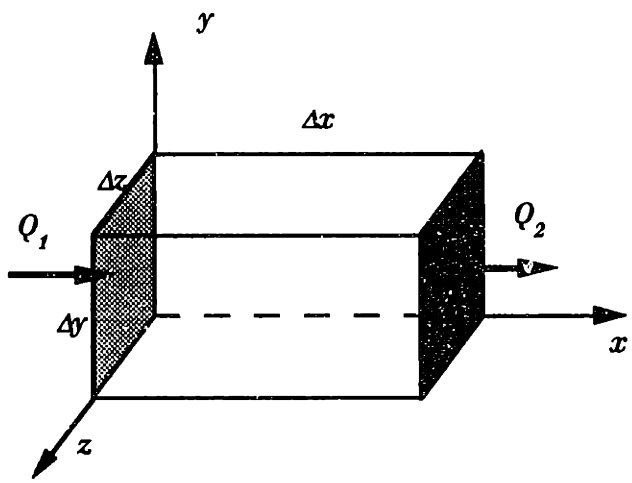


Fig. 2.1: Heat transfer in an infinitesimal element.

Therefore, the net heat transfer per unit volume in the x direction is:

$$\dot{Q}_1 - \dot{Q}_2 = \frac{\left(k\frac{dT}{dx}\right)\Big|_{x+\Delta x} - \left(k\frac{dT}{dx}\right)\Big|_{x}}{\Delta x}\Delta V \qquad (4)$$

If we take the limit of eq. 4 as the element becomes infinitesimally small ($\Delta V \rightarrow 0$), then eq. 4 becomes:

$$\dot{Q}_1 - \dot{Q}_2 = \frac{\partial}{\partial x}\left(k\frac{\partial T}{\partial x}\right)\Delta V \qquad (5)$$

If we take the difference between the heat input at each side of the element using the principle of conservation of energy, the change of internal energy per unit volume is:

$$\rho c\frac{\partial T}{\partial t} = \dot{Q}_G + \frac{\partial}{\partial x}\left(k\frac{\partial T}{\partial x}\right) + \frac{\partial}{\partial y}\left(k\frac{\partial T}{\partial y}\right) + \frac{\partial}{\partial z}\left(k\frac{\partial T}{\partial z}\right) \qquad (6)$$

where Q_G is the heat generated inside the element (by chemical reactions or by the arc current), ρ is the density of the material, and c is the specific heat of the material. If we assume that the thermal conductivity, k, is a function of temperature only, then eq. 6

becomes:

$$\rho c\frac{\partial T}{\partial t} = \dot{Q}_G + k\left(\frac{\partial^2 T}{\partial x^2}+\frac{\partial^2 T}{\partial y^2}+\frac{\partial^2 T}{\partial z^2}\right)+\frac{\partial k}{\partial T}\left[\left(\frac{\partial T}{\partial x}\right)^2+\left(\frac{\partial T}{\partial y}\right)^2+\left(\frac{\partial T}{\partial z}\right)^2\right] \quad (7)$$

If additionally the thermal conductivity k is assumed constant, this equation may be further simplified:

$$\frac{\partial T}{\partial t} = \frac{\dot{Q}_G}{\rho c}+\alpha\left(\frac{\partial^2 T}{\partial x^2}+\frac{\partial^2 T}{\partial y^2}+\frac{\partial^2 T}{\partial z^2}\right) \quad (8)$$

where $\alpha = \frac{k}{\rho c}$ is the thermal diffusivity of the material, k is the thermal conductivity of the material, Q_G is the heat generated inside the element, ρ is the density of the material, and c is the specific heat of the material. Eq. 8 is the differential equation of heat conduction for a stationary, homogenous, isotropic solid with constant thermal conductivity, k. The solution to this equation is strongly dependent on the boundary conditions and the initial conditions (at time $t = 0$ sec.) because eq. 8 gives the solution to heat flow *inside* the body which depends on the heat flow between the body and its environment. In the following derivations, we assume a semi-infinite plate which means that there are no boundary conditions to be accounted for.

2.2 Finite Element Model:

A first attempt to model the effects of high-frequency weaving was made using a finite element model using a modified version of the model developed by Doumanidis (1988). The model uses two meshes: a coarse mesh away from the torch and a fine mesh close to the torch to represent the melting region (see Fig. 2.2). The explicit expression for the temperature at any element of the mesh is calculated as a function of the temperatures at neighboring elements according to:

$$T^{t+\Delta t} = T^t + \Delta Fo\left(\sum_{i=1}^{6} T_i^t - 6T^t\right)$$

where $F_0 = \alpha\, t/\Delta s^2$ is the dimensionless Fourier number, Δs is the mesh size, and Δt is the time step.

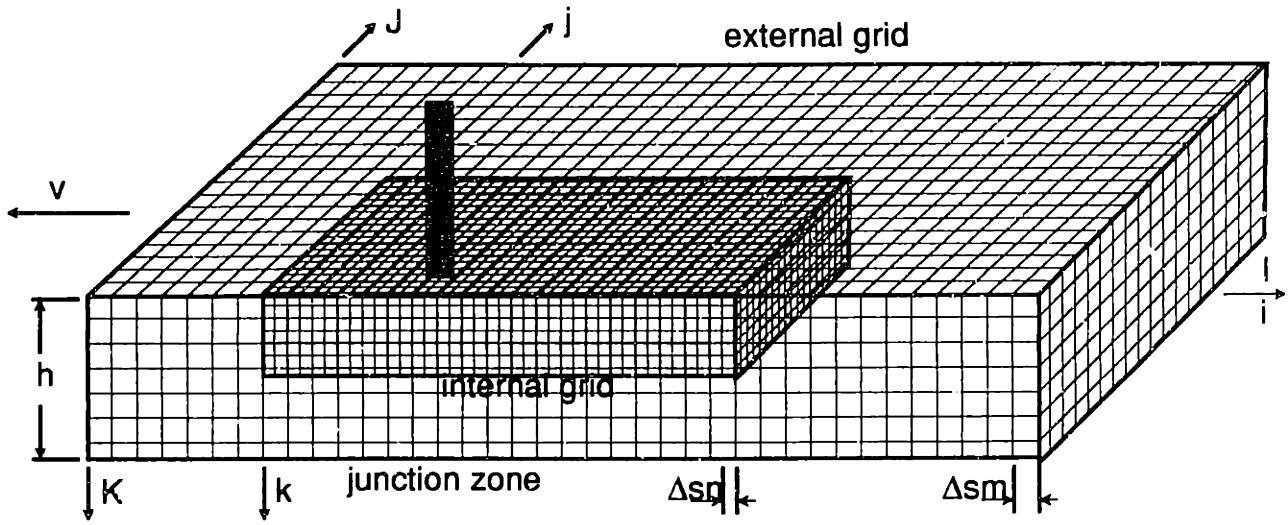


Fig. 2.2: Mesh elements used in the finite-element model

Figure 2.3 shows an example of the temperature profiles obtained with this finite element model. Finite element simulations were performed on a Cray II supercomputer, and the results are shown in figures 2.4-2.8.

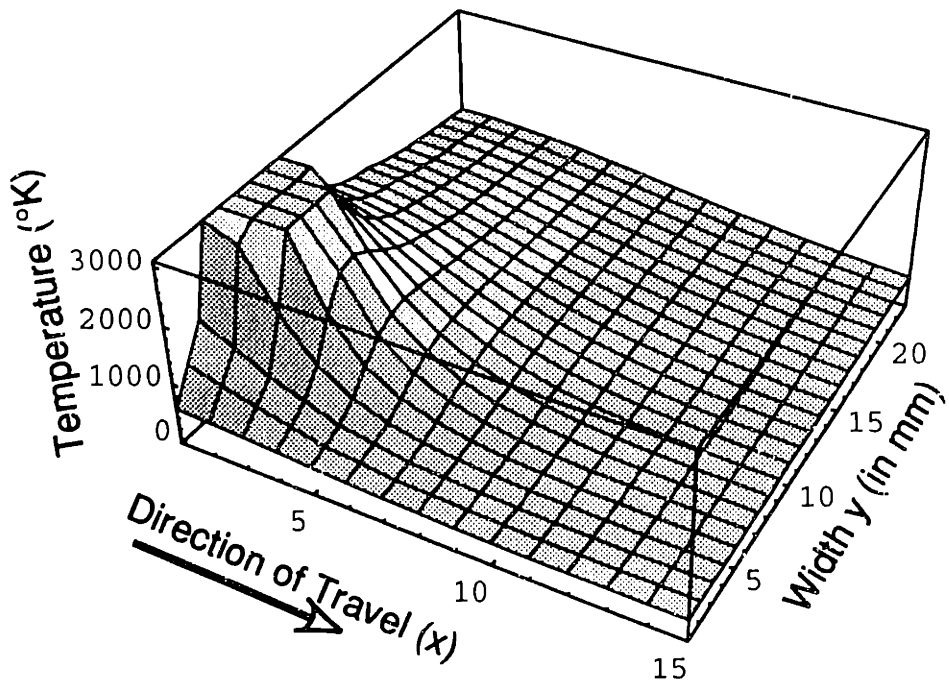


Fig. 2.3: Temperature profile on top surface obtained with the finite-element model.

The model was modified to allow for lateral torch motion to model the effects of high frequency weaving. However, in order to model high-frequency weaving (up to 10 Hz) as travel speeds of up to 10 mm/s, the mesh size must be extremely small (of the order of 0.01 mm). This leads to long computation time and reduced accuracy. Fig. 2.4 shows the simulated response to a step increase in the heat input. The response of the HAZ width is second order non-minimum phase. This means that the HAZ width first decreases and then increases as a result of a step increase in heat input. The reason for this non-minimum phase behavior is that HAZ is the distance between two isotherms, both of which have a first order response to heat input but with different time constant. As the heat input is increased, the inner isotherm increases more quickly than the outer isotherm.

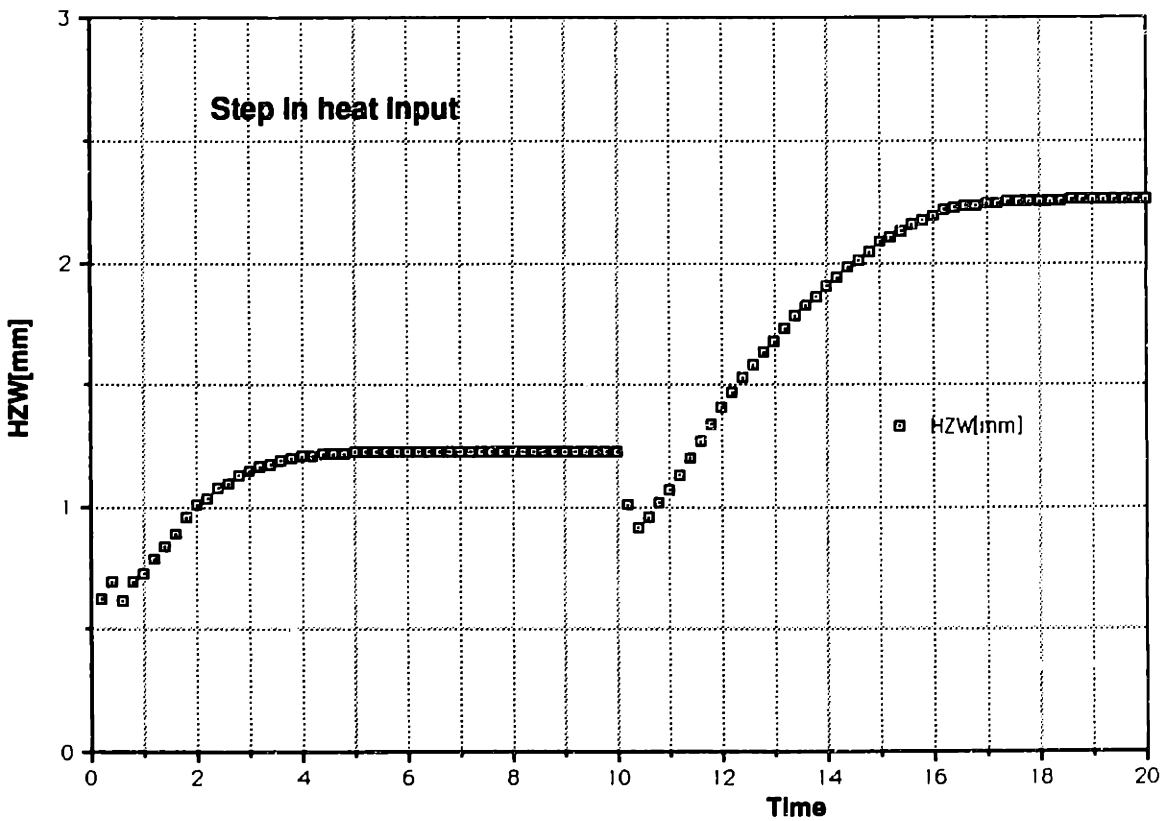


Fig. 2.4: Effect of a step change in heat input on heat-affected zone width (heat input changes from 1000 J/s to 2000 J/s)

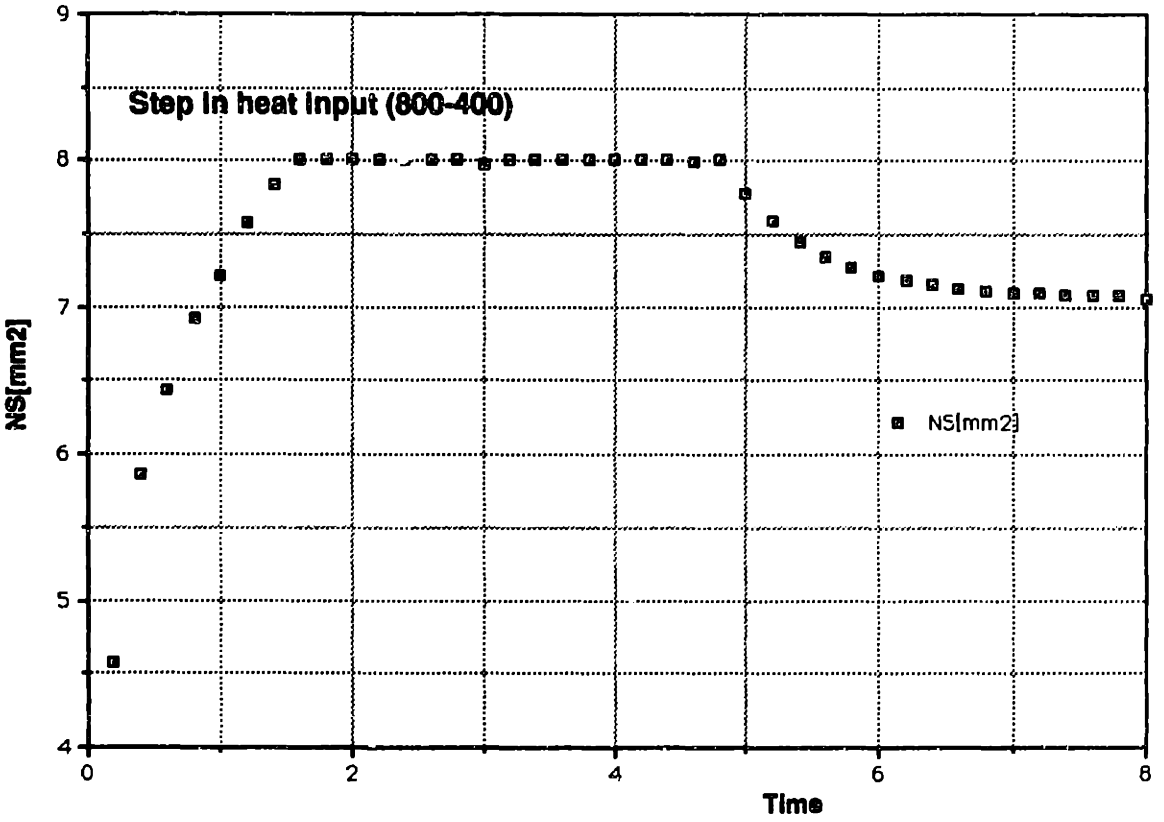


Fig. 2.5: Effect of a step change in heat input on nugget size (heat input changes from 800 J/s to 400 J/s)

Figure 2.5 shows the response of the nugget size (NS) as a result of a step decrease in heat input. Nugget size is the cross section area of the molten pool (approximated by the product of pool width and pool depth). The response of the NS can be approximated by a first order transfer function with a time constant of about 1 sec.

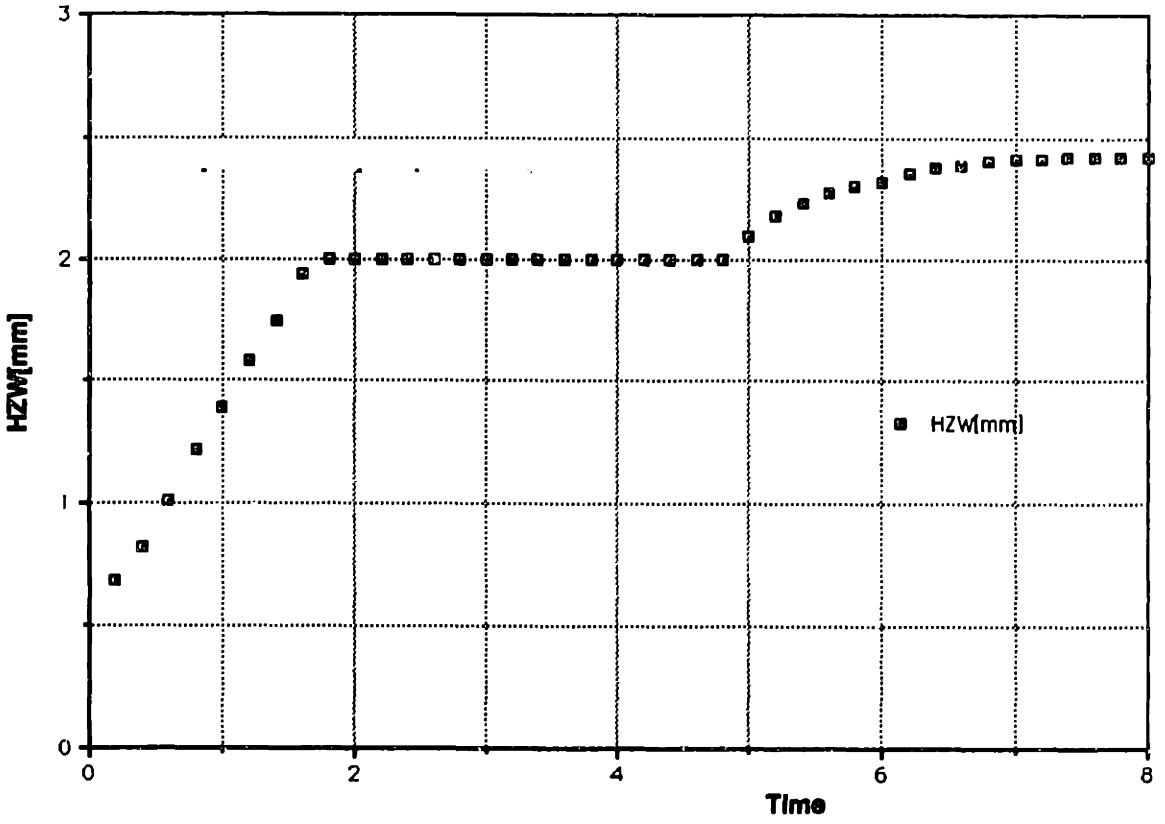


Fig. 2.6: Effect of a step increase in weave amplitude on HAZ width (weave amplitude from 2 mm. to 4 mm.)

Figure 2.6 shows the response of HAZ width to a step increase in weave amplitude. The response of the HAZ width can be modeled as a first order transfer function with a time constant of about 1.2 second.

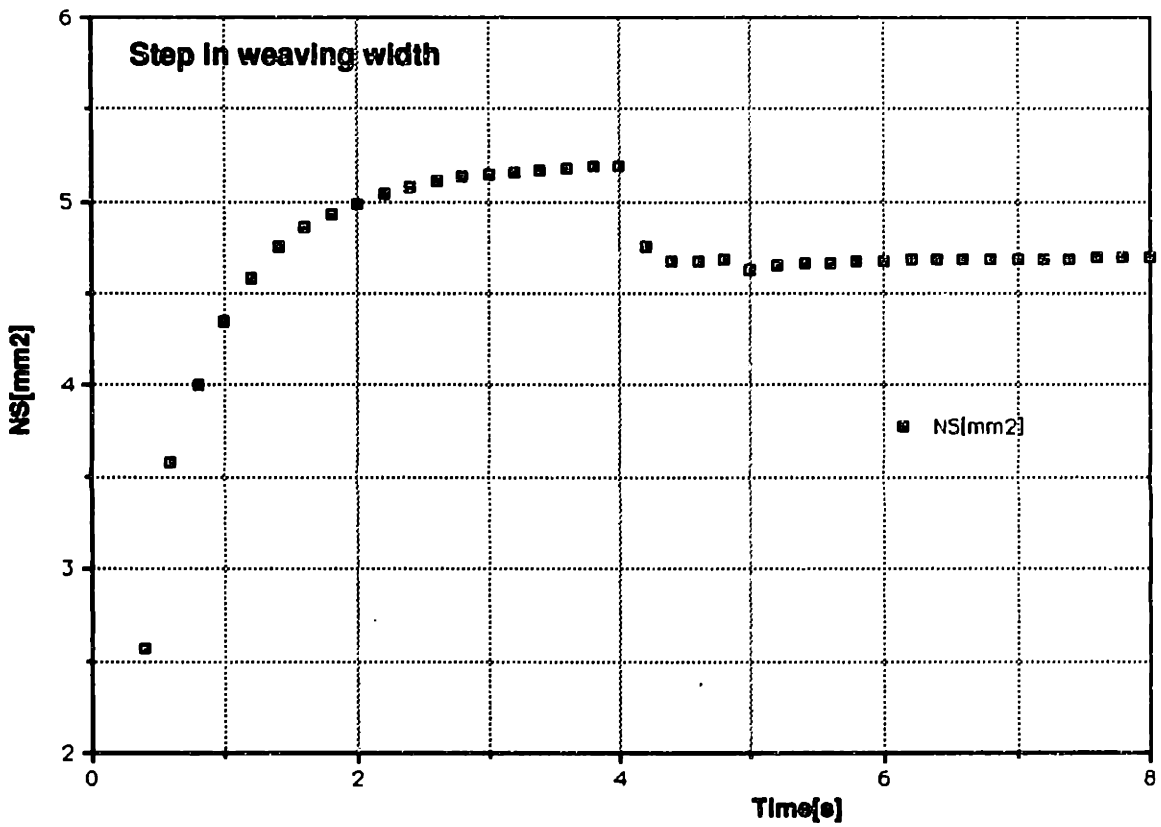


Fig. 2.7: Effect of a step change in weave width on nugget size (amplitude changes from 2 mm to 4 mm)

As shown in Fig. 2.7, a step increase in weave amplitude leads to a decrease in nugget size. A larger amplitude leads to an increased pool width and a smaller pool depth. As a result, the nugget size is decreased.

Fig. 2.8 shows the effect of an increased weave frequency on HAZ width. A larger weave frequency results initially in a larger HAZ width but then in a smaller HAZ width. The finite element model used to perform these simulations cannot accurately represent the effect of large weave frequency because of the finite mesh size. Simulations performed with very small mesh sizes (less than 0.25 mm), showed a large deterioration in the accuracy of the model.

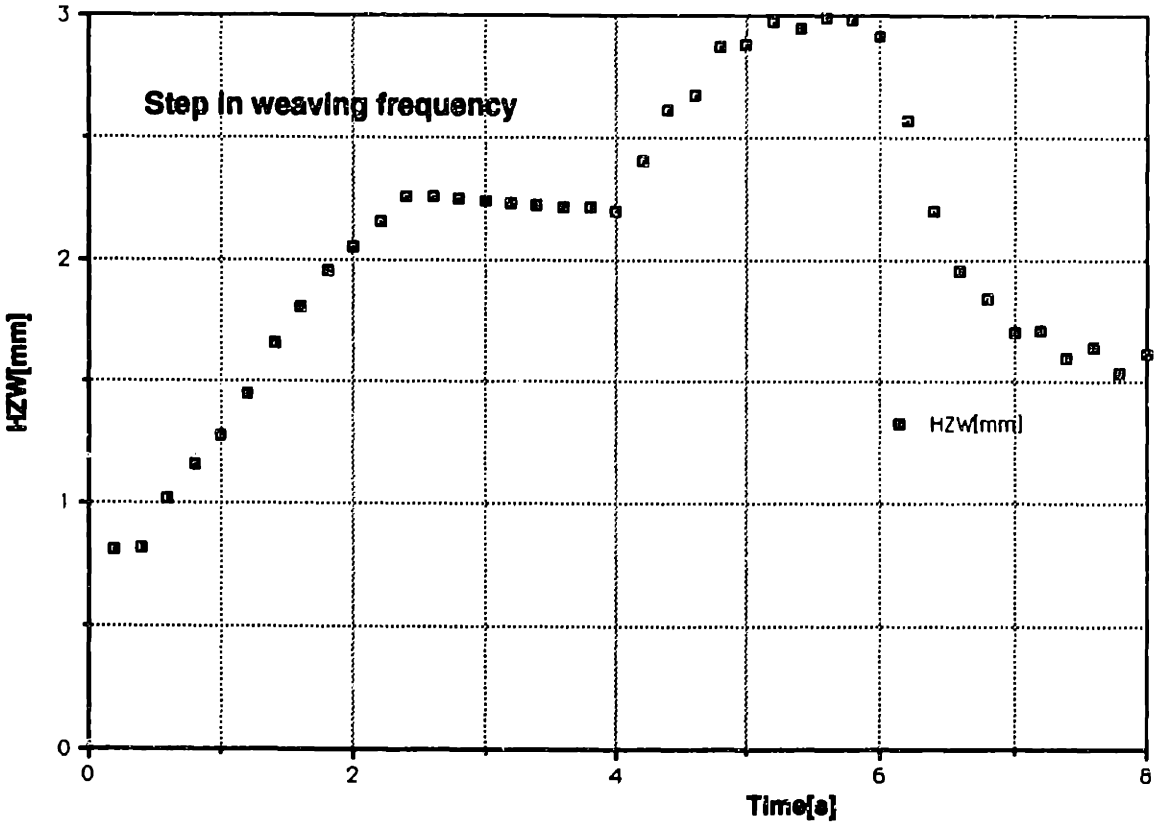


Fig. 2.8: Effect of a step change in weave frequency on heat-affected zone width (weave frequency changes from 3 Hz to 7 Hz)

In order to model high-frequency weaving (up to 10 Hz.), with travel speeds as large as 10 mm/s, the mesh size must be of the order of 0.01 mm. This leads to long computation time and reduced accuracy. Therefore, this finite element model is not very practical, and we are interested in a computationally-simple model that can be used to simulate the welding process quickly and inexpensively.

2.3 Analytical solutions of the heat conduction equation:

The fundamental heat conduction equation (Eq. 8) has been solved first by Rosenthal in 1946 for a moving point heat source. Since then several refinements and extensions have been made over the years to model more practical applications. In 1956, Grosh studied the effect of using temperature-dependent coefficients on the point source solution, and showed that it results in 15% more accuracy. In 1965, Christensen compared the point source solution with experimental results for various welding methods and developed a non-dimensional solution which can be used for all types of welding applications. Christensen showed that the point source solution underestimates the size of the weld zone by as much as 60%. In 1974, Malmuth studied the effect of the latent heat on the weld penetration and width. He showed that the latent heat has a relatively small influence on the weld width and penetration. The distributed heat source solution was offered first in 1974 by Rykalin who showed that it gives more accurate predictions than a point source. Since then, the distributed heat source solution has received much more attention and several solutions were derived for varying applications.

Most analytical solutions are based on some simplifying assumptions in order to solve the heat conduction equation. In deriving Eq. 8, it was assumed that heat transfer occurs by conduction only, and convection and radiation were neglected. It was also assumed that all material properties (density, specific heat, and thermal diffusivity) are constant, and that no phase transformations take place inside the material. Rosenthal's steady state solution of the temperature distribution produced by a point heat source moving at constant velocity is given by:

$$T - T_0 = \frac{q}{4\pi kR} e^{-\frac{v(w+R)}{2\alpha}} \qquad (9)$$

where T_0 is the initial temperature of the material, R is the distance between the source and the point of interest, v is the constant forward velocity, and w is the relative distance $w = x - vt$.

Following the steady-state solution proposed by Carslaw (1959) (see derivation in Appendix 1), and accounting for the fact that the heat source is moving both longitudinally and transversely, the change in temperature is given by:

$$dT_{t'} = \frac{\delta Q\, dt'}{\rho c\left(4\pi\alpha[t-t']\right)^{\frac{3}{2}}}\; e^{-\frac{(x-x_s)^2+(y-y_s)^2+(z-z_s)^2}{4\alpha(t-t')}} \tag{10}$$

where δQ is the amount of heat added during time $dt' = t - t'$ and (x_s, y_s, z_s) represent the position coordinates of the heat source at time t. It can be easily verified that eq. 10 satisfies the differential equation of heat conduction (eq. 8). If the heat source is assumed to have a Gaussian distribution, then the heat added at every point on the surface is:

$$Q(x,y) = \frac{q}{2\pi\sigma^2}\; e^{-\frac{(x^2+y^2)}{2\sigma^2}} \tag{11}$$

Therefore, the total heat addition by the Gaussian heat source may be obtained by integrating eq. 10 over the entire upper surface (Tsai, 1983):

$$dT_{t'} = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} \frac{q}{2\pi\sigma^2}\,\frac{dt'}{\rho c\left(4\pi\alpha[t-t']\right)^{3/2}}\; e^{-\frac{(x^2+y^2)}{2\sigma^2}-\frac{(x-x_s)^2+(y-y_s)^2+(z-z_s)^2}{4\alpha(t-t')}}\, dx'\, dy'$$

$$= \frac{q\, dt'}{2\pi\rho c\left(4\pi\alpha[t-t']\right)^{1/2}\left[2\alpha(t-t')+\sigma^2\right]}\; e^{-\frac{(x-x_s)^2+(y-y_s)^2}{4\alpha(t-t')+2\sigma^2}-\frac{(z-z_s)^2}{4\alpha(t-t')}} \tag{12}$$

Equation 12 gives the incremental temperature change between time t' and time $t'+dt'$. In order to calculate the absolute temperature at time t, we have to integrate eq. 12 from 0 to t. However, since we are interested in the instantaneous temperature and not only in the final temperature, this integration is performed stepwise ($dt' = t - t'$):

$$T(t) = T(t') + \int_{t'}^{t} \frac{q}{2\pi\rho c\sqrt{4\pi\alpha[t-t']\left[2\alpha(t-t')+\sigma^2\right]}}\; e^{-\frac{(x-x_s)^2+(y-y_s)^2}{4\alpha(t-t')+2\sigma^2}-\frac{(z-z_s)^2}{4\alpha(t-t')}}\, dt' \tag{13}$$

Equation 13 gives the incremental temperature change between time t' and time $t'+dt'$. It is the solution for 3-D point source heat flow in an infinite medium. By symmetry, the temperature distribution in a semi-infinite medium can be obtained by

multiplying the total heat input by a factor of two. Also, since the heat source is assumed to be at the top surface (z_S=0), this solution becomes:

$$T(t) = T(t') + \int_{t'}^{t} \frac{q}{\pi\rho c\sqrt{4\pi\alpha[t-t'][2\alpha(t-t')+\sigma^2]}}\; e^{-\frac{(x-x_s)^2+(y-y_s)^2}{4\alpha(t-t')+2\sigma^2} - \frac{z^2}{4\alpha(t-t')}}\, dt' \qquad (14)$$

Equation (14) may be used to calculate the temperature history at any point (x,y,z) starting from any initial condition $T(0)$. Numerical integration of Eq. (14) was implemented in a C-language code on a Macintosh II personal computer using a Fifth-order Runge-Kutta smart routine with monitoring of local truncation error to ensure accuracy and adjust the stepsize accordingly (See appendix D for program listing).

It is useful here to re-state the assumptions used in deriving the above model:

- The material is solid at all times and no phase changes occur,
- The material is homogeneous, isotropic, and its thermal properties are constant,
- The workpiece is infinite and there are no heat losses at the boundaries,
- Heat transfer through convection or radiation has been neglected,
- The moving heat source is modeled as a Gaussian distribution.

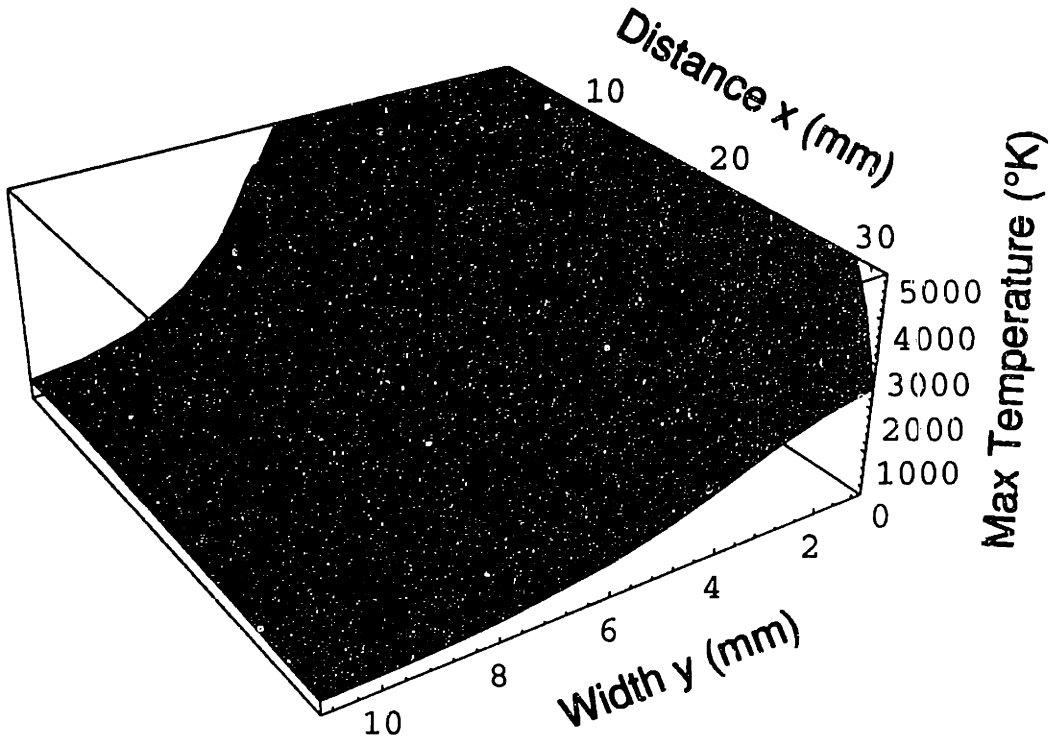


Fig. 2.9: Temperature Distribution using the Conduction Heat Flow Model (with step in heat input at x =10 mm)

Note that equation (14) is not defined at $t = t_s$, and therefore we have to integrate from $t_s + \varepsilon$, where ε is a small number (0.01 sec). This numerical truncation results in a smaller temperature when equation (14) is integrated repeatedly from time t=0 to time $t=t_{final}$. As a result, the pool width and the HAZ width calculated will be smaller than expected.

Fig. 2.9 shows an example of the heat distribution obtained on the top surface when the heat source moves from x=0 to x=30 mm, and the rate of heat input changes at x=10 mm from 3000 J/s to 1500 J/s.

2.4 Model Validation:

Figure 2.10 shows the result of simulating the model for the case of mild steel with a heat input Q between 500 J/s and 2500 J/s and a travel speed v between 1 mm/s and 5 mm/s with no weaving. Fig. 2.10 shows that as the pool width increases due to an increase in the heat input per unit length, the HAZ width decreases regardless of the value of Q and v. This is because the HAZ width is the difference between two isotherms, and as the heat input per unit length is increased, the inner isotherm expands more than the outer isotherm therefore the difference becomes smaller but always in a coupled fashion. This is explained from a pure conduction model which neglects latent heat of fusion and assumes constant material properties. This result however does not match the experimental data, which indicate that as the heat input per unit length is increased, the HAZ also increases (see section 3.5). Changing the welding inputs or the material properties does not change this intrinsic property in the conduction model. Fig. 2.11 shows the effect varying heat input (Q), thermal diffusivity (α), specific heat (c), and the distribution of the Gaussian source (σ). It is seen that changing these parameters influences the absolute value of HAZ, but preserves the same decreasing trend.

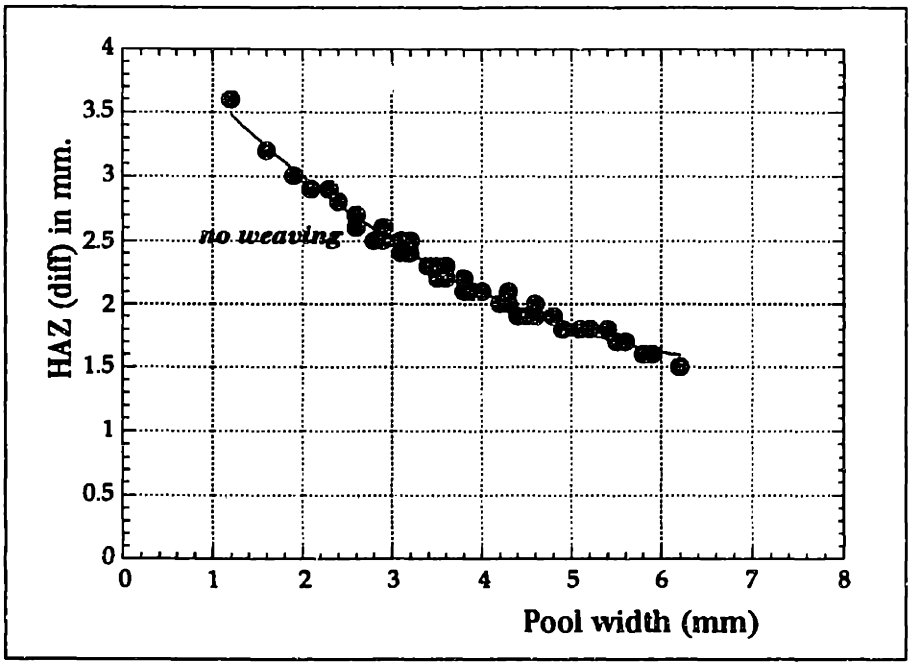


Fig. 2.10: Reachability range without weaving (pure conduction model)

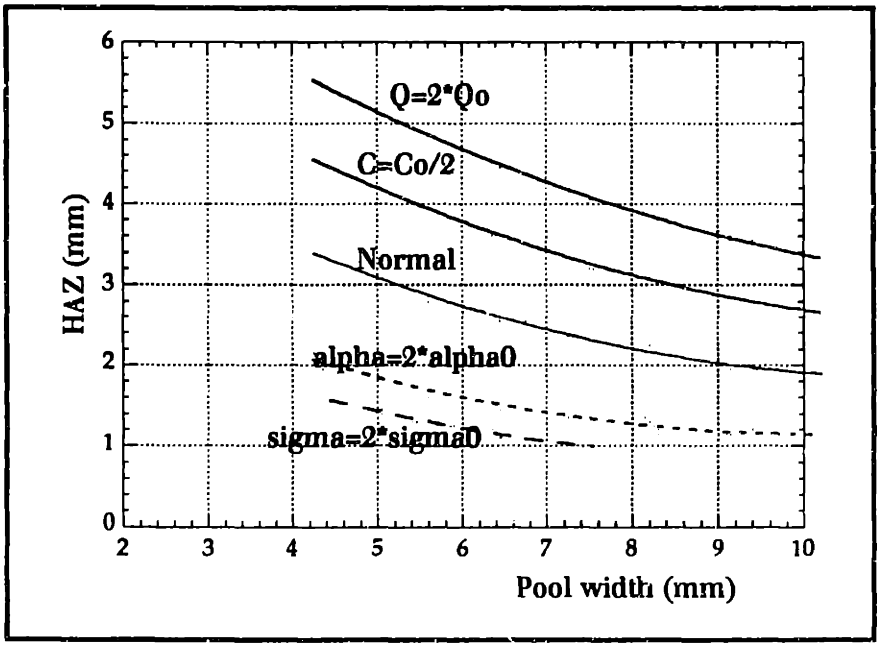


Fig. 2.11: Effect of process parameters on Reachability range

In order to change this trend to match the experimental data, the latent heat of fusion and temperature-dependent heat coefficients must be included in the model. In the molten pool region, the dominant heat loss is due to the arc efficiency, and additional heat is stored due to latent heat of fusion which is temporarily absorbed during melting. Thus, the heat stored in the molten pool region by latent heat of fusion is a function of the pool volume. In the simulations, this was modeled as a cubic function of the width y (the constants a and d are obtained by fitting experimental data):

$$q = q_0\left(\eta_{arc} - [a * y + b]^3\right)$$

where q_0 is the electrical energy supplied by the power supply, y is the pool width in mm, and η_{arc} represents the arc efficiency.

In the HAZ region, the latent heat is released when the liquid solidifies, and the dominant heat losses in the HAZ region are due to heat convection and radiation at the top surface. Therefore, heat losses in the HAZ were modeled as a function of the top surface area (square function of width):

$$q = q_0\left(\eta_m - [c * y + d]^2\right)$$

where η_m is the melting efficiency, and c and d are obtained by empirical calibration. By matching these curves to the experimental data (presented in chapter 3), the following coefficients are obtained:

$\eta_{arc} = 0.9$	$\eta_m = 0.8$
a = 0.15	c = 0.105
b = 0.3	d = 0.21

Figure 2.12 shows a plot of the heat efficiency functions inside the pool and the HAZ regions for Q=3000 J/s as a function of width using the values shown above. The difference between the two curves is the latent heat of fusion which is absorbed during welding so it does not contribute to temperature increase in the molten pool, but is released during solidification so it does contribute to the heat inside HAZ.

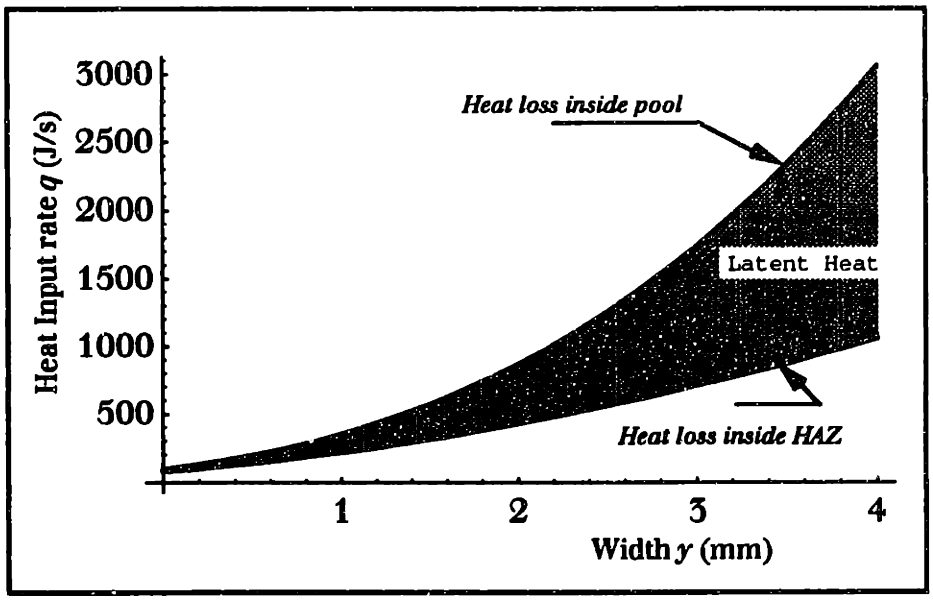


Fig. 2.12 Heat loss and storage inside the pool and HAZ regions.

The second factor which is important for decoupling the dynamics of the pool width from that of the HAZ is the fact that material thermal properties (such as thermal conductivity and diffusivity) change with temperature. This change can be as much as 300% (Touloukian, 1967). Therefore, in order to model heat transfer in the pool and the HAZ accurately, it is useful to include temperature-dependent coefficients. The following linear interpolations were included in the model as an approximation of the thermal coefficients:

$$\begin{cases} k = 85 - 0.052T \ (\mathrm{W/m^\circ K}) & \text{for } T < 1000^\circ \mathrm{K} \\ k = 30 \ (\mathrm{W/m^\circ K}) & \text{for } T > 1000^\circ \mathrm{K} \end{cases}$$

$$\begin{cases} c = 171.4 + 0.82T \ (\mathrm{J/Kg^\circ K}) & \text{for } T < 1000^\circ \mathrm{K} \\ c = 650 \ (\mathrm{J/Kg^\circ K}) & \text{for } T > 1000^\circ \mathrm{K} \end{cases}$$

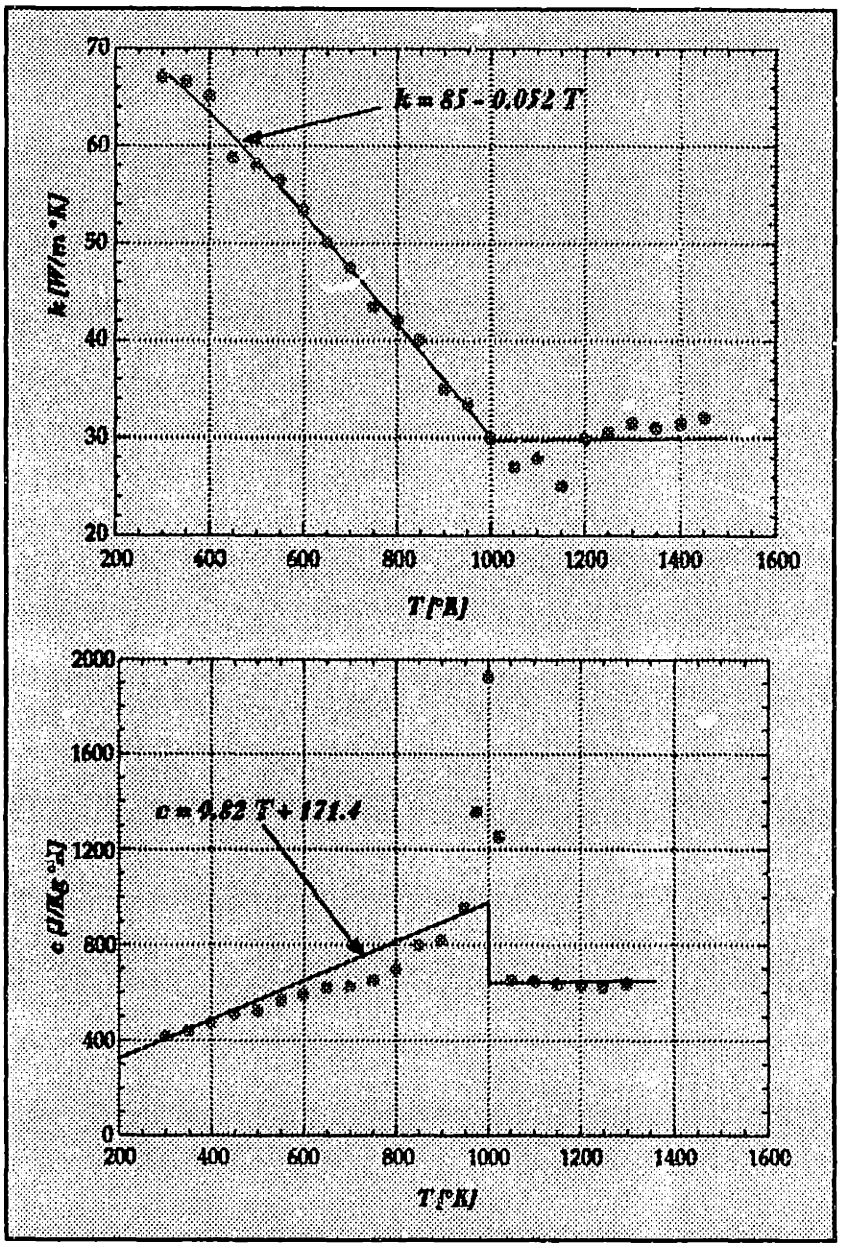


Fig. 2.13: Temperature-dependent thermal coefficients for mild steel (Touloukian, 1967)

The linear interpolations shown in Fig. 2.13 were included in the model as an approximation of the temperature-dependent thermal coefficients. Figure 2.14 shows the simulation results with heat losses and temperature-dependent coefficients. It is

shown that these two modifications to the model result in a significant improvement since the behavior exhibited in Fig. 2.14 is much closer to the experimental data (shown in Fig. 3.18).

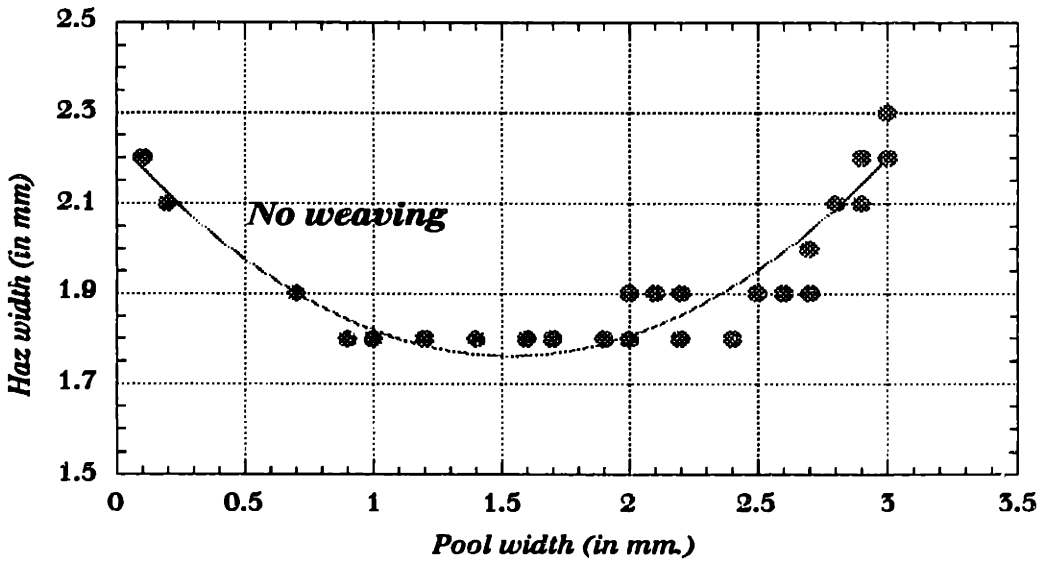


Fig. 2.14: I/O map with latent heat of fusion and temperature-dependent thermal coefficients.

2.5 Dynamic Process Model

A dynamic model of the welding process is useful for performing dynamical simulations and evaluating alternative control methods. The model used so far predicts steady state temperature distribution within a semi-infinite plate. It can be slightly modified to represent the transient behavior of the temperature distribution in the weldment. This was done by using a small mesh (of 20 elements) placed at the desired width as shown in Fig. 2.15. The model was implemented in a MATLAB Executable function (MEX-function) that solves the heat conduction equation, to calculate the temperature at any position with any initial value. The function has the following structure:

$$T(x,y,t) = temp(init\_t,\ delta\_t, init\_Temp, x, y, heat, vel, amp)$$

where:

init_t	Initial time t in seconds.
delta_t	Time step dt in seconds.
init_Temp	Initial temperature in °K.
x	position x of desired point in mm.
y	position y of desired point in mm.
heat	heat input Q in J/s.
vel	velocity in mm/s.
amp	Weaving amplitude in mm.

This dynamic model calculates the instantaneous temperature at a fixed point in a moving frame by creating a small line mesh (20 elements) along the direction of motion of the torch. It then solves the 3-D heat conduction equation at each of those points by using the temperature of the preceding point as initial conditions. This is an efficient way of calculating the transient temperature at a point without having to solve a large finite-element model. Using this small mesh (Fig. 2.15), allows us to calculate the initial temperatures at the point of interest for each time step. Otherwise, if the temperature at the same point and the previous time step is used, temperature will keep increasing even when the heat input is decreased.

