DEVELOPMENT OF A SCADA SYSTEM TO MONITOR AND CONTROL CONTINUOUS CASTING OF STEEL BILLETS

by

VLADIMIR RAKOCEVIC
B.Sc. in Electrical Engineering, University of Belgrade, Yugoslavia, 1991.

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

MASTER OF APPLIED SCIENCE

in

THE FACULTY OF GRADUATE STUDIES

Department of Mining and Mineral Process Engineering

We accept this thesis as conforming

to the required standard

THE UNIVERSITY OF BRITISH COLUMBIA

August 1995

In presenting this thesis in partial fulfilment of the requirements for an advanced degree at the University of British Columbia, I agree that the Library shall make it freely available for reference and study. I further agree that permission for extensive copying of this thesis for scholarly purposes may be granted by the head of my department or by his or her representatives. It is understood that copying or publication of this thesis for financial gain shall not be allowed without my written permission.

Department of Mining & Mineral Process Engineering

The University of British Columbia
Vancouver, Canada

Date Aug. 30, 1995.

ABSTRACT

Continuous casting of steel billets, blooms and slabs is one of the dominant processes in the steel making industry. Production of high quality steel, with no defects, has become a very important issue in the current highly-competitive market conditions. To satisfy quality control standards, steel mills must now incorporate "state of the art" technologies in continuous casting. New techniques from different fields are being applied to the process to assist in casting "perfect" steel.

Computers are now essential elements in applying advanced technologies for process and quality improvements. In this work, an attempt has been made to create an intelligent Supervisory Control and Data Acquisition System (SCADA) for the billet casting process. The "brain" of this system consists of an Artificial Intelligence entity that combines low-level numerical processing of sensor inputs with high-level symbol processing, i.e. an Expert System. The mastered process knowledge and experience, along with intelligent numerical computation, have been captured into a system called "Smart" Mould.

This thesis focuses on the evolution of the hardware framework and software support for the SCADA system. The concept of intelligent computation as a prerequisite to intelligent process control with respect to the continuous casting of steel billets is also introduced.

TABLE OF CONTENTS

List of Tables

List of Figures

List of Symbols

t_N	Negative Strip Time (s)
π	3.14
f	Oscillation frequency (Hz)
V_c	Casting Speed (m/s)
S	Oscillation stroke lenght (m)

ACKNOWLEDGMENTS

I would like to express my sincere gratitude to my supervisors Dr. John Meech, Dr. Indira Samarasekera, and Dr. Keith Brimacombe, for providing consistent encouragement, outstanding guidance, and invaluable critique throughout this work. Without their help, support and assistance this thesis would never have been completed.

I am greatly indebted to Branko Zatezalo and Comdale Technologies Inc. for their support in designing the data acquisition software. Manitoba Rolling Mills, Alta Steel, Hatch Associates and the Natural Sciences and Engineering Research Council of Canada are greatly acknowledged for support of this study.

I am grateful to Sunil Kumar and Neil Walker for providing many useful comments and discussions during this work.

Finally, the tremendous gratitude I dedicate to my wife Ana who provided me with every assistance throughout my studies at UBC. Without her enormous help in designing the software, this work would be unfeasible.

Chapter 1

Introduction

To remain competitive internationally and face the challenges of quality, productivity, cost and the environment, Canadian mini steel mills will have to apply intelligent process control systems, particularly for continuous casting. Such "smart" processes must have a brain which receives sensor inputs and, based on knowledge, experience and intelligent numerical processing, makes decisions to adjust operating parameters or informs the operator about existing process conditions. The systems must first be "aware" of what is happening in the process, then must provide control or alarm action.

The "heart" of a billet casting process is the water-cooled oscillating mould. To build a "smart" caster, we must connect this unit to the "brain", a computer program in which the human thought-process and sensory input system have been captured. The "eyes" of the system consist of instrumentation that senses mould temperature, friction, metal level, mould oscillation, casting speed and cooling water velocity. The "brain" of the system is software that combines an Artificial Intelligence (AI) entity with a module capable of low-level numerical processing of sensor inputs to generate high-level symbol processing, i.e. an Expert System (see Figure 1).

The use of an expert system to control the casting process was first proposed by J.K Brimacombe [1,2]. The "**Intelligent Mould**" research program was initiated at the

University of British Columbia and is funded by industry and NSERC. The objectives of this research program are as follows:

[1] To develop an intelligent continuous casting process via a smart mould system capable of maximizing billet quality and productivity;

[2] To complete the knowledge base for oil lubricants by developing relationships among design and operating parameters, and billet quality and productivity for different steel grades;

[3] To establish techniques to interpret mould thermocouple and load cell signals with respect to billet quality and casting problems, with both oil or mould flux lubricant;

[4] To establish a knowledge base for mould behavior and quality/operating problems;

[5] To develop relationships among mould heat extraction, mould powder composition, casting speed, steel composition and superheat, oscillation characteristics, wall thickness and cooling water velocity for casting operations utilizing mould flux lubrication;

[6] To complete the fundamental understanding of the linkages between mould design/operation and billet quality.

Based on the nature of these goals, a multi-disciplinary research team was formed at U.B.C. The execution of the project was divided between two main tasks:

1. Acquire new knowledge about the continuous casting process; and

2. Create hardware and software support for the mastered process knowledge.

Beside the necessity for experts in continuous casting, there was a clear need for an individual to take charge of the computer system development and software evolution. The main focus of this thesis is the development of the overall software and hardware required to be the repository for the "smart" mould knowledge. The developed system is referred as a Supervisory Control and Data Acquisition (SCADA) System.

The task of designing the "Smart" mould consisted of two separate issues: definition of hardware requirements and development of the necessary software. Figure 1 is a schematic representation of the components of the "Smart" mould system.

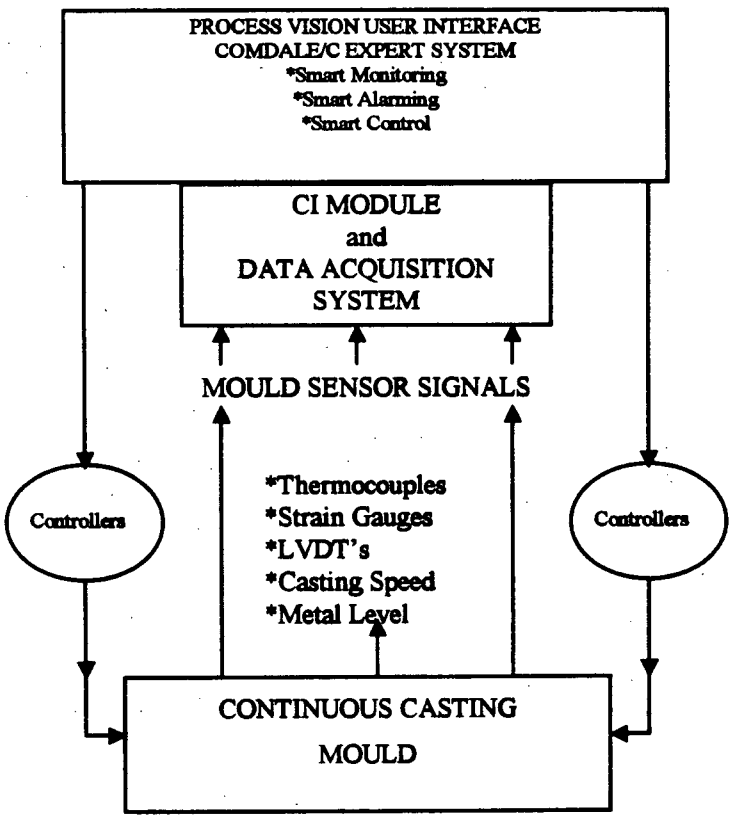

Figure 1: Structure of the "Smart Mould"

Hardware requirements included decisions about:

- the computer platform to use as the environment for the system "brain";

- the sensors to acquire process data;

- the data acquisition system/card to use for data collection;

- the best operating system to provide real-time multitasking features.

Software development involved building a multi-tasking high frequency data acquisition driver with a Computational Intelligence (CI) module for data processing. The CI module supports the beginnings of Symbolic Processing at the higher intelligence level. A new approach to developing a real-time Expert System has been applied, in which hierarchical connections have been established between the fast numerical data-processor and the decision-making system. As a first step in creating the knowledge base, a real-time Expert System within a SCADA environment was developed to trace transverse depression defects as they occur at the meniscus and then move down the mould.

This thesis consists of seven chapters. Chapter 2 gives an overview of Supervisory Control and Data Acquisition (SCADA) Systems and the evolution of AI techniques in process control. Chapter 3 presents the objectives of this thesis. Chapter 4 details the steps involved in creating the hardware framework and software support for the "Smart" mould. The structure of the intelligent driver is described and the concept of a unique shape recognition function is discussed. Chapter 5 reviews the results from several plant trials. Chapter 6 outlines the conclusions that arise from this work. Chapter 7 makes recommendations for future work.

Chapter 2

Literature Review

2.1 Outline.

This chapter presents a short overview of the historical involvement of Artificial Intelligence in process control. It consists of four sections:

Process Control and SCADA Systems. This section gives a retrospective on the basic ideas in control theory and explains some common terms. The Supervisory Control and Data Acquisition (SCADA) System is introduced as an integrated computer system for process monitoring and control.

Introduction to Artificial Intelligence. This section describes the genesis of the Artificial Intelligence field with an introduction to Expert Systems, Fuzzy Logic, Artificial Neural Networks and Genetic Algorithms as these components relate to this work.

Integration of AI with Process Control. This part deals with AI techniques applied in process control to accomplish different tasks. Additional requirements to apply AI methods in a real-time environment are pointed out. An insight into the architecture for real-time expert systems is presented.

Proposed Approach - Computational Intelligence (CI). The concept of Computational Intelligence is defined. The hierarchical connection between fast, intelligent numerical processing and real-time expert systems is proposed. The different levels of intelligence in biochemical and electronic processing are contrasted.

2.2 Process Control and SCADA Systems

A *system* is a collective entity defined and characterized by a finite number of attributes and elements. A *control system* is a biochemical or man-made system that applies a strategy (control logic) to achieve the best results, related to a set of defined objectives. The applied control strategy may be based on numerical theory and analysis, or on heuristics and complex decision-making procedures. In the biochemical case, the human feedback loop may appear to be instinctive or intuitive without any obvious "thinking" taking place. The more complex the control requirements, the more complex the strategy.

In general, the term control system, refers to certain specific man-machine interactions. The objective of the system is to control outputs in some prescribed manner by manipulating inputs through the elements of the overall system. Accordingly, a *process control system* is "made up of a process involved in a controlled system and its control equipment or hardware and operators"[3].

To obtain desired system behavior, the applied control strategy has to provide accurate and *stable* control. The control system is considered to be *stable* if the output remains bounded for a restricted input, i.e. it does not grow or oscillate with ever increasing magnitude. Figure 2 presents a feedback system with two inputs: desired temperature (set point value) and outside temperature (process load). Here, the simple system described can be operated in two main ways: to respond to "supervised" changes in the setpoint (servo control) or to adjust the control signal in a way to compensate for changes in outside temperature (regulatory control).

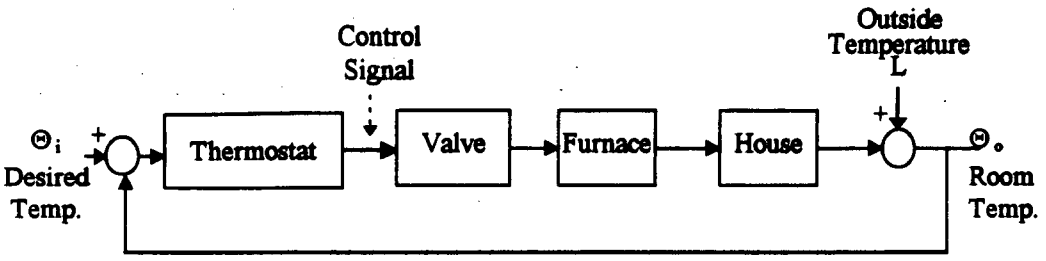

Figure 2: Closed loop control system

In automatic control, only machines are involved, as in Figure 2, where the furnace is turned off and on depending on a thermostat reading. Within the human body, blood pressure, blood sugar, eye-pupil diameter are a few of the many variables controlled by biological mechanisms that are equivalent to automatic control and can be studied by methods of feedback control.

When the control signal is made to be linearly proportional to the error in the measured output (Θ_i *minus* Θ_o), the system is called *proportional feedback*. When the feedback controller performs an integral operation on the error over time, the system is using *integral feedback*, and when the slope of the error-time curve is used, *derivative feedback* is the control mode. Feedback that uses all three methods - proportional plus integral plus derivative - is referred to as ***PID control***.

The purpose of regulatory control is to keep process variables close to specified values in spite of external process disturbances or variations in internal process dynamics. Regulator design involves discovering a "good" control solution for the mathematical expression Θ_o / L (see Figure 2).

In the servo problem, the task is to make process variables respond to changes in a *command signal* (set point) in a given way. The servo design provides a correct control strategy for mathematical correlation between system output Θ_o and set point Θ_i.

Considering the servo problem, another hierarchical control level can be introduced, the supervisory control loop, to manage the overall performance of the designed system. Supervisory control may take into account the performances of separate interacting systems and may involve adjusting controller parameters (proportional, integral or derivative actions) to maintain the best overall performance of the total system. This particular technique is often referred to as Adaptive Control. This might involve using a Reference Model in parallel with the process to provide Predictive Control of future output values from the process.

A major difficulty in control-system design is to reconcile large-scale, fuzzy, real problems with the simple, well-defined problems that control theory can handle [4]. The natural presence of heuristics in real life control asks for implementing Artificial Intelligence tools, particularly Expert Systems and Fuzzy Logic, to handle ill-defined process parameters, and large variations in control signals. Heuristics derive from unknown and unexpected input/output relationship changes from external factors not yet understood or defined. These include other variables and/or complex relationships. In particular, Fuzzy Expert Systems are suited to heuristic analysis in the supervisory control loop.

Digital computers have been integrated into process control systems to support fast and accurate execution of various control algorithms or strategies. Recent hardware

and, particularly, software development have expanded the commitment of computers to process control. The term *process control* nowadays, includes more than straight implementation of algorithms for direct control. It also comprises:

1. a process to collect sensor inputs
2. the graphic presentation of the overall process that helps an operator to visualize, better understand and handle (control) the process. This is referred to as the Man Machine Interface (MMI);
3. automatic monitoring and alarming; and
4. supervisory control to apply different inferences including human-like thought processes.

A computer connected to the process instrumentation, together with software that supports data collection, process graphics, alarming, and supervisory control is considered a SCADA System. The role of SCADA is to provide overall monitoring and control of complex, large scale processes. It is connected to the direct control units and may perform tuning of such devices. The control algorithm in SCADA can be realized by applying an algorithm for system optimization or by utilizing operator experience in running the process.

Considering the continuous casting process, there are several integrated computer systems that have been implemented in slab and bloom casting: mould level control [5], breakout detection [6,7], and on-line monitoring and diagnostics [8,9]. However, in billet casting there are few quality and process control systems, and the area lags behind slab

and bloom casting in achieving defect-free surface quality [1]. This work was undertaken to build quality control SCADA system for continuous casting of steel billets.

To build a SCADA System, in addition to the inevitable hardware decisions, a software development tool is necessary. Such a tool must have facilities for implementing control logic. The appropriate tool should include Fuzzy Logic, Expert System and/or Artificial Neural Network modules. The **ProcessVision** SCADA development tool was selected in building the intelligent system described in this work. An Expert System was built for on-line prediction of billet defects. The obtained process knowledge about mechanisms of surface defects formation has been captured in the reasoning system. The ultimate goal of this work is to create eventually a control module that will supervise the performance of PID controllers applied to regulate the position of the molten steel interface and/or other control variables in the process.

2.3 Introduction to Artificial Intelligence

> "*Artificial intelligence is the science of making machines do things that require intelligence if done by men.*" Marvin Minsky

Artificial Intelligence (AI) can be considered as a collection of programming algorithms and techniques which provide the impression that a computer is thinking, reasoning, making decisions, solving problems and learning [10]. It attempts to capture and present human reasoning on a machine.

The components of the human thought-processes with regard to decision making are: Symbolic Processing, Heuristics and Pattern Recognition.

In the AI sense, a symbol is a string of characters that stands for some real-word concept. Humans use symbols, apply IF-THEN-ELSE decision structures, utilize heuristics, and handle uncertainty to define and master a problem.

Heuristics are kinds of empirical knowledge which help a man or machine to describe, understand and solve a problem. These include rules-of-thumb based on trial-and-error, tricks, procedures and other relevant inputs that enhance the search process [11].

Pattern-matching or recognition involves automatic identification of patterns (figures, characters, shapes, forms, etc.) according to some predetermined condition or standard. Usually, an input pattern is compared to a stored template and the closeness of fit is determined.

Expert Systems (ES), Fuzzy Logic (FL), Artificial Neural Networks (ANN) and Genetic Algorithms (GA) are the most widely-used AI techniques for performing process control tasks. All of these techniques are appropriate for specific types of control problems. Expert Systems are able to simulate the reasoning process using IF-THEN-ELSE rule structures in problem-solving. Fuzzy Logic assists Expert Systems to reach solution in an uncertain environment, where some of the analytical parameters are unclear. Neural Networks are able to learn, or map any multi-dimensional non-linear relationship between input and output patterns. Genetic Algorithms provide very rapid solution for complex optimization problems, based on Darwin's Survival-of-the-Fittest theory. These techniques can be applied alone or can be combined together for better managing an obscure environment.

2.3.1 Expert Systems

An Expert System (ES) is a computer program which captures human knowledge and cognition in a way that emulates a human expert solving specific types of problems [12]. The technology of ES is a subfield of AI which advances the capabilities of the computer beyond conventional utilization; a reasoning structure and decision-making logic are applied in addition to interpreting incomplete information in an uncertain environment. It consists of equations, models, rules of thumb, do's and don'ts., etc. [13].

The main components of an ES are:

- Knowledge base
- Inference Engine
- User Interface

A knowledge base is a collection of data, rules, inferences and procedures arranged into frames, semantic networks, scripts, instructions, and other formats. It contains everything needed to formulate, comprehend and solve a specific problem. It includes facts, theory, experience, and rules to guide the application of knowledge to handle a problem. The necessary wisdom which comprises the knowledge base is based on pieces of evidence identified by an expert(s). The wisdom can be supported by theory or could come from experiential knowledge and informal judgment.

The Inference Engine is the "thinking" part of an ES. It is a computer program which processes knowledge contained in the knowledge base. It contains methodology to reason with the information in the knowledge base and to achieve conclusions. The Inference Engine is capable of: 1. analyzing a statement and decomposing it into its

components, 2. assigning degrees of belief to facts, 3. examining rules, 4. applying a search strategy, 5. explaining and justifying solutions, and 6. communicating with a user or external programs and processes.

The User Interface is a facility which allows communication between the user and the computer (ES). The user can ask for explanation and more detail about the subject, select items from a menu or a form, and provide an answer or presumption to a specific question.

In summary, the Expert System technique is an appropriate method to represent the way a human thinks while performing problem-solving. An operator observes and controls a process by utilizing logic. The way he/she reasons about the process phenomena is captured directly into an ES program. The modular approach used in these systems is an appropriate and effective element which is beneficial to designing a real-time control system. For Expert Systems to be able to deal with imprecise, unclear and/or noisy inputs, and to reach a solution in highly uncertain environments or in situations where crisp representation of knowledge is limiting, another AI technique is needed - Fuzzy Logic.

2.3.2 Fuzzy Logic

According to Webster's dictionary, *logic* is the science of the normative formal principles of reasoning. Hence, "fuzzy logic is concerned with the formal principles of approximate reasoning, with precise reasoning viewed as a limiting case" [14]. Unlike the conventional approach, fuzzy logic tries to model obscure techniques of reasoning that play an essential role in the human ability to make decisions in an uncertain environment.

Fuzzy logic provides a mechanism to represent the meaning of propositions expressed in a natural language when such meaning is unclear.

Considering process control applications, Fuzzy Logic is utilized in a supervisory control loop, as a fuzzy controller [15,16] (see Figure 3) or as a Fuzzy Expert System that reasons and generates a solution based on process inputs or comprehended operational logic.

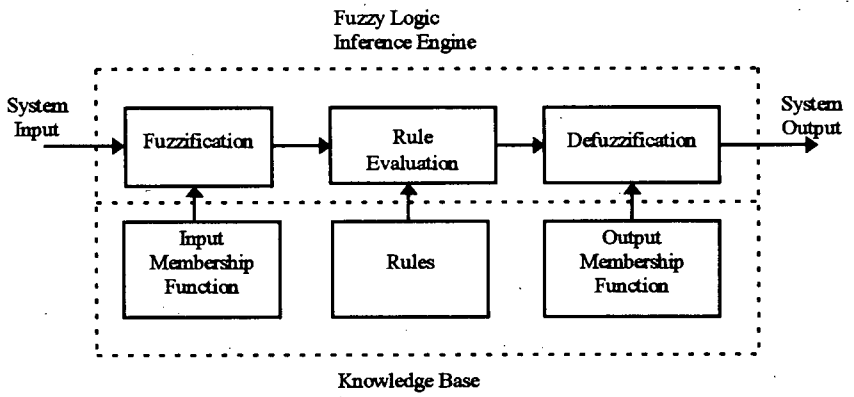

Figure 3: Structure of Fuzzy Logic Controller

The inputs in a fuzzy system are translated into fuzzy terms. The inference engine, using rules from the knowledge base, calculates fuzzy output, that in supervisory control is defuzzified into crisp numbers to provide a control action. Fuzzy Expert Systems can be utilized to assist an operator in handling the process, or to provide extended information about present conditions. Also, real-time prediction of process irregularities or product quality can be made, as it is done in a continuous billet casting application.

Fuzzy ES have the capability of adapting to changing circumstances in real-time by either adjusting membership function definitions [17], changing the method of defuzzification [18], or by adjusting weights associated with the Fuzzy Rules [10]. Adaptation generally takes place using sets of heuristics or meta-rules that are added to

the system one at a time until all uncertain situations are eventually covered. These heuristics act to create very complex multi-variable relationships that can be built up one fact at a time. Fuzzy Associative Memory (FAM), is a processing model that integrates fuzzy theory with neural networks and can be used to store the entire set of the weight matrixes for all covered situations.

True machine learning however, that occurs by applying a rigorous mathematical approach is the domain of a technique called Artificial Neural Networks. It is the learning ability that makes ANN very atractive for various process control applications.

2.3.3 Artificial Neural Networks

Artificial neural networks (ANN) are a modeling approach based on the exact way that information is processed within the brain. A neural network consists of processing elements (PE), called neurons or nodes, grouped into layers, with weighted connections between the elements in each layer. Each PE in an ANN receives signals from all its input connections, performs a mathematical operation, and produces a single output value. The described processing paradigm is also referred as a **Connectionist model**, highlighting the structure of highly interconnected processing units (see Figure 4).

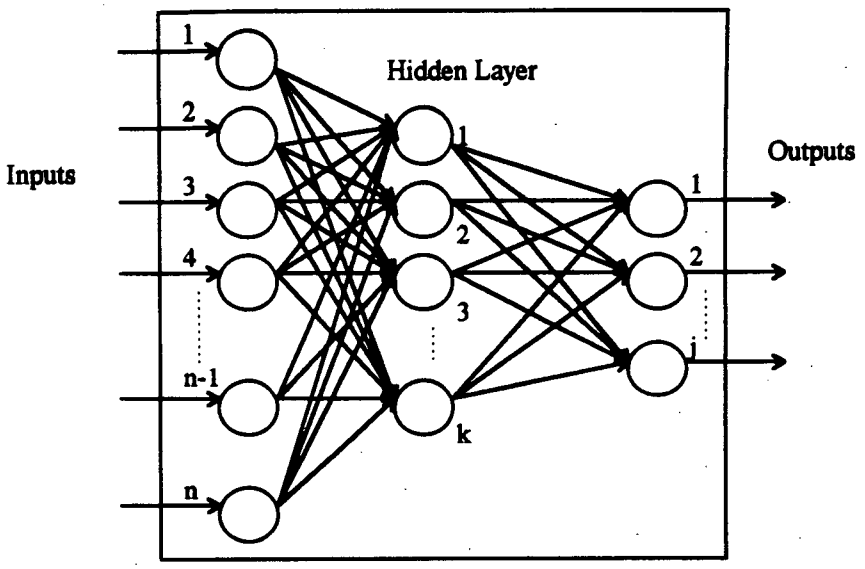

Figure 4: Connectionist model with a single hidden layer

In a broad sense, an ANN consists of three principle elements[19]:

1. *Topology* - a structure and connections of ANN layers
2. *Learning* or *Training* - a way to store information in an ANN
3. *Recall* or *Testing* - a way to retrieve information from an ANN

The first step in applying an ANN is to select a network architecture that encompasses the requirements of an application. The topology of an ANN refers to the architecture into which neurons are arranged and connected. The *layers* of neurons and the *connections* made between them are two of the most important characteristics that define a network's architecture. Both the number of layers and the size of each layer can be varied. The number of PEs in each layer is arbitrary and difficult to specify when designing networks. It is a function of the form of input and output data, and the type and complexity of the processing task.

Learning or training an ANN is the process of obtaining a desired response by changing the connecting weights. The backpropagation learning algorithm is one of the most utilized methods in supervised learning. This approach is based on computing the necessary changes in connection weights to gain desired behavior. The input data along with the desired output vector are presented to the network, the obtained response is compared to the targeted one and then the weights are changed in the direction of decreasing error. Two arbitrarily selected ANN parameters, a Learning coefficient and Momentum, guide the searching algorithm through numerous iterations. The learning coefficient and momentum are used to allow the present network error and the previous weight change to influence the weight change in the current iteration. The learning phase is over when the output error becomes less than a predefined value, usually 5% [20,21]. Once learning is completed, the ANN simply acts as a "black-box" to provide immediate and quick Input/Output mappings.

Artificial Neural Networks are very beneficial tools for real-time process control problems. They can be implemented at different levels and provide numerous tasks for direct process control, such as pattern matching and noise removal.

In this work, an ANN technique has not yet been considered. The learning capability, that is a big advantage of ANN in an off-line mode, becomes a drawback when performing in a real-time environment, since it is inherently slow and mathematically intensive. However, the structure of our CI module has been designed to allow for future incorporation of ANN methods should the speed issue be resolved.

Holographic Neural Networks [22] may be the key to real-time learning issues in the future. The network structure is not based on the connectionist model and the learning algorithm is not executed in an iterative fashion. The holographic network consists of a single neuron which contains complete information about the Input/Output mappings. The search for an appropriate mapping is performed using complex numbers. The entire input data sets and targeted outputs are presented to the network at one time. The learning process is reduced to a single iteration, i.e. calculation of the complex elements of the Input/Output matrix is accomplished in seconds. The network precision is at least equal to that obtained by the connectionist model.

2.3.4 Genetic Algorithms

A third methodology evolving out of the AI field is called Genetic Algorithms (GA). These techniques are used to look for an optimal solution to large industrial multi-variable problems. In the case of this work, GA could be used as a method to adapt fuzzy set definitions in an FAM or link-weight values in an ANN in an optimal fashion.

GA transform a set of individual objects, called a population, into a new set (new population or next generation) using operations based on the Darwinian principle of survival of the fittest and natural genetic recombination. An object's characteristics are mapped into a fixed-length character string (chromosome) associated with the fitness value, i.e. objective function. The optimization is achieved by evolving, via mating two "good" input strings (parents) at a time and generating a new string with better characteristics than either parent [23]. The following steps are utilized in GA:

1. map object characteristics into binary strings and define an objective function to depict each object of the population,

2. mate two "good" objects of the population using a number of cross-over transformations and mutations,

3. evaluate the outcoming offspring and replace one of the parents,

4. continue the process until the objective function is maximized or minimized.

Genetic Algorithms have become very popular methods in designing control systems where the inputs and set up values can change significantly during the operation. GA provide the optimum output space for such changes. However, in developing the intelligent SCADA system for billet casting it was not found necessary to apply Genetic Algorithms at this stage. The method can be added to the CI module, if there is a requirement in future work.

2.4 Integration of AI with Process Control

In the last decade there has been an expanding interest in utilizing Artificial Intelligence techniques in every engineering domain. In Process Control, AI methods can be encountered in almost every step of the analysis of processes and the design and implementation of control algorithms: straight advisory systems for designers and operators (off-line applications), diagnosis and supervisory systems (on-line applications), and direct control (real-time applications). However, the real-time environment adds extra requirements to the problems arising when using AI in process control.

The necessity for analytical representation of a process that must be controlled has driven research into fields such as process modeling, system identification, parameter estimation, etc. In cases where a process cannot be depicted by a linear model and the control requirements are not translated to straight criteria, analytical solutions are extremely difficult to find (if not impossible), and design becomes a numerical optimization problem. People have approached this type of problem by applying model-based predictive methods in which the design procedure is embedded into the prediction of the process output over a certain prediction horizon, based on a model of the process or on time-series analysis of input and output signals.

In cases where a mathematical model of the process is not available, the control logic must be based on qualitative expressions and experience of people working with the process. Actions can be performed either as "the result of evaluating rules or as unconscious actions based on presented process behavior after a learning phase" [24]. It provides the opportunity to use experience collected from operators and process engineers. Uncertainty about the knowledge can be handled as well as ignorance about the structure of the system. In this approach, the attributes of conventional control theory (such as frequency response, complex domain, stability, system identification, controllability, etc.) are not issues. People who confront difficulties in describing a process mathematically, are eager to apply AI methods, because the experience obtained from running a real process can be implemented regardless of whether the system is linear, non-linear, time-variant, etc.

In analyzing computer control of an industrial process, several levels can be delineated: Level 0 - Process Instrumentation, Level 1 - Direct control, Level 2 - Supervisory control, Level 3 - Plant-Wide control, and Level 4 - Enterprise as depicted in Figure 5.

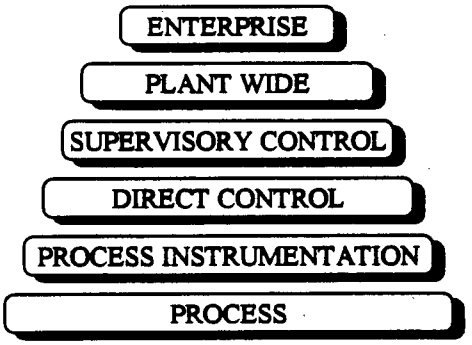

Figure 5: Control System Hierarchy

At the lowest level, instruments sense, monitor, and manipulate process variables. Such devices are connected to control units capable of implementing control strategy. These units consist of single-loop controllers, Programmable Logic Controllers (PLC's), or Distributed Control Systems (DDS), each of which may apply some combination of sequential or continuous-time control logic. The process of collecting, presenting, and managing sensor data is based on numerical methods. At this level, a control system responds to external events extremely fast, carrying out several activities simultaneously. Such a system must operate on a computer under a real-time multitasking operating system to provide such speed.

A real-time system can be considered:

An integrated computer system that responds fast enough to the interrupting external events to provide accurate and fast control (or alarm) action.

Usually, systems with responses of 10 ms to perhaps, a few seconds are considered real-time depending on the process and/or circumstances.

At the next level is the supervisory control computer. This computer is connected to the primary control devices by network communication. The supervisory host maintains applications and databases that sit above the direct control function. Like a supervisor, the system gathers relevant information from the lower levels at relatively slow rates. The operating mode can be called "pseudo" real-time defined as:

An integrated computer system that responds 'fairly' fast to external interrupts and carries out a control action at the speed of several seconds up to perhaps one minute.

Similar to a real-time system, "pseudo" real-time provides fast and accurate control signals, but in a different time domain. Obviously, both definitions are context-dependent.

In many plant environments, the supervisory computer is connected by another communication path to a plant-wide computer system that maintains the business and overall manufacturing aspects of the plant. Finally, this plant-wide system can be connected to a corporate computing system that runs the corporate-wide applications. Neither of these two levels need be real-time or "pseudo" real-time environments.

AI techniques have been extensively applied in process monitoring and control systems. In the paper "Artificial Intelligence and Feedback Control" Verbruggen and Åström discussed the different domains and levels in applying AI techniques in process control [25,26]. They proposed separation of a control system into algorithms and logic, possibly in a hierarchical structure. For direct Real-Time Expert Control they suggest

division of the knowledge base into structural and numerical knowledge. Some other authors also support multi-layered structure in an intelligent control concept, integrating various methods of information processing like symbolic AI methods and neural networks for basic mathematical computations [27,28].

The *progressive reasoning theory* was proposed by Lattimer [29], where reasoning is divided into several levels. Each new part of a rule goes into more detail about the subject. This methodology is considered as progressive deepening. When a rule base has been evaluated and there is still time left, another deeper rule is estimated in order to try to produce a better conclusion.

Work done by Wickramarachchi et al., 1995 [30] utilized a hierarchical structure for fuzzy control in a fish processing application. Lukas et al., 1989 in the paper "Evolution of Expert Systems for Real-Time Process Management" [31], introduced the concept of an embedded expert system for interfacing with a real-time distributed data acquisition and control system. The embedded AI concept also has been described by Musliner et al., 1995 [32]. They offer the following three principle ways to combine AI techniques and real-time into a single system:

[1] embedding AI into a real-time system.

[2] embedding real-time reactions into an AI system, and

[3] coupling AI and real-time subsystems as parallel, cooperating components.

The major issue in applying AI techniques in real-time, or direct process control, is that symbolic reasoning is relatively slow on machines designed for fast and efficient numerical processing (von-Neumann machines). On one side, computers are capable of

very fast processing of complex numerical algorithms, while on the other, AI techniques process symbols instead of numbers. As the number of symbols and their interrelationships grow, the resulting complexity lends to poor turn-around time.

The solution is to divide overall system tasks between those requiring numerical algorithms and those based on heuristics and logic. But numerical processing must support the beginnings of symbol processing to provide intercommunication between these modules. In this way the numerical processor must acquire some "intelligence".

2.5 The Concept of Computational Intelligence

Symbolic processing on conventional "von-Neumann" type computers is inherently inefficient because of the architectural design of the hardware. The Arithmetic Logic Unit, the heart of information processing in a microprocessor, is designed for binary manipulation. A new design which deals with symbols instead of crisp on/off inputs, would speed up AI techniques. However, the hardware solution is not yet widely available and people are turning to software to overcome the speed issue. So how can AI be moved effectively into an environment (real-time, direct control, etc.) that demands intensive and rapid numerical computing? The answer is found in a new paradigm evolving from the field of AI - *Computational Intelligence*.

Computational Intelligence (CI) is a term first coined by Bezdek in 1993 to describe *"low level" knowledge in the style of the mind* [33]. CI consists of very "primitive" concepts, in the AI sense, that support the beginnings of symbolic knowledge

which he called "tid-bits", These "elements" can become the inputs to an artificial intelligence structure that processes the symbols heuristically or in other ways.

Primitives are the fundamental processes conducted on numbers: addition, subtraction, multiplication, division, and comparison.

These make up the basic steps in any complicated numerical structure that produce "elements" as output. Current hardware can deal with these complex numerical structures in a efficient manner and so, CI can become the underlying support structure for AI methodologies.

The components of CI may include Fuzzy Logic maps, Artificial Neural Network connections, or Genetic Algorithm optimizations, all of which use numerical methods to support symbolic knowledge.

Fast Fourier Transform (FFT) [34] and Wavelets theory [35,36] are two types of straight mathematical methods that are widely used in signal processing and can be implemented in CI to support real-time AI applications. FFT is a fast algorithm for calculating Fourier coefficients in the discrete domain. The technique provides a frequency spectrum of a process signal and is primarily used to detect signal noise or process disturbances. The developed CI module will include a FFT algorithm to analyze the operating conditions of a mould oscillator.

However, a definition of CI that is limited to numerical techniques seems lacking in that Computational Intelligence should not rely only on pure mathematics. Rules of thumb based on trial-and-error and other relevant inputs can enhance the search process to

provide symbolic output for the higher intelligence levels. This could help to increase performance speed, even though gross error may be contained in the output.

To bring AI into the lower levels of the control hierarchy of a real-time environment, CI modules for creating "primitive" symbols must be very fast. By implementing AI and heuristic approaches into CI modules, we introduce certain error, but gain on speed. The key trade-off in real-time computing is always: accuracy versus processing speed.

Considering that error can derive from applying heuristics in CI, it is proposed that direct connection between CI and AI levels can assist with error detection and interpretation at the AI level. The AI module can test symbolic output from cooperating sensors and recognize, tune out or reduce such error (Figure 6).

