Fault Detection and Path Optimisation for a Meat-Processing Robot

A thesis

submitted in fulfilment

of the requirements for the Degree

of

Doctor of Philosophy in

Physics and Electronic Engineering

at the

University of Waikato

by

Shaun Anthony Hurd

The University of Waikato

Te Whare Wananga o Waikato

University of Waikato

2005

For Sister D

ABSTRACT

An automated Y-cutting system has been developed by the Automation Systems team of Industrial Research Limited. This robotic device performs the Y-cut operation on sheep carcasses. The robotic Y-cutting system must deal with a variety of carcass shapes and sizes, and it is important that process faults are detected, diagnosed and corrected as quickly as possible. This thesis addresses the fault detection and path optimisation requirements of the Y-cutting system.

The development of a neural network-based fault detection module is documented. This module classifies process faults using axial motor current data from the Y-cutting robot. The module successfully classifies 98% of the presented cut signals during offline training, and 100% of cuts during an extended trial in an Australian meat-processing plant. An online training scheme is implemented to allow for the retraining of the neural network weights as required. The fault detection module is extended to handle a greater number of fault conditions and to detect variations in the process load.

A path optimisation algorithm is developed to optimise the parameters that define the cut-path of the robot based on the output of the fault detection module. A line-search within the parameter space is used to estimate the position of the optimum parameter value. The optimisation of fifteen path parameters requires 4760 simulated Y-cuts, equating to approximately 1.5 days of processing in a typical meat-plant. This is significantly faster than the existing method for manually tuning the Y-cutting system. The fault detection and path optimisation systems can be generically applied to other robotic systems produced by Industrial Research Limited for the handling and processing of highly varying natural products.

ACKNOWLEDGEMENTS

I would like to take this opportunity to thank the people who assisted and supported me during this thesis.

I wish to express my sincere gratitude to Dr. Dale Carnegie, for his supervision and support through some difficult times. Special thanks also to Dr. Niven Brown and Dr. Paul Gaynor for their supervision, advice and tireless proofreading.

Thanks to Andrew Osborn and Jeffery Yang, for sharing the Goulburn experience with me and for their data collection efforts while I was absent. My thanks also to all of the members of Automation Systems, DeviceWorks and the extended IRL family for their help and assistance throughout this project.

I am especially grateful to Dr. Richard Templer for providing me with this research opportunity and welcoming me into the Automation Systems team.

I could not have completed this thesis without the financial support of the Foundation for Research, Science and Technology through the Bright Future Scholarship scheme. Additionally, Meat New Zealand and Meat and Livestock Australia have provided funding for the development of the automated Y-cutting system.

Thanks to Fiona, Alan and Keegan for their enduring love and support. Thanks also to my extended family and friends for their encouragement.

Finally to Jennifer, for her continuing inspiration and for reminding me of the more important things in life.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1. INTRODUCTION

1.1 OVERVIEW

Commercial robotic systems have been available for several decades for a variety of simple automation tasks (Kopacek, 1999), including assembly, machining and palletising (Friedrich, 1995). These tasks usually deal with identical products in highly structured environments, with the robot endlessly repeating the same motions.

In contrast, there has been only limited development of commercial robotic systems for the handling or processing of highly varying natural products, such as meat, timber or fruit (Spooner & Rodrigo, 1998). These products are generally non-uniform in size, shape and appearance. This means that any developed robotic system needs to be adaptive to deal with these product variations, and requires sophisticated sensing techniques and robust control algorithms (Malone *et al.*, 1994).

The Automation Systems team of Industrial Research Limited (IRL) specialise in the development of robotic devices for the handling and processing of natural products. In particular, IRL focuses on the development of flexible automation solutions for the meat-processing industry (Templer *et al.*, 2000). This work has culminated in the production of the world's first sheep-meat processing robot, which performs the Y-cut operation on sheep carcasses (Taylor & Templer, 1997). All previous Y-cut automation work by IRL has taken place within New Zealand (Templer *et al.*, 1999). To demonstrate the international potential of the Y-cutting technology, IRL have developed a prototype commercial system for the Australian meat-processing market.

1.2 COMMERCIAL OPPORTUNITY

The sheep-meat industry is an important contributor to both the New Zealand and Australian economies, employing thousands of people in both countries. New Zealand produced 562,000 tonnes of sheep-meat in 2002, with export earnings of NZ$2.3 billion

(New Zealand Meat Board, 2002), while Australia produced 746,000 tonnes of sheep-meat worth A$2.1 billion in the same year (Meat and Livestock Australia, 2002). The New Zealand sheep-meat industry is dominated by the production of export-quality lamb, with a focus also on high-quality pelts. The Australian sheep-meat industry is far more diverse. While there are abattoirs that produce high-quality lamb, many plants focus more on the production of lower-grade mutton for export to Middle-Eastern markets. Australian sheep are generally subject to harsher climatic conditions than New Zealand livestock, with the environment tending to be hotter, dryer and less arable. These environmental differences mean that Australian sheep can arrive at the abattoir in poor condition with dust and sand embedded in their pelts. There are also a large number of Merino animals in Australia, with these animals having large folds of skin around the neck region. These differences necessitate the demonstration of the Y-cutting system in an Australian abattoir to determine if the automated process developed for New Zealand conditions is appropriate.

Despite the size of the meat industry in both New Zealand and Australia, there has been only limited development of modern meat-processing automation solutions (Templer *et al.*, 2000). If the industry is to remain internationally competitive, then the uptake of meat-processing automation will need to increase in the coming years (Templer *et al.*, 1998). The economics of the meat-processing industry demand extremely robust and reliable technology, with commercial clients typically requiring a success rate of greater than 98%. This performance requirement means that it is important to detect, diagnose and correct faults as quickly as possible so that the performance of the system is not adversely affected.

At present, the Y-cutting and other robotic systems developed by IRL lack the ability to identify and quantify inaccuracies in their own performance and cannot independently modify the behaviour that produced these errors. Therefore, the implementation of an adaptive fault detection and correction scheme would be an important enhancement to the automated Y-cutting system and other IRL robotic devices.

1.3 THESIS OBJECTIVES

This thesis intends to meet the following objectives:

- Define and characterise faults that occur in the Y-cut process.
- Develop a system for detecting the incidence of process faults automatically and reliably, with a fault classification rate of at least 95%.
- Improve the existing manual tuning process of the Y-cutting system by reducing the time taken to achieve a cutting success rate of 98%.
- Provide generic fault detection and tuning solutions that can be applied to other IRL robotic devices.

1.4 THESIS PROGRESSION AND EVOLUTION

The data used for this thesis were obtained from two different Y-cutting installations. The first installation used an older Y-cutting system that was temporarily installed at Auckland Meat Processors (AMP) in Auckland, New Zealand in October 2002. This installation was primarily used to test a new design of Y-cut tool. It also enabled the capture of motor current data from the Y-cutting robot. These data were used for the initial signal characterisation and fault detection development work detailed in Chapter 4. Because the performance of the Y-cutting system was sub-optimal, there was a propensity for the cut to terminate prematurely. This meant there was a high incidence of two different process faults: insertion failures and end-point failures (refer Section 4.1). Therefore, the initial fault detection strategies were designed to classify both of these fault conditions.

The second Y-cut installation occurred at Southern Meats in Goulburn, Australia in August 2003 with a prototype of the commercial Y-cutting system. The main purpose of this trial installation was to demonstrate the system in Australian conditions and on Australian livestock. This was also the first installation to use a new KUKA industrial robot. The Y-cut tool was modified slightly based on observations from the AMP trial. The improvements to the tool combined with the additional degrees-of-freedom

provided by the KUKA robot resulted in a vast improvement in the performance of the Y-cutting system, with insertion failures being the only observed process fault. This meant that the detection of end-point failures was not possible with the fault detection module developed in Chapter 5. However, it was recognised that a generic solution should also detect other fault conditions, with Chapter 7 extending the fault detection system to allow for the classification of end-point failures and load variations.

1.5 CONTRIBUTIONS TO PRACTICAL WORK

The author became involved with the development of the Y-cutting system at IRL in June 2002. His initial work involved re-commissioning the older Y-cutting robot, which had been removed from a New Zealand plant several years previous. He also conducted all of the process tuning and monitoring while the system was installed at AMP. It was during this time that he captured the dataset used in Chapter 4.

The author was also involved extensively with the development of the KUKA Y-cutting system during 2003. He was primarily responsible for the development of sensor measurement programs, a software-synchronised conveyor tracking methodology and the interfacing of associated sensors. He was also involved in the development of the main robot motion program. Between August and October of 2003, he was jointly responsible for the installation, commissioning and tuning of the KUKA Y-cutting system at Southern Meats in Goulburn, Australia. The datasets described in Section 5.3 were obtained by a colleague of the author in November 2003, using a methodology defined by the author. Following the development of a fault detection module by the author, he returned to Goulburn in January 2004 to implement and test the module. This trip also allowed the author to conduct a series of cutting trials, producing the data that are described in Section 6.1.1. All remaining analysis, testing and algorithm development is the author's own work.

1.6 THESIS STRUCTURE

Chapter 2 gives an overview of adaptive robotic systems and their application to natural product handling and processing tasks. Possible fault detection methods are identified

and three neural network architectures are detailed. The chapter concludes with an examination of potential optimisation techniques and their application to the Y-cutting system.

Chapter 3 describes the development of an automated Y-cutting system for use on sheep carcasses. The manual Y-cutting process is defined, and the key drivers behind the development of an automated system are discussed. Previous design work is detailed, progressing through to a description of the prototype commercial Y-cutting system. This includes an overview of the sensing system, software and path-planning algorithm.

Chapter 4 details the initial testing of different fault detection methods using an older version of the Y-cutting system. Various process faults are identified and are related to load data obtained from the robot. Several fault detection strategies are developed and tested, and a hybrid strategy is proposed.

The development of a fault detection system for the prototype commercial Y-cutting system is given in Chapter 5. The software used to implement the fault detection module is described. The module is tested during normal operation of the Y-cutting system, and a modification is made to allow the module to adapt to changes in the process.

Chapter 6 describes the development of a path optimisation and fault correction algorithm. The chapter begins with an analysis of key cut-path parameters and observations from gathered data. The optimisation algorithm is detailed and results are presented from simulations with a reduced set of parameters. The algorithm is re-tested using an expanded set of path parameters and its overall performance is evaluated.

Chapter 7 proposes the expansion of the fault detection and path optimisation system to allow for more generic implementation. The fault detection module is extended to handle a greater number of faults and to detect variations in the cutting load. The extension of the path optimisation algorithm is also discussed, along with issues relating to the generic application of both systems.

The thesis concludes with an evaluation of the developed fault detection and path optimisation systems, and outlines possible improvements and areas for future research.

2. BACKGROUND

2.1 ADAPTIVE ROBOTIC SYSTEMS

There has been a shift towards greater mechanisation and automation of industrial production facilities in recent decades (Kopacek, 1999). The development of industrial robots has been integral to the rapid up-take of this automation technology. Industrial robots have improved productivity and plant flexibility, and have reduced the use of human labour in physically demanding or dangerous tasks. These robots were initially very simple, but they have increased in complexity with the advent of faster computers and microprocessors. Improvements to programming languages and software environments have also made it easier to develop and implement robotic systems. These advances have resulted in the use of robots for increasingly complex and demanding tasks. As task complexity rises, the ability to supply robotic devices with some form of artificial intelligence is proving increasingly advantageous (de Silva, 1995).

A large amount of research has focussed on the creation of "intelligent" robot learning systems (Brooks & Mataric, 1993; Murphy, 2000). Previous research has included the development and application of artificial neural networks (Kartalopoulos, 1996), fuzzy-logic control (de Silva, 1995) and evolutionary learning algorithms (Floreano & Urzelai, 2000). These techniques have been incorporated into controllers for robotic arms and manipulators (Cheah & Wang, 1998), and have demonstrated superior performance to traditional feedback control systems (Gupta & Sinha, 2000). The use of machine intelligence has allowed the fusion of motion controllers with sophisticated sensing systems and provides advanced robot functionality. Examples of this fusion include vision-based motion tracking systems (Chen & Hseuh, 2001) and real-time obstacle avoidance (Yang & Meng, 2000; Mbede *et al.*, 2001).

2.1.1 Natural Product Handling and Processing

The processing and handling of natural products, such as meat, timber and fruit, are particularly challenging tasks to automate. The non-uniformity and variability of these products demand sophisticated sensing systems, flexible tool design and robust robotic control procedures. These requirements are necessary to ensure that the given task can be completed autonomously with a high degree of efficiency and repeatability. It is also important that the task be completed as quickly as possible to make the automation a feasible alternative to manual processing.

Various mobile robotic systems have been developed to automate agricultural planting and harvesting tasks (Torii, 2000; Kassler, 2001). Much of this research has focussed on improving the reliability of sensing systems and navigation algorithms, rather than on the handling or processing of agricultural or horticultural products. An example of an automated handling task is the robotic harvesting of cucumbers (Van Henten *et al.*, 2003). The robot uses a stereo vision system to locate the hanging cucumbers on the plant. A manipulator grips, cuts and packs the fruit. The robotic system achieved a success rate of 74.4% and an average cycle time of 124 seconds per harvested cucumber. However, neither this success rate nor processing speed is considered adequate for commercial implementation.

Other natural product handling and processing applications include a mobile robotic device for harvesting asparagus (Spooner & Rodrigo, 1998), and a robotic wood carving system (Andersson & Johansson, 2001). The wood carving system uses a force/torque sensor to control the cutting forces of the milling tool, and allows the robot to automatically deal with woods of different species and relative humidity.

2.1.2 Robotic Systems in Meat Processing

The Automation Systems team of Industrial Research Limited specialise in the development of robotic systems for meat processing tasks. In addition to the development of an automated Y-cutting system (refer Section 3.2), IRL has also produced a variety of other meat processing robots. These systems include a robot used

for brisket sawing of beef carcasses (Templer *et al.*, 1998), a beef belly-rip robot and a robot manipulated waterjet cutter for the boning of fish fillets (Malone *et al.*, 1994). A robotic contour following system has also been developed for the steam vacuuming of beef carcasses (Hildreth *et al.*, 2003).

Other robotic meat-processing systems have been developed internationally, although little of this work has been published due to the commercially sensitive nature of the research. Any published material tends to give only an overview of the techniques and technologies that have been used, rather than providing substantive details. From the point-of-view of a researcher, this lack of detail makes it hard to establish what the current state-of-the-art is, what problems have been encountered and what solutions have been developed or proposed. Worldwide, there are only a small number of research organisations and companies involved in developing robotic meat-processing systems, which also limits the amount of available literature. Apart from IRL, Dunedin-based Scott Automation is the only other New Zealand company actively developing robotic systems for the meat industry. Scott Automation has developed a system using a KUKA robot to remove the aitchbone from lamb carcasses (Country-Wide Publications, 2004). This system uses a clamping mechanism to hold and tension the carcass while the robot removes the bone using an actuated knife. It is not known what sensing is used by this system.

Outside of New Zealand, Food Science Australia (FSA) has developed a robotic system for the scribing of beef carcasses (Li & Hinsch, 2003). The scribing operation consists of a set of carcass cuts and bone scribes that eventually produce speciality cuts of meat. A machine vision system using a laser scanner determines the cut-path. FSA are the only Australian company known to be currently developing robotic meat-processing systems. However, during the 1980s and 1990s the Meat Research Corporation of Australia spent many millions of dollars developing the FUTUTECH abattoir and semi-automated beef dressing system (Australian Government, 1994). FUTUTECH was intended to demonstrate the ultimate potential of automation technology by automating the majority of tasks involved in dressing beef carcasses (White, 1994). However, there were many technical failings and the abattoir was never able to operate at commercially viable speeds. The failure of this project has contributed to a decline in meat-processing research and development within Australia, as well as reinforcing industry perceptions

of automation technology being unreliable and unsuitable for real-world meat-plant environments.

Elsewhere in the world, a robotic primal cutting system for pork has been installed in a plant in Norway (Purnell & Brown, 2003). The robot uses a rotary saw blade to cut sides of pork into smaller primal pieces, with a separate machine vision system generating the cut-path. This robotic system has been shown to produce more anatomically accurate cuts than the manual human process, with 97% of the cuts being within ±5 mm of the optimal anatomical cut. Another robotic pork system has been developed in Canada (Tessier, 2000). This system removes the side ribs from pork primal cuts, with a vision system used to determine the location of the ribs. Although both of these robotic systems have been demonstrated in commercial abattoirs, it is unclear whether either has reached sufficient maturity to become commercially available products.

While all of these meat-processing systems use sophisticated sensing techniques to deal with the highly variable products, none of them appear to employ fault detection strategies to determine if the process has been successfully completed.

2.2 FAULT DETECTION

A definition of a fault relating to a robotic system is "a non-permitted deviation of a characteristic property of the process itself, the actuators, the sensors and controllers" (Isermann, 1993). It is important that faults are detected and diagnosed as quickly as possible so that the performance of the robot is not adversely affected. Fault detection can also ensure that the safety of the system is not compromised.

The simplest form of fault detection involves limit checking to determine if the process is operating outside of normal tolerances. This method works well if the monitored process is operating in a steady state, but is of limited value if the operating set point continually changes (Isermann & Ayoubi, 1996). If an efficient closed-loop system controls the process then it may be impossible to detect the influence of a fault from the output signals, provided the process inputs remain within the normal operating range. A

fault may only be detectable if it occurs suddenly and with a large amplitude or if it results in a long-term increase or decrease in the process inputs. More advanced fault detection methods are needed to identify smaller faults or for the monitoring of processes that undergo rapid state changes. There are three broad classifications of advanced fault detection: analytical model-based, knowledge-based, and pattern recognition-based methods (Genovesi *et al.*, 2000).

2.2.1 Analytical Model-Based Methods

Analytical model-based fault detection relies on the existence of an accurate mathematical description of the robotic system and process (Freyermuth, 1991). If a model exists, then techniques from modern control theory allow the monitoring of the system for the occurrence of fault conditions. These techniques include the use of observers, state estimators, parameter estimators and parity equations (Franklin *et al.*, 1994). The resulting parameters, states and residuals determine if the process is operating in a fault-free or faulty manner.

Many robotic processes are difficult or impossible to model because of the non-linear and non-stationary interactions between the robot and the environment (Rezayat, 1999). Approximating or simplifying the system can overcome this difficulty, although usually to the detriment of the fault detection (Patton, 1994).

2.2.2 Knowledge-Based Methods

Knowledge-based fault detection uses qualitative models to relate the incidence of faults to specific symptoms or data measurements. These symptoms can result from either analytic or heuristic knowledge of the process (Isermann, 1998).

Analytic symptoms include residuals and state-variables generated from mathematical models, as used in the model-based fault detection methods (Angeli & Chatzinikolaou, 1999). If a mathematical model of the process does not exist, then signal-processing techniques can generate characteristic values from sensor measurements. The simplest technique is to use limit value checking to determine if the measurement signal has

exceeded specified tolerances. Other techniques include the use of correlation functions, spectral analysis, and autoregressive moving average models (Isermann & Ayoubi, 1996). The characteristic values allow for the extraction of features pertaining to the faulty and non-faulty process.

It is often cheaper and easier to have a person act as the main sensing system rather than using a variety of complicated sensors. Qualitative information obtained from human observation of the process produce heuristic symptoms. These observations can relate to parameters such as colour, smell, vibration, and wear and tear. The qualitative information can also take the form of statistical data obtained through human experience, such as the mean time between failure and the probability of observing particular faults. Heuristic symptoms can be represented by linguistic variables (e.g. 'small', 'medium', 'large'), or as fuzzy membership sets (Berkan & Trubatch, 1997). Both of these representations lend themselves to the application of an expert system (Visinsky *et al.*, 1994) or fuzzy logic control for fault detection (de Silva, 1995; Lu *et al.*, 1998).

Knowledge-based fault detection methods are useful when there is sufficient process information and expert knowledge, and when the dynamics of the robotic system are not easily defined in terms of a mathematical model (Genovesi *et al.*, 2000). However, these methods tend to be very process specific. They also suffer from limited resolution due to the often ambiguous nature of qualitative reasoning (Gomez *et al.*, 1998). If the knowledge base is not sufficiently detailed then faults may go undetected. If the application requires a large number of rules for reliable fault detection, then the system can become very complex. This complexity can limit the usefulness of the fault detection since real-time operation might not be possible.

2.2.3 Pattern Recognition-Based Methods

Pattern recognition-based methods use classification techniques to relate input data to output data, via either statistical or non-statistical techniques. Although the two techniques are fundamentally equivalent (Jain *et al.*, 2000), traditional statistical classifiers are not adaptive and make strong assumptions about the nature of underlying

distributions (Lippmann, 1988). These limitations make statistical techniques unsuitable for real-time monitoring of non-linear processes. The most common form of non-statistical classifier is the artificial neural network, which attempts to mimic the functioning of biological nervous systems (Kartalopoulos, 1996). Neural networks are robust to noise, and can handle incomplete data. They are very effective at modelling non-linear relationships and offer good real-time fault detection performance (Terra & Tinós, 2001). Neural networks can perform better than traditional statistical classifiers when the monitored process is non-linear and the distributions are strongly non-Gaussian (Lippmann, 1988). Neural networks have been used to detect faults in various mechanical and robotic systems (Vemuri *et al.*, 1998; Oaufi *et al.*, 2001; Campa *et al.*, 2002; Wang & Too, 2002).

2.3 ARTIFICIAL NEURAL NETWORKS

The development of artificial neural networks began with the work of McCulloch and Pitts (1943), who proposed a hardware model that was inspired by the operation of a biological neuron. Hebb (1949) developed the first neural learning procedures with an adaptation mechanism that strengthened pathways between frequently used neurons. Rosenblatt (1959) integrated learning into an artificial neuron model called the perceptron, which classified geometric patterns from optical inputs. Widrow and Hoff (1960) produced simplified neuron models with the ADALINE (ADAptive LInear NEuron) and MADALINE (Multiple ADALINE) networks.

Research interest waned during the 1970's, as a vigorous debate took place concerning the classification ability of the perceptron and other developed models (Minsky & Pappert, 1969). This debate focussed on the fact that the perceptron could only linearly separate a parameter space, resulting in an inability to classify supposedly simple patterns correctly, such as the output from an exclusive-OR logic device. However, there was a resurgence of interest in the field following the work of Hopfield (1982), who re-demonstrated the potential application of neural networks. In particular, the development of multilayer neural networks solved the problem of linear separability encountered with the earlier single-layer perceptron. Enthusiasm has grown in subsequent years, with numerous research groups now working exclusively on the

design and implementation of neural networks. While early neuron models were constructed from electronic components, the advent of faster digital computers means that networks can now be built entirely from software. This development has given rise to a number of real-world applications in areas such as financial-modelling, communications and image-processing (Kartolopoulos, 1996).

There are two classes of neural network paradigm: supervised and unsupervised. The paradigms are characterised by the different methods used to train the networks. The training of supervised networks works by applying a known input and observing the output. If the observed output differs from the desired target output, then a learning rule adjusts the network to minimise this error. Unsupervised networks do not require a target output. Instead, inputs are organised into categories or clusters depending on how they stimulate the network. The unsupervised learning rule either uses a fixed number of clusters or creates a new cluster if the input results in a sufficiently different excitation from the other clusters.

The flow of information through a network distinguishes different types of neural networks. Feed-forward networks have connections between neurons in one direction only, so that information flows from the input to the output. Feedback networks allow information to flow in both directions, with additional connections from the output back to the input. These recurrent networks perform particularly well in control applications (Ziemke, 2000).

The most commonly used neural networks for pattern recognition and classification tasks are feed-forward networks (Jain *et al.*, 2000). This family includes the multilayer perceptron (MLP), radial basis function (RBF) network and learning vector quantisation (LVQ) network.

2.3.1 Multilayer Perceptron

The multilayer perceptron is a variation of the original perceptron developed by Rosenblatt (1959). Figure 2.1 shows the general MLP structure. The neurons are

arranged in a hierarchical structure, with one or more hidden layers between the input and output nodes. The connections between layers have associated modifiable weights.

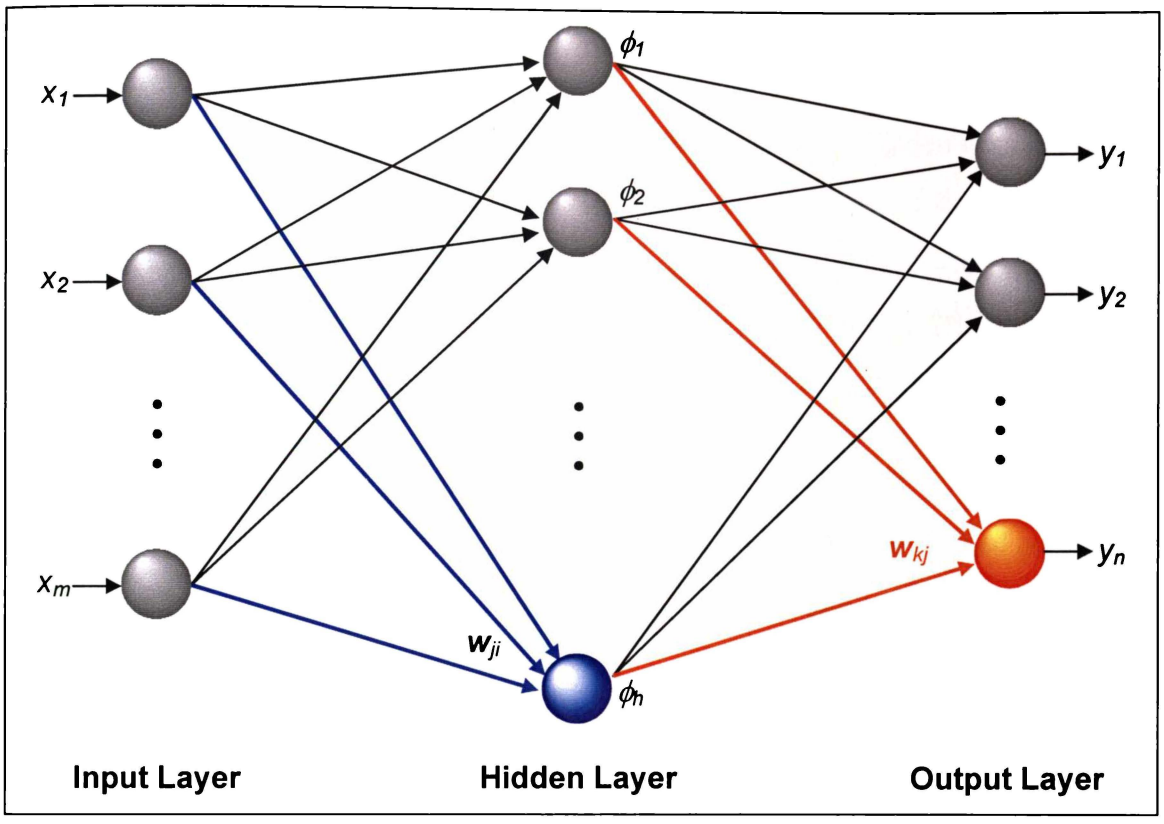

FIGURE 2.1 The structure of the MLP neural network.

The m input layer neurons are responsible for the distribution of the input vector **x** to all of the h hidden layer neurons, with $\boldsymbol{x} = (x_1, x_2, \ldots, x_m)$. The weighted sum of the inputs activates each hidden neuron, with the activation of neuron j being given by:

$$net_j = w_{j0} + \sum_{i=1}^{m} x_i w_{ji} \tag{1}$$

where j indexes the hidden neurons, w_{ji} is the weight connecting input neuron i with hidden neuron j, and w_{j0} is a bias or threshold weight. If a unit input $x_0 = 1$ is added to the input vector to represent this threshold input, then the activation of the hidden neuron is:

$$net_j = \sum_{i=0}^{m} x_i w_{ji} \tag{2}$$

The output ϕ_j of the hidden neuron is a non-linear function of the activation:

$$\phi_j = f_1(net_j) \tag{3}$$

The most commonly used nonlinearities are hard-limiting and sigmoid activation functions (refer Figure 2.2). The sigmoid functions are preferred, because they are continuous and easily differentiable. These properties are important for many neural network training methods.

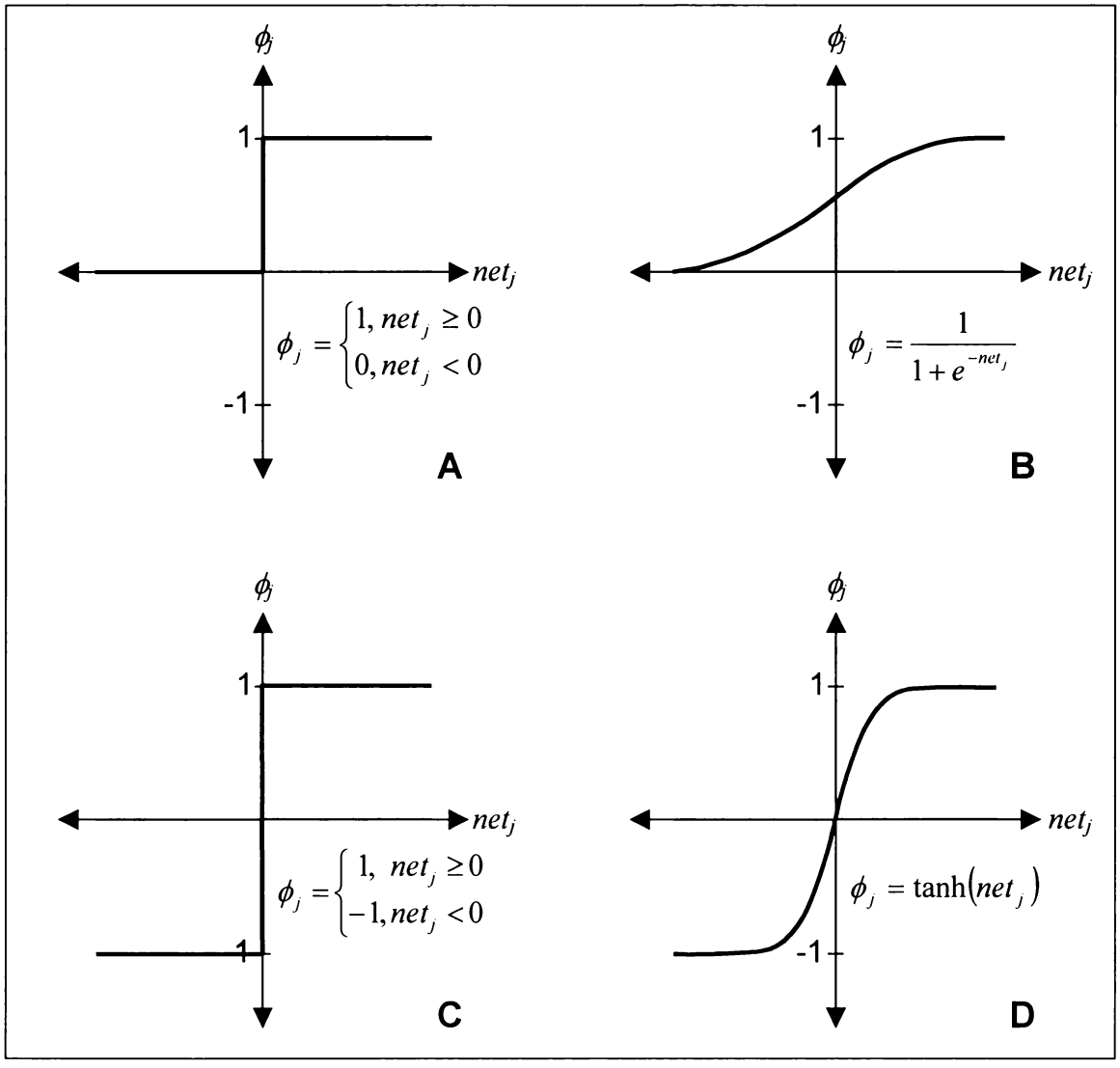

FIGURE 2.2 **Common non-linear activation functions: a) hard-limiter, b) logarithmic sigmoid, c) symmetrical hard-limiter, and d) hyperbolic tangent sigmoid.**

The n output neurons are activated in the same manner as the hidden neurons:

$$net_k = w_{k0} + \sum_{j=1}^{h} \phi_j w_{kj} = \sum_{j=0}^{h} \phi_j w_{kj} \tag{4}$$

where k indexes the output neurons, w_{kj} is the weight connecting hidden neuron j with output neuron k, and w_{k0} is a threshold weight (applied to a unit input of $\phi_0 = 1$).

The final output of the network is again a nonlinear function of the activation:

$$y_k = f_2(net_k) \tag{5}$$

Equations 2, 3, 4, and 5 lead to the relationship between the network input and output:

$$y_k = f_2\left(\sum_{j=0}^{k} w_{kj} f_1\left(\sum_{i=0}^{m} w_{ji} x_i\right)\right) \tag{6}$$

The number of input and output neurons is equivalent to the size of the input and output data vectors. The number of neurons in the hidden layer is usually determined via a trial-and-error process. If too few neurons are used, then classification will be poor. If too many neurons are used, then generalisation will be poor.

The number of hidden layers depends on the complexity of the classification problem. An MLP with no hidden layers can only form a linear decision boundary between regions R_1 and R_2 (refer Figure 2.3). An MLP with one hidden layer can form any arbitrary decision boundary if there are sufficient neurons within the layer, and can therefore approximate any continuous function (Duda *et al.*, 2000). This approximation ability means there is often no advantage in using more than one hidden layer.

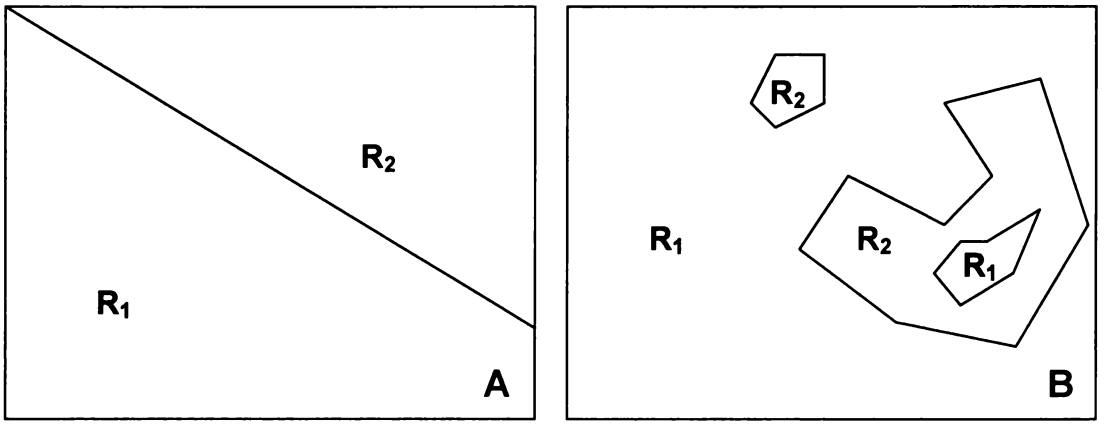

FIGURE 2.3 MLP decision boundaries: a) no hidden layer, b) one hidden layer.

2.3.2 Back-Propagation Algorithm

The MLP is trained with input-output data pairs to optimise the classification performance of the network. This training is analogous to the learning process that takes place in biological neural networks. Werbos (1974) and Rumelhart, Hinton and Williams (1986) are independently credited with the development of the back-propagation algorithm, which is the most commonly used method for training feed-forward neural networks. Back-propagation uses a gradient-descent method for modifying weights so that the training error is minimised. The training error is defined as the sum of the squared difference between the desired target output t_k and the actual output y_k:

$$E = \sum_{k=1}^{n} \tfrac{1}{2}(t_k - y_k)^2 \tag{7}$$

The back-propagation algorithm is an iterative process that adjusts the inter-neuron weights w according to the learning rule:

$$w(a+1) = w(a) + \Delta w(a) \tag{8}$$

where a is the training iteration. The weights shift in a direction that will reduce the output or training error E:

$$\Delta w = -\eta \frac{\partial E}{\partial w} \tag{9}$$

where η is the learning rate. The learning rate can be either a constant or a monotonically decreasing function of the training iteration number. If the learning rate decreases as training progresses, then the initial rate has to be large enough to allow the error to converge within an acceptable number of iterations. If the initial learning rate is too large, then the error can overshoot or even diverge. Other methods have been proposed to modify the learning rate based on the performance of the network, and are shown to improve the speed and stability of convergence (Magoulas *et al.*, 1999).

The weights in both the hidden and output layers require modification. The dependence of the training error on the output weights w_{kj} is determined using the chain rule for differentiation:

$$\frac{\partial E}{\partial w_{kj}} = \frac{\partial E}{\partial net_k} \frac{\partial net_k}{\partial w_{kj}} = \frac{\partial E}{\partial net_k} \phi_j \tag{10}$$

The partial derivative term on the right-hand side of Equation 10 describes the effect that the output neuron activation has on the training error. This sensitivity effect can be defined as:

$$\delta_k = -\frac{\partial E}{\partial net_k} \tag{11}$$

Using the chain rule again, δ_k becomes:

$$\delta_k = -\frac{\partial E}{\partial y_k} \frac{\partial y_k}{\partial net_k} = (t_k - y_k) f'(net_k) \tag{12}$$

From Equation 12 it is apparent that the neuron activation function $f(net_k)$ must be differentiable. This is a key reason for the popularity of the sigmoid transfer functions described in Section 2.3.1. From Equation 8, the update rule for the output weights becomes:

$$\Delta w_{kj} = \eta \phi_j \delta_k = \eta \phi_j (t_k - y_k) f'(net_k) \tag{13}$$

The dependence of the training error on the hidden weights w_{ji} is determined in a similar manner to Equation 10:

$$\frac{\partial E}{\partial w_{ji}} = \frac{\partial E}{\partial net_j} \frac{\partial net_j}{\partial w_{ji}} = \frac{\partial E}{\partial net_j} x_i \tag{14}$$

The sensitivity δ_j of the hidden layer neurons can be defined as:

$$\delta_j = -\frac{\partial E}{\partial net_j} = -\frac{\partial E}{\partial \phi_j} \frac{\partial \phi_j}{\partial net_j} \tag{15}$$

If the training error is defined in terms of the hidden neuron output ϕ_j:

$$E = \sum_{k=1}^{n} \tfrac{1}{2} \left(t_k - f\left(\sum_{j=1}^{h} \phi_j w_{kj} \right) \right)^2 \tag{16}$$

then the partial derivative with respect to ϕ_j is:

$$\frac{\partial E}{\partial \phi_j} = -\sum_{k=1}^{n} (t_k - y_k) f'(net_k) w_{kj} \tag{17}$$

Therefore, the sensitivity δ_j becomes:

$$\delta_j = f'(net_j) \sum_{k=1}^{n} (t_k - y_k) f'(net_k) w_{kj} = f'(net_j) \sum_{k=1}^{n} \delta_k w_{kj} \tag{18}$$

Equation 18 implies that the sensitivity of a single hidden neuron is dependent on the sum of the sensitivities of the output neurons. This result is central to the success of the back-propagation algorithm. It means that the training error observed at the output can

propagate back through the network to modify the hidden neuron weights. The update rule for the hidden neuron weights becomes:

$$\Delta w_{ji} = \eta x_i \delta_j = \eta x_i f'(net_j) \sum_{k=1}^{n} (t_k - y_k) f'(net_k) w_{kj} \tag{19}$$

Equations 13 and 19 form the basis of the back-propagation algorithm. The algorithm modifies the neuron weights so that the performance of the network converges to a local optima (although not necessarily the global optimum). Note that if the output weights w_{kj} are all zero then Δw_{ji} will also be zero and the hidden neuron weights will not change. The initialisation of the weights with random values at the commencement of the training prevents this from happening.

2.3.3 MLP Training Methods

Several MLP training methods use the back-propagation algorithm. Stochastic training is the most commonly used method, and involves the random selection of input-target data pairs. The input vector determines the output of the network, and Equations 13 and 19 give the weight changes for the hidden and output neurons. The process of random data selection and weight modification continues until the training error E is smaller than some pre-defined threshold.

Batch training involves the sequential presentation of all the training data. The calculated weight changes for each data pair are added together, and are only used to update the weights once all of the training data has been presented. Batch training is typically slower than stochastic training, particularly if there is a degree of redundancy within the training data (Duda *et al.*, 2000).

Online training presents a data pair to the network once and only once. The data pair is immediately discarded once the neuron weights are updated, and the next data pair is acquired. This training method is useful if there is a large amount of training data, or if there is a limited amount of memory storage available.

The speed with which an MLP network is trained is usually defined in terms of epochs, where one epoch equates to the single presentation of all of the training data. This measure is used for both stochastic and batch training. For online training, the number of presented data pairs is a more appropriate metric.

2.3.4 Radial Basis Function Network

The radial basis function (RBF) network is another popular feed-forward neural network design (Powell, 1987). The RBF network structure is essentially the same as the MLP structure (refer Figure 2.1), but uses different neuron activation functions in the hidden and output layers.

The hidden layer neurons use a Gaussian activation function. The output of hidden neuron j is given by:

$$\phi_j = \exp\left(-\sum_{i=1}^{m} \frac{(x_i - w_{ji})^2}{2\sigma_j^2} \right) \tag{20}$$

where x_i is the signal from input neuron i, w_{ji} is the weight connecting input neuron i with hidden neuron j, and σ_j^2 is the variance of the Gaussian. Each neuron in the hidden layer creates a localised receptive field in the m-dimensional input space. The neuron only produces a non-zero response if the input falls within this field. This receptive field is centred about the neuron weights w_{ji}, and the parameter σ_j specifies the width or spread of the field (refer Figure 2.4). If the input is a radial distance σ_j from the neuron weights, then the output of the neuron is 0.6065. The radial symmetry of the Gaussian is the reason for the naming of the radial basis function network.

The output layer neurons use a linear transfer function to form a linear combination of the radial basis neurons:

$$y_k = w_{k0} + \sum_{j=1}^{h} w_{kj}\phi_j = \sum_{j=0}^{h} w_{kj}\phi_j \tag{21}$$

where w_{k0} is a threshold weight, and w_{kj} is the weight connecting output neuron k with hidden neuron j. The combination of the hidden radial basis neurons and the linear output neurons means that the RBF network performs a nonlinear transformation from the m-dimensional input space to the n-dimensional output space.

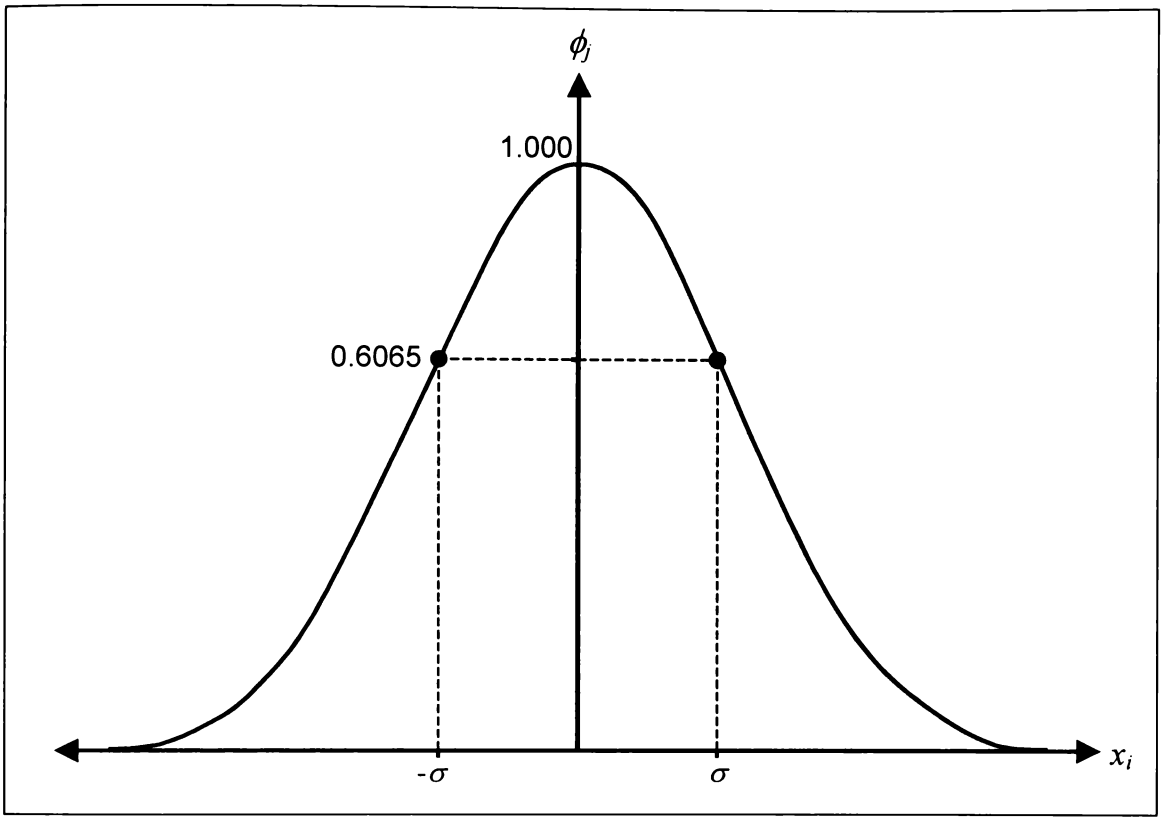

FIGURE 2.4 Gaussian radial basis function.

2.3.5 RBF Network Training

Training of RBF networks takes place in two stages. The first stage determines the radial basis function weights w_{ji} and widths σ_j. The simplest method for specifying the RBF weights is for the network to have the same number of hidden neurons and training vectors, with the weights for each neuron set equal to a different training vector. This results in a single neuron being completely activated ($\phi_j = 1$) by each training vector. A network of this form will have zero error when tested with the training data. However, the network can become unacceptably large if there are many training vectors. A more efficient method uses an orthogonal least squares algorithm (Chen *et al.*, 1991), with single neurons added to the network until the training error falls below a certain level.

The weights for each added neuron are set equal to the training vector that produces the biggest decrease in the training error. This method can produce a network with fewer neurons, although it does not necessarily result in the smallest possible network for a given training error (Sherstinsky & Picard, 1996). A third method uses a clustering algorithm to group the training vectors, and then places a neuron at the centre of each cluster (Moody & Darken, 1989). However, there is no attempt to minimise the training error, meaning this method may be unsuitable for modelling nonlinear systems.

The second stage of the network training involves the specification of the output weights w_{kj}. The linear transfer function used by the output neurons allows the direct calculation of the weights. From Equation 21, the output of the RBF network can be rewritten in matrix form as:

$$\boldsymbol{y} = \boldsymbol{w}\boldsymbol{\phi} \tag{22}$$

where:

$$\boldsymbol{y} = [y_1 \; y_2 \cdots y_n]^T \tag{23}$$

$$\boldsymbol{w} = \begin{bmatrix} w_{10} & w_{11} & \cdots & w_{1h} \\ w_{20} & w_{21} & \cdots & w_{2h} \\ \vdots & \vdots & \ddots & \vdots \\ w_{n0} & w_{n1} & \cdots & w_{nh} \end{bmatrix} \tag{24}$$

$$\boldsymbol{\phi} = [1 \; \phi_1 \; \phi_2 \cdots \phi_h]^T \tag{25}$$

If there are p pairs of training data $(\boldsymbol{x},\boldsymbol{t})$, then the p radial basis activations and target output vectors form a rectangular set of linear equations:

$$\boldsymbol{T} = \boldsymbol{w}\boldsymbol{\Phi} \tag{26}$$

where:

$$\boldsymbol{T} = [\boldsymbol{t}_1 \; \boldsymbol{t}_2 \cdots \boldsymbol{t}_p]^T \tag{27}$$

$$\boldsymbol{\Phi} = \left[\boldsymbol{\phi}_1\ \boldsymbol{\phi}_2 \cdots \boldsymbol{\phi}_p\right]^T \tag{28}$$

Equation 26 can be solved directly for the network output weights:

$$\boldsymbol{w} = \boldsymbol{\Phi}^{\dagger}\boldsymbol{T} \tag{29}$$

where $\boldsymbol{\Phi}^{\dagger}$ is the pseudo-inverse of $\boldsymbol{\Phi}$ (Duda *et al.*, 2000):

$$\boldsymbol{\Phi}^{\dagger} = \left(\boldsymbol{\Phi}^T\boldsymbol{\Phi}\right)^{-1}\boldsymbol{\Phi}^T \tag{30}$$

Although the pseudo-inverse calculation is relatively straightforward, it does require the inversion of a potentially large matrix. This calculation can be computation intensive if there are a moderate number of hidden neurons.

RBF networks tend to have more neurons than an MLP network trained with the same data. The sigmoid activation functions used for the MLP network can produce an output over a large part of the input space, whereas the radial basis function is sensitive to a much smaller region. Therefore, an RBF network generally has to have more neurons than an MLP network to respond to the same input space.

The accuracy of an RBF network is very high when the input data is similar to the training data. The ability of the network to generalise beyond the training data can be poor, since the activation of the radial basis neurons diminishes away from the centre of the receptive field (Koppen-Seliger & Frank, 1999).

2.3.6 Learning Vector Quantisation Network

The third type of neural network considered is the learning vector quantisation (LVQ) network (Kohonen, 1997). The LVQ network categorises input data into different classes, with training data consisting of input vectors and target classifications. Figure 2.5 gives the LVQ network structure.

The neurons in the hidden layer use a competitive activation function that employs a "winner-takes-all" strategy (Kartalopoulos, 1996). Each neuron calculates the Euclidean distance between the input $\boldsymbol{x}$ and the neuron weights:

$$net_j = \sqrt{\sum_{i=1}^{m} (x_i - w_{ji})^2} \tag{31}$$

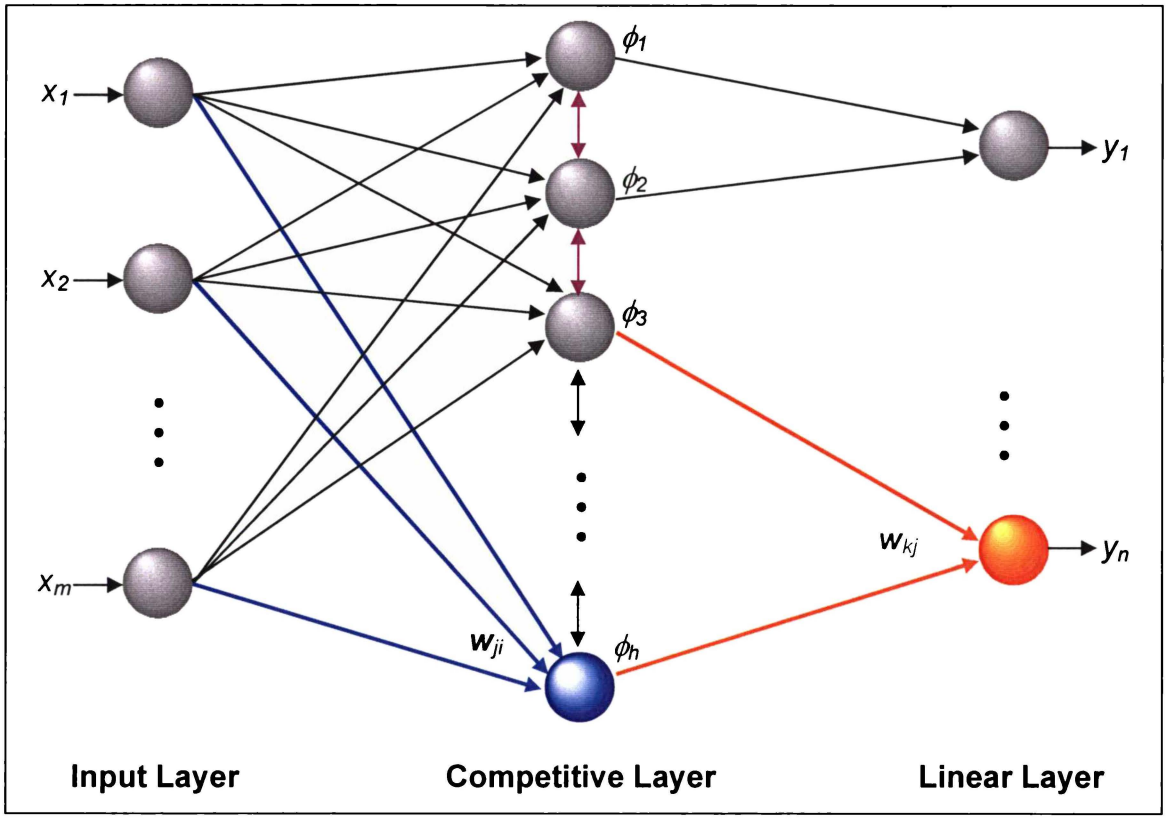

FIGURE 2.5 The structure of the LVQ network.

where w_{ji} is the weight connecting input neuron i with hidden neuron j. A competitive function determines the output of all of the hidden neurons. The neuron with the smallest Euclidean distance produces an output ϕ_j of one, and all of the other neurons output a zero:

$$\phi_j = \begin{cases} 0, & net_j \neq \max_{j=1\ldots h}(net_j) \\ 1, & net_j = \max_{j=1\ldots h}(net_j) \end{cases} \tag{32}$$

This competitive function means that the closest neuron to the input vector is the only one activated. The output neurons use a linear transfer function:

$$y_k = \sum_{j=1}^{h} w_{kj}\phi_j \tag{33}$$

with the weight w_{kj} being equal to one if output neuron k is connected to hidden neuron j, otherwise the weight is equal to zero. Each hidden neuron connects to only one output neuron.

The simplicity and speed of the LVQ network means that it is particularly useful when there is a large amount of training data. The user specifies the numbers of hidden and output neurons in the LVQ network. The number of output neurons is equivalent to the number of target classes as specified in the training data. The number of hidden neurons gives the number of subclasses used to categorise the input data.

A limitation of the LVQ network is that it can over-fit the training data if the network is too complex for the given problem (Bermejo & Cabestany, 2004). This over-fitting can occur if there are more hidden neurons than necessary to define the decision boundary between neighbouring classes. This can limit the ability of the network to generalise beyond the training data.

2.3.7 LVQ Network Training

LVQ network training uses both supervised and unsupervised methods. Unsupervised clustering techniques, such as k-means clustering (Makhoul *et al.*, 1985), are used to group the input vectors. The number of clusters determines the number of hidden neurons in the LVQ network, and the cluster centres provide initial values for the neuron weights w_{ji}. The forming of connections between the hidden and output neurons uses a voting system, with a connection between neurons if the majority of the data in a cluster belongs to a particular output target class.

The hidden neuron weights are modified using supervised techniques to optimise the classification performance of the LVQ network. There are two key algorithms used for the supervised training of LVQ networks: LVQ1 and LVQ2.1 (Kohonen, 1997).

2.3.7.1 LVQ1

The LVQ1 training algorithm only modifies the weights for the neuron that is closest to a presented input vector. This neuron is responsible for the classification of the input vector. If the calculated class is the same as the target output class, then the weights for the activated subclass neuron move towards the input data:

$$w_{ji}(q+1) = w_{ji}(q) + \eta(x_i - w_{ji}(q)) \tag{34}$$

where η is the learning rate and q indexes the training iteration. If the calculated class is different from the target class, then the weights move away from the input data:

$$w_{ji}(q+1) = w_{ji}(q) - \eta(x_i - w_{ji}(q)) \tag{35}$$

The change to the neuron weights effectively repositions the decision boundary between neighbouring output classes (Kohonen *et al.*, 1996). Network training is very fast with only one neuron modified during each training step.

2.3.7.2 LVQ2.1

The LVQ2.1 algorithm is supplementary to LVQ1, and acts to fine-tune the boundaries between neighbouring classes. The LVQ2.1 algorithm can improve the performance of a network already trained with the LVQ1 algorithm because it acts to move the boundary between neighbouring neurons closer to the optimal decision boundary (McDermott, 1997).

LVQ2.1 modifies the weights for the two neurons that are closest to the input vector. This weight modification only occurs if the two neurons produce different classifications and the input vector falls within a window that surrounds the midpoint between the two neurons. The neuron that produces the correct classification moves closer to the input using Equation 34. The other neuron moves away from the input using Equation 35.

The size of the window is defined in terms of the Euclidean distances ϕ_1 and ϕ_2 between the input vector and the two neurons, where $\phi_1 < \phi_2$. The input vector falls inside the window if:

$$\left(\frac{\phi_1}{\phi_2}\right) > \left(\frac{1-w}{1+w}\right) \tag{36}$$

where w is the relative width of the window. The width of the window must be large enough to allow effective optimisation of the boundary between different classes, yet small enough to preserve the training from the initial LVQ1 application. A recommended window width is between 0.2 and 0.3 (Kohonen, 1997).

2.3.8 Fault Detection Using Neural Networks

MLP, RBF and LVQ networks have been used to detect faults in automated systems in a variety of ways.

An MLP network was used to detect faults in an air compressor and cooling system (Juuma & Parkkinen, 1994). The network was trained to detect eight different faults using pressure and temperature sensor data. The network was found to successfully detect faults even when random noise was added to the training data. An MLP network was also used to model errors associated with a multi-axis CNC machine (Oaufi *et al.*, 2001). The generated residuals were fed into an online compensator that acted to reduce the effect of the error.

Both MLP and RBF networks were used to generate residuals relating to robotic manipulators (Terra & Tinós, 2001). The MLP was used to approximate the state equation of the robotic system, and separate MLP and RBF networks were employed to classify the residuals. The RBF network was trained with a self-organising algorithm to determine the centres of the radial basis neurons. The centres were also tuned using the LVQ2.1 algorithm. The RBF network produced better classification results than the MLP when the input space was small.

An LVQ network was used to identify faults in rotating machinery (Wang & Too, 2002). Higher-order statistics were used to extract features from vibration signals, which were then clustered using a self-organising algorithm. The LVQ network used the cluster centres to successfully identify eight different process faults.

2.4 FAULT CORRECTION AND PATH OPTIMISATION

If it is possible to detect a process fault (either using a neural network or some other method), the obvious next step is to attempt to correct for the behaviour that caused the fault, thereby improving the overall performance of the system. This fault correction process requires knowledge of the relationship between the symptom of the fault and the cause of the fault. These relationships can be highly complex and nonlinear in most real-world situations, making it extremely difficult to develop a model-based fault correction strategy. Fault correction strategies instead often rely on expert knowledge of the monitored system. In the case of the automated Y-cutting system, the correction of process faults occurs by optimising the cut-path to increase the success rate of the system.

2.4.1 Current Y-Cut Path Optimisation

Current optimisation of the cut-path uses an iterative observational process. The outcome of an individual cutting trial determines the success rate for a particular set of path parameters. Each trial uses an appropriate number of animals, depending on the observed success rate. If the observed success rate is very low, then the trial only requires a few Y-cuts, since the system is obviously performing sub-optimally. More animals need to be cut as the process improves, ensuring that the calculated success rate is statistically significant. The optimisation of the cut-path requires many cutting trials, with the selected optimum parameters being those that result in the highest trial success rate.

Past Y-cutting experience has demonstrated that the path optimisation process is long and laborious. It can be relatively straightforward to get the system to perform with a moderate level of success, but it is more difficult to attain a consistently high success

rate. Each cutting trial requires at least one person to monitor the automated Y-cutting system constantly. This person must record the outcome of each cut while simultaneously observing the process to determine the likely cause of any cut failures. These process observations are the basis of subsequent changes to the path parameters. The observations often relate to one particular part of the cut-path, requiring the operator to focus on this feature for an extended period. It is usually more efficient to have one person recording and another observing, although this also increases the financial cost of the tuning process. An alternative is to use a video camera to record the process for later observation, although the time taken to review the footage can increase the overall tuning period. Previous Y-cut installations have taken weeks or months to reach satisfactory performance levels, regardless of the number of people that have been available.

An automated path optimisation system could streamline the tuning process and reduce both the associated time and financial cost. This enhancement could potentially increase the uptake of the automated Y-cutting technology. It could also automatically adapt the path to seasonal stock variations that may affect the success rate. This path optimisation would also effectively act as a fault correction system.

2.4.2 Optimisation Problem Definition

The success rate J of the Y-cut system can be defined as:

$$J = f(\boldsymbol{x}) + \theta \tag{37}$$

where $f(\boldsymbol{x})$ is an unknown function, $\boldsymbol{x}$ is a vector of n path parameters $(x_1, x_2, \ldots, x_n)$, and θ is an error term. The error θ allows for the inaccuracy associated with the fault detection. It also takes into account external factors that may influence the success rate of the system. Adjustments or improvements to the Y-cut tool can mitigate the influence of external mechanical factors, such as the blade sharpness or the pressure exerted by the leg-guide. A small component of this error will be due to random noise within the system.

The optimum parameter vector $\boldsymbol{x}^*$ is given by:

$$x^* = \arg\max_{x \in D} f(x) \tag{38}$$

where D is the domain or parameter space within $\Re^n$. The objective of the optimisation process is to obtain an adequate estimate $\hat{x}^*$ of this optimum parameter vector, such that:

$$\left| f(x^*) - f(\hat{x}^*) \right| < \varepsilon \tag{39}$$

where ε is some minimum error.

2.4.3 Local Optimisation Methods

The objective function $f(x)$ can have one extremum (uni-modal) or many extrema (multi-modal). If the objective function is uni-modal and can be expressed analytically to allow the calculation of derivatives, then it may be possible to solve a set of equations explicitly. If a direct solution cannot be obtained, then local optimisation methods can be used to determine the optimum parameters.