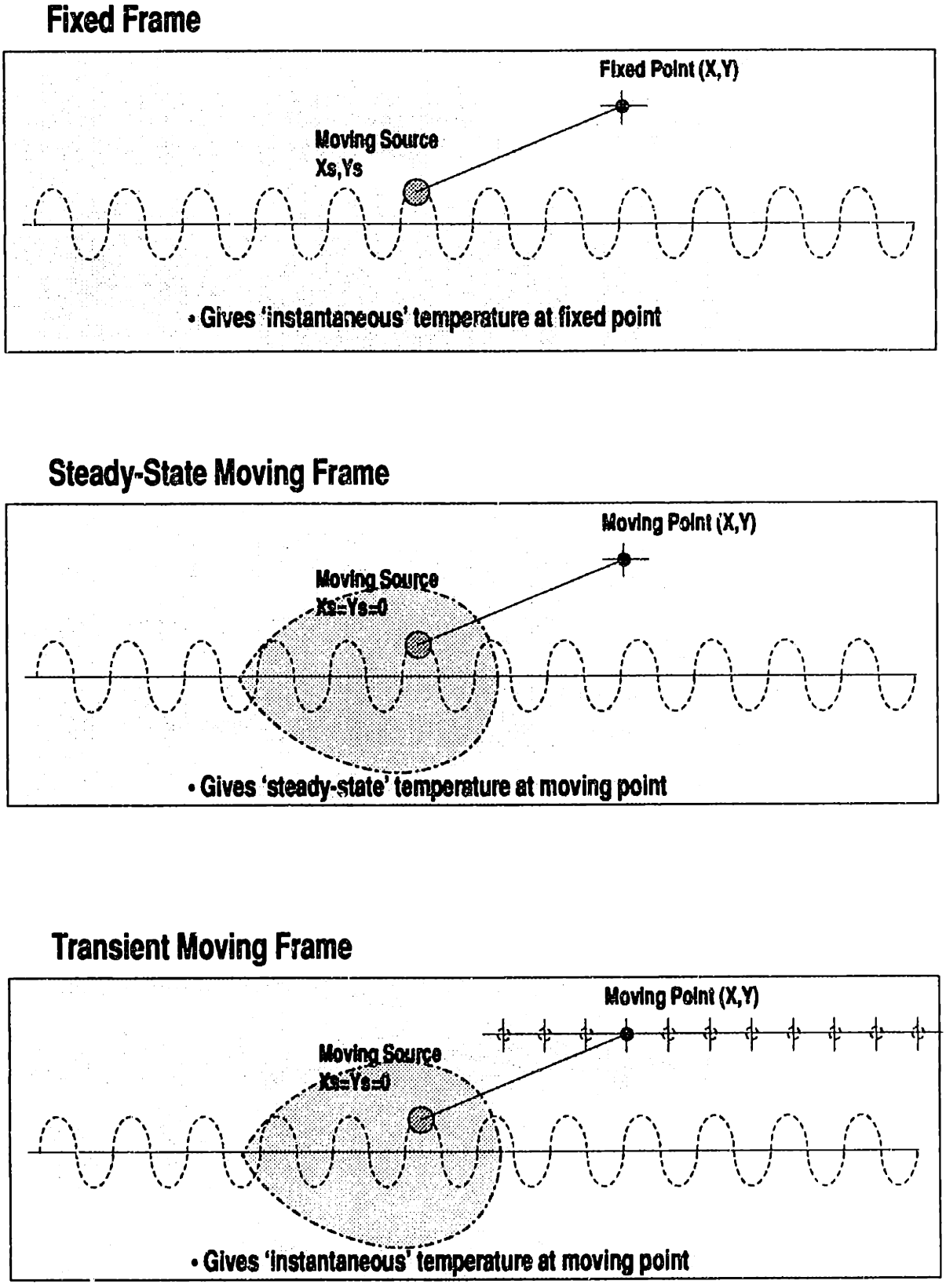


Fig. 2.15: Dynamic modeling using a 'fixed mesh' in a moving frame.

Figure 2.16 shows the dynamic response of the model when the inputs (weave amplitude and torch travel speed) are changed in an open-loop manner. The dynamic response exhibited in Fig. 2.16 is dependent on the operating conditions, which is a typical characteristic of nonlinear systems. This shows precisely the problem with using linearized models for arc welding. A linearized model is needed at each of the different operating conditions. Yet, the model used here captures the nonlinear dynamic properties quite easily, and is therefore helpful for dynamic simulations and control system design.

The model is shown to represent the transient behavior of the temperature distribution in arc welding accurately, and therefore is useful in performing dynamic simulations of alternative closed-loop control methods. The dynamic model can also be used in the input selection phase to select the best input pair, in a Two-input-two-output (TITO) control system. In this case, the two outputs that have been chosen for control are the molten pool width and the heat-affected zone (HAZ) width. Both of these outputs capture the geometric as well as thermal properties of the weld, and therefore give a good indication of the overall quality and strength of the weld joint.

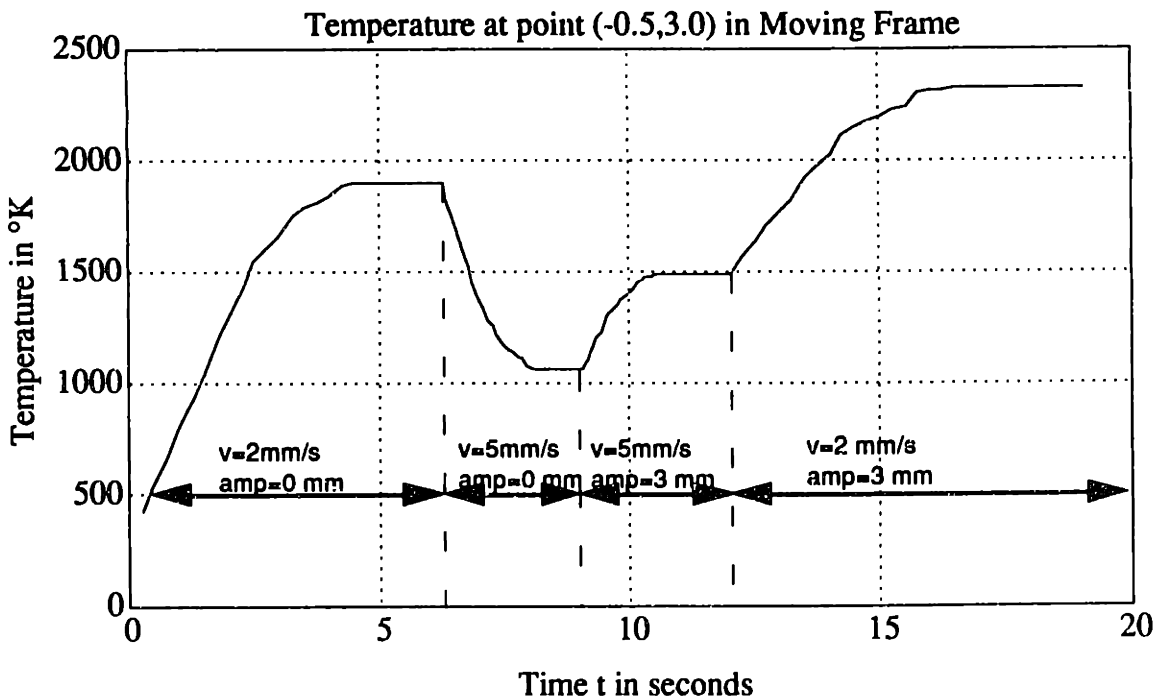


Fig. 2.16: Dynamic response of temperature as a result of changes in the welding parameters (heat input Q = 2000 J/s)

2.6 Conclusions:

An analytical model of heat conduction during welding was presented. This model was shown to accurately estimate the pool width and the heat-affected zone. The model requires calibration with experimental data.

A finite element model, first developed by Doumanidis (1988), was modified to include torch weaving and ported to the CRAY-II supercomputer. However, the granularity of the mesh makes it difficult to select the weave path freely. A simple heat transfer model based on numerical integration of the conduction equation has been developed and used to simulate rapid lateral motions of a gaussian-distribution heat source. This simple 3 -D heat transfer model was numerically integrated to calculate the temperature field in a semi-infinite plate due to a moving gaussian-distribution heat source.

A pure heat conduction model was shown to be inadequate for representing the complex dynamic response of the welding outputs. As a result, the model was modified to include nonlinear properties such as latent heat of fusion and temperature-dependent thermal coefficients. The modified nonlinear model was shown to accurately represent the welding process and was used to develop a dynamic model of the welding process. This model, implemented in a MATLAB subroutine, is useful for dynamic simulation and control system design.

Chapter 3

Effects of High-Frequency Weaving

A major obstacle to the successful implementation of multivariable control of welding processes, in addition to the complexity of the process itself and the lack of reliable sensors, has been the strong coupling between the process outputs. Simultaneous multivariable control of geometric characteristics of the molten pool and of thermal properties of the weld requires the ability to control the shape and size of the temperature isotherms, which is not possible with conventional welding processes. Previous experiments (Hale, 1989) using feedback control show that for a given pool width, the HAZ can be modulated at most about 0.5 mm and in many cases much less. Other studies (Doumanidis, 1990) show that a similar coupling exists between HAZ and the centerline cooling rate thus precluding independent control of both variables.

To achieve multivariable control of welding, it is therefore necessary to change the process by adding a new control input. Essentially, we are interested in controlling the heat distribution in the workpiece. Pre-heating and post-weld heat treatments are one way to achieve this, but they do not lend themselves easily to on-line process control. Another method to achieve output decoupling is to move the torch transversely at frequencies high enough to be averaged out by the heat diffusion in the weldment. In this chapter, we shall refer to this transverse motion as high-frequency 'weaving', to distinguish it from traditional manual weaving which is used to fill wider weld joints. The purpose here is not to make larger welds, but to modify and control the temperature distribution in the weldment as shown schematically in Fig. 3.1.

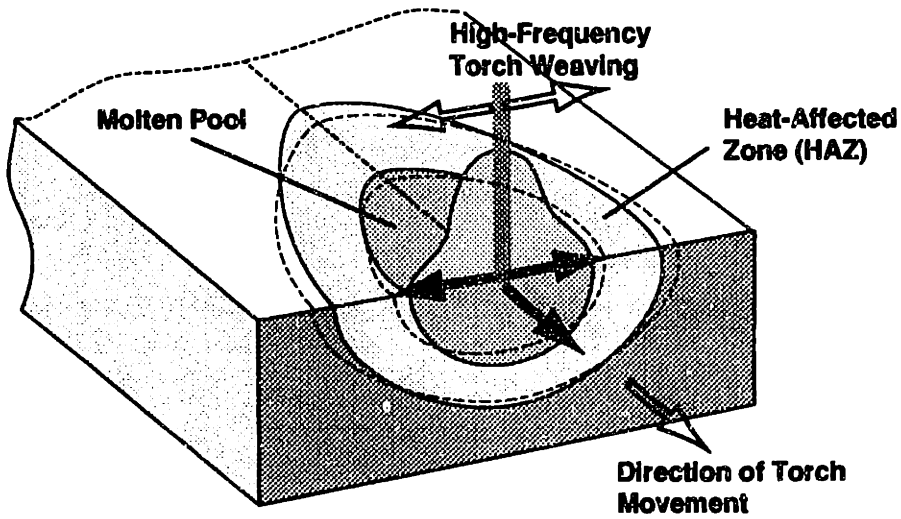


Fig. 3.1: Concept of variable distribution heat source using high-frequency weaving.

The effect of high frequency torch weaving as an additional control input of the welding process has not been studied in the literature, although the idea of torch weaving has a strong heuristic basis. Weaving is often used in manual welding, and previous research work has shown that weaving improves the microstructure of the joint due to the stirring effect (Makara and Kushnirenko, 1967). Weaving also creates less cracking and steam formation due to the lower cooling rate. Weaving is sometimes used in automated welding to provide through-arc sensing for seam tracking.

Experiments were conducted with High-frequency weaving using GTA bead-on-plate welds of mild steel plates. Input-Output correlation matrices indicate that the HAZ width is determined by the heat per unit length, and is basically independent of weaving amplitude. The weaving amplitude can be used to control the pool width in a decoupled feedback control system.

Both simulation and experimental data show that high-frequency torch weaving results in a significant increase in output decoupling. This will allow the development of strategies for independent multivariable control of geometric and thermal properties of arc weld.

3.1 Weaving in the Literature:

The metallurgical effects of weaving are not included in this study, but should be considered for a complete evaluation of High Frequency weaving. There have been several studies conducted to look at the effects of weaving on the metallurgical composition of the weld. In general, these studies have shown that weaving tends to improve the mircrostructure of the joint (particularly in the HAZ region) and results in less cracking and steam formation due to the lower cooling rate. Weaving has also been shown to reduce grain size in molten pool and HAZ, and to compensate for poor joint fit-up. The stirring effect of weaving also makes the microstructure more homogenous and less oriented.

In 1967, Makara and Kushnirenko showed that weaving with an amplitude of 3.5 mm and a frequency of 3 Hz. improves the weld microstructure by creating a more uniform heat distribution in the pool thus resulting in less columnar texturing and heterogeneity. They also showed that the mechanical properties of the joint improve as a result of weaving. Shtrikman and Pavlov (1983) showed that weaving causes reduction of the grain size and disorientation of the primary structure by changing the direction of solidification.

In 1986, Grong and Christensen developed a two-dimensional heat flow model based on the Rosenthal solution with a distributed heat source (constant intensity distribution) to model weaving of the torch. They showed that a distributed heat source of net power density q/2L on semi-infinite body gives better predictions of pool geometry than the classical point source solution.

In 1987, Yahata and Kamo showed that weaving improves the weld quality using mild-steel. They developed an improved weaving technique, consisting of pendulum and parallel motion, for the automation of pipe welding. Siewert et al. (1987) concluded that weaving changes the shape of the weld interface, but has little effect on the weld toughness of HY-80 welds. Hughes and Welduck (1987) built a ceramic torch/deflector for magnetic arc weaving in Robotic Plasma Welding. The deflector was capable of frequencies up to 100 Hz., while the amplitude depends on arc length. This method requires arc length measurement and calibration.

3.2 Simulation Results of Pure Conduction Model:

3.2.1: Without weaving:

A heat flow model based on numerical integration of the heat conduction solution, presented in Chapter 2, was used to simulate the effects of weaving on the desired outputs, namely the pool width and HAZ. Figures 3.2 and 3.3 show the results of simulations using this analytical heat transfer model.

Figure 3.2 shows that without weaving, the reachability range corresponds to a thin line in the output space. This illustrates the high coupling between the outputs and confirms the same results obtained by Hale (1990).

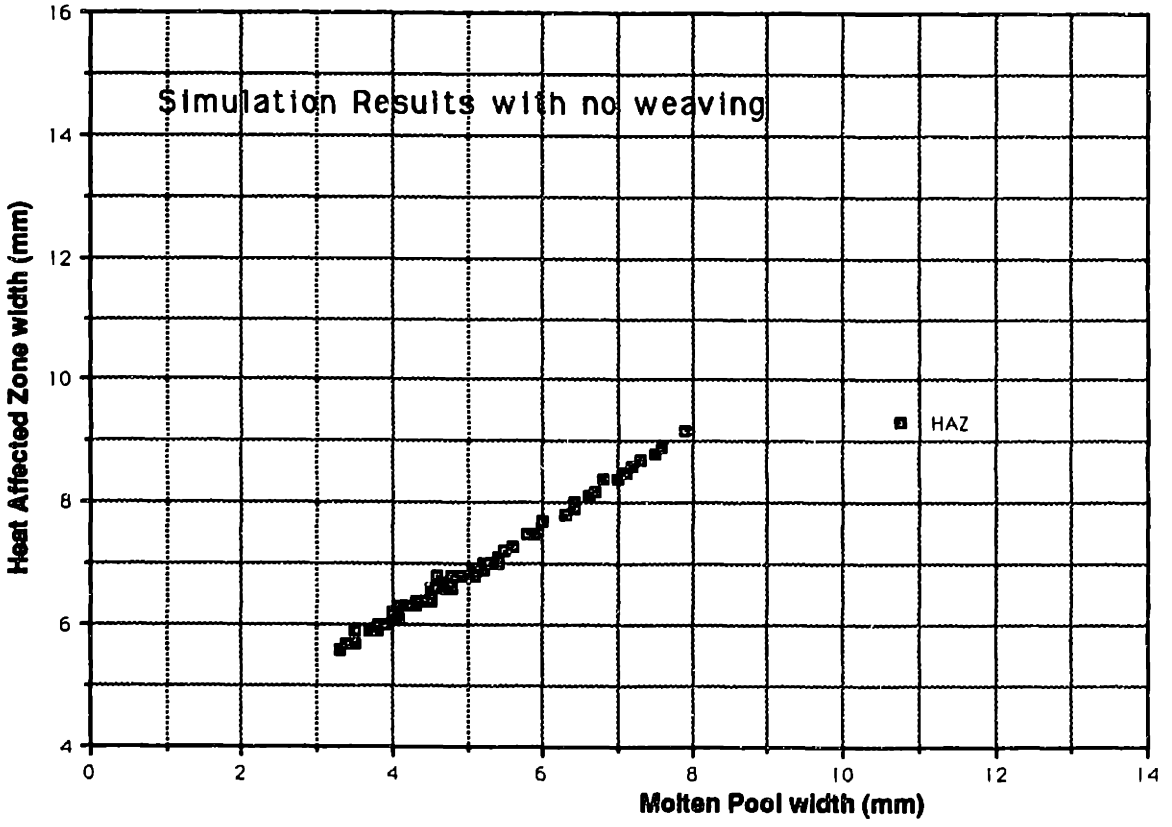


Fig 3.2: Reachability Range without weaving (1000 J/s < Q < 4000 J/s and 1 mm/s< v < 5 mm/s).

3.2.2 Modeling of High-frequency weaving:

In order to model high-frequency weaving of the torch, equation 2.14 was numerically integrated to calculate the temperature distribution on the upper surface of a plate using a moving source whose coordinates are given by:

$x_s = v\,t$ where v is the longitudinal velocity

$y_s = y_0 \sin(wt)$ where w is the weave frequency.

The model uses four control inputs:

The rate of heat input:	q(t) in Joules/sec
The torch velocity:	v in mm/sec
The weave amplitude:	y_0 in mm
The weave frequency:	w in rad/sec.

The outputs of the simulation are:

The pool width:	P_{width} in mm
The Heat-Affected Zone width:	HAZ in mm.

The maximum temperature at a given point is given by:

$$\frac{\partial T}{\partial t} = 0$$

and is reached at time:

$$t = \frac{r^2}{6\alpha}$$

Heat-per-unit length:

To be able to compare the simulation results for welding with weaving with welding without weaving, the same amount of heat per unit length of weld (q/v) must be used. Since we have a sinusoidal weave motion ($y = y_0 \sin(w\,t)$), the total velocity V_{total} is given by:

$$V_{total}^2 = V_x^2 + V_y^2$$

Thus choosing: $V_x = V_0 \cos(w\,t)$,

and : $V_y = w\,y_0 \cos(w\,t)$,

gives:

$$V_{total} = \cos(\omega t)\sqrt{V_x^2 + \omega y_0^2}$$

Therefore, in order to keep (q/v) constant at all times, the rate of heat input q must be chosen such that:

$$q = q_0 \cos(w t)$$

The results shown in figure 3.3 include data for three cases of heat input modulation. One using the above equation, $q = q_0 \cos(w t)$, thus keeping a constant heat per unit length. The other uses a constant heat input, $q = q_0$, thus releasing more heat on the outside areas than on the inside, and the third method uses $q = q_0 \sin(w t)$, which means putting no heat on the centerline and most of the heat on the outside points. It is noted that each one of these methods increases the output range in a particular region i.e. larger HAZ widths for $q = q_0$, and smaller HAZ widths for $q = q_0 \sin(w t)$.

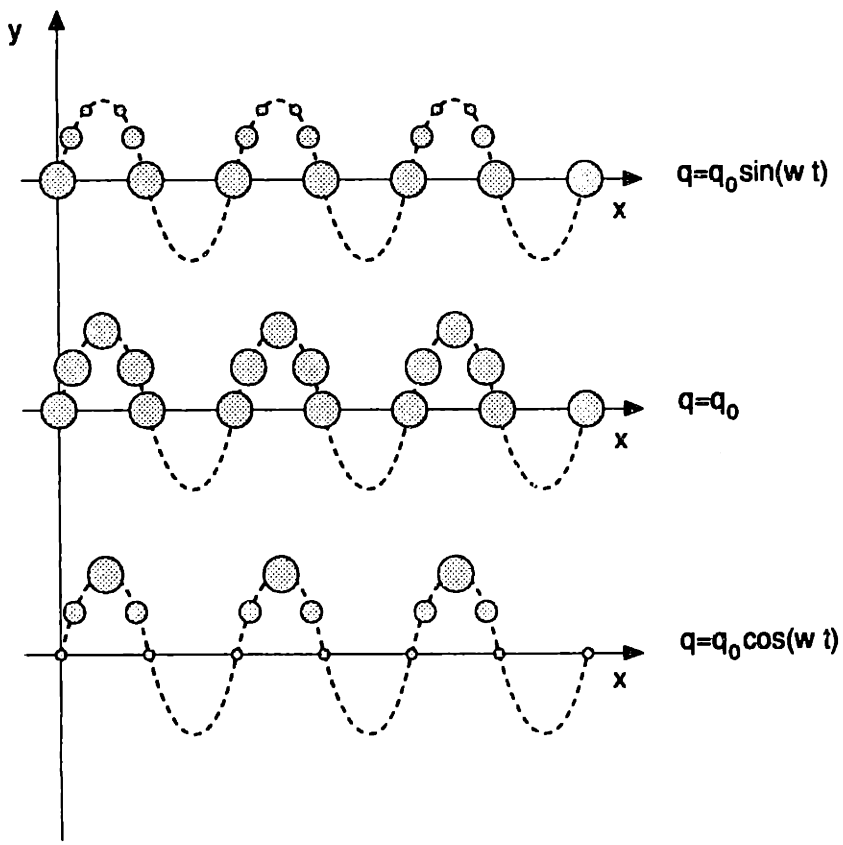


Fig 3.3: Possible Heat input modulation during weaving

Figure 3.3 shows that modulation of the heat input as a function of torch position produces a different heat distribution in the weldment. This property can be useful in decoupling the outputs for different materials and applications. Fig. 3.4 shows simulation results with weaving when the heat input is modulated a function of distance from the centerline. Modulating heat input as a sine function of time produces welds with small pool widths because most of the heat is deposited on the outside of the weld. Modulating heat input as a cosine function produces weld joints with larger pool widths and smaller HAZ because most of the heat is deposited on the inside of the weld. Cant heat input produces weld joints with large pool and HAZ widths.

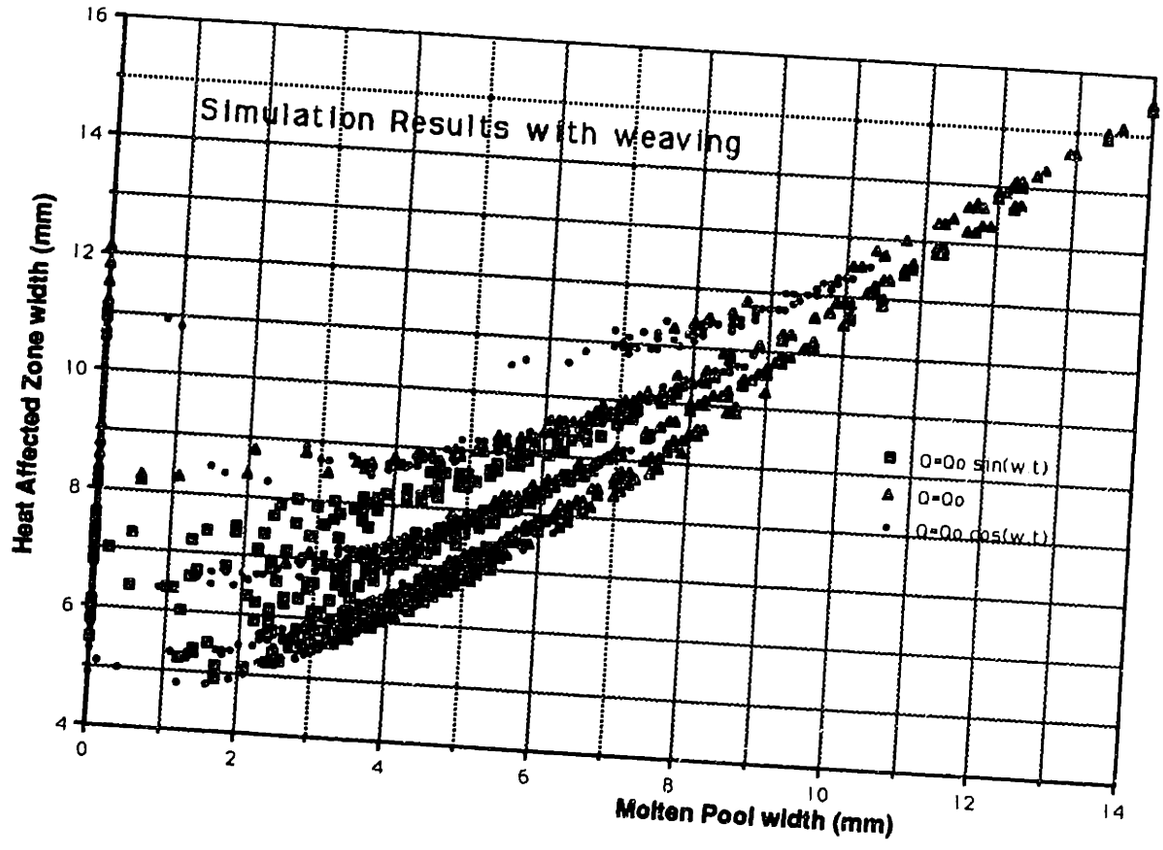


Fig. 3.4: Reachability range with weaving and heat input modulation (1000 J/s < Q < 4000 J/s and 1 mm/s< v < 5 mm/s).

Fig. 3.5 shows lines of constant heat-input-per-unit-length. The data points are shown along with second order polynomial curve fits (goodness of fit R = 0.95). It is shown that varying the heat-input-per-unit-length shifts the line inside the reachability range up or down. The first data points on the left of the curves represent the no weaving case. As the weave amplitude is increased, both the pool width and the HAZ are increased (the pool width increases much more).

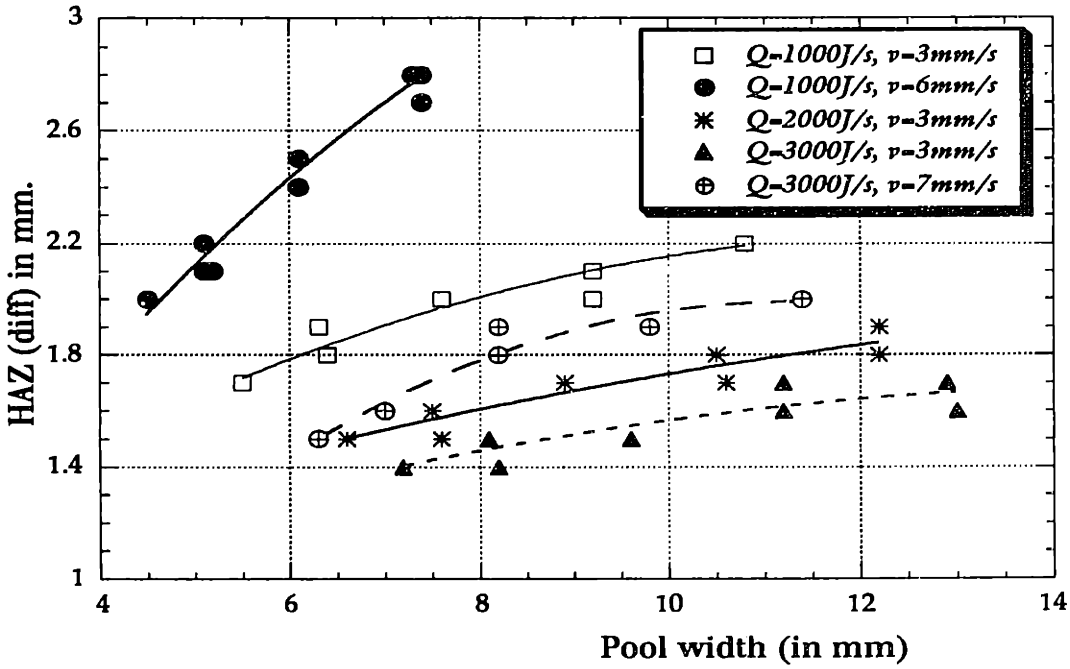


Fig. 3.5: Lines of constant heat-input-per-unit length

Fig. 3.6 shows lines of constant weave amplitude while varying all other inputs. It is shown that weave amplitude has a significant effect on the outputs as each amplitude gives a separate curve on the output map. This means that weave amplitude can be used as a control input to decouple the two desired outputs. In Fig. 3.6, the HAZ width is defined as the difference between the Ac_1 and the Ac_2 isotherms. Plotting the difference rather than the total HAZ width is more descriptive of the second order dynamic behavior that is intrinsic in the process, and will therefore be used in the remaining sections of this thesis.

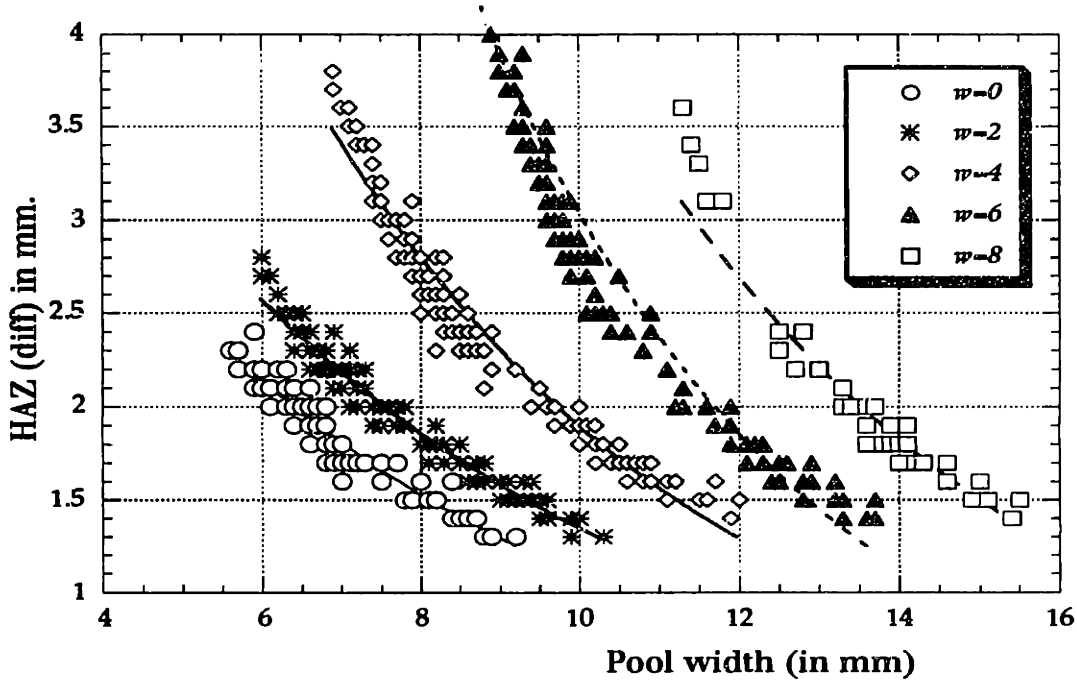


Fig. 3.6: Effects of weave amplitude on reachability range.

Figure 3.6 shows that weaving significantly increases the area of reachability as each weave amplitude produces a separate line on the I/O map. Along the same constant weave amplitude line, HAZ width decreases with increasing pool width. As explained in Chapter 2, this does not agree with experimental data, and is an intrinsic limitation of a pure conduction model. This trend can be changed if the latent heat of fusion and temperature-dependent coefficients are included in the model, as explained in the next section.

The combination of Fig. 3.5 and 3.6 reveals an interesting property of this system. As shown in Fig. 3.7, one may pick a prescribed point inside the reachability region defined by a specific pool width and HAZ width, and that point will lie at the intersection of a constant weave amplitude curve and a constant heat-per-unit-length (Q/v). Therefore the weave amplitude y_0, should be used as an input along with either the rate of heat input Q or the forward velocity v. Using travel speed as a control input yields a more flexible system since it can generate an almost infinite heat-per-unit-length as v approaches zero.

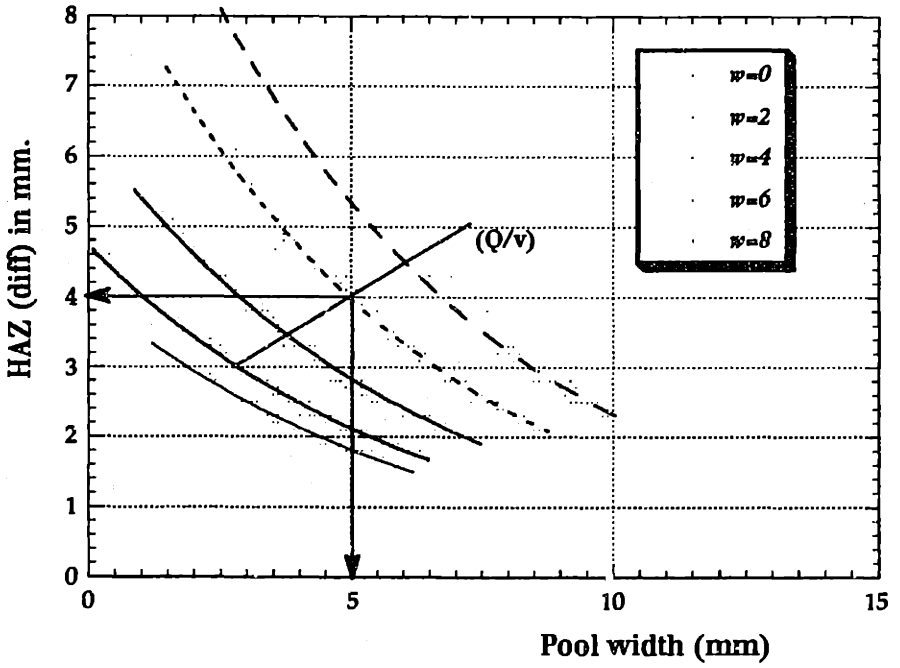


Fig. 3.7: Effect of weave amplitude and heat-input-per-unit-length on output Map

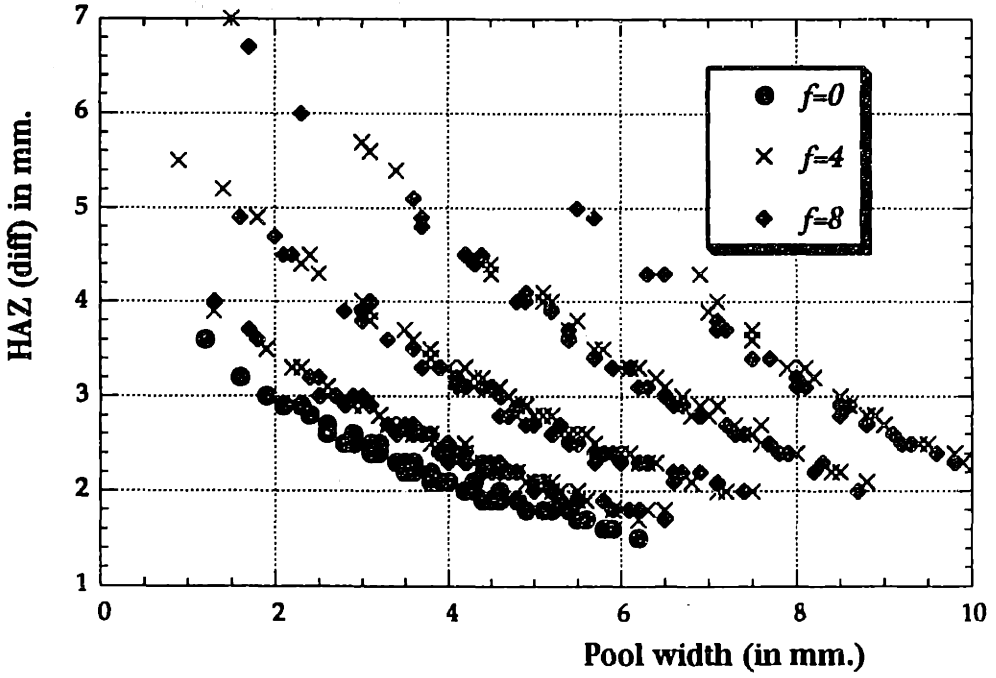


Fig. 3.8: Effects of weaving frequency on I/O map (simulation)

Figure 3.8 shows the effects of varying the weave frequency. Data points for $f = 4$ Hz and $f = 8$ Hz are shown to cover the entire reachable range. Thus, weave frequency does not have much effect on the outputs as long as it is larger than the bandwidth of the heat diffusion in the weldment (about 2 Hz. for mild steel).

3.3 Effects of Latent heat of fusion and temperature-dependent coefficients:

As shown in the previous section, a pure conduction model does not accurately predict the dynamics of the welding process. There are significant nonlinearities that are intrinsic in the welding processes and which must be included in the model. In this section, the model is modified to include latent heat of fusion, heat losses due to convection and radiation, and temperature-dependent thermal coefficients, as determined in chapter 2. Simulation results, Fig. 3.9, show that these modifications increase the model's accuracy in comparison with experimental data (both those of Hale and those presented in section 3.5). At low heat-input-per-unit-length, heat conduction is the dominant factor, and therefore the pure conduction model is a good approximation.

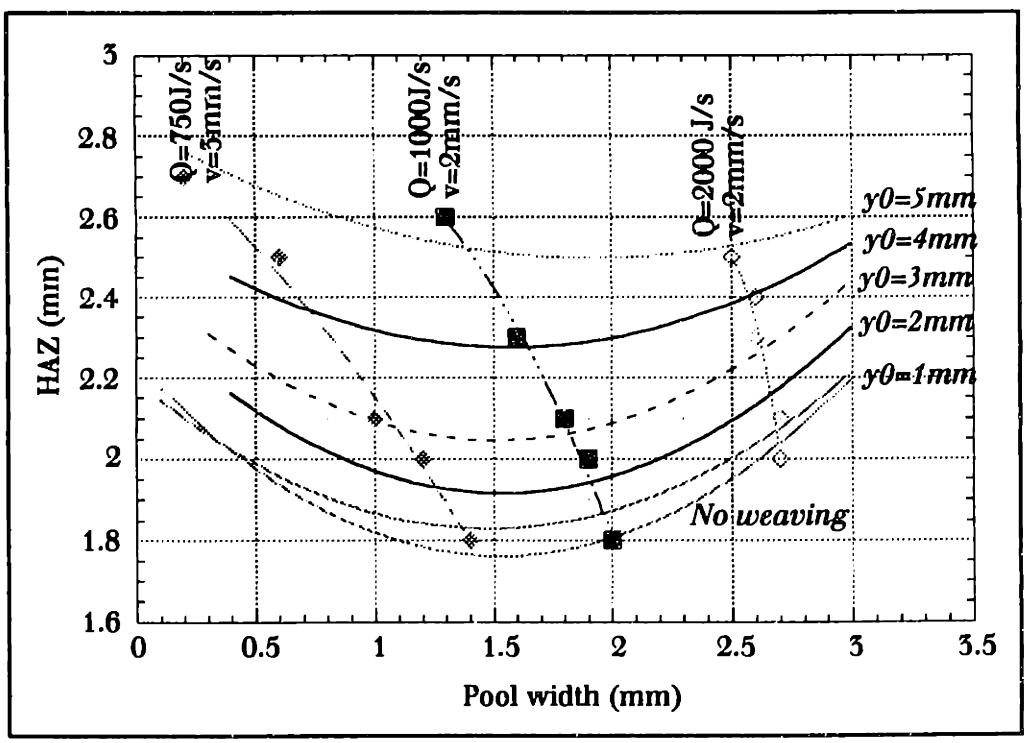


Fig. 3. 9: Effect of weave amplitude on output Map

However, at higher heat inputs, the nonlinear effects of latent heat of fusion and temperature-dependent coefficients become more important and drastically change the response of the process. The pure conduction model is hence not valid at large heat inputs. As shown in Fig. 3.9, the weave amplitude and the heat input-per-unit-length (Q/v) are sufficient to independently control the process outputs within the area of reachability.

3.4 Effects of weaving on the isotherms:

The effects of weaving on the heat distribution in the weld joint are illustrated in Fig. 3.10. The isotherms of the melting temperature and the HAZ temperature are shown with and without weaving for Q=2000 J/s and v=2 mm/s. Without weaving, the shape of the isotherms were fixed for a given welding process, and changing the heat-input-per-unit-length changes only the size of the isotherms and not their general

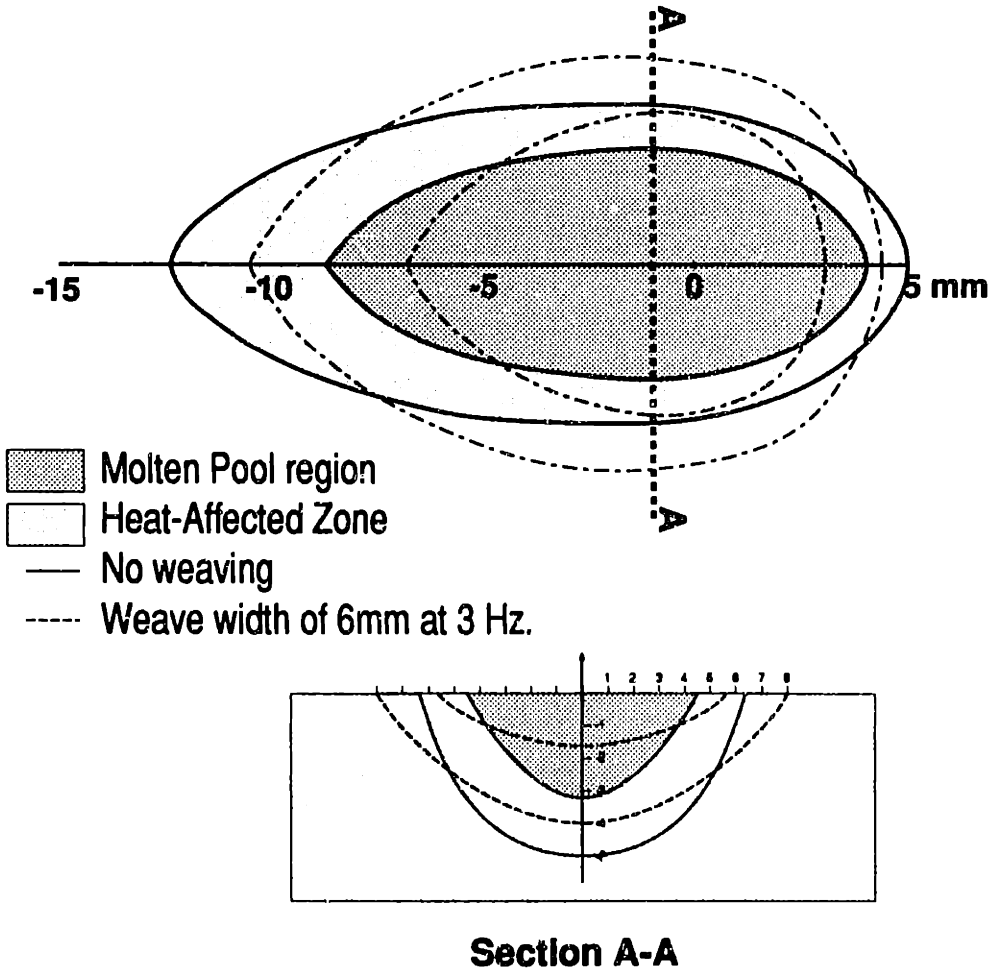


Fig. 3.10 Effect of High-frequency weaving on temperature isotherms for mild steel

shape. Weaving provides control over the shape as well as the size of the isotherms thus resulting in a variable heat distribution in the workpiece. This makes multivariable control possible. Fig. 3.10 shows that high-frequency weaving makes the isotherms shorter along the x-direction hence making the welding process more efficient.

3.4.1. Effect of material properties: Aluminum

Arc welding of Aluminum has a different dynamic response than mild steel because of its much higher diffusivity ($\alpha = 6.1\ 10^{-5}\ m^2/sec$). A much higher heat input must be used to melt Aluminum because heat is diffused much more quickly.

Fig. 3.11 shows the simulation of Aluminum welding using the conduction model with latent heat of fusion, heat losses due to convection and radiation, and temperature-dependent thermal coefficients. A heat input of 12,000 J/s, and a travel speed of 7 mm/s were used in the simulation. The continuous lines show the isotherms without weaving and the dotted lines show the isotherms with weaving.

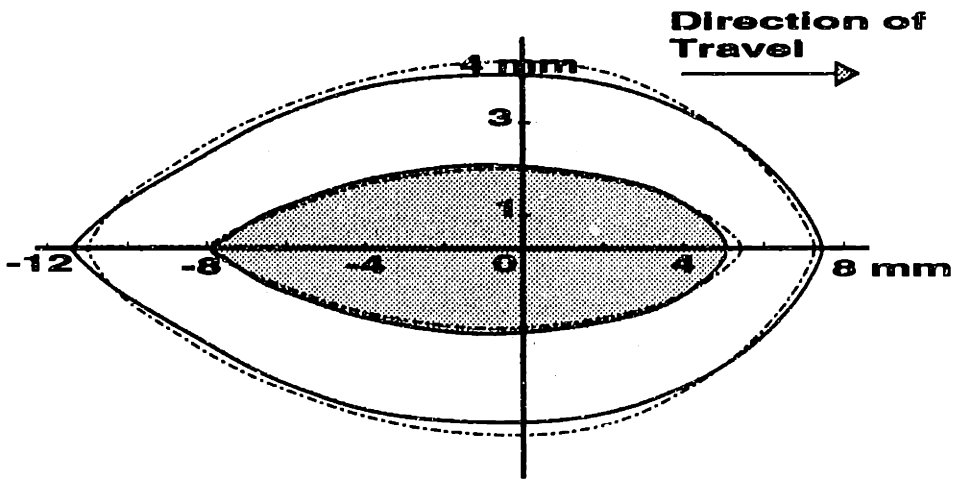


Fig. 3.11 Simulation of Aluminum Welding with high-frequency weaving (Q=12,000 J/s and v=7 mm/s).

It is seen in Fig 3.11 that weaving has less effect in the case of aluminum than in the case of mild steel. The latent heat of fusion and temperature-dependent coefficient are

modeled as (Touloukian, 1967):

$$\begin{cases} k = 260 - 0.052T \ (\mathrm{W/m^\circ K}) & \text{for } T < 725^\circ \mathrm{K} \\ k = 188 \ (\mathrm{W/m^\circ K}) & \text{for } T > 725^\circ \mathrm{K} \end{cases}$$

$$\begin{cases} c = 630 + 0.697T \ (\mathrm{J/Kg^\circ K}) & \text{for } T < 725^\circ \mathrm{K} \\ c = 1130 \ (\mathrm{J/Kg^\circ K}) & \text{for } T > 725^\circ \mathrm{K} \end{cases}$$

3.5 Experimental Investigation of High-Frequency Weaving:

Open-loop GTAW experiments were conducted to study the effects of high-frequency weaving on the weld pool width and Heat-Affected Zone width. Experimental data was collected off-line by sectioning and measuring the pool and HAZ shape for different inputs of weaving amplitude and frequency. Gas Tungsten Arc Welding was used in these experiments because GTAW does not include metal addition, contrary to Gas Metal Arc Welding, and is therefore much closer to the assumptions used in deriving the model.

A straightforward and inexpensive way to achieve high frequency weaving is by electromagnetic arc modulation. An externally controlled magnetic field can be used to deflect the arc at very high frequencies (up to 100 Hz). However, this method creates unpredictable changes in field/arc characteristics and the arc may become unstable at high frequencies and amplitudes. Electromagnetic fields create arc deflections and uncertainties about the exact location of the arc. These uncertainties tend to confuse the causal relationships between the inputs and the outputs. Therefore, it was decided to build a servo-mechanism with microprocessor-based control which can accurately control the torch position (see Fig. 3.12). The main advantages of this high-precision, high-frequency weaving mechanism is its ability to instantly vary the weave width and frequency, and to select various weave patterns (sinusoidal, triangular, and square waves). The weaving mechanism has an accuracy of 0.05 mm.

3.5.1 Experimental Setup:

The GTA Welding apparatus used in the welding experiments is shown in Fig. 3.12. A 33 MHz. 386 IBM Compatible computer was used to control the inputs to the system consisting of a Hobart Cybertig III power source, a weaving mechanism, and an X-

position servo-system to regulate the longitudinal velocity. These experiments are basically open-loop, except for the weaving mechanism which employs position and velocity feedback control.

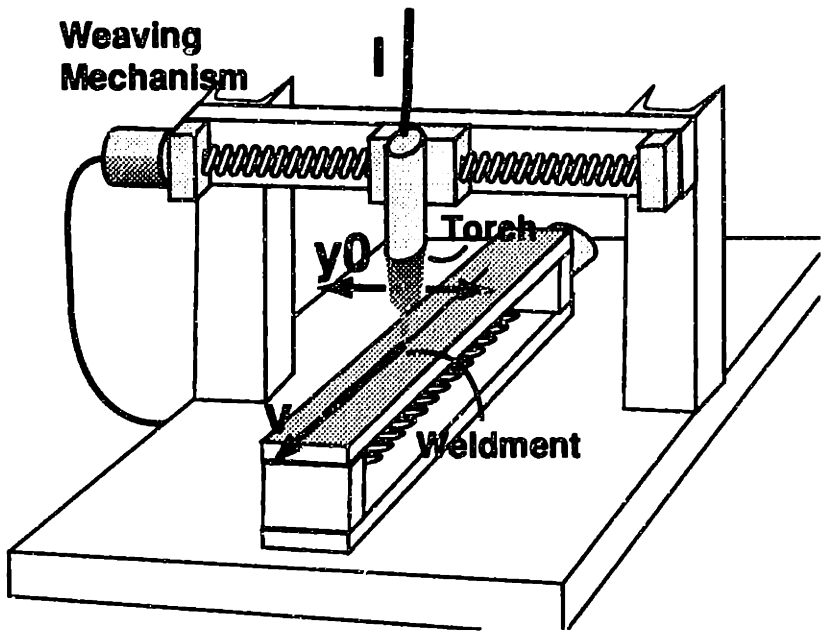


Fig. 3.12: Experimental apparatus for high-frequency weaving.

One advantage of this system, compared with electromagnetic arc weaving, is its ability to select various shapes (sinusoidal, triangular, and square waves) and also to vary the weave amplitude and frequency. Fig. 3.13 shows a functional block diagram of the control system used in the weaving servo-mechanism. Fig. 3.14 shows some of the traditional weave patterns used in manual welding.