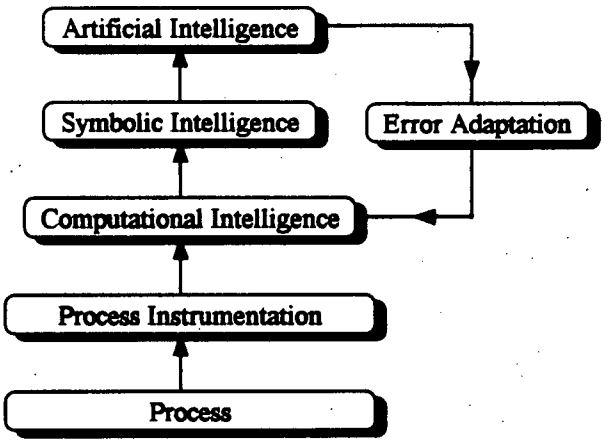

Figure 6: Error Detection in AI Correcting CI methods

Following an analogy from Bezdek, an individual given an apple can immediately recognize it by shape and colour. With eyes closed however, other sensory inputs must be used: touch, smell, taste. Smell and taste inputs will produce real-time decisions about an apple, but touch and feel may require additional processing to visualize in the mind the

shape and texture of an apple. The symbols of shape and texture must be recalled and compared with measurements that are less accurate than sight.

To assist AI in making rapid decisions intelligently, the CI module needs the following:

- IF/THEN rules (inferences and relationships)
- prior knowledge (to direct the CI process)
- symbolic "primitives" (output from CI module)

The approach resembles the hierarchy of human intelligence depicted in Figure 7.

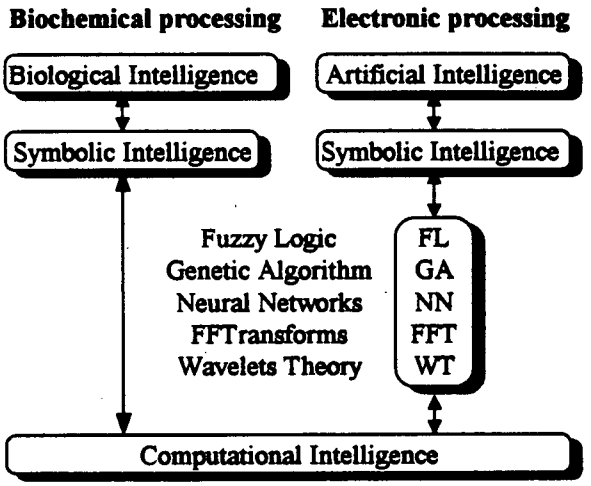

Figure 7: Comparison of Biological and Machine Intelligence

Biological Intelligence consists of manipulating symbols supported by low-level numerical processing to generate belief in a particular symbol. This hierarchy is mirrored in the arrangement of AI with CI to form the basis for rapid problem-analysis.

Within Biological Intelligence, the ability of autistic savants to carry out rapid and accurate data calculation, musical recall, etc., are examples of how the human brain can perform unusually accurate real-time computation. Whether output is intelligent or not is

determined by those who interact with such exceptional people. Perhaps they use fuzzy sets with very broad support characteristics (see Figure 8) [37]. Such set definitions can provide complex Input/Output mappings not achievable with "properly" defined sets. Unfortunately, the application of each individual set will produce rather useless output (100% medium, 60% low, 20% high), etc. So, while savants may be able to tell the day of the week for a particular date, they rarely understand the significance of the date in question.

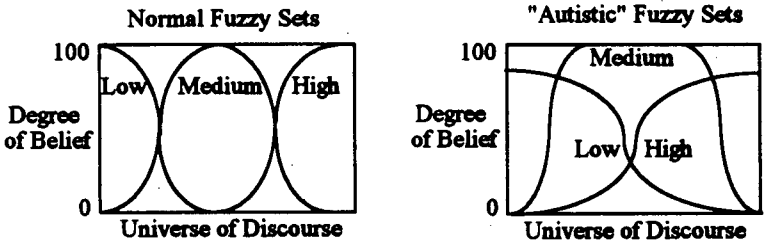

Figure 8: Normal and "Savant" Fuzzy Sets

This hierarchical intelligence concept has been selected in building the AI entity for intelligent billet casting. The character of the application required separation of fast numerical algorithms for data processing from the logic for real-time quality prediction. The CI module performs fast and intelligent computation of sensory inputs and supports the beginnings of Symbolic Processing at the higher level. The filtering functions within the CI module accomplish rapid feature extraction and apply heuristics to guide the search process.

The development of the CI module was one of the main tasks in this work together with construction of the Intelligent SCADA system for billet casting.

Chapter 3

Objectives of this work

An integrated computer system that utilizes AI techniques (Fuzzy Expert System and Computational Intelligence) has been built for monitoring and on-line defect detection in the continuous casting process. To create this system, research work was undertaken to meet the following objectives:

[1] define a computer platform, operating system, and development tool for Supervisory Control And Data Acquisition that can be integrated into a control system for Continuous Casting of Steel Billets.

[2] define hardware requirements for data acquisition.

[3] create software for high speed data acquisition.

[4] create a Computational Intelligence (CI) module to support symbolic processing.

[5] build a real-time Expert System for on-line detection of transverse depression formation in steel billets.

[6] test the data acquisition software, CI module and real-time Expert System at several Canadian mini-mills to verify its performance and reliability.

Chapter 4

Methodology

4.1 Description of the Billet Casting Process

In the continuous casting of steel billets, molten steel is brought from the steelmaking shop in refractory-lined ladles and discharged into a vessel, called a tundish. The tundish is used as a container for the liquid steel and helps to control the delivery of liquid metal to the mould. Steel is poured from the tundish through a nozzle into a water-cooled copper mould. The semi-solid strand is continually pulled from the mould with the help of a set of rotating pinch-rolls. Rotational or casting speed, is linked to the level of molten metal in the mould through a standard PID control loop. Changes in this level from its set position are reflected by changes in the casting speed. Metal level varies appreciably when turbulent conditions exist from a "ropey" stream.

The mould oscillates up and down to help strip the newly formed solid shell from the mould wall. Displacement is usually sinusoidal but when sticking or binding occurs, these signals can become distorted. Mineral oil or powdered mould flux lubricates the strand during casting. Lubricant behavior, whether oil or powder, is primarily linked to heat transfer conditions prevalent at the time of casting.

Beneath the mould is a water spray system that cools the semi-solid strand. A torch is used to cut the cooled strand into desired lengths to make billets (see Figure 9).

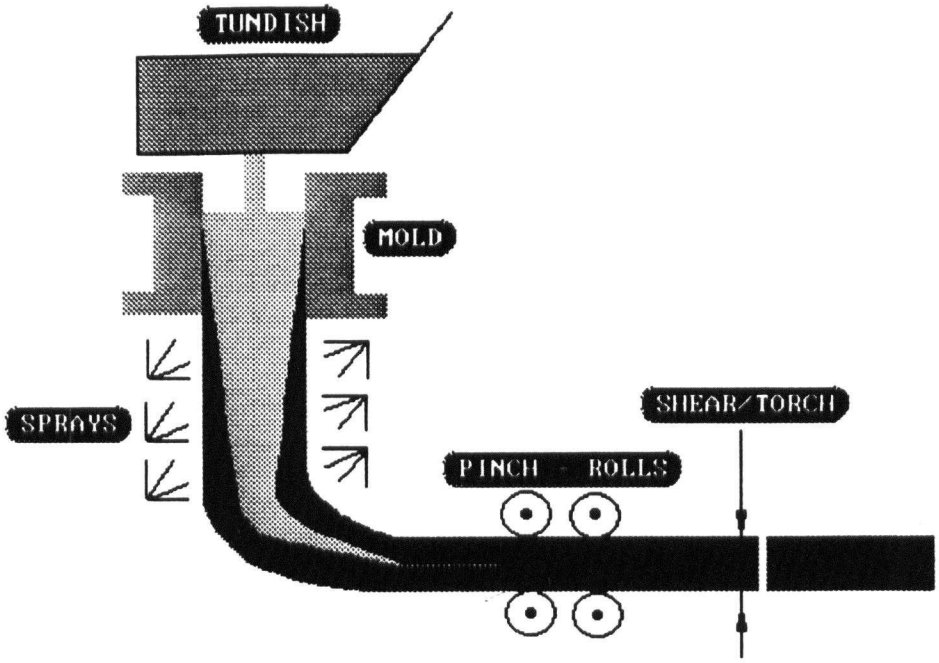

Figure 9: Schematic diagram of Continuous Casting process

The stream of liquid metal must be straight and smooth. The stream condition influences the behaviour of the metal level, also referred to as the meniscus. Many billet defects originate at the meniscus and are caused by a "ropey" stream. The tendency is to keep metal level position constant, and hence casting speed as well [38]; however changes in flow rate can lead to fluctuating levels. The mould oscillation frequency, together with displacement stroke and casting speed, are process parameters that must be monitored and controlled during casting in order to prevent surface defects. Negative strip time is a process variable that combines these three inputs to produce information about the relative ratio between casting speed and mould velocity. It is defined as the duration when the mould moves downward faster than the strand, i.e. casting speed. Uniform distribution of mould strand lubricant and appropriate cooling water flow rate are additional requirements in performing ideal casting .

The following variables are currently used by the research team at UBC in the monitoring of continuous billet casting:

- Metal level position - provides information about the current meniscus position. It can be used for calculation of metal level standard deviation. Consecutive rise and drop in metal level can produce defects in a billet. Metal level variation can be applied as an early indicator of some quality problems [38].
- Casting speed - provides the current withdrawal speed. It can be applied in on-line calculation of negative strip time. Also, it is used for monitoring process upsets.
- Cooling water flow - gives the rate of current water flow. A reduction in water flow during casting can cause a rise in mould temperature and affect the heat transfer.
- Water temperature - provides the current temperatures in the cooling water system. The inlet and outlet water temperatures are measured during casting to allow direct calculation of heat transfer in the mould.
- Mould temperature - gives essential information about mould-strand interaction. These measurements directly reflect the behaviour of the billet shell. Usually, several thermocouples are placed on all four faces down the mould to provide key information about billet defects.
- Mould displacement - this measurement supports understanding of the mechanical aspects of mould behaviour during casting. It can be used for on-line calculation of oscillation frequency, displacement distortion and negative strip time.
- Stream quality - poor liquid metal stream quality has an important effect upon metal level turbulence. More air is entrained by the ropier streams producing a turbulent metal level in

the mould. The research team at UBC uses video, 35 mm photography, mould thermocouple measurements and metal level signal to monitor the effect of the stream turbulence on mould temperatures.

The typical sensors and devices used in the process are:

- tachometer for casting speed measurement
- radioactive source coupled with a radioactive detector (metal level sensor)
- mould thermocouples
- Linear Variable Displacement Transducers (LVDT), and
- single loop PID controller

To build an intelligent computer system for continuous casting, all of these sensors and instruments must be connected to the system and observed during the operation. There has to be appropriate hardware to support data collection. Correct decisions about hardware platform, data acquisition board and SCADA system development tool, must be made to achieve the proposed objectives.

4.2 Selection of Hardware Platform and Software Development Tool

Process control applications usually consist of several independent activities that run simultaneously. Response time is very important and hence the computer system involved in these applications must provide real-time multitasking capabilities as defined in Chapter 2. The character of the continuous casting application provided the selection criteria for a computer platform, an operating system (OS) and the software development tool.

The hardware design decisions were made based on the following criteria :

- distributed real-time multitasking capabilities
- networking capabilities
- Intelligent SCADA system development tool
- price
- ease of use

In the case of real-time multi-tasking and networking, we need to examine these properties as they relate to selecting an overall operating system. Table 1 presents several widely used OS.

Table 1: Examined Operating Systems

OS	Multi-tasking	Real-Time	Distributed	Networking
MS-DOS	No	No	No	No
Windows	Yes	No	No	No
Windows NT	Yes	Yes	Yes	Yes
Windows 95	Yes	Yes	Yes	Yes
OS/2	Yes	Yes	No	No
OS/2 Warp	Yes	Yes	Yes	Yes
UNIX	Yes	No	Yes	Yes
QNX	Yes	Yes	Yes	Yes

The main responsibility of an operating system is to manage a computer's resources. All activities in the system - scheduling application programs, writing files to disk, sending data across a network, etc. - should function together as seamlessly and transparently as possible. Real-time applications depend on the OS to handle multiple events within fixed time constraints. Hence, multitasking, priority-driven preemptive scheduling, and fast context-switching are essential elements of a real-time system. The

structure of such a system is based on a kernel, the heart of any OS, and a group of cooperating processes. In a real-time OS the kernel should be very small and dedicated to only two essential functions: message passing and scheduling.

MS-DOS, the most widely used current PC-based OS is definitely not appropriate for these problems. RAM is limited and the architecture is purely serial in nature.

Windows is in fact, a Graphic User Interface overlaying MS-DOS with some multi-tasking capabilities. Resource management is not as rigorous as in a real-time OS. Task scheduling is not priority-driven and preemptive. The elements of an application can not be run on several personal computers simultaneously. Obviously, Windows is not suitable for this type of application. However, Microsoft has produced Windows NT and Windows 95 for distributed, real-time, networking applications. They could be the right decision for the "Smart" mould, when they become widely available and used.

OS/2 Warp is also an operating system of choice. It has real-time networking capabilities. It can be distributed over several machines and can share their hardware resources. It supports priority-driven preemptive scheduling. OS/2 Warp could be also the right choice for the billet casting application, when it becomes widely accepted.

The most commercialized multitasking operating system for industrial applications is UNIX. Even though the UNIX kernel was not designed for real-time applications, the OS can be used in real-time. The networking capabilities makes UNIX very attractive for various applications.

QNX, produced by Quantum Software Systems, is a truly distributed real-time multitasking system designed for the PC platform. It supports all essential features of

modern real-time computing. Its modular structure allows an application to be executed on several QNX nodes simultaneously. Considering the speed issue, QNX outperforms any of the listed OS.

Several SCADA System building tools were examined for use with the "Smart" Mould application. The major criterion in selecting the development tool was availability of AI techniques in supervisory control. Table 2 lists the tools widely accepted in industry.

Table 2: Examined SCADA Systems Development Tools

SCADA Development Tool	OS and Hardware Platform	Include AI
G2 from Gensym	Windows;Unix; Work Stations;	Yes
ProcessVision from Comdale	QNX; PC	Yes
Real Flex - Quantum Software	QNX; PC	No
InTouch from Wonderware	Windows & Windows NT; PC	No
Factory Link from US Date	Windows(NT) and OS/2; PC	No
FIX-DMACS from Intellution	Windows; PC	No

Although there are many SCADA system development software on the market, there are only a few that include AI technology in supervisory control. **G2** is one of the most popular building tools with AI capabilities for UNIX platforms. **G2** was originally designed for process modeling and is the best overall Graphic-modeling tool; for unique MMI and AI methods it is not as flexible, but prototype development using object blocks is rapid.

For personal computers, there are many SCADA systems that run in Windows or OS/2, but they do not involve intelligence (see Table 2). The **ProcessVision** SCADA development tool, produced by **Comdale Technologies**, surpasses this disadvantage; it runs on a PC, under the QNX OS and contains a real-time fuzzy expert system development shell. It is also widely used in industry [39,40,41,42,43].

The cost of software and hardware are important considerations in this work as it is desired to produce a system for distribution to industry. The price of the "Smart" Mould can be an inhibiting factor in its implementation so the issue is one of low price versus loss in flexibility.

PC-based systems are more widely accepted by industry not only because of price but also due to the availability of trained personnel who accept the hardware without reservation.

According to the above criteria the PC platform, along with the QNX OS and **ProcessVision** building tool were selected in creating the "Smart Mould" system.

Figure 10 describes the main parts of **ProcessVision**, indicating the multi-tasking features of a real-time system.

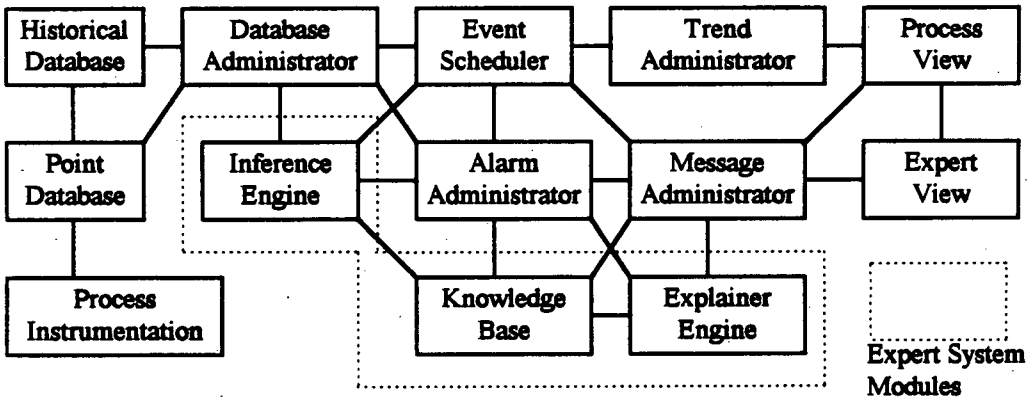

Figure 10: Typical Configuration of ProcessVision

The high-level supervisory decision-making modules consist of the inference engine, the knowledge base and the explainer module.

4.3 Selecting a Data Acquisition Board

QNX OS and ProcessVision provide data acquisition facilities over a computer's serial communication port. The sensor inputs can be collected by some external data acquisition device or computer and passed directly to ProcessVision's point database through networking connections (see Figure 10).

Alternatively, a dedicated data acquisition board can be plugged into a computer and used directly for fast and accurate data collection. A software driver is a necessity to run data acquisition tasks. A ProcessVision Utility called the Third Party Interface Library is used to support connection between the output from the data acquisition program and ProcessVision.

The current steel-plant instrumentation does not provide enough sensory inputs for creating a "Smart" Mould system. The UBC research team uses thermocouples located around a mould to sense strand surface temperature. To produce a stand-alone intelligent system, a dedicated data acquisition board was necessary to record the mould thermocouple data. The networking connection to the external acquisition devices or a computer is still available and can be used if necessary.

Several data acquisition boards were considered: from straight high precision data acquisition boards (DAS 20 designed by Keithley Metrabyte) to Digital Signal Processing boards with Intell 80486 microprocessor (DAP 3200e designed by Microstar Laboratories) (see Table 3).

Table 3: Considered Data Acquisition Boards

The Board Characteristics	DAS 20	DAP - 800/1	DAP - 3200e/101
Analog Inputs	16	8	16
Analog Inputs expandable to	128	32	512
Samples per second (x1000)	100	75	330
Selectable ranges	7	4	3
Digital Inputs	16	8	16
Analog Outputs (AO)	2	2	2
AO updates per second (x1000)	130	75	330
Digital Outputs	16	8	16
Processor	No	80C188 10 MHz	80486SX 24 MHz
On-board OS	No	DAPL	DAPL
RAM (Kbytes)	2	256	4096
QNX low-level driver	Yes	No	No
Price	$2259	$2206	$3945

The basic criteria for selecting DAS 20 was the availability of QNX software for the board interface. Because of the availability of an existing QNX driver, this was the fastest way to write the data acquisition and processing software to support a real-time Expert System. Microstar Laboratories did not provide software support for a QNX OS and their boards were relatively expensive. The big benefit for employing "intelligent" boards would be the use of a dedicated processor for data filtering, leaving the main CPU for decision-making tasks. This maybe the best approach in the future if the necessary software becomes available. However, using the multi-tasking capabilities and priorities under QNX and/or its distributed networking capabilities, the CPU time can be controlled and used more efficiently even with the DAS 20 board.

4.4 Isolation Amplifiers

In practice, sensors are located on and grounded through the process equipment. On the contrary, the plug-in data acquisition board is grounded through the computer. These two ground points are not at exactly the same voltage and the difference between the ground voltages is included as an input signal (see Figure 11).

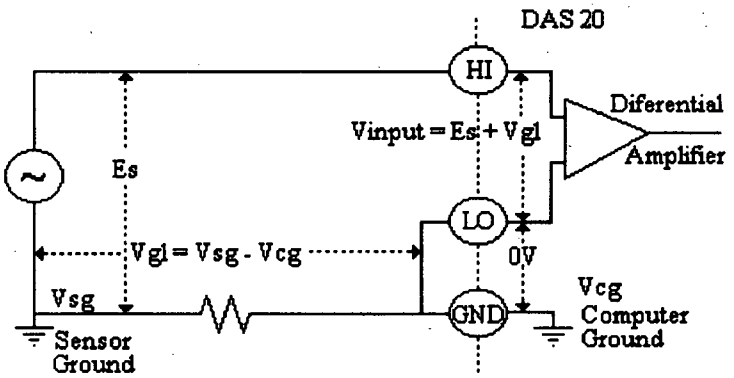

Figure 11: The creation of a ground loop

The ground loop problem can be avoided using floating signal sources (not locally connected to the ground). However, this solution was not applicable in this work, because the drilled-in thermocouples are always grounded through the mould, thus two common grounds are formed.

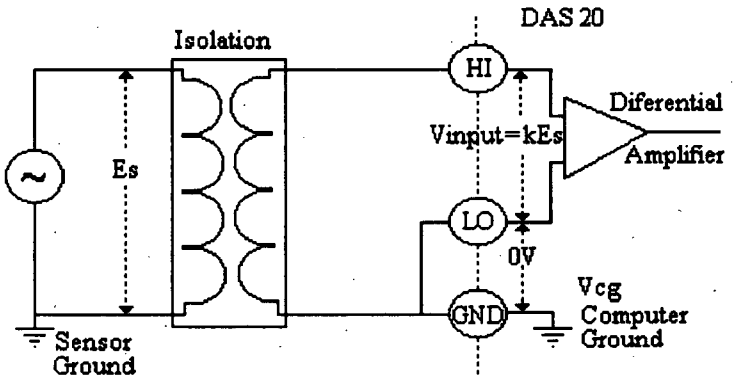

Figure 12: Avoiding ground loop problems

To build a SCADA system that depends on correct and reliable process data, signal specific isolation amplifiers and isolated thermocouple converters were purchased and implemented in the "Smart" mould system. This isolation system separates the plant current circuit from SCADA, and hence completely prevents the ground looping problem, as illustrated in Figure 12. In addition, this technique results in amplification of the thermocouple signals so measurements are recorded in volts rather than milivolts, providing better input resolution and more accurate data values.

4.5 Software development

Software development consisted of three phases:

1. Design and testing of data acquisition driver
2. Design and testing of CI module - filtering functions
3. Design and testing of the Expert System
4. Development of Man Machine Interface

4.5.1 SCADA driver

The data acquisition driver is a program developed in "C", using the Watcom C compiler for QNX environment. The driver collects data from up to 16 analog inputs from process sensors at a user-configurable sampling frequency. The design of the data acquisition software was based on the Data Acquisition (DAS) Library for the DAS 20 board obtained from the QNX Bulletin Board on the Internet.

The first step in creating the driver was to build the DAS library and DAS manager. This was an undocumented procedure and took several weeks to obtain a correctly compiled version of the library. The second step included design of an algorithm for data acquisition. The idea was to read data from each active input channel at the desired sampling frequency over a specified time interval, store the data in a 2-dimensional table, perform calculations over the collected data, and begin to read again, hence creating an infinite loop for continuous data acquisition. The first index in the data table specifies the channel number, and the second one references the sampled variable.

The DAS 20 board has a 16-bit input buffer where the converted analog signal is captured. The first four bits are reserved to define the input gain range (-50 +50 mV, 0 +100 mV, -0.5 +0.5V, 0 +1V, -5 +5V, -10 +10V, 0 +10V) and the operating mode (16 single-ended or 8 differential inputs). The remaining 12 bits are reserved for the converted analog input [44].

To acquire sensor data, the DAS 20 board must be initialized and the channel input range must be delineated properly. Initialization is completed by applying the **das_open** function from the DAS library. This reads a default DAS configuration file, sets the port address, the interrupt request, direct memory access channel, range control (local = 0, remote = 1), operating mode (single ended or differential), pacer-clock frequency, and A/D and D/A converter resolution (2 bytes). This function must be included in the driver code to control the operation of all other library functions. The function **das_ad_set_range** sets the analog input voltage range, which must be applied

before reading. Finally the function **das_ad_sync** performs a single synchronous Analog to Digital conversion on a channel and is used for input readings [45].

At the beginning of execution, the data acquisition program reads a configuration file (a different one from the default DAS configuration file). This file contains information about the settings for the input channels (input range, sensor type, activity, number and types of applied filtering functions) and is stored under the **/usr/** directory (see Appendix A). The configuration file also contains common information about sampling frequency (how often data is collected expressed in Hz), scanning interval, trigger values, etc. If the configuration file is incorrect, suitable error messages are displayed and the data acquisition program can not work properly.

After board initialization (**das_open**), the program uses **das_ad_set_range** function to establish proper channel settings. Data collection is performed in a loop: active input channels are polled consecutively using **das_ad_sync**. The sampling frequency is achieved by applying the **delay** QNX system function right after the final sampled channel. This provides "sleeping" of the current QNX running task for a calculated time interval expressed in ms, i.e. user specified frequency. Next, polling is applied after the sampling delay. The data collection cycle is completed when the configured scanning time is over. A 2-dimensional table is used for storing channel inputs (see Table 4).

Table 4: Stored sensors input

Channel	Sampled Points			
	0	1	....	6000
0	1014	1012	....	980
1	560	562	....	578
...	....	....	....	...
16	971	975	....	1100

After accomplishing data acquisition, the program creates a parallel process by applying the **fork** system function. This new process performs the data filtering (CI) task. In this case, multi-threading programming features were utilized. The main thread continues onto a new acquisition cycle, while the processing task filters the previously recorded inputs. When the data processing task finishes the filtering routine, it "kills" itself. The main program recreates the processing task each time it completes another acquisition cycle. It is clear that the CI must complete data processing before the main task finishes the current acquisition cycle. Figure 13 depicts the applied multi-threading technique.

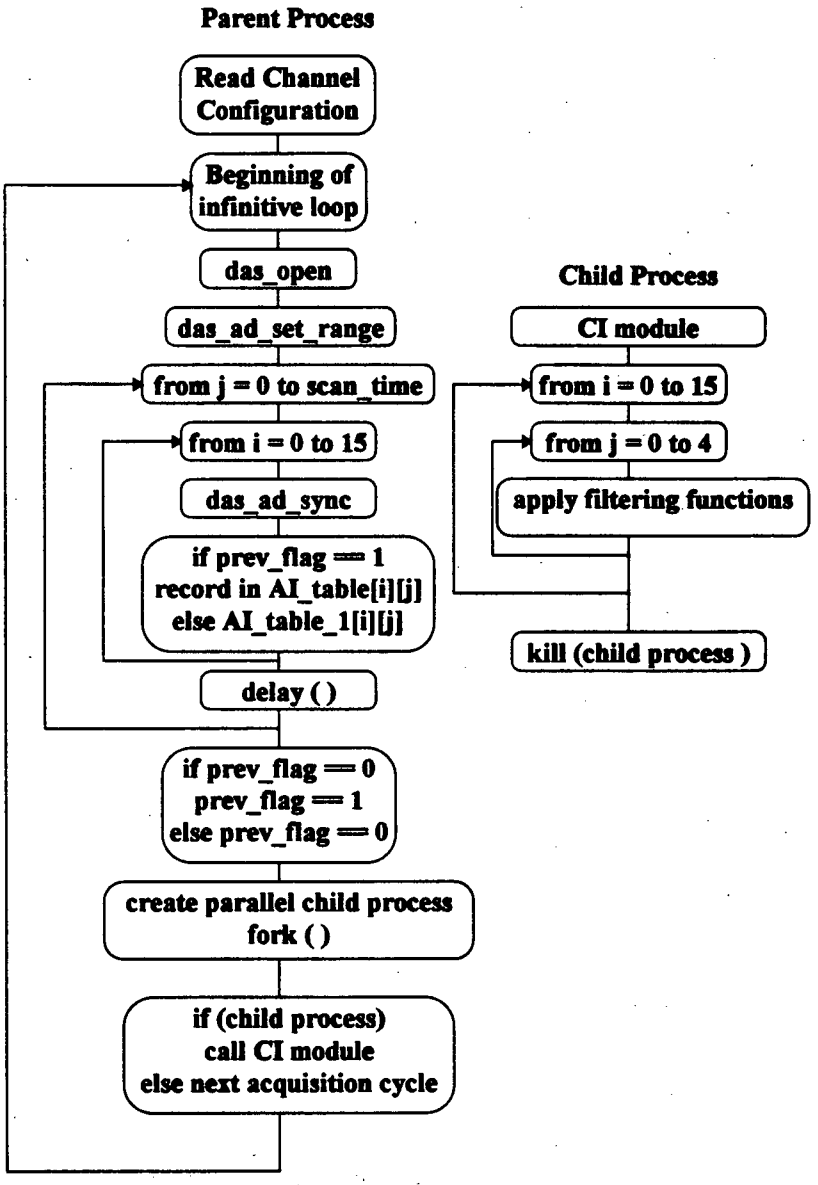

Figure 13: Data Acquisition and Data Filtering tasks run in parallel

4.5.2. Design and testing of CI module - filtering functions

The CI processing task receives pointers to the data table and sequentially filters inputs from the different channels. The configuration file provides the essential information about the functions that are to be performed on each designated input channel. This information is captured in an array of channel structures that constitute a global variable. The program runs in a loop, completing each function one by one for the first, then the second, etc., sampled channel, as illustrated in Figure 13. The number of filtering functions is limited to 5 per channel, but can be increased for future work. The CI module allows new processing routines to be appended as they are created. Genetic Algorithms, Artificial Neural Networks and FFT methods can be added to filter sensors data. The method to append a new filtering function is described in Appendix B.

Each of the applied functions use the **calibration** routine to transfer the digital presentation of analog inputs into actual values. At the end of data manipulation, the results are passed to the ProcessVision point database using the Third Party Interface Library, mentioned previously in section 4.3.

The currently available functions are:

average - calculates average(s) over a specified number of points in the recorded data table and feeds ProcessVision with this/these average(s) as key-word-triplet(s).

minmax - looks for minimum and maximum values over recorded data and passes two values to the ProcessVision point database as two key-word-triplets defined by a developer.

storedata - stores collected data, expressed in volts, in a file. Filename is defined by a user.

calibration - this function converts input data expressed in volts to actual values.

compare - this function combines data from two input channels to calculate negative strip time on-line. The function provides two outputs: negative strip time based on a mathematical expression and negative strip time based on a comparison method.

valley - this function is an example of shape recognition and feature extraction. The function looks for "valley" shapes in thermocouple data. Up to 5 valleys in the data table can be recognized and passed as 20 key-word-triplets to ProcessVision. The triplets passed for each "valley" are: temperature drop, base temperature, time of occurrence, and valley span. The total number of detected drops is also reported to the Expert System.

extreme - this routine calculates the first derivative over digital thermocouple data and records all signal extremes and their types (minimum or maximum). The function searches for "maximum-minimum-maximum" patterns in the array of the extreme structures and reports a "valley" shape to ProcessVision. Similar to the **valley** function, the triplets passed for each "valley" are: temperature drop, base temperature, time of occurrence, and valley span.

The CI routines **average**, **minmax**, **storedata**, and **calibration** are quite straight forward. However, the algorithms applied in **compare**, **valley**, and **extreme** functions are more complex and need to be explained in more detail.

4.5.2.1 Compare function

The compare function requires inputs from two process sensors: Casting Speed sensor (provided by the plant) and Linear Variable Displacement Transducer (LVDT). The received process data are first smoothed using a forward moving point average. The mould velocity profile is obtained from the mould displacement signal as a five-point-derivative (difference between fifth and first point divided by the time interval between these two points). Figures 14 - 16 presents the output from the compare function. Figure 14 displays the mould displacement obtained as 5-point moving average from the LVDT signal. Figure 15 illustrates the velocity profile procured from the mould displacement table, as a 5-point derivative. Figure 16 shows the applied technique for calculating negative strip time. The sampled points for casting speed signal and mould velocity are outlined.

The first zero-crossing from a positive to a negative value in the velocity profile table is considered as the start point in the calculation (see Figure 16). The second consecutive zero-crossing (from negative to positive) is referred to as the end of the calculation (NST end point in Figure 16). The applied function compares mould velocity and casting speed, point by point, between these two limits. Whenever mould velocity is less than negative casting speed a counter is incremented. When the calculation is completed, the content of the counter decremented by 1 is multiplied by the time interval between two consecutive points to provide the dynamic negative strip time. The negative strip time obtained by straight comparison between mould velocity and casting speed, point by point, will be from point A2 to point A3 (Figure 16). This is a result of sampling

analog inputs and presenting them in point form. An extrapolation technique is necessary to get an accurate negative strip time, i.e. from point B1 to point B2.

As well, the theoretical negative strip time (Eq. 4.1) is calculated within the same function for each data acquisition cycle. From the mould displacement table, the difference between the up stroke and down stroke, expressed in mm, is used in a mathematical expression of negative strip time. The number of points between the same two consecutive zero-crossings (from positive to negative value, or vice-verse; **Start point** *minus* **B1** in Figure 16), multiplied by the time interval between two consecutive points, produces the mould oscillation frequency. The casting speed is applied in the formula as the average speed over one second of data.

$$t_N = \frac{1}{\pi f}\cos^{-1}\left(\frac{V_c}{\pi f S}\right) \quad \text{.....Eq. 4.1}$$

where V_c is casting speed expressed in mm/s, f is oscillation frequency expressed in Hz, and S is oscillation stroke expressed in mm.

4.5.2.2 Valley function

This function was the first attempt to recognize temperature drops in thermocouple responses. Process data is filtered initially using a central moving point average over one second. The raw thermocouple data sampled at 60 Hz is illustrated in Figure 17, and the filtered trend in Figure 18. It can be noticed that the averaging method shifts data and reduces the size of drops and peaks. Other algorithms can be applied for more accurate filtering, such as Savitzky-Golay method [46], but they are more mathematically intensive. Our goal has been directed at minimizing computational delays.

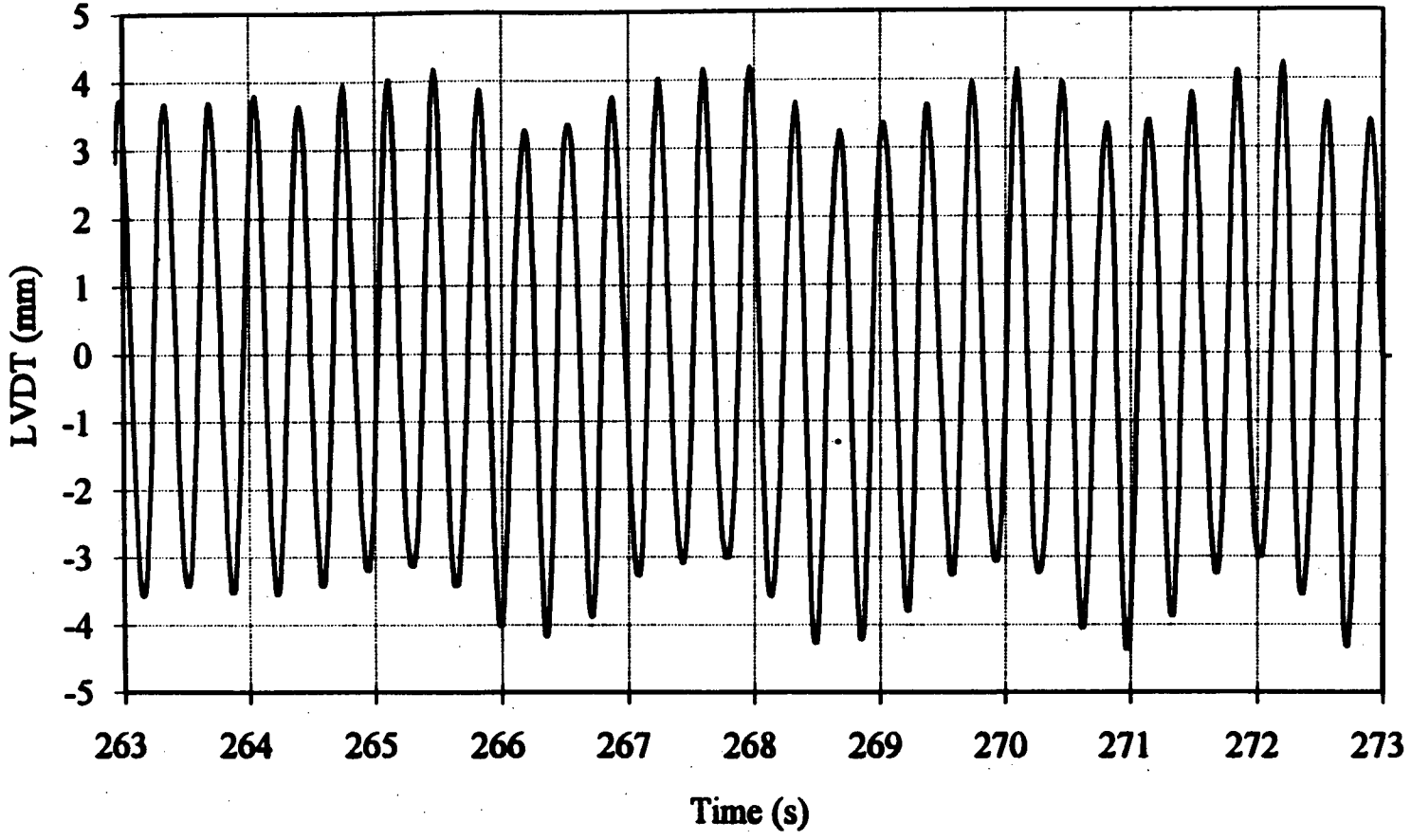

Figure 14: Mould Displacement obtained from the LVDT signal employing 5-point moving average, Heat #E33767, Company A.