The most common local optimisation method is gradient descent, which also forms the basis of the back-propagation algorithm discussed in Section 2.3.2. Back-propagation moves the neural network weights in the direction of the greatest change in the error gradient. The same process can be used to locally optimise an unknown objective function. Since the cut-path optimisation problem requires the determination of a global maximum in Equation 38, gradient ascent rather than descent is used. This optimisation process is expressed as:

$$\hat{x}^*_{k+1} = \hat{x}^*_k + \eta(k) \nabla J(\hat{x}^*_k) \tag{40}$$

where $\hat{x}^*_k$ is the current estimate of the optimum parameter vector, $\nabla J(\hat{x}^*_k)$ is the instantaneous gradient estimate, and $\eta(k)$ is the learning rate.

A number of techniques increase the speed of convergence of the basic gradient descent method. A momentum term can be added to Equation 40 to give:

$$\hat{x}^*_{k+1} = \hat{x}^*_k + \eta(k)\nabla J(\hat{x}^*_k) + \beta(\hat{x}^*_k - \hat{x}^*_{k-1}) \tag{41}$$

where β is the momentum learning rate (Orr & Leen, 1997). The momentum term allows for the inclusion of some fraction of the previous change to the optimum estimate. This momentum term is analogous to the physical concept of momentum; objects continue to move unless an external force is applied. The new estimate for the optimum parameter vector will still be different from the current estimate, even if the gradient term is very small. This increases the speed of convergence in flatter regions of the objective function (Duda *et al.*, 2000).

Adaptive learning rate schedules for $\eta(k)$ can be used to improve the convergence. If the learning rate decreases with each training step according to:

$$\eta(k) = \frac{\eta_0}{k} \tag{42}$$

then the gradient descent algorithm converges in a similar manner to simulated annealing (refer Section 2.4.4.2), with the iterative shift in the objective function estimate decaying even if the instantaneous gradient estimate remains unchanged. This form of annealed learning rate can result in an optimal rate of convergence (Murata *et al.*, 1997), but is highly dependent on the initial learning rate η_0. If η_0 is too small then the algorithm may not converge completely. If η_0 is too big then the algorithm can diverge from the optimal solution. A search-then-converge (STC) algorithm uses a constant learning rate during initial training, and then switches to an annealed learning rate to improve the final convergence (Darken *et al.*, 1992). The STC learning rate can allow for the avoidance of poor local extrema, meaning that this method could be applicable to global optimisation problems.

2.4.4 Global Optimisation Methods

If the objective function is multi-modal, then there is no guarantee that a local optimisation method will locate the global extremum, and a global optimisation method must perform a complete search of the parameter space. There are two distinct classes

of global optimisation methods: deterministic methods and stochastic methods (Zabinsky, 1998). Deterministic methods require the specification of the objective function in terms of a set of algebraic-differential equations. The equations can be solved simultaneously using numerical approximation methods, or sequentially using an integration routine (Esposito & Floudas, 2000). There is no such set of equations to describe the Y-cutting success rate, so a stochastic optimisation method must be used.

Stochastic optimisation relies on the introduction of random elements to attempt to perturb the estimate of the optimal parameter vector $\boldsymbol{x}^*$ towards the true optimum. Although the randomness associated with stochastic optimisation means that convergence can never be theoretically guaranteed, it can be used to gradually shift the estimate towards the optimal vector, resulting in probabilistic convergence over successive iterations. Stochastic optimisation is an online learning strategy, and is particularly convenient when the set of available training data is large, or if there are significant sources of noise within the data. The randomness associated with this form of learning is useful if the objective function is multi-modal, since it allows for the escape from a local extrema and the eventual convergence to a global solution.

The simplest form of stochastic optimisation is a random search technique, with the objective function evaluated at a number of random points within the parameter space. However, this is an extremely inefficient technique, which becomes unfeasible as the number of optimising parameters increases. There is no guarantee of locating the global maximum without an exhaustive search of the entire parameter space.

2.4.4.1 Sequential Random Search

A modification of the basic random search is the sequential random search algorithm (Rastrigin, 1960). The algorithm generates a candidate point based on the current point, with the new point accepted if it improves the objective function. The candidate point $\boldsymbol{u}_k$ is defined as being in a direction D_k and step size S_k:

$$\boldsymbol{u}_k = \hat{\boldsymbol{x}}_k^* + D_k S_k \tag{43}$$

where $\hat{x}_k^*$ is the current estimate. The next estimate of the optimal point is determined by:

$$\hat{x}_{k+1}^* = \begin{cases} u_k & f(u_k) > f(\hat{x}_k^*) \\ \hat{x}_k^* & \text{otherwise} \end{cases} \tag{44}$$

Variants to the sequential random search algorithm predominantly deal with the modification of the search direction and step size based on the local behaviour of the objective function (Masri *et al.*, 1980). Sequential random search algorithms can converge to a global optimum, although convergence can be slow and the performance can be problem-dependent (Cox & John, 1997).

2.4.4.2 Simulated Annealing

Simulated annealing is a form of adaptive random search algorithm (Kirkpatrick *et al.*, 1983). The basis of the optimisation method is the physical process of heating a substance, then allowing it to slowly cool into a low-energy equilibrium state. With simulated annealing, there is a finite probability of accepting a candidate point even if this new point does not improve the objective function. This probability is given by:

$$P = e^{-\Delta E / T(k)} \tag{45}$$

where $T(k)$ is a temperature function and ΔE is:

$$\Delta E = f(u_k) - f(\hat{x}_k^*) \tag{46}$$

The temperature decreases according to a cooling schedule, which reduces the probability of accepting an inferior estimate. The probabilistic nature of these transitions allows the system to escape from local minima if the cooling schedule is sufficiently slow. The schedule also affects the speed of convergence, with annealing algorithms typically requiring a large number of iterations.

2.4.4.3 Controlled Random Search

The controlled random search (CRS) algorithm is another variation of the basic random search technique. It starts with an initial sample of N evaluation points, and iteratively replaces the worst point with a better point (Price, 1978). A new point is found by randomly selecting a subset of the N points and calculating the centroid of this sample. If the new point produces a better objective function estimate then it replaces the worst point, otherwise the algorithm chooses another random subset. This contraction process continues until the difference between the best and worst points is less than some minimum error.

Modifications to the original CRS algorithm use different centroid calculations and local search techniques to improve the efficiency of the algorithm (Ali *et al.*, 1997). Although CRS is heuristic in nature and has no theoretical convergence properties, a modified algorithm can ensure probabilistic convergence to the global optimum (Törn *et al.*, 1999).

2.4.4.4 Other Global Optimisation Methods

There are several other methods for stochastic global optimisation. The multi-start method applies local search techniques (such as gradient descent) to a random sample of points. Each search determines a local optimum, from which the global optimum is selected. This form of search can waste significant time by reaching the same local optimum on multiple occasions. Clustering methods reduce the number of random samples and hence speed up the multi-start algorithm (Rinnooy Kan & Timmer, 1987). Problems can arise if there is more than one local optimum associated with each cluster, and it is often difficult to identify the clusters accurately in higher-dimensioned parameter spaces.

The use of genetic algorithms for optimisation has received significant research attention (Mitchell, 1996). These algorithms attempt to mimic biological evolutionary processes to produce an improving estimate of the global optimum. An initial population of points is created, with each point represented by a chromosome data

string. The points that result in the poorest estimates are discarded, and the remaining points produce the next generation of points. Mutation and crossover operators introduce random changes to the reproduced population, effectively resulting in a stochastic search of the parameter space (Duda *et al.*, 2000). Genetic algorithms usually require a substantial number of generations before an acceptable solution is obtained, and convergence to the global optimum is not guaranteed. There are also significant computational requirements associated with genetic algorithms.

2.4.5 Application to Y-Cut Path Optimisation

There are specific challenges presented by the optimisation of the Y-cut path. The greatest challenge is the evaluation of the objective function at a particular point in the parameter space. Since the outcome of an individual Y-cut is binary, it is only possible to estimate the success rate by averaging the outcome of a series of Y-cuts. This means that each evaluation of the success rate is a time-consuming process. The number of such evaluations needs to be minimised and the optimisation must converge within a reasonable number of iterations. A meat plant operating at eight carcasses per minute will slaughter approximately 3000 animals per shift. If the optimal path parameters are to be found within a single day, then the optimisation algorithm will need to converge in less than 3000 iterations.

The form of the objective function is currently unknown since there is limited prior knowledge that specifically relates the success rate to the path parameters. It may be possible to develop a model of the objective function based on some sort of data-collection exercise. It is also possible to give the optimisation process a head start, since it is reasonably easy to specify an initial set of path parameters so that the system begins with a moderate level of success. Some of these initial parameters can be specified by making physical measurements of the Y-cut installation. An example of this is the starting point of the cut-path, which should correspond to the height of an opening cut on the foreleg of the carcass (refer Section 3.2.2). Other parameters relating to the general shape of the cut-path can be estimated based on previous heuristic knowledge of the process.

3. DEVELOPMENT OF AN AUTOMATED Y-CUTTING SYSTEM

The Automation Systems team of Industrial Research Limited has developed a robotic system to automate the Y-cutting of sheep carcasses. This work began in 1991 and progressed through various design iterations to a prototype commercial system.

3.1 MANUAL Y-CUT PROCESS

The Y-cut is an integral part of the process of removing the pelt from a sheep carcass. This cut is one of the first operations performed on the carcass following the slaughter of the animal.

3.1.1 Inverted Carcass Dressing

The 'inverted dressing' system is the standard method for removing the pelt from a sheep carcass (Authier, 1992). This system uses a continuously moving conveyor-line, with the carcass hung from the moving conveyor chain by all four feet. The chain moves past stationary workers, who perform the necessary cutting, clearing and evisceration tasks (refer Figure 3.1).

After the animal has been slaughtered and hung from the conveyor by all four legs, a butcher makes a vertical cut down the length of the neck. This cut allows access to the neck cavity, whereupon the butcher cuts and ties the oesophagus and severs the main artery. The butcher then drops the forelegs from the chain, allowing the blood to drain fully from the carcass. The forelegs are re-hung once bleeding ceases, presenting the carcass in a horizontal position.

FIGURE 3.1 Inverted carcass dressing at Goulburn, Australia.

The Y-cut is usually the next operation. The cut begins at the vertical opening in the neck, and proceeds underneath the skin up the foreleg to exit above the knee. Cuts to both forelegs result in a Y-shaped slit in the pelt – from the neck to the end of each leg (refer Figure 3.2). Subsequent dressing tasks clear the pelt from the foreleg, shoulder and neck of the carcass. Either the same Y-cutting butcher or another down-stream butcher completes this clearing task, depending on the dressing system employed by the plant. The delegation of these tasks depends on the speed that the conveyor chain is moving. In New Zealand and Australia these chain speeds can range from three to twelve carcasses per minute (ccs/min). Plants with slower chains will employ two or three people, who will each Y-cut and clear the legs of a carcass as it moves along the conveyor. Plants with faster chains will have two or three butchers solely doing the Y-cut and another two or three people completing the clearing of the legs.

FIGURE 3.2 A manually Y-cut carcass (note that folds of skin obscure the neck opening).

3.1.2 Contamination Issues

It takes approximately eight seconds to Y-cut each carcass, including sterilisation of the knife blade prior to cutting the next animal. This cycle time does not include sharpening of the knife-edge, which occurs on a regular basis due to the dulling action of the pelt, wool and imbedded grit. Nor does it include sterilising the blade between each foreleg. Some plants employ a "double-knifing" system, where one knife stands in sterilising hot water while the butcher uses another knife to cut the current animal.

Informal observation indicates that many meat-plant workers fail to sterilise knives after every cut. There is often significant time-pressure placed on workers to maximise the carcass throughput: by keeping the conveyor running continuously at the maximum possible speed, and by employing the minimum number of people. If a Y-cut takes

longer than expected on a particular animal, then there may not be time to sterilise before the next carcass arrives. The repetitive nature of the job (workers in high-speed plants may be processing up to 5000 carcasses per shift) combined with a lackadaisical attitude means that butchers can view sterilising as unnecessary extra work that often occurs only sporadically if plant supervisors do not rigorously enforce standards.

Contamination of the carcass can also occur if the quality of the Y-cut is poor. The knife blade has a blunted point to minimise the damage to the subcutaneous membrane layer that separates the pelt from the underlying meat. It is important not to rupture this membrane, as it acts to protect the meat from direct microbial contamination (refer Figure 3.3). Tiredness, inattention and poor technique can all contribute to the incidence of this membrane damage. The risk of contamination increases if the knife has not been regularly sterilised.

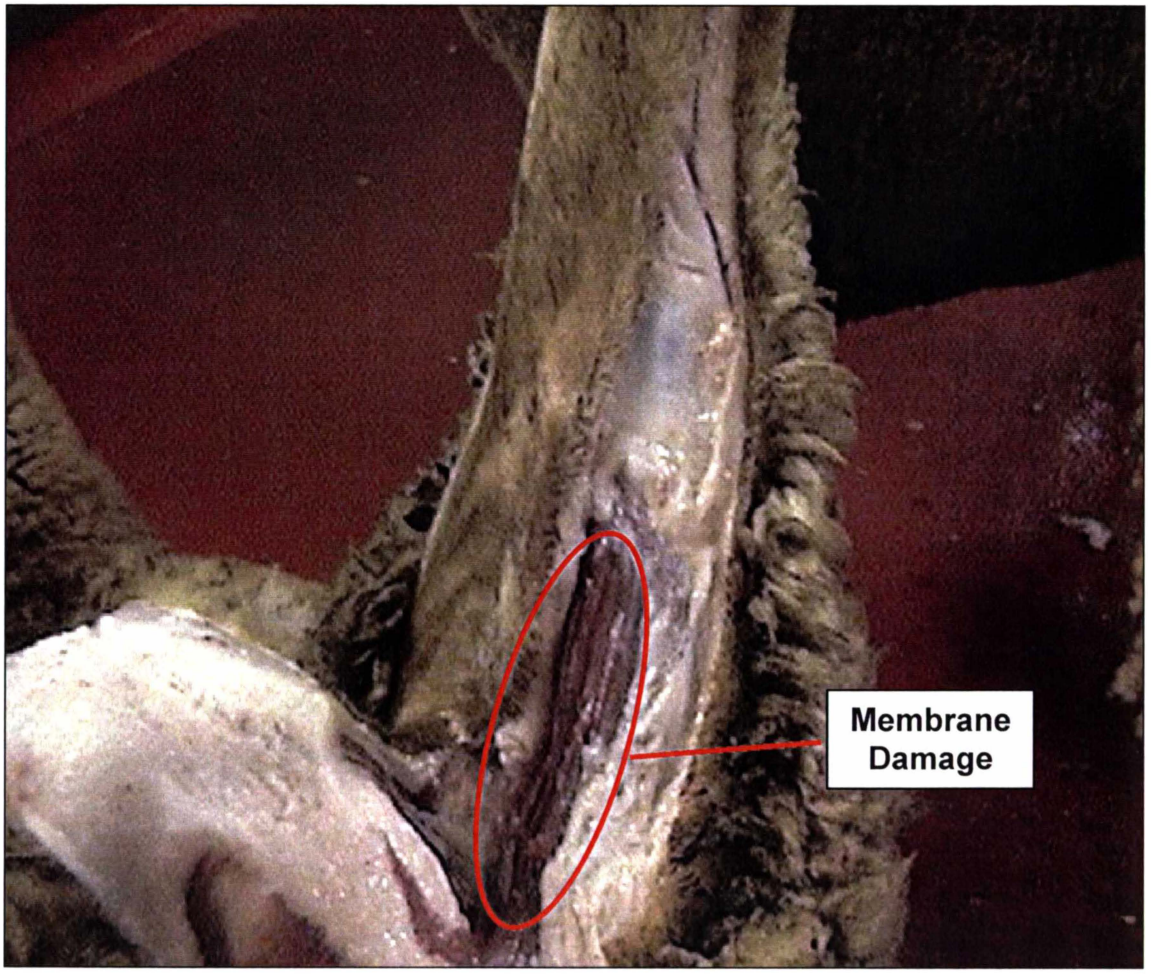

FIGURE 3.3 Unacceptable damage to the subcutaneous membrane of the carcass.

3.1.3 Worker Safety

The manual Y-cut is a two-handed operation, with one hand holding the knife and the other tensioning the skin along the cut-path. This makes the cut a very dextrous task, requiring the butcher to be highly skilled and experienced. The proximity of the tensioning hand to the knife, in combination with the upward direction of the blade and the variability in pelt strength between animals means that Y-cutters have a high incidence of injury. Self-inflicted stab wounds to the hand, arm, chest or face can occur and there is significant risk to neighbouring butchers, particularly as there are often a number of butchers working in the same general workspace.

The arduous nature of the work has also led to a high incidence of occupational overuse syndrome (OOS) among butchers. These injuries are particularly prevalent amongst Y-cutters, with the task placing a continual load on the finger, wrist, elbow and shoulder joints. The resulting accident compensation claims, worker downtime, and the possibility of long-term disabilities are all of concern to the industry. In 2001, the Accident Compensation Corporation (ACC) of New Zealand spent NZ$11 million on meat industry claims. New Zealand meat-processing companies also spent an additional NZ$15-20 million (excluding ACC premium costs), giving a total industry cost of NZ$26-31 million annually (Accident Compensation Corporation, 2002).

3.1.4 Towards an Automated Y-Cutting System

Contamination and worker safety are two of the main drivers behind the development of automation systems for use in the modern meat-processing environment. An automated process is potentially more precise, with greater controllability and repeatability. This means that the sterilisation of contact surfaces after each cut can be guaranteed. It also means a drastic reduction in the incidence of OOS associated with the long-term execution of the task, and the immediate elimination of self-inflicted injuries. Other advantages of pursuing automation in the meat-processing industry include:

- More stringent compliance with food hygiene and disease control regulations.
- Tighter control over plant-related expenses, such as water and electricity.

- Direct financial savings from reduced labour costs and lower accident compensation premiums.
- Potential for running multiple shifts or even continuous operation.
- Continuing economic viability for individual plants in the face of domestic and international competition.
- Freeing skilled butchers to work in more value-added processing areas, such as boning or speciality cuts.

Many of the individual tasks are difficult to automate, requiring a high level of dexterity and specialised tool design. The slaughter-floor environment is particularly harsh with all equipment having to withstand high-pressure water and caustic cleaning products. Inter-animal variations mean that a sophisticated sensing system is required to handle the highly variable carcasses. The economics of the meat-processing industry demand extremely robust and reliable technology, with automated systems typically needing to demonstrate a success rate of greater than 98%.

3.2 AUTOMATED Y-CUTTING DEVELOPMENT AT IRL

Development of an automated Y-cutting system began at IRL in 1991. Initial research focussed in three key areas: the design of a suitable Y-cutting tool, the development of a motion system for the tool, and the provision and integration of an appropriate sensing system.

3.2.1 Tool Development

Work began with an examination of the human Y-cutting motion, using a standard knife and attached strain gauge sensors to capture the cutting forces and moments (Taylor, 1993). This work identified several key success factors, including tactile sensing of the required cutting forces and the ability to fuse this information with visual cues to coordinate the cutting motion. While the intention was not to replicate the human cutting method, the understanding gained from this exercise was crucial to give a better appreciation of the task and to help identify different ways to automate the process.

3.2.1.1 Cutting Method

Subsequent trials used a range of different cutting techniques, including knife-based systems, rotary and shearing actions, and water-jet cutting (Taylor & Brooking, 1994). The most effective cutting technique found is a variation of a knife-based system, incorporating two blades moving with a reciprocating action. This motion keeps the pelt under tension and ensures that the action of the tool provides the majority of the cutting forces.

An important feature of the blade design is the profile of the blade-tip, which is blunted to prevent penetration of the subcutaneous membrane and to stop the tool from prematurely breaking out of the pelt (refer Figure 3.4). A second feature is the positioning of coarse feeder teeth in front of the sharper cutting blades. These teeth pull the pelt up onto the cutting surface, which helps to maintain the quality of the cut by keeping the pelt under tension.

Initial cutting tests used a U-shaped cut-path; from the top of one leg, down to the neck, and then up to the top of the second leg. This path worked well for slow cutting speeds, but the upward motion of the cut-path tended to lift the second leg out of the conveyor spreaders for faster cutting speeds. This observation resulted in the decision to cut both legs with a downward motion, rather than attempting to duplicate the human upward Y-cut.

3.2.1.2 Motor Selection

Early tool iterations used a compressed-air motor to drive the cutting blades. This type of motor had the advantage of not being electrically powered, thereby reducing the need for sealing against water ingress. The motors were also lightweight and compact, minimising the overall mass of the Y-cut tool. However, extensive testing found these motors to be mechanically unreliable, excessively noisy, and unable to handle the cutting load. When the tool had no applied load, the motor would run at approximately 5000 rpm, but the speed could drop below 3000 rpm when cutting. The motors were expensive to run, consuming excessive quantities of compressed air.

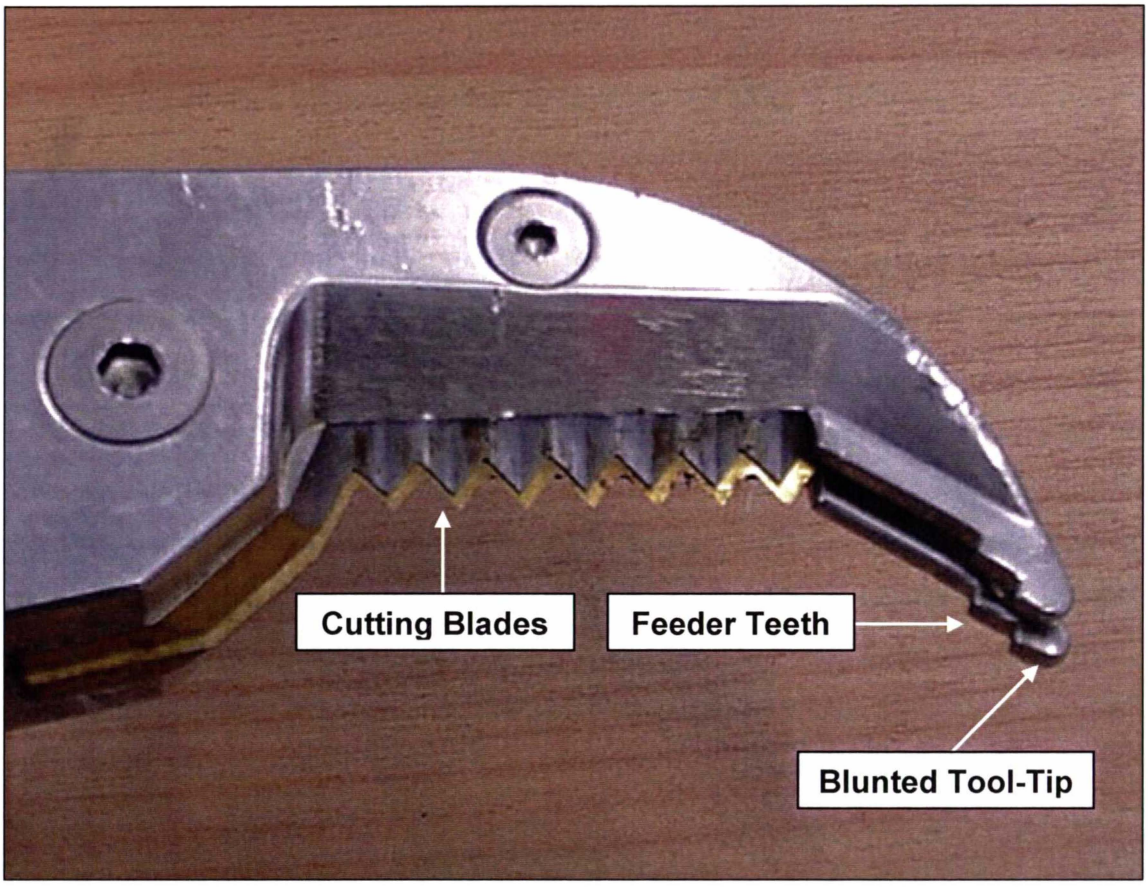

FIGURE 3.4 Detail of the Y-cut tool blades.

Later iterations have incorporated DC electric motors with separate pulse-width modulated (PWM) drive controllers. While these motors have increased the bulk and weight of the tool, they are extremely robust, reliable, quiet, and relatively inexpensive to operate or replace. The PWM drive is very effective at coping with the varying loads that the motor and blades have to deal with, whilst maintaining a constant cutting speed.

3.2.1.3 Sterilisation

To meet hygiene requirements, any surface that could potentially contact the meat of the carcass must be sterilised between each cut (Ministry of Agriculture and Forestry, 2002). Sterilisation is usually carried out by immersing the cutting device in a hot-water bath (82° C) for several seconds, or by placing the surface underneath a series of spray nozzles (also maintained at 82° C). A better sterilising method for the Y-cut tool is the spray option, since it allows for the precise metering of the water quantity while

still maintaining the same aseptic standards. Initial development focussed on the provision of a separate sterilisation unit that moved in front of the tool (Templer & Wichman, 1997). However, the motion of the sterilisation unit adversely increased the overall cycle-time of the automated process. The same restriction also applied when the tool moved to a stationary sterilisation unit.

The solution is to incorporate a spray nozzle into the design of the tool (refer Figure 3.5). This requires the attachment of a hot-water line to the tool, but provides the flexibility for the system to sterilise as and when appropriate. The spray run-off can be directed into a dedicated drainage conduit, although many plants have adequate in-floor drainage. The tool is directed away from the carcass during the sterilisation cycle to ensure that the spray does not deflect onto the carcass.

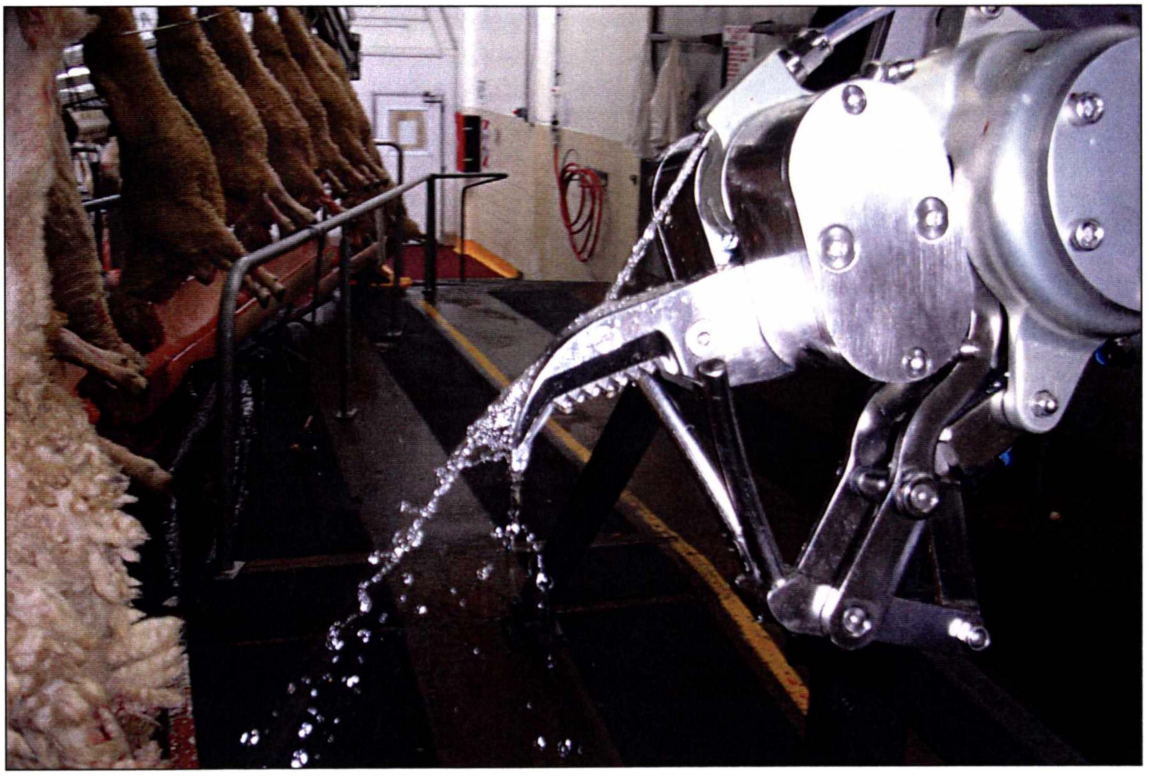

FIGURE 3.5 Sterilisation of the blades using an integrated spray nozzle.

Microbiological testing of cutting surfaces has demonstrated the effectiveness of the tool sterilisation. Although results are commercially sensitive, tests conducted by the Ministry of Agriculture and Forestry (MAF) indicate that the log of the aerobic plate count for a trial of 50 carcasses is approximately 10% lower for robotic Y-cut carcasses than for manually Y-cut carcasses (Templer & Wichman, 1997). The log of *E. Coli*

count shows a similar reduction for carcasses that have been Y-cut with the automated system.

3.2.1.4 Mechanical Compliance

Because of the high variability of the carcasses, a sophisticated sensing system is required to quantify most of the variations. However, it is often easier and cheaper to use some form of compliant design. This is especially evident when cutting over the knee region, as there are a number of tough ligaments that have to be broken through or avoided. The process loads experienced by the tool are highly variable, with the strength of the knee ligaments dependent on the age, breed and health of the animal. These factors are very difficult to determine prior to the commencement of the cut, meaning that the automated system must be able to handle these load variations without prior knowledge of the expected magnitude.

Mechanical compliance can be added to the tool or the motion system, or both. The mechanics required to generate the cutting action mean that the tool is particularly rigid, and not amenable to compliant components. Similarly, the motion system requires a great deal of strength and rigidity. The point where the tool mounts to the motion system is the most appropriate place to incorporate mechanical compliance. Tests using different mechanical solutions, including rubberised mounting blocks and pins, have shown signs of fatigue or failure after short durations of use. The current tool uses a torsional rubber suspension unit, with the load experienced by the tool producing a moment about the centre of rotation of the mounting (refer Figure 3.6).

3.2.1.5 Leg-Guide Development

The tool needs to press against the leg with a reasonable amount of force to insert reliably into an opening cut at the top of the leg (refer Section 3.2.2). This force results in the tool sliding off the side of the leg if it does not meet the leg in a completely perpendicular fashion. The semi-complaint nature of the conveyor chains and spreaders in many meat-plants exacerbates the problem. This means that the tool can easily push aside carcasses on these chains, and the alignment of the tool with the leg can be lost.

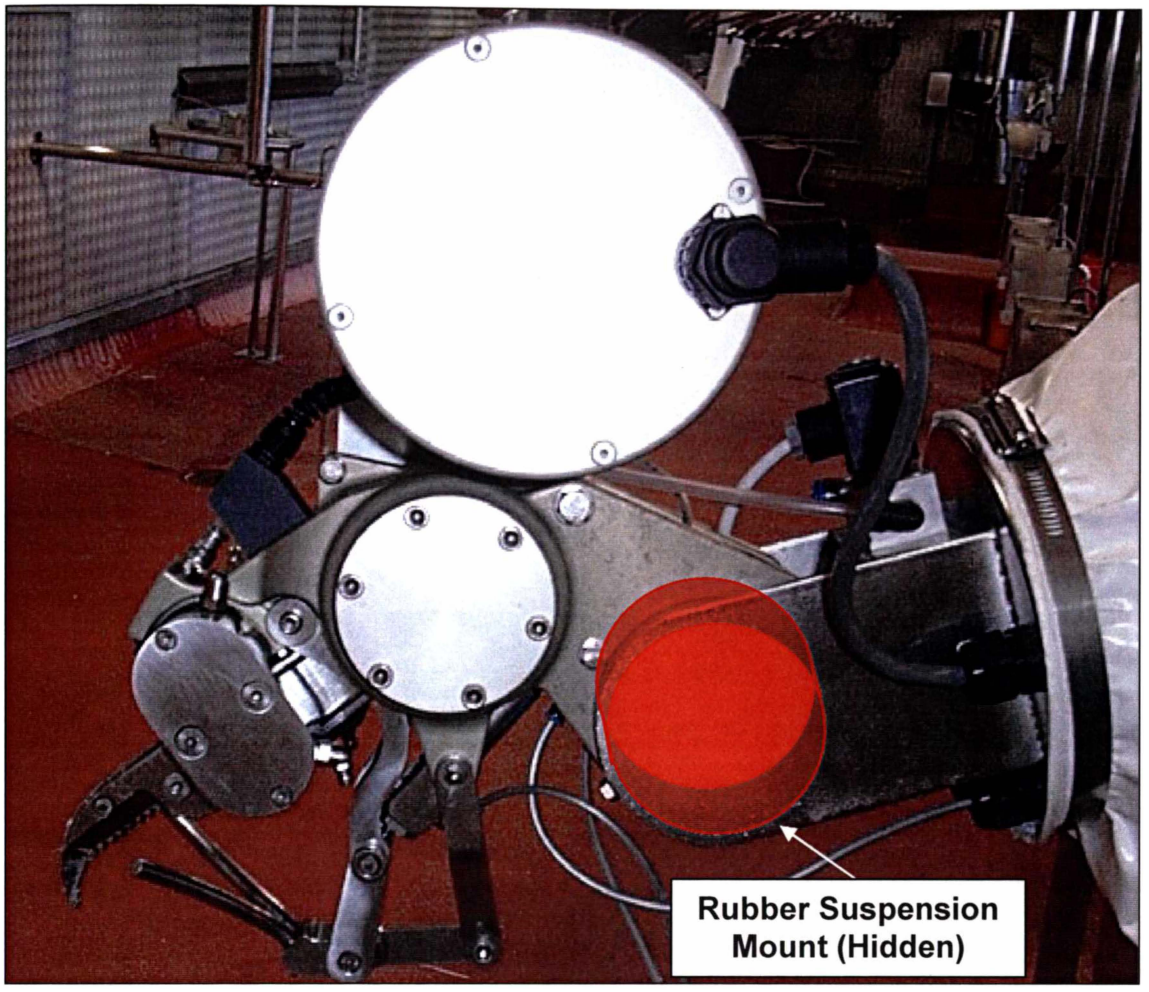

FIGURE 3.6 The current Y-cut tool mount.

A collapsible leg-guide prevents the misalignment of the tool. It consists of a set of splayed forks driven by a small pneumatic cylinder (refer Figure 3.7). The actuation of the forks enables the guide to retract once the tool has inserted into the opening cut. Tuning the cylinder air-pressure allows the leg-guide to act in a semi-compliant manner when pressed against the leg, resulting in more consistent insertion success rates.

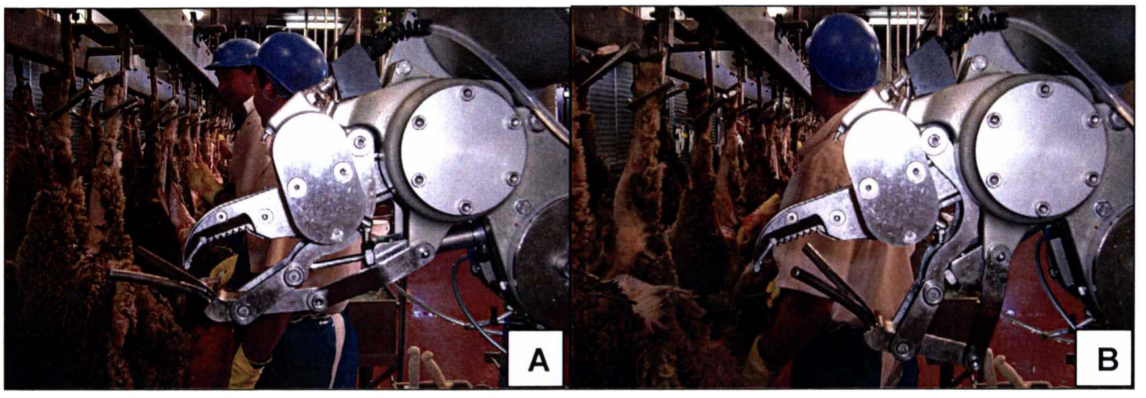

FIGURE 3.7 Leg-guide actuation: a) extended and b) retracted.

3.2.2 Sock-Ringing

With the blade-tip blunted to prevent damage to the pelt or membrane, the tool requires a separate system to allow it to enter underneath the pelt and begin the cut. A sock-ringing system is used to cut through the pelt to the depth of the bone, leaving a pocket of skin that the tool can insert into (refer Figure 3.8). The sock-ringer has a slowly rotating serrated circular saw blade mounted on the conveyor railing. It is positioned so that the opening cut is as high up the leg as possible.

3.2.3 Sensing Techniques

The automated Y-cut system needs to sense several critical carcass parameters. These parameters provide the means to generate a suitable trajectory for the tool to follow. In particular, it is necessary to determine where the tool should insert, where it should complete the Y-cut, and how it should move between these two points.

3.2.3.1 Insertion Point Detection

Finding the position of the opening cut produced by the sock-ringer is paramount for the success of the automated Y-cutting system. This localisation is straightforward, since the sock-ringer and the opening cut are always at a constant height relative to the conveyor rail. The lateral position of the opening cut is determined by placing a mechanical switch at the height of the sock-ringer in the path of the moving carcass foreleg. It is unnecessary to sense the third remaining Cartesian depth dimension accurately, since the leg-guide directs the leg to the blades. The tool must also exert sufficient force in this plane to achieve reliable insertion.

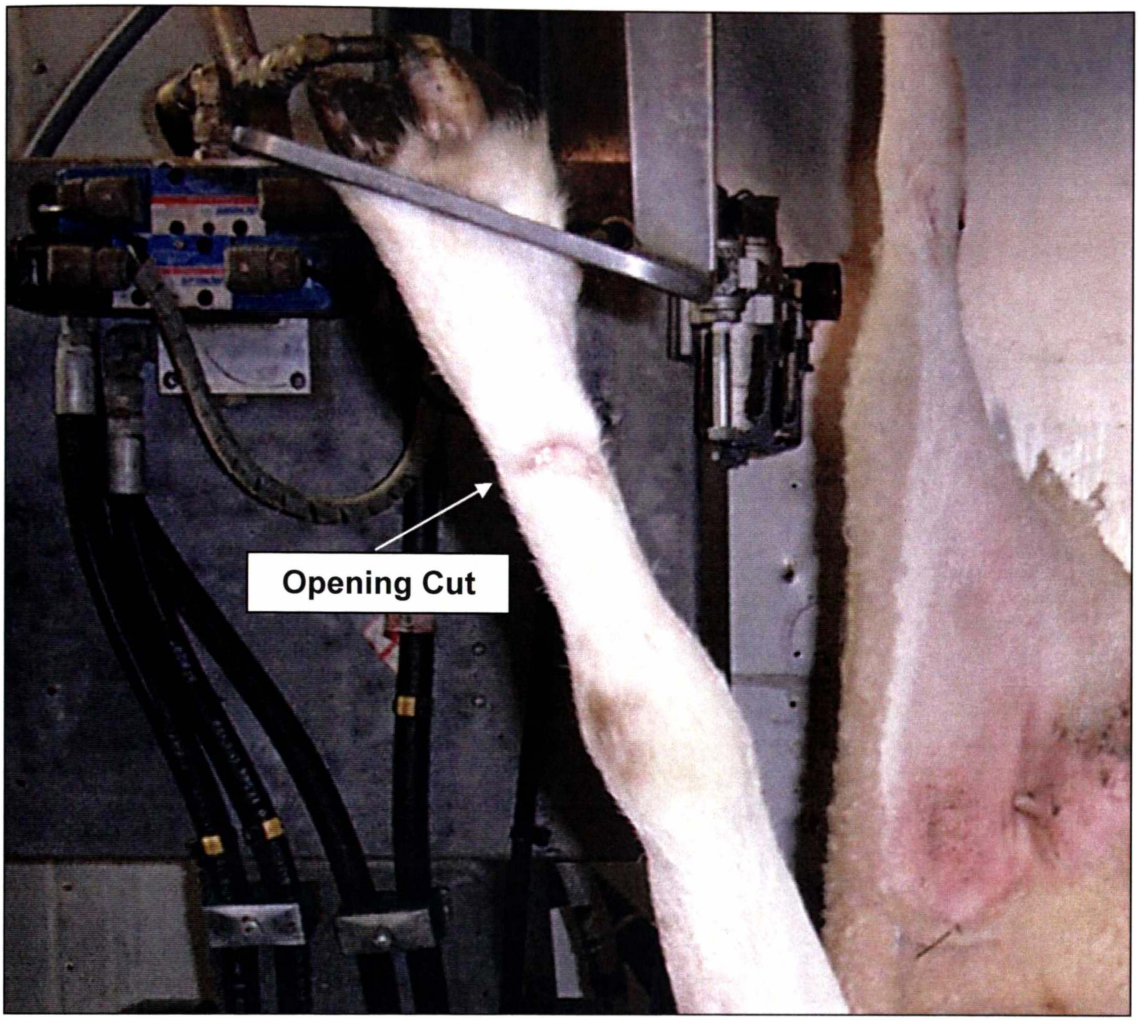

FIGURE 3.8 A typical opening cut at the top of a leg.

3.2.3.2 Cut End-Point Detection

The desired end-point for each leg cut is the top of the neck opening (refer Figure 3.9). Different sensing technologies have been tested for their ability to localise this opening. These techniques include machine vision, microwave sensing, ultrasound, and structured-lighting projection. All of these methods suffer from an inability to detect the desired end-point reliably if the neck opening is obscured. This is often the case, especially for breeds of sheep with large folds of skin around the neck region (refer Figure 3.2).

The developed sensing system uses a pair of tactile wands, each driving a rotary encoder. One wand rotates vertically and measures the height of the brisket, which is the flat region across the chest of the carcass. The other wand rotates horizontally and

measures the depth of the neck (refer Figure 3.9). These two measurements in conjunction with a linear model give an estimate of the location of the neck cut. This model is prone to a reasonable amount of error: typically ±50 mm. However, extensive testing has demonstrated that there is a natural tendency for the cut to migrate towards the neck opening when the Y-cut tool is moving in this general direction. This tendency is due to the underlying direction of the pelt grain, and is contingent on the tool maintaining sufficient pelt tension.

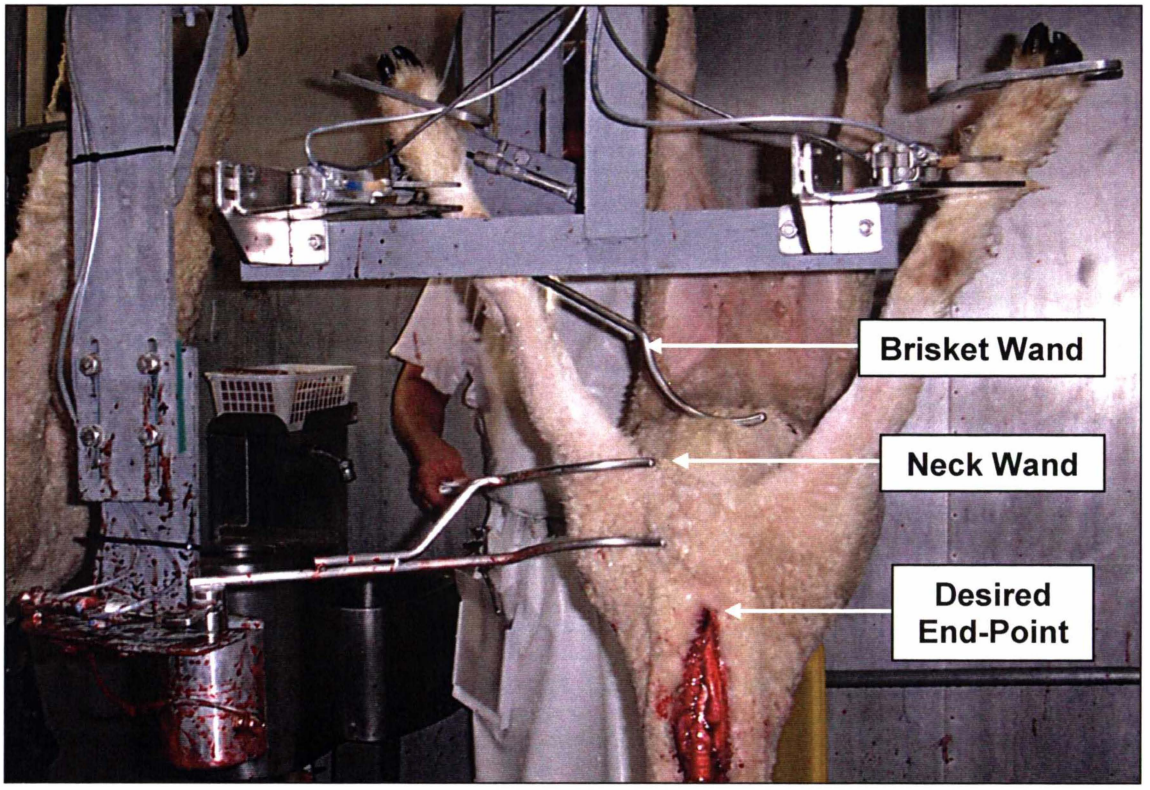

FIGURE 3.9 Tactile wands used for the determination of the Y-cut end-point.

3.2.3.3 Intermediate Leg Detection

Various non-contact sensing techniques have been used to localise intermediate points on the leg. Most of these sensors suffer from an inability to resolve the true location of the carcass pelt through varying wool depths. Earlier abattoir installations used tactile sensors to determine the location of the knee. Repeated testing over time demonstrated that these measurements added little value to the planning of a suitable cut trajectory. This has resulted in the use of a cut-path that is dependent solely on the insertion and end-point. A basic linear path between these two points is adjusted using constant

offsets tuned to best match the leg profile. Section 3.4 describes this path-planning algorithm.

3.2.3.4 Conveyor Tracking

It is important to know the position of a carcass on the moving dressing chain at all times. The ability to track a moving object on a conveyor traditionally uses an encoder attached to a driveshaft of the conveyor, and some sort of software to convert the encoder stream to a usable metric. In many plants, a driveshaft may be some distance from the Y-cutting system, and there is often a significant amount of slack in the intervening conveyor-chain. Both of these factors can add significant noise to the tracking signal, requiring some form of filtering or a separate drive system for the encoder.

3.2.4 Robot Development

In parallel with the design of the Y-cut tool, a motion system is required to transport the tool. Key factors influencing the design include the likely tool payload, the size of the workspace, the articulation and dexterity of the Y-cut task, and the level of control needed to achieve the required speed and motion precision.

Initial laboratory and abattoir testing used a six degree-of-freedom (DOF) IRB 2000 industrial robot manufactured by ABB (refer Figure 3.10). However, the design and construction of the robot meant that it was not suitable for the harshness of the meat-plant environment, and would not have sustained direct hosing or chemical cleaning. No attempt was made to permanently enclose or protect the robot from this environment since the robot was only being used for initial development work. Very few industrial robots could handle this kind of environment, and any that could were considered prohibitively expensive.

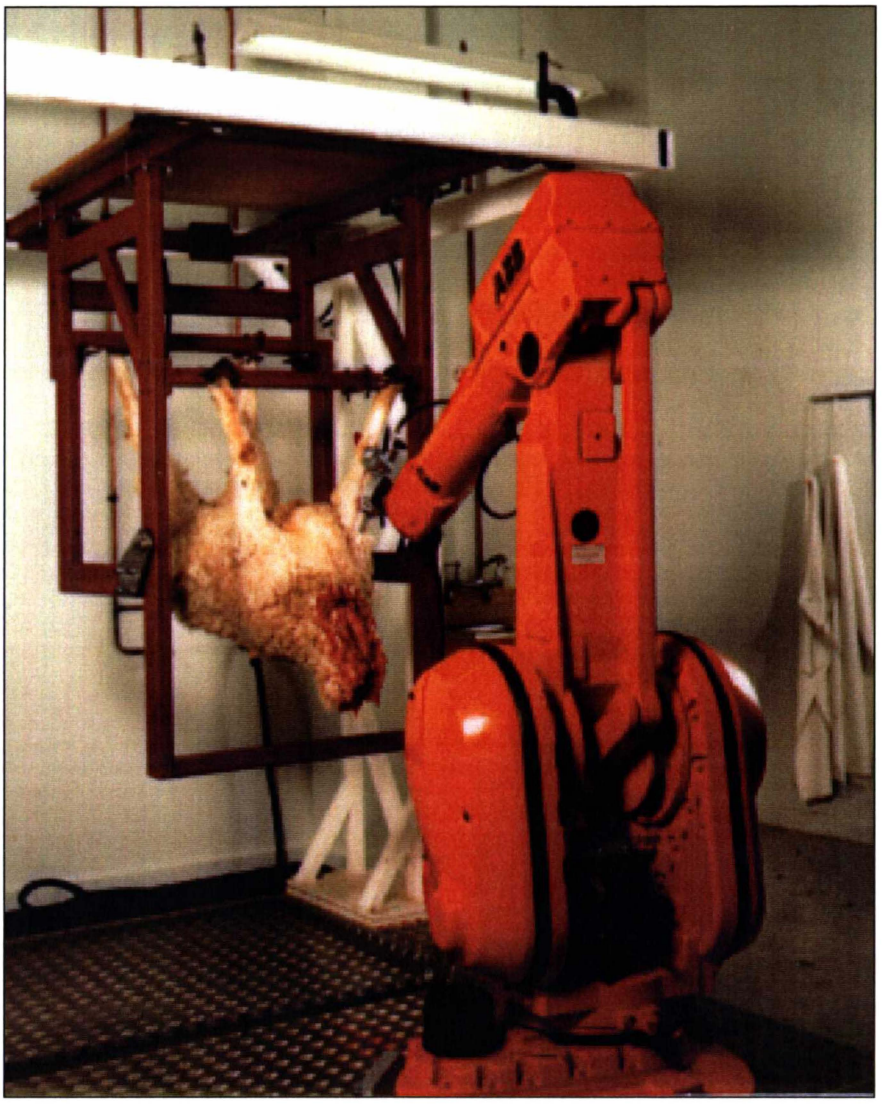

FIGURE 3.10 The ABB IRB 2000 robot used for initial Y-cut development.

IRL commissioned Auckland-based Motion Design Limited to design and build a custom-made robot in 1995. The project specifications required the robot to be highly water and corrosion resistant, as well as being suitable for future food handling and processing tasks. The resulting IRL7L and IRL8L robots are of a four-axis cylindrical design (Taylor & Templer, 1997). The main traversing axis permits horizontal tracking of the conveyor chain for up to 1200 mm. The shoulder axis allows vertical rotation of the probe axis, which can extend telescopically by 600 mm. At the end of the probe is a wrist axis, allowing ±180° rotation of the tool. Individual DC servomotors drive the four axes, with a separate PC-based controller. The exterior of the robot is built from food-grade stainless steel (refer Figure 3.11), with environmental sealing to at least an IP67 rating (Australian Standard AS1939, 1990).

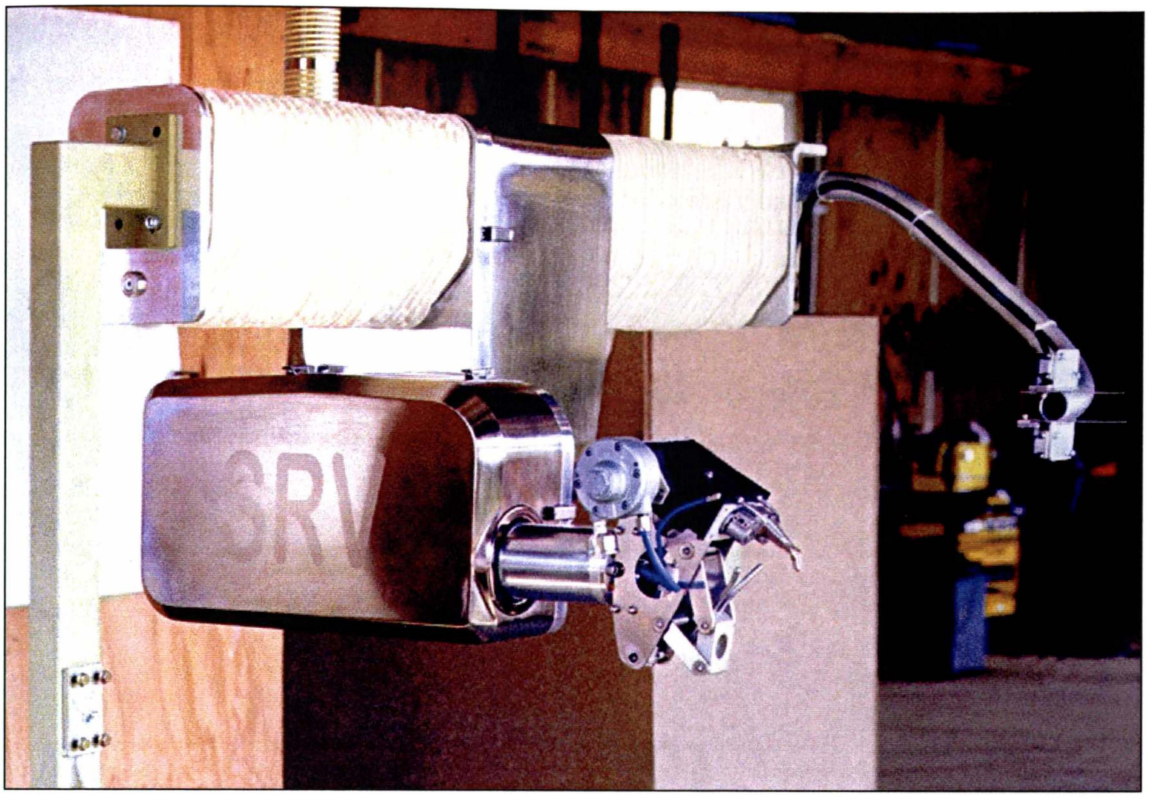

FIGURE 3.11 An IRL8L robot with an early-generation Y-cut tool.

3.2.5 Automated Y-Cut Installations in New Zealand

Several New Zealand meat-processing plants have been involved in the testing and installation of prototype automated Y-cut systems. The first installation was at the Ruakura research abattoir in Hamilton, and used the ABB IRB 2000 robot to test a variety of cutting techniques (Taylor & Brooking, 1994).

The first IRL7L robot installation was at Progressive Meats in Hastings in 1995. This plant operated at a relatively slow processing speed of 2.5 – 3.0 ccs/min, and provided the opportunity to improve the sensing and control software. The culmination of the work at Progressive Meats involved running the system for a full seven-hour production shift on over one thousand animals. The overall success rate was 99.1%, with the trial including a range of different-sized carcasses with a variety of wool lengths (Templer & Wichman, 1997).

The next challenge was to demonstrate the system operating in a plant with a faster chain speed. An IRL8L robot was installed at Alliance Smithfield in Timaru, which

operated at 8.3 ccs/min (refer Figure 3.12). To achieve this processing rate, the robot had to complete the Y-cut with a cutting speed 5-6 times faster than was used at Progressive Meats. This resulted in substantial modifications to the tool design and cut-path. The system achieved success rates of over 98% in continuous cutting trials.

FIGURE 3.12 The IRL8L Y-cutting robot in action at Alliance Smithfield.

A number of other trial Y-cutting systems were installed in plants throughout New Zealand, based on the IRL8L robot and various tool iterations (Templer *et al.*, 1999). These installations demonstrated the commercial viability of the full system (Templer, 2000). However, mechanical reliability, cost and manufacturing concerns prior to the full commercialisation of the system necessitated the move to an alternative robotic platform.

3.3 KUKA Y-CUTTING SYSTEM

Because of the concerns with the custom-built robots, the decision was made to move to a standard industrial robotic platform produced by an internationally reputable supplier. A number of different robots were considered, including ABB (used during initial development work), Fanuc, Kawasaki, KUKA and Motoman robots. A KUKA robot was selected, as it met the payload and speed requirements of the Y-cut process and was competitively priced.

3.3.1 KUKA Robot

The KUKA KR60/2 is a six DOF robot that can accommodate a 60 kg payload. This payload capacity greatly exceeds the weight of the Y-cut tool, which is approximately 15 kg, and provides additional capacity to handle the process cutting loads. Other relevant specifications of the KR60/2 include:

- Weight: 875 kg (without tool)
- Horizontal reach: 2041 mm
- Vertical reach: 2498 mm
- Positional accuracy: 0.2 mm
- Maximum linear speed: 2 m/s

The KR60/2 is not wash-down proof, so a separate enclosure protects it from water ingress. The robot sits in a stainless steel base, and is covered by a tailored PVC bag (refer Figure 3.13). The base has a removable stainless steel "lobster-back", which clamps onto the main base and seals with an O-ring. The robot bag clamps against the lobster-back with a ratchet-clip, and to the end of the robot wrist with a standard hose-clamp. The fan in the ceiling space inflates the bag, forcing air into the robot base via a stainless steel conduit. This inflation acts to keep the bag away from the robot while it is operating, and allows the robot to complete a full range of motions without damaging the protective bag. The positive air-pressure inside the bag also acts to drive out impinging water that may have circumvented the other sealing barriers.

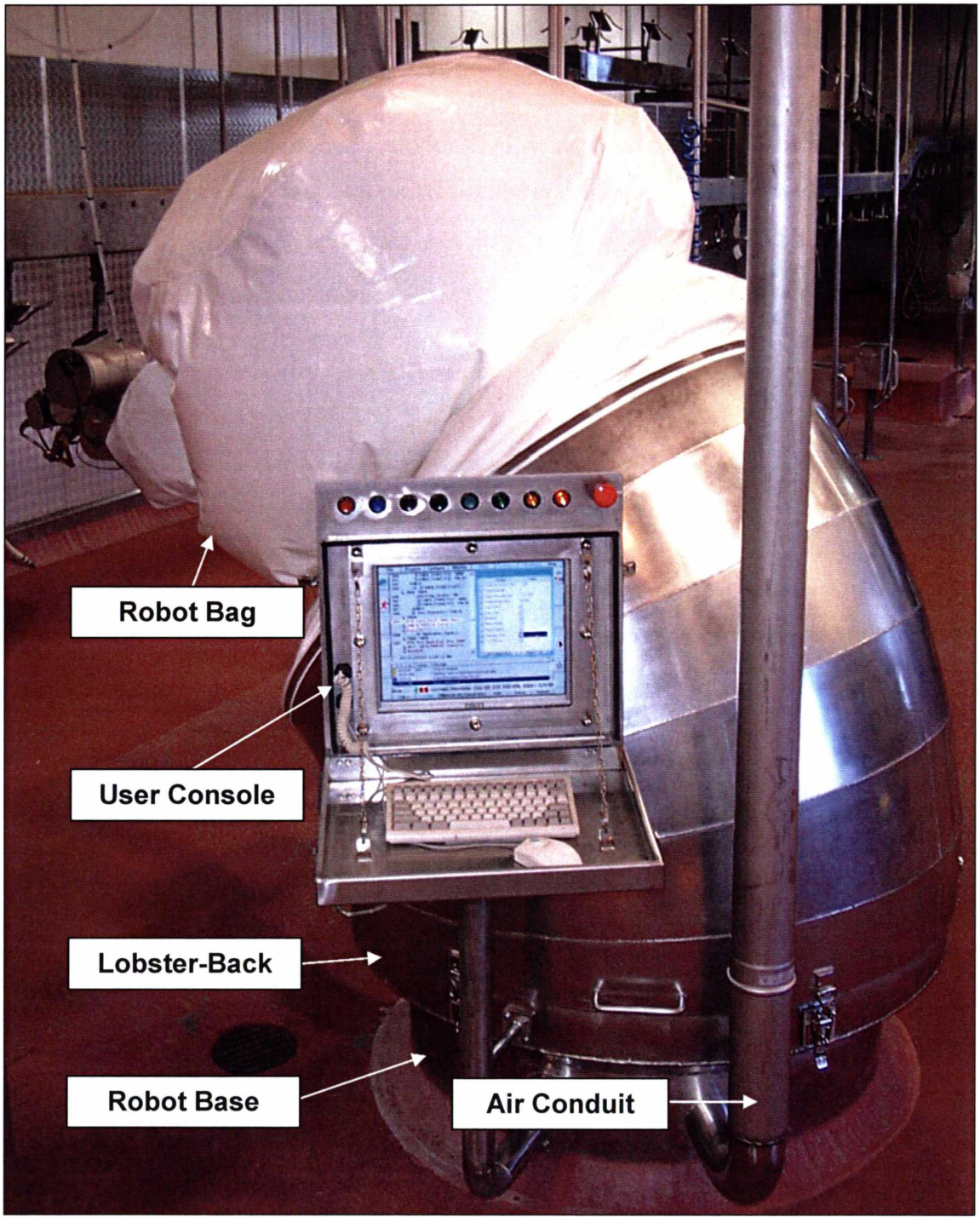

FIGURE 3.13 The KUKA Y-cutting system.

Figure 3.14 shows the configuration of the six robot axes. The padding that covers axis A3 prevents the robot bag from contacting the exterior of the hidden servomotors. These motors become heated following continuous operation and could damage the PVC material of the bag.

