The Development of Computer-Assisted Techniques for the Classification of Nerve Spike Signals.

Mark Hamilton Browning

being a thesis submitted in fulfilment of the requirements for the degree of Master of Science in the University of Glasgow, Faculty of Science, Division of Environmental and Evolutionary Biology.

September 2000.

ProQuest LLC.
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106 – 1346

Abstract.

Since the development of electronic amplification and signal recording facilities, there has been considerable interest in separating and classifying nerve spike events. Initially techniques were developed to identify spikes on the basis of amplitude but, as technology has progressed, the main interest has been in the development of techniques for classifying nerve spikes on the basis of shape. A range of strategies have been developed for performing the separation, but these strategies have been (and possibly still are) limited by the capabilities of the available hardware. These strategies and their implementations are described.

A novel method for performing automated spike shape classification is described. Software has been written to implement this method, and it is applied to nerve spike data from extracellular recordings of the superficial flexor nerve of the Norway lobster (*Nephrops norvegicus*) and from the coxo-basal chordotonal organ and cuticular stress detector one of the crayfish (*Procambarus clarkii*). The results are assessed by the use of contemporaneous intracellular recordings and compared with the performance of a commercially available spike classifier, voltage thresholding techniques and an implementation of a pre-existing technique for classifying spikes.

The relative merits of different strategies are considered, as well as the fundamental limitations of attempting to segregate spike data on the basis of shape alone. Technical issues relating to the implementation of a software based spike classifier are also considered.

Contents.

List of Tables

List of Figures

Acknowledgements.

I wish to thank Maria Denheen, Cornelia Leibrock and Daniel Cattaert for access to data gathered by them and for advice on matters relating to that data. I also wish to thank John Riddell and David Halliday for providing access to equipment.

I particularly wish to thank my supervisor, Jon Barnes, and also Max Huxham without whose advice and general encouragement this thesis would never have reached a successful conclusion.

Finally, I would like to thank everyone who supported and encouraged me during the course of this work.

Author's Declaration.

I hereby affirm that the thesis entitled "The Development of Computer-Assisted Techniques for the Classification of Nerve Spike Signals" represents, except where a note is made to the contrary, work carried out by myself. The text was composed by myself.

Mark Hamilton Browning

September, 2000.

1 Introduction.

1.1 The nature of nerve spike recordings.

A single neuron conveys information in the form of an all or nothing electrical pulse. However, there is a fundamental difference between this and the digital information transported along a fibre-optic cable. This is obvious enough to a biologist, but the concept of digital communication is fast becoming so culturally ingrained that the idea that biological systems differ from electronic ones, and that this difference is important to the operation of the systems concerned may soon seem strange. In essence then, an electronic system uses sequences of pulses to code information. Over short distances these are square-wave pulses, but over longer distances square-waves tend to spread out and merge due to the differing velocities of their frequency components. The use of soliton waveforms as the binary pulses in digital communication lines is now becoming accepted since these do not degrade with the same rapidity (Hasegawa and Kodama, 1981). Also, long distance digital communication lines have relays at regular intervals to prevent unacceptable levels of signal decay over these distances.

Nerve axons correspondingly transfer information in what can be considered to be binary pulses (nerve spikes). However, that is where the similarity ends. These pulses are discrete, the time between pulses is irregular and, even though the distances are short by engineering standards, the membrane properties ensure the signal is constantly returned to its original shape - thus sidestepping the engineering issues of separating a degraded signal from the background noise.

The key point is that the bulk of signal processing research is geared to engineering and physical applications and not to the understanding of biological signals. The scientific culture which is therefore applied to the development of spike signal analysis methods is thus largely foreign to the biologist studying the physiological importance of these waveforms. This has the obvious potential to lead to fundamental problems when applying new methodology in this area.

1.2 Single-unit versus multi-unit recordings.

A nerve spike activity recording can be effected either by inserting an electrode into a neuron (an intracellular recording), resulting in a record containing of necessity only that neuron's activity (that is a single-unit recording), or by placing an electrode in close proximity to several neurons (an extracellular recording) - resulting in a multi-unit recording. Other methods may also be applied, such as the use of voltage sensitive dyes to measure neuron activity optically (Yamada *et al.*, 1992). In practical terms the only problem in analysing (as opposed to recording) a single-unit record is the separation of the spike signal from the background noise. As well as being easier to obtain, multi-unit records have the enormous advantage of containing information about the activity of anything from a few neurons (D'Hollander and Orban, 1979) to dozens if not hundreds of neurons (Abeles and Goldstein, 1977; Jansen and Maat, 1992). However, the use of this type of recording leaves the problem of how to separate the resultant activity correctly, as well as the disadvantages of not seeing synaptic potentials or being able to identify cells anatomically by the use of (say) Lucifer yellow.

1.3 Motivation for the separation of multi-unit recordings.

Before considering how separation of multi-unit recordings may be achieved, it is worth considering in greater detail why such a separation may be useful, or more accurately why it should be attempted for recordings in which it is a non-trivial task. To consider specifically the system used in this study (described in detail in section 3.1), previous studies (Harris-Warwick and Kravitz, 1984) have not been able to distinguish reliably between the activity of the F1 and F2 tonic motor neurons or the F3 and F4 tonic-phasic motor neurons of the third root of the superficial flexor nerve of the Norway lobster (*Nephrops norvegicus*), and related systems have exhibited corresponding problems (Sokolove and Tatton, 1975). However studies of the effect of neuromodulators such as serotonin or octopamine on the motor output patterns in this and related systems require the ability to perform these separations reliably, as do

studies which aim to determine neuronal coupling (Tatton and Sokolove, 1975; Denheen, 1992) by cross-correlation of the activity of separate neurons.