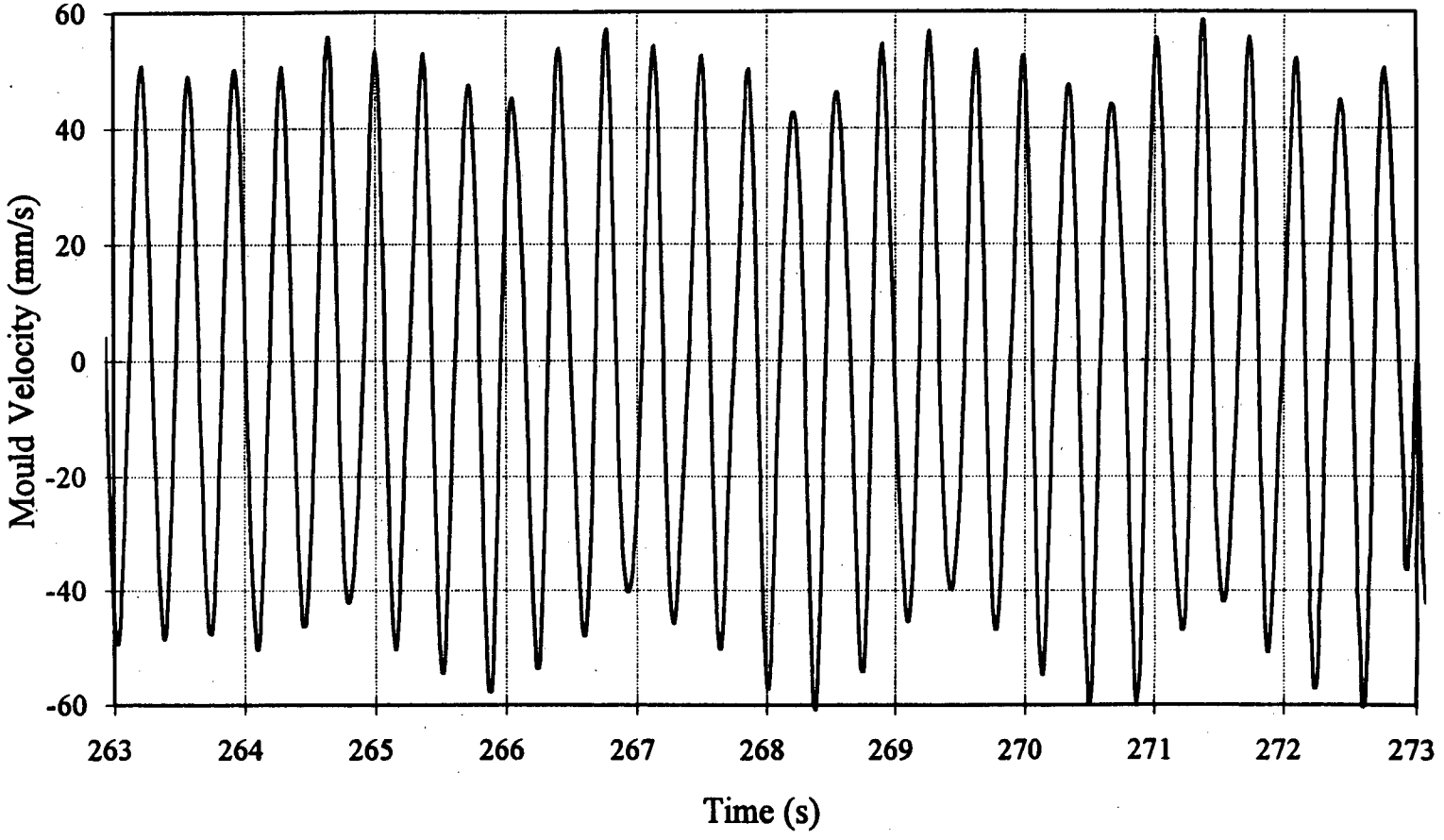

Figure 15: The calculated mould velocity using the 5-point derivative from the LVDT signal; Heat #E33767, Company A.

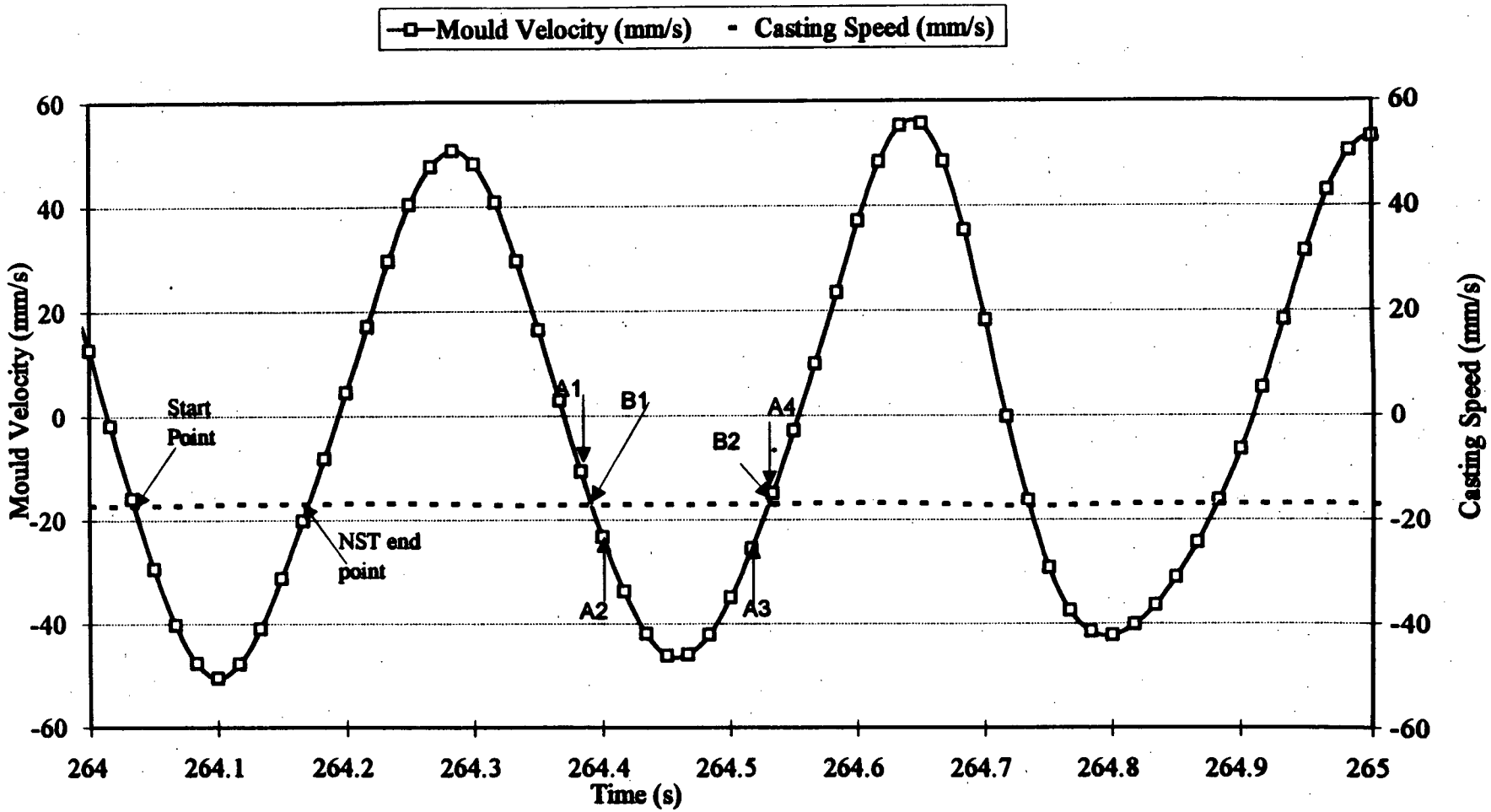

Figure 16: Graph illustrating the approach applied for calculating negative strip time using 5-point derivative; Heat #E33767, Company A.

The shape recognition algorithm consists of prior-knowledge (window size and threshold temperature drop) to direct the shape search, and a "window" technique to locate a minimum and left and right maximums. Two parameters guide the shape search and are provided by the user: the threshold temperature drop, expressed in degree C, and the probable "valley" span, expressed in seconds. As well, the span parameter can be updated in real-time by the Expert System, if the higher intelligence level concludes that the span is currently inappropriate. The delineated span is translated into the number of points used to define the "window" size. The routine detects the minimum and maximum temperature within the window, and if their difference is greater than the threshold drop, the minimum point is positioned into the centre of the window. Then the function identifies the maximum values in the right and left half, and compares to the minimum value. If both differences are greater than the threshold, the function reports the "valley" shape (see Figure 18). A span is calculated as the difference in time between the right and left maximums, the base temperature as the greater of the two maximum values, and the drop as the difference between base and minimum temperatures. The drop occurrence is reported to guide the Expert System in its search for defects. When a drop is detected, the window is moved to an adjacent position in the data table and the procedure is repeated until the whole table has been ratified (see Figure 18). The function output for a specific channel, passed to the ES is depicted in Table 5.

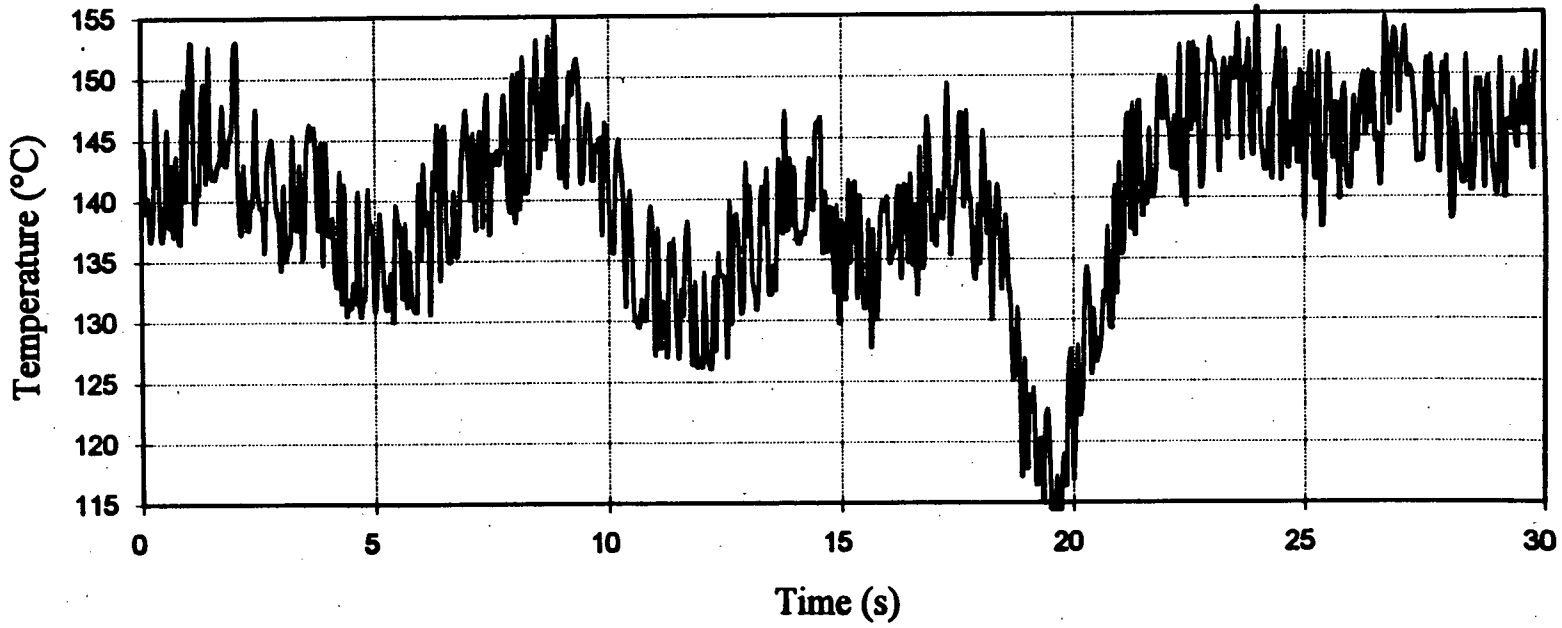

Figure 17: THC E18 raw data, Heat #333, Company D.

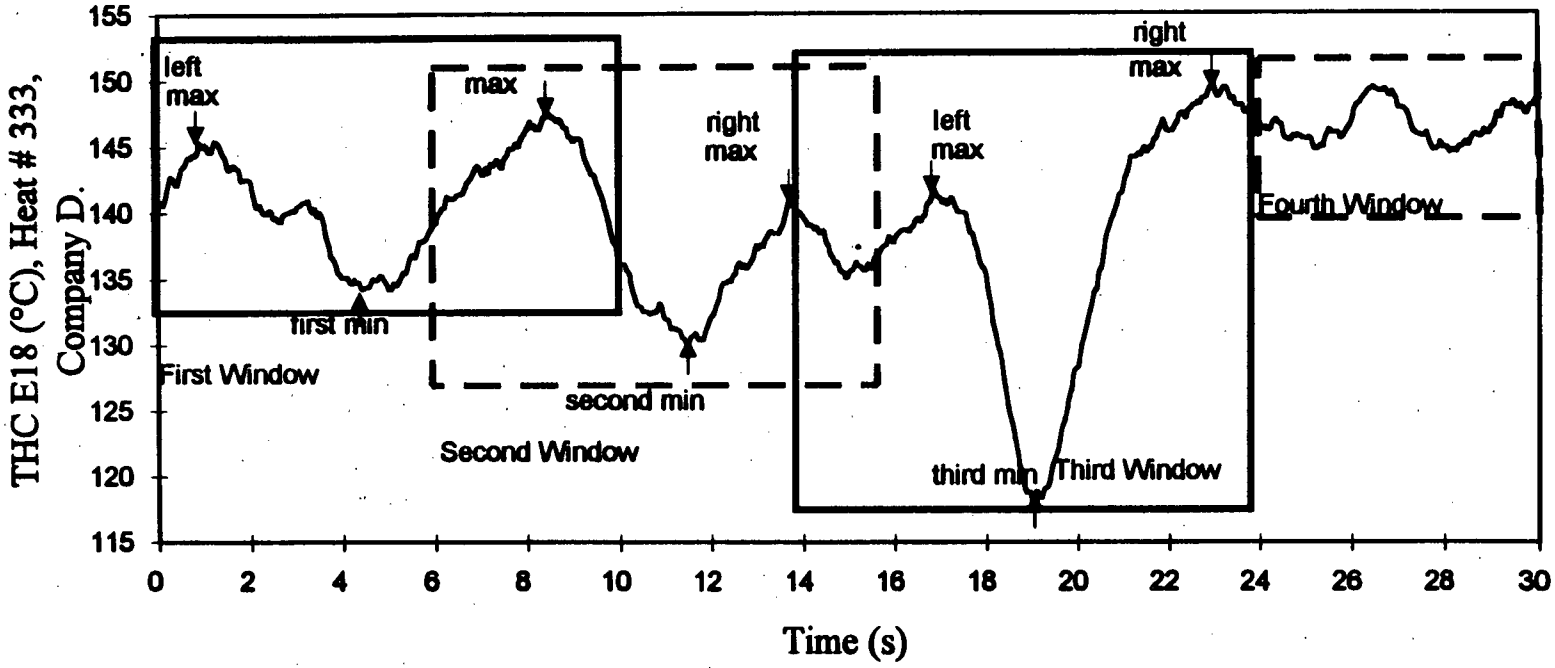

Figure 18: Applied "window" method

Table 5: The output from valley function

Keyword Triplet	Value	Degree of belief
TC1.t_drop.@f	40	100
TC1.t_base.@f	138	100
TC1.t_span.@f	10	100
TC1.t_time.@f	9	100

4.5.2.3 Extreme function

This function receives a pointer to the delineated channel and performs a forward moving-point-average. If the data acquisition frequency is 60 Hz, a 60-point moving average will be calculated, for 20 Hz frequency - 20-points, and so on. The first derivative is obtained from the filtered data, based on a time interval of one second. Therefore, the calculated first derivative mirrors every change in the trend in filtered data that takes place over one second intervals. The raw thermocouple data sampled at 60 Hz is illustrated in Figure 16, and the filtered and first derivative data are shown in Figure 19.

The first derivative is obtained in point form and stored in a new data vector. Searches for all zeros and zero-crossings in the derivative vector is accomplished to derive information about the filtered data local extremes. An array of structures is utilized to keep records of all extreme locations, values and types (in this case minimum and maximum). The program continue to search through the extreme array looking for the consecutive extremes that create the "maximum-minimum-maximum" pattern. Once the match is found, the program reports a "valley" shape, if the valley drop is greater than the threshold value provided by the user. Considering thermocouple precision and DAS 20 resolution, 5 °C is accepted as the overall system resolution. The keyword triplets

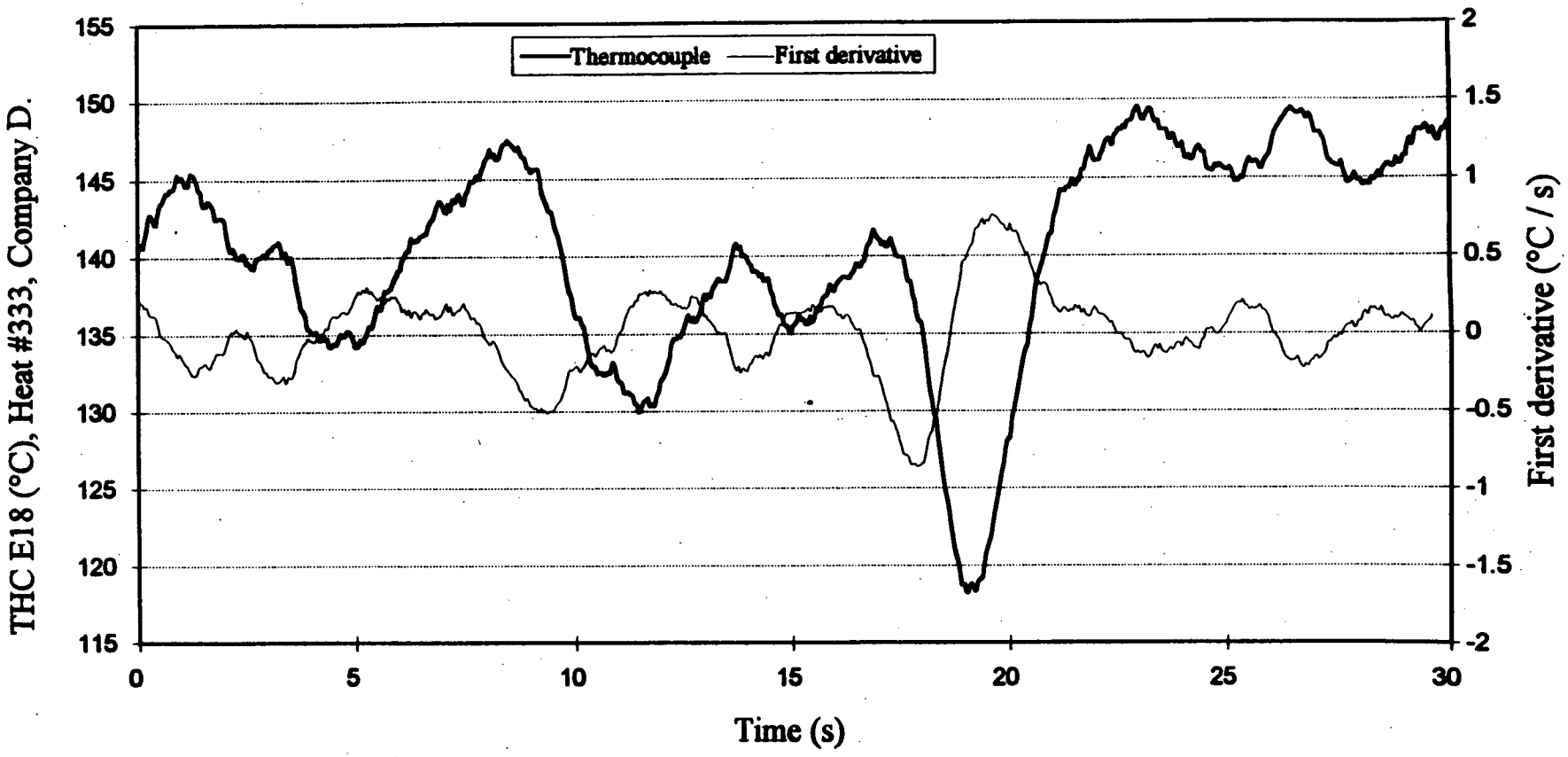

Figure 19: Applied "derivative" method

describing each detected "valley" are passed to the ProcessVision point database in the same manner as described in section 4.5.2.2.

4.5.3 Expert System development

The real-time quality control Expert System for continuous billet casting was developed at the Centre for Metallurgical Process Engineering, using the Comdale/C development tool. Comdale/C is an expert systems development shell for creating supervisory control modules within ProcessVision applications. It is an integral part of the overall system. Communications between the billet casting Expert System and CI software is accomplished through the point database as depicted in Figure 20. The RAM-resident point database acts as a medium where different real-time modules exchange their information.

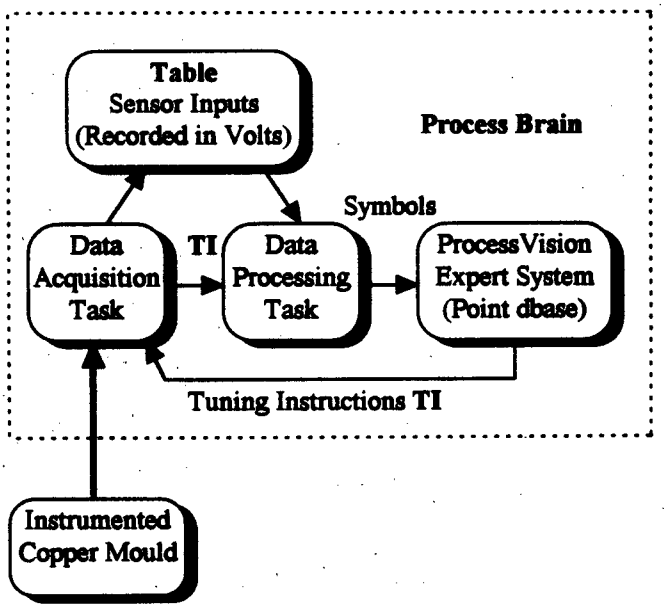

Figure 20: Multitasking concept of the "Smart" Mould

The research team at UBC has been working on interpreting patterns from sensor responses taken from numerous field trials conducted over the past 20 years. Specific curve shapes from thermocouple (THC) time responses (temperature peaks, drops, etc.) were found to be related to specific billet defects. At this point, these correlations make up a part of the knowledge base for on-line detection of the following surface defects: bleeds/laps and depressions.

It was discovered that when successive temperature drops and rises propagate down the mould during casting, this indicates that some particular defect has occurred. The ratio between temperature drop and the base temperature measured before the drop defines the significance of the drop. The time interval between a drop and rise is also very vital information. The relative drop and time span are two features extracted from the thermocouple responses that define the extent of the defect created.

To predict bleeds/laps and depressions, the knowledge base examines these features. A data acquisition rate of 5-20 Hz is adequate to capture all important changes related to these defects. The "pseudo" real-time Expert System is not designed to process input data intelligently at this rate. So, applying a CI module was a necessity.

The knowledge base is currently designed to trace 4 thermocouple signals per face to predict bleeds/laps and depressions. Initially, the Expert System looks for the "start" flag, set up by the CI task at the end of the filtering process. When the "start" flag becomes 1, the ES counter is set to the number of reported drops in the first THC (THC1), and the ES checks the first detected drop from THC1. Fuzzy logic is applied to present the significance of the detected drop. The source in the fuzzy presentation is the

ratio between the temperature drop and base temperature, expressed in percentage. Figure 21 illustrates the shape of fuzzy membership function for temperature drop.

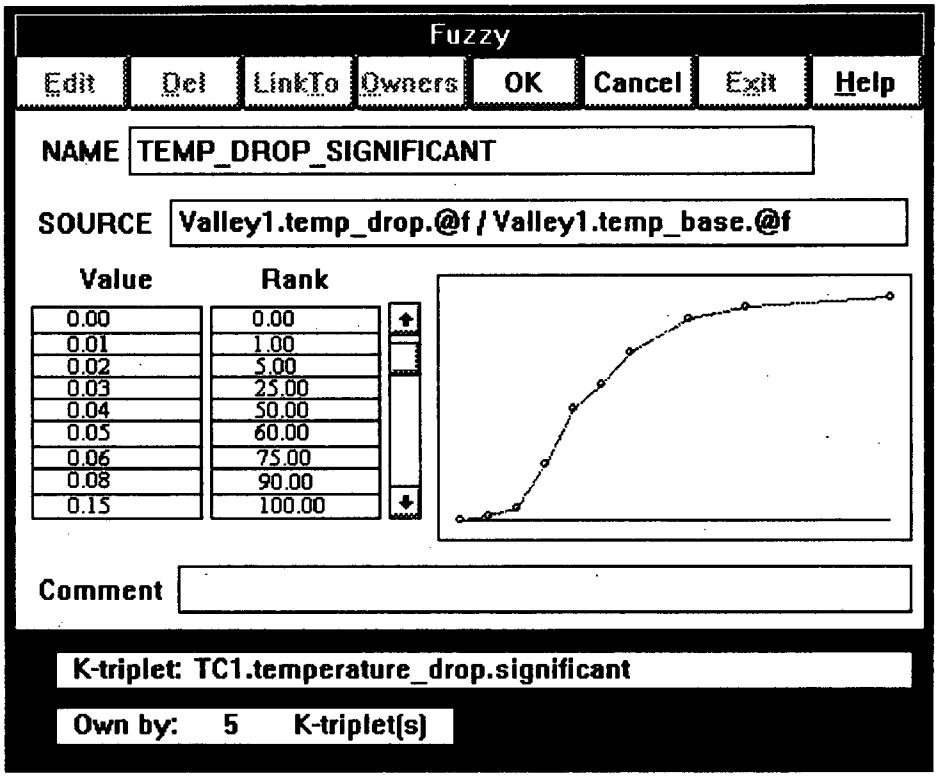

Figure 21: Fuzzy set used for presenting significance of a temperature drop

When the first drop is depicted by its significance value, the ES checks the output from the second THC (THC2). The same drop has to be seen at the THC2 to increase the degree of belief that a certain billet defect will occur. Equation 4.2 presents the applied formula to express the degree of belief for the predicted defect.

CONCLUSION=
(CERTAINTY (TC1.temperature_drop.significant) * 0.60 +
CERTAINTY (TC2.temperature_drop.significant) * 0.15 +
CERTAINTY (TC3.temperature_drop.significant) * 0.15 +
CERTAINTY (TC4.temperature_drop.significant) * 0.10) ..... Eq. 4.2

The applied weights in equation Eq. 4.2 (0.60, 0.15, 0.15, and 0.10) are derived from the billet casting experts. An ANN or Genetic Algorithm can be employed to define these coefficients, based on recorded process data that reflects the real defects, in future revisions of the "Smart" Mould.

If the position of the drop reported at THC2, is within certain time boundaries, defined as (*position_of_THC1* - *position_of_THC2*) [mm] / *casting_speed* [mm/s] + 0.5 * *THC1_span1*, the drop is considered as the same one seen at THC1. The search continues on until the outputs from all four THC are rectified. The counter is decreased every time a search routine is completed for a delineated drop. When the ES counter becomes equal to 0, the ES provides a list of detected defects along with degrees of belief., through the ExpertView module. If the system concludes that some of the drop searches can not be completed, because the drop detected at the first THC, did not have time to appear at the next thermocouple, the ES saves the necessary information in a data file, records the search status as incomplete, and after the next acquisition cycle, starts the search procedure from thermocouple number 2. Figure 22 presents a typical rule used in the ES to describe a valley that propagates down the mould.

```
Rule
@name = TC1
IF          TRUE
THEN Valley1.temp_drop.@float = Thermocouple1_drop1.drop_size.@float
THEN Valley1.temp_span.@float = Thermocouple1_drop1.span_size.@float
THEN Valley1.temp_base.@float = Thermocouple1_drop1.base_temp.@float
THEN Valley1.position.@float = Thermocouple1_drop1.position.@float
THEN valley_span.avg_value.@float = Valley1.temp_span.@float
THEN FIND ( "TC1.temperature_drop.significant" )
THEN time_range.avg_val.@float = Thermocouple1_drop1.position.@float + (Thermocouple2.location.@float -
Thermocouple1.location.@float ) / casting.speed.@float
THEN time_range.min_val.@float = time_range.avg_val.@float - 0.5 * Thermocouple1_drop1.span_size.@float
THEN time_range.max_val.@float = time_range.avg_val.@float + 0.5 * Thermocouple1_drop1.span_size.@float
THEN TC1.rule_examined.variable is TRUE
THEN FREERULE ( $Rule, "time_range_*" )
THEN MACRO ( "time_range_*" )
THEN MACRO ( "check_time_range" )
endRule
```

Figure 22: The rule applied for tracing the THC drops

4.5.4. Design of the Man Machine Interface

The continuous casting application was initially designed by creating the configuration files needed by the individual ProcessVision (PV) modules (see Figure 10). Each module within PV which uses data, upon request for data values or upon writing of data values to the point database, causes these items to be created and to be an integral part of the overall system. This data, assigned to a unique key-word-triplet, is then accessible to any other module that needs it. A set of process graphics was designed to describe the process and reflect real-time data as it changes in the physical process. A view of the current state of a process is provided by the modules contained within the Process Interface. PV holds information in short-term memory that represents the current state of the process and/or historical data to allow real-time trend analysis.

The configuration files typically include details of the following process parameters:

-Alarm Monitoring of process conditions.

-Graphic display of process conditions.

-Scheduling of time dependent events such as data logging and scheduled process checks.

-Application Message Class definition for warning message filtering.

-Definition of Control strategy.

-Explanation of Control strategy.

Each module is instructed in its role in the execution of the application through the development of the system configuration files.

The minimum number of modules for any application is four (4): *cc_admin*(to start an application and to monitor the health of all active modules), *db_admin* (for locating data), *pd_dbase* (holds a snapshot of the current state of process variables at anytime), *mg_admin* (manages messages generated by the system for the human operator). The configuration files must be defined for each module used in an application except *cc_admin*, *db_admin*, and *pd_dbase* modules [47].

The Process View module is a windows-based graphical Man Machine Interface. It allows the operators to interact with the process by displaying graphical representations of process data. The Process View Editor is used to configure Process View groups (graphic display windows). All Process View groups (files) have the extension .grp. Groups are interconnected by specifying which group to display when symbols and buttons are clicked on by the user during execution of Process View.

The icon bar is used to access the different dynamic objects that are available to the developer. Dynamic objects can be designed to display active process data, change process data values and access other functions and graphic screens. The dynamic objects used for portraying active data are trends, meters, sliders, dials, bars, and gauges. Buttons, text objects, and symbols, designed by the developer, are used for linking different graphic screens within the application. Moreover, the buttons are used to run the procedures within the ES, to set up search boundaries (Tuning Instructions from Figure 19) and signals (start and stop flags for recording process data) for the CI processing functions.

Trends are applied to present dynamics of several process variables concurrently, providing better understanding of the complex relationship among casting speed, metal level, negative strip time, mould displacement and thermocouple signals.

ProcessVision also includes a Hypertext development tool. The continuous casting hypertext is a vital part of the application and contains necessary information about the billet casting process. It is an electronic book with delineated "hot" text and image objects that open a new page or provide a short description of the specified topics.

Data recorded during one of the last plant trials, were used for creating a demo version or simulation of the continuous casting process. The data acquisition program reads channel inputs from files which stored the original data, instead of from the DAS 20 board. It creates the data processing task (CI module), and the CI module passes filtered data to the PV application. The MMI output from the demo version is presented in Figures 23 - 26.

The application starts with the screen illustrated in Figure 23. It presents all components of the billet casting process. The process is divided into 5 segments: tundish, mould, spray zone, pinch-rolls and shear/torch zone. The specific part of the process can be accessed by positioning and clicking on the button assigned to this component. To initiate the ES search for billet defects, the button "Check bleeds/laps & depressions" has to be applied. The Hypertext document can be accessed by clicking on "Description" button. The process trends can be opened by pressing the "Trend" button.

Figure 24 depicts four simultaneous process trends. Trend 1 is the thermocouple above the meniscus. It portrays the metal level position at the delineated face. Trend 2

gives the metal level signal obtained from the plant sensor, plotted from the bottom of the mould. Trend 3 represents the casting speed and mirrors the metal level signal. Trend 4 illustrates the calculated negative strip time.

Figure 25 consists of several opened windows, each of which outline a different aspect of the process. The HyperDisplay screen is an integrated part of the application and provides concise information about the billet casting process. A novice operator can click on the hot text or objects on the hypertext window to obtain necessary information related to running the process or particular maintenance issues.

Figure 26 presents the calculated t_N based on equation Eq. 4.1, and the actual casting speed and mould velocity profile.

