Investigation of a Hybrid Switching Control System

Jacek A Narożny

Thesis prepared in fulfilment of the requirements for the Degree of
Master of Science in Electrical Engineering at the University of Cape Town.

Thesis Supervisor: Associate Professor Martin Braae

Date: December 1995

Acknowledgements

As in many undertakings of this nature, there are always a number of people behind the scenes who in one way or another made it all possible. For their invaluable contributions, the author would like to thank:

- Associate Professor Martin Braae, University of Cape Town; for the ideas, the many discussions and the occasional push in the right direction.
- Professor Edward J. Davison, University of Toronto; for the numerous papers on Switching Control techniques.
- Professor A. Stephen Morse, Yale University; for making available a copy of his unpublished work.
- Rob Bowyer, Cameron Haines, Karl Prince, and Warren Carew; for the company, the jokes and the friendship.
- Jenny; for the support and understanding - I admire your patience.

Synopsis

A servo motor is to be used to position the cutting arm in a hypothetical pattern generation application. The motor is controlled in closed-loop in order to track, with zero asymptotic error, a reference signal represented by either a sinusoidal, triangular, or square wave. In addition, the schedule of reference signal type changes is not known *a priori* and the controlled system must achieve asymptotic tracking without operator intervention.

As no *simple* single controller can satisfy these requirements for all setpoint types, a Hybrid Switching Control System is proposed which combines intuitive logic with standard control techniques. Under the guidance of a simple supervisor, the controller corresponding to each type of setpoint is switched in and out of the active feedback loop as required.

A simple Multi-layer Perceptron neural network was selected to identify the type of signal being tracked and hence initiate controller switching. This network performed very well even in the presence of measurement noise, and the hybrid system automatically tracked each of the three types of reference signal over a wide range of signal amplitude and frequency. However, the reconfiguration interval was quite long (although still acceptable in terms of the proposed application), and the size of the neural net structure had to be limited for the system to work in real-time.

Other methods of implementing switching control were investigated, but were found to be quite complex and non-intuitive. The switching criteria were mostly not well formulated and the resultant hybrid systems could not automatically track more than one type of setpoint without modification. However, these techniques do not rely on lengthy reference signal identification to initiate controller switching and thus could potentially improve the efficiency of the hypothetical pattern generation system. Consequently, it was recommended that these methods should be adapted to automatically track multiple set-

point types with a view towards subsequent implementation for the hypothetical application being considered here.

Contents

List of Figures

List of Tables

Glossary and List of Abbreviations

FSM Finite State Machine

HCS Hybrid Control System, Hybrid switching Control System

MLP Multi-Layer Perceptron

PC Personal Computer

RBF Radial Basis Function

RBT Real-time/Boolean Translator

SCL Supervisory Control Logic

SNR Signal-to-Noise Ratio

activation function	associated with each node in a neural network, it sets the node's output as a particular function of its input(s).
batch learning	neural network training method where the network weights are adapted only after the presentation of several training sets.
bounding function	an increasing function of some index k, it defines the limits on the values of some system variables. If the variables exceed these limits some specific action is taken (e.g. controller switching).
connection	a link between neural network nodes used to pass data from one node to another. Each connection has an adjustable value called a weight.
desired output set	a "correct" result included with each input pattern in a training or testing data set.
error surface	a multi-dimensional surface formed in the weight space from the (mean) squared error associated with the network outputs.
expected SNR	for the purposes of this work this is defined as 25.0dB, as per the derivation in Appendix E.2.
fitting interval	time interval between successive applications of the smoothing filter used to generate filtered signal samples. It is a multiple of the true sampling interval.

generalisation	a neural network's ability to respond correctly to data not used to train it.
global minimum	the unique point of least error during gradient descent, metaphorically the true "bottom" of the error surface.
gradient descent	a learning process that changes a neural network's weights to ideally follow the steepest path towards the global minimum of the error surface. However, at times the local minimum is found instead.
hidden layer	a layer of nodes not directly connected to a neural network's input or output.
hybrid control system	a control system which combines some form of high-level logic with standard controllers.
input layer	a layer of nodes that forms a (usually) passive conduit for data entering a neural network.
input space	the entire anticipated range of the data used to form the neural network input pattern.
layer	a set of nodes connected to common inputs, outputs, or both.
local minimum	a point of regionally low error during gradient descent along the error surface.
network training	a process during which a neural network passes through a data set repeatedly, changing the values of its weights to improve its performance.
neural network	an implementation of a learning algorithm derived from research about the brain. It typically contains layers of so-called artificial neurons composed of weights, connections, and nodes.
node	a single neuron-like element in a neural network. It typically has many inputs but only one output.
objective function	from a function minimisation aspect, this is the function of several variables whose minimum is to be found by, for example, a gradient descent search algorithm.
output layer	the node layer which produces the network's results.
pattern learning	neural network training method where the network weights are adapted after the presentation of each training set.

pattern recognition	identification of shapes, forms, or configurations by artificial means.
PC	for the purposes of this work (and unless otherwise stated) this refers to an IBM AT compatible running on a 80286, 16Mhz processor.
real-time	in a Control Engineering perspective refers to the requirement that the controller must interact with the physical system on the physical system's own time scale.
real-time/Boolean translator	converts continuous system data into a format suitable for analysis by the supervisory control logic.
reconfiguration interval	the time elapsed between a setpoint type change and the subsequent instant at which the appropriate controller takes over and achieves the control objectives.
root mean squared error	a measure of accuracy used by several neural networks, calculated by summing the square of the difference between actual and desired network outputs, dividing by the number of outputs, and taking the square root.
setpoint	reference signal, usually for a closed-loop system.
sigmoid, sigmoidal function	a continuous differentiable non-linearity used in neural networks as an activation function; defined as $f(x)=\frac{1}{1+e^{-x}}$
signal-to-noise ratio	defined as $20\cdot\log\left(\frac{\text{signal amplitude}}{\text{noise amplitude}}\right)$, in dB.
supervised learning	a neural network learning process requiring an input data set with a matching desired output set.
supervisory control logic	the topmost control layer in a Hybrid Control System; it performs high-level logical and analytical functions.
$t_{5\%}$	time to settle to within 5% of the final value.
unsupervised learning	a learning process that does not require matching desired output sets for the input patterns.
weight	an adjustable value associated with a connection between nodes in a neural network.

1. Introduction

Modern real-time digital control systems are an amalgamation of two distinct components: real-time control loops, and discrete-state logical decision-makers. Such a combination is usually met in practice, where often some form of logic is used to select between the real-time loops which provide control of the process dynamics. This logic is designed mostly on an *ad hoc* basis, without any serious consideration given to the resultant structure of the overall control system. Automatic control theory provides a foundation for the design of the real-time loops; it does not, however, contain decision-making logic as an integral component. Such logic has to be synthesised independently from the control loops, as and when it is required. This separate theoretical treatment of the two components can be a serious limitation in the design of modern control systems.

In their paper on automatic control design practice **[Bencze 1995]**, Bencze and Franklin state that "the addition of an analysis and decision-making capability to a control system greatly increases its flexibility". For example, individual controllers can be designed for specific plant operating points, and then switched in and out when the operating point changes. In addition, the switching rules can be made intuitive, without requiring a strict mathematical formulation. Not only is the design of such controllers much simpler than the design of a single controller capable of handling all likely operating points, but it would also result in a more robust and stable system **[Antsaklis 1995]**.

To provide this additional flexibility to the control system, the real-time control loops are surrounded by logical structures which decide when and how to modify these loops in the face of changing system parameters and control objectives. The resultant combined structure is known as a *Hybrid Control System (HCS)* - a controller that contains both real-time feedback loops and logical decision-making components. This structure has been found to be compatible with expert system-based intelligent control systems, and can indeed employ expert system techniques when necessary and effective **[Bencze 1995]**.

Bencze and Franklin go on to propose a design methodology for such systems, based on their interpretation of the hybrid controller's architecture, which is shown in Figure 1-1. This method, developed from control engineering practice, separates the overall design task into three smaller sub-tasks:

1. design of the real-time control loops;
2. synthesis of the control logic (or the *supervisory controller*) for the real-time loops;
3. design of the translators that allow the two to interact in a stable manner.

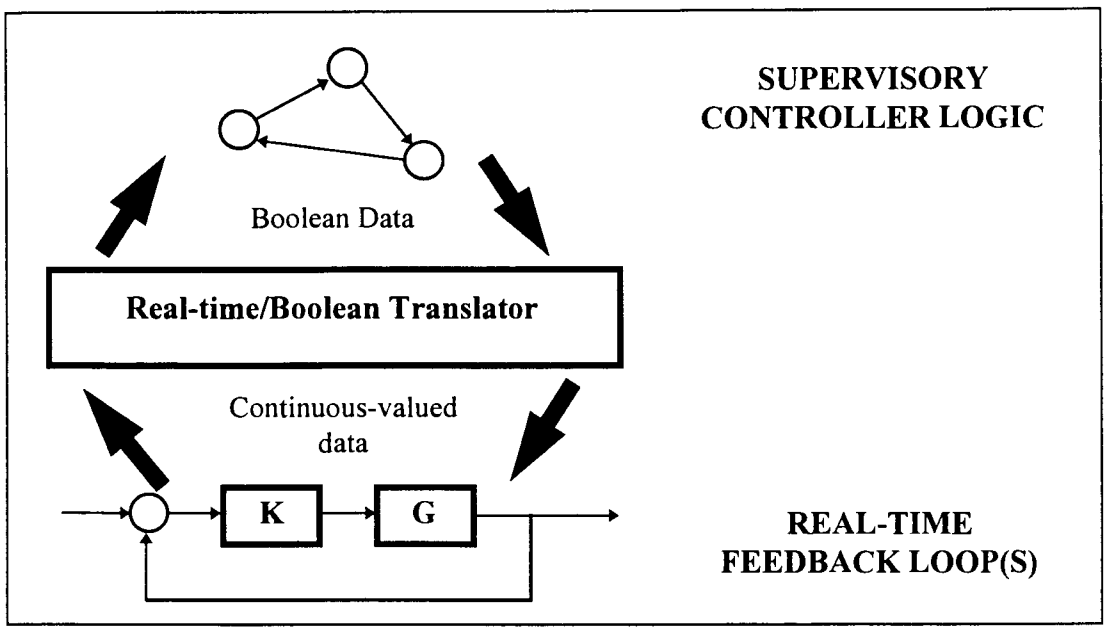


Figure 1-1. Hybrid Controller Architecture.

The real-time control loops are designed for each significant operating point of the plant **G**, with the emphasis being on achieving a given closed-loop performance criterion. The function of the supervisory controller is to combine the real-time loops so that smooth transfer of active plant control occurs as the operating points and conditions change. Finally, the translators are required to convert continuous-time plant and system data into discrete status *events* and vice versa, to enable the supervisory controller to de-

termine the plant's current operating point and activate the appropriate real-time feedback loop.

A hypothetical application which can be viewed in the context of hybrid control systems is automated pattern generation using an x-y table, as depicted in Figure 1-2.

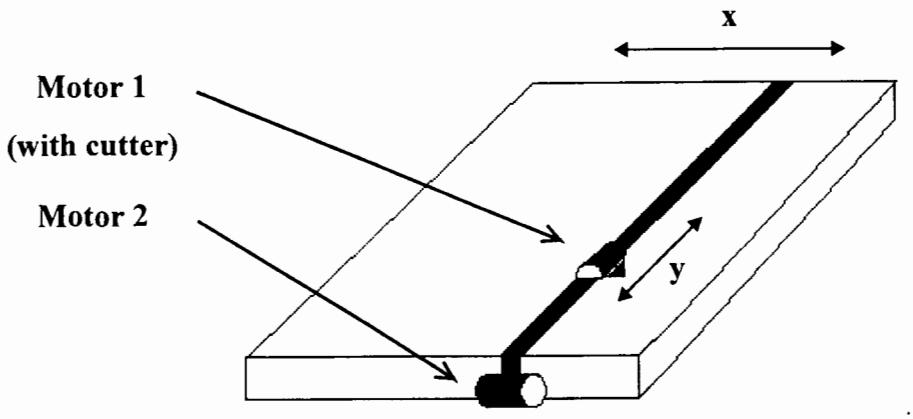


Figure 1-2. X-Y Table for pattern generation.

A pair of DC servo motors would be used to move the cutting tool on the table surface, with one motor for each of the two possible directions of motion and with no operator intervention. It has been noted that the required patterns could be obtained by using one of three different demand signals for the *velocity* of the controlling motor: namely a sine wave, a square wave, and a triangular wave, all of different amplitudes and frequencies. Furthermore, it is proposed to control the motors using a feedback system, with each motor's closed-loop controller implemented digitally on an available personal computer (PC). The above waveforms would then be the desired velocity reference signals for these control loops.

From a control engineering perspective, the objective becomes the tracking of these reference signals with zero asymptotic error and minimum control effort. However, no single controller can satisfy the performance requirements for all operating conditions. For example, a controller designed to track triangular waves will produce excessive overshoot and use more control effort if used with a square wave reference signal, even

though it may track with zero asymptotic error. Hence, it would be necessary to *reconfigure* the motor controller to reflect the current operational requirements. Furthermore, since the reference signals will be in no specific order, and the process is to be automated, this reconfiguration must be performed solely by the computer.

This fits in with the idea of HCS, in that separate *real-time control loops* would be required for each type of setpoint to be tracked, and some form of *supervisory control logic (SCL)* would have to be synthesised to perform smooth autonomous control transfer between these loops. The desired reference signal would be inferred by the *real-time/Boolean translator (RBT)*, and the information passed on to the supervisory controller.

The aim of this study was to evaluate the feasibility of applying the hybrid switching control system (or HCS for short) approach to the hypothetical automated pattern-generation problem outlined above. Therefore the objectives of this thesis were:

1. To investigate different methods of implementing the RBT;
2. To design and implement the real-time control loops for a single servo motor;
3. To synthesise a suitable SCL block;
4. To test the resultant HCS, both in simulation and on an actual servo motor in real-time;
5. To investigate alternative methods of implementing the HCS scheme and compare their performance to that of the system developed in this work.

This investigation was limited to a system consisting of a single servo motor, as that was felt to be representative enough of the hypothetical application. Should satisfactory performance be achieved with this system, it would be an easy task to extend the control strategy to the other motor and hence the entire pattern-generation system. Other limitations include the use of a relatively slow PC for real-time work, to evaluate the effect this would have on the designed HCS components. Furthermore, advanced control methods such as model reference adaptive control, predictive control, and sliding-mode

control were not considered here, as the emphasis was on simple, robust control laws that fitted easily into the HCS structure as outlined by Bencze and Franklin.

This report begins by describing and evaluating various methods of implementing the RBT. Due to its simplicity in this particular case, the SCL is also included in this discussion. Next, the design of the real-time control loops using continuous-time s-plane techniques is detailed. Simulation and real-time control results for the HCS so formed are also presented. An alternative switching control strategy is then evaluated for the same servo motor system. Finally, conclusions are drawn about the supervisory control system implemented and some recommendations are made with regard to further development of the HCS approach for this application.

2. Real-time/Boolean Translator and the Supervisory Control Logic

In the context of Bencze and Franklin's design methodology, these two components form the upper two layers of the hybrid control system (HCS) architecture. They play an important role in the overall performance of the system, particularly in the case of the proposed pattern generation application. This chapter describes and evaluates various methods of implementing these components from the perspective of this application.

2.1 Real-time/Boolean Translator

The RBT forms the interface between the real-time control loops and the supervisory control logic (SCL), and is custom-tailored to the specific needs and goals of the overall control system. It's complexity is dictated by, among other things, the degree of automation required of the overall control system, the accessibility of the necessary data, and the amount of information to be processed.

2.1.1 Function

The RBT must provide data to enable the supervisory controller to determine the current state of the system and hence infer controller reconfiguration requirements, if any. It must further translate the SCL's commands into control loop-level actions, such as selecting the correct controller equations for computing the control signal u.

In the pattern generation application specifically, controllers are to be switched based on the type of waveform used as the reference signal. No other information is available which influences the switching process. Hence, the RBT must discriminate between three types of waveforms with unknown periods and amplitudes. Furthermore, this must be done without any assistance from the operator, in a completely *stand-alone* fashion.

Perhaps the most important part of this *digital waveform recognition* process is the identification of unique features of the waveforms, which could then be used to discriminate between them. These features must be chosen and grouped such that they form clearly separated regions in the *feature space* of the waveforms, as demonstrated below.

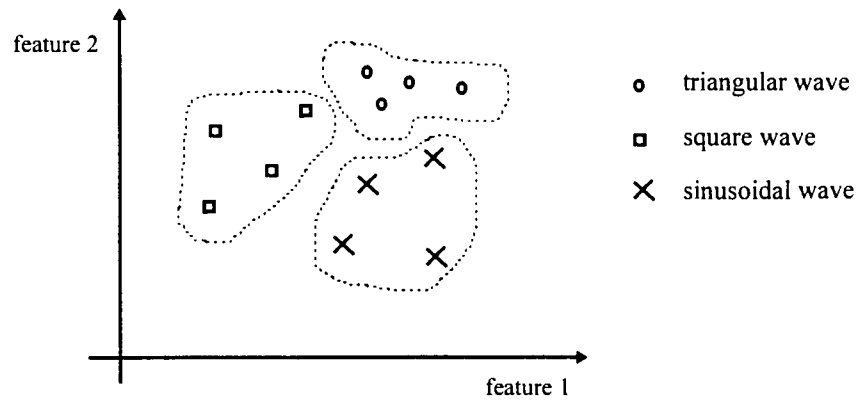


Figure 2-1. Sample waveform feature space.

If a given [*feature1, feature2*] vector falls within a certain pre-identified region, it can be classified as representing that waveform type. However, should the chosen features not be unique enough, and the regions overlap, no conclusion can be reached about the type of waveform being analysed. The problem is further complicated by the expected presence of measurement and quantisation noise[1], and the structural similarity of the sinusoidal and the triangular waveforms. It should be noted, though, that the waveform features can be identified either *a priori* by the designer, which means the RBT has only to extract and analyse them, or else in real-time by the RBT itself, which further increases its complexity.

An important aspect of the RBT design is that it should be able to perform the waveform analysis on-line and in real-time (or faster). This is motivated by the need to minimise the "down time" of the pattern generator, where it waits for the control objectives to be satisfied before generating the next pattern.

[1] As a result, the expected minimum *Signal-to-Noise Ratio (SNR)* is 25.0dB, as per the calculations in §E.2. All subsequent mention of the *expected SNR* in this work will refer to this figure.

The functions of the RBT can therefore be summarised as follows:

1. identify and extract unique features of the waveforms, unless this forms part of *a priori* design knowledge;
2. quantify these features; analysis of the resultant location in the feature space will determine the type of waveform present in the reference signal;
3. pass the result of the analysis on to the SCL;
4. perform steps 1-3 on-line and in real-time, in order to minimise the *reconfiguration interval* (the time elapsed between setpoint change and the instant when the appropriate controller takes over and achieves the control objectives);

2.1.2 Implementation

This problem falls into the *pattern recognition* category, with the possible values of the waveform features forming the patterns to be recognised, or classified. Several approaches to waveform recognition are proposed in literature **[Chen 1982]**, with most of them requiring extensive pre-processing of waveform samples. These methods are usually based on statistical techniques which rely heavily on probability theory and require a large number of signal samples spanning an entire waveform period. Inevitably, these methods also require significant computing power.

To satisfy the requirements of the hypothetical application considered in this study, the reference signal must be identified in as small a fraction of the waveform period as possible. Furthermore, this must be done without overloading the available computing resources, so that the entire procedure can be performed on-line, in real-time. For these reasons, classical statistical techniques will not be considered in the following discussion of waveform recognition methods. Instead, intuitive methods of feature extraction and waveform identification will be evaluated.

It should be noted at this stage that, due to the inherent symmetry properties of the waveforms being considered here, most of their features are also contained in each *quarter of the period.* Therefore, by analysing only that section of the signal, it should be possible

to classify it, and emphasis will be placed on the ability of each evaluated method to make use of this fact.

2.1.2.1 The Gradient Method

This is an intuitive method, based on the analysis of the first and second derivatives of the signal. In the case of the waveforms under consideration, these two features could be used to classify them. However, from a strictly mathematical perspective the derivatives of the square wave do not exist, as the original waveform is discontinuous. The potential conflict with theory can be averted, though, by considering only continuous sub-sections of the waveform. In this way, derivatives can be taken without violating this principle.

Waveform	1st derivative	2nd derivative
sinusoidal	sinusoidal	sinusoidal
square	0	0
triangular	non-zero constant	0

Table 2-1. Waveform derivatives.

From Table 2-1, it can be seen that the type of waveform making up the signal could be inferred according to the following decision tree:

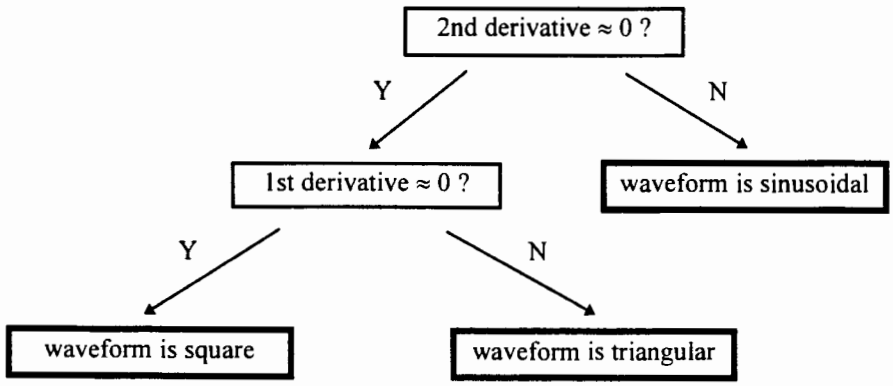


Figure 2-2. Decision tree for gradient-based classification.

During evaluation of this method, samples of the original signal were taken over the continuous sections of each waveform. The gradient was calculated between each pair of consecutive samples, to give a "moving window" of 50 samples of the first derivative $\delta r/\delta t$.

$$\frac{\partial r}{\partial t} \cong \frac{r(t+h) - r(t)}{h}$$

These were averaged to obtain the mean, and the gradient was again taken between each consecutive pair of $\delta r/\delta t$ values to obtain another moving window containing 50 samples of the second derivative, $\delta^2 r/\delta t^2$.

$$\frac{\partial^2 r}{\partial t^2} \cong \frac{\frac{\partial r}{\partial t}(t+h) - \frac{\partial r}{\partial t}(t)}{h}$$

The mean of these samples was also found, and the two averages were checked to see if they fell within pre-defined bounds of zero.

2.1.2.1.1 Results

With ideal waveforms not contaminated by signal noise or other distortions, this method proved very successful, as can be seen from Figure 2-3 on page 11. The two features extracted from the waveforms were sufficiently unique to enable classification. However, this was not the case when noise[2] was added to the signal (the noise levels used were chosen so as to give *signal-to-noise ratios (SNRs)* consistent with the *expected SNR* of 25.0dB). The process of taking the derivatives further compounded the noise problem, to the extent that it proved impossible to conclusively discriminate between the three waveform types.

In order to eliminate the adverse effects of the signal noise, a Savitzky-Golay smoothing filter **[NumRec 1992]** (as described in Appendix C) was applied to both the original sig-

[2] This term refers to *additive zero-mean White noise* when used in a simulation context throughout this work.

nal and its derivatives. However, as can be seen from Figure 2-4 below, even this measure failed to resolve the problem. The features extracted were no longer unique, and the waveforms were often misclassified. Furthermore, the threshold limits used for the zero test described above had to be adjusted with every change in noise level. No straightforward formula for setting these limits could be found.

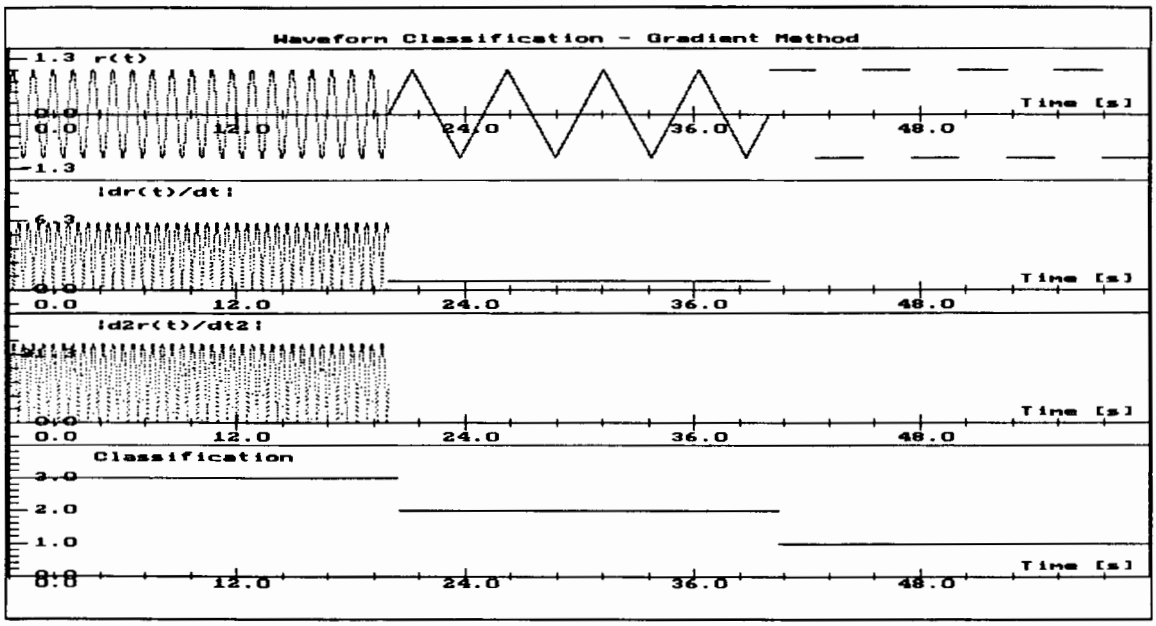


Figure 2-3. Gradient method: classification results with no signal noise.

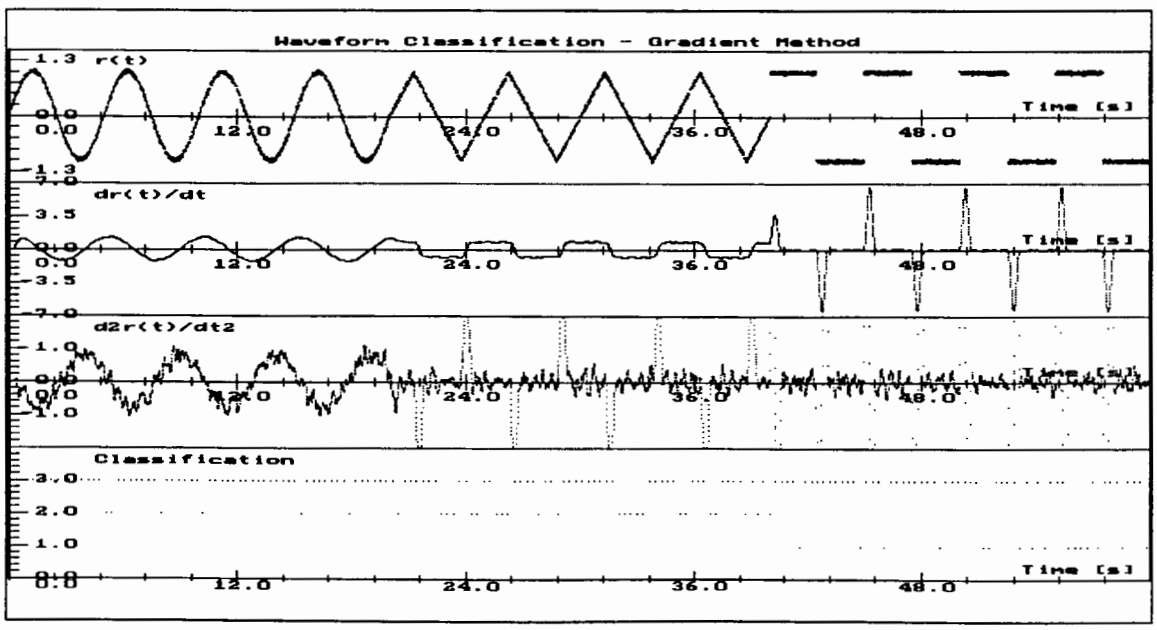


Figure 2-4. Gradient method: classification results with a SNR of 25.0dB.