It may initially seem obvious that it is worthwhile to separate a multi-unit record - particularly when the alternative to one extracellular electrode is the use of two or three intracellular electrodes. However, the effort required to effect the separation and the general applicability of hardware or software developed for the purpose must be set against the effort saved by simplifying the experimental procedure. In this context it must be noted that no worthwhile experiment requires so much effort or takes such great resources as to be unrepeatable. Some preparations obviously work better than others. However, unlike certain areas of physics or astronomy, progress is not based on data derived from experiments which are unique. On the contrary, if an experimental result cannot be replicated then the result must be viewed with considerable suspicion. Either the quantity of data to be analysed must be significant or the results must be unobtainable by other means in order to justify the effort involved.

1.4 Strategies for the separation of multi-unit recordings.

The nature of the task has been discussed by Glaser and Ruchkin (1976) and the range of approaches has been reviewed by Schmidt (1984a; 1984b). It is instructive to consider the pattern which emerges from this. Over the last forty years there have been various attempts, both hardware and software based, to separate multi-unit records using threshold detectors (MacNichol and Jacobs, 1955; Littauer and Walcott, 1959; Hermann *et al.*, 1962; Landolt and Milliken, 1970), single window discriminators (Hermann *et al.*, 1962; Bradley *et al.*, 1967; Freeman, 1971; Bak and Schmidt, 1977; Millar, 1983), multiple window discriminators (Simon, 1965; Schmidt, 1971), clustering of reduced (Dinning and Sanderson, 1981; Wörgötter *et al.*, 1986; Salganicoff *et al.*, 1986; Kreiter *et al.*, 1989) or non-reduced (Gerstein and Clark, 1964; Jansen, 1990; Marion-Poll and Tobin, 1991; Bergman and DeLong, 1992; Jansen and Maat, 1992) feature waveforms, contour fitting algorithms (Kent, 1971; Akker *et al.*, 1982), Fourier analysis (Bessou and Perl, 1969), principal component analysis (Abeles and Goldstein, 1977; Eggermont

et al., 1983), curve fitting (Remmel, 1983) and various strategies considering timing of peak or peak-to-peak amplitudes (O'Connell *et al.*, 1973; Mishelevich, 1970; McCann, 1973). Multiple electrodes have been used to provide conduction velocity information (Schmidt and Stromberg, 1969; Heetderks and Williams, 1975; Roberts and Hartline, 1975; Kanz *et al.*, 1978) and double electrodes (Camp and Pinsker, 1979; McNaughton *et al.*, 1983) to provide two recordings with slightly different electrode positions.

Most of these systems have been on-line (Simon, 1965; Mishelevich, 1970; Schmidt, 1971; D'Hollander and Orban, 1979; Dinning and Sanderson, 1981; Cohen and Landsberg, 1983; Wörgötter *et al.*, 1986; Salganicoff *et al.*, 1988; Bergman and DeLong, 1992), although some of the more technically ambitious have been off-line (McCann, 1973; Roberts and Hartline, 1975; Abeles and Goldstein, 1977; Camp and Pinsker, 1979; Jansen, 1990; Jansen and Maat, 1992). Some have relied partially or exclusively on hardware for the separation (Schmidt, 1971; Bak and Schmidt, 1977; Cohen and Landsberg, 1983; Millar, 1983; Wörgötter *et al.*, 1986; Kreiter *et al.*, 1989), while others have been implemented almost entirely in software (Simon, 1965; McCann, 1973; Roberts and Hartline, 1975; Camp and Pinsker, 1979; D'Hollander and Orban, 1979; Studer *et al.*, 1984; Salganicoff *et al.*, 1988; Jansen, 1990; Marion-Poll and Tobin, 1991; Jansen and Maat, 1992; Bergman and DeLong, 1992; Yamada *et al.*, 1992), although the extent to which a system that requires a dedicated mini-computer (McCann, 1973; Camp and Pinsker, 1979; D'Hollander and Orban, 1979) can be said to be hardware independent is perhaps debatable. There has been a steady growth in the processing power required to perform the separation (Schmidt, 1971; D'Hollander and Orban, 1979; Kreiter *et al.*, 1989), and this has been more or less in line with the most powerful facilities reasonably available at any particular time. Systems have been challenged with synthesised waveforms, automatically selected events, and live recordings. Many systems rely on an initial "training" phase for the definition of events of interest. This training may be performed on the data to be classified (Millechia and McIntyre, 1978; Jansen and Maat, 1992), on a specific training segment (Salganicoff *et al.*, 1988; D'Hollander and Orban, 1979) or on manually selected events (Kent, 1971; Jansen, 1990). A number of investigators have made broad claims as to the utility and

applicability of the systems which they have devised (Kanz *et al.*, 1978; Wörgötter *et al.*, 1986; Jansen and Maat, 1992). However, with a few exceptions (notably threshold detectors and single window discriminators), there is little evidence in the literature to suggest that these systems, and sometimes even the principles underlying them, have been used extensively except by their originators. In conclusion therefore, the literature pertaining to this problem indicates that the available computer hardware has been, and perhaps remains, a major limiting factor. Also some systems which have been developed, have become unusable because the computer hardware for which they were developed has become obsolete.

1.5 Considerations in the separation of multi-unit recordings.

Having considered the strategies underlying the attempts to separate multi-unit recordings, it is now appropriate to consider the concerns and difficulties experienced by various investigators in the pursuit of these goals.

The first and most fundamental consideration is the signal to noise (S/N) ratio of the recording. This of course depends on the conduct of the experiment as well as the performance of the analog to digital (A/D) converter. Any consideration of the separation of signals from background noise, or the analysis of badly degraded signals is outwith the scope of the present study, and as such will not be considered further.