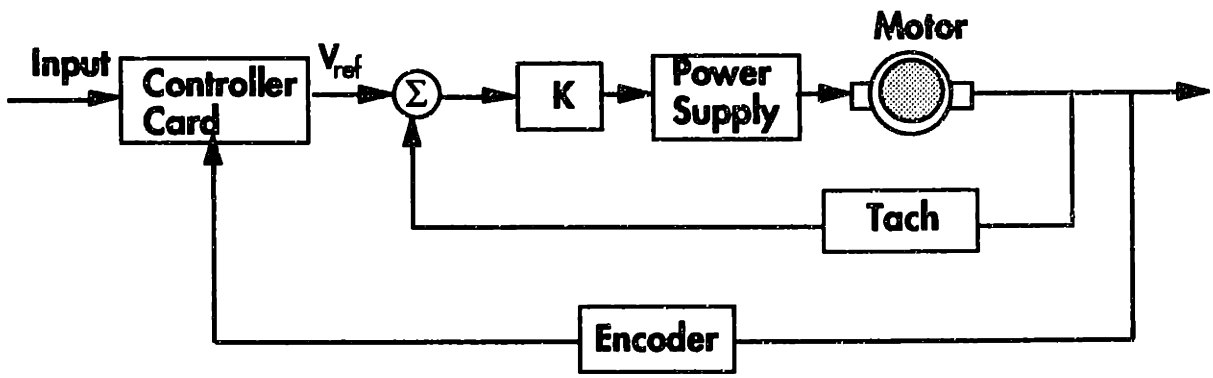


Fig. 3.13: Feedback Loops for the Servo-Controller

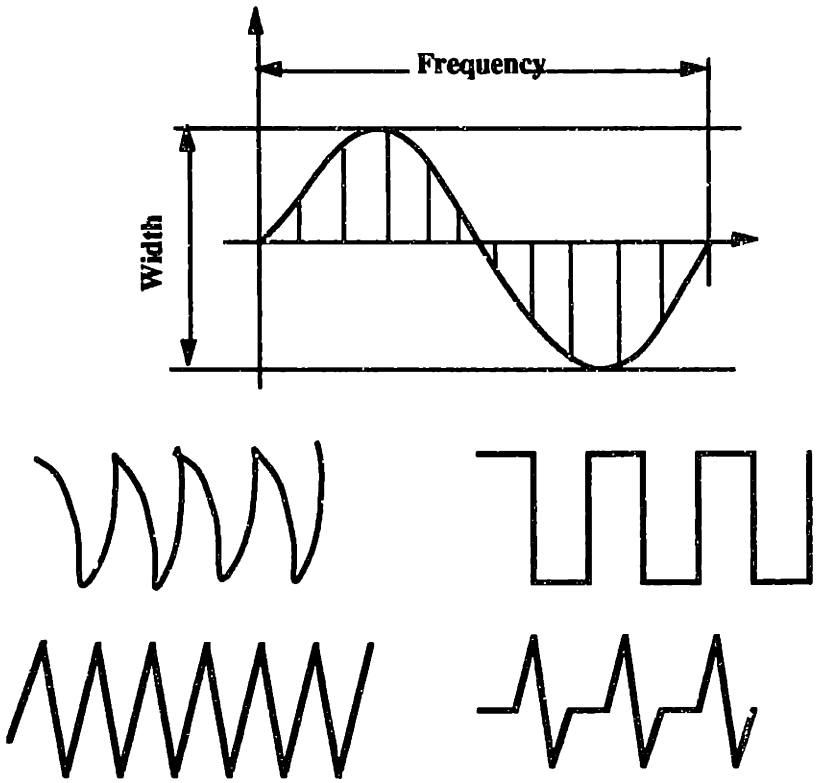


Fig. 3.14: Possible weaving Patterns.

3.5.2 Description of the experiments:

More than 300 open-loop experiments were performed bead-on-plate using A36 mild steel 0.5 in. thick, 2 in. wide, and 21 in. long. The plate surface was thoroughly cleaned with a rust remover prior to welding. The hardware configuration used in the experiments is illustrated in Fig. 3.15. A Heliarc HG-20 water-cooled torch equipped with 1/36 in. diameter 2% thriated tungsten electrode was used in the experiments. The shielding gas used is 98% Argon, and 2% Oxygen flowing at 30 cu ft/hr. The power supply unit is a Hobart Cybertig III GTA welding supply with a bandwidth of 0-1 kHz. for operation of up to 200 Amps. The forward velocity was varied from 1 to 7 mm/sec. The input current was varied from 70 to 130 A. Weaving amplitudes range between 0 and 5 mm and frequencies between 0 and 7 Hz.

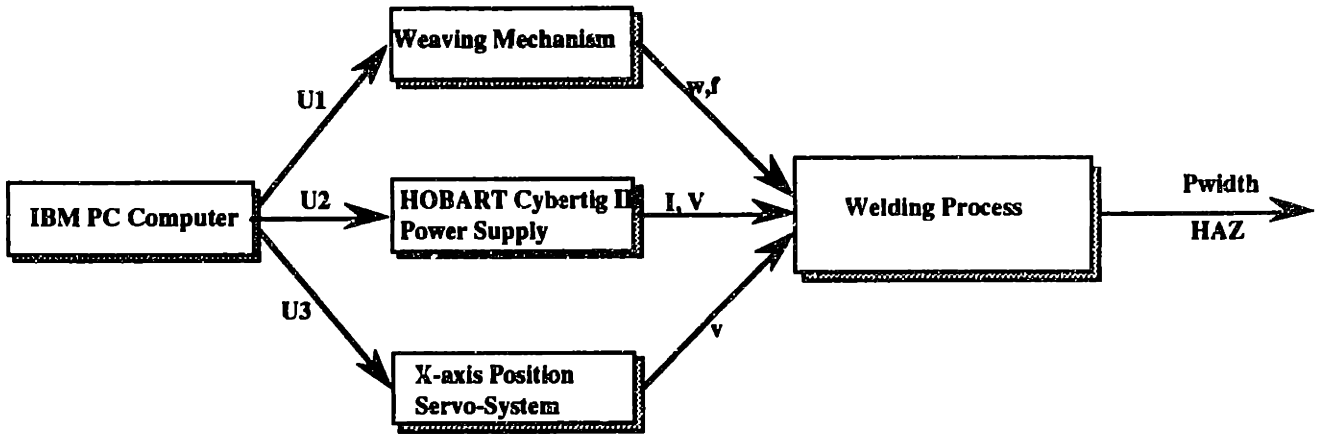


Fig. 3.15: Open-Loop GTAW Experimental Setup

After welding, the joints were sectioned, milled, ground, and etched using 2% nitric acid. Each weld was sectioned twice, and one side was photographed. Measurements of pool width, depth, and HAZ width were performed on one side directly with a caliper, and on the other side from the photographs. These measurements were averaged to reduce the effects of measurement errors. As shown in Fig. 3.16, the measurements are very noisy (standard deviation = 0.4 mm). This noise is due in part to measurement errors and also to the sensitivity of the GTAW process itself (these tests were performed without closed loop control).

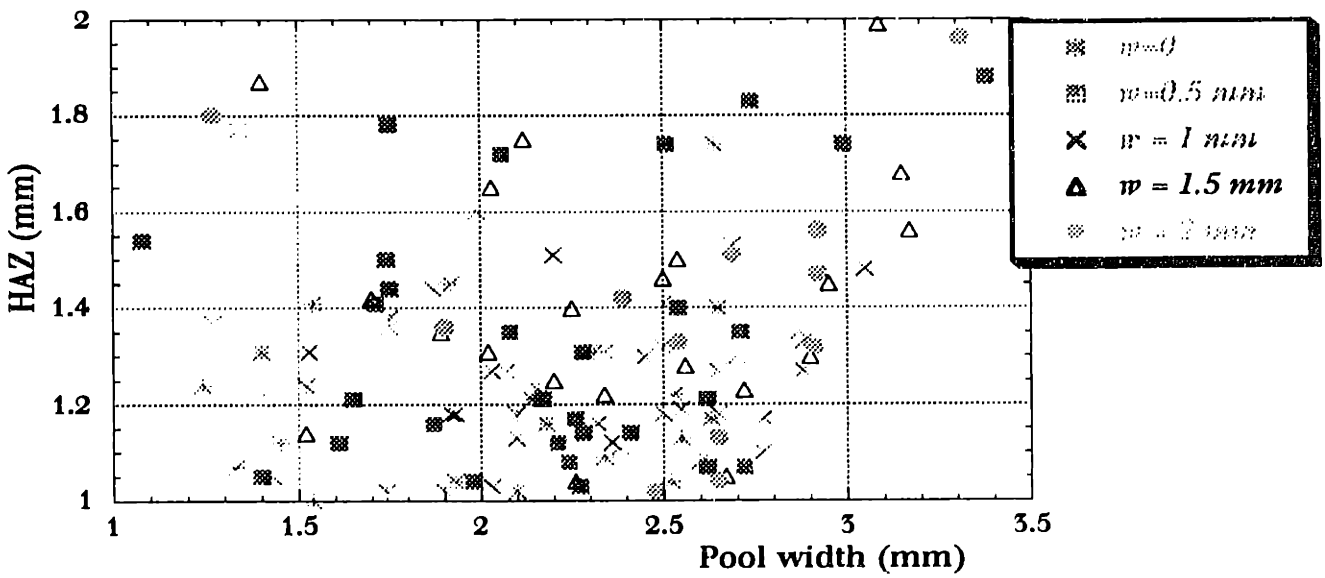


Fig. 3.16 GTAW experimental data as a function of weave amplitude (raw data)

The problem of noisy measurements is always present in the GTAW experiments shown in this chapter. Therefore, instead of showing the noisy data, the graphs shown in the remaining of this chapter will contain second order polynomial curve fits in order to look for any trend in the data. The standard deviation of the measurements was always between 0.25 and 0.5 mm.

Figure 3.17 shows second order polynomial curve fits of the raw GTAW data. Despite the noisy measurements, a clear trend emerges, and the effects of high-frequency weaving on the reachability range are clearly illustrated. Without weaving, this range is limited to the bottom line on the plot, which is in agreement with previous experimental data (Hale, 1990) and illustrates the high level of coupling between the pool width and the HAZ width. Using weaving moves this line up and provides a larger range of reachability. Fig. 3.18 also indicates that the weave amplitude y_0 has a strong effect on output decoupling since every value of y_0 gives a new line on the plot.

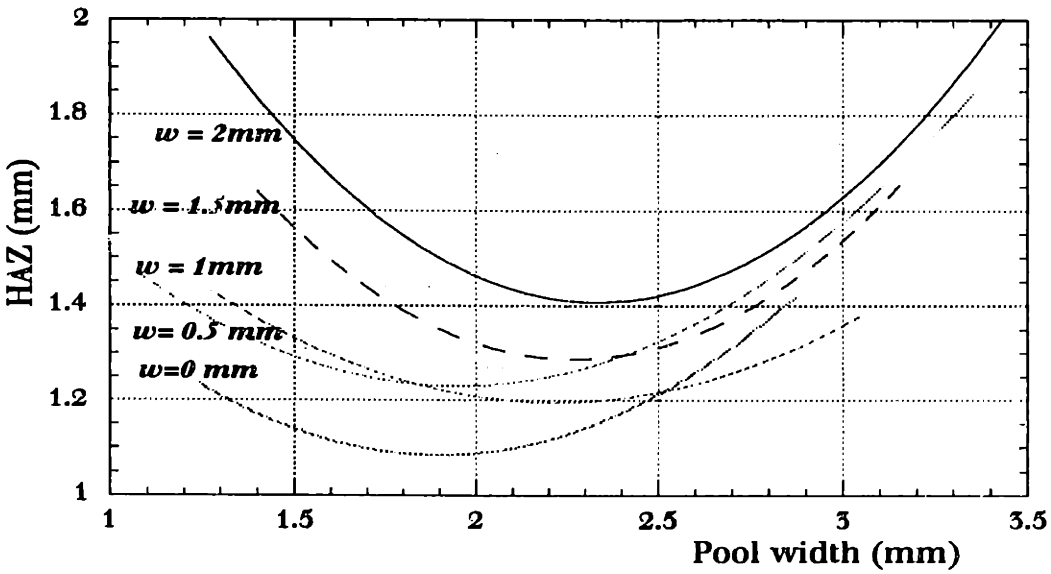


Fig. 3.17 GTAW experiments with high-frequency weaving (polynomial curve fits)

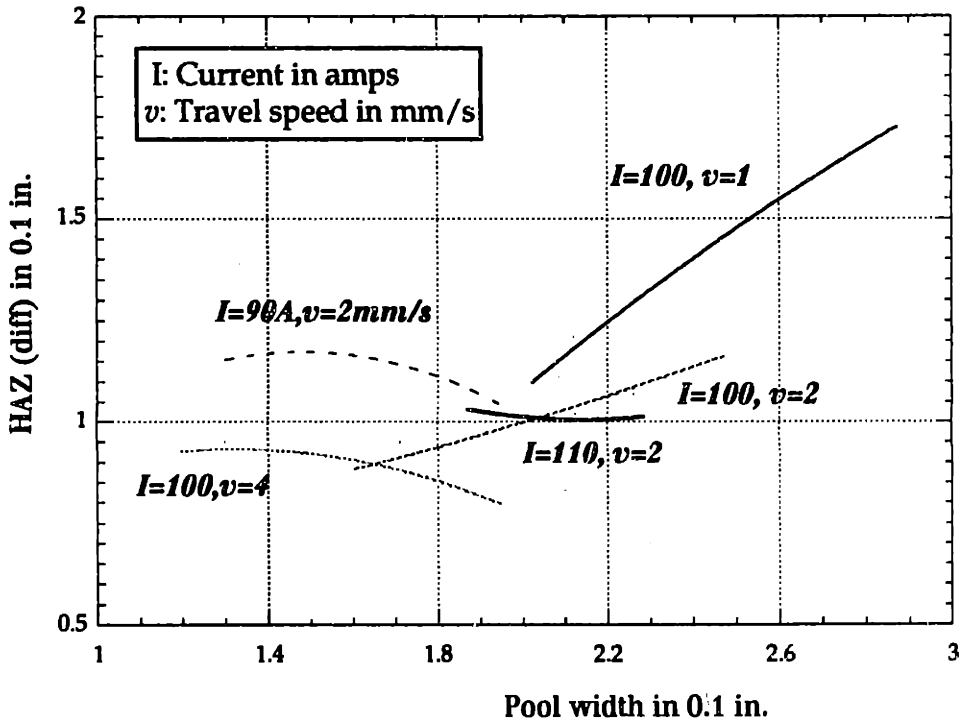


Fig. 3.18: GTAW experiments: Lines of constant heat-input-per-unit-length

Figure 3.18 shows lines of constant heat input per unit length. The point on the extreme left represents the no-weaving case. At low heat inputs, the lines are almost horizontal suggesting that weaving increases the pool width while keeping HAZ width almost the same. Increasing weave amplitude increases the pool width for the same heat input per unit length. At lower travel speeds (higher heat input per unit length), HAZ increases with increasing pool width. This reflects the smaller time constant of the HAZ in comparison withthe pool dynamics.

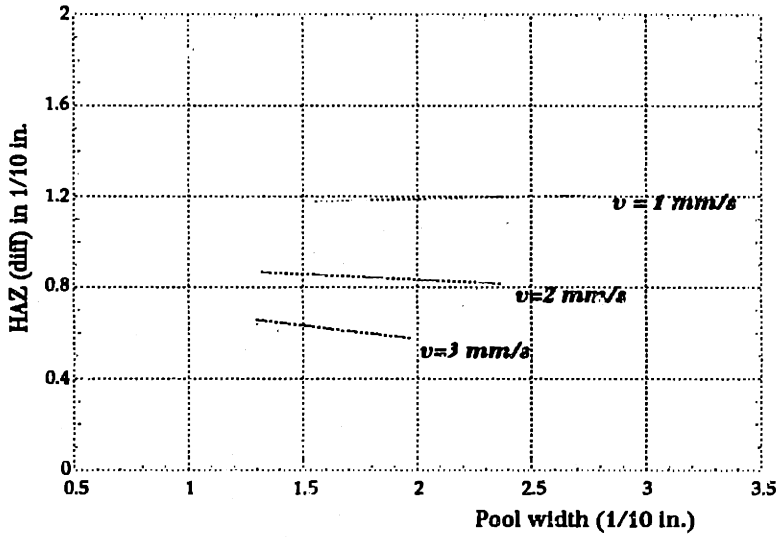


Fig. 3.19: GTAW Experimental Results: Lines of constant heat-per-unit-length (current I = 85 A)

Figure 3.19 shows three curves for a relatively low heat input (current of 85 A). The curves are almost horizontal suggesting that weaving increases the pool width while keeping HAZ width constant. For each curve, the heat input per unit length (Q/v) is constant while the weaving amplitude is changed.

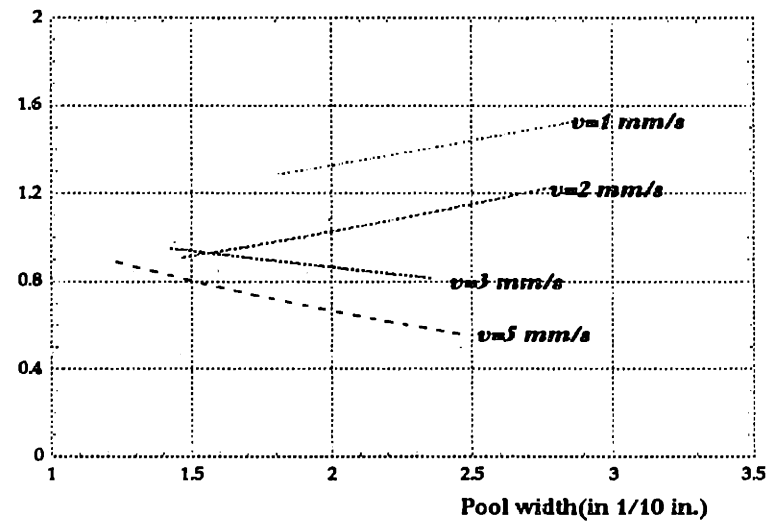


Fig. 3.20: GTAW Experimental Results: Lines of constant heat-per-unit-length (current I = 100 A)

Fig. 3.20 shows the constant longitudinal velocity lines for a current of 100 A. At large travel speeds (low heat per unit length), weaving causes a decrease in HAZ width and a simultaneous increase in pool width. At slower velocities, HAZ increases with increasing pool width. This reflects the smaller time constant of the HAZ with respect to that of the pool dynamics.

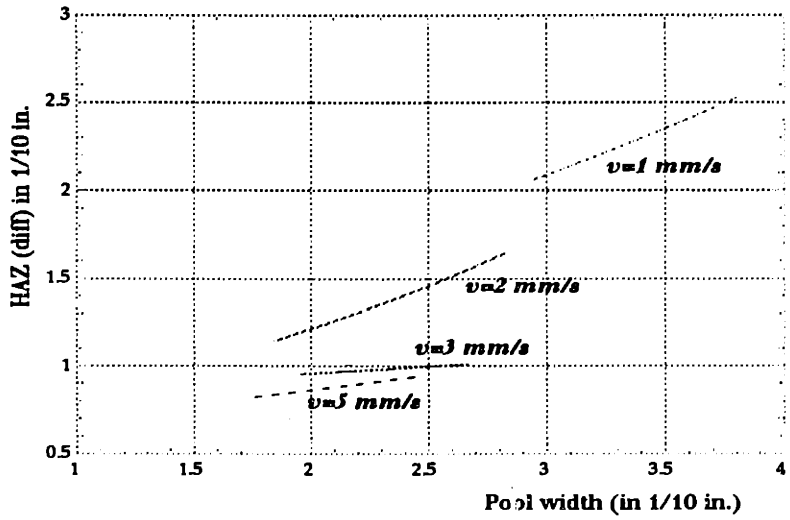


Fig. 3.21: GTAW Experimental Results: Lines of constant heat-per-unit-length (current I = 130 A)

Fig. 3.21 shows the constant velocity lines for a large heat input (current I = 130 A). This is a high heat input for the kind of material and electrode we are using. As before. for high velocities (low heat per unit length), HAZ is almost constant for increasing pool width. However, for smaller velocities, HAZ increases with increasing pool width.

3.6 Input-Output Relations

Input/Output maps are useful for looking at the relationships between each the inputs and the outputs and also for finding the steady-state gains used for controller design purposes. Fig. 3.22 shows the simulation I/O map for a constant rate of heat input of Q=2000 J/s. It is shown that travel speed has a strong effect on the pool width and little effect on the HAZ width. The weave amplitude has a strong effect on the HAZ width as shown in the Amplitude-HAZ plot and the velocity-HAZ plot. Each value of the weave amplitude is represented by a separate curve in the velocity-HAZ map hence weave amplitude can be used for decoupling pool width and HAZ width.

Pool width is controlled by the heat input per unit length (Q/v) so increasing the heat input has the same effect as decreasing the velocity. In controller design, therefore, it is simpler to use either v or Q to control the pool width, and keep the other constant. The velocity v is easier to use as a control input since it can provide infinite heat-per-unit-length (at v=0 mm/s). Therefore, in the control system design and experiments of chapters 4 and 5, the heat input Q will be fixed, and travel speed v and weave amplitude w, will be used as control inputs.

Figure 3.22 shows the Input/Output map for a constant heat input of 2000 J/s. Looking at the amplitude vs. pool width map, it is seen that travel speed strongly affects pool width while weave amplitude does not have a big effect on pool width, especially at low travel speeds.

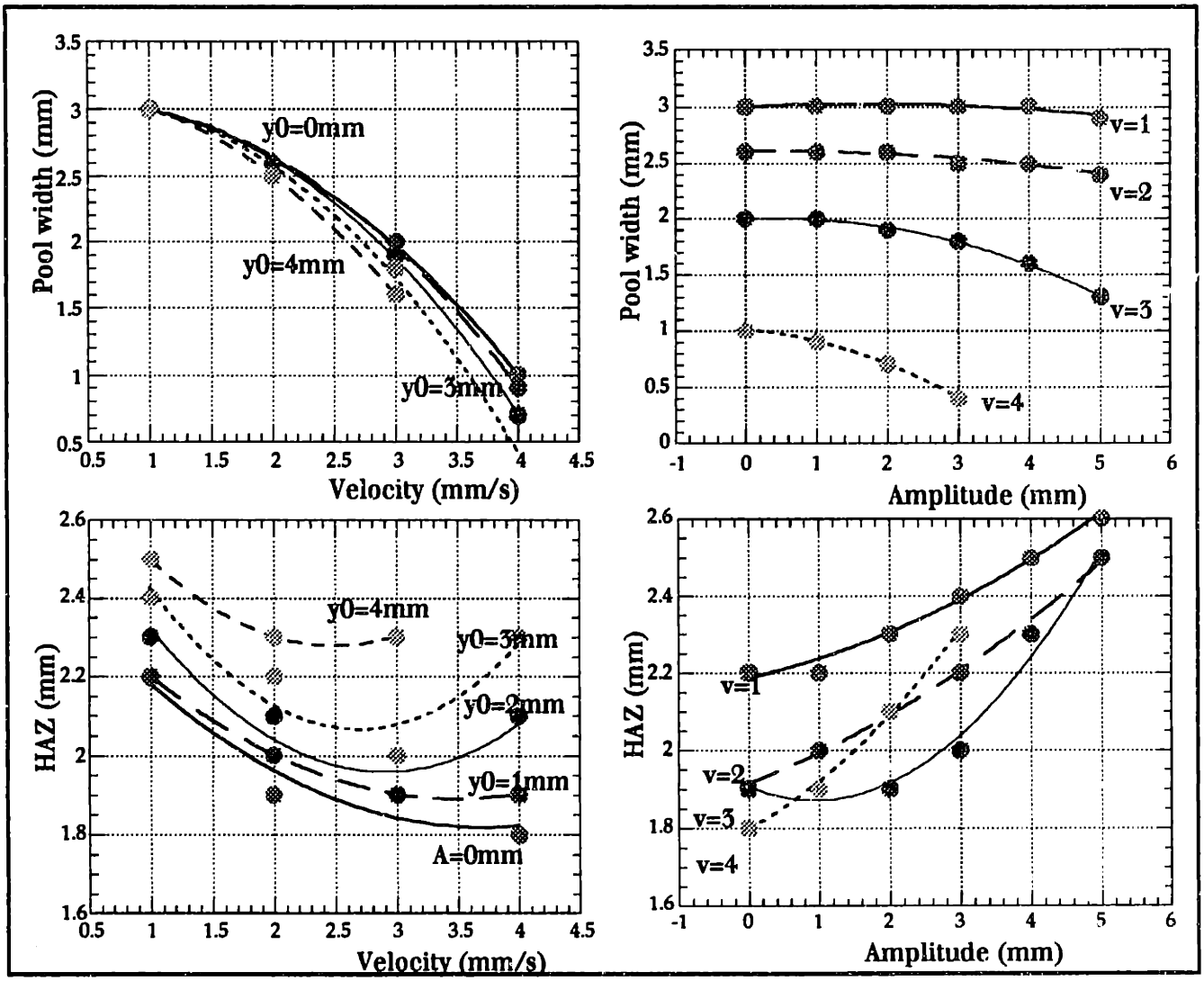


Fig. 3.22: Simulation Input-Output Steady-State relationships (Heat input Q = 2000 J/s)

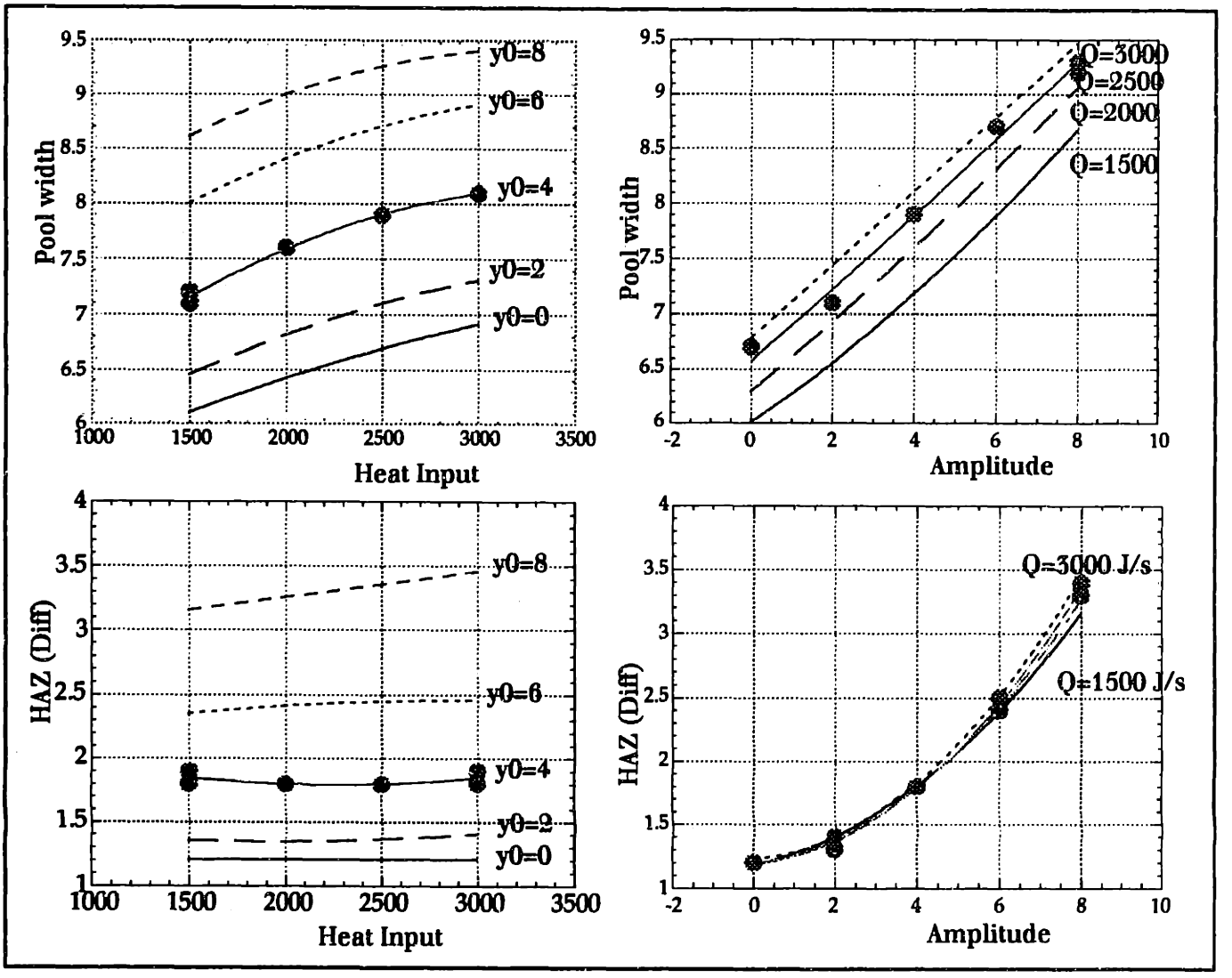


Fig. 3.23: Simulation Input-Output Steady-State relationships (travel speed = 2 mm/s)

Figure 3.23 shows the simulated steady-state input-output relationships for a constant travel speed (v = 2 mm/s). The amplitude vs. HAZ plot also shows that the weave amplitude can be used to control the HAZ width in a decoupled multivariable control system.

Figures 3.24-3.25 shows the Input-Output relationships obtained with the experimental data. The results are very similar to the simulation data. GTAW experiments confirm that the heat input per unit length (Q/v) can be used to control the pool width, while it has very little effect on the HAZ width. The weave amplitude has a significant,

although smaller than that shown by simulations, effect on the HAZ width. Travel speed v can be used to obtain infinite heat per unit length (at v=0 mm/s). Therefore this leads to a basically decoupled closed-loop control system using velocity and weave amplitude as inputs and pool width and HAZ width as outputs. The reachability range obtained with travel speed and weave amplitude as control inputs (Fig. 3.24) is larger than that obtained with heat input (current) and weave amplitude (Fig. 3.25).

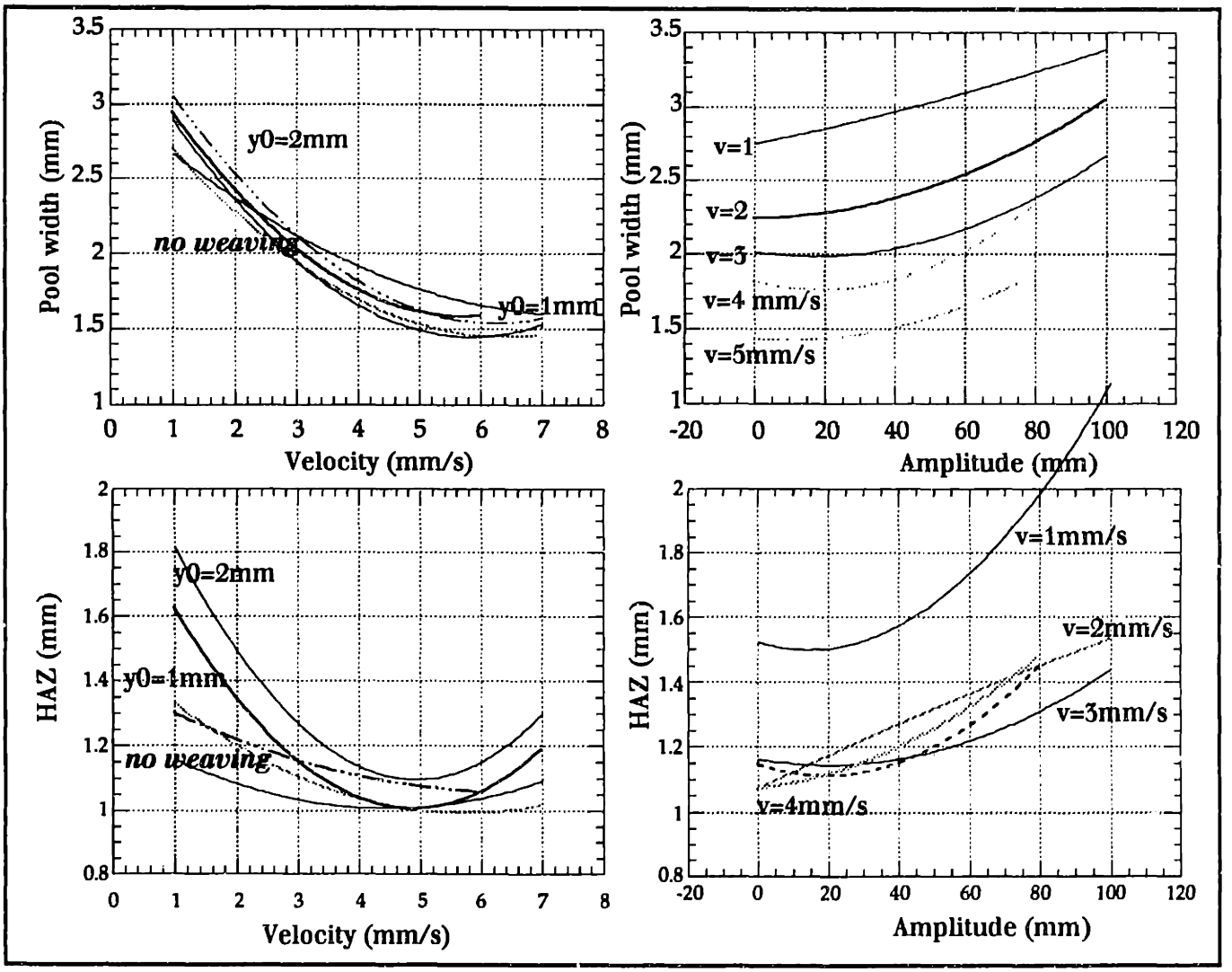


Fig. 3.24: Experimental Input-Output Steady-State relationships (Current I = 100 A)

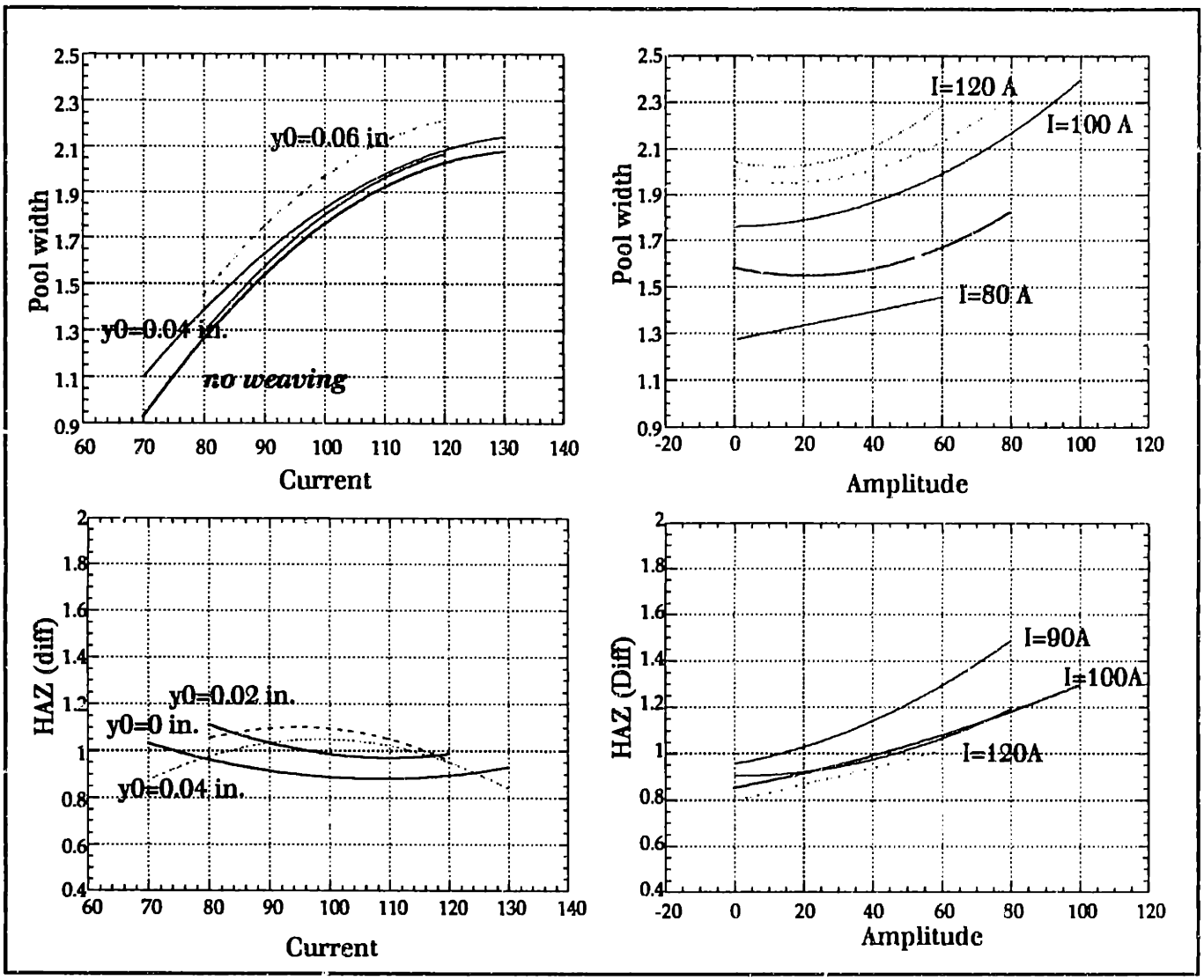


Fig. 3.25: Experimental Input-Output Steady-State relationships (velocity v = 2 mm/s)

There are, however, some differences between simulation and experimental data. First, the simulation model predicts no melting (pool width equal to 0) at velocities higher than 5 mm/s for a heat input of 2000 J/s. The experiments show that melting occurs up to a velocity of 7 mm/s. Another discrepancy is that the model predicts that increasing the weave amplitude slightly decreases the pool width, while the experiments show the opposite behavior. In both cases, however, the effect is very small. Note

also that both simulations and experiments show that the effect of the weave width on the HAZ depends on the magnitude of the forward velocity v.

These discrepancies between the model simulations and the experimental data can be attributed to either model errors and mismatch, or to the large measurement errors in the experimental data. The standard deviation of the pool width measurements is 0.9 mm, or 35 % of the absolute value of the pool width. The standard deviation of the HAZ width measurements is 0.6 mm, or about 30% of the absolute value of HAZ width. These measurements errors are due to the uncertainty associated wit the etching and polishing of the surface, and also to the sensitivity of the GTA welding process itself.

3.7 Conclusions

Multivariable control of arc welding processes requires the ability to exert precise control over heat diffusion in the weldment. Previous research has shown that the outputs of the welding process are highly coupled using heat input and travel speed as control inputs, and that the control range or *reachability* is very limited. Control methods alone cannot overcome this inherent process limitation, and therefore some type of process modification is required.

In this chapter, it was shown that output decoupling can be achieved by high-frequency weaving of the torch. This effectively allows more control over the distribution of the heat in the workpiece. The concept of a variable heat input distribution can be effectively exploited to implement multivariable control of geometric and thermal properties of the weld joint.

The dynamic model presented in chapter 2 was used to simulate the effects of high-frequency torch weaving on arc welding processes. Simulation results show that weave frequency should be at least 3 Hz. (faster than the time constant of heat diffusion in the workpiece). Weave amplitude has a strong effect on HAZ width, while travel speed has a strong effect on pool width. This leads to a basically decoupled two-inputs-two-outputs (TITO) control system.

A high-precision high-frequency weaving servo-mechanism was built and used to verify the model predictions. Electromagnetic arc weaving was not used because it distorts the physics of the arc and the effects of metal and heat input. This high-

precision, high-frequency weaving mechanism is capable of instantly varying the weave amplitude and frequency, and of selecting various weave patterns (sinusoidal, triangular, and square waves). Thus it is possible to control not only the amount but also the precise location of the heat input.

From a physical standpoint, both heat input Q and travel speed v have similar effects on the process because they both regulate the heat-input-per-unit-length (Q/v). This in itself provides a good explanation for the high coupling between the process outputs when Q and v are both used as control inputs. Weaving provides an additional control input that allows the decoupling of both process outputs. Weaving allows a better control over the heat distribution in the workpiece and therefore a better control of cooling rates and temperature gradients inside the weldment.

Chapter 4

Control System Design & Simulations

This chapter presents the Time Delay Control (TDC) theory, and its application to the control of arc welding. A dynamic model, presented in chapter 2, is used to simulate the welding process and to design the control system. The performance of the TDC controller is compared through simulations to the PID controller, and is shown to exhibit better transient response and robustness to disturbances and model uncertainties. The control task is to regulate the pool width and the heat-affected zone width, in an independent fashion, using travel speed and weave amplitude as control inputs. This will be used to confirm the results of chapter 3 that high-frequency weaving leads to decoupling of the process outputs.

Previous research at the MIT Laboratory for Manufacturing and Productivity has separately addressed the problem of thermal control (Doumanidis and Hardt, 1990) and geometric control of easily measurable features, such as width and bead height (Hale and Hardt, 1990a, 1990b, and Song, 1992). In this thesis, a comprehensive approach to in-process control of welding including both geometric features of the bead (such as pool width) and thermal characteristics (such as heat affected zone width) is developed. The geometric features can be related to the basic mechanics of the joint, while the thermal distribution indicates the metallurgical properties of the weldment.

The control goal consists of implementing a robust control method using weaving as a control input (along with total heat input or travel speed). This includes developing the necessary sensors to measure the pool width and the heat-affected zone width. As

shown in chapter 2, the welding process is inherently nonlinear with varying parameters and several uncertainties in the model. Time Delay Control (TDC) is a nonlinear control method developed by Prof. Youcef-Toumi (1990) at MIT. This method has been shown to give excellent results even in the presence of unmodeled dynamics (Youcef-Toumi, 1991). The method uses previous measurements of the state variables and estimates of their derivatives to compensate for the unknown dynamics of the system.

Control of nonlinear systems is an ongoing research topic. In the past, most applications have relied on linearization techniques around an operating point to transform the nonlinear system into a linearized system and using simple linear control methods. The problem with this approach is that a linear controller is valid only in a small neighborhood around the point at which the system was linearized. When the system has a wide operating range, gain scheduling may be used to control the system throughout the operating regions. However, gain scheduling is an ad-hoc method and does not provide any stability guarantees.

In addition, there are many situations when the system cannot be linearized around its operating point because of 'hard nonlinearities' contained in the system. For example, coulomb friction, backlash, and dead-zone cannot be approximated by linear functions. Also there are many instances when the system has no specific operating point, and gain scheduling becomes time-consuming as well as inaccurate. Thus, there is a strong interest in developing new methods to control nonlinear systems. The Time Delay Control (TDC) is one of these promising methods which have shown excellent properties when applied to a number of applications.

In this chapter, Time Delay Control is used to achieve input/output linearization and its robustness is tested in the presence of model uncertainties and sensor noise. The nonlinear dynamic welding model presented in chapter 2 is used to design the TDC controller. The control system is simulated under realistic noise and velocity saturation conditions in order to test its robustness to model uncertainties and sensor noise. TDC is compared with the linear PID method and is shown, through simulations, to provide superior performance and robustness to modeling errors.

4.1 Input/Output Linearization

Several approaches, e.g. variable structure systems (VSS), have been proposed recently for the control of uncertain nonlinear systems. Input/Output linearization is one such method that can be used to control a class of nonlinear systems. I/O Linearization is very straightforward and provides the designer the freedom to select the outputs that must be controlled.

In general, a nonlinear system can be represented by,

$$\dot{x}(t) = f(x(t)) + g(x(t),t)u(t) \tag{1}$$

$$y(t) = h(x(t),t) \tag{2}$$

In the Input/Output Linearization procedure, the output y_i is differentiated with respect to time several times until one of the control inputs u_j appears,

$$\dot{y}(t) = \frac{\partial h(x,t)}{\partial x}\dot{x} = L_f h(x,t) + L_{gu} h(x,t) \tag{3}$$

where L_f h(x,t) is the lie derivative (directional derivative) of h(x,t) along f. If the input u (or at least one element of the input vector) does not appear in equation (3), take the second derivative of the output,

$$\ddot{y}(t) = L_f^2 h(x,t) + L_{gu} L_f h(x,t) \tag{4}$$

In general, we must differentiate r times, where r is the relative degree of the system in order to get access to the input vector.

Definition:

A nonlinear system is said to have a relative degree r if:

$$\begin{cases} L_{gu} L_f^i h(x,t) = 0 & (0 \le i \le r-1) \\ L_{gu} L_f^{r-1} h(x,t) \neq 0 & \end{cases} \tag{5}$$

Following this procedure, the r^{th} derivative of the output will be:

$$\begin{aligned} y^{(r)}(t) &= L_f^r h(x,t) + L_{gu} L_f^{r-1} h(x,t) \\ &= a(x) + b(x,t)u \end{aligned} \tag{6}$$

By choosing an exogenous control function which satisfies:

$$y^{(r)}(t) = a(x) + b(x,t)u = v \tag{7}$$

The resulting system will have *r* stable variables under I/O L control, and (n-r) state variables 'free' (with no control). The designer must check that these 'internal' states remain stable, or the whole control system will be unstable.

Therefore, the control input is chosen as;

$$u = \frac{v}{b(x,t)} - \frac{a(x)}{b(x,t)} \tag{8}$$

The exogenous control can be chosen freely. For tracking a desired trajectory, it is selected as (for *r*=2):

$$v = \ddot{y}_d + k_2(\dot{y}_d - \dot{y}) + k_1(y_d - y) \tag{9}$$

Note that we need to differentiate the output *r* times. This leads to a stable error dynamics,

$$\ddot{e} + k_2\dot{e} + k_1 e = 0 \tag{10}$$

Example:

To illustrate the I/O Linearization procedure, let us design a controller for the following third order system (Youcef-Toumi, 1990):

$$\mathbf{x} = \begin{bmatrix} x_2 - x_1^3 \\ -x_1 \\ x_1^3 - x_3 \end{bmatrix} + \begin{bmatrix} 0 \\ -1 \\ 1 \end{bmatrix} u$$

$$y = x_1$$

The first derivative of the output does not give access to the input u, but the second derivative does (relative degree 2):

$$\ddot{y} = a(x) + b(x)u$$

where:

$$a(x) = L_f^2 h(x,t) = x_1^2 + 3x_2^3 - x_3$$
$$b(x) = L_{gu} L_f h(x,t) = 1 + 3x_2^2$$

The gains of the exogenous control u are selected by using the error dynamics (Eq.10). If we choose a damping coefficient $\zeta = 0.707$ and a natural frequency $\omega_n = 5$ Hz., we obtain:

$$k_2 = 2\zeta\omega_n = 7.07$$

$$k_1 = \omega_n^2 = 25$$

If the full state vector is available for feedback, then Eq. 8 can be used to exactly linearize the system. The output will then follow the desired trajectory, and the internal states will be stable (the zero dynamics is asymptotically stable). Figure 4.1 shows the simulation of this control system with full state feedback.

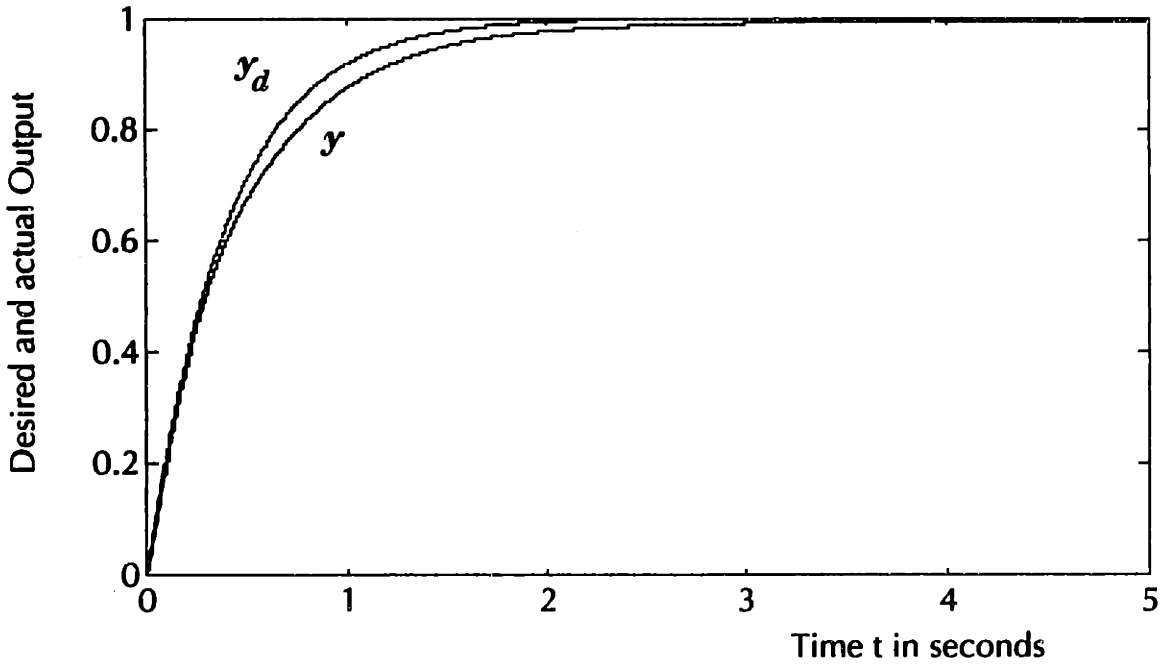


Fig. 4.1: Time response of output & desired trajectory with I/O Linearization

The Input-Output Linearization method assumes perfect knowledge of the system nonlinearities in order to invert them, and requires full state feedback. This method is not robust to model errors. Fig. 4.2 shows the same simulation as in Fig. 4.1 except that the state equation for x_1 was changed from $x_3 - x_2^3$ to $x_3 - \sin(x_2^2)$. The system remains stable but its performance is greatly degraded. Fig. 4.3 shows that the 'internal states' remain stable as well.

However, if access to the full state vector is not available, then the functions a(x) and b(x) must be approximated with some parameter estimation schemes. In this case, the system's stability is no longer guaranteed, and a more robust control law should be used.

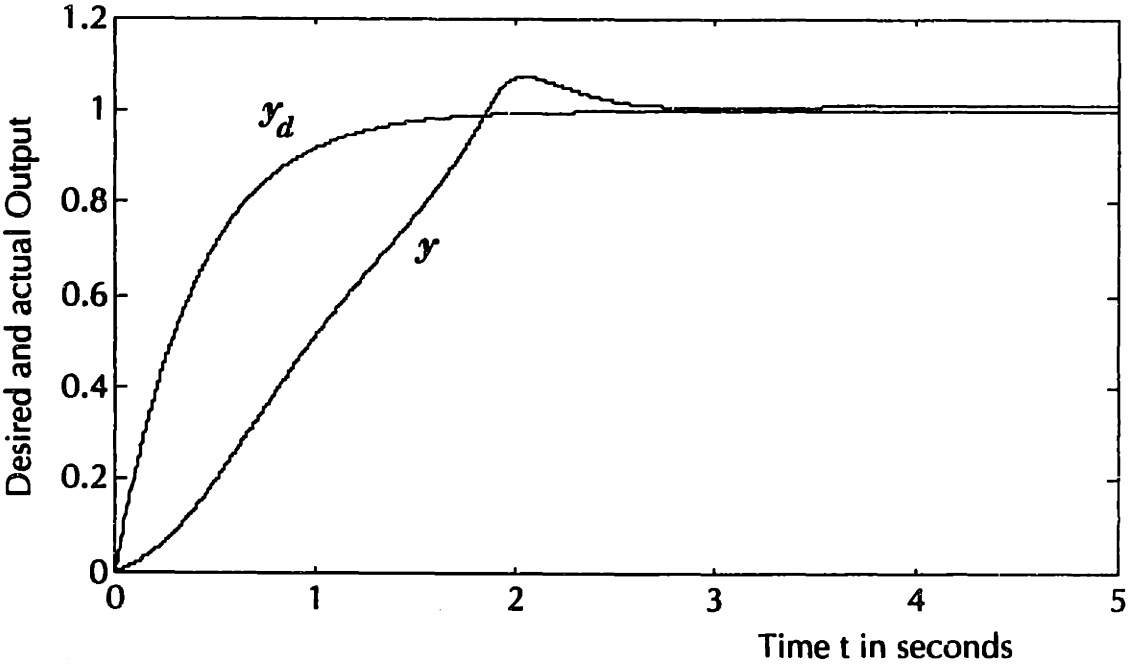


Fig. 4.2: Time response of output & desired trajectory with I/O Linearization (with model errors)

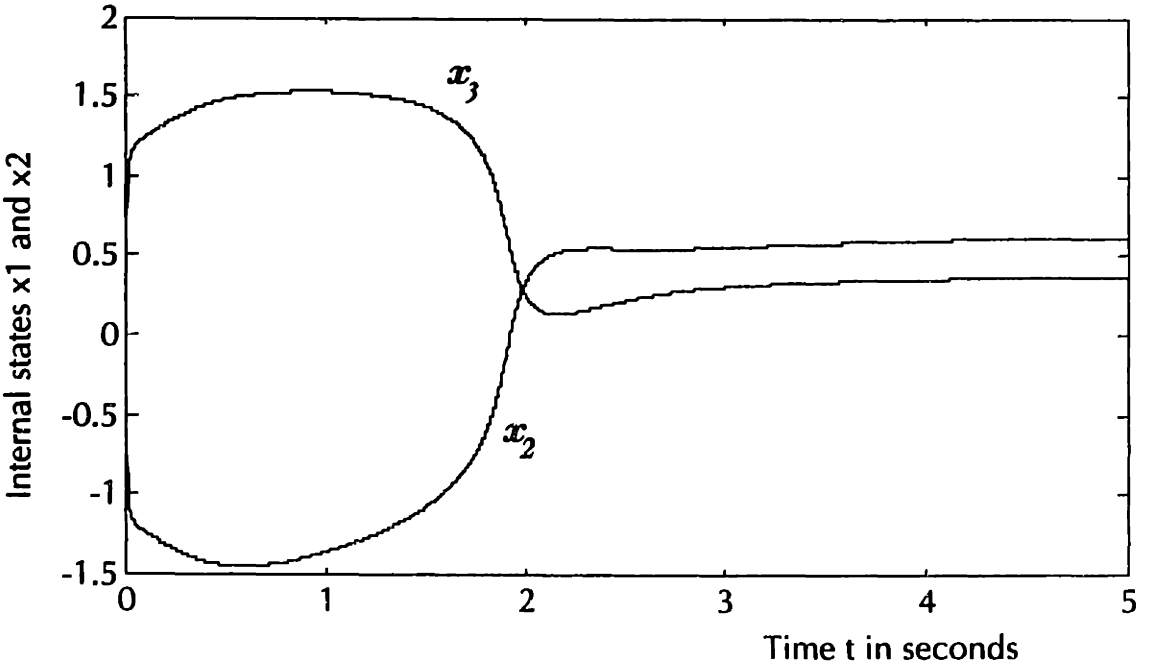


Fig. 4.3: Time response of 'internal states' with I/O Linearization (with model errors)

4.2 Time Delay Control (TDC)

The control of systems with uncertain dynamics and unpredictable disturbances has raised some challenging problems. Time Delay Control (TDC) has been suggested as an alternative control scheme which does not require an explicit plant model. Time Delay Control uses adaptation with past observation of the outputs to estimate the plant dynamics and cancel the effects of any disturbances. Therefore, TDC is well suited for a large class of control problems the dynamics of the system are difficult to predict and where the parameters of the model are highly variable. Let us briefly review the theory of Time Delay Control.