2.1.2.2 Fourier Transform Techniques

Each of the three waveforms under consideration has (theoretically) a different Fourier Transform representation which can be obtained by taking the Discrete Fourier Transform (DFT) of the sampled signal. Should these representations be sufficiently unique even in the presence of signal noise, some features could be extracted from them for waveform classification purposes.

However, a complete Fourier transform representation is obtained only by considering an entire waveform period **[Oppenheim 1989] [Morrison 1991]**, and the Fourier coefficients so obtained still have to be processed further to identify the signal they represent. Furthermore, tests of the DFT of the respective waveforms show that the results obtained for the sine and triangular waves were too similar, even in the presence of relatively little additive signal noise (see Figure 2-5 below). Subsequent increases in noise levels further enhanced this similarity, and no useful features could be extracted from the Fourier transform of the signals.

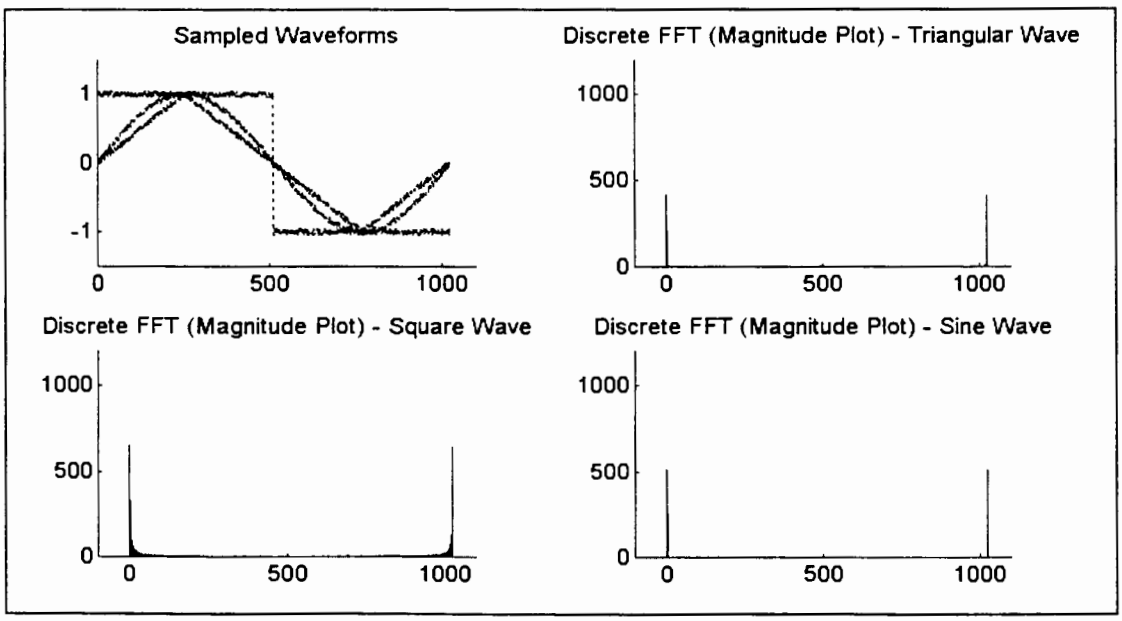


Figure 2-5. DFTs of waveforms with a SNR of 20.0dB.

2.1.2.3 Simple Statistical Methods

For a feature set to have the largest discriminatory power it should be constructed from all the available features of the waveforms or signals under investigation **[Chen 1982]**. Simple statistical analysis of these waveforms results in several characteristics which can be used to form just such a feature set, providing they yield values which are sufficiently unique for each type of signal. Again, only a quarter-period section of each waveform is used for the analysis, for reasons already stated on page 8.

The following statistical characteristics[3] were investigated as possible waveform features:

1. Means
2. Medians
3. Standard Deviations
4. Cumulative Sums and Products

2.1.2.3.1 Results

In the case of features 3 and 4, the values obtained failed to uniquely classify the three waveforms investigated, and so were discarded as possible features. However, as can be seen from Figure 2-6(a) on page 15, the mean of the sampled waveform sections gave values which were separable for SNRs as low as 15.0dB. Similarly, the median values of the waveform sections began to overlap only at SNRs below 20.0dB (Figure 2-6(b)). The plots were generated by computing 50 values each of the mean and of the median, for SNRs ranging from approximately 0.5dB to infinity (no noise). The means and medians so obtained are amplitude dependent, but this is a very minor problem as the signal samples can be normalised to unity prior to the evaluation of the two features.

[3] The formulae used to evaluate these characteristics are given in §E.3.

The resultant feature space is shown in Figure 2-6(c), from which it can be seen that the regions formed overlap only at SNRs of approximately 20.0dB and less. As the actual SNRs are rarely expected to fall below 20.0dB during sampling, these features could be used to form a feature space with relatively good discriminatory powers.

Despite this, it is felt that this feature set may still not prove to be robust enough in its classification abilities. The triangular and sinusoidal waveform regions are not far enough apart to achieve reliable discrimination, especially in the presence of signal noise, and the feature set is constructed from only two features of the waveforms. Several linear combinations of these features were tried as possible ways of obtaining better spaced regions in the feature space, but without success.

If as many features as possible were included in the feature set, and different non-linear combinations of them were formed, it is felt that this would produce a feature space with clearly separated regions which could be used for robust signal classification. As so many possible features exist, and infinite non-linear combinations of them can be formed, a quick method of generating such a set is required.

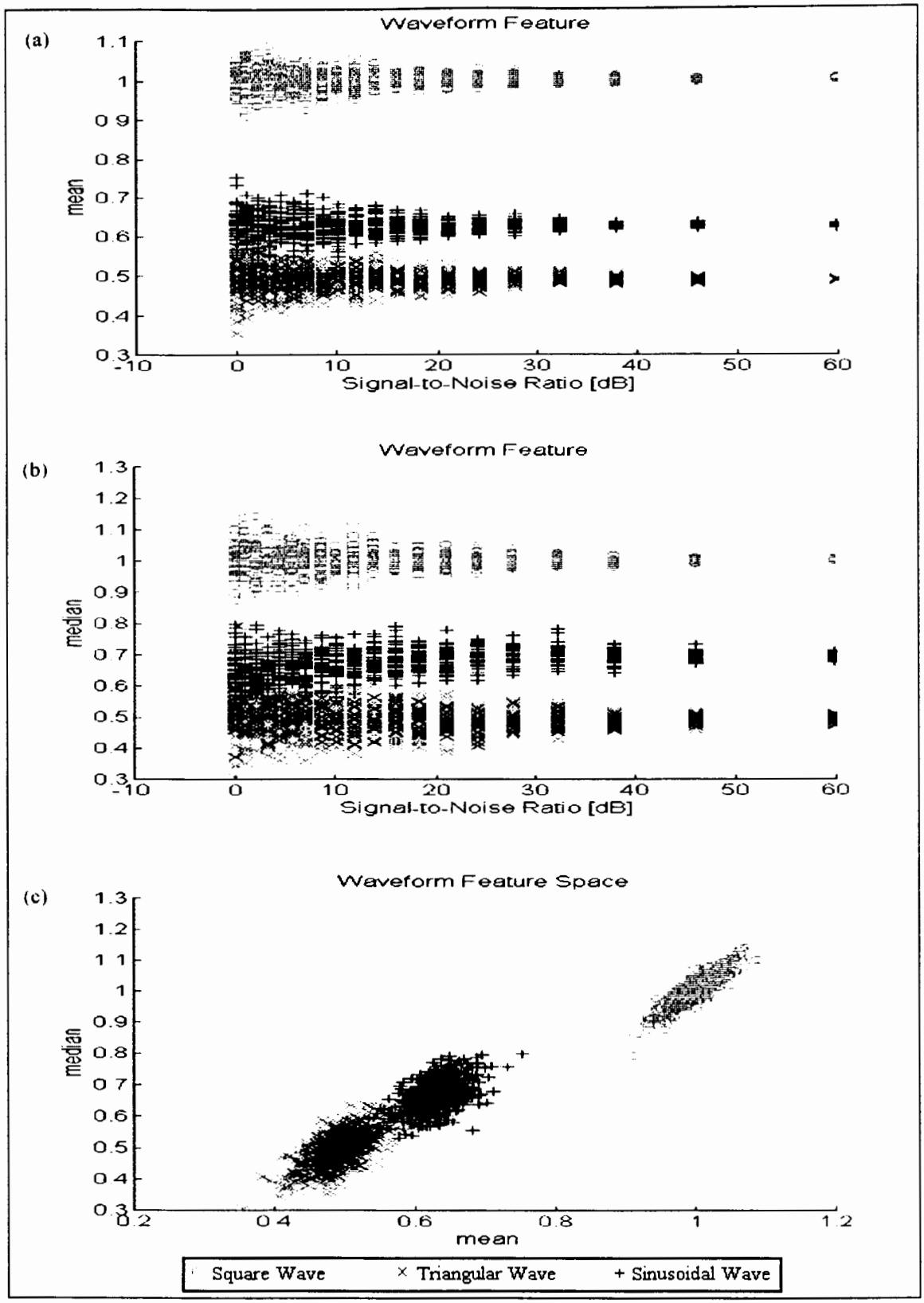


Figure 2-6. Use of medians and means as features.

2.1.2.4 The Neural Network Approach

Neural Networks are the result of an attempt to model mathematically the learning and reasoning abilities of the human brain, which together combine to give us the quality known as *intelligence*.

The field of neural network research is extremely wide, with many of its proponents concentrating on different methods of achieving the goal of an *artificially intelligent system*. As a result, there exist a large variety of network[4] structures and learning algorithms which approximate the behaviour of the human brain to different extents. Each of these are suited to different applications, and few criteria exist to aid the designer applying neural networks to a given problem in deciding which structure and algorithm best fits the requirements.

In control engineering, the appeal of neural networks stems from their proven ability to model any non-linear function, given only the inputs and corresponding outputs of that function **[Trossbach 1994]**. This is extremely useful for obtaining process models, where the underlying mathematical formulation of the model is not required or is intractable in the first place. Similarly, neural networks can be used to model particular non-linear control laws which may be difficult to derive using conventional theory and methods. In general, though, neural networks do not replace standard control techniques, but instead augment and enhance them.

Over the last few years, the field of *intelligent control* has received substantial interest from the control community **[Antsaklis 1994]**, mainly due to the realisation that it can address control problems that cannot be formulated in the language of conventional control. In particular, neural networks have been used in a *pattern recognition and classification* role, forming part of a supervisory controller with built-in reasoning and logic capabilities. Such *hybrid* structures have been successfully used in work on Control Law

[4] The terms *neural network, network,* and *net* will be used interchangeably in this discussion.

Scheduling or Reconfiguration, involving the monitoring of system states or operating conditions **[Antsaklis 1991] [Garcia 1991]**, and failure detection **[Narendra 1992]**.

In the waveform classification problem being considered here, a neural network used as a pattern recogniser would be able to distinguish between the different waveforms **[Narożny 1995].** By virtue of its inherent non-linearity, the neural network would extract the required waveform features in a totally transparent manner. In effect, it would learn to *model* the feature extraction and identification process, classifying the waveform when supplied with the raw waveform sample data. Through repetitive training the net would learn to select those waveform features (and non-linear combinations thereof) which form the most discriminatory feature set. Such a set would be a vast improvement on one produced manually, and should provide robust classification. Furthermore, the approximation and generalisation properties of neural nets should enable the network to adequately handle noisy inputs.

2.1.2.4.1 Selection of Neural Network Structure

The limitations imposed by the computing resources available for this hypothetical application (a low-end desktop PC) meant that the chosen network structure (and corresponding training algorithm) had to fulfil several criteria. Furthermore, the envisioned use of the network with noisy data in real-time added even more constraints, resulting in a fairly detailed specification of the requirements.

The neural net representation had to:

1. be suitable for pattern classification applications;
2. be able to handle noisy inputs well;
3. be small enough not to require excessive storage space and processing power;
4. have a training algorithm suitable for pattern learning, and which was easy to implement. This algorithm should be fairly fast for the selected network size and should ensure good convergence leading to a high rate of correct classifications.

Several representations were considered, based on their recent evaluation and use in numerous applications **[Antsaklis 1993] [Trossbach 1994]**. The most suitable candidates are listed below, with more details of each type of network being given in Appendix A:

1. Kohonen Feature Maps;
2. Cerebellar Model Articulation Controller;
3. Multi-layer Perceptrons (MLPs);
4. Radial Basis Function (RBF) networks.

Of these, the MLP and the RBF networks, together with their associated training algorithms, were selected as the most promising implementations. The other network structures considered were discarded due to a combination of their large size, the complexity of their training algorithms, and their somewhat average performance in classification problems **[Lippmann 1987] [Antsaklis 1993]**.

Further investigation of the RBF nets indicated that training of these networks would be a complex, trial-and-error process which did not guarantee good classification results even after convergence during learning **[Koivo 1994]**. The complexity of training was caused by the additional parameters which needed to be set for the hidden nodes in the RBF network, namely the centres and widths of the Gaussian activation functions associated with those nodes. Admittedly, these parameters could be iteratively set by the training procedure itself, but Koivo discovered that a network trained in this fashion failed to classify test data properly.

The MLP, on the other hand, offered a structure with a proven record in pattern classification applications **[Antsaklis 1993]**, which was relatively easy to implement and train. It required less storage space than the corresponding RBF network **[Trossbach 1994]**, and handled noisy inputs well. However, the associated Error Backpropagation algorithm also involved *ad hoc* setting of some learning parameters, a task which was not often straightforward. In addition, the choice of the number of nodes in the hidden layers

is another MLP parameter which must be set by trial-and-error, and which can be quite critical to the network's generalisation (and hence classification) performance.

Despite this, Koivo showed that it was a fairly easy task to construct and successfully train a MLP network, achieving good convergence in a reasonable number of iterations through the training set. Furthermore, the resultant network produced better classification than a RBF net. For these reasons, the Multilayer Perceptron was selected for the waveform identification application being considered here.

2.1.2.4.2 Pre-processing of Network Input Data

The raw waveform data has to be sampled and presented in a specific way to satisfy several system and neural network-level requirements.

Firstly, as the classification method needs to be independent of input signal amplitude, the signal samples will be normalised to the range [0.0–1.0]. Secondly, the waveform samples must be as noise-free as possible, in order to increase the likelihood of correct classification. Hence, the sample data is filtered using the Savitzky-Golay best-fit filter prior to presentation to the neural network.

Finally, the finite number of signal samples used for classification purposes must cover exactly a *quarter* of the waveform period, irrespective of the signal frequency. This will serve to reduce the reconfiguration interval, as was described earlier, and also makes the classification method frequency independent. The **first** quarter of the waveform period was chosen as the signal section to be used for classification purposes, as depicted in Figure 2-7 below.

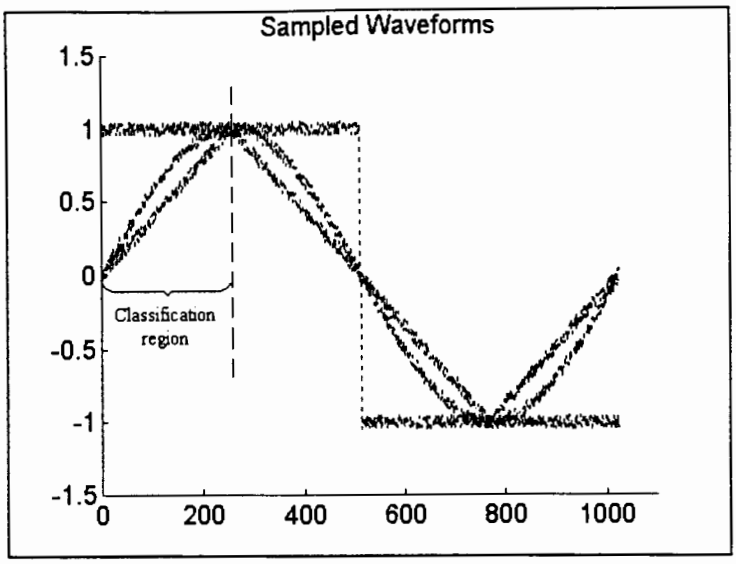


Figure 2-7. The classification region of the waveforms.

The problem of frequency dependence arises in the following situation: if the number of samples and the sampling interval are kept the same for each sampling run, but the frequency of the sampled signal is changed between runs, the waveform section represented by those samples will be different for each frequency. An example of this is given in Figure 2-8 below, where the period of the original waveform is changed and the waveform is re-sampled using the original sampling interval.

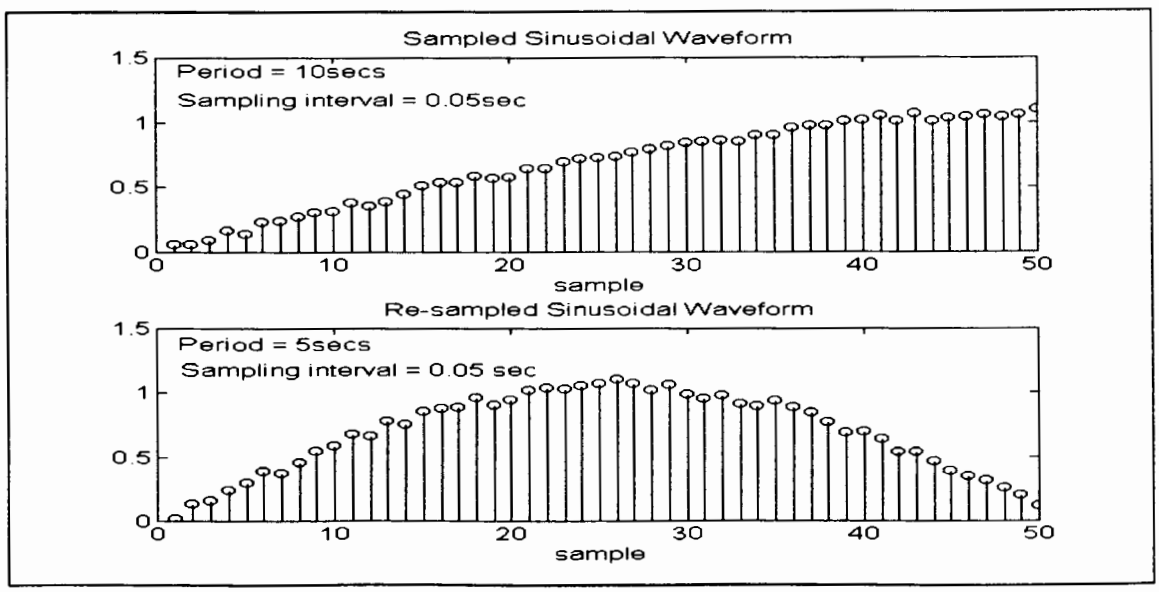


Figure 2-8. Waveform sampling with fixed sampling interval.

However, to minimise network size and training time the same waveform *"shape"* (as obtained by plotting consecutive signal samples) must be presented to the neural network for classification, irrespective of the waveform period. If this is not done, the network will have to be trained on an infinite number of patterns to cover all the possible waveform sections and sub-sections which might be generated by sampling signals of different frequencies. Clearly, such a procedure would be extremely tedious and inefficient.

There exists, though, a linear relationship between the waveform period and the sampling interval, given a fixed number of samples. If, for example, the waveform period is halved, then halving the sampling interval results in the consecutive samples forming the original waveform shape. This is illustrated in Figure 2-9 below, and is true for each of the three waveform types considered here.

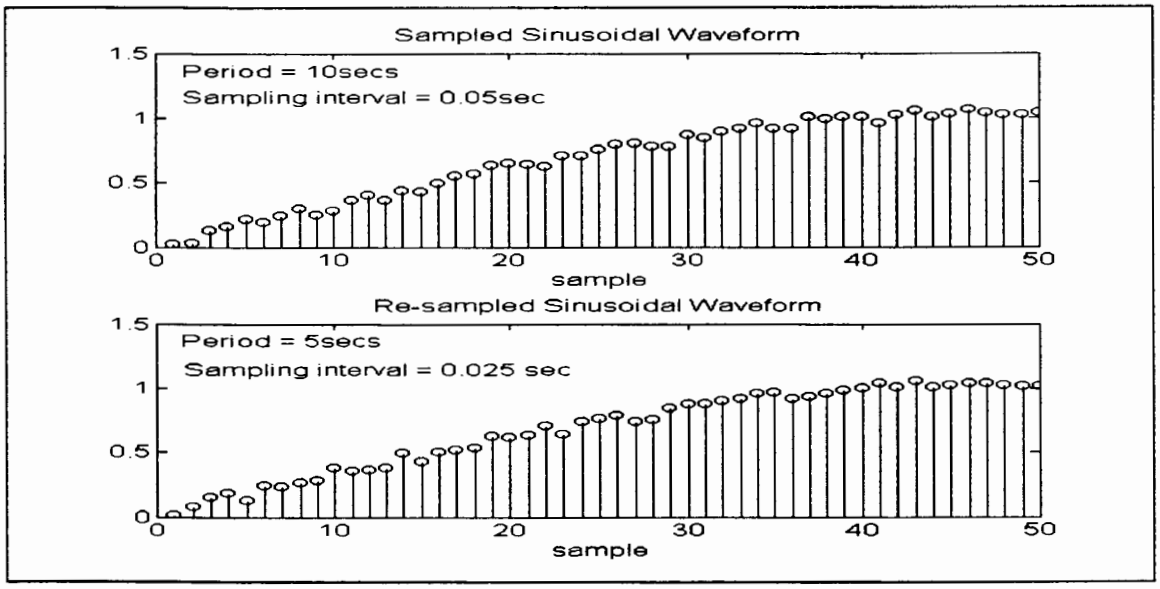


Figure 2-9. Waveform sampling with variable sampling interval.

For our purposes, the number of samples required for classification was set to 50, in order to minimise network dimensions while still obtaining sufficient signal samples for feature extraction. The signal was sampled every 0.01 second, but the *fitting interval*[5]

[5] This is the interval between successive fitted (or filtered) signal samples, which form the input pattern for classification purposes. It is a multiple of the true sampling interval of 0.01 second.

was varied according to the following formulae to give exactly 50 filtered samples which covered the entire quarter-period of the waveform:

sampling interval $T_s = 0.01$ seconds

fitting interval $f\_dt = \dfrac{period}{4 \cdot number\_of\_samples}$ э $f\_dt \geq 10 \cdot T_s$ for a good fit

where:

period	is to be measured
number_of_samples	is 50

These samples formed a *pattern* to be fed into the neural network, and the waveform period was measured from the signal itself using threshold detection and two consecutive threshold crossings. The procedure for obtaining the pattern samples is illustrated graphically in Figure 2-10 below, and the algorithm for variable-interval sampling is outlined in the flow-chart of Figure 2-11 on the following page.

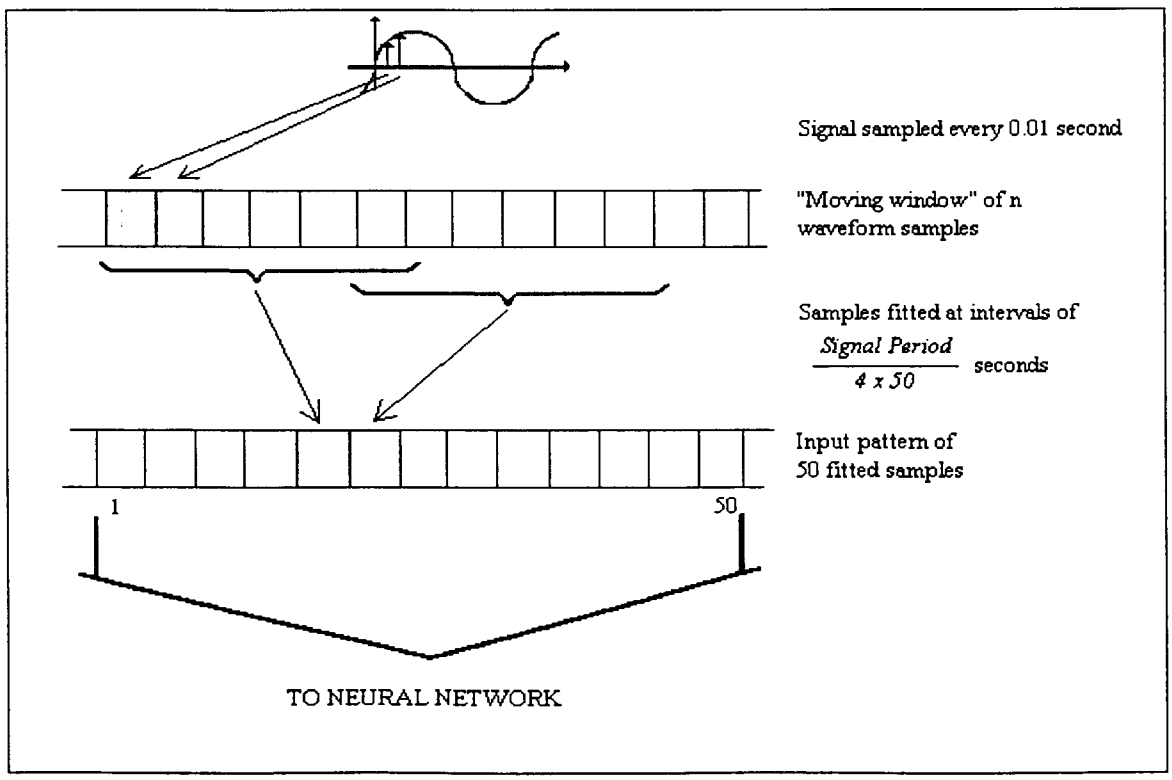


Figure 2-10. Input pattern generation.