The next important consideration is the number of datapoints required to represent the spike waveform accurately. Obviously, more datapoints result in a more accurate waveform but this is correspondingly more expensive both in terms of the data storage and processing requirements. Fortunately an upper limit can be derived for any particular type of impulsive waveform data, namely twice the bandwidth (Stremler 1990). The bandwidth is defined by the highest frequency component of the waveform, which is obtained from the Fourier transform. Abeles and Goldstein (1977) point out that the maximum bandwidth of a nerve spike is around 10kHz, and consequently 20 datapoints are required per millisecond. Anything less than this will result in a loss of meaningful information about the spike shape. However it must also be considered that

this estimate is for the representation of a waveform in terms of its component frequencies. In other words, the shape of the waveform can be accurately calculated from the sample values but this data will not in general contain the maxima or minima of the waveform since these will usually lie between sample points. This becomes an issue because much software (including the *Spike2* program produced by Cambridge Electronic Design Ltd. (CED)) uses the simplifying assumption that the values between two datapoints are linear combinations of the datapoint values, hence cropping the peaks and troughs of a waveform sampled at only twice the bandwidth.

Having established an adequate sampling regime and captured the data, it is next necessary to separate the spike waveforms from the background. This is most commonly achieved by selecting a suitable threshold voltage (MacNichol and Jacobs, 1955; Hermann *et al.*, 1962; Landolt and Miliken, 1970; Millar 1983) and defining a spike in terms of the waveform contained in a time interval around any crossing of the threshold (Schmidt, 1971; Bak and Schmidt, 1977; Abeles and Goldstein, 1977; Kreiter *et al.*, 1989; Jansen, 1990; Jansen and Maat, 1992; Bergman and DeLong, 1992).

At this point strategies diverge. On the one hand are attempts to classify spikes as they are sampled (the on-line approach). This must of course be effected in real time, and leads to the possibility of software based systems suffering from a "dead" period after each spike, hence encouraging interest in hardware based spike sorters. On the other hand, off-line systems make no attempt to operate in real time, and hence can employ more sophisticated identification strategies which may not even operate on the data in its original time sequence. Investigators following on-line strategies find that speed is of crucial importance, while those employing off-line strategies sacrifice immediate answers in favour of more manageable hardware requirements.

There have been major advances in the capabilities of computer hardware available since the early 1960s, and this is reflected in the complexity of the software applied to spike classification. In 1970, Mishelevich described a system running on a Spear Micro-LINC 300 in which the entire computer had the equivalent of 12 kilobytes (KB) of random access memory (RAM), of which less than 1.5 KB was required to load the whole program. Not surprisingly, the software was correspondingly simple. At the

opposite extreme is the off-line system described by McCann (1973) in which a suite of programs offering multiple forms of analysis was written for an IBM 360/44 mainframe with 128 KB of RAM. This used a GUI (graphical user interface) and was controlled by light pen. For its time the program was an impressive achievement, and even today it would be a very respectable system. However it would have taken a large research project to justify using such a (then) powerful computer (O'Connell *et al.*, 1973). On a more realistic scale are systems such as that described by D'Hollander and Orban (1979) using a PDP-11/40 with 28 KB of RAM for on-line classification using a template method. Software based systems described in the late 1980s and early 1990s are typically IBM PC based (Jansen, 1990; Marion-Poll and Tobin, 1991; Jansen and Maat, 1992; Bergman and DeLong, 1992) but are not invariably so (Salganicoff *et al.*, 1988; Yamada *et al.*, 1992).

The complexity of hardware based systems has correspondingly increased from the simple electronic circuits described by MacNichol and Jacobs (1955), Littauer and Walcott (1959), Landolt and Milliken (1970), Schmidt (1971) or Bak and Schmidt (1977) through simple microprocessor based systems such as that described by Cohen and Landsberg (1983) to the system described by Kreiter *et al.*, (1989) which incorporates an MC68000 central processor unit (CPU) and 128 KB of RAM, all of which is controlled by an RS232 connection to an IBM PC. This level of hardware complexity approaches that of a PC, to the degree that if developed further such a system would virtually be a full fledged computer. On different lines is the work described by Wörgötter *et al.*, (1986) in which adjustable analog delay lines are used to effect the matching process.

1.6 Testing strategies.

Any hardware or software for separating multi-unit recordings must be tested, and a range of testing strategies have been adopted by different investigators. Data can be synthesised with varying degrees of realism. At the simpler end of the scale is the sine wave approximation used by Mishelevich (1970) to which noise was added by

Wörgötter *et al.*, (1986), and the triangular waveform with added noise used by Cohen and Landsberg (1983). More complex is the method described by Bergman and DeLong (1992) in which two principal component waveforms were defined and combined according to user specified ratio ranges, with added band-limited noise. Synthetic data has the great advantage that the correct match is known in advance, giving an opportunity for the system to be tested objectively. Real data does not have this advantage. However it does fully reflect the natural variability which any worthwhile system must be capable of handling. All of the systems referred to have been tested using real data. Some investigators are keen to compare their systems with other approaches. However, worthwhile comparisons are complicated by the diversity of data sources and qualities (Schmidt, 1984b). Little serious comparison of techniques has been effected, an exception being the work of Wheeler and Heetderks quoted by Schmidt (1984b).

1.7 Complications of multi-unit separation methods.

It is difficult enough to separate spike waveforms at the best of times. However, there are two further problems which have to be considered. The first of these is waveform overlap. A multi-unit recording contains the activity of a number of neurons, and there is absolutely nothing to prevent two (or more) of these from being active at substantially the same time. The result is the existence of composite events representing the sum of the activity at that instant. It is important to realise that these events are in general the outcome of non-aligned spikes, and therefore do not themselves have characteristic waveforms. This can result in problems both in the training or template generation phase and in the main data analysis phase.