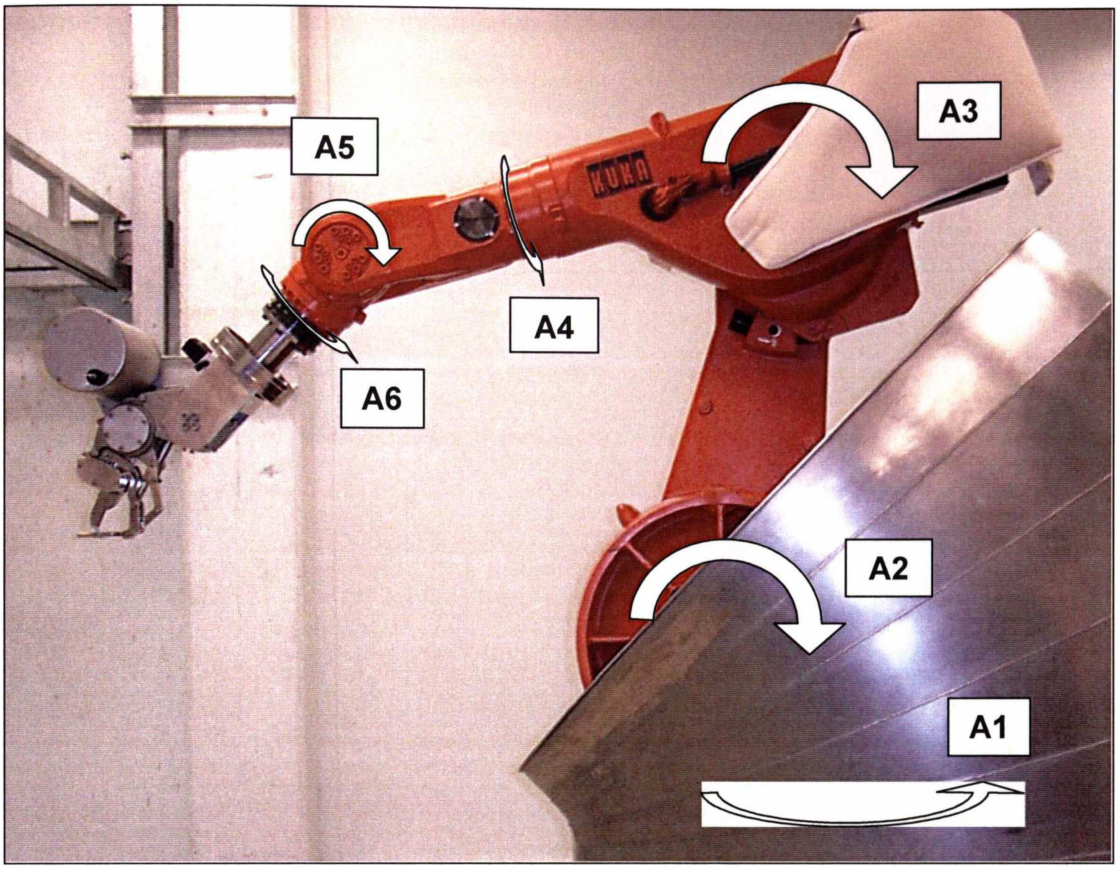

FIGURE 3.14 The axis configuration of the KUKA KR60/2 robot.

3.3.2 Robot Controller

A KUKA KRC2 robot control system powers and controls the KR60/2 robot. This controller incorporates:

- 6 servo-drives for the robot axis motors (plus space for 2 additional drives)
- PC-based controller
  - Pentium III 733 MHz
  - Combined Windows/VxWorks operating system
  - LP Elektronik PCI Controller Area Network (CAN) bus scanner card
- Integrated electronic safety circuitry
- Battery back-up for PC
- Weight: 178 kg

As with the robot, a stainless steel enclosure contains and protects the KRC2 controller (refer Figure 3.15). The controller enclosure also contains the PWM drive for the Y-cut tool, which mounts on top of the controller inside a separate cabinet. The enclosure has sufficient clearance from the controller to meet KUKA air-circulation specifications. A conduit connects the controller enclosure with the robot base, allowing a constant airflow from the fan that inflates the robot bag. This conduit contains and protects the robots power and resolver cables. A venting outlet runs from the enclosure into the ceiling space.

3.3.3 User Console

The standard method for controlling and programming a KUKA robot is via the use of a KUKA control pendant (KCP). The KCP incorporates a screen, 6-DOF mouse, keypad, and other feature-selection buttons. As with the robot and the controller, the KCP is not suited to the harshness of the meat-plant environment.

A separate console replaces most of the functionality of the KCP. The console features a flat-screen LCD display mounted behind a sealed glass plate, a standard keyboard and mouse, and a number of IP67-rated buttons and indicator lamps (refer Figure 3.13). A fold-down tray serves as a working surface during normal operation, and protects the screen, keyboard and mouse during the cleaning of the plant. The buttons and lamps remain visible when the tray folds up, providing a simple interface for the plant to operate the robot under normal conditions.

The console mounts on the rear of the robot base, with all components constructed from stainless steel. The KCP is stored in the controller enclosure, and remains connected to the controller at all times. This allows access to the KCP for quick manual movement of the robot in emergencies, which is a capability that the console interface cannot provide.

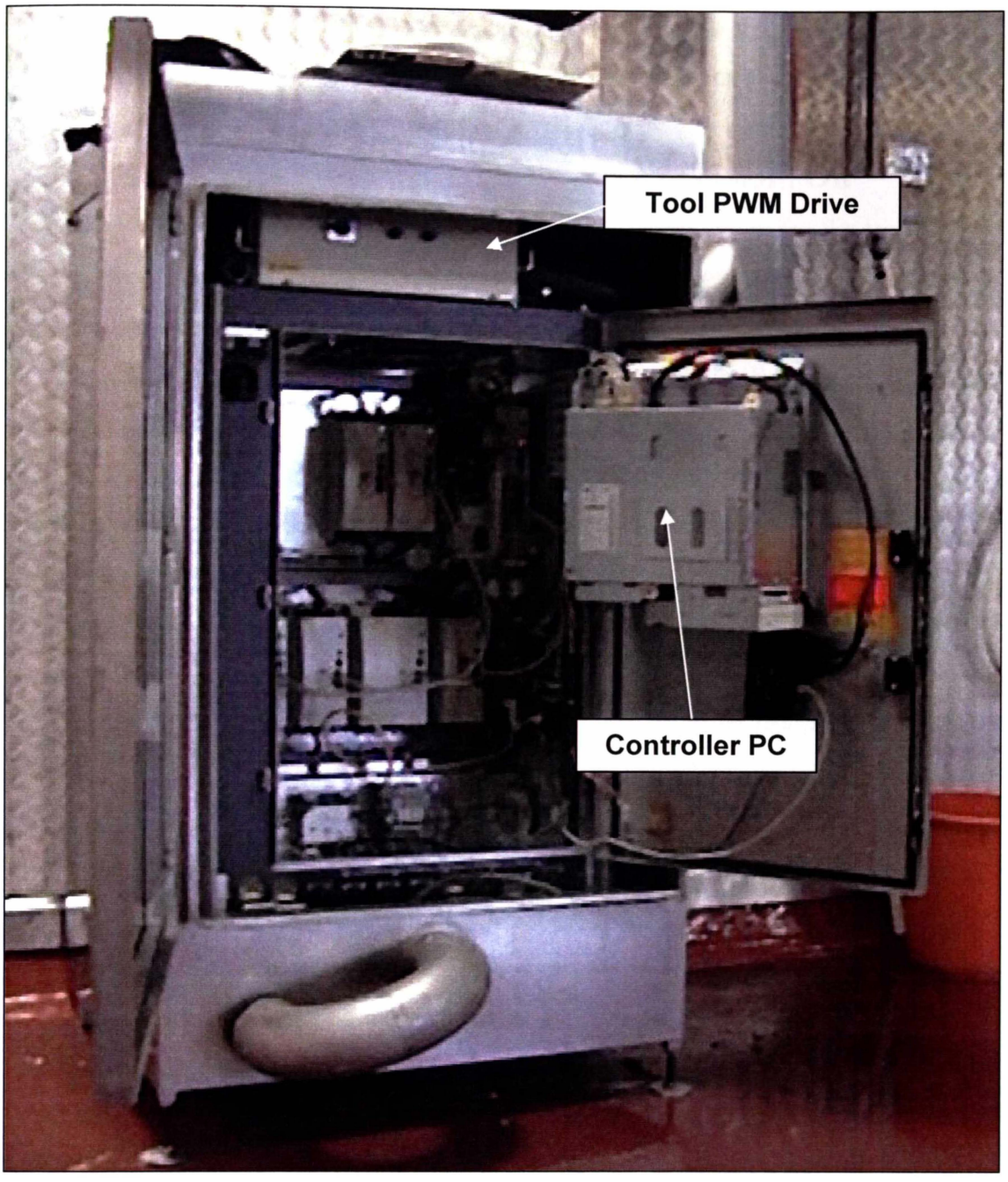

FIGURE 3.15 The KRC2 controller inside its enclosure.

3.3.4 Sensing System

The KRC2 controller can interface to a number of industry-standard communication field-bus technologies. One of the most widely used systems is DeviceNet, which involves the RS-422 communications protocol implemented on CAN bus hardware. This technology allows the connection of a distributed network of compatible sensing,

data and I/O devices to a single common bus, reducing the complexity and quantity of cabling, and providing sophisticated diagnostic capabilities.

3.3.4.1 I/O Modules

Allen-Bradley POINT I/O modules supply the basic DeviceNet interface for the KUKA Y-cutting system. Individual analog and digital modules provide either input or output capability. Two Allen-Bradley Communication Interface modules connect the I/O modules to the main DeviceNet trunk cable, which terminates at the PCI CAN bus scanner card. One interface module is located inside the robot controller, and provides I/O connections to the user console and the Y-cut tool. The second interface module is contained in a separate stainless steel enclosure that sits above the conveyor rail, and connects the main Y-cut sensing devices.

3.3.4.2 Brisket Sensor

The dressing system employed at Goulburn is slightly different to that used in New Zealand meat-plants. Prior to Y-cutting, a butcher clears a strip of skin from the brisket region between the legs, and makes a cut down the neck to join up with the initial neck opening. The manual Y-cut begins where the cleared area of brisket intersects the shoulder, and then proceeds up the leg. The automated version will also have to exit into this brisket region.

A tactile wand cannot measure the height of the carcass in Goulburn because the brisket is exposed prior to Y-cutting. This means that a wand that brushes across the pelt and onto the brisket could contaminate the meat. However, the lack of wool on the brisket means that non-contact sensing is a viable alternative.

A Leuze Optical Distance Sensor measures the height of the brisket. This sensor is a laser triangulation device, with the following specifications:

- Measurement range: 200 – 2000 mm
- Resolution: $\leq$5 mm

- Accuracy: ±2% of measured distance
- Measurement frequency: 10 – 100 Hz
- Output: 1 – 10 V (linear with distance)

A stainless steel enclosure houses the sensor, with a glass window for the laser to pass through. This laser enclosure sits above the carcass at the height of the conveyor rail. The orientation angle of the laser is adjustable so that it best measures the height of the brisket. The sensor output connects to an analog I/O module in the separate DeviceNet enclosure above the conveyor rail.

3.3.4.3 Leg Wand

Because the Goulburn Y-cut does not end at the neck opening, there is no need for a horizontally rotating neck wand. However, a tactile wand at the base of the leg allows for the accurate determination of the end of the path.

The leg wand lies in the path of the forelegs of the moving carcass (refer Figure 3.16). The height of the wand is set so that it contacts the leg between the knee and the shoulder. The wand will be closer to the shoulder on smaller carcasses, while on larger animals it will be closer to the knee. The wand deflects as the leg brushes past, and a rotary encoder detects the angular rotation of the arm. The maximum rotation of the wand occurs when the nearest point on the leg is contacting the wand arm. This maximum rotation allows the calculation of the Cartesian distance to the leg. The wand arm returns into the path of the next leg using a pneumatic cylinder.

The redesigned leg wand incorporates a larger DeviceNet-compatible encoder and improved environmental sealing. The selected Allen-Bradley 842D DeviceNet absolute encoder has 26-bit multi-turn resolution, giving 8192 steps per revolution for 8192 revolutions. Each step is equivalent to 0.044°. With a nominal wand length of 400 mm, the maximum measurement resolution is 0.3 mm.

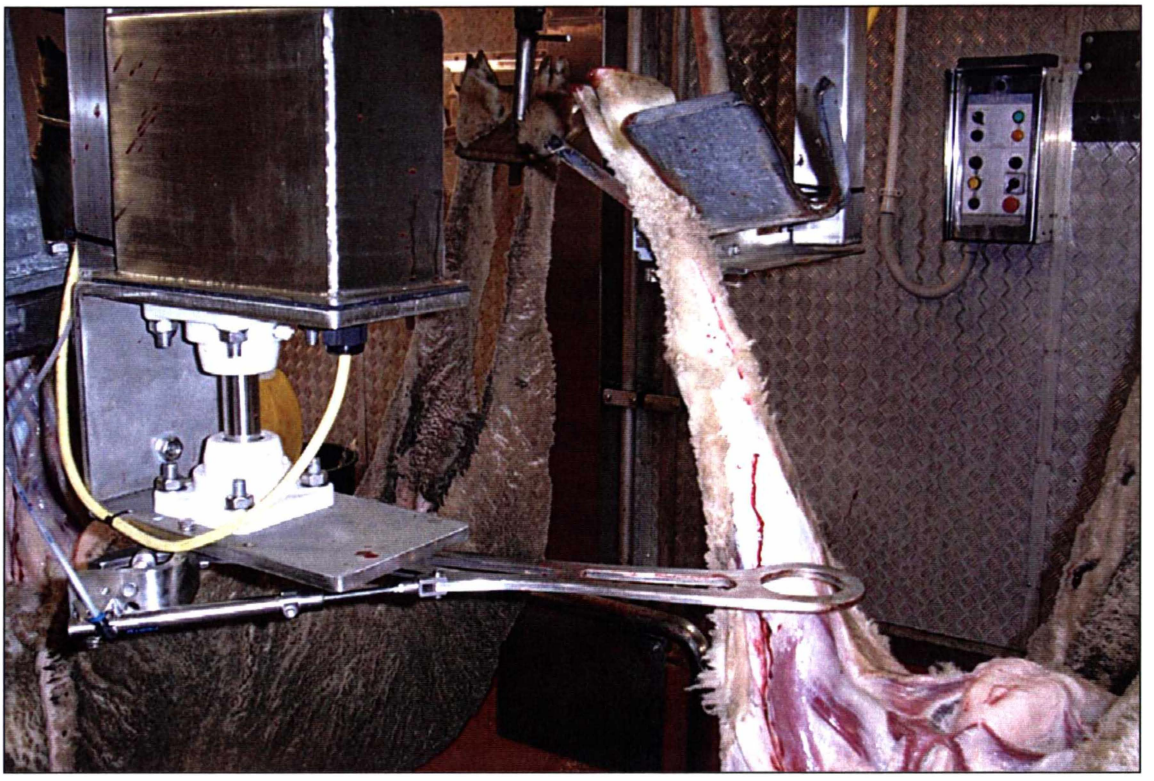

FIGURE 3.16 The redesigned leg wand (note that this leg has already been manually Y-cut).

3.3.5 Software Environment

Control of the robot uses a high-level, PASCAL-based KUKA Robot Language (KRL). The Windows operating system handles the graphical user interface, and the creation and editing of robot programs. VxWorks executes programs and manages the operation of the robot. Communication between the two operating systems is via TCP/IP protocol. A KUKA interface provides the user with a file manager, program editor, status and message windows, and a number of robot functionality keys. The interface can also display program variables, and allows the user to monitor and set I/O states.

When the robot controller is executing a program, an auxiliary "Submit" program is also running. The Submit program handles certain error conditions, and can control peripheral equipment. The Submit program gives the appearance of executing in parallel with the main program, although KUKA only guarantee the execution of the Submit control loop every 50 ms.

3.3.6 Y-Cut Program Structure

The automated Y-cut system uses two concurrently operating programs: one program contains the robot motion instructions, and the other gathers the measurement data from the sensing system. This demarcation allows a carcass to be Y-cut by the main robot program, while the sensing program measures another carcass at the same time. A separate subprogram calculates the Y-cut path based on the measurement data pertaining to the current carcass.

The Submit control loop calls the measurement program, with interrupt routines capturing sensor values and leg-switch events. The interrupts are polled at 12 ms intervals, which is the interpolation or control cycle of the KUKA controller. This interpolation cycle-time limits the accuracy of the sensor measurements. If the chain is moving at 10.2 ccs/min (the maximum speed of the conveyor at Goulburn), then in 12 ms the conveyor only moves 1.9 mm. It is expected that the Y-cut process can tolerate this level of inaccuracy.

3.3.7 Conveyor Tracking

KUKA produce a ConveyorTech module to allow the robot to track a moving conveyor. This system incorporates a synchronisation and counter module that interfaces with the DeviceNet bus, and a software package to facilitate the development of conveyor-based applications. Each hardware module has two separate counters, allowing the tracking of two different conveyors. The software coordinates the motion of the robot so that the tool centre-point (TCP) of the robot moves with the conveyor.

A Hertzler R159 incremental encoder (10000 counts per revolution) is the input for the counter module. This encoder has an IP67 rating, and can tolerate significant axial loading. A conveyor chain idler wheel drives the encoder, which is approximately 6 m upstream of the Y-cutting location. Each encoder count equates to 0.012 mm of conveyor travel.

A typical robotic application will have only one object moving through the workspace at any given time, meaning that a single encoder stream is sufficient to track consecutive objects. Usually a switch triggering off an object on the conveyor provides a synchronisation signal. This signal allows the object to be located relative to the conveyor, and sets the ConveyorTech counter to zero. However, a single ConveyorTech module cannot track multiple objects because the counter is referenced for each object, destroying data pertaining to other objects. The Y-cut process requires the tracking of multiple objects, because more than one leg can be within the workspace at any given time. Physical workspace limitations can necessitate the positioning of the sensors some distance from the workspace of the robot, requiring the tracking of intervening legs.

To avoid this problem, the chain encoder signal is fed into both ConveyorTech counter modules (refer Figure 3.17). One counter generates a continuous encoder stream, which determines the position of each passing leg. A single switch placed in the path of the legs provides the necessary trigger signal to capture the current chain encoder value. The measurement program completes the capture of the encoder value, which is stored in a global data file. The ConveyorTech software uses the second counter to coordinate the motion of the TCP. A software signal sent from the Y-cut program synchronises this counter. The robot sends this signal when it is ready to cut the current leg. The ConveyorTech software offsets the frame-of-reference of the robot by the distance that the current leg has moved since triggering the leg switch, allowing the robot to determine the location of the leg.

The leg switch used in the KUKA Y-cutting system is an E50 Cutler-Hammer limit switch. This switch has an IP67 rating and does not require additional environmental protection. It has a specified electrical life of 1,000,000 operations, which is equivalent to approximately half a year of operation in the Goulburn meat-plant. The switch connects to the DeviceNet modules in the separate enclosure above the conveyor rail.

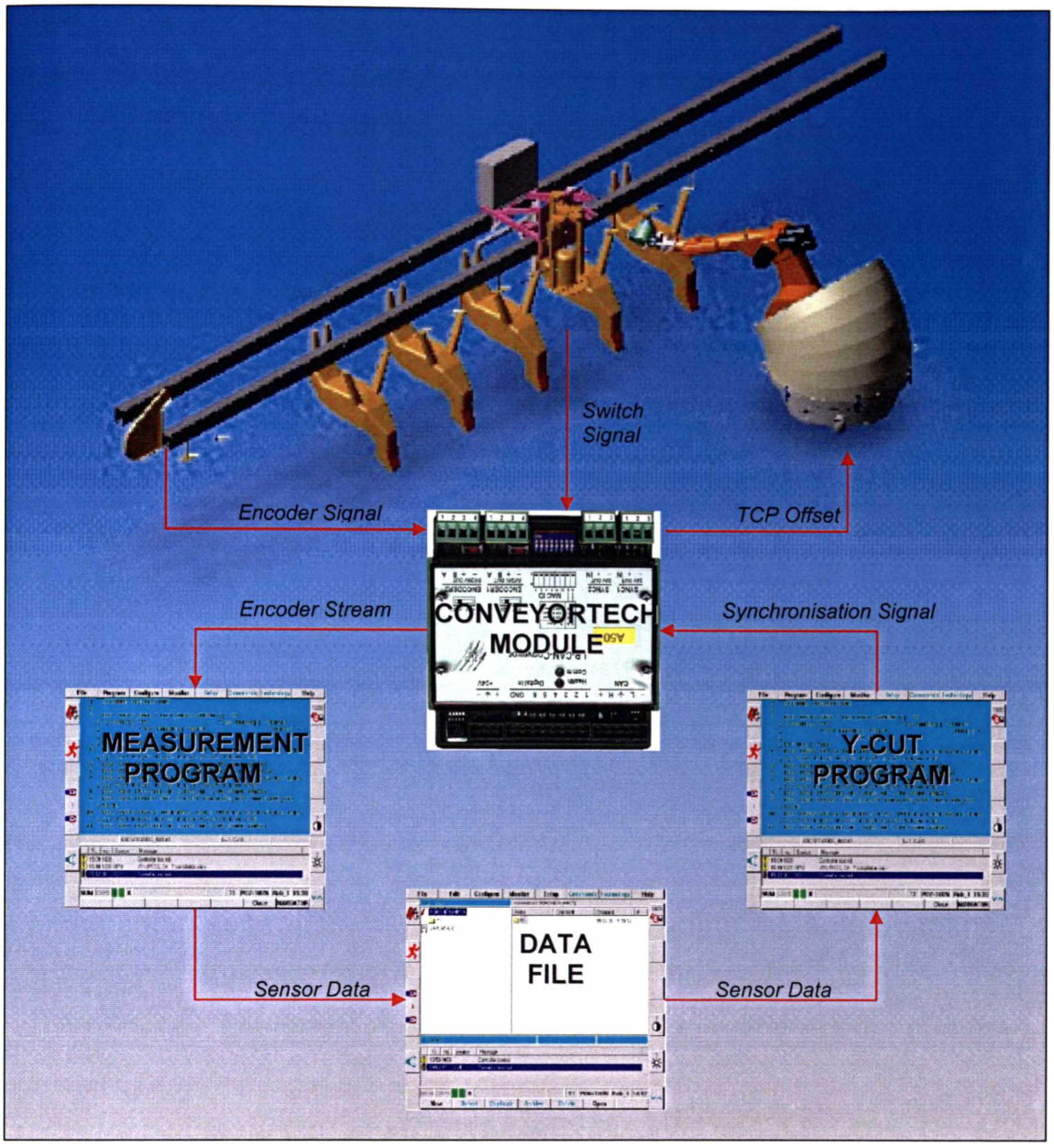

FIGURE 3.17 Y-cutting conveyor synchronisation.

3.3.8 Improvements from the IRL8L System

The KUKA Y-cutting system has significant advantages over the previous IRL8L system. The KUKA robot is mechanically robust and reliable, with a proven record in a multitude of industrial automation applications. There are established networks of trained technicians in both New Zealand and Australia that can repair and service KUKA robots. This is not the case with the IRL8L robot, with expertise limited to a few people who were involved in the initial development of the robot.

The KUKA offers a more advanced user interface than the IRL8L. The KUKA is easy to program, with most control and motion commands being accessible via a menu-driven interface. The Windows interface of the KUKA can be easily reconfigured to display different system parameters or I/O data. There are also a number of built-in diagnostic tools. The IRL8L robot is controlled with a lower level object-oriented architecture that requires the user to have an intimate knowledge of the program structure. The IRL8L uses a DOS-based operating system with a comparatively primitive interface that displays limited information and is difficult to modify.

Another improvement is the user console of the KUKA system, which is completely external of the main robot controller. The IRL8L system is controlled via a standard computer keyboard and LCD monitor connected to the PC inside the controller enclosure. However, there are no external connections for these peripheral devices, meaning that the door of the controller has to remain open whenever the system is being reprogrammed. This means that there is a significant risk of water or other contaminants entering the controller and damaging the electronic hardware of the IRL8L. The simple connectivity of the DeviceNet system means that additional sensors and I/O modules can be easily incorporated into the KUKA system. The IRL8L system is more restricted, with the PC interface cards of the controller offering limited I/O resources and expansion capacity.

3.4 PATH-PLANNING ALGORITHM

An integral part of the KUKA Y-cut process is the determination of the cut-path for each animal. Path-planning algorithms used in previous installations have applied offsets to waypoints along a straight line. The Goulburn Y-cutting system employs a similar strategy.

3.4.1 Definition of Coordinate System

Figure 3.18 defines the coordinate system for the conveyor. The robot uses this coordinate system when it is Y-cutting a carcass. The X-axis is in the direction of conveyor travel, to meet the requirements of the ConveyorTech software. The Y-axis

points horizontally away from the robot, and the Z-axis vertically upwards. The origin of the coordinate system is located at the lower front edge of the conveyor rail, at a point perpendicular to the base of the robot.

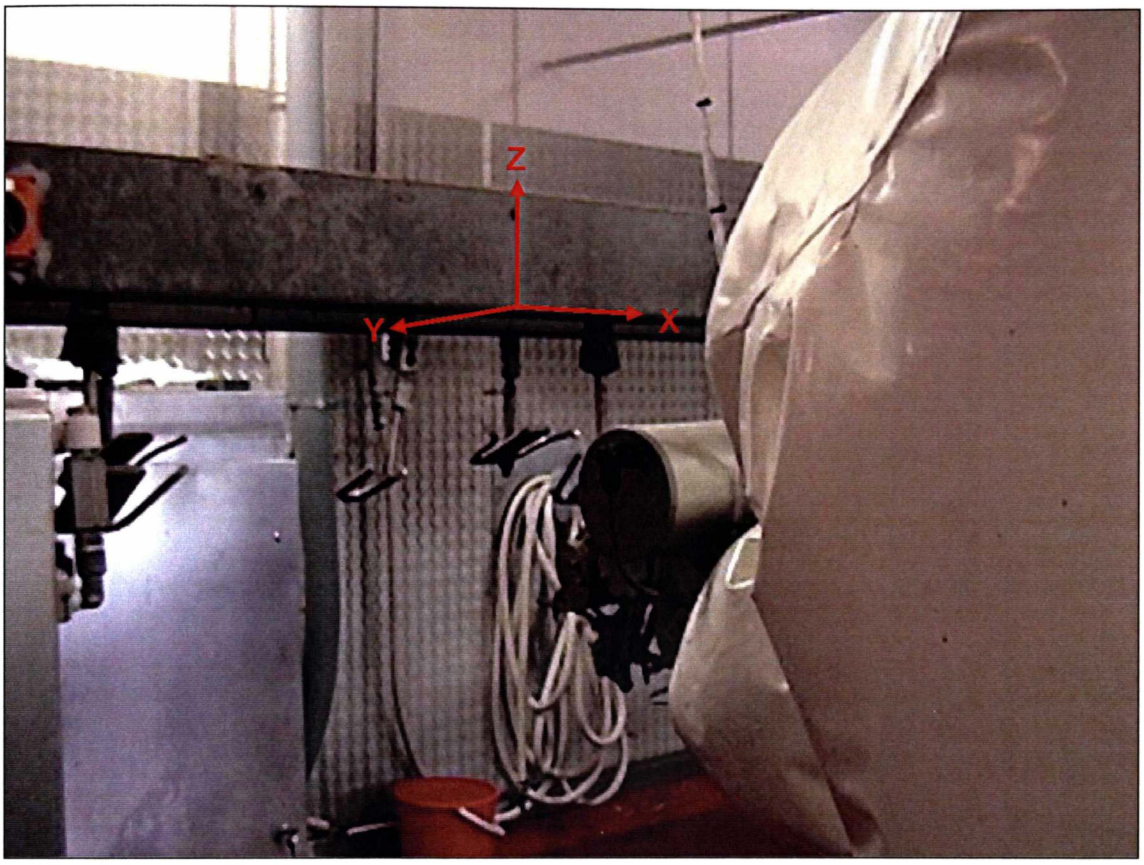

FIGURE 3.18 The conveyor coordinate system, as seen from the point-of-view of the robot.

3.4.2 Description of Path-Planning Algorithm

Four key sensor measurements are obtained for each carcass (refer Figure 3.19a). They are:

- *Leg_Switch_Pos*: the leading edge of the leg at the height of the sock-ringing cut (X-direction)
- *Brisket_Measure*: the height of the brisket (Z-direction)
- *Wand_Measure*: the depth measurement from the leg wand (Y-direction)
- *Wand_Measure_Pos*: the lateral measurement from the leg wand (X-direction)

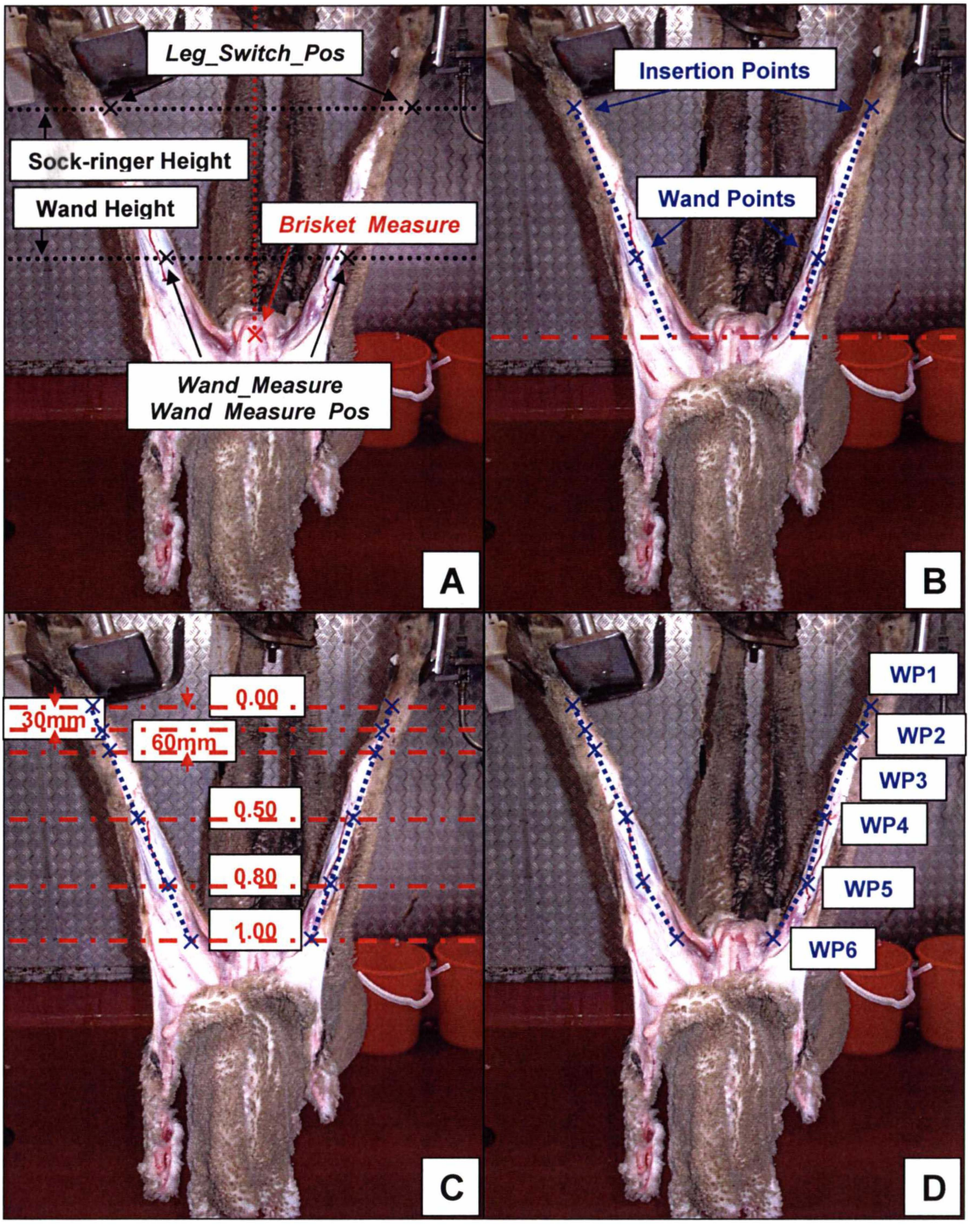

FIGURE 3.19 The Y-cut path-planning algorithm: a) sensor measurements, b) baseline projection, c) waypoint height definition, d) inclusion of offsets to give final path.

The baseline of the path is defined by the location of the insertion point and the measured wand-point (refer Figure 3.19b). The Cartesian coordinates (X, Y, Z) of the insertion point are given by (*Leg_Switch_Pos*+*IPX, IPY, IPZ*), where *IPX*, *IPY* and *IPZ* are constants. The offset *IPX* shifts the insertion point towards the centre of the foreleg. The constant *IPY* is the depth of the leg, while *IPZ* is the height of the sock-ringing cut.

The coordinates of the wand-point are (*Wand_Measure_Pos+Wand_X*, *Wand_Measure+Wand_Y*, *Wand_Height*), with *Wand_X* and *Wand_Y* being constant offsets. *Wand_Height* is equal to the height of the leg wand. The path baseline projects to a height given by *Brisket_Measure+Brisket_Measure_Offset*. The constant *Brisket_Measure_Offset* allows the tuning of the overall length of the path so that it can finish above or below the height of the brisket.

Six waypoints define the Y-cut path. These waypoints are distributed down the baseline so that they roughly correspond to physical features of the leg (refer Table 3.1). Waypoints WP1 and WP6 are at either end of the baseline, and are the insertion and end-points of the Y-cut. Waypoint WP2 is critical for successful insertion, ensuring that the tool hooks into the pelt. The main function of WP3 is to provide sufficient pullback on the pelt so that the tool avoids the main knee ligaments. Waypoints WP4 and WP5 provide the flexibility to change the shape of the cut-path depending on particular plant requirements.

TABLE 3.1 Y-cut path waypoints, and the corresponding physical features and heights.

Waypoint	*Leg Feature*	*Height (ratio)*	*Height (mm)*
WP1	Insertion	0.00	-
WP2	Hock	-	30
WP3	Knee	-	60
WP4	Leg	0.50	-
WP5	Shoulder	0.80	-
WP6	Brisket	1.00	-

All six waypoints had been located at a set proportion of the cut height during previous New Zealand Y-cutting trials. However, it was found that the height of WP2 that resulted in successful insertion was not dependent on the brisket height, but on the distance from the insertion point. Hence, the current algorithm has WP2 a constant height below WP1.

Previous testing shows that animals with the same brisket measurement have knees that are located at a range of heights (approximately ±30 mm). This poor correlation between knee height and brisket measurement means that it is unreliable to make the height of WP3 dependent on the overall cut height. It is better for the tool to pull back

early to avoid the knee, rather than pulling back too late and snagging on a knee ligament. Thus WP3 is set at a constant height below WP2 (and hence a constant height below WP1). WP2 and WP3 are located 30 mm and 60 mm below WP1. WP4 and WP5 are located at a set proportion of the overall cut height (0.50 and 0.80 respectively) (refer Figure 3.19c). The final cut-path results by offsetting each waypoint in both the X and Y directions. This produces a cut-path that best matches the contour of the leg and the requirements of the plant (Figure 3.19d). The tuning of these offsets currently requires an iterative manual process.

3.4.3 Tool Orientation

Along with the Cartesian position of the waypoints, the tool orientation angle at each waypoint needs setting. The KUKA system uses Euler angles A, B and C (about axes Z, Y and X respectively), which correspond to the yaw-pitch-roll angles that are commonly used in aviation (refer Figure 3.20).

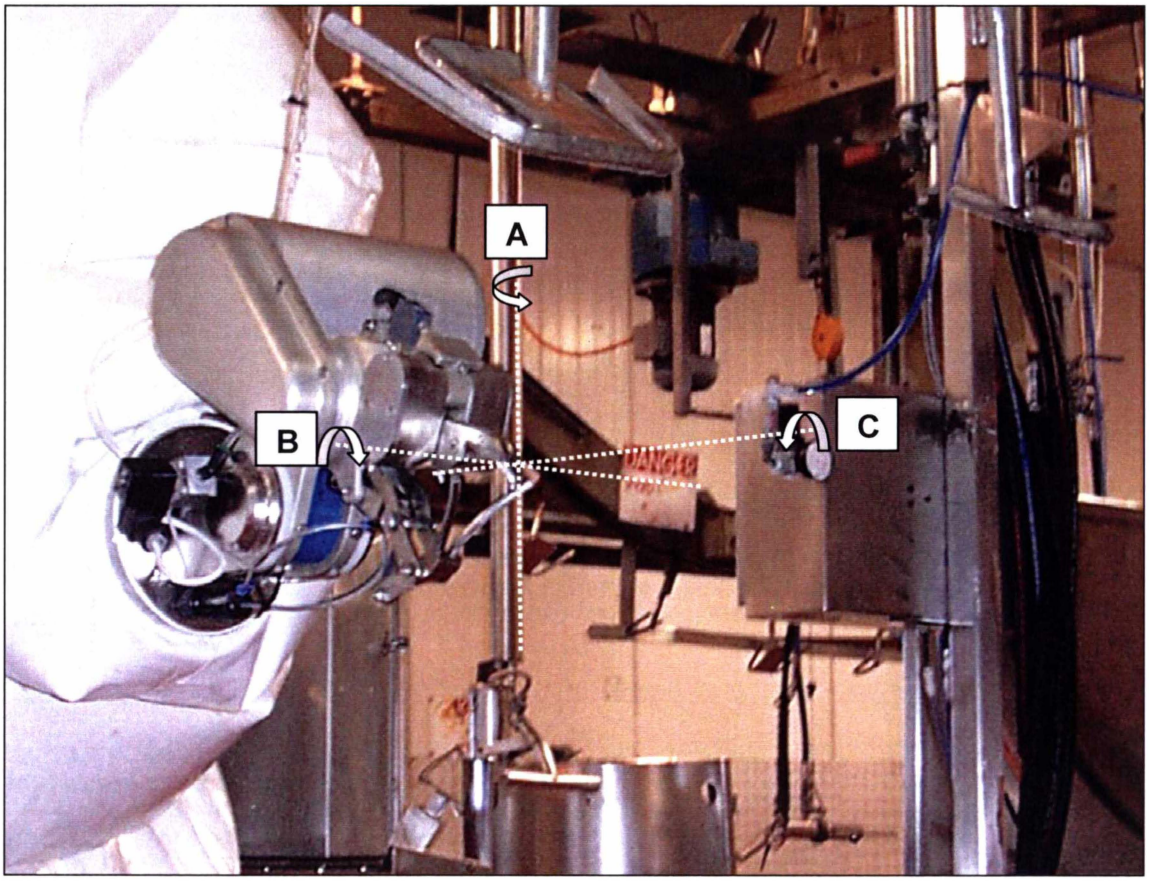

FIGURE 3.20 Tool orientation angles.

However, the coordinate system of the tool has a different orientation to that of the conveyor, and a simple transformation converts between the two. The waypoint orientation angle A is set to 90° to align the two coordinate systems. This yaw angle results in the tool being normal to the conveyor and the leg. Angles B and C then correspond with the pitch and roll of the tool. When B = 0° the tool is horizontal, when C = 0° the tool is vertical. The orientation angles at each waypoint are tuned using an iterative manual process.

3.4.4 Tuning of the Cut-Path

The most time-consuming part of commissioning an automated Y-cutting system is the tuning of the cut-path offsets and angles. Current optimisation of these path parameters uses an iterative process (refer Section 2.4.1), requiring at least one person to be observing the robot and modifying the cut-path continuously over a period of weeks or months. This time and labour requirement is a potential barrier to the uptake of the Y-cutting system. Therefore, there is a real need for some form of automated path optimisation. However, this form of optimisation requires the outcome of each cut to be determined automatically. This determination can be provided by a fault detection system.

4. FAULT DETECTION FOR AUTOMATED Y-CUTTING

While previous automated Y-cutting installations have demonstrated success rates of over 98%, this level of performance has only been achieved after months of continual tuning and optimisation of the cut-path. With the development of a prototype commercial Y-cutting system, there is now a real need to speed up the tuning and path optimisation process. The provision of an automatic fault detection system is the first step towards achieving this objective. Prior to the work described in this chapter, there had been no attempt to detect Y-cutting process faults automatically.

4.1 PROCESS FAULT DESCRIPTIONS

During previous testing and installation work, the automated Y-cutting system had to be adapted and tuned to meet the requirements of each specific plant. This tuning process required the correction of a number of key process faults. They are:

1. Insertion failure.
2. End-point failure.
3. Incorrect cut-path.
4. Meat or membrane damage.
5. Robot overload.

4.1.1 Insertion Failure

Insertion failure occurs when the Y-cut tool does not hook into the opening cut produced by the sock-ringer. If the tool fails to insert then the automated Y-cut cannot commence. Causes of insertion failure include:

- The sock-ringer imparts insufficient cutting force, producing an opening cut that is too narrow for the tool.
- Some breeds of sheep have particularly woolly legs, and although the sock-ringer may have created an adequate cut in the pelt, the surrounding wool can obscure the opening cut and prevent the tool from inserting.
- The leg drops or falls out of the chain spreader prior to reaching the robot.
- The leg-guide fails to capture the leg or prevents the tool from reaching the leg.
- The profile of the tool-tip is too blunt.
- The presentation angle of the tool is incorrect.
- The position of waypoint WP1 relative to the opening cut is not optimal.
- The path from WP1 to WP2 does not hook the pelt onto the feeder-teeth of the tool.

4.1.2 End-Point Failure

The Y-cut tool is supposed to finish the Y-cut in a specified location. The traditional New Zealand Y-cut has an end-point target at the top of the neck opening, while in Goulburn the end-point is the cleared area of brisket between the forelegs. End-point failure occurs if the tool pulls out of the pelt before reaching the target region. It is important that the cuts terminate at the same point on all carcasses, as down-stream processes are dependent on this reliability. The causes of an incorrect cut end-point are:

- There is excessive tool pullback and pelt tension, resulting in the tool ripping out of the pelt prematurely.
- The definition of the cut-path end-point is incorrect.

4.1.3 Incorrect Cut-Path

The definition of an incorrect cut-path is one that produces a Y-cut that does not meet the requirements of the plant. Most plants require the cut to be down the centre of the leg, although there can be variations to this depending on downstream processes. Causes of this fault include:

- The hanging of the legs from the conveyor spreaders is inconsistent, with the hooves orientated in a variety of directions. This inconsistency means that the articulation of the knees and shoulders is unreliable, and the shape of the legs can vary from one animal to the next.
- The geometry of the legs of the sheep is highly abnormal.
- The waypoint offsets have been tuned incorrectly.

4.1.4 Meat or Membrane Damage

Damage to the meat or membrane layer beneath the pelt is unacceptable since it affects the quality of the carcass and increases the risk of microbial contamination. Causes of this form of damage are:

- The tool-tip profile is too sharp and rips into the meat instead of sliding between the membrane and pelt.
- The orientation of the tool-tip is not parallel to the leg.
- The tool catches on a knee ligament, and rips through the knee joint and into the leg below.
- The cut-path imparts excessive force on the leg.

4.1.5 Robot Overload

An overload condition in the robot occurs when one of the axis drive units exceeds a current limit and results in the halting of the executed robot program. This condition is usually the result of the tool catching in the knee joint. It can also occur if a programmed motion will exceed individual axis acceleration or torque limits. The causes of an overload are:

- The cut-path is incorrect, bringing the tool too close to the knee.
- The programmed motion is beyond the capabilities of the robot.

4.1.6 Towards Automatic Fault Detection

Many of the causes of these faults can be easily rectified; mechanical adjustments or improvements can be made to the sock-ringer, leg-guide and tool-tip profile. The remaining causes all relate to the path of the tool in relation to the carcass, and can be rectified by further tuning of the path waypoint offsets. The manual tuning of these offsets can be a laborious task. It requires an operator to conduct numerous trials to establish the effect that an offset change has on the success-rate and fault incidence. The majority of the tuning effort focuses around increasing the insertion success-rate, as the Y-cut cannot be attempted if the tool does not insert. An automatic path tuning and optimisation system would be of great benefit, particularly if it helped to reduce the tuning time.

The highlighted process faults must be reliably detected prior to attempting to develop such an optimisation system. An obvious symptom for most of these faults is a variation in the load experienced by the robot or tool motors. The load is expected to increase if insertion is successful, and should decrease once the tool has completed cutting. If the membrane is penetrated and the underlying meat is cut, then there might be an observable variation in the process loads. A robot overload condition should also be easily detectable.

An incorrect cut-path is the only fault that is not expected to be detectable from process load variations. Correction of this fault does not require significant process tuning. Waypoints WP4, WP5 and WP6 principally determine the shape of the cut-path. The offsets for these waypoints can be easily changed to produce a cut-path with a different shape. The other cause of an incorrect cut-path is inconsistent hang-up of the forelegs, which can be negated via effective supervision of workers conducting this task.

4.2 ANALYSIS FROM INITIAL TOOL TESTING

The development of a new Y-cut tool was part of the full commercialisation of the automated Y-cutting system. The main feature of the new design was the replacement of the air motor with an electric DC motor and separate PWM drive. The new design

also included other mechanical changes to improve the reliability and serviceability of the tool. A temporary Y-cut installation at Auckland Meat Processors (AMP) provided a test-bed for the new tool design. The installation used an older IRL8L robot and controller (refer Figure 4.1). The tool PWM drive unit is housed in a separate enclosure that sits on top of the IRL8L controller.

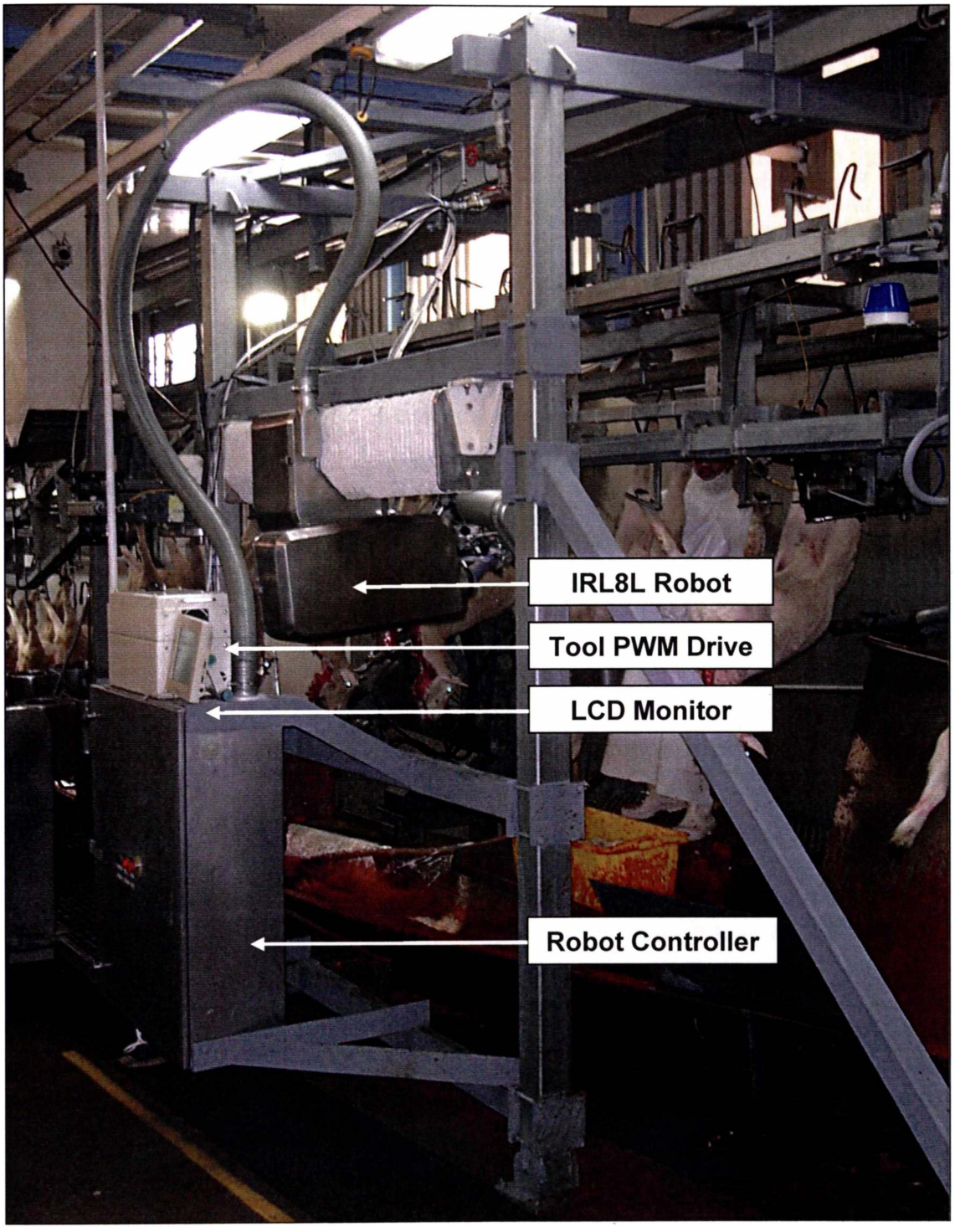

FIGURE 4.1 The IRL8L robot at AMP.

4.2.1 Robot Data Acquisition

The IRL8L controller uses Pacific Scientific SC902 and SC903 servo-drives to power the four robot axes. These drives each have two digital-to-analog converter (DAC) outputs that allow the monitoring of certain internal parameters, such as velocity, current and temperature. Servo-programming software maps each DAC output to a particular parameter. Specifications of the DAC outputs are:

- ±5 V output range
- 8-bit resolution (0.039 V)
- 400 Hz update rate
- 10 kHz low-pass filtered output

To determine the load experienced by each axis during the Y-cut operation, the "Measured Torque Current" was selected as the DAC output on all four servo-drives. This torque current is the current supplied to the axis motor. The conversion from DAC output to torque current is 1 V = 1 A, giving an output range of ±5 A.

A National Instruments PC-516 DAQ card recorded the DAC signals. Specifications of this DAQ card include:

- 16-bit successive-approximation ADC (analog-to-digital converter)
- 8 single-ended or 4 differential inputs
- 4 digital inputs + 4 digital outputs
- 50 kHz sampling rate
- PCMCIA card format

The DAQ card was installed in a splash-proof laptop that was capable of operating in the humid meat-plant environment. The inputs were configured as differential inputs, recording the difference between the servo DAC outputs and the signal ground. The use of differential inputs reduced the amount of common-mode noise present in the signal. An internal battery powered the laptop during data capture, reducing the noise conducted via ground loops.

A LabVIEW program captured the signal data. The program captured data from the DAQ card with a sampling rate of 1 kHz. This sampling rate is greater than the 400 Hz update rate of the servo drive DAC outputs. A video camera simultaneously recorded each Y-cut. This visual record allows the captured load data and the relating Y-cut to be synchronised during subsequent analyses. The time that a fault occurs can then be established and related back to the load data.

The axis positions of the robot were logged to a data-file while the Y-cut was being attempted. The positional data were captured approximately nine times per second within a control-loop of the main Y-cut program. These data can be linked to the motor current signal to determine the load experienced by the robot at a particular spatial location.

4.2.2 Preliminary Signal Analysis

Axis motor current data for 32 Y-cuts were acquired. All cuts were performed on the leading foreleg of the carcasses. Eight of these cuts were unsuccessful because of insertion failure and another sixteen were end-point failures. The remaining eight cuts were successfully completed cuts. There was no incidence of damage to the carcass membrane or meat, although on several occasions the tool appeared to encounter resistance while traversing the knee.

Figure 4.2 gives the torque current data for each robot axis. A sharp increase in the shoulder axis torque current identifies the start of the downwards Y-cut motion, and provides a reference for the plotted data. The plots in Figure 4.2 commence from this reference point.

The shoulder axis data show the biggest difference between insertion failures, end-point failures and completed cuts, since the shoulder axis is responsible for most of the vertical motion of the Y-cut. This axis also experiences the majority of the process load. The traverse and probe axes show some minimal variations, while the wrist axis exhibits no discernable difference between the three classes of cut.

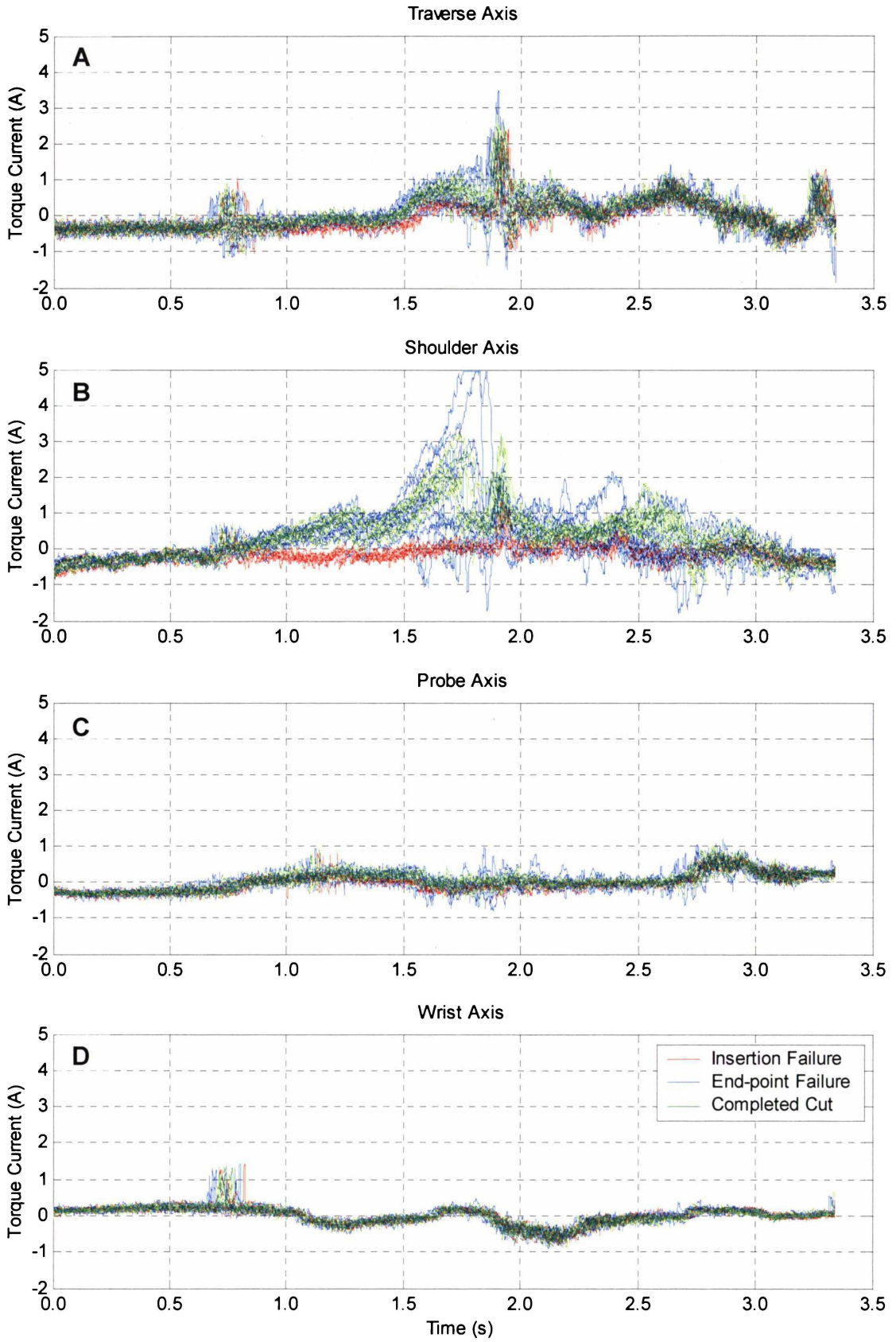

FIGURE 4.2 **Current data from the IRL8L robot: a) traverse axis, b) shoulder axis, c) probe axis, d) wrist axis.**

4.2.3 Identification of Key Features

After reviewing the video record of the cuts, features of the cut-path can be related to the load data from the shoulder axis (Figure 4.2b). The most obvious feature is the traversal of the knee, which occurs 1.5 – 2.0 seconds after the start of the cut. There is a large increase in the torque current during this interval – provided the cut is not an insertion failure.

The increase in torque current corresponds to the increase in the load as the shoulder axis cuts across the knee. The magnitude of this load appears to vary significantly between animals, with the maximum current ranging between 1.4 and 5.0 A for cuts that have successfully started. In two cases, the torque current saturated the DAC output at 5.0 A, although this is well short of the maximum permissible current of 15 A for the shoulder axis servo-drive. In both of these cases, the tool lodges in the knee temporarily before eventually driving over the joint.

Other cut-path features are identifiable from the torque current signals. Figure 4.3 plots the shoulder axis current for three different Y-cuts. Each plot includes a representative insertion failure, with labels for the different stages of the cut-path. Timings taken from the video of the process provide estimations of the cut end-point.

The first stage of the cut-path is the insertion of the tool into the opening cut. There is no discernable difference between an insertion failure and a successfully started Y-cut during this stage. There is initially minimal load applied to the tool, with the robot moving at a slow velocity. This helps to feed the pelt onto the blades, ensuring that the cut begins smoothly. The tool must fully insert into the pocket of the opening cut, resulting in a delay before the tool-tip experiences the process load.

If insertion has been successful, then the motor current increases during the pre-knee stage. This increase occurs as the cutting of the pelt begins, and the robot starts to speed up. If insertion is unsuccessful, the current remains constant.

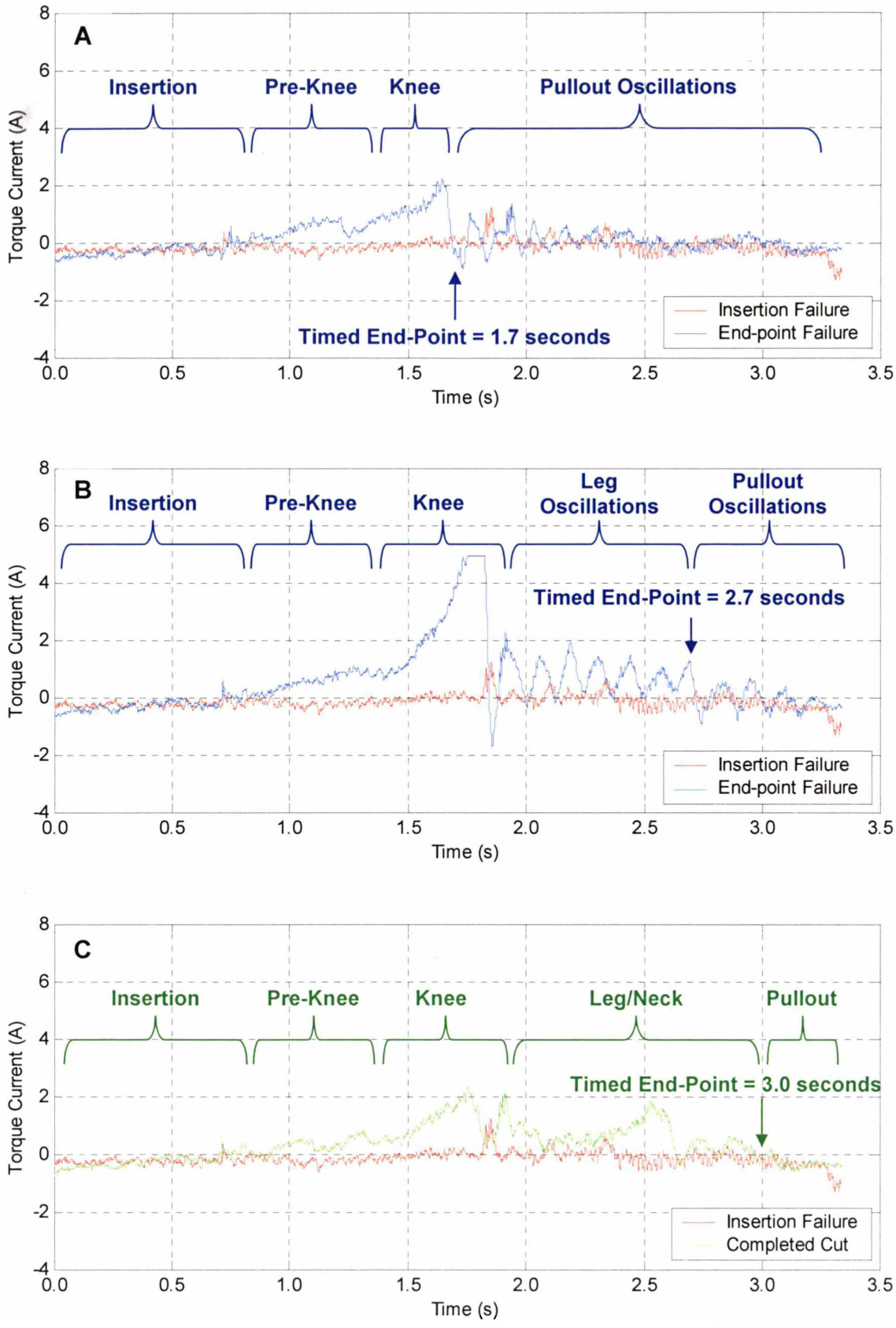

FIGURE 4.3 **Shoulder axis current for three successfully started Y-cuts: a) an end-point failure at the knee, b) an end-point failure down the leg, and c) a completed cut.**

When the tool reaches the knee, the load experienced by the robot increases further. The current differs significantly between Y-cuts, due to the variety of knee ligament sizes and pelt strengths.

If the tool pulls out of the pelt prematurely (resulting in an end-point failure), then the torque current eventually returns to the insertion failure level. Figure 4.3a shows a cut that ended just below the knee. The timed end-point of the cut corresponds to a rapid decay in the current signal. There is a small amount of oscillation in the signal during this pullout phase. If the tool has been exerting sufficient force on the pelt to pull the carcass towards the robot, then as the tool pulls through the pelt the load experienced by the axis motors instantly reduces. In order for the robot to maintain the programmed path and trajectory, the motion controller manages the load reduction via a PID control-loop, resulting in the observed oscillation of the current to the axis motors.

Figure 4.3b shows a cut where the robot experienced a very high load while traversing the knee. In this case the tool becomes wedged in the knee, before ripping through the joint and then on down the leg. There is significant oscillation present in this signal, both during the cut down the leg and then after the pullout of the tool. Note that the oscillations during the cutting of the leg were all above the insertion failure level. The timed end-point of the cut again corresponds with the signal dropping below the insertion failure level.