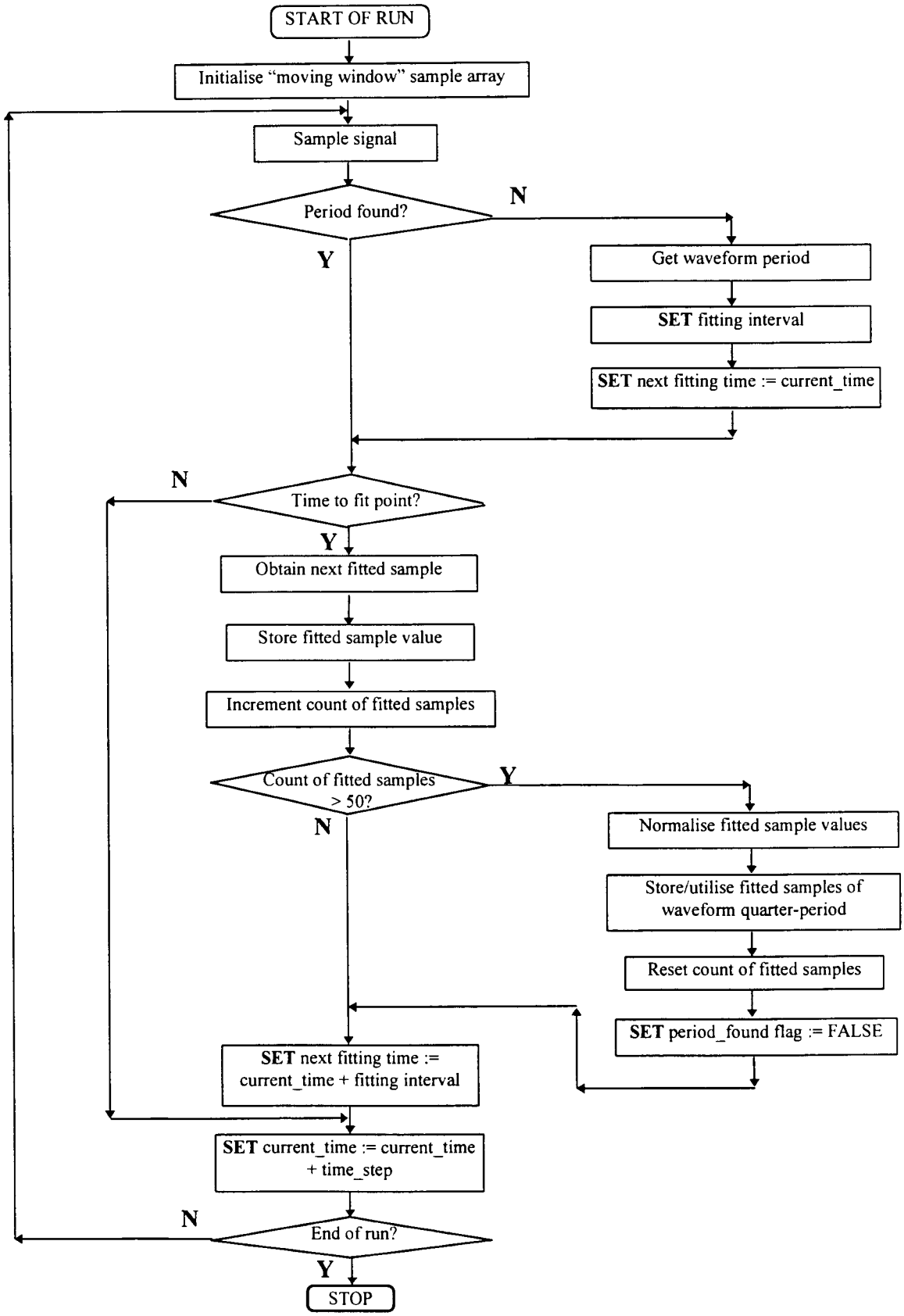


Figure 2-11. Outline flowchart of variable-interval sampling algorithm.

By sampling the waveform using a fixed interval and requiring 50 fitted samples per quarter-period, the period of the reference signal used with this system is constrained to fall within certain limits. Simple application of the anti-aliasing theorem (also known as the sampling theorem) leads to the following:

sample interval $T_S = 0.01$ second

$\Rightarrow$ sampling frequency $f_S = 100$ Hz

For anti-aliasing: $2f_m < f_S$ where f_m is the maximum signal frequency

$\Rightarrow$ $f_m < 50$ Hz

$\Rightarrow$ Minimum signal period = $T_{min} = 0.02$ second

However, from fitting considerations at least 10 signal samples are needed to obtain a good fitted sample (see the discussion in Appendix C), and 50 of these fitted samples are required per quarter-period. Hence:

$$\begin{aligned} \tfrac{1}{4} \cdot T_{\min} &= 50 \cdot 10 \cdot T_s \\ &= 5 \\ \therefore T_{\min} &= 20 \text{ seconds} \end{aligned}$$

Increasing this by 25% to give a reasonable margin of error, the minimum reference signal period becomes:

$$\boldsymbol{T_{min} = 25 \text{ seconds}}$$

The corresponding maximum frequency is 0.04 Hz, and this is used in the subsequent design of the feedback controllers (described in Chapter 3).

To summarise, the following conventions are assumed in relation to the waveform data used by the neural network classifier:

- the period of the reference signal must be *at least* 25 seconds;
- signal samples are taken at fixed intervals of 0.01 second;
- the samples are amplitude-normalised to unity;

- 50 filtered samples are taken per quarter-period to obtain the input pattern;
- the shape of the waveform represented by the input pattern is not dependent on the frequency of the original sampled signal.

2.1.2.4.3 Network Training and Network Size Optimisation

The actual size of the network, in terms of the number of nodes per layer, could only be determined after the selection and testing of a training algorithm. However, for the purpose of continuity, the pre-determined parts of the network structure are described here, with the rest given as the section develops.

Number of Input Nodes

Each pattern which is presented to the network consists of 50 samples of the waveform. Hence the number of input nodes is also set to 50.

Number of Output Nodes

The network is required to classify the waveform into one of three categories, hence three binary output nodes are needed. These would correspond to three Boolean flags with a 1 indicating a TRUE state, as required by the SCL to which these outputs are fed. In the ideal case, the desired output of the network was arbitrarily chosen to be:

	Output Node 1	*Output Node 2*	*Output Node 3*
For a sine wave	1	0	0
For a triangular wave	0	1	0
For a square wave	0	0	1

Table 2-2. Desired output patterns.

In addition, since the network outputs will almost never be perfect binary numbers, a *decision threshold* was used to decide if a particular value was a binary 1. As very good classification was required, this threshold was set at 0.1.

$$\Rightarrow \quad 1.0 \pm 0.1 \equiv 1$$

Activation Functions

The input layer is merely required to pass the inputs on to the hidden layer for classification, while the output layer simply sums the outputs of the hidden layer. In the hidden layer, a non-linearity is necessary to give the net its approximation properties, and either the *hyperbolic tangent (tanh)* or the *sigmoid* can be used. It was found during training that the *tanh* function caused convergence problems, hence the following activation functions were used:

Input Layer - linear
Hidden Layer - sigmoidal non-linearity
Output Layer - linear

Number of Hidden Layers

There is much discussion in literature about the ideal number of hidden layers that a MLP should have, with Lippmann stating that no more than two need to be used to model an arbitrarily complex non-linear mapping **[Lippmann 1987]**. Indeed, Niesler found this structure to be adequate for his purposes **[Niesler 1993]**. However, according to Trossbach there is in fact no difference between the optimal performance of single and double hidden layer networks, although on average, single layer networks are better classifiers **[Trossbach 1994]**. He goes on to state that double hidden layer nets are more susceptible to the local minima problem during training, and this further motivates the choice of a single hidden layer implementation of the MLP for this work.

Training Algorithm

In the case of the MLP, the most popular training methods by far are versions of the Backpropagation algorithm developed by Rumelhart et al. No definitive rule exists for the choice of a training algorithm, with the selection usually being dictated by performance during evaluation tests using data specific to the application, as was done by Niesler **[Niesler 1993]**. In his case, he found that the Resilient Propagation algorithm gave him the best performance with a MLP, although several other techniques were also promising. In general, the choice of algorithm is very subjective, and depends to a large extent on the requirements of the intended application.

Several authors have successfully used Backpropagation training for classification problems **[Lippmann 1987] [Koivo 1994]**, and this motivated its selection here. However, during the course of testing it was found that the training time and convergence rate could be significantly improved by modifying the training parameters during a run, resulting in a two-stage Backpropagation. This effectively broke the training up into a coarse, quick phase, followed by a fine, slow one. The original Backpropagation algorithm of Rumelhart and the modified version developed in this work are both described in detail in Appendix B.

The following thresholds were used with the modified algorithm to achieve the two-stage training:

1. First threshold: Modify learning parameters when **rms. error**[6] ϕ <= **0.001**;
2. Second threshold: Terminate training when **rms. error** ϕ <= **0.0001**.

These threshold values were selected so that successful training would result in a robust and reliable neural network. Such a network should produce an output pattern whose elements have a high probability of falling within the decision threshold of the desired ones.

Training Data

The neural network classifier was required to correctly identify a signal from samples taken over a quarter-period, even in the presence of noise. In addition, for the training to be comprehensive the training data should ideally span the entire expected input space. As such, it was felt that the neural network would be best trained on patterns consisting of *noise-corrupted* data, as opposed to ones representing perfect waveforms. The random nature of the zero-mean White noise added to the training patterns would cause them to excite a wider region of the input space, improving the trained network's generalisation

[6] The *rms. error* referred to here is the Root Mean Square of the difference between the actual and the desired network outputs after the presentation of an input pattern.

ability. Results to this effect have been reported in experiments using the Backpropagation algorithm, and Matsuoka proves mathematically that the addition of random noise to the training inputs enhances the MLP's generalisation and noise-handling capabilities **[Matsuoka 1992]**

Hence, the training data consisted of sets of 50 signal samples taken over a quarter-period of the waveforms and corrupted by band-limited, zero-mean, additive White noise. Band-limiting was achieved by filtering using the Savitzky-Golay smoothing filter, and the level of noise was selected so as to give a SNR of 20.0dB. This figure was below the 25.0dB expected during sampling, but would result in the excitation of a larger region of the input space than the actual input data would occupy. This in turn would ensure that as much of the expected input space was used during training as possible. Larger noise levels were not used as Matsuoka implies that a network trained on such data could become *too* general, necessitating a bigger structure to achieve the same classification performance.

One hundred such sets were generated for each of the three waveforms, and were then repeatedly used in random order to train the network.

Number of Hidden Nodes

The number of hidden nodes was initially set to an upper bound of 75, as this represented (for the given input and output layer sizes) the largest network permitted by the stack size limit of 64 Kbytes, as set by the simulation software[7]. In addition, several other (smaller) hidden layer sizes were tried in order to determine the smallest network structure which attained not only quick convergence of the training algorithm, but also good classification performance.

The results of these trials are summarised in the following table. The network was trained several times using each set of learning parameters, to determine the consistency of the convergence (if any) observed and eliminate unstable results. Only those cases in

[7] Borland Turbo Pascal for DOS, version 7.

which the number of iterations required for convergence was fairly consistent were subjected to further classification tests. These cases are marked by an asterisk (*) in the table.

Number of hidden nodes	*Learning rate* η		*Momentum term* α		*Iterations to convergence*	*Comment*
	INITIAL	*FINAL*	*INITIAL*	*FINAL*		
75*	0.20	0.05	0.35	0.10	26,000	consistent
40	0.95	0.30	0.60	0.20	---	training stuck
40*	0.30	0.10	0.50	0.10	7,500	fairly consistent
40*	0.25	0.30	0.50	0.15	14,200	fairly consistent
20*	0.30	0.10	0.50	0.10	18,000	fairly consistent
20	0.95	0.30	0.60	0.20	---	training stuck
10	0.25	0.40	0.50	0.15	11,400	very inconsistent
10	0.95	0.30	0.60	0.20	---	training stuck

Table 2-3. Network training results with different hidden layer sizes.

From the table, it can be seen that 10 hidden nodes were not enough to successfully approximate the underlying non-linear function. With large training parameters, the process stuck in a local minimum, while smaller parameter values did not produce consistent convergence. Increasing the hidden layer size to 20 and then 40 nodes resulted in more stable learning, although large learning parameter values again led to the local minimum problem. Finally, the original 75 hidden nodes produced stable convergence, with the only drawback being the resultant large network size.

Each of the marked networks in the table were then compared using a classification test, as good convergence does not necessarily imply good classification ability. The test consisted of repeated waveform identification attempts by the trained networks, using new

sets of waveform sample data (i.e. not the training sets) corrupted by increasing amounts of signal noise. The signal noise levels were varied so as to give SNRs ranging from 6.0dB to infinity (no noise), and several runs were carried out for each noise level setting. These were then averaged, giving the results detailed in Appendix E and summarised below.

In the following discussion, *Identification Errors* are cases where the network misclassified the presented waveform as being of another type, and *Indecisions* are cases where the net could not decide conclusively on the type of the presented waveform.

1. MLP with 75 hidden nodes

This network displayed almost perfect behaviour at SNRs above 20.0dB, with no indecisions or identification errors. Performance deteriorated progressively at lower SNRs, although the square wave was always recognised, irrespective of the SNR. As a SNR in excess of 20.0dB represented the noise levels in which the net was expected to operate (the *region of interest*), this particular structure would be reliable enough for the identification task.

2. MLP with 40 hidden nodes, small α_{final} and η_{final}

Generally bad classification results were produced, even in the region of interest. The sinusoidal wave especially was not identified correctly, with the network unable to decide on the type of waveform presented. Hence, this network could not be considered reliable, and as such could not be used for waveform identification.

3. MLP with 40 hidden nodes, large α_{final} and η_{final}

The performance obtained here was very similar to that of the 75 hidden node net, with no classification errors in the region of interest. Outside this region, the percentage of successful identifications of the sinusoidal and triangular waves decreased with decreasing SNR, as expected. The square wave, though, was correctly recognised at all noise levels. In general, the results showed that this network was very reliable and robust, and therefore could be used for waveform classification.

4. MLP with 20 hidden nodes

In terms of reliability and classification in the region of interest, almost identical performance to cases (1) and (3) above was obtained. Hence, this structure too could be used to identify the waveforms.

Three possible hidden layer sizes were therefore identified as representing network structures which could be trained to produce extremely good classification performance: 75 hidden nodes, 40 hidden nodes, and 20 hidden nodes. A smaller network, though, has less connection weights and so will take less time to process. As this is an important factor in the eventual real-time use of the neural network, the network structure with 20 hidden nodes was selected here.

2.1.2.4.4 Performance Results for the Selected Network

Identification of square, triangular and sinusoidal waveforms

In addition to the very good classification performance described in the previous section, during tests the 20 hidden node MLP repeatedly achieved correct identification in a quarter-period of the waveform, and the classification process was not too computationally intensive.

Rejection of other waveform types

The neural network was also tested on waveforms it was not trained to identify, i.e. *not* square, triangular or sinusoidal signals. In such cases, the network should produce an inconclusive result, unless the waveform is structurally similar to one of the three recognised types. Using a periodic exponential waveform, the network produced only inconclusive results, which was equivalent to a 100% rejection rate as required.

Hence, the neural network approach can be successfully used to discriminate between the three waveforms, even in the presence of signal noise.

2.1.2.5 Summary

Of the methods considered for implementing the real-time/Boolean translator (RBT), the gradient and the Fourier transform approaches proved to be of little use. However, promising results were obtained with the other methods evaluated, in particular the neural networks.

The advantage of using medians and means as a feature set is that the waveform features are identified in advance, thereby simplifying the RBT which then has only to extract and classify them. Implementation of the pre-processing techniques described in §2.1.2.4.2 above would further make this method amplitude and frequency independent. However, its disadvantage is that such a feature set may not be discriminatory enough, especially in the presence of noise. Additionally, tests have shown that the regions representing the sinusoidal and the triangular waveforms are extremely closely situated in the feature space, even at low noise levels. This could lead to a significant number of classification errors.

A neural network could overcome the proximity problem of the regions by learning and internally forming a non-linear combination of the two features, resulting in a more discriminatory feature set. This adds more processing requirements to the identification procedure though, and will extend the reconfiguration interval.

Training the network with raw waveform data, on the other hand, would cause it to extract and learn a non-linear combination of some features that, due to the learning process, maximises the discriminatory power of the resultant feature set. Such networks proved very successful during evaluation, and good classification was obtained even in the presence of significant noise levels. Although the training of the network was a somewhat difficult procedure requiring knowledge of the learning algorithm, the neural network obtained was small enough not to require excessive processing time.

For these reasons, a neural network was implemented as the Real-time/Boolean Translator for this hypothetical application. The Multilayer Perceptron was chosen, with a three-layered structure:

- Input Layer : 50 nodes, linear activation function;
- Hidden Layer : 20 nodes, sigmoidal activation function;
- Output Layer : 3 nodes, linear activation function.

It was trained using the modified Backpropagation algorithm with the following parameters, as determined by the tests described earlier in this section:

- learning rates η = 0.3 (initial) and 0.1 (final);
- momentum terms α = 0.5 (initial) and 0.1 (final);
- error thresholds ϕ = 0.001 (initial) and 0.0001 (final).

Lastly, the signals which the network could recognise were limited to square, triangular and sinusoidal waves with a minimum period of 25.0 seconds. Other signal types would cause network indecision, thus effectively being rejected.

2.2 Supervisory Control Logic

The SCL is the *supercontroller* which performs all analysis of available system data and carries out any logical or 'reasoning' tasks in the hybrid control system (HCS). It forms the highest level in the HCS architecture.

2.2.1 Function

The main function of the SCL is to interpret the Boolean-format data passed to it by the Real-time/Boolean Translator, and from it infer the current state of the system and hence the control loop reconfiguration requirements. In the pattern generation application, the

SCL has to decide which reference signal is being presented to the system, and then to invoke the appropriate controller.

Its operation can be summarised as follows:

RBT output (binary)[8]	Interpretation	Action
1 0 0	sine wave present	request Type ω controller
0 1 0	triangular wave present	request Type II controller
0 0 1	square wave present	request Type I controller
0 0 0	no decision reached	no action
more than 1 bit set high	inconclusive decision	no action

Table 2-4. Summary of SCL operation.

The request generated in this way is first passed through a *moderation routine*, which ensures that spurious controller switching does not result and is the secondary function of the SCL module. Spurious switching can occur if the RBT passes inconsistent system information on to the SCL by, for example, first indicating the presence of a sine wave, then a square wave, and then a sine wave again (during three consecutive identification attempts). This would result in constant controller reconfiguration (*chattering*), with the system not being allowed to settle, and could further lead to instability and large reference signal tracking errors. The reasons for any inconsistencies in the RBT output are discussed in §D.5.

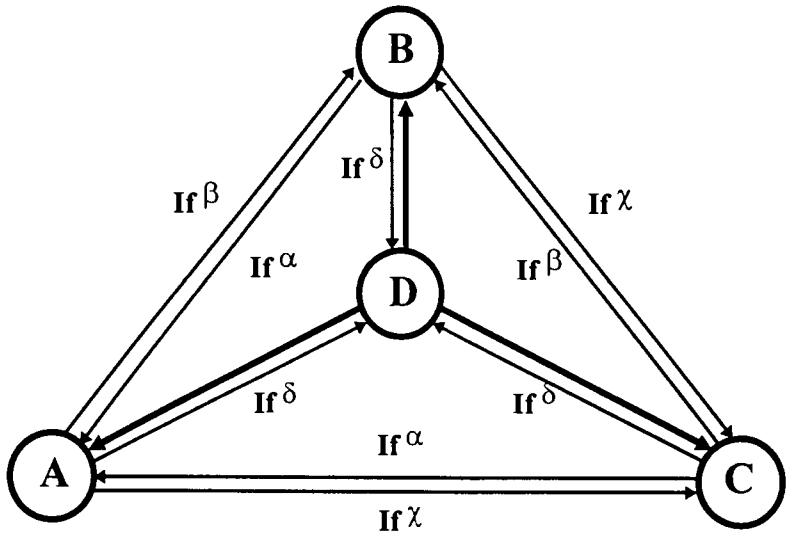
The moderation routine used here works simply on a *repeated request* basis, such that control loop reconfiguration is initiated only if two identical requests for a specific controller are received in sequence. Once this is the case, the SCL commands the appropriate controller (as indicated in Table 2-4) to take over active plant control from the current one. Bumpless transfer between controllers is a feature of their particular discrete difference equation representation, as discussed in §3.3. Furthermore, the design of the

[8] The RBT outputs are discussed in detail in §2.1.2.4.3.

controllers themselves is such that the control objectives are fulfilled simply by switching to the controller corresponding to the current reference signal.

2.2.2 Implementation

The SCL is implemented as a simple set of *if-then* conditionals, which can be viewed in finite state machine representation for clarity. Such a representation is shown in Figure 2-12 below. These conditionals are implemented in both the simulation and real-time control software, which can be found on the attached diskette.



where:

A = sine wave controller in use
B = triangular wave controller in use
C = square wave controller in use
D = no action taken, return to previous state

α = repeated request for Type ω controller
β = repeated request for Type II controller
χ = repeated request for Type I controller
δ = repeated request for current controller, or no valid request

Figure 2-12. Finite State Machine representation of the Supervisory Control Logic

3. Dynamic Controllers for the Real-time Loops

This section describes the design of the controllers which form the real-time loops of the Hybrid Control System (HCS) architecture. Each controller's performance is analysed in terms of noise and disturbance rejection, speed of response and stability. In addition, simulation and real-time control results for the overall hybrid system are shown.

3.1 System Overview

The control objectives for this hypothetical application are the tracking of three different types of reference signal *r(s)* with zero asymptotic error and minimum control effort. For the sinusoidal, triangular and square waveforms being considered here, no single fixed controller can easily achieve these objectives. Hence three separate controllers are used, and these must be switched in and out as required, under the guidance of the control logic discussed in Chapter 2.

The structure of the overall system is shown in Figure 3-1. Each of the three controllers operates independently in a closed-loop feedback configuration, with the controller choice being dictated by the output of the Neural Network waveform identifier. The "system" being controlled here is one of the two servo motors which form part of the hypothetical pattern generator.

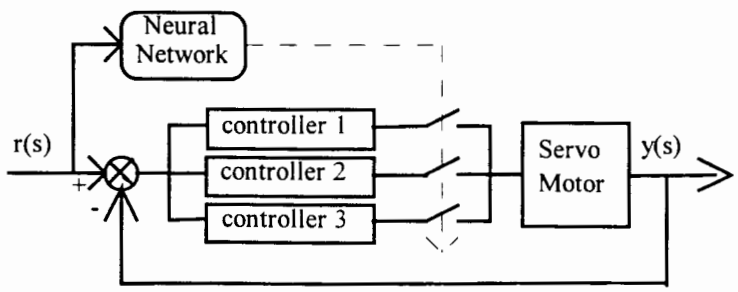


Figure 3-1. Control system block diagram.

3.1.1 System Identification

In order to form a mathematical model of the servo motor, it was subjected to several step tests, as detailed in §E.1. From these, a nominal first order model was derived and used in the subsequent design of the closed-loop controllers. A first order representation was used as it was a good enough approximation of the motor's characteristics for controller switching purposes.

The (stable) motor model obtained was:

$$g(s) = \frac{1.70}{1 + 0.80 \cdot s} \quad \frac{[\text{Volts}]}{[\text{Volts}]}$$

3.2 Controller Design

Three separate controllers were designed for this system, and were then converted to discrete difference equation representations for implementation on a PC. The following criteria were used in the design:

1. the velocity reference setpoint to be tracked does not have a fixed amplitude, but the mean amplitude is 1.0 TachoVolt[9]. In addition, the setpoint has an offset of 4.0 TachoVolts;
2. the controlled system must track the desired setpoint with zero asymptotic error;
3. to prevent damage to the motor, the amplitude of the *dynamic* control signal supplied to it must not exceed 2.5 Volts when the dynamic reference signal amplitude is 1.0 TachoVolts;
4. the controlled system must be stable;

[9] A measure of the motor's speed of rotation, obtained using a tachometer.

5. the controlled system must exhibit good output disturbance rejection - the noise due to sampling of the real-time system's output can be considered a form of output disturbance;
6. following a significant change in a square or triangular wave setpoint (e.g. step or ramp in the opposite direction respectively), the closed-loop system must achieve asymptotic tracking before the next such change occurs half a waveform period later. If the controlled system settles to within 5% of the desired speed within a *quarter-period*, this goal will be met. As the reference setpoint has a minimum period of 25.0 seconds, this in turn translates to the requirement that the closed-loop system (under Type I or II control) settle to within 5% of the desired speed in a maximum time of 6.0 seconds ($t_{5\%} \leq 6.0$ *seconds*). In the case of the sinusoidal setpoint, such a requirement is not as critical and is therefore relaxed to ($t_{5\%} \leq 8.0$ *seconds*) for a closed-loop system under Type ω (sinusoid-tracking) control.

NOTE: From standard s-plane theory, for the closed-loop system's output to settle to within 5% of its final value in 6.0 seconds its time constant τ must be such that $3\tau < 6.0$. Therefore, the Type I and Type II controllers were both designed such that the system's dominant closed-loop pole(s) λ_i satisfied the relation $Re(\lambda_i) < -0.5$ in each case. The corresponding relation for the Type ω controller was $Re(\lambda_i) < -0.4$.

The design procedure involved selecting different controller parameters with the aid of a Root Locus plot and a Nyquist diagram. Those parameters which resulted in the best simulation performance were then used. In order to test their robustness, each of these control loops were also subjected to disturbance and noise rejection tests using an additive output disturbance.

3.2.1 Type I controller

The Type I controller is typically used for constant setpoint tracking applications. It achieves the desired control objectives with minimum control effort when the reference setpoint is a step, or a combination thereof:

$$r(s) = \frac{A}{s}$$

Manipulation of several controller parameters, in conjunction with Root-Locus and Nyquist plot analysis, produced a controller with the following transfer function:

$$k(s) = \frac{K \cdot (s + a)}{s} \qquad \begin{cases} a = 2.0 \\ K = 0.8 \end{cases}$$

The resultant closed-loop system has poles λ_i at $s \approx (-1.5 \pm 1.1j)$ and therefore its output should settle to within 5% of its final value after 2.0 seconds, which is less than the design requirement of 6.0 seconds. In addition, from the Nyquist plot of Figure 3-3(a) on page 41 it can be seen that the curve passes well within the critical (-1,0) point. Hence the closed-loop system will be stable in this configuration.

Lastly, application of the final-value theorem to the error signal in the closed-loop system formed using this controller shows that e_∞ is zero, which satisfies the setpoint tracking design criterion.

3.2.1.1 Control-loop Simulation

From the simulation output shown in Figure 3-2 below, it can be seen that $t_{5\%}$ is approximately 2.0 seconds as expected, and that the amplitude of the dynamic control effort (u) does not exceed 2.5 Volts. Furthermore, the reference signal is asymptotically tracked as required.

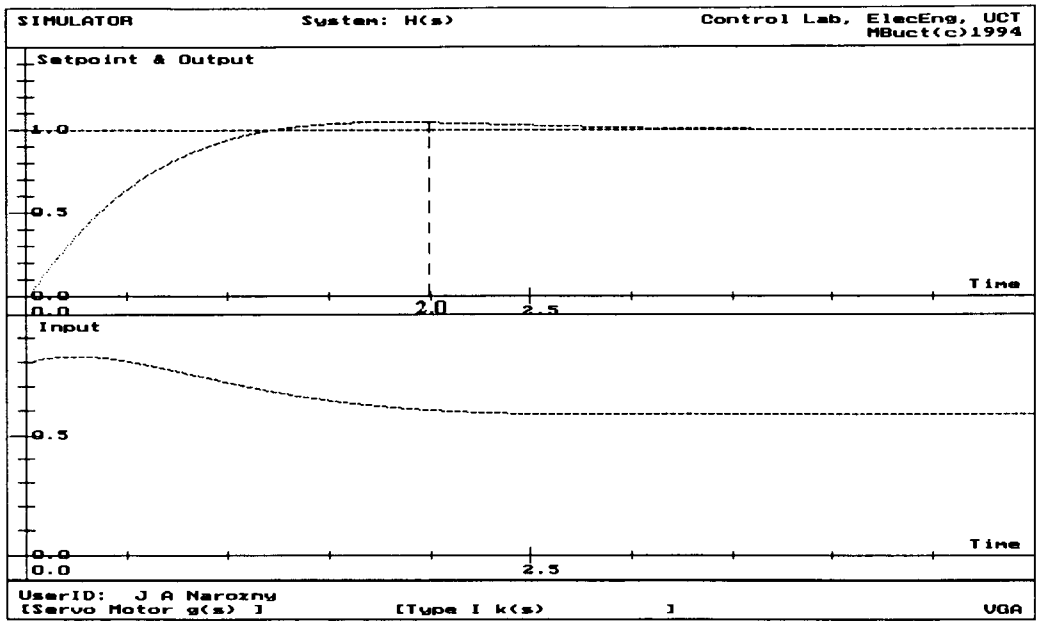


Figure 3-2. Closed-loop system response - Type I controller.

3.2.1.2 Disturbance Rejection

The output disturbance added during simulation of the closed-loop system consisted of a step of amplitude 0.2 Volts, combined with White noise to give a SNR of 20.0dB. The disturbance was applied before the system had settled, and no filtering was employed at this stage.