Consider the class of nonlinear systems described by the following differential equations:

$$\dot{\mathbf{x}}(t) = \mathbf{F}(\mathbf{x},t) + \mathbf{G}(\mathbf{x},\mathbf{u},t) + \mathbf{D}(t) \tag{11}$$

where $x(t) \in \Re^n$, and $u(t) \in \Re^r$, are the system state vector and input vector respectively. **F**(x,t), **G**(x,u,t), and **D**(t) represent the known, unknown and disturbance components of the system dynamics respectively.

Equation 11 can rewritten as:

$$\dot{\mathbf{x}}(t) = \mathbf{F}(\mathbf{x},t) + \mathbf{H}(\mathbf{x},\mathbf{u},t) + \mathbf{B}\mathbf{u} + \mathbf{D}(t) \tag{12}$$

where **B** is a known matrix of rank r to be selected by the designer and the term **H**(x,u,t) is defined as:

$$\mathbf{H}(\mathbf{x},\mathbf{u},t) = \mathbf{G}(\mathbf{x},\mathbf{u},t) - \mathbf{B}\mathbf{u}$$

A linear time-invariant model can be used as a reference model to generate the desired trajectory,

$$\dot{\mathbf{x}}_m(t) = \mathbf{A}_m\mathbf{x}_m(t) + \mathbf{B}_m\mathbf{r}(t) \tag{13}$$

The error between the plant state vector and the desired trajectory is:

$$\mathbf{e}(t) = \mathbf{x}_m(t) - \mathbf{x}(t) \tag{14}$$

The error dynamics is therefore given by:

$$\dot{\mathbf{e}}(t) = \mathbf{A}_m\mathbf{x}_m(t) + \mathbf{B}_m\mathbf{r}(t) - [\mathbf{F}(\mathbf{x},t) + \mathbf{H}(\mathbf{x},\mathbf{u},t) + \mathbf{B}\mathbf{u} + \mathbf{D}(t)]$$

By adding and subtracting the term $\mathbf{A}_m\mathbf{x}_m$, this can also be written as;

$$\dot{\mathbf{e}}(t) = \mathbf{A}_m\mathbf{e}(t) + [\mathbf{A}_m\mathbf{x}(t) + \mathbf{B}_m\mathbf{r}(t) - \mathbf{F}(\mathbf{x},t) - \Psi(\mathbf{x},\mathbf{u},t) - \mathbf{B}\mathbf{u}] \tag{15}$$

where the term $\Psi(\mathbf{x},\mathbf{u},t)$ represents the unknown dynamics $[\mathbf{H}(\mathbf{x},\mathbf{u},t) + \mathbf{D}(t)]$. The control $\mathbf{u}$ can be chosen such that the term between brackets in Eq.15 is zero if the matrix $\mathbf{B}$ is invertible, otherwise $\mathbf{u}$ is determined on a least squares approach using the pseudo-inverse of $\mathbf{B}$. In this case, the error dynamics will be as fast as that of the reference model. In general, this is not sufficient and it is desirable to have the error vanish much quicker than the reference model dynamics. This can be achieved by introducing a new control input $\mathbf{v}$;

$$\mathbf{v} = \mathbf{u} + \mathbf{K}\mathbf{e}$$

where $\mathbf{K}$ is a feedback gain matrix to be chosen. The error dynamics now becomes:

$$\dot{\mathbf{e}} = [\mathbf{A}_m + \mathbf{B}\mathbf{K}]\mathbf{e} + \mathbf{p} \tag{16}$$

where $\mathbf{p}$ is:

$$p = \mathbf{A}_m\mathbf{x}(t) + \mathbf{B}_m\mathbf{r}(t) - \mathbf{F}(\mathbf{x},t) - \Psi(\mathbf{x},\mathbf{u},t) - \mathbf{B}\mathbf{v} \tag{17}$$

The control law is chosen as:

$$\mathbf{v} = \mathbf{B}^{+}\left[\mathbf{A}_m\mathbf{x}(t) + \mathbf{B}_m\mathbf{r}(t) - \mathbf{F}(\mathbf{x},t) - \hat{\Psi}(\mathbf{x},\mathbf{u},t)\right] \tag{18}$$

where $\mathbf{B}^{+}$ is the pseudo-inverse of $\mathbf{B}$. One main requirement for the successful implementation is to have a good estimate of the unknown dynamics $\Psi(\mathbf{x},\mathbf{u},t)$. Several approaches for calculating this estimate have been proposed in the literature (Youcef-

Toumi, 1991), but one simple method that will be used here is to assume that, for a sufficiently small delay time L, the vector functions do not change significantly. Thus, the unknown dynamics can be approximated by:

$$\begin{aligned}\mathbf{H}(\mathbf{x},\mathbf{u},t)+\mathbf{D}(t) &\cong \mathbf{H}(\mathbf{x},\mathbf{u},t-L)+\mathbf{D}(t-L)\\ &\cong \dot{\mathbf{x}}(t-L)-\mathbf{F}(\mathbf{x},t-L)-\mathbf{B}\mathbf{u}(t-L)\end{aligned}$$

4.3 I/O Linearization with Time Delay Control:

The basic idea behind time delay control is to use past information of the state variables and their rates of change to predict present dynamics without using an accurate model. The assumption is that the time delay is sufficiently small to assume that these quantities have not changed significantly during that time delay interval.

This same idea can be used to achieve input/output linearization without full state feedback. In equation 7, the r^{th} derivative of the output (r being the relative degree of the system) was written as:

$$y^{(r)}(t) = a(x) + b(x,t)u = v$$

If b(x,t) is known exactly, then the control law can be chosen as:

$$u = b^{-1}(x,t)[-a(x) + v]$$

If b(x,t) is not known, it can be approximated by a constant b_r and a(x) can be approximated by $y^{(r)}$-$b_r u$, thus:

$$u(t) = b_r^{-1}\left[-y^{(r)}(t-L) + b_r u(t-L) + v\right] \qquad (19)$$

Note that in the above control law, we do not use any explicit model. Current values for unknown quantities are approximated by delayed information (assuming that the time delay is very small). The exogenous control u is chosen as in Eq. 9 for tracking.

Youcef-Toumi and Wu (1991) show that in order to guarantee stability of the overall control system, b_r must satisfy the following condition:

$$\left\|\mathbf{B}(x,t)\mathbf{B}_r^{-1}-\mathbf{I}\right\| \le \beta \le 1 \qquad \text{for some t} > 0$$

Additionally, it is shown that if the control action of Eq.19 is filtered using a low pass filter of the form:

$$u_{(k)} = \frac{\alpha}{1+\alpha} u'_{(k)} + \frac{1}{1+\alpha} u_{(k-1)} \tag{20}$$

where:

$$u'_{(k)} = b_r^{-1}\left[-y^{(r)}{}_{(k-1)} + b_r u_{(k-1)} + v_{(k)}\right]$$

then the bound on b_r becomes;

$$\left\|\frac{\alpha}{1+\alpha}\mathbf{B}(x,t)\mathbf{B}_r^{-1}-\mathbf{I}\right\| \le 1 \qquad \text{for some t} > 0 \tag{21}$$

Figure 4.4 shows a block diagram of the TDC controller with Input-Output Linearization that was used to implement the above control law (Eq. 20). Note that in this case b_r is a constant. One way to make this system even more adaptive to model changes is to use on-line parameter estimation techniques to estimate br.

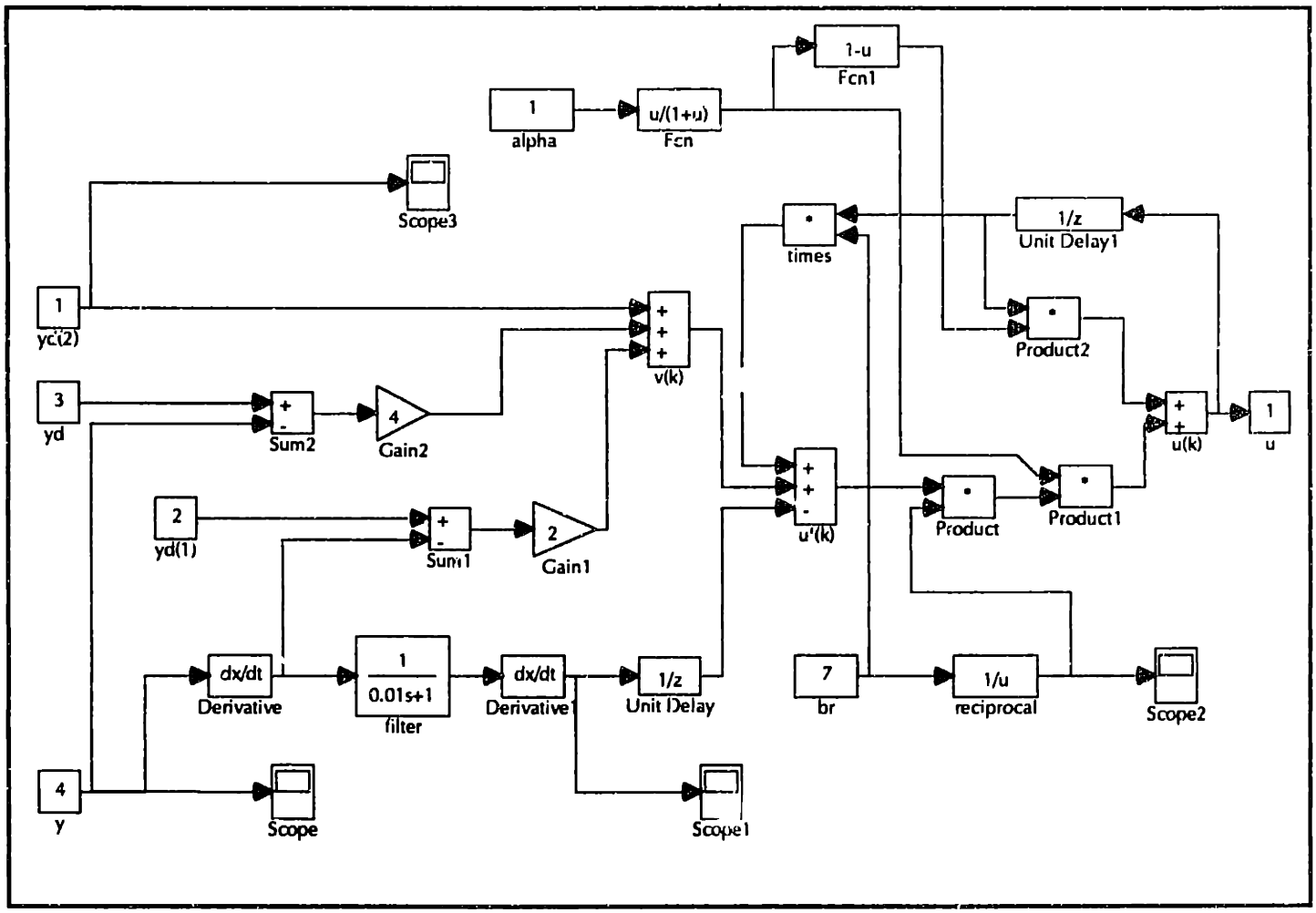


Fig. 4.4: SIMULAB Block Diagram for TDC with I/O Linearization

Figure 4.4 shows the Simulab block diagram for the TDC controller used in these simulations. The controller output u is calculated using equation 20, with filtering of the control action to increase the system's robustness to model errors and disturbances. After the derivative of the input y is calculated, a filter is used to take out the high frequency noise before calculating the second derivative of y. This is one of the main problem associated with the TDC, which makes it sensitive to to excessive noise in the measurements.

4.3.1 Simulations

In the following simulations, we use Time Delay Control to achieve input/output linearization of the same model presented in the previous section using the same desired trajectory. Fig. 4.5 shows the time response of the output and the desired output using $b_r = 4$, and $\alpha = 1000$ (no filtering of the control action). Notice that the response is fast and the output follows closely the desired trajectory even though no specific model was used in the derivation of the Time Delay Control law.

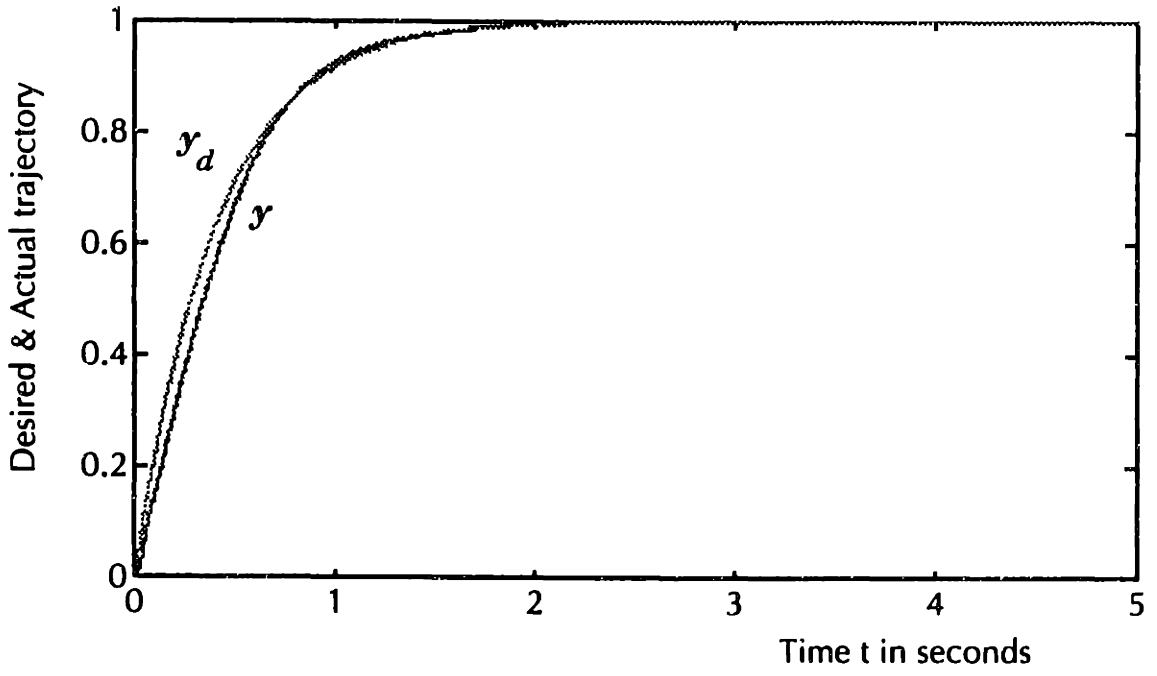


Fig. 4.5: Dynamic response of TDC controller ($b_r = 4$, and $\alpha = 1000$)

4.3.2 Robustness to Modeling errors

In order to test the robustness of the Time Delay Controller to model errors and uncertainties, the same controller as before was simulated with a different plant model defined by the following differential equations:

$$\begin{cases} \dot{x}_1 = x_3 + \sin x_2^2 \\ \dot{x}_1 = -x_2 + u \\ \dot{x}_3 = 2x_1 - x_3 + u \end{cases}$$

Figure 4.6 shows the simulation of the TDC with Input-Output linearization control system with model errors (same as those shown in Fig. 4.2). The Time Delay Controller is much more robust to model uncertainties because the control algorithm does not use an explicit model. This is one of the main advantages of TDC control, in addition to the ease of application to nonlinear systems and the fact that it does not need full state feedback. Note that TDC (Fig. 4.6) uses only one sensor measurement (that of the output $y=x_1$) while I/O Linearization without time delay control (Fig. 4.2) requires full state feedback (three sensors or observers).

Figure 4.7 shows that TDC requires high-frequency high frequency control action required to make the output follow the desired trajectory. One way to reduce this high frequency chattering is through filtering of the control input u as shown in Eq. 20 by using a smaller value of α. Fig. 4.8 shows the same controller as Fig. 4.6, except that filtering of the control input has been used (a = 1). As shown in Fig. 4.9, this filtering eliminates high frequency chattering of the control input u. Youcef-Toumi and Wu (1991) showed that using this filter also improves the robustness of the TDC Controller.

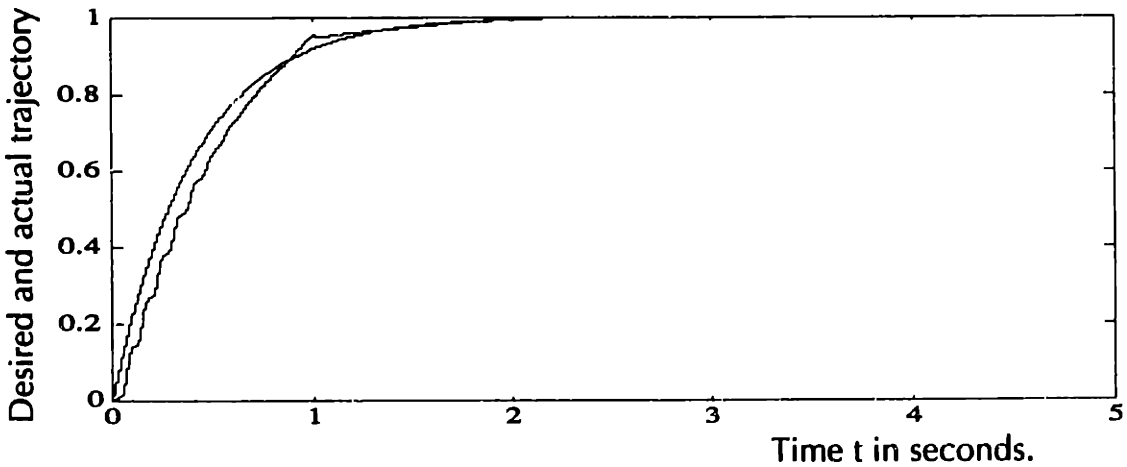


Fig. 4.6: Desired & Actual trajectory of TDC with I/O Linearization in the presence of model errors (α=1000, b_r=4)

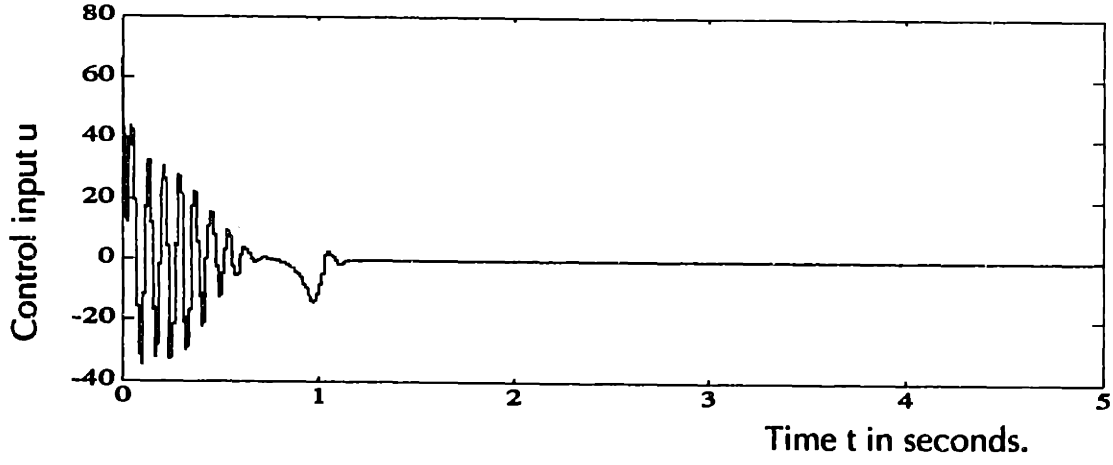


Fig. 4.7: Control input of TDC with I/O Linearization in the presence of model errors (α=1000, b_r=4)

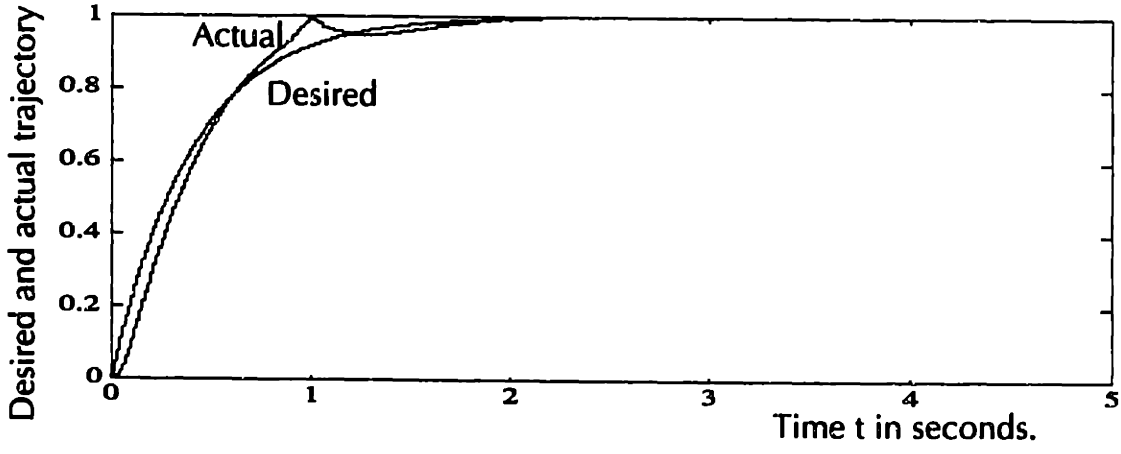


Fig. 4.8: Desired & Actual trajectory of TDC with I/O Linearization in the presence of model errors (α=1, b_r=4)

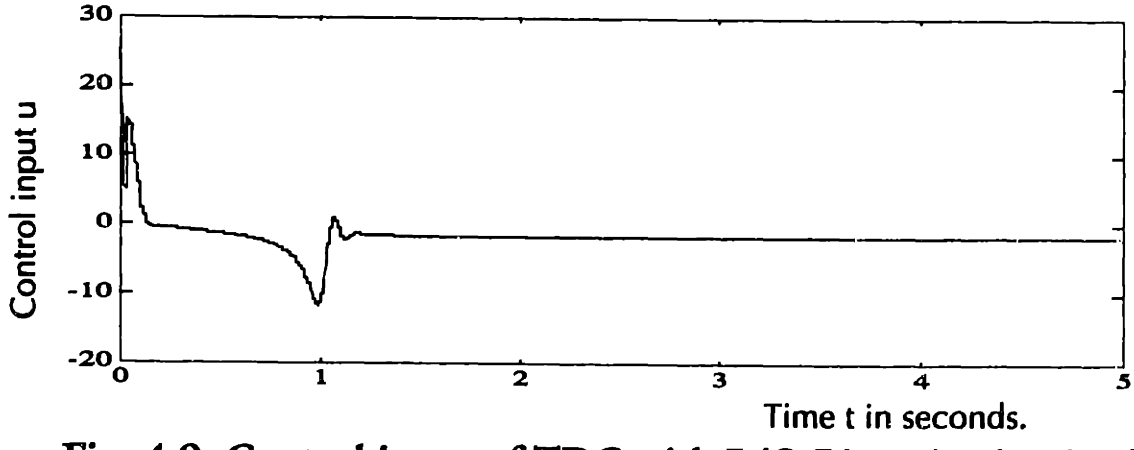


Fig. 4.9: Control input of TDC with I/O Linearization in the presence of model errors (α=1, b_r=4)

4.4 Closed-loop Control Simulation of Arc Welding

In this section, closed-loop control systems are simulated and designed for multivariable control of arc welding process outputs. One way to model the welding process is through empirically-developed transfer functions. Previous research (Doumanidis, 1990, and Hale, 1990b) showed that the dynamic response of arc welding may be approximated by second order linear transfer functions with constant damping ratio and velocity-dependent natural frequency. Transfer functions of the following structure are obtained by fitting impulse response data:

$$\begin{Bmatrix} y_1(s) \\ y_2(s) \end{Bmatrix} = \begin{bmatrix} \dfrac{b_{11}s+b_{12}}{s^2+a_{11}s+a_{12}} & \dfrac{b_{13}s+b_{14}}{s^2+a_{11}s+a_{12}} \\ \dfrac{b_{21}s+b_{22}}{s^2+a_{21}s+a_{22}} & \dfrac{b_{23}s+b_{24}}{s^2+a_{21}s+a_{22}} \end{bmatrix} \begin{Bmatrix} u_1(s) \\ u_2(s) \end{Bmatrix}$$

The problem with this approach is that the linearized model is valid only in a very small neighborhood of the operating point around which the model was obtained. In order to cover the entire operating range of the welding process. Linearization of the process must be repeated many times, and gain scheduling must be used. This difficulty is further aggravated by the fact that there are many exogenous process parameters (such as arc length, electrode contamination, turbulent fluid flow in the molten pool, etc..) which are not under direct control and can change the process dynamical properties significantly.

One solution, proposed by Suzuki (1990), Doumanidis (1990), and Song (1992), is to use on-line parameter estimation to vary the model according to the operating conditions. This technique is then used in conjunction with a model-based adaptive controller to accomplish feedback control (Fig. 4.10). The problem with this approach is that the error signal drives both the parameter estimation scheme and the adaptive controller. Therefore, the system can become unstable.

Another method of modeling arc welding processes is to use finite element or finite difference models. This approach was tried in chapter 2 and was found to be too slow for modeling the effects of arc weaving. Therefore a simple dynamic model was developed in chapter 2, and will be used here to simulate the welding process. The

model was implemented in a Matlab function subroutine and is therefore well-suited for dynamic simulation and control system design. The main advantages of this model are its simplicity, accuracy, and computational ease of implementation in comparison with large and complex finite-element or finite difference models.

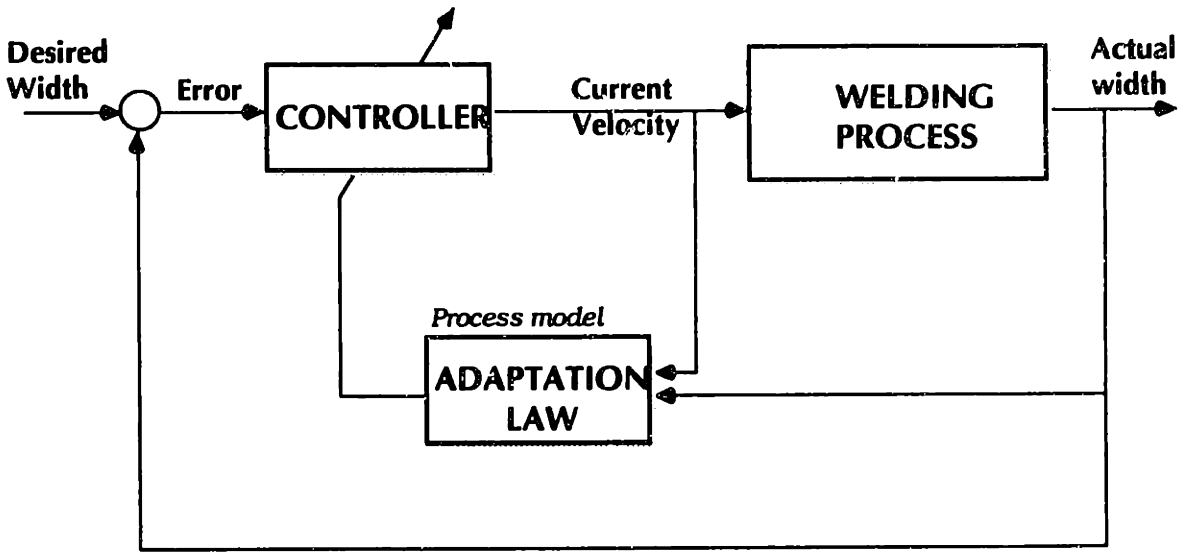


Fig. 4.10: Adaptive Control Law Block Diagram

The control approach adopted here is to use a nonlinear control method that is model-independent such as Time Delay Control (TDC). This method uses past and current information about the system inputs and outputs to estimate the plant dynamics and is very robust to model uncertainties. Assuming that the time delay is small compared with the time constant of the plant, past measurements of the inputs and outputs can be used to predict the current dynamic response of the system. The stability of TDC controllers is assured if the time delay is small enough.

4.5 PID Control Simulations:

In this section, the dynamic model developed in chapter 2 is used to simulate the response of the welding process under PID control. PID control is a simple linear feedback control method that calculates an input as a function of the output error, its derivative, and its integral over time:

$$U = K_p e + K_i \int e dt + K_d \frac{de}{dt}$$

where K_p is the proportional gain, K_i is the integral term gain, K_d is the derivative term gain, e is the error between the desired and the actual output, and U is the control input.

The inputs available for GTA welding are: current, travel speed, and weave width. Weave frequency was kept constant at 3 Hz. Arc current in GTAW is almost a liner function of the heat input Q. The voltage is kept constant using the Automatic voltage controller (AVC). The heat input Q and the reciprocal of travel speed ($1/v$) have a similar effect because they both determine the amount of heat input per unit length (Q/v). Increasing the heat input Q or decreasing the travel speed v have similar effect on the process. However, the heat input Q usually saturates quite quickly, and there is much more freedom in selecting the travel speed. In fact, using a velocity of zero, we can obtain an infinite heat per unit length (Q/v). Very small travel speeds can simulate very large heat inputs which would otherwise be unattainable with the available power supplies. Therefore, using travel speed as the second input, with weave amplitude, provides more control authority over the process.

The goal in these control simulations is to control the temperature at a specified width using travel speed (v) and weave amplitude (w). So if the desired pool width for example is 4 mm, then the temperature at that width is controlled to be the melting temperature (1800 °K for mild steel) and similarly for HAZ (1000 °K for mild steel).

Figure 4.11 shows the response of the model under proportional control. The response to a step increase in desired temperature (from 1000 to 1500 °K) is second order and very oscillatory (small damping ratio) and is quite different from the response to a step decrease in desired temperature (from 1500 to 1000 °K). The input, travel speed, is plotted in Fig. 4.12. Figure 4.11 illustrates the nonlinear nature of the welding process. The response of the process is different depending on the operating conditions and the inputs. This means that a different PID controller must be designed for each condition.

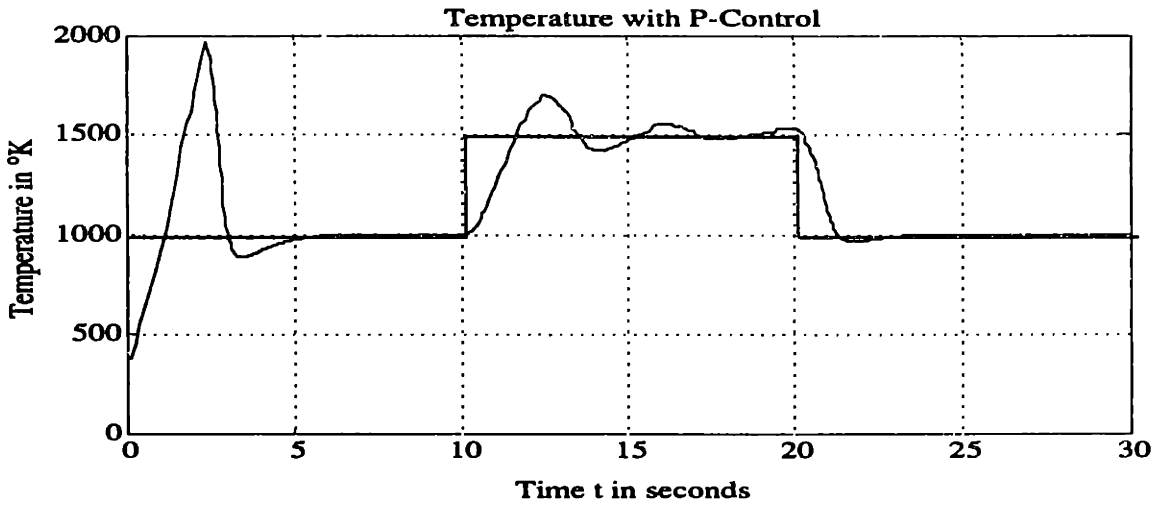


Fig. 4.11: Dynamic response of temperature under proportional control

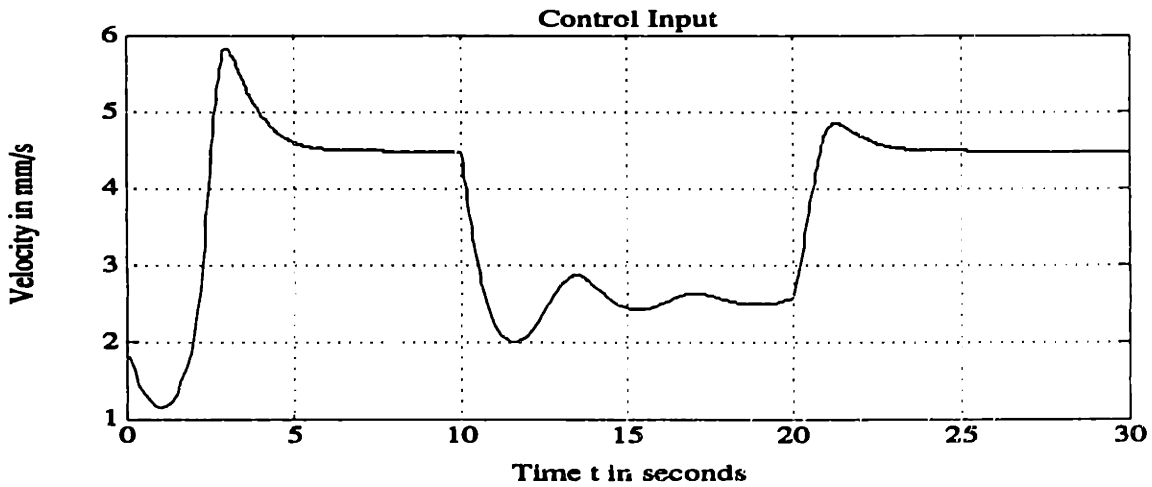


Fig. 4.12: Input travel speed as a function of time (proportional control)

Figures 4.13 and 4.14 show the response of a proportional+derivative (PD) controller in the presence of white noise. The proportional controller does not guarantee zero steady-state error and has poor transient response, and therefore we must use derivative and integral action to improve the properties of the control system. As a result of noise, the control input, travel speed v, exhibits high-frequency chattering, which is not desirable, therefore in the simulation, saturation bounds on velocity and acceleration

were included to model real hardware limitations. The PD Controller has improved transient response but has a non-zero steady-state error as shown in Fig. 4.13. Therefore, integral control action must be added to obtain zero steady state error as shown in Fig. 4.15 and 4.16.

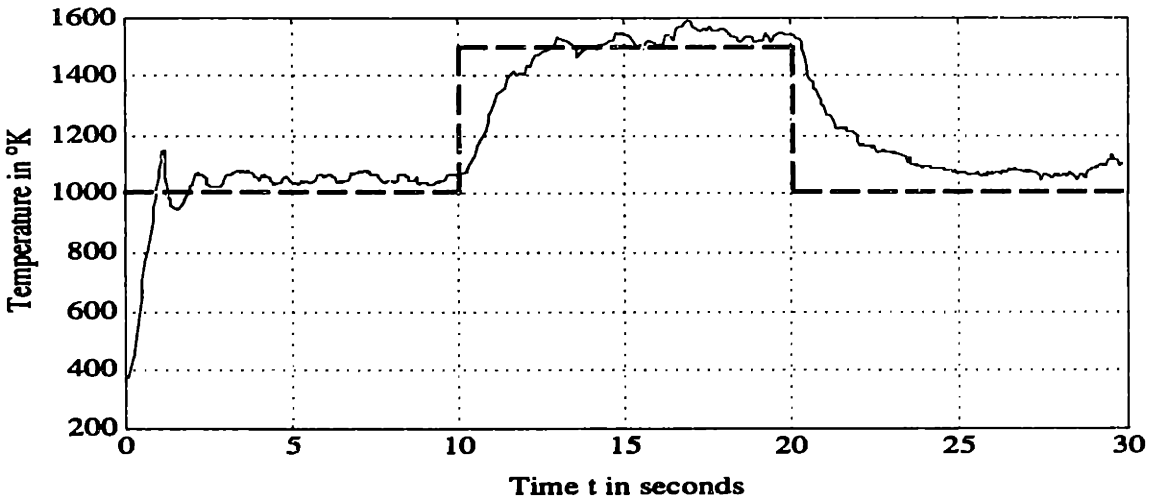


Fig. 4.13: Dynamic response of temperature under PD control in the presence of white noise (standard deviation = 30 °K)

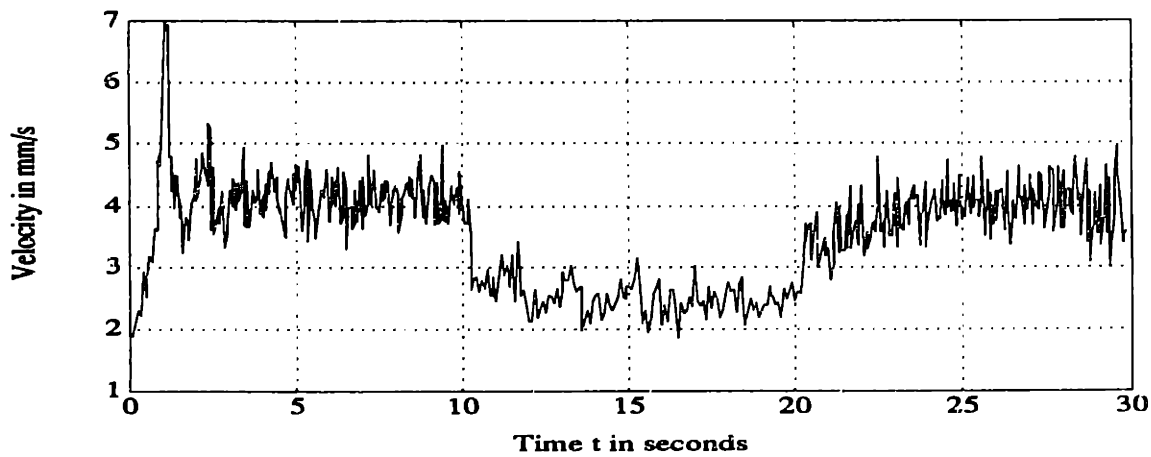


Fig. 4.14: Control input with PD control in the presence of white noise (standard deviation = 30 °K)

Figures 4.15 and 4.16 show the simulation plots of a PID controller. As expected, the error converges to zero, but the transient behavior is worse than with PD control. The gains used in Fig. 4.15 and 4.16 are: $K_p = K_d = K_i = 5x10^{-4}$.

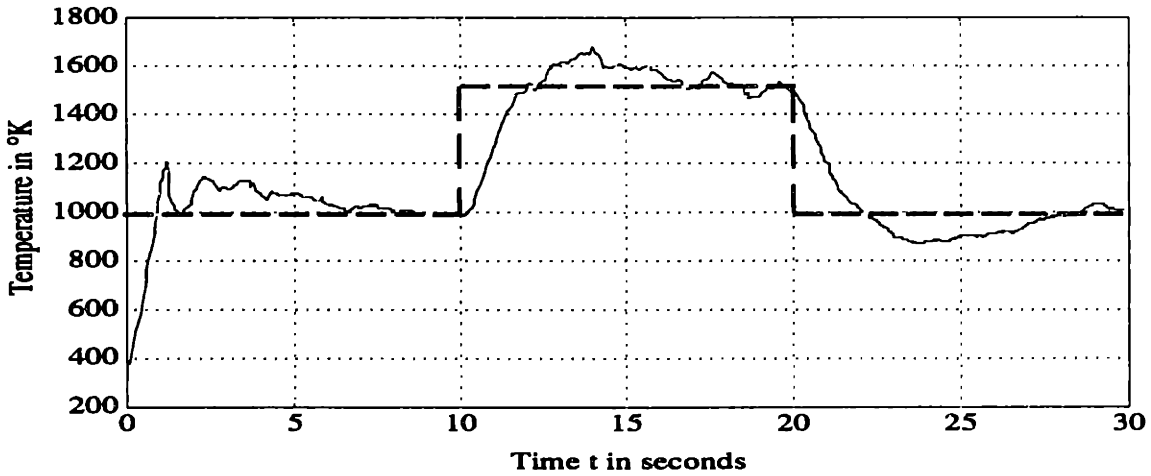


Fig. 4.15: Dynamic response of temperature under PID control in the presence of white noise (standard deviation = 30)

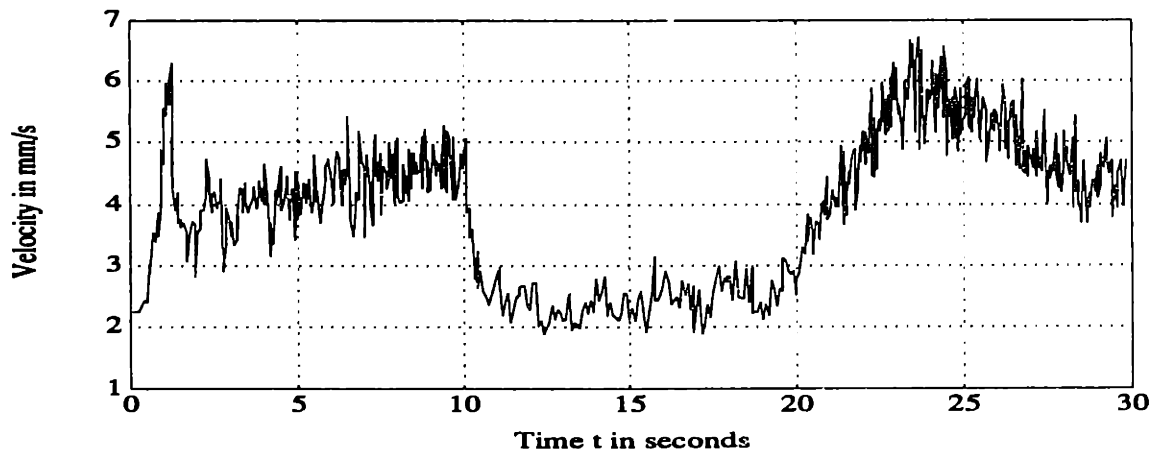


Fig. 4.16: Control input of temperature under PID control in the presence of white noise (standard deviation = 30)

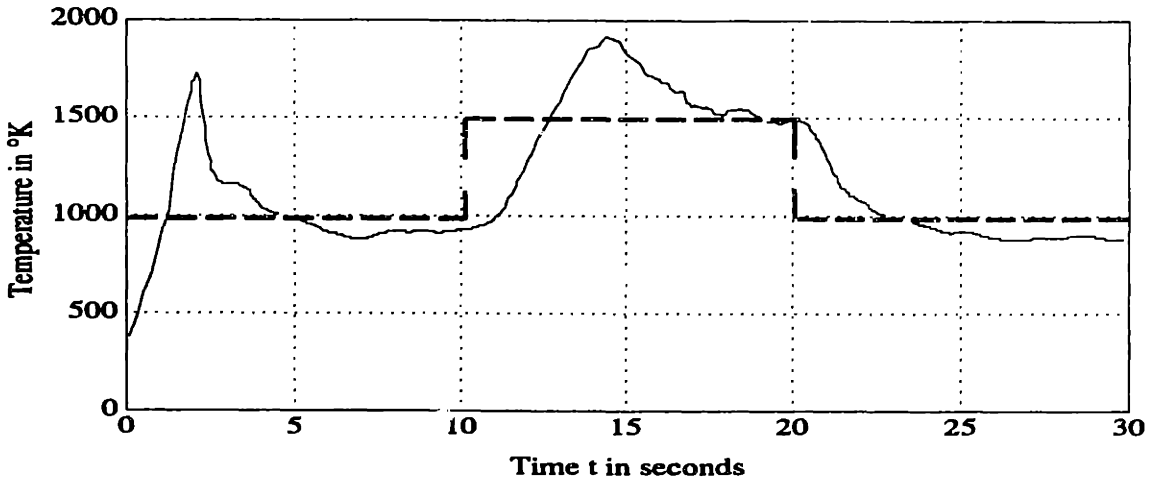


Fig. 4.17: Dynamic response of temperature under PID control in the presence of white noise and acceleration saturation.

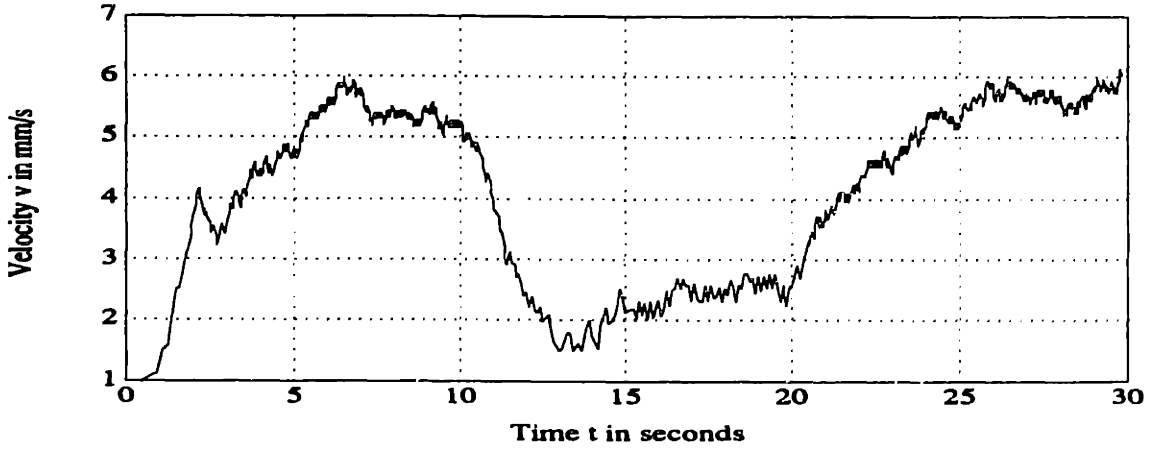


Fig. 4.18: Control input under PID control in the presence of white noise and acceleration saturation.

Figures 4.17 and 4.18 show the same simulation of 4.15 with the addition of velocity and acceleration saturation. The upper limit on travel speed was fixed at 7 mm/s and that of acceleration was set at 5 mm/s^2. The addition of saturation reduces the bandwidth of the control system and slows down its response as shown in Fig. 4.17. The gains used in Fig. 4.17 are the same as those used in Fig. 4.15.

4.6 TDC Control Simulations:

This section presents simulation results of the TDC controller. The first choice when designing a TDC control system is to determine the order of the system (which should be equal to the relative order of the plant (see section 4.2)). In the case of the welding process, it is known that the response is second order, so the controller should be second order. If a first order TDC controller is used instead, large oscillations and instability are obtained as shown in Figures 4.19 and 4.20.

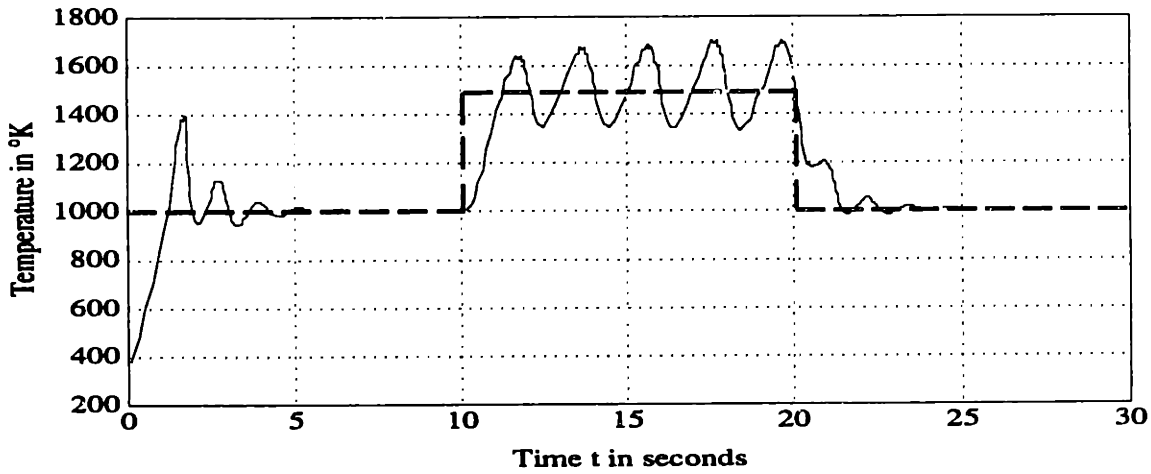


Fig. 4.19: Dynamic response of first order TDC Controller.

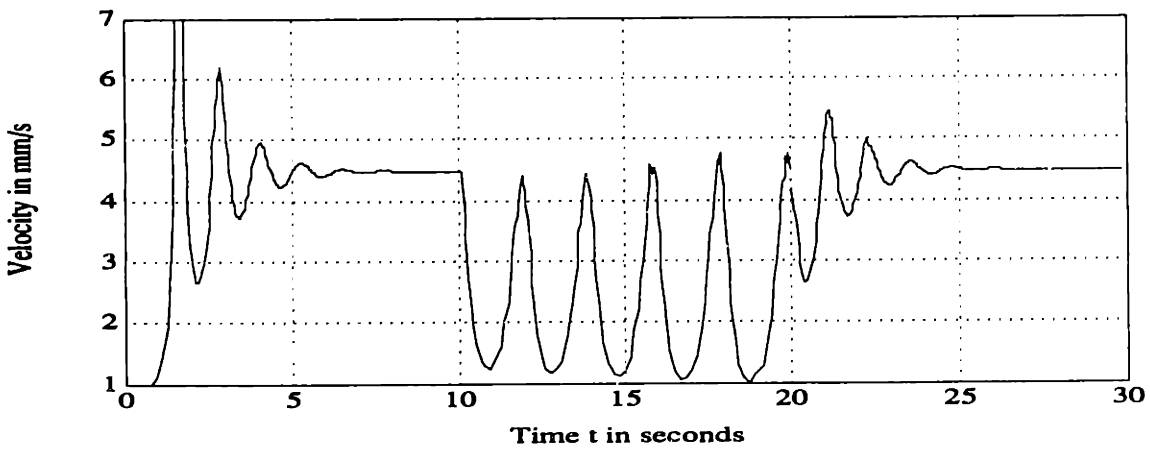


Fig. 4.20: Control input with first order TDC Controller.

Figures 4.21 and 4.22 show that a first order TDC controller oscillatory response in the presence of sensor noise and input saturation (travel speed cannot exceed 7 mm/s and acceleration cannot exceed 5 mm/s^2).

Figures 4.23-4.28 show that a second order TDC controller gives a much better response as expected because the system is second order. The Matlab subroutine used for simulating the TDC controller is listed in Appendix 2.