In the training phase, composite waveforms can be registered as rare but distinct spike classes, resulting in a subsequent waste of processing capacity attempting to identify what is in fact a unique event. This has been overcome in a number of ways. Operator-directed training is one approach (Jansen, 1990). The forming of templates on clusters with more than a specified number of occurrences is another (Jansen and Maat,

1992). In many studies training is effected by using a clustering technique (D'Hollander and Orban, 1979; Dinning and Sanderson, 1981; Salganicoff *et al.*, 1988; Kreiter *et al.*, 1989). However, it is often not clear how (or even if) overlapping events are handled by these systems. Clearly overlapping events give rise to cluster outliers which if added to cluster based average waveforms will have a minor (but potentially significant) effect on the template, while outliers which form their own clusters (and hence templates) will slow the analysis process.

In the analysis phase, processing time is wasted by searching for matches to overlap events. This is potentially significant but is by no means the major concern. The failure to identify the spikes causing an overlap results (obviously) in a loss of information. In an on-line system, there will be insufficient spare capacity to separate overlap events - judging by the specifications of the hardware typically used in such systems. This either relegates overlap processing to being an off-line task (Salganicoff *et al.*, 1988) conflicting with the very idea of on-line processing, or results in such information as could be extracted from overlaps being discarded. This loss of information is potentially significant, there is after all no point in performing accurate spike classifications in order to generate crude statistical summaries of the activity. The analysis of burst activity or neuronal coupling could be seriously affected by the removal of significant numbers of events, particularly if the overlap is non-random in a particular system. The actual frequency of overlap events also affects how they should be resolved. As rare occurrences it may be appropriate to discard them as not being worth the effort of separation, but if they are common or if the proposed analysis of activity is sensitive to data censoring then strategies must be adopted to resolve as many occurrences as possible.

Various investigators have described spike classification systems in which they omit details of the handling of spike overlap situations, or alternately make only cursory reference to the problem (Millechia and McIntyre, 1978; D'Hollander and Orban, 1979; Dinning and Sanderson, 1981; Wörgötter *et al.*, 1986; Salganicoff *et al.*, 1988; Kreiter *et al.*, 1989; Jansen, 1990; Bergman and DeLong, 1992; Jansen and Maat, 1992). This is not to suggest that all their systems fail to handle overlaps. Indeed, some like the

neural net systems (Jansen, 1990; Jansen and Maat, 1992) could be expected to perform well in such situations. Some investigators have considered how to extend their technique to cope with overlaps (Abeles and Goldstein, 1977), and the extent to which overlap can be tolerated (Mishelevich, 1970). However, no-one has described a single electrode system in which the overlap problem is systematically tackled - other than by the vigilance of a trained operator (Prochazka *et al.*, 1972; Prochazka and Kornhuber, 1973; D'Hollander and Orban, 1979).

However, work by Roberts and Hartline (1975) which has been further refined by Oguztöreli and Stein (1977), Andreassen *et al.*, (1979), Roberts (1979) and Stein *et al.*, (1979) presents a multielectrode solution to the overlap problem. Their technique essentially depends on the differences in conduction velocities to provide a non-overlapping event pair at one of the recording electrodes. According to the data presented by Roberts and Hartline (1975) and Andreassen *et al.*, (1979), their approach is highly effective at classifying spike events accurately. Their method, although one of the most theoretically robust and practically successful of those reported, has the obvious practical limitation of requiring multiple electrodes evenly spaced along the nerve fibre - which is clearly not a practical arrangement in many studies (McNaughton *et al.*, 1983).

The second problem of waveform separation is the effect of waveform drift. This can itself take two forms. In the first, the baseline voltage changes either periodically or randomly over an extended time. This has the obvious effect of nullifying any separation method based on series of window discriminators which does not use a baseline correction (Bergman and DeLong, 1992), but can readily be overcome by differentiating the waveform numerically and using the derivative waveform in the matching process (Marion-Poll and Tobin, 1991). The second and more serious problem is that of amplitude variation, both over the course of a particular experiment and during a single burst (Bergman and DeLong, 1992). Most investigators who have considered this problem have added newly matched spikes to the templates, hence any slow bias will be accommodated over the time of its manifestation (Cohen and Landsberg, 1983; Struder *et al.*, 1984). On the other hand, the problem has also been resolved by using templates with an error margin of up to six standard deviations (Bergman and DeLong, 1992).

1.8 The nature of classification errors.

Several investigators have discussed the types of classification error which can be made (Sarna *et al.*, 1988; Bergman and DeLong, 1992). In short, there are four potential errors which can be described: a *false positive* in which noise is identified as a spike, a *false negative* in which a spike is identified as noise, a *false match* in which a spike is incorrectly classified, and a *double match* in which a spike is matched to two categories. Within each spike class the false match errors can be further divided into *inclusion errors* in which a spike of some other class is sorted into the specified class, and *exclusion errors* in which a spike of the specified class is assigned to some other class.

Of these various possible errors the most important and difficult to identify is undoubtedly the false match, and it is here that considerations of the classification reliability come to be of importance. For some purposes there may require to be a high degree of confidence that any spike identified as belonging to a particular class should in fact be of that class. This implies that the template (or other method) should be set up to minimise the inclusion errors. For other purposes it may be sufficient to identify possible members of a particular class, resulting in a minimisation of exclusion errors (presumably without including everything), or equalise the rates of inclusion and exclusion errors to obtain general activity over a period.