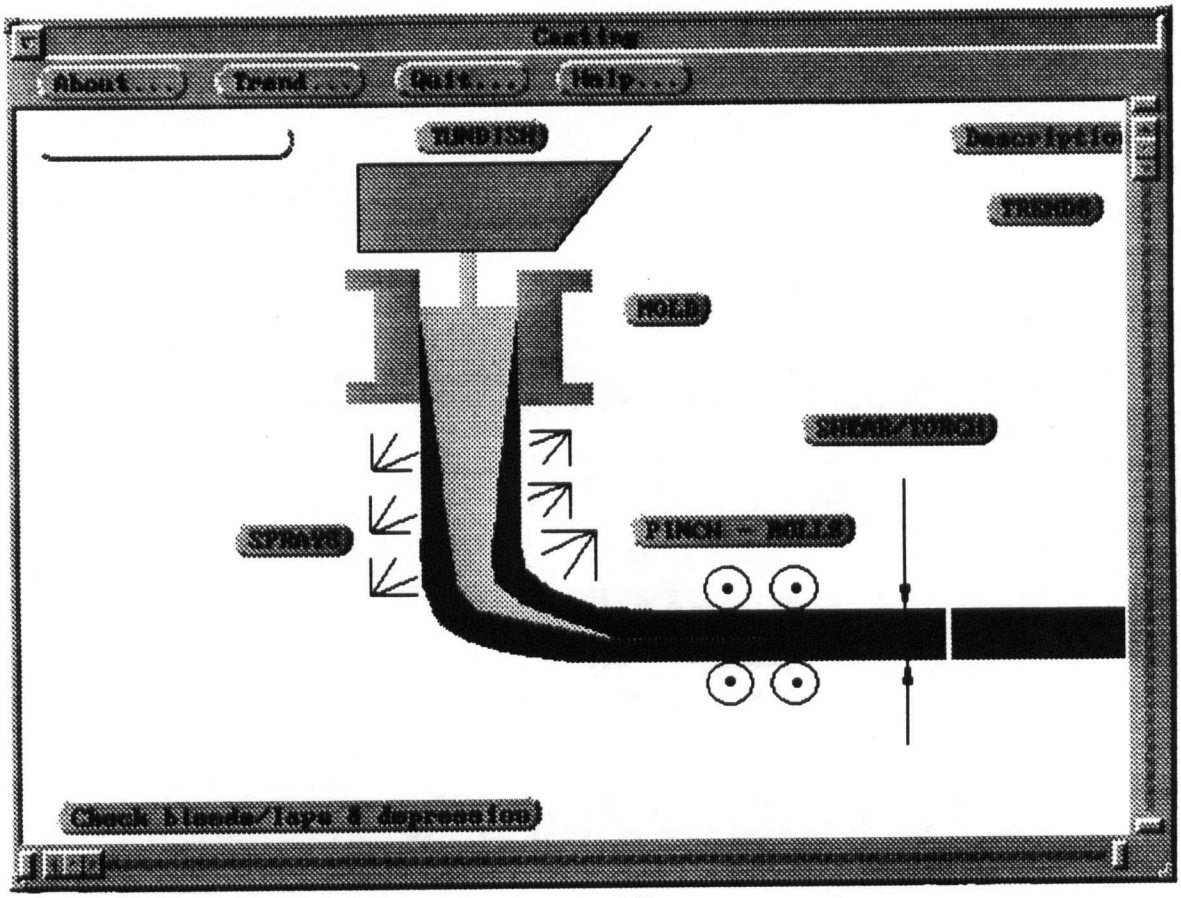

Figure 23: "Smart Mould" Introductory screen

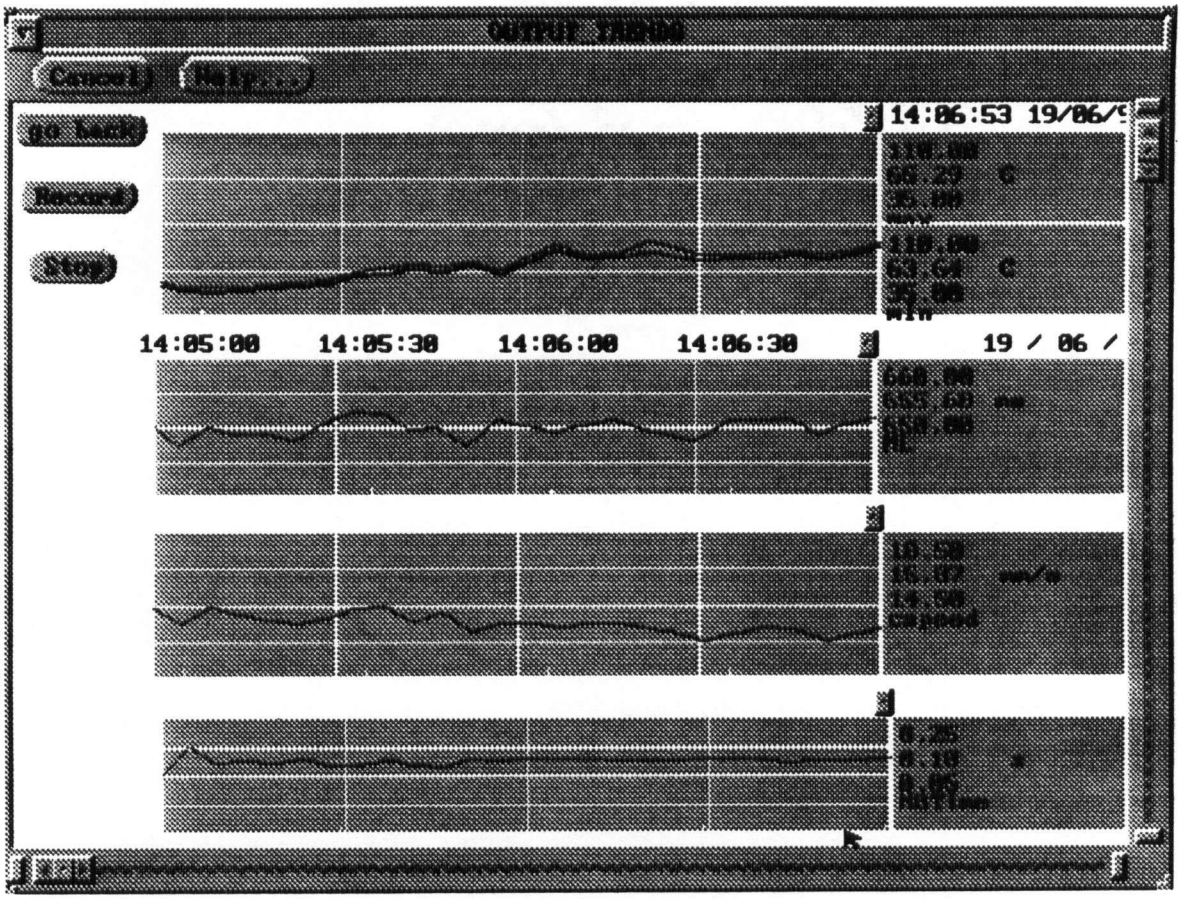

Figure 24: The four process trends: THC above meniscus, Metal Level, Casting Speed and Negative Strip Time

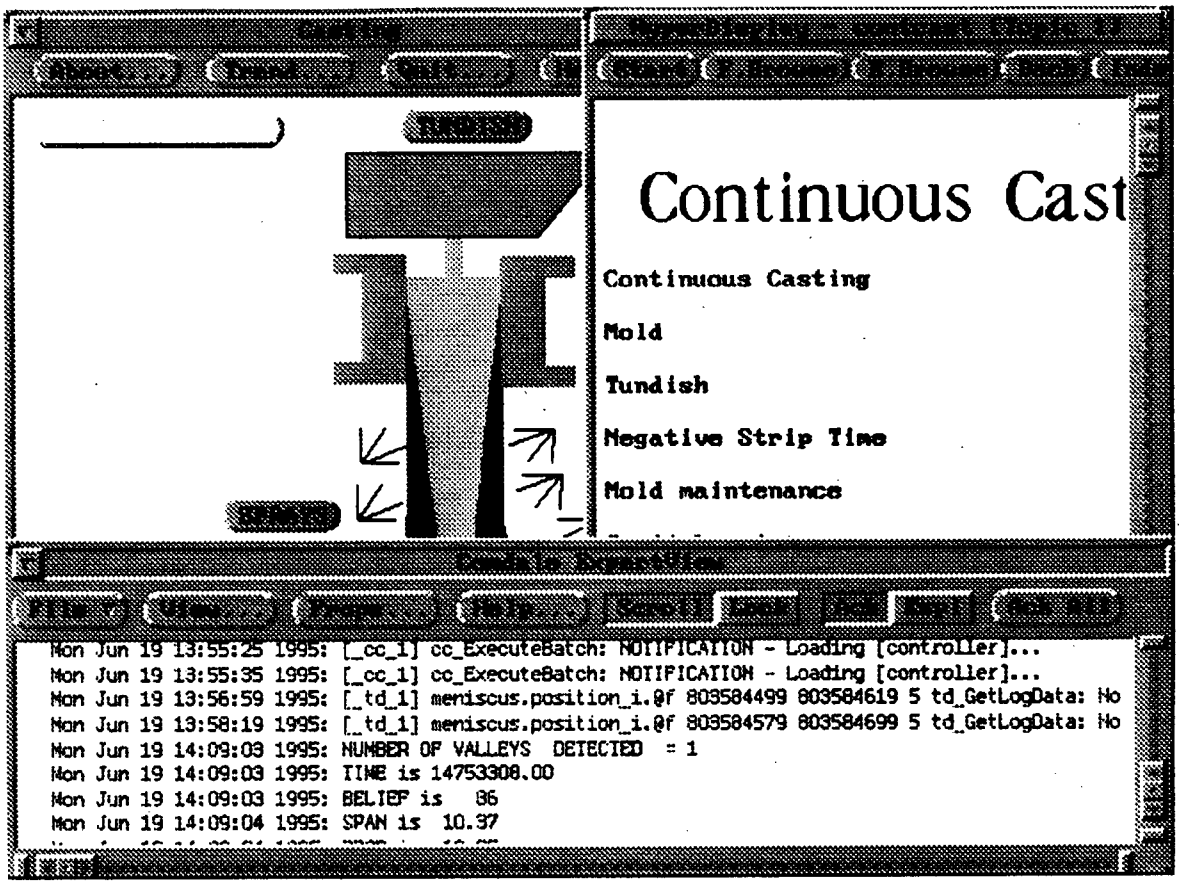

Figure 25: "Smart Mould" application with opened ExpertView, ProcessView and Hypertext modules

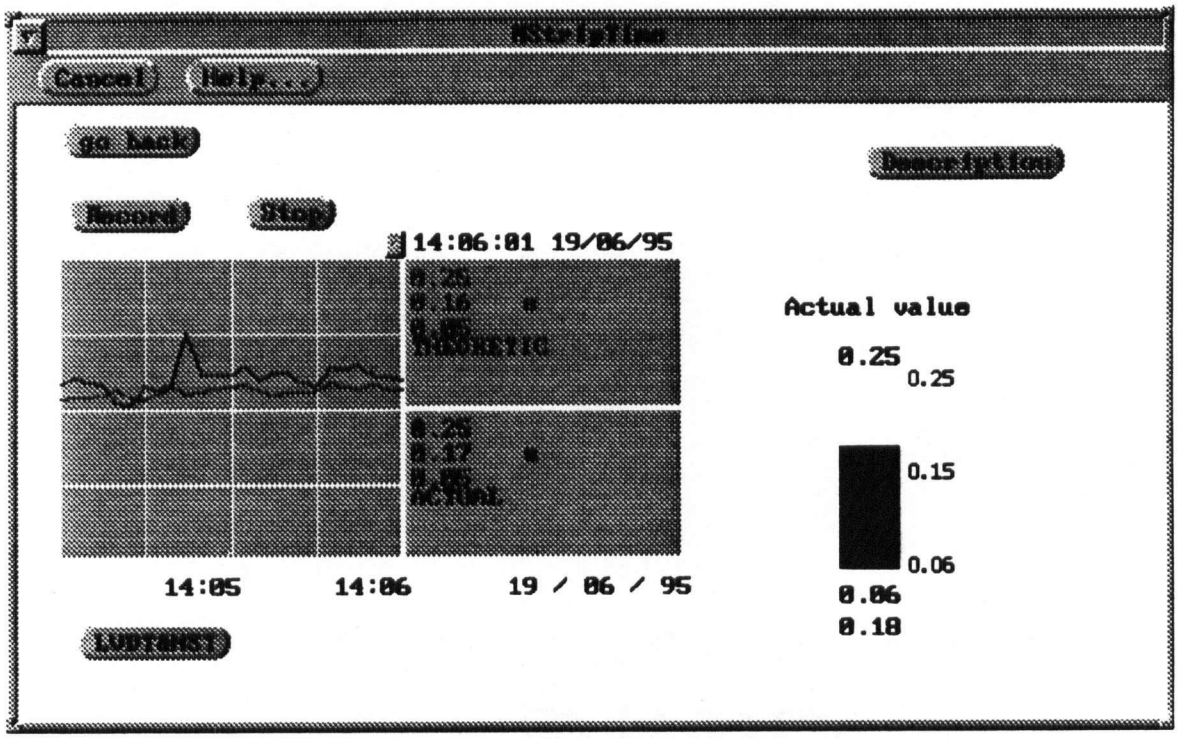

Figure 26: Negative Strip Time trend

Chapter 5

Results and Discussion

5.1 Experimental Procedure

The developed Supervisory Control and Data Acquisition System for continuous casting of steel billets has been implemented and tested at two Canadian steel mini mills. The trials were conducted to test the SCADA system hardware components, their performance in an industrial environment, and software robustness and correctness under plant operating conditions. The software evaluation was divided into three steps:

- Testing the precision of data acquisition.
- Testing the correctness of applied filtering algorithms.
- Testing the truth of the applied logic and predictions made within the pseudo real-time Expert System.

To test the SCADA system, sensor data was collected using two parallel Data Acquisition Systems. The original UBC acquisition system, consisting of DAS-8 board and 8 expandable boards (EXP-16), recorded data in parallel with the SCADA system, using a commercial software package called "Labtech Notebook for Windows". Data was stored in a file so that at a later time, data manipulation was available to check the output from the CI module. The hardware configuration of the two systems working in parallel is depicted in Figure 27.

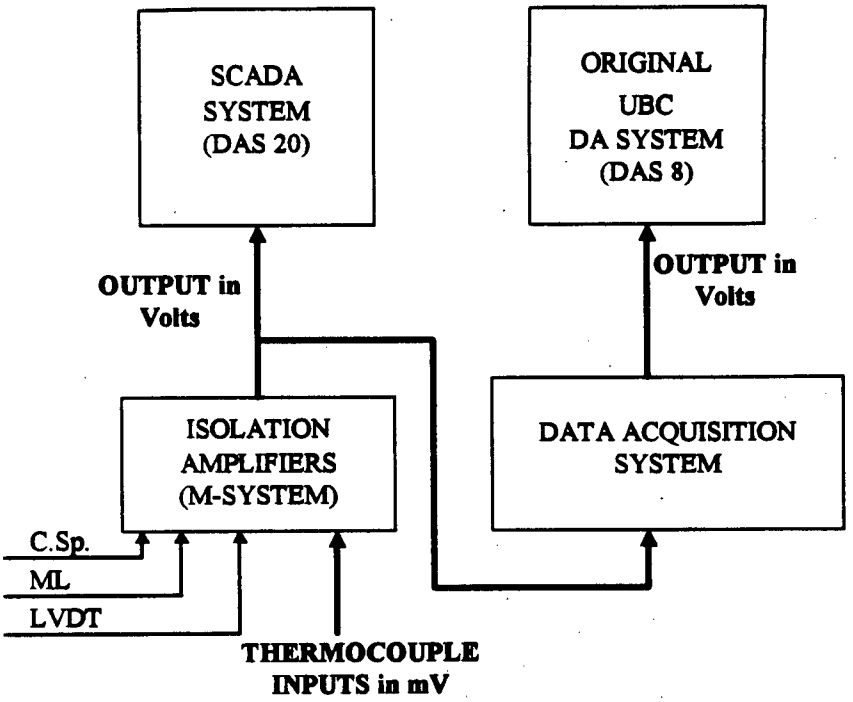

Figure 27: Hardware configuration for simultaneous data acquisition

During the first plant trial, the UBC acquisition system acquired data at 60 Hz for nine sensors that included casting speed, metal level, LVDT and six mould thermocouples. The SCADA system sampled data at 200 Hz for casting speed, metal level, LVDT and at 60 Hz for the six selected thermocouples. Thermocouples were located on the south side of the mould at the following positions: 210, 235, 260, 290, 320 and 420 mm from the top of the mould. The casting speed and metal level signals were used for on-line calculation of negative strip time. The thermocouple inputs were applied to on-line detection of billet defects, such as bleeds, laps and transverse depressions.

The continuous casting process was monitored for four heats that used powder for mould lubrication. The data was recorded along with the output from the SCADA System and CI module. Also, billets were inspected to correlate SCADA results with actual defects.

The second plant trial was conducted to map the prediction from the ES to real surface defects and to tune the fuzzy sets defined to describe belief in a significant surface depression. Again, the UBC acquisition system collected data in parallel in order to justify the SCADA readings. Afterwards data analysis from the first plant trial led to the conclusion that a 60 Hz sampling frequency for the LVDT, casting speed, and metal level signal, and 20 Hz for thermocouple sensors were adequate. The SCADA outputs were presented on the screen as trends, in order to establish correlation with visual observations of metal level and casting speed changes, as well as surface defects. The system was used for monitoring several powder and oil heats. Predictions derived from the ES were recorded for 3 oil heats and the corresponding billets were inspected and measured to confirm the ES predictions.

The hardware components of the SCADA system (plug-in DAS 20 board and isolation amplifiers) worked satisfactorily in the plant environment. The overall system, (data acquisition driver, the CI module, pseudo real-time ES and PV application), ran for several heats continuously, without any failure. Data acquisition software was accurate and robust. The collected data set mirrored the parallel one gathered by the Labtech Notebook for Windows software system. Output from the CI module, presented as trends within ProcessVision, was accurate and corresponded very well to the visual observations.

The thermocouple data collected by the SCADA along with the isolation amplifiers, did not contain any irregularities. In absence of the isolation amplifiers thermocouple signals exhibited sudden and unpredicted maximum negative values (due to probable floating ground effects). Thus, isolation amplifiers must be used for the all sensors in the "Smart" Mould.

5.2 Data acquisition software

The applied polling method in the data acquisition program along with the **delay** QNX system function defines the upper limit for sampling frequency per number of channels being read.

In the multi-tasking environment, whenever a particular task is in a "sleep" mode (discussed in section 4.5.1), it does not consume CPU time; the other tasks share the microprocessor time. When the specified time delay is over, the "sleeping" task is ready to use the CPU time. The task acquires CPU resources within one tick. Ticks are system timer interrupts used only for software timers. If the QNX tick size is set to 0.5 mS (the minimum value), the total delay time in mS will be: *specified_delay + 0.5 [mS]*.

At the beginning of each acquisition cycle the program reads the first active channel one thousand times, to establish the necessary time for a single reading. This time depends on hardware configuration and on how busy the system resources are. Usually, this single reading takes around 0.132 mS. This figure is multiplied by the total number of active channels and is then added to *specified_delay + 0.5 [mS]*, to get the real sampling time. The data acquisition program accepts a user defined sampling frequency from the

configuration file, and calculates *specified_delay* in a way to obtained the correct acquisition rate.

The data acquisition program records the beginning of each acquisition cycle. At the end of each polling procedure, the beginning time is compared to the current time. If the time difference is greater than the user specified scanning interval, the acquisition cycle is abandoned. The program calculates the real sampling interval (i.e. frequency) based on the recorded time difference divided by the number of sampled points. The actual scanning interval may differ from the desired one by several mS up to 40 mS.

Figure 28 presents the maximum available sampling frequency based on number of input channels, for a 486-DX2, 66 MHz microprocessor. As shown, the designed hardware/software combination is able to provide extremely rapid sampling rates (between 270 and 600 Hz) depending on the number of input channels being used.

5.3 Results from Negative Strip Time calculation

The CI output for trend presentation is calculated and updated every second, by applying suitable filtering functions. The trend update interval is limited to 1 second by the ProcessVision development tool. The data acquisition task reads the input channels for 1 second at 60 Hz (or 100 Hz), creates the processing task, and then the processing task sequentially filters data and passes on the corresponding keyword-triplet values to the associated trends within ProcessVision. Trends are available for metal level signal, casting speed, LVDT maximum and minimum position, negative strip time (t_N) and thermocouple temperatures, located above and below the meniscus. The output from the negative strip

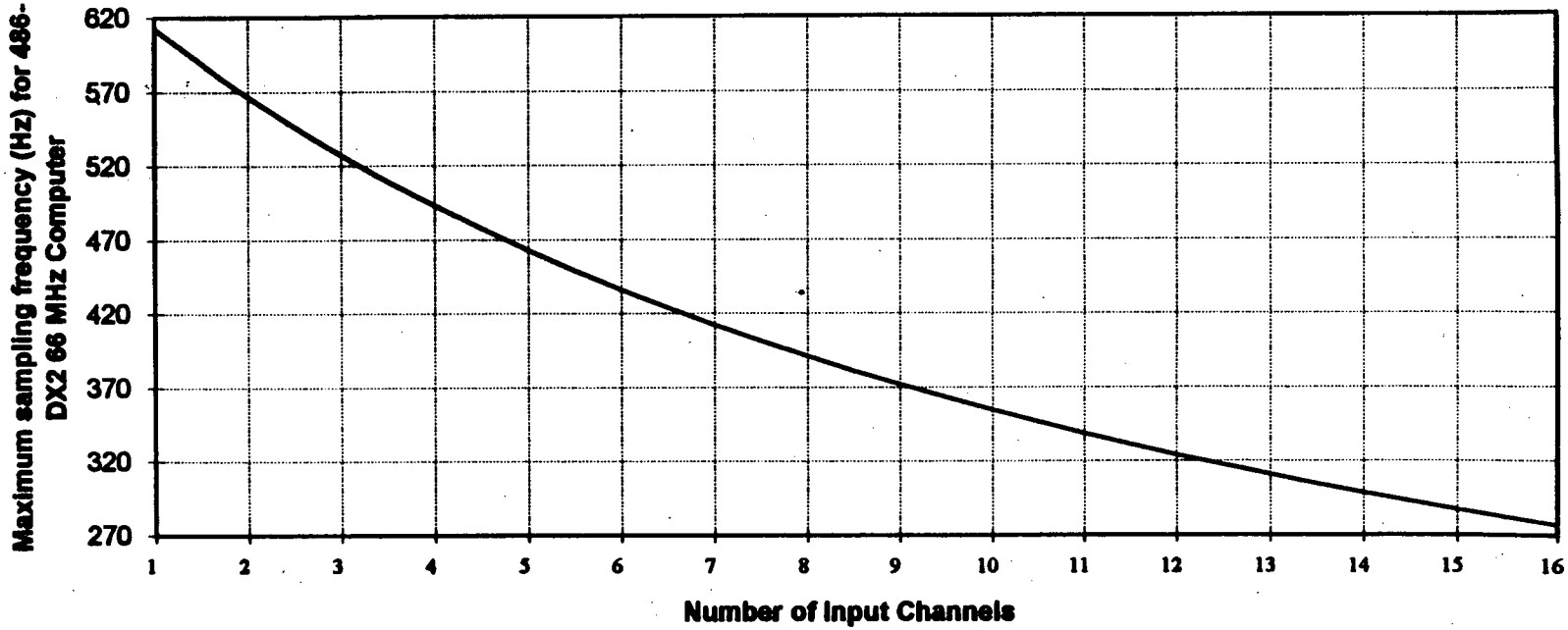

Figure 28: Maximum Data Acquisition Frequency per Number of Input Channels, per single task

time calculation is also saved to a data file, along with average casting speed and LVDT maximum and minimum values. This file was used later to analyze the t_N calculation.

Figures 29 - 33 present the output from the negative strip time calculation for data collected at 60 Hz, during the first plant trial conducted in November 1994. The observed heat was cast with powder lubricants and the set-up values for casting speed, oscillation frequency and stroke length were 40 inches/min (around 17 mm/s), 160 cpm (2.67 Hz), and 8 mm, respectively.

Figure 29 portrays the changes in minimum and maximum values in the LVDT signal, over 10 minutes of casting conditions. As can be seen over the interval from 250 to 350 seconds, the displacement signal becomes erratic at two distinct time periods. These are rather instantaneous changes, and perhaps, result from mould-strand interaction (sticking or binding problems) or due to some external oscillator upsets. These unexpected changes are reflected in the negative strip time calculation shown in Figure 30. The variation of t_N is very important for the billet quality and is examined in parallel studies.

Figure 30 displays the influence of mould displacement changes on negative strip time and also portrays the difference in the two methods applied to calculate t_N discussed in section 4.5.2.1. It can be seen that the 5-point-derivative method is more sensitive to stroke length changes than that derived from the standard mathematical equation (Eq. 4.1). The 5-point-derivative method is more accurate because it does not assume a consistent sinusoidal displacement signal; it tends to stress the influence of abrupt changes in mould displacement. Either calculation can provide instantaneous t_N values and can be used to monitor oscillator behaviour, stroke and billet quality, however the 5-point

derivative method clearly provides a more sensitive and accurate representation of the "true" t_N value.

Figure 31 presents the casting speed signal together with output from the two t_N calculations. It is clear, that over this time interval, the casting speed does not vary much and so does not influence changes in t_N. The changes in the negative strip time mirrors variations in the mould displacement signal.

Figure 32 illustrates the influence of casting speed on the negative strip time over 600 seconds (10 minutes) of casting conditions. It can be seen that casting speed continually decreased over 400 seconds and caused a gradual rise in t_N. The steady decline in casting speed was perhaps, a result of the decrease in metal level (ferro-static pressure) in the tundish due to the startup of another strand.

Figure 33 presents the behaviour of the meniscus and casting speed during this time. It is obvious that the meniscus position did not change significantly over this period. The PID controller, used to regulate the casting speed to keep the meniscus position steady, maintains constant metal level.

An attempt was made to derive the optimum sampling frequency and first derivative for the LVDT signal to obtain reliable negative strip time. It is common sense to have as high a sampling frequency as possible to map most accurately an analog signal into a digital one. The first derivative from such a signal would also be the most precise. However, in the plant environment, the higher the frequency, the more noisy will be the signal received. According to Shannon's sampling theorem [48], the sampling frequency should be at least 2 times greater than the frequency of variations in the sampled analog

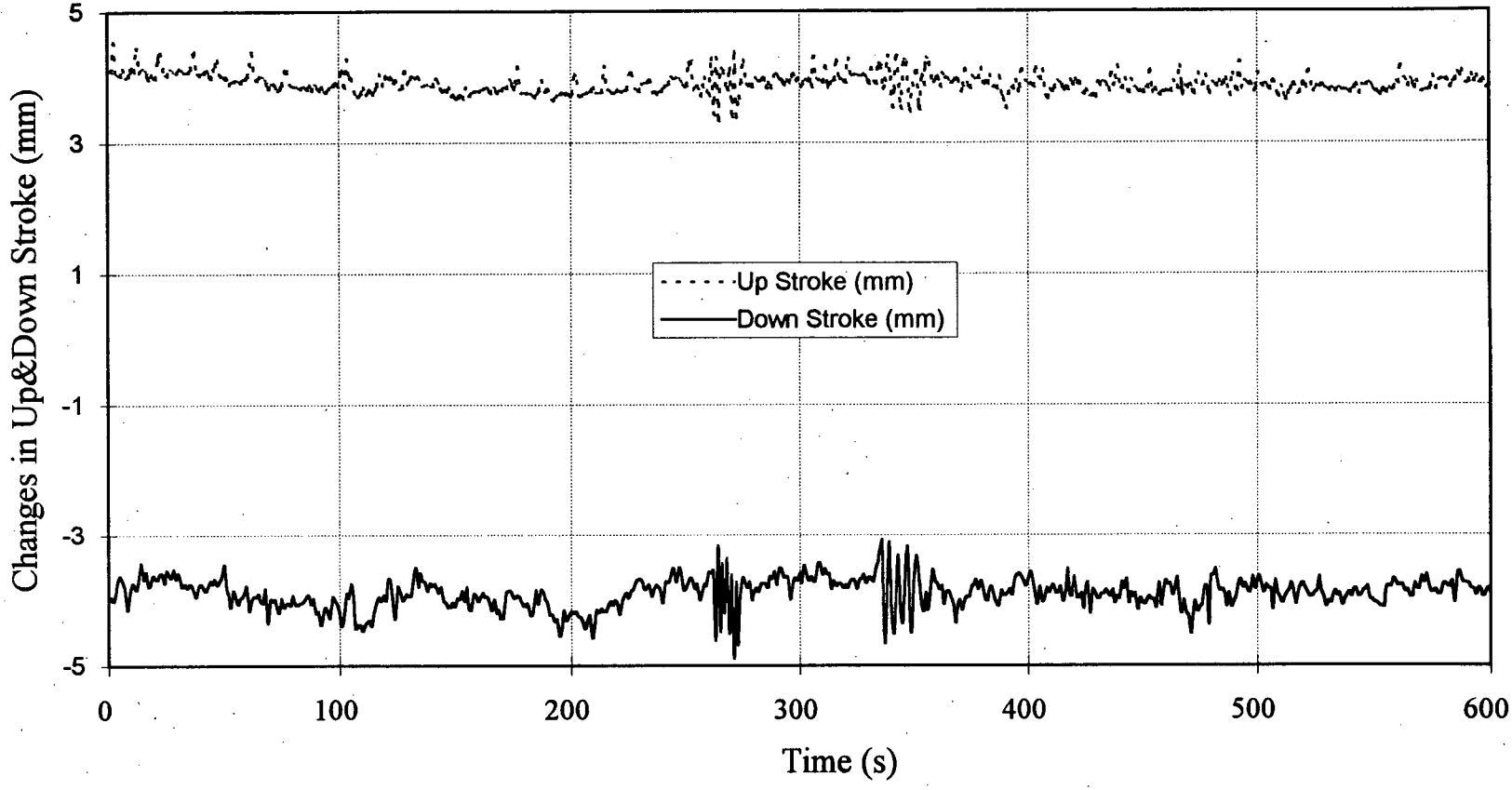

Figure 29: Variations in the minimum and maximum stroke position (LVDT signal) over a 5 minute duration in Heat #E33767, Company A. Note the erratic behavior at 250 and at 350 seconds.

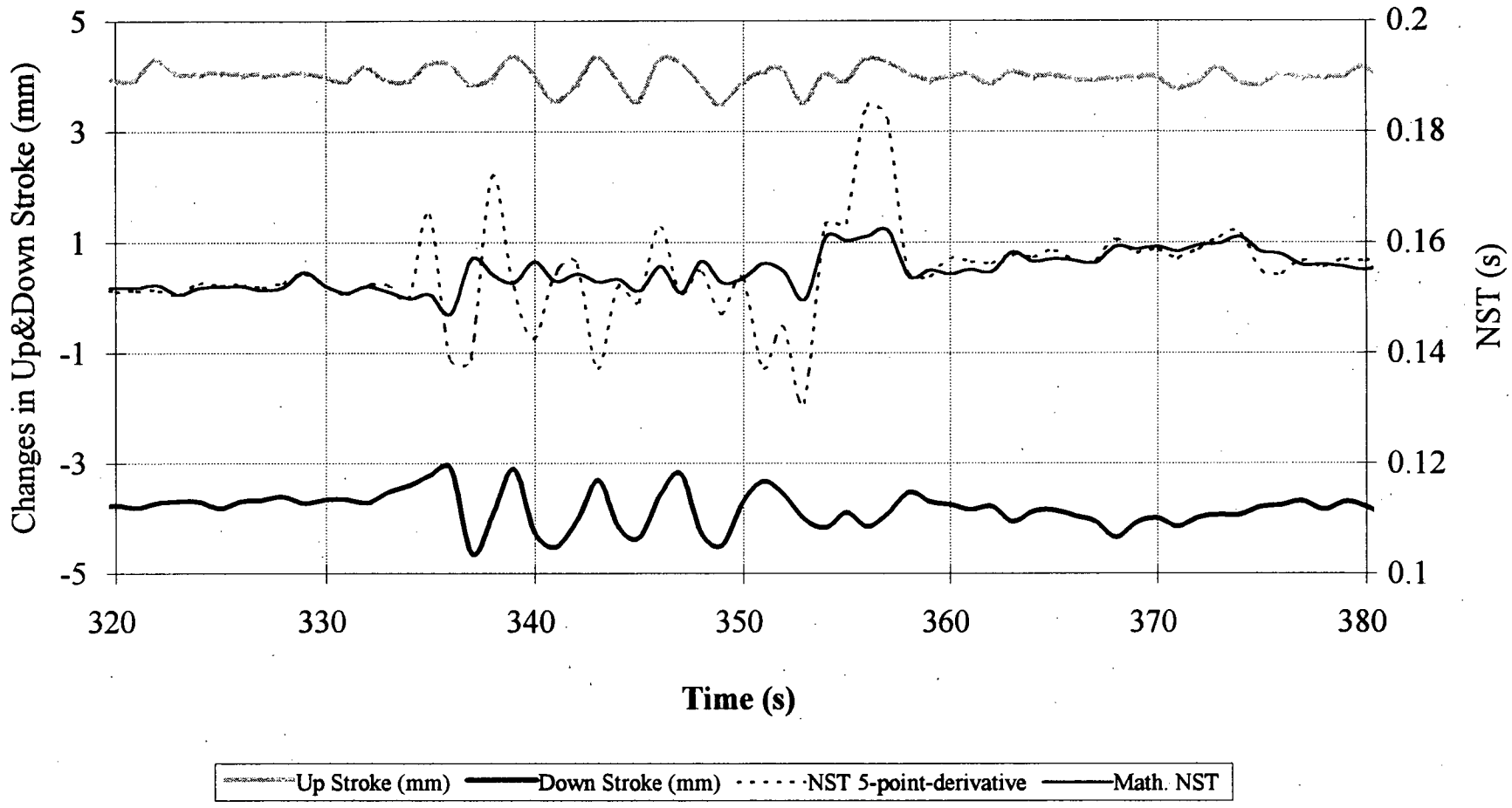

Figure 30: Influence of mould displacement on negative strip time during displacement instability, Company A, Heat #E33767.

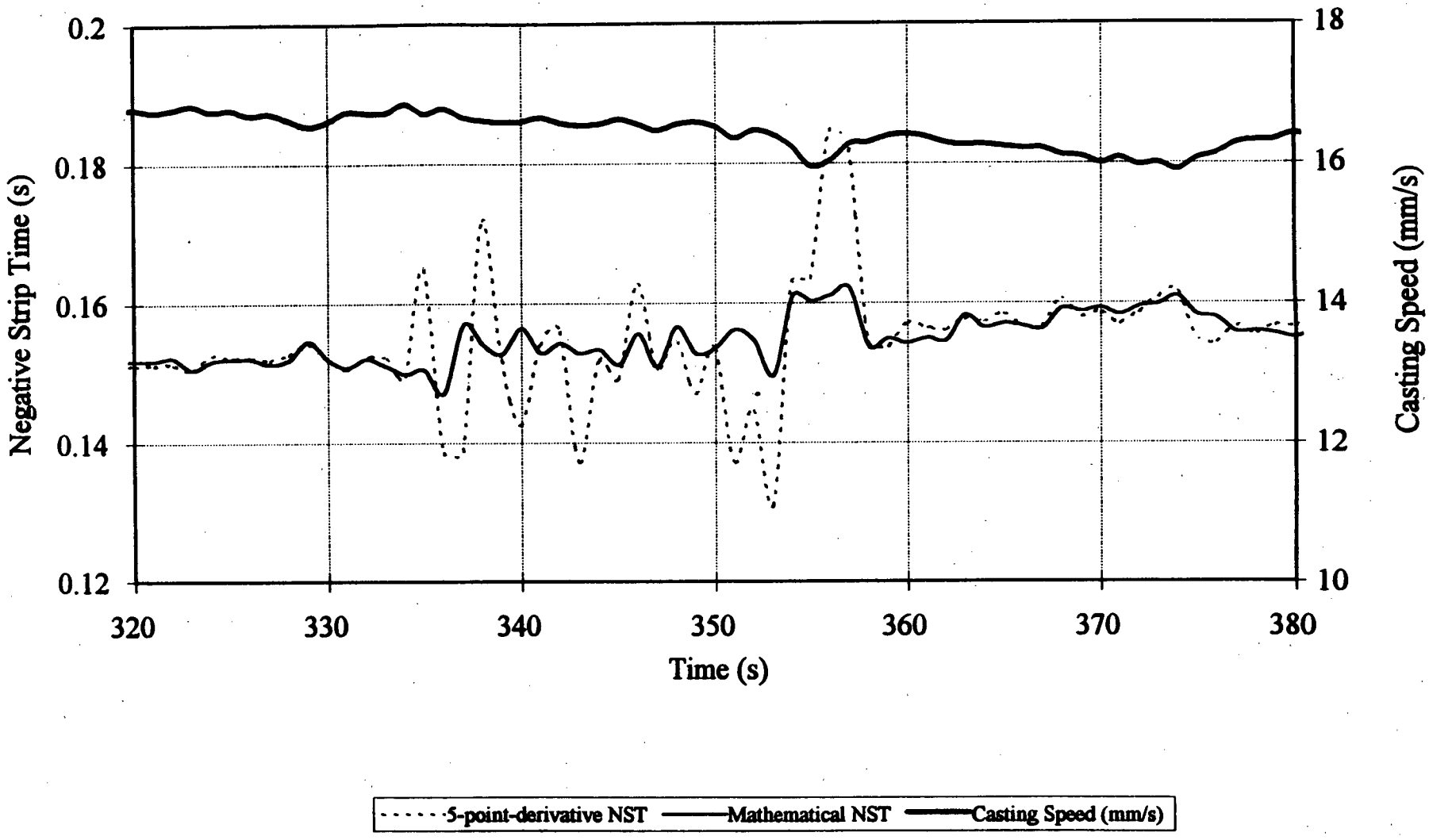

Figure 31: Influence of casting speed on negative strip time during displacement instability, Company A, Heat #E33767.