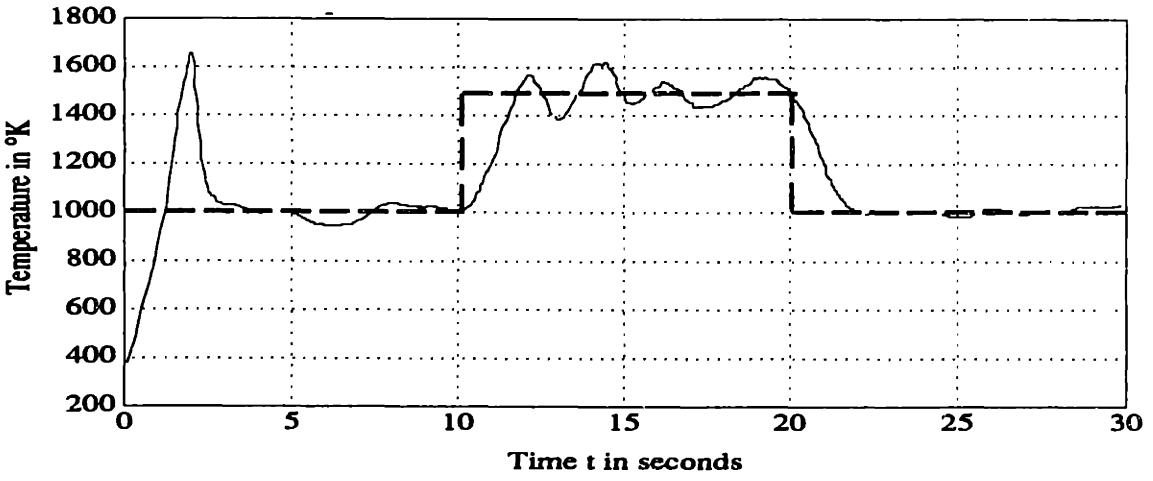


Fig. 4.21: Dynamic Simulation of temperature with first order Time Delay Control

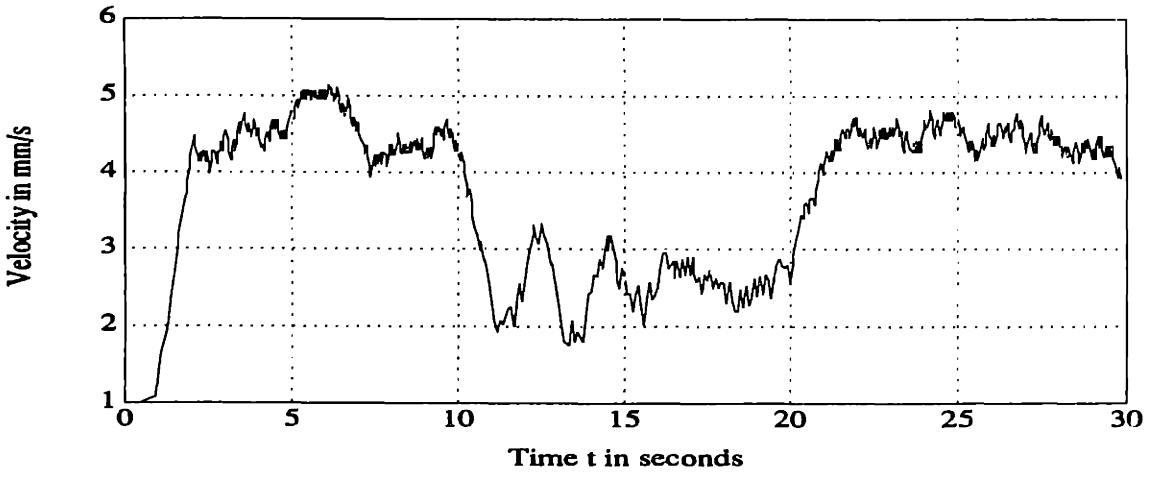


Fig. 4.22: Control input used with first order TDC Controller

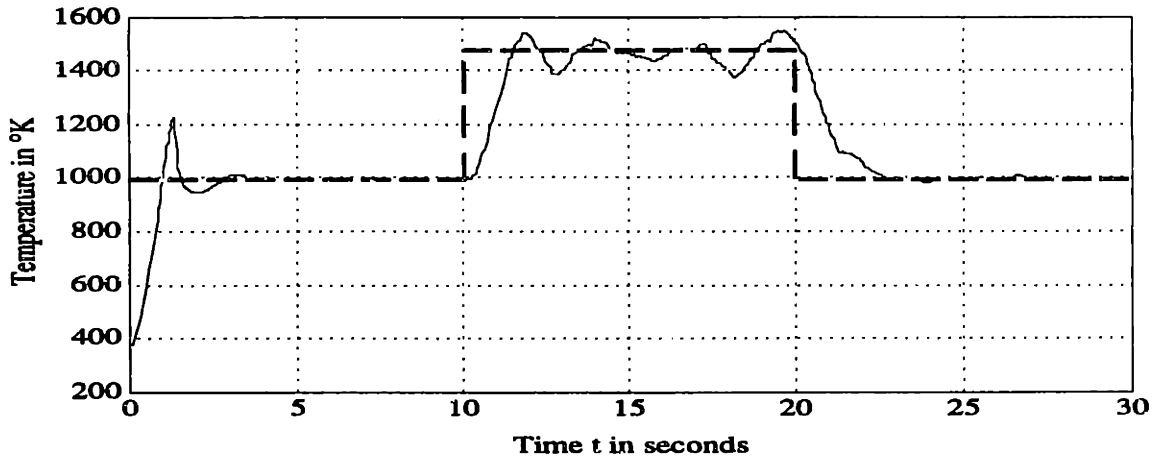


Fig. 4.23: Dynamic response with second order TDC Controller

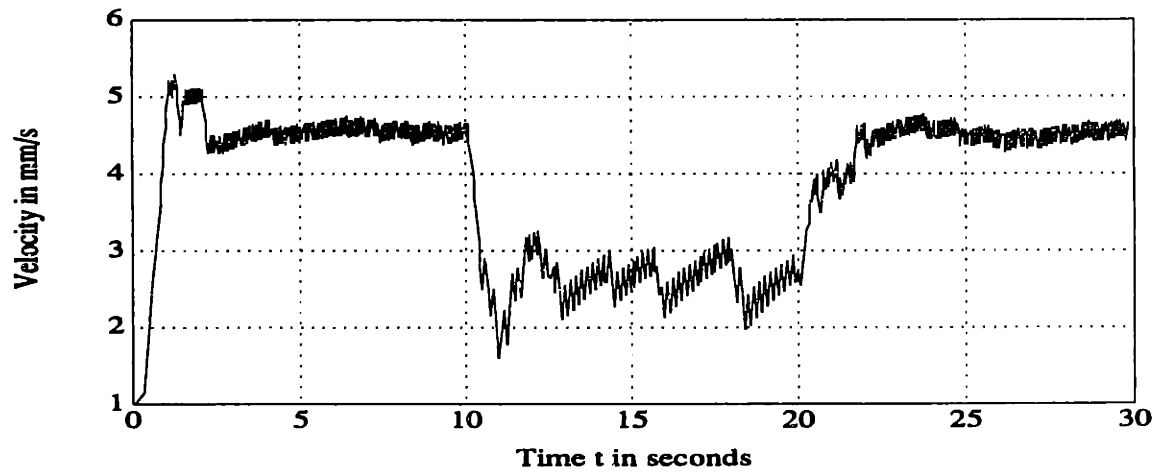


Fig. 4.24: Control input with second order TDC Controller

Figures 4.23 and 4.24 show the dynamic response of a second order TDC controller with noise and saturation, but without control input filtering ($\alpha = 5$). The gains used are: $k_1=25$; $k_2=7.07$; and $b_r=2000$. k_1 and k_2 determine the desired dynamics of the error (Eq. 10), and br represents our estimate of the feedforward term (Eq. 19).

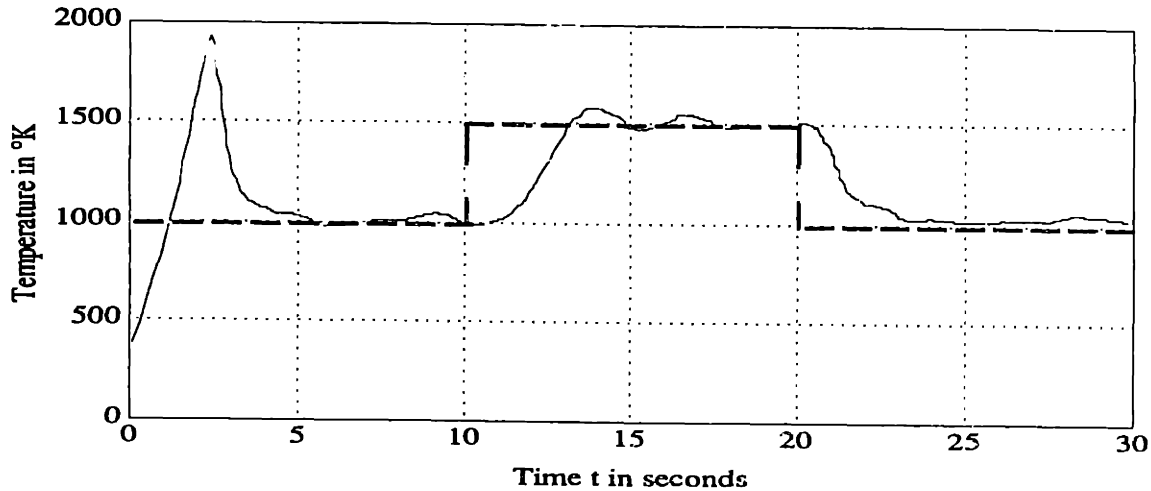


Fig. 4.25: Dynamic response with second order TDC Controller (acceleration saturation and filter)

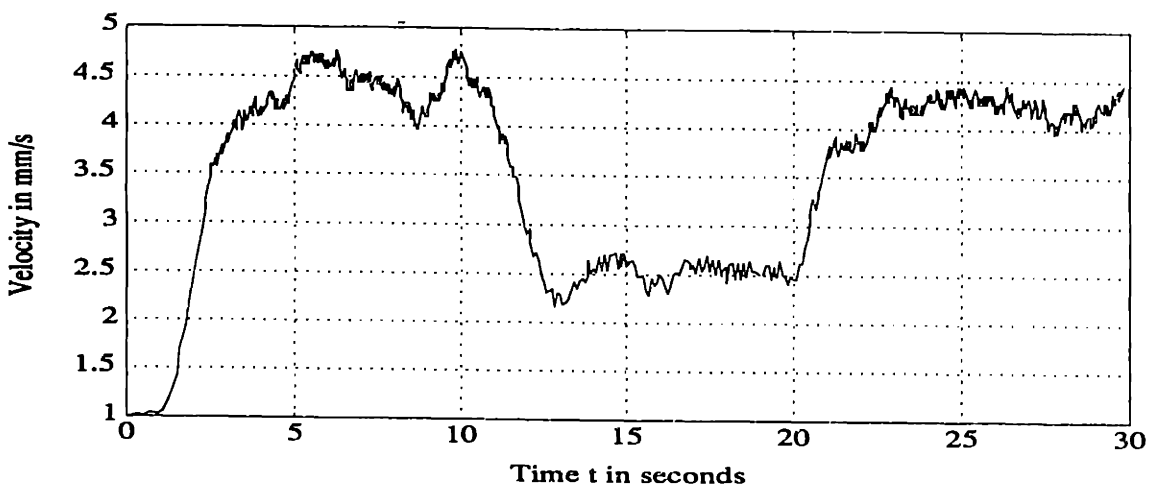


Fig. 4.26: Control input with second order TDC Controller (acceleration saturation, noise and filter)

The gains used in Fig. 2.25 and 4.26 are the same as those used in Fig. 4.23 and 4.24, except that more control action filtering is used (α= 0.05). Note that this results in slower response and much less oscillations.

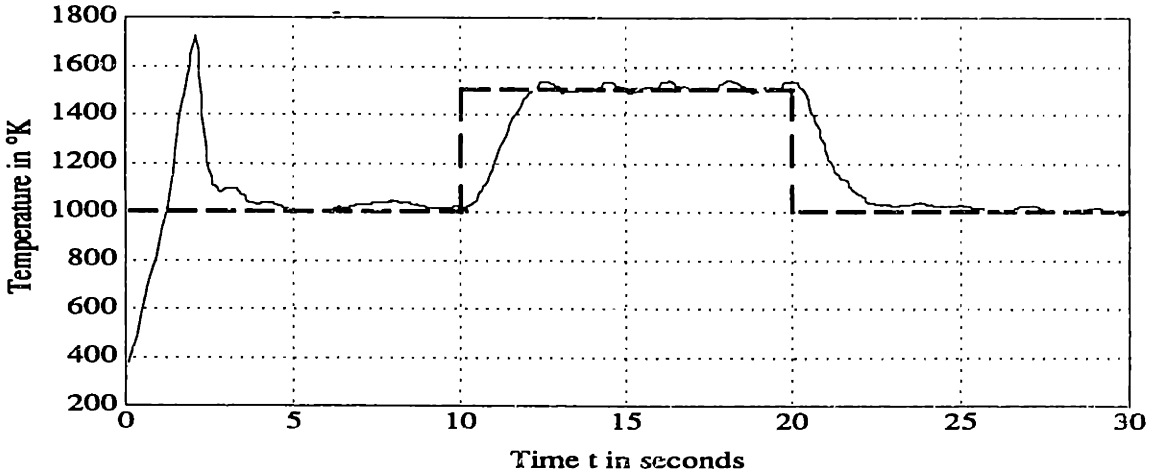


Fig. 4.27: Dynamic response with second order TDC Controller (acceleration saturation, noise and filter)

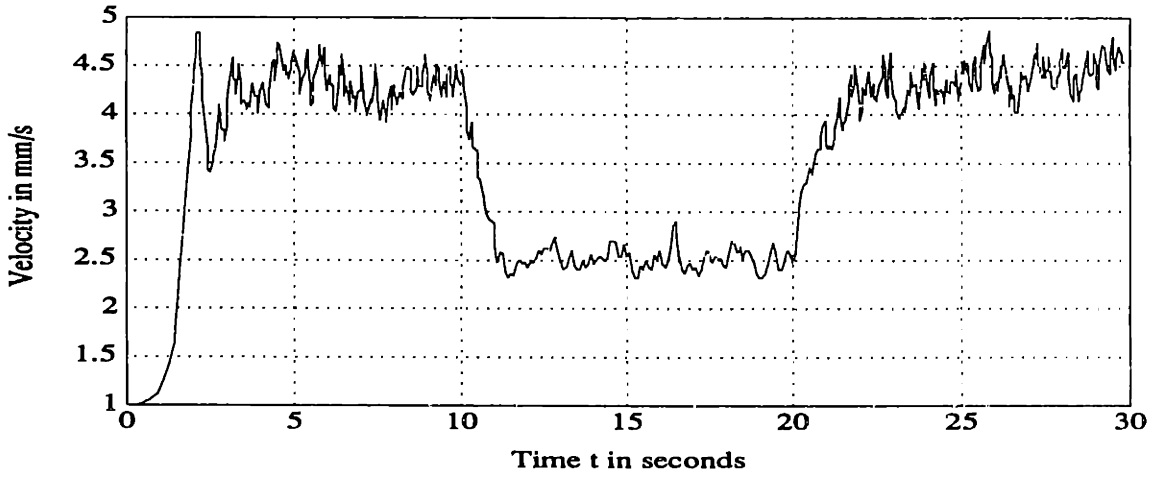


Fig. 4.28: Control input with second order TDC Controller (acceleration saturation, noise and filter)

The gains used in Fig. 4.27 and 4.28 are $k_1 = 9$, $k_2 = 4.2$, and $b_r = 3000$. Filtering of the control action is also used ($\alpha = 0.2$). Note that this results in slower error dynamics but also a much more stable control system.

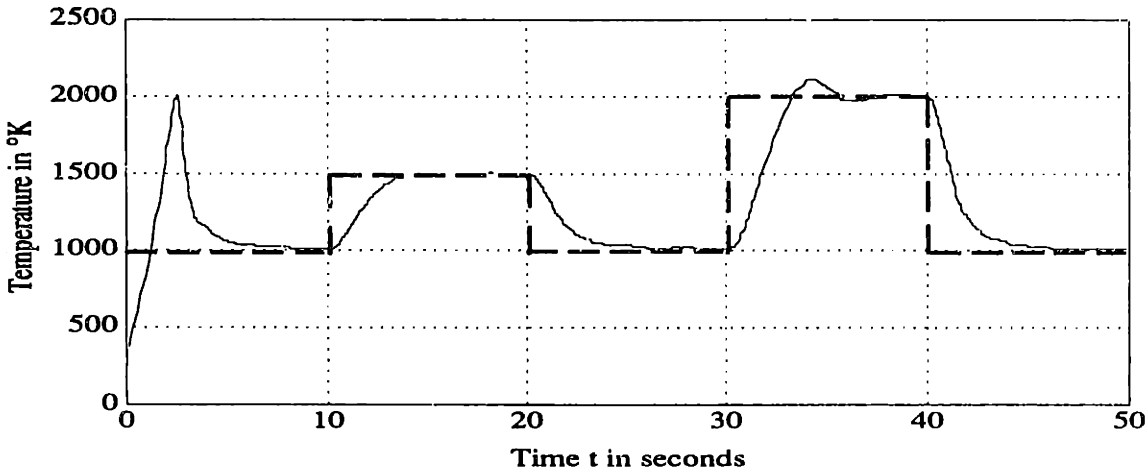


Fig. 4.29: Dynamic response with second order TDC Controller (acceleration saturation, noise and filter)

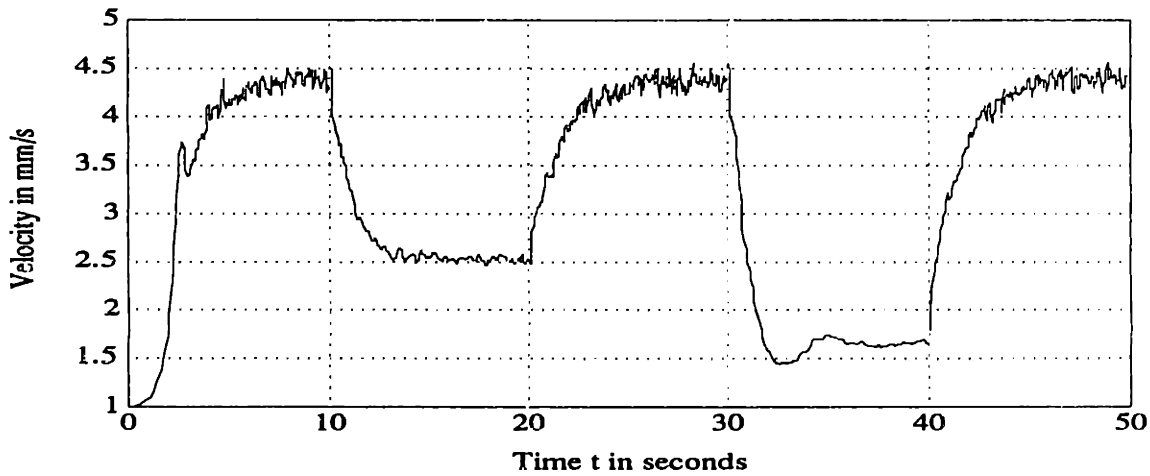


Fig. 4.30: Control input with second order TDC Controller (acceleration saturation, noise and filter)

Figures 4.29 and 4.30 show the simulation of the TDC controller using gains k_1=9 and k_2=4.2 (k_1 and k_2 determine the desired error dynamics), and b_r =4000. Input filtering is used to improve the stability of the controller (α = 0.1).

Figures 4.29 and 4.30 include sensor noise, and saturation of velocity and acceleration. These are exactly the same conditions for the simulations shown in Fig. 4.31 and

4.32 with PID control. Clearly, TDC control gives better transient response than PID control. The gains used in Fig. 4.31 and 4.32 are $K_p = K_d = 5x10^{-4}$, and $K_i = 2.5x10^{-4}$. Tuning of PID gains was attempted through simulations, and these were the gains that gave the best results.

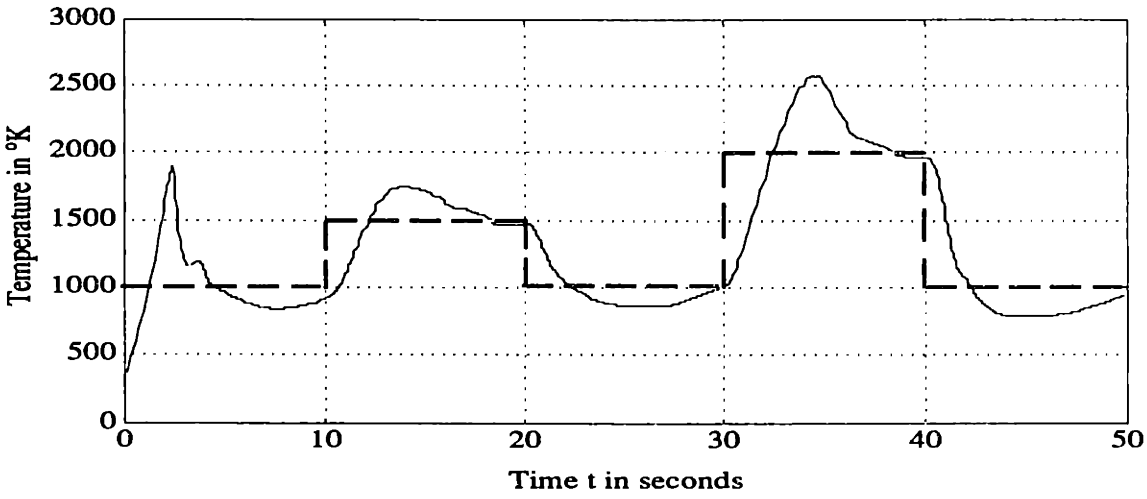


Fig. 4.31: Dynamic response with PID Controller (acceleration and velocity saturation, sensor noise and filter)

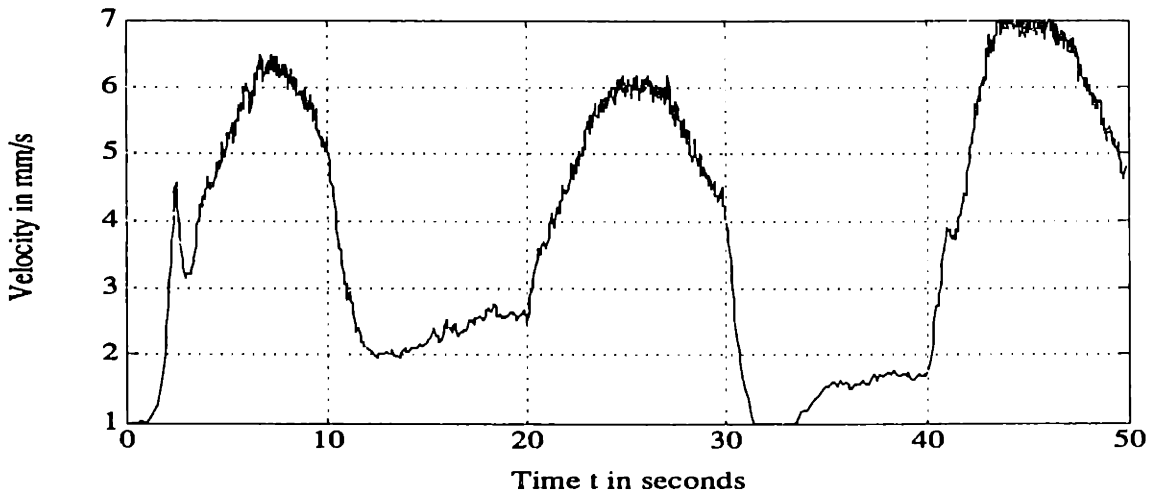


Fig. 4.32: Control input with PID Control (acceleration and velocity saturation, sensor noise and filter)

4.6.1 SISO Simulation with weave amplitude:

This section contains simulations using weave amplitude as a control input to control the temperature at a given width. Figures 4.33 and 4.34 show a SISO control system with PID control using weave amplitude as a control input. The gains are K_p= K_d= $1.6x10^{-3}$, and K_i = $0.8x10^{-3}$. Heat input is 2000 J/s, and travel speed is 3 mm/s. Weave amplitude varies between 1 and 3 mm as shown in Fig. 4.34.

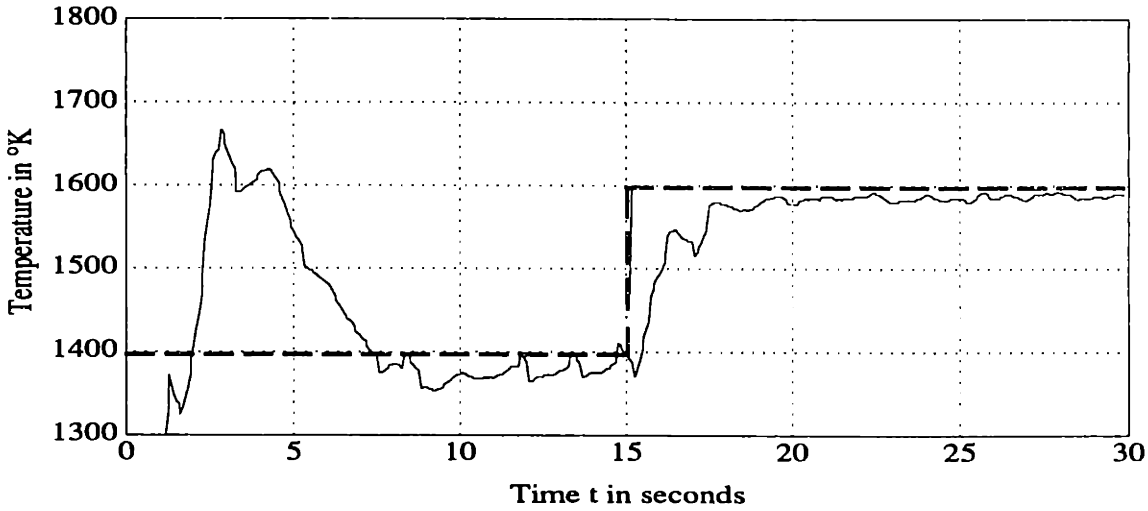


Fig. 4.33: Dynamic response of temperature with PID controller using weave amplitude as a control input

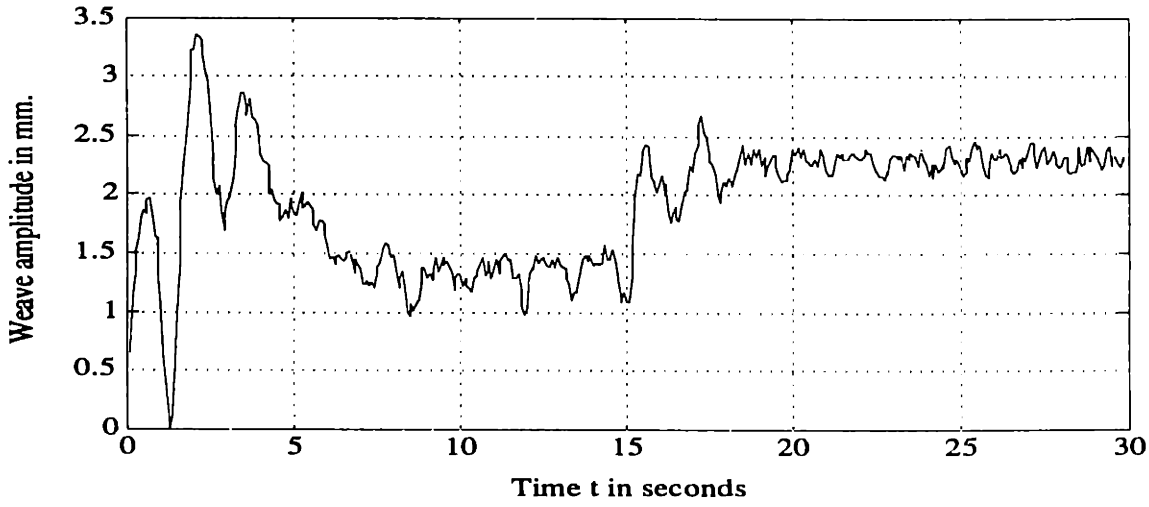


Fig. 4.34: Weave amplitude as a control input using PID control

Figures 4.35 and 4.36 show the simulation of TDC control using weave amplitude as a control input. The gains are $k_1 = 25$, $k_2 = 7.07$, and $b_r = 4000$. Input filtering is also used ($\alpha = 0.1$). In these simulations, both heat input and travel speed were kept constant, while weave amplitude was used to control temperature at the desired width. TDC gives a faster response than PID with much less oscillations. This result confirms that high frequency weaving can be used to control the temperature distribution in an arc welding process.

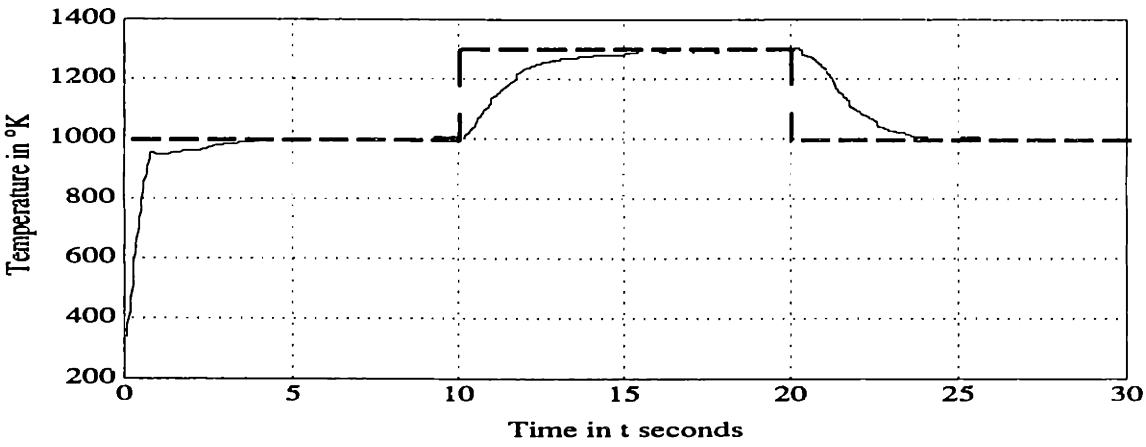


Fig. 4.35: Dynamic response with TDC controller using weave amplitude as a control input

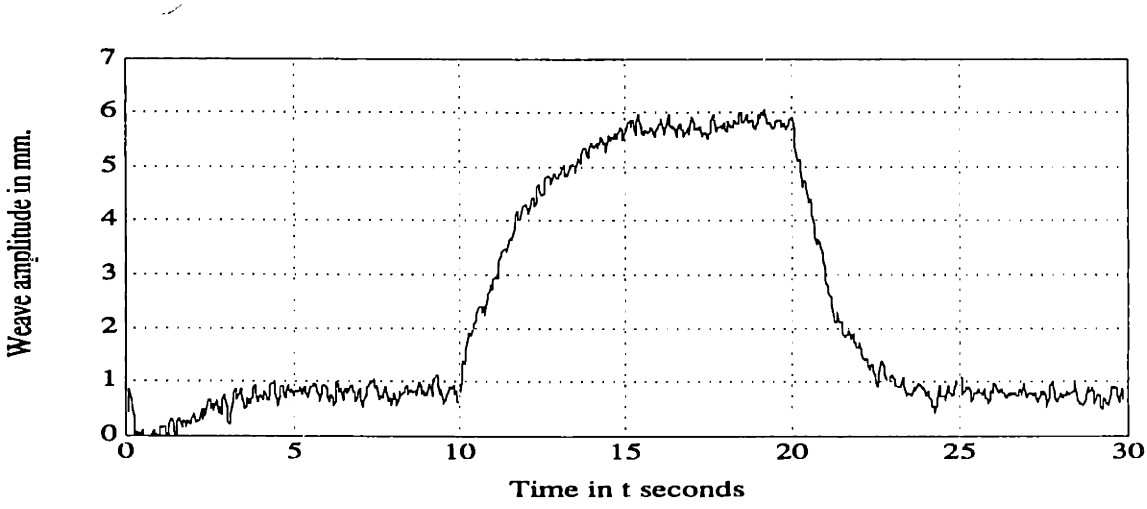


Fig. 4.36: Weave amplitude as a control input with TDC control

4.7 Multivariable Control of Arc Welding:

This section contains the simulation of a multivariable TDC control system for arc welding. The process inputs are the travel speed (v in mm/s) and the weave amplitude (w in mm.). The controlled outputs are the temperature at the desired molten pool width (set at 3 mm in the simulation) and the temperature at the desired HAZ width (set at 4.5 mm in the simulation). The gains used for the travel speed are: k_1=9, k_2=4.2, b_r=4000, and α = 0.1, and for the weave amplitude; ak_1=25, ak_2=7.07, ab_r=4000, and α=0.1. The Matlab subroutine used for simulating this multivariable decoupled control system is listed in Appendix A. Fig. 4.37 includes both sensor noise (standard deviation = 30 °K) and saturation limits for travel speed and acceleration. Fig. 4.37 shows that it is possible to control the temperature at the HAZ width independently the molten pool width. This decoupling effect is achieved by using high-frequency weaving. This allows real multivariable control of arc welding, and makes the process much more robust to disturbances and model uncertainties.

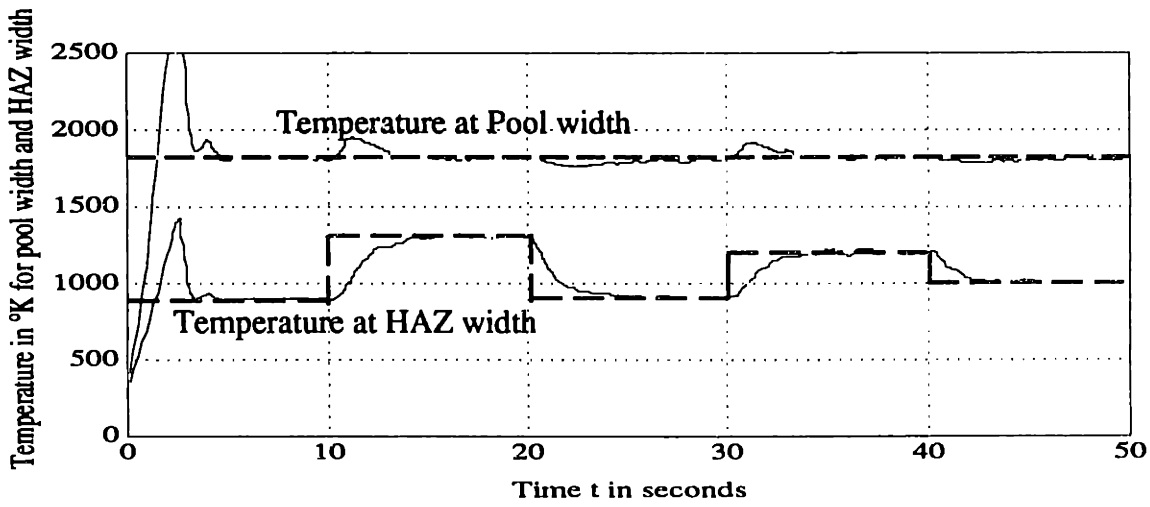


Fig. 4.37: Dynamic response with Multivariable TDC controller (inputs are travel speed and weave amplitude)

Fig. 4.38 shows the control inputs used for the multivariable control of pool width and HAZ width. The inputs are travel speed and weave amplitude. Note that this system uses two decoupled loops for controlling the pool width with travel speed and the HAZ width with weave amplitude.

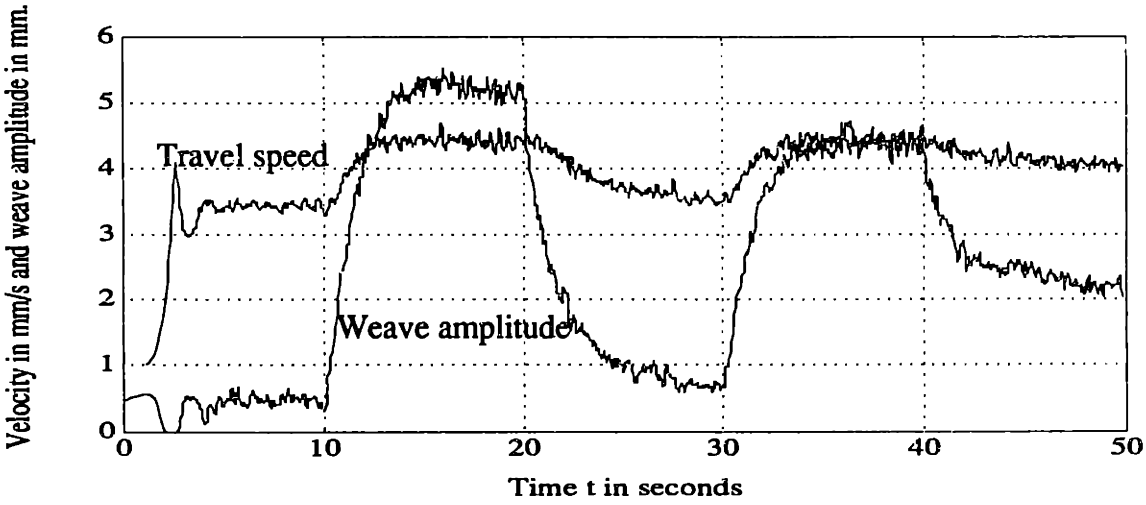


Fig. 4.38: Control inputs, travel speed and weave amplitude, with Multivariable TDC controller

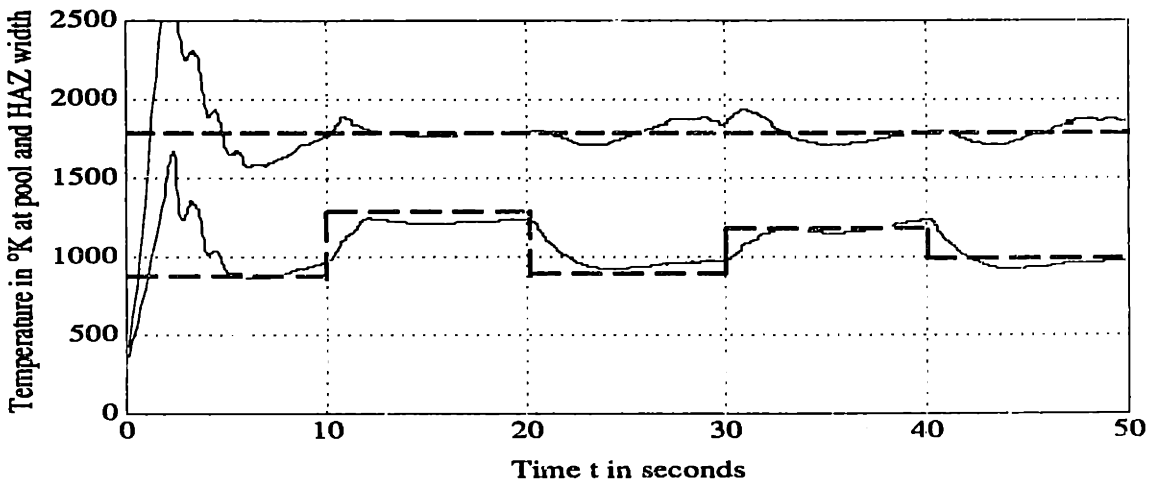


Fig. 4.39: Control inputs, travel speed and weave amplitude, with PID control

Fig. 4.39 shows the simulation of a PID controller using travel speed and weave amplitude as control input. The PID controller is much slower than the TDC controller, and does not achieve perfect decoupling. The temperature at the pool width varies as the controller tries to change the HAZ width.

As shown in Fig. 4.40, using heat input and travel speed as control inputs does not

achieve the desired results because of the high coupling between the outputs. The same TDC controller used in the previous section cannot independently control pool width and HAZ width because heat input and travel speed are both related to the heat per unit length. Hence, both control inputs are coupled and the system is in effect a single-input system. This is precisely why conventional arc welding process are not capable of achieving independent multivariable control of thermal and geometric properties, and why high frequency weaving is needed to achieve the desired decoupling.

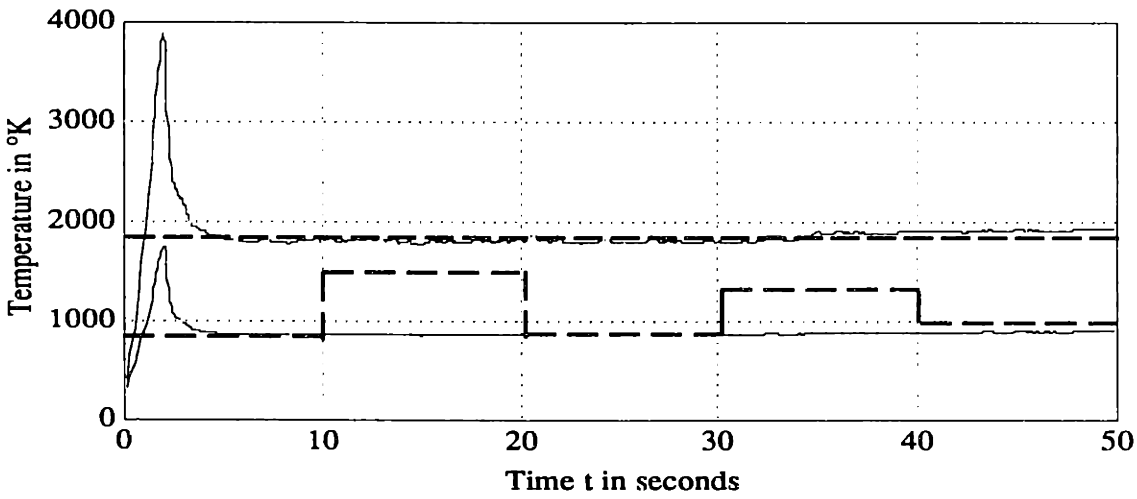


Fig. 4.40: Multivariable TDC controller with travel speed and heat input as control inputs

4.8 Robustness to disturbances:

In this section, the robustness of the TDC and PID control systems is tested through simulations. Variations are introduced in the thermal conductivity of the material, as listed in the Matlab subroutine (appendix B). For the first 10 seconds, the correct thermal conductivity *k* is used. In the second 10 second, *k* is reduced by 50 %, in the third 10 seconds, *k* is doubled, and in the last 10 seconds, a random error of between 0 and 100% is added. The gains used in the TDC simulations shown in Fig. 4.41 are:

a) for travel speed: $k_1=9$, $k_2=4.2$, $b_r=4000$, $\alpha = 0.1$,

b) for weave amplitude: $ak_1=25$, $ak_2=7.07$, $ab_r=4000$, $\alpha_a =0.1$.

As shown in Fig. 4.41, the TDC controller gives excellent robustness to modeling errors. The control task is to keep both pool and HAZ widths constant despite the variations in the thermal conductivity coefficient. This is accomplished quite accurately in Fig. 4.41, even though to a lesser extent when random noise is included in the last 10 seconds. Fig. 4.42 contains the control inputs that are used to accomplish this task. The simulations of Fig. 4.41 and 4.42 also include white noise in the measurements and saturation limits for travel speed and acceleration.

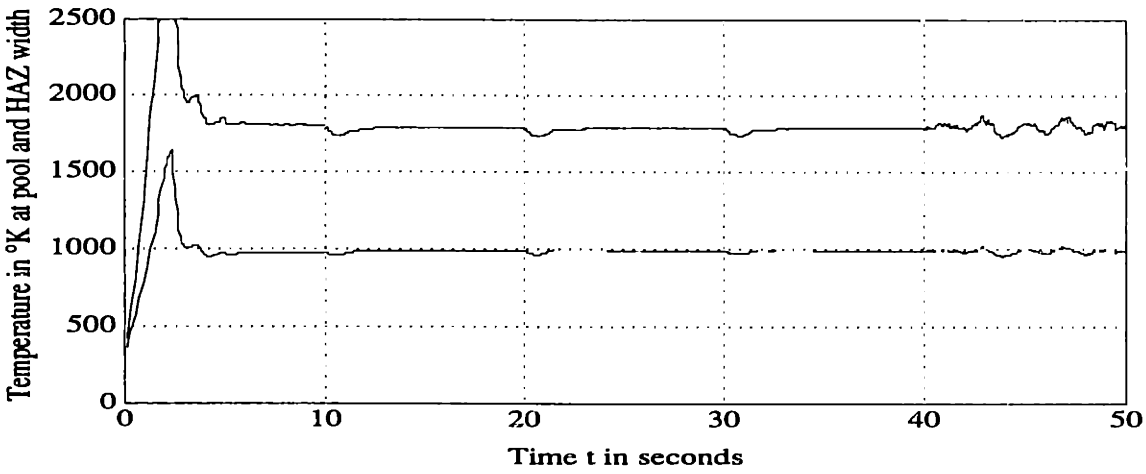


Fig. 4.41: Robustness of Multivariable TDC controller to variations in the thermal conductivity coefficient.

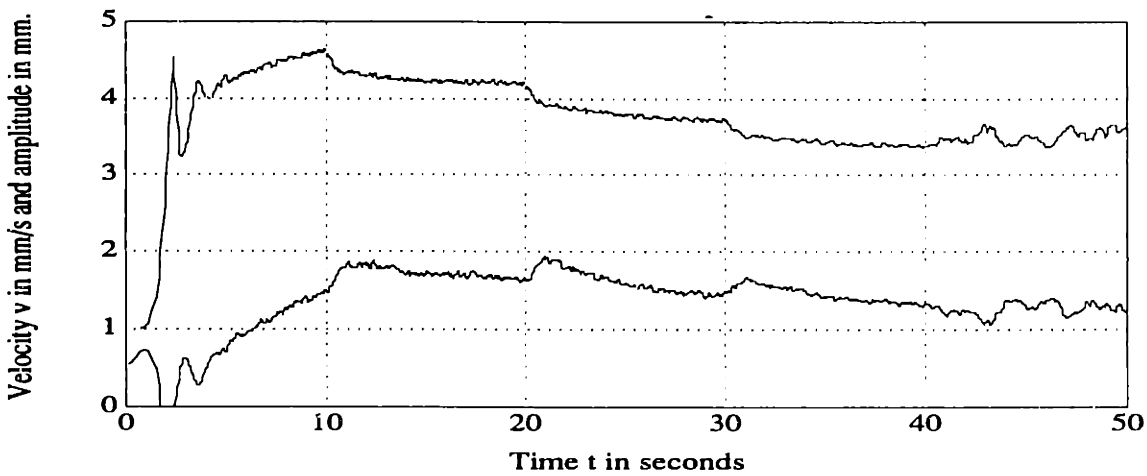


Fig. 4.42: Control inputs for Multivariable TDC controller with variations in the thermal conductivity coefficient.

Figures 4.43 and 4.44 show the robustness of the PID controller to disturbances of the same magnitude as those used with the TDC controller in Fig. 4.41. The gains used in these simulations are: $K_p = K_d = 1.6 \times 10^{-3}$, and $K_i = 0.8 \times 10^{-3}$. Tuning of the gains was performed through simulations, and these gains provide the best conditions for the PID controller. The response of the PID controller to variations in the coefficient of thermal conductivity, k, is however not very robust. The control algorithm is not capable of holding temperature at the desired locations constant. From these simulations, it can be concluded that Time Delay Control is much more robust to disturbances and modeling errors than PID.

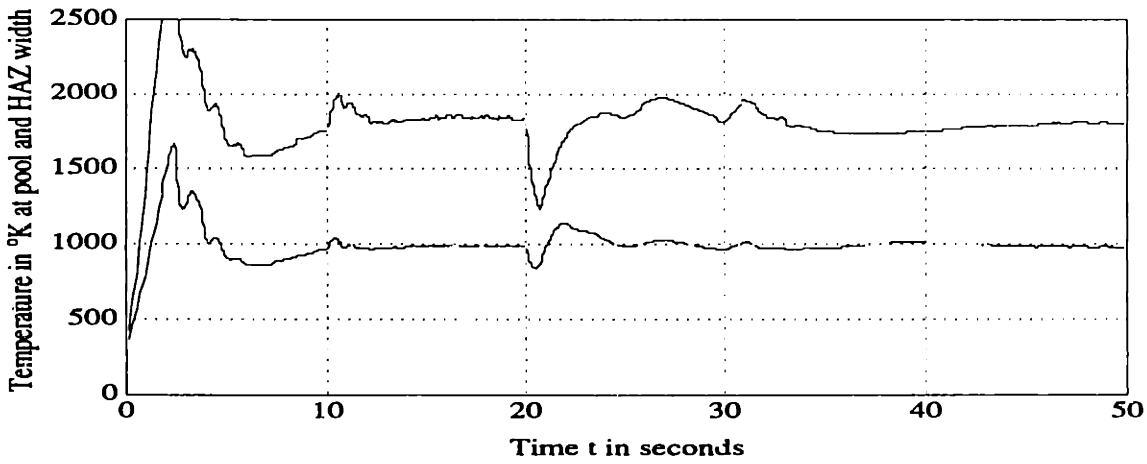


Fig. 4.43: Robustness of Multivariable PID control to variations in the thermal conductivity coefficient.

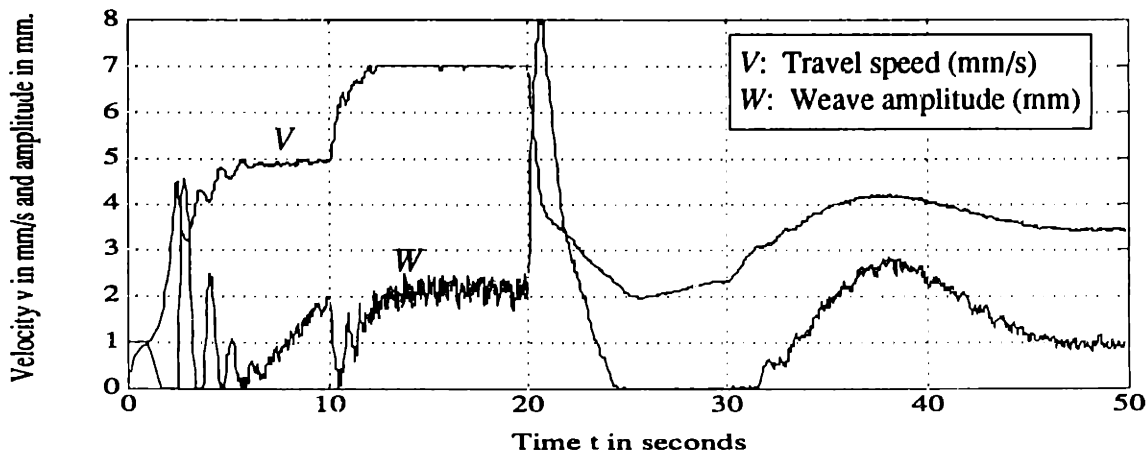


Fig. 4.44: Control inputs of Multivariable PID control with variations in the thermal conductivity coefficient.

4.9 Conclusions:

In this chapter, two types of control systems for arc welding were designed and simulated. The first type is the traditional linear PID controller, and the second method uses the theory of Time Delay Control with input/output linearization. Simulations were performed using the nonlinear dynamic MATLAB model for arc welding developed in chapter 2. Simulation results show that TDC provides superior performance and better transient response than PID control. This can be easily explained by the fact that the PID method is a linear technique, and is not very suitable for inherently nonlinear systems such as the arc welding processes. Gain scheduling was not attempted, however the controller gains were tuned through simulations. TDC uses past information of the inputs and the outputs to estimate the dynamics of the process, assuming that the sample delay time is much smaller than the fastest time constant of the process. In the case of arc welding with mild steel, the heat diffusion time constant is approximately 0.5 seconds, thus the sample time has to be faster than 0.25 seconds.

A TDC multivariable control system was developed and simulated using travel speed and weave amplitude as control inputs. Simulations confirm that high-frequency weaving allows independent control of pool width and HAZ width. The TDC control system of independently controlling the HAZ width while keeping the pool width constant. This decoupling between the outputs cannot be achieved by using heat input and travel speed as control inputs. This is due to the fact that heat input Q and travel speed v are both related to the heat-per-unit length (Q/v). High frequency weaving therefore provides more control of the heat distribution during arc welding. The TDC controller was also shown to have excellent robustness to model errors and uncertainties. Simulations were performed with large variations in the coefficient of thermal conductivity, and the TDC controller was much more stable and robust than PID.