One implication of this is that by permitting only one attempt at classification an on-line system necessarily lacks the flexibility of an off-line system to re-analyse the data set while adjusting the permitted rates of inclusion and exclusion error. To assume that data (of any description) can necessarily be analysed correctly at the first attempt is to endow the process of analysis with mystical properties. The involvement of computers certainly does not improve this situation, and the most prudent course is always to allow for the possibility of re-analysis.

1.9 The approach used here.

The basic approach used is an off-line system, with manual template creation (the

learning phase) and two quite distinct matching methods. The first is a variation of a contour fitting approach in which the templates are defined on characteristic parts of the waveform and then linked together after the main analysis. The second is a departure from the methods described in the literature, in that the distance comparison is not performed in multi-dimensional space or a transformed multi-dimensional space. Rather it involves a measure of the similarity of information content of the spike and the template. The methods used also attempt to address the problem of spike overlap, in the first by looking for sets of characteristic features rather than complete waveforms, and in the second by the information comparison itself.

2 Computational Materials and Methods.

2.1 Computer hardware and its recent development.

Computer hardware has developed at a breathtaking rate over its short history, and all classes of operating systems and application software have moved apace. The design, construction and testing of a piece of software are heavily influenced both by the theoretical state-of-the-art and by the tools and operating systems available at the time. In particular, it must be realised that when this project was begun in 1990 the first practically useable version of the *Microsoft Windows* operating system - version 3.0 - had not been released. It is therefore difficult simply to refer to previous work as a basis for describing the methods used here. An expanded description of relevant technical issues is therefore included to provide an understanding of the technology involved, the rapidity of its development and how this relates to this project.

2.1.1 The basis of digital computing.

Many of the early computers were analog machines, that is they represented numbers by voltages which could assume any value within a given range. This approach was superseded by the digital representation of numbers as sets of binary on/off signals which are grouped together into bytes (most commonly containing eight bits) and which consequently represent 2^8 or 256 values. Assemblies of bytes give rise to standard data types such as characters, integers and floating point numbers, as well as higher order data structures.

Computer hardware and software has developed in parallel, each feeding the further development of the other. Howeyer, a number of distinct types of hardware have emerged - supercomputers, mainframes, high performance workstations and desktop computers (specifically the IBM personal computer (PC) and the Apple Macintosh). At the commencement of this project a typical desktop PC of the type used here had an Intel 286 CPU, 2 megabytes (MB) of RAM and 40 MB of disk capacity. By the time the main

part of the work described here was complete the CPU speed and storage capacity of other components of a similar machine had increased by a factor of about 30.

The single most significant feature of the PC is its use of a segmented memory architecture, a legacy of the 8 bit memory addressing schemes of the early predecessors of the PC. However it has important implications for the design and implementation of software, a point which will be returned to later. Originally the PC had only one processor mode - real mode - which permitted software to access up to 1 MB of RAM via a pointer to a physical memory location, and is the mode used by MS-DOS. With the Intel 386 came 32 bit memory addressing, permitting the use of up to 4 gigabytes (GB) of RAM and another processor mode - virtual 8086 mode (which is the processor mode used by *Microsoft Windows*).

2.2 Operating systems.

An operating system is the link between the application software and the computer hardware, and as such has developed in parallel with the hardware. It provides a method by which software can operate without having to consider irrelevant details such as how data is stored on a disk, and how to retrieve that data. It is also responsible for loading and running the application software, managing system resources and providing an application program interface (API).

An operating system in its simplest form will allow a single user to operate a single piece of software (or task) at any one time. More complex systems will allow multiple users to share hardware (as in mainframes) and may allow multiple tasks to be run concurrently. This last may use either non-preemptive multitasking (as in *Microsoft Windows*) or preemptive multitasking (as in UNIX or IBM OS/2). Some systems permit multi-threading, that is the capacity for a single program to operate several separate (but not totally independent) internal processes concurrently. As if this was not enough complexity, there is a specific problem with some operating systems (such as MS-DOS) in which some parts of the system require access to other parts in a fashion which precludes any other operation making the same demand at the same time. This is

referred to as non-reentrance and it is a particularly significant problem with MS-DOS, even though MS-DOS is a single-user, single-tasking system.

Most operating systems take the approach of forcing a degree of insulation between the application software and the hardware, a process known as virtualisation. Taken to its conclusion virtualisation provides a mechanism for insulating separate tasks from each other's faults. Systems which take this approach also must cope with the concept that certain pieces of software, notably the operating system kernel itself but also device drivers, must have a degree of privileged access to the hardware which is denied to application software.

2.2.1 MS-DOS.

When PCs were first developed, existing operating systems such as UNIX placed too great a demand on the hardware to be used as the operating system. Thus when IBM introduced the PC in 1981 it used the MS-DOS operating system developed by Microsoft Corporation. This owed much to the then common operating system CP/M and also borrowed heavily from UNIX. However it was a single-user, single-tasking operating system, and it has many internal features which limit its potential (Dettmann, 1989).

At the commencement of this project DOS was by far the most commonly used operating system in the world, being used by almost all IBM PCs and it could not have achieved such success without its simplicity - it is quick and easy to develop a small DOS application, relatively little expert knowledge is required to run the development tools or work with the API, and the system is highly error tolerant (perhaps too much so).

On the other hand, the DOS API lacks anything but the most rudimentary of user interface capabilities (and little support for interface standards). The precise error tolerance which makes a small program easy to construct makes the task of debugging a large program far harder (since the errors have a cumulative effect). Worse still, MS-DOS only supports the real mode of the Intel processors on which all PCs are based, and this combines with the location of the system ROM at the memory address 0xA000:0000

to limit the available memory to 640 KB - which must be shared by DOS, device drivers, application software and data.