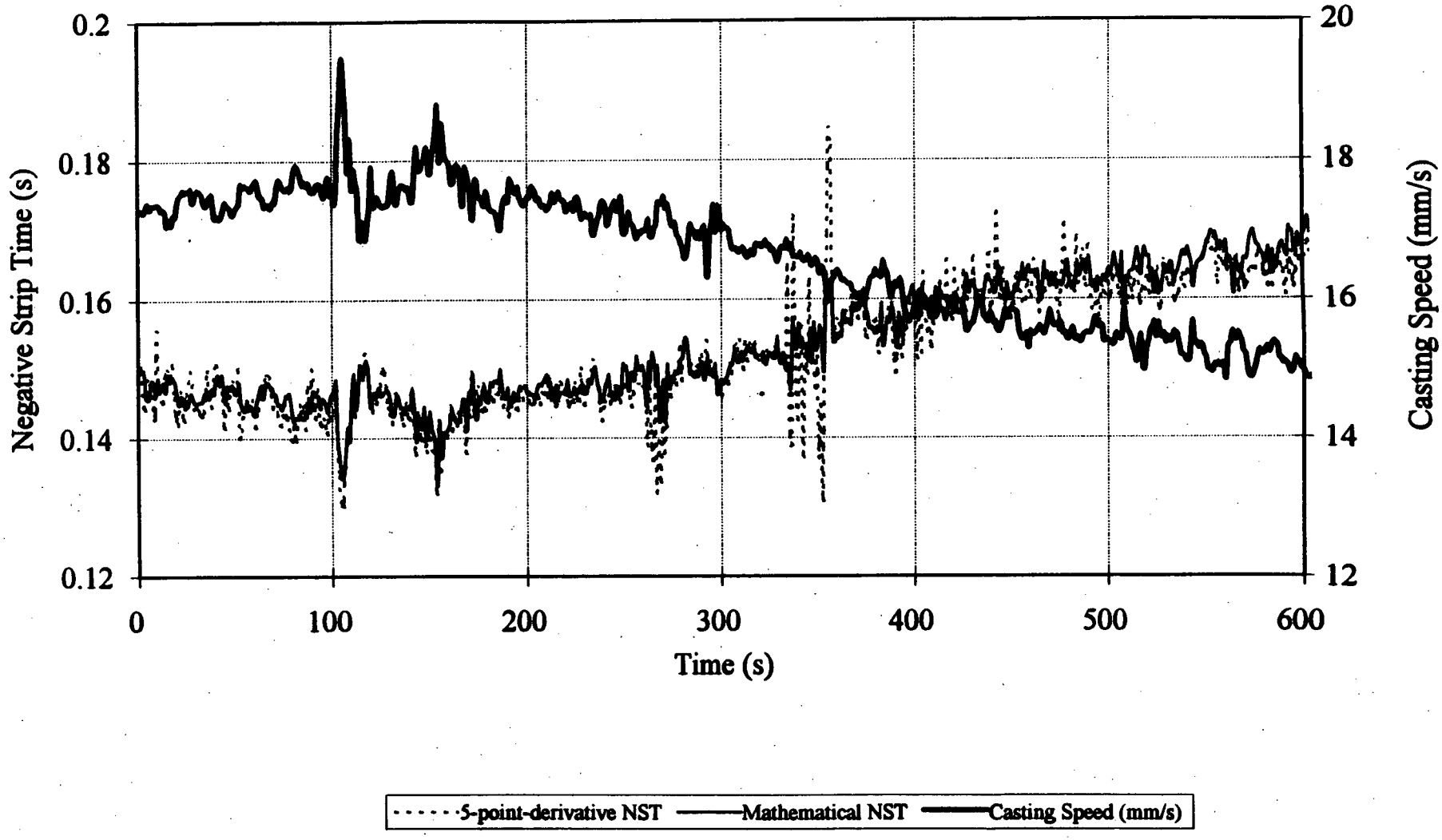

Figure 32: Influence of casting speed on negative strip time for 600 seconds of casting, Company A, Heat #E33767.

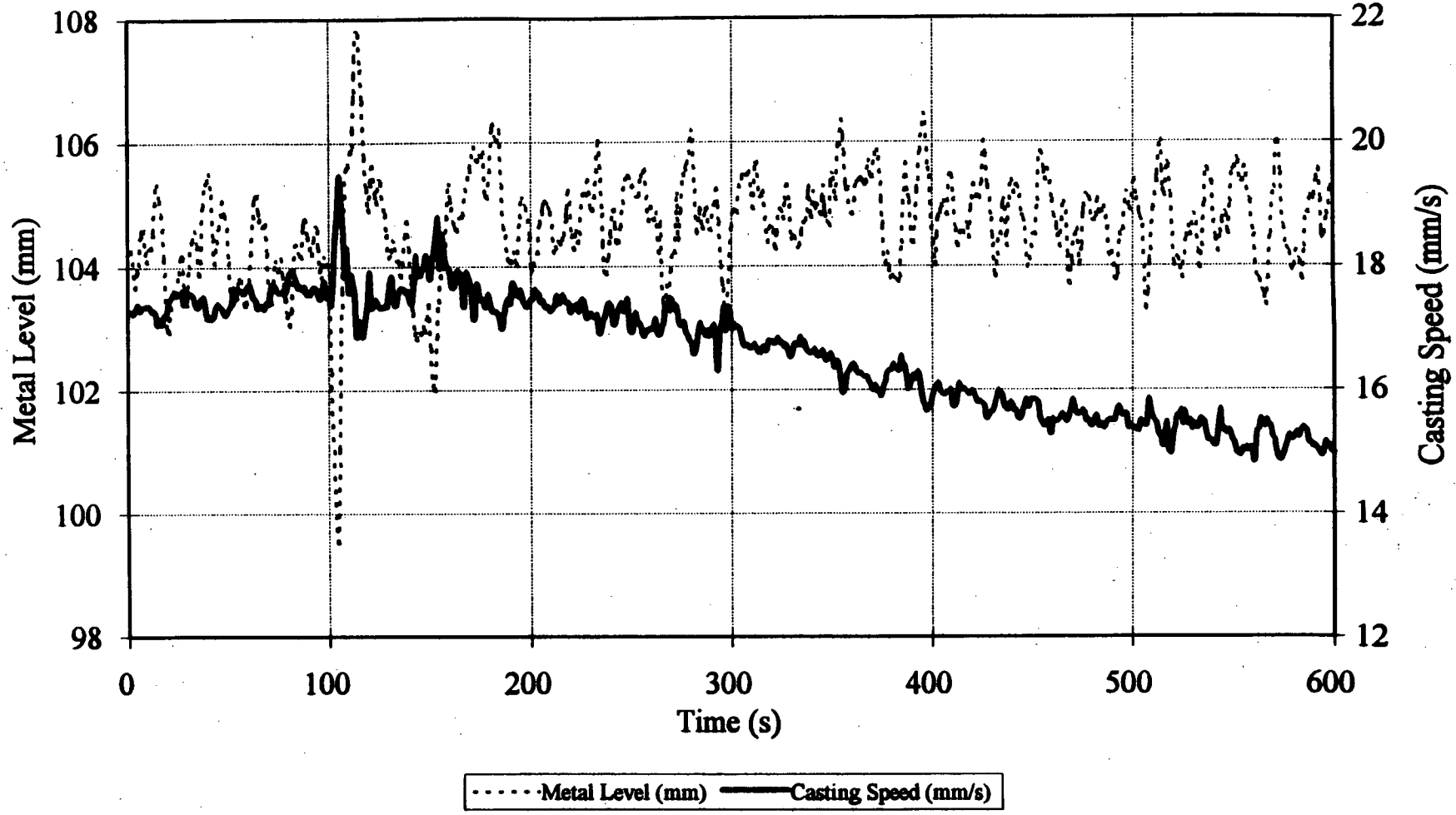

Figure 33: Graph comparing casting speed and metal level signal for 600 seconds of casting; Company A, Heat #E33767.

signal. Since the mould oscillates in the range of 2 - 4 Hz, depending on operating conditions, a sampling frequency of 9 Hz should be sufficient to reconstruct the mould displacement signal. If some other signal at a higher frequency is superimposed on the mould displacement, then a higher sampling frequency is necessary to pick up such oscillations and avoid aliasing [49]. Two analog signals with different frequencies may have the same values at all sampling instants. Hence, the higher frequency signal will be retrieved as a signal with lower frequency, if the sampling rate is not adequate. This processing error is known as aliasing.

During the second plant trial, negative strip time was calculated from 200 Hz sampled LVDT and casting speed signals. Although, the 5-point-moving-average together with 3-point and 5-point-derivative were applied, the negative strip time value jumped between 0 and 0.2 s, every other acquisition cycle. When the sampling rate was reduced to 60 Hz, the calculated t_N became reasonable and did not change in such an erratic fashion. Using the higher acquisition frequency caused very irregular velocity profiles, as derived from the first derivative from the LVDT signal, with numerous zero-crossings. It is certainly difficult to distinguished the right time interval for comparing casting speed and mould velocity, as discussed in section 4.5.2.1.

However, if a sampling frequency below 60 Hz is used this does not provide enough information for the t_N calculation based on the mathematical equation. The oscillation stroke calculated as the difference between LVDT maximum and minimum position, from 30 Hz sampled signal varied appreciably, although the actual analog signal did not change. If the mould oscillates at 3 Hz and the LVDT signal is sampled at 30 Hz,

one full oscillation cycle is depicted by only 10 points. This is insufficient for accurate calculation of oscillation stroke, frequency and hence t_N.

5.4 Results from shape recognition functions

The first attempt to extract depression drops and spans from thermocouple time responses was made by applying the valley filtering function, described in section 4.5.2.2. The early stages of the "Smart" mould project has focused on on-line detection of surface defects such as bleeds, laps and transverse depressions. The idea was to recognize up to 5 temperature drops over every 30 second sampling period, and to trace their positions in time as they moved down the mould. The meaning of some patterns in thermocouple data was unclear. For example, the same drop can be considered as two separate depressions, or as one that superimposes two coupled depressions. This complicates the mathematical methods used to describe the thermocouple responses. An ANN was considered as a tool for feature extraction that can produce correct output for similar and noisy data. However, this would involve the tedious work of teaching an ANN numerous input patterns, that might be considered as depressions. As our manual approach to this pattern-recognition problem was filled with considerable heuristics, an ANN approach was rejected at this time.

To be successful, the outputs from the associated thermocouples must be checked in order to reach a conclusion about the probable defects. The ES must be applied to analyze the output from several shape recognition modules.

The **valley** function was tested on the thermocouple data, recorded at previous plant trials, at company C, for heats with real surface defects. The output from the function was satisfactory; all major depressions were detected and processing time was acceptable. Depending on the ratio between data acquisition time and sampling frequency (i.e. number of points that have to be processed), as well as available computer resources, the processing time varied from 0.001-0.03 of the acquisition time. The limiting factor in the **valley** function is the size of the window that slides over the data table. It was left up to the user to define this parameter, based on the size of depressions being experienced. If the window size is too big the function will add two depressions together and generate incorrect output. If the size is too small, the function can not recognize the entire drop. A tuning rule within the ES examines if the window size is inadequate, based on the magnitude of the reported spans. If the spans are consistently close to the window size, the ES will increase the window, and the next processing cycle will be accomplished with a new window parameter.

Figures 34 and 35 present the thermocouple data sampled at 1 Hz over 120 minutes. Table 6 shows the function output for the data illustrated in Figures 34 and 35. The first, second and third columns give the positions of the left maximum, minimum and right maximum value, respectively, while the fourth and fifth present the span size of a detected drop in seconds and the temperature drop in °C. It is apparent that each "big" drop is detected, but small ones are disregarded. However, for the last twenty minutes of data in Figure 35, the **valley** function generated incorrect output: obviously, there is no

straight drop of 9.7 °C and span of 12 seconds at this time. The function considered two joined drops as one.

Table 6: Recognized drops by the valley function

The First Acquisition Cycle

left (s)	min_position (s)	right (s)	Span (s)	Temp_drop (°C)
0	0	7	7	29.12
51	56	63	12	24.18
63	68	72	9	12.82
72	79	85	13	13.74
99	103	108	9	9.71

The Second Acquisition Cycle

8	12	19	11	18.86
19	25	29	10	10.07
29	36	42	13	17.03
79	83	89	10	29.49
102	109	114	12	9.71

The extreme shape recognition function which was developed after the second plant trial clarified that two joined depressions can cause two distinguishable defects on a billet. The research team examined a number of billets for which detailed temperature trends were recorded. Billet surface defects at midface (i.e. at the location position of the thermocouple) were measured according to their distribution, position, and surface depth. These measurements served as input to a 2-dimensional mould heat-transfer model. The depth and distribution of billet defects were translated into upsets in mould heat flux, assuming that the deepest depression represented about a 85% upset in heat extraction. Output from the model was obtained for several thermocouples located around and down the mould. Model output was plotted concurrently with real sensor data obtained during casting, as depicted in Figure 36.

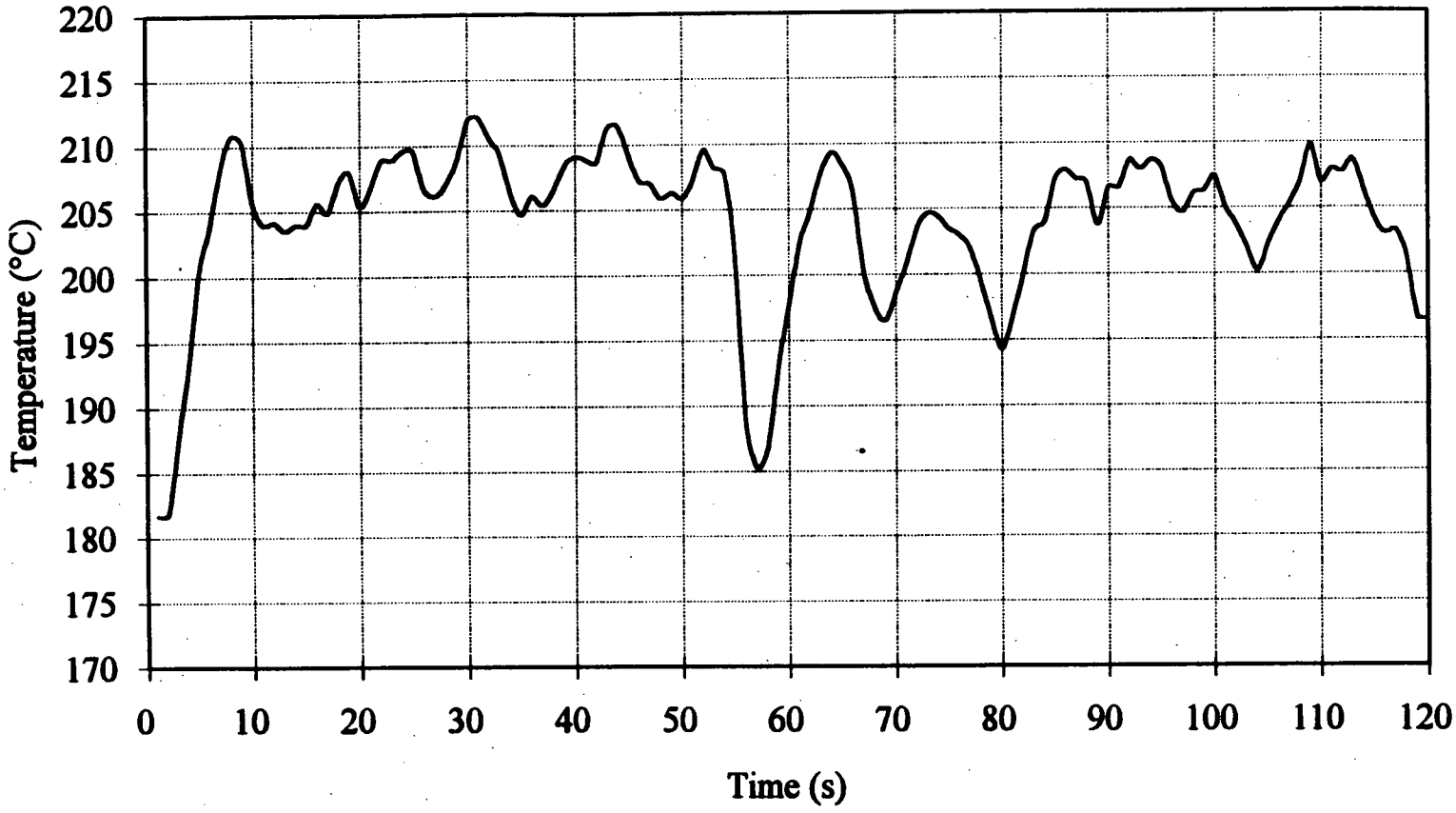

Figure 34: THC 191 mm from the top of the mould; the first acquisition cycle; Heat #D6131, Company C

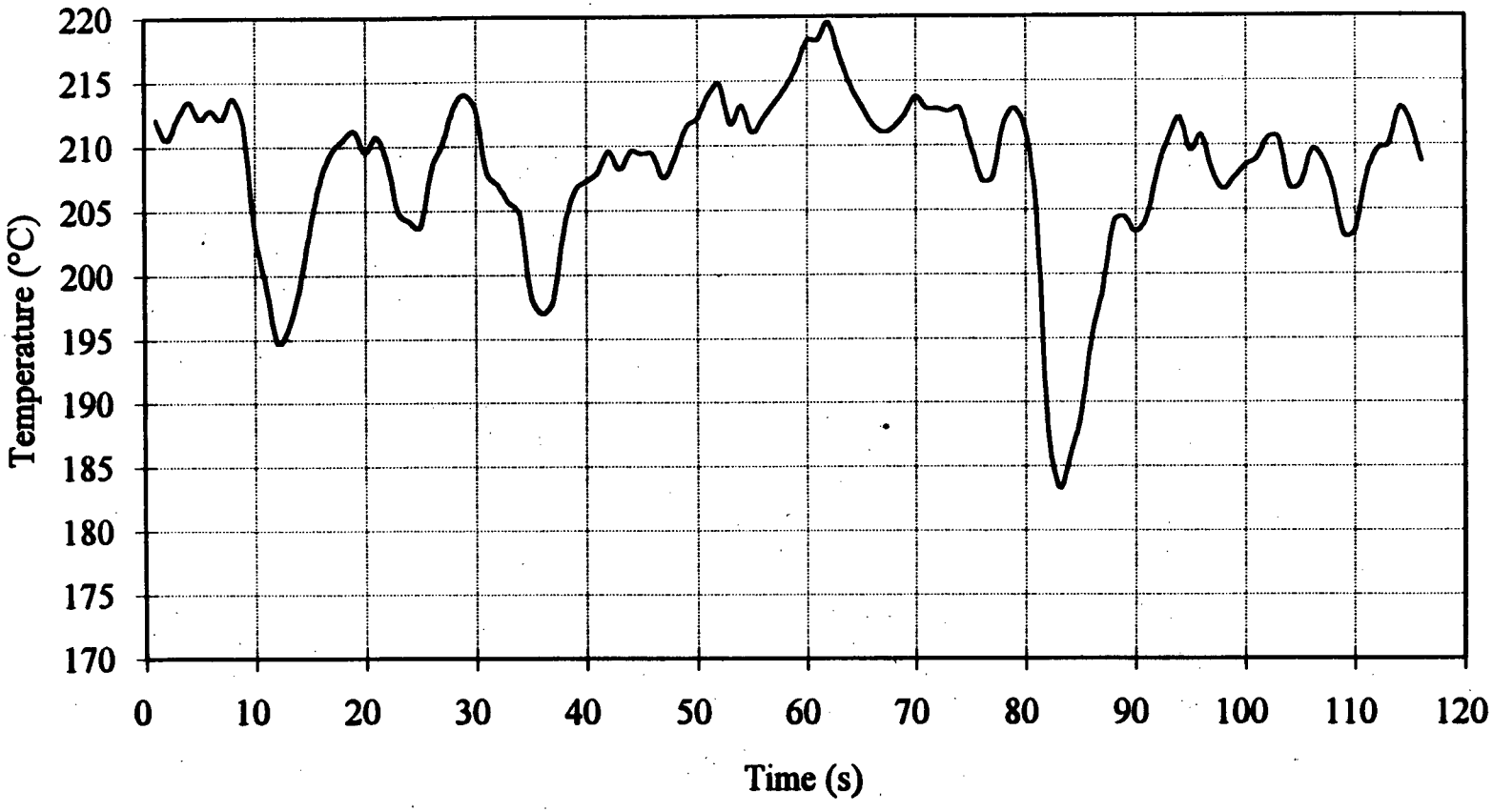

Figure 35: THC 191 mm from the top of the mould; the second acquisition cycle; Heat #D6131, Company C.

Figure 36 shows a clear correlation between these trends - one for the model temperature profile based on depression measurements and the second for the direct measurements taken from the thermocouple. The model output has the same general curve shape as the real thermocouple-time response. The model assumes a constant casting speed, although speed varies during casting. The analysis establishes a clear correlation with existing defects on the billets, sensor data, and the mould heat-flux model.

Figure 37 presents the thermocouple output from the mould model for the entire billet, with 45 measured depressions. The curve shape is obviously very smooth. The model output was run through the CI **extreme** function and the result is presented in Table 7. The **extreme** function recognized every single valley in the thermocouple trend, with extremely precise span size and temperature drop. The span size and temperature drop calculated for each detected depression are illustrated in Figures 38 and 39.

The corresponding thermocouple data were manually examined by the research team, and drop locations, size and spans, were recorded in a spreadsheet. Then, sensor data were run through the CI **valley** and **extreme** functions: the window parameter for the **valley** function was tested at 10 and 15 seconds. Table 8 provides the comparison between these three cases. The **extreme** function detected all 45 depressions with correct span sizes and drops, while the **valley** technique recognized only 24 and 20 depressions, for 10 and 15 second window sizes, respectively. It is evident that the biggest depressions are detected, but many small ones are disregarded.

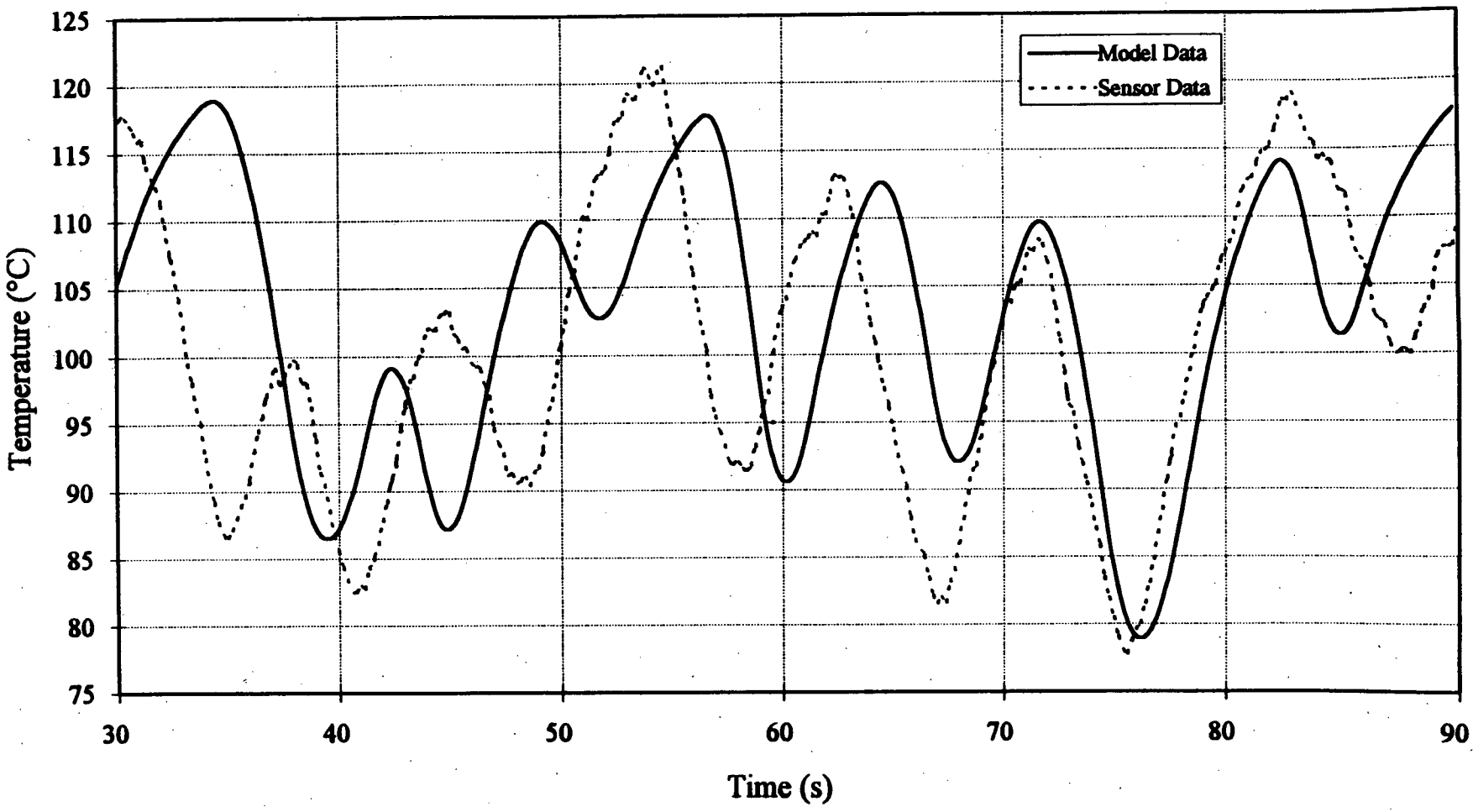

Figure 36: Sensor and model data for the THC E22, 535 mm from the top of the mould, Heat #333, Company D.

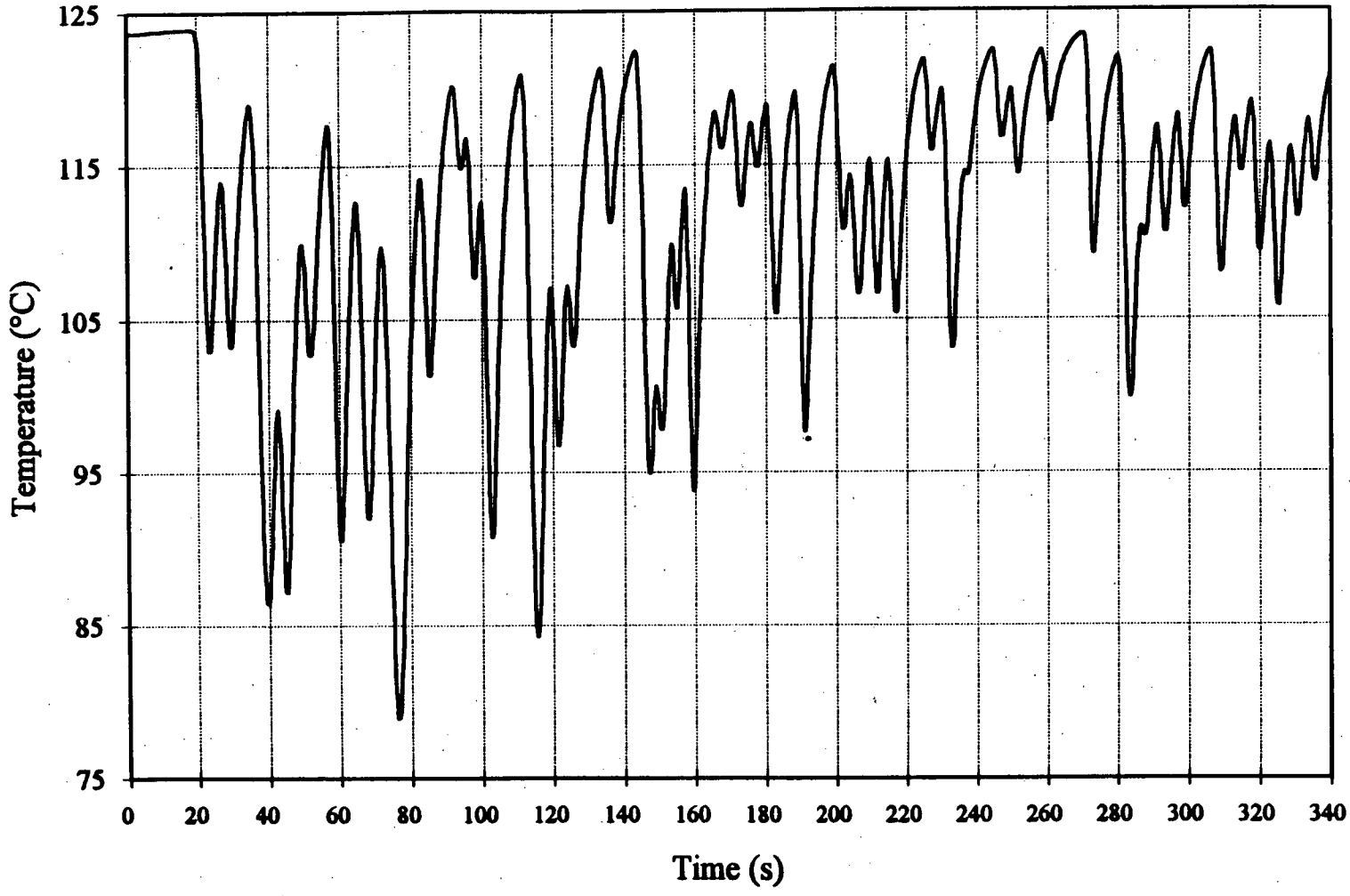

Figure 37: Model Data for the THC E22, 535 mm from the top of the mould, Heat #333, Company D.

Table 7: Output from the extreme function from thermocouple data obtained from the mould model

Extreme								
Depression	left (s)	L_max(°C)	min_pos (s)	Min_Temp(°C)	right (s)	R_max(°C)	Span(s)	Drop(°C)
1	18.00	123.90	23.60	103.01	26.80	113.98	8.80	20.89
2	26.80	113.98	29.60	103.33	34.80	118.96	8.00	15.63
3	34.80	118.96	39.80	86.49	42.80	99.12	8.00	32.47
4	42.80	99.12	45.20	87.19	49.60	109.91	6.80	22.72
5	49.60	109.91	52.00	102.77	57.00	117.65	7.40	14.88
6	57.00	117.65	60.60	90.58	64.80	112.65	7.80	27.07
7	64.80	112.65	68.40	92.05	72.00	109.72	7.20	20.60
8	72.00	109.72	76.60	78.96	83.00	114.12	11.00	35.16
9	83.00	114.12	85.40	101.41	92.00	120.11	9.00	18.70
10	92.00	120.11	94.40	114.83	95.80	116.74	3.80	5.28
11	95.80	116.74	98.20	107.77	100.00	112.57	4.20	8.97
12	100.00	112.57	103.00	90.81	111.20	120.85	11.20	30.04
13	111.20	120.85	115.80	84.32	119.60	107.02	8.40	36.53
14	119.60	107.02	121.80	96.80	124.40	107.15	4.80	10.35
15	124.40	107.15	126.00	103.27	133.60	121.29	9.20	18.02
16	133.60	121.29	136.20	111.30	143.40	122.39	9.80	11.09
17	143.40	122.39	147.40	94.96	149.20	100.61	5.80	27.43
18	149.20	100.61	150.80	97.82	153.80	109.84	4.60	12.02
19	153.80	109.84	155.00	105.70	157.40	113.42	3.60	7.72
20	157.40	113.42	159.80	93.79	166.00	118.42	8.60	24.63
21	166.00	118.42	167.80	116.09	170.80	119.73	4.80	3.64
22	170.80	119.73	173.20	112.33	176.00	117.70	5.20	7.40
23	176.00	117.70	177.80	114.89	180.40	118.88	4.40	3.99
24	180.40	118.88	183.20	105.35	188.60	119.71	8.20	14.36
25	188.60	119.71	191.40	97.67	199.40	121.36	10.80	23.69
26	199.40	121.36	202.40	110.81	204.20	114.31	4.80	10.55
27	204.20	114.31	206.60	106.64	209.80	115.35	5.60	8.71
28	209.80	115.35	212.00	106.68	215.00	115.30	5.20	8.67
29	215.00	115.30	217.20	105.41	225.00	121.83	10.00	16.42
30	225.00	121.83	227.40	115.90	230.20	119.87	5.20	5.93
31	230.20	119.87	233.20	103.12	236.80	114.53	6.60	16.75
32	236.80	114.53	237.40	114.31	244.60	122.41	7.80	8.10
33	244.60	122.41	247.20	116.78	249.80	119.87	5.20	5.63
34	249.80	119.87	252.00	114.43	258.60	122.37	8.80	7.94
35	258.60	122.37	261.00	117.77	270.20	123.44	11.60	5.67
36	270.20	123.44	273.20	109.22	280.20	121.95	10.00	14.22
37	280.20	121.95	283.80	99.90	286.80	110.93	6.60	22.05
38	286.80	110.93	287.80	110.35	291.00	117.46	4.20	7.11
39	291.00	117.46	293.40	110.64	297.00	118.18	6.00	7.54
40	297.00	118.18	299.00	112.20	306.20	122.36	9.20	10.16
41	306.20	122.36	309.40	108.00	313.20	117.95	7.00	14.36
42	313.20	117.95	315.00	114.52	317.80	119.04	4.60	4.52
43	317.80	119.04	320.20	109.38	323.00	116.27	5.20	9.66
44	323.00	116.27	325.40	105.82	329.00	116.04	6.00	10.45
45	329.00	116.04	331.00	111.58	334.00	117.89	5.00	6.31

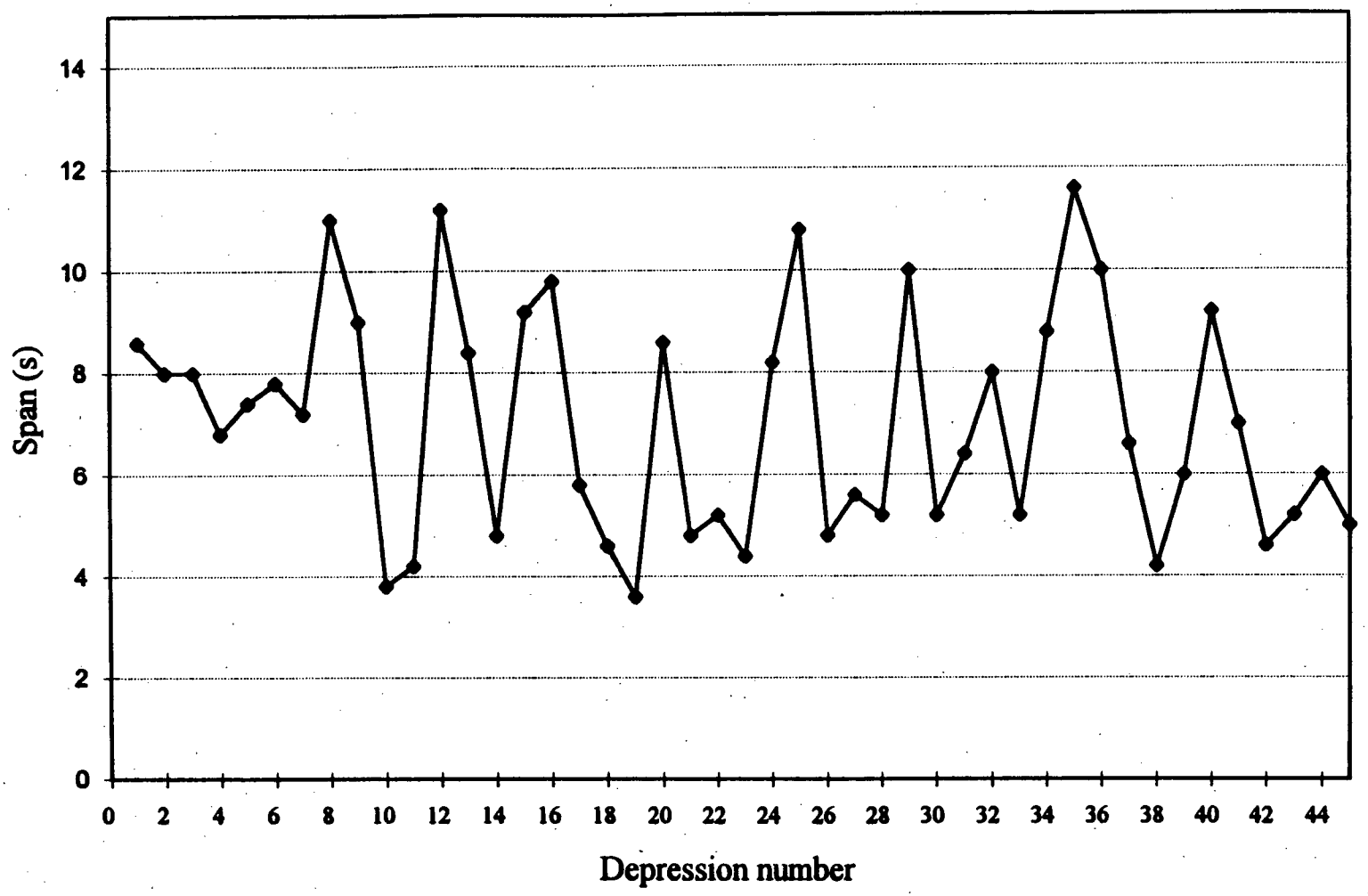

Figure 38: 45 detected depressions and their spans, Heat #333, Company D.