Chapter 5

Control System Implementation

This chapter explains the hardware and implementation issues that were faced during the process of implementing the control algorithm developed in chapter 4, and explains how these issues were resolved. These implementation issues are always present with any feedback control system. An important issue in the implementation of any control system is the existence of accurate and consistent sensors. This issue is especially complex when dealing with extremely rough environments such as the case with arc welding processes.

The feedback control system developed in chapter 4 requires the measurement of pool width and heat-affected zone width; the two outputs that we are interested in controlling (Fig. 5.1). The pool width measurement is obtained using a vision system with a CCD camera and frame grabber board to digitize the image for image analysis by the computer. The HAZ width is obtained using an Infra-Red temperature sensor with a one dimensional scanning mechanism. The torch is weaved at a frequency of 3 Hz. by a DC motor-based servomechanism.

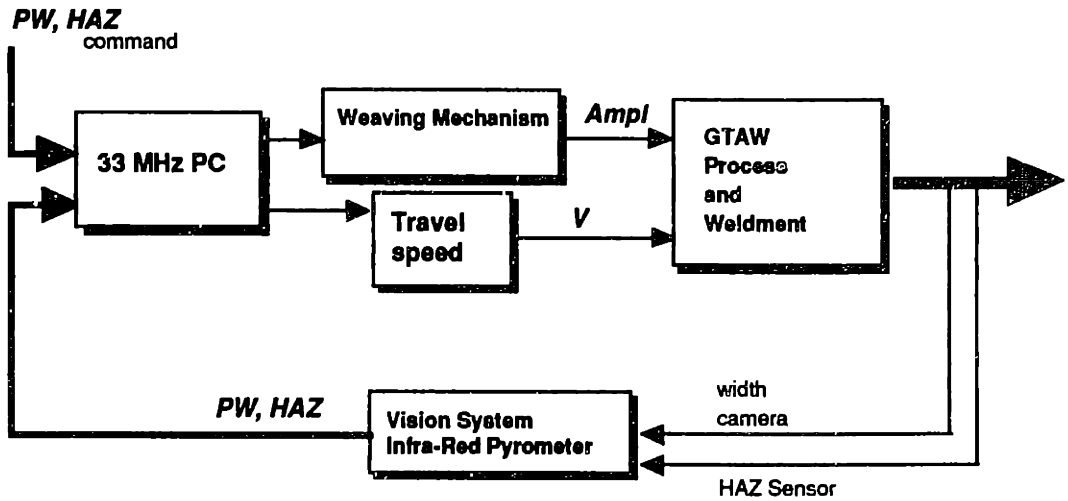


Fig. 5.1: Block Diagram of Arc Welding Process Control System

The hardware components built and used in the course of implementing closed-loop control are explained in this chapter, and PID closed-loop experimental results are presented at the end of the chapter. TDC control could not be implemented with the current hardware because of insufficient computation power. One 33 MHz Personal computer was used to perform all the calculations required for temperature measurements, image analysis, control of the weaving mechanism, and calculation of travel speed and weave amplitude. The computer was not capable of performing all these computations at a rate of at least 4 Hz, which is the minimum sampling time for the TDC controller. Therefore, to implement TDC, either a faster computer has to be used or a dedicated Digital Signal Processor (DSP) board has be dedicated to the weaving mechanism. Due to these hardware limitations, TDC could not be implemented and the closed-loop experiments presented in this chapter all use PID control.

5.1 High-Frequency Torch Weaving Mechanism:

Figure 5.2 shows the block diagram for the torch weaving mechanism. The system consists of a DC motor that drives a leadscrew-equipped table on which the torch is mounted. The DC motor is controlled by a Programmable Motion Controller (PMC) board that communicates with the PC through the bus. The system uses a tachometer and a position encoder for velocity/position feedback. There are two limit switches on both sides of the table in order to prevent the table from reaching the edges. This is a flexible system that can be used for accurate positioning of the torch. The total length of the table is 400 mm., and the system is capable of weaving frequencies up to 10 Hz. with an amplitude of 10 mm.

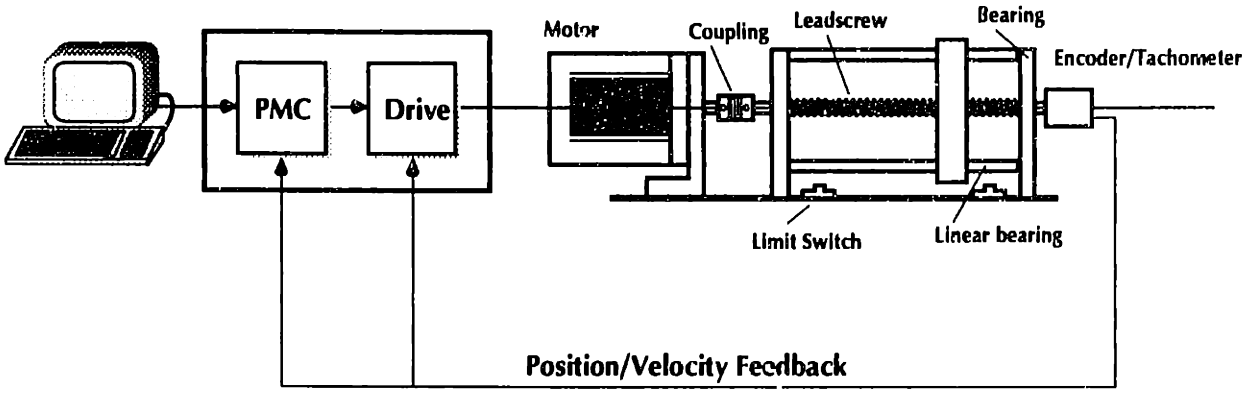


Fig. 5.2: Diagram of the High-Frequency Weaving Mechanism

Figure 5.3 shows the schematic of the weaving mechanism control system which includes one PC, one PID controller card, two servo-amplifier and a DC power supply, two Permanent Magnet DC motors, with incremental encoders and tachometers for position and velocity feedback. The encoder output goes directly to the controller card, which calculate the number of encoder counts per sample time, while the tach output goes to an Op-Amp circuitry that calculates the difference between the reference velocity and the actual velocity and inputs a signal proportional to this error. During experimentation with the hardware system, it was found that the tachometer feedback significantly increases the bandwidth of the system without decreasing its stability. This is specially important since the motor will be oscillating at high speeds.

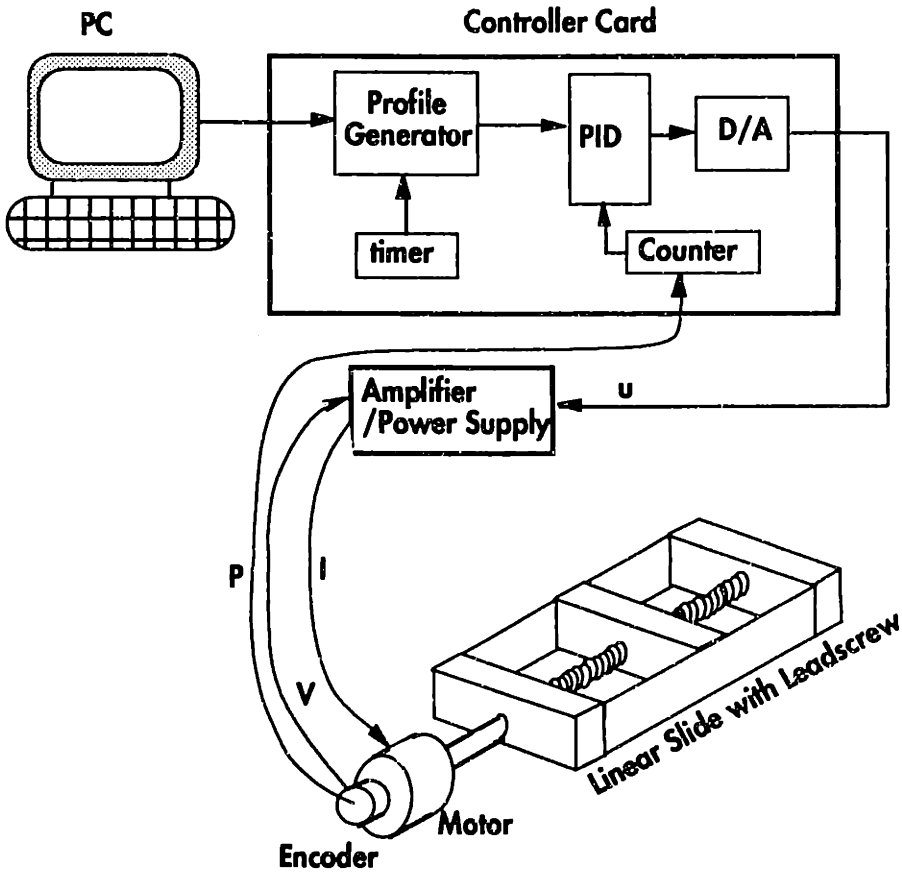


Fig. 5.3: Weaving Mechanism Control System Diagram

5.2 Scanning Mechanism for Infra-Red Sensor :

The Infra-red sensor is equipped with a line scanning mirror that scans at a frequency up to 20 Hz. The reason for using a scanning mirror is the interest in finding the peak temperature at a specified width in order to control the HAZ width. The location of the peak temperature along a particular width depends on the heat input and the travel speed.

The scanning mechanism consists of a mirror mounted on stepper motor driven by a SAC-560 programmable stepper controller (1992). The programmable controller communicates with the PC through the serial port and has storage memory that can store motion trajectory programs. This storage capability relieves the PC from doing real time control of the stepper motor and saves valuable computation time. Each step of the stepper motor corresponds to 0.9°.

5.3 Infra-Red Temperature Sensor:

An Accufiber Model 100 Multi-Channel Optical Fiber Thermometer High Temperature Measurement Device was used for measuring point temperatures. This is a sophisticated patented Optical Fiber Thermometry (OFT) technology which overcomes many of the problems associated with infra-red temperature measurements. The Accufiber 100 system is illustrated in Fig. 5.4.

The principle of thermography is straightforward. A single crystal of aluminum oxide sapphire is coated at its tip with a thin film of a precious metal in order to form a blackbody cavity. When the cavity is heated, it emits radiation, which is accurately described by the Planck equation. The sapphire crystal transmits the radiant energy to a low temperature optical fiber, which in turn transmits it to a remote optical detector. At the detector, the radiation is converted to an electrical signal, linearly amplified, and digitized. A microprocessor is used to convert this signal into the temperature of the blackbody (see Accufiber (1990) for further explanation of thermography.)

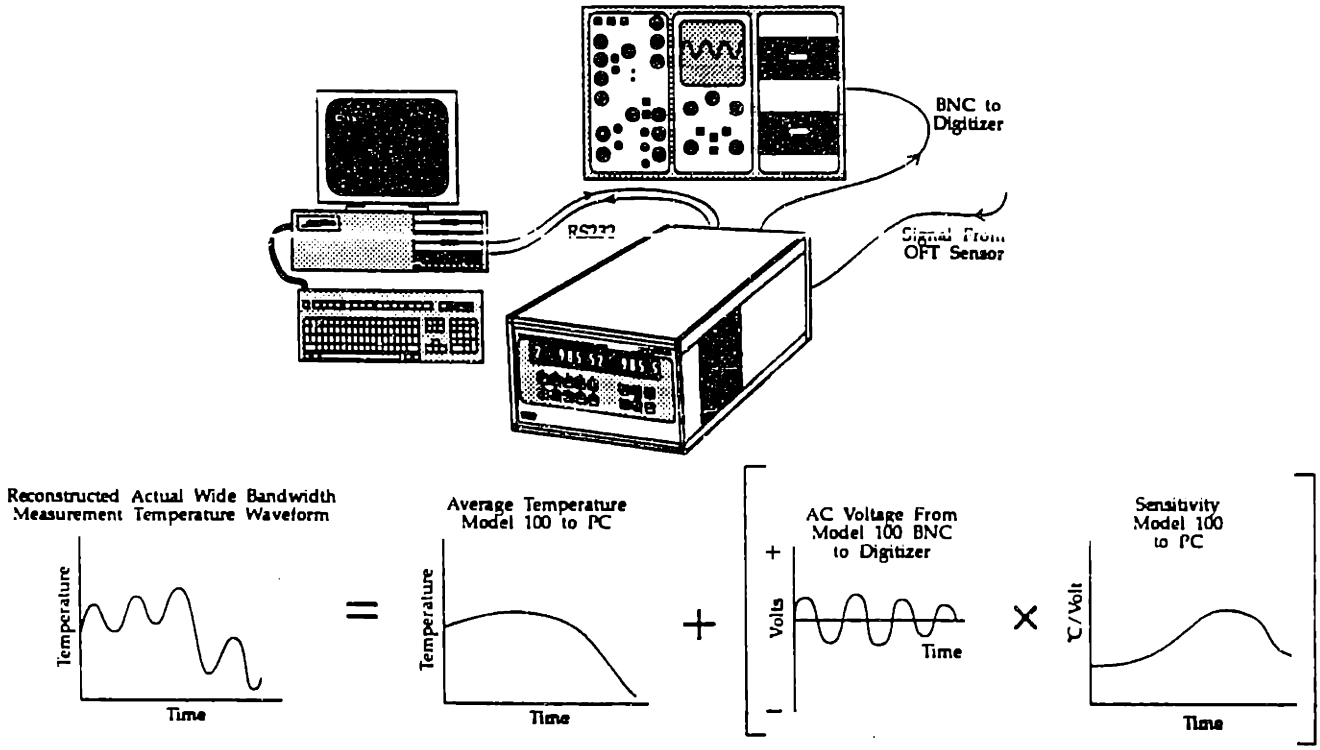


Fig. 5.4 Accufiber 100 Infra-Red Pyrometer (Accufiber, 1990)

The main advantages of this system are:

1- Remote temperature sensing: The pyrometer head containing the optical lenses is placed at a distance of 12 in. away from the weld.

2- Small spot size: At 12 in. from the object, the spot size is 0.8 mm.

3- Dual wavelength: This feature separates the incoming radiant energy into two wavelengths; 800 nm and 950 nm These two measurements can be used independently or by using the two-color ratio method (the ratio method was found unacceptable close to the arc because of the high noise content.) Accufiber 100 uses two independent filters and photodiodes for the two wavelengths.

4- Wide bandwidth output: provides a 0-10 KHz analog signal for subsequent digitization and post-external processing.

5- RS-232 Serial Port: Communicates with the PC at a rate of approximately 25 Hz.

6- High temperature range: The system is calibrated for measuring temperatures in the range of 700 to 2400 °K.

7- Built-in programmable digital filter: uses a low-pass filter with an adjustable cutoff frequency of 0-25 Hz. in order to reduce the high-frequency noise.

8- Built-in PID control: This feature, which outputs a signal to correct for deviations from a setpoint, was not used since we are interested in nonlinear control methods.

9- Programmable emissivity: The emissivity can be programmed as a second order polynomial function of temperature (if such a function is known a-priori).

10- Host message interface: Acccufiber 100 is programmable through the serial port. Any of the functions or variables may be read and changed by using simple high-level commands. For more information, see the Accufiber 100 manual (1990).

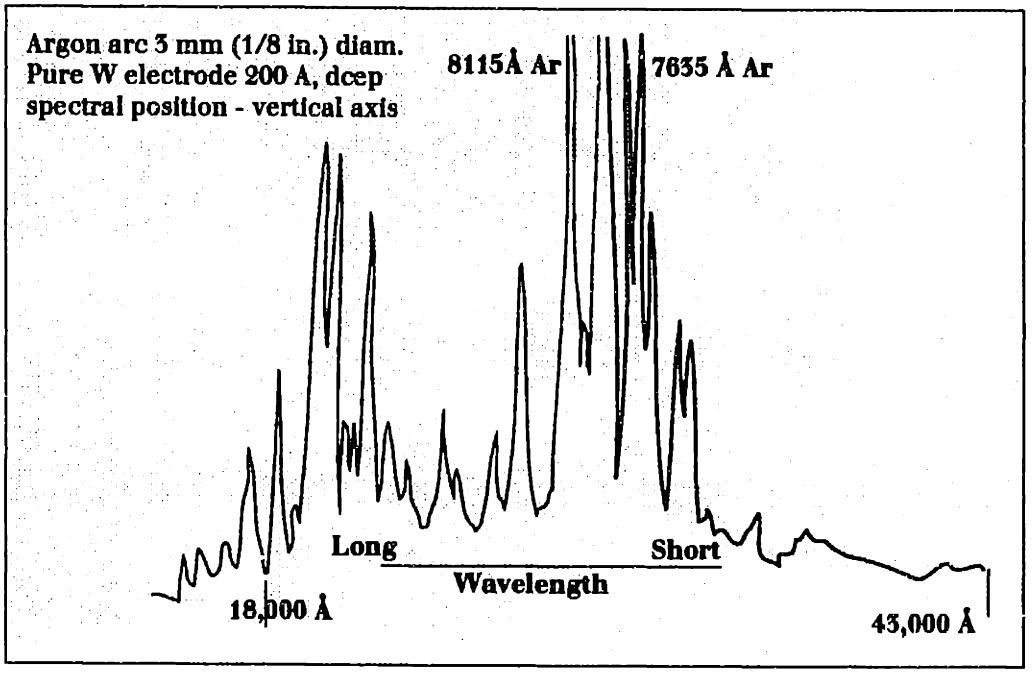


Fig. 5.5: Spectrum of the Argon shielded Gas Tungsten Arc (The Welding Handbook, 1989)

The hardware which was built and used for implementing multivariable control of arc welding processes is shown in the two photographs on next page. Photo 1 shows the computer which was used for data acquisition and real-time control, the vision system which was used for measuring the pool width with a CCD camera, and the Accufiber Infra-Red sensor used for temperature measurement and monitoring. Photo 2 shows the GTAW torch mounted on a high-frequency weaving mechanism, the infra-red pyrometer head, the scanning mirror used to measure the temperature along a specified width, and the Automatic Voltage Controller. A DC motor was also used to move the workpiece at a controlled travel speed while the torch is weaving in the y-direction.

As shown in Fig. 5.5, the GTAW arc emits radiation at almost all wavelengths on the spectrum which means that measuring the temperature close to the pool will be contaminated by arc radiation noise. There are two ways to reduce this problem. The first method is to use the two-color pyrometer and take the average of the two measurements. The second way is to use the wavelength that has less noise in the signal. With the welding system used, it was found that temperature measurements performed at a wavelength of 950 nm contained much less noise, and therefore it was used in the feedback control law.

5.3.1 Open-Loop Temperature Measurements:

Figure 5.6 shows two graphs for open-loop temperature measurement for GTAW bead-on-plate welding of 0.5 in. by 2 in. by 36 in. A36 mild steel. The first one (a) shows the measurements obtained while scanning one line with the scanning mirror. It is shown that the IR sensor picks up the maximum and minimum temperatures at the correct scanning frequency (about 6 Hz.). This result is important because it allows us to find the peak temperature at a specified width location. The second plot (b) shows the temperatures obtained at the two different wavelengths without scanning, as well as their average. Calibration of these results is necessary if we are interested in the absolute value of the temperature to determine whether it has reached the desired temperature (T_{c1} in the case of HAZ). However, for HAZ width measurements, we can calibrate the measured signal directly to the HAZ width, rather than to absolute temperature.

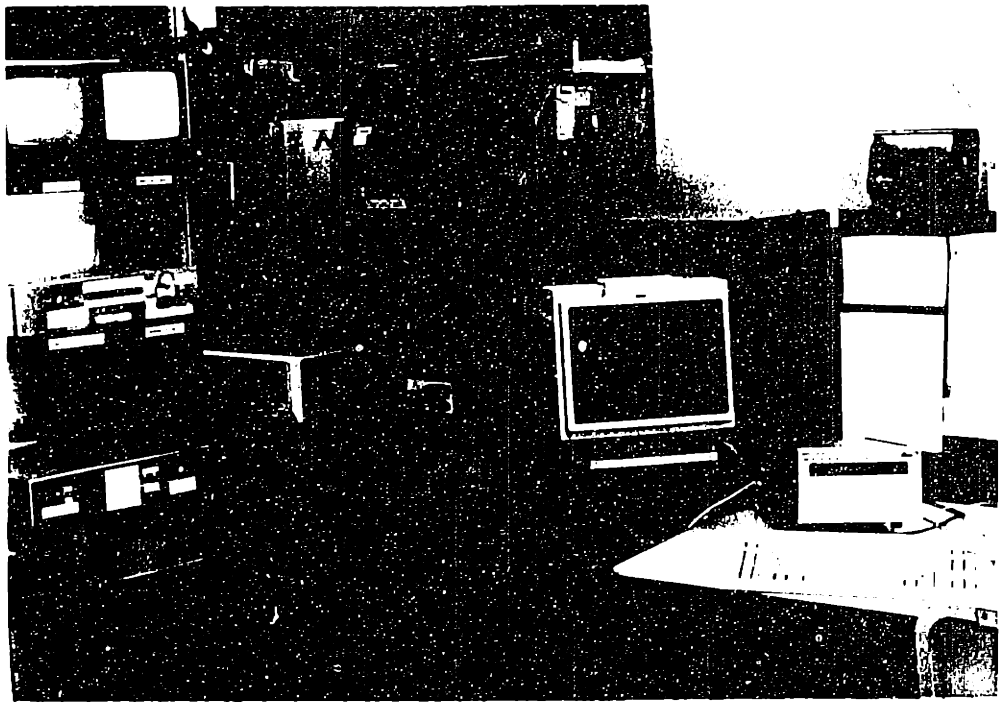


Photo 1: Hardware used to control and monitor the pool width and HAZ width during closed-loop experiments:
Computer, Vision system, and Accufiber temperature measurement system.

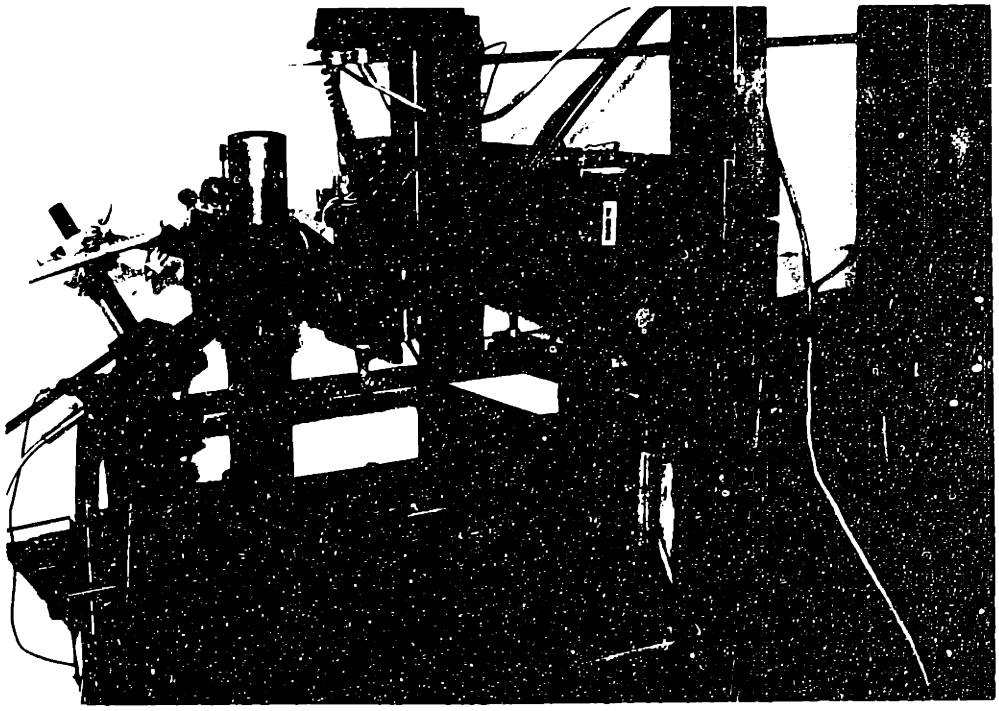
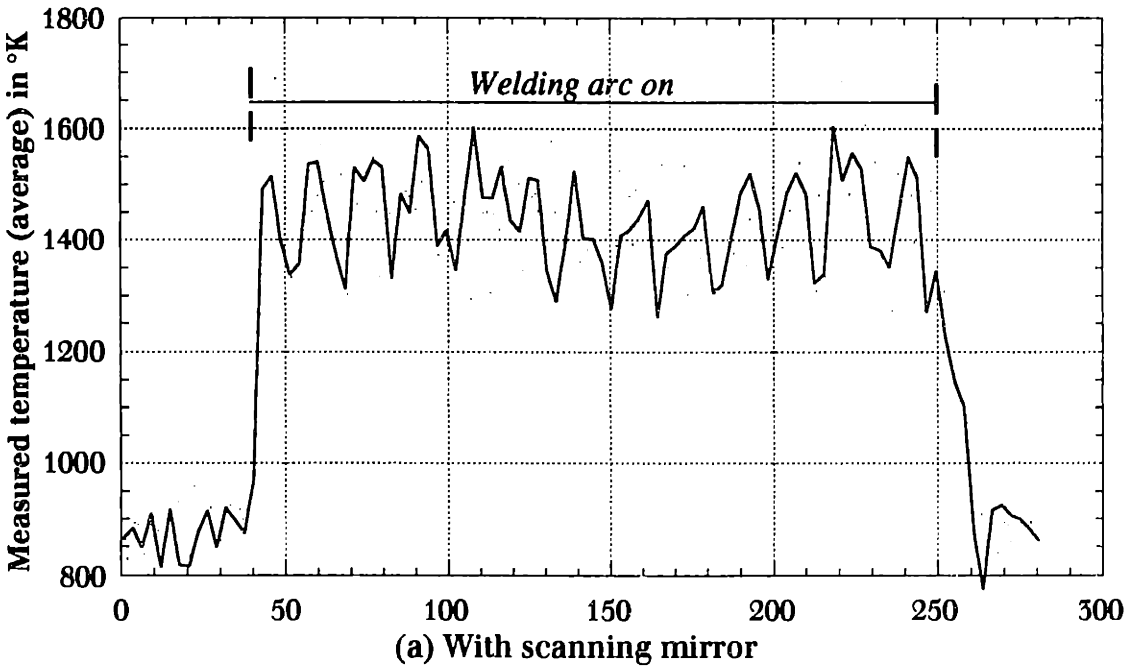
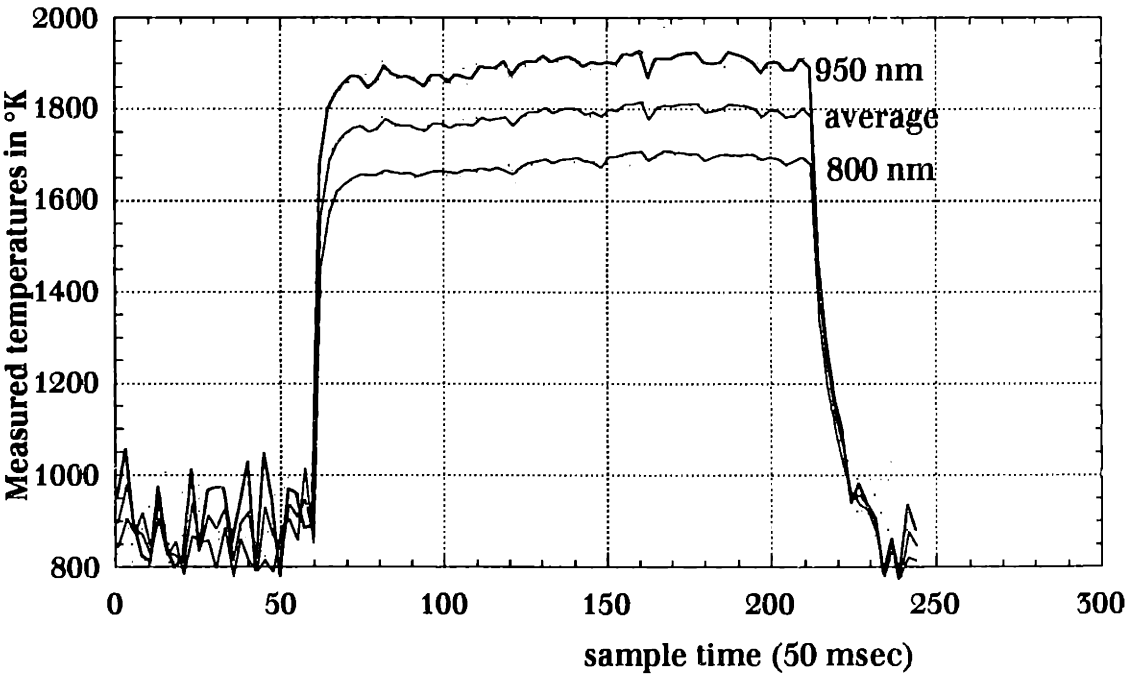


Photo 2: GTAW welding equipment used to carry out the closed-loop experiments:
Weaving system, scanning mirror, IR pyrometer, and Automatic Voltage Controller



(a) With scanning mirror



(b) Without scanning mirror (at 2 wavelengths)

Fig. 5.6: Open-loop Infra-Red temperature measurements without weaving. (a) with scanning mirror (b) at a fixed point.

Figure 5.6 (a) contains noisy data which is curve-fitted with a cubic spline. The power spectral density of the measurement data is plotted in Fig. 5.7. In addition to the noise, there is a peak magnitude at 6 Hz. which is the frequency of the scanning mirror. The purpose of using a scanning mirror is to find the peak temperature at the specified width. The location of the peak temperature behind the torch is not a constant and varies

as a function of travel speed. There is also another peak at a frequency of 2 Hz. which corresponds to the time constant of the heat diffusion in mild steel.

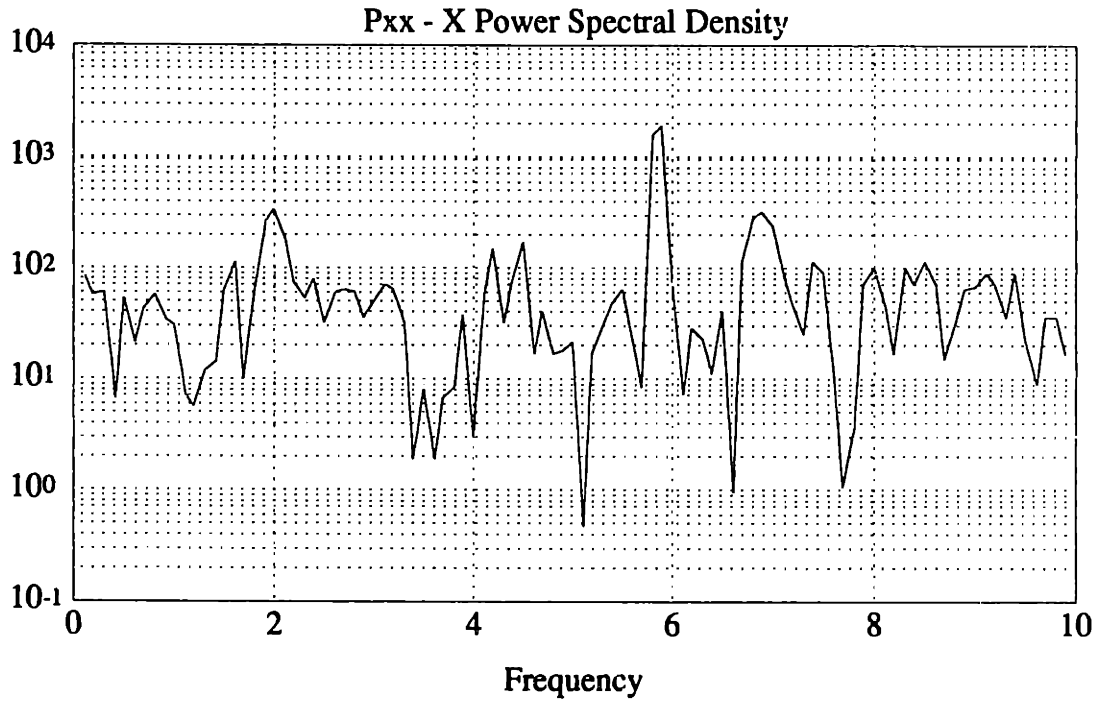
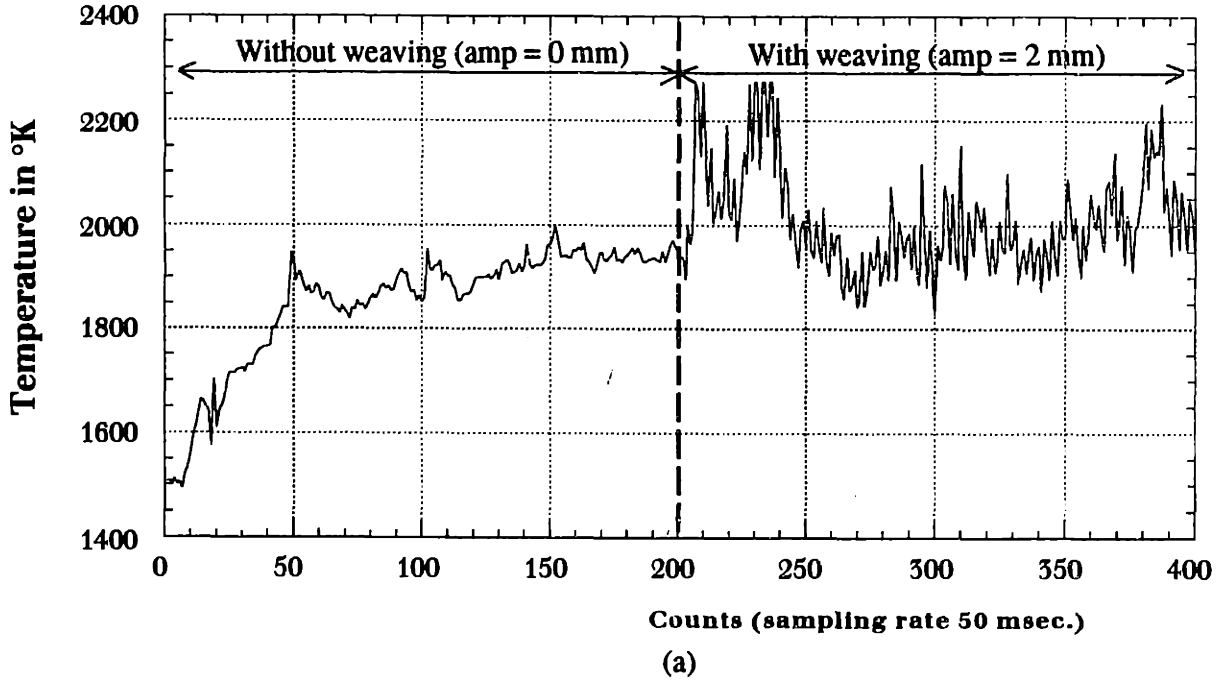
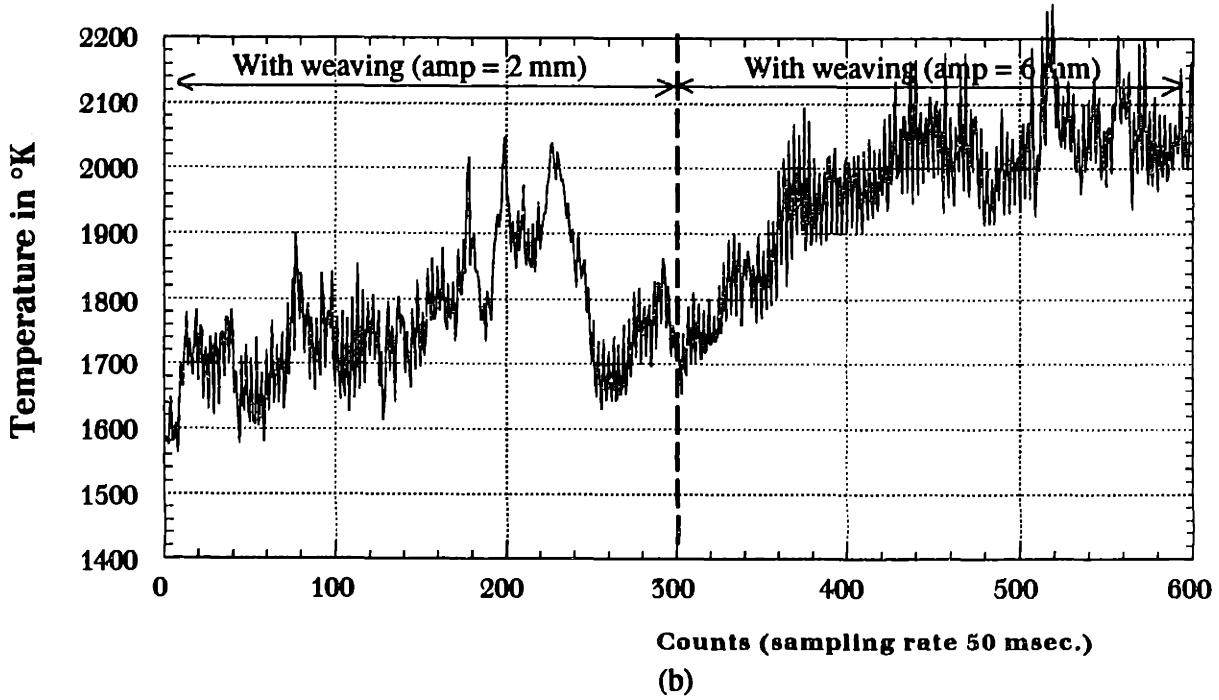


Fig. 5.7: Power Spectral Density of temperature with scanning mirror at 6 Hz. (data plotted in Fig. 5.6)

Figure 5.8 shows open-loop temperature measurement experimental results with weaving. The first one (a) shows the effects of weaving which not only increases the absolute value of the temperature (by about 75 °K in this case), but also increases the level of noise in the signal due to the moving arc. In plot (a), the digital filter bandwidth was set at 25 Hz. In plot (b), no filtering of the signal was used, and the signal is much more noisy.



(a)

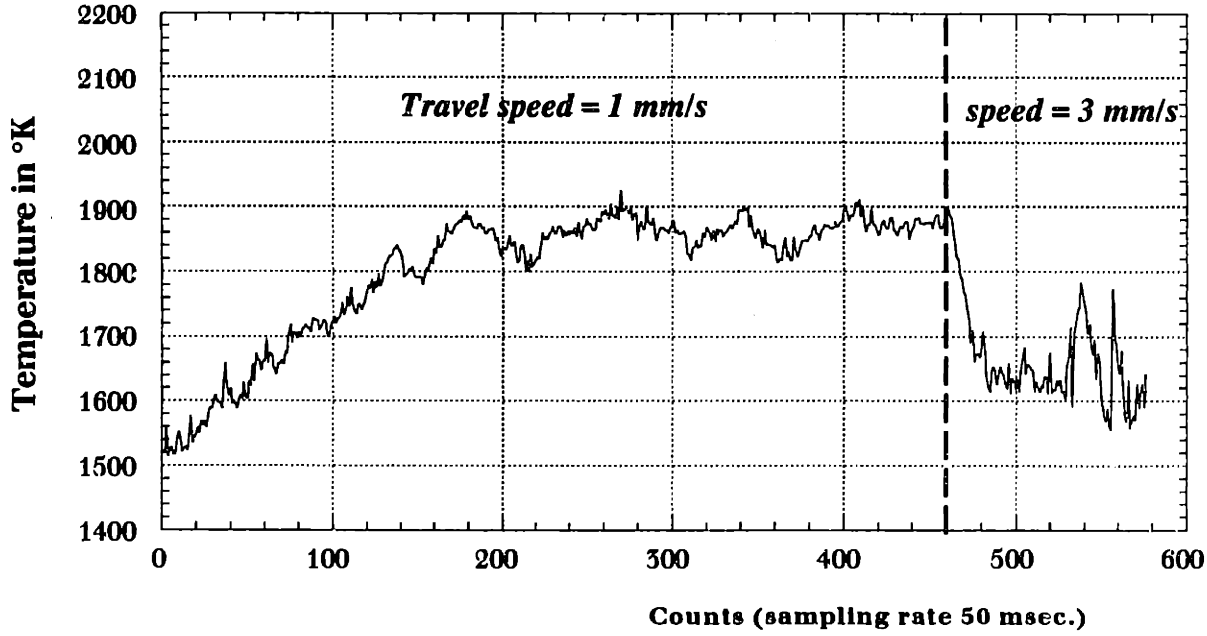


(b)

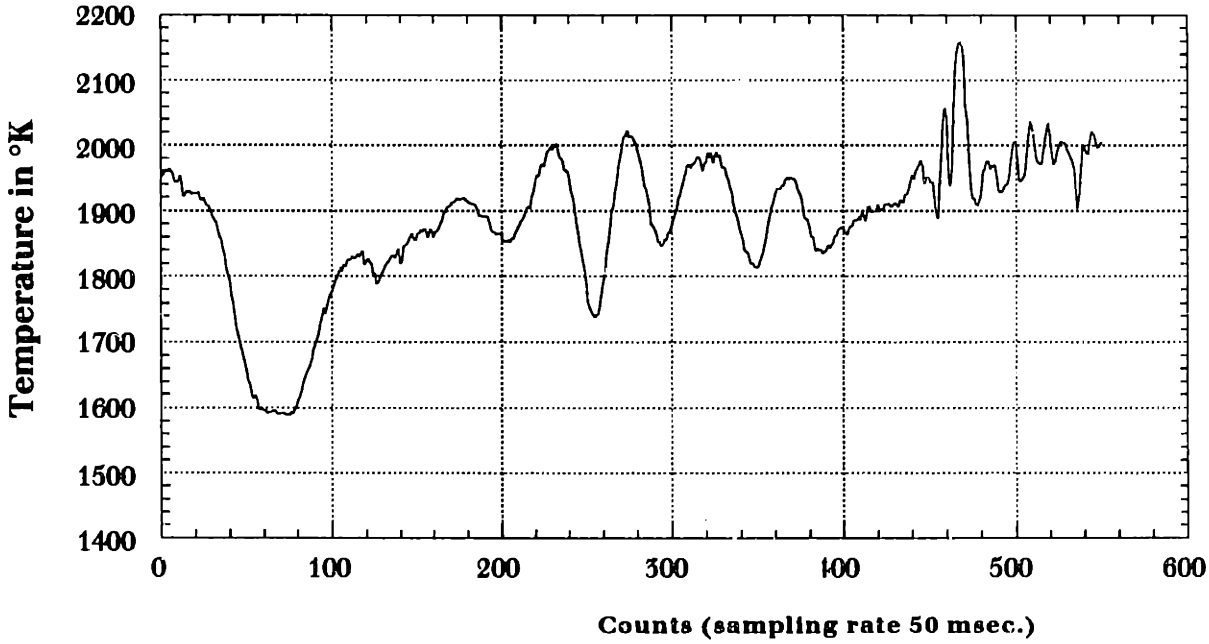
Fig. 5.8: Open-loop temperature measurements with a step change in weave amplitude: (a) filter at 25 Hz, (b) no filter.

Figure 5.9 shows more open-loop temperature measurements. In the first plot (a), the temperature transient is shown for a change in travel speed from v=1mm/s to v=3mm/s. The temperature increases (from ambient temperature which is outside the sensor's range) to 1850 °K in about 20 seconds. Then travel speed is increased from 1 mm/s to 3 mm/s, and temperature drops to approximately 1600 °K in 3 seconds. The second plot (b) shows the temperature measurements obtained with an unstable arc. This plot indicates that this sensor can also be used to monitor the stability of the arc, although that is beyond the scope of this work.

One major shortcoming with communicating through the serial port for temperature measurements is that the serial port is too slow. Each time that new data is available for transmission, the serial port must be initiated and its status must be checked. There are basically two ways for communicating through the serial port. The first method, and the simplest, is called ***polled I/O***. At pre-specified time intervals, the computer must check the serial port to see if new data is available by checking if DATA_SEND bit is set. If it is, then it sets DATA_RECEIVE bit, and the transmission begins. Polled I/O is a slow process because of all the checking that must be done continuously whether new data is available or not. The second method, much harder to implement but also much faster, is called ***interrupt driven I/O***. In this method, whenever new data is ready, an interrupt is issued to the computer. The computer then stops execution of the current program, reads the data and stores it in a buffer, then returns to the currently interrupted program. Interrupt-driven I/O is much faster because there is no periodic checking of the status of the serial port. In the following experiments, interrupt-driven communications (at 9600 baud) was used through serial port.



(a)



(b)

Fig. 5.9: Open-loop temperature measurements (a) change in travel speed, (b) unstable arc.

5.3.2 Closed-Loop Temperature Control:

The following graphs show the results of closed-loop experiments using PID control. The control input is weave amplitude, with voltage V=12 V, current I=120 A, travel speed v =1mm/s, and weave frequency f =1Hz. Welding is started without weaving, then weaving is used to increase the temperature by 150 °K. Because weaving at 1 Hz. is slower than the time constant of the heat diffusion, the oscillations in the temperature due to torch weaving are visible in the plot. The gains used in Fig. 5.10 are: (a) K_p=0.1, and K_i=0.5, and (b) K_p=0.01, and K_i=0.05.

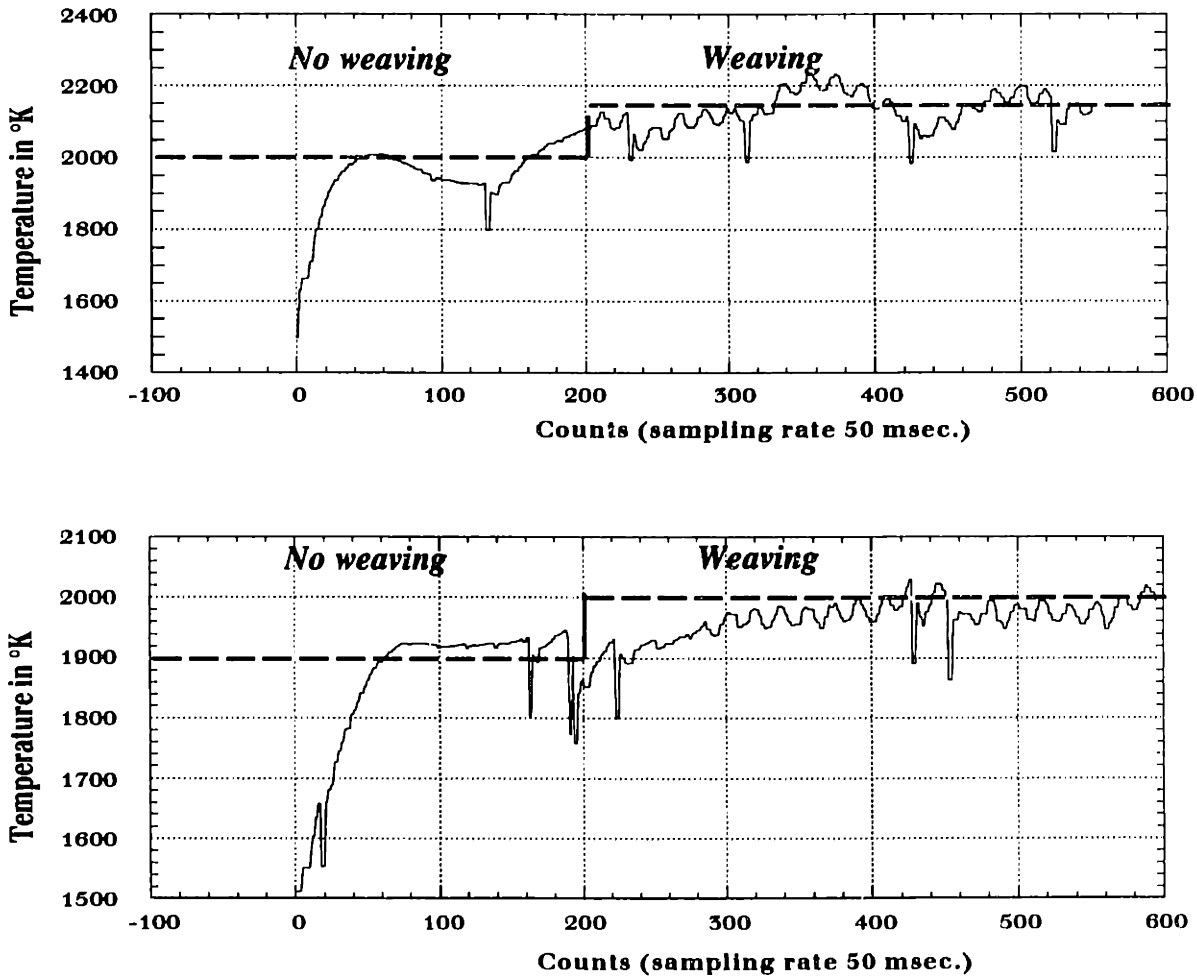


Fig. 5.10: Closed-loop temperature control using weave amplitude (V=2V, I=120A, v=1mm/s, and f= 1 Hz.)

Figure 5.11- 5.12 show the results of closed loop experiments similar to Fig. 5.10 but with a weaving frequency of 3 Hz. and a travel speed of 3 mm/s. Weaving results in an increase of 200 °K in the temperature in approximately 7.5 seconds. The gains used in Fig. 5.11 are the same as those used in Fig. 5.10 (b).

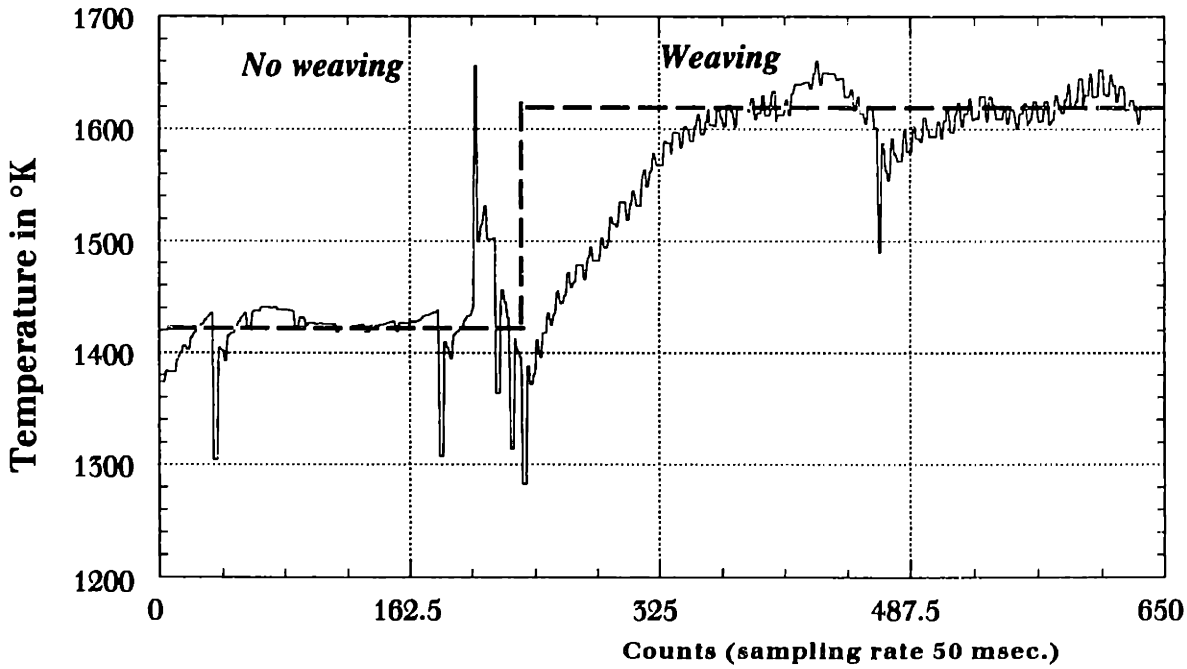


Fig. 5.11: Closed-loop temperature control using weave amplitude (V=2V, I=120A, v=3 mm/s, and f = 3 Hz.)