2.2.2 Microsoft Windows as an operating environment.

Technically *Microsoft Windows* (or *Windows* for short) was, at least until version 4 (also known as Windows 95), an operating environment rather than an operating system. The difference is that it requires DOS in order to operate. This may seem to be splitting hairs, but *Windows* depends for many of its basic operations on the capabilities of DOS, and so must either handle the limitations of DOS or pass these limitations on to *Windows* application software.

Windows has two basic components, a GUI and a DOS extender. The GUI provides the user with a CUA compliant environment, and provides a large range of useful API facilities. These include support for higher level interface objects such as bitmaps, dialog boxes, menus, buttons and scroll bars along with the required support for window management services and a standardised help facility. Early versions of *Windows* (prior to version 3.0) were restricted by the 640 KB limit of DOS real mode operation, but *Windows* 3.0 introduced two additional operating modes: standard and 386 enhanced, corresponding to the protect and virtual 8086 modes of the Intel CPUs. This was achieved by including a DOS Protect Mode Interface (DPMI) DOS extender as an integral part of *Windows* to provide all applications with access to up to 16 MB of RAM, and incidentally providing access to hardware based memory access checking facilities. *Windows* also provided a standard mechanism for allowing third party hardware manufacturers (particularly of graphics adapters and printers) to interface their products with all *Windows* applications. This sidesteps all the problems DOS has with successfully supporting a wide range of hardware.

However, *Windows* has a number of drawbacks, some merely irritating but others potentially very serious. It uses non-preemptive multitasking (Petzold, 1990), so any processor intensive operations bring the entire system grinding to a halt. Worse, the use of a single message queue and the sharing of resources mean that a malfunction in

one task can readily compromise system integrity (King, 1994). On top of this, certain API functions either do not work as documented (by Microsoft) or differ sufficiently from their documented modes of operation to appear to be unreliable. Overall, *Windows* applications tend to require phenomenal amounts of computing resources (processor speed, RAM, disk space, display capability) relative to their DOS cousins. However, this is not comparing like with like. *Windows* applications tend to provide substantially more functionality than their DOS based predecessors, and part of the rationale for developing *Windows* applications is precisely the ability to access these resources.

2.3 Programming languages and other tools.

Programming languages have developed alongside the hardware and operating systems, from the early plug board (Augarten, 1984) through to languages as diverse as Smalltalk, Fortran and C. Each language also evolves as new requirements arise (Metcalf, 1992) and due to commercial pressures.

Currently there are two distinct approaches to the generation of an executable program. The first is to use an interpreter to take the text file containing the language instructions (source code) and turn these into machine instructions at execution time. This approach has one major drawback - the interpreter must process the source code and keep track of what is happening as it happens, consequently the process is slow and not amenable to efficiency savings. This is usually exacerbated by interpreters performing elaborate error checking to avoid potential problems.

The second approach is to perform the translation into machine instructions and then run the program. There are several steps in this process. First a compiler operates on each source file, undertaking a range of performance enhancing optimisations, or alternately adding special checking and debugging information. Once each file has been compiled, all the resulting object files are passed to the linker which makes the appropriate connections within and between the object files and with the relevant libraries (collections of previously bundled together object files). This approach actively encourages the use of multiple source files.

Systems such as *Windows* require an additional step following linking. *Windows* makes great use of resources (items such as dialog boxes, bitmaps, menus or cursors) and these are held in a separate resource file. This is passed to a resource compiler (analogous to the language compiler), but rather than being hooked in by the linker, the resource file and the executable file produced by the linker are passed to a binder (which is in essence another type of linker).

The process of compiling and linking is usually more time consuming than interpreting when a program is only going to be run once or twice, but it provides great scope both for automated error checking and optimisation as well as removing the run-time time penalty inherent to interpreting. It therefore provides a more robust approach to development and a better end result.

2.3.1 The use of C.

The language used for all the code developed in this project was C. C is a compiled language (unlike Basic), with a wide range of development tools readily available for the PC environment (unlike Fortran or Pascal). Syntactically its structure is similar to the underlying machine operations, so C code compiles to produce applications which run almost as fast as the hardware itself permits (unlike C++). While the basic command set is extremely limited, it is intended to be extended by the use of third party facilities - indeed the American National Standards Institution specification for C (ANSI-C) insists on a substantial additional command set, and this provides a natural route for the provision of additional facilities. In common with some other languages C permits the definition of new data types, and also permits the definition of data structures containing any combination of pre-defined data types. However, there are some serious drawbacks with C. It is arguably the hardest high level language to learn (with the obvious exception of C++), and the ANSI specification provides only for a teletype user interface. Unless software is being developed for *Windows* there is a very substantial requirement for user interface development. Also, the very power and flexibility of C permit bad coding practices to result in extremely

unstable software which can potentially do great damage.

2.3.2 C development tools and their recent history.

The choice of programming language depends on suitable development tools being available. As mentioned earlier, C offers a range of PC based development systems. The available tools themselves have developed significantly over the course of the project, and so it is necessary to review both the range available, and the development of these in the relevant period.

In this period Microsoft distributed versions 5.1, 6 and 7 of their C compiler and versions 1 and 1.5 of their Visual C++ compiler. Versions 5 and 6 provided a DOS only compiler, linker, debugger and related support utilities. Source code editing was undertaken via third party software, the integrated development environment (IDE) added in version 6 being unworkable. Version 7 added C++ compiler support, and also provided tools for developing *Windows* applications. Visual C++ version 1 was the direct follow on to version 7, and provided a usable *Windows* based IDE and extensive *Windows* debugging and resource editing facilities. The minimum hardware requirements for version 6 were a PC with 640 KB of RAM and 5 MB of disk space for a full installation (plus the requirements for the editor). Visual C++ 1 on the other hand required at least a 386 based PC with at least 4 MB RAM and 65 MB of disk space.