Figure 4.3c shows the motor current for a completed cut. The current is consistently above the insertion failure level during the pre-knee, knee and leg cutting stages, and then quickly decays following the pullout of the tool.

The signal for an insertion failure remains relatively constant throughout the cut-path at approximately 0 A, although there are some noticeable perturbations. The cause of these perturbations is the motion-controller of the IRL8L robot, with the disturbance occurring when the robot approaches a path waypoint. The controller reduces the speed of the axes as the robot approaches the waypoint, so that the robot can stop at the end of the current motion if necessary. There is also a slight acceleration of the axes as the robot starts to move towards the next waypoint. These two speed changes produce the observed perturbations. Figure 4.4 gives a typical insertion failure signal. The

locations of waypoints WP1 to WP7 have been determined from the captured positional data. The largest perturbation occurs as the robot moves through waypoint WP3, with the programmed cut-speed increasing from 35% to 65% of the maximum speed.

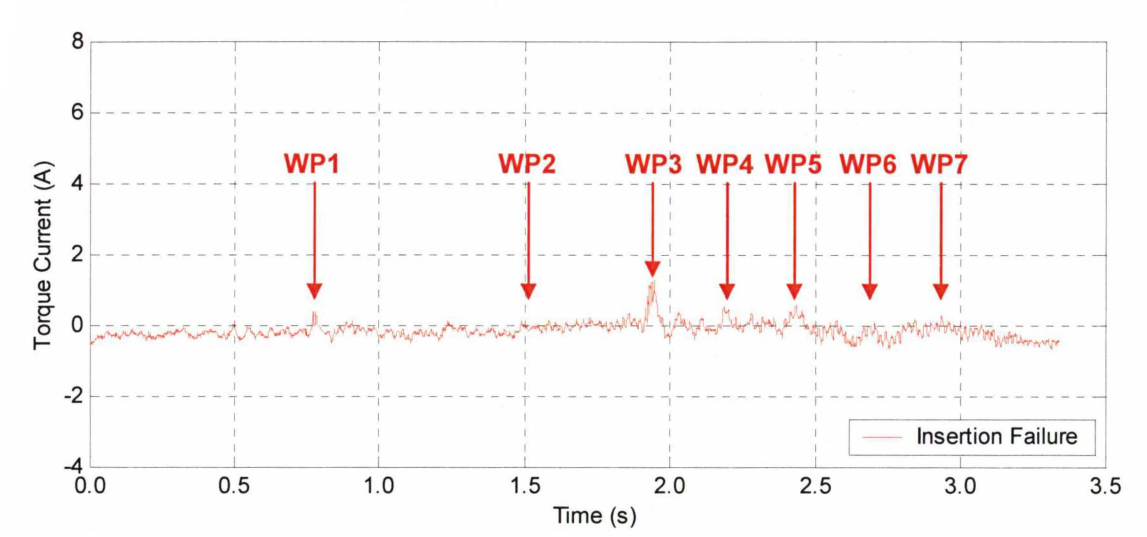

FIGURE 4.4 Shoulder axis torque current for an insertion failure, with cut-path waypoint perturbations.

4.3 KNOWLEDGE-BASED FAULT DETECTION

Initial observations of the IRL8L motor current data indicate that some general signal characteristics could be useful for the detection of specific process faults. These characteristics are:

- Insertion Failure
  - The shoulder axis current does not increase during the pre-knee stage of the cut and remains relatively constant throughout the remainder of the cut.
- End-Point Failure
  - The current increases during the pre-knee stage.
  - When the tool withdraws from the pelt, the current rapidly decays past the insertion failure signal before oscillating and settling at this level.
- Robot Overload
  - The current becomes increasingly large during the knee stage of the cut.

4.3.1 Initial Fault Detection Strategy

These signal characteristics form the basis of a knowledge-based fault detection strategy. The strategy applies a threshold level to the shoulder axis signal. The cut is deemed an insertion failure if the signal has not exceeded the threshold after a certain period of monitoring. The cut is said to have ended when the signal falls and remains below the threshold for a certain decay interval. If the cut end-point occurs before the robot has reached the end of the cut-path then an end-point failure has occurred, otherwise the cut has completed successfully.

This fault detection strategy was implemented using MATLAB. Initial testing determined the relevant threshold parameters. The best performance was achieved with a threshold level of 0.00 A, an initial delay of 1.00 seconds, and a decay interval of 0.04 seconds (refer Figure 4.5). The strategy was tested on the 32 captured Y-cut signals, with the results given in Table 4.1.

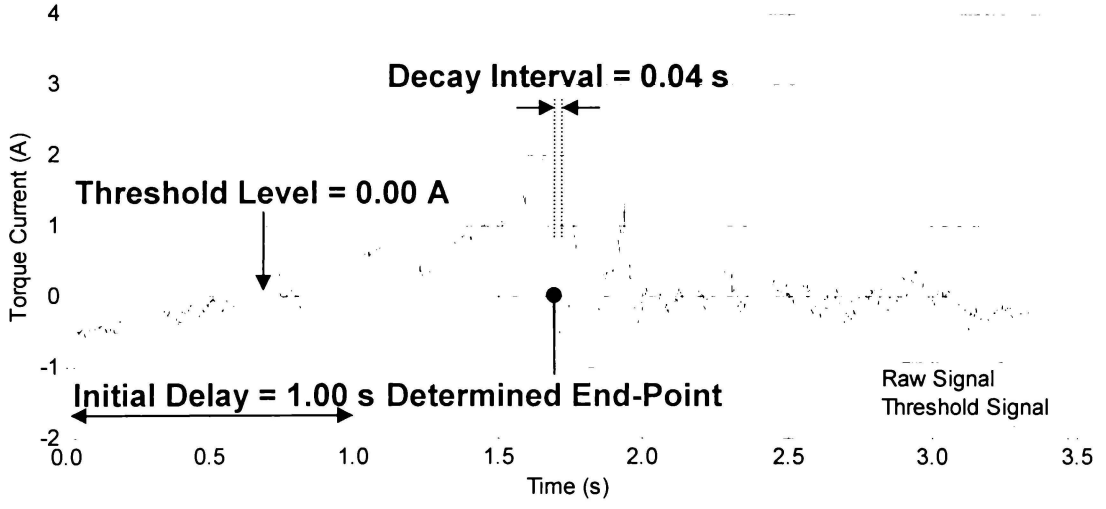

FIGURE 4.5 Initial knowledge-based fault detection strategy.

TABLE 4.1 Classification success of the initial knowledge-based fault detection strategy.

	Insertion Failures	*End-Point Failures*	*Completed Cuts*	*Overall*
Observed Cut Outcomes	8	16	8	32
Correctly Classified Cuts	8	14	3	25
Fault Detection Success Rate	100%	87.5%	37.5%	78.1%

The developed strategy performs remarkably well, with all eight insertion failures being successfully detected. The two incorrectly classified end-point failures are classed as completed cuts, and the five incorrectly classified completed cuts as end-point failures. Of the five incorrectly classified completed cuts, three are determined to have occurred within 5 mm of the end of the cut-path, and all five were within 30 mm of the end-point.

The end-point time determined by the fault detection system is compared to the observed end-point time for the 24 cuts that are deemed to have started. Figure 4.6 shows the distribution of end-point errors, which are defined as the difference between the observed end-point time and the determined end-point time. The standard deviation of this error distribution is 0.20 seconds. The mean absolute magnitude of the end-point errors is 0.17 seconds. The strategy detects 20 out of 24 end-points within 0.3 seconds of when they actually occurred.

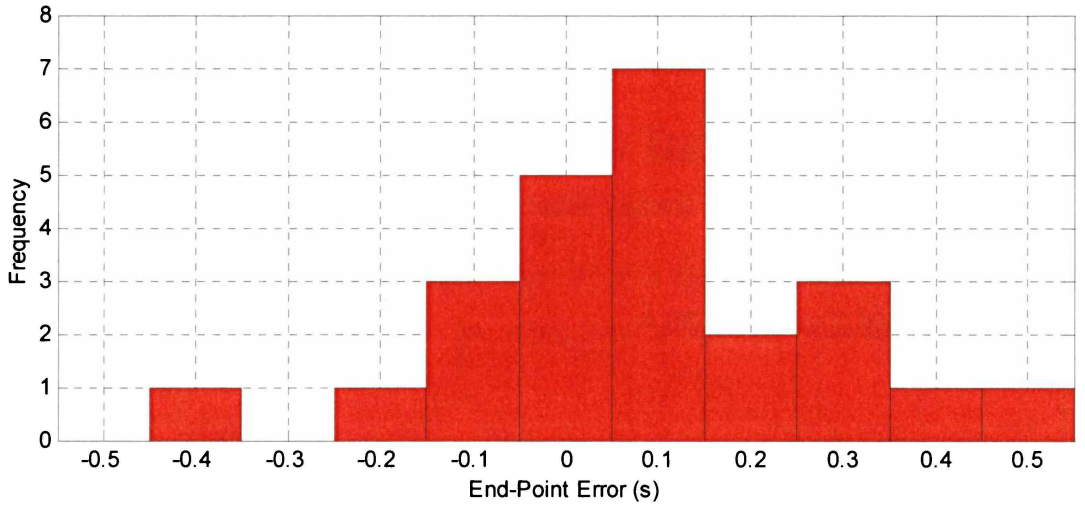

FIGURE 4.6 Distribution of end-point errors for initial fault detection strategy.

4.3.2 Modified Fault Detection Strategy

A second fault detection strategy is developed as an alternative to the initial strategy. This strategy again applies a threshold level to the shoulder axis signal, and examines the signal for regions that are above the threshold level, indicating that the robot axis is experiencing an external load. If the identified region exceeds some minimum width,

then the load is considered the result of the pelt being cut. The cut is said to have ended when the final such qualifying region is encountered.

The implementation of the strategy again uses MATLAB. The best performance is achieved with a threshold level of 0.00 A and a minimum width of 0.11 seconds (refer Figure 4.7). The width is selected so that oscillations during the pullout phase of the cut are ignored. Table 4.2 gives the classification success of this strategy.

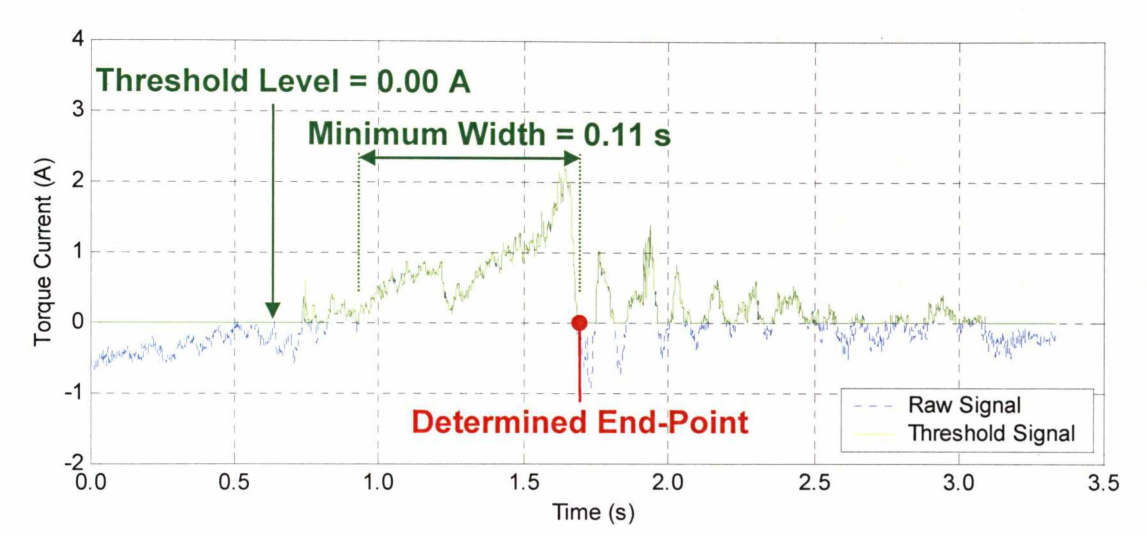

FIGURE 4.7 Modified knowledge-based fault detection strategy.

TABLE 4.2 Classification success of modified knowledge-based fault detection strategy.

	Insertion Failures	*End-Point Failures*	*Completed Cuts*	*Overall*
Observed Cut Outcomes	8	16	8	32
Correctly Classified Cuts	8	14	4	26
Fault Detection Success Rate	100%	87.5%	50.0%	81.3%

The classification performance of the modified strategy is slightly better than the initial strategy, with the correct classification of one additional completed cut. Of the four incorrectly classified completed cuts, all four determined end-points were within 20 mm of the actual end-point. Figure 4.8 shows the distribution of end-point errors. The modified strategy produces a much narrower error distribution when compared to the initial strategy, confirming that the former is producing a more reliable estimate of the cut end-point time. The standard deviation of this error distribution is 0.16 seconds.

The mean absolute magnitude of the end-point errors is 0.14 seconds. The strategy detects 23 out of 24 end-points within 0.3 seconds of when they actually occurred. All of these statistics indicate that the modified strategy is superior to the initial strategy.

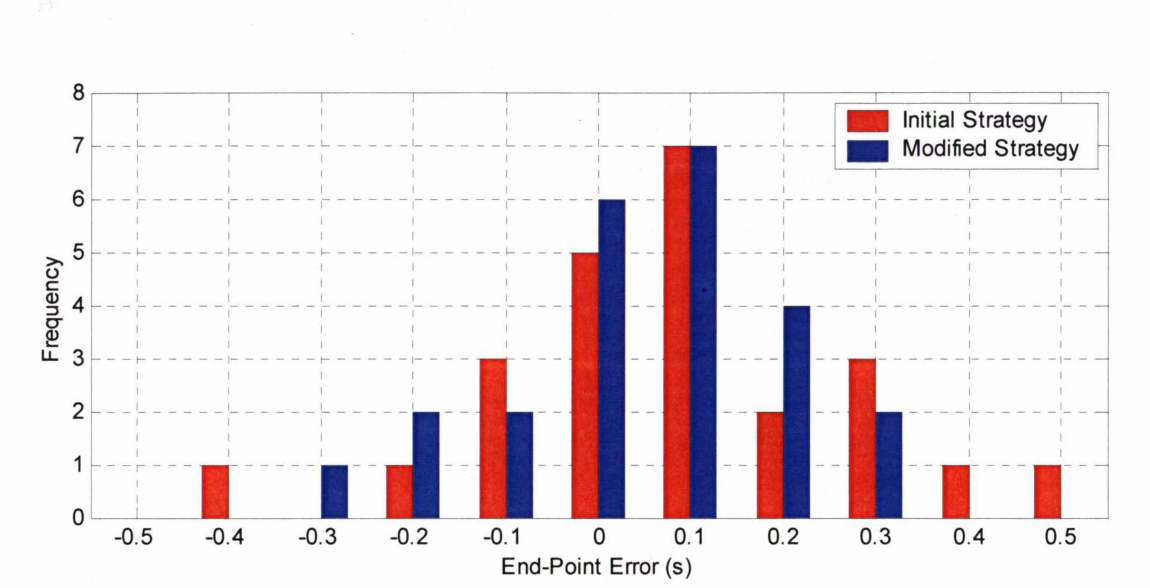

FIGURE 4.8 Comparison of the distributions of end-point errors for the initial and modified fault detection strategies.

4.4 NEURAL NETWORK FAULT DETECTION

Following the application of the two knowledge-based fault detection methods, the fault detection performance of different types of artificial neural network was investigated. Three specific varieties of neural network were tested; the multilayer perceptron (MLP), radial basis function (RBF) neural network, and learning vector quantisation (LVQ) neural network. These three networks were tested for their ability to determine the fault classification of a cut based on the input signal. They were also evaluated for their capacity to determine the cut end-point time.

4.4.1 Multilayer Perceptron Fault Detection

The multilayer perceptron has a parallel architecture, requiring the concurrent presentation of the input signal. The MLP must have an input neuron for each data-point in the signal. Network training uses a target output to represent the timed end-point of the cut. This target is a discrete signal (zero or one), with a falling edge when

the timed end-point occurs (refer Figure 4.9). The datasets are down-sampled to reduce the number of input and output neurons. To down-sample the data by a factor of n, every n^{th} data-point is selected. A down-sampling factor of 20 is used, effectively decreasing the sampling frequency from 1 kHz to 50 Hz and giving a Nyquist frequency of 25 Hz. Even with this level of data-reduction, the down-sampled signal is still an excellent representation of the original input signal (refer Figure 4.9). The majority of the datasets have 3340 data-points prior to down-sampling, resulting in networks with 167 input and 167 output neurons. Note that one dataset (an end-point failure) has less than 3340 samples, so is not suitable for presentation to a network of this size.

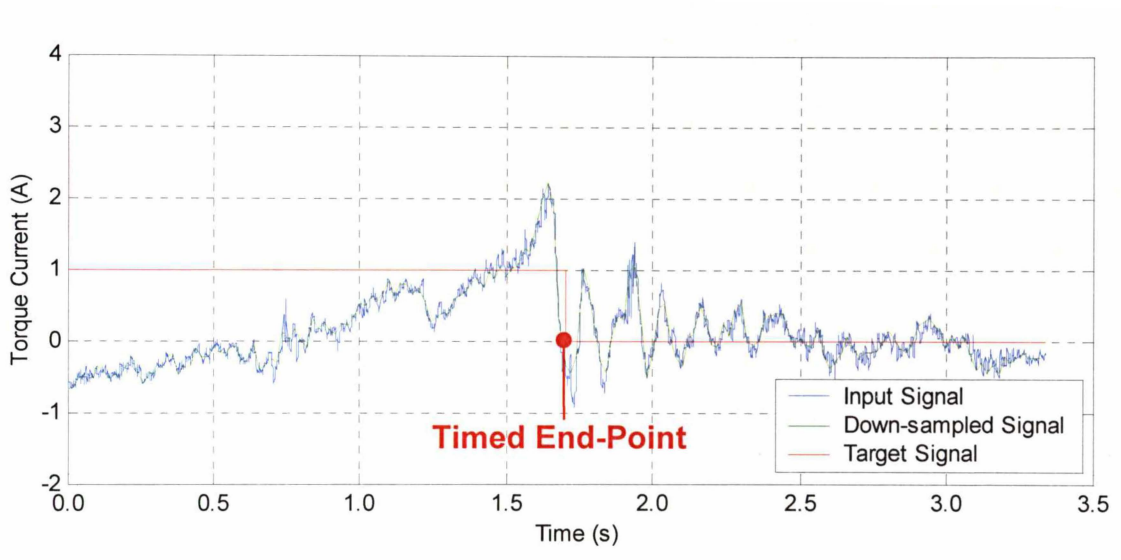

FIGURE 4.9 The target output for the MLP fault detection strategy.

Due to the limited amount of training data, a "leave-one-out" training strategy is used (Jain *et al.*, 2000). The network is trained with 30 of the datasets, and tested on the one remaining dataset. This process is repeated until all 31 datasets have been tested. The leave-one-out strategy has the advantage of producing an unbiased estimate of the network performance, whereas "holdout" methods that split the data into independent training and testing sets can be pessimistically biased if the number of datasets is small (Jain *et al.*, 2000).

The networks are developed using the Neural Network Toolbox of MATLAB. A two-layer structure is used, with a hidden layer of 50 neurons. The number of neurons in the hidden layer is selected following testing with various numbers of neurons to determine

the best performance. The hidden and output neurons use logarithmic sigmoid transfer functions (refer Section 2.3.1). Network training uses a resilient back-propagation algorithm (refer Section 2.3.2). The training for each network typically takes 35 epochs and stops when the minimum-squared-error (MSE) gradient is less than 1×10^{-6}. The final MSE is typically 1×10^{-7}.

The output of the MLP is generally a good representation of the target vector, although there can be some small perturbations present in the signal (refer Figure 4.10). To facilitate the determination of the cut end-point, a hard-limiting function is applied to the network outputs. A threshold of 0.5 is applied to the outputs, forcing the signal to zero if the output is less than the threshold, otherwise to one. The cut end-point is determined to be when this hard-limited signal stabilises at zero.

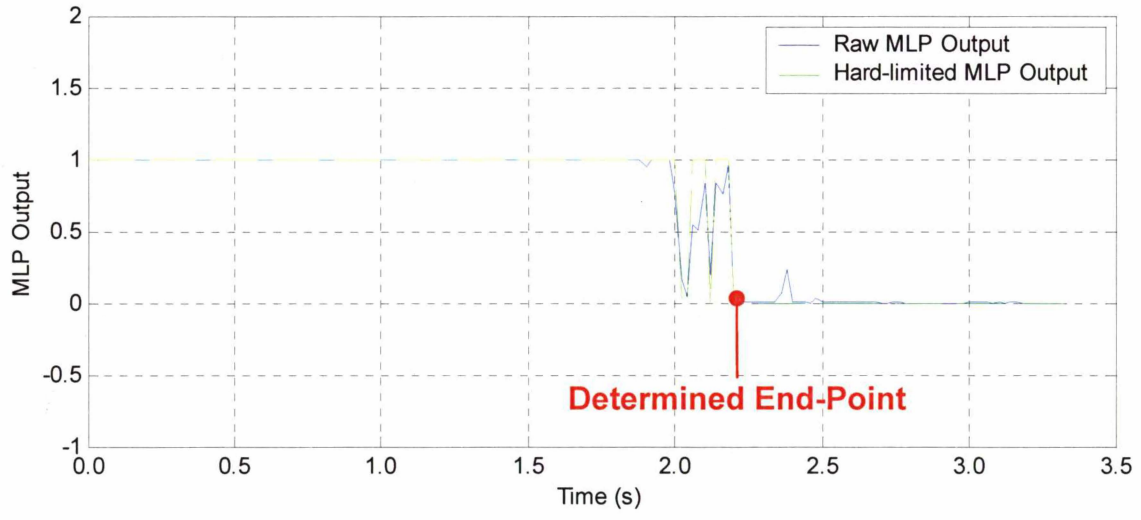

FIGURE 4.10 The raw and hard-limited MLP output.

The performance of the MLP is determined by averaging the results from 10 runs (refer Table 4.3). The strategy reliably detects insertion failures, but the classification success for completed cuts is particularly poor. Figure 4.11 shows the distribution of the end-point errors. The standard deviation of this error distribution is 0.36 seconds. The mean absolute magnitude of the end-point errors is 0.26 seconds. An average of 15.4 out of 23 end-points are detected within 0.3 seconds of when they actually occurred. All of these performance statistics are inferior to the knowledge-based techniques tested in the previous section.

TABLE 4.3 Classification success of the MLP fault detection strategy.

	Insertion Failures	*End-Point Failures*	*Completed Cuts*	*Overall*
Observed Cut Outcomes	8	15	8	31
Correctly Classified Cuts	7.7	11.7	2.5	21.9
Fault Detection Success Rate	96.3%	78.0%	31.3%	70.6%

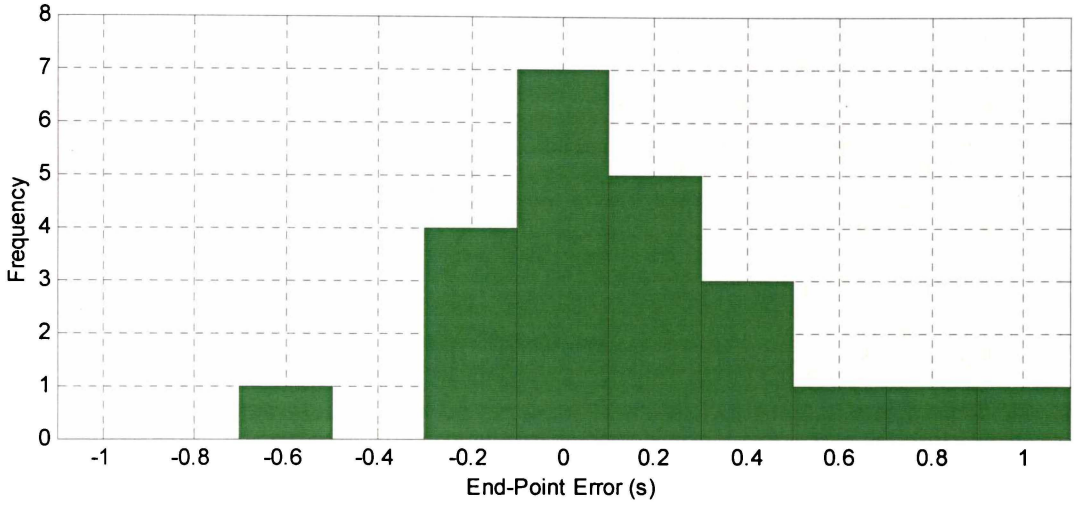

FIGURE 4.11 Distribution of end-point errors for the MLP fault detection strategy.

It is thought that the number of training datasets that are available is limiting the performance of the MLP-based strategy. The networks also appear to have particular difficulty classifying signals with significant pullout oscillations. These oscillations can result in an unstable network output, which produces an unreliable determination of the cut end-point.

4.4.2 RBF Network Fault Detection

A fault detection strategy using an RBF neural network is developed as an alternative to the multilayer perceptron design. This network also employs concurrently presented inputs, requiring the network to have an input neuron for each data-point in the signal. Down-sampling is used to reduce the size of the network and a down-sampling factor of 20 is again selected, resulting in the same number of input and output neurons as used for the MLP-based strategy. The RBF network is trained using the leave-one-out strategy, with the same target output as used to train the feed-forward networks. The

network has a hard-limiting function applied to the output, as was used for the MLP output.

The Neural Network Toolbox provides two methods for creating RBF networks: an exact design that creates as many neurons as there are input vectors, and an optimised design that adds neurons one at a time until the network error is minimised or the maximum number of neurons is reached. If the optimised design method is employed then the network generally has 25 neurons, as opposed to 30 neurons if the exact method is used. However, there is no improvement in the performance of the optimised network, so the exact design method is used during subsequent testing.

Networks are trained with a range of basis function widths σ (refer Section 2.3.4) to ensure that the output error is minimised (refer Figure 4.12). The minimum mean error in the output signal occurs when $\sigma = 5.50$. This minimum value is the same for both the raw and hard-limited output signals. Table 4.4 gives the classification success for the RBF strategy.

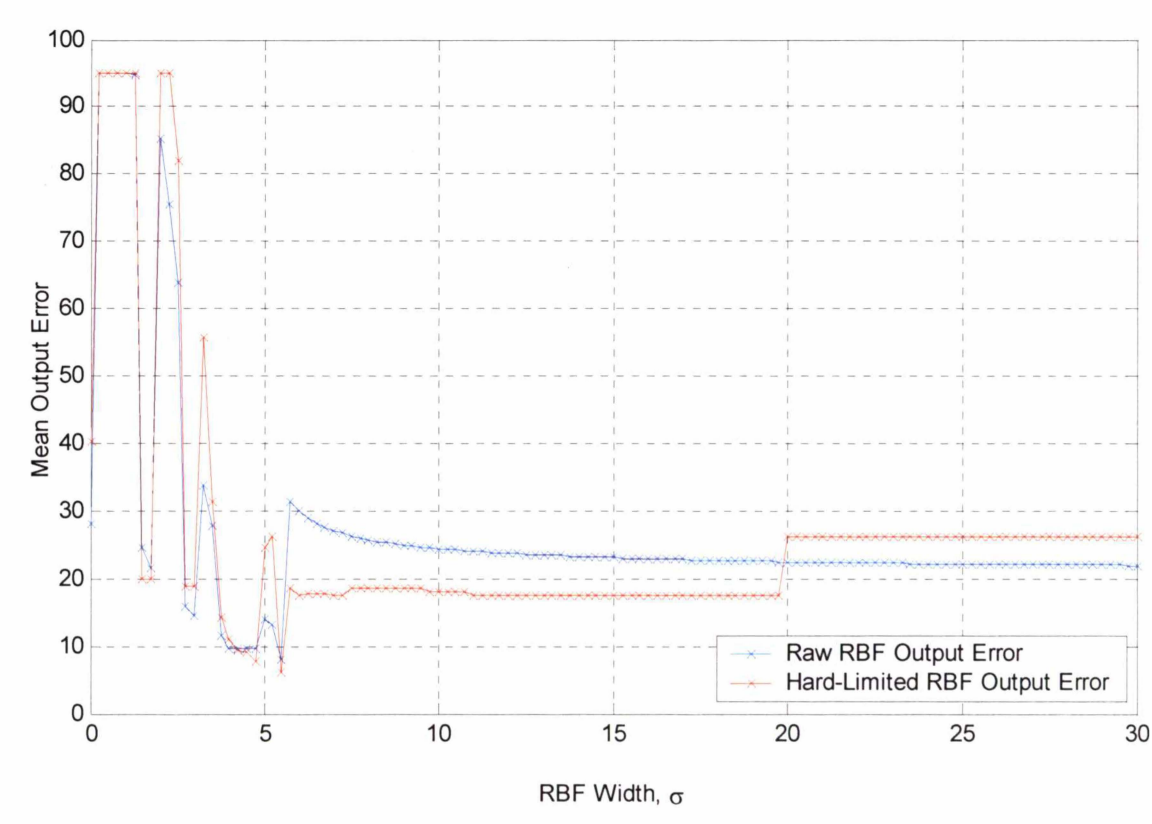

FIGURE 4.12 Mean output error for the RBF network with various neuron widths.

TABLE 4.4 Classification success of the RBF network fault detection strategy.

	Insertion Failures	*End-Point Failures*	*Completed Cuts*	*Overall*
Observed Cut Outcomes	8	15	8	31
Correctly Classified Cuts	8	14	0	22
Fault Detection Success Rate	100%	93.3%	0%	71.0%

The RBF network correctly classifies more of the end-point failures than the MLP, but does not classify any of the completed cuts. The overall classification performance is almost identical to that of the MLP strategy. Figure 4.13 gives the end-point error distributions for both the RBF and MLP strategies. The standard deviation of this error distribution is 0.27 seconds. The mean absolute magnitude of the end-point errors is 0.21 seconds, which is an improvement from the 0.26 seconds that resulted from using the MLP strategy. However, this error is still worse than obtained with the modified knowledge-based fault detection strategy. The RBF strategy detects 18 out of 23 of the cut end-points within 0.3 seconds of the actual end-point time, which is also an improvement from the MLP strategy.

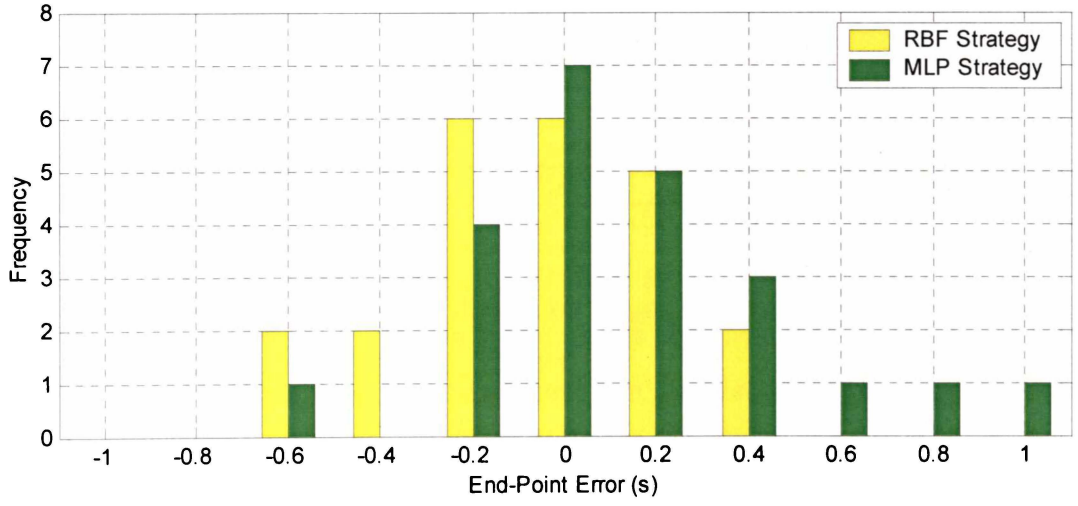

FIGURE 4.13 Distribution of end-point errors for the RBF and MLP fault detection strategies.

The poor classification of completed cuts appears to be the result of the low number of training vectors. This is exacerbated by the variation between the available completed

cut signals. Both of these factors result in a fault detection strategy with a poor ability to generalise beyond the training data.

4.4.3 LVQ Network Fault Detection

LVQ networks are used for the classification of input vectors. In general, an LVQ network cannot be trained to output the end-point time of a particular cut, but they can be used to indicate whether a particular type of fault has occurred.

The leave-one-out training strategy is again used, with a down-sampling factor of 20. The LVQ networks have 167 inputs and 3 outputs, with 1 output for each class of cut (insertion failure, end-point failure or completed cut). The learning algorithm LVQ1 is initially used (refer Section 2.3.7.1). The network is tested with various numbers of hidden neurons, with the performance remaining relatively constant. The final network structure has 10 hidden neurons, and is trained for 50 epochs. Table 4.5 gives the classification success of the LVQ network, with the results being the average of 10 runs.

TABLE 4.5 Classification success of the LVQ network fault detection strategy with the LVQ1 learning algorithm.

	Insertion Failures	*End-Point Failures*	*Completed Cuts*	*Overall*
Observed Cut Outcomes	8	15	8	31
Correctly Classified Cuts	8	6.8	3.2	18.0
Fault Detection Success Rate	100%	45.3%	40.0%	58.1%

The LVQ network demonstrates an excellent ability to classify insertion failures. However, both end-point failures and completed cuts are poorly classified and the overall fault detection success rate of 58.1% is disappointing. Because the insertion failure signals are all very similar, it is relatively straightforward for the LVQ network to characterise an insertion failure. The other two cut-classes have signals that are very dissimilar, meaning that the LVQ1 learning algorithm cannot effectively differentiate between the classes.

The LVQ2.1 algorithm (refer Section 2.3.7.2) is used in an identical test to determine if it produces a more effective fault classifier. This learning algorithm acts to separate

neighbouring neurons that generate different classifications and can improve the performance of an LVQ network trained with LVQ1. Table 4.6 gives the classification success after using the LVQ2.1 algorithm and is again the average of 10 runs.

TABLE 4.6 Classification success of the LVQ network fault detection strategy with the LVQ2.1 learning algorithm.

	Insertion Failures	*End-Point Failures*	*Completed Cuts*	*Overall*
Observed Cut Outcomes	8	15	8	31
Correctly Classified Cuts	7.5	13.2	0.9	21.6
Fault Detection Success Rate	93.8%	88.0%	11.3%	69.7%

The performance of the LVQ2.1 network is superior to that of the LVQ1 network, with almost twice as many end-point failures being correctly classified. Some of this improvement has been at the expense of the completed cut detection rate, which has decreased from the LVQ1 trial. The majority of the completed cuts are consistently misclassified as end-point failures. While this means that the introduction of a different learning algorithm has not necessarily improved the differentiation between end-point failures and completed cuts, it has increased the overall success rate of the LVQ fault detection strategy. The overall performance of the LVQ2.1 network is almost equal to that of the MLP and RBF fault detection strategies.

4.5 DISCUSSION

The classification performance of the knowledge-based and neural network fault detection strategies are summarised in Table 4.7.

All of the tested strategies reliably detect insertion failures, with four systems successfully classifying 100% of the available cases. The RBF network exhibits the best classification performance for end-point failures, although it also classifies none of the completed cuts correctly. There is a similar trade-off in performance for all of the neural network strategies. The best performing strategy could only correctly classify 14.2 of the 23 end-point failures and completed cuts. A superior performance level would be achieved by simply classifying all of these 23 cases as end-point failures,

resulting in 15 successful classifications. The classification of completed cuts is poor for all of the tested strategies, with the best result being a 50% success rate for the modified knowledge-based system.

TABLE 4.7 Classification success of the tested fault detection strategies.

	Fault Detection Success Rate (%)			
	Insertion Failures	*End-Point Failures*	*Completed Cuts*	*Overall*
Initial Knowledge-Based	100.0	87.5	37.5	78.1
Modified Knowledge-Based	100.0	87.5	50.0	81.3
MLP Network	96.3	78.0	31.3	70.6
RBF Network	100.0	93.3	0.0	71.0
LVQ Network – LVQ1	100.0	45.3	40.0	58.1
LVQ Network – LVQ2.1	93.8	88.0	11.3	69.7

When the overall success rate is considered, none of the neural network strategies are as effective as either of the knowledge-based fault detection systems. The key reason is that the knowledge-based strategies employ expert knowledge of the process and of the observed features in the axis current signals. These specifically tailored strategies maximise the classification success rate for this particular set of signals. Because of the small size of the dataset, it is reasonably easy to select some key fault indicators that are present in the majority of the signals. This would not necessarily be the case if the set of available signals were larger (320 Y-cuts instead of 32). The knowledge-based strategies are also very specific to this particular iteration of the automated Y-cut process.

All of the neural network strategies appear to suffer from a poor ability to generalise beyond the set of training data, which is a common failure identified in previous neural network research (Duda *et al.*, 2000). An instance of this poor generalisation occurred during the testing of the feed-forward network, with pullout signal oscillations producing spurious network outputs. There were also some small oscillations in the RBF output. All three types of network fail to differentiate between end-point failures and completed cuts reliably. This failing could be due to the small number of training samples and that the available completed cut samples are significantly disparate. A

larger population of samples should give a better representation of a typical completed cut signal and should improve the classification success.

Table 4.8 gives the errors in the determination of the end-point time. The two knowledge-based strategies result in a smaller absolute end-point error than the neural network models. The error distribution is also narrower for the knowledge-based strategies, indicating that these systems are providing a more reliable estimate of the true end-point time.

TABLE 4.8 End-point time determination of the tested fault detection strategies.

	Absolute End-Point Error (s)	*Error Distribution Standard Deviation (s)*
Initial Knowledge-Based	0.17	0.20
Modified Knowledge-Based	0.14	0.16
MLP Network	0.26	0.36
RBF Network	0.21	0.27

4.6 FAULT DETECTION USING A HYBRID STRATEGY

A hybrid strategy is developed by combining the modified knowledge-based strategy with the neural network strategies. The input signal is conditioned using the modified knowledge-based strategy, rather than directly presenting the down-sampled signal to the networks. Figure 4.14 shows the conditioned input signal that results from the application of the modified knowledge-based strategy. A threshold level of 0.00 A and a minimum width of 0.11 seconds are again used. If the input signal is greater than the threshold level for a period longer than the minimum width, then the conditioned signal is set to 1.00 A, otherwise the conditioned signal becomes 0.00 A.

The conditioned input data are used to train the three types of neural network. The network architectures and target outputs are identical to those used in Section 4.4. The classification performance of the hybrid strategies are summarised in Table 4.9 and the errors in the determination of the end-point time are given in Table 4.10. The performances of the previous strategies have been included in both tables for reference purposes.

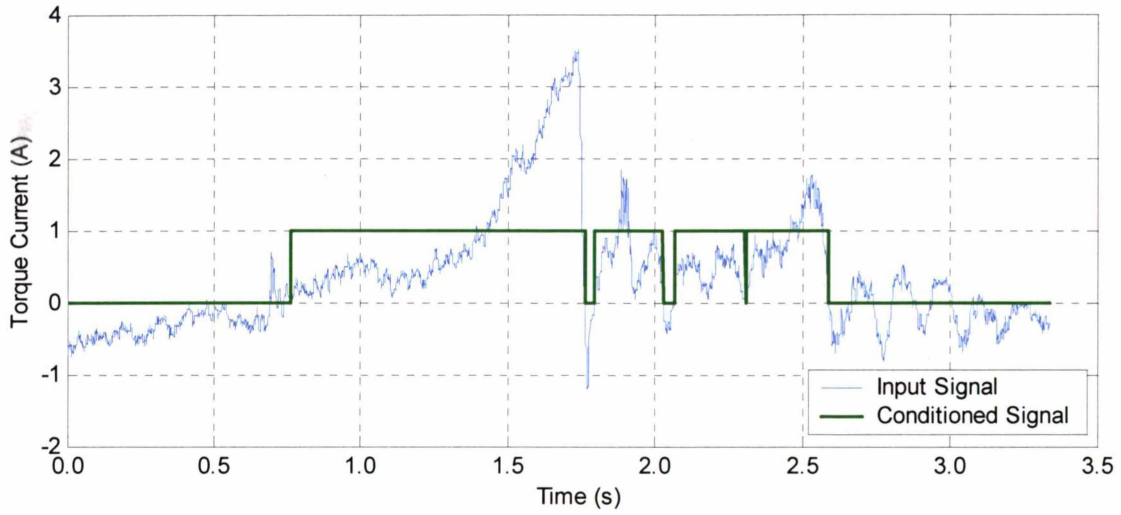

FIGURE 4.14 Conditioned input signal for the hybrid fault detection strategy.

TABLE 4.9 Classification success of the hybrid fault detection strategies.

	Fault Detection Success Rate (%)			
	Insertion Failures	*End-Point Failures*	*Completed Cuts*	*Overall*
Initial Knowledge-Based	100.0	87.5	37.5	78.1
Modified Knowledge-Based	100.0	87.5	50.0	81.3
MLP Network	96.3	78.0	31.3	70.6
Hybrid MLP Network	**97.5**	**87.3**	**45.0**	**79.0**
RBF Network	100.0	93.3	0.0	71.0
Hybrid RBF Network	**100.0**	**100.0**	**12.5**	**77.4**
LVQ Network – LVQ1	100.0	45.3	40.0	58.1
Hybrid LVQ Network – LVQ1	**100.0**	**53.3**	**96.3**	**76.5**
LVQ Network – LVQ2.1	93.8	88.0	11.3	69.7
Hybrid LVQ Network – LVQ2.1	**100.0**	**77.3**	**20.0**	**69.0**

TABLE 4.10 End-point time determination of the hybrid fault detection strategies.

	Absolute End-Point Error (s)	*Error Distribution Standard Deviation (s)*
Initial Knowledge-Based	0.17	0.20
Modified Knowledge-Based	0.14	0.16
MLP Network	0.26	0.36
Hybrid MLP Network	**0.15**	**0.20**
RBF Network	0.21	0.27
Hybrid RBF Network	**0.21**	**0.27**

The performance of the hybrid strategies is superior to the basic neural network strategies in all cases except the LVQ network trained with the LVQ2.1 algorithm. The classification success of the MLP network increased from 70.6% to 79.0%. This is the best performing neural network strategy and is bettered only by the modified knowledge-based strategy.

The classification success of the RBF network improved from 71.0% to 77.4%. The biggest change in performance is for the LVQ network trained with the LVQ1 algorithm, with an increase in classification performance from 58.1% to 76.5%. The performance of the LVQ1 network is now superior to the LVQ2.1 network. The LVQ1 strategy also correctly classifies the highest number of completed cuts of any of the tested strategies, although it still has the worst performance at classifying end-point failures with only 53.3% correctly classified. The overall classification success of the hybrid LVQ2.1 network has decreased slightly from 69.7% to 69.0%.

The accuracy of the end-point determination has improved for the hybrid MLP strategy. The mean absolute end-point error for this strategy has decreased from 0.26 s to 0.15 s, which is bettered only by the modified knowledge-based strategy with a mean absolute error of 0.14 s. The standard deviation of the error distribution has decreased from 0.36 s to 0.20 s. The accuracy of the hybrid RBF strategy is identical to the basic RBF strategy.

This work has demonstrated that the tested neural networks have the potential to detect faults from process load variations, even though they have not performed as robustly as a developed knowledge-based strategy. However, the small size of the available dataset means that it is unclear if any of the strategies will be suitable for implementation within the automated Y-cutting system. Therefore, it will be necessary to obtain larger datasets for further analysis.

5. KUKA Y-CUT FAULT DETECTION

Testing using the IRL8L robot has demonstrated that the Y-cutting process loads produce observable variations in the current supplied to the motors of the robot axes. The load variations can distinguish between faulty and non-faulty operation of the Y-cut system. However, it is unclear whether these load variations are observable with the new tool design and KUKA robot, particularly given the different axis configuration and kinematics of the KUKA. The commissioning of a full KUKA Y-cutting system in Goulburn, Australia enabled the capture and analysis of larger datasets than could be obtained from the temporary IRL8L installation.

5.1 ROBOT AND TOOL MODIFICATIONS

A key advantage of the KUKA-based system over the IRL8L system is the additional degrees-of-freedom that the six-axis robot provides. The extra two axes of the KUKA allow the positioning of the tool with a full range of orientations. The IRL8L robot could not offer this range of motion, and the performance of the system during the AMP trial suffered as a result. The configuration of the IRL8L axes means that there is always a compromise between the presentation angle of the tool at insertion, and the angle of the tool down the remainder of the cut-path. The tip of the tool needs to be angled slightly with respect to the leg during insertion. However, if this angle is too steep then the tool-tips are not parallel with the leg, and there is a greater risk of damaging the meat or membrane layer. The only mechanisms to change the presentation angle at insertion are to modify the tool mount or shift the height of the entire robot – both of which require a great deal of time and effort.

Some minor modifications were made to the tool design due to the outcome of the trial at AMP. One of the most important improvements was a reduction in the separation between the cutting blades (refer Figure 5.1). The AMP tool had a separation of 2.5

mm between the tool-tips and approximately 1.5 mm between the two cutting blades. This separation was considered one of the main contributors to the poor performance observed during the AMP trial. The Goulburn tool has a 0.2 mm separation between both the tool-tips and the blades. The tool-tip profile has been modified making them thinner and sharper. This change is made in an attempt to improve the insertion success-rate with the new tool-tips being easier to insert into the opening cut. The tool mounts on the KUKA robot with a torsional rubber suspension unit (refer Section 3.2.1.4), which helps absorb the process loads identified at AMP.

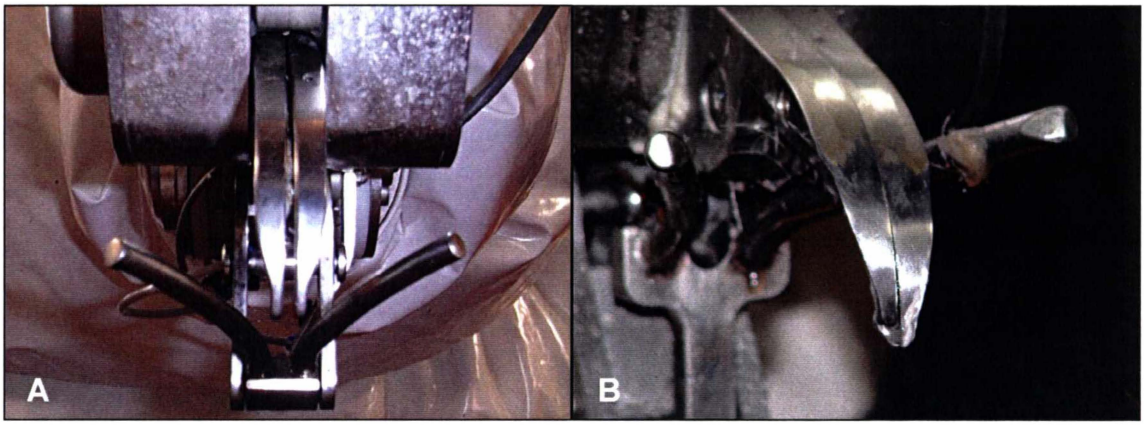

FIGURE 5.1 The cutting blades of a) the AMP tool and b) the Goulburn tool.

5.2 PROCESS IMPROVEMENTS

Observations during initial testing at Goulburn showed that the new robot and tool modifications have made a vast improvement to the performance of the system. Changes to the presentation angle of the tool at the opening cut are straightforward, with a resulting improvement to the insertion success-rate. The cuts made by the new tool are precise and clean, and the traversal of the knee is easier. The new tool has not prematurely withdrawn from a single Y-cut, meaning that there has been no observed incidence of end-point failure. If insertion is successful then the tool appears to always complete the cut. This dramatic performance improvement is a direct result of the modifications to the blade separation.

There are some limited instances of incorrect cut-path faults. However, these highly subjective occurrences are dependent on the preference of the individual supervisors,

making it very difficult to take remedial action to correct the cut-path. For this reason the detection and correction of these faults is not considered in the subsequent analysis.

There has been no observed damage to the meat or membrane layer. The ability of the KUKA to specify the orientation angle completely means that the tool-tips can be kept parallel to the leg, regardless of the presentation of the tool at insertion. The combination of the new blades and the mounting suspension unit greatly reduce the process loads experienced by the robot. The KR60/2 robot is significantly more powerful than the IRL8L. There is also provision within the KUKA programming environment to handle robot overload conditions automatically by defining the amount that the command torque can vary from the actual torque. The performance of the new system is so superior to the version used at AMP that there is now only one process fault that needs correction: insertion failure.

5.3 DATA ACQUISITION

The KUKA software environment includes a diagnostic oscilloscope program. The oscilloscope can capture specific axis parameters and I/O data, and then display this data via a simple user interface. This program is used to capture the motor current for all six axes of the robot. This data is captured every 2 ms. The output signal to extend and retract the leg guide is captured simultaneously, although this occurs with a slower sampling rate of 12 ms. This signal partitions the actual current signals into individual cut datasets, with each foreleg cut of the Y-cut operation treated as a separate case; the leading leg is the downstream leg of the carcass, and the trailing leg is the upstream leg. A video camera obtains a visual record of the cut, allowing the subsequent determination of the cut outcome.

The captured dataset consists of 308 leg-cuts, with 171 leading legs and 137 trailing legs. The 171 leading leg cuts resulted in 142 completed cuts and 29 insertion failures. The 137 trailing leg cuts resulted in 113 completed cuts and 24 insertion failures. The sock-ringer was switched off periodically to artificially increase the number of insertion failures. Figure 5.2 gives the axis currents for the all of the leading leg cuts, and Figure 5.3 gives the axis currents for the all of the trailing legs.

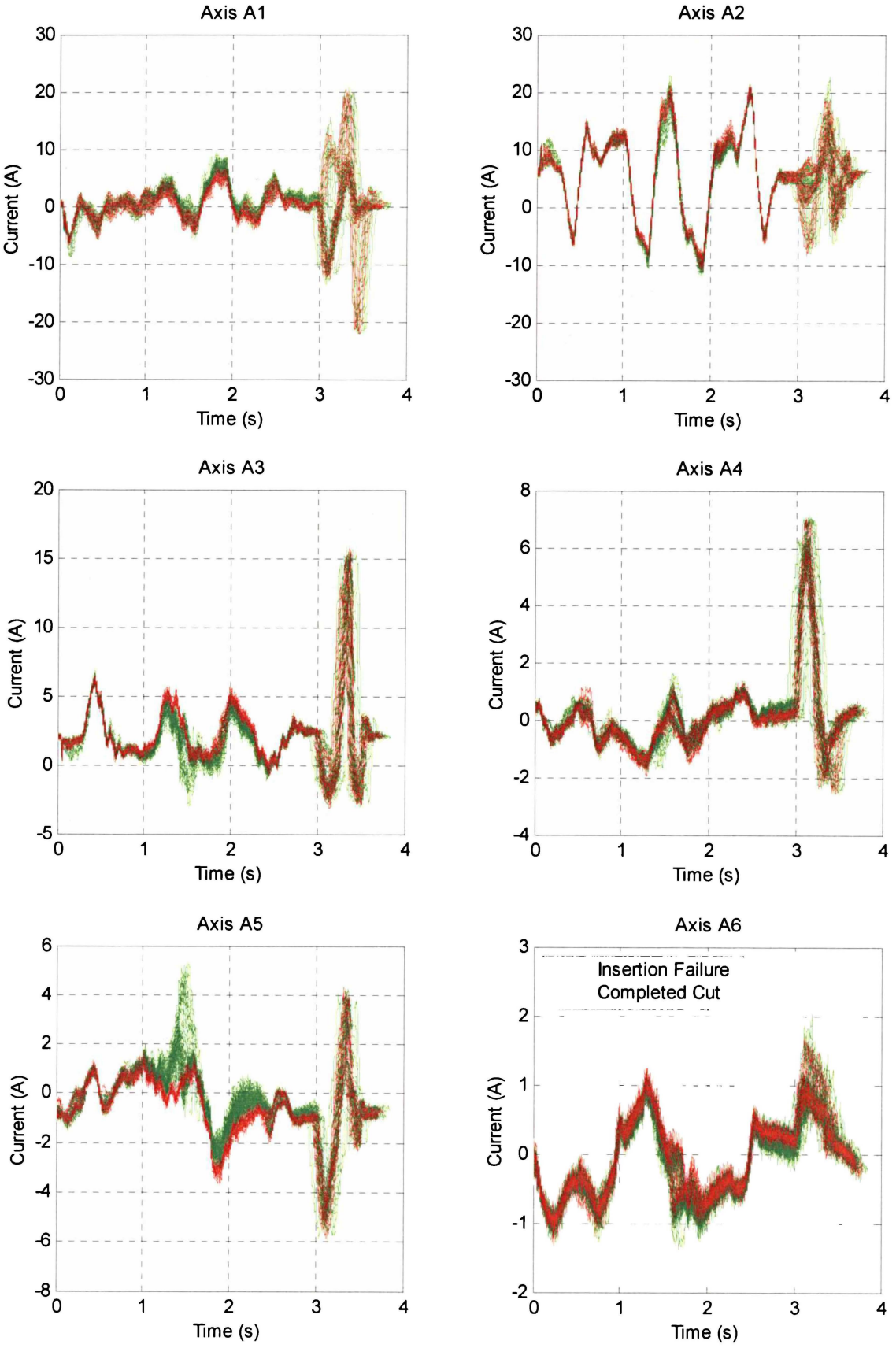

FIGURE 5.2 Current data from the KUKA robot for leading leg cuts.

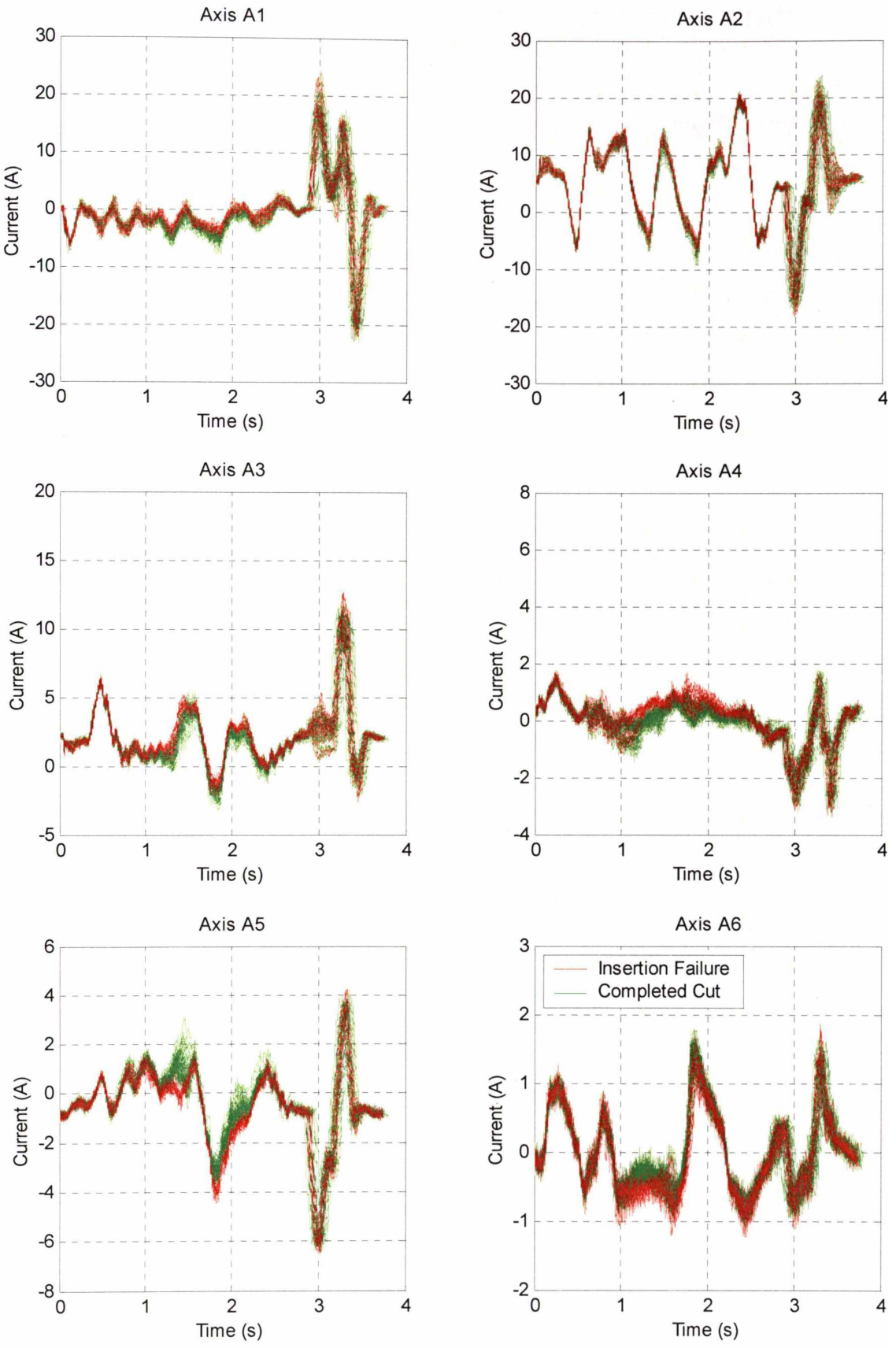

FIGURE 5.3 Current data from the KUKA robot for trailing leg cuts.

5.4 PRELIMINARY SIGNAL ANALYSIS

The clearest differentiation between the insertion failures and completed cuts occurs for the axis A5 signal, indicating that the majority of the process load is experienced by A5 (refer Figures 5.2 and 5.3). There are also some smaller differences in the other axes, particularly A3 and A4. The complicated kinematics of the KUKA robot distributes the process load through more than one axis, since more than one axis is usually involved in any particular motion. The IRL8L robot has much simpler kinematics, with a more direct link between the process load and the shoulder axis. Moving the IRL8L through a vertical path only uses the shoulder and probe axes, whereas the same motion with the KUKA requires the coordination of all six axes.

Figure 5.4 plots the A5 current for a representative leading leg insertion failure and completed cut, and includes the identification and labelling of the cut-path features.

FIGURE 5.4 Identification of cut-path features in axis A5 motor current data.

The output signal driving the leg-guide references the individual cut datasets. The leg-guide extension occurs prior to the tool moving towards the opening cut. The tool then approaches the leg with the leg-guide extended. The current increases during this approach phase of the cut due to the increase in the load experienced by the robot as the tool impacts with the leg. This load increase occurs for both completed cuts and insertion failures since the tool strikes the leg regardless of the cut outcome.

The robot pauses temporarily when the tool is positioned at the opening cut to give the tool-tips time to nibble into the pelt pocket. The tool then begins to move down the leg. As noted with the IRL8L shoulder axis data, there is no discernable difference between an insertion failure and a completed cut during this insertion phase of the process.

During the pre-knee phase, a very small increase in the current for the completed cut is observed, although the difference between the two signals is minimal. This is in contrast to the IRL8L data, where there was a steady departure of the completed cut signal from the insertion failure signal.

There is an increase in the current for the completed cut as the tool traverses the knee. As with the IRL8L data, the load experienced while cutting the knee is again a key feature discriminating between the two classes of cut. This loading over the knee can also be seen in the other robot axes (refer Figures 5.2 and 5.3).

There is a small difference between the completed cut and insertion failure signals while the remainder of the leg is cut. This difference decays during the pullout phase of the cut. The final part of the dataset is a point-to-point (PTP) move up to the start of the next leg. While this is not part of the actual cut-path, it is interesting to note the axis currents that are required to complete this very rapid motion.

5.5 OFF-LINE TRAINING OF AN LVQ NETWORK

Because of the improvements made to the process, insertion failures are the only faults that require detection. It is no longer important to isolate the timing of a fault, as was needed for the detection of end-point failures during the AMP trial. Fault detection can

now be achieved by classifying cuts as being either insertion failures or completed cuts. This greatly reduces the complexity of the required fault detection strategy. The previously tested neural network strategies all demonstrated an excellent ability to detect insertion failures. Of these neural network models, the LVQ network is considered the easiest to implement.

Because of the kinematics of the KUKA robot, the process load is no longer confined to a single axis. This presents the possibility that a network trained with data from more than one axis could perform better than a network trained with data from a single axis.

5.5.1 Single Axis Training

To establish which axis results in the best classification of insertion failures and completed cuts, a series of LVQ networks are trained with the data from a single axis. The individual cut datasets are truncated after 2.8 seconds, since the PTP motion at the end of the signal is not considered part of the cut-path. The datasets are then split into leading and trailing leg groups. Each network is trained using one leg-type only, with a random 50/50 split of the data for training and testing. The leave-one-out training method is not used, as this is a computationally intensive process that requires a different network for each testing sample. There is also greater amount of data than used during the development work in Chapter 4, which should minimise any bias introduced by using a holdout training method. Each network has two neurons, and is trained for 50 epochs. Training and testing data are randomly selected for each run, with 100 runs completed. Summary statistics from these 100 runs are given in Tables 5.1 (for the leading leg data) and 5.2 (for the trailing leg data).

TABLE 5.1 Leading leg classification success of LVQ networks trained with single axis data.