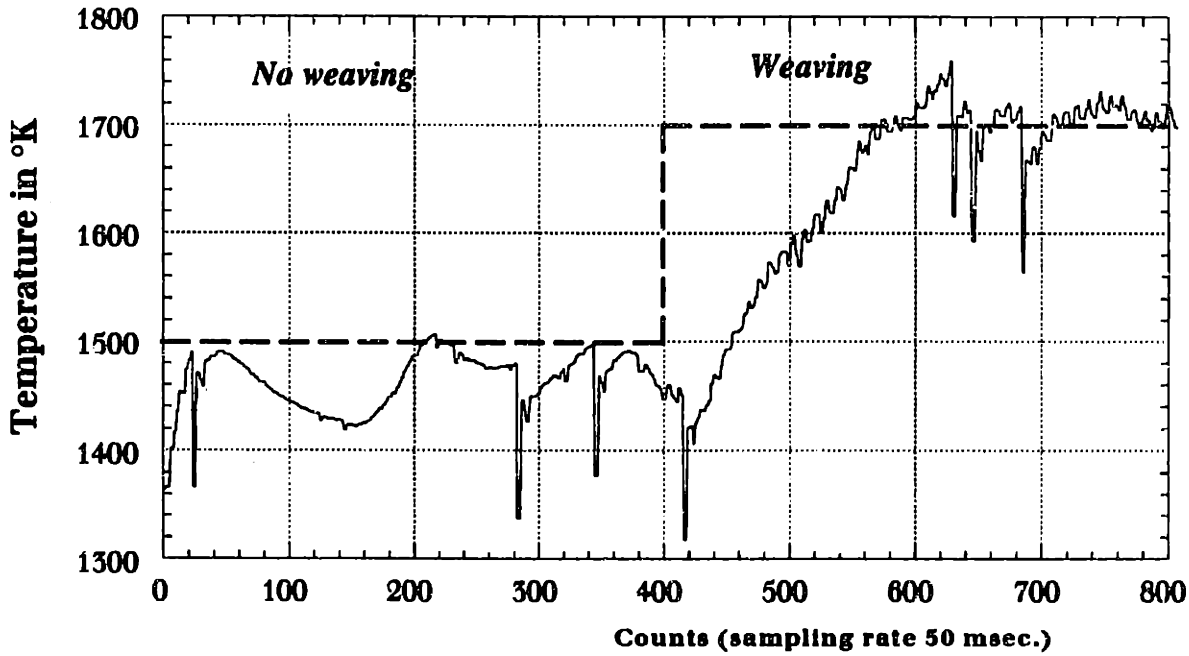


Fig. 5.12: Closed-loop temperature control using weave amplitude (V=2V, I=120A, v=2 mm/s, and f = 3 Hz.)

As seen in Fig. 5.11-5.12, weave amplitude can be used to control temperature at a specified width in a SISO control system. The time constant of the response of Fig. 5.11 is 4 seconds with a travel speed of 3 mm/s. The time constant increases when the speed is decrease, as shown in Fig. 5.12, because the IR sensor is positioned at a distance x_s = 2mm behind the torch.

The experimental response seen in Fig. 5.11 and 5.12 compares very well with the simulation results of Fig. 4.33 which has a time constant of 3.5 seconds for the same welding conditions. This confirms the validity of the dynamic model used in the simulations and control design stage in chapter 4.

5.4 Vision System:

The pool geometry is measured using a CCD camera that sends a real-time digitized image of the weld to the PC. The computer performs image analysis to estimate the pool width, and provide that estimate to the control algorithm. There are 3 issues that are pertinent to the vision system used:

1- The contrast between the light emitted by the molten pool and the background light was not sufficiently large to use a constant threshold for finding the edge of the pool. Thus an elaborate algorithm was developed to calculate the threshold from information about the light intensity of the background which was not always constant due to the radiation noise from the arc.

2- The camera cannot look directly at the arc because the light intensity is too high. This means that we can measure the pool width only at some distance (3 or 4 mm) behind the torch which introduces a measurement delay. This measurement delay puts a strong limitation on the bandwidth of the control system. As a result, the controller was run at 2 Hz. because a faster cycle would create instability in the loop.

3- The maximum width of the pool, as well as the general shape of the temperature isotherms, depend on the travel speed of the torch. The light intensity inside the molten pool also changes with travel speed. Therefore, the location at which the pool width is measured must be changed as a function of travel speed.

A sophisticated software algorithm (see Appendix D) was developed to perform the image analysis and calculate the pool width. The algorithm changes the threshold value for finding the edge according to the background light. It also searches in different sections depending on the travel speed. It searches for the edges of the molten pool in a specified area close to the previous edge location since the edge position cannot change too quickly in one time step (sampling rate of 0.5 sec.). This guarantees less variation in the pool width measurement due to noise.

Figure 5.13 shows pool width measurement obtained in open-loop after a step change in travel speed (from 3 mm/s to 1 mm/s) that occurs at t =11 sec .

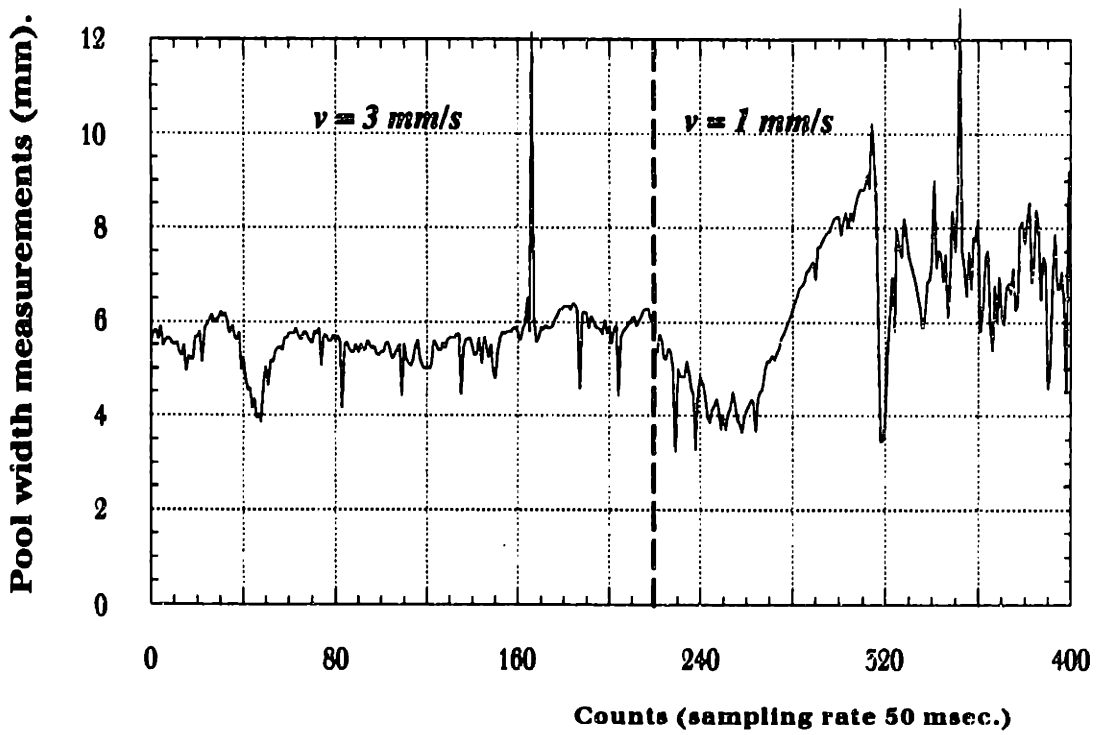
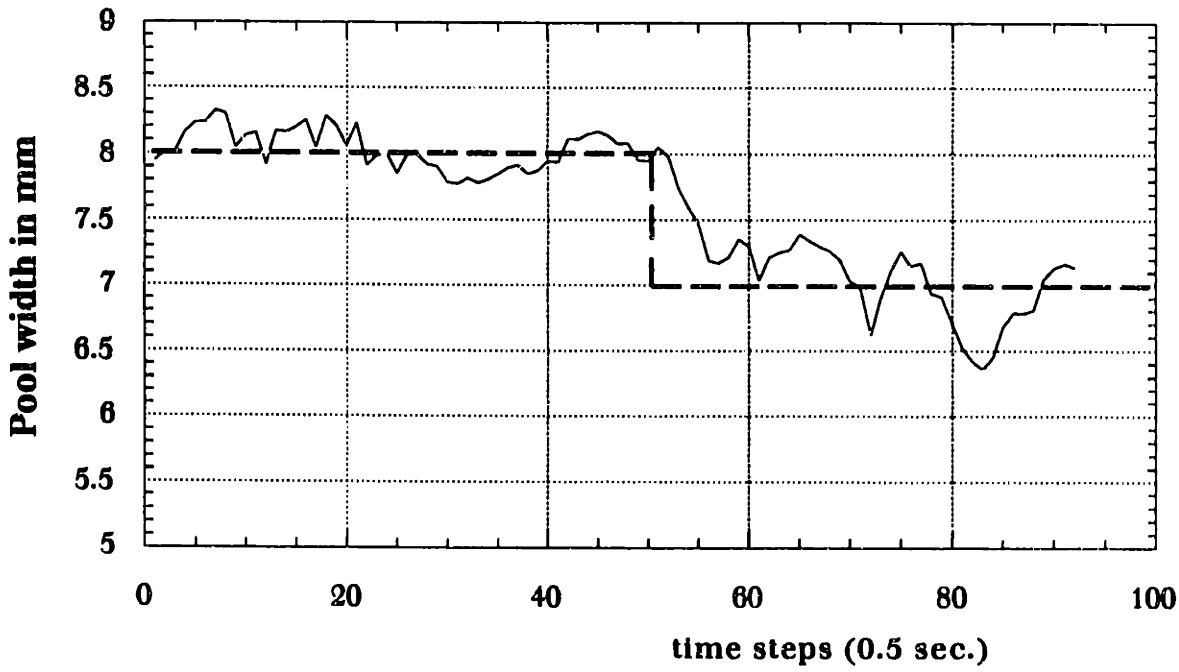


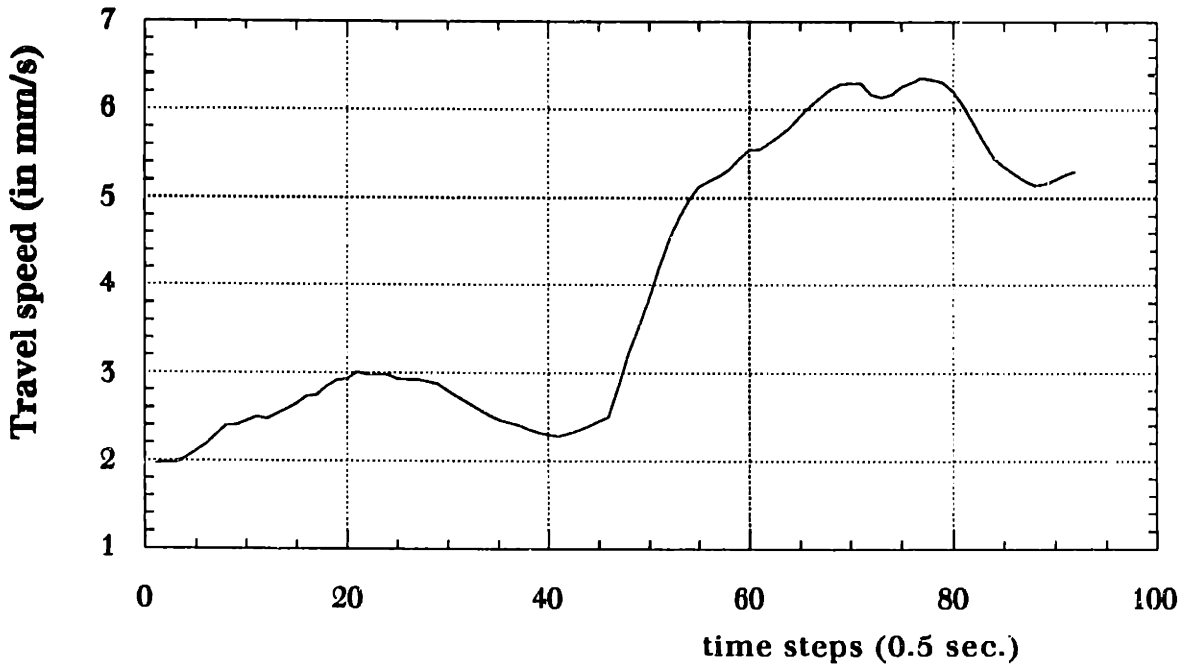
Fig. 5.13: Open-loop pool width measurements with step change in travel speed at t =11 sec.

5.4.2. Closed-Loop Pool Width Control:

Figure 5.14 shows the results of closed loop PID control of pool width using torch travel speed as a control input. The desired pool width changes from 8 mm to 7 mm at t=25 sec. Note that at higher travel speeds, the weld pool width measurement becomes more noisy. Fig. 5.14(b) shows the command input, travel speed, that was used to achieve the PID control of Fig. 5.14(a). The gains used are K_p=0.1 and K_i=0.05. This result is similar to the simulation results shown in Fig. 4.19. The arc voltage was 12 V, the arc current was 120A, and no weaving was used in this experiment. Because the algorithm used to analyze the picture frame and estimate the pool width takes more than 0.2 seconds, the sampling time used in these experiments is 0.5 seconds.



(a)



(b)

Fig. 5.14: Closed-loop PID Control of pool width using travel speed (a) Pool width, (b) Command input.

5.5 Conclusions

This chapter presented some important and relevant issues for implementing closed loop control of arc welding. The most critical issue is the lack of accurate and dependable sensors due to the extremely harsh environment created by arc welding (high temperatures and radiation noise). For instance, measuring weld pool width by vision system is difficult due to low contrast between the weld pool electromagnetic radiation and the background radiation. Additionally, we had to measure the pool width at some distance (about 2 or 3 millimeters) behind the torch because of the high radiation of the arc. This creates a measurement delay of about 3 seconds which puts a severe restriction on the controller bandwidth and stability.

Similarly, Infra-Red temperature measurements poses several difficulties because of the high noise emitted by the arc. A state-of-the-art Accufiber 100 two-color pyrometer was used for temperature measurements and provided accurate and reliable temperature measurements. This sensor is able to measure temperature, without too much noise, at close proximity to the arc (2 or 3 mm). Interrupt-Driven Serial port communications was used to speed up the data acquisition process and free the computer for other computations. Yet, it was found that the PC was too slow to do simultaneous control of pool width and HAZ width. Single-input single-output PID Control was implemented and shown to achieve satisfactory performance.

MIMO control of pool width and HAZ width was not implemented because the computer was not fast enough to do all the calculations necessary for output measurements (temperature and pool width measurements and filtering) and perform real-time control of weaving and travel speed. The image analysis algorithm alone required 0.2 seconds to estimate the pool width using a 33 Mhz 386 PC. This was not enough to implement Time Delay Control which requires a fast sample time (less than 0.1 sec.). To implement multivariable TDC control, either a faster computer must be used or a separate DSP board must used for the control of weaving mechanism. In any case, the PID closed-loop control experiments, presented in this chapter, showed the feasibility of using high-frequency weaving to independently control the pool width and the HAZ width during GTAW welding.

Chapter 6

Conclusions

Real-time control of arc welding has been an area of active research during the last two decades. There has been many achievements in the area of understanding the basic phenomena associated with arc welding and how they effect the final mechanical and thermal properties of the weld joint. Much work has been conducted in the area of correlating process inputs and parameters with the desired properties of the final weld joint. Even though arc welding has progressed from an 'art' to a science, there are still unresolved issues such as maintaining a consistent quality during automated welding. The difficulty lies in the intrinsic nonlinearity of the process as well as the lack of accurate and noise-free sensors. In addition, previous work has shown that the process outputs, in particular geometry and thermal properties of the weld joint, are highly coupled which precludes real multivariable control. This coupling between the outputs also makes the process sensitive to disturbances which tend to shift the range of reachability.

In order to implement multivariable control of arc welding that is capable of maintaining on-line quality control, one must deal with the multivariable, nonlinear, and highly coupled nature of the arc welding process. No control method can overcome these inherent process limitations, and therefore re-design and modifications of the process must be used to increase its controllability.

In this thesis, research on modeling and control of the multi-input multi-output arc welding process was presented. Decoupling of the process outputs was achieved by using high-frequency torch weaving. It was shown, both analytically and experimentally, that high-frequency weaving reduces the coupling between the pool width and the HAZ width, and thus allows implementation of 'real' multivariable control systems

using travel speed and weave amplitude as control inputs. Basically, high-frequency weaving provides greater control over the heat distribution inside the workpiece, and therefore can be thought of as a 'variable distribution heat source .'

A nonlinear dynamic model of heat conduction during welding was presented. The model includes latent heat of fusion, heat losses due to convection and radiation, and temperature-dependent thermal coefficients, and has been shown to capture most of the nonlinear dynamic behavior of arc welding. This simplified dynamic is therefore extremely useful for dynamic simulations and control system design.

GTAW experiments confirm the simulation results and show that decoupling between the pool width and the *HAZ* can be achieved by using high-frequency weaving. It was shown that the amplitude of the weave motion has a large effect on process output decoupling, while the frequency of weaving changes the shape of the molten weld pool. From these open-loop experimental results, it was shown that:

- Travel speed should be used to control the pool width,
- Weave amplitude should be used to control the HAZ width,
- Both the input current and the weave frequency are kept constant during closed-loop control.

Closed-loop control simulations were performed with PID and Time Delay Control (TDC) control systems. The TDC controller was shown to exhibit excellent transient response and robustness to model uncertainties and disturbances. The method is based on a simple algorithm which uses current and past information of the system inputs and outputs, and estimates about their rates of change, to estimate the plant dynamics.

The hardware used to conduct GTAW experiments consist of a high-precision high-frequency weaving servomechanism, an infra-red pyrometer with a scanning mirror for temperature measurement, and a vision system for pool width measurement. A line scanning mirror allows the measurement of peak temperatures at a specified width, but does not allow the measurement of pool width or HAZ width because the position of the peak temperatures is not stationary. Closed-loop control experiments were performed using SISO PID control. The weave amplitude was used to control the HAZ width, and travel speed was used to control the pool width. PID control experiments show that PID can provide satisfactory performance if the gains are tuned properly and if different gains are used in different operating conditions.

Chapter 7

References

Agapakis, J., Somers, T., and Masubuchi, K., *"Annotated Bibliography on Welding Automation and Robotics with Emphasis on Shipbuilding Applications"*, MIT Report, December 20, 1983.

Anonymous, *Weld Width Indicates Weld Strength*, NASA Tech Briefs, 6(3), 1981, MFS-25648.

Bates, B.E., and Hardt D.E., *"A Real-Time Calibrated Thermal Model for Closed-Loop Weld Bead Geometry Control"*, ASME Journal of Dynamic Systems, Measurement, and Control, 107, 1985, 25-33.

Boughton, P., Rider, G., and Smith, C.J, *Feedback Control of Weld Penetration in 1978, Advances in Welding Processes*, Welding Institute International Conference, 1978, 203-215.

Bradstreet, B.J., *Effect of Welding Conditions on Cooling Rate and Hardness in the Heat-Affected Zone*, Welding Journal, 48(11), 1969, 499s-504s.

Brown, D.C., Crossley, F.A., Rudy, J.F., and Schwartzbart, H., *The Effect of Electromagnetic Stirring and Mechanical Vibration on Arc Welds*, Welding Journal, 41(6), 1962, 241s-250s.

Carlson, N.M., and Johnson, J.A., *Ultrasonic Detection of Weld Bead Geometry, Proceedings of Review of Progress in Quantitative NDE*, LaJolla, California, August 3-8, 1986.

Carslaw, H. S., and Jaeger, J.C., *"Conduction of Heat in Solid"*, Oxford University Press, 1959, pp. 255-267.

Chen, X. Q., and Lucas, J.; *A fast vision system for Control of Narrow Gap TIG Welding;*

International Conference on Advances in Joining and Cutting Processes; Harrogate, U.K.; pp:480-488, 1990

Christensen, N., Davies, V., and Gjermundsen, K., *"The Distribution of Temperature in Arc Welding"*, British Welding Journal, 12(2), 1965, pp.54-75

Dornfeld, D. A., Tomizuka, M., and Langari, G., *Modelling and Adaptive Control of Arc Welding Processes, Measurement and Control for Batch Manufacturing*, ASME, New York, New York, 1982.

Doumanidis, C.C., *A Review of Arc Welding Models*, LMP Publication, 1986.

Doumanidis, C.C., and Hardt, D.E., *"A Model for In-Process Control of Thermal Properties during Welding"* ASME Journal of Dynamic Systems, Measurement, and Control, Mar, 1989.

Doumanidis, C.C., *"Modelling and Control of Thermal Phenomena in Welding"*, Ph.D Thesis, Dept. of Mechanical Engg, MIT, February 1988.

Essers, W.G., and Walter, R., *Heat Transfer and Penetration Mechanisms with GMA and Plasma-GMA Welding*, Welding Journal, 60(2), 1981, 37s-42s.

Glickstein, S.S, and Yeniscavich, W., *A Review of Minor Element Effects on the Welding Arc and Weld Penetration*, Welding Research Council Bulletin, 226, May 1977, 1-18.

Grong, Ø., Christensen, N.; *Effects of weaving on temperature distribution in fusion welding*; Materials Science and Technology; vol.2, pp:967-973, Sept. 1986

Grosh, R.J., and Trabant, E.A., *"Temperature Distribution insolids of Variable Thermal Properties Heated by Moving Heat Sources"*, Quarterly Appl. Mech., 13(2), 1955, pp. 161-167.

Hale, M., Doumanidis, C., and Hardt, D.E., *Multivariable Control of Arc Welding Processes*, International Trends in Welding Research, ASM, Metals Park, Ohio, May 18-22, 1986.

Hale, M.B. *"Multivariable Dynamic Modeling and Control of GMAW Pool Geometry"*, Ph. D.Thesis, Department of ME, MIT, 1989.

Hale, M.B., and Hardt, D.E., *"Multivariable Geometry Control of Welding - Part I: Process*

Modeling" Symposium on Manufacturing Process Modeling and Control, ASME, Dec, 1990 a.

Hale, M.B., and Hardt, D.E., "*Multivariable Geometry Control of Welding - Part II: Process Control*" Symposium on Manufacturing Process Modeling and Control, ASME, Dec, 1990 b.

Hardt, D.E., Doumanidis, C., and Hale, M., *Non-Linearities in Welding System Dynamics*, Proceedings of the Fifth Symposium on Energy Engineering Sciences, June 17-19, 1987, 184-197.

Hardt, D.E., Garlow, D.A., and Weinert, J.B., "*A Model of Full Penetration Arc-Welding for Control System Design*", ASME Journal of Dynamic Systems, Measurement, and Control, 107, 1985, 40-46.

Hardt, D.E., and Katz, J.M., "*Ultrasonic Measurement of Weld Penetration*", Welding Journal, Vol. 63, No. 9, 1984, 273s - 281s

Hibbitt, H.D., and Marcal, P.V., "*A Numerical Thermo-Mechanical Model for the Welding and Subsequent Loading of a Fabricated Structure*", Computers and Structures, 3(5), 1973, 1145-1174.

Hughes, R. V., and Welduck, R. P.; *Electromagnetic Arc Path Control in Robot Plasma Welding;* Robotic Welding, IFS Publications. pp:243-263, 1987.

Hunter, J.J., Bryce, G.W., Doherty, J., *On-Line Control of the Arc Welding Process*, Developments in Mechanised Automated and Robotic Welding, Welding Institute International Conference, November 1980, paper 37.

Jackson C.E., and Shrubsall, A.E., *Control of Penetration and Melting Ratio with Welding Technique*, Welding Journal, 32(4), 1953, 172s-178s.

Jackson, C.E., and Shrubsall, A.E., *Energy Distribution in Electric Welding*, Welding Journal, 29(5), 1950, 231s-241s.

Khan, M.A., Madsen, N.H., Gooding, J.S., and Chin, B.A., "*Infrared Thermography as a Control for the Welding Process*", Optical Engineering, Vol. 25, No. 6, June 1986, 799 -805

Kim, Y-S., *Metal Transfer in Gas Metal Arc Welding*, PhD Thesis, Massachusetts Institute of Technology, 1989.

L'vov, M.S., and Igoshin, A.P., *A System for the Automation of Arc Welding with Self Adjustment by Depth of Penetration*, Automatic Welding, 24(5), 1971, 42-43.

Lawrence, B.D., and Jackson, C.E., *Variable Frequency Gas Shielded Pulsed Current Arc Welding*, Welding Journal, 48(3), 1969, 97s-104s.

Lesnewich, A., *Control of Melting Rate and Metal Transfer in Gas-Shielded Metal Arc Welding*, Welding Journal, 37(8), 1958, 343s-353s and 418s-425s.

Lund, Richard Alan, *Multivariable Computer Control of a Gas Metal Arc Welding Process*, PhD Thesis, University of Wisconsin-Madison, 1977.

Makara, A. M.; and Kushnirenko, B. N.; *Weaving the arc as a factor in improving the structure and properties of welded joints*; Automatic Welding; vol. 20, pp:35-40, Jan. 1967

Malmuth N.D., Hall W. F., Davis, B.I., and Rosen, C.D., *Transient Thermal Phenomena and Weld Geometry in GTAW*, Welding Journal, 53(9), 1974, 388s-400s.

Marquardt, D.W., *"An Algorithm for Least-Squares Estimation of Nonlinear Parameters"*, Journal of the Society for Industrial and Applied Mathematics, Vol. 11, No. 2, June 1963, 431-441.

McGlone, J.C., *"Weld Bead Geometry Prediction - A Review"*, Metal Construction, 14(9), 1982, 378-384.

Mills, G.S., *Use of Emission Spectroscopy for Welding Arc Analysis*, Welding Journal, 56(3), 1977, 93s-96s.

Moody, W.V., *Automatic Welding Control Using a State Variable Model*, Master's Thesis, Massachusetts Institute of Technology, Department of Ocean Engineering, 1979.

Myers, P.S., Uyehara, O.A., and Borman, G.L., *Fundamentals of Heat Flow in Welding*, Welding Research Council Bulletin, 123, July 1967 1-46.

Needham, J.C., Cooksey, C.J., and Milner D.R. , *Metal Transfer in Inert-gas Shielded-Arc Welding*, British Welding Journal, February 1960, 101-114.

Nomura, H., Satoh, Yukihiko, Tohno, K., Satoh, Yoshikazu, and Kurotori, M, *"Arc Light Intensity Controls Current in SA Welding System"*, Welding and Metal Fabrication, September, 1980, pp. 457 - 463

Nunes A.C., *"An Extended Rosenthal Model"*, Welding Journal, Vol. 62, No. 6, June, 1983.pp 165s-170s.

Peterson, D.W., and Ransom, P.L., *The Calibration and Cataloging of Spectral Emissions from Gas Metal Arc Welding of Steel from 4100A to 7450A*, Radio Research Laboratory Publication No. 522, University of Illinois, January 1983.

Potap'evskii, A.G., Lapchinskii, V.F., and Buchinskii, V.N., *Transfer of Electrode Metal in Argon-Shielded Pulsed Arc Welding*, Automatic Welding, 18(6), 1965 18-22.

Press, W. H., et al. *"Numerical Recipes in C: The Art of Scientific Computing"*, Cambridge University Press, 1989, pp. 574-580.

Richardson, R. W., Gutow, D. A., and Rao, S. H., *A Vision Based System for Arc Weld Pool Size Control*, Measurement and Control for Batch Manufacturing, ASME, New York, New York, 1982, 65-74.

Richardson, R.D., Gutow, R.A., Anderson, R.A., and Farson, D.F., *"Coaxial Arc Weld Pool Viewing for Process Monitoring and Control"*, Welding Journal , Vol. 63, No. 3, March 1984, pp43-50.

Roberts, D.K, and Wells, A.A., *A Mathematical Examination of the Effect of Bounding Planes on the Temperature Distribution due to Welding*, British Welding Journal, (1), 1954, 553-5609.

Rosenthal, D., and Schmerber, R., *"Thermal Study of Arc Welding Experimental Verification of Theoretical Formulas"*, Welding Journal, 17(4), 1938, 2s-8s.

Rosenthal, D., *"Mathematical Theory of Heat Distribution During Welding and Cutting"*, Welding Journal, Vol. 20, No. 5, 1941, 220s-234s

Rosenthal, D., *"The Theory of Moving Sources of Heat and Its Application to Metal Treatments"*, Transaction of the ASME, November 1946, pp. 849-866

Schellhase, M., and Weinschenk, H.E., *Dynamic Behaviour of Consumable Metal Arcs, Arc*

Physics and Weld Pool Behavior, Welding Institute International Conference, May 1979, 59-66.

Shinoda, T., and Doherty, J., "*The Relationship Between Arc Welding Parameters and Weld Bead Geometry - A Literature Survey*", Welding Institute Report, 74/1978/PE, October 1978.

Shtrikman, M. M., and Pavlov, A. S.; *Solidification of welds made with a slit gap and transverse weaving of the electrode* (translation), Automatic welding; vol. 35, pp: 53-56, June 1983.

Shultz, B.L. and Jackson, C.E., *Influence of Weld Bead Area on Weld Metal Mechanical Properties*, Welding Journal, 52(1), 1973, 26s-37s.

Siewert, T.A., Trevisan, R.E., Purtscher, P.T.; *The Effect of Electrode Weave Procedure on HY-80 GMA Welds*; Welding Journal; vol. 66, pp:203-209, July 1987.

Somers, T., "*Control of GTA Weld Penetration Based on Infrared, Temperature Measure*", S.M. Thesis, Department of Mechanical Engineering, MIT, January 1986.

Song, J.B., *Multivariable Adaptive Control in GMA Welding Using a Thermally Based Depth Estimator*, PhD Thesis, Massachusetts Institute of Technology, February 1992.

Suzuki, A., Hardt, D.E., and Valvani, L. "*Application of Adaptive Control Theory to On line GTA Weld Geometry Regulation*", ASME J. of Dynamic Systems Measurement and Control, 1991.

Tam, A., and Hardt, D.E., "*Weld Pool Impedance for Pool Geometry Measurement: Stationary and Non-Stationary Pools*", ASME Journal of Dynamic Systems, Measurement, and Control, Dec, 1989.

Thorn, K., Feenstra, M., Young, J.C., Lawson, W.H.S., and Kerr, H.,W., *The Interaction of Process Variables - Their Influence on Weld Dimensions in GMA Welds on Steel Plate*, Metal Construction, 14(3), 1982, 128-133.

Touloukian, Y. S., "*Thermophysical Properties of High Temperature Solid Materials*", MacMillan Co., 1967, pp. 587-590.

Tsai, N., "*Heat Distribution and Weld Bead Geometry in Arc Welding*", Ph.D. Thesis, MIT,

Department of Materials Science and Engineering, April 1983

Tsao, K.C., and Wu C.S., *"Fluid Flow and Heat Transfer in GMA Weld Pools"*, Welding Journal, 67(3), 1988, 70s-75s.

Vroman A.R., Brandt H., *"Feedback Control of GTA Welding Using Puddle Width Measurement"*, Welding J., Sept. 1978, pp. 742-746

Wells, A.A., *"Heat Flow in Welding"*, Welding Journal, 31(5), 1952, 263s-267s.

Willgoss, R.A., *Mathematical Model Predicts Equilibrium*, Welding and Metal Fabrication, 52(9), 1984, 340-351.

Wilson, J.L., Claussen, G.E., and Jackson, C.E., *The Effect of IR Heating on Electrode Melting Rate*, Welding Journal, 35(1), 1956, 1s-8s.

Yahata, H., and Kamo, K.; *Automation of pipe welding for steam electrical generating power plant;* Welding International; vol.1, pp: 73-79, 1987.

Youcef-Toumi, K., and Ito, O.; *"A Time Delay Controller for Systems with Unknown Dynamics"*; ASME Journal of Dynamic Systems, Measurement and Control; vol.112, No.1,pp: 133-142, March 1990.

Youcef-Toumi, K., and Wu, S.T.; *"Input-Output Linearization using Time Delay Control"*; ASME Journal of Dynamic Systems, Measurement and Control; vol.114, No.3, March 1991.

Zacharia, T., Eraslan, A.H. and Aidun, D.K., *"Modeling of Autogenous Welding"*, Welding Journal, 67(3), 1988, 53s-62s.

Zacharia, T., Eraslan, A.H. and Aidun, D.K., *"Modeling of Non-Autogenous Welding"*, Welding Journal, 67(1), 1988, 18s-27s.

Zachsenhouse, M., and Hardt, D., *Weld Pool Impedance Identification for Size Measurement and Control*, ASME Journal of Dynamic Systems, Measurement, and Control, 105, 1983, 77-88.

Özisik, M. N., *"Heat Conduction"*, Wiley-Interscience publication, 1980, pp. 217-221.

Chapter 8

Appendices

Appendix A

Derivation of the Incremental Solution of the heat flow in a semi-infinite plate.

1- Homogeneous Solution:

In this appendix, the one-dimensional homogeneous solution for heat conduction in an infinite medium is derived using the separation of variables method [8]. The 3-Dimensional solution (eq. 10) can be derived in similar fashion. The homogeneous differential equation for heat conduction in one direction is:

$$\frac{\partial T(x,t)}{\partial t} = \alpha\left(\frac{\partial^2 T(x,t)}{\partial x^2}\right) \qquad (A.1)$$

By separating the variables in the form $T(x,t) = X(x)\,\Gamma(t)$, the solution for the function $\Gamma(t)$ is given as:

$$\Gamma(t) = e^{-\alpha\beta^2 t} \qquad (A.2)$$

and the function $X(x)$ satisfies the equation

$$\frac{d^2X(x)}{dx^2} + \beta^2 X(x) = 0 \qquad (A.3)$$

Two linearly independent solutions of this equation are $\cos(bx)$ and $\sin(bx)$ corresponding to each value of b (the eigenvalues). The general solution of the heat-

conduction problem is obtained by the superposition of $X(x)$ and $G(t)$:

$$T(x,t)=\int_{\beta=0}^{\infty} e^{-\alpha\beta^2 t}\left[a(\beta)\cos\beta x+b(\beta)\sin\beta x\right]d\beta \qquad (A.4)$$

The initial conditions for this problem at time $t = 0$ is (for $-\infty<x<\infty$):

$$T(x,0)=\int_{\beta=0}^{\infty}\left[a(\beta)\cos\beta x+b(\beta)\sin\beta x\right]d\beta \qquad (A.5)$$

Eq. A.5 is the Fourier formula for the integral representation of an arbitrary function. The coefficients $a(b)$ and $b(b)$ are then the Fourier coefficients:

$$a(\beta)=\frac{1}{\pi}\int_{-\infty}^{\infty} T(x',0)\cos\beta x'\,dx' \qquad (A.6-a)$$

$$b(\beta)=\frac{1}{\pi}\int_{-\infty}^{\infty} T(x',0)\sin\beta x'\,dx' \qquad (A.6-b)$$

When equations A.6-a and A.6-b are substituted into eq. A.5, we get:

$$T(x,0)=\int_{\beta=0}^{\infty}\left[\frac{1}{\pi}\int_{x'=-\infty}^{\infty} T(x',0)\cos\beta(x-x')\,dx'\right]d\beta \qquad (A.7)$$

Substituting A.5 and A.7 into A.4, we obtain:

$$T(x,t)=\frac{1}{\pi}\int_{\beta=0}^{\infty} e^{-\alpha\beta^2 t}\int_{x'=-\infty}^{\infty} T(x',0)\cos\beta(x-x')\,dx'\,d\beta \qquad (A.8)$$

Using the integral equation:

$$\int_{\beta=0}^{\infty} e^{-\alpha\beta^2 t}\cos\beta(x-x')\,d\beta=\sqrt{\frac{\pi}{4\alpha t}}\,exp\left[-\frac{(x-x')^2}{4\alpha t}\right]$$

This solution becomes:

$$T(x,t)=\frac{1}{\sqrt{4\pi\alpha t}}\int_{x'=-\infty}^{\infty}T(x',0)exp\left[-\frac{(x-x')^2}{4\alpha t}\right]dx' \qquad (A.9)$$

2- Non-Homogeneous Solution:

Eq. A.9 is the homogeneous solution (with no heat addition). To derive the non-homogeneous solution, we include the heat generation in the differential equation, and use the Green's function. The mathematical formulation of the non-homogeneous case is:

$$\frac{\partial T(x,t)}{\partial t}=\frac{Q(x,t)}{\rho c}+\alpha\left(\frac{\partial^2 T(x,t)}{\partial x^2}\right) \qquad (A.10)$$

The solution eq. A.9 can be written in terms of Green's function:

$$T(x,t)=\int_{-\infty}^{\infty}G(x,t|x',\tau)\Big|_{\tau=0}T(x,o)dx' \qquad (A.11)$$

By comparing eq. A.9 and A.11, the Green's function is:

$$G(x,t|x',\tau)\Big|_{\tau=0}=\frac{1}{\sqrt{4\pi\alpha t}}exp\left[-\frac{(x-x')^2}{4\alpha t}\right] \qquad (A.12)$$

The Green's function notation convention is:

$$G(x,t|x',\tau)\equiv G(effect|impulse) \qquad (A.13)$$

The general form of the Green's function $G(x,t \mid x',t)$ can be obtained from $G(x,t \mid x',t)_{\mid t=0}$ merely by replacing t by $(t-t)$ in the latter [8]. Thus, the desired Green's

function solution is:

$$G(x,t|x',\tau)=\frac{1}{\sqrt{4\pi\alpha(t-\tau)}}\,exp\left[-\frac{(x-x')^2}{4\alpha(t-\tau)}\right] \qquad (A.14)$$

Therefore, the overall solution to the heat conduction problem is obtained by summing the homogenous and the non-homogeneous solutions:

$$T(x,t)=\frac{1}{\sqrt{4\pi\alpha t}}\int_{x'=-\infty}^{\infty}T(x',0)\,exp\left[-\frac{(x-x')^2}{4\alpha t}\right]dx' + \frac{1}{\rho c}\int_{\tau=0}^{t}\frac{1}{\sqrt{4\pi\alpha(t-\tau)}}\int_{x'=-\infty}^{\infty}Q(x',\tau)\,exp\left[-\frac{(x-x')^2}{4\alpha(t-\tau)}\right]dx'\,d\tau \qquad (A.15)$$

Equation A.15 is the general solution for heat conduction in an infinite medium in the presence on non-uniform initial conditions and heat generation. In the case of uniform initial condition, the homogeneous solution becomes zero, and we are left with the non-homogeneous solution. For a point source, this becomes:

$$T(x,t)=\frac{1}{\rho c}\int_{\tau=0}^{t}\frac{Q(\tau)}{\sqrt{4\pi\alpha(t-\tau)}}\,exp\left[-\frac{(x-x')^2}{4\alpha(t-\tau)}\right]d\tau \qquad (A.16)$$

In the case of 3-D heat flow, this solution becomes:

$$T(x,t)=\frac{1}{\rho c}\int_{\tau=0}^{t}\frac{Q(\tau)}{\left(4\pi\alpha(t-\tau)\right)^{3/2}}\,exp\left[-\frac{(x-x')^2+(y-y')^2+(z-z')^2}{4\alpha(t-\tau)}\right]d\tau \qquad (A.17)$$

which is the solution for 3-D heat conduction in infinite medium with a point source $Q(t)$ at (x',y',z').

Appendix B

MATLAB Subroutines

Open-Loop Simulation

```
init_temp=300; t1=0;
deltax=0.5; deswidth=3.0;
Npoints=20; x=-0.5:deltax:Npoints*deltax;
T=init_temp*ones(1,Npoints); T_old=T;
time=[]; temperature=[];
vel=2; amp=0; q=3000;
heat=q*(0.75 - (0.07*(3*deswidth + 3))^3);
while t1<19
if t1>6
vel=5;
end
if t1>9
amp=3;
end
if t1>12
vel=2;
end
deltat=deltax/vel;
T(Npoints)=temp(0,deltat,init_temp,x(Npoints),deswidth,heat,vel,amp);
for i=Npoints-1:-1:1
        T_old(i)=T(i);
        T(i)=temp(0,deltat,T_old(i+1),x(i),deswidth,heat,vel,amp);
 end
temperature=[temperature; T(1)];
t1=t1+deltat;
time=[time; t1];
end
plot(time,temperature)
clear mex;
```

SISO PID Control

```
init_temp=300; t1=0; des_T=1000; integral=0;
v_old=0; max_acc=5; index=0;
P_gain=1/2000; P_deriv=1/2000; P_int=1/4000;
deltax=0.2; deswidth=3.0; rand('normal'); integral_v=[];
Npoints=20; x=-0.5:deltax:Npoints*deltax;
T=init_temp*ones(1,Npoints); T_old=T;
time=[]; temperature=[]; velocity=[];
vel=2; amp=0; q=3000;
heat=q*(0.75 - (0.07*(3*deswidth + 3))^3);
while t1<49.8
if t1>10
if index==0
        des_T=1500;
        integral=0;
        index=1;
 end
end
if t1>20
if index==1
        des_T=1000;
        integral=0;
        index=2;
 end
end
if t1>30
if index==2
        des_T=2000;
        integral=0;
        index=3;
 end
end
if t1>40
if index==3
        des_T=1000;
```

```
        integral=0;
        index=4;
 end
end
deltat=deltax/vel;
T(Npoints)=temp(0,deltat,init_temp,x(Npoints),deswidth,heat,vel,amp);
for i=Npoints-1:-1:1
        T_old(i)=T(i);
        T(i)=temp(0,deltat,T_old(i+1),x(i),deswidth,heat,vel,amp);
 end
temperature=[temperature; T(1)];
t1=t1+deltat;
time=[time; t1];
T(1) = T(1) + 30*rand;                          %add measurement noise
 T(1) = (T(1) + T_old(1))/2;                    % Low pass filter
deriv=(T(1)-T_old(1))/deltat;
integral=integral +(T(1)-des_T)*deltat;
vel = vel +P_gain*(T(1)-des_T) +P_deriv*deriv+P_int*integral;
integral_v=[integral_v; integral];
vel = (vel + v_old)/2;

%%%%%%%%%% Check for vel & accel saturation
if ((vel-v_old)/deltat) > max_acc
vel = v_old + max_acc*deltat;
 end
if ((vel-v_old)/deltat) < -max_acc
        vel = v_old - max_acc*deltat;
 end
if vel>7
        vel=7;
 end
if vel<1
        vel=1;
 end
v_old=vel;
velocity=[velocity; vel];
end
plot(time,temperature)
clear mex;
```

SISO TDC Control

```
%%%%%%%%%%%%%%% Setting up initial values and inputs
init_temp=300; t1=0; des_T=1000; integral=0; v_old=0; max_acc=5;
k1=9; k2=4.2; br=3000; alpha=0.2;
deltax=0.2; deswidth=3.0; rand('normal');
Npoints=20; x=-0.5:deltax:Npoints*deltax; deriv_old=0; deriv2_old=0;
T=init_temp*ones(1,Npoints); T_old=T;
time=[]; temperature=[]; velocity=[];
vel=2; amp=0; q=3000;
heat=q*(0.75 - (0.07*(3*deswidth + 3))^3);


%%%%%%%%%%%%%%%%%%%%%%%% Desired dynamics
num=[2]; numd=[2 0]; den=[1 2];
t_des=0.1:0.1:30;
u=[1000*ones(100,1); 1500*ones(100,1); 1000*ones(100,1)];
[y_des,x_des]=lsim(num,den,u,t_des);
[yd_des,x_des]=lsim(numd,den,u,t_des);
[y2d_des]=(yd_des-[0; yd_des(1:299,1)])/0.1;


%%%%%%%%%%%%%%%%%%%%%%%% Start Simulations
while t1<29.8
if t1>10
des_T=1500;
end
if t1>20
des_T=1000;
end
deltat=deltax/vel;
T(Npoints)=temp(0,deltat,init_temp,x(Npoints),deswidth,heat,vel,amp);
for i=Npoints-1:-1:1
        T_old(i)=T(i);
        T(i)=temp(0,deltat,T_old(i+1),x(i),deswidth,heat,vel,amp);
 end
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%Control loop
temperature=[temperature; T(1)];
t1=t1+deltat;
time=[time; t1];
 T(1) = T(1) + 30*rand;                          %add measurement noise
 T(1) = (T(1) + T_old(1))/2;                          % Low pass filter
deriv=(T(1)-T_old(1))/deltat;
deriv=(deriv+deriv_old)/2;                          % Low pass filter
deriv2=(deriv-deriv_old)/deltat;
deriv2=(deriv2+deriv2_old)/2;                                % Low pass filter
deriv_old=deriv;
deriv2_old=deriv2;
j=floor(10*t1);
exo_input=y2d_des(j)+k2*(yd_des(j)-deriv)+k1*(y_des(j)-T(1));
vel = (br*v_old + deriv2 - exo_input)/br;
vel = (alpha*vel + v_old)/(1+alpha);

%%%%%%%%%%%%%%%%%%%% Check for vel & accel saturation
if ((vel-v_old)/deltat) > max_acc
vel = v_old + max_acc*deltat;
  end
if ((vel-v_old)/deltat) < -max_acc
          vel = v_old - max_acc*deltat;
  end
if vel>7
          vel=7;
  end
if vel<1
          vel=1;
  end
v_old=vel;
velocity=[velocity; vel];
end
plot(time,temperature)
clear mex;
```

Multivariable TDC Control

```
%%%%%%%%%%%%%%%%% Setting up initial values and inputs
init_temp=300; t1=0; v_old=0; max_acc=5; amp_old=0;
k1=9; k2=4.2; br=4000; alpha=0.1;
ak1=25; ak2=7.07; abr=4000; aalpha=0.1; aderiv_old=0; aderiv2_old=0;
deltax=0.2; deswidth=2.7; adeswidth=3.9; rand('normal');
Npoints=20; x=-0.5:deltax:Npoints*deltax; deriv_old=0; deriv2_old=0;
T=init_temp*ones(1,Npoints); T_old=T;
aT=init_temp*ones(1,Npoints); aT_old=aT;
time=[]; temperature=[]; velocity=[]; amplitude=[];
vel=2; amp=0; q=1300;
heat=q*(0.9 - (0.15*deswidth + 0.3)^3);
aheat=q*(0.8 - (0.11*adeswidth + 0.2)^2);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Desired dynamics
num=[2]; numd=[2 0]; den=[1 2];
t_des=0.1:0.1:50;
u=[1800*ones(100,1); 1800*ones(100,1); 1800*ones(100,1);
1800*ones(100,1); 1800*ones(100,1)];
[y_des,x_des]=lsim(num,den,u,t_des);
[yd_des,x_des]=lsim(numd,den,u,t_des);
[y2d_des]=(yd_des-[0; yd_des(1:499,1)])/0.1;
u2=[1000*ones(100,1); 1300*ones(100,1); 1000*ones(100,1);
1200*ones(100,1); 1000*ones(100,1)];
[ay_des,x_des]=lsim(num,den,u2,t_des);
[ayd_des,x_des]=lsim(numd,den,u2,t_des);
[ay2d_des]=(ayd_des-[0; ayd_des(1:499,1)])/0.1;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% StartSimulations
while t1<49.9
deltat=deltax/vel;
T(Npoints)=temp(0,deltat,init_temp,x(Npoints),deswidth,heat,vel,amp);
for i=Npoints-1:-1:1
        T_old(i)=T(i);
        T(i)=temp(0,deltat,T_old(i+1),x(i),deswidth,heat,vel,amp);
```

```
 end
aT(Npoints)=temp(0,deltat,init_temp,x(Npoints),adeswidth,aheat,vel,amp);
for i=Npoints-1:-1:1
        aT_old(i)=aT(i);
        aT(i)=temp(0,deltat,aT_old(i+1),x(i),adeswidth,aheat,vel,amp);
 end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%Control loop
temperature=[temperature; [T(1),aT(1)]];
t1=t1+deltat;
time=[time; t1];
 T(1) = T(1) + 30*rand;                          %add measurement
noise
 T(1) = (T(1) + T_old(1))/2;                     % Low pass filter
deriv=(T(1)-T_old(1))/deltat;
deriv=(deriv+deriv_old)/2;                  % Low pass filter
deriv2=(deriv-deriv_old)/deltat;
deriv2=(deriv2+deriv2_old)/2;               % Low pass filter
deriv_old=deriv;
deriv2_old=deriv2;

 aT(1) = aT(1) + 30*rand;                        %add measurement
noise
 aT(1) = (aT(1) + aT_old(1))/2;                      % Low pass filter
aderiv=(aT(1)-aT_old(1))/deltat;
aderiv=(aderiv+aderiv_old)/2;               % Low pass filter
aderiv2=(aderiv-aderiv_old)/deltat;
aderiv2=(aderiv2+aderiv2_old)/2;            % Low pass filter
aderiv_old=aderiv;
aderiv2_old=aderiv2;

j=floor(10*t1);
exo_input=y2d_des(j)+k2*(yd_des(j)-deriv)+k1*(y_des(j)-T(1));
vel = (br*v_old + deriv2 - exo_input)/br;
vel = (alpha*vel + v_old)/(1+alpha);

aexo_input=ay2d_des(j)+ak2*(ayd_des(j)-aderiv)+ak1*(ay_des(j)-aT(1));
amp = (abr*amp_old - aderiv2 + aexo_input)/abr;
amp = (aalpha*amp + amp_old)/(1+aalpha);

%%%%%%%%%%%%%% Check for vel & accel saturation
if ((vel-v_old)/deltat) > max_acc
```

```
vel = v_old + max_acc*deltat;
 end
if ((vel-v_old)/deltat) < -max_acc
        vel = v_old - max_acc*deltat;
 end
if vel>7
        vel=7;
 end
if vel<1
        vel=1;
 end
v_old=vel;
velocity=[velocity; vel];
if amp>8
        amp=8;
 end
if amp<0
        amp=0;
 end
amp_old=amp;
amplitude=[amplitude; amp];
end
plot(time,temperature)
clear mex;
```

Robustness

```
%%%%%%%%%%%%%%%% Setting up initial values and inputs
init_temp=300; t1=0; v_old=0; max_acc=5; amp_old=0;
k1=9; k2=4.2; br=4000; alpha=0.1;
ak1=25; ak2=7.07; abr=4000; aalpha=0.1; aderiv_old=0; aderiv2_old=0;
deltax=0.2; deswidth=2.7; adeswidth=3.9; rand('normal');
Npoints=20; x=-0.5:deltax:Npoints*deltax; deriv_old=0; deriv2_old=0;
T=init_temp*ones(1,Npoints); T_old=T;
aT=init_temp*ones(1,Npoints); aT_old=aT;
time=[]; temperature=[]; velocity=[]; amplitude=[];
vel=2; amp=0; q=1300;
heat=q*(0.9 - (0.15*deswidth + 0.3)^3);
aheat=q*(0.8 - (0.11*adeswidth + 0.2)^2);
k_percent = 1;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Desired dynamics
num=[2]; numd=[2 0]; den=[1 2];
t_des=0.1:0.1:50;
u=[1800*ones(100,1); 1800*ones(100,1); 1800*ones(100,1);
1800*ones(100,1); 1800*ones(100,1)];
[y_des,x_des]=lsim(num,den,u,t_des);
[yd_des,x_des]=lsim(numd,den,u,t_des);
[y2d_des]=(yd_des-[0; yd_des(1:499,1)])/0.1;
u2=[1000*ones(100,1); 1000*ones(100,1); 1000*ones(100,1);
1000*ones(100,1); 1000*ones(100,1)];
[ay_des,x_des]=lsim(num,den,u2,t_des);
[ayd_des,x_des]=lsim(numd,den,u2,t_des);
[ay2d_des]=(ayd_des-[0; ayd_des(1:499,1)])/0.1;
rand('uniform');

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Start Simulations
while t1<49.9
if t1>10
        k_percent = 0.5;
end
if t1>20
        k_percent = 2.0;
```

```
    end
    if t1>30
            k_percent = 1 + rand;
    end
    deltat=deltax/vel;
    T(Npoints)=temp(0,deltat,init_temp,x(Npoints),deswidth,heat,vel,amp,k_percent);
    for i=Npoints-1:-1:1
            T_old(i)=T(i);
            T(i)=temp(0,deltat,T_old(i+1),x(i),deswidth,heat,vel,amp,k_percent);
     end
    aT(Npoints)=temp(0,deltat,init_temp,x(Npoints),adeswidth,aheat,vel,amp,k_percent);
    for i=Npoints-1:-1:1
            aT_old(i)=aT(i);
aT(i)=temp(0,deltat,aT_old(i+1),x(i),adeswidth,aheat,vel,amp,k_percent);
     end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%Con-
trol loop
    temperature=[temperature; [T(1),aT(1)]];
    t1=t1+deltat;
    time=[time; t1];
     T(1) = T(1) + 30*rand;                          %add measurement noise
     T(1) = (T(1) + T_old(1))/2;                     % Low pass filter
    deriv=(T(1)-T_old(1))/deltat;
    deriv=(deriv+deriv_old)/2;              % Low pass filter
    deriv2=(deriv-deriv_old)/deltat;
    deriv2=(deriv2+deriv2_old)/2;           % Low pass filter
    deriv_old=deriv;
    deriv2_old=deriv2;

     aT(1) = aT(1) + 30*rand;                        %add measurement noise
     aT(1) = (aT(1) + aT_old(1))/2;                        % Low pass filter
    aderiv=(aT(1)-aT_old(1))/deltat;
    aderiv=(aderiv+aderiv_old)/2;           % Low pass filter
    aderiv2=(aderiv-aderiv_old)/deltat;
    aderiv2=(aderiv2+aderiv2_old)/2;                       % Low pass filter
    aderiv_old=aderiv;
    aderiv2_old=aderiv2;

    j=floor(10*t1);
    exo_input=y2d_des(j)+k2*(yd_des(j)-deriv)+k1*(y_des(j)-T(1));
```

```
vel = (br*v_old + deriv2 - exo_input)/br;
vel = (alpha*vel + v_old)/(1+alpha);

aexo_input=ay2d_des(j)+ak2*(ayd_des(j)-aderiv)+ak1*(ay_des(j)-aT(1));
amp = (abr*amp_old - aderiv2 + aexo_input)/abr;
amp = (aalpha*amp + amp_old)/(1+aalpha);

%%%%%%%%%%%%% Check for vel & accel saturation
if ((vel-v_old)/deltat) > max_acc
vel = v_old + max_acc*deltat;
 end
if ((vel-v_old)/deltat) < -max_acc
        vel = v_old - max_acc*deltat;
 end
if vel>7
        vel=7;
 end
if vel<1
        vel=1;
 end
v_old=vel;
velocity=[velocity; vel];
if amp>8
        amp=8;
 end
if amp<0
        amp=0;
 end
amp_old=amp;
amplitude=[amplitude; amp];
end
plot(time,temperature)
clear mex;
```