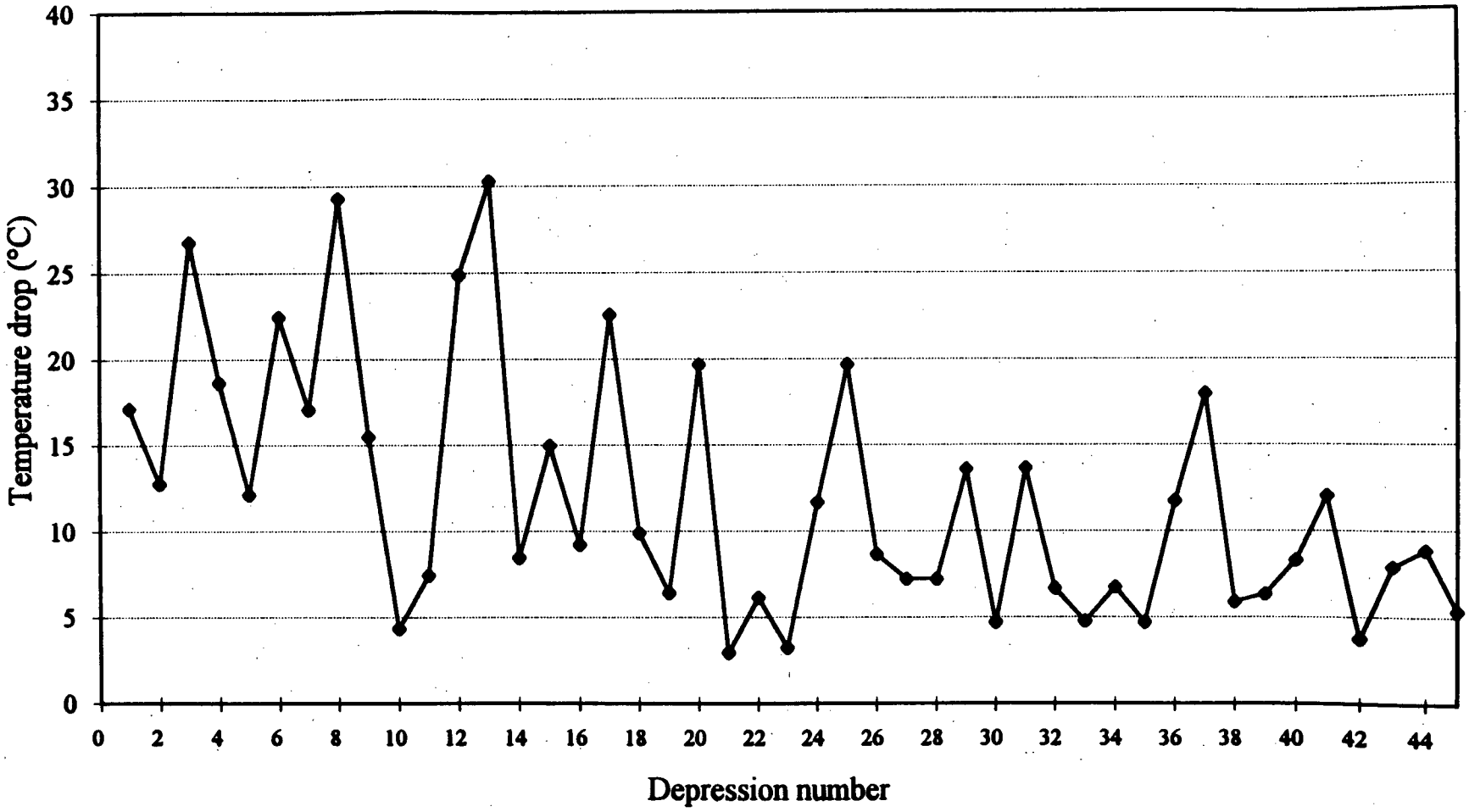

Figure 39: 45 detected depressions and the corresponding temperature drops, Heat #333, Company D.

Table 8: Output from the **extreme, 10s-window, and 15s-window valley function** over thermocouple data obtained from Heat #333, Company D.

Depression	Extreme		Win_10s		Win_15s	
number	Span(s)	Drop(°C)	Span(s)	Drop(°C)	Span(s)	Drop(°C)
1	8.80	20.89	8.2	20.63	8.20	20.63
2	8.00	15.63	-	-	-	-
3	8.00	32.47	8	32.06	12.60	32.06
4	6.80	22.72	6.8	22.17	-	-
5	7.40	14.88	-	-	-	-
6	7.80	27.07	7.8	26.45	7.80	26.45
7	7.20	20.60	-	-	-	-
8	11.00	35.16	9.6	31.28	10.80	34.61
9	9.00	18.70	7.8	16.77	9.20	18.27
10	3.80	5.28	-	-	-	-
11	4.20	8.97	-	-	-	-
12	11.20	30.04	8	24.69	14.80	29.15
13	8.40	36.53	8.4	36.07	8.40	36.07
14	4.80	10.35	4.8	9.74	-	-
15	9.20	18.02	-	-	-	-
16	9.80	11.09	8	9.64	-	-
17	5.80	27.43	9	27.01	10.40	27.01
18	4.60	12.02	-	-	-	-
19	3.60	7.72	-	-	-	-
20	8.60	24.63	7.4	22.36	8.60	23.87
21	4.80	3.64	-	-	-	-
22	5.20	7.40	5.4	7.07	-	-
23	4.40	3.99	-	-	-	-
24	8.20	14.36	7.8	13.79	8.00	13.89
25	10.80	23.69	8	21.37	10.60	22.99
26	4.80	10.55	-	-	-	-
27	5.60	8.71	5.6	8.22	10.40	14.39
28	5.20	8.67	-	-	-	-
29	10.00	16.42	7.4	13.72	15.00	16.00
30	5.20	5.93	-	-	-	-
31	6.60	16.75	8	16.24	15.20	17.70
32	7.80	8.10	-	-	-	-
33	5.20	5.63	-	-	-	-
34	8.80	7.94	7.2	7.05	14.00	7.72
35	11.60	5.67	-	-	-	-
36	10.00	14.22	8.2	13.79	10.00	13.79
37	6.60	22.05	8.8	21.64	11.00	21.64
38	4.20	7.11	-	-	-	-
39	6.00	7.54	5.8	7.15	5.80	7.15
40	9.20	10.16	-	-	-	-
41	7.00	14.36	7.2	14.03	11.00	14.03
42	4.60	4.52	-	-	-	-
43	5.20	9.66	-	-	-	-
44	6.00	10.45	6	9.61	15.20	12.72
45	5.00	6.31	-	-	-	-

5.5 The results from the Expert System

Along with the SCADA System and data acquisition software, the output from the pseudo real-time Expert System was also tested. The ES was not the focus of the first plant trial; hence only a preliminary attempt was made to establish a rough correlation between the ES prediction and real defects. Unfortunately, production constraints (billets produced were almost immediately shifted into the rolling phase of the process) did not allow precise inspection. However, the rough check-over justified the ES prediction: occasional bleeds and laps were usually observed on the billets.

The objective of the second plant trial was to test the correctness of the ES. Several oil cast heats were monitored, and the output from the ES was recorded in a spreadsheet. Monitored thermocouples were located on the east face at 335, 435, 535, and 630 mm from the top of the mould, respectively. The corresponding billets were inspected, and the position, span and depth of the surface defects were measured.

The output from the ES contained time of occurrence, span size and temperature drop for each depression. A high degree of mapping between predicted and actual defects was obtained. Correlation with surface measurements was virtually perfect. Table 9 presents 45 detected depressions at thermocouples located on the east side, 335, 435, 535, and 630 mm from the top of the mould and the corresponding ES predictions. Only three measured depressions on the billet were predicted with a degree of belief (DoB) less than 50 %. The lowest DoB is 29 % for depression number 21. For this particular case, the actual depth of the depression on the billet is 0.024 inch (0.6096 mm). This is a rather small depression, and is not considered a "severe" billet defect. Future work will focus on

Table 9: Output from the extreme function and associated Expert System Degree of Belief

Drop	THC 335mm			THC 435mm			THC 535mm			THC 630mm			The ES
	pos(s)	Span(s)	Drop(°C)	pos(s)	Span (s)	Drop(°C)	pos(s)	Span (s)	Drop(°C)	pos(s)	Span (s)	Drop(°C)	DoB
1	13.60	8.60	28.08	18.60	8.20	25.17	23.60	8.80	20.89	28.60	8.80	16.44	100
2	19.60	7.80	21.27	24.60	8.00	18.88	29.60	8.00	15.63	34.60	8.00	12.22	99
3	30.00	8.00	43.23	35.00	8.00	38.81	39.80	8.00	32.47	44.80	8.00	25.39	100
4	35.40	6.80	30.41	40.40	6.80	27.24	45.20	6.80	22.72	50.20	6.80	17.83	100
5	42.20	7.40	20.09	47.20	7.40	17.86	52.00	7.40	14.88	57.00	7.40	11.88	98
6	50.60	8.00	36.52	55.60	8.00	32.61	60.60	7.80	27.07	65.60	7.80	21.32	100
7	58.40	7.20	27.61	63.40	7.20	24.53	68.40	7.20	20.60	73.40	7.20	16.23	100
8	66.60	10.80	47.11	71.60	10.80	41.96	76.60	11.00	35.16	81.60	10.80	27.66	100
9	75.60	9.20	25.21	80.60	9.20	22.52	85.40	9.00	18.70	90.40	9.20	14.85	100
10	84.60	3.80	7.11	89.40	3.80	6.31	94.40	3.80	5.28	99.40	3.80	4.11	55
11	88.40	4.20	12.18	93.20	4.00	10.81	98.20	4.20	8.97	103.20	4.20	7.11	90
12	93.20	11.20	40.59	98.00	11.40	36.10	103.00	11.20	30.04	108.00	11.20	23.47	97
13	105.80	8.20	48.80	110.80	8.20	43.66	115.80	8.40	36.53	120.60	8.40	28.57	100
14	111.80	4.80	13.93	116.80	4.80	12.31	121.80	4.80	10.35	126.80	4.60	8.09	95
15	116.00	9.20	24.05	121.00	9.20	21.46	126.00	9.20	18.02	131.00	9.40	14.31	98
16	126.40	10.00	14.94	131.40	9.80	13.30	136.20	9.80	11.09	141.20	9.80	8.62	93
17	137.40	5.80	36.72	142.40	5.80	32.72	147.40	5.80	27.43	152.40	5.80	21.59	100
18	140.80	4.40	16.37	145.80	4.60	14.38	150.80	4.60	12.02	155.60	4.40	9.37	96
19	145.20	3.80	10.62	150.20	3.80	9.27	155.00	3.60	7.72	160.00	3.80	6.10	83
20	149.80	8.40	32.94	154.80	8.40	29.30	159.80	8.60	24.63	164.80	8.60	19.40	94
21	158.00	4.80	4.92	163.00	4.80	4.40	167.80	4.80	3.64	172.80	4.60	2.91	29
22	163.40	5.40	10.10	168.40	5.20	8.83	173.20	5.20	7.40	178.20	5.40	5.78	78
23	168.00	4.40	5.41	172.80	4.40	4.80	177.80	4.40	3.99	182.80	4.40	3.13	37
24	173.20	8.00	19.51	178.20	8.20	17.17	183.20	8.20	14.36	188.00	8.00	11.25	97
25	181.40	10.80	31.91	186.40	10.80	28.37	191.40	10.80	23.69	196.20	11.00	18.60	100
26	192.40	4.80	14.15	197.40	4.80	12.62	202.40	4.80	10.55	207.40	4.80	8.28	92
27	196.80	5.60	11.77	201.80	5.60	10.32	206.60	5.60	8.71	211.60	5.60	6.79	88
28	202.00	5.20	11.68	207.00	5.20	10.35	212.00	5.20	8.67	217.00	5.00	6.77	88
29	207.20	10.20	22.10	212.20	10.00	19.68	217.20	10.00	16.42	222.20	10.20	12.94	99
30	217.40	5.00	7.90	222.40	5.20	7.03	227.40	5.20	5.93	232.40	5.20	4.62	57
31	223.20	6.60	22.51	228.20	6.60	20.03	233.20	6.60	16.75	238.20	6.40	13.14	99
32	227.60	7.80	10.97	232.40	7.80	9.83	237.40	7.80	8.10	242.40	8.00	6.41	81
33	237.40	5.40	7.55	242.40	5.20	6.74	247.20	5.20	5.63	252.20	5.20	4.38	57
34	242.00	8.60	10.61	247.00	8.80	9.46	252.00	8.80	7.94	256.80	8.60	6.17	80
35	251.20	11.60	7.61	256.00	11.60	6.71	261.00	11.60	5.67	266.00	11.60	4.44	60
36	263.40	10.00	19.16	268.20	10.00	16.89	273.20	10.00	14.22	278.20	10.00	11.17	97
37	273.80	6.60	29.65	278.80	6.60	26.41	283.80	6.60	22.05	288.60	6.80	17.34	100
38	277.80	4.40	9.59	282.80	4.40	8.48	287.80	4.20	7.11	292.60	4.20	5.58	77
39	283.60	5.80	10.18	288.40	5.80	9.03	293.40	6.00	7.54	298.40	5.80	5.98	79
40	289.00	9.20	13.68	294.00	9.20	12.09	299.00	9.20	10.16	303.80	9.40	7.99	91
41	299.60	7.20	19.34	304.40	7.20	17.13	309.40	7.00	14.36	314.40	7.00	11.29	97
42	305.20	4.40	6.13	310.00	4.40	5.36	315.00	4.60	4.52	320.00	4.60	3.52	48
43	310.40	5.20	12.96	315.40	5.20	11.46	320.20	5.20	9.66	325.20	5.20	7.59	91
44	315.60	6.00	14.06	320.60	6.00	12.41	325.40	6.00	10.45	330.40	5.80	8.20	93
45	321.00	5.00	8.49	326.00	5.00	7.53	331.00	5.00	6.31	335.80	5.20	4.97	68

translating the temperature drops and spans recorded in the mould into the real depths and spans on the billets. Regarding the actual span and depth of billet defects, a type of "depression severity index" has to be defined: each defect with an index value lower than the threshold will not be reported.

5.6 On-line billet quality prediction

The nature of the billet casting process has been guiding development of the CI module and the ES. After two plant trials and discussion with plant personnel, it appears to be more useful for the pseudo real-time ES to generate some type of report for each cast billet with an associated quality index that can be used to reject "bad" billets. Such a report would contain: 1. the number and kind of detected surface defects, such as depressions, bleeds, laps, off-corner cracks, craze cracks, and off-squereness, with the corresponding "severity" index; 2. metal level standard deviation; 3. negative strip time on-line calculation; 4. mould displacement variation, 5. changes in inlet and outlet water temperature. The overall quality index perhaps, could be obtained by applying an ANN: each type of defect found, associated with the highest detected severity index, can be input to an ANN structure. The connections weights could be strict, or could vary according to the operating conditions, steel chemistry, and production objectives. A Genetic Algorithms and/or Fuzzy Associative Memory module could be applied to define the optimum connections Weight matrix. The derived quality index would summarize the overall quality of a cast billet and provide on-line inspection: billets with a quality index below 50% would be diverted for inspection.

Chapter 6

Conclusions

The following conclusions arise from this work:

1. A new paradigm known as Computational Intelligence is evolving in which intelligent numerical manipulation will form the underpinning of successful AI applications. CI has been shown to be essential for moving AI into a real-time control environment. CI can be used to acquire data rapidly to allow control and/or monitoring of a continuous casting process.

2. Plant trials have proven that the selected real-time multitasking operating system, SCADA system development tool, the isolation transformers, and DAS-20 board are an adequate environment and hardware support for the "Smart" Mould application.

3. The data acquisition software provides necessary sampling frequencies, and correctly records sensor data.

4. Negative strip time obtained from the **5-point-derivative** method is proven to be more sensitive and accurate than t_N obtained from the mathematical equation.

5. The developed shape recognition functions extract the temperature drops in thermocouple data successfully. The use of the **extreme** function which uses first derivative information is more successful than the "window" technique at detecting overlapping and small depressions.

6. The pseudo real-time expert system is able to trace and predict surface defects successfully.

Chapter 7

Excellent

Recommendations for future work

The recommendations for future development of the "Smart" Mould can be divided into two parts: hardware components and further software development.

1. Hardware recommendations:

a. The complete version of the "Smart" Mould system should include 20 thermocouples (5 per each mould face), Linear Variable Displacement Transducer, Casting Speed and Metal Level signals; altogether 23 input channels. The current DAS-20 board supports just 16 analog inputs. Keithley MetraByte offers a 16-channel multiplexer to expand the number of input channels monitored by the DAS-20. The EXP-20 provides 16 inputs that can be multiplexed into a single DAS-20 input channel. Test work will be necessary to determine the influence of high numbers of input signals on system turn-around time.

b. Additional isolation amplifiers for all thermocouple signals have to be purchased and integrated into the SCADA system.

2. Software recommendations :

a. The use of an EXP-20 board will require changes in the data acquisition software. The **das_digital_out** DAS library function has to be employed to address different input channels on the EXP-20.

b. The Fast Fourier Transform can be applied to analyze the operating conditions of the mould oscillator. This function will be an integral part of the CI module. This facility can be used to establish the need for maintenance of the oscillation system.

c. A knowledge base must be developed to map detected temperature drops in the mould into actual defect depth (for bleeds, laps, and transverse depressions).

d. A knowledge base for billet off-squareness has to be created and incorporated into "Smart" Mould. Use of off-corner THC signals can be used to establish the degree of rhomboidity.

e. A knowledge base for off-corner cracks has to be incorporated into "Smart" Mould. This rule set would also rely on THC signals used for depressions detection, but processed in a different manner.

f. Some parts of the existing off-line expert system to diagnose quality problems in the continuous casting of steel billets (CRAC/X) [13] have to be translated into "Smart" Mould. The rules describing defects caused by steel chemistry can be easily transferred into the real-time ES.

g. Work has to be conducted to produce the report as the result of the "Smart" Mould analysis of the steel chemistry and casting conditions.

h. Work has to be conducted to establish and create a quality index for each billet produced from the "Smart" Mould.

References

[1] J.K. Brimacombe: "Empowerment with Knowledge-Toward The Intelligent Mould for The Continuous Casting of Steel Billets", Iron and Steelmaker, 1993, Vol. 20 (II), pp.35-47.

[2] J.K. Brimacombe, I.V. Samarasekera: "Future Trends in the Development of Continuous Casting Moulds", In *Mould Operation for Quality and Productivity*, ISS-AIME, Warrendale, PA., 1991, pp 153-160.

[3] F. Jovic: Process Control Systems: Principles of Design Operation and Interfacing, Chapman & Hall, London, UK, 1992.

[4] K.J. Åström and Björn Wittenmark: Computer-Controlled Systems, Theory and Design, Prentice Hall Information and System Sciences Series, 1990.

[5] R.V.Williams: Control and Analysis in Iron and Steelmaking, Butterworths Monographs in Materials, Boston, USA, pp.166-176.

[6] W.H. Emling and S. Dawson: "Mould Instrumentation for Breakout Detection and Control", Proc. Steelmaking Conf., ISS, Warrendale, PA., 1991, Vol. 74, pp. 197-217.

[7] F. Haers and S.G. Thornton: "Application of mould thermal monitoring on the two strand slab caster at Sidmar", Iron and Steelmaking 1994, Vol.21 No.5.

[8] K.D. Schnelle and R.S.H. Mah: "Product Quality Management Using a Real-time Expert System", ISIJ International, Vol. 34, 1994, No. 10, pp. 815-821.

[9] M.J. Hague: "Diagnostic aids for quality improvement and maintenance in continuous caster", Iron and Steel Engineer, May 1988, pp. 36-42.

[10] J.A. Meech, S. Kumar: A Hypermanual on Expert Systems v.2.0, electronic book, CANMET, 1993.

[11] M.M. Veiga: A Heuristic System for Environmental Risk Assessment of Mercury from Gold Mining Operations, Ph.D. Thesis, University of British Columbia, Dept.of Mining and Mineral Processing, 1994, pp. 64-67.

[12] D. Waterman: A Guide to Expert Systems, Addison-Wesley, Reading, MA. USA, 1986.

[13] S. Kumar: An Expert System to Diagnose Quality Problems in Continuous Casting of Steel Billets, M.A.Sc. Thesis, University of British Columbia, Dept. of Metals and Materials Eng., pp. 5-8.

[14] L.A. Zadeh: Fuzzy Logic, IEEE Computer Mag., Apr. 1988, pp. 83-93

[15] C. A. Harris:Fuzzy Logic: A Potential Control Technique for Mineral Processes, MSc. Thesis, Queen's University at Kingston, Department of Mining Engineering, 1986, pp. 25-27

[16] J.M. Sibigtroth, D. Mazuelos: Basic Training: Fuzzy Logic for 8-bit MCUs, Conference Proceedings, Fuzzy Logic, July 1993, San Francisco, CA., USA, Sesion #T11.

[17] J.A. Meech, L.A. Jordan: Development of a Self-Tuning Fuzzy Logic Controller, Minerals Engineering, Vol. 6, No. 2, printed in Great Britain, 1993, pp. 119-131.

[18] M.H. Smith, H. Takagi: Optimization of Fuzzy Systems by Switching Reasoning Methods Dynamically, presented at the International Conference on Fuzzy Systems, Seoul, Korea, June 993.

[19] P.K. Simpson: Foundations of Neural Networks, Artificial Neural Networks: Paradigms, Applications and Hardware Implementations, IEEE Press, Piscataway, NJ, 1992, pp. 3-20.

[20] G. Wells: An Introduction to Neural Networks, Application of Artificial Intelligence in Process Control, edited by L. Boullart, A. Krijgsman and R.A. Vingerhoeds, Pergamon Press, Oxford, UK, 1992, pp. 176-183.

[21] V. Rakocevic, J.A. Meech: Application of Artificial Neural Network to interpret Froth Images from a Copper Flotation Process, in press.

[22] AND Corporation: HNet Discovery, Version 1.4- User's Manual, 1994.

[23] J.R. Koza: Genetic Programming, A Bradford Book, The MIT Press, Cambridge, Massachusetts, USA, 1993.

[24] H.B. Verbruggen, A.J. Krijgsman, P.M. Bruijn: Towards Intelligent control: Integration of AI in Control, Application of Artificial Intelligence in Process Control, edited by L. Boullart, A. Krijgsman and R.A. Vingerhoeds, Pergamon Press, Oxford, UK, 1992, pp. 223-247.

[25] H.B. Verbruggen, K.J. Åström: Artificial Intelligence and Feedback Control, IFAC Workshop, Shenyang, People's Republic of China, September 1989, pp. 1-11.

[26] K.J. Åström: Autonomous Process Control, Proceedings of The Second IEEE Conference on Control Applications, Vancouver, British Columbia, Canada, September 1993, pp. 573-580.

[27] A.J. Krijgsman, R. Jager, H.B Verbruggen and P.M. Bruijn: DICE: A Framework for Intelligent Real-Time Control, IFAC Workshop, California, USA, September 1991, pp. 13-19.

[28] R. Jager: Direct Real-time Control using Knowledge-Based Techniques, Proceedings of the European Simulation Symposium, Ghent, Belgium, 1990.

[29] W.M. Lattimer and co-workers: An expert system for real-time control, IEEE Software, March 1986, pp. 16-24,.

[30] N.K. Wickramarachchi: Development of a Knowledge-Based Hiearchical Control Structure for Process Automation, Ph.D. Thesis, University of British Columbia, Dept. of Mechanical Eng., March 1995.

[31] M.P. Lukas, R.A. Oye, M.A. Keyes, and A. Kaya: Evolution of Expert Systems for Real-Time Process Management: A Case Study on Motor Control, IFAC Workshop, Shenyang, People Republic of China, September 1989, pp. 79-84.

[32] D.J. Musliner, J.A. Hendler, and A.K. Agrawala: The Challenges of Real-Time AI Computer, January 1995.

[33] J.C. Bezdek: "What is Computational Intelligence?", Computational Intelligence - Imitating Life, 1994 IEEE World Congress on Computational Intelligence (WCCI), pp. 1-12.

[34] C.S. Williams: Designing Digital Filters, Prentice-Halll, INC., Englewood Clliffs, New Jersey, 1986., pp. 257-320.

[35] G. Kaiser: A Friendly Guide to Wavelets, Birkhäuser Boston, Cambridge, MA., USA., 1995.

[36] Yves Meyer: Wavelets Algorithms&Applications, The Society for Industrial and Applied Mathematics, Philadelphia, USA, 1994.

[37] J.A. Meech: AI Applications in the Mining Industry into 21st Century, Proceedings of APCOM XXV, Brisbane, Australia, July 1995.

[38] S. Kumar, B.N. Walker, I.V. Samarasekera, J.K. Brimacombe: Chaos at the Meniscus - The Genesis of Defects in Continuously Cast Steel Billets, in press

[39] Brenda Flotation Supervisor, Brenda Mines, Kelowna, BC., Canada.

[40] Wabush Mines SAG Mill, Wabush Mines, SW Labrador, Newfoundland, Canada.

[41] St. Lawrence Cement Kiln, Joliette, Que., Canada.

[42] Mount Isa Mines Copper Concentrator, Mount Isa Mines, Australia.

[43] HVC Froth Recognition, Highland Valley Copper, Logan Lake, BC., Canada

[44] Keithley Data Acquisition: DAS-20 User's Guide, April 1993

[45] QNX Software Systems Ltd.: QNX Data Acquisition Toolkit - Programmer's Guide, 1993.

[46] W.H. Press, S.A. Teukolsky, W.T. Vetterling, B.P. Flannery: "Numerical Recipes in FORTRAN - The Art of Scientific Computing", Cambridge University Press, Cambridge, USA, 1992.

[47] Comdale Technologies Inc: ProcessVision Version 5.2, User's Manual and Reference Guide, 1993.

[48] R. Iserman: Digital Control Systems, Springer-Verlag Berlin Heidelberg, Germany, 1989.

[49] P.P. Vaidyanathan: Multirate Systems and Filter Banks, Prentice Hall, Englewood Cliffs, New Jersey, USA, 1993.

Sample Configuration file

```
0                                   //trigger channel
1                                   //trigger value (in Volts)
30                                  //scan interval (in seconds)
10                                  //delay_low (in milliseconds)
50                                  //sampling frequency (Hz)
8                                   //num_of_channels
20                                  //room temperature in degree C - for calibration

//Setup for channel 0
0                                   //range_low (in Volts) for channel 0
10                                  //range_high (in Volts) for channel 0
5                                   //channel type - 5 for new amplifier for T type THC
1                                   //active state of channel ( 0 - non-active )
2                                   //number of functions applied to this channel
//      Function name and passed key-word-triplets
minmax
2                                   //number of passed kwt
TC_above_ML.min.@f
TC_above_ML.max.@f
//      Function name and passed key-word-triplets
average
6                                   //number of passed kwt
TC_above_ML.av1.@f
TC_above_ML.av2.@f
TC_above_ML.av3.@f
TC_above_ML.av4.@f
TC_above_ML.av5.@f
TC_above_ML.av6.@f

//Setup for channel 1
0                                   //range_low  (in Volts) for channel 1
10                                  //range_high (in Volts) for channel 1
6                                   //channel type for LVDT signal
1                                   //active state of channel ( 0 - non-active )
2                                   //number of functions applied to this channel
//      Function name and passed key-word-triplets
minmax
2                                   //number of passed kwt
LVDT.min.@f
LVDT.max.@f
//      Function name and passed key-word-triplets
compare
4                                   //number of passed kwt
3                                   //set up frequency
10                                  //set up stroke
negative_strip_time.average.@f
negative_strip_time.actual.@f

//Setup for channel 2
0                                   //range_low  (in Volts) for channel 2
```

```
10                              //range_high (in Volts) for channel 2
7                               //channel type for casting speed
1                               //active state of channel ( 0 - non-active )
1                               //number of functions applied to this channel
//      Function name and passed key-word-triplets
average
1                               //number of passed kwt
casting.speed.@f

//Setup for channel 3
0                               //range_low  (in Volts) for channel 3
10                              //range_high (in Volts) for channel 3
0                               //channel type
0                               //active state of channel ( 0 - non-active )
0                               //number of functions applied to this channel

//Setup for channel 4
0                               //range_low  (in Volts) for channel 4
10                              //range_high (in Volts) for channel 4
5                               //channel type - 5 for new amplifier for T type THC
1                               //active state of channel ( 0 - non-active )
3                               //number of functions applied to this channel
//      Function name and passed key-word-triplets
extreme
1                       //number of files (only one allowed)
extr_ch4.dat
//      Function name and passed key-word-triplets
valley
23                      //number of function variables
5                       //Temperature drop to search for (degree C)
15                      //size of depression to search for (span in sec)
TC11.t_span.@f
TC11.t_drop.@f
TC11.t_base.@f
TC11.m_time.@f
TC12.t_span.@f
TC12.t_drop.@f
TC12.t_base.@f
TC12.m_time.@f
TC13.t_span.@f
TC13.t_drop.@f
TC13.t_base.@f
TC13.m_time.@f
TC14.t_span.@f
TC14.t_drop.@f
TC14.t_base.@f
TC14.m_time.@f
TC15.t_span.@f
TC15.t_drop.@f
TC15.t_base.@f
TC15.m_time.@f
number_of_valley.in_TC1.@i
//      Function name and passed key-word-triplets
storedata
```

```
1                                   //number of files (only one allowed)
channel4.dat

//Setup for channel 5
0                                   //range_low  (in Volts) for channel 5
10                                  //range_high (in Volts) for channel 5
5                                   //channel type - 5 for new amplifier for T type THC
1                                   //active state of channel ( 0 - non-active )
3                                   //number of functions applied to this channel
//      Function name and passed key-word-triplets
extreme
1                           //number of files (only one allowed)
extr_ch5.dat
//      Function name and passed key-word-triplets
valley
23                          //number of function variables
5                           // temperature drop to search for (degree C)
15                          //size of depression to search for (span in sec)
TC21.t_span.@f
TC21.t_drop.@f
TC21.t_base.@f
TC21.m_time.@f
TC22.t_span.@f
TC22.t_drop.@f
TC22.t_base.@f
TC22.m_time.@f
TC23.t_span.@f
TC23.t_drop.@f
TC23.t_base.@f
TC23.m_time.@f
TC24.t_span.@f
TC24.t_drop.@f
TC24.t_base.@f
TC24.m_time.@f
TC25.t_span.@f
TC25.t_drop.@f
TC25.t_base.@f
TC25.m_time.@f
number_of_valley.in_TC2.@i
//      Function name and passed key-word-triplets
storedata
1                                   //number of files (only one allowed)
channel5.dat

//Setup for channel 6
0                                   //range_low  (in Volts) for channel 6
10                                  //range_high (in Volts) for channel 6
5                                   //channel type - 5 for new amplifier for T type THC
1                                   //active state of channel ( 0 - non-active )
3                                   //number of functions applied to channel 6
//      Function name and passed key-word-triplets
extreme
1                           //number of files (only one allowed)
extr_ch6.dat
```

```
//      Function name and passed key-word-triplets
valley
23                      //number of function variables
5                       //temperature drop to search for (degree C)
15                      //size of depression to search for (span in sec)
TC31.t_span.@f
TC31.t_drop.@f
TC31.t_base.@f
TC31.m_time.@f
TC32.t_span.@f
TC32.t_drop.@f
TC32.t_base.@f
TC32.m_time.@f
TC33.t_span.@f
TC33.t_drop.@f
TC33.t_base.@f
TC33.m_time.@f
TC34.t_span.@f
TC34.t_drop.@f
TC34.t_base.@f
TC34.m_time.@f
TC35.t_span.@f
TC35.t_drop.@f
TC35.t_base.@f
TC35.m_time.@f
number_of_valley.in_TC3.@i
//      Function name and passed key-word-triplets
storedata
1                               //number of files (only one allowed)
channel6.prn

//Setup for channel 7
0                               //range_low (in Volts) for channel 7
10                              //range_high (in Volts) for channel 7
5                               //channel type - 5 for new amplifier for T type THC
1                               //active state of channel ( 0 - non-active )
3                               //number of functions applied to this channel
//      Function name and passed key-word-triplets
extreme
1                       //number of files (only one allowed)
extr_ch7.dat
//      Function name and passed key-word-triplets
valley
23                      //number of function variables
5                       //temperature drop to search for (degree C)
15                      //size of depression to search for (span in sec)
TC41.t_span.@f
TC41.t_drop.@f
TC41.t_base.@f
TC41.m_time.@f
TC42.t_span.@f
TC42.t_drop.@f
TC42.t_base.@f
TC42.m_time.@f
```

```
TC43.t_span.@f
TC43.t_drop.@f
TC43.t_base.@f
TC43.m_time.@f
TC44.t_span.@f
TC44.t_drop.@f
TC44.t_base.@f
TC44.m_time.@f
TC45.t_span.@f
TC45.t_drop.@f
TC45.t_base.@f
TC45.m_time.@f
number_of_valley.in_TC4.@i
//          Function name and passed key-word-triplets
storedata
1                                   //number of files (only one allowed)
channel7.dat
```

UBC Data Acquisition Module

for

DAS-20 acquisition board

Introduction

The UBC Data Acquisition Module collects and processes sensor data from the DAS-20 board. It runs concurrently with other ProcessVision modules, and can be configured for collecting data from up to all sixteen single-ended analog inputs at frequency of several hundreds Hz (configurable).

Installing the ProcessVision - Data Acquisition Software

Before installing the ProcessVision - Data Acquisition software, make sure that your tick size under QNX 4 is set to 0.5 ms The **ticksize** utility queries or changes the rate at which timer interrupts (called ticks) are applied to the system. Setting the tick size will not affect the system date; ticks are used for software timers only. For more information refer to the QNX - Utilities Reference.

At the prompt type:

ticksize 0.5 <RETURN>

It is recommended that this is added to your **sysinit** file for setting the ticksize every time you boot QNX.

To install the Data Acquisition software under QNX 4, insert the ProcessVision - Data Acquisition driver distribution diskette in a floppy drive. Make sure the drive from which you are installing the software is "mounted". For example, to mount the floppy drive /dev/fd0 as /fd0 you would type the command:

mount /dev/fd0 /fd0 <RETURN>

Substitute as necessary for your drive number.

After the drive is mounted you can copy the driver to your system. First, create a new directory das as a subdirectory of /usr/lib/. At the prompt type:

cd /usr/lib <RETURN>

mkdir das <RETURN>

cd das <RETURN>

Copy from the ProcessVision - Data Acquisition distribution diskette files to your system by issuing the following commands:

cp -R /<mounted_drive_name>/das/ /<hard_drive_name>/usr/lib/das

at the QNX prompt then press the <RETURN> key.

Where:

<mounted_drive_name> is the name of your mounted floppy drive

<hard_drive_name> is the name of your mounted hard drive

Since the drive is usually "/fd0" or "/fd1" and the hard drive name is usually "/" then the issued command takes on one of these two forms:

cp -R /fd0/das/ /usr/lib/das

cp -R /fd1/das/ /usr/lib/das

Now, copy the driver files from the root on distribution diskette to your ProcessVision directory typing the following commands:

cp -v /<mounted_drive_name>/* /<hard_drive_name>/ProcessVision

at the QNX prompt then press the <RETURN> key.

Where, again:

<mounted_drive_name> is the name of your mounted floppy drive

<hard_drive_name> is the name of your mounted hard drive

The usual two forms are:

cp -v /fd0/* /ProcessVision

cp -v /fd1/* /ProcessVision

Installing the Data Acquisition Module

Before running **das_task**, you need to install the Keithley Metrabyte DAS-20 plug-in board in your host computer (see Keithley documentation).

UBC Data Acquisition Module Mode of Operation

The module you are shipped is called: **das_task**. This program is an executable program which is designed to run in the background in the QNX operating system. Although it can be run in the foreground, and you may wish to do so for debugging purpose, it is designed for long term use as a background process. This means that the driver should be started with the ampersand ('&') operator to specify background use.

Upon commencing execution, the **das_task** module will read a configuration file called **config.met**. This file is expected to contain information about the settings for analog input channels (input range, sensor type, activity, number and types of applied functions) and is stored under the **/usr/** directory. The configuration file also contains information about data acquisition frequency (how often data is collected expressed in Hz), scanning interval, trigger value, etc... If the configuration file is incorrect, an error message will be displayed and the **cast_task** will not work properly.

If the configuration file is read successfully, the **cast_task** module will start to collect and process data continuously.

The UBC Data Acquisition Module acquires sensor data from every active input channel over a desired time, applies specified functions to the stored data and updates the ProcessVision point database with the current filtered values of variables defined in the configuration file. It repeats this process continuously.

Several ProcessVision Data Acquisition Modules can be run simultaneously. Each module can collect data at a different frequency.

Data Acquisition Configuration File

The Data Acquisition configuration file contains information that is used by the module to perform data reading and filtering continuously. Trigger channel, trigger value, time interval over which filtering functions are applied, polling frequency for trigger event, data acquisition frequency and room temperature are specified in a common section of a configuration file. The following is the general format of a common section:

<setup_value> <tab> <tab> <//Comment>

An empty line separates the common section from channel 0 configuration. This part of configuration file begins with the comment:

// Setup for channel 0

The next lines define the minimum and maximum values for input range, expressed in volts, channel type (required for calibration), active state of a channel and a number of applied functions to the channel. The following is the general format:

<setup_value> <tab> <tab> <//Comment>

Where "setup_value" is numerical value (integer or float).