Closed-loop system performance under these conditions can be accurately predicted from the Nyquist plots of Figure 3-3 below. From part (a) of that figure, it can be seen that the q-plot approaches the |S|=1 circle[10] as the signal frequency $\omega \rightarrow \infty$. Similarly, part (b) of the same figure indicates that the closed-loop system has a -3dB bandwidth of approximately 0.39Hz. Hence, it can be expected that this system will tend to reject all but very high frequency disturbances, and its tracking performance will deteriorate for signal frequencies exceeding 0.39Hz. (The maximum reference signal frequency is 0.04Hz[11], which is well within the -3dB bandwidth.)

---

[10] This refers to the $\left|S_q^h(j\omega)\right| = 1$ circle.

[11] From $f_{max}=(T_{min})^{-1}$, T_{min}=25.0 seconds.

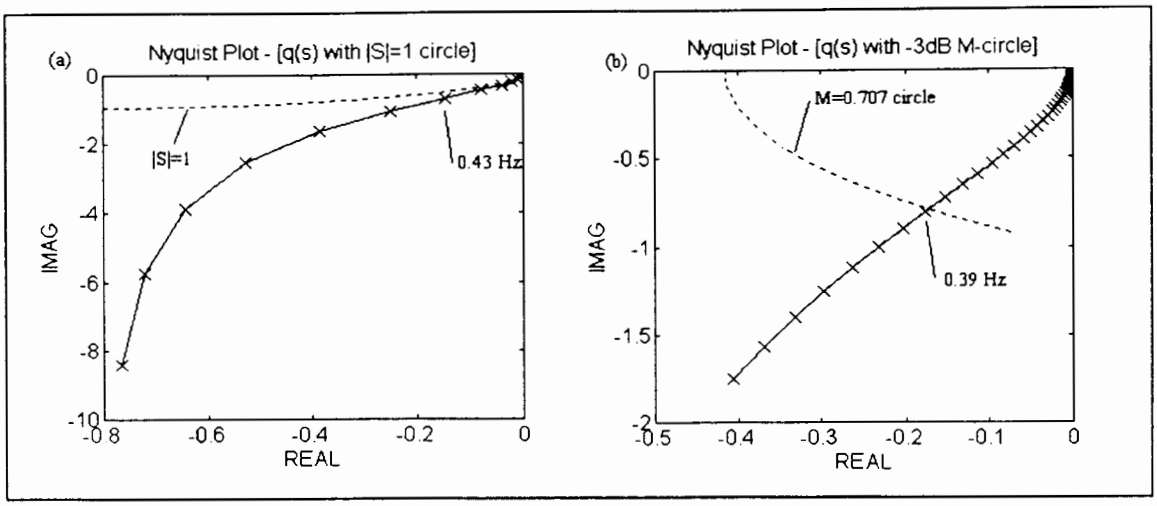


Figure 3-3. Nyquist plots - Type I controller.

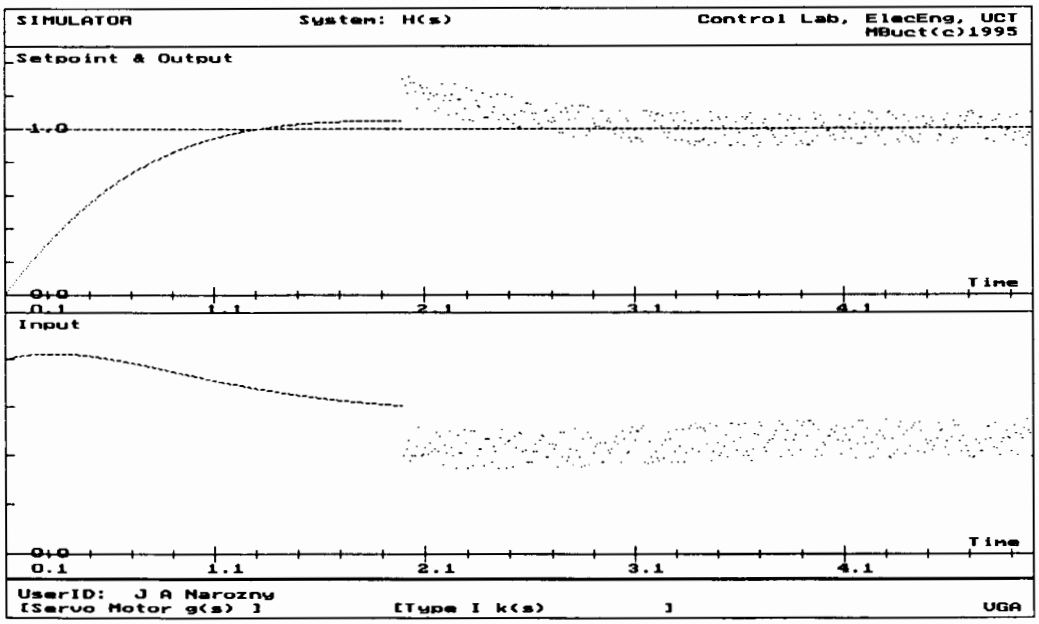


Figure 3-4. Output disturbance rejection - Type I controller.

Inspection of the additional simulation results shown in Figure 3-4 indicates that the system rejected the step disturbance, while the noise was not totally compensated for, as expected. The unfiltered noise contained some high frequency components, and as at those frequencies the system's q-plot enters the |S|=1 circle, these components were not

rejected by the system. A filter will band limit the noise to lower frequencies, hence the noise rejection performance will improve when it is implemented.

3.2.1.3 Discrete Representation

The continuous-time controller equation was converted to discrete difference equation format in order to simplify implementation of the controller on a computer. Since the difference equation format is causal, *in this case* the approach would also easily lend itself to approximately "bumpless" transfer between controllers, as discussed later in this chapter.

The type I controller was discretised using the step-invariant z-transform:

$$\begin{aligned} \mathrm{k}(z) &= \left(\frac{z-1}{z}\right) \cdot \mathbf{Z}\left[\frac{1}{s} \cdot \left(\frac{K \cdot (s+a)}{s}\right)\right]_T \\ &= \frac{0.8000 + (1.600 \cdot T - 0.8000) \cdot z^{-1}}{1 - z^{-1}} \\ &= \frac{\mathrm{u}(z)}{\mathrm{e}(z)} \end{aligned}$$

where: *Z[..]* is the z-transform;

T is the discrete sampling interval;

u(z) is the z-transform of the control effort;

e(z) is the z-transform of the tracking error.

Controller pole location: *z=1.0*

No ringing or unstable poles are present, and the corresponding difference equation was:

$$\mathrm{u}(k) = \mathrm{u}(k-1) + 0.8000 \cdot \mathrm{e}(k) + (1.600 \cdot T - 0.8000) \cdot \mathrm{e}(k-1)$$

The index *k* indicates the signal sample number; for example, *u(k)* is the value of the control effort at the current sample instant, *u(k-1)* is the value at the previous sample instant, and so on.

3.2.2 Type II controller

This controller is used mostly to track a constant (non-zero) gradient setpoint, although it can also be used to track a step signal albeit with larger overshoot than the Type I controller would produce. The Type II controller will achieve the control objectives when the reference setpoint is a ramp, or a combination thereof:

$$\mathrm{r}(s) = \frac{A}{s^2}$$

Application of the design methods led to the following final Type II controller configuration which asymptotically tracks the ramp setpoint as required:

$$\mathrm{k}(s) = \frac{K \cdot (s+a) \cdot (s+b)}{s^2} \qquad \begin{cases} K = 0.9 \\ a = 0.5 \\ b = 1.9 \end{cases}$$

The resultant closed-loop system has a dominant pole at $s \approx (-0.6)$, hence $t_{5\%}$ will be approximately 5.0 seconds, which is less than the design limit of 6.0 seconds. Furthermore, as the system's Nyquist plot of Figure 3-6(a) on page 45 passes well within the (-1,0) critical point, this configuration will be stable in closed-loop.

3.2.2.1 Control-loop Simulation

Simulation of the controlled system (Figure 3-5 below) showed that it tracked within 5% of the desired signal in less than the expected 5.0 seconds, and that the dynamic control signal did not exceed 2.5 Volts (on the interval where the setpoint ranged from 0.0 to 1.0 TachoVolts).

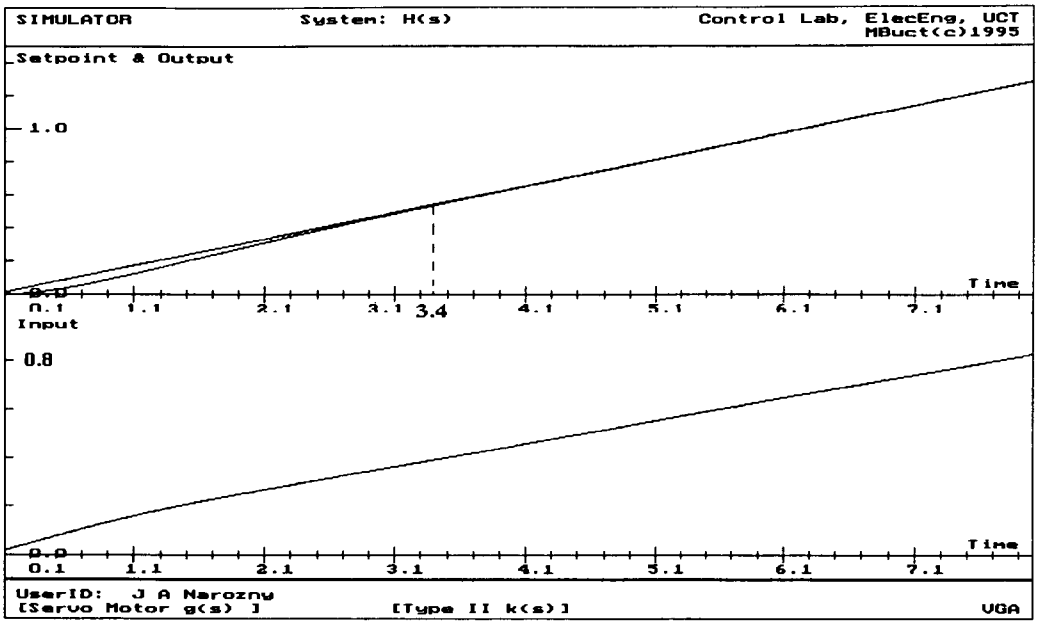


Figure 3-5. Closed-loop system response - Type II controller.

3.2.2.2 Disturbance Rejection

A step of amplitude 0.2V, combined with White noise to give a SNR of approximately 20.0dB, was used as the additive output disturbance during simulation of the closed-loop system. The disturbance was added before the system had settled to the desired trajectory, and no filtering was used at this stage.

The closed-loop system's response to such disturbances can be predicted from the Nyquist plots of Figure 3-6 below. As the q-plot enters the |S|=1 circle at a frequency of approximately 0.57Hz, the system will not reject disturbances whose frequency exceeds that value. Therefore this system is more noise-sensitive than the corresponding Type I controller configuration analysed previously, although filtering will band limit the noise and hence decrease the sensitivity. Tracking performance will also deteriorate for all signals with frequencies higher than the -3dB frequency of approximately 0.47Hz (from part (b) of the figure). However, this bandwidth is sufficient for the maximum reference signal frequency of 0.04Hz.

The predicted performance was observed during the additional simulation run of Figure 3-7 below. The step disturbance was completely rejected, while the unfiltered high frequency noise components were not compensated for, resulting in the noisy output shown in the graph.

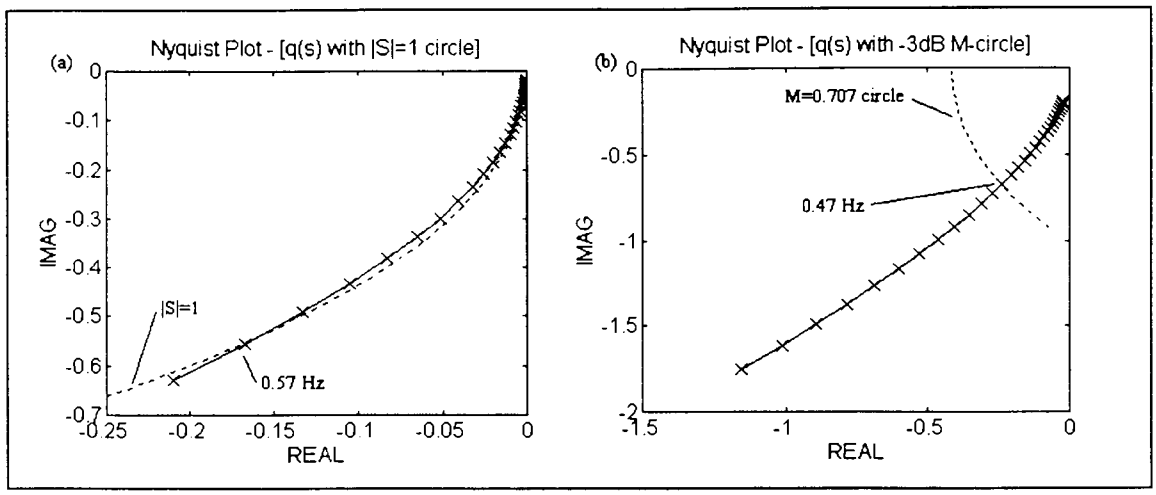


Figure 3-6. Nyquist plots - Type II controller.

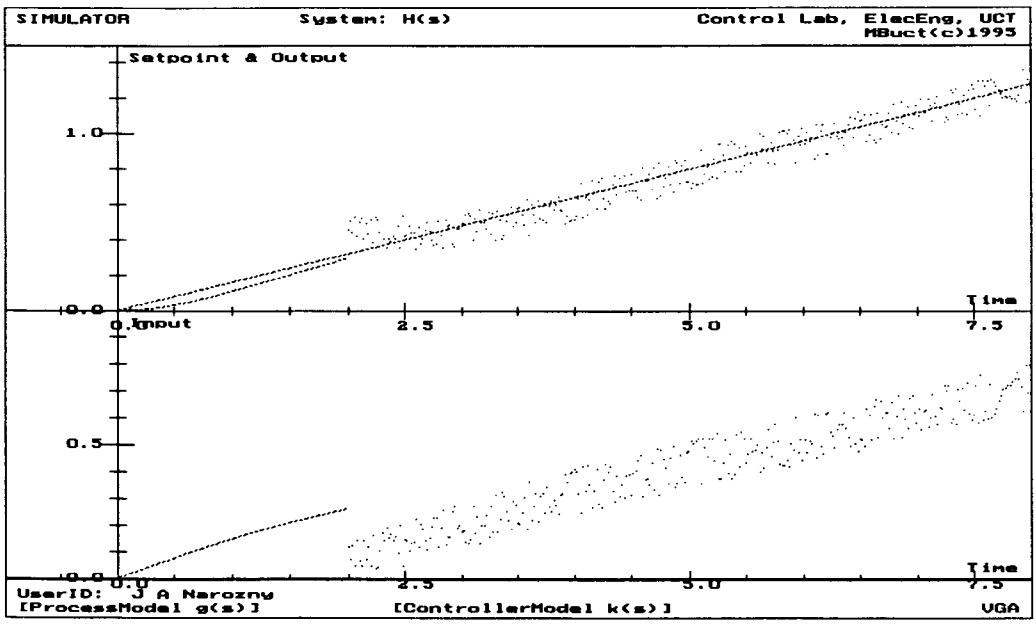


Figure 3-7. Output disturbance rejection - Type II controller.

3.2.2.3 Discrete Representation

The type II controller equation was discretised using the step-invariant z-transform to obtain the following z-plane formulation:

$$k(z) = \left(\frac{z-1}{z}\right) \cdot Z\left[\frac{1}{s} \cdot \left(\frac{K \cdot (s+a) \cdot (s+b)}{s^2}\right)\right]_T$$

$$= \frac{0.9000 + (0.4275 \cdot T^2 + 2.160 \cdot T - 1.800) \cdot z^{-1} + (0.4275 \cdot T^2 - 2.160 \cdot T + 0.9000) \cdot z^{-2}}{1 - 2 \cdot z^{-1} + z^{-2}}$$

$$= \frac{u(z)}{e(z)}$$

where: *Z[..]* is the z-transform;

T is the sampling interval;

u(z) is the z-transform of the control effort;

e(z) is the z-transform of the tracking error.

Controller pole locations: *z=1.0, z=1.0*

No ringing or unstable poles are present in the controller equation, and the corresponding difference equation was:

$$\begin{aligned} u(k) = {} & 2 \cdot u(k-1) - u(k-2) \\ & +0.900 \cdot e(k) \\ & +(0.4275 \cdot T^2 + 2.160 \cdot T - 1.800) \cdot e(k-1) \\ & +(0.4275 \cdot T^2 - 2.160 \cdot T + 0.900) \cdot e(k-2) \end{aligned}$$

The index *k* indicates the signal sample number; for example, *u(k)* is the value of the control effort at the current sample instant, *u(k-1)* is the value at the previous sample instant, and so on.

3.2.3 Type ω Controller

There is no prescribed controller structure for tracking a sinusoidal signal with an offset:

$$r(s) = \frac{A}{s} + \frac{B}{s^2 + \omega^2} \qquad \begin{cases} \omega \text{ is the frequency of} \\ \text{the sinusoidal component} \end{cases}$$

However, according to the Internal Model Principle **[Braae 1994]**, the resultant closed-loop system will asymptotically track the reference signal *r(s)* if the controller equation contains terms equal to *r(s)*. Hence, the following controller structure was used:

$$k(s) = \frac{K_a \cdot (s+a)^3}{s \cdot (s^2 + \omega^2)} \qquad \begin{cases} K_a = 1.0 \\ a = 0.8 \\ \omega = \text{reference signal frequency in } rad \cdot s^{-1} \end{cases}$$

The controller equation is a function of the frequency ω of the sinusoidal signal to be tracked, therefore the location of the closed-loop poles of the system also depends on ω. Nevertheless, inspection of the extended Root Locus plot (Figure 3-8 below) for $0.0 \leq \omega \leq \omega_{max}$, $\omega_{max}=0.25 rads^{-1}$, K_a and a fixed, indicates that the pole movement is negligible over this *permitted* frequency range.

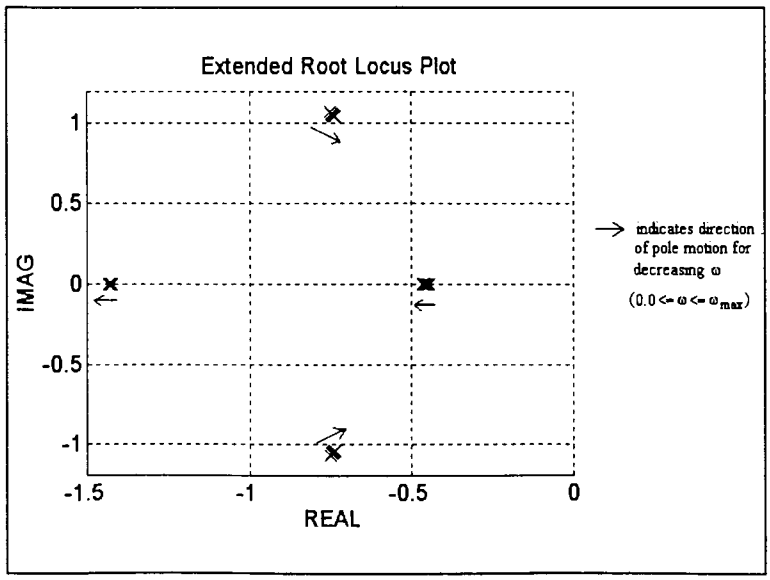


Figure 3-8. Poles of H(s) for varying ω - Type ω controller.

From the plot it can also be seen that, for a given set of controller parameters (K_a, a), the settling time ($t5\%$) of the closed-loop system increases slightly with ω. As the controlled system has to satisfy the design criterion $t5\% \leq 8.0$ *seconds* for all setpoint frequencies in the allowed range, it is therefore necessary to ensure that this is the case for the highest permitted frequency. With this in mind, the controller parameters (K_a, a) were optimised for a reference setpoint frequency $\omega = \omega_{max}$, giving the values $K_a=1.0$, $a=0.8$. These parameter values were subsequently used in the controller equation for all reference signal frequencies.

The resultant closed-loop system has a dominant pole at $s \approx -0.45$ for $\omega=0.25 rads^{-1}$, which gives a settling time $t5\%$ of approximately 6.7 seconds, well within the design limit. The corresponding settling time for $\omega=0.0 rads^{-1}$ is 6.4 seconds. The system also has a complex pole pair at $s \approx (-0.75 \pm 1.06j)$ for $0.0 \leq \omega \leq \omega_{max}$, thus some overshoot and oscillation is expected prior to settling.

Although this controller is of a non-linear nature, it was shown previously that the closed-loop poles do not move significantly over the permitted ω range and therefore it is possible to infer closed-loop stability for $0.0 \leq \omega \leq \omega_{max}$ from the system's behaviour at the two extremes of this range. The Nyquist plots of Figure 3-10 and Figure 3-11 on page 50 represent this, and from them it can be seen that the system will be stable in closed-loop, since its q-plot passes well inside the critical point (-1,0) point in each case.

Lastly, error analysis using this controller shows that it will track with zero asymptotic error:

$$
\begin{aligned}
e_\infty &= \lim_{s\to 0}\{s \cdot e(s)\} \\
&= \lim_{s\to 0}\left\{s \cdot \frac{r(s)}{1 + g(s)\cdot k(s)}\right\} \qquad \left\{g(s) = \frac{C}{(1 + s \cdot D)}\right. \\
&= \lim_{s\to 0}\left\{s \cdot \frac{\left(A \cdot s^2 + B \cdot s + A \cdot \omega^2\right)\cdot(1 + s \cdot D)}{\left(s^3 + s \cdot \omega^2\right)\cdot(1 + s \cdot D) + C \cdot K_a \cdot (s + a)^3}\right\} \\
&= 0
\end{aligned}
$$

3.2.3.1 Control-loop Simulation

From the simulation results of Figure 3-9, it can be seen that the controlled system took the expected 6.5 seconds (approximately) to settle to within 5% of the desired value, and that the offset sinusoid was asymptotically tracked as required. In addition, the dynamic control effort did not exceed 2.5 Volts, as can be seen by the maximum deviation from its average (offset) value, despite the presence of the predicted overshoot and oscillation.

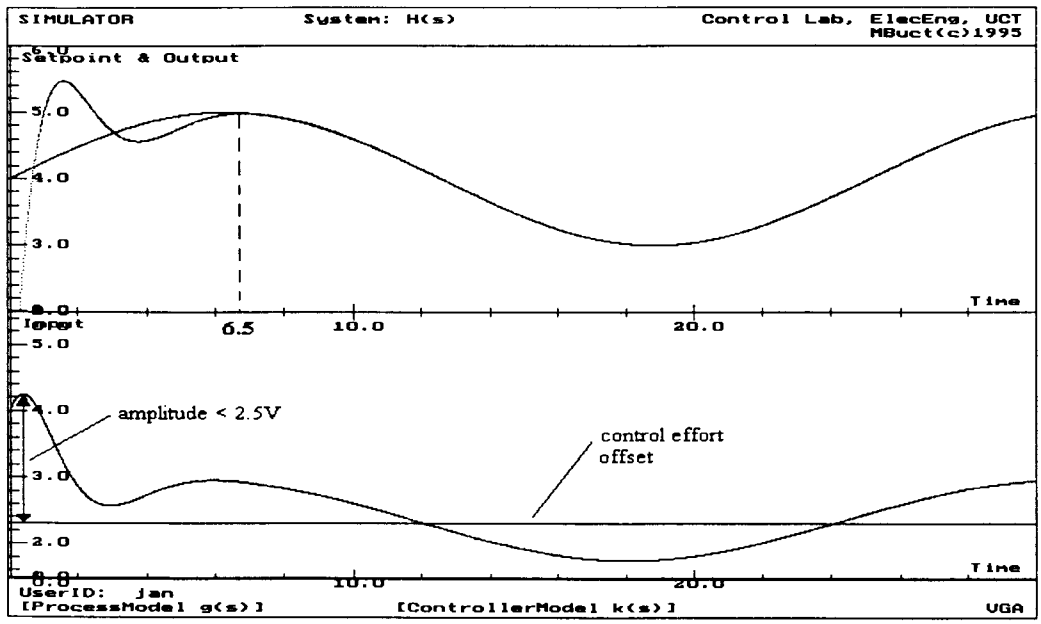


Figure 3-9. Closed-loop system response - Type ω controller, $\omega=\omega_{max}$.

3.2.3.2 Disturbance Rejection

During simulation tests, the output disturbance consisted of a step of amplitude 0.2Volts combined with White noise to give a SNR of approximately 20.0dB. This disturbance was added before the system had settled, and the signals were not filtered at this stage.

The closed-loop system's performance under these conditions can be predicted from the Nyquist plots of Figure 3-10 and Figure 3-11 below. These span the operational frequency range of the controller and, because the closed-loop poles do not move signifi-

cantly over this range, can be used to indicate system stability. In part (a) of both figures, the q-plot enters the |S|=1 circle at a frequency of 0.35Hz and 0.39Hz respectively. The system will therefore not reject disturbances with frequencies in excess of approximately 0.35Hz, thereby being quite noise sensitive. Similarly, part(b) of both figures shows that the closed-loop system has a -3dB bandwidth of approximately 0.49Hz. Hence tracking performance will deteriorate for all reference signals with frequency greater than 0.49Hz.

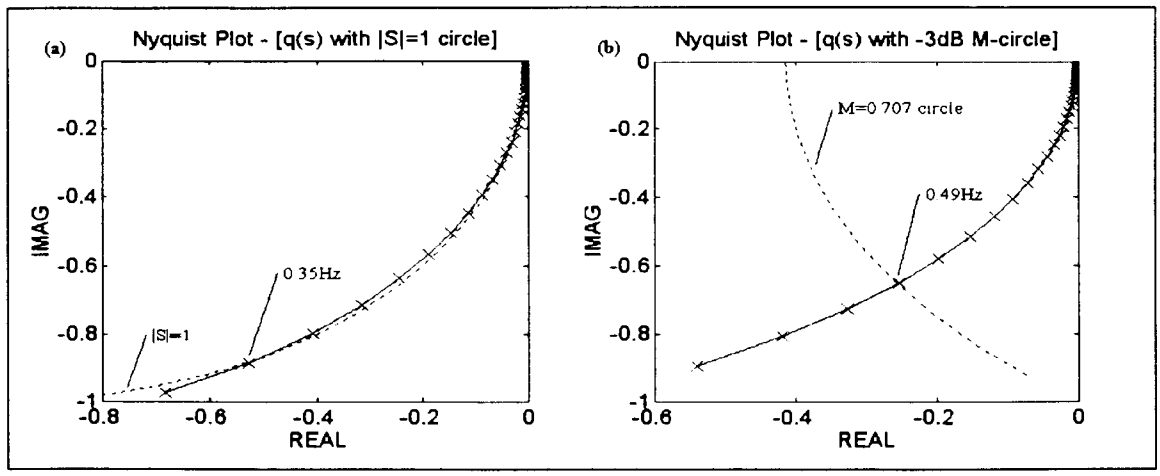


Figure 3-10. Nyquist plots - Type ω controller, ω=0.0rads^{-1}.