Appendix C

C Program for Open Loop Welding Simulations

This program solves the heat diffusion equation using 4th order Runga-Kutta integration

---

```
#include <math.h>
#include <stdio.h>
#include <console.h>
#include "nr.h"
#include "nrutil.h"
#include <stdlib.h>


#define N 1                                        /* N is number of differential equations  */
#define SQR(a) ((a)*(a))                  /* Defining a square function */
#define PI 3.1415926536
#define tmax 20
#define ntotal 200

extern float dtsav,*tp,**tempp;           /* defined in odeint.c  */
extern int kmax;                             /* defined in odeint.c  */
int kount=0;
float v,v0;
float t1,t2,tiny;
float x,y,z;
float  ys,sigma=0.003,xs=0.,y0,t_old=0.;
float q,q0 = 10000.0;                                  /* heat input rate in J/s */
float f;
double rho = 7800;                                     /* density in kg/m3 */

/********  double k,k_cold=30.0,k_hot=80.0;          conductivity in W/m.K */
double c,c_cold=420.0,c_hot=630.0;                                  /* specific heat in J/kg.K */
double alpha,alpha_cold=21.0e-6,alpha_hot=6.1e-6; /* thermal diffusivity in m2/s */
/*************************************************/

void derivs (t,temp,dtempdt)
float t,temp[],dtempdt[];
{
        double r_squared, delta_t;
        double c2,c3,c4,tau;                                    /* distances x,y,z are in mm
*/
```

```
        xs = xs + v*(t-t_old);
        t_old = t;
        ys = y0*sin(2.*PI*f*t);
/************ Guassian distribution ******************/
  if (t < 0.02) tiny = 0.02; else tiny =0.0;
  tau = t+tiny;

  c2 = q/((PI*rho*c)*pow(4.0*PI*alpha*tau,0.5)*((2.*alpha*tau) + pow(sigma,2.)));
  c3 = -1.0e-6*(pow((x+xs),2.)+pow((y-ys),2.))/(4*alpha*tau + 2*pow(sigma,2.));
  dtempdt[1] = c2 * exp(c3);

/*  c4 = -1.0e-6*pow((z),2.)/(4*alpha*tau);
  dtempdt[1] = c2 * exp(c3 + c4);*/
}
/*************************************************/


main()
{
        int i,nok=0,nbad=0;
        float eps,h1,hmin,*tempstart;
        char ch;

        tp=vector(1,ntotal);
        tempstart=vector(1,N);
        tempp=matrix(1,ntotal,1,N);

        eps=1.0e-1;
        h1=0.01;
        hmin=0.0001;
        kmax=1;
        dtsav=1./20.0;

        x = 0.5;
        y = 1.1;
        z = 0.0;

        q0 = 3000.;
        v = 2.0;
        y0 = 0.0;
        f = 3.0;
        tempstart[1]=300.0;
        q=q0*(0.7 - pow((0.07*(3.*y + 4.)),3));

for(i=0;i<10;i++){
        t1 = 0.1*i;
        t2 = t1+0.1;                                        /* Sampling time Ts=0.1
sec.*/

        if (tempstart[1] > 1000) {alpha=alpha_hot;c=c_hot;}
                else{
                c = 0.82*tempstart[1] + 171.4;
                alpha=(85.-0.052*tempstart[1])/(7800.*c);}
```

```
/********************** Pool Width Temperature Calculations ********/
                        odeint(tempstart,N,t1,t2,eps,h1,hmin,&nok,&nbad,derivs,rkqc);


printf("  ys = %2.1f \t v = %7.2f\t t2 = %4.2f\t xs = %f\t T = %f\n",ys,v,t2,xs,tempstart[1]);
}       free_matrix(tempp,1,ntotal,1,N);
        free_vector(tp,1,ntotal);
        free_vector(tempstart,1,N);
        printf("\n   **   Good Bye   **\n");
}

/*****************************************************************************/

#define MAXSTP 10000
#define TINY 1.0e-30

int kmax=1000;                                          /* if kmax = 0 do not save inter results */
float *tp=0,**tempp=0,dtsav=0.0;                        /* save at dtsav intervals  */
extern kount;

void odeint(tempstart,nvar,t1,t2,eps,h1,hmin,nok,nbad,derivs,rkqc)
float tempstart[],t1,t2,eps,h1,hmin;
int nvar,*nok,*nbad;
void (*derivs)( float,float *,float *);
void (*rkqc)(float *,float *,int,float *,float,float,float *,float *,float *,void(*)());
{
        int nstp,i,tick=1;
        float tsav,t,hnext,hdid,h;
        float *tempscal,*temp,*dtempdt,*vector();
        void nrerror();
        void free_vector();

        tempscal=vector(1,nvar);
        temp=vector(1,nvar);
        dtempdt=vector(1,nvar);
        t=t1;
        h=(t2 >t1) ? fabs(h1): -fabs(h1);
        for (i=1;i<=nvar;i++) temp[i]=tempstart[i];
        if (kmax > 0) tsav=t-dtsav*2.0;                         /* storage of first step */
        for (nstp=1;nstp<=MAXSTP;nstp++) {                      /* take at most MAXSTP steps */
                (*derivs)(t,temp,dtempdt);
                for (i=1;i<=nvar;i++)
                        tempscal[i]=fabs(temp[i])+fabs(dtempdt[i]*h)+TINY;  /* scaling to monitor
accuracy */
                if (kmax > 1) {
                        if (fabs(t-tsav) > fabs(dtsav)) {               /* store intermediate results */
                                if (kount < kmax-1) {
                                        tp[++kount]=t;
                                        tempp[kount][1]=temp[1];
                                        tsav=t;
                                }
                        }
                }
```

```
                if ((t+h-t2)*(t+h-t1) > 0.0) h=t2-t;            /* if step overshoots, cut down */
                (*rkqc)(temp,dtempdt,nvar,&t,h,eps,tempscal,&hdid,&hnext,derivs);
                if (hdid == h) ++(*nok); else ++(*nbad);
                if ((t-t2)*(t2-t1) >= 0.0) {                            /* are we done? */
                        for (i=1;i<=nvar;i++) tempstart[i]=temp[i];
                        if ((kmax==1) && (t2==(kount+1))){
                                tp[++kount]=t;                                          /* save final
step */
                                tempp[kount][1]=temp[1];
                        }
                        free_vector(dtempdt,1,nvar);
                        free_vector(temp,1,nvar);
                        free_vector(tempscal,1,nvar);
                        return;
                }
                if (fabs(hnext) <= hmin) nrerror("Step size too small in ODEINT");
                h=hnext;
        }
        nrerror("Too many steps in routine ODEINT");
}
/*************************************************/
#define PGROW -0.20
#define PSHRNK -0.25
#define FCOR 0.06666666                 /*1.0/15.0*/
#define SAFETY 0.9
#define ERRCON 6.0e-4

void rkqc(temp,dtempdt,n,t,htry,eps,tempscal,hdid,hnext,derivs)
float temp[],dtempdt[],*t,htry,eps,tempscal[],*hdid,*hnext;
void (*derivs)(float,float*,float *);
int n;

{
        int i;
        float tsav,hh,h,tempor,errmax;
        float *dtempsav,*tempsav,*temptemp,*vector();
        void rk4(),nrerror(),free_vector();

        dtempsav=vector(1,n);
        tempsav=vector(1,n);
        temptemp=vector(1,n);
        tsav=(*t);                                                      /* save initial values */
        for (i=1;i<=n;i++) {
                tempsav[i]=temp[i];
                dtempsav[i]=dtempdt[i];
        }
        h=htry;                                                         /* set step size to initial value */
        for(;;) {
                hh=0.5*h;                                               /* take two half steps */
                rk4(tempsav,dtempsav,n,tsav,hh,temptemp,derivs);
                *t=tsav+hh;
                (*derivs)(*t,temptemp,dtempdt);
                rk4(temptemp,dtempdt,n,*t,hh,temp,derivs);
                *t=tsav+h;
```

```
                if (*t == tsav) nrerror("Step size too small in routine RKQC");
                rk4(tempsav,dtempsav,n,tsav,h,temptemp,derivs);          /* take the large step */
                errmax=0.0;
                for (i=1;i<=n;i++) {
                        temptemp[i]=temp[i]-temptemp[i];                 /* temptemp now contains the
error estimate */
                        tempor=fabs(temptemp[i]/tempscal[i]);
                        if (errmax < tempor) errmax=tempor;
                }
                errmax/=eps;                                             /* scale relative to required
tolerance */
                if (errmax<=1.0) {
                        *hdid=h;
                        *hnext=(errmax > ERRCON ?
                                SAFETY*h*exp(PGROW*log(errmax)) : 4.0*h);
                        break;
                }
                h=SAFETY*h*exp(PSHRNK*log(errmax));                      /* truncation error too large

reduce step size  */
        }
        for (i=1;i<=n;i++) temp[i] += temptemp[i] * FCOR;        /* Mop up fifth order truncation error
*/
        free_vector(temptemp,1,n);
        free_vector(dtempsav,1,n);
        free_vector(tempsav,1,n);
}
/*************************************************/
void rk4(temp,dtempdt,n,t,h,tempout,derivs)
float temp[],dtempdt[],t,h,tempout[];
void (*derivs)(float,float *,float *);
int n;

{
        int i;
        float th,hh,h6,*dtempm,*dtempt,*tempt,*vector();
        void free_vector();

        dtempm=vector(1,n);
        dtempt=vector(1,n);
        tempt=vector(1,n);
        hh=h*0.5;
        h6=h/6.0;
        th=t+hh;
        for (i=1;i<=n;i++) tempt[i]=temp[i]+hh*dtempdt[i];       /* first step */
        (*derivs)(th,tempt,dtempt);                                                  /* second
step */
        for (i=1;i<=n;i++) tempt[i]=temp[i]+hh*dtempt[i];
        (*derivs)(th,tempt,dtempm);
        for (i=1;i<=n;i++)  {
                tempt[i]=temp[i]+h*dtempm[i];
                dtempm[i]+=dtempt[i];
        }
        (*derivs)(t+h,tempt,dtempt);                                                 /*fourth step
```

```
*/
        for (i=1;i<=n;i++)                                  /* accumulate increments with proper
weights */
                tempout[i]=temp[i]+h6*(dtempdt[i]+dtempt[i]+2.0*dtempm[i]);
        free_vector(tempt,1,n);
        free_vector(dtempt,1,n);
        free_vector(dtempm,1,n);
}
/*************************************************/
void nrerror(error_text)
char error_text[];
{
        void exit();

        fprintf(stderr,"Numerical Recipes run-time error...\n");
        fprintf(stderr,"%s\n",error_text);
        fprintf(stderr,"...now exiting to system...\n");
        exit(1);
}



float *vector(nl,nh)
int nl,nh;
{
        float *v;

        v=(float *) malloc((unsigned) (nh-nl+1)*sizeof(float));
        if (!v) nrerror("allocation failure in vector()");
        return v-nl;
}

int *ivector(nl,nh)
int nl,nh;
{
        int *v;

        v=(int *)malloc((unsigned) (nh-nl+1)*sizeof(int));
        if (!v) nrerror("allocation failure in ivector()");
        return v-nl;
}

double *dvector(nl,nh)
int nl,nh;
{
        double *v;

        v=(double *)malloc((unsigned) (nh-nl+1)*sizeof(double));
        if (!v) nrerror("allocation failure in dvector()");
        return v-nl;
}
```

```
float **matrix(nrl,nrh,ncl,nch)
int nrl,nrh,ncl,nch;
{
        int i;
        float **m;

        m=(float **) malloc((unsigned) (nrh-nrl+1)*sizeof(float*));
        if (!m) nrerror("allocation failure 1 in matrix()");
        m -= nrl;

        for(i=nrl;i<=nrh;i++) {
                m[i]=(float *) malloc((unsigned) (nch-ncl+1)*sizeof(float));
                if (!m[i]) nrerror("allocation failure 2 in matrix()");
                m[i] -= ncl;
        }
        return m;
}

double **dmatrix(nrl,nrh,ncl,nch)
int nrl,nrh,ncl,nch;
{
        int i;
        double **m;

        m=(double **) malloc((unsigned) (nrh-nrl+1)*sizeof(double*));
        if (!m) nrerror("allocation failure 1 in dmatrix()");
        m -= nrl;

        for(i=nrl;i<=nrh;i++) {
                m[i]=(double *) malloc((unsigned) (nch-ncl+1)*sizeof(double));
                if (!m[i]) nrerror("allocation failure 2 in dmatrix()");
                m[i] -= ncl;
        }
        return m;
}

int **imatrix(nrl,nrh,ncl,nch)
int nrl,nrh,ncl,nch;
{
        int i,**m;

        m=(int **)malloc((unsigned) (nrh-nrl+1)*sizeof(int*));
        if (!m) nrerror("allocation failure 1 in imatrix()");
        m -= nrl;

        for(i=nrl;i<=nrh;i++) {
                m[i]=(int *)malloc((unsigned) (nch-ncl+1)*sizeof(int));
                if (!m[i]) nrerror("allocation failure 2 in imatrix()");
                m[i] -= ncl;
        }
        return m;
}
```

```
float **submatrix(a,oldrl,oldrh,oldcl,oldch,newrl,newcl)
float **a;
int oldrl,oldrh,oldcl,oldch,newrl,newcl;
{
        int i,j;
        float **m;

        m=(float **) malloc((unsigned) (oldrh-oldrl+1)*sizeof(float*));
        if (!m) nrerror("allocation failure in submatrix()");
        m -= newrl;

        for(i=oldrl,j=newrl;i<=oldrh;i++,j++) m[j]=a[i]+oldcl-newcl;

        return m;
}



void free_vector(v,nl,nh)
float *v;
int nl,nh;
{
        free((char*) (v+nl));
}

void free_ivector(v,nl,nh)
int *v,nl,nh;
{
        free((char*) (v+nl));
}

void free_dvector(v,nl,nh)
double *v;
int nl,nh;
{
        free((char*) (v+nl));
}



void free_matrix(m,nrl,nrh,ncl,nch)
float **m;
int nrl,nrh,ncl,nch;
{
        int i;

        for(i=nrh;i>=nrl;i--) free((char*) (m[i]+ncl));
        free((char*) (m+nrl));
}

void free_dmatrix(m,nrl,nrh,ncl,nch)
double **m;
int nrl,nrh,ncl,nch;
```

```
{
        int i;

        for(i=nrh;i>=nrl;i--) free((char*) (m[i]+ncl));
        free((char*) (m+nrl));
}

void free_imatrix(m,nrl,nrh,ncl,nch)
int **m;
int nrl,nrh,ncl,nch;
{
        int i;

        for(i=nrh;i>=nrl;i--) free((char*) (m[i]+ncl));
        free((char*) (m+nrl));
}



void free_submatrix(b,nrl,nrh,ncl,nch)
float **b;
int nrl,nrh,ncl,nch;
{
        free((char*) (b+nrl));
}



float **convert_matrix(a,nrl,nrh,ncl,nch)
float *a;
int nrl,nrh,ncl,nch;
{
        int i,j,nrow,ncol;
        float **m;

        nrow=nrh-nrl+1;
        ncol=nch-ncl+1;
        m = (float **) malloc((unsigned) (nrow)*sizeof(float*));
        if (!m) nrerror("allocation failure in convert_matrix()");
        m -= nrl;
        for(i=0,j=nrl;i<=nrow-1;i++,j++) m[j]=a+ncol*i-ncl;
        return m;
}



void free_convert_matrix(b,nrl,nrh,ncl,nch)
float **b;
int nrl,nrh,ncl,nch;
{
        free((char*) (b+nrl));
}
```

Appendix D

C Program for MATLAB Dynamic Model MEX Function

This program implements the welding dynamic model in a Matlab MEX function

```
/*****************************************************************************
 WELD.C
 The calling syntax is:

                    [temp] = temp(initt,deltat,inittemp,x,y,heat,vel,amp,k)

 Radhouan Masmoudi  April 30, 1992
*****************************************************************************/

#include <math.h>
#include "cmex.h"

#ifndef DOUBLE
#ifdef THINK_C /* THINK C doubles are extended, ints are short */
#define DOUBLE          short double    /* need true double for MATLAB matrices */
#define INT             long                    /* need long for user_fcn() */
#else                   /* everyone else does the normal thing */
#define DOUBLE          double
#define INT             int
#endif
#endif

/* Input Arguments */

#define  INT_IN         prhs[0]         /* Initial Time t in seconds   */
#define  DELT_IN        prhs[1]         /* Time step  dt in seconds    */
#define  INTEMP_IN      prhs[2]         /* Initial Temperature  in •K*/
#define  X_IN           prhs[3]         /* position x of the point of interest in mm*/
#define  Y_IN           prhs[4]         /* position y of the point of interest in mm*/
#define  HEAT_IN        prhs[5]         /* Heat input in J/s   */
#define  VEL_IN         prhs[6]         /* Travel speed in mm/s   */
#define  AMP_IN         prhs[7]         /* Weave amplitude in mm   */
#define  K_IN           prhs[8]         /* Disturbance factor in thermal diffusivity
                                           for robustness analysis   */


/* Output Arguments */

#define  TEMP_OUT       plhs[0]
```

```
#define max(A, B)       ((A) > (B) ? (A) : (B))
#define min(A, B)       ((A) < (B) ? (A) : (B))


user_fcn(nlhs, plhs, nrhs, prhs)
INT nlhs, nrhs;
Matrix *plhs[], *prhs[];
{
        DOUBLE          *temp;
        DOUBLE          *initt,*deltat,*inittemp,*x,*y,*heat,*vel,*amp,*k;


        /* Check for proper number of arguments */

        if (nrhs != 9) {
                mex_error("TEMP requires 9 input arguments.");
        } else if (nlhs != 1) {
                mex_error("TEMP requires 1 output argument.");
        }

        /* Create a matrix for the return argument */

        TEMP_OUT = create_matrix(1, 1, REAL);

        /* Assign pointers to the various parameters */

        temp = TEMP_OUT->pr;

        initt = INT_IN->pr;
        deltat = DELT_IN->pr;
        inittemp = INTEMP_IN->pr;
        x = X_IN->pr;
        y = Y_IN->pr;
        heat = HEAT_IN->pr;
        vel = VEL_IN->pr;
        amp = AMP_IN->pr;
        k = K_IN->pr;


        /* Do the actual computations in a subroutine */

        weld(temp,initt,deltat,inittemp,x,y,heat,vel,amp,k);
}


#include <stdio.h>
#include "nr.h"
#include "nrutil.h"
#define N 1                                     /* N is number of differential equations */
#define SQR(a) ((a)*(a))                /* Defining a square function */
#define PI 3.1415926536
#define tmax 20
```

```
#define ntotal 200

extern float dtsav,*tp,**tempp;          /* defined in odeint.c  */
extern int kmax;                         /* defined in odeint.c  */
int kount=0;
float v,v0;
float t1,t2,tiny;
float x,y,z;
float  ys,sigma=0.003,xs=0.,y0,t_old=0.;
float q;                                 /* heat input rate in J/s */
float f;
double rho = 7800;                                   /* density in kg/m3 */

/********  double k,k_cold=30.0,k_hot=80.0;           conductivity in W/m.K */
double c,c_cold=420.0,c_hot=630.0;                              /* specific heat in J/kg.K */
double alpha,alpha_cold=21.0e-6,alpha_hot=6.1e-6; /* thermal diffusivity in m2/s */
/*************************************************/

void derivs (t,temp,dtempdt)
float t,temp[],dtempdt[];
{
        double r_squared, delta_t;
        double c2,c3,c4,tau;                                    /* distances x,y,z are in mm */

        xs = xs + v*(t-t_old);
        t_old = t;
        ys = y0*sin(2.*PI*f*t);
/************* Guassian distribution ******************/
  if (t < 0.02) tiny = 0.02; else tiny =0.0;
  tau = t+tiny;

  c2 = q/((PI*rho*c)*pow(4.0*PI*alpha*tau,0.5)*((2.*alpha*tau) + pow(sigma,2.)));
  c3 = -1.0e-6*(pow((x+xs),2.)+pow((y-ys),2.))/(4*alpha*tau + 2*pow(sigma,2.));
  dtempdt[1] = c2 * exp(c3);

/*  c4 = -1.0e-6*pow((z),2.)/(4*alpha*tau);
  dtempdt[1] = c2 * exp(c3 + c4);*/
}
/*************************************************/


weld(temp,initt,deltat,inittemp,xp,yp,heat,vel,amp,k)
DOUBLE      temp[];
DOUBLE      initt[],deltat[],inittemp[],xp[],yp[],heat[],vel[],amp[],k[];
{
        int nok=0,nbad=0;
        float eps,h1,hmin,*tempstart;
        double k_percent;
        char ch;

        tp=vector(1,ntotal);
        tempstart=vector(1,N);
        tempp=matrix(1,ntotal,1,N);
```

```
        eps=1.0e-1;
        h1=0.01;
        hmin=0.0001;
        kmax=1;
        dtsav=1./20.0;

        x = xp[0];
        y = yp[0];
        z = 0.0;

        q = heat[0];
        v = vel[0];
        y0 = amp[0];
        k_percent = k[0];      /* k[0] is the percent error in the thermal conductivity */
        f = 3.0;

        tempstart[1]=inittemp[0];
        if (tempstart[1] > 1000) {alpha = k_percent * alpha_hot;c=c_hot;}
                else{
                c = 0.82*tempstart[1] + 171.4;
                alpha = k_percent * (85.-0.052*tempstart[1])/(7800.*c);}

        t1 = initt[0];
        t2 = t1+deltat[0];                                          /* Sampling time Ts=0.1
sec.*/


/********************** Pool Width Temperature Calculations ********/
                        odeint(tempstart,N,t1,t2,eps,h1,hmin,&nok,&nbad,derivs,rkqc);
        xs=0;
        temp[0]=tempstart[1];
/*printf("  ys = %2.4f \t v = %7.2f\t t2 = %4.2f\t xs = %f\t T = %f\r",ys,v,t2,xs,tempstart[1]);          */
        cfree_matrix(tempp,1,ntotal,1,N);
        free_vector(tp,1,ntotal);
        free_vector(tempstart,1,N);
}
```

Appendix E

C Program for Real-time Welding Control

This program implements the following:

- Takes temperature measurements (interrupt driven I/O)
- Image processing for pool width measurements
- Controls travel speed, and weave motion

```
/* ************************************************************************************************
  File: mimo.c
  By:   Radhouan Masmoudi
  Date: 7/31/92
  Description
   * Performs PID Control of pool width and HAZ width
   * Computes HAZ temp using gtemp.c, and pool width using get_width().
************************************************************************************************ */
#include <stdio.h>
#include <math.h>
#include <time.h>          /* declare and define time_t */
#include <stdlib.h>
#include <conio.h>
#include "pclerrs.h"
#include "pcldefs.h"
#include <asynch_1.h>
#include "support.c"


#define NORMAL 7
#define TRAVEL_SPD_CAL 102.0      /* count/(mm/sec) */
#define clrscr printf("\x1B[2J")    /* for these cursor commands to work, */
#define null 0
#define or | |
#define and &&
#define pi 3.14159265

#define cs1  10   /* y axis       */

#define total    0 /* reset all flags                        */
#define tc_brfl 187 /* reset trajectory complete and breakpoint reached */
#define breakpf 191 /* reset breakpoint reached flag              */
#define poserrf 223 /* reset position error flag                */
#define indexpf 247 /* reset index pulse flag                   */
#define trjcmpf 251 /* reset trajectory complete flag             */
```

```
#define cmderrf 253 /* reset command error flag                */

int axes,ax1,ax2,ax3;
int kpy, kiy, kdy, ily;
int base;
int tim_check,mode,incflag;
long int velmag;
long int accell;
int qlim = 18;          /*qlim = number of coordinate triplets*/
int WIDTH_CAL;
int tmin;
long int npoints;
long int freq = 3, calcdel;

int mo,br,epe,wo,ip,tc,ce;
time_t ti_weld, tf_weld;   /* time_t is declared in <time.h> */
int board=1;
long int amp;
float B[20];            /* array of y coordinates */
float const1, const2;

char *ibuffer,*obuffer;          /* Communication queues   */
int in_row = 6;                  /* Current row number of  */
int in_col = 3;                  /* the incoming messages. */
int out_row = 16;
int out_col = 3;
int in_port,out_port;

float vel, rad=1.0;
int q=1;

int serial(void);
void out_char(char ch);
void port_init1(void);
void port_init2(void);
void errmsg(char *pmsg, int code, unsigned status);

int ercode,iqsize;
unsigned status;
unsigned char ch;

extern int thresh;
void initialize_dt();
float get_temp(void);
void initialize_vision(void);
float get_width(void);

char filename[20];
FILE *outfile;
/*************************************************************/
/********************* MAIN PROGRAM ********************/
/*************************************************************/

main()
```

```
{
 clock_t ti, tf;
 int length, travel_spd_ct;
 int i,ii,k, nsteps, first_time;
 unsigned short stop = 2048;

 float sample_t = 0.5;
 long WELD_DURATION;
 float error, errort;
 float width =5.0;
 float temp = 1000.0;
 int des_width[4];
 float des_temp[4];
 float deriv, integral=0;
 float old_w=0, v_old=0, old_temp=1000, rad_old=0, alpha=1;
 float Pt_gain,P_gain, P_deriv, P_int;
 float deltat=0, t_weld=0, temp_diff;

 void reset_intr();
 void sync_start(),stop_motor_fast(),follow_path(),wait_pos(),
 load_vel(),load_PID(),update_traj(),resetlm628(),set_base(),check_command();
 void waitkey(),is_it_busy();

  printf("Enter name of file to save temperature measurements:\n");
  gets(filename);
  if ((outfile = fopen(filename,"w")) == NULL){
        printf("Error opening file : %s\n",filename);
        exit(0);
        }

set_base();

                incflag = 0;     /* all position commands absolute */
                tim_check = 0;  /* not performing a timing check now */
                old_temp = 1800.0;     /* to reduce initial error */
                rad_old = 10.0;
                old_w = 6.0;     /* to reduce initial error */
                v_old = 2.0;
                const1=20;
                const2=0;
                qlim=9;
                velmag=1000000;

initialize_dt();       /* Initialize the DT-2801 board */
initialize_vision();       /* Initialize the DT-2851 board */

          axes = cs1;
          ax2 = 1;
                resetlm628(cs1);
                load_PID(cs1);
                load_vel();
                update_traj(cs1,0);
                sync_start();
/*****************************************************/
```

```
        printf("*** DATA FOR DESIRED SET POINTS ***\n");
        printf(" \n Enter number of steps: ");
        scanf("%d",&nsteps);
        printf(" \n Enter total weld time (50 sec): ");
        scanf("%d",&length);
        printf(" \n **** Enter Control Gains **** ");
        printf(" \n Enter Temp Proportional Gain (P): ");
        scanf("%f",&Pt_gain);
        printf(" \n Enter Vel Proportional Gain (P): ");
        scanf("%f",&P_gain);
        printf(" \n Enter Derivative Gain (PD): ");
        scanf("%f",&P_deriv);
        printf(" \n Enter Integral Gain (PI): ");
        scanf("%f",&P_int);
        printf(" \n Enter smoothing factor (alpha): ");
        scanf("%f",&alpha);
        printf(" \n Enter sampling time (0.5 sec.): ");
        scanf("%f",&sample_t);
        printf("\n Enter Desired temperature increase (200) : ");
        scanf("%f", &temp_diff);
        printf(" \n Enter Calibration number (190 pix/10mm.): ");
        scanf("%d",&WIDTH_CAL);

for(i=0;i<nsteps;i++){
        printf("\n Enter Desired Width%d (mm) : ",i);
        scanf("%d", &des_width[i]);
   }
/*****************************************************/
        serial(); /* setup serial port and interrupts */
/*****************************************************/
/******** INITIAL STARTUP: OPEN LOOP ****************/
/*****************************************************/
/* Start to supply the shielding gas and cooling water */
 printf("\nEnter 's' to start motion");
 while ( getch() != 's'){}
 port_init2();                  /* Initialize input port */
 printf("\n  ====> Hit any key to stop welding");
        rad=1;
        amp =30;
        data_input();

        vel=2.;                 /* initial velocity */
/* Convert the physical data into the count values in D/A of RTI802. */
        travel_spd_ct = (int) (vel * TRAVEL_SPD_CAL) + 2048 ;
        dac_value(1,&travel_spd_ct);
                ti = clock();
                deltat = 0.;
          while(deltat < 8){
                get_width();
                temp = get_temp();
                tf = clock();
                deltat = (tf - ti)/1000.;
           }
                        first_time = 1;
```

```
                    q= 1;

                    old_temp = temp;
                    des_temp[0] = ceil(temp);
                    des_temp[1] = des_temp[0] + temp_diff;
/************************************ Start Closed loop control */
  for(i=0;i<nsteps;i++){
                WELD_DURATION = length/nsteps;
                t_weld = 0;
                deltat = 0.1;
                integral=0;

        while (!kbhit() && (t_weld < WELD_DURATION))
                {
                ti = clock();
                width = get_width();

                 if (first_time == 1) {
                        old_w = width;
                        deltat = 0.1;
                        first_time = 0 ;
                        des_width[0] = ceil(width);
                        des_width[1] = des_width[0] - 1;
                   }

                deltat = 0.;
          while(deltat < sample_t){
                 get_width();
                 temp = get_temp();

                        errort = temp - des_temp[i];
                        rad = rad - (Pt_gain*errort);
                        rad = (alpha*rad + rad_old)/(1+alpha);
                        if(rad>30.0) rad=30.0;
                        if(rad<0.) rad=0.;

                        amp = (long int) (rad * B[q]);

                        update_traj(cs1,amp);
                        sync_start();
                        wait_pos();

                q = q +1;
                if (q > qlim) q = 1;
                rad_old = rad;
                old_temp = temp;

                 tf = clock();
                 deltat = (tf - ti)/1000.;
                }
                printf("\t%f",deltat);
                t_weld = t_weld + deltat;      /*elapsed time */

                deriv = (width - old_w)/deltat;
```

```
                error = width - des_width[i];
                integral = integral + (error * deltat);
                vel = vel + (P_gain*error) + (P_deriv*deriv) + (P_int*integral);
                vel = (alpha*vel + v_old)/(1+alpha);
                        if(vel>5.0) vel=5.0;
                        if(vel<0.7) vel=0.7;

/* Convert velocity into count values in D/A of RTI802. */
            travel_spd_ct = (int) (vel * TRAVEL_SPD_CAL) + 2048 ;
                dac_value(1,&travel_spd_ct);

printf("\t%f\t%f\t%f \n", temp, errort, rad);
fprintf(outfile,"  T= %f  e= %f  rad= %f \n", temp, errort, rad);

printf("\t%d\t%f\t%f\t%f \n", thresh, width, error, vel);
fprintf(outfile,"  w= %f  e= %f  v= %f \n", width, error, vel);

                v_old=vel;
                old_w = width;
          }

        }
        update_traj(cs1,0);
        sync_start();
        wait_pos();
        stop_motor_fast(cs1);
        reset_intr(cs1,191);
        check_command(cs1);
        dac_value(1,&stop);       /* set travel speed to 0 */
        terminate();
/*********************************************/
  fclose(outfile);
  cacurset(0,0);              /* Just clean up, close  */
  caclear();                  /* the ports and exit.   */
  out_char(27);               /* stop stepper motor    */
  close_a1(in_port);
  close_a1(out_port);
  cacursor(0,12,13);

}   /* end of main() */
/*-------------------------------------------------------
  Function initialize_dt()
    This function initializes and sets up the DT2801 board.
  -------------------------------------------------------*/
void initialize_dt()
{
 int error, unit_id;
 unsigned short stop = 2048;

 initialize();
 reset_dt(&unit_id);
 dac_value(1,&stop);       /* set travel speed to 0 */

/* Set the D/A for software trigger with internal clock
```

```
 at the channel 1. */
 error = setup_dac(0,1);
 if(error != 0x00)
  printf("\n Setup_dac error occurred !!! ");

} /* end of initialize_dt() */
/*************************************************************************/
/*This void checks to see if data was transferred correctly.          */
/*************************************************************************/
void check_command(select)
int select;
{
   ce = 0;
   is_it_busy(select);
   if (ce == 1)
         {
            clrscr;
            gotoxy(36,1);
            printf("ATTENTION !");
            gotoxy(22,3);
            printf("The 628 is reporting a command error");
            waitkey();
         }
}
/*************************************************************************/
/* this void waits for an input from the keyboard.                     */
/*************************************************************************/
void waitkey()
{
   char ch = null;
   gotoxy(26,22);
   printf("Press any key to continue\n");
   while (!ch)
         ch = getch();
   clrscr;
}
/*************************************************************************/
/*                                                                       c*/
/*************************************************************************/
unsigned char peek(address)
int address;
{
         return(inp(address));
}
/*************************************************************************/
/*                                                                       c*/
/*************************************************************************/
void poke(address,byte)
int address;
unsigned char byte;
{
         outp(address,byte);
}
/*************************************************************************/
```

```
/* This void checks the status of the busy bit and returns values     */
/* corresponding to the RDSTAT command.                               */
/*********************************************************************/
void is_it_busy(select)
int select;
{
   int address, busy, stat;

   address = base + select;        /*set PS low to read status byte*/

   busy = 1;
   while (busy)
          {
          stat = peek(address);
          busy = (stat & 1);
          }
   br = (stat &  64) >> 6;
}
/*********************************************************************/
/* This void sends the command to the desired axis of motion.         */
/*********************************************************************/
void send_cmd(select,command)
int select;
unsigned char command;
{
   int address;
   address = base + select;
   is_it_busy(select);
   poke(address,command);
}
/*********************************************************************/
/*This void sends data to the desired axis of motion.                */
/*********************************************************************/
void send_data(select,data)
int select;
int data;
{

          int address;
          address = base + select + 1;       /*set PS high for data printf*/
          is_it_busy(select);
          poke(address,data);
          is_it_busy(select);

}
/*********************************************************************/
/*This void outputs a zero drive signal.                             */
/* SEE LTRJ COMMAND and the corresponding coontrol word.             */
/*********************************************************************/
void stop_motor_fast(select)
int select;
{
   send_cmd(select,0x1f);          /*send traj command*/
   send_data(select,2);           /*turn off motor*/
```

```
   send_data(select,0);          /*dummy send*/
   send_cmd(select,0x01);
}
/*****************************************************************/
/* These functions return the high and low words of a long integer.  c*/
/*****************************************************************/
unsigned int hi_word(LI)
long int LI;
{
        int *IP;
        IP = ( int * ) &LI;
        return(*(IP+1));
}

unsigned int lo_word(LI)
long int LI;
{
        int *IP;
        IP = ( int * ) &LI;
        return(*IP);
}

/*****************************************************************/
/* These functions return the high and low bytes of an integer.     c*/
/*****************************************************************/
unsigned char hi(I)
unsigned int I;
{
        char *CP;
        CP = ( int * ) &I;
        return(*(CP+1));
}

unsigned char lo(I)
unsigned int I;
{
        char *CP;
        CP = ( int * ) &I;
        return(*CP);

}
/*****************************************************************/
/* This void loads the filter parameters.                        */
/* SEE LFIL COMMAND                                              */
/*****************************************************************/
void load_filter(select,kp,ki,kd,il)
int select;
int kp,ki,kd,il;
{
   send_cmd(select,0x1e);     /*filter loading command          */

   send_data(select,0x0);     /*0 means 341.33 us sampling      */
   send_data(select,0xf);     /*meaning load kp,ki,kd and then il */
```

```
    send_data(select,hi(kp));   /*send MSByte of KP          */
    send_data(select,lo(kp));   /*send LSByte of KP          */

    send_data(select,hi(ki));   /*send MSByte of KI          */
    send_data(select,lo(ki));   /*send LSByte of KI          */

    send_data(select,hi(kd));   /*send MSByte of KD          */
    send_data(select,lo(kd));   /*send LSByte of KD          */

    send_data(select,hi(il));   /*send MSByte of IL          */
    send_data(select,lo(il));   /*send LSByte of IL          */

    send_cmd(select,0x04);      /*send Update Filter command  */

    check_command(select);      /*check that everything was ok */

}/*************************************************************************/
/*This void loads the trajectory control parameters.          */
/* SEE LTRJ COMMAND                                           */
/*************************************************************************/
void load_traj(select,accel,velocity,position)
int select;
long int accel;
long int velocity;
long int position;
{
    stop_motor_fast(select);

    send_cmd(select,0x1f);          /*trajectory load command   */

    send_data(select,0);            /*send hi 16 bits of traj   */
    send_data(select,42);           /*send lower 16 bits of traj */
                                    /*control word, 42 means accel,*/
                                    /*vel and position will be   */
                                    /*downloaded                 */

    send_data(select,hi(hi_word(accel)));   /*send top 8 bits of acc    */
    send_data(select,lo(hi_word(accel)));   /*send next 8 bits of acc   */
    send_data(select,hi(lo_word(accel)));   /*send next 8 bits of acc   */
    send_data(select,lo(lo_word(accel)));   /*send bottom 8 bits of acc */

    send_data(select,hi(hi_word((velocity))));/*send top 8 bits of vel    */
    send_data(select,lo(hi_word((velocity))));/*send next 8 bits of vel   */
    send_data(select,hi(lo_word((velocity))));/*send next 8 bits of vel   */
    send_data(select,lo(lo_word((velocity))));/*send bottom 8 bits of vel */

    send_data(select,hi(hi_word(position)));/*send top 8 bits of pos    */
    send_data(select,lo(hi_word(position)));/*send next 8 bits of pos   */
    send_data(select,hi(lo_word(position)));/*send next 8 bits of pos   */
    send_data(select,lo(lo_word(position)));/*send bottom 8 bits of pos */

    check_command(select);          /*check that everything was ok*/

}
```

```
/*********************************************************************/
/*This void loads the desired trajectory from the keyboard.      */
/*********************************************************************/
void load_trajectory(select)
int select;
{
   stop_motor_fast(select);
   check_command(select);
   if (!tim_check)
   {
   is_it_busy(select);
   load_traj(select,accell,velmag,0L);

   }                              /* timing check. Will be */
   else                             /* loading new ones soon */
         {                          /* anyway.          */
            is_it_busy(select);
            load_traj(select,0L,0L,0L);

         }
}
/*********************************************************************/
void load_vel()
{
   load_trajectory(cs1);
   mode = 0;
}
/*********************************************************************/
/*This void loads the PID parameters from the keyboard.         */
/*********************************************************************/
void load_PID(select)
int select;
{
   int kp, ki, kd, il;

   if (!tim_check)

   {
            kp = 300;
            ki = 200;
            kd = 100;
            il = 10;

            kpy = kp;
            kiy = ki;
            kdy = kd;
            ily = il;


   is_it_busy(select);
   load_filter(select,kp,ki,kd,il);
   }
   else
         {
```

```
            is_it_busy(select);        /* will be replaced anyway.    */
            load_filter(select,2,2,2,2);

        }
}
/*****************************************************************/
/*This void sets the position error for stopping.                */
/* SEE LPES COMMAND                                              */
/*****************************************************************/
void set_pos_error(select,pos_error)
int select,pos_error;
{
   send_cmd(select,0x1a);          /*load position error for stop*/
   is_it_busy(select);
   send_data(select,hi(pos_error));
   is_it_busy(select);
   send_data(select,lo(pos_error));

}/*****************************************************************/
/*This void resets the interrupts.                               */
/* SEE RSTI COMMAND                                              */
/*****************************************************************/
void reset_intr(select,data)
int select;
int data;
{
   is_it_busy(select);
   send_cmd(select,0x1d);       /*send reset interrupt command*/
   is_it_busy(select);
   send_data(select,255);       /*reset only selected interrupt */
   is_it_busy(select);
   send_data(select,lo(data));

/* the lower 8 bits of data should be 223 to reset position error, 0
for total reset, 187 for breakpoint and trajectory complete both.
Use 191 for breakpoint only, 247 for index pulse only, 251 for
trajectory complete, and 253 for command error interrupt */

}
/*****************************************************************/
/* This void resets the LM628.                                   */
/* SEE RESET COMMAND                                             */
/*****************************************************************/
void resetlm628(select)
int select;
{
        send_cmd(select,0);
        send_cmd(select,0x05);       /* set for 8 bits */
        send_cmd(select,0x02);       /* define_home  */
        set_pos_error(select,10000);
        reset_intr(select,0);
        send_cmd(select,3);    /*set index position*/
}
```

```
/*************************************************************************/
/* This void reads the dip switch settings from the file dip.dat       */
/* and sets the base address.                                          */
/*************************************************************************/
void set_base()
{
        int ch,dip[5],i=0;
        FILE *fptr;

  if ((fptr = fopen("dip.dat","r")) == NULL){
        printf("Cannot find file for PMC628 address :  %s\n","dip.dat");
        exit(0);
        }

        for (i=0;i<5;i++)
                {
                fscanf(fptr,"%d\n",&dip[i]);
                }
        fclose(fptr);

        /* Base represents the base address of the PMC MOTION CONTROLLER */
        /* Base is set to the desired address location by setting the   */
        /* dip switch settings (see above) to match the desired address */

        base = (0x200 + dip[0]*16 + dip[1]*32 + dip[2]*64 + dip[3]*128 + dip[4]*256);

}
/*************************************************************************/
void wait_pos()
{
   int i;
   reset_intr(cs1,191);
        for(i=0;i<calcdel;i++){}    /* do nothing loop */
   reset_intr(cs1,191);

}
/*************************************************************************/
gotoxy(x,y)
int x,y;
{
        printf("\x1B[%d;%df", y, x);
}

/*************************************************************************/
data_input()
{
   long int dia;
   double increment;
   double deg;

        freq=3;

        calcdel = (long int) (const2/freq);
        printf("\n calcdel = %ld     ", calcdel);
```

```
        accell = const1 * freq * abs(amp);
        printf("accell = %ld     ", accell);

        increment = 360/qlim;
        deg = 0;
        q = 0;
        load_trajectory(cs1);
   while (q < qlim)
        {
          q = q + 1;
          deg = deg + increment;
          B[q] = cos(deg*pi/180)*rad;
          printf(" B[%d] = %f \n",q, B[q]);
        }

}/*********************************************************************/
/* This command changes the velocity & position data & vel/pos mode    */
/*********************************************************************/
void update_traj(select,pos)
int select;
long int pos;
{
   send_cmd(select,0x1f);            /*trajectory load command    */

   send_data(select,0);        /* send upper 16 bits */

        send_data(select,2);          /*send lower 16 bits of traj  */
                                      /*control word, 10 means      */
                                      /*vel and position will be    */
                                      /*downloaded                  */

   is_it_busy(select);
   send_data(select,hi(hi_word((pos))));      /*send top 8 bits of vel     */
   send_data(select,lo(hi_word((pos))));      /*send next 8 bits of vel    */
   send_data(select,hi(lo_word((pos))));      /*send next 8 bits of vel    */
   send_data(select,lo(lo_word((pos))));      /*send bottom 8 bits of vel  */

   check_command(select);            /*check that everything was ok */
}
/*********************************************************************/
/*  This sends a start command to multiple axes at the same time.     c*/
/*********************************************************************/
void sync_start()
{
        is_it_busy(cs1);
        outp(base+axes,0x01);
}
/*********************************************************************/
serial()
{
  unsigned char scan;
  unsigned char ch;
  int ii=0;
```

```
  port_init1();                /* Initialize output port */
  printf(" TYPE: CALL SCAN  to start scanning");
  printf(" TYPE: ANY FUNCTION KEY to end communication with stepper controller");

  ii=0;
  while (ii<100)
  {
    if (kbhit())               /* A keystroke is ready.  */
          {
           if ((ch = cainkey(&scan)) != 0)  /* Standard character. */
                  {
                   ii = ii + 1;
                   cawrite(ch,1);    /* Echo the character.    */
                   out_char(ch);     /* Output to the queue    */
                   }
          else
            {
                    ii=200;
                    cabell;
           }
           }
 /*    read_ch(); */
  }
  return(0);
}
/* OUT_CHAR  Write a character to the output queue          */

void out_char(char ch)
{
   if ((ercode = wrtch_a1(out_port,ch)) != 0)
         errmsg("Cannot write to the output queue (wrtch_a1).",ercode,0);
}

/* PORT_INIT  Initialize the COM port(s).                   */
void port_init2()
{
  int ercode;
  static char *pbaud[] =
          {"110","150","300","600","1200","2400","4800","9600","19200"};
  static char *pparity[] = {"No","Odd","Even"};
  static char *pdata[]  = {"Five","Six","Seven","Eight"};
  static char *pstop[]  = {"One","Two"};

  ercode = 1;
  while (ercode)
  {
    if ((ibuffer = malloc(32384)) == NIL)
          errmsg("Cannot allocate space for input buffer.",0,0);
    ercode = open_a1(in_port,32150,128,0,0,ibuffer);
    if (ercode != 0)
          errmsg("Cannot open input port.",ercode,0);

  }
```

```
  ercode = 1;
  while (ercode)
  {
    if ((ercode = setop_a1(in_port,1,7)) != 0)
          errmsg("Cannot set BAUD rate for the input port.",ercode,0);
  }
  ercode = 1;
  while (ercode)
  {
    if ((ercode = setop_a1(in_port,2,0)) != 0)
          errmsg("Cannot set Parity for the input port.",ercode,0);
  }
  ercode = 1;
  while (ercode)
  {
    if ((ercode = setop_a1(in_port,3,3)) != 0)
          errmsg("Cannot set Data Bits for the input port.",ercode,0);
  }
  ercode = 1;
  while (ercode)
  {
    if ((ercode = setop_a1(in_port,4,0)) != 0)
          errmsg("Cannot set Stop Bits for the input port.",
                                                        ercode,0);
  }

  cacursor(1,0,0);
  cascroll(1,0,24,79,0,NORMAL);
  cacurset(1,0);
  caattrib(15,0,32,160);
  printf(" BAUD rate is %s   %s parity   %s data bits   %s stop bit(s)\n",
                              pbaud[7],pparity[0],
                              pdata[3],pstop[0]);
  printf(" CTS is not required.\n");

  return;

}
/* PORT_INIT  Initialize the COM port(s).                    */
void port_init1()
{
   int ercode;

  clrscr;
  cadspmsg(2,26,15,0,"I N I T I A L I Z A T I O N");
  ercode = 1;
  while (ercode)
  {
    cascroll(4,0,24,79,0,NORMAL);
    cacurset(4,0);
    cacursor(0,12,13);

    out_port  = cardnum(" Transmission Port          ");
    if ((obuffer = malloc(260)) == NIL)
```

```
        errmsg("Cannot allocate space for output buffer.",0,0);
        ercode = open_a1(out_port,128,128,0,0,obuffer);
        if (ercode != 0)
          errmsg("Cannot open output port.",ercode,0);
  in_port  = cardnum("   Receiving Port        ");
 }
 ercode = 1;
 while (ercode)
 {
  if ((ercode = setop_a1(out_port,1,7)) != 0)
        errmsg("Cannot set BAUD rate for the output port.",ercode,0);
 }
 ercode = 1;
 while (ercode)
 {
  if ((ercode = setop_a1(out_port,2,0)) != 0)
        errmsg("Cannot set Parity for the output port.",ercode,0);
 }
 ercode = 1;
 while (ercode)
 {
  if ((ercode = setop_a1(out_port,3,3)) != 0)
        errmsg("Cannot set Data Bits for the output port.",ercode,0);
 }
 ercode = 1;
 while (ercode)
 {
        if ((ercode = setop_a1(out_port,4,0)) != 0)
        errmsg("Cannot set Stop Bits for the output port.",
                                                    ercode,0);
 }
 if ((ercode = setop_a1(out_port,9,0)) != 0)
  errmsg("Cannot remove requirement for CTS",0,0);

 return;
}
/*************************************************************/
/* ERRMSG  Display an error message, error code and port status  */
/* if they are nonzero.  The bottom four lines of the transmit  */
/* window are used to display the error messages.          */

void errmsg(char *pmsg, int code, unsigned status)
{

 int row,col,tcur,bcur;
 static char *pcodemsg[] =
        {"Successful",
         "Reserved for future use",
         "Invalid COM port number",
         "COM port is not opened",
         "Invalid parameter or function value",
         "Reserved for future use",
         "No serial port found",
         "Output queue is full",
```

```
        "Reserved for future use",
        "COM port is already open",
        "Input queue is empty"};

 cacurst(&row,&col,&tcur,&bcur);    /* Current cursor position */
 cacursor(1,0,0);                   /* Turn the cursor off    */
 cacurset(21,0);                    /* Position the cursor,   */
 printf(" %s\n",pmsg);              /* display the message,   */
 if (code != 0)                     /* and possibly the code. */
   printf(" Error code - %02d  %s",code,pcodemsg[code]);
 if (status != 0)
   printf(" Port status - %04x",status);
 printf("\n");
 capause("");                       /* Wait for a key to be   */
 cacursor(0,12,13);                 /* pressed. Turn the cursor*/
 cascroll(21,0,24,79,0,NORMAL);     /* back on and clear the  */
 cacurset(row,col);                 /* window and go back.    */

 return;

}
/*************************************************************************/
float get_temp(void)
{
 int i,j,ii,digit, valid = 0;
 static float old_t1 = 1000.0, old_t2 = 1000.0, old_t3 = 1000.0;
 float tempe = 0.;
 i=0;
/******  read temperatures from input queue  ******/
ercode = 1;
while(ercode != 10)
{
        i++;
         ercode = rdch_a1(in_port,&ch,&iqsize,&status);
         if (status & 0x00ff)
         errmsg("Error accessing the input queue (rdch_a1).",0,status);

         if(i==1 && ch==50) valid = 1;
         if((valid == 1) && (i>2) && (i<7)){
                digit = (int) ch - 48;
                if((i==3) && (ch != 32)) tempe = digit * 1000;
                if (i==4) tempe = tempe + (digit * 100);
                if (i==5) tempe = tempe + (digit * 10);
                if (i==6) tempe = tempe + digit ;
          }
        }
          if ((valid == 0) || (tempe < 700) || (tempe > 2500)) return(old_t1);
/************************** Compute filtered temperature */
 tempe = (tempe + old_t1 + old_t2 + old_t3)/4.;
 old_t3 = old_t2;
 old_t2 = old_t1;
 old_t1 = tempe;

return(tempe);
}
```