	Completed Cuts				*Insertion Failures*				*Overall*			
Axis	*Mean*	*SD*	*Min*	*Max*	*Mean*	*SD*	*Min*	*Max*	*Mean*	*SD*	*Min*	*Max*
A1	0.97	0.02	0.91	1.00	0.98	0.06	0.59	1.00	0.97	0.03	0.82	1.00
A2	0.96	0.03	0.83	1.00	0.94	0.13	0.40	1.00	0.95	0.04	0.84	1.00
A3	**0.98**	0.02	0.87	1.00	**1.00**	0.00	1.00	1.00	**0.98**	0.02	0.89	1.00
A4	0.92	0.05	0.78	1.00	0.46	0.25	0.07	1.00	0.84	0.06	0.69	0.99
A5	**0.98**	0.03	0.90	1.00	**1.00**	0.00	1.00	1.00	**0.98**	0.02	0.92	1.00
A6	0.97	0.03	0.82	1.00	0.94	0.14	0.35	1.00	0.97	0.04	0.80	1.00

TABLE 5.2 Trailing leg classification success of LVQ networks trained with single axis data.

	Completed Cuts				*Insertion Failures*				*Overall*			
Axis	*Mean*	*SD*	*Min*	*Max*	*Mean*	*SD*	*Min*	*Max*	*Mean*	*SD*	*Min*	*Max*
A1	0.97	0.03	0.89	1.00	0.99	0.04	0.75	1.00	0.97	0.03	0.91	1.00
A2	0.93	0.04	0.79	1.00	0.93	0.13	0.40	1.00	0.93	0.04	0.81	1.00
A3	0.96	0.03	0.86	1.00	0.89	0.09	0.60	1.00	0.95	0.03	0.85	1.00
A4	**0.98**	0.02	0.86	1.00	**1.00**	0.00	1.00	1.00	**0.98**	0.02	0.88	1.00
A5	**0.98**	0.02	0.89	1.00	**1.00**	0.00	1.00	1.00	**0.98**	0.02	0.91	1.00
A6	0.96	0.04	0.75	1.00	0.96	0.11	0.46	1.00	0.96	0.05	0.75	1.00

Two axes produced networks with the best overall classification performance for the leading leg data. On average, axes A3 and A5 both successfully classified 100% of the insertion failures and 98% of the completed cuts, giving an overall classification performance of 98% on all testing data. The A5 networks have a slightly higher minimum success-rate compared to the A3 networks, from the 100 runs that have been conducted.

Trials with the trailing leg data also resulted in two axes having nearly equivalent classification performances. Data from axes A4 and A5 produced networks that successfully classified 100% of the insertion failures and an average of 98% of the completed cuts, with an overall classification success-rate of 98%.

Although A3 produced networks with the equal-best performance for the leading leg data, the axis also had the second-worst performance for the trailing leg data, with a classification success-rate of 95%. A similar observation is made for A4, which performed well for the trailing leg data, but produced the worst performing networks based on the leading leg data, with a classification success-rate of 84%.

5.5.2 Multiple Axis Training

After training with single axis data, LVQ networks are trained using the best two performing axes for each leg-type. These are axes A3 and A5 for the leading leg, and axes A4 and A5 for the trailing leg. Networks are also trained with data from all six axes. The data are combined by concatenating the individual axis signals. A 50/50 split is again used to partition the training and testing data. Each network has two neurons

trained for 50 epochs, and 100 runs are completed. The classification success is given in Tables 5.3 and 5.4, with the performance of the single axis networks included for comparative purposes.

TABLE 5.3 Leading leg classification success of LVQ networks trained with multiple axis data.

	Completed Cuts				*Insertion Failures*				*Overall*			
Axis	*Mean*	*SD*	*Min*	*Max*	*Mean*	*SD*	*Min*	*Max*	*Mean*	*SD*	*Min*	*Max*
A3+A5	**0.98**	0.02	0.92	1.00	**1.00**	0.00	1.00	1.00	**0.98**	0.01	0.93	1.00
AllAxes	**0.98**	0.02	0.91	1.00	**1.00**	0.00	1.00	1.00	**0.98**	0.02	0.93	1.00
A3	**0.98**	0.02	0.87	1.00	**1.00**	0.00	1.00	1.00	**0.98**	0.03	0.89	1.00
A5	**0.98**	0.02	0.90	1.00	**1.00**	0.00	1.00	1.00	**0.98**	0.02	0.92	1.00

TABLE 5.4 Trailing leg classification success of LVQ networks trained with multiple axis data.

	Completed Cuts				*Insertion Failures*				*Overall*			
Axis	*Mean*	*SD*	*Min*	*Max*	*Mean*	*SD*	*Min*	*Max*	*Mean*	*SD*	*Min*	*Max*
A4+A5	**0.98**	0.02	0.89	1.00	**1.00**	0.00	1.00	1.00	**0.98**	0.02	0.91	1.00
AllAxes	0.97	0.03	0.88	1.00	**1.00**	0.00	1.00	1.00	0.97	0.02	0.90	1.00
A4	**0.98**	0.02	0.86	1.00	**1.00**	0.00	1.00	1.00	**0.98**	0.02	0.88	1.00
A5	**0.98**	0.02	0.89	1.00	**1.00**	0.00	1.00	1.00	**0.98**	0.02	0.91	1.00

There is no improvement to the classification success when the data from two axes are used to train the LVQ network. In the case of the leading leg cuts, the network trained with A3 and A5 data has a slightly better minimum performance than either of the single-axis networks. For the trailing leg data, the combined A4 and A5 network is identical in performance to the network trained using just A5 data. There is also no improvement when data from all six axes are used, with a slight decrease in performance for the trailing leg cuts.

Because the multiple axis data do not improve the classification performance, axis A5 is selected for use in the final fault detection implementation. This axis provides consistent results for both leading and trailing leg cuts. The selection of a single axis also limits the potential computational requirements.

5.5.3 Down-Sampling of the Axis Data

Even with the datasets truncated at 2.8 seconds, each cut signal is still made up of 1400 data points, resulting in an LVQ network with 2800 weights. While the KUKA

oscilloscope program can sample data at 500 Hz, external robot programs can only access the axis motor current data every 12 ms, which is equivalent to a sampling frequency of 83.3 Hz. This sampling frequency corresponds to the 12 ms interpolation cycle of the robot controller.

The leading leg A5 current data is down-sampled to determine if a reduced sampling frequency affects the classification ability of the LVQ network. The down-sampled data are then used in the same manner as before (50/50 split, 2 neurons, 50 epochs, 100 runs). Figure 5.5 shows the variation in classification success with the effective sampling frequency. The original data has to be significantly down-sampled before the overall classification success is adversely affected.

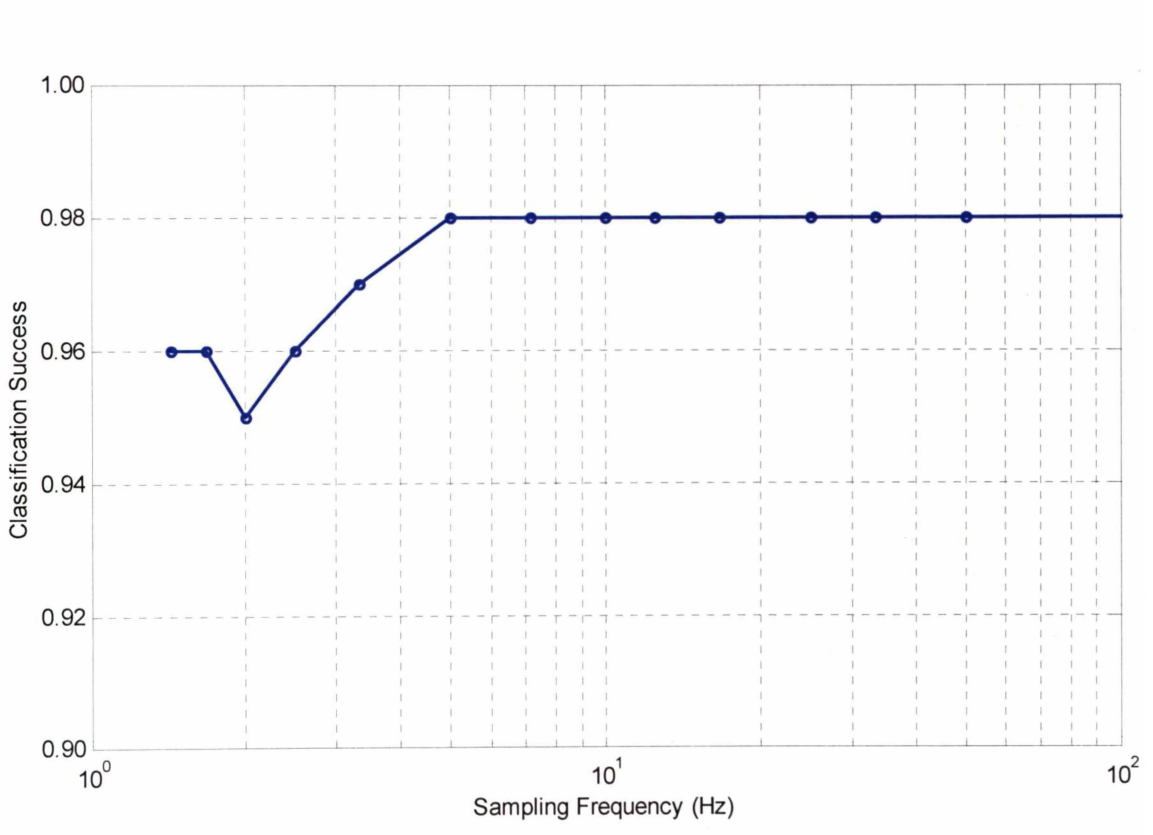

FIGURE 5.5 The effect of down-sampling on the classification performance of the LVQ network.

The classification success remains constant at 98% for sampling frequencies greater than 5 Hz, which is equivalent to a sampling rate of less than 200 ms. The classification success reduces slightly for frequencies less than 5 Hz, although the minimum observed performance is still 95%. This indicates that the main differences between the two cut

types are at a very low frequency level, with higher frequency components not greatly improving the classification. This observation suggests that the sustained increase in the load experienced by the tool during the initial pelt cutting and knee traversal is the key discriminating feature used by the LVQ network.

5.5.4 Final Network Training

It is advantageous to have a sampling rate that is a multiple of the 12 ms interpolation cycle of the KUKA controller since this is the update rate for the variables containing the motor current data. This is also the polling rate for the software interrupts. Therefore, a sampling rate of 180 ms is selected for the initial implementation of the network in the KUKA system, with this rate being less than the previously observed 200 ms cut-off.

A sampling rate of 180 ms equates to a down-sampling factor of 90, resulting in the capture of 16 samples over 2.7 seconds. This down-sampling factor is applied to the A5 current datasets. The down-sampled data is split into leading and trailing sets, and an LVQ network is trained on each (2 neurons, 50 epochs). Final testing with this sampling rate produced networks with an average classification performance of 98%.

Figure 5.6 gives the insertion failure and completed cut input weights for the trained network. The sixteen LVQ weights are numbered 0 to 15. The leading leg network weights for insertion failures and completed cuts are very similar to each other. The completed cut activation is slightly higher from weight six onwards, which is immediately prior to the tool reaching the top of the knee. The trailing leg weights have a larger difference between the completed cut and insertion failure cases, particularly during the traversal of the knee.

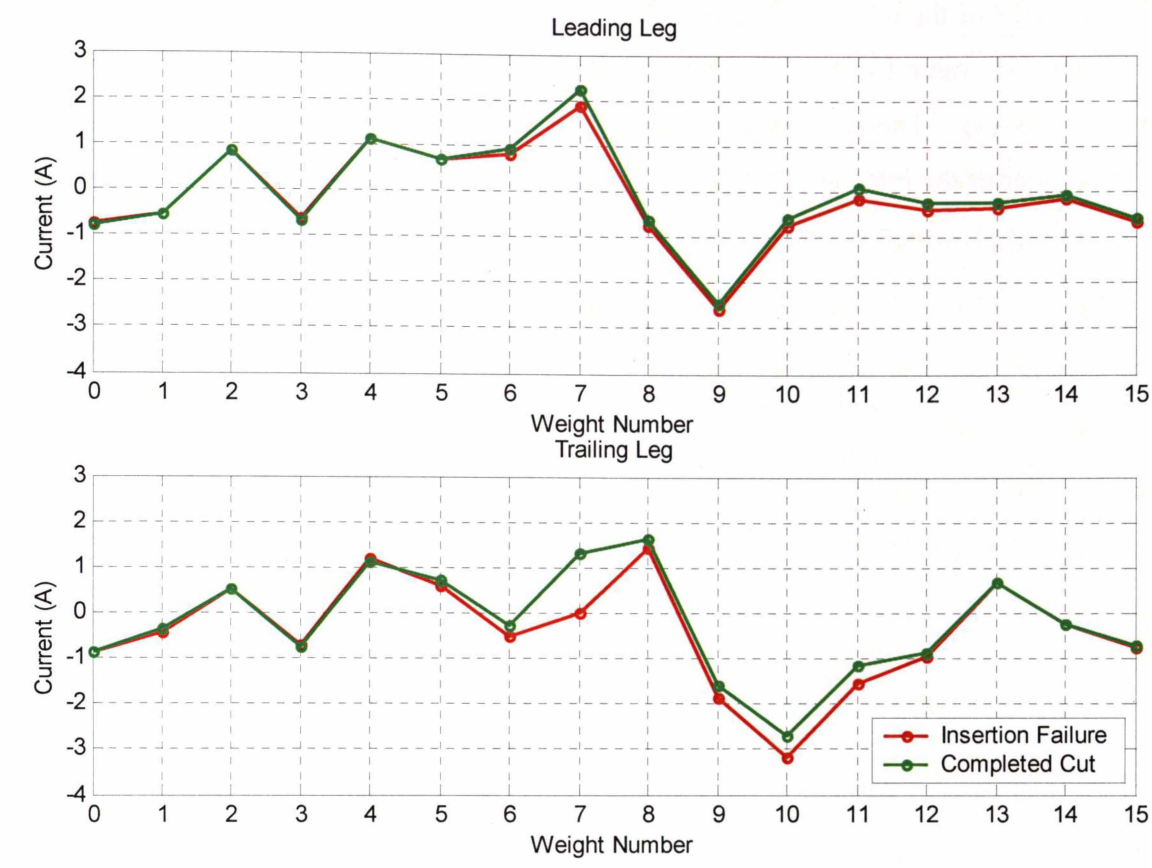

FIGURE 5.6 Trained weights for implementation of LVQ network.

5.6 IMPLEMENTATION OF FAULT DETECTION MODULE

The MATLAB-based LVQ network used in testing was transferred into KRL for implementation in the Goulburn Y-cutting system. The developed fault detection system resides within the "measurement" program (refer Section 3.3.6), and executes via the use of interrupt routines.

5.6.1 KRL Implementation

Two interrupt routines are used. The first interrupt is triggered by the leg-guide extension signal (refer Figure 5.7). This signal is set from within the main Y-cutting program, and occurs prior to the tool moving in to the leg. The initial action of the first interrupt routine is to obtain a sample of the motor current for axis A5. This is done via a system variable `$CURR_ACT[5]`, which gives the actual motor current for A5 as a

percentage of the maximum permissible current. The percentage current is converted to an absolute value by multiplying by $CURR_MAX[5] (the maximum permissible current for axis A5). The motor current is stored in a 16-element array A5_Curr[]. The remainder of the interrupt routine deals with the initialisation of an internal system timer $TIMER[13], which is used to trigger a second interrupt routine.

```
GLOBAL DEF Leg_Guide_Interrupt()

A5_Curr[1] = $CURR_ACT[5]*$CURR_MAX[5]/100
                                ;Get A5 motor current value
A5_Curr_I = 1                   ;Reset array index variable
$TIMER_STOP[13] = TRUE          ;Ensure timer has stopped
$TIMER_FLAG[13] = FALSE         ;Reset timer flag
$TIMER[13] = -180               ;180 ms timer
$TIMER_STOP[13] = FALSE         ;Start timer
INTERRUPT ON 33                 ;Enable fault detection interrupt

END
```

FIGURE 5.7 Leg-guide interrupt routine.

The second routine completes the remainder of the data collection and implements an LVQ network to determine the classification of the cut (refer Figure 5.8). The interrupt is triggered when the flag $TIMER_FLAG[13] is set, which occurs when $TIMER[13] increments to zero. If sixteen A5 motor current samples have not been obtained, then the timer is reinitialised so that the interrupt can be triggered 180 ms later.

When all sixteen samples have been obtained, the interrupt is disabled and the timer is stopped. The type of leg that has just been cut is determined by checking the data structure-element Current_Leg_Data.Leg_Type. The Current_Leg_Data structure contains the measurement data needed by the path-planning algorithm, and is used within the main Y-cutting program. The leg-type determination allows the referencing of the correct set of LVQ weights. The negative Euclidean distance between the LVQ weights and the sixteen current samples is calculated, and an Insertion_Failure output is set if the samples are closer to the insertion failure weights.

```
GLOBAL DEF Fault_Detection()

INT N                                ;Variable declaration
A5_Curr_I = A5_Curr_I + 1            ;Increment array index
A5_Curr[A5_Curr_I] = $CURR_ACT[5]*$CURR_MAX[5]/100
                                     ;Get A5 motor current value
IF (A5_Curr_I < 16) THEN             ;Is array filled yet?
  $TIMER_STOP[13] = TRUE             ;Stop timer
  $TIMER_FLAG[13] = FALSE            ;Reset timer flag
  $TIMER[13] = -180                  ;180 ms timer
  $TIMER_STOP[13] = FALSE            ;Start timer
ELSE
  INTERRUPT OFF 33                   ;Disable fault detection interrupt
  $TIMER_STOP[13] = TRUE             ;Stop timer
  $TIMER_FLAG[13] = FALSE            ;Reset timer flag
  IF (Current_Leg_Data.Leg_Type == #Leading) THEN
    LT = 1                           ;Current leg is leading
  ELSE
    LT = 2                           ;Current leg is trailing
  ENDIF
  Sum_CC = 0.0                       ;Reset temporary variable
  Sum_IF = 0.0                       ;Reset temporary variable
  FOR N = 1 TO 16                    ;Calculate sum of squares
    Sum_CC = Sum_CC+((IW_CC[LT,N]-A5_Curr[N])*(IW_CC[LT,N]-A5_Curr[N]))
    Sum_IF = Sum_IF+((IW_IF[LT,N]-A5_Curr[N])*(IW_IF[LT,N]-A5_Curr[N]))
  ENDFOR
  Neuron_CC = -SQRT(Sum_CC)          ;Calculate negative Euclidean distance
  Neuron_IF = -SQRT(Sum_IF)
  IF (Neuron_CC > Neuron_IF) THEN
    Insertion_Failure = FALSE        ;Reset fault output signal
  ELSE
    Insertion_Failure = TRUE         ;Set fault output signal
  ENDIF
ENDIF

END
```

FIGURE 5.8 Fault detection interrupt routine.

5.6.2 Initial Module Testing

Following the inclusion of the fault detection code in the Goulburn controller, the system was tested by cutting several sheep with the sock-ringer switched off. This motion should have resulted in an insertion failure being detected, but instead produced an output indicating that the cut had been "completed".

A number of changes had been made to the Y-cut process in the months following the capture of the original axis data. The waypoint offsets had been modified, and the speed of the robot during the cut-path had decreased. A new tool-mount was also being

used, which incorporated an angled flange. This mount shifted the TCP off-axis, reducing the occurrence of a singularity-related error. All of these modifications had changed the motions of the individual axes, meaning that the weights that had been trained off-line were no longer relevant.

5.7 PROVISION OF ONLINE TRAINING SCHEME

An online training scheme was developed to allow for the automatic adjustment of the LVQ weights. This scheme responds to a hardware input signal from an operator who indicates if an insertion failure has occurred by pushing a button on the user console. The weights are modified with the same LVQ1 learning rule that is used during the offline training of the LVQ network (refer Section 2.3.7.1).

5.7.1 KRL Implementation

The LVQ1 learning rule is implemented within the fault detection interrupt routine (refer Figure 5.9). The operator enables learning by setting a global Boolean variable `LVQ_Learning` from within the Y-cut user interface. The operator monitors each leg cut, and presses a button on the user console to indicate if the cut is an insertion failure.

The operator button links directly to the input signal `Target_Output`. If the signalled cut-class is different from the calculated class, then the weights of the calculated class shift away from the motor current data. If the signalled class is the same as the calculated class, then the weights move towards the motor current data. The `Learning_Rate` is nominally set at 0.01.

```
GLOBAL DEF Fault_Detection()

...
IF (A5_Curr_I < 16) THEN
  ...
ELSE
  ...
  IF (Neuron_CC > Neuron_IF) THEN          ;Completed cut determined
    Insertion_Failure = FALSE              ;Reset fault output signal
    IF (LVQ_Learning == TRUE) THEN         ;Online learning is enabled
      IF (Target_Output == TRUE) THEN      ;User indicates insertion failure
        FOR N = 1 TO 16                    ;Shift CC weights away from cut
          IW_CC[LT,N]=IW_CC[LT,N]-(Learning_Rate*(A5_Curr[N]-IW_CC[LT,N]))
        ENDFOR
      ELSE                                 ;User indicates completed cut
        FOR N = 1 TO 16                    ;Shift CC weights towards cut
          IW_CC[LT,N]=IW_CC[LT,N]+(Learning_Rate*(A5_Curr[N]-IW_CC[LT,N]))
        ENDFOR
      ENDIF
    ENDIF
  ELSE                                     ;Insertion failure determined
    Insertion_Failure = TRUE               ;Set fault output signal
    IF (LVQ_Learning == TRUE) THEN         ;Online learning is enabled
      IF (Target_Output == TRUE) THEN      ;User indicates insertion failure
        FOR N = 1 TO 16                    ;Shift IF weights towards cut
          IW_IF[LT,N]=IW_IF[LT,N]+(Learning_Rate*(A5_Curr[N]-IW_IF[LT,N]))
        ENDFOR
      ELSE                                 ;User indicates insertion failure
        FOR N = 1 TO 16                    ;Shift IF weights away from cut
          IW_IF[LT,N]=IW_IF[LT,N]-(Learning_Rate*(A5_Curr[N]-IW_IF[LT,N]))
        ENDFOR
      ENDIF
    ENDIF
  ENDIF
ENDIF

END
```

FIGURE 5.9 **Modified interrupt routine for the online training of the fault detection system (modifications are highlighted in bold).**

5.7.2 Online Training Results

The online training scheme was enabled for a continuous series of leading leg cuts, and the LVQ weights were periodically recorded. Figure 5.10 shows the change in the value of the weights as training progressed.

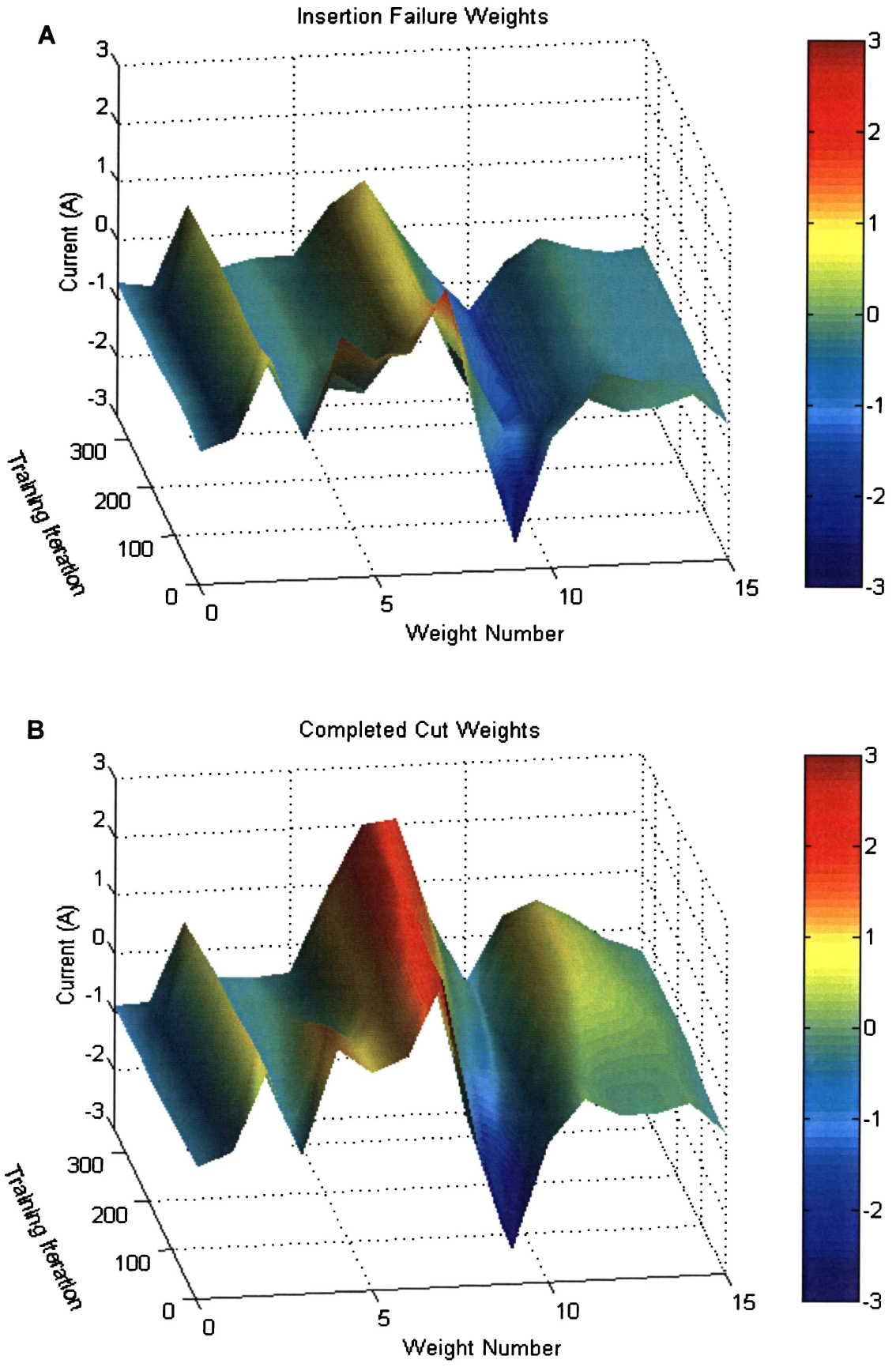

FIGURE 5.10 **Changes in the LVQ weights during online training: a) insertion failure weights b) completed cut weights.**

During testing of the online training scheme, the Y-cutting system was operating with a success-rate of approximately 95%, meaning that 95% of attempted cuts were successfully completed. Because of the low number of insertion failures observed, the neural system was artificially stimulated by initially cutting 50 sheep with the sock-ringer switched off. These cuts are identical to consecutive insertion failures, and produce a significant change to the insertion failure weights during the first 50 training iterations (refer Figure 5.10a). The completed cut weights do not change during this time (refer Figure 5.10b).

The remaining cuts had the sock-ringer switched on, with the majority of the cuts being successfully completed. The insertion failure weights remain reasonably constant during subsequent training. However, the completed cut weights undergo significant modification, with a temporal expansion of the weights as the training progresses (refer Figure 5.10b). This dilation is a direct consequence of the decrease in the cutting speed of the robot since the offline training of the LVQ weights. The speed change means that key features of the motor current signal, such as the peak loading during the knee traversal, occur at a later point in time.

The online training stopped after 350 cuts, which included the initial 50 insertion failures. Figure 5.11 shows the weights at the completion of the online training. Both the insertion failure and completed cut weights have experienced a slight temporal expansion. The new insertion failure weights generally have a lower activation level than the old weights. This reduction is likely to be the result of changes made to the waypoint offsets and orientation angles, which have produced a smoother robot motion along the cut-path.

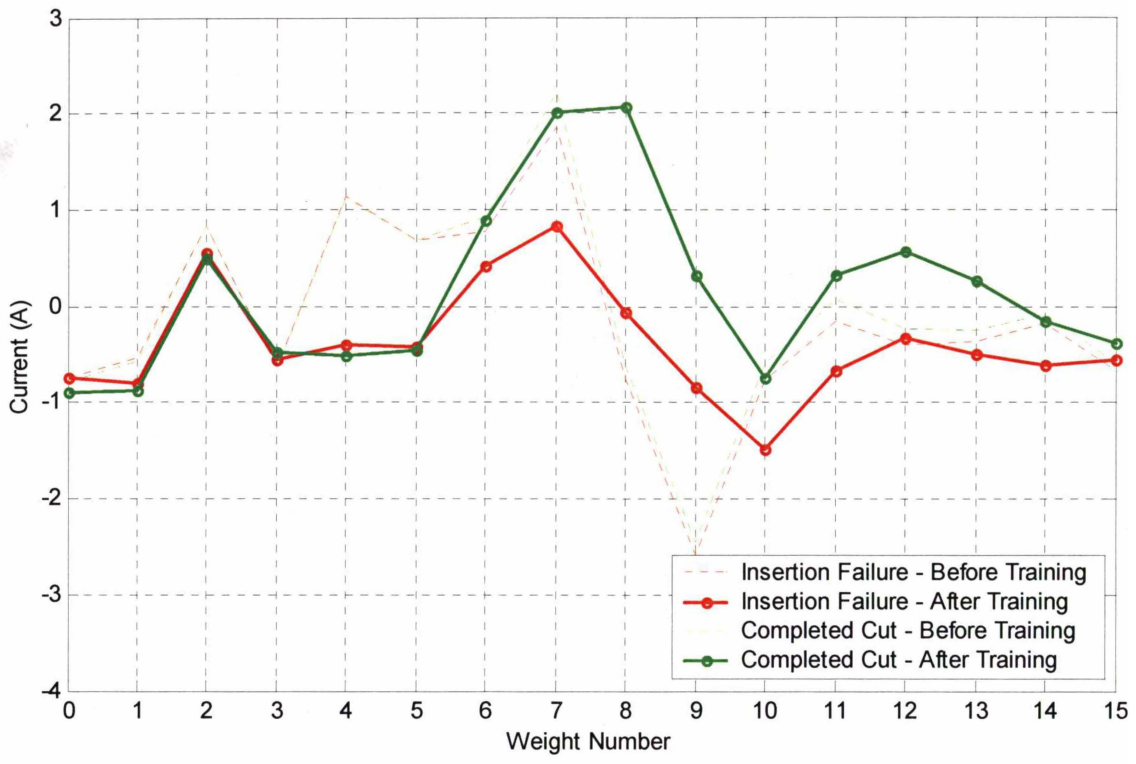

FIGURE 5.11 The leading leg LVQ network weights before and after online training.

5.8 FINAL RESULTS

With the LVQ weights retrained, the fault detection module was tested during normal operation of the Y-cutting system. This test included switching off the sock-ringer occasionally to increase the number of observed insertion failures. The system attempted 502 leading leg cuts, resulting in 322 completed cuts and 180 insertion failures. Of the 180 insertion failures, approximately 100 were real insertion failures that occurred when the sock-ringer was switched on. The LVQ fault detection module successfully classified 100% of the cuts (refer Table 5.5).

TABLE 5.5 Classification success of the LVQ fault detection system.

	Insertion Failures	*Completed Cuts*	*Overall*
Observed Cut Outcomes	180	322	502
Correctly Classified Cuts	180	322	502
Fault Detection Success-Rate	100%	100%	100%

5.9 DISCUSSION

The developed LVQ-based fault detection system has performed exceptionally well. The system successfully classified 98% of the cut data during offline testing. The system has been integrated within the KUKA environment and an online training algorithm has been implemented to allow the neural network weights to be retrained as needed. The final test of the system has resulted in 100% successful classification of 502 cuts.

This fault detection system has future application in signalling to down-stream workers that a particular carcass needs to be manually Y-cut. The system could also allow for the automatic collection of Y-cut performance statistics. These data would be useful for plant management, allowing them to monitor the performance of the automated system remotely. More importantly, the provision of a reliable fault detection method opens the way to the development of an automatic fault correction system. Having detected an insertion failure, the robot can attempt to correct the behaviour that caused the fault, thereby improving the overall performance of the system.

6. PATH OPTIMISATION AND FAULT CORRECTION

The development of the LVQ fault detection scheme provides the continuous and reliable means to observe the outcome of each cut, eliminating the need for a person to compile trial statistics manually. The ultimate progression is to develop some form of automatic path optimisation, which would automate the path tuning process entirely. This path optimisation would also effectively act as a fault correction system.

6.1 PRELIMINARY ANALYSIS OF PATH PARAMETERS

The path-planning algorithm of the Y-cutting system depends on the specification of various waypoint offsets (refer Section 3.4). The first two waypoints of the cut-path are the key for successful insertion since these points control the way that the tool hooks into the opening cut of the sock-ringer. The path parameters pertaining to these two waypoints must be optimised so that the success rate of the Y-cutting system is maximised.

Waypoint WP1 is defined by the three Cartesian coordinates of the insertion point, two Cartesian offsets and three Euler angles for the tool orientation. Waypoint WP2 is defined by three Cartesian offsets and three Euler angles. This means that there are 14 parameters used to define these two waypoints. The speed of the robot as it travels between WP1 and WP2 is also crucial, since this affects the ability of the pelt to feed onto the blades of the tool. This gives 15 parameters in total that must be optimised:

- Insertion point coordinates – *IPX*, *IPY* and *IPZ*
- WP1 Cartesian offsets – *BA1X* and *BA1Y*
- WP1 Euler angles – *BA1A*, *BA1B* and *BA1C*
- WP2 Cartesian offsets – *BA2X*, *BA2Y* and *BA2Z*
- WP2 Euler angles – *BA2A*, *BA2B* and *BA2C*

- Robot speed – *WP2VEL*

The prefix "*BA*" refers to the *B*aseline *A*djustment of the waypoints.

6.1.1 Data Collection

A series of cutting trials were conducted at various points in the parameter space to determine the general nature of the success-rate objective function. The trials began with a set of supposedly optimal parameters following an extended period of manual tuning. The parameters were detuned individually to force the process to perform sub-optimally. Only leading legs were cut during the trials, with the success rate for each cutting trial being the average of approximately 100 cuts.

The brisket measurements for the individual cuts were also recorded during the cutting trials. The leg wand was not operating during the data collection trials. The trailing legs were being manually Y-cut upstream of the automated system and there were concerns from the plant supervisors that the wand would contaminate the meat as it brushed over the pelt. Instead, constant values were used for the two wand sensor measurements *Wand_Measure* and *Wand_Measure_Pos*. Although the use of constant values may have affected the quality of the cut, this was not expected to change the insertion success rate, with the wand measurements having no influence on the position of waypoint WP1 and minimal impact on WP2 (estimated as ±2 mm in both the X and Y directions).

It was unlikely that all of the 15 parameters could be de-tuned during the cutting trials because of time constraints. This restriction forced the prioritisation of the parameters:

Priority 1: Cartesian insertion point coordinates.
Priority 2: Cartesian waypoint offsets.
Priority 3: Robot speed.
Priority 4: Tool orientation angles.

Ultimately, cutting trials pertaining to only six parameters were completed. These parameters were the three Cartesian insertion point coordinates (*IPX*, *IPY* and *IPZ*) and three of the Cartesian waypoint offsets (*BA1Y*, *BA2X* and *BA2Y*).

6.1.2 Heuristic Analysis of Tested Parameters

Previous knowledge of the process can be used to gain an initial understanding of the possible shape of the success-rate objective function. This heuristic analysis is illustrated in Figure 6.1, with each of the six tested parameters dealt with individually.

- *IPX*
  - This parameter gives the lateral position of the insertion point.
  - Variations in the value of *IPX* will be partially negated by the action of the leg-guide, which acts to guide the leg towards the tool-tip. This should produce a plateau of the success-rate objective function near the optimum parameter value (refer Figure 6.1a).
  - The success rate should decrease rapidly for extreme values of *IPX* that do not permit the leg-guide to capture the leg.

- *IPY*
  - This parameter gives the horizontal depth of the insertion point.
  - An increase in the value of *IPY* pushes the tool closer to the leg. However, there is a physical limitation on how far the tool can move in this direction. The tool will eventually clash with the metal spreaders that hang from the conveyor-chain. This limitation appears as a boundary in Figure 6.1b.
  - It is unclear what will happen to the success rate if the tool pushes beyond an optimum position. One consequence is that the force applied by the tool would shift the leg and effectively shift the height of the opening cut upwards. This movement should reduce the success rate.
  - The success rate should decrease rapidly as the tool is pulled away from the optimum position.

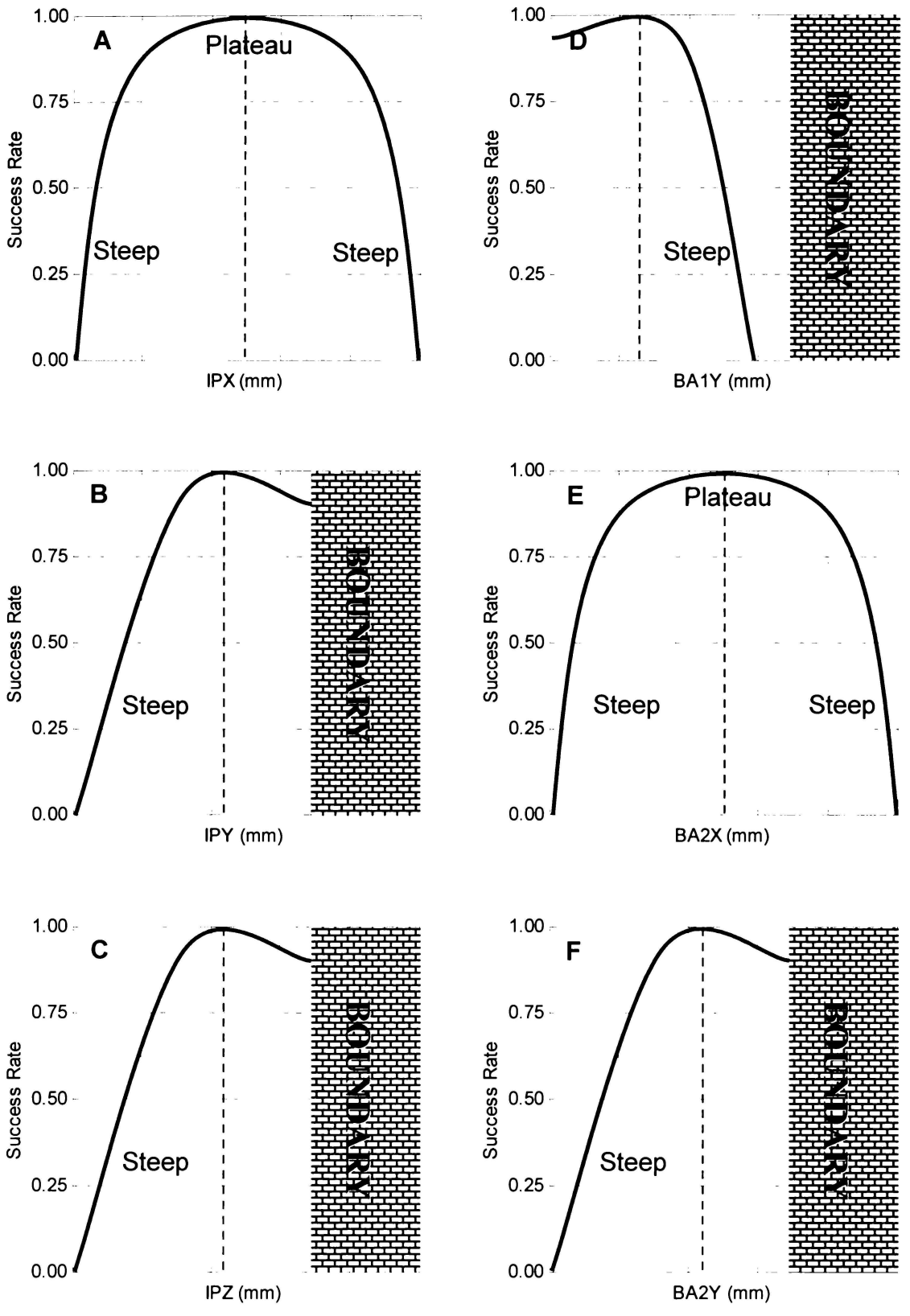

FIGURE 6.1 Heuristic analysis of selected path parameters.

- *IPZ*
  - This parameter specifies the vertical height of the insertion point.
  - The value of *IPZ* should be approximately equal to or slightly above the height of the opening cut, allowing the tool to drag down into the opening cut pocket.
  - If *IPZ* is too low then the success rate should drop away rapidly, since the tool will never be able to hook into the opening cut pocket. However, the decay in the success rate will not be a sudden drop-off because the size of the pocket depends on the breed and age of the individual animal.
  - If *IPZ* is too far above the opening cut then the success rate will also decline, although this depends on the rest of the cut-path (the tool could be directed into the opening cut pocket regardless of the height of *IPZ*). Note that there is also a physical limitation on this parameter (refer Figure 6.1c) with the tool unable to exceed a certain height because of interference with the conveyor-chain spreaders.

- *BA1Y*
  - This parameter offsets waypoint WP1 in the Y-direction. While *IPY* defines the baseline for the cut-path definition, *BA1Y* allows WP1 to vary from a location at one end of this line. Modifying *BA1Y* does not offset the whole cut-path but just WP1.
  - *BA1Y* works in tandem with *BA2Y* to define the hook of the cut-path. As *BA1Y* decreases, WP1 moves away from the foreleg and increases the angle of the cut-path relative to the leg. Although the increase in this angle should increase the chances of the tool hooking into the opening cut pocket, at some point the success rate will drop away once WP1 is too far from the foreleg (refer Figure 6.1d).
  - As *BA1Y* increases, WP1 moves closer to the foreleg and the angle of the cut-path decreases. This angle decrease will produce a decrease in the success rate. There is a physical limitation on the upper value of *BA1Y*, although the location of this boundary depends on the value of *IPY*.

- *BA2X*
  - This parameter offsets waypoint WP2 in the X-direction.

- As with *IPX*, the success rate should fall off either side of the optimum value of *BA2X*. There will be a plateau in the objective function near the optimum value because of the influence of the leg-guide (refer Figure 6.1e). The leg-guide does not retract until the tool reaches WP2.

- *BA2Y*
  - This parameter offsets waypoint WP2 in the Y-direction.
  - *BA2Y* is similar to *BA1Y*, with both of these parameters affecting the hook of the cut-path into the opening cut pocket. The objective function should roughly mirror *BA1Y* since moving *BA2Y* towards the leg increases the angle of the cut-path relative to the leg (refer Figure 6.1f).
  - This parameter may have an upper boundary, depending on whether the tool will interfere with the spreaders at the height of WP2. Note that there is probably also a physical restriction on how hard the tool can push against the foreleg.

6.2 INITIAL DATA ANALYSIS

Following the collection of the cutting trial data, some initial data analysis and modelling is completed to determine the nature of the success-rate objective function.

6.2.1 Process Observations

Figure 6.2 gives the observed success rates for each of the cutting trials, with the heuristic analysis of Figure 6.1 overlaying the data. The observed success rates largely match the expected form of the objective function. The locations of the "optimum" parameters that served as the basis of the de-tuning process are included for reference purposes.

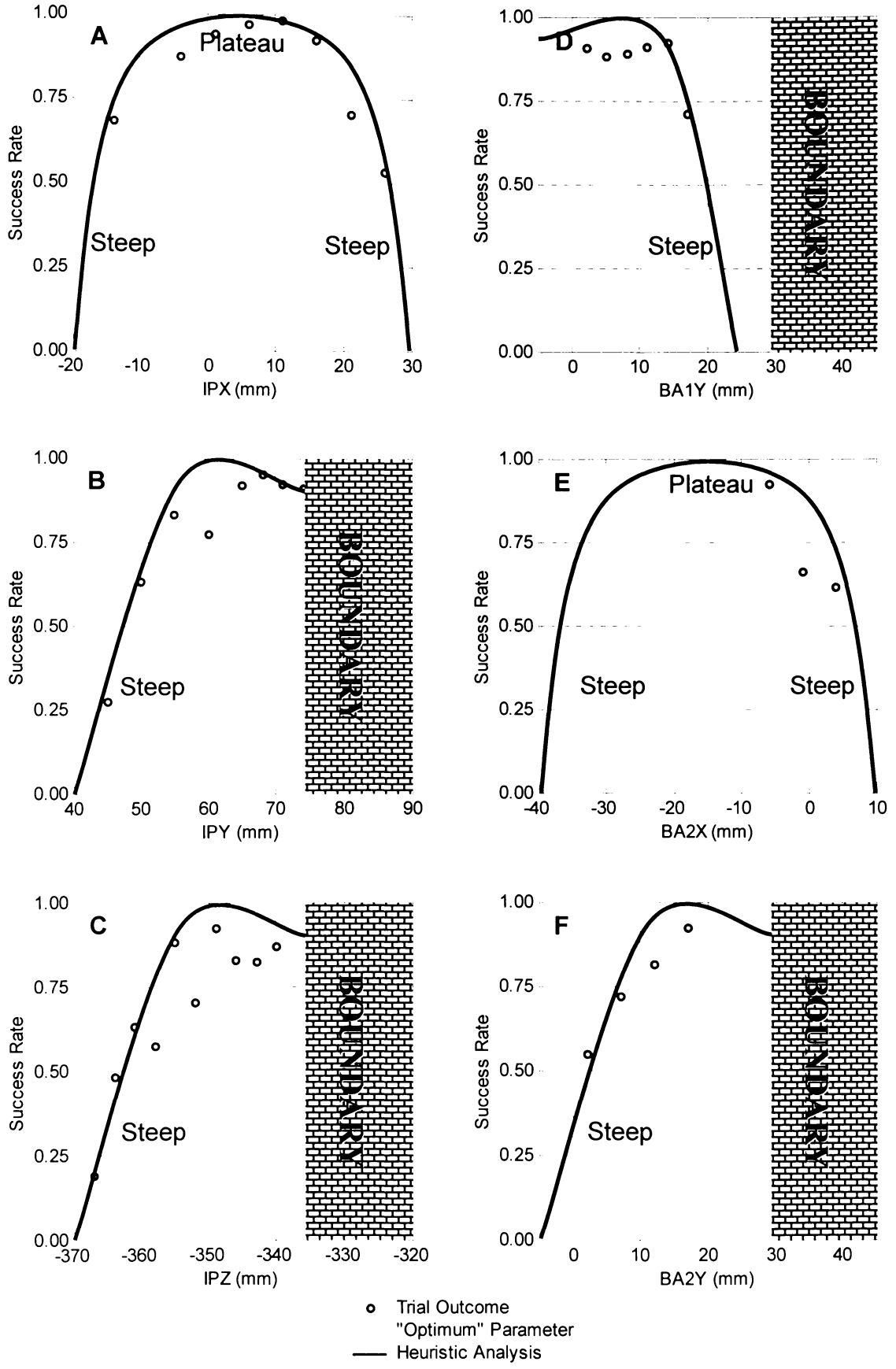

FIGURE 6.2 **Heuristic analysis overlaid with trial outcome data.**

Several different values of *IPX* result in a higher success rate than the supposedly optimum parameter of 16 mm. One value of *IPY* gives a better performance than the optimum parameter of 71 mm. Both of these results indicate that the manually optimised parameters have not been sufficiently refined.

Note that *BA2X* and *BA2Y* have had only three and four cutting trials respectively. The limited amount of data for these parameters was a result of insufficient time to complete as many cutting trials as had been planned. There were only limited periods of time when the cutting trials could be conducted due to the commercial requirements of the plant. The cutting trials for these two parameters varied the supposedly optimum values in only one direction. These data appear to match part of the heuristic curves, although further cutting trials would be required to determine the nature of the objective function in the opposite direction.

6.2.2 Brisket Measurement Correlation

Figure 6.3 shows the brisket measurement for all of the individual Y-cuts of the cutting trial. There is no immediately discernable relationship between the brisket measurement and the observed success rate, with the spread of insertion failures in each trial appearing to be random. This observation is confirmed by calculating the correlation coefficient between the brisket measurement and cut outcome for each trial. The correlation coefficient R_{ij} between variables x_i and x_j is defined as:

$$R_{ij} = \frac{\sigma_{ij}^2}{\sqrt{\sigma_{ii}^2 \sigma_{jj}^2}} \tag{47}$$

where σ_{ij}^2 is the covariance between variable x_i and x_j, and σ_{ii}^2 and σ_{jj}^2 are the variances of x_i and x_j. The correlation coefficient gives a measure of the strength of the linear relationship between variables (The MathWorks, Inc., 2000). A coefficient of zero indicates that the variables are uncorrelated and a coefficient of one indicates that the variables are equivalent. A negative correlation coefficient indicates that the linear relationship has a negative slope. The mean trial correlation coefficient for the brisket measurement and cut outcome data is -0.051, with a minimum coefficient of -0.3137

and a maximum of 0.2298. This indicates that there is a very poor linear correlation between the brisket measurement and the cut outcome.

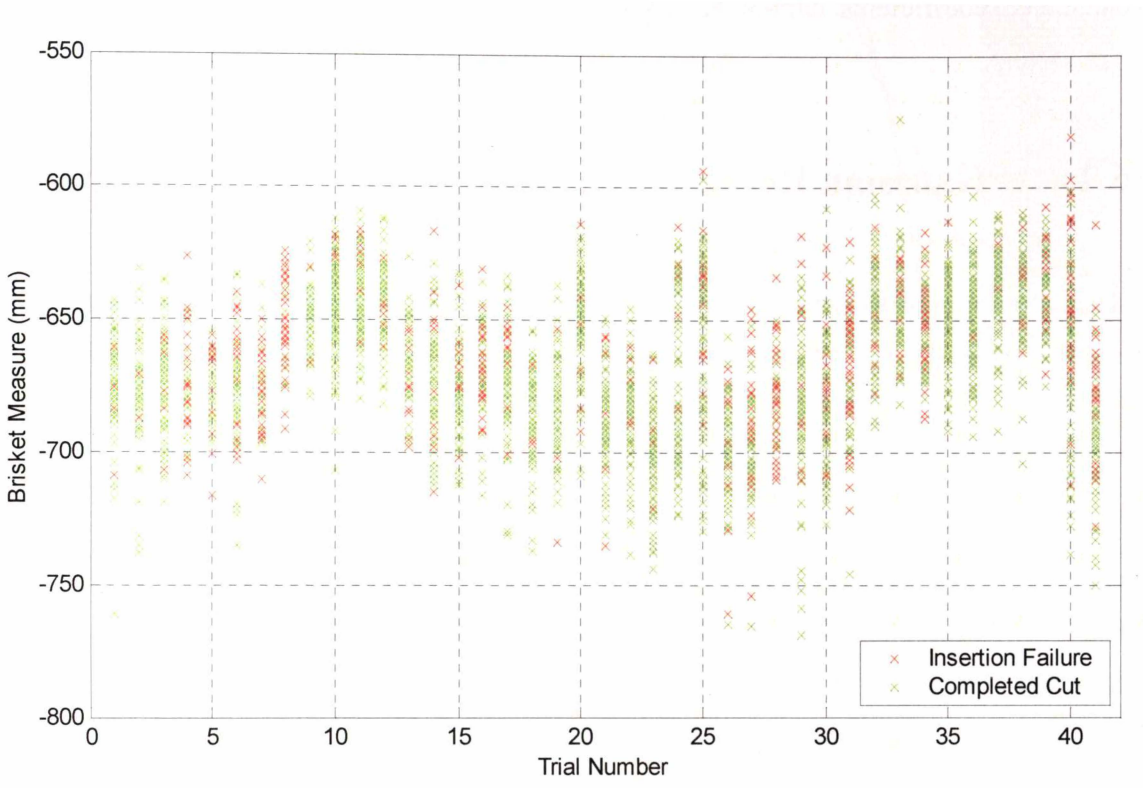

FIGURE 6.3 Observed brisket measurement for each cutting trial.

6.2.3 Path Parameter Correlation

The correlation coefficients are also determined for the tested path parameters (refer Table 6.1). These coefficients correspond to the strength of the linear relationship between path parameters and the success rate of the individual cutting trials. The squares of the correlation coefficients R^2 are also included, indicating the proportion of the variance explained by the linear relationship.

TABLE 6.1 Correlation coefficients for the tested path parameters.

	IPX	*IPY*	*IPZ*	*BA1Y*	*BA2X*	*BA2Y*
Correlation Coefficient	-0.17	0.30	0.53	-0.24	-0.16	0.16
R^2	0.03	0.09	0.28	0.06	0.03	0.03

However, the correlation coefficient only gives an indication of the strength of the linear relationship between variables. From the comparison of the heuristic analysis and

trial outcomes in Section 6.2.1, it seems likely that the relationship between the path parameters and the success rate is nonlinear in nature. This observation means that the calculated coefficients cannot be used to solely quantify the correlation for the path parameters.

6.2.4 Gaussian Regression

Based on the heuristic analysis of the path parameters, a Gaussian regression is used to model the success rate of the individual parameters. The regression is of the form:

$$f(x_i) = e^{-\frac{(x_i-\mu)^2}{2\sigma^2}} \tag{48}$$

where x_i is the path parameter, and μ and σ are the mean and standard deviation of the Gaussian. The software package STATISTICA is used to obtain an estimate of the unknown variables μ and σ. The regression minimises a loss function:

$$Loss = \left(f(x_i) - \hat{f}(x_i)\right)^2 \tag{49}$$

where $f(x_i)$ is the observed value of the Gaussian regression and $\hat{f}(x_i)$ is the predicted value from the cutting trial. Table 6.2 gives the estimated Gaussian mean and standard deviation variables. The table also gives the final value of the loss function and the R^2 value for the goodness of the regression fit. Figure 6.4 shows the Gaussian regressions, with the heuristic curves included for comparison.

TABLE 6.2 The Gaussian mean and standard deviation from the regression of the observed success rate for individual path parameters.

Parameter	*Mean* μ *(mm)*	*Standard Deviation* σ *(mm)*	*Loss*	R^2
IPX	4.8	20.0	0.013	0.933
IPY	66.8	15.4	0.050	0.864
IPZ	-347.0	12.4	0.139	0.704
BA1Y	6.3	13.8	0.033	0.884
BA2X	-15.8	18.9	0.094	0.830
BA2Y	25.7	22.1	0.001	0.992

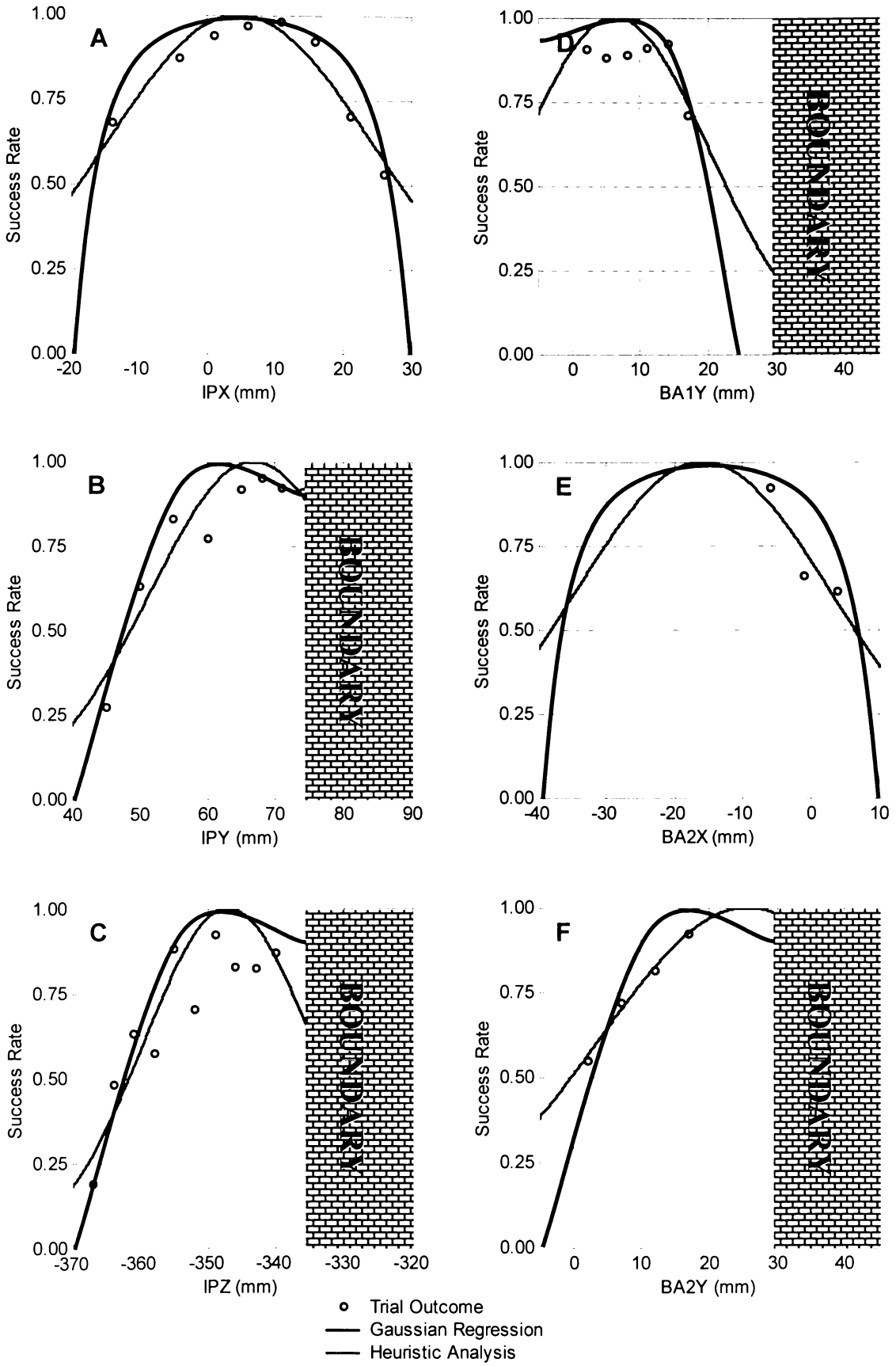

FIGURE 6.4 **Gaussian regression of the success rate from the conducted cutting trials.**

A comparison of the R^2 values in Table 6.1 and Table 6.2 demonstrates that the Gaussian model is a much better fit to the data than the linear model. Figure 6.4a shows that the Gaussian regression provides a good approximation to the success rate variation for *IPX*. This is confirmed in Table 6.2, with the R^2 value of 0.933 indicating that the Gaussian regression accounts for 93.3% of the variability in the observations. The regressions for *IPY* in Figure 6.4b, *IPZ* in Figure 6.4c and *BA1Y* in Figure 6.4d do not match the data points as closely as *IPX*. The regressions for *BA2X* in Figure 6.4e and *BA2Y* in Figure 6.4f are not particularly significant since these are based on only three and four data points respectively. However, these regressions will be adequate for initial off-line development and simulation purposes. The Gaussian regressions also correspond approximately to the shape of the heuristic objective function.

If the regression is an accurate model of the success-rate objective function, then the standard deviation indicates the relative importance of the individual path parameter. The smaller the value of σ then the greater the impact on the success rate if the path parameter is non-optimal. The calculated Gaussian regression indicates that the success rate is most sensitive to changes in the height of the insertion point *IPZ*. The parameter that the success rate is least sensitive to is the offset *BA2Y*.

6.3 DEVELOPMENT OF OPTIMISATION ALGORITHM

As stated in Section 2.4.5, the biggest challenge for the optimisation of the cut-path is the evaluation of the success-rate objective function for a given set of path parameters. This evaluation requires the completion of a number of Y-cuts, from which an estimate of the success rate can be calculated. It is important to ensure that the optimisation process converges to an acceptable solution within an acceptable timeframe, meaning that the number of objective function evaluations must be kept to a minimum.

6.3.1 Line Search Optimisation Algorithm

A line search algorithm developed to optimise the success rate is shown in Figure 6.5. This algorithm is a variation of the sequential random search algorithm described in Section 2.4.4.1 and is based on the current one-dimensional manual tuning process. The

algorithm commences with the supposedly optimal set of n parameters as a starting point for the search. The parameters are ordered according to the estimate of the standard deviation estimates obtained from the Gaussian regressions. Each path parameter is individually offset from the current optimum estimate by a distance λ_j. This offset is proportional to a constant ratio R and the standard deviation σ_i of the path parameter. The ratio R allows for the scaling of the search region based on the distribution estimate, effectively increasing the length of the search line.