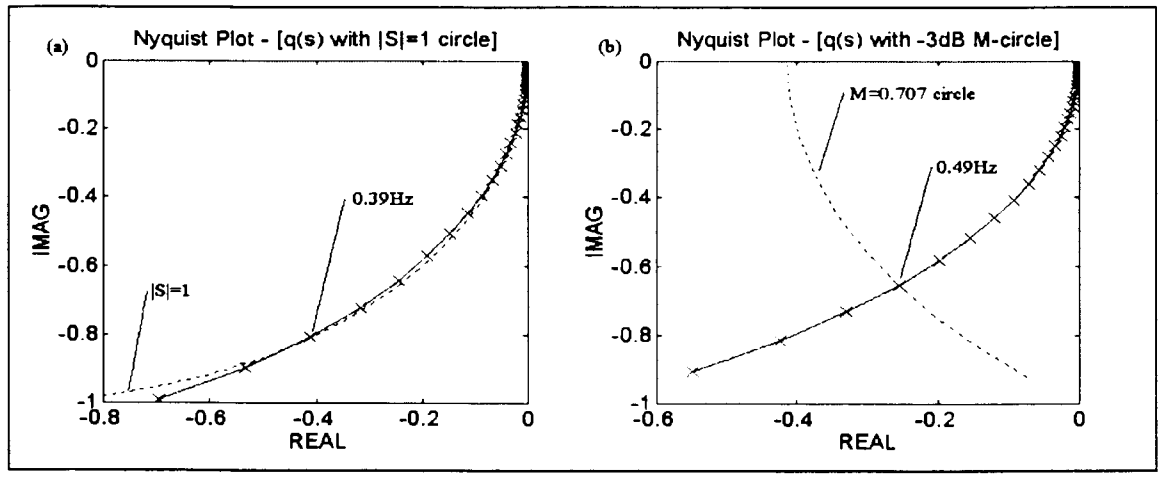


Figure 3-11. Nyquist plots - Type ω controller, ω=0.25rads^{-1}.

The predicted disturbance rejection performance can be observed in Figure 3-12 below, where the step disturbance was successfully attenuated by the system, while the high-frequency noise components were not compensated for. As filtering will band limit the

noise to lower frequencies, the system will become less noise sensitive when this is implemented.

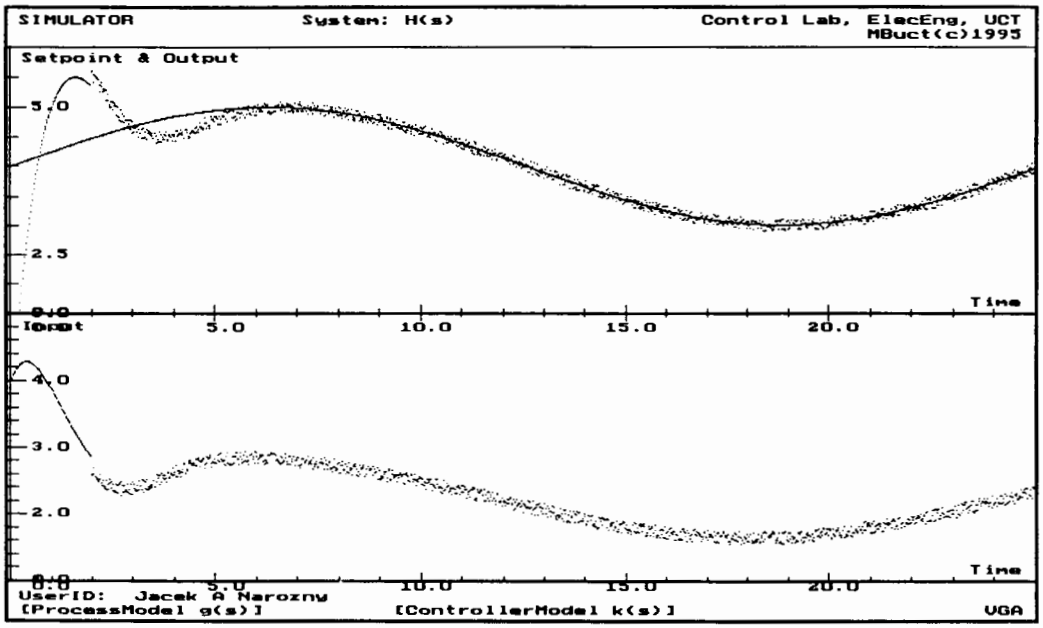


Figure 3-12. Output disturbance rejection - Type ω controller, $\omega=\omega_{max}$.

3.2.3.3 Discrete Representation

The type ω controller was also discretised using the step-invariant transform to give this z-plane formulation:

$$k(z)=\left(\frac{z-1}{z}\right)\cdot Z\left[\frac{1}{s}\cdot\left(\frac{K_a\cdot(s+a)^3}{s\left(s^2+\omega^2\right)}\right)\right]_T$$

$$=\frac{\left\{\begin{array}{l}1.000\\ +\left[\dfrac{0.5120\cdot T}{\omega^2}+A\cdot\sin(w\cdot T)-(2+\cos(\omega\cdot T))+\dfrac{1.920}{\omega^2}\cdot(1-\cos(\omega\cdot T))\right]\cdot z^{-1}\\ +\left[\left(1+2\cdot\cos(\omega\cdot T)\right)-2\cdot A\cdot\sin(\omega\cdot T)-\dfrac{1.024\cdot T}{\omega^2}\cdot\cos(\omega\cdot T)\right]\cdot z^{-2}\\ +\left[\dfrac{0.5120\cdot T}{\omega^2}-\cos(\omega\cdot T)+A\cdot\sin(\omega\cdot T)-\dfrac{1.920}{\omega^2}\cdot\left(1-\cos(\omega\cdot T)\right)\right]\cdot z^{-3}\end{array}\right\}}{1-(1+2\cdot\cos(\omega\cdot T))\cdot z^{-1}+(1+2\cdot\cos(\omega\cdot T))\cdot z^{-2}-z^{-3}}$$

$$=\frac{u(z)}{e(z)}$$

where: $Z[..]$ is the z-transform;

T is the sampling interval;

ω is the reference signal frequency;

$u(z)$ is the z-transform of the control effort;

$e(z)$ is the z-transform of the tracking error;

$$A = \frac{2.400 \cdot \omega^2 - 0.5120}{\omega^3}$$

Controller pole locations: $z=1.0,\ z=\cos(\omega T) \pm j\sin(\omega T)$

No unstable or ringing poles are present for the ω and T values used here. The equivalent difference equation representation is:

$$\begin{aligned} u(k) = {} & \left(1 + 2 \cdot \cos(\omega \cdot T)\right) \cdot u(k-1) - \left(1 + 2 \cdot \cos(\omega \cdot T)\right) \cdot u(k-2) + u(k-3) \\ & + e(k) \\ & + \left[\frac{0.5120 \cdot T}{\omega^2} + A \cdot \sin(\omega \cdot T) - \left(2 + \cos(\omega \cdot T)\right) + \frac{1.920}{\omega^2} \cdot \left(1 - \cos(\omega \cdot T)\right)\right] \cdot e(k-1) \\ & + \left[\left(1 + 2 \cdot \cos(\omega \cdot T)\right) - 2 \cdot A \cdot \sin(\omega \cdot T) - \frac{1.024 \cdot T}{\omega^2} \cdot \cos(\omega \cdot T)\right] \cdot e(k-2) \\ & + \left[\frac{0.5120 \cdot T}{\omega^2} - \cos(\omega \cdot T) + A \cdot \sin(\omega \cdot T) - \frac{1.920}{\omega^2} \cdot \left(1 - \cos(\omega \cdot T)\right)\right] \cdot e(k-3) \end{aligned}$$

The index k indicates the signal sample number; for example, $u(k)$ is the value of the control effort at the current sample instant, $u(k-1)$ is the value at the previous sample instant, and so on.

3.3 Bumpless Transfer

Switching between the three different control loops had to be performed as smoothly as possible, in order to avoid switching transients which could either damage the servo motor or else cause the system to become unstable.

Several "bumpless transfer" techniques are used in practice, depending on the controller implementation and the time constants involved. These techniques force the new controller to start off with initial conditions which ensure *continuity* in the control signal supplied to the plant. However, by definition a function *f(t)* is continuous at a given point $t=T$ if and only if:

$$f(T-\delta t) = f(T+\delta t) \qquad \{\delta t \text{ is small}$$

$$\left.\frac{\partial f}{\partial t}\right|_{(T-\delta t)} = \left.\frac{\partial f}{\partial t}\right|_{(T+\delta t)}$$

Hence, for smooth control transfer the controller output signal *u(t)* has to satisfy the above conditions at the switching instant. The corresponding discrete-domain conditions for *u[k]* are:

$$u_{K1} = u_{K2} \qquad \{\text{switching from controller K1 to K2} \qquad \ldots(1)$$

$$du|_{K1} = du|_{K2} \qquad \ldots(2)$$

Some of the more common methods of obtaining smooth switching are:

- Specifying a *transfer period* during which both the new and the current controllers provide control input in parallel. The new controller progressively provides more input as the current one is faded out; the transfer period is usually shorter than the time constant of the system being controlled. As the control signal immediately after the switching instant is taken mostly from the current controller, the conditions for smooth switching are satisfied.
- Forcing all the available controllers to *track* the output of the active controller. The states and initial conditions of each off-line controller are continually updated so that its output matches that of the active controller at the given

moment. The smooth switching criteria are therefore automatically met at the instant of active control transfer.

The discrete difference equation format of the controllers used in this work can be considered as a special case of the *controller tracking* approach to bumpless transfer. Here the continual update of controller states and initial conditions is replaced by the use of historical signal values in the computation of the difference equation for each controller:

Type I controller

$$u[k] = u[k-1] + Ae[k] + Be[k-1] \qquad ...(3)$$

Type II controller

$$u[k] = 2u[k-1] - u[k-2] + Ce[k] + De[k-1] + Ee[k-2] \qquad ...(4)$$

Type ω controller

$$u[k] = Fu[k-1] - Fu[k-2] + u[k-3] + Ge[k] + He[k-1] + Ie[k-2] + Je[k-3] \qquad ...(5)$$

N.B. 1. *A-J* are functions of the discrete sampling interval *T* and setpoint frequency *ω*.

2. Each of (3)-(5) use the *same* set of past values of *u* and *e*.

Assumption 1:

At the switching instant, $u[k-1] \approx u[k-2] \approx u[k-3]$ for small sample intervals *T* and sufficiently small signal noise levels (here defined by SNR > 20.0dB).

Equations (3)-(5) then become:

Type I controller

$$u[k] = \underbrace{u[k-1]}_{u_{ic}} + \underbrace{Ae[k] + Be[k-1]}_{du} \qquad ...(6)$$

Type II controller

$$u[k] = \underbrace{u[k-1]}_{u_{ic}} + \underbrace{Ce[k] + De[k-1] + Ee[k-2]}_{du} \qquad ...(7)$$

Type ω controller

$$u[k] = \underbrace{u[k-1]}_{u_{ic}} + \underbrace{Ge[k] + He[k-1] + Ie[k-2] + Je[k-3]}_{du} \qquad ...(8)$$

In order to satisfy the condition of (2), the terms *e[k-i], i=0..3*, were reset to zero at each switching instance. Hence, as the term u_{ic} in each of (6)-(8) was identical (to an approximation), the condition represented by (1) was also satisfied.

However, for noise levels represented by a SNR < 20.0dB Assumption 1 will not hold and control transfer will not be smooth. This will not pose a serious problem though, as the expected SNR is 25.0dB and the signals will also be filtered (thereby increasing the actual SNR).

Hence the control signal *u(t)* is approximately continuous, and the difference equation method (as outlined above) was used to implement sufficiently smooth transfer between the real-time control loops of the HCS.

3.4 System Simulation

The Hybrid Control System (HCS) structure was initially evaluated in a simulated environment, in order to determine the performance likely to be encountered in a real-time situation. To make the simulation as realistic as possible, noisy reference signal samples (filtered where appropriate) were used. The noise level was set to give a SNR of approximately 20.0dB in all cases.

3.4.1 The Simulation Algorithm

Three major components made up the simulated system: the process g(s), the discrete controllers k(z), and the neural network classifier. Due to the disparate requirements of each, three potentially different *time intervals* were used in the simulation:

1. The servo motor process was simulated using a Runga-Kutte approximation method, with a time step *sim_dt* set to 0.01 second.
2. The reference and process output signals were sampled using an interval T_S set to 0.01 second. This ensured that the 50 filtered samples obtained during a quarter-period would each be fitted using at least 10 non-overlapping, noisy samples of the actual signal:

   $$\text{Minimum interval between fitted samples} = \frac{T_{\min}}{(4 \cdot 50)}$$
   $$= 0.125 \text{ seconds}$$
   $$\geq 10 \cdot T_S$$

   The Savitzky-Golay causal filter (Appendix C) produced optimum performance under such conditions.
3. The discrete controller equations were evaluated using an interval *discrete_dt* set to 0.1 second. This choice was motivated by the need to reduce the sensitivity of the sinusoidal controller difference equations to numerical precision, as shorter intervals produced inconsistent tracking performance.

For example, with a *discrete_dt* of 0.01 second and a reference signal frequency of 0.25 rads^{-1} (equal to ω_{max}) the Type ω controller has a complex pole pair at

$$z=0.999997 \pm j0.002500.$$

Rounding errors in the PC could "move" these poles to $z=1.0$, which would eliminate the controller's oscillatory mode and thereby its sinusoid-tracking ability. Alternatively, using *discrete_dt*=0.1 second would place the pole pair at

$$z=0.999700 \pm j0.024997,$$

which is two orders of magnitude less sensitive to numerical precision.

The chosen discrete "sampling" interval is still small enough for the signals under consideration, as shown by the following application of the sampling theorem:

$$discrete\_dt \equiv 0.1 \text{ seconds}$$
$$\therefore\ sampling\_f = 10.0 \text{ Hz}$$

$$\text{We require } sampling\_f \geq 2 \cdot f_{max},$$
$$\text{and } f_{max} \equiv 0.04 \text{ Hz}$$

$$\Rightarrow sampling\_f \gg 2 \cdot f_{max}$$

Simulation was carried out according to the following basic algorithm. Note that for clarity the additional steps used to measure the signal period have been omitted. It should be assumed though, that the period is measured as and when required.

```
REPEAT
    • sample reference and output signals
    • IF a quarter-period has been sampled THEN
        • identify waveform
        • select appropriate controller
    • IF current time > discrete sampling time THEN
        • filter both reference and output signals
        • calculate e := r - y
        • calculate u from appropriate difference
          equation
        • discrete sampling time := current time +
          discrete_dt
    • simulate process using sim_dt for the remainder of
      the current sampling interval T_s
UNTIL end of simulation run
```

3.4.2 Expected Performance

Based on the performance characteristics already described for both the control loops and the neural network Real-time/Boolean Translator (RBT), the behaviour of the HCS can be predicted in terms of noise handling, control transfer, and the reconfiguration interval.

3.4.2.1 Noise Handling and Control Transfer

Tests of the RBT showed that it can successfully classify the three waveforms when the Signal-to-Noise Ratio (SNR) is greater than 20.0dB. These are the SNR levels expected during sampling, and therefore used in the simulation. Hence, no indecisions or misclassifications are expected unless extreme noise levels (SNR<14.0dB) are encountered.

As the sampled signals are first filtered to band-limit them and eliminate as much of the noise as possible, the dynamic controllers should be able to compensate adequately for the small noise levels expected. This also applies to the sinusoidal controller, which was shown to be the most sensitive to signal noise during disturbance-rejection analysis. Thus good tracking and adequate settling times are expected of all the control loops. Furthermore, control transfer between the different control loops is expected to be quite smooth due to the filtering. Only in the presence of extreme noise levels should any significant "bumps" become evident, as the assumptions made for bumpless transfer do not hold under those conditions.

3.4.2.2 Maximum Reconfiguration Interval

Various conditions can affect the length of this interval, ranging from the amount of signal noise present to the exact time during the identification process at which the reference signal changed type. A number of maximum reconfiguration interval values can therefore be obtained under different combinations of these conditions, hence only the typical, best and worst cases for a SNR of 20.0dB are given here.

In all cases, the interval includes the effect of the moderation routine as well as the maximum time required for the system to settle under the new control law. At a SNR of 20.0dB, the latter is on the order of 0.35 periods (sinusoidal controller, $t_{5\%}/T_{\min}$) and will increase with noise. Furthermore, the reconfiguration interval is stipulated in units of the reference signal period as the waveform identification process operates on fixed sections of this period, irrespective of its actual duration.

This discussion assumes that the reference signals to be tracked are of the type the network was trained to recognise: square, triangular and sinusoidal waves. Arbitrary signal types will indefinitely extend the reconfiguration interval, unless they are structurally similar to one of the recognised types, in which case the intervals derived below will apply.

3.4.2.2.1 Typical case

This is the reconfiguration interval expected under typical operating conditions, when the signal period has to be measured prior to identification, and the signal does not change type during either period measurement or the subsequent sampling and identification.

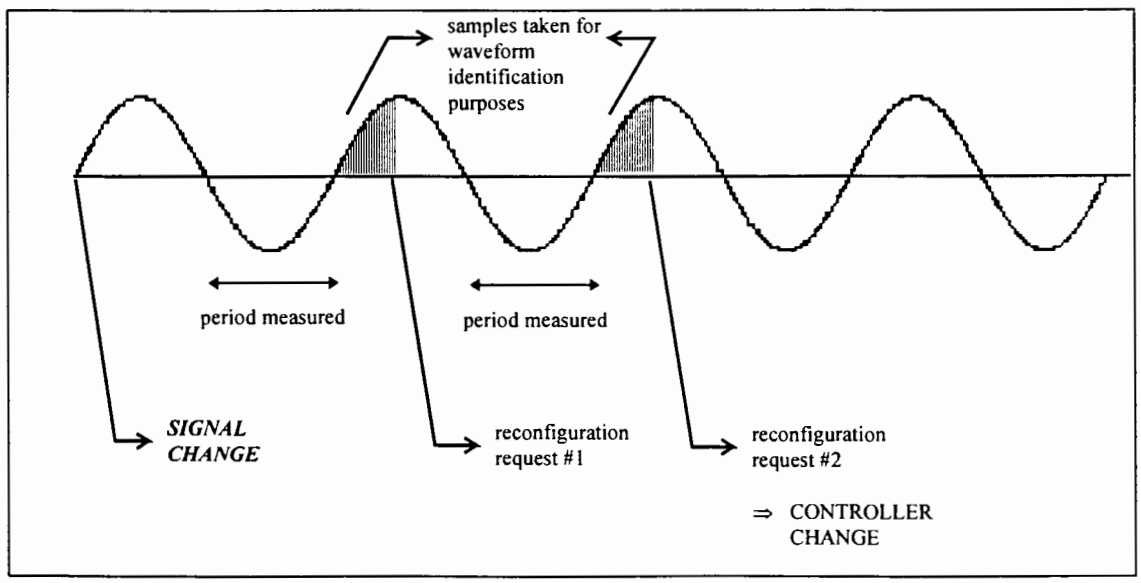


Figure 3-13. Reconfiguration interval under typical conditions.

As can be seen from Figure 3-13, 2.25 periods will elapse before a controller change is commanded. Adding the maximum expected settling time of 0.35 periods, this yields a typical maximum reconfiguration interval of 2.60 periods.

3.4.2.2.2 Best case

In the best case, the waveform period has been measured just prior to the signal change, and the new signal type has approximately the same period. Once again, the signal type is assumed to remain constant during subsequent sampling.

From Figure 3-14 below, the controller change will be commanded after 1.25 periods. Inclusion of the maximum expected settling time results in a best-case reconfiguration interval of 1.60 periods.

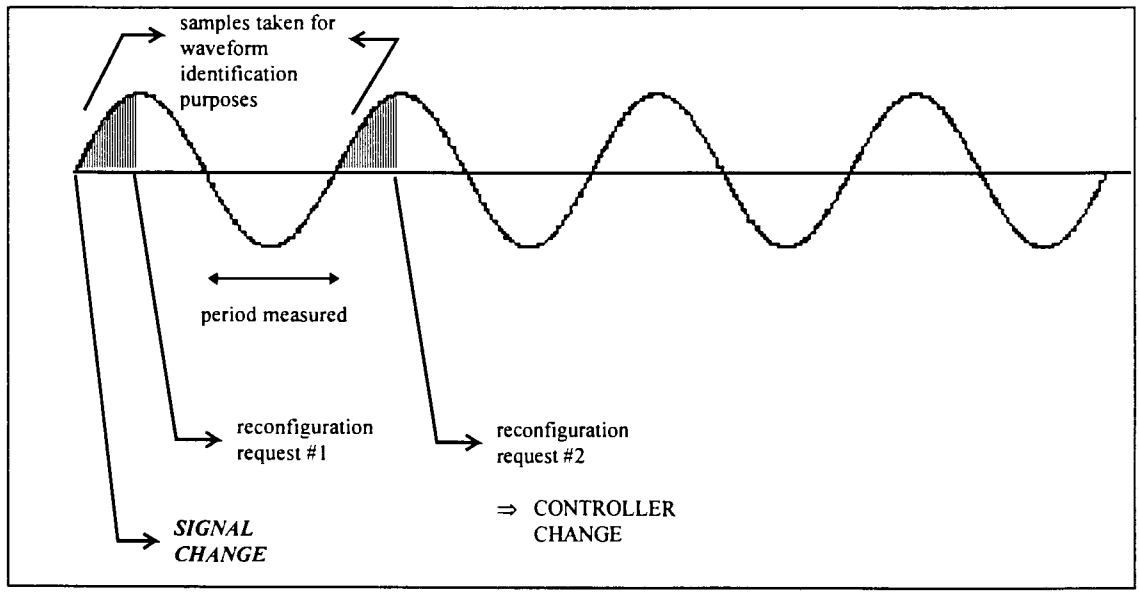


Figure 3-14. Reconfiguration interval under best-case conditions.

3.4.2.2.3 Worst case

At the noise levels being considered here, the worst case interval will occur if the set-point changes type immediately after the period has been measured, and has a shorter period than the previous setpoint type. The first samples of the new signal will then not span a quarter-period, and a misclassification or an indecision may result initially.

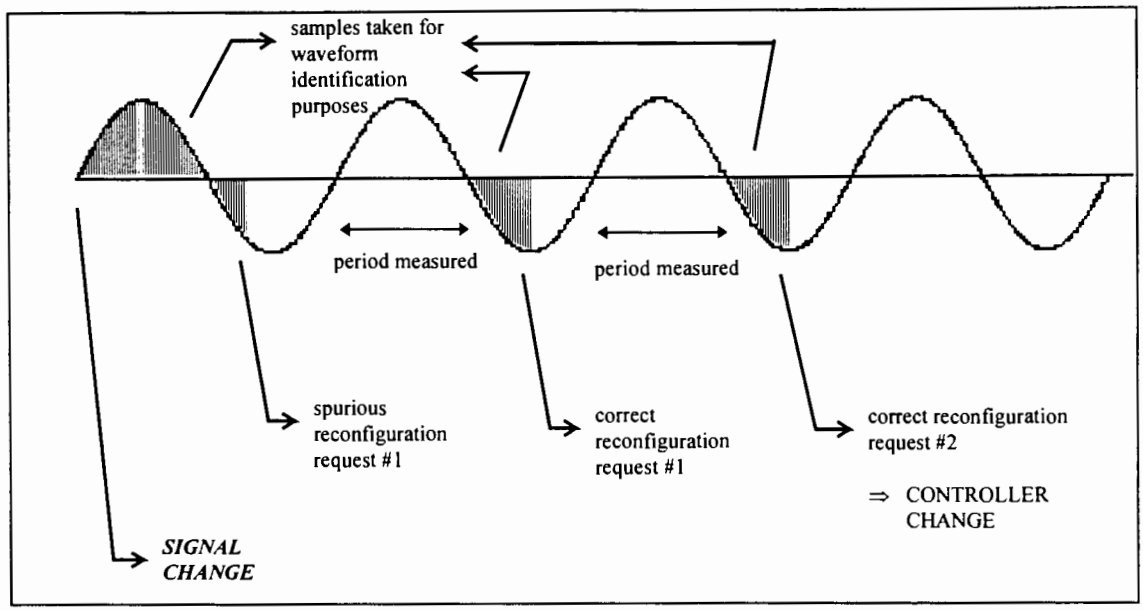


Figure 3-15. Reconfiguration interval under worst-case conditions.

In this case, controller change will be commanded after only 2.75 periods, leading to a worst-case reconfiguration interval of 3.10 waveform periods.

Note: Extraordinary levels of signal noise (SNR<14.0dB) could extend this worst case interval indefinitely. In this scenario, the noise would cause misclassifications, indecisions, and incorrect period measurements, which if combined in a given sequence can cause repeatedly inconsistent reconfiguration requests leading to no controller changes. Similar results would be observed if the reference signal changed again before the system had settled under the new controller (i.e. before reconfiguration was complete). However, such noise levels are not likely, and provided the reference signal is not changed prior to complete reconfiguration, the worst case interval will not be encountered in practice.

3.4.3 Actual Performance

During the simulation run, the HCS performed exactly as expected. For comparison purposes, the following discussion details this performance under similar headings to those used in the analysis of the HCS' expected behaviour.

In each of the simulation plots shown below, the closed-loop system was initialised with the Type II controller active and a square wave as the reference signal. A third of the way into the simulation run the reference signal was changed to a triangular wave, and after an equal time interval to a sinusoidal wave. Each change of the active controller is indicated by a dashed vertical line, with the legend between these lines (along the top of the plot) specifying which controller was active during that period of the simulation. The three signals displayed in the plot are the process output *y(t)*, the control signal *u(t)* and the scaled asymptotic tracking error *e(t)*, all with units of Volts.

For the purposes of these tests, the order of the reference waveforms and the initial controller type is not important.

3.4.3.1 Noise Handling and Control Transfer

With a SNR of 20.0dB (Figure 3-16 below), no indecisions or misclassifications were observed during the simulation run. The controlled system tracked the reference signal well and settled within the expected intervals, indicating good noise rejection performance. This further resulted in quite smooth switching between control loops (even for the noise-sensitive Type ω controller), as evidenced by the system output (*y(t)*) and control effort (*u(t)*) plots in the figure. In addition, the control effort did not exceed the design limits.

However, at the more extreme noise level represented by a SNR of 10.5dB, a number of indecisions and misclassifications were observed (see Figure 3-17). These were confined to the sinusoidal and triangular waveforms, as per the classification performance of the neural network RBT. Signal tracking behaviour and settling times were also not as good under these noise conditions, and switching to the Type ω controller was somewhat bumpy. Despite this, the magnitude of the control effort *u(t)* did not exceed design limits, although the bad tracking performance meant that all the control objectives were not achieved.

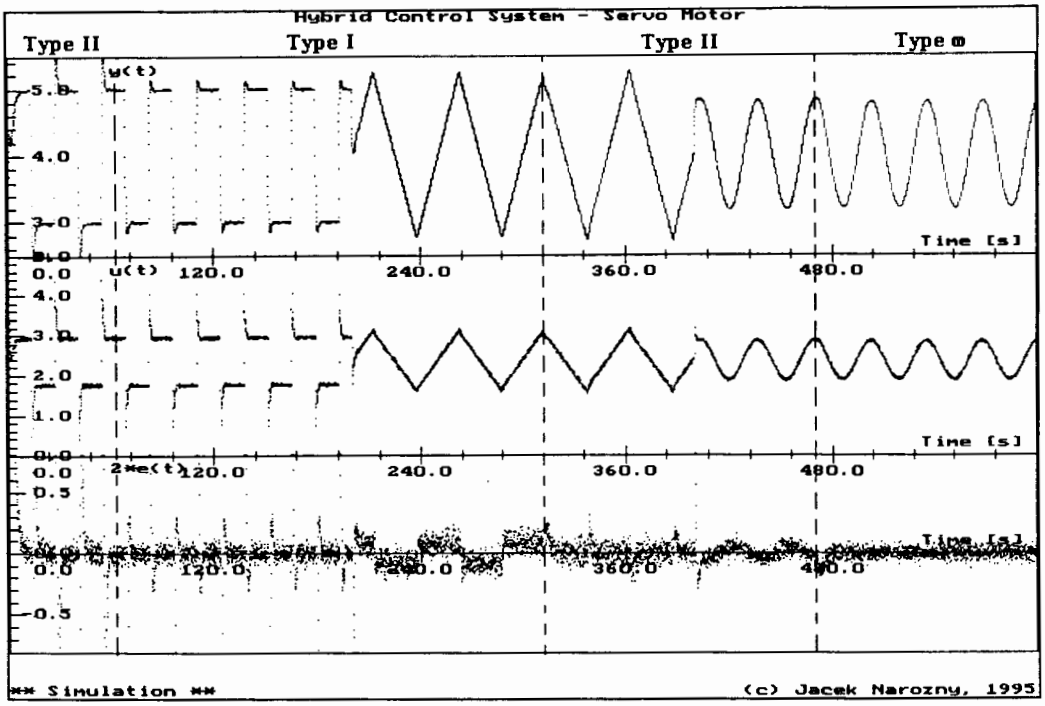


Figure 3-16. Simulation results: SNR of 20.0dB.