In the same period Borland distributed both *Borland C++* and *Turbo C++* (a cut down version) versions 2, 3, 3.1, and 4. All versions of *Borland C++* provide a compiler, linker, a consistent range of debuggers, many support utilities, an IDE (DOS based in versions 2, 3, and 3.1, *Windows* based in version 4) and full support for the production of *Windows* applications, including all the necessary debuggers and resource editors. Unlike *Microsoft C* where every version is significantly different to use, the Borland IDE and debugger have developed in a consistent manner. Version 4 requires a minimum of a 386 with 4 MB of RAM and 75 MB of disk capacity.

Both of these systems have certain common features, specifically a move from DOS based systems primarily intended for DOS development to *Windows* based systems

for *Windows* development, in line with the move towards *Windows* as the dominant operating system for PCs. The IDEs for both systems also improved considerably over the period in question, however there was a substantial increase in the specification of hardware required to use either system. The precise versions used here will be described more fully in section 2.8.

2.3.3 DOS extenders and other resources.

One of the benefits of compiled languages is the capacity to link in facilities produced by (usually commercial) third parties. There are a whole range of libraries available for C and class-libraries for C++ covering most common requirements from graph plotting to database manipulation. Additionally, specialist libraries are available relating to particular products.

One particular category of third party resource warranting specific attention is the DOS extender. As previously discussed, DOS suffers from a limit of 640 KB on the available memory. Early DOS software was sufficiently small, and worked with sufficiently small data sets to fit comfortably into 640 KB. Later, overlaying techniques permitted parts of the software to be loaded from disk as required. This reduced program execution speed but was mostly a satisfactory compromise. However it was still at the mercy of whatever overlay development support the compiler suppliers chose to provide. Additionally, this only provides a solution for program code segments, data is not provided for. One approach proposed by a consortium of Lotus, Intel and Microsoft (Forney, 1989) was to provide hardware support in the Intel 386 for page mapped expanded memory. This provided a 64 KB "page frame" in upper memory (i.e. in a free space between memory addresses 0xA000:0000 and 0xFFFF:0000) into which four 16 KB pages could be mapped at any one time from the total pool of up to 32 MB of expanded memory. As an alternate approach, the device driver HIMEM.SYS was created to permit software to access high memory (i.e. above 1 MB) by copying 64 KB blocks of memory into conventional memory. This process does not rely on hardware support and is consequently significantly slower.

Neither access to expanded memory nor access to extended memory via HIMEM.SYS is a particularly satisfactory approach - both methods require special modifications within the source code. These approaches are particularly unsatisfactory when coupled with the use of overlays, since this also places artificial constraints on the structuring of the code. None of these approaches address the fundamental problem - the limitations of real mode DOS.

There is however no constraint on the software changing the operating mode of the processor (Forney, 1989), providing that it also traps and handles all the relevant operating system functions. A number of commercially provided "DOS extenders" do precisely this, all that is needed is to link with the replacement libraries and use a real mode stub loader (see the next section) to load the DOS extender kernel and the application software. Suddenly all the available memory in the PC is there to be used by code or data, complete with all the hardware based protect mode memory checking capabilities. No code modifications are required.

2.4 Program segmentation.

A brief description of how programs are laid out in PC memory is warranted since details of this will be referred to later. As previously mentioned the PC uses a segmented memory architecture, with each segment being up to 64 KB in size. DOS provides a means (the segmented .EXE file) to create software spanning multiple 64 KB segments. On startup a "stub loader" is invoked to load these segments into memory and connect them correctly. These are the code segments.

There is also one default data segment which contains the global data area (or "near heap") and the stack. The near heap uses an area of memory starting at the lowest address in the data segment, while the stack starts at the highest address and uses successively lower addresses as data is added to it. The proximity of these two elements, and the lack of any insulation between them can be a significant source of problems - mostly when the stack (used to store automatic variables and other function information) overflows into the global data held in the near heap. The "far heap" is all the free

memory apart from the code segments and the data segment. It is available for allocation in chunks of up to 64 KB.

Each program has a single unique entry point, that is the memory address of the *main* (or in *Windows* the *WinMain*) function. The execution of the entire program starts when *main* starts, and ends when *main* ends. However, *main* is free to call other functions, each of which can in turn call yet other functions. Each function adds its automatic (i.e. temporary) variables to the stack when it starts and removes them when it ends. The stack thus expands and contracts depending on how many levels of function are currently active and the number and size of variables used.

2.5 Program architecture.

At the level of individual commands, any program is simply a sequence of instructions, and in the early days of computers, programs were sufficiently small that this was an adequate model for the entire program. However it is naive to expect a model appropriate to a few dozen or a few hundred lines of essentially linear code to be effective in projects consisting of dozens of modules, each containing thousands of lines of code with multiple potential execution paths.

At this level of complexity there is a specific requirement for high level program design (McConnell, 1993). The quality of the resulting architecture determines in large part the quality of the finished program. Poor coding can of course undermine the best designed software, but good coding absolutely cannot replace good design. Furthermore, design deficiencies detected during coding (or worse still during testing) are much harder to remedy than those found during the design phase (McConnell, 1993).

Any program architecture must in the first instance address certain basic issues, specifically (but not exclusively):

- how the modules are defined, what they do, and how they interact
- the capacity for change which is allowed
- the third party components available
- the necessary degree of fault tolerance

- the error handling strategy

- the required degree of overall robustness

- the hardware limitations.