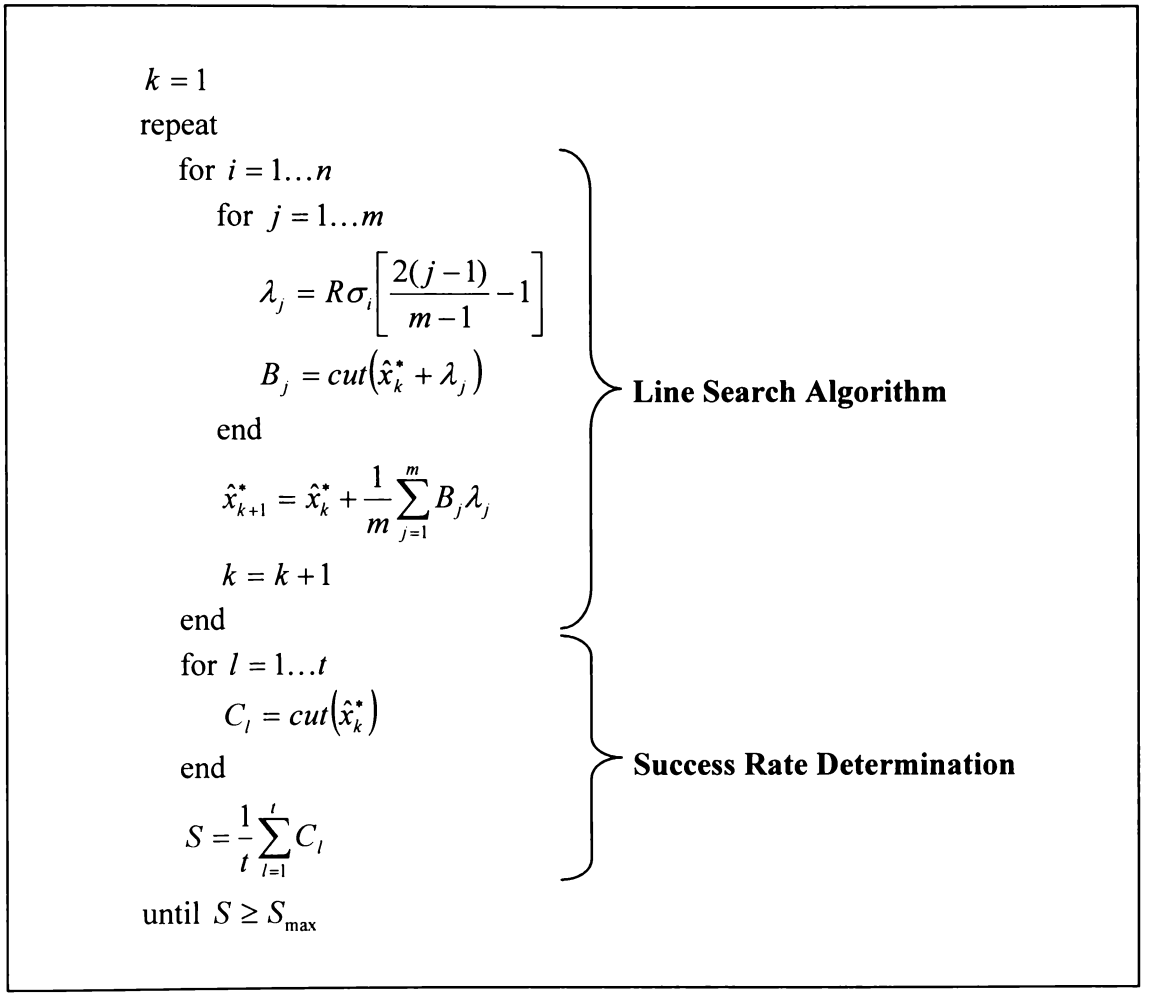

FIGURE 6.5 The line search optimisation algorithm.

A single Y-cut is attempted at m uniformly distributed points along the line, provided that the point is not outside the physical boundaries of the system. For simulation purposes, the Gaussian regression model is used to determine the probability of success for each cut based on the search point. The multivariate Gaussian regression is of the form:

$$f(\boldsymbol{x}) = e^{-\frac{(x-\mu)^2}{2\sigma^2}} \tag{50}$$

where $\boldsymbol{x}$ is a vector of path parameters, and μ and σ are the mean and standard deviation vectors of the Gaussian. The probability $f(\boldsymbol{x})$ is compared with a randomly generated number on the interval (0,1), in order to obtain a binary decision for the cut outcome. If $f(\boldsymbol{x})$ is greater than the random number then the cut is deemed to have been completed, otherwise it is an insertion failure. The cut outcome $cut(x)$ is then given by:

$$cut(x) = \begin{cases} 1 & \text{if completed cut} \\ 0 & \text{if insertion failure} \end{cases} \tag{51}$$

The parameter estimate $\hat{x}_k^*$ moves to the mean location of all of the completed cuts. This line search is repeated for all n path parameters. The algorithm then performs a trial of t Y-cuts with the new parameter estimate. If this cutting trial has a success rate greater than S_{max}, then the line search process concludes, otherwise the line search routine repeats.

6.3.2 Statistically Dependent Parameters

This form of one-dimensional parameter search is valid for statistically independent path parameters. Parameters x_i and x_j are considered statistically independent if $\sigma_{ij}^2 = 0$. For the more general case where the parameters have some level of dependence ($\sigma_{ij}^2 \neq 0$), a search involving only a single parameter is not an efficient method (refer Figure 6.6). This inefficiency is caused by the search direction not being orthogonal to the gradient of the Gaussian objective function, meaning that subsequent line searches can spoil the contribution of previous iterations.

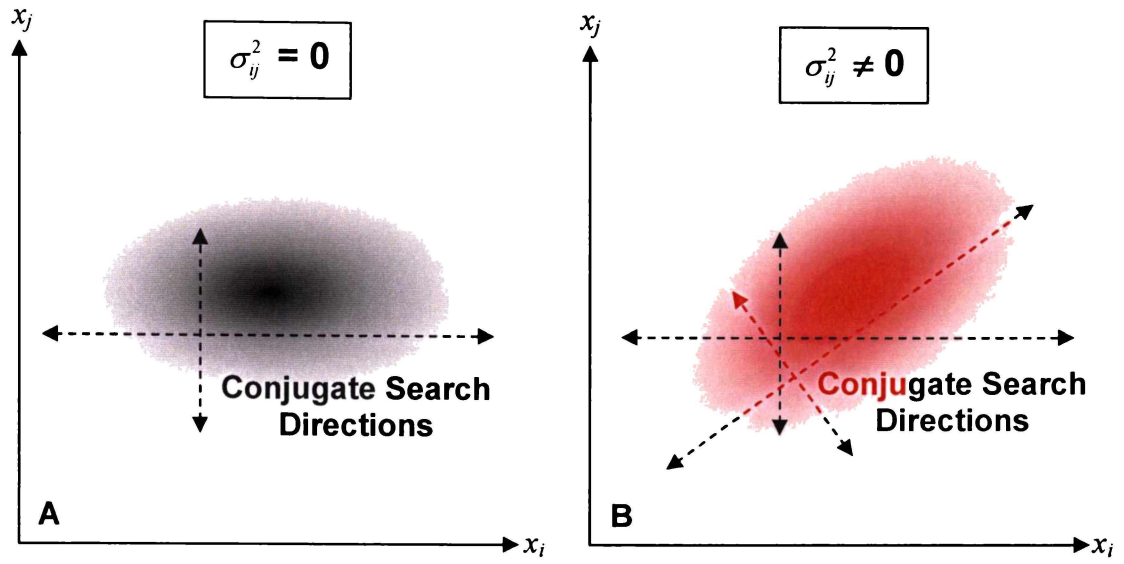

FIGURE 6.6 Conjugate search directions for a) independent and b) dependent parameters.

The use of the eigenvectors and eigenvalues of the covariance matrix ensures that the search directions are orthogonal and aligned with the principal axes of the Gaussian hyper-ellipsoid distribution. The eigenvalues of the covariance matrix determine the length of the line search for each parameter. The eigenvectors give the conjugate direction of the line search for dependent parameters. This means that each line search can span the entire parameter space rather than being restricted to a simple one-dimensional space.

The path parameters are assumed to be independent for initial simulation purposes, since the cutting trials used to develop the Gaussian model are one-dimensional and do not allow for the estimation of the parameter covariance.

6.4 SIMULATION RESULTS

The line-search optimisation algorithm is simulated using MATLAB. This initial simulation uses only the six parameters from the cutting trials to represent the success-rate objective function. This simplified model is used to investigate the convergence characteristics of the developed algorithm, and to determine if the algorithm has the potential to converge faster than the current manual tuning process.

6.4.1 Various Line Densities and Lengths

The convergence speed of the line search algorithm is evaluated for various ratios R and line sample sizes m (refer Figure 6.7). The ratio R is varied from 0.4 to 2.0 in steps of 0.1, and m is varied from 11 to 51 in steps of 10. An odd number is used for m to ensure that a simulated Y-cut is made at the location of the current parameter estimate $\hat{x}_k^*$. A success rate criteria $S_{\max}$ of 0.98 (98%) is used to terminate the search algorithm, with this criteria matching the success rate commonly demanded by the meat industry (refer Section 3.1.4). The size of the success rate trial sample t is 100 Y-cuts. The convergence speed is the total number of simulated Y-cuts required by the algorithm to meet the success rate criteria, and is the average of 100 runs.

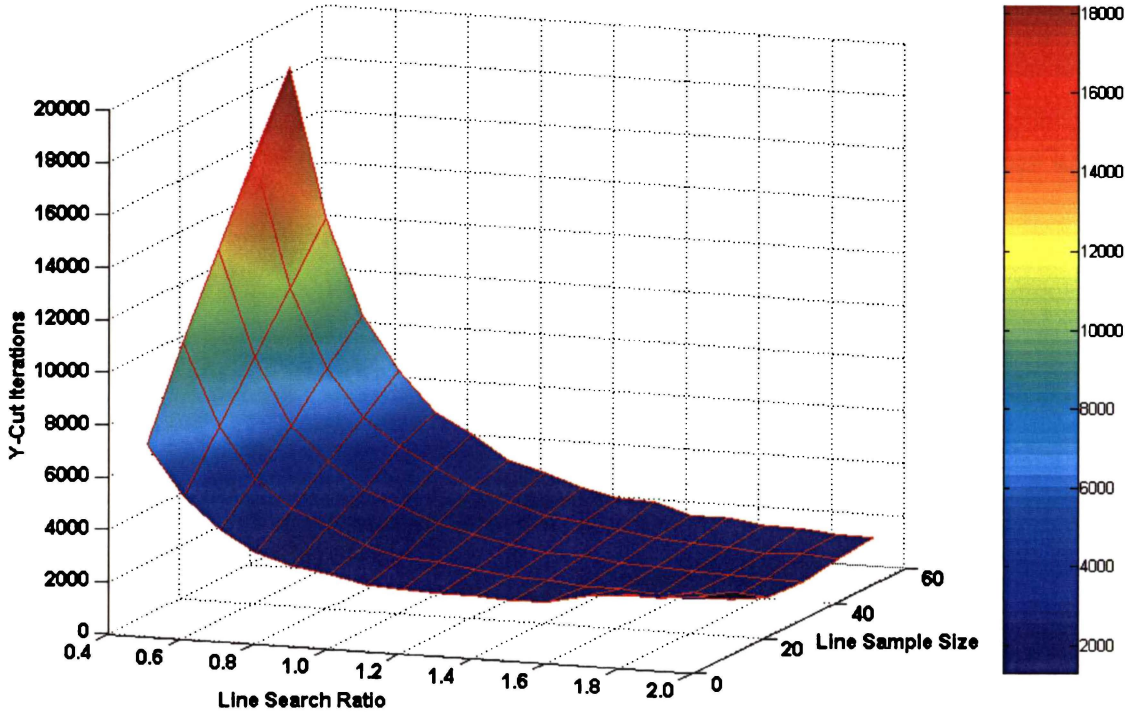

FIGURE 6.7 The convergence speed of the line search algorithm.

Figure 6.7 shows that convergence is slower for smaller values of R and larger values of m. The fastest convergence occurs for R = 1.7 and m = 21, with the mean number of required Y-cuts being 1320. All of the tested line sample sizes are capable of converging in less than 2000 iterations, depending on the line search ratio that is used.

There is a linear relationship between the number of Y-cut iterations and the line sample size for a given line search ratio. This is an expected result since an increasing sample size corresponds to an increase in the density of sampling points along the search line. While a higher sample density will theoretically improve the estimation of the mean location of the completed cuts, this density increase does not improve the convergence speed of the algorithm. However, it is noted for $m = 11$ that when $R > 1.5$ the number of iterations required for each run becomes erratic and the mean increases slightly. This is a result of the sample density being too low for smooth convergence, leading to the observed instability.

6.4.2 Improved Success Rate Determination

The convergence speed of the line search algorithm can be improved by modifying the sampling procedure used to determine the success rate at the current parameter estimate. The success rate sampling of t Y-cuts is halted prematurely if the number of observed insertion failures exceeds the number permitted by the criterion S_{max}. This modification stops the algorithm from unnecessarily collecting samples if the success rate is destined to be less than S_{max}. The modified line-search optimisation algorithm is shown in Figure 6.8.

The algorithm is tested using the same parameters as before ($R = 0.4,\ldots,2.0$, $m = 11,\ldots,51$, $S_{max} = 0.98$, $t = 100$), with results averaged over 100 runs. Figure 6.9 shows the convergence of the modified algorithm with the performance of the original algorithm overlaid. The fastest convergence occurs for $R = 1.6$ and $m = 11$, with the mean number of Y-cuts being 960. A similar convergence rate occurs for several different combinations of R and m. The convergence of the modified algorithm is significantly faster than the original algorithm, with the modified algorithm typically requiring 25% fewer Y-cuts. Because of this improvement, the modified algorithm is used for all subsequent testing.

$k = 1$
repeat
Line Search Algorithm
...
for $l = 1 \ldots t$
$C_l = cut\left(\hat{x}_k^*\right)$
if $\sum_{i=1}^{l} C_i < (1 - S_{\max})t$ then
break
end
end
$S = \frac{1}{t}\sum_{l=1}^{t} C_l$
Success Rate Determination
until $S \geq S_{\max}$

FIGURE 6.8 The modified line-search optimisation algorithm.

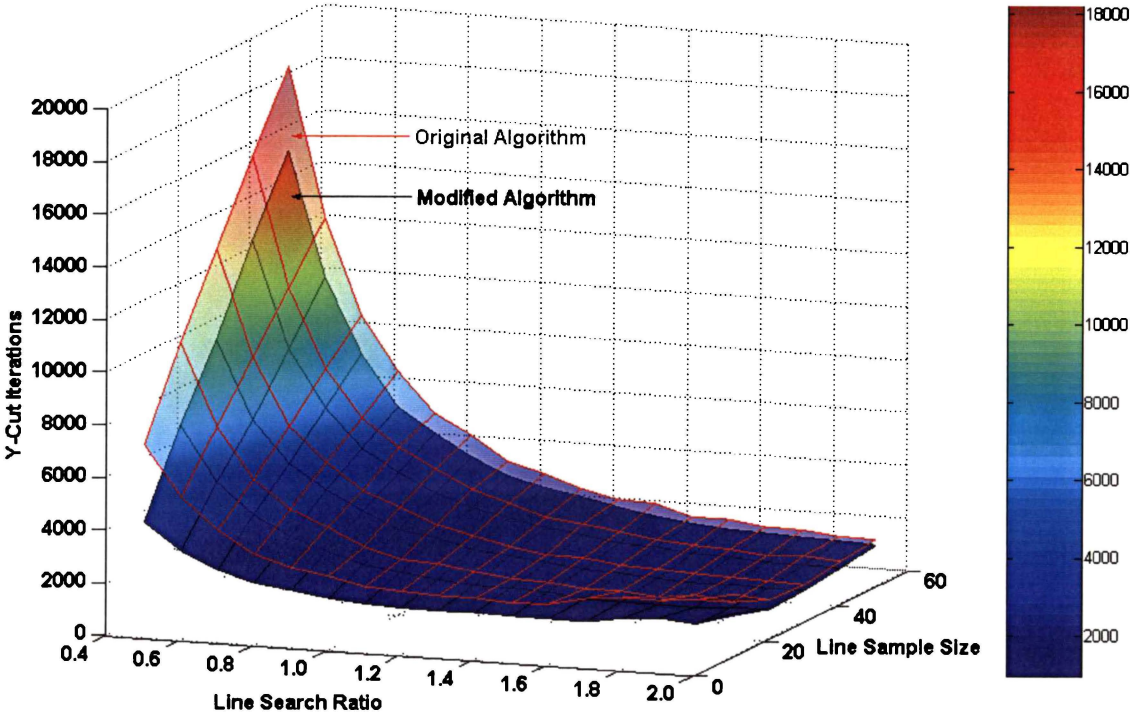

FIGURE 6.9 The convergence speed of the modified line search algorithm.

6.4.3 Different Success Criteria and Sampling

Different success rate criteria S_{max} and trial sizes t for the determination of the success rate are tested. Two values of S_{max} (0.98 and 1.00) and two values of t (100 and 200) are used. One possible advantage of using S_{max} = 1.00 is that the success rate determination is terminated if a single insertion failure is observed, rather than having to wait for three insertion failures when S_{max} = 0.98. This stronger criterion could be useful if the current parameter estimate is sub-optimal. The modified search algorithm is used with line sample size m = 21. Results are averaged over 100 runs. Figure 6.10 shows the convergence for the four permutations of S_{max} and t.

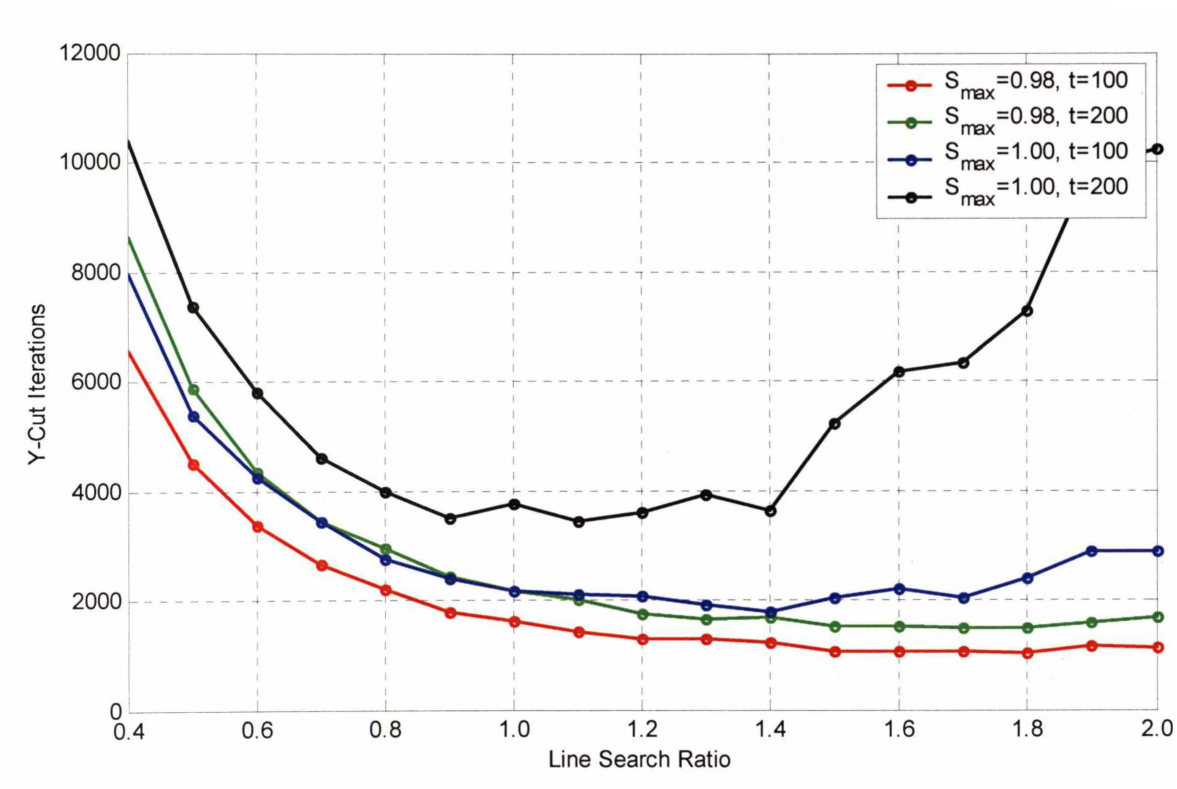

FIGURE 6.10 The effect of varying the success rate criteria S_{max} and trial size t on the convergence speed of the modified line search algorithm.

The permutation that converges the fastest is the case S_{max} = 0.98, t = 100, which is the weakest criteria of the four tested. The slowest converging case is for S_{max} = 1.00, t = 200, which is the strongest criteria. This case also has a steady increase in the required number of Y-cuts when the ratio R is greater than 1.4. The lower sample density causes the increase when R is large, and is exacerbated by the stronger termination criteria of S_{max} and t. The convergences of the two intermediate cases (S_{max} = 0.98, t = 200 and

S_{max} = 1.00, t = 100) are almost identical for $R < 1.4$. All four of the tested permutations have an increase in the convergence time for larger values of R, with the rate of increase being commensurate with the strength of the termination criteria.

Figure 6.11 shows the residual error for the four test cases. The residual error is defined to be the Euclidean distance between the parameter estimate at the conclusion of the search algorithm and the optimum parameter vector. The permutation with the smallest residual error is the strongest criteria S_{max} = 1.00, t = 200, with an average error of 2.4 mm. The permutation with the largest residual error is the weakest criteria S_{max} = 0.98, t = 100, with an average error of 4.3 mm.

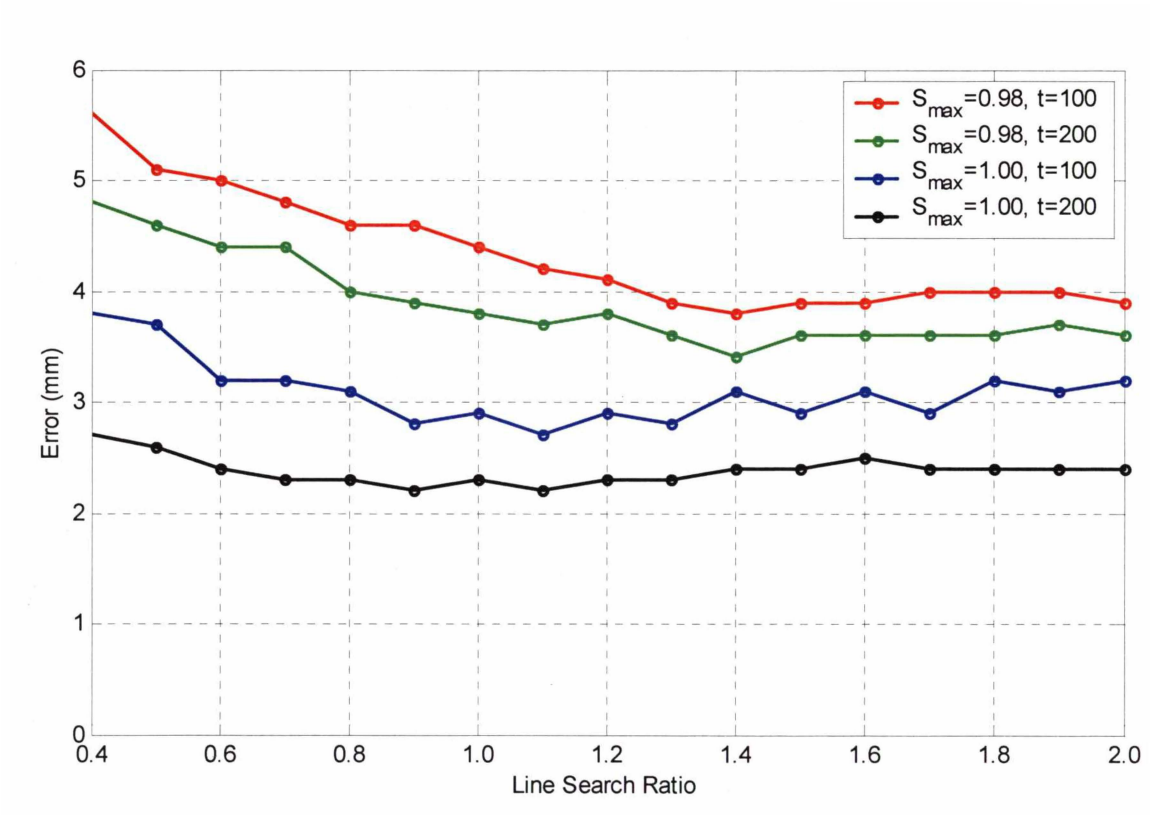

FIGURE 6.11 The effect of varying the success rate criteria S_{max} and trial size t on the residual error for the modified line search algorithm.

6.4.4 Continuous Optimisation

The optimisation algorithm is expanded to allow for continuous optimisation of the parameter estimate (refer Figure 6.12). This optimisation is intended to work in parallel with the normal operation of the Y-cutting system. The success rate of the system is

continually monitored after the conclusion of the line search routine, becoming the average of the t most recent cuts. If the success rate S drops below S_{max} then the search algorithm is reapplied to adjust the parameter estimate.

repeat
$l = l + 1$
$C_l = cut(\hat{x}_k^*)$
$S = \frac{1}{t}\sum_{i=l-t}^{l} C_i$
if $S < S_{max}$ then
{Apply Line Search Algorithm}
end
until $l = \infty$

FIGURE 6.12 Continuous optimisation algorithm.

Figure 6.13 shows the residual error and success rate from a representative simulation with R = 1.0, m = 21, S_{max} = 0.98 and t = 100. The simulation runs for 15000 Y-cuts, which is approximately equivalent to the number of carcasses processed by a typical meat-plant in a five-day working week. The line search algorithm terminates after 1102 Y-cuts, with the error reducing from 19.4 mm to 4.0 mm in this time. The search algorithm is reapplied on five occasions during the remainder of the simulation whenever the success rate falls below S_{max}. The line search part of the optimisation algorithm required 2420 of the 15000 simulated Y-cuts, although 600 of these cuts are used for the determination of the success rate before the termination of the line search. If these 600 cuts are included, then 13180 iterations are part of the normal Y-cutting operation. The final error at the conclusion of the simulation is 2.4 mm.

During the initial application of the line search algorithm, the success rate is established periodically via the algorithm cutting trials of t Y-cuts, producing step-like changes in the success rate (refer Figure 6.13b). Early termination of the success rate determination due to an excessive number of insertion failures can produce large dips in the success rate. Once the success rate exceeds S_{max} and the search algorithm

terminates, then the success rate fluctuates according to the average of the t most recent cuts.

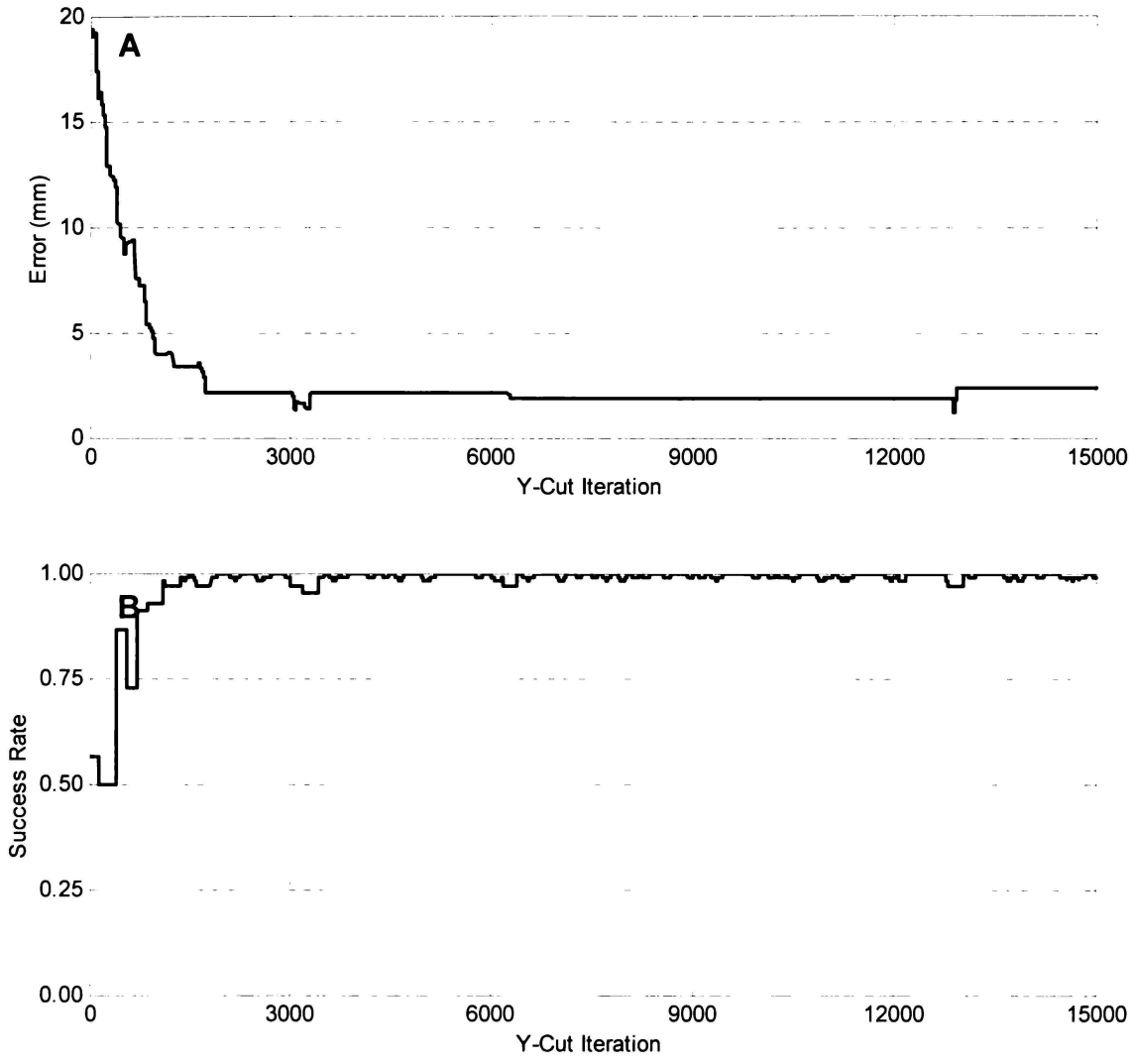

FIGURE 6.13 Plots of a) the residual error and b) the success rate for the continuous optimisation algorithm.

6.4.5 Step Response of the Optimisation Algorithm

It is important that the optimisation algorithm is able to track changes in the optimum parameter vector over time. This tracking ability would allow the Y-cutting system to optimise its performance automatically: responding to sudden parameter fluctuations or to gradual process changes due to seasonal stock variations.

The insertion point height *IPZ* is selected for modification during this simulation since this parameter has the smallest standard deviation and hence the greatest effect on the success rate. The step response of the optimisation algorithm is tested by changing *IPZ* from -347.0 mm to -337.0 mm after 9000 Y-cut iterations. This modification to *IPZ* could occur in the real Y-cut system if maintenance staff decided to raise the height of the sock-ringer. The optimisation algorithm uses R = 1.0, m = 21, S_{max} = 0.98 and t = 100, and the simulation runs for 15000 Y-cuts. The initial estimate of *IPZ* is -349.0 mm, which corresponds to the optimum value identified during the manual tuning process.

Figure 6.14 shows the response of the algorithm to the step change in *IPZ*. The line search terminates after 1886 Y-cuts with a residual error of 4.4 mm. This error reduces to 1.3 mm during subsequent retuning (refer Figure 6.14a). The step change to *IPZ* after 9000 iterations produces an immediate decrease in the success rate (refer Figure 6.14b) and an increase in the error, and forces the restart of the line search part of the optimisation algorithm. The line search terminates after another 2131 Y-cuts with a residual error of 1.5 mm. Figure 6.14c shows the changes in the estimate of *IPZ* during the simulation. The estimate initially settles to a level very close to the optimum value. Following the step change after 9000 Y-cut iterations, the estimate gradually shifts back to the level of the optimum value.

6.4.6 Ramp Response of the Optimisation Algorithm

The same test is undertaken using a ramp change to a parameter. The insertion height *IPZ* changes from -337.0 mm to -347.0 mm over the period of the simulation. This form of parameter change could occur if the height of the sock-ringer slowly dropped over time, although this is considered an unlikely possibility. The magnitude of the parameter change is also considered excessive, but will be useful to demonstrate the tracking ability of the optimisation algorithm. The algorithm again uses R = 1.0, m = 21, S_{max} = 0.98 and t = 100, and the simulation runs for 15000 Y-cuts. The initial estimate of *IPZ* is again -349.0 mm. Figure 6.15 shows the response of the algorithm to the ramp change in *IPZ*.

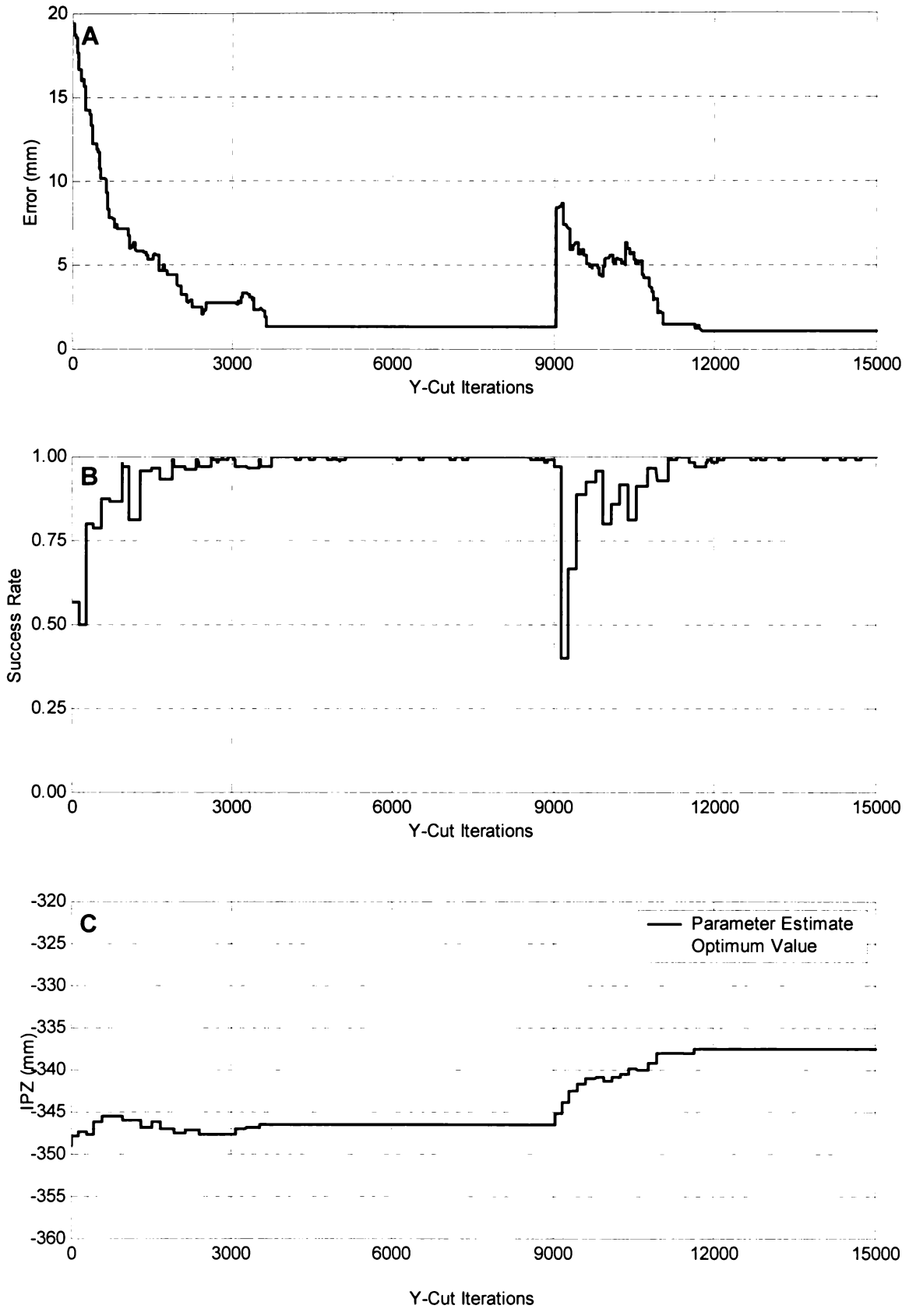

FIGURE 6.14 **The response of the optimisation algorithm to a step change in the optimum value of parameter *IPZ*; a) residual error, b) success rate and c) parameter estimate.**

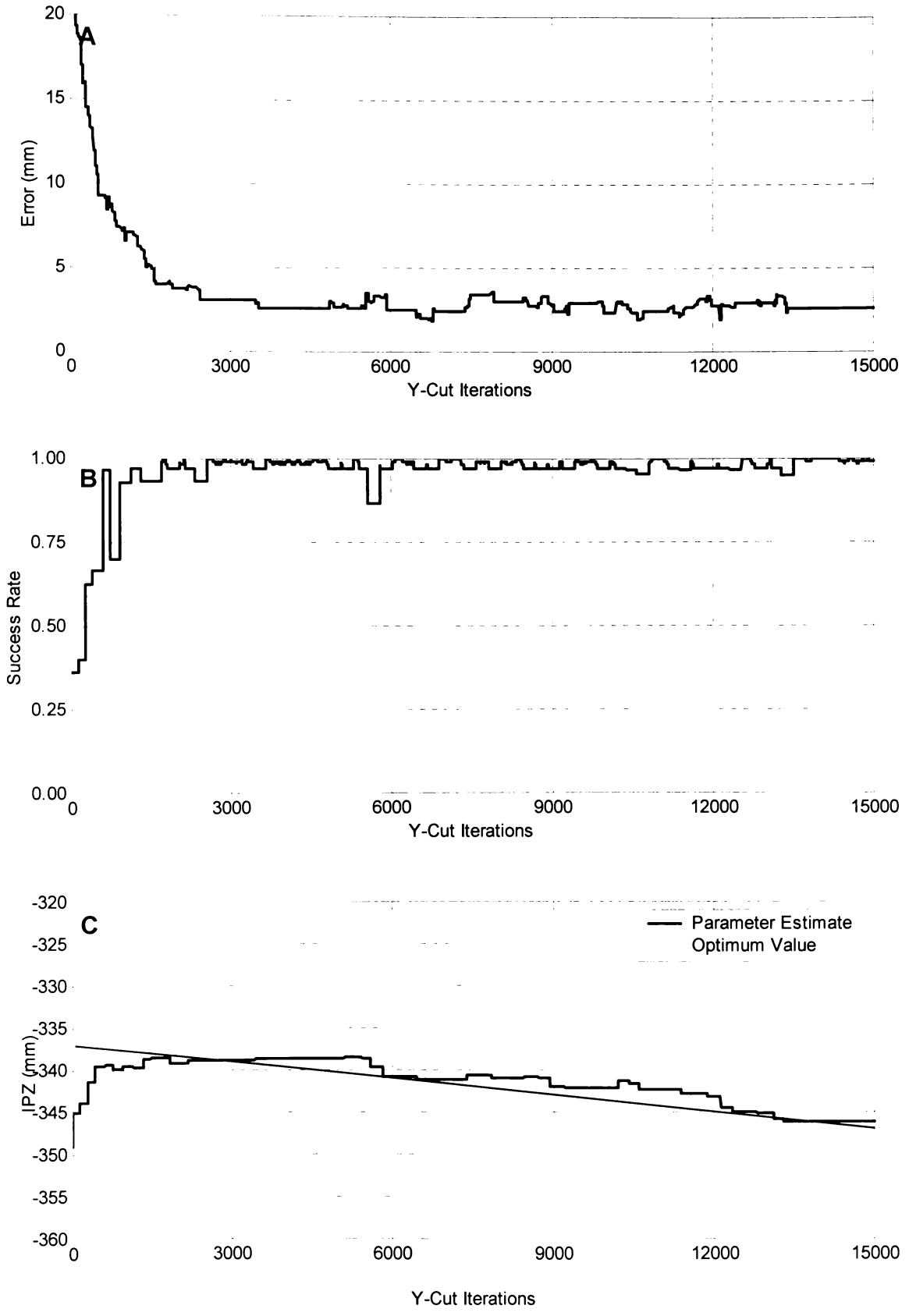

FIGURE 6.15 **The response of the optimisation algorithm to a ramp change in the optimum value of parameter *IPZ*; a) residual error, b) success rate and c) parameter estimate.**

The line search terminates after 1703 Y-cuts with a residual error of 4.0 mm. The line search is reapplied 22 times during the remainder of the simulation, with the error having an average value of 2.7 mm during this time. Figure 6.15c shows the change to the parameter estimate during the simulation. The estimate tracks the decaying optimum value of *IPZ* effectively. Despite the constant drift of *IPZ* and the regular application of the line search algorithm, over 60% of the iterations are part of the normal operation of the Y-cutting system. This is a promising result, given that the severity of the parameter magnitude change (10 mm during a week of operation) is considered unlikely.

6.4.7 Variable vs. Constant Line Search Length

While the assumption made in the initial development of the line search algorithm was that the search region should be scaled by the estimate of the standard deviation, the validity of this scaling regime has not been tested. It may be possible that a search using a line of constant length is just as effective as a variable line length. The performance of a constant line search is established to determine if there is a difference between the two methods. A constant length of 17.1 mm is used to define the length of each line search, with this length being the average of the standard deviations for the six parameters. This choice for the length of the search line is arbitrary since the line is also scaled by the ratio R during this test. R is again varied from 0.4 to 2.0 in steps of 0.1, and m is varied from 11 to 51 in steps of 10. The termination criteria of the algorithm uses $S_{max} = 0.98$ and $t = 100$.

Figure 6.16 gives the convergence of the line search algorithm for both a constant line length and the standard deviation-dependent variable line length of Section 6.4.2. The variable method converges consistently faster than the constant method. Figure 6.17 shows the difference in the number of Y-cut iterations between the constant and the variable methods. The variable method takes an average of 13.7% fewer Y-cuts to converge than the constant method. While this result indicates that a line length based on the parameter distribution improves the convergence of the algorithm, the relatively small decrease in performance for the constant line length means that this method could still be used if the parameter distributions are unknown.

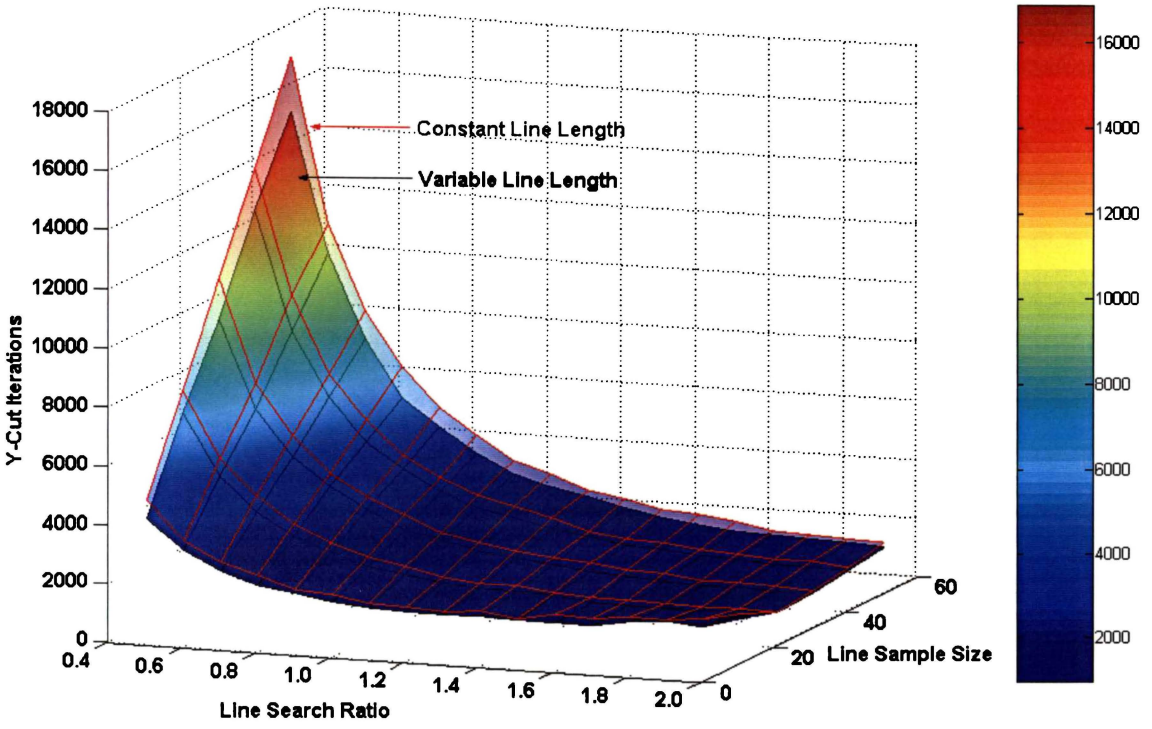

FIGURE 6.16 Convergence speed of the optimisation algorithm with variable and constant line-search lengths.

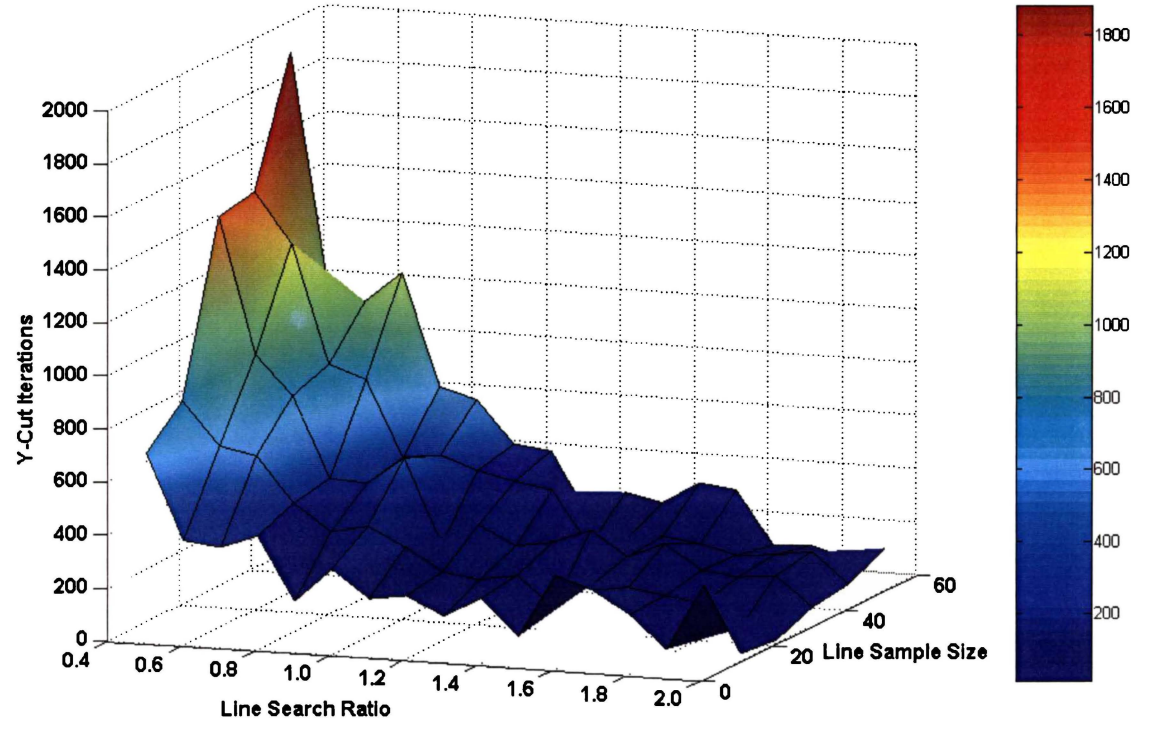

FIGURE 6.17 Difference between the convergence of the constant and variable line search algorithms.

6.5 SIMULATION WITHIN FULL PARAMETER SPACE

Previous testing has all been based on a simplified model using only the six path parameters pertaining to the cutting trial data. However, up to fifteen parameters need to be optimised to maximise the Y-cutting success rate of the real system (refer Section 6.1). Therefore, the remaining nine parameters must be estimated to simulate the full performance of the optimisation algorithm.

6.5.1 Estimation of Remaining Path Parameters

Table 6.3 gives the mean and standard deviation values used to describe the fifteen path parameters. The estimated means are the supposedly optimal parameters resulting from the initial manual tuning phase of the Goulburn installation (refer Table 6.2). A standard deviation of 20 mm is selected for *BA1X* since this parameter is expected to have a similar influence as *IPX* (σ = 20.0 mm) and *BA2X* (σ = 18.9 mm). The standard deviation for *BA2Z* is estimated as 15 mm, being of a similar magnitude to the other Cartesian offsets. The six Euler angle parameters are assigned identical standard deviations of 5°, which is based of observation of the Y-cutting process. The parameter *WP2VEL* has a standard deviation of 20%, which is specified as a percentage of the maximum speed of the robot and is also based on process observations.

6.5.2 Simulation Using Full Set of Parameters

The performance of the optimisation algorithm on the expanded set of path parameters is established using the same procedure as before (R = 0.4,...,2.0, m = 11,...,51, S_{max} = 0.98, t = 100). Figure 6.18 gives the convergence of the algorithm for both the full set of fifteen parameters and the previous results for six parameters. It clearly takes the algorithm a longer time to converge to an optimal solution for the expanded set of parameters. The fastest convergence occurs for R = 1.0 and m = 21, with the mean number of required Y-cuts being 4760. This equates to approximately 1.5 days of tuning and is a factor of 5.0 slower than the fastest convergence of the algorithm for the six-parameter case (960 for R = 1.6 and m = 11).

TABLE 6.3 The Gaussian mean and standard deviation for all 15 path parameters (estimated values in bold).

Path Parameter	*Mean*	*Standard Deviation*
Insertion Point Coordinates		
IPX	4.8 mm	20.0 mm
IPY	66.8 mm	15.4 mm
IPZ	-374.0 mm	12.4 mm
WP1 Offsets and Angles		
BA1X	**-3.0 mm**	**20 mm**
BA1Y	6.3 mm	13.8 mm
BA1A	**0°**	**5°**
BA1B	**-20°**	**5°**
BA1C	**6°**	**5°**
WP2 Offsets and Angles		
BA2X	-15.8 mm	18.9 mm
BA2Y	25.7 mm	22.1 mm
BA2Z	**0.0 mm**	**15 mm**
BA2A	**0°**	**5°**
BA2B	**-20°**	**5°**
BA2C	**13°**	**5°**
Robot Speed		
WP2VEL	**75%**	**20%**

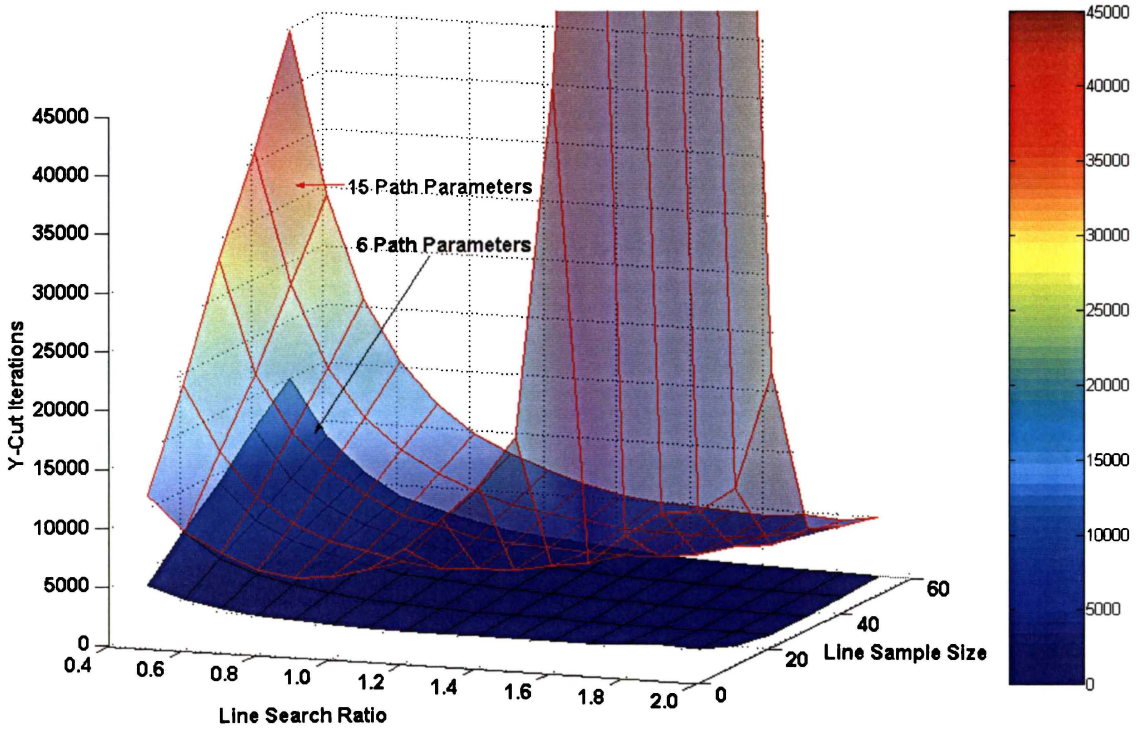

FIGURE 6.18 Convergence of the line search algorithm for the optimisation of 15 path parameters.

If the convergence rates are compared for the same values of R and m, then the expanded set of parameters takes approximately 3 times longer to converge. This is illustrated in Figure 6.19, which shows the ratio of the convergence times between the fifteen-parameter and the six-parameter cases. The only part of the surface that does not indicate the three-fold increase in convergence times is for larger values of R and smaller values of m, where the convergence ratio rapidly increases. The lower right-hand part of the plot in Figure 6.19 is not shown because the convergence ratio is much greater than the maximum scale of the graph. The lower sample density for these values of R and m causes the increase in convergence times. In the case of the expanded set of parameters, there is a very large increase in convergence times within this lower density region. This increase can be seen in Figure 6.18, with the number of Y-cut iterations dramatically rising in the front right-hand corner of the convergence surface.

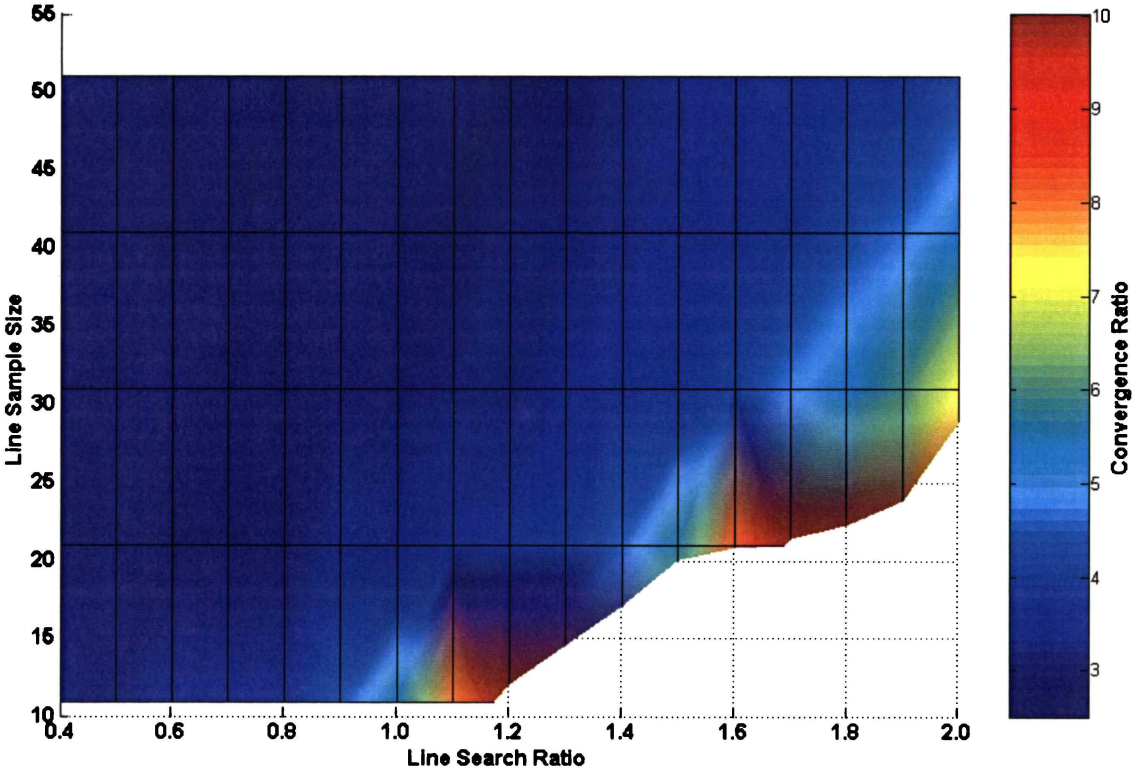

FIGURE 6.19 **The ratio of the convergence for fifteen parameters to the convergence for six parameters.**

The three-fold increase in the convergence time is slightly greater than the ratio of the number of parameters 15:6 = 2.5. This indicates that the increase in convergence times is more than just an increase in the number of parameters n that the algorithm must search. There is also an increase in the number of iterations of the line search part of the algorithm required to meet the terminating success rate criteria.

6.5.3 Continuous Optimisation

The continuous optimisation procedure (refer Section 6.4.4) is applied to the expanded set of parameters. Figure 6.20 shows the residual error and success rate from a representative simulation with R = 1.0, m = 21, S_{max} = 0.98 and t = 100. The simulation runs for 15000 Y-cuts.

The line search algorithm terminates after 4712 Y-cuts, with the error reducing from 23.3 mm to 3.8 mm in this time. The search algorithm is reapplied on thirteen occasions during the remainder of the simulation. The line search part of the optimisation algorithm required 11341 of the 15000 simulated Y-cuts, which is a much higher proportion than was observed for the six-parameter case (1820 out of 15000 simulated Y-cuts). The final error at the conclusion of the simulation is 4.4 mm, which is greater than the observed error at the conclusion of the initial line search.

Examination of Figure 6.20a reveals that there are significant oscillations in the residual error after the initial decay from 23.3 mm to 3.8 mm. If the simulation is extended to 60000 Y-cuts (equivalent to four working weeks of operation) then these oscillations remain and the optimisation algorithm spends the majority of the time tuning the parameter estimates. It is possible that similar oscillations are also present when the optimisation algorithm is continuously applied to the six-parameter case, although the magnitude of the oscillations is significantly smaller (refer Figure 6.13). It is thought that the length of the search line and the sample density is causing these oscillations during subsequent applications of the line search part of the optimisation algorithm. While having R = 1.0 and m = 21 may produce the fastest convergence to an initial solution satisfying the success rate criteria, these values may not be the best choice for the on-going optimisation of the path parameters.

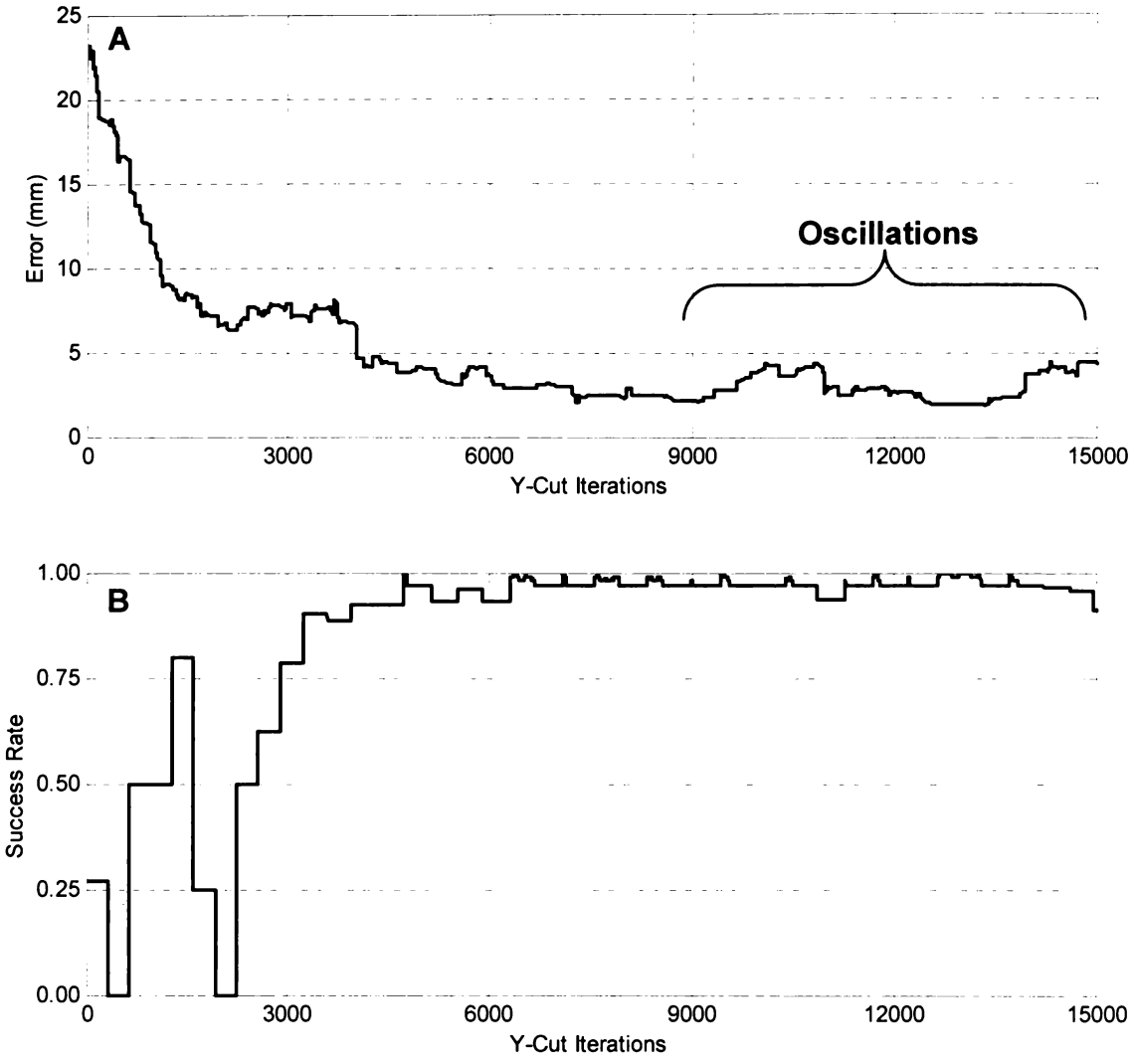

FIGURE 6.20 Plots of a) the residual error and b) the success rate for the continuous optimisation algorithm and the full set of fifteen parameters.