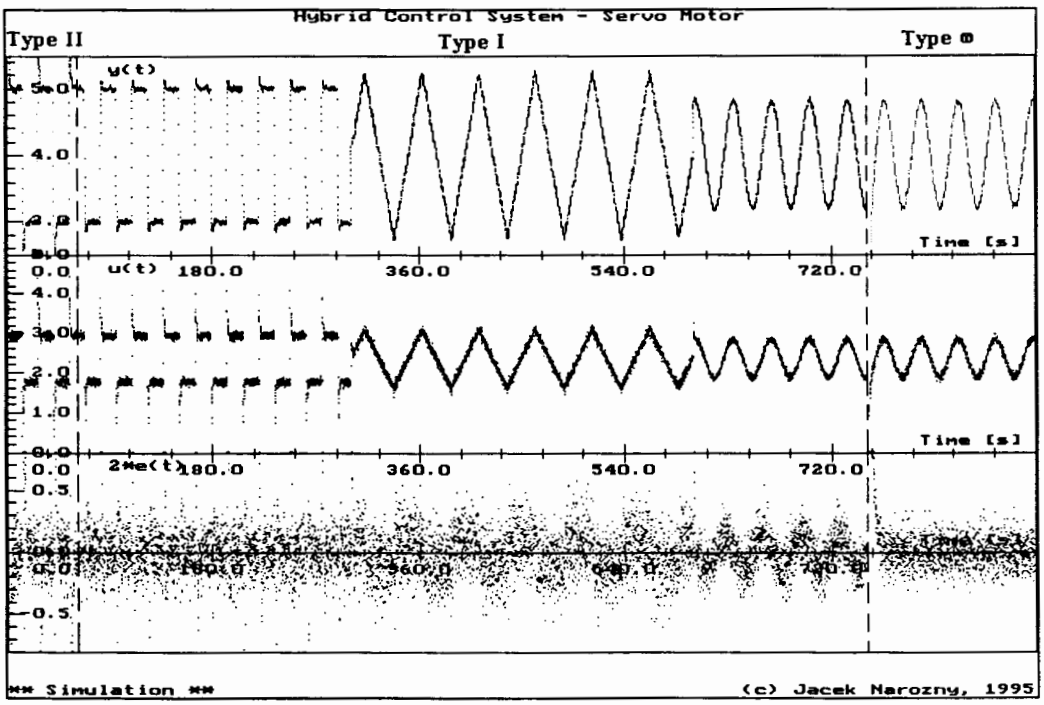


Figure 3-17. Simulation results: SNR of 10.5dB

3.4.3.2 Maximum Reconfiguration Interval

The exact values of the reconfiguration intervals varied with the duration of the simulation run, but not by significant amounts. On average, at a SNR of 20.0dB the reconfiguration intervals corresponded to those observed for the simulation run of 600 seconds' duration (Figure 3-16). As can be seen from Table 3-1, these reconfiguration intervals fell into the Typical category, as expected at this level of signal noise.

Switching to:	*Reconfiguration Interval [periods]*
Type I controller	2.5
Type II controller	2.5
Type ω controller	2.5

Table 3-1. Reconfiguration intervals: 600sec simulation run, SNR of 20.0dB.

Increasing the noise level to give a SNR of 10.5dB caused the reconfiguration intervals to approach the worst-case scenario, due to the occurrence of indecisions and misclassifications. Longer settling times, especially in the case of the Type ω controller, also contributed to this, resulting in the long reconfiguration intervals listed in Table 3-2.

Switching to:	*Reconfiguration Interval [periods]*
Type I controller	2.5
Type II controller	> 3.1
Type ω controller	5.5

Table 3-2. Reconfiguration intervals: 600sec simulation run, SNR of 10.5dB.

The Type I controller's reconfiguration interval was similar to that in the low-noise case because the RBT always identifies the square waves correctly, irrespective of noise levels. Hence, no indecisions or misclassifications occurred with this waveform, unlike the triangular wave which was not identified successfully before the signal switched to the sinusoid. Thus the Type II controller was never switched to in this simulation run.

The benefit of the moderation routine can be seen by comparing Figure 3-17 with Figure 3-18 below. In the latter case, the moderation routine was not used and some spu-

rious controller reconfiguration occurred. The sinusoidal signal was first misclassified as a square wave and then as a triangular wave, before finally being correctly identified. The corresponding controller was activated each time, and although the reconfiguration interval was not significantly affected, such low frequency *chattering* could damage the servo motor.

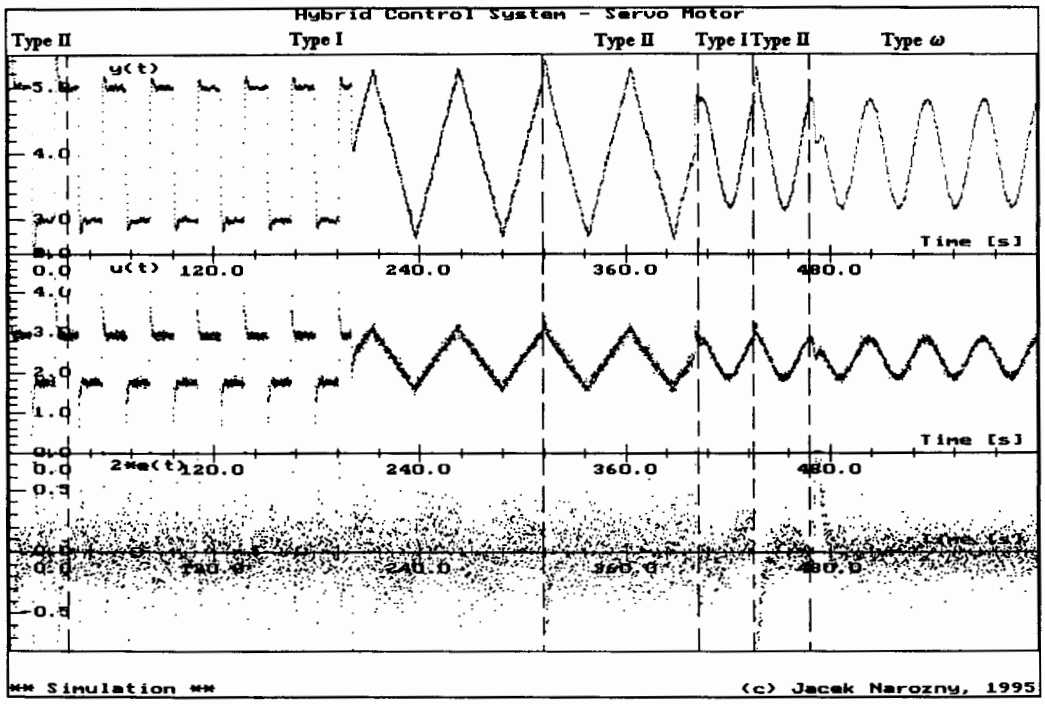


Figure 3-18. Simulation results: SNR of 10.5dB, no moderation routine.

3.4.3.3 Computational speed

Most of the simulations were performed on a 40MHz 80486-based PC for speed of execution reasons, and the average execution time of the 600.0 second simulations was 65.0 seconds in real-time. This shows that the computational complexity of the neural network waveform identifier did not adversely affect the HCS' performance, and that the necessary calculations could be comfortably performed in real-time on such a machine. Tests using a slower machine (16MHZ 80286-based) gave an average execution time of 295.0 seconds for simulations of 600.0 seconds' duration.

The effect of a slower PC on the system's performance was investigated during tests of the HCS on the real-time servo motor system, and is detailed in the following section.

3.5 Real-time Control

Real-time control of the servo-motor process was carried out using a slow 16MHz 80286-based PC, with all parts of the HCS architecture simulated on the computer.

3.5.1 Control Algorithm

The algorithm used for real-time control was very similar to that used in the system simulations. The main difference was that instead of generating both the reference and process output signals, the latter was sampled. Two timing intervals were used, one for obtaining signal samples (T_S), and one for simulating the discrete controllers (*discrete_dt*), with the same values as in the simulation algorithm.

The additional steps used to measure the signal period have again been omitted for clarity. It should be assumed though, that the period is measured as and when required.

```
REPEAT
    • sample reference and output signals
    • IF a quarter-period has been sampled THEN
        • identify waveform
        • select appropriate controller
    • IF current time > discrete sampling time THEN
        • filter both reference and output signals
        • calculate e := r - y
        • calculate u from appropriate difference
          equation
        • discrete sampling time := current time +
          discrete_dt
```

- wait out the remainder of the current sampling interval T_S

UNTIL end of test run

3.5.2 Expected Performance

Based on the results obtained during the simulation and the analysis carried out in §3.4.2, the performance of the real-time control system should match that of the simulated system for signals with a Signal-to-Noise Ratio (SNR) of 20.0dB. No indecisions or misclassifications are expected, control transfer should be quite smooth, and typical reconfiguration intervals should be observed. The control objective of reference setpoint tracking with minimum control effort should also be achieved.

In addition, from the execution times obtained in §3.4.3.3 it is expected that the HCS should be able to provide adequate real-time control of the process. The computations involved with the use of the neural network RBT should not compromise the control loop through the introduction of delays of any sort.

3.5.3 Actual Performance

The HCS performed exactly to expectations in real-time. The SNRs of the sampled signals were within the predicted 25.0dB, hence no indecisions or misclassifications occurred. In addition, the control effort *u(t)* did not exceed the design limits once during the test runs.

A sample output showing the system's performance is given in Figure 3-19 below. In this plot, the closed-loop system was initialised with the Type II controller active and a square wave as the reference signal. A third of the way into the simulation run the reference signal was changed to a triangular wave, and after an equal time interval to a sinusoidal wave. Each change of the active controller is indicated by a dashed vertical line, with the legend between these lines (at the top of the plot) specifying which controller

was active during that section of the simulation run. The three signals displayed in the plot are the process output *y(t)*, the control signal *u(t)* and the scaled asymptotic tracking error *e(t)*, all in Volts.

The order of the reference waveforms and the initial controller type is not important for the purposes of this test.

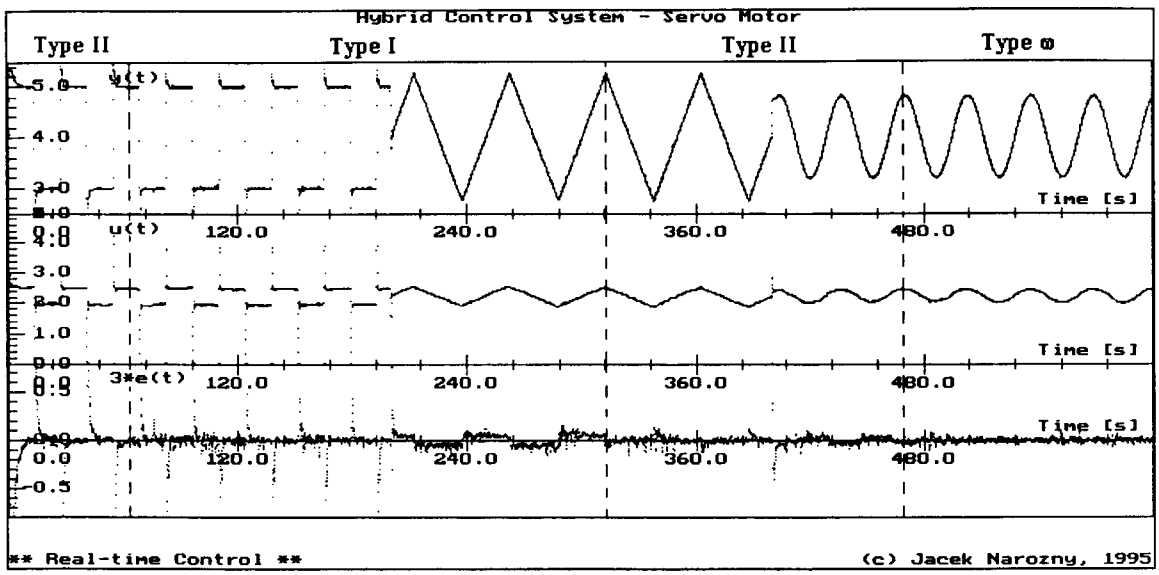


Figure 3-19. Real-time control results.

From Figure 3-19, it can be seen that the system tracked the reference signal well, and settled within the expected intervals once the correct controller was chosen. This is especially evident from inspection of the plot in the area of the Type I-to-Type II controller transfer on the triangular wave signal; the tracking error *e(t)* asymptotes to zero once the Type II controller is active. Furthermore, switching between the controllers was quite smooth, with the discontinuities observed on the plot being due to harsh changes in the reference signal during waveform switches. However, these were well compensated for by the controllers, showing good input disturbance rejection.

The average reconfiguration intervals encountered during the real-time runs generally fell into the typical category, as shown in Table 3-3 below. Slight increases in these intervals were observed during longer test runs, although the behaviour of the overall system satisfied the control objectives in all cases.

Switching to:	*Reconfiguration Interval [periods]*
Type I controller	2.5
Type II controller	2.5
Type ω controller	2.5

Table 3-3. Reconfiguration intervals: real-time run.

Lastly, the good performance of the HCS when implemented on such a slow PC shows that the computational requirements of the neural network RBT did not prevent the system from achieving control objectives in real-time. It should be noted, though, that this was the case only when a small network was used (20 hidden nodes). Real-time trials using a larger network structure (75 hidden nodes) indicated that the time required to process it compromised the performance of the control loops **[Narożny 1995]**.

3.6 Summary

Individually, each of the designed controllers achieved the control objectives set out at the beginning of this chapter. Combined into the HCS in a simulation environment, they performed well at SNRs of 20.0dB and above. Good noise rejection, tracking and quite smooth control transfer, as well as typical reconfiguration intervals were observed throughout.

At higher noise levels, the RBT produced some misclassifications and indecisions and the HCS' performance deteriorated. Reference signal tracking was not that good, the control law switching was quite bumpy, and worst-case reconfiguration intervals occurred. Even though the control effort remained within limits, the control objectives were not met because of this behaviour.

In real-time though, the HCS performed extremely well. Good tracking, noise rejection and almost bumpless transfer were produced during each run. Typical reconfiguration intervals were obtained, and the overall system achieved its control objectives. Further-

more, the slow PC used for real-time control did not adversely affect performance, provided a small neural network structure was used for the RBT.

Finally, due to the modularity of the design, this system can be easily extended to track other types of reference signals. This can be achieved by implementing the appropriate feedback control loops and simply re-training the neural network to also recognise the corresponding waveforms.

4. Alternative Switching Control Methods

Several other interesting approaches to switching control have been presented in recent literature (**[Morse 1994] [Davison 1995]**), and in this section some of these methods are briefly examined to evaluate their potential as alternative implementations of the hybrid control system (HCS) for the hypothetical pattern generation application being considered in this work.

4.1 Overview

Many of the methods considered here use some form of "output estimation error" or *performance signal* (also called a *switching criterion* in certain implementations) to infer the controller switching requirements; if this signal exceeds limits defined by a suitable *bounding function* the controllers must be repeatedly switched until one of them causes the performance signal to be minimised for the given operating conditions. This is similar to the concept of certainty equivalence as found in parameter adaptive control, and indicates that these "heuristic" switching control techniques actually have a strong basis in adaptive control theory **[Morse 1994]**. In addition, the implementation of the switching "logic" often relies on the explicit mathematical formulation of the switching criteria which govern the selection of particular controllers. However, such a formulation is not always trivial or intuitive.

A majority of the published work in this field has been carried out by Davison **[Davison 1992-1995]**, although he mainly deals with adaptive stabilisation in the case of uncertainty in the plant model. In his early research on the subject **[Davison 1992]**, he assumed that the uncertain model G_u falls into a subset of the set of known plant models $\sum G$, where for each *known* model G_i there exists a matching stabilising controller gain K_i. By switching to the controller K_i, corresponding to the known plant model G_i to which G_u is similar, the uncertain plant would be stabilised. This approach can therefore be viewed as a form of automatic gain scheduling.

In the particular case considered in **[Davison 1992]**, the control objective is asymptotic error regulation for reference and disturbance signals which are described by a known unforced differential equation (the step, ramp and sinusoid fall into this category). The controller structure consists of a servo-compensator for asymptotic tracking of the reference signal (as dictated by the Internal Model Principle **[Braae 1994]**) and a stabilising term for state bounding, scaled by the above-mentioned gains K_i.

The gains required for asymptotic error regulation are selected according to a *tuning function h(t)*, which reflects the magnitude of the plant states and indicates whether or not the tracking error is going to zero. However, in this early implementation of his method the servo-compensator is not modified by the switching process - thus only one class of setpoints can be tracked by a given design.

The choice of the tuning function is very much problem-specific and even arbitrary, and it appears from tests (see §4.2 below) and other work by the same authors (e.g. **[Davison 1995]**) that a particular *h(t)* is unlikely to produce the desired results for all classes of setpoints and all plant models. Furthermore, the only guideline given by the authors for choosing *h(t)* is that it must be a strictly increasing function which grows "sufficiently fast".

Davison demonstrates this method for a constant setpoint and a simple minimum phase plant model, and achieves asymptotic tracking with a particular choice of *h(t)*. In subsequent work, he presents similar systems for adaptive tracking with a control input constraint **[Davison 1993]**, and reduces the amount of plant information that has to be known *a priori* **[Davison 1994] [Davison 1995]**. However, all of these methods have been developed for constant reference setpoints and it is assumed that the controllers stop switching at some time, thereby simplifying to LTI (Linear Time-Invariant) models. Hence in order to extend it to asymptotic error regulation for multiple setpoints *without* the inherent cessation of switching, as required by the hypothetical application considered in this work, this approach has to be quite extensively modified.

Davison *et al* readily admit that the choice of the tuning functions is an *ad hoc* process, and that a plant's matching (by design) controller is not always settled on when switching stops. This would present a problem in our case (assuming for completeness that the setpoint forms part of an augmented plant model), as asymptotic tracking depends on the appropriate controller being switched in. Furthermore, as each candidate controller is tried on-line before its tracking performance can be quantified, large switching transients are present. No attempt is made to minimise or eliminate these transients, and this system is not well behaved in the presence of measurement noise **[Davison 1992]**. These and other problems associated with extending these methods to the control problem being considered in this work are described in §4.2, during the more thorough analysis of the method of **[Davison 1995]**.

Remark Yet another approach to switching control, this time with the use of an "intelligent" supervisor, is presented in **[Morse 1994]** for SISO plants. Here a number of controllers work in parallel, although only one of them is actually used to drive the plant. Each controller also produces a performance index, which is a measure of that controller's ability to "identify" the plant and track the required setpoint; the supervisor switches controllers based on this index. This method is proven to be robust in the presence of bounded and constant disturbances and measurement noise **[Morse 1994]**, and it appears that it does not inherently assume the cessation of switching in finite time. However, it was developed for constant setpoints and with corresponding special controller structures (the integrator ensuring asymptotic tracking is outside the controller block affected by the switching) and performance signals, and as such it may be difficult to extend it to encompass the requirements of the HCS application. This approach has not been tested further in this work.

4.2 Evaluation of Davison's Switching Control Method

The technique considered in this section is taken from **[Davison 1995]**, and represents the general approach taken by Davison throughout his work on the subject. It is selected

for evaluation over his other methods as it allows the most leeway in the design of the controllers, which do not have to be specifically structured. Thus, the controllers presented in §3.2 can be easily incorporated. In the discussion which follows, proof of the assumptions and results taken from **[Davison 1995]** is omitted and can be found in that reference if required.

4.2.1 The Switching Control Methodology

Central to this entire approach is a *Modified Strong Bounding Function f(k)* which is used to define the limits on the performance indices at which switching occurs. The actual specification of *f(k)* is somewhat *ad hoc*, with Davison only requiring that it be strictly increasing over *k*. An example of such a function is:

$$f(k) = k \cdot e^{k^2} \qquad \{k \in \mathrm{N}$$

However, it appears that the definition of *f(k)* is very problem-specific, and a given function is extremely unlikely to produce satisfactory switching performance for all classes of setpoints and all plant models **[Davison 1995]**.

This approach assumes the following general state-space format for the plants and their corresponding controllers:

Plant

$$G_i : \begin{cases} \dot{x} = A_i x + B_i u \\ y = C_i x \end{cases} \qquad \ldots(1)$$

$$e = y_{ref} - y$$

where:

	x	is the plant state;
	u	is the control input;
	y	is the output to be regulated;
	y_{ref}	is the reference input;
	e	is the asymptotic tracking error;
	i	is the plant index.

Controllers

$$K_i : \begin{cases} \dot{\eta} = H_i\eta + J_i y + L_i y_{ref} \\ u = M_i\eta + N_i y + R_i y_{ref} \end{cases} \quad \ldots(2)$$

where: η is the controller state;

i is the controller index (equal to the plant index).

After each k^{th} switch, the active controller index i is given by the formula

$$i = ((k - 1) \bmod s) + 1, k = 1,2,3,\ldots$$

where s is the number of controllers (and by implication, matching plant models), and *a mod b* returns the integer remainder after the division of a by b. Hence the set of controllers is switched through cyclically, with each controller being possibly tried out more than one time. This apparently produces "better" transient response **[Davison 1995]** and in fact cyclic switching is necessary for the pattern generation application, where the reference signals corresponding to each controller can occur more than once during a run.

Controller switching is initiated according to the following simple algorithm:

```
let k=1 at time t=0;
let s be the number of controllers;
REPEAT
    • i = ((k - 1) mod s) + 1;
    • use controller K_i;
    • IF (|η(t)| OR |e(t)|) >= f(k) THEN
        • k = k + 1;
UNTIL end of run;
```

From this it can be seen that switching will cease in finite time, provided $f(k)$ grows sufficiently fast enough. Furthermore, ever increasing deviations from nominal $\eta(t)$ and $e(t)$

are tolerated, so that if switching again becomes necessary *after* it has ceased, it may actually not be initiated. This will be problematic with regard to the hypothetical application, since there switching will be necessary every time the setpoint changes. The solution to this could be to reset the switching index k once an acceptable steady state has been reached, but such an approach may not be robust enough.

Lastly, the following assumptions must hold for the system:

1. $|\eta(0)| < f(1)$;
2. $|e(0)| < f(1)$;
3. for each plant and each corresponding controller K_i, the closed-loop system is stable and achieves the control objectives;
4. each plant is observable and controllable.

4.2.2 Simulation Results

In order to implement this switching control method as a HCS for the application being considered here, the servo motor plant model was used with the three controllers designed in Chapter 3, one for each type of setpoint. Although each of the controllers will stabilise the plant in closed-loop, only the correct controller will produce tracking error which asymptotes to zero for a given setpoint. Hence the switching should be initiated mainly due to limits on $e(t)$ being exceeded.

The respective continuous state-space formulations were:

Plant

$$\dot{x} = -1.250x + u, \quad x(0) = 0$$
$$y = 2.125x$$

Type I controller

$$\dot{\eta} = y_{ref} - y, \quad \eta(0) = 0$$
$$u = 1.600\eta + 0.800y_{ref} - 0.800y$$

Type II controller

$$\dot{\eta} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}\eta + \begin{bmatrix} 0 \\ 1 \end{bmatrix} y_{ref} - \begin{bmatrix} 0 \\ 1 \end{bmatrix} y, \quad \eta(0) = 0$$

$$u = \begin{bmatrix} 0.8550 & 2.160 \end{bmatrix}\eta + 0.900\, y_{ref} - 0.900\, y$$

Type ω controller

$$\dot{\eta} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & -\omega^2 & 0 \end{bmatrix}\eta + \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} y_{ref} - \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} y \qquad \{\omega = 0.25 rads^{-1},\ \eta(0) = 0$$

$$u = \begin{bmatrix} 0.5120 & (1.920 - \omega^2) & 2.400 \end{bmatrix}\eta + y_{ref} - y$$

The plant and the controllers conform to the format of equations (1) and (2) respectively, and assumptions (3) and (4) are easily verified for the plant. A suitable choice of the bounding function *f(k)* will additionally satisfy assumptions (1) and (2):

f(1)> *η(0)* implies *f(1)>0*

e(0)=y_{ref}(0)-y(0), and assuming a typical max(*y_{ref}(0)*)=5.0V,

e(0)=5.0-2.125x(0)

e(0)=5.0, and therefore

f(1)>e(0) implies *f(1)>5.0*

Hence the function *f(k)* must be chosen so that *f(1)>5.0.*

No other guidelines for choosing *f(k)* are available, hence the typical bounding function given by Davison was used initially:

$$f(k) = \begin{cases} 20 \cdot k, & 1 \le k \le 10 \\ \left(\frac{k}{6}\right)^2 \cdot \exp\left(\frac{k}{6}\right)^3, & k > 10 \end{cases}$$

This satisfies the requirement on *f(1)*, and some switching results obtained using this function are summarised in Table 4-1 below. Note that initially only the square wave and the triangular wave setpoints, with their respective controllers and without additive noise, were used during these tests. (If satisfactory switching performance is obtained, the tests can be extended to include the sinusoidal setpoint and the Type ω controller, as well as measurement noise.)

The objective of the tests was to have the correct controller active when switching stopped, irrespective of the initial controller choice, and the system had to settle in less than the required 6.0 seconds defined in Chapter 3. The setpoint type was not changed during a test run.

Reference Signal Type (correct controller Type)	*Initial Controller Type*	*Final Controller Type*	*Comment*
Square Wave (I)	I	I	switching stopped after 2.13 seconds; switching transients were present;
Square Wave (I)	II	I	switching stopped after a single switch at 0.62 seconds; switching transients were present;
Triangular Wave (II)	I	I	no switching took place;
Triangular Wave (II)	II	I	switching stopped after 3.50 seconds; switching transients were present

Table 4-1. Simulation results - Davison's switching control method.

For a square wave setpoint, the results show that switching stopped with the correct controller (Type I) selected irrespective of the starting controller choice, and that the system settled well within the required 6.0 seconds. However, when the setpoint was changed to a triangular wave, the system always selected the Type I controller and the *e(t)* limit was never exceeded to cause switching. Subsequent selection of an alternative bounding function

$$f(k) = \begin{cases} 10 \cdot k, & 1 \le k \le 3 \\ \left(\frac{k}{6}\right)^2 \exp(k^2), & k > 3 \end{cases}$$

and the scaling of the tracking error contribution to the switching criteria resulted in the Type II controller being switched to as required, but only when the initial controller was of Type I.

These results show that the performance of this switching control method is heavily dependent on the choice of the bounding function *f(k)*. A different function would have to be used with each type of reference signal to select the correct controller, thus implying that the setpoint type has to be known *a priori*. However, this is not possible in the case of the application being considered in this work, and although numerous variations of this function were tried, one which ensured correct switching for all setpoint types (irrespective of initial controller choice) was not found. Hence, this method is not robust enough as it stands, and extensive redesign would be required to further modify it for use with the pattern generation application.