If the number of applied functions is greater than 0, the next line is:

// **Function name and passed key-word-triplets**

The following lines define the name of an applied function, the total number of passed variables (key-word-triplets) to the applied function, and a list of key-word-triplets. The following is the general format:

<function_name>

<number_of_passed_key_word_triplets> <tab> <tab> <//Comment>

<key_word_triplet>

...

<key_word_triplet>

If there is more than one applied function (but no more than five per channel - upper limit) the next lines will define the next function in much the same manner. The keyword-triplet can not be longer than 30 characters and there can not be more than 30 key-word-triplets per function.

An empty line separates the configuration section of each channel. An example configuration file is given at the end of this document.

Creating and Applying filter functions

Source code and executable files of seven data processing functions are shipped along with the Data Acquisition Module. The seven function are:

minmax - looks for minimum and maximum value over recorded data and passes two values to the ProcessVision point database as two key-word-triplets defined by a developer.

average - calculates average(s) over specified number of points in the recorded data table and feeds ProcessVision with this/these average(s) as key-word-triplet(s). Example: You record data for a specific channel over 30 seconds (sampling interval is defined as 30) at an acquisition frequency of 100 Hz (= 10 ms). You will end up with 3000 points (= 30 s *X* 100 Hz). If you apply function **average** and pass 10 key-word-triplets, you will get 10 key-word-triplets in the point data base and every one will represent average over 3 seconds (first key-word-triplet is average over first 3 seconds, etc ...)

storedata - stores collected data, expressed in volts, in a file. Filename is defined by a user.

compare - this is an example of how to combine data from two input channels. The example presented is designed for calculating negative strip time in continuous casting of steel billets. For this calculation we need inputs from an LVDT and Metal Level sensor.

valley - this function is an example of shape recognition and feature extraction. It looks for "valley" shape in row data. It can recognize up to 5 valleys in any data table and passes 20 key-word-triplets to ProcessVision. These triplets for every detected valley are: temperature drop, temperature before drop happened, time when drop happened expressed in absolute seconds since the beginning of the year, and valley span. User also defines the time domain over which depression is observed (in 30 seconds collected data you can look for valleys that have spans less than 5 seconds, 10 seconds or 20 seconds) and temperature drop that is recognized as significant (in some cases 3 degree C maybe significant drop, but in another it is just measurement noise).

extreme - this function calculates the first derivative over smoothed sensor data and searches for "maximum-minimum-maximu" patterns. It records span, drop, and position of each detected "valley" shape in sensor data.

calibration - this function converts input data expressed in volts to actual values. For every applied sensor, you should provide a mapping function and assign the channel type within the source code.

Based on these provided examples you should be able to write your own data processing function and incorporate it in the Data Acquisition Module. The main loop within the Data Acquisition Module (**das_task.c** source code) reads and records input data. After recording, it calls the data processing function **processing_of_data** (the same source code). Inside the body of this function are several **"if"** statements that compare names and call applied functions. Your new processing function has to be done in much the same way as minmax, average, ... It has to include the same header files and has to be compiled with the same options (make files are also provided).

Starting theUBC Data Acquisition Module

Assuming you are already inside QNX Windows and the necessary ProcessVision database modules are running, load up a new Shell. Type in:

cast_task & <RETURN>

The Data Acquisition Module will be started.

Stopping the Data Acquisition Module

You can stop the Data Acquisition Module by killing its ID or its name.

Open a new shell in QNX Windows and type **sin** command to display system information. Remember ID number of **das_task** process. Issue the following command:

kill <cast_task_ID> <RETURN>

The following is a much simpler way:

slay das_task <RETURN>

Developing new filter function

The Data Acquisition Module is open for building and adding new filtering functions. If a user needs a specific function for preprocessing sensor data, the new routine can be added very easily.

First, a developer has to edit "das_task.c" source code, typing at the prompt:

vedit /ProcessVision/das_task.c <RETURN>

When the source code is open, go to line 429. This is the beginning of the **processing_of_data** function. Add in the following lines:

```
if ( ( s = strncmp ( channel[i]. function[j].function_type,
        "user_function", strlen ("user_function")) ) == 0 )
        {
                list = channel[i].function[j].parameters;
```

```
                    user_function ( AI_table[i], &channel[i], i,
                            channel[i].function[j].num_of_kwt, list);
                    continue;
                }
```

Where:

channel[i].function[j].function_type is a string that contains name of "j-th" applied function, on the "i-th" input channel. This function name was read from configuration file and stored in the structure channel[i].

user_function is the name of added user function.

list is a pointer variable to the structure **par** that contains the list of function variables (including passed key-word-triplets).

AI_table[i] is a pointer to the table that contains data for "i-th" input channel.

i is the channel number. It is an integer type variable.

channel[i].function[j].num_of_kwt is an integer variable that contains the number of passed variables from configuration file to a function.

After adding these new lines of code, save the **das_task.c** file.

Now, you have to define the prototype of *user_function*. Open the file **fun_type.h** typing at the prompt:

vedit /ProcessVision/fun_type.h <RETURN>

Go to the end of the file and type in:

```
extern  void    user_function (short __huge*, struct channel *, int, int, struct par *);
```

Save the **fun_type.h** file and close it.

You are ready to write your own data processing function. At the prompt type:

vedit *user_function*.**c <RETURN>**

Include the following header files in your *user_function*.c file editing the following lines:

```
#include "fun_type.h"
#include "globals.h"
#include "ch_structure.h"
#include "struct.h"
#include "proto.h"
#include "cc_lib.h"
#include <errno.h>
#include <stdio.h>
...
```

Start to write the body of your function. Type in:

```
void user_function ( short __huge *AI_tab_ptr, struct channel *chan, int chan_num, int num_of_kwt,
struct par * kwt)
{
        ...
        body of your program
        ...
}
```

Where:

user_function is the name of added user function.

AI_tab_ptr is a pointer to the table that contains data for the input channel.

chan is a pointer to the structure **channel**. Structure **channel** contains all relevant information for specific input channel.

chan_num is the channel number which data user will process.

num_of_kwt is an integer variable that contains a number of passed variables from configuration file to a function.

kwt is a pointer variable to the structure **par** that contains the list of function variables (including passed key-word-triplets).

Data stored in a table (you refer to this data by using pointer **AI_tab_ptr**) is presented in format from 0 - 4096, where 0 is lower limit and 4096 is upper limit. To convert data to voltage representation, you have to use within your program the following lines:

```
if ( ( chan->range_low < 0 ) && ( value > 2047 ) )
        value = (float) value - 4096;
if ( ( chan->range_low < 0)
        volt = (float) ( ( float ) ( value / 4096 ) * 2 * ( chan->range_high / 1000000 ) );
else
        volt = (float) ( ( float ) ( value / 4096 ) * ( chan->range_high / 1000000 ) );
```

Where:

value is a short type variable, obtained by **AI_tab_ptr** pointer (value = *AI_tab_ptr;)

2047 is the upper limit when the measured input variable is in the range of -1 +1 V, -5 +5 V and -10 +10 V (2047 presents 1V, 5V, 10V respectively). Negative voltage is expressed from 2047 up to 4096.

volt is float type variable that presents input variable in volts.

To obtain an actual value, a developer has to call **calibration** function within his/her program, before passing this value to point database. Here is the prototype of the **calibration** function:

float calibration (float , struct channel *)

To call this function type in:

real_value = calibration (volt, chan);

Where:

real_value is return float type variable from **calibration** function

volt is passed variable from your program expressed in volts.

chan is pointer to **structure channel**.

When you are finished with coding your processing function, open file **calibration.c**. Go to line 92 and type in:

```
case 9:
/*Comment*/
        real_value = some_function (volt);
        break;
```

Where:

case 9 is a new case within the body of calibration function . **9** will be your new channel type defined in your configuration file

real_value is a returned variable that presents actual value.

some_function **(volt)** is the calibration equation to convert voltage into an actual value. **volt** is the passing variable from the calling program. For example:

real_value = 75 * volt - 75;

Save and exit file **calibration.c**.

You have to recompile the Data Acquisition Module for all these changes to take effect. Open file **_das** issuing the following command:

vedit _das <RETURN>

At the end of third line (**- o** option during compilation) type in:

calibration.c user_function.c

Save and exit **_das** file.

Now, at the prompt type in:

concas <RETURN>

You have started compilation of your new function.

EXPERT SYSTEM

```
Object
@name = casting
@attribute = speed.@float
endObject

Object
@name = counter
@attribute = number.@integer
endObject

Object
@name = depression
@attribute = warning.@float
endObject

Object
@name = driver
@attribute = flag.@float
endObject

Object
@name = number_of_valley
@attribute = in_TC1.@float
endObject

Object
@name = rep1
@attribute = a_span.@float, m_time.@float, t_drop.@float
endObject

Object
@name = rep2
@attribute = a_span.@float, m_time.@float, t_drop.@float
endObject

Object
@name = rep3
@attribute = a_span.@float, m_time.@float, t_drop.@float
endObject

Object
@name = rep4
@attribute = a_span.@float, m_time.@float, t_drop.@float
endObject

Object
@name = rep5
@attribute = a_span.@float, m_time.@float, t_drop.@float
endObject
```

```
Object
@name = rep6
@attribute = a_span.@float, m_time.@float, t_drop.@float
endObject

Object
@name = REPORT
@attribute = addition_value.@integer, extension.@string, number.@integer,
        rule_name.@string
endObject

Object
@name = scan_window
@attribute = max_time.@float, min_time.@float
endObject

Object
@name = show
@attribute = belief1.@string, belief2.@string, belief3.@string,
        belief4.@string, belief5.@string, belief6.@string
endObject

Object
@name = start
@attribute = check.@float
endObject

Object
@name = start_looking_for
@attribute = depression.@float
endObject

Object
@name = TC
@attribute = ignore_rule_name.@string, next_number.@integer
endObject

Object
@name = TC0
@attribute = rule_examined.variable, temperature_drop.significant
endObject

Object
@name = TC1
@attribute = extension.@string, location.@float, rule_examined.variable,
        start_number.@integer, temperature_drop.significant, valley_number.@integer
endObject

Object
@name = TC11
@attribute = m_time.@float, t_base.@float, t_drop.@float,
        t_span.@float
endObject
```

```
Object
@name = TC12
@attribute = m_time.@float, t_base.@float, t_drop.@float,
        t_span.@float
endObject

Object
@name = TC13
@attribute = m_time.@float, t_base.@float, t_drop.@float,
        t_span.@float
endObject

Object
@name = TC14
@attribute = m_time.@float, t_base.@float, t_drop.@float,
        t_span.@float
endObject

Object
@name = TC15
@attribute = m_time.@float, t_base.@float, t_drop.@float,
        t_span.@float
endObject

Object
@name = TC2
@attribute = extension.@string, location.@float, rule_examined.variable,
        temperature_drop.significant, valley.zero, valley_number.@integer
endObject

Object
@name = TC21
@attribute = m_time.@float, t_base.@float, t_drop.@float,
        t_span.@float
endObject

Object
@name = TC22
@attribute = m_time.@float, t_base.@float, t_drop.@float,
        t_span.@float
endObject

Object
@name = TC23
@attribute = m_time.@float, t_base.@float, t_drop.@float,
        t_span.@float
endObject

Object
@name = TC24
@attribute = m_time.@float, t_base.@float, t_drop.@float,
        t_span.@float
endObject
```

```
Object
@name = TC25
@attribute = m_time.@float, t_base.@float, t_drop.@float,
        t_span.@float
endObject

Object
@name = TC2a
@attribute = rule_examined.variable, temperature_drop.significant
endObject

Object
@name = TC3
@attribute = extension.@string, location.@float, rule_examined.variable,
        temperature_drop.significant, valley.zero, valley_number.@integer
endObject

Object
@name = TC31
@attribute = m_time.@float, t_base.@float, t_drop.@float,
        t_span.@float
endObject

Object
@name = TC32
@attribute = m_time.@float, t_base.@float, t_drop.@float,
        t_span.@float
endObject

Object
@name = TC33
@attribute = m_time.@float, t_base.@float, t_drop.@float,
        t_span.@float
endObject

Object
@name = TC34
@attribute = m_time.@float, t_base.@float, t_drop.@float,
        t_span.@float
endObject

Object
@name = TC35
@attribute = m_time.@float, t_base.@float, t_drop.@float,
        t_span.@float
endObject

Object
@name = TC3a
@attribute = rule_examined.variable, temperature_drop.significant
endObject
```

```
Object
@name = TC4
@attribute = extension.@string, location.@float, rule_examined.variable,
        temperature_drop.significant, valley.zero, valley_number.@integer
endObject

Object
@name = TC41
@attribute = m_time.@float, t_base.@float, t_drop.@float,
        t_span.@float
endObject

Object
@name = TC42
@attribute = m_time.@float, t_base.@float, t_drop.@float,
        t_span.@float
endObject

Object
@name = TC43
@attribute = m_time.@float, t_base.@float, t_drop.@float,
        t_span.@float
endObject

Object
@name = TC44
@attribute = m_time.@float, t_base.@float, t_drop.@float,
        t_span.@float
endObject

Object
@name = TC45
@attribute = m_time.@float, t_base.@float, t_drop.@float,
        t_span.@float
endObject

Object
@name = TC4a
@attribute = rule_examined.variable, temperature_drop.significant
endObject

Object
@name = TCVL
@attribute = extension.@string
endObject

Object
@name = time_range
@attribute = avg_val.@float, max_val.@float, min_val.@float
endObject
```

```
Object
@name = valley
@attribute = avg_value.@float, default_span.@float, extension.@string,
        status.incomplete
endObject

Object
@name = valley_span
@attribute = avg_value.@float
endObject

Object
@name = VL
@attribute = number.@integer
endObject

Object
@name = VL0
@attribute = dummy.@string, size.significant, status_number.@integer
endObject

Object
@name = VL01
@attribute = m_time.@float, t_base.@float, t_drop.@float,
        t_span.@float
endObject

Object
@name = VL02
@attribute = m_time.@float, t_base.@float, t_drop.@float,
        t_span.@float
endObject

Object
@name = VL03
@attribute = m_time.@float, t_base.@float, t_drop.@float,
        t_span.@float
endObject

Object
@name = VL04
@attribute = m_time.@float, t_base.@float, t_drop.@float,
        t_span.@float
endObject

Object
@name = VL1
@attribute = dummy.@string, size.significant, status_number.@integer
endObject
```

```
Object
@name = VL11
@attribute = m_time.@float, t_base.@float, t_drop.@float,
        t_span.@float
endObject

Object
@name = VL12
@attribute = m_time.@float, t_base.@float, t_drop.@float,
        t_span.@float
endObject

Object
@name = VL13
@attribute = m_time.@float, t_base.@float, t_drop.@float,
        t_span.@float
endObject

Object
@name = VL14
@attribute = m_time.@float, t_base.@float, t_drop.@float,
        t_span.@float
endObject

Object
@name = VL2
@attribute = dummy.@string, size.significant, status_number.@integer
endObject

Object
@name = VL21
@attribute = m_time.@float, t_base.@float, t_drop.@float,
        t_span.@float
endObject

Object
@name = VL22
@attribute = m_time.@float, t_base.@float, t_drop.@float,
        t_span.@float
endObject

Object
@name = VL23
@attribute = m_time.@float, t_base.@float, t_drop.@float,
        t_span.@float
endObject

Object
@name = VL24
@attribute = m_time.@float, t_base.@float, t_drop.@float,
        t_span.@float
endObject
```

```
Object
@name = VL3
@attribute = dummy.@string, size.significant, status_number.@integer
endObject

Object
@name = VL31
@attribute = m_time.@float, t_base.@float, t_drop.@float,
        t_span.@float
endObject

Object
@name = VL32
@attribute = m_time.@float, t_base.@float, t_drop.@float,
        t_span.@float
endObject

Object
@name = VL33
@attribute = m_time.@float, t_base.@float, t_drop.@float,
        t_span.@float
endObject

Object
@name = VL34
@attribute = m_time.@float, t_base.@float, t_drop.@float,
        t_span.@float
endObject

Object
@name = VL4
@attribute = dummy.@string, size.significant, status_number.@integer
endObject

Object
@name = VL41
@attribute = m_time.@float, t_base.@float, t_drop.@float,
        t_span.@float
endObject

Object
@name = VL42
@attribute = m_time.@float, t_base.@float, t_drop.@float,
        t_span.@float
endObject

Object
@name = VL43
@attribute = m_time.@float, t_base.@float, t_drop.@float,
        t_span.@float
endObject
```

```
Object
@name = VL44
@attribute = m_time.@float, t_base.@float, t_drop.@float,
        t_span.@float
endObject

Object
@name = VL5
@attribute = dummy.@string, size.significant, status_number.@integer
endObject

Object
@name = VL51
@attribute = m_time.@float, t_base.@float, t_drop.@float,
        t_span.@float
endObject

Object
@name = VL52
@attribute = m_time.@float, t_base.@float, t_drop.@float,
        t_span.@float
endObject

Object
@name = VL53
@attribute = m_time.@float, t_base.@float, t_drop.@float,
        t_span.@float
endObject

Object
@name = VL54
@attribute = m_time.@float, t_base.@float, t_drop.@float,
        t_span.@float
endObject

Object
@name = wind
@attribute = size.true, size.@float
endObject

Inference
@name = BELIEF_1
@and = $min
@or = $max
@mathexpr = $min
@conclusion =  ( CERTAINTY ( TC1.temperature_drop.significant ) * 0.600000 )
endInference
```

```
Inference
@name = BELIEF_2
@and = $min
@or = $max
@mathexpr = $min
@conclusion = ( CERTAINTY ( TC1.temperature_drop.significant ) * 0.600000 + CERTAINTY (
TC2.temperature_drop.significant ) * 0.150000 ) * 0.800000 / 0.750000
endInference

Inference
@name = BELIEF_3
@and = $min
@or = $max
@mathexpr = $min
@conclusion = ( CERTAINTY ( TC1.temperature_drop.significant ) * 0.600000 + CERTAINTY (
TC2.temperature_drop.significant ) * 0.150000 + CERTAINTY ( TC3.temperature_drop.significant ) *
0.150000 ) * 0.950000 / 0.900000
endInference

Inference
@name = BELIEF_4
@and = $min
@or = $max
@mathexpr = $min
@conclusion = ( CERTAINTY ( TC0.temperature_drop.significant ) * 0.600000 + CERTAINTY (
TC2.temperature_drop.significant ) * 0.150000 + CERTAINTY ( TC3.temperature_drop.significant ) *
0.150000 + CERTAINTY ( TC4.temperature_drop.significant ) * 0.100000 )
endInference

Inference
@name = BELIEF_4a
@and = $min
@or = $max
@mathexpr = $min
@conclusion = ( CERTAINTY ( TC1.temperature_drop.significant ) * 0.600000 + CERTAINTY (
TC2.temperature_drop.significant ) * 0.150000 + CERTAINTY ( TC3.temperature_drop.significant ) *
0.150000 + CERTAINTY ( TC4.temperature_drop.significant ) * 0.100000 )
endInference

Fuzzy
@name = TEMP_DROP_SIGNIFICANT
@source = ( \VL\<valley.extension.@string>.t_drop.@float ) / (
\VL\<valley.extension.@string>.t_base.@float )
@range = 10
@value = 0.000000, 0.010000, 0.020000, 0.030000, 0.040000, 0.050000, 0.060000, 0.080000, 0.100000,
0.150000
@rank = 0.000000, 1.000000, 5.000000, 25.000000, 50.000000, 60.000000, 75.000000, 90.000000,
95.000000, 100.000000
endFuzzy

Rule
@name = a
IF TRUE
THEN ASKCC ( driver.flag.@float )
```

```
THEN ASKCC ( start_looking_for.depression.@float )
THEN FREERULE ( $Rule, "a1" )
THEN GOTO ( "a1" )
endRule

Rule
@name = a1
IF  driver.flag.@float  > 0
AND  start_looking_for.depression.@float  > 0
THEN FORGET ( "*.*.*" )
THEN driver.flag.@float  = 1
THEN TC1.location.@float  = 220
THEN TC2.location.@float  = 335
THEN TC3.location.@float  = 435
THEN TC4.location.@float  = 535
THEN ASKCC ( casting.speed.@float )
THEN ASKCC ( TC11.m_time.@float )
THEN ASKCC ( TC11.t_base.@float )
THEN ASKCC ( TC11.t_drop.@float )
THEN ASKCC ( TC11.t_span.@float )
THEN ASKCC ( TC12.m_time.@float )
THEN ASKCC ( TC12.t_base.@float )
THEN ASKCC ( TC12.t_drop.@float )
THEN ASKCC ( TC12.t_span.@float )
THEN ASKCC ( TC13.m_time.@float )
THEN ASKCC ( TC13.t_base.@float )
THEN ASKCC ( TC13.t_drop.@float )
THEN ASKCC ( TC13.t_span.@float )
THEN ASKCC ( TC14.m_time.@float )
THEN ASKCC ( TC14.t_base.@float )
THEN ASKCC ( TC14.t_drop.@float )
THEN ASKCC ( TC14.t_span.@float )
THEN ASKCC ( TC15.m_time.@float )
THEN ASKCC ( TC15.t_base.@float )
THEN ASKCC ( TC15.t_drop.@float )
THEN ASKCC ( TC15.t_span.@float )
THEN ASKCC ( TC21.m_time.@float )
THEN ASKCC ( TC21.t_base.@float )
THEN ASKCC ( TC21.t_drop.@float )
THEN ASKCC ( TC21.t_span.@float )
THEN ASKCC ( TC22.m_time.@float )
THEN ASKCC ( TC22.t_base.@float )
THEN ASKCC ( TC22.t_drop.@float )
THEN ASKCC ( TC22.t_span.@float )
THEN ASKCC ( TC23.m_time.@float )
THEN ASKCC ( TC23.t_base.@float )
THEN ASKCC ( TC23.t_drop.@float )
THEN ASKCC ( TC23.t_span.@float )
THEN ASKCC ( TC24.m_time.@float )
THEN ASKCC ( TC24.t_base.@float )
THEN ASKCC ( TC24.t_drop.@float )
THEN ASKCC ( TC24.t_span.@float )
THEN ASKCC ( TC25.m_time.@float )
THEN ASKCC ( TC25.t_base.@float )
```

```
THEN ASKCC ( TC25.t_drop.@float  )
THEN ASKCC ( TC25.t_span.@float  )
THEN ASKCC ( TC31.m_time.@float  )
THEN ASKCC ( TC31.t_base.@float  )
THEN ASKCC ( TC31.t_drop.@float  )
THEN ASKCC ( TC31.t_span.@float  )
THEN ASKCC ( TC32.m_time.@float  )
THEN ASKCC ( TC32.t_base.@float  )
THEN ASKCC ( TC32.t_drop.@float  )
THEN ASKCC ( TC32.t_span.@float  )
THEN ASKCC ( TC33.m_time.@float  )
THEN ASKCC ( TC33.t_base.@float  )
THEN ASKCC ( TC33.t_drop.@float  )
THEN ASKCC ( TC33.t_span.@float  )
THEN ASKCC ( TC34.m_time.@float  )
THEN ASKCC ( TC34.t_base.@float  )
THEN ASKCC ( TC34.t_drop.@float  )
THEN ASKCC ( TC34.t_span.@float  )
THEN ASKCC ( TC35.m_time.@float  )
THEN ASKCC ( TC35.t_base.@float  )
THEN ASKCC ( TC35.t_drop.@float  )
THEN ASKCC ( TC35.t_span.@float  )
THEN ASKCC ( TC41.m_time.@float  )
THEN ASKCC ( TC41.t_base.@float  )
THEN ASKCC ( TC41.t_drop.@float  )
THEN ASKCC ( TC41.t_span.@float  )
THEN ASKCC ( TC42.m_time.@float  )
THEN ASKCC ( TC42.t_base.@float  )
THEN ASKCC ( TC42.t_drop.@float  )
THEN ASKCC ( TC42.t_span.@float  )
THEN ASKCC ( TC43.m_time.@float  )
THEN ASKCC ( TC43.t_base.@float  )
THEN ASKCC ( TC43.t_drop.@float  )
THEN ASKCC ( TC43.t_span.@float  )
THEN ASKCC ( TC44.m_time.@float  )
THEN ASKCC ( TC44.t_base.@float  )
THEN ASKCC ( TC44.t_drop.@float  )
THEN ASKCC ( TC44.t_span.@float  )
THEN ASKCC ( TC45.m_time.@float  )
THEN ASKCC ( TC45.t_base.@float  )
THEN ASKCC ( TC45.t_drop.@float  )
THEN ASKCC ( TC45.t_span.@float  )
THEN ASKCC ( number_of_valley.in_TC1.@float  )
THEN FREERULE ( $Rule, "a2" )
THEN FREERULE ( $Rule, "a3" )
THEN FREERULE ( $Rule, "no_valley" )
THEN FREERULE ( $Rule, "report0" )
THEN report.number.@integer = 0
THEN valley.status.incomplete is TRUE CF=0.00
THEN counter.number.@integer = 1
THEN TC1.start_number.@integer = 1
THEN IMPORT ( "file.xxx", 0, 100 )
THEN FREERULE ( $Rule, "no_valley" )
THEN MACRO ( "no_valley" )
```

```
THEN FREERULE ( $Rule, "a1a" )
THEN GOTO ( "a1a" )
ELSE WAIT ( $Rule, "*", 5 )
ELSE FREERULE ( $Rule, "a" )
ELSE GOTO ( "a" )
endRule

Rule
@name = a1a
IF   valley.status.incomplete is TRUE
THEN FREERULE ( $Rule, "a2" )
THEN GOTO ( "a2" )
ELSE FREERULE ( $Rule, "a3" )
ELSE GOTO ( "a3" )
endRule

Rule
@name = a2
IF   valley.status.incomplete is TRUE
AND  counter.number.@integer <= number_of_valley.in_TC1.@float
THEN FORGET ( "VL*.dummy.@s" )
THEN FREERULE ( $Rule, "Valley_*" )
THEN FIND ( "VL0.dummy.@s" )
THEN FIND ( "VL1.dummy.@s" )
THEN FIND ( "VL2.dummy.@s" )
THEN FIND ( "VL3.dummy.@s" )
THEN FIND ( "VL4.dummy.@s" )
THEN FIND ( "VL5.dummy.@s" )
THEN report.rule_name.@string is STRCONCAT ( "report", FORMAT ( report.number.@integer, "%ld"
) )
THEN FREERULE ( $Rule, report.rule_name.@string )
THEN MACRO ( report.rule_name.@string )
THEN IGNORE ( $Rule, "a3" )
THEN driver.flag.@float  = 0
THEN start_looking_for.depression.@float  = 0
THEN start.check.@float  = 70
THEN FREERULE ( $Rule, "a1" )
THEN GOTO ( "a1" )
ELSE driver.flag.@float  = 0
ELSE start_looking_for.depression.@float  = 0
ELSE start.check.@float  = 60
ELSE FREERULE ( $Rule, "a" )
ELSE GOTO ( "a" )
endRule

Rule
@name = a3
IF   valley.status.incomplete is FALSE
AND  counter.number.@integer <= number_of_valley.in_TC1.@float
THEN FORGET ( "VL*.dummy.@s" )
THEN FREERULE ( $Rule, "Valley_*" )
THEN FIND ( "VL1.dummy.@s" )
THEN FIND ( "VL2.dummy.@s" )
THEN FIND ( "VL3.dummy.@s" )
```

```
THEN FIND ( "VL4.dummy.@s" )
THEN FIND ( "VL5.dummy.@s" )
THEN report.rule_name.@string is STRCONCAT ( "report", FORMAT ( report.number.@integer, "%ld"
) )
THEN FREERULE ( $Rule, report.rule_name.@string )
THEN MACRO ( report.rule_name.@string )
THEN driver.flag.@float  = 0
THEN start_looking_for.depression.@float  = 0
THEN start.check.@float  = 70
THEN FREERULE ( $Rule, "a1" )
THEN GOTO ( "a1" )
ELSE driver.flag.@float  = 0
ELSE start_looking_for.depression.@float  = 0
ELSE start.check.@float  = 60
ELSE FREERULE ( $Rule, "a" )
ELSE GOTO ( "a" )
endRule

Rule
@name = belief_calc_1
@inference = BELIEF_1
IF  \VL\<VL.number.@integer>.status_number.@integer == 1
THEN valley.extension.@string is STRCONCAT ( FORMAT ( VL.number.@integer, "%ld" ), "1" )
THEN VL01.t_drop.@float  = \VL\<valley.extension.@string>.t_drop.@float
THEN VL01.t_span.@float  = \VL\<valley.extension.@string>.t_span.@float
THEN VL01.t_base.@float  = \VL\<valley.extension.@string>.t_base.@float
THEN VL01.m_time.@float  = \VL\<valley.extension.@string>.m_time.@float
THEN valley.status.incomplete is TRUE
THEN counter.number.@integer = 0
THEN ASNCERTAINTY ( valley.status.incomplete, 100 )
THEN ASNCERTAINTY ( counter.number.@integer, 100 )
THEN ASNCERTAINTY ( VL01.t_drop.@float , 100 )
THEN ASNCERTAINTY ( VL01.t_span.@float , 100 )
THEN ASNCERTAINTY ( VL01.t_base.@float , 100 )
THEN ASNCERTAINTY ( VL01.m_time.@float , 100 )
THEN EXPORT ( "file.xxx", "valley.status.incomplete", 0, 100 )
THEN EXPORT ( "file.xxx+", "counter.number.@i", 0, 100 )
THEN EXPORT ( "file.xxx+", "VL01.0.@f file.xxx", 0, 100 )
THEN EXPORT ( "file.xxx+", "VL01.1.@f file.xxx", 0, 100 )
THEN EXPORT ( "file.xxx+", "VL01.2.@f file.xxx", 0, 100 )
THEN EXPORT ( "file.xxx+", "VL01.3.@f file.xxx", 0, 100 )
THEN EXPORT ( "file.xxx+", "VL01.4.@f file.xxx", 0, 100 )
THEN EXPORT ( "file.xxx+", "VL01.5.@f file.xxx", 0, 100 )
THEN IGNORE ( $Rule, "belief_*" )
THEN report.addition_value.@integer = 0
THEN ASNCERTAINTY ( report.addition_value.@integer, 100 )
ELSE report.addition_value.@integer = 1
ELSE ASNCERTAINTY ( report.addition_value.@integer, 100 )
endRule

Rule
@name = belief_calc_2
@inference = BELIEF_2
IF  \VL\<VL.number.@integer>.status_number.@integer == 2
```

```
THEN \VL\<VL.number.@integer>.size.significant is TRUE
THEN valley.status.incomplete is TRUE CF=0.00
THEN ASNCERTAINTY ( valley.status.incomplete, 0 )
THEN EXPORT ( "file.xxx", "valley.status.incomplete", 0, 100 )
THEN IGNORE ( $Rule, "belief_*" )
endRule

Rule
@name = belief_calc_3
@inference = BELIEF_3
IF  \VL\<VL.number.@integer>.status_number.@integer == 3
THEN \VL\<VL.number.@integer>.size.significant is TRUE
THEN valley.status.incomplete is TRUE CF=0.00
THEN ASNCERTAINTY ( valley.status.incomplete, 0 )
THEN EXPORT ( "file.xxx", "valley.status.incomplete", 0, 100 )
THEN IGNORE ( $Rule, "belief_*" )
endRule

Rule
@name = belief_calc_4
@inference = BELIEF_4
IF  VL.number.@integer == 0
AND  \VL\<VL.number.@integer>.status_number.@integer == 4
THEN \VL\<VL.number.@integer>.size.significant is TRUE
THEN valley.status.incomplete is TRUE CF=0.00
THEN ASNCERTAINTY ( valley.status.incomplete, 0 )
THEN EXPORT ( "file.xxx", "valley.status.incomplete", 0, 100 )
THEN IGNORE ( $Rule, "belief_*" )
endRule

Rule
@name = belief_calc_4a
@inference = BELIEF_4a
IF  VL.number.@integer > 0
AND  \VL\<VL.number.@integer>.status_number.@integer == 4
THEN \VL\<VL.number.@integer>.size.significant is TRUE
THEN valley.status.incomplete is TRUE CF=0.00
THEN ASNCERTAINTY ( valley.status.incomplete, 0 )
THEN EXPORT ( "file.xxx", "valley.status.incomplete", 0, 100 )
endRule

Rule
@name = check_time_range
IF  KNOWN ( \TC\<TC.next_number.@integer>.valley_number.@integer )
THEN \TC\<TC.next_number.@integer>.extension.@string is FORMAT (
\TC\<TC.next_number.@integer>.valley_number.@integer, "%ld" )
THEN \VL\<VL.number.@integer>.status_number.@integer = TC.next_number.@integer - 1
THEN TC.ignore_rule_name.@string is STRCONCAT ( "XX", FORMAT ( TC.next_number.@integer,
"%ld" ) )
THEN IGNORE ( $Rule, TC.ignore_rule_name.@string )
ELSE TC.ignore_rule_name.@string is STRCONCAT ( "TC", FORMAT ( TC.next_number.@integer,
"%ld" ) )
ELSE IGNORE ( $Rule, TC.ignore_rule_name.@string )
endRule
```

```
Rule
@name = no_valley
IF  number_of_valley.in_TC1.@float  == 0
AND  valley.status.incomplete is FALSE
THEN driver.flag.@float  = 0
THEN FREERULE ( $Rule, "a" )
endRule

Rule
@name = pre_report
IF  \VL\<VL.number.@integer>.status_number.@integer > 1
THEN report.extension.@string is STRCONCAT ( FORMAT ( VL.number.@integer, "%ld" ), "1" )
THEN \rep\<report.number.@integer>.m_time.@float  =
\VL\<report.extension.@string>.m_time.@float
THEN \rep\<report.number.@integer>.t_drop.@float  = CERTAINTY (
\VL\<VL.number.@integer>.size.significant )
THEN \rep\<report.number.@integer>.a_span.@float  = valley_span.avg_value.@float
endRule

Rule
@name = report0
IF  report.number.@integer == 0
THEN IGNORE ( $Rule, "report*" )
THEN driver.flag.@float  = 0
THEN FREERULE ( $Rule, "a1" )
THEN GOTO ( "a1" )
endRule

Rule
@name = report1
IF  report.number.@integer == 1
THEN TEXT ( "NUMBER OF VALLEYS  DETECTED  = 1", "alarm" )
THEN TEXT ( "TIME is !$FORMAT(rep1.m_time.@f,"%6.2lf")$!", "alarm" )
THEN TEXT ( "BELIEF is !$FORMAT(rep1.t_drop.@f,"%4.0lf")$!", "alarm" )
THEN TEXT ( "SPAN is !$FORMAT(rep1.a_span.@f,"%6.2lf")$!", "alarm" )
THEN IGNORE ( $Rule, "report*" )
THEN depression.warning.@float  = 1
THEN start.check.@float  = 70
THEN show.belief1.@string = FORMAT ( rep1.t_drop.@float  , "%ld" )
THEN FIND ( "wind.size.true" )
endRule

Rule
@name = report2
IF  report.number.@integer == 2
THEN TEXT ( "NUMBER OF VALLEYS  DETECTED  = 2", "alarm" )
THEN TEXT ( "First TIME is !$FORMAT(rep1.m_time.@f,"%6.2lf")$!", "alarm" )
THEN TEXT ( "First BELIEF is !$FORMAT(rep1.t_drop.@f,"%4.0lf")$!", "alarm" )
THEN TEXT ( "First SPAN is !$FORMAT(rep1.a_span.@f,"%6.2lf")$!", "alarm" )
THEN TEXT ( "Second TIME is !$FORMAT(rep2.m_time.@f,"%6.2lf")$!", "alarm" )
THEN TEXT ( "Second BELIEF is !$FORMAT(rep2.t_drop.@f,"%4.0lf")$!", "alarm" )
THEN TEXT ( "Second SPAN is !$FORMAT(rep2.a_span.@f,"%6.2lf")$!", "alarm" )
THEN IGNORE ( $Rule, "report*" )
```

```
THEN depression.warning.@float  = 1
THEN start.check.@float  = 70
THEN show.belief1.@string = FORMAT ( rep1.t_drop.@float , "%ld" )
THEN show.belief2.@string = FORMAT ( rep2.t_drop.@float , "%ld" )
THEN FIND ( "wind.size.true" )
endRule

Rule
@name = report3
IF  report.number.@integer == 3
THEN TEXT ( "NUMBER OF VALLEYS  DETECTED  = 3", "alarm" )
THEN TEXT ( "First TIME is !$FORMAT(rep1.m_time.@f,"%6.2lf")$!", "alarm" )
THEN TEXT ( "First BELIEF is !$FORMAT(rep1.t_drop.@f,"%4.0lf")$!", "alarm" )
THEN TEXT ( "First SPAN is !$FORMAT(rep1.a_span.@f,"%6.2lf")$!", "alarm" )
THEN TEXT ( "Second TIME is !$FORMAT(rep2.m_time.@f,"%6.2lf")$!", "alarm" )
THEN TEXT ( "Second BELIEF is !$FORMAT(rep2.t_drop.@f,"%4.0lf")$!", "alarm" )
THEN TEXT ( "Second SPAN is !$FORMAT(rep2.a_span.@f,"%6.2lf")$!", "alarm" )
THEN TEXT ( "Third TIME is !$FORMAT(rep3.m_time.@f,"%6.2lf")$!", "alarm" )
THEN TEXT ( "Third BELIEF is !$FORMAT(rep3.t_drop.@f,"%4.0lf")$!", "alarm" )
THEN TEXT ( "Third SPAN is !$FORMAT(rep3.a_span.@f,"%6.2lf")$!", "alarm" )
THEN IGNORE ( $Rule, "report*" )
THEN depression.warning.@float  = 1
THEN start.check.@float  = 70
THEN show.belief1.@string = FORMAT ( rep1.t_drop.@float , "%ld" )
THEN show.belief2.@string = FORMAT ( rep2.t_drop.@float , "%ld" )
THEN show.belief3.@string = FORMAT ( rep3.t_drop.@float , "%ld" )
THEN FIND ( "wind.size.true" )
endRule

Rule
@name = report4
IF  report.number.@integer == 4
THEN TEXT ( "NUMBER OF VALLEYS  DETECTED  = 4", "alarm" )
THEN TEXT ( "First TIME is !$FORMAT(rep1.m_time.@f,"%6.2lf")$!", "alarm" )
THEN TEXT ( "First BELIEF is !$FORMAT(rep1.t_drop.@f,"%4.0lf")$!", "alarm" )
THEN TEXT ( "First SPAN is !$FORMAT(rep1.a_span.@f,"%6.2lf")$!", "alarm" )
THEN TEXT ( "Second TIME is !$FORMAT(rep2.m_time.@f,"%6.2lf")$!", "alarm" )
THEN TEXT ( "Second BELIEF is !$FORMAT(rep2.t_drop.@f,"%4.0lf")$!", "alarm" )
THEN TEXT ( "Second SPAN is !$FORMAT(rep2.a_span.@f,"%6.2lf")$!", "alarm" )
THEN TEXT ( "Third TIME is !$FORMAT(rep3.m_time.@f,"%6.2lf")$!", "alarm" )
THEN TEXT ( "Third BELIEF is !$FORMAT(rep3.t_drop.@f,"%4.0lf")$!", "alarm" )
THEN TEXT ( "Third SPAN is !$FORMAT(rep3.a_span.@f,"%6.2lf")$!", "alarm" )
THEN TEXT ( "Fourth TIME is !$FORMAT(rep4.m_time.@f,"%6.2lf")$!", "alarm" )
THEN TEXT ( "Fourth BELIEF is !$FORMAT(rep4.t_drop.@f,"%4.0lf")$!", "alarm" )
THEN TEXT ( "Fourth SPAN is !$FORMAT(rep4.a_span.@f,"%6.2lf")$!", "alarm" )
THEN IGNORE ( $Rule, "report*" )
THEN depression.warning.@float  = 1
THEN start.check.@float  = 70
THEN show.belief1.@string = FORMAT ( rep1.t_drop.@float , "%ld" )
THEN show.belief2.@string = FORMAT ( rep2.t_drop.@float , "%ld" )
THEN show.belief3.@string = FORMAT ( rep3.t_drop.@float , "%ld" )
THEN show.belief4.@string = FORMAT ( rep4.t_drop.@float , "%ld" )
THEN FIND ( "wind.size.true" )
endRule
```

```
Rule
@name = report5
IF   report.number.@integer == 5
THEN TEXT ( "NUMBER OF VALLEYS  DETECTED  = 5", "alarm" )
THEN TEXT ( "First TIME is !$FORMAT(rep1.m_time.@f,"%6.2lf")$!", "alarm" )
THEN TEXT ( "First BELIEF is !$FORMAT(rep1.t_drop.@f,"%4.0lf")$!", "alarm" )
THEN TEXT ( "First SPAN is !$FORMAT(rep1.a_span.@f,"%6.2lf")$!", "alarm" )
THEN TEXT ( "Second TIME is !$FORMAT(rep2.m_time.@f,"%6.2lf")$!", "alarm" )
THEN TEXT ( "Second BELIEF is !$FORMAT(rep2.t_drop.@f,"%4.0lf")$!", "alarm" )
THEN TEXT ( "Second SPAN is !$FORMAT(rep2.a_span.@f,"%6.2lf")$!", "alarm" )
THEN TEXT ( "Third TIME is !$FORMAT(rep3.m_time.@f,"%6.2lf")$!", "alarm" )
THEN TEXT ( "Third BELIEF is !$FORMAT(rep3.t_drop.@f,"%4.0lf")$!", "alarm" )
THEN TEXT ( "Third SPAN is !$FORMAT(rep3.a_span.@f,"%6.2lf")$!", "alarm" )
THEN TEXT ( "Fourth TIME is !$FORMAT(rep4.m_time.@f,"%6.2lf")$!", "alarm" )
THEN TEXT ( "Fourth BELIEF is !$FORMAT(rep4.t_drop.@f,"%4.0lf")$!", "alarm" )
THEN TEXT ( "Fourth SPAN is !$FORMAT(rep4.a_span.@f,"%6.2lf")$!", "alarm" )
THEN TEXT ( "Fifth TIME is !$FORMAT(rep5.m_time.@f,"%6.2lf")$!", "alarm" )
THEN TEXT ( "Fifth BELIEF is !$FORMAT(rep5.t_drop.@f,"%4.0lf")$!", "alarm" )
THEN TEXT ( "Fifth SPAN is !$FORMAT(rep5.a_span.@f,"%6.2lf")$!", "alarm" )
THEN IGNORE ( $Rule, "report*" )
THEN depression.warning.@float  = 1
THEN start.check.@float  = 70
THEN show.belief1.@string = FORMAT ( rep1.t_drop.@float , "%ld" )
THEN show.belief2.@string = FORMAT ( rep2.t_drop.@float , "%ld" )
THEN show.belief3.@string = FORMAT ( rep3.t_drop.@float , "%ld" )
THEN show.belief4.@string = FORMAT ( rep4.t_drop.@float , "%ld" )
THEN show.belief5.@string = FORMAT ( rep5.t_drop.@float , "%ld" )
THEN FIND ( "wind.size.true" )
endRule

Rule
@name = report6
IF   report.number.@integer == 6
THEN TEXT ( "NUMBER OF VALLEYS  DETECTED  = 6", "alarm" )
THEN TEXT ( "First TIME is !$FORMAT(rep1.m_time.@f,"%6.2lf")$!", "alarm" )
THEN TEXT ( "First BELIEF is !$FORMAT(rep1.t_drop.@f,"%4.0lf")$!", "alarm" )
THEN TEXT ( "First SPAN is !$FORMAT(rep1.a_span.@f,"%6.2lf")$!", "alarm" )
THEN TEXT ( "Second TIME is !$FORMAT(rep2.m_time.@f,"%6.2lf")$!", "alarm" )
THEN TEXT ( "Second BELIEF is !$FORMAT(rep2.t_drop.@f,"%4.0lf")$!", "alarm" )
THEN TEXT ( "Second SPAN is !$FORMAT(rep2.a_span.@f,"%6.2lf")$!", "alarm" )
THEN TEXT ( "Third TIME is !$FORMAT(rep3.m_time.@f,"%6.2lf")$!", "alarm" )
THEN TEXT ( "Third BELIEF is !$FORMAT(rep3.t_drop.@f,"%4.0lf")$!", "alarm" )
THEN TEXT ( "Third SPAN is !$FORMAT(rep3.a_span.@f,"%6.2lf")$!", "alarm" )
THEN TEXT ( "Fourth TIME is !$FORMAT(rep4.m_time.@f,"%6.2lf")$!", "alarm" )
THEN TEXT ( "Fourth BELIEF is !$FORMAT(rep4.t_drop.@f,"%4.0lf")$!", "alarm" )
THEN TEXT ( "Fourth SPAN is !$FORMAT(rep4.a_span.@f,"%6.2lf")$!", "alarm" )
THEN TEXT ( "Fifth TIME is !$FORMAT(rep5.m_time.@f,"%6.2lf")$!", "alarm" )
THEN TEXT ( "Fifth BELIEF is !$FORMAT(rep5.t_drop.@f,"%4.0lf")$!", "alarm" )
THEN TEXT ( "Fifth SPAN is !$FORMAT(rep5.a_span.@f,"%6.2lf")$!", "alarm" )
THEN TEXT ( "Sixth TIME is !$FORMAT(rep6.m_time.@f,"%6.2lf")$!", "alarm" )
THEN TEXT ( "Sixth BELIEF is !$FORMAT(rep6.t_drop.@f,"%4.0lf")$!", "alarm" )
THEN TEXT ( "Sixth SPAN is !$FORMAT(rep6.a_span.@f,"%6.2lf")$!", "alarm" )
THEN depression.warning.@float  = 1
```

```
THEN start.check.@float  = 70
THEN show.belief1.@string = FORMAT ( rep1.t_drop.@float , "%ld" )
THEN show.belief2.@string = FORMAT ( rep2.t_drop.@float , "%ld" )
THEN show.belief3.@string = FORMAT ( rep3.t_drop.@float , "%ld" )
THEN show.belief4.@string = FORMAT ( rep4.t_drop.@float , "%ld" )
THEN show.belief5.@string = FORMAT ( rep5.t_drop.@float , "%ld" )
THEN show.belief6.@string = FORMAT ( rep6.t_drop.@float , "%ld" )
THEN FIND ( "wind.size.true" )
endRule

Rule
@name = TC0a
IF  counter.number.@integer == 0
AND TC11.m_time.@float  <= ( VL01.m_time.@float  + VL01.t_span.@float  / 2 )
THEN valley.extension.@string is STRCONCAT ( FORMAT ( VL.number.@integer, "%ld" ), "1" )
THEN TC1.extension.@string is FORMAT ( counter.number.@integer, "%ld" )
THEN VL01.t_drop.@float  = MAX ( TC11.t_drop.@float , VL01.t_drop.@float  )
THEN VL01.t_span.@float  = MAX ( TC11.t_span.@float , VL01.t_span.@float  )
THEN VL01.t_base.@float  = MAX ( TC11.t_base.@float , VL01.t_base.@float  )
THEN VL01.m_time.@float  = ( TC11.m_time.@float  )
THEN valley_span.avg_value.@float  = VL01.t_span.@float
THEN FIND ( "TC0.temperature_drop.significant" )
THEN valley.default_span.@float  = VL01.t_span.@float
THEN time_range.avg_val.@float  = VL01.m_time.@float  + ( TC2.location.@float  -
TC1.location.@float  ) / casting.speed.@float
THEN time_range.min_val.@float  = time_range.avg_val.@float  - 0.500000 * VL01.t_span.@float
THEN time_range.max_val.@float  = time_range.avg_val.@float  + 0.500000 * VL01.t_span.@float
THEN TC.next_number.@integer = 2
THEN VL0.status_number.@integer = 1
THEN valley.extension.@string is STRCONCAT ( FORMAT ( VL.number.@integer, "%ld" ), FORMAT
( TC.next_number.@integer, "%ld" ) )
THEN TC0.rule_examined.variable is TRUE
THEN IGNORE ( $Rule, "TC1" )
THEN IGNORE ( $Rule, "TC0b" )
THEN FREERULE ( $Rule, "time_range_*" )
THEN MACRO ( "time_range_*" )
THEN MACRO ( "check_time_range" )
THEN counter.number.@integer = 1
endRule

Rule
@name = TC0b
IF  counter.number.@integer == 0
AND TC11.m_time.@float  > ( VL01.m_time.@float  + VL01.t_span.@float  / 2 )
THEN valley.extension.@string is STRCONCAT ( FORMAT ( VL.number.@integer, "%ld" ), "1" )
THEN TC1.extension.@string is FORMAT ( counter.number.@integer, "%ld" )
THEN VL01.t_drop.@float  = VL01.t_drop.@float
THEN VL01.t_span.@float  = VL01.t_span.@float
THEN VL01.t_base.@float  = VL01.t_base.@float
THEN VL01.m_time.@float  = VL01.m_time.@float
THEN valley_span.avg_value.@float  = VL01.t_span.@float
THEN FIND ( "TC0.temperature_drop.significant" )
THEN valley.default_span.@float  = VL01.t_span.@float
```

```
THEN time_range.avg_val.@float = VL01.m_time.@float + ( TC2.location.@float -
TC1.location.@float ) / casting.speed.@float
THEN time_range.min_val.@float = time_range.avg_val.@float - 0.500000 * TC11.t_span.@float
THEN time_range.max_val.@float = time_range.avg_val.@float + 0.500000 * TC11.t_span.@float
THEN TC.next_number.@integer = 2
THEN \VL\<VL.number.@integer>.status_number.@integer = 1
THEN valley.extension.@string is STRCONCAT ( FORMAT ( VL.number.@integer, "%ld" ), FORMAT
( TC.next_number.@integer, "%ld" ) )
THEN TC0.rule_examined.variable is TRUE
THEN IGNORE ( $Rule, "TC1" )
THEN FREERULE ( $Rule, "time_range_*" )
THEN MACRO ( "time_range_*" )
THEN MACRO ( "check_time_range" )
endRule

Rule
@name = TC1
IF TRUE
THEN valley.extension.@string is STRCONCAT ( FORMAT ( VL.number.@integer, "%ld" ), "1" )
THEN \VL\<valley.extension.@string>.t_drop.@float =
\TC1\<counter.number.@integer>.t_drop.@float
THEN \VL\<valley.extension.@string>.t_span.@float =
\TC1\<counter.number.@integer>.t_span.@float
THEN \VL\<valley.extension.@string>.t_base.@float = \TC1\<counter.number.@integer>.t_base.@float
THEN \VL\<valley.extension.@string>.m_time.@float =
\TC1\<counter.number.@integer>.m_time.@float
THEN valley_span.avg_value.@float = \VL\<valley.extension.@string>.t_span.@float
THEN FIND ( "TC1.temperature_drop.significant" )
THEN valley.default_span.@float = \TC1\<counter.number.@integer>.t_span.@float
THEN time_range.avg_val.@float = \TC1\<counter.number.@integer>.m_time.@float + (
TC2.location.@float - TC1.location.@float ) / casting.speed.@float
THEN time_range.min_val.@float = time_range.avg_val.@float - 0.500000 *
\TC1\<counter.number.@integer>.t_span.@float
THEN time_range.max_val.@float = time_range.avg_val.@float + 0.500000 *
\TC1\<counter.number.@integer>.t_span.@float
THEN TC.next_number.@integer = 2
THEN \VL\<VL.number.@integer>.status_number.@integer = 1
THEN valley.extension.@string is STRCONCAT ( FORMAT ( VL.number.@integer, "%ld" ), FORMAT
( TC.next_number.@integer, "%ld" ) )
THEN TC1.rule_examined.variable is TRUE
THEN FREERULE ( $Rule, "time_range_*" )
THEN MACRO ( "time_range_*" )
THEN MACRO ( "check_time_range" )
endRule

Rule
@name = TC2
IF TC2.valley.zero is FALSE
THEN \VL\<valley.extension.@string>.t_drop.@float = \TC2\<TC2.extension.@string>.t_drop.@float
THEN \VL\<valley.extension.@string>.t_span.@float = \TC2\<TC2.extension.@string>.t_span.@float
THEN \VL\<valley.extension.@string>.t_base.@float = \TC2\<TC2.extension.@string>.t_base.@float
THEN \VL\<valley.extension.@string>.m_time.@float =
\TC2\<TC2.extension.@string>.m_time.@float
```

```
THEN valley_span.avg_value.@float = ( valley_span.avg_value.@float +
\VL\<valley.extension.@string>.t_span.@float ) / 2
THEN FIND ( "TC2.temperature_drop.significant" )
THEN time_range.avg_val.@float = \TC2\<TC2.extension.@string>.m_time.@float + (
TC3.location.@float - TC2.location.@float ) / casting.speed.@float
THEN time_range.min_val.@float = time_range.avg_val.@float - 0.500000 *
\TC2\<TC2.extension.@string>.t_span.@float
THEN time_range.max_val.@float = time_range.avg_val.@float + 0.500000 *
\TC2\<TC2.extension.@string>.t_span.@float
THEN TC.next_number.@integer = 3
THEN \VL\<VL.number.@integer>.status_number.@integer = 2
THEN valley.extension.@string is STRCONCAT ( FORMAT ( VL.number.@integer, "%ld" ), FORMAT
( TC.next_number.@integer, "%ld" ) )
THEN TC2.rule_examined.variable is TRUE
THEN FREERULE ( $Rule, "time_range_*" )
THEN MACRO ( "time_range_*" )
THEN MACRO ( "check_time_range" )
endRule

Rule
@name = TC3
IF TC3.valley.zero is FALSE
THEN \VL\<valley.extension.@string>.t_drop.@float = \TC3\<TC3.extension.@string>.t_drop.@float
THEN \VL\<valley.extension.@string>.t_span.@float = \TC3\<TC3.extension.@string>.t_span.@float
THEN \VL\<valley.extension.@string>.t_base.@float = \TC3\<TC3.extension.@string>.t_base.@float
THEN \VL\<valley.extension.@string>.m_time.@float =
\TC3\<TC3.extension.@string>.m_time.@float
THEN valley_span.avg_value.@float = ( 2.000000 * valley_span.avg_value.@float +
\VL\<valley.extension.@string>.t_span.@float ) / 3
THEN FIND ( "TC3.temperature_drop.significant" )
THEN time_range.avg_val.@float = \TC3\<TC3.extension.@string>.m_time.@float + (
TC4.location.@float - TC3.location.@float ) / casting.speed.@float
THEN time_range.min_val.@float = time_range.avg_val.@float - 0.500000 *
\TC3\<TC3.extension.@string>.t_span.@float
THEN time_range.max_val.@float = time_range.avg_val.@float + 0.500000 *
\TC3\<TC3.extension.@string>.t_span.@float
THEN TC.next_number.@integer = 4
THEN \VL\<VL.number.@integer>.status_number.@integer = 3
THEN valley.extension.@string is STRCONCAT ( FORMAT ( VL.number.@integer, "%ld" ), FORMAT
( TC.next_number.@integer, "%ld" ) )
THEN TC3.rule_examined.variable is TRUE
THEN FREERULE ( $Rule, "time_range_*" )
THEN MACRO ( "time_range_*" )
THEN MACRO ( "check_time_range" )
endRule

Rule
@name = TC4
IF TC4.valley.zero is FALSE
THEN \VL\<valley.extension.@string>.t_drop.@float = \TC4\<TC4.extension.@string>.t_drop.@float
THEN \VL\<valley.extension.@string>.t_span.@float = \TC4\<TC4.extension.@string>.t_span.@float
THEN \VL\<valley.extension.@string>.t_base.@float = \TC4\<TC4.extension.@string>.t_base.@float
THEN \VL\<valley.extension.@string>.m_time.@float =
\TC4\<TC4.extension.@string>.m_time.@float
```

```
THEN valley_span.avg_value.@float  = ( 3.000000 * valley_span.avg_value.@float  +
\VL\<valley.extension.@string>.t_span.@float  ) / 4
THEN FIND ( "TC4.temperature_drop.significant" )
THEN \VL\<VL.number.@integer>.status_number.@integer = 4
THEN TC4.rule_examined.variable is TRUE
ELSE \VL\<valley.extension.@string>.m_time.@float  = time_range.avg_val.@float
ELSE FREERULE ( $Rule, "TC_default_rule" )
ELSE MACRO ( "TC_default_rule" )
ELSE FIND ( "TC4.temperature_drop.significant" )
ELSE \VL\<VL.number.@integer>.status_number.@integer = 4
ELSE TC4.rule_examined.variable is TRUE
endRule

Rule
@name = TC_default_rule
IF  TRUE
THEN \VL\<valley.extension.@string>.t_drop.@float  = 0.000000
THEN \VL\<valley.extension.@string>.t_span.@float  = 0.000000
THEN \VL\<valley.extension.@string>.t_base.@float  = 1.000000
THEN \VL\<valley.extension.@string>.m_time.@float  = time_range.avg_val.@float
THEN time_range.min_val.@float  = time_range.avg_val.@float  - 0.500000 *
valley.default_span.@float
THEN time_range.max_val.@float  = time_range.avg_val.@float  + 0.500000 *
valley.default_span.@float
endRule

Rule
@name = time_range_0
IF  time_range.min_val.@float  > scan_window.max_time.@float
THEN IGNORE ( $Rule, "XX*" )
THEN IGNORE ( $Rule, "TC*" )
THEN IGNORE ( $Rule, "time_range_*" )
THEN IGNORE ( $Rule, "check_time_range" )
ELSE TCVL.extension.@string is STRCONCAT ( FORMAT ( TC.next_number.@integer, "%ld" ), "1" )
ELSE FREERULE ( $Rule, "TC*" )
ELSE FREERULE ( $Rule, "XX*" )
ELSE \TC\<TC.next_number.@integer>.valley.zero is TRUE
ELSE ASNCERTAINTY ( \TC\<TC.next_number.@integer>.valley.zero, 100 )
ELSE FREERULE ( $Rule, "check_time_range" )
endRule

Rule
@name = time_range_1
IF  \TC\<TCVL.extension.@string>.m_time.@float  >= time_range.min_val.@float
AND  \TC\<TCVL.extension.@string>.m_time.@float  < time_range.max_val.@float
THEN \TC\<TC.next_number.@integer>.valley_number.@integer = 1
THEN IGNORE ( $Rule, "time_range_*" )
THEN \TC\<TC.next_number.@integer>.valley.zero is FALSE
THEN ASNCERTAINTY ( \TC\<TC.next_number.@integer>.valley.zero, 0 )
ELSE TCVL.extension.@string is STRCONCAT ( FORMAT ( TC.next_number.@integer, "%ld" ), "2" )
ELSE \TC\<TC.next_number.@integer>.valley.zero is TRUE
ELSE ASNCERTAINTY ( \TC\<TC.next_number.@integer>.valley.zero, 100 )
endRule
```

```
Rule
@name = time_range_2
IF  \TC\<TCVL.extension.@string>.m_time.@float  >= time_range.min_val.@float
AND \TC\<TCVL.extension.@string>.m_time.@float  < time_range.max_val.@float
THEN \TC\<TC.next_number.@integer>.valley_number.@integer = 2
THEN IGNORE ( $Rule, "time_range_*" )
THEN \TC\<TC.next_number.@integer>.valley.zero is FALSE
THEN ASNCERTAINTY ( \TC\<TC.next_number.@integer>.valley.zero, 0 )
ELSE TCVL.extension.@string is STRCONCAT ( FORMAT ( TC.next_number.@integer, "%ld" ), "3" )
ELSE \TC\<TC.next_number.@integer>.valley.zero is TRUE
ELSE ASNCERTAINTY ( \TC\<TC.next_number.@integer>.valley.zero, 100 )
endRule

Rule
@name = time_range_3
IF  \TC\<TCVL.extension.@string>.m_time.@float  >= time_range.min_val.@float
AND \TC\<TCVL.extension.@string>.m_time.@float  < time_range.max_val.@float
THEN \TC\<TC.next_number.@integer>.valley_number.@integer = 3
THEN IGNORE ( $Rule, "time_range_*" )
THEN \TC\<TC.next_number.@integer>.valley.zero is FALSE
THEN ASNCERTAINTY ( \TC\<TC.next_number.@integer>.valley.zero, 0 )
ELSE TCVL.extension.@string is STRCONCAT ( FORMAT ( TC.next_number.@integer, "%ld" ), "4" )
ELSE \TC\<TC.next_number.@integer>.valley.zero is TRUE
ELSE ASNCERTAINTY ( \TC\<TC.next_number.@integer>.valley.zero, 100 )
endRule

Rule
@name = time_range_4
IF  \TC\<TCVL.extension.@string>.m_time.@float  >= time_range.min_val.@float
AND \TC\<TCVL.extension.@string>.m_time.@float  < time_range.max_val.@float
THEN \TC\<TC.next_number.@integer>.valley_number.@integer = 4
THEN IGNORE ( $Rule, "time_range_*" )
THEN \TC\<TC.next_number.@integer>.valley.zero is FALSE
THEN ASNCERTAINTY ( \TC\<TC.next_number.@integer>.valley.zero, 0 )
ELSE TCVL.extension.@string is STRCONCAT ( FORMAT ( TC.next_number.@integer, "%ld" ), "5" )
ELSE \TC\<TC.next_number.@integer>.valley.zero is TRUE
ELSE ASNCERTAINTY ( \TC\<TC.next_number.@integer>.valley.zero, 100 )
endRule

Rule
@name = time_range_5
IF  \TC\<TCVL.extension.@string>.m_time.@float  >= time_range.min_val.@float
AND \TC\<TCVL.extension.@string>.m_time.@float  < time_range.max_val.@float
THEN \TC\<TC.next_number.@integer>.valley_number.@integer = 5
THEN \TC\<TC.next_number.@integer>.valley.zero is FALSE
THEN ASNCERTAINTY ( \TC\<TC.next_number.@integer>.valley.zero, 0 )
THEN IGNORE ( $Rule, "time_range_*" )
ELSE \TC\<TC.next_number.@integer>.valley.zero is TRUE
ELSE ASNCERTAINTY ( \TC\<TC.next_number.@integer>.valley.zero, 100 )
endRule

Rule
@name = Valley_0
IF  valley.status.incomplete is TRUE
```

```
AND  counter.number.@integer == 0
THEN VL.number.@integer = 0
THEN FREERULE ( $Rule, "TC*" )
THEN FREERULE ( $Rule, "xx*" )
THEN FORGET ( "TC*.rule_examined.variable" )
THEN FORGET ( "TC*.temperature_drop.significant" )
THEN FIND ( "TC0.rule_examined.variable" )
THEN FIND ( "TC2.rule_examined.variable" )
THEN FIND ( "TC3.rule_examined.variable" )
THEN FIND ( "TC4.rule_examined.variable" )
THEN FREERULE ( $Rule, "belief_*" )
THEN MACRO ( "belief_*" )
THEN VL0.dummy.@string is "test"
THEN counter.number.@integer = counter.number.@integer + 1
THEN report.number.@integer = report.number.@integer + report.addition_value.@integer
THEN FREERULE ( $Rule, "pre_report" )
THEN MACRO ( "pre_report" )
ELSE VL0.dummy.@string is "test"
ELSE report.number.@integer = 0
endRule

Rule
@name = Valley_1
IF   counter.number.@integer == 1
AND  counter.number.@integer <= number_of_valley.in_TC1.@float
THEN VL.number.@integer = 1
THEN FREERULE ( $Rule, "TC*" )
THEN FREERULE ( $Rule, "xx*" )
THEN FORGET ( "TC*.rule_examined.variable" )
THEN FORGET ( "TC*.temperature_drop.significant" )
THEN FIND ( "TC1.rule_examined.variable" )
THEN FIND ( "TC2.rule_examined.variable" )
THEN FIND ( "TC3.rule_examined.variable" )
THEN FIND ( "TC4.rule_examined.variable" )
THEN FREERULE ( $Rule, "belief_*" )
THEN MACRO ( "belief_*" )
THEN VL1.dummy.@string is "test"
THEN counter.number.@integer = counter.number.@integer + 1
THEN report.number.@integer = report.number.@integer + report.addition_value.@integer
THEN FREERULE ( $Rule, "pre_report" )
THEN MACRO ( "pre_report" )
ELSE VL1.dummy.@string is "test"
endRule

Rule
@name = Valley_2
IF   counter.number.@integer == 2
AND  counter.number.@integer <= number_of_valley.in_TC1.@float
THEN VL.number.@integer = 2
THEN FREERULE ( $Rule, "TC*" )
THEN FREERULE ( $Rule, "xx*" )
THEN FORGET ( "TC*.rule_examined.variable" )
THEN FORGET ( "TC*.temperature_drop.significant" )
THEN FIND ( "TC1.rule_examined.variable" )
```

```
THEN FIND ( "TC2.rule_examined.variable" )
THEN FIND ( "TC3.rule_examined.variable" )
THEN FIND ( "TC4.rule_examined.variable" )
THEN FREERULE ( $Rule, "belief_*" )
THEN MACRO ( "belief_*" )
THEN VL2.dummy.@string is "test"
THEN counter.number.@integer = counter.number.@integer + 1
THEN report.number.@integer = report.number.@integer + report.addition_value.@integer
THEN FREERULE ( $Rule, "pre_report" )
THEN MACRO ( "pre_report" )
ELSE VL2.dummy.@string is "test"
endRule

Rule
@name = Valley_3
IF  counter.number.@integer == 3
AND  counter.number.@integer <= number_of_valley.in_TC1.@float
THEN VL.number.@integer = 3
THEN FREERULE ( $Rule, "TC*" )
THEN FREERULE ( $Rule, "xx*" )
THEN FORGET ( "TC*.rule_examined.variable" )
THEN FORGET ( "TC*.temperature_drop.significant" )
THEN FIND ( "TC1.rule_examined.variable" )
THEN FIND ( "TC2.rule_examined.variable" )
THEN FIND ( "TC3.rule_examined.variable" )
THEN FIND ( "TC4.rule_examined.variable" )
THEN FREERULE ( $Rule, "belief_*" )
THEN MACRO ( "belief_*" )
THEN VL3.dummy.@string is "test"
THEN counter.number.@integer = counter.number.@integer + 1
THEN report.number.@integer = report.number.@integer + report.addition_value.@integer
THEN FREERULE ( $Rule, "pre_report" )
THEN MACRO ( "pre_report" )
ELSE VL3.dummy.@string is "test"
endRule

Rule
@name = Valley_4
IF  counter.number.@integer == 4
AND  counter.number.@integer <= number_of_valley.in_TC1.@float
THEN VL.number.@integer = 4
THEN FREERULE ( $Rule, "TC*" )
THEN FREERULE ( $Rule, "xx*" )
THEN FORGET ( "TC*.rule_examined.variable" )
THEN FORGET ( "TC*.temperature_drop.significant" )
THEN FIND ( "TC1.rule_examined.variable" )
THEN FIND ( "TC2.rule_examined.variable" )
THEN FIND ( "TC3.rule_examined.variable" )
THEN FIND ( "TC4.rule_examined.variable" )
THEN FREERULE ( $Rule, "belief_*" )
THEN MACRO ( "belief_*" )
THEN VL4.dummy.@string is "test"
THEN counter.number.@integer = counter.number.@integer + 1
THEN report.number.@integer = report.number.@integer + report.addition_value.@integer
```

```
THEN FREERULE ( $Rule, "pre_report" )
THEN MACRO ( "pre_report" )
ELSE VL4.dummy.@string is "test"
endRule

Rule
@name = Valley_5
IF  counter.number.@integer == 5
AND  counter.number.@integer <= number_of_valley.in_TC1.@float
THEN VL.number.@integer = 5
THEN FREERULE ( $Rule, "TC*" )
THEN FREERULE ( $Rule, "xx*" )
THEN FORGET ( "TC*.rule_examined.variable" )
THEN FORGET ( "TC*.temperature_drop.significant" )
THEN FIND ( "TC1.rule_examined.variable" )
THEN FIND ( "TC2.rule_examined.variable" )
THEN FIND ( "TC3.rule_examined.variable" )
THEN FIND ( "TC4.rule_examined.variable" )
THEN FREERULE ( $Rule, "belief_*" )
THEN MACRO ( "belief_*" )
THEN VL5.dummy.@string is "test"
THEN counter.number.@integer = counter.number.@integer + 1
THEN report.number.@integer = report.number.@integer + report.addition_value.@integer
THEN FREERULE ( $Rule, "pre_report" )
THEN MACRO ( "pre_report" )
endRule

Rule
@name = windows
IF  wind.size.@float  > ( 1 + rep1.t_drop.@float  )
THEN wind.size.@float  = wind.size.@float  + 1.000000
THEN wind.size.true is TRUE
endRule

Rule
@name = XX2
IF  TC2.valley.zero is TRUE
THEN \VL\<valley.extension.@string>.t_drop.@float  = 0.000000
THEN \VL\<valley.extension.@string>.t_span.@float  = 0.000000
THEN \VL\<valley.extension.@string>.t_base.@float  = 1.000000
THEN \VL\<valley.extension.@string>.m_time.@float  = time_range.avg_val.@float
THEN time_range.min_val.@float  = time_range.avg_val.@float  - 0.500000 *
valley.default_span.@float
THEN time_range.max_val.@float  = time_range.avg_val.@float  + 0.500000 *
valley.default_span.@float
THEN TC.ignore_rule_name.@string is STRCONCAT ( "TC", FORMAT ( TC.next_number.@integer,
"%ld" ) )
THEN IGNORE ( $Rule, TC.ignore_rule_name.@string )
THEN TC.next_number.@integer = 3
THEN \VL\<VL.number.@integer>.status_number.@integer = 2
THEN valley.extension.@string is STRCONCAT ( FORMAT ( VL.number.@integer, "%ld" ), "3" )
THEN ASNCERTAINTY ( TC2.temperature_drop.significant, 0 )
THEN TC2.rule_examined.variable is TRUE
THEN FREERULE ( $Rule, "time_range_*" )
```

```
THEN MACRO ( "time_range_*" )
THEN MACRO ( "check_time_range" )
endRule

Rule
@name = XX3
IF   TC3.valley.zero is TRUE
THEN \VL\<valley.extension.@string>.t_drop.@float   = 0.000000
THEN \VL\<valley.extension.@string>.t_span.@float   = 0.000000
THEN \VL\<valley.extension.@string>.t_base.@float   = 1.000000
THEN \VL\<valley.extension.@string>.m_time.@float   = time_range.avg_val.@float
THEN time_range.min_val.@float   = time_range.avg_val.@float   - 0.500000 *
valley.default_span.@float
THEN time_range.max_val.@float   = time_range.avg_val.@float   + 0.500000 *
valley.default_span.@float
THEN TC.ignore_rule_name.@string is STRCONCAT ( "TC", FORMAT ( TC.next_number.@integer,
"%ld" ) )
THEN IGNORE ( $Rule, TC.ignore_rule_name.@string )
THEN TC.next_number.@integer = 4
THEN \VL\<VL.number.@integer>.status_number.@integer = 3
THEN valley.extension.@string is STRCONCAT ( FORMAT ( VL.number.@integer, "%ld" ), "4" )
THEN ASNCERTAINTY ( TC3.temperature_drop.significant, 0 )
THEN TC3.rule_examined.variable is TRUE
THEN FREERULE ( $Rule, "time_range_*" )
THEN MACRO ( "time_range_*" )
THEN MACRO ( "check_time_range" )
endRule

Rule
@name = XX4
IF   TC4.valley.zero is TRUE
THEN \VL\<valley.extension.@string>.t_drop.@float   = 0.000000
THEN \VL\<valley.extension.@string>.t_span.@float   = 0.000000
THEN \VL\<valley.extension.@string>.t_base.@float   = 1.000000
THEN \VL\<valley.extension.@string>.m_time.@float   = time_range.avg_val.@float
THEN time_range.min_val.@float   = time_range.avg_val.@float   - 0.500000 *
valley.default_span.@float
THEN time_range.max_val.@float   = time_range.avg_val.@float   + 0.500000 *
valley.default_span.@float
THEN TC.ignore_rule_name.@string is STRCONCAT ( "TC", FORMAT ( TC.next_number.@integer,
"%ld" ) )
THEN IGNORE ( $Rule, TC.ignore_rule_name.@string )
THEN \VL\<VL.number.@integer>.status_number.@integer = 4
THEN ASNCERTAINTY ( TC4.temperature_drop.significant, 0 )
THEN TC4.rule_examined.variable is TRUE
THEN FREERULE ( $Rule, "time_range_*" )
THEN MACRO ( "time_range_*" )
endRule

Facets
@triplet = TC0.temperature_drop.significant
@fuzzy = TEMP_DROP_SIGNIFICANT
endFacets
```

```
Facets
@triplet = TC1.temperature_drop.significant
@fuzzy = TEMP_DROP_SIGNIFICANT
endFacets

Facets
@triplet = TC2.temperature_drop.significant
@fuzzy = TEMP_DROP_SIGNIFICANT
endFacets

Facets
@triplet = TC3.temperature_drop.significant
@fuzzy = TEMP_DROP_SIGNIFICANT
endFacets

Facets
@triplet = TC4.temperature_drop.significant
@fuzzy = TEMP_DROP_SIGNIFICANT
endFacets

!*** LoadStrategy must go at the end of the Knowledge Base ***!
LoadStrategy
@name = "fback.stg"
EndLoadStrategy
```