Different values for the line search ratio R are tested for use following the initial convergence of the line search algorithm (which uses a ratio R_0). The initial application of the algorithm still uses $R_0 = 1.0$, $m = 21$, $S_{max} = 0.98$ and $t = 100$. The simulation runs for 60000 simulated Y-cuts, allowing time for any oscillations to become apparent. The percentage of simulated Y-cuts spent on normal cutting (rather than tuning) is recorded, along with the final residual error. This procedure is repeated 10 times for each value of R and the average results are shown in Figure 6.21.

Previous continuous applications of the optimisation algorithm have used a value of $R = 1.0$ throughout the simulation, which corresponds to the right-hand edge of Figure 6.21. This value of R is clearly sub-optimal, having a low cutting percentage of 20.6% and a

relatively high final error of 3.6 mm. The best performance is obtained for $R = 0.4$, with a cutting percentage of 61.8% and a final error of 1.5 mm. Note also that this graph indicates that there is a clear correlation between the residual error and the cutting percentage, with a low error producing a high cutting percentage.

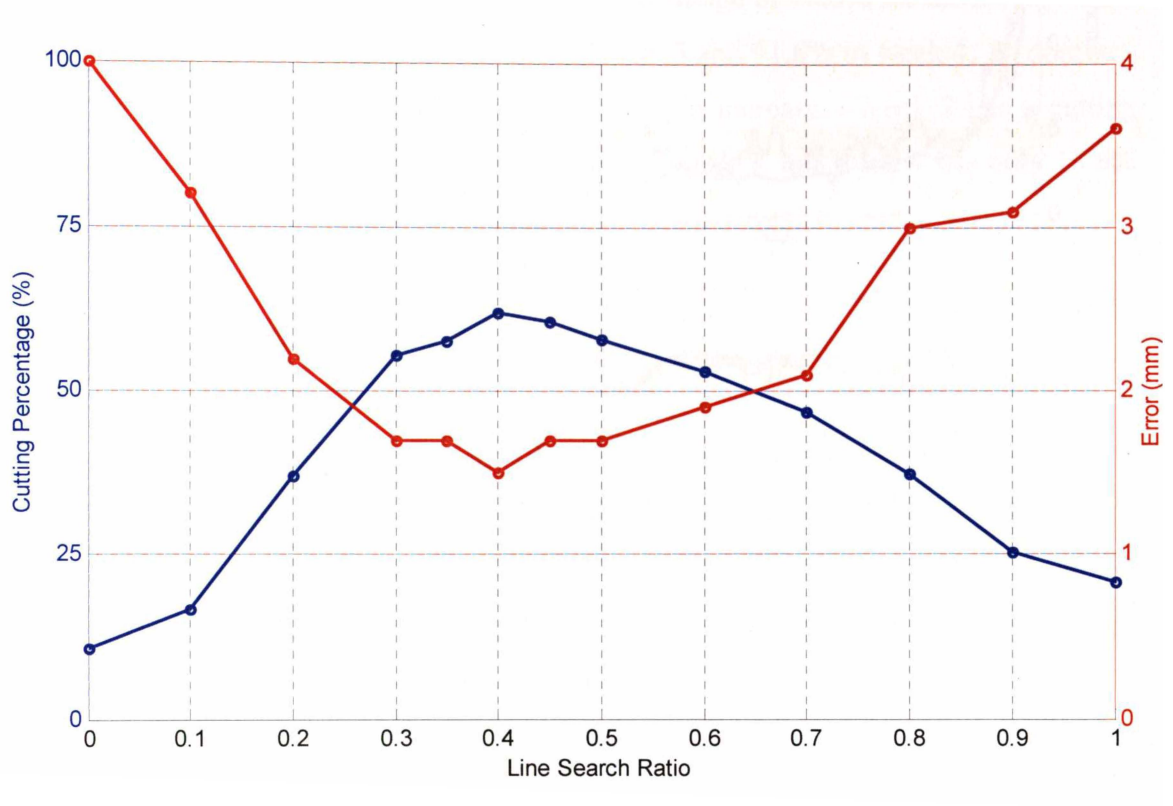

FIGURE 6.21 The cutting percentage and final error for various line search ratios used following the initial application of the line search algorithm.

Figure 6.22 shows the residual error, success rate and cutting percentage for the best performing ratio $R = 0.4$ and the original ratio $R = 1.0$. For the $R = 0.4$ case, the initial application of the line search algorithm with $R_0 = 1.0$ takes 5492 iterations to achieve a 98% success rate. The final error is 1.4 mm and 65.7% of the 60000 simulated cuts comprise the normal Y-cutting operation of the system. For the $R = 1.0$ case, the initial convergence takes 4799 iterations. The final error is 3.4 mm and only 25.9% of the 60000 simulated cuts are part of the normal Y-cutting operation. Figure 6.22a illustrates the difference in the convergence between the two ratios. In the $R = 0.4$ case, the error continues to decrease following the initial application of the line search algorithm. However, the error for the $R = 1.0$ case oscillates throughout the simulation.

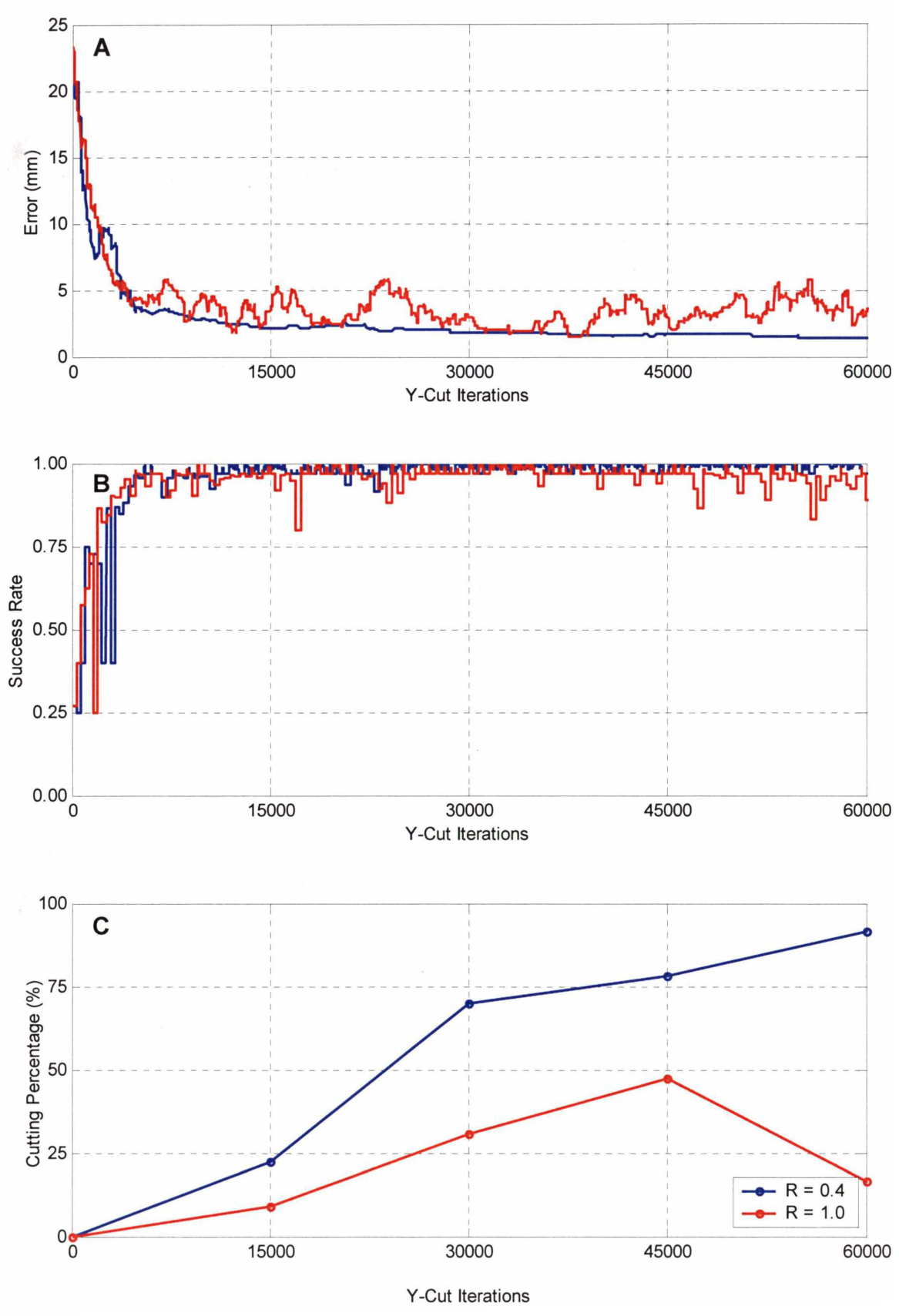

FIGURE 6.22 **Plots of a) the residual error, b) the success rate and c) the cutting percentage for the continuous optimisation algorithm using different line search ratios following the initial application of the line search algorithm.**

Figure 6.22c shows the percentage of the simulated cuts spent on normal Y-cutting as calculated for each "week" of operation. In both test cases, the first week (or 15000 iterations) are predominantly spent tuning the parameter estimate, with 22.2% of the time consisting of normal Y-cutting for the $R = 0.4$ case and only 8.8% for the $R = 1.0$ case. However, the second week has a cutting percentage of 70.0% for the $R = 0.4$ case. This cutting percentage increases to 78.3% in week 3 and 91.6% in week 4. In contrast, the $R = 1.0$ case does not experience the same rapid increase. Week 2 has a cutting percentage of 30.7%, which increases to 47.2% in week 3, but week 4 has only 16.6% of the time consisting of normal Y-cutting.

6.5.4 Step and Ramp Response

The inclusion of a smaller line search ratio for use following the initial tuning period has improved the stability of the algorithm during the continuous optimisation of the parameter estimate. However, this smaller ratio affects the response of the algorithm to any changes in the optimal value of the parameters. Figure 6.23 shows the effect of this ratio change on the step response and the ramp response of the algorithm. The step and ramp are the same as used to test the six-parameter model in Section 6.4.5 and Section 6.4.6, although the simulation runs over 30000 iterations to give a better indication of the response. In both cases, the smaller value for R means that the algorithm takes longer to respond to changes in the optimum parameter value. In the case of the ramp response, the algorithm cannot maintain the rate of change and quickly lags behind the optimum parameter. This lag produces a significant decrease in the success rate.

A solution to the poor response of the algorithm is to use an adaptive ratio R that depends on the observed success rate. If the success rate of the system is unacceptably low then the larger ratio $R = 1.0$ is used, otherwise the smaller ratio $R = 0.4$ is used to fine-tune the parameter estimate. Figure 6.24 shows the effect that a variable ratio has on the step and ramp response of the optimisation algorithm. The algorithm switches from $R = 0.4$ to $R = 1.0$ if the success rate S is less than 0.90 (90%). While this is a relatively high acceptability threshold, this value appears to give good tracking performance. The algorithm is able to respond more effectively to changes in the parameter estimate.

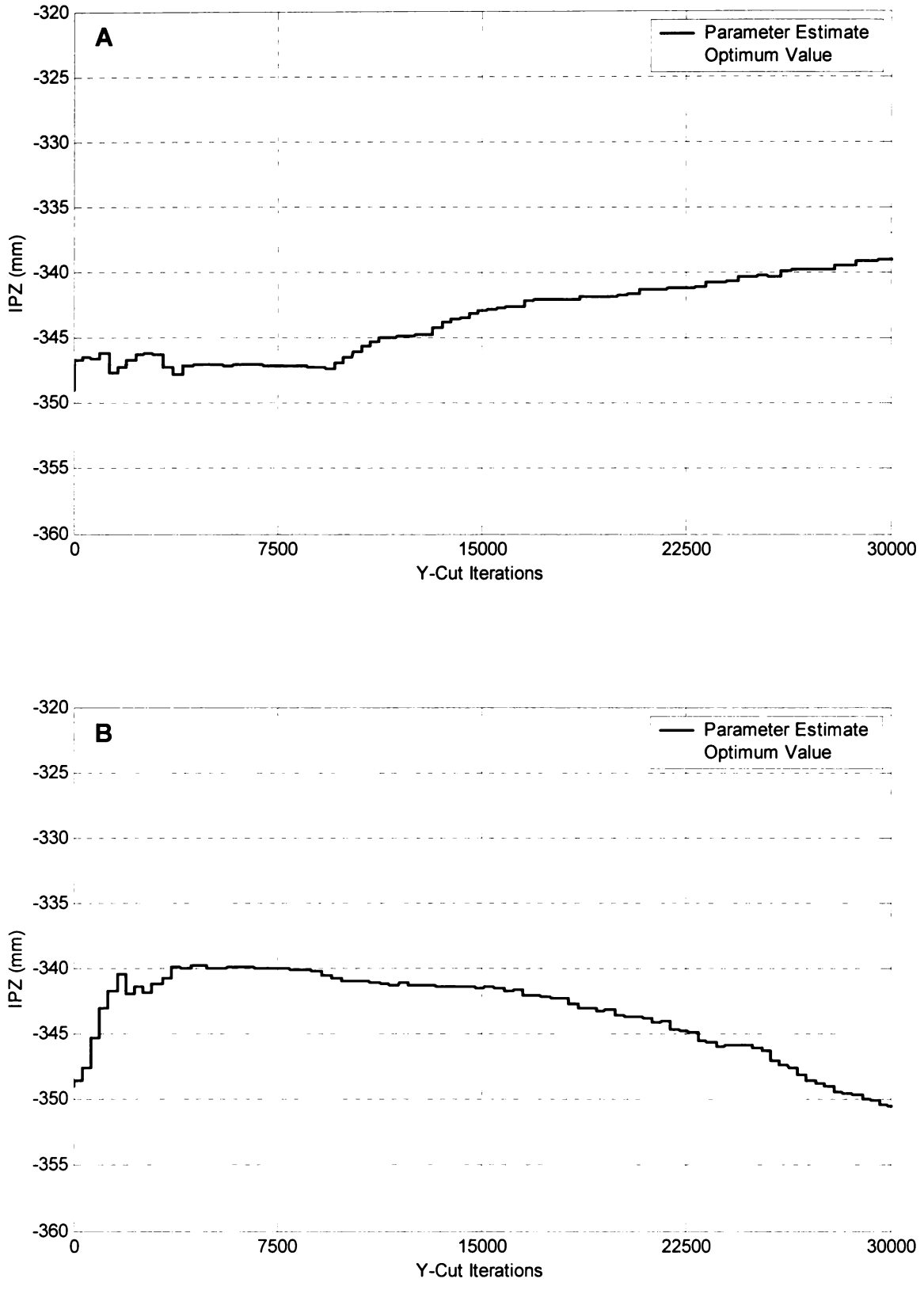

FIGURE 6.23 **The response of the optimisation algorithm to a) a step change and b) a ramp change in the path parameter *IPZ*.**

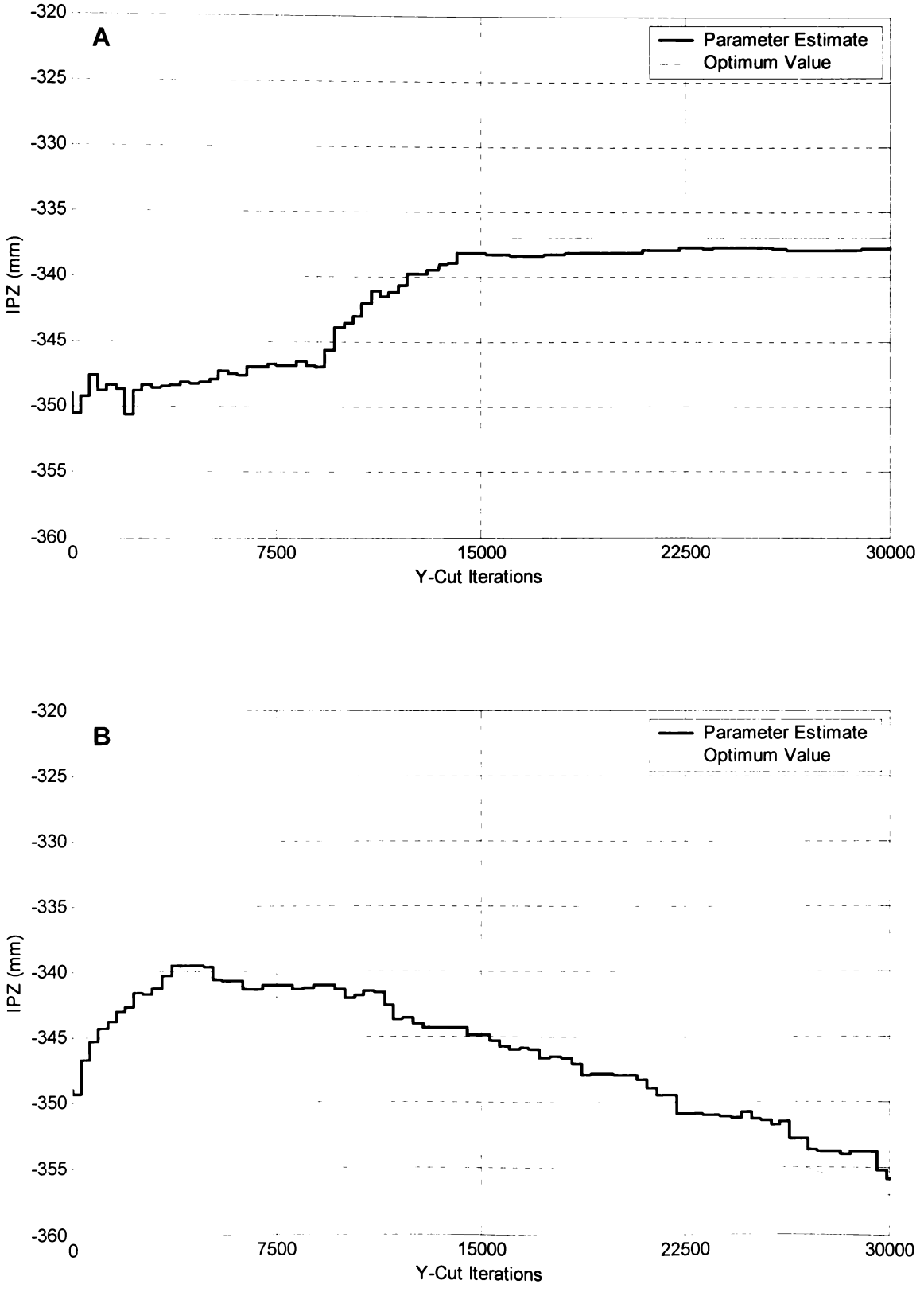

FIGURE 6.24 **The response of the optimisation algorithm to a) a step change and b) a ramp change in the path parameter *IPZ* using an adaptive line search ratio.**

6.6 DISCUSSION

Several changes have been made to the original algorithm proposed in Section 6.3.1. The convergence time of the algorithm has been improved by modifying the sampling procedure used to determine the success rate at the current parameter estimate. This modified algorithm typically converges with 25% fewer simulated Y-cuts when tested on the simplified six-parameter model.

The optimisation algorithm has been shown to converge quickly to an acceptable set of path parameters. In the case of the simplified six-parameter model, the algorithm converged in 960 simulated Y-cuts when R = 1.6 and m = 11. Testing with the expanded set of parameters has shown that convergence times rapidly increase for lower sample densities. The fastest convergence for the expanded fifteen-parameter model occurred when R = 1.0 and m = 21, and required 4760 Y-cuts. This is a five-fold increase from the best performing six-parameter case, although it generally only takes three times longer to converge than the six-parameter model for the same values of R and m. The 4760 simulated Y-cuts required by the optimisation algorithm equates to approximately a day and a half of processing in a real meat-plant. This is substantially faster than the existing manual tuning process and illustrates that the developed algorithm is a viable alternative.

Different criteria for the termination of the line search algorithm have been tested. The stronger success rate criteria take longer to converge but result in a lower residual error at the conclusion of the search algorithm. However, there is the risk of the convergence time increasing if the sample density is too low. Although the majority of the testing has used the criteria S_{max} = 0.98 and t = 100 there could be an advantage in using S_{max} = 1.00 and t = 100 in the real-world implementation of the algorithm. The stronger criteria take approximately 500 extra iterations to converge to an acceptable solution but result in a smaller residual error.

The optimisation algorithm is able to respond effectively to changes in the optimal parameters. Initial testing using the simplified six-parameter model has demonstrated that the algorithm is suitable for the on-going optimisation of the parameter estimate

and can track both step and ramp changes. However, the same algorithm is not suitable for the continuous optimisation of the expanded set of parameters, with the residual error oscillating and the line search continually attempting to improve the parameter estimate. This means that the majority of the simulated cuts are used to tune the parameters rather than being part of the normal Y-cutting operation of the system. The algorithm is modified so that a smaller ratio $R = 0.4$ is used for applications of the line search following the initial convergence with $R_0 = 1.0$. This modification increased the normal Y-cutting percentage and allowed the algorithm to fine-tune the parameter estimate. However, it also means that the system cannot respond quickly to changes in the optimal parameters. A compromise solution is developed that changes the ratio R depending on the observed success rate. If the success rate is less than 90% then the algorithm uses $R = 1.0$ to effect a larger change in the parameter estimate. If the success rate is greater than 90% then the estimate is deemed to only require fine-tuning with $R = 0.4$. This solution improves the response of the algorithm to parameter changes.

There is a possibility that some of the fifteen identified path parameters would not need to be optimised in the real Y-cutting system. An example is the two Euler angles *BA1A* and *BA2A*. During testing at Goulburn, it was found that the speed of the robot could be increased if the orientation of the tool did not change substantially along the cut-path. One suggestion was for the tool to maintain the same yaw-angle A throughout the cut-path, especially as having the tool perpendicular to the foreleg ($BA1A = BA2A = 0°$) appeared to result in the best performance. If the decision were made to not change these angles then this would reduce the dimensionality of the parameter space and would reduce the convergence time required by the optimisation algorithm. If the success rate distribution for these angles is estimated via a series of cutting trials then the standard deviation could determine whether the parameter can be made a constant. A large standard deviation indicates that the parameter can tolerate significant variation without adversely affecting the overall success rate. In this case, it would be reasonable to hold the parameter at a constant value since a non-optimal estimate will not significantly reduce the performance of the system. If the standard deviation is small then the success rate is more sensitive to a non-optimal parameter and the estimate should continue to be optimised by the algorithm.

Testing with the six-parameter model indicates that a constant line search takes longer to converge than a line search based on the estimate of the success-rate objective function. However, when this estimate is not available, a constant search method can still be used as part of an effective optimisation technique. An alternative would be to conduct a further series of cutting trials to more accurately establish the nature of the success-rate objective function as it pertains to these parameters, although this would be a time-consuming process. This will be the case during subsequent implementation of the algorithm within the KUKA Y-cutting system since most of the path parameters have only been estimated for the purposes of these simulations. It is also important that the algorithm is flexible enough to serve as a generic solution for other automation tasks.

7. TOWARDS GENERIC FAULT DETECTION AND PATH OPTIMISATION

Most of the robotic systems developed by IRL are used to process highly varying natural products. While the products and processes may differ, the robots are all performing similar tasks. These tasks largely consist of planning an appropriate path for the robot based on sensor information and then following the designated path. Unless the task is non-contact, there will be some sort of interaction between the robotic tool and the product, producing similar load variations to those observed in the automated Y-cutting process. Therefore, the developed fault detection and path optimisation algorithms have the potential for generic application within a number of different automated systems. However, the algorithms must be able to handle different process variations and a greater number of faults.

7.1 NETWORK TRAINING IN THE PRESENCE OF NOISE

If the fault detection module is to be used in conjunction with different automated tasks then it must be tolerant to process variations of different magnitudes. In the case of the automated Y-cutting system, carcasses of different sizes and shapes produce differing axial loads, particularly as the tool traverses the knee. These variations can be approximated by superimposing noise of varying magnitude on the underlying signal. This means that it is important to establish the sensitivity of the LVQ network to noise.

To test the effect of noise on the performance of the LVQ network, zero-mean Gaussian noise (variance $\sigma^2 = 1$) is added to the leading leg datasets from Goulburn (refer Section 5.3). The added noise is modified by a current scale factor that varies from 0.0 A to 2.0 A in steps of 0.1 A. The data are down-sampled by a factor of 90, which gives the same sampling rate as the previously implemented fault detection module (refer

Section 5.5.4). A 50/50 split is used to partition the data into training and testing sets. Each network has two neurons trained for 50 epochs, and 100 runs are completed.

Figure 7.1 shows the effect of the noise on the classification performance of the LVQ network. The classification success decreases for noise scale factors greater than 0.1 A, although 100% of insertion failures are still detected for a factor of 0.2 A. The classification of insertion failures rapidly decreases for larger noise factors. However, the classification of completed cuts remains relatively high throughout the test. This disparity suggests that as the amount of noise increases the network is classifying most of the cuts as completed, with the noise masking any discriminating features. This is an expected result when the difference between the insertion failure and completed cut weights for the noiseless case is considered (refer Figure 7.2). Most of the weights differ by less than 0.2 A in magnitude, so it would not require a significant amount of noise to mask the discrimination between the two cut classes. However, it should be noted that the difference between the weights was much greater following the online training of the LVQ network in Goulburn (refer Figure 5.10). It is expected that this re-trained network would be more tolerant to noise because of this greater differential.

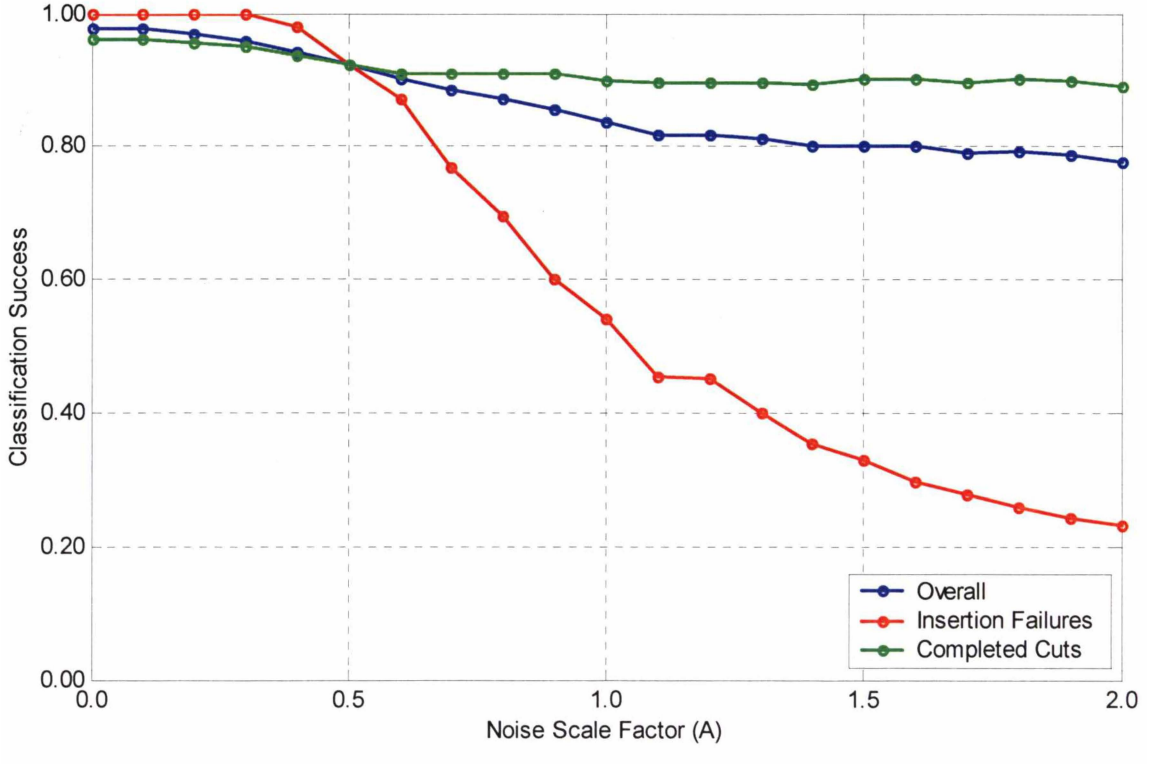

FIGURE 7.1 The effect of noise on the classification success of the LVQ fault detection module.

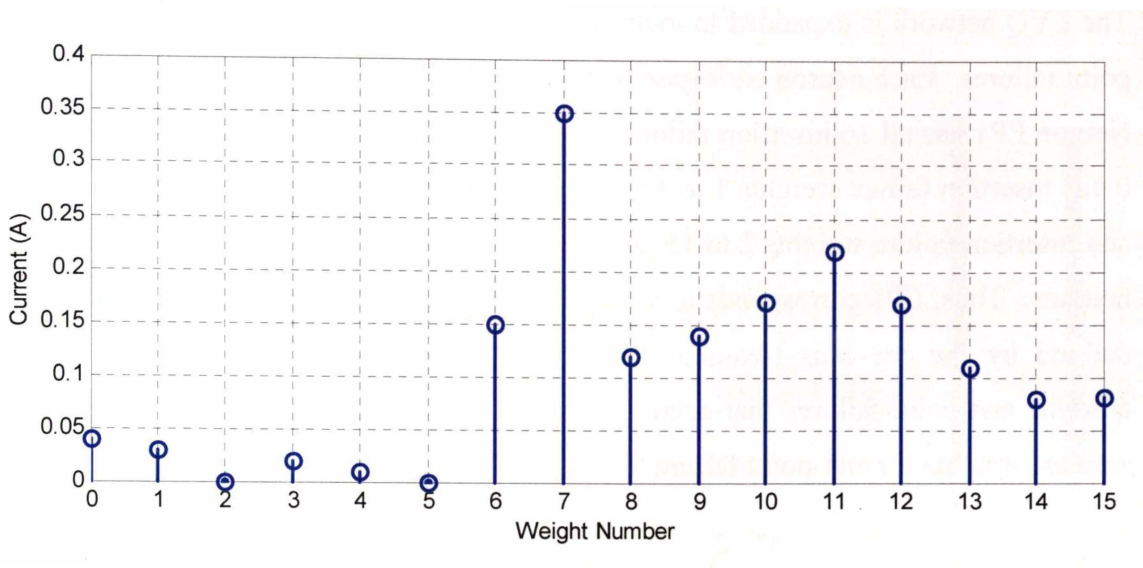

FIGURE 7.2 **The absolute difference between the insertion failure and completed cut LVQ weights.**

7.2 DETECTION OF ADDITIONAL FAULTS

The LVQ module developed in Chapter 5 has been successfully applied to the detection of insertion failures in the KUKA automated Y-cutting system. The module only needed to detect insertion failures because this was the only process fault observed at Goulburn. However, the earlier testing conducted with the IRL8L robot illustrated that end-point failures could occur if the process or tool was not optimised sufficiently. Therefore, the extension of the existing module to detect end-point failures will be beneficial to future meat-automation work.

7.2.1 Expansion of Existing LVQ Network

During the IRL8L trial, it was observed that the load for an end-point failure began as a completed cut but then rapidly decayed to the level of an insertion failure after the tool withdrew from the pelt. This means that an end-point failure should be able to be described by a combination of a completed cut signal and an insertion failure signal. This also means that an end-point failure should be detectable using a combination of the completed cut and insertion failure weights from the existing LVQ network.

The LVQ network is expanded to include 17 neurons to allow for the detection of end-point failures. Each neuron corresponds to a particular end-point, labelled EP1 to EP17. Neuron EP1 has all 16 insertion failure weights. Neuron EP2 has completed cut weight 0 and insertion failure weights 1 to 15. Neuron EP3 has completed cut weights 0 and 1 and insertion failure weights 2 to 15, and so on. Neuron EP17 has all 16 completed cut neurons. Thus, EP1 corresponds to an insertion failure and EP17 to a completed cut as defined by the previous two-neuron LVQ network. The 15 intervening neurons describe end-point failures that occur at various points in time. Figure 7.3 shows the selected weights for end-point failure EP9.

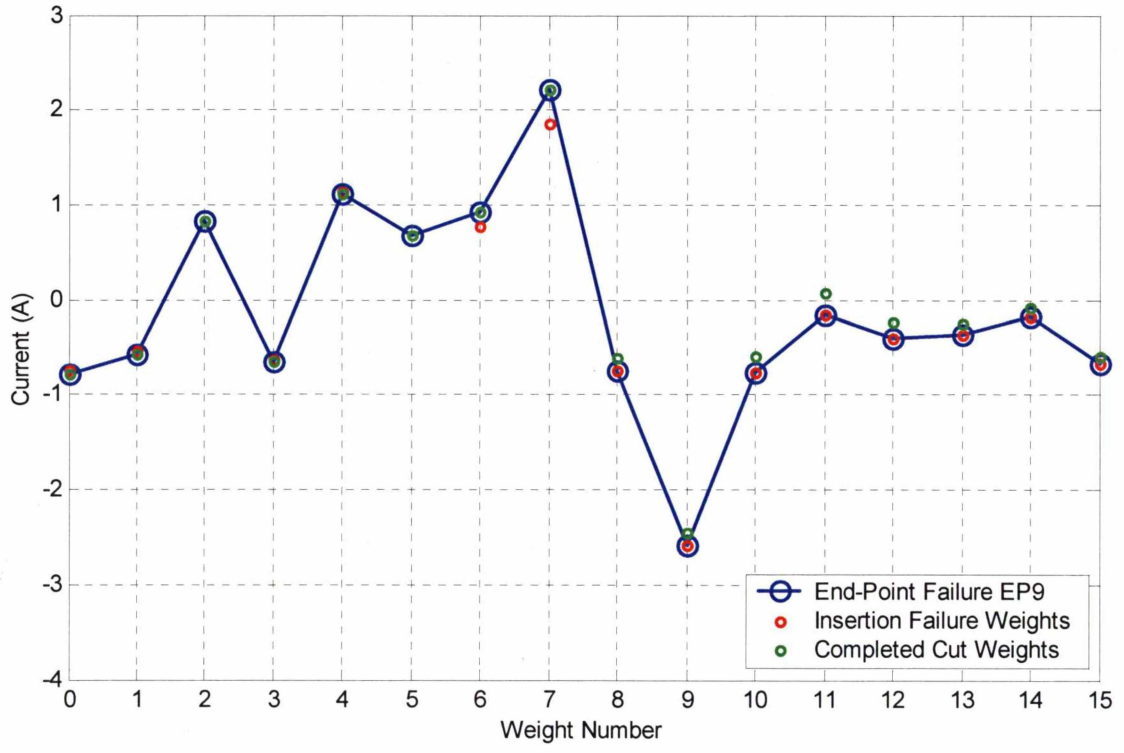

FIGURE 7.3 The construction of neuron EP9 from insertion failure and completed cut weights.

7.2.2 Classification of Goulburn Data

The expanded LVQ network is initially tested with the leading leg data from Goulburn. Figure 7.4 shows the cumulative classification output of the LVQ network for the presentation of the 171 cuts (142 completed cuts and 29 insertion failures).

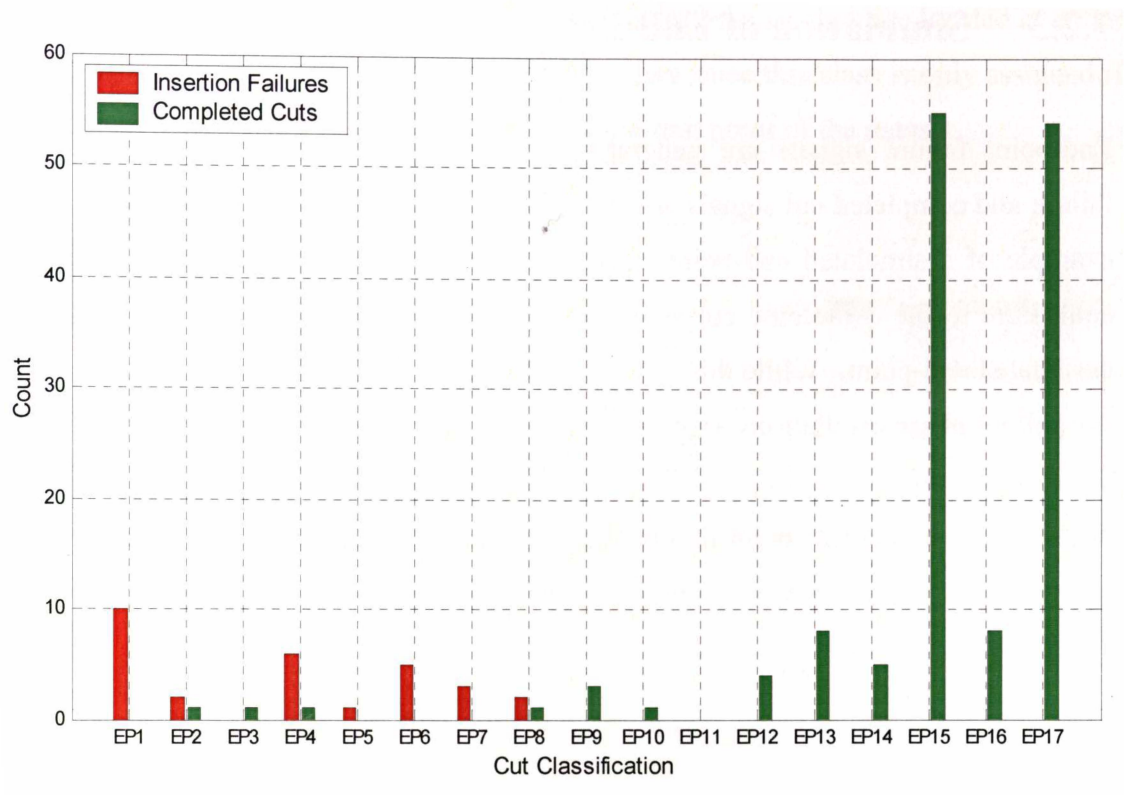

FIGURE 7.4 Classification of leading leg data using the expanded LVQ network.

The insertion failures are all classified as end-points EP1 to EP8. End-point EP1 has the highest number of insertion failure classifications, accounting for 34.5% of the insertion failures. Note that the first six insertion failure and completed cut weights are very similar to each other (refer Figure 7.2), which makes it difficult for the network to distinguish between the first eight end-point classifications.

Most of the completed cuts appear on the right-hand side of the graph, with the majority of these cuts classed as end-points EP15 or EP17. This is an expected result, with the neuron weights for EP17 being equivalent to the weights for a completed cut. Several outlying completed cuts have been classified as end-points EP2, EP3, EP4, and EP8. These outlying cuts are the cause of the network previously classifying only 98% of the datasets correctly, with these four outlying cuts having been classed as insertion failures.

7.2.3 Simulation of End-Point Failures

End-point failure signals are generated by combining randomly selected insertion failure and completed cut signals at randomly selected end-points. Figure 7.5 shows an example of a simulated end-point signal. The generated end-point failure is initially equivalent to the completed cut signal, but tracks the insertion failure signal after the designated end-point. While this approximation of an end-point signal does not include the pullout phase oscillations observed in Section 4.2.3, it should be sufficiently similar for initial simulation purposes. The expanded LVQ network is tested with 100,000 randomly generated end-point failure signals. Each simulated cut is associated with the closest target end-point class, based on the proximity of the end-point to the neuron weights along a common time-base.

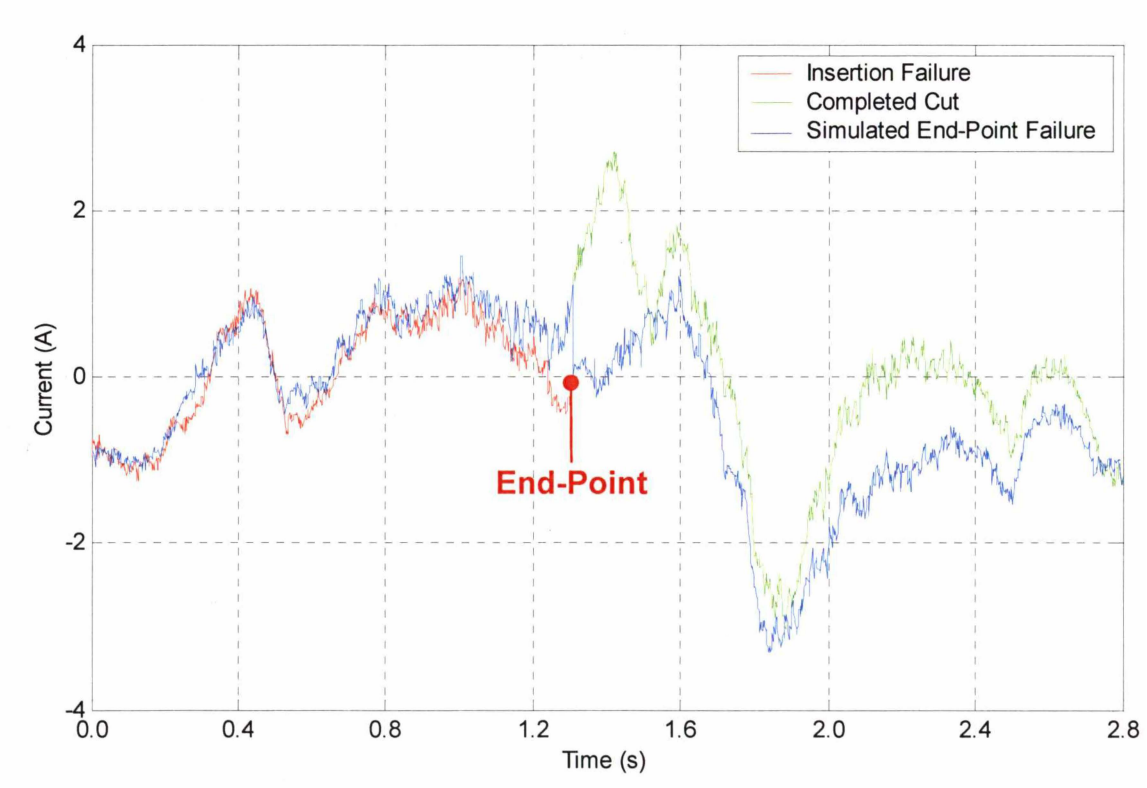

FIGURE 7.5 An example of a simulated end-point failure.

Figure 7.6 gives the cumulative number of Y-cuts belonging to each end-point class, as well as the number of cuts determined by the LVQ network for each class. The number of cuts assigned to each target classification is constant over most end-point classes, which is expected since the end-point is selected randomly. Classes EP2 and EP17 are

associated with fewer cuts than the intervening classes because they are located at either end of the sampling line. EP1 has very few members since this class is only assigned if the randomly selected end-point coincides with the first point of the dataset.

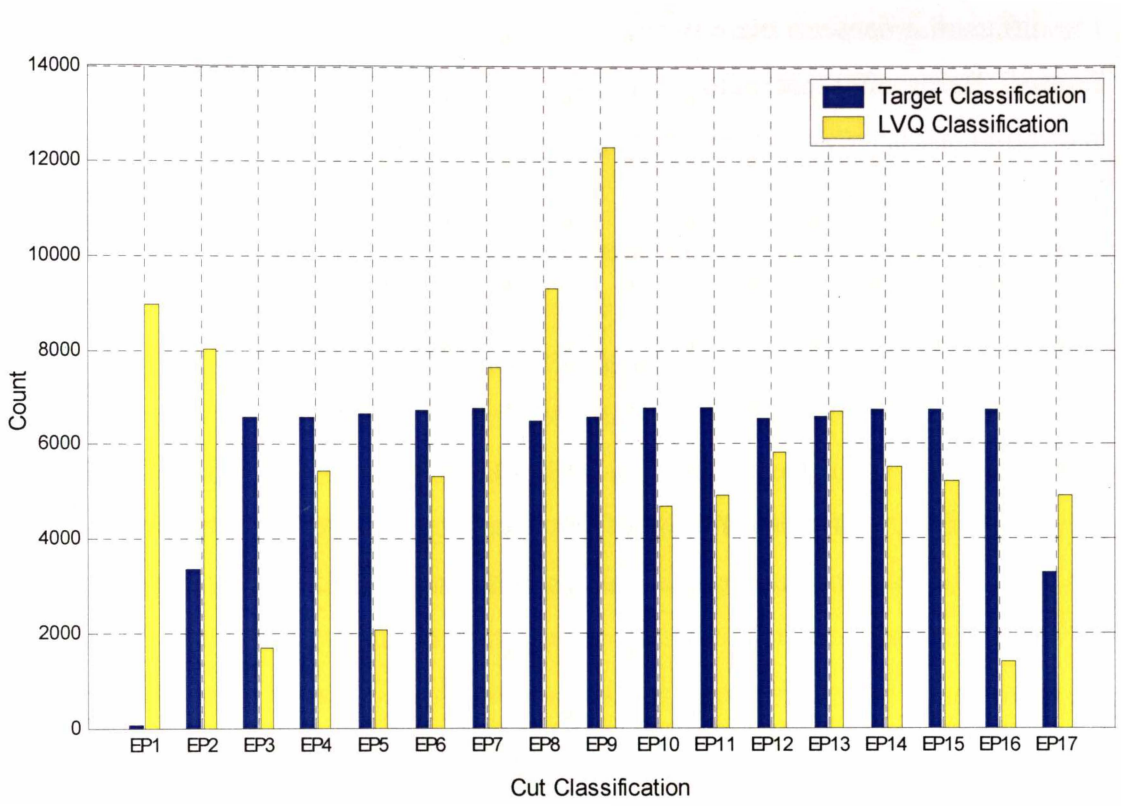

FIGURE 7.6 Comparison of the target end-point classification and the LVQ classification output.

EP9 is the class most frequently assigned by the LVQ network. This class is the first to include the completed cut neuron weight 7 that corresponds to the peak loading as the tool traverses the knee (refer Figure 7.3). This loading is one of the key discriminating features between the two cut classes, with the difference between the completed cut and insertion failure weights being the greatest at this point (refer Figure 7.2). The disproportionate number of cuts assigned to this class is due to the presence of this feature and because subsequent differentials are much smaller in magnitude. A comparison of the LVQ classifications in Figure 7.6 with the weight differentials in Figure 7.2 shows that the shapes of the two plots are very similar, particularly from EP9/Weight-7 onwards. This correspondence indicates that the end-point output of the LVQ network is closely related to the magnitude of the weight differential.

A large number of cuts are classified as EP1, despite the fact that there are only a few cuts that actually correspond to this classification. There are also a disproportionate number of cuts classified as EP2. These two end-point classes differ by the first neuron weight only: EP1 uses insertion failure weight 0 and EP2 uses completed cut weight 0. The differential between these two is larger than for the subsequent five weights (refer Figure 7.2), meaning that there is a propensity for the LVQ network to use this weight to discriminate between classes.

Figure 7.7 shows the classification success for each end-point. Also shown is the ratio of cuts that are within one end-point classification of the correct class (including the correctly classified cuts). End-point EP9 has the highest percentage of correctly classified cuts, with 47.0%. End-point EP10 has the highest percentage of cuts within one class of the correct classification, with 93.6%. The lowest classification success occurs for EP3, with only 7.9% of cuts correctly classified. EP4 has the lowest percentage of almost correct classifications, with only 25.2% of cuts being closer than one end-point class.

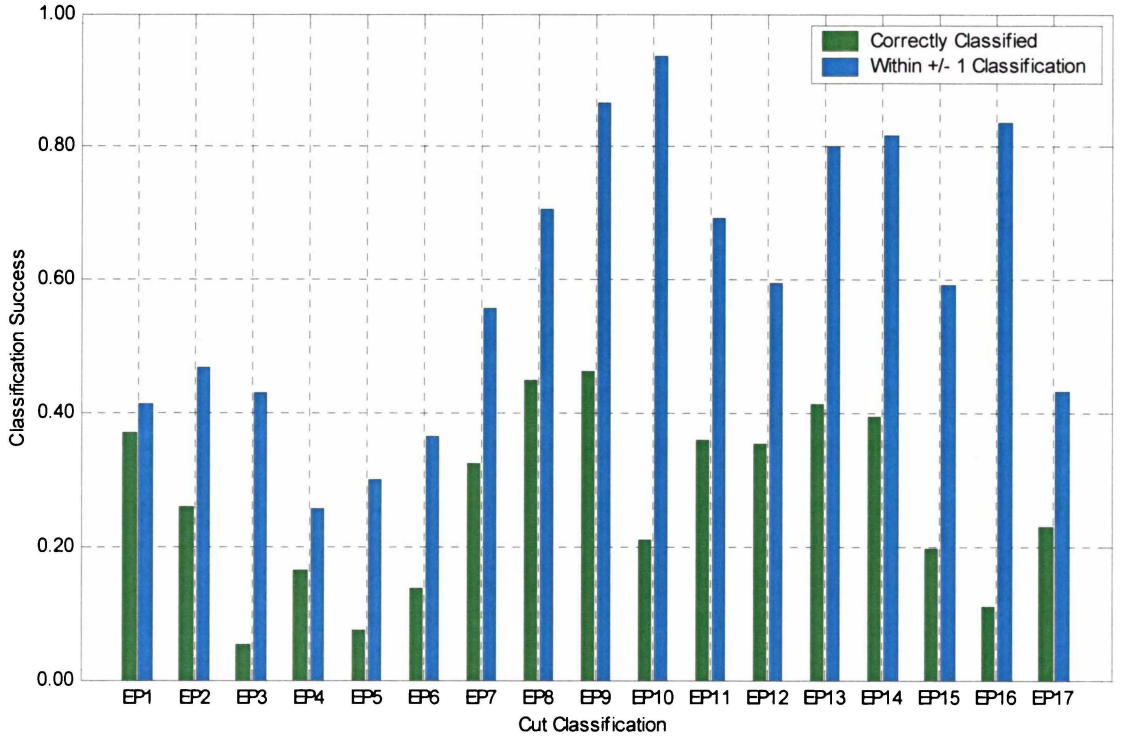

FIGURE 7.7 The end-point classification success of the LVQ network.

The earlier end-points (EP1 to EP6) are not detected as reliably as the later end-points (EP7 to EP16). This shows that the accuracy of the end-point detection is dependent on the size of the discriminating features. This also highlights a general observation about all pattern recognition methods, in that the reliability and accuracy of the classification can only ever be as good as the difference between the discriminating features. It is unreasonable to expect that an algorithm can reliably differentiate between neighbouring classes if the feature spaces overlap significantly.

Figure 7.8 shows a histogram of the classification errors associated with the simulated data. The classification error is defined as the difference between the LVQ classification output (EP1 to EP17) and the target output (EP1 to EP17). Most of the simulated cuts have an error of -1 or 0, with these errors accounting for 54.9% of the cuts. The classification of 75.7% of the cuts is within two end-point classes of the target end-point class. The average classification error over the 100,000 simulated cuts is 1.72 end-point classes. With the time difference between neighbouring samples being 180 ms (refer Section 5.5.4), this average classification error is equivalent to an end-point error of 0.31 s. This indicates that the expanded network is reasonably successful at determining the cut end-point, although this end-point error is greater than the errors observed during testing with the IRL8L data (refer Section 4.6).

The end-point determination error can be isolated for each end-point class (refer Figure 7.9). The smallest error occurs for cuts with end-point EP10, with an error of 0.19 s. End-points EP9 and EP11 also have an error of less than 0.20 s. There is a general trend for the "completed cuts" (EP9 to EP17) to have smaller end-point errors than the "insertion failures" (EP1 to EP8). This observation highlights again the importance of the relative difference between discriminating features.

7.2.4 KRL Implementation

Figure 7.10 gives an example implementation of a modified fault detection interrupt routine for the classification of end-point failures. The modified routine uses the same insertion failure and completed cut weights as the scheme developed in Chapter 5, with these weights acting as a basis for the definition of the end-points EP1 to EP17.

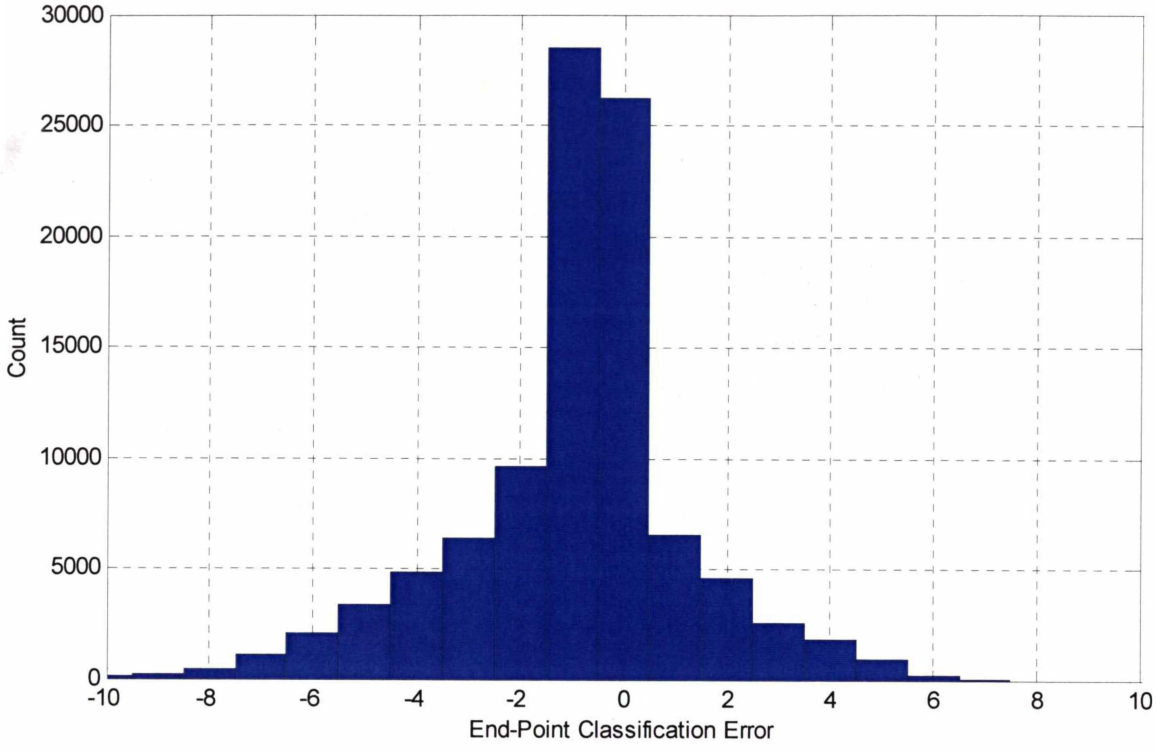

FIGURE 7.8 **Histogram of the classification errors from the expanded LVQ network tested on the simulated end-point failures.**

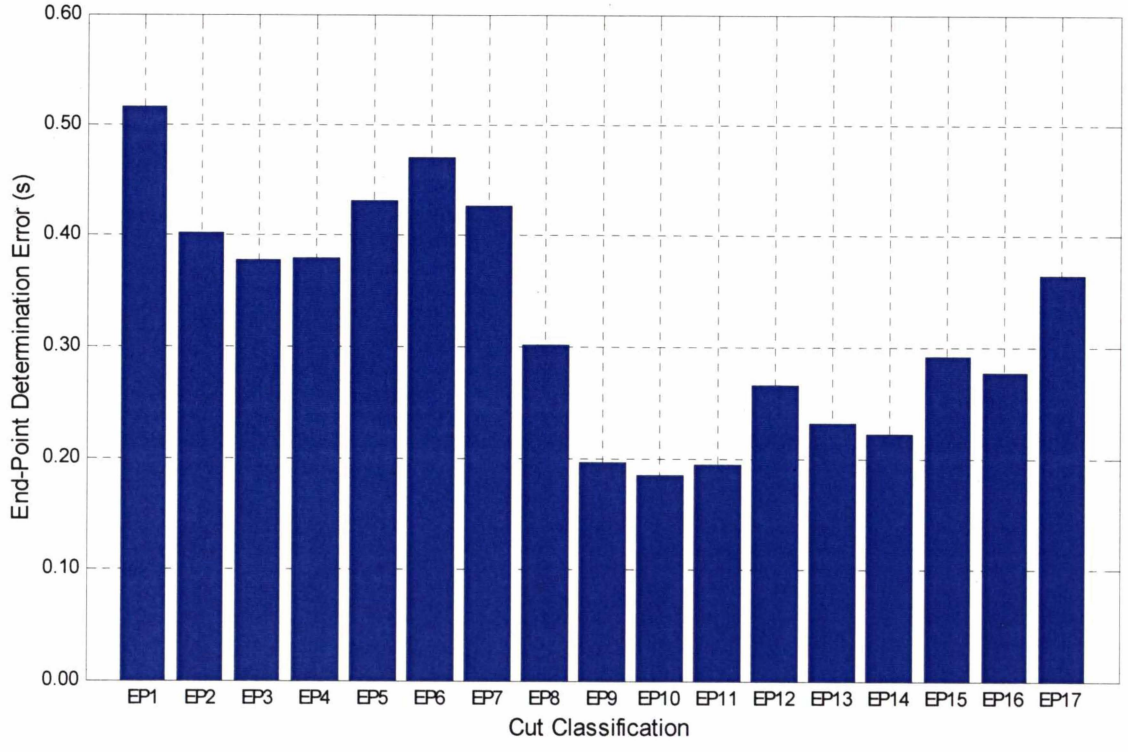

FIGURE 7.9 **The end-point determination error for each end-point class.**

```
GLOBAL DEF Fault_Detection()

...
IF (A5_Curr_I < 16) THEN                    ;Is array filled yet?
  ...
ELSE
  ...
  IF (LVQ_Learning == TRUE) THEN            ;Online learning is enabled
    Sum[1] = 0.0                            ;Reset temporary variable
    Sum[17] = 0.0                           ;Reset temporary variable
    FOR N = 1 TO 16
      Sum[1] = Sum[1]+((IW_IF[LT,N]-A5_Curr[N])*(IW_IF[LT,N]-A5_Curr[N]))
      Sum[17] = Sum[17]+((IW_CC[LT,N]-A5_Curr[N])*(IW_CC[LT,N]-A5_Curr[N]))
    ENDFOR
    Neuron[1] = -SQRT(Sum[1])               ;Calculate IF neuron activation
    Neuron[17] = -SQRT(Sum[17])             ;Calculate CC neuron activation
    IF (Neuron[17] > Neuron[1]) THEN        ;Completed cut determined
      IF (Target_Output == TRUE) THEN       ;User indicates insertion failure
        FOR N = 1 TO 16                     ;Shift CC weights away from cut
          IW_CC[LT,N] = IW_CC[LT,N]-Learning_Rate*(A5_Curr[N]-IW_CC[LT,N]))
        ENDFOR
      ELSE                                  ;User indicates completed cut
        FOR N = 1 TO 16                     ;Shift CC weights towards cut
          IW_CC[LT,N] = IW_CC[LT,N]+Learning_Rate*(A5_Curr[N]-IW_CC[LT,N]))
        ENDFOR
      ENDIF
    ELSE                                    ;Insertion failure determined
      IF (Target_Output == TRUE) THEN       ;User indicates insertion failure
        FOR N = 1 TO 16                     ;Shift IF weights towards cut
          IW_IF[LT,N] = IW_IF[LT,N]+Learning_Rate*(A5_Curr[N]-IW_IF[LT,N]))
        ENDFOR
      ELSE                                  ;User indicates completed cut
        FOR N = 1 TO 16                     ;Shift IF weights away from cut
          IW_IF[LT,N] = IW_IF[LT,N]-Learning_Rate*(A5_Curr[N]-IW_IF[LT,N]))
        ENDFOR
      ENDIF
    ENDIF
  ELSE                                      ;Online learning is disabled
    FOR K = 1 TO 17
      Sum[K] = 0.0                          ;Reset temporary variable
      N = 1                                 ;Reset weight index
      WHILE (N < K)                         ;Endpoint weight = CC weight
        Sum[K] = Sum[K]+((IW_CC[LT,N]-A5_Curr[N])*(IW_CC[LT,N]-A5_Curr[N]))
        N = N + 1                           ;Increment weight index
      ENDWHILE
      WHILE (N <= 16)                       ;Endpoint weight = IF weight
        Sum[K] = Sum[K]+((IW_IF[LT,N]-A5_Curr[N])*(IW_IF[LT,N]-A5_Curr[N]))
        N = N + 1                           ;Increment weight index
      ENDWHILE
      Neuron[K] = -SQRT(Sum[K])             ;Calculate neuron activation
    ENDFOR
    Neuron_Max = Neuron[1]                  ;Initialise maximum neuron value
    Endpoint_Failure = 1                    ;Initialise end-point indicator
    FOR K = 2 TO 17
      IF (Neuron[K] > Neuron_Max) THEN
        Neuron_Max = Neuron[K]              ;Set maximum to current value
        Endpoint_Failure = K                ;Set indicator to current index
      ENDIF
    ENDFOR
  ENDIF
ENDIF

END
```

FIGURE 7.10 **Modified fault detection interrupt routine for the detection of end-point failures (modifications are highlighted in bold).**

The online training scheme detailed in Section 5.7 is retained so that the base insertion failure and completed cut weights can be automatically adjusted. For the purposes of this implementation, it is assumed that the operator can still broadly class the cuts as either insertion failures or completed cuts, and that distinct sets of neuron weights for each end-point are unnecessary. However, the scheme could easily be adapted to allow the operator to input any one of seventeen cut-path end-points and only update the weights pertaining to a specific point.

If the online training is disabled then the activation of the seventeen end-point neurons is calculated. The weights for each neuron are a combination of the completed cut and insertion failure weights `IW_CC[]` and `IW_IF[]`. The neuron with the smallest Euclidean distance is classed as the winner and a corresponding integer output `Endpoint_Failure` is set.

7.3 DETECTING LOAD VARIATIONS

As well as detecting end-point failures, additional process information can be obtained from the expanded LVQ network by examining the activation of the individual LVQ neurons. In particular, it should be possible to isolate cuts that deviate from a normal completed cut signal and establish regions with higher or lower loads.