To summarise, the following problems were identified when using this switching control method to implement the HCS:

1. no well-formulated guidelines for the choice of the bounding function *f(k)* exist;
2. switching ceases in finite time, unless the index *k* is reset to 1 after some suitable interval. However, this is not a robust approach;
3. if the index *k* is not reset the switching criteria become progressively more relaxed. Therefore if switching again becomes necessary, it may not be initiated due to the performance indices not exceeding the larger limits;
4. the correct controller is not always selected when switching stops, even when *f(k)* is optimal **[Davison 1995]**;
5. this method is apparently not applicable in the presence of measurement noise **[Davison 1995]**.

4.3 Summary

The switching control methods reviewed here all depend on the often *ad hoc* mathematical formulation of the switching criteria and the associated bounding functions. Such a formulation is far from intuitive, and successful implementations vary with each problem being considered. This was demonstrated by the analysis of the method presented in **[Davison 1995]**, for which a generally applicable bounding function could not be found.

In their present form, these methods are therefore not robust enough to be used with the hypothetical pattern generation application, but despite this some of them could be applied after more extensive analysis and modification. As the techniques discussed here do not rely on lenghty setpoint identification to initiate switching, this could in turn result in shorter reconfiguration intervals for the HCS.

The potentially useful techniques are those described for adaptive tracking with control input constraints **[Davison 1993]**, and supervisory control using a well-defined switching criterion **[Morse 1994]**.

5. Conclusions and Recommendations

In order to identify different types of waveforms some unique features of these signals have to be extracted and used to form clearly separated feature-space regions, one for each waveform. Furthermore, the sets comprising these features must have good discriminatory properties and must therefore be formed using the greatest possible number of *different* features.

Linear combinations of intuitive features such as first and second derivatives cannot form such a set for the sinusoidal, triangular and square wave signals under consideration. Based on such features, tests showed that the waveforms could not be identified with sufficient certainty in the presence of even moderate levels of measurement noise. Hence, to obtain good signal discrimination *non-linear* combinations of *non-intuitive* features need to be formed.

A simple neural network can achieve this, as was shown by the extremely good classification results obtained. For typical levels of measurement noise, the three signal types can be successfully identified over a wide range of signal amplitude and frequency ($\omega_{max} = 0.25 rads^{-1}$). In addition the required network can be processed easily by the slow computer used for real-time control, hence this method fits into the proposed switching control scheme quite well. However, as the training of the neural net is a specialised procedure this system does not lend itself to end-user maintenance unless such a user is familiar with neural networks.

Combination of this signal identification module with the control loops produced a system which automatically switches in different controllers based on the reference set-point. The simulation and real-time results obtained show that this switching is quite smooth, and that the reconfiguration intervals are acceptable in terms of the proposed application although they could be further shortened to improve overall efficiency. Therefore the control objectives were achieved and this switching control system can be

used to track reference signals which change type randomly. Furthermore, although the waveforms which can be tracked are restricted to sinusoidal, triangular and square waves at present, additional signal types can be easily catered for by re-training the neural network and implementing the corresponding feedback control loops.

Investigation of alternative switching control schemes shows that such methods are invariably designed to track a single fixed setpoint and that the formulation of the switching criteria is often an *ad hoc* process. Extension of these methods for use with the hypothetical pattern generation application is therefore far from trivial. Despite this, the techniques proposed in **[Morse 1994]** and **[Davison 1993]** can potentially be adapted to multiple-setpoint tracking and, because they could substantially shorten the reconfiguration interval by eliminating the signal identification module, merit further study.

In general, the Hybrid Control System designed here successfully implements the proposed switching control scheme for a servo motor. The method is intuitive, robust in the presence of measurement noise, and much simpler to implement than other currently available techniques. It will also be relatively easy to extend it to the entire pattern generation system.

5.1 Recommendations

Although the switching control system performed to expectations and achieved its objectives, there is some scope for further improvement. In view of this, the following recommendations are made:

1. The reconfiguration intervals should be reduced further, perhaps by reducing the time required to identify a particular signal. Ultimately, this will improve the efficiency of the pattern generation system by minimising the delay necessitated by the time taken to achieve asymptotic tracking.

2. The alternative switching control schemes of **[Morse 1994]** and **[Davison 1993]** should be adapted and implemented for multiple setpoints. As these methods continually monitor the performance of the closed-loop system and do not rely on setpoint identification, they should simplify the overall tracking system and improve its performance.

Bibliography

[Antsaklis 1991] : Antsaklis P.J., Sartori M.A.; *A Gaussian Neural Network Implementation for Control Scheduling*, Proceedings IEEE International Symposium on Intelligent Control, August 1991.

[Antsaklis 1993] : Antsaklis, P.J. & Taylor, J.G. (Editors), *Control Theory Approach*, Mathematical Approaches to Neural Networks, Elsevier Science Publishers B.V., 1993.

[Antsaklis 1994] : Antsaklis P.J., *Defining Intelligent Control - Report of the Task Force on Intelligent Control*, IEEE Control Systems Magazine, June 1994, pp. 4-5 & 58-66.

[Antsaklis 1995] : Antsaklis P.J., *Intelligent Learning Control*, IEEE Control Systems Magazine, June 1995, pp. 5-7.

[Askew 1993] : Askew C., et al.; *A Neural Network Pattern Classification Approach for Payload Adaptive Regulation of Flexible Manipulators*, Proceedings of the American Control Conference, June 1993, pp. 2518-2519.

[Bencze 1995] : Bencze W.J., Franklin G.F.; *A Separation Principle for Hybrid Control System Design*, IEEE Control Systems Magazine, Vol.15, No.2, April 1995, pp. 80-85.

[Braae 1994] : Braae M.; *Control Theory for Electrical Engineers*, UCT Press, 1994.

[Chen 1982] : Chen C.H.; *Digital Waveform Processing and Recognition*, CRC Press Inc., 1982.

[Davison 1992] : Davison E.J., Miller D.E.; *An Adaptive Tracking Problem*, IJACSP, Vol.6, 1992, pp. 45-63.

[Davison 1993] : Davison E.J., Miller D.E.; *An Adaptive Tracking Problem with a Control Input Constraint*, Automatica, Vol.29, No.4, 1993, pp. 877-887.

[Davison 1994] : Davison E.J., Chang M.; *Control of Unknown Systems using Switching Controllers: An Experimental Study*, Proceedings of the American Control Conference, June 1994, pp. 2984-2989.

[Davison 1995] : Davison E.J., Chang M.; *Switching Control of a Family of Plants*, Proceedings of the American Control Conference, June 1995, pp. 1015-1020.

[Garcia 1991] : Garcia H.E., Ray A., Edwards R.M.; *Reconfigurable Control of Power Plants Using Learning Automata*, IEEE Control Systems Magazine, January 1991, pp. 85-92.

[Koivo 1994] : Koivo H.N.; *Artificial Neural Networks in Fault Diagnosis and* Control, Control Engineering Practice, Vol.2, No.1, February 1994, pp. 89-101.

[Lippmann 1987] : Lippmann R.P.; *An Introduction to Computing with Neural Nets*, IEEE ASSP Magazine, April 1987, pp. 4-22.

[Matsuoka 1992] : Matsuoka K.; *Noise Injection into Inputs in Back-Propagation Learning*, IEEE Trans. SMC, Vol.22, No.3, 1992, pp. 436-440.

[Morrison 1991] : Morrison N.; *Introduction to Fourier Analysis*, Department of Applied Mathematics, University of Cape Town, 1991.

[Morse 1994] : Morse A.S.; *Supervisory Control of Families of Linear Setpoint Controllers - Part 1: Exact Matching*, To be published.

[Narendra 1992] : Narendra K.S., Mukhopadhyay S.; *Intelligent Control Using Neural Networks*, IEEE Control Systems Magazine, April 1992, pp. 11-18.

[Narożny 1995] : Narożny J.A.; *Reconfigurable Control of a Position Servo in a Pattern Cutting Application*, Proceedings of the International Conference on Engineering Applications of Neural Networks, Helsinki, Finland, August 1995, pp. 185-188.

[Niesler 1993] : Niesler T.R.; *Time-optimal Control by means of Neural Networks*, M.Sc. Thesis, University of Stellenbosch, 1993.

[NumRec 1992] : Press W.H., et al.; *Numerical Recipes in C (2nd Edition)*, Cambridge University Press, 1992.

[Oppenheim 1989] : Oppenheim A.V., Schafer R.W.; *Discrete-Time Signal Processing*, Prentice-Hall, Inc., 1989.

[Trossbach 1994] : Trossbach W.; *Neural Networks in Control Engineering*, M.Sc. Thesis, University of Cape Town, June 1994.

Appendix A. Neural Network Structures

Due to the many different approaches to the study of artificial intelligence, several neural network structures and training algorithms exist. Each of these has its particular advantages and disadvantages which make it well suited to a specific area of application.

Over the last few years, several authors have attempted to provide a concise summary of the more successful representations, both from a general viewpoint **[Lippmann 1987]**, and one specifically geared toward Control Engineering **[Antsaklis 1993]** **[Trossbach 1994]** **[Koivo 1994]**. This appendix will present a review of some of the more relevant structures described in this literature, describing their size and consequent storage requirements, the complexity of their associated training algorithms, and their applications. The discussion is not intended to be exhaustive, and more detailed information on the individual approaches can be found in the references quoted above.

A.1. Kohonen Feature Maps

As their name suggests, this class of networks is a self-organising, unsupervised feature classifier. Originally proposed by Kohonen in 1980, the nets have found use as vector quantisers and clustering methods. Good performance in the presence of noise has been reported, mainly due to slow weight adaptation during training and the *off-line* nature of the training itself, which precludes weight changes after training has completed.

Kohonen nets consist of an input layer of nodes corresponding to the number of elements in the input pattern, and a *feature map* consisting of *mapping elements* which are each connected to every input node by a *weighted link*. Training is an unsupervised iterative process, during which the weights are adapted in order to map different input patterns into unique regions of the feature map. Such *clustering* does not make use of a desired output pattern as with most other training algorithms. Instead, the network itself 'decides' on the output for a specific input pattern, and if patterns are sufficiently close may assign them to the same cluster.

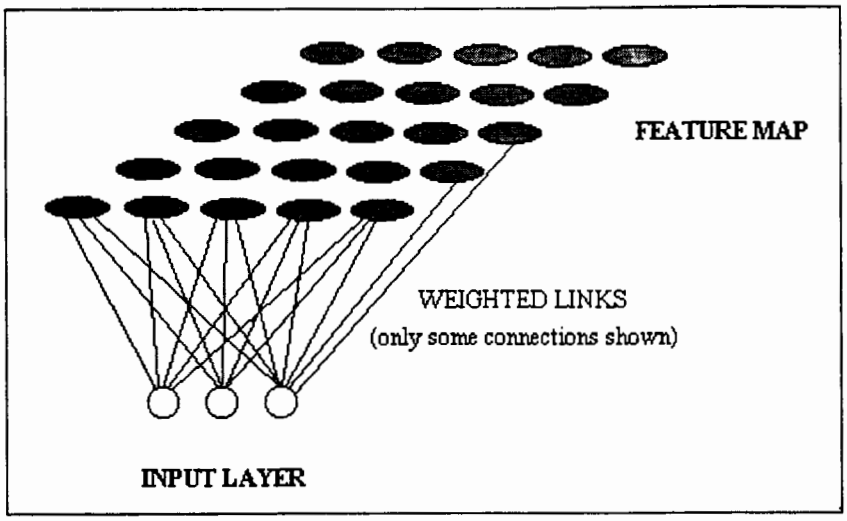


Figure A - 1. Kohonen feature map structure.

For reliable classification, though, the number of classes that the presented patterns must fall into has to be known in advance. Based on this, the experienced designer can make a rough estimate of the required number of feature layer elements. Even then, complete classification is not always robust, with the alternative being to increase the number of Kohonen (feature) layer elements in the network. This further increases an already large storage requirement, and slows down network training, itself a complex process.

A.2. Cerebellar Model Articulation Controller (CMAC)

Introduced in 1975 by Albus, CMAC networks have been gaining popularity in robotics and signal, speech and image processing. Their appeal stems from a fast training algorithm and the fact that hardware implementation is relatively easy using logic cell arrays. Furthermore, this class of neural networks accepts real-valued numbers as input and outputs also real numbers, as well as exhibiting local generalisation properties, all of which make it useful for classification.

The CMAC approach bears some similarity to that of the Kohonen feature maps outlined previously, in that the input is also quantised and mapped to a number (C) of consecutive *association cells*. Each association cell is excited by a number of input levels, and this

overlap determines the generalisation properties of the network. Every association cell is assigned an address which is then mapped by some Hash coding to a memory location where the *weights* are stored. These addresses are intended to be unique, but collisions do occur when two cells are mapped to the same memory location. The weights of all active association cells are then summed to produce the output.

CMAC weights are determined by the training algorithm, typically a supervised learning method like Least Mean Squares. Here, however, CMAC departs further from Kohonen maps in that it is not *self-organising*. A *desired output* is used to determine the weight changes during training, and this indicates which association cells are to be activated by a specific pattern. All other parameters, such as C, overlap and size of memory, are fixed in advance by the designer.

These networks are currently being used to model dynamic systems, in the implementation of fuzzy controllers, and for classification purposes. Their biggest drawbacks to date are the storage space required (in a hardware implementation this is not so critical), and problems caused by address mapping collisions.

A.3. Multi-layer Perceptrons (MLPs)

This neural net representation is by far the most popular for control applications, and it has been extensively used and studied over many years. Usual implementations show it as consisting of an input layer, a number of hidden layers, and an output layer. Nodes are interconnected by weighted links in a *feedforward* topology, with varying activation functions assigned to the nodes of each layer.

MLPs are mostly trained with variations of the popular Backpropagation algorithm, which adapts the weights according to a gradient search method. An input pattern is presented to the network, which then propagates it forward through the hidden layer(s) to produce an output. The mean square of the difference between this and the desired output is then used to change the network weights. The weights are adapted by layers, from

the output layer back to the input layer. In this way, the mean square error is *backpropagated* through the network. Training stops when the error drops below some acceptable maximum value. A detailed discussion of the full algorithm is presented in Appendix B.

The choice of the number of hidden layers to use, and the number of nodes in those layers, is a difficult one and is often done on a trial-and-error basis for each specific application. Network training parameters are treated in a similar manner. Despite this, the network has been successfully used for modelling, function approximation and pattern classification, handles noisy inputs well, and has relatively small storage requirements. Furthermore, the training algorithm is by and large quick, and has a proven record of reliable convergence.

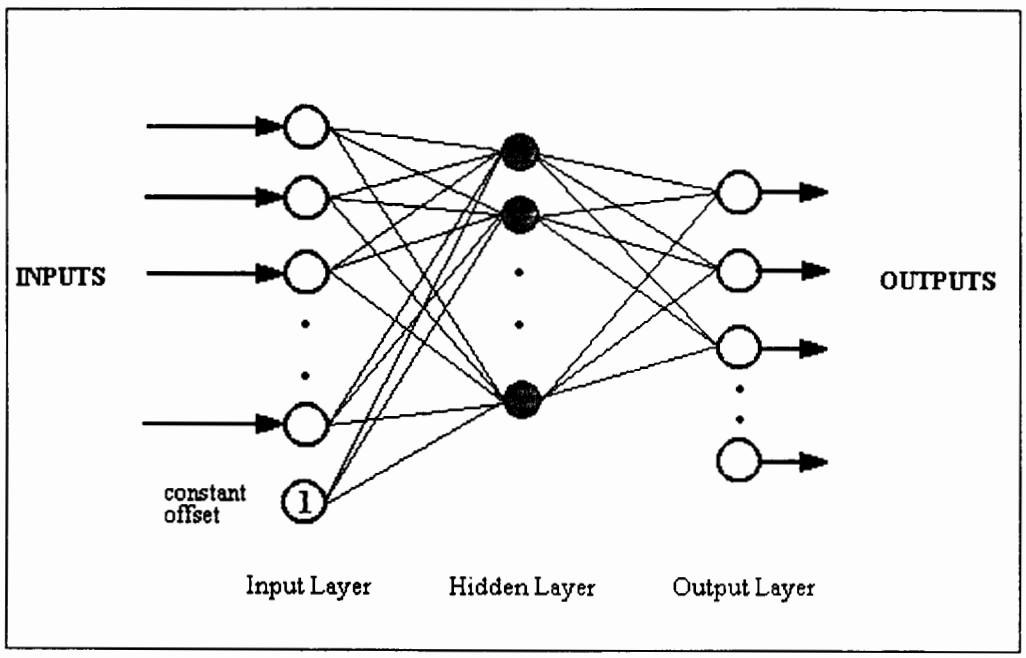


Figure A - 2. Structure of a 3-layer perceptron.

A.4. Radial Basis Function (RBF) Networks

RBF networks are also feedforward networks, with a structure similar to the MLP. However, they use hidden layer activation functions which exhibit local behaviour (e.g. Gaussian function). This makes RBF neural nets the preferred choice for pattern classification problems, where they have been extensively applied **[Trossbach 1994]**.

The different activation functions entail additional training parameter, such as the centres and widths of the Gaussian functions, and this complicates the training algorithm. As a result, training does not always lead to convergence to the minimum of the error function, and the networks have problems obtaining classifications. In addition, the local action of the nodes means that more of them are required for a specific mapping, and so RBF networks require large amounts of storage space.

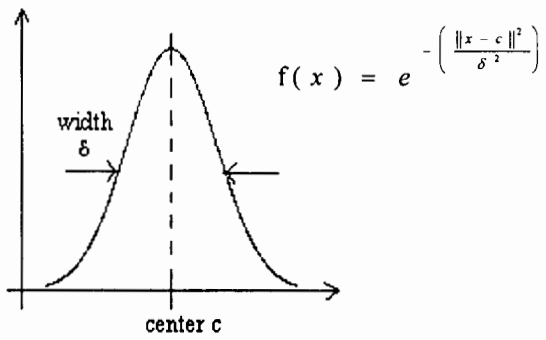


Figure A - 3. The Gaussian activation function used in RBF networks.

Appendix B. The Backpropagation Training Algorithm

The Backpropagation algorithm was first suggested by Rumelhart et al. as a supervised training method for the feedforward Multilayer Perceptron (MLP) neural network model **[Lippmann 1987]**. Since the algorithm's original publication, several variants have appeared, with the majority using modified activation functions, register shifts instead of multiplication, or progressively modified learning parameters. The goals of these changes are quicker convergence to the global minimum of the objective function, an increased immunity to the problem of local minima, and faster network processing for larger structures.

This section describes Rumelhart's standard Backpropagation training algorithm for neural networks, as well as a version of it with progressively modified training parameters. The latter was used by the author in the training of the neural network implemented as the RBT, or waveform classifier, for the hypothetical pattern generation application.

B.1. Standard Backpropagation

This is the original algorithm proposed by Rumelhart et al. in 1986, and implemented in Pascal by Trossbach **[Trossbach 1994]**. It is an iterative gradient descent algorithm designed to minimise an objective function consisting of the root mean square error between the actual output of a MLP and the desired output. In the search for the global minimum, the algorithm traces a path along this *error surface*, ideally in the direction of steepest descent. Convergence depends, to a large extent, on the choices made for the initial values of several parameters, which is often an *ad hoc* process requiring an in depth knowledge of the training method itself.

The algorithm assumes a network whose nodes are arranged into sequentially connected layers, numbered from 0 to M, with one output layer [M], one input layer [0], and one or more hidden layers [1 - (M-1)]. It requires continuous differentiable non-linearities as activation functions, such as the *sigmoidal* non-linearity often used in the hidden nodes:

$$f(x) = \frac{1}{1+e^{-x}}$$

with first order derivative:

$$f'(x) = \frac{e^{-x}}{\left(1+e^{-x}\right)^2} = f(x)\cdot\left[1-f(x)\right]$$

The steps of the algorithm are:

1. Initialise all network weights to small random numbers, and other training parameters to pre-selected values.
2. Present the input pattern to the input nodes of the network, and set the corresponding desired network outputs.
3. For the current pattern, calculate each node's input and output (input layer nodes have their inputs set in step 2 above):

$$x_k^s = \sum_{j=1}^{N_{(s-1)}} w_{kj}^s \cdot y_j^{(s-1)} \qquad s = 1..M;$$

$$y_k^s = f\left(x_k^s\right) \qquad s = 0..M;$$

where:

s	=	the generic layer number
N_S	=	number of nodes in layer s
k	=	index of node in layer s
j	=	index of node in layer *(s-1)*
x_k^S	=	input to node k in layer s
y_k^S	=	output of node k in layer s

w_{kj}^s = weight of the link between node *j* in layer *(s-1)* and node *k* in layer *s*

4. Adapt the weights of the connections between the network layers. Initially, for all nodes *k* in each layer *s*, starting at the output layer and working back, compute:

$$\delta_k^s = \mathrm{f}'\left(x_k^s\right) \cdot \begin{cases} d_k - y_k^s & \mathrm{s} = \mathrm{M}; \\ \sum_{j=1}^{N_{(s+1)}} w_{jk}^{(s+1)} \cdot \delta_j^{(s+1)} & \mathrm{s} = 1..(\mathrm{M}-1); \end{cases}$$

Next, starting at the output nodes and working back to the first hidden layer (*s*=1), calculate the required weight changes:

$$\Delta w_{kj}^s = \eta \cdot \delta_k^s \cdot y_j^{(s-1)} + \alpha \cdot \left(w_{kj}^s(t) - w_{kj}^s(t-1)\right) \quad \begin{cases} \mathrm{k} = 1..\mathrm{N}_s; \\ \mathrm{j} = 1..\mathrm{N}_{(s-1)}; \end{cases}$$

If the weights are to be updated at this instance, then:

$$w_{kj}^s(t+1) = w_{kj}^s(t) + \Delta w_{kj}^s$$

where:

η = learning rate, or gain term

α = momentum term, added to speed up convergence and smooth weight changes ($0 < \alpha < 1$)

t = current time index

δ_k^s = error term for node *k* in layer *s*

d_k = desired output for node *k*

5. if the *root mean square* (rms.) error between the desired and actual network outputs is less than a desired threshold, then end, otherwise go to step 2 with a new network input-output pattern

B.2. Modified Backpropagation

The actual training algorithm used by the author progressively modified the η and α training parameters in order to achieve faster convergence to the global minimum of the objective function.

The training of the neural network was broken up into two phases; a coarse, quick phase, followed by a finer, slow one. During the coarse phase, the learning rate η and the momentum term α were kept fairly big, to enable large steps to be taken along the error surface and allow the search process to escape from local minima "troughs". Once the rms. error fell below a certain threshold, the values of η and α were reduced, so permitting finer steps to be taken along the error surface in search of the global minimum.
This assumed that after the initial coarse phase the general location of the global minimum had been identified by the algorithm. This assumption was not always valid, as indicated by the need to use several values of α and η on a trial-and-error basis until a combination that produced convergence was achieved.

A *pattern-learning* approach was used with Rumelhart's algorithm, to enable higher learning rates to be used than in the corresponding *batch-learning* approach, while still maintaining algorithm stability **[Trossbach 1994]**. This involved updating the connection weights after the presentation of **each** network input-output training pattern, with the patterns presented in random order to improve the network's generalisation capabilities.

The standard Backpropagation algorithm was modified in the following way:

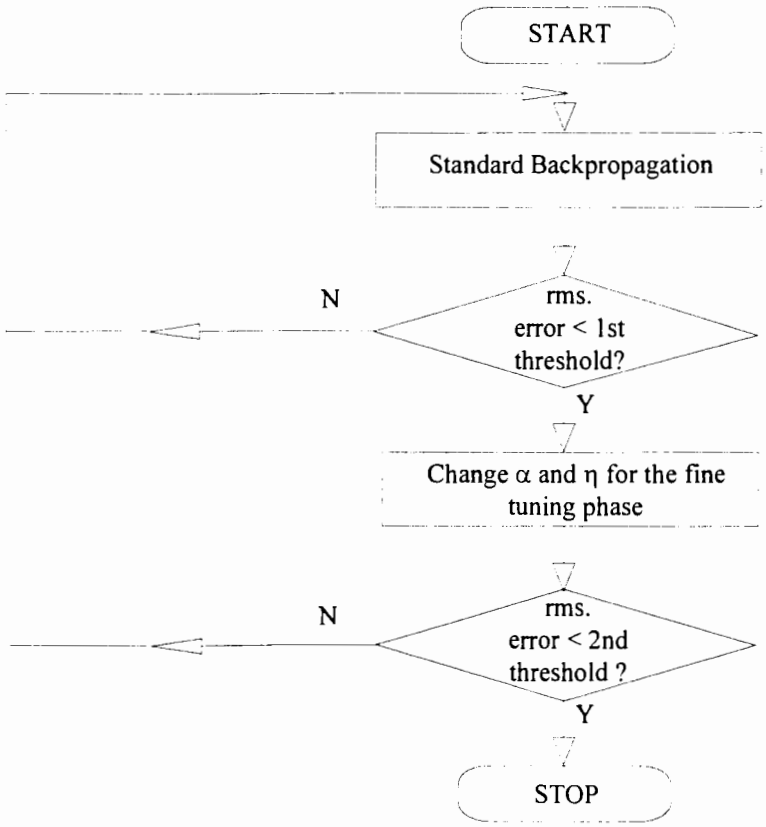


Figure B - 1. Outline of modified Backpropagation algorithm.

This standard training algorithm was coded in the unit FFNETS.PAS, and the modified version in the program NN_TRN_C.PAS, both of which can be found on the attached diskette.

Appendix C. The Savitzky-Golay Smoothing Filter

Data corrupted by measurement noise needs to be smoothed in order to extract the underlying function. A *low-pass* filter is typically used for this, and the Savitzky-Golay smoothing filter is a discrete implementation of one. This appendix summarises the discussion of Savitzky-Golay filters presented in **[NumRec 1992]**, pp.650-655. The Pascal code implementation of this filter (a direct translation of the C source found in **[NumRec 1992]**) can be found in the unit JN_UTILS.PAS on the attached diskette.

C.1. The Theoretical Aspect

Let a series of equally spaced data values be $f_i \equiv f(t_i)$, where $t_i = t_0 + i\Delta$, Δ is the sample interval and $i = ...,-2,-1,0,1,2,....$ The simplest type of digital filter replaces each f_i by a linear combination g_i of itself and some number of nearby points which together form a "moving window".

$$g_i = \sum_{n=-n_L}^{n_R} c_n f_{i+n} \qquad \begin{cases} n_L \text{ is the number of data points preceding } f_i \text{ in time;} \\ n_R \text{ is the number of data points succeeding } f_i \text{ in time;} \\ c_n \text{ is the n}^{\text{th}} \text{ filter coefficient} \end{cases} \qquad ...(1)$$

A causal filter would have $n_R = 0$, for example.

If the underlying function is constant or linearly decreasing or increasing with time, a "moving average" filter will suffice ($c_n = (1+n_R+n_L)^{-1}$ then). However, if as in our case the underlying function has a nonzero second derivative, it may need to be approximated by polynomials of higher degree and therefore c_n cannot be constant for all f_i.

Thus for each point f_i, a polynomial $p(i)$ of degree M in i is fitted in the least-squares sense to all $(1+n_R+n_L)$ points in the moving window containing f_i. Then $g_i=p(i)$ gives the necessary approximation.

However, this can be computationally demanding if done each time for all data points f_i, and is therefore an impractical approach. The Savitzky-Golay filter overcomes this problem by utilising a property of least-squares fitting which implies that the coefficients of a fitted polynomial are themselves linear in the value of the data **[NumRec 1992]**. This means that all the fitting can be done *in advance* for fictitious data consisting of all zeros except for a single 1, and then a fit to the real data can be obtained simply by taking linear combinations of the coefficients so derived.