Once the purpose of the program and the fundamental design considerations have been established there are a host of detailed concerns of stylistic consistency such as naming and calling conventions to be addressed. A consistent coding style increases code readability and consequentially reduces coding errors. Eventually a design will have split the program into subsystems, which will in turn be split into modules, these will then be split into functions. Usually the details of how the functions operate are left to the actual implementation.

2.5.1 Structured design.

There are two basic approaches to the actual design. The first is the structured design approach proposed in the early 1970s (McConnell, 1993). Structured design is usually accomplished either using a top-down or a bottom-up approach or some combination of these. In the top-down approach the most general level of organisation is considered first, it is defined, formally described, and verified. Each part of this is then defined, described and verified in turn, and the process continues until the stage is reached that implementing the subcomponents as individual functions is appropriate. The bottom-up approach is a composition strategy based on describing individual relevant functions and synthesising these into successively more general levels of description until the top level is reached. This is more than likely to be a prelude to further top-down refinement.

The advantages of structured design are that it provides a means for decomposing the problem into manageable pieces while deferring all detailed considerations of how to implement the individual components. However, it does assume that whatever problem is being addressed is actually capable of being decomposed into neat, non-overlapping pieces and it suggests that the data flow and execution paths are tree-like. These assumptions are often only partially true.

2.5.2 Object-oriented design.

The second design strategy is object-oriented design (OOD). This approach is characterised by the identification of real world and abstract objects which can then be represented by program objects (McConnell, 1993). The principle underlying this is the closer the program structure is to the real-world problem it aims to solve, the more understandable it will be and the less likely that spurious interactions will occur. OOD is thus a process of identifying objects and classes of objects, and identifying required operations on object classes.

A number of concepts used here characterise OOD, but are not necessarily restricted to object orientated programming (OOP) languages such as C++, and central to this is the idea of abstraction. Hierarchical systems perform abstraction at the level of functions (for operations) and structures (for data), object-oriented systems however take this to its logical conclusion with objects encapsulating both the data and the functions to access and manipulate that data, with some or all of the data only being available to the object functions. This is a rigorous implementation of the idea of information hiding, and serves to permit the internal data organisation to be altered as needed without disturbing any interaction with other objects. There is no inherent reason why functions cannot be associated with their data in C provided a reasonable degree of discipline is used in the coding process.

2.5.3 The relative merits of hierarchical and object oriented techniques.

No discussion of hierarchical and OOD techniques would be complete without a consideration of the drawbacks of each. This needs to be done in the context of the practical effects of these approaches, and in relation to the languages commonly used to translate designs into operational code.

Hierarchical design (even when using abstraction and encapsulation techniques to their limit) leaves much to the discipline of the programmer. It is easy to crosslink items, and even when strict separation is observed, modification of one part of the system can

have repercussions for others. This type of problem is squarely addressed by OOP languages such as C++, however the use of these languages leads to somewhat less obvious problems. First is excessive encapsulation and abstraction, where base classes are used to derive operating classes to the extent that even libraries and API functions are considered to be unavailable unless encapsulated into classes. Linked to this is the problem caused by modern C++ compilers provide enormous sets of base classes which, if used, require vast amounts of additional hardware resources. This arises in part because the classes from which the application classes are derived are themselves derived - perhaps many times over. Each layer adds bulk and reduces execution speed. Overall there is a very real risk that exclusive use of OOP techniques will lead to a fundamental lack of understanding of program operation and an inability to work outwith the confines of derived classes.

2.6 Sequential input and event input.

DOS was designed to use the then current interaction metaphor - the so called "glass teletype" - in which information is displayed line by line on a video screen, commands are typed on the current line and the whole display scrolls up and off the screen as more lines are added. The great advantage of this system is that it is technically simple, but it forces the user to be relatively expert.

This form of interface lends itself to linear interactions, and is particularly useful in such situations. However, for most purposes a more sophisticated interface is required. In particular, much software operates by guiding the user through a maze of hierarchical menus. This effect can be achieved by an extension of the linear interaction model, with each menu supporting only a limited number of current options. In terms of the underlying code, a hierarchy of decision making functions can quite satisfactorily cope with this situation, and it is even possible to implement a limited GUI using this approach. The problem is that code complexity increases as the number of available choices increases, and full support for GUIs such as *Windows* is impossible to achieve.

A more recent approach is to consider each keystroke or mouse movement a

single event in a stream of events. Taken with any required supplementary information it can be passed to the message handler for the appropriate window, dialog box, or other on-screen gadget. The message handler then processes the message based on the current state of the system and returns control to the message dispatching routine. This is an intrinsically more complex process than handling linear input, however it is much more flexible. Differing kinds of input can be handled, even events generated by the operating system or other application software. This approach also parallells the development of object-oriented techniques for designing the application software, and it is readily apparent that such a messaging system could be implemented in an object-oriented program. However, in terms of GUIs no current alternative to such a message handling system has the required flexibility to respond to input in multiple windows.

Windows uses an event pushing mechanism to feed event handlers for each window. It thus makes sense to use this mechanism rather than ignoring it in favour of a sequential input mechanism. Additionally, *Windows* (up to version 3.1) uses cooperative or non-preemptive multitasking, implemented by the message dispatching function passing messages to each active task in turn. Thus each task has exclusive control of the PC while the message handler is processing the message (Petzold, 1990), making use of this mechanism all but essential.

2.7 General considerations in software development.

The primary concerns for any piece of software are: first, is the software *stable* (i.e. does it run reliably) to the required degree? Without meeting this requirement, the software is next to useless. Second, is the code *maintainable* (i.e. can modifications be made later) to the necessary degree? In reality whatever