7.3.1 Using the Neuron Output

One method for defining an abnormal cut is to use the output of the LVQ neurons as a measure of the deviation from a normal operation. The output of each neuron corresponds to the Euclidean distance between the neuron weights and the input axial load signal (refer Section 2.3.6). Figure 7.11 shows the distribution of Euclidean distances output by the completed cut neuron in response to the leading leg completed cut data from Goulburn. Most of the datasets are located between 4.0 A and 5.5 A from the completed cut neuron, with 80.0% of the cuts being within this range. The loads experienced by these cuts could be defined as being normal with the remaining cuts having abnormal loads.

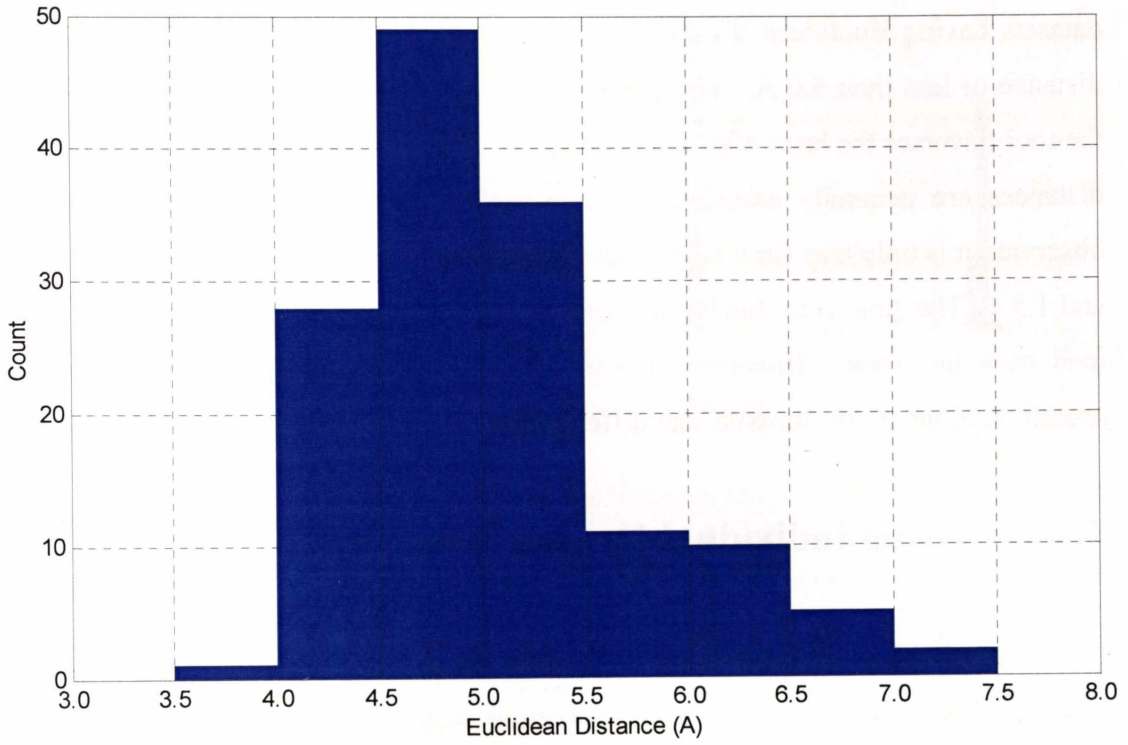

FIGURE 7.11 **Histogram of the Euclidean distances between the completed cut weights and the completed cut signals.**

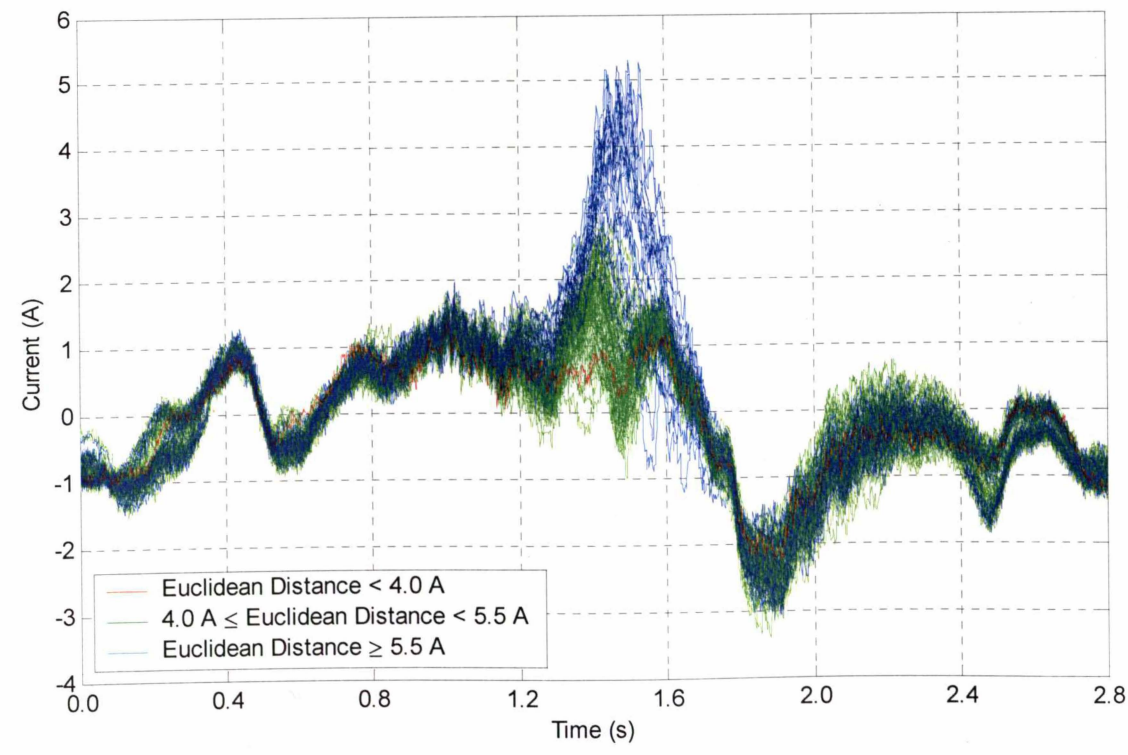

FIGURE 7.12 **Leading leg completed cuts with a Euclidean distance less than 4.0 A and greater than 5.5 A.**

The signals that are outside the range 4.0 – 5.0 A are highlighted in Figure 7.12, with 28 datasets having Euclidean distances greater than 5.5 A and a single dataset having a distance of less than 4.0 A. The greatest difference between these cuts can be seen as the tool traverses the knee after approximately 1.5 s. The cuts with the larger Euclidean distances are generally associated with a higher load over the knee, although this observation is only true for a small part of the overall signal between approximately 1.3 and 1.5 s. The single cut that has a distance of less than 4.0 A has a lower than average load over the knee. However, this technique does not allow the identification of specific regions of the cut-path that differ from a normal cut.

7.3.2 Using Individual Neuron Weights

A more direct and accurate method for defining abnormalities in the cut-path is to examine the difference between the individual neuron weights and the input signal. Figure 7.13 shows the differences between the sixteen completed cut weights and the completed cut signals. The distributions for some of the weights are narrow, particularly those that relate to the start of the cut (weights 0 to 6). This is expected since there is only limited variation in the load signal during this initial cut-phase. Weights 0 to 6 are all subject to completed cut signal variations of less than 1.0 A.

There are large variations encountered by some of the weights, particularly weights 7 to 13. These weights encounter completed cut signals that vary by over 1.0 A. Weight 8 experiences the largest variation in input signal, with a difference of 5.04 A between the biggest and the smallest. Weights 14 and 15 have signal variations of less than 1.0 A. These two weights correspond to the pullout phase of the cut-path where there is only limited variation in the load experienced by the robot. The large input signal variations can be used to detect abnormalities in specific regions of the cut-path. Two examples of this abnormality detection are shown in Figure 7.14. Figure 7.14a highlights cuts with a variation of greater than 3.5 A for both weights 8 and 9. The threshold variation of 3.5 A is obtained by observing the input variations in Figure 7.13 and selecting a current value that corresponds to an arbitrary upper limit for the individual distributions. The highlighted cuts all have high loads over the knee region, but comparatively normal loads over the remainder of the cut-path.

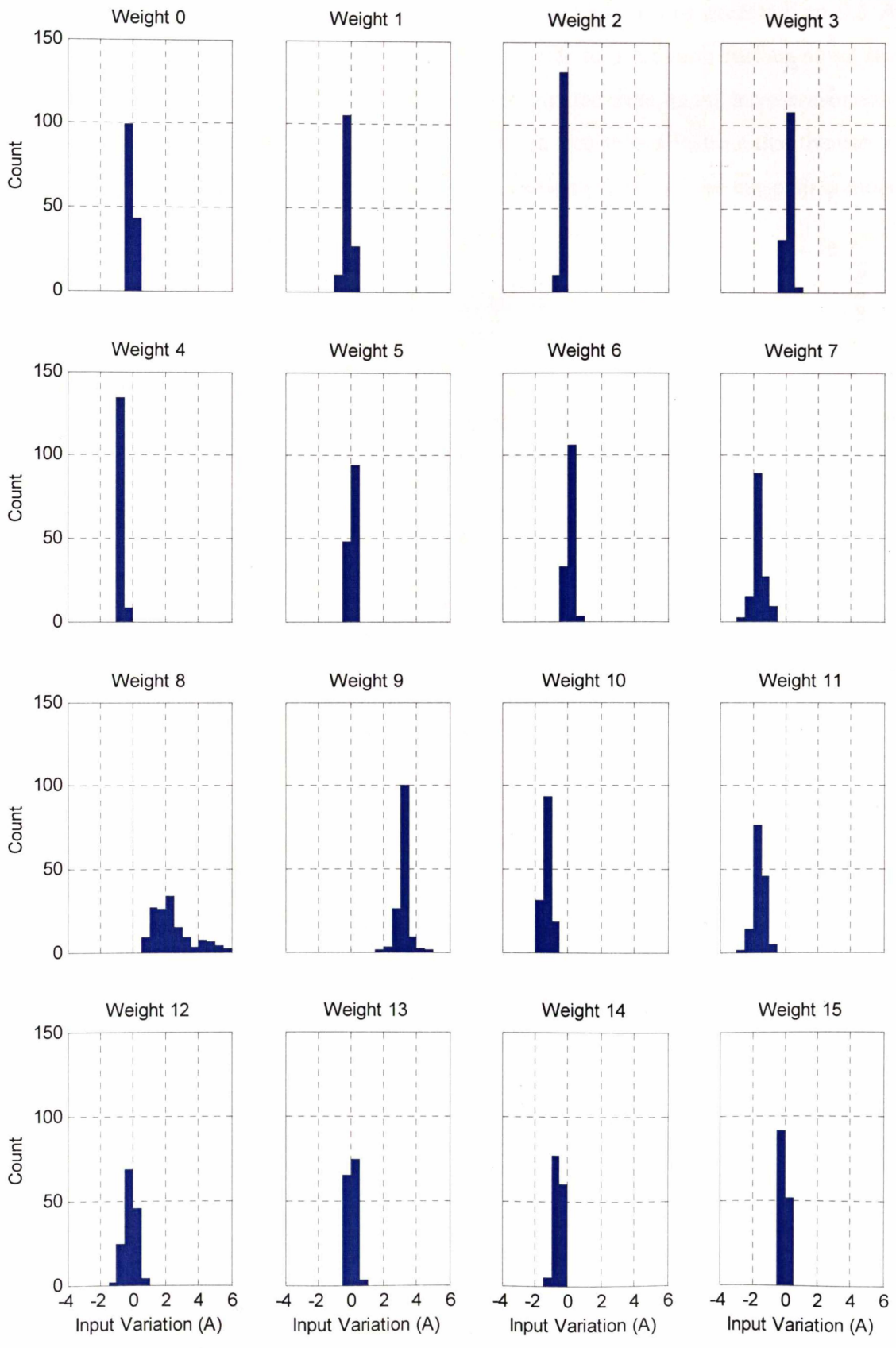

FIGURE 7.13 **Histograms of distances between individual completed cut weights and completed cut signals.**

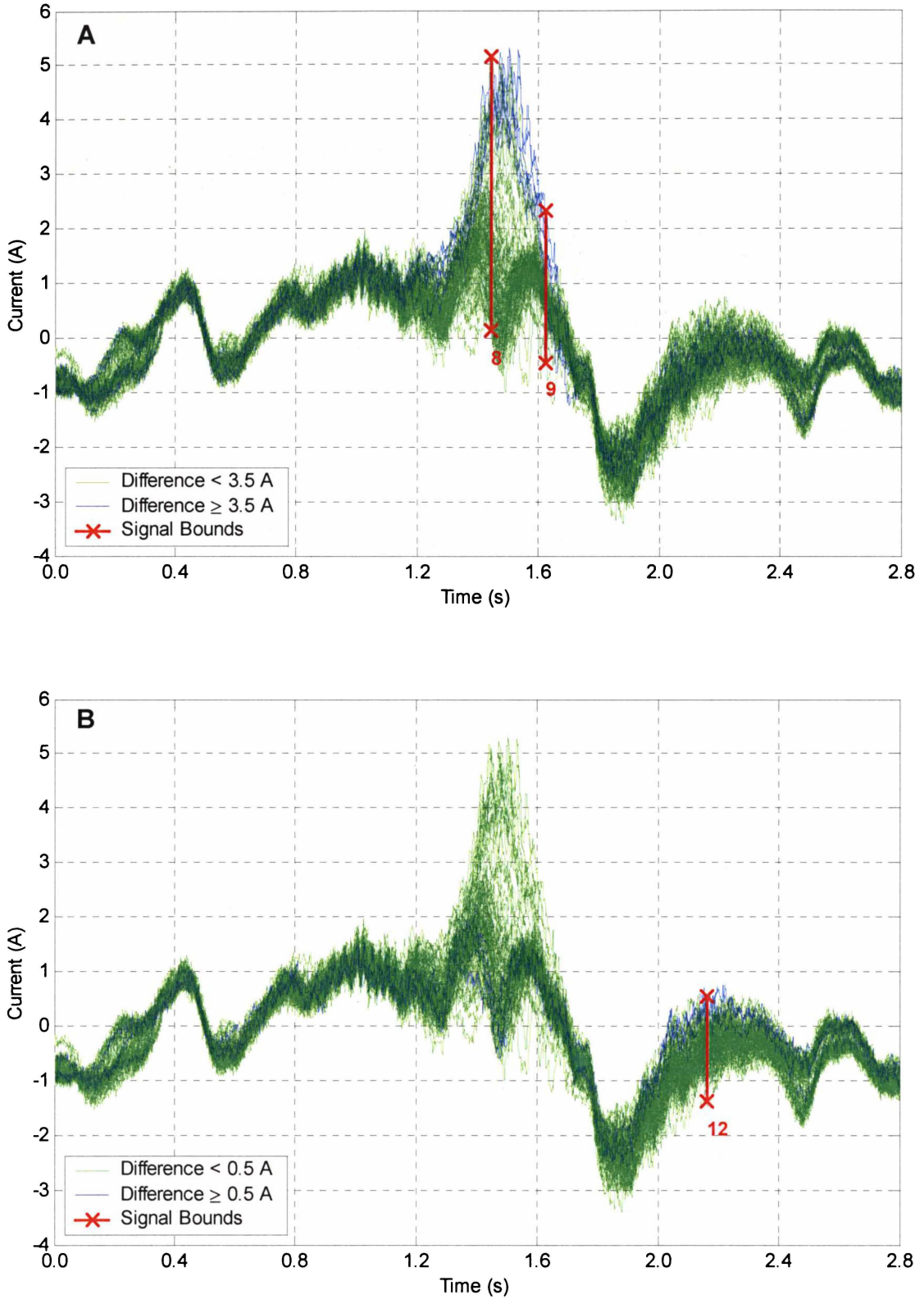

FIGURE 7.14 **Leading leg completed cuts with a variation of a) greater than 3.5 A for completed cut weights 8 and 9, and b) greater than 0.5 A for completed cut weight 12.**

A second example in Figure 7.14b shows cuts with a variation of greater than 0.5 A from the value of weight 12. This weight corresponds to a location further down the foreleg towards the end of the cut-path. The highlighted cuts again have reasonably normal loads over the rest of the signal. Both of these examples illustrate that the use of individual weights for isolating load variations in particular parts of the cut-path is more effective than using the cumulative neuron output.

7.3.3 KRL Implementation

The fault detection interrupt routine described in Section 7.2.4 must be modified slightly to provide access to the individual weight variations (refer Figure 7.15).

```
GLOBAL DEF Fault_Detection()

...
IF (A5_Curr_I < 16) THEN                    ;Is array filled yet?
  ...
ELSE
  ...
  IF (LVQ_Learning == TRUE) THEN            ;Online learning is enabled
    ...
  ELSE                                      ;Online learning is disabled
    FOR N = 1 TO 16                         ;Calculate signal variations
      IF_Weight[N] = (IW_IF[LT,N]-A5_Curr[N])
      CC_Weight[N] = (IW_CC[LT,N]-A5_Curr[N])
      IF (CC_Weight[N] > Threshold[N]) THEN
        Excessive_Load[N] = TRUE            ;Set excessive load indicator
      ELSE
        Excessive_Load[N] = FALSE           ;Reset excessive load indicator
      ENDIF
    ENDFOR
    FOR K = 1 TO 17
      Sum[K] = 0.0                          ;Reset temporary variable
      N = 1                                 ;Reset weight index
      WHILE (N < K)                         ;Endpoint weight = CC weight
        Sum[K] = Sum[K]+(CC_Weight[N]*CC_Weight[N])
        N = N + 1                           ;Increment weight index
      ENDWHILE
      WHILE (N <= 16)                       ;Endpoint weight = IF weight
        Sum[K] = Sum[K]+(IF_Weight[N]*IF_Weight[N])
        N = N + 1                           ;Increment weight index
      ENDWHILE
      Neuron[K] = -SQRT(Sum[K])             ;Calculate neuron activation
    ENDFOR
    ...
  ENDIF
ENDIF

END
```

FIGURE 7.15 Modified interrupt routine for the detection of excessive loads (modifications are highlighted in bold).

The difference between the neuron weights and the input signal are calculated separately and stored in arrays `IF_Weight[]` and `CC_Weight[]`. The prior calculation of `IF_Weight[]` and `CC_Weight[]` also improves the efficiency of the interrupt routine since the individual weight contributions are no longer calculated every time `Sum[K]` is updated. The signal variation for the completed cut weight is then compared to a `Threshold[]`, and an `Excessive_Load[]` flag is either set or reset.

The values specified in `Threshold[]` can be based on observations of the system, similar to those obtained via inspection of Figure 7.13. An alternative method would be to analyse the load variations continuously and use a statistical measure to set the threshold values.

7.4 PROPOSED EXTENSION OF PATH OPTIMISATION

The path optimisation algorithm developed in Chapter 6 can modify the top of the cut-path in order to maximise the success rate of the Y-cutting system. However, generic application of the algorithm could require the optimisation of the entire path. The ability to detect end-point failures and quantify load variations provides pertinent information for the modification of specific parts of a robotic cut-path.

7.4.1 Localisation of Cut-Path Load Data

The temporal detection of faults and load abnormalities need to be augmented with spatial information to allow the localisation of specific events in relation to the defined path waypoints. Figure 7.16 shows the modifications made to the leg-guide and fault detection interrupt routines. The current position of the robot in the conveyor coordinate system (refer Section 3.4.1) can be obtained via the system variable `$POS_ACT`, which is a structure containing the three Cartesian coordinates and three Euler angles. This positional information is captured at the same time as the motor current samples from axis A5, and is stored in an array `Path_Position[]`. This positional information can then be used to synchronise the load data with the cut-path.

```
GLOBAL DEF Leg_Guide_Interrupt()

A5_Curr[1] = $CURR_ACT[5]*$CURR_MAX[5]/100
                                        ;Get A5 motor current value
Path_Position[1] = $POS_ACT             ;Get current path position
A5_Curr_I = 1                           ;Reset array index variable
...

END

-------------------------------------------------------------------------

GLOBAL DEF Fault_Detection()

...
A5_Curr[A5_Curr_I] = $CURR_ACT[5]*$CURR_MAX[5]/100
                                        ;Get A5 motor current value
Path_Position[A5_Curr_I] = $POS_ACT     ;Get current path position
...

END
```

FIGURE 7.16 Modified interrupt routines to capture robot positional information concurrently with axis load data (modifications in bold).

7.4.2 Optimisation of the Entire Cut-Path

The ability to synchronise end-point failures and load abnormalities with the actual cut-path means that optimisation of individual path waypoints should be possible. If an end-point failure or excessive load condition is detected at a certain location then the waypoint offsets near that point should be modified to correct the situation.

An end-point failure is caused by the cut-path being too far from the foreleg, resulting in the tool ripping out of the pelt prematurely. Conversely, an excessive load is caused by the cut-path being too close to the foreleg, particularly around the knee of the carcass. A basic strategy can be developed based on these observations:

- If the load is excessive near a particular waypoint then pull the path away from the foreleg.
- If an end-point failure is detected near a waypoint then shift the path towards the foreleg.

An obvious method for moving the path relative to the foreleg is to change the Y-coordinate offset for the particular waypoint. Alternatively, it is possible that a modification of the X-coordinate would be more appropriate, shifting the path laterally relative to the foreleg. However, there is little justification for selecting one coordinate direction over another.

A second method is to use a modified version of the line search algorithm proposed in the previous chapter. An end-point failure or excessive load condition would trigger the algorithm and the line search applied to the nearest waypoint. The search captures the load variation at each sample point along with the occurrence of end-point failures. The optimisation algorithm would then adjust the waypoint offsets to minimise the occurrence of both conditions. This minimisation effectively acts to balance these two conditions and obtain an optimum cutting load, since one condition is caused by an excessive load and the other caused by an insufficient load (being equivalent to an insertion failure). A limitation of this method is that it requires a time-consuming line search to be completed, rather than making a simple ad-hoc change to the waypoints as proposed with the previous method. However, the optimisation algorithm would complete a more rigorous search of the surrounding parameter space and has the potential to converge to an optimal solution. Either method would require rigid constraints placed on the permitted changes to the waypoints to ensure that the modified path does not produce a cut that is unacceptable to the plant.

It is difficult to justify changes to the cut-path based solely on the load experienced by the robot. An optimisation algorithm will ultimately require some form of feedback relating to the quality and acceptably of the cut-path. A possible source of feedback would be a machine vision system used to determine the position of the cut-path relative to the carcass. However, the isolation of the completed path is difficult given that wool can obscure the cut. A more convenient form of feedback would be from a human operator or plant supervisor via some sort of user interface. The interface would allow the user to specify if a particular part of the cut-path is acceptable or not. This feedback would also allow for the correction of incorrect cut-path faults, which had previously been identified as subjective events dependent on the observer (refer Section 5.2).

7.5 GENERIC IMPLEMENTATION

There are several outstanding factors that must be addressed before the developed fault detection and path optimisation systems can be employed in other robotic applications.

Fault detection for the Y-cutting system uses data from only one robot axis. While data from other axes did not improve the classification performance, this simplification may not always be possible. Other applications may produce process loads that are observable in all six axes. In these cases, it will be necessary to change the sections of the interrupt routines that deal with the capture of axis data, although this is a relatively straightforward modification.

Other automated processes may require higher sampling frequencies to capture discriminating variations in the load signals. This change also increases the required number of weights for each LVQ neuron. Both of these changes can be incorporated via some very simple code changes to the interrupt routines. Note that the KUKA controller only allows sampling of system variables every 12 ms, so processes that undergo rapid load variations may require an external data acquisition system. However, it is expected that most processes involving natural products would exhibit similar characteristics to the Y-cutting system.

Although the KUKA robotic system has proven to be a robust and reliable platform for automated Y-cutting, it is possible that future applications could use robots from different manufacturers. Therefore, it is important that the developed algorithms can be easily transferred to different robotic platforms. The fault detection module also needs to have access to motor current data from the robots axes. Fortunately, most major robot manufacturers provide high-level programming environments with access to such data. While the language syntax and structure may vary, it should be a straightforward task to rewrite the KUKA code for a different type of robot.

8. CONCLUSIONS

8.1 THESIS EVALUATION

This thesis has primarily focussed on the development of an automatic fault detection module and path optimisation algorithm for use with a robotic Y-cutting system.

8.1.1 Initial Fault Detection Development

Initial development of fault detection strategies used an older IRL8L Y-cutting system. Data relating to 32 Y-cuts were acquired, and features of the cut-path were related to motor current data from the shoulder axis of the robot. The largest increase in the load experienced by the axis occurred as the tool traversed the knee. There was a clear difference between a cut that failed to insert and a completed Y-cut. If the tool pulled out of the cut prematurely, the current signal quickly returned to the level of an insertion failure.

Two knowledge-based fault detection strategies were developed to classify captured signal data into three classes of cut and to determine the cut end-point. The first strategy applied a threshold of 0.00 A to the current signal, and the cut end-point was deemed to have been reached when the signal remained below this threshold level for more than 0.04 seconds. This strategy correctly classified 78.1% of the available datasets. The error in the end-point determination was 0.17 seconds. The second strategy applied the same threshold of 0.00 A, but detected regions of the signal that were above this level for more than 0.11 seconds. This strategy had an improved classification rate of 81.3% and a more accurate end-point determination error of 0.14 seconds.

Four neural network strategies were tested using different architectures. A multilayer perceptron (MLP) strategy correctly classified 70.6% of the cuts, with an end-point error of 0.26 seconds. A strategy using a radial basis function (RBF) network had an

almost equivalent performance with 71.0% of cuts correctly classified and an end-point error of 0.21 seconds. Two strategies used learning vector quantisation (LVQ) networks trained with different learning algorithms. The LVQ network classified only 58.1% of the cuts correctly when trained with the LVQ1 algorithm. This performance improved when the LVQ2.1 training algorithm was used, with a classification rate of 69.7%. The LVQ strategies were not used to determine the cut end-points. None of the tested neural network strategies were as effective as the knowledge-based strategies at detecting end-point failures, although all of the tested strategies reliably detected insertion failures.

A series of hybrid strategies were developed by combining the second knowledge-based strategy with the four neural network strategies. The performance of these strategies was superior to the basic neural networks in all cases except the LVQ network trained with the LVQ2.1 algorithm. The best performing hybrid strategy used the MLP network, resulting in a classification success of 79.0% for the three classes of cut and an end-point error of 0.15 seconds. This level of performance approached that of the second knowledge-based strategy. However, none of the tested strategies were able to meet the fault classification requirement of 95%. Even though the fault detection performance was limited by the small size of the available dataset, this work demonstrated the potential of a neural network-based system to detect faults from process load variations. It was thought that the classification performance could be improved by obtaining a larger dataset of Y-cut signals, with a bigger population of samples providing a better representation of the different classes of cut.

8.1.2 Implementation of a Fault Detection Module

The commissioning of the full KUKA Y-cutting system enabled the capture of a greater number of axial load datasets. The process had been substantially improved since the development work with the IRL8L robot, resulting in the need to only detect insertion failures. Although the kinematics of the KUKA were more complicated than the IRL8L, the Y-cutting process loads were still observable in the motor current data from several of the robot axes.

An LVQ-based fault detection module was developed to classify insertion failures from the motor current of axis A5. The LVQ network successfully classified 98% of the presented datasets during offline training. The addition of data from other axes did not improve the classification success. The sampling rate was decreased from 2 ms to 180 ms without adversely affecting the classification performance of the network.

The LVQ fault detection module was implemented in the KUKA Y-cutting system using two interrupt routines. An online training scheme was also developed to allow for the automatic adjustment of the weights of the LVQ network. This training scheme was shown to quickly modify the weights to match changes in the Y-cutting process, requiring only 350 cuts to retrain the network. The retrained fault detection module was tested during normal operation of the Y-cutting system and successfully classified 100% of 502 cuts.

The provision of the online training scheme is an important feature of the fault detection module. If major changes are made to the path or the speed of the robot, then the sock-ringer can be switched off to retrain the insertion failure weights. The completed cut weights require the sock-ringer to be switched on, but can be retrained while the system continues Y-cutting.

A key advantage of the developed module is that it does not require any additional hardware. The initial fault detection work with the custom-built IRL8L robot required a separate computer and data acquisition system. While this would have been relatively straightforward to implement, the extra cost may have restricted the uptake of the system. Instead, the KUKA robot provided automatic access to the necessary motor current data and the LVQ network was easily integrated within the structure of the existing KRL Y-cutting program.

8.1.3 Development of a Path Optimisation Algorithm

The provision of this fault detection module allowed the development of an automated path optimisation system. This system was designed to optimise the first two waypoints of the cut-path and maximise the insertion success rate. Initial analyses identified

fifteen path parameters that needed to be optimised. A series of cutting trials were conducted using the Goulburn Y-cutting robot to determine the nature of the success-rate objective function. Due to time constraints, data was collected for only six parameters.

The observed success rates generally matched the expected form of the objective function. No correlation was found between the brisket measurement for the individual carcasses and the outcome of each cut. A linear model gave very poor correlations between the six tested parameters and the success rate for each cutting trial, with R^2 values less than 0.28. A Gaussian model provided a better fit to the data with R^2 values greater than 0.70. This model was used for subsequent development and simulation work.

The standard deviation of each Gaussian model provided an indication of the relative importance of the parameters. Smaller standard deviations indicated that the success rate was more susceptible to changes in the particular parameter. Using this metric, the height of the insertion point *IPZ* was the most important path parameter. The least important of the tested parameters was the waypoint offset *BA2Y*. This ranking indicated the order in which the parameters should be optimised.

An optimisation algorithm was developed based on the existing one-dimensional manual tuning process. The algorithm used a line search along each parameter to estimate the position of the optimum parameter value. Key internal parameters of the algorithm were the line search ratio R and the line sample size m. The fastest convergence to a 98% success rate occurred for $R = 1.7$ and $m = 21$, with the mean number of required Y-cuts being 1320. All of the tested values of m were capable of converging in less than 2000 iterations.

The algorithm was improved by modifying the sampling procedure used to determine the success rate at the conclusion of each line search. The success rate sampling was halted prematurely if the number of observed insertion failures exceeded the permitted number of failures. The modified optimisation algorithm typically required 25% fewer Y-cuts to converge than the original algorithm. The faster convergence for the modified algorithm occurred for $R = 1.6$ and $m = 11$, with the mean number of Y-cuts being 960.

Various success rate criteria were also tested for the termination of the line search algorithm. Stronger criteria took longer to converge but resulted in lower residual errors in the estimate of the parameters.

The optimisation algorithm was shown to continually improve the parameter estimate during simulated operation. The optimisation algorithm was also able to respond to changes in the optimum parameter values, tracking both step and ramp changes successfully.

A constant line search length was tested to contrast with the standard deviation-scaled variable length used in prior testing. The variable method took an average of 13.7% fewer Y-cuts to converge than the constant method, depending on the value of R and m. While the constant method required more iterations than the variable method, this test demonstrated that a constant line search could still be applied without significantly affecting the convergence in situations where an estimate of the success-rate objective function was not available. This is important if the optimisation algorithm is to be generically applied to other processes without the need for exhaustive preliminary testing to determine the nature of the objective function.

Following the successful application of the optimisation algorithm to the simplified six-parameter model, the Gaussians for the remaining nine critical parameters were estimated. The fastest convergence for the expanded fifteen-parameter model occurred for $R = 1.0$ and $m = 21$, with the mean number of simulated Y-cuts being 4760. This equates to approximately a day and a half of processing in a plant operating at 8 ccs/min. This convergence time is significantly faster than the existing manual tuning process, which has previously required weeks or months of tuning to achieve a success rate of 98%. If an automation engineer has an indicative charge-out rate of NZ$1,000 per day and two engineers are required to observe and tune the system, then each day of tuning costs NZ$2,000. There are also additional costs incurred by the plant while the y-cutting system is being tuned. An extra downstream butcher (at an indicative cost of NZ$200 per day) could be required to manually Y-cut any carcasses that the robot misses. Even if the optimisation algorithm conservatively reduces the tuning time by 5 days, then this could equate to a financial saving of NZ$11,000.

The optimisation algorithm was modified to allow the more effective tracking of step and ramp changes. An adaptive line search ratio was used, with $R = 1.0$ if the success rate was less than 90%, otherwise a smaller ratio $R = 0.4$ was used. This modification allowed the algorithm to fine-tune the parameter estimate and increased the number of iterations spent on normal Y-cutting rather than tuning. The modified algorithm was successfully applied to the expanded set of path parameters, tracking both step and ramp changes in the optimum parameter values and reducing the residual error over continuous periods of operation.

8.1.4 Expansion of the Fault Detection Module

The fault detection module for insertion failures was expanded to allow the detection of end-point failures. Extra neurons were added to the LVQ network, with these neurons having a combination of the weights from the base insertion failure and completed cut neurons. Each neuron corresponded to a particular end-point class labelled EP1 to EP17, with neuron EP1 equivalent to an insertion failure and EP17 equivalent to a completed cut. End-point failure signals were simulated by combining randomly selected insertion failure and completed cut signals at randomly selected points. The expanded network was tested with 100,000 simulated signals. The average classification error was 1.72 end-point classes or 0.31 seconds. The classification of 75.7% of the cuts was within two end-point classes of the target class. The end-point determination error was smaller for end-points that corresponded to larger discriminating differences between the insertion failure and completed cut weights. The smallest error occurred for end-point EP10, with an error of 0.19 seconds. Neighbouring end-points EP9 and EP11 also had an error of less than 0.20 seconds. These three end-points all corresponded to the peak loading as the tool traversed the knee, which was the largest discriminating feature of the cut-path.

The fault detection module was also modified to detect variations in the load experienced by the robot. One method used the output of the LVQ neurons as a measure of the deviation from a normal cutting operation. However, this lacked the ability to identify specific regions of the cut-path that differed from a normal cut. A more direct method used the difference between individual weights and the input signal.

Regions of excessive signal variation were identified in captured datasets and modifications to the KUKA interrupt routines were proposed to detect these cut abnormalities.

8.2 CONTRIBUTIONS TO ORIGINAL RESEARCH

This thesis has made a number of significant contributions to the field of robotics, particularly related to the robotic handling and processing of highly varying natural products:

- Process faults relating to the automated Y-cutting of sheep carcasses have been characterised from robotic axial motor current signals. More advanced sensing techniques were not possible because of the harsh meat-plant environment. However, the limited dataset of motor current signals were successfully used to identify specific process faults and served as the basis for the subsequent development of a fault detection module and path optimisation algorithm.
- The effectiveness of MLP, RBF and LVQ neural networks has been investigated for detecting faults and localising cut end-points from concurrently presented motor current signals.
    - The performance of these neural networks has been compared to two knowledge-based fault detection strategies.
    - The optimum neuron width to minimise the output error has been established for a fault detecting RBF network.
    - The classification success of LVQ networks trained with the LVQ1 algorithm has been compared with networks trained with the LVQ2.1 algorithm.
    - A hybrid neural network strategy has been developed to detect process faults from conditioned axial motor current signals.
- The developed LVQ fault detection module is the first known application of neural network-based fault detection for a robotic system used to process highly varying natural products.
    - An investigation has been completed into the use of single and multiple robotic axis data for detecting Y-cutting process faults.

  - The effect of reducing the sampling frequency of robotic axial load data on the classification performance of the LVQ network has been established for the Y-cutting process.
  - The LVQ module has been successfully integrated within a commercial automated Y-cutting system to provide continuous and reliable observation of the outcome of each cut, eliminating the need for a person to compile trial statistics manually.
  - An online training scheme for an LVQ network has been developed and implemented within a KUKA robotic platform.
- The effect of additive noise on the classification performance of an LVQ network has been investigated.
- The fault detection performance of an expanded LVQ network has been evaluated.
  - An investigation has been completed into the ability of the expanded LVQ network to detect and isolate cut end-points.
  - The LVQ network has been extended to isolate cutting load variations using individual neuron weights.
- A novel path optimisation algorithm based on an adaptation of the sequential random search algorithm has been examined.
  - A heuristic analysis of parameters used to define a robotic Y-cut path has been completed.
  - The Y-cutting success-rate objective function has been modelled from cutting trial data using a multivariate Gaussian distribution.
  - An adaptive line search ratio has been developed to improve the tracking performance of the path optimisation algorithm.
  - The developed path optimisation algorithm has demonstrated significant timesavings over the existing manual tuning process.
  - In conjunction with the fault detection module, the optimisation algorithm can provide the means to automate the entire path tuning process.
- Path parameters used to define the top of the Y-cut have been ranked according to their effect on the success rate of the system. This importance ranking has not been identified during previous Y-cut installation work.
- The ConveyorTech counter module used by the KUKA robot for tracking moving conveyors has been adapted to allow for the simultaneous tracking of multiple

objects. This is a novel application of existing technology, providing additional functionality that was considered unworkable by KUKA representatives.

8.3 RECOMMENDATIONS FOR FUTURE WORK

There are several opportunities for further research and development that stem from this thesis.

The optimisation algorithm developed in Chapter 6 requires implementation and testing as part of the full KUKA Y-cutting system. The algorithm converged significantly faster than the manual tuning process during computer-based simulations, but must be tested under real-world conditions. This will require the integration of the algorithm within the overall structure of the Y-cutting program. The main barrier to the immediate implementation of the optimisation algorithm is that the Y-cutting system has been removed from the Goulburn meat-plant following the successful demonstration of the technology under Australian conditions. However, a number of other plants in both Australia and New Zealand have indicated that they wish to purchase the system, meaning that there will soon be the opportunity to test the optimisation algorithm.

It may be possible to reduce the number of path parameters that need to be optimised from the fifteen that have been identified. This parameter reduction could be possible following the testing of the optimisation algorithm in the real Y-cutting system and after further observation of the process. If the tool maintains a constant yaw-angle A throughout the cut-path (refer Section 6.6), then both $BA1A$ and $BA2A$ would not need to be optimised. There is also the possibility that a constant robot speed could be used, given that the overall quality of the cut would decrease for very fast speed settings. During the automated tuning of the system, it is important that the upstream handling and processing of the carcass be of a high standard, thereby reducing the noise that the optimisation algorithm will encounter. The legs should be aligned in the spreaders with a consistent orientation, and the sock-ringer should be adjusted to provide a consistent opening cut at the top of the leg. The maintenance of these processing standards along with a decrease in the dimensionality of the parameter space should result in the

reduction of the overall tuning time from the approximately one and a half days of processing required by the optimisation algorithm.

The proposed extensions to the fault detection module also need to be tested under real-world conditions. It should be relatively easy to test the modified interrupt routines for their ability to detect load variations. However, it may be difficult to test the detection of end-point failures since the current Y-cutting system has never exhibited this fault condition. One option would be to detune the cut-path so that the tool ripped out of the pelt near the knee. A second option would be to manually make a horizontal cut across the foreleg prior to the Y-cutting operation, which could cause the tool to prematurely withdraw from the pelt.

An immediate application of the fault detection module is that a Y-cut could be reattempted if a fault is detected. This might not be possible if the conveyor-chain is moving relatively fast (>6 ccs/min), since the foreleg may have already moved out of the workspace of the robot. However, for slower chain speeds (<6 ccs/min) there should be sufficient opportunity for the robot to repeat the Y-cut.

The fault detection module also presents several product development opportunities. An option that could be provided to purchasers of the Y-cutting system is a down-stream signalling device. Currently, every carcass has to be manually inspected to determine if the automated Y-cut has been successfully completed. A signalling system would allow the identification of particular carcasses that the robotic Y-cutter had missed or incorrectly cut and that needed to be manually Y-cut by a down-stream butcher. A possible implementation of this signalling system would be a row of indicator lights placed along the front of the conveyor rail. If a fault was detected then the lights could be illuminated in sequence to track the relevant carcass along the conveyor-chain.

Another opportunity presented by the fault detection module is the automatic collection and calculation of statistics relating to the performance of the automated Y-cutting system. This information is readily available, since the module has demonstrated the ability to determine the outcome of each cut reliably. The controller of the KUKA robot also has the capacity to be interrogated via an Ethernet network connection. This

would allow the performance of the Y-cutting system to be monitored from a remote location by plant management. Service personnel could also access this information to assist with diagnosis and maintenance issues; a prolonged deterioration in performance could indicate that the tool requires servicing.

It is also possible that the fault detection and path optimisation algorithms could be integrated with a flexible robotic programming environment that is being developed by the Automation Systems team (Rajapaske & Hildreth, 2004). This environment incorporates a LabVIEW toolkit that allows a user to program an ABB industrial robot from a remote computer via a graphical interface. This environment is being extended to allow a range of robots to be programmed and interfaced to a variety of sensors, all from a remote location. The fault detection and path optimisation algorithms could be included as separate software functions within the toolkit to provide additional functionality to the programmed robot.

8.4 SUMMARY

This thesis has addressed the provision of automatic fault detection and path optimisation for a robotic Y-cutting system. The following requirements were specified at the beginning of the thesis:

- Define and characterise faults that occur in the Y-cut process.
- Develop a system for detecting the incidence of process faults automatically and reliably, with a fault classification rate of at least 95%.
- Improve the existing manual tuning process of the Y-cutting system by reducing the time taken to achieve a cutting success rate of 98%.
- Provide generic fault detection and tuning solutions that can be applied to other IRL robotic devices.

Five process faults have been identified and defined, and two of these conditions were characterised using robotic axial motor current signals. A neural network-based fault detection module was developed to detect insertion failures. The fault detection module successfully classified 98% of the presented cut signals during offline training. Testing

of the fault detection system in an Australian meat-processing plant resulted in 100% successful classification of 502 cuts. This performance exceeded the specified classification rate of 95% required by the fault detection system. Additionally, an online training scheme was developed to allow for the retraining of the neural network weights as required. Although the module was only required to detect one process fault, provision has been made for the expansion of the module to detect additional fault conditions. This module can also be generically applied to other robotic processes.

An adaptive path optimisation algorithm was developed to meet the third requirement of the thesis. This algorithm was designed to work in conjunction with the fault detection module to provide an automated system to tune a robotic path. Simulations involving the optimisation of fifteen path parameters required an average of 4760 Y-cuts to achieve a cutting success rate of 98%, equating to approximately 1.5 days of processing in a typical meat-plant. This is significantly faster than the manual tuning process, which had required weeks or months of continuous tuning. The optimisation algorithm was shown to continually improve the parameter estimate and track changes in the optimum parameter values over time. Six of the path parameters have been ranked according to the standard deviation of the Gaussian regression model. The most important parameter for an optimum success rate is the height of the insertion point *IPZ*, followed by waypoint offset *BA1Y* and insertion point depth *IPY*. The least important parameter was the waypoint offset *BA2Y*.

In conclusion, the developed fault detection module and path optimisation algorithm provide the means to automatically tune the cut-path of a robotic Y-cutting system. The fault detection module is the first known application of a neural network in a meat-processing robot, and the optimisation algorithm has the potential to significantly reduce time, labour and financial costs associated with the commissioning of a new Y-cutting system. These developments have the potential to increase the uptake of meat automation technology and represent an important competitive advantage for current and future robotic systems produced by Industrial Research Limited.

APPENDIX A - REFERENCES

Accident Compensation Corporation (2002), *New Zealand Meat Industry Health and Safety Plan*, HTTP://WWW.ACC.ORG.NZ.

Ali, M.M., Törn, A. & Viitanen, S. (1997), "A numerical comparison of some modified controlled random search algorithms," *Journal of Global Optimization*, vol. 11, pp. 377-385.

Andersson, J.-E. & Johansson, G. (2001), "Robot control for wood carving operations," *Mechatronics*, vol. 11, pp. 475-490.

Angeli, C. & Chatzinikolaou, A. (1999), "Fault prediction and compensation functions in a diagnostic knowledge-based system for hydraulic systems," *Journal of Intelligent and Robotic Systems*, vol. 25, no. 2, pp. 153-165.

Australian Government (1994), "Industry commission: meat processing," *Australian Government Publishing Service*, vol. 1, report no. 38.

Australian Standard AS1939 (1990), *Classification of Degrees of Protection Provided by Enclosure for Electrical Equipment.*

Authier, J.F. (1992), "Mechanical dressing and beef processing/boning," *MIRINZ 27th Meat Industry Research Conference*, Hamilton, New Zealand, pp. 327-338.

Berkan, R.C. & Trubatch, S.L. (1997), *Fuzzy Systems Design Principles: Building Fuzzy IF-THEN Rule Bases*, New York: IEEE Press.

Bermejo, S. & Cabestany, J. (2004), "Local averaging of ensembles of LVQ-based nearest neighbor classifiers," *Applied Intelligence*, vol. 20, pp. 47-58.

Brooks, R.A. & Mataric, M.J. (1993), "Real robots, real learning problems," *Robot Learning*, Connell, J.H. & Mahadevan, S. ed., Kluwer Academic, pp. 193-213.

Campa, G., Fravolini, M.L., Seanor, B., Napolitano, M.R., Del Gobbo, D., Yu, G. & Gururajan, S. (2002), "On-line learning neural networks for sensor validation for the flight control system of a B777 research scale model," *International Journal of Robust and Nonlinear Control*, vol. 12, pp. 987-1007.

Cheah, C.C. & Wang, D. (1998), "Learning impedance control for robotic manipulators," *IEEE Transactions on Robotics and Automation*, vol. 14, no. 3, pp. 452-465.

Chen, S., Cowan, C.F.N. & Grant, P.M. (1991), "Orthogonal least squares learning algorithm for radial basis function networks," *IEEE Transactions on Neural Networks*, vol. 2, pp. 302-309.

Chen, Y.M. & Hseuh, C.S. (2001), "Complementary data fusion in vision-guide and control of robotic tracking," *Robotica*, vol. 19, pp. 53-58.

Country-Wide Publications (2004), "Meat plant technology attracts Aussie funding", *Country-Wide South Magazine*, HTTP://WWW.COUNTRY-WIDE.CO.NZ.

Cox, D.D. & John, S. (1997), "SDO: a statistical method for global optimization," *Multidisciplinary Design Optimization: State of the Art*, Alexandrov, N. & Hussaini, M.Y. ed., Philadelphia: SIAM, pp. 315-329.

Darken, C., Chang, J. & Moody, J. (1992), "Learning rate schedules for faster stochastic gradient search," *Neural Networks for Signal Processing 2 – Proceedings of the 1992 IEEE Workshop*, White, D.A. & Sofge, D.A. ed., Piscataway: IEEE Press, pp. 3-12.

De Silva, C.W. (1995), *Intelligent Control: Fuzzy Logic Applications*, Boca Raton: CRC Press.

Duda, R.O., Hart, P.E. & Stork, D.G. (2000), *Pattern Classification*, 2nd Edition, New York: John Wiley & Sons.

Esposito, W.R. & Floudas, C.A. (2000), "Deterministic global optimization in nonlinear optimal control problems," *Journal of Global Optimization*, vol. 17, pp. 97-126.

Floreano, D. & Urzelai, J. (2000), "Evolutionary robots with on-line self-organisation and behavioural fitness," *Neural Networks*, vol. 13, pp. 431-443.

Franklin, G.F., Powell, J.D. & Emami-Naeini, A. (1994), *Feedback Control of Dynamic Systems*, 3rd Edition, Reading: Addison-Wesley.

Freyermuth, B. (1991), "An approach to model based fault diagnosis of industrial robots," *Proceedings of the IEEE International Conference on Robotics and Automation*, Sacremento, USA, pp. 1350-1356.

Friedrich, W.E. (1995), "Increased reliability by effective use of sensor information: a shop floor application of sensor-aided robotic handling," *Proceedings of the 2nd New Zealand International Two-Stream Conference on Artificial Neural Networks and Expert Systems*, Dunedin, New Zealand, pp. 359-364.

Genovesi, A., Harmand, J. & Steyer, J-P. (2000), "Integrated fault detection and isolation: application to a winery's wastewater treatment plant," *Applied Intelligence*, vol. 13, pp. 59-76.

Gomez, M.R., Ventosa, J.E. & Aramendia, G.A. (1998), "Expert system hardware for fault detection," *Applied Intelligence*, vol. 9, no. 3, pp. 245-262.

Gupta, P. & Sinha, N.K. (2000), "Intelligent control of robotic manipulators: experimental study using neural networks," *Mechatronics*, vol. 10, pp. 289-305.

Hebb, D.O. (1949), *The Organisation of Behaviour*, New York: John Wiley & Sons.

Hildreth, N., Lim, P., Yang, J. & Friedrich, W. (2003), "Integration of user/robotic tasks using shared telerobotic control," *Proceedings of the 7th World Multiconference on Systemics, Cybernetics and Informatics*, Orlando, USA.

Hopfield, J.J. (1982), "Neural networks and physical systems with emergent collective computational abilities," *Proceedings of the National Academy Of Science*, USA, vol. 79, pp. 2554-2558.

Hurd, S.A., Carnegie, D.A., Brown, N.R. & Gaynor, P.T. (2005), "Development of an intelligent robotic system for the automation of a meat-processing task," *International Journal of Intelligent Systems, Technologies and Applications*, vol. 1, nos. 1/2, pp. 32-48.

Isermann, R. (1993), "Fault diagnosis of machines via parameter estimation and knowledge processing – tutorial paper," *Automatica*, vol. 29, no. 4, pp. 815-835.

Isermann, R. (1998), "On fuzzy logic applications for automatic control, supervision, and fault diagnosis," *IEEE Transactions on Systems, Man, and Cybernetics – Part A: Systems and Humans*, vol. 28, no. 2, pp. 221- 235.

Isermann, R. & Ayoubi, M. (1996) "Fault detection and diagnosis with neuro-fuzzy-systems," *Proceedings of the 4th European Congress on Fuzzy Intelligent Technologies*, Aachen, Germany, pp. 1479-1491.

Jain, A.K., Duin, R.P.W. & Mao, J. (2000), "Statistical pattern recognition: a review," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 22, no. 1, pp. 4-37.

Juuma, T. & Parkkinen, R. (1994), "Application of a neural network to condition monitoring and fault diagnosis in a pressure system," *Proceedings of the IFAC Symposium on Fault Detection, Supervision and Safety for Technical Processes*, Espoo, Finland, pp. 603-609.

Kartalopoulos, S.V. (1996), *Understanding Neural Networks and Fuzzy Logic: Basic Concepts and Applications*, New York: IEEE Press.

Kassler, M. (2001), "Agricultural automation in the new millennium," *Computers and Electronics in Agriculture*, vol. 31, pp. 237-240.

Kirkpatrick, S., Gelatt, C.D. & Vecchi, M.P. (1983), "Optimization by simulated annealing," *Science*, vol. 220, pp. 671-680.

Kohonen, T. (1997), *Self-Organizing Maps*, 2nd Edition, New York: Springer-Verlag.

Kohonen, T., Hynninen, J., Kangas, J., Laaksonen, J. & Torkkola, K. (1996), "LVQ_PAK: The learning vector quantization program package," *Laboratory of Computer and Information Science Report A30*, Helsinki University of Technology.

Kopacek, P. (1999), "Intelligent manufacturing: present state and future trends," *Journal of Intelligent and Robotic Systems*, vol. 26, no. 3, pp. 217-229.

Köppen-Seliger, B. & Frank, P.M. (1999), "Fuzzy logic and neural networks in fault detection," *Fusion of Neural Networks, Fuzzy Sets, and Genetic Algorithms: Industrial Applications*, Jain, L.C. & Martin, N.M. ed., Boca Raton: CRC Press, pp. 171-209.

Li, Z. & Hinsch, A. (2003), "A new approach to detect the cutting positions for a robotic beef carcass scribing system," *Proceedings of the 2003 Australasian Conference on Robotics and Automation*, Brisbane, Australia.

Lippmann, R.P. (1988), "An introduction to computing with neural nets," *Artificial Neural Networks: Theoretical Concepts*, V.Vemuri ed., Computer Society Press, pp. 36-54.

Lu, Y., Chen, T.Q. & Hamilton, B. (1998), "A fuzzy diagnostic model and its application in automotive engineering diagnosis," *Applied Intelligence*, vol. 9, pp. 231-243.

Magoulas, G.D., Vrahatis, M.N. & Androulakis, G.S. (1999), "Improving the convergence of the backpropagation algorithm using learning rate adaptation methods," *Neural Computation*, vol. 11, pp. 1769-1796.

Makhoul, J., Roucos, S. & Gish, H. (1985), "Vector quantisation in speech coding," *Proceedings of the IEEE*, vol. 73, no. 11, pp. 1551-1588.

Malone, D.E., Friedrich, W.E., Spooner, N.R. & Lim, P.P.K. (1994), "Knowledge based control in the processing of highly varying products," *Proceedings of the IEEE International Conference on Robotics and Automation*, San Diego, USA, pp. 2903-2908.

Masri, S.F., Bekey, G.A. & Safford, F.B. (1980), "A global optimization algorithm using adaptive random search," *Applied Mathematics and Computation*, vol. 7, pp. 353-375.

Mbede, J.B., Wei, W. & Zhang, Q. (2001), "Fuzzy and recurrent neural network motion control among dynamic obstacles for robot manipulators," *Journal of Intelligent and Robotic Systems*, vol. 30, no. 2, pp. 155-177.

McCulloch, W.S. & Pitts, W. (1943), "A logical calculus of the ideas imminent in nervous activity," *Bulletin of Mathematical Biophysics*, vol. 5, pp. 115-133.

McDermott, E. (1997), *Discriminative Training for Speech Recognition*, Ph.D. Thesis, Waseda University.

Meat and Livestock Australia (2002), *Meat and Livestock Australia Annual Report*, HTTP://WWW.MLA.COM.AU.

Ministry of Agriculture and Forestry (2002), *Industry Standard 5: Slaughter and Dressing*, HTTP://WWW.MAF.GOVT.NZ.

Minsky, M.L. & Papert, S. (1969), *Perceptrons*, Cambridge: MIT Press.

Mitchell, M. (1996), *An Introduction to Genetic Algorithms*, Cambridge: MIT Press.

Moody, J. & Darken, C. (1989), "Fast learning in networks of locally-tuned processing units," *Neural Computation*, vol. 1, pp. 281-294.

Murata, N., Müller, K.-R. & Ziehe, A. (1997), "Adaptive on-line learning in changing environments," *Advances in Neural Information Processing Systems*, Mozer, M.C., Jordan, M.I. & Petsche, T. ed., Cambridge: MIT Press, vol. 9, pp. 599-605.

Murphy, R.R. (2000), *Introduction to AI Robotics*, Cambridge: MIT Press.

New Zealand Meat Board (2002), *New Zealand Meat Board 2002 Annual Report: Statistical Information*, HTTP://WWW.MEATNZ.CO.NZ.

Orr, G.B. & Leen, T.K. (1997), "Using curvature information for fast stochastic search," *Advances in Neural Information Processing Systems*, Mozer, M.C., Jordan, M.I. & Petsche, T. ed., Cambridge: MIT Press, vol. 9, pp. 606-612.

Ouafi, A.E., Guillot, M. & Bedrouni, A. (2001), "Accuracy enhancement of multi-axis CNC machines through on-line neurocompensation," *Journal of Intelligent Manufacturing*, vol. 11, pp. 535-545.

Patton, R.J. (1994), "Robust model-based fault diagnosis: the state of the art," *Proceedings of the IFAC Symposium on Fault Detection, Supervision and Safety for Technical Processes*, Espoo, Finland, pp. 1-24.

Powell, M.J.D. (1987), "Radial basis function for multi-variable interpolation: a review," in *Algorithms for Approximation*, Mason, J.C. & Cox, M.G. ed., London: Clarendon Press, pp. 143-167.

Price, W.L. (1978), "A controlled random search procedure for global optimization," *Towards Global Optimization 2*, Dixon, L.C.W. & Szegö, G.P. ed., Amsterdam: North-Holland, pp. 71-84.

Purnell, G. & Brown, T. (2003), "Automation for the meat industry," *New Food Magazine*, Issue 3, pp. 80-85.

Rajapakse, B. & Hildreth, N. (2004), "Remote access toolkit for industrial robots," *Proceedings of the Institution of Mechanical Engineers – Part B: Journal of Engineering Manufacture*, vol. 218, pp. 1017-1022.

Rastrigin, L.A. (1960), "Extremal control by the method of random scanning," *Automation and Remote Control*, vol. 21, pp. 891-896.

Rezayat, F. (1999), "Constrained SPSA controller for operations processes," *IEEE Transactions on Systems, Man, and Cybernetics – Part A: Systems and Humans*, vol. 29, no. 6, pp. 645-649.

Rinnooy Kan, A.H.G. & Timmer, G.T. (1987), "Stochastic global optimization methods – part I: clustering methods," *Mathematical Programming: Series A*, vol. 39, pp. 27-56.

Rosenblatt, F. (1959), *Principles of Neurodynamics*, New York: Spartan Books.

Rumelhart, D.E., Hinton, G.E. & Williams, R.J. (1986), "Learning internal representations by back-propagating errors," *Nature*, vol. 323, no. 99, pp. 533-536.

Sherstinsky, A. & Picard, R.W. (1996), "On the efficiency of the orthogonal least squares training method for radial basis function networks," *IEEE Transactions on Neural Networks*, vol. 7, no. 1, pp. 195 – 200.

Spooner, N.R. & Rodrigo, T.S. (1998), "Integrated architectures for a horticultural application," *Proceedings of the SPIE International Symposium on Intelligent Systems in Design and Manufacturing*, Boston, USA, vol. 3517, pp. 151-161.

Taylor, M.G. (1993), "Automated Y-cutting of sheep carcasses," *MEAT '93 Australian Meat Industry Research Conference*, Gold Coast, Australia, Session 7B, pp. 1-4.

Taylor, M.G & Brooking, G.N. (1994), "Y-cut dressing of sheep carcases using a robot," *MIRINZ 28th Meat Industry Research Conference*, Auckland, New Zealand, pp. 181-186.

Taylor, M.G. & Templer, R.G. (1997), "A washable robot suitable for meat processing," *Computers and Electronics in Agriculture*, vol. 16, pp. 113-123.

Templer, R.G. (2000), "How to benefit from advances in automation and robotics," *Food Technology Forum 2000*, Auckland, New Zealand.

Templer, R.G., Neuhaus, P., Nanu, A., Osborn, A. & Wichman, T.H. (1999), "New automation techniques for sheep and beef processing," *Proceedings of the 2nd Plenary Meeting of MACA*, Parma, Italy.

Templer, R.G., Nicholls, H.R. & Nicolle, T. (1998), "Robotics for meat processing – from research to commercialisation," *Proceedings of the 1st International Workshop on Future Automation for Meat Processing*, Frankfurt, Germany.

Templer, R.G., Nicolle, T., Nanu, A., Osborn, A. & Blenkinsopp, K. (2000), "New automation techniques for sheep and beef processing 2000," *Proceedings of the 1st International Meat Automation Congress*, Malaga, Spain, pp. 18-24.

Templer, R.G. & Wichman, T.H. (1997), "High speed robotic Y-cutting of lamb carcasses," *Proceedings of the 43rd International Congress of Meat Science and Technology*, Auckland, New Zealand, pp. 246-247.

Terra, M.H. & Tinós, R. (2001), "Fault detection and isolation in robotic manipulators via neural networks: a comparison among three architectures for residual analysis," *Journal of Robotic Systems*, vol. 18, no. 7, pp. 357-374.

Tessier, C. (2000), "Advanced robotic vision in pork plants," *Proceedings of the 1st International Meat Automation Congress*, Malaga, Spain, pp. 14-17.

The MathWorks, Inc. (2000), *Using MATLAB*, Version 6, Natick, MA, USA.

Torii, T. (2000), "Research in autonomous agriculture vehicles in Japan," *Computers and Electronics in Agriculture*, vol. 25, pp. 133-153.

Törn, A., Ali, M.M. & Viitanen, S. (1999), "Stochastic global optimization: problem classes and solution techniques," *Journal of Global Optimization*, vol. 14, pp. 437-447.

Van Henten, E.J., Van Tuijl, B.A.J., Hemming, J., Kornet, J.G., Bontsema, J. & Van Os, E.A. (2003), "Field test of an autonomous cucumber picking robot," *Biosystems Engineering*, vol. 86, no. 3, pp. 305-313.

Vemuri, A.T., Polycarpou, M.M. & Diakourtis, S.A. (1998), "Neural network based fault detection in robotic manipulators," *IEEE Transactions on Robotics and Automation*, vol. 14, no. 2, pp. 342-348.

Visinsky, M.L., Cavallaro, J.R. & Walker, I.D. (1994), "Expert system framework for fault detection and fault tolerance in robotics," *Computers in Electrical Engineering*, vol. 20, no. 5, pp. 421-435.

Wang, C.C. & Too, G.P.J. (2002), "Rotating machine fault detection based on HOS and artificial neural networks," *Journal of Intelligent Manufacturing*, vol. 13, pp. 283-293.

Werbos, P.J. (1974), *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences*, Ph.D. Thesis, Harvard University.

White, R.M. (1994), "FUTUTECH," *MIRINZ 28th Meat Industry Research Conference*, Auckland, New Zealand, pp. 165.

Widrow, B & Hoff, M.E. (1960), "Adaptive Switching Circuits," *1960 IRE WESCON Convention Record*, Part4, pp. 96-104.

Yang, S.X. & Meng, M. (2000), "Real-time collision-free path planning of robotic manipulators using neural network approaches," *Autonomous Robots*, vol. 9, no. 1, pp. 27-39.

Zabinsky, Z.B. (1998), "Stochastic methods for practical global optimization," *Journal of Global Optimization*, vol. 13, pp. 433-444.

Ziemke, T. (2000), "Remembering how to behave: recurrent neural networks for adaptive robot behaviour," *Recurrent Neural Networks: Design and Applications*, Medsker, L.R. & Jain, L.C. ed., Boca Raton: CRC Press, pp. 355-389.