The standard least-squares design matrix for this problem is:

$$\mathbf{A}_{ij} = i^j \quad \begin{cases} i = -n_L, \ldots, n_R; \\ j = 0, \ldots, M; \\ M = \text{ order of fitting polynomial;} \end{cases}$$

which gives the filter coefficients according to:

$$c_n = \sum_{m=0}^{M} \left[\left(\mathbf{A}^{\mathrm{T}} \cdot \mathbf{A} \right)^{-1} \right]_{0m} n^m \quad \{ -n_L \le n \le n_R; \qquad \ldots(2)$$

These coefficients are computed once at the start of a signal processing routine, and subsequently used in (1) whenever the filter is applied to a set of data.

In the computer implementation of equation (2) **[NumRec 1992]**, an additional parameter *ld* is used to set whether the returned coefficients will produce the best-fit to the underlying function (*ld=0*) or to its k^{th} derivative (*ld=k*). The selection of *m* is specific to each application, although *m*=2 or *m*=4 are the usual choices; a larger value of *m* increases the cut-off frequency of the filter. Finally, *m* must satisfy:

$$m \le (n_L + n_R);$$
$$m \ge ld;$$

The need to compute its coefficients only once during its application gives the Savitzky-Golay filter an advantage over other digital filters and inspired its selection for this study. Using this filter, smoothing of the noisy reference signal will be fast and nearly

optimal and will not adversely affect the setpoint tracking system for which signal processing time must be minimised in order to achieve adequate real-time control.

C.2. Practical Implementation

For the purposes of this application, a best-fit to noisy samples of sinusoidal, triangular and square wave signals is required. Furthermore, the filter must be *causal* in order to be realisable.

During tests of the filter, it was found that a first order polynomial approximation (m=1) produced the best results for all the functions under consideration[12], and that 10 data points were sufficient for a good fit. To achieve consistently good filtering, this was ultimately increased to 30 data points. Hence the following parameters were used to obtain the coefficients c_n with the *savgol* function defined in **[NumRec 1992]**:

$$\begin{aligned} n_L &= 30 \\ n_R &= 0 \\ ld &= 0 \\ m &= 1 \end{aligned}$$

A *fitted* point was then obtained using the function *fit_data_1* which implemented the required causal combination of coefficients c_n and data points f_i, as described in **[NumRec 1992]**. The function *fit_data_1* is defined in the unit JN_UTILS.PAS which is included on the attached diskette.

[12] The polynomial was fitted to very small subsections of each waveform at a time.

Appendix D. Results of Neural Net Classification Tests

This appendix lists the results obtained from classification tests performed on Multilayer Perceptron (MLP) neural networks of several different sizes. For each network size, details of the classification performance are given and the trends in the results for that structure are discussed. Furthermore, some general remarks applicable to the results for all the cases considered here are presented at the conclusion of this chapter.

In the following sections, *Identification Errors* are cases where the network misclassified the presented waveform as being of another type, and *Indecisions* are cases where the net could not decide conclusively on the type of the presented waveform. *Successful Identifications* do not fall into either of these categories, and *SNR* is the Signal-to-Noise Ratio in decibels [dB].

D.1. MLP with 75 hidden nodes

As can be seen from Figure D - 1 below, extremely good classification was obtained with this network. The waveforms were almost always identified correctly for SNRs above 20.0dB (the *region of interest*), a range which covered the expected SNR of 25.0dB more than adequately. At higher noise levels, performance decreased for the sinusoidal and the triangular waveforms. This was expected, since the net was not trained at such high noise levels and would not be able to generalise to the required extent. The square wave, on the other hand, was classified perfectly irrespective of the noise level present.

Inspection of Table D - 1 further shows that until a SNR below 9.1dB, unsuccessful classifications were not due to identification errors. Therefore the network did not confuse waveforms over a noise region exceeding the *region of interest*, and hence is quite robust and reliable. Such performance indicates that this particular implementation of the MLP is suited to the task of identifying the waveforms.

SNR	Sinusoidal Wave				Triangular Wave				Square Wave			
[dB]	*Identification Attempts*	*Identification Errors*	*Indecisions*	*%Successful Identifications*	*Identification Attempts*	*Identification Errors*	*Indecisions*	*%Successful Identifications*	*Identification Attempts*	*Identification Errors*	*Indecisions*	*%Successful Identifications*
∞	1190	0	0	100.00	1190	0	0	100.00	1190	0	0	100.00
26.0	1190	0	0	100.00	1190	0	0	100.00	1190	0	0	100.00
20.0	1190	0	32	97.31	1190	0	7	99.41	1190	0	0	100.00
16.5	1190	0	124	89.58	1190	0	66	94.45	1190	0	0	100.00
14.0	1190	0	232	80.50	1190	0	144	87.90	1190	0	0	100.00
12.0	1190	0	295	75.21	1190	0	199	83.28	1190	0	0	100.00
10.5	1190	0	382	67.90	1190	0	261	78.07	1190	0	0	100.00
9.1	1190	0	418	64.87	1190	0	360	69.75	1190	0	0	100.00
8.0	1190	1	487	58.99	1192	2	418	64.77	1190	0	0	100.00
6.9	1190	3	509	56.97	1191	3	429	63.73	1190	0	0	100.00
6.0	1190	7	573	51.26	1193	5	513	56.58	1190	0	0	100.00

Table D - 1. Classification results: MLP with 75 hidden nodes.

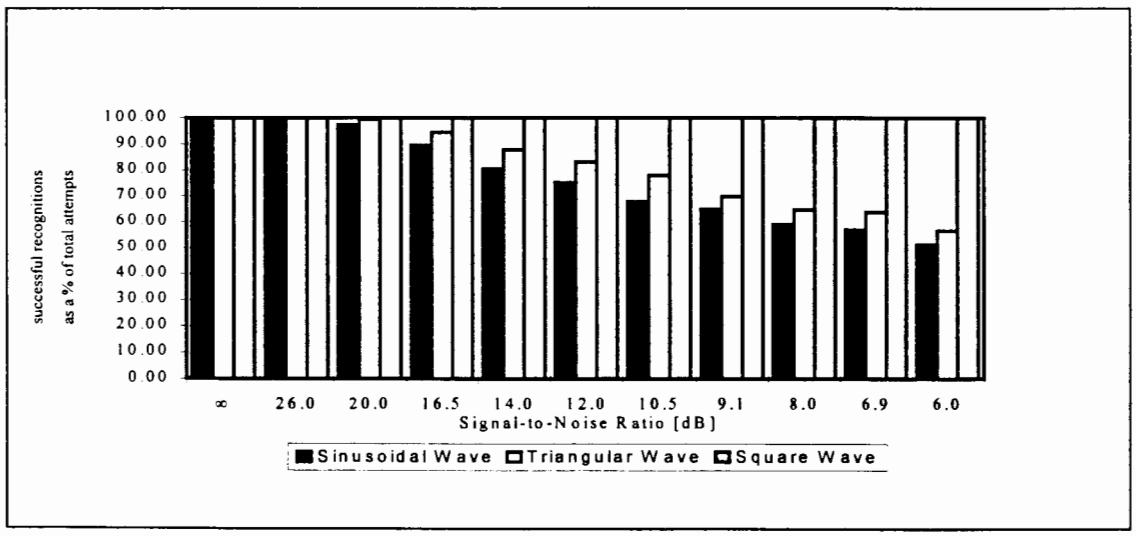


Figure D - 1. Signal recognition rates: 75 hidden nodes.

D.2. MLP with 40 hidden nodes, small α_{final} and η_{final}[13]

Very poor performance was obtained with this MLP structure, although the square wave was again correctly identified at all noise levels (Figure D - 2). For the sinusoidal wave-form in particular, bad results were produced even without noise being present in the signal. Inspection of Table D - 2 shows that this was caused by network indecision though, and not misclassifications. As similar behaviour is seen for the triangular wave, it can be surmised that the network did not form clearly separated feature-space regions

[13] α and η are learning algorithm parameters.

for these two waveforms. Instead the regions overlap, causing the network to produce a 'high' output at more than one output node when presented with either waveform.

In this case, the failure to form separable feature-space regions can be attributed to insufficient network training. As the length of the learning process was found to be fairly consistent during several runs with this network, it can be concluded that training was terminated prematurely due to convergence to a *local* minimum of the objective function. Hence this network is in fact not fully trained and will produce unreliable performance. As such, it should not be used for the classification task.

SNR	Sinusoidal Wave				Triangular Wave				Square Wave			
[dB]	*Identification Attempts*	*Identification Errors*	*Indecisions*	*%Successful Identifications*	*Identification Attempts*	*Identification Errors*	*Indecisions*	*%Successful Identifications*	*Identification Attempts*	*Identification Errors*	*Indecisions*	*%Successful Identifications*
∞	1190	0	490	58.82	1190	0	0	100.00	1190	0	0	100.00
26.0	1190	0	223	81.26	1190	0	86	92.77	1190	0	0	100.00
20.0	1190	0	306	74.29	1190	0	235	80.25	1190	0	0	100.00
16.5	1190	0	377	68.32	1190	0	318	73.28	1190	0	0	100.00
14.0	1190	0	425	64.29	1190	0	421	64.62	1190	0	0	100.00
12.0	1190	0	478	59.83	1190	0	513	56.89	1190	0	0	100.00
10.5	1190	0	502	57.82	1190	0	564	52.61	1190	0	0	100.00
9.1	1190	0	600	49.58	1191	0	632	46.94	1190	0	0	100.00
8.0	1190	3	591	50.08	1191	0	687	42.32	1190	0	0	100.00
6.9	1190	3	634	46.47	1190	1	749	36.97	1190	0	0	100.00
6.0	1190	10	617	47.31	1191	6	794	32.83	1190	0	0	100.00

Table D - 2. Classification results: MLP with 40 hidden nodes, small α_{final} and η_{final}.

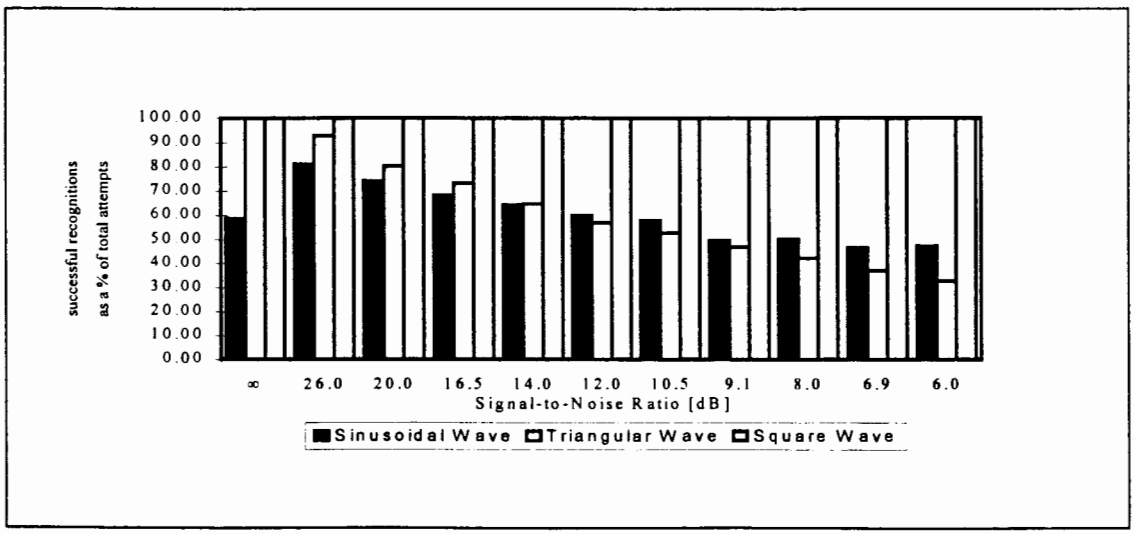


Figure D - 2. Signal recognition rates: 40 hidden nodes, small α_{final} and η_{final}.

D.3. MLP with 40 hidden nodes, large α_{final} and η_{final}

This version of the network considered in §D.2 produced classification performance comparable to and slightly better than that of the 75 node network discussed in §D.1. The square waveform was again correctly recognised under all noise conditions, and the reliability shown by the results in Figure D - 3 and Table D - 3 makes this network very suitable for waveform identification.

SNR	Sinusoidal Wave				Triangular Wave				Square Wave			
[dB]	*Identification Attempts*	*Identification Errors*	*Indecisions*	*%Successful Identifications*	*Identification Attempts*	*Identification Errors*	*Indecisions*	*%Successful Identifications*	*Identification Attempts*	*Identification Errors*	*Indecisions*	*%Successful Identifications*
∞	1190	0	0	100.00	1190	0	0	100.00	1190	0	0	100.00
26.0	1190	0	0	100.00	1190	0	0	100.00	1190	0	0	100.00
20.0	1190	0	12	98.99	1190	0	19	98.40	1190	0	0	100.00
16.5	1190	0	86	92.77	1190	0	78	93.45	1190	0	0	100.00
14.0	1190	0	193	83.78	1190	0	164	86.22	1190	0	0	100.00
12.0	1190	0	247	79.24	1190	0	262	77.98	1190	0	0	100.00
10.5	1190	0	304	74.45	1190	0	369	68.99	1190	0	0	100.00
9.1	1190	0	383	67.82	1192	0	455	61.83	1190	0	0	100.00
8.0	1190	0	446	62.52	1191	2	486	59.03	1190	0	0	100.00
6.9	1190	3	542	54.20	1192	2	576	51.51	1190	0	0	100.00
6.0	1190	7	526	55.21	1193	5	616	47.95	1190	0	0	100.00

Table D - 3. Classification results: MLP with 40 hidden nodes, large α_{final} and η_{final}.

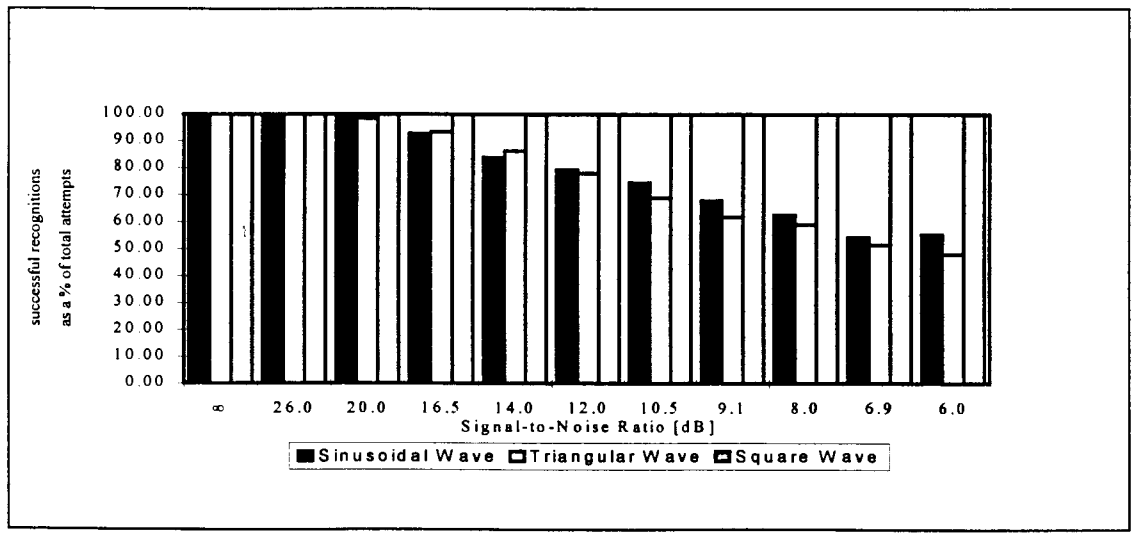


Figure D - 3. Signal recognition rates: 40 hidden nodes, large α_{final} and η_{final}.

D.4. MLP with 20 hidden nodes

A relatively small structure, this network nevertheless gave classification results which were virtually identical to those observed in §D.3 above. The square wave was again identified correctly at all noise levels, and the network proved to be robust, reliable and well suited to the waveform identification task. Table D - 4 and Figure D - 4 below present the classification results for this network.

SNR [dB]	Sinusoidal Wave				Triangular Wave				Square Wave			
	Identification Attempts	*Identification Errors*	*Indecisions*	*%Successful Identifications*	*Identification Attempts*	*Identification Errors*	*Indecisions*	*%Successful Identifications*	*Identification Attempts*	*Identification Errors*	*Indecisions*	*%Successful Identifications*
∞	1190	0	0	100.00	1190	0	0	100.00	1190	0	0	100.00
26.0	1190	0	0	100.00	1190	0	0	100.00	1190	0	0	100.00
20.0	1190	0	15	98.74	1190	0	3	99.75	1190	0	0	100.00
16.5	1190	0	82	93.11	1190	0	48	95.97	1190	0	0	100.00
14.0	1190	0	190	84.03	1190	0	124	89.58	1190	0	0	100.00
12.0	1190	0	223	81.26	1190	0	242	79.66	1190	0	0	100.00
10.5	1190	0	326	72.61	1190	0	329	72.35	1190	0	0	100.00
9.1	1190	0	395	66.81	1190	0	435	63.45	1190	0	0	100.00
8.0	1191	3	438	62.97	1192	1	476	59.98	1190	0	0	100.00
6.9	1191	3	476	59.78	1190	3	568	52.02	1190	0	0	100.00
6.0	1190	10	540	53.78	1191	4	596	49.62	1190	0	0	100.00

Table D - 4. Classification results: MLP with 20 hidden nodes.

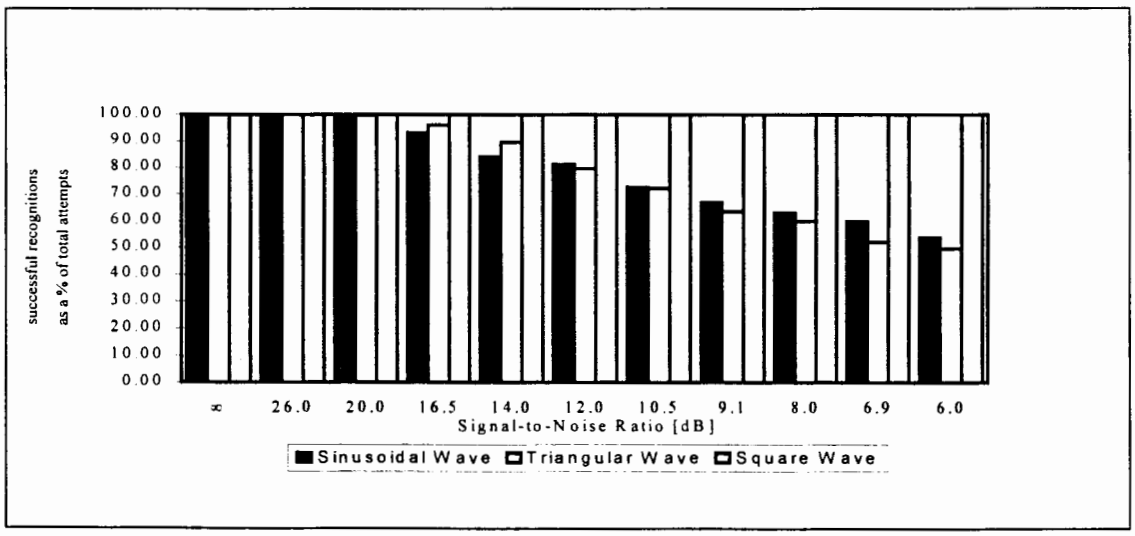


Figure D - 4. Signal recognition rates: 20 hidden nodes.

D.5. General Remarks on the Results

Small incidence of signal misclassification

A global trend evident from the presented results is that all the networks put through the classification test displayed very minimal instances of actual misclassification, and then only at SNRs below 9.1dB (see the *Identification Errors* columns in the respective Classification Results tables above). This behaviour is very important in this application, as it means that at the expected SNR the chances of the network mistakenly identifying a waveform as being of another type are very small. Hence it is unlikely that such a network would cause spurious controller reconfiguration, which would lead to system instability.

Reasons for misclassifications and for inconclusive results

Although most[14] of the neural network classifiers considered here exhibited exceptional behaviour during testing, there were instances when such a network either could not classify a signal or else classified it as belonging to the wrong category. The reasons for this are set out below:

1. *Excessive levels of signal noise*:- the most errors and inconclusive results were observed at fairly high noise levels (SNR < 12.0dB), when the network's generalisation capabilities were at their limit. The network was trained on data with a SNR of 20.0dB, so would exhibit best generalisation in the region of that value;
2. *Waveform sampling using the wrong fitting interval*:- this can occur if the waveform period is measured in a particular quarter-period section and the fitting interval *f_dt* is set accordingly, but then the signal type or period is changed before or during the subsequent sampling process. The resultant samples may not be of a quarter-period section and will form an "arbitrary" shape. Such a shape may resemble either one of the other known signal types

[14] Although these reasons can also apply to the neural network of §D.2, that net's poor classification performance is due mainly to it being insufficiently trained, as discussed in that Section.

(misclassification) or else none that the network was trained on (inconclusive result);

3. *The similarity between the sinusoidal and the triangular waveforms*:- from the point of view of the *shape* of the waveforms (which are amplitude and frequency normalised), these two types can be very similar, leading to some confusion. This is especially the case in the presence of high signal noise levels, when the network may be confused between the two. A misclassification would then occur;
4. *Attempt to identify an arbitrary waveform*:- if the signal that the network is to identify is not of the type it was trained on (square, triangular, sinusoidal), the network will not be able to classify it and will return an inconclusive result. (However, if the arbitrary waveform is *structurally* similar to one of the three recognised waveforms, it will be classified as such.)

Perfect classification of the square wave

For each of the noise levels used during the tests, all of the tested networks correctly identified the square wave. This performance was expected, and the reasons for it are based on the fact that the square wave is substantially different in *shape* from the sinusoidal and the triangular waveforms.

Samples of the square wave occupy a much smaller *subspace* of the input space, centred around a relatively constant value (e.g. 0.8 Volts) even at high noise levels. In contrast, the corresponding samples of the other two signals always span the entire input space (0.0–1.0 Volts). Furthermore, due to the nature of the best-fit filter used to smooth the waveforms, this *subspace* is made even smaller after filtering.

The neural network classifies input patterns based on the feature-space regions they fall into. Both the size and the location of these regions in the feature-space depend to an extent on the size and location (in the input space) of the original input *subspace*. Hence, for a given noise level the feature-space region corresponding to the square wave is not only smaller than, but also located further away from both the sinusoidal wave region and the triangular wave region. This in turn means that it is easier for the neural network

to form clearly separating hyperplanes between the square wave region and the other two feature-space regions, thereby greatly increasing the probability of correctly classifying square wave samples.

Appendix E. Miscellaneous Derivations and Formulae

E.1. Plant Transfer Function g(s)

The plant under consideration was a simple servo motor, for which a first order representation was required:

$$g(s) = \frac{C}{1 + s \cdot D} \quad \frac{[\text{Volts}]}{[\text{Volts}]}$$

Several values of the parameters C and D were obtained from standard open loop step tests (both step *up* and step *down*) of the plant, and these were then averaged to produce the parameters used in the final model (here given with their standard deviation):

Step Test #	*C [Volts/Volts]*	*D [seconds]*
1	1.80	0.90
2	1.70	0.90
3	1.65	0.70
4	1.65	0.70
Average	**1.70±0.07**	**0.80±0.12**

Table E - 1. Plant step test results.

Therefore, the final plant model used was:

$$g(s) = \frac{1.70}{1 + 0.80 \cdot s} \quad \frac{[\text{Volts}]}{[\text{Volts}]}$$

E.2. Minimum Expected Signal-to-Noise Ratio

The plant output and the reference signal need to be measured at given intervals to monitor the system's performance. Sampling of these signals will introduce some noise due to quantisation errors, and the signals themselves may contain noise. As the level of noise present is important to the performance of the hybrid control system (HCS) and

influences its design, an *expected Signal-to-Noise Ratio (SNR)* was determined. This SNR is an indicator of the average levels of noise that are expected to be present at any one time during sampling of the necessary signals.

The expected SNR was determined by measuring a constant signal of known amplitude with the analogue-to-digital converter (ADC) that will be used with the HCS. The difference between the measured and the known values was used as an approximation of the sampling noise level. Several known signal amplitudes were used, spanning the entire input range of the ADC, and repeated readings were taken for each amplitude. The average measured values for each amplitude are shown in Table E - 2 below, together with the corresponding SNR.

Known Amplitude A_1 *[Volts]*	*Measured Amplitude* A_2 *[Volts]*	*Noise Amplitude* $abs(A_1-A_2)$ *[Volts]*	*SNR* $20log[A_1/abs(A_1-A_2)]$ *[dB]*
1.00	1.01	0.01	40.0
2.00	2.01	0.01	46.0
4.00	4.01	0.01	52.0
6.00	6.03	0.03	46.0
8.00	8.03	0.03	48.5
10.00	9.99	0.01	60.0
Average SNR			**48.8**

Table E - 2. Measured Signal-to-Noise Ratios.

Halving this average SNR to obtain a reasonable margin of error, then rounding for ease of working:

$$\text{expected SNR} = 24.4\text{dB}$$
$$\cong 25.0\text{dB}$$

Hence the HCS was designed for an *expected SNR* of **25.0dB**.

E.3. Statistical Formulae

1. Arithmetic Mean

$$\bar{x} = \frac{\sum_{i=1}^{n} x_i}{n} \quad \begin{cases} n, i \in \mathbf{Z} \\ x_i \in \mathbf{R} \end{cases}$$

2. Median

For a set of n observations x_i which are ranked from 1 to n in order of magnitude,

$$median = \begin{cases} x_{m+1} & \forall\ n = (2 \cdot m + 1),\ m \in \mathbf{Z}; \\ \dfrac{(x_m + x_{m+1})}{2} & \forall\ n = 2 \cdot m,\ m \in \mathbf{Z}; \end{cases}$$

3. Standard deviation

$$\sigma = \sqrt{\frac{\sum_{i=1}^{n} (x_i - \bar{x})^2}{n-1}} \quad \begin{cases} i, n \in \mathbf{Z} \\ x_i \in \mathbf{R} \end{cases}$$

4. Cumulative Sum

$$\text{sum} = \sum_{i=1}^{n} x_i \quad \begin{cases} i, n \in \mathbf{Z} \\ x_i \in \mathbf{R} \end{cases}$$

5. Cumulative Product

$$\text{product} = \prod_{i=1}^{n} x_i \quad \begin{cases} i, n \in \mathbf{Z} \\ x_i \in \mathbf{R} \end{cases}$$

Appendix F. Index of Enclosed Software

This appendix lists the software included on the attached diskette.

FILE	DESCRIPTION
20_a.wts	Neural network weights, 20 hidden nodes.
egavga.bgi **herc.bgi**	Turbo Pascal video drivers.
ffnets.pas **ffnets.tpu**	Unit containing neural network construction and training utilities. Created by Werner Trossbach, UCT, 1994.
hcs_rtc.pas **hcs_rtc.exe**	Source code and executable for the HCS real-time control module.
hcs_sim.pas **hcs_sim.exe**	Source code and executable for the Hybrid Switching Control System (HCS) simulation.
id_set_c.pas **id_set_c.exe**	Performs classification tests on a given neural network represented by its weight file.
jn_ctrl.pas **jn_ctrl.tpu**	Unit containing supervisory control utilities.
jn_dt280.pas **jn_dt280.tpu**	Unit containing utilities for interfacing to the DT2801 ADC/DAC card.
jn_utils.pas **jn_utils.tpu**	Unit containing miscellaneous utilities implementing neural network and mathematical functions.
mbsg.pas **mbsg.tpu**	Unit containing signal generation utilities. Created by Martin Braae, UCT, 1995.
mbwndw.tpu **mbconst.tpu**	Graphing utilities. Created by Martin Braae, UCT, 1995.
nn_dta_c.pas **nn_dta_c.exe**	Generates network training data for a particular setpoint type.
nn_trn_c.pas **nn_trn_c.exe**	Trains a particular network structure using sample data from specified training files.
test_dac.pas **test_dac.exe**	Simple utility to test the functionality of the ADC/DAC card.
vgadrv.obj **herc.obj**	Binary object equivalents of the Turbo Pascal video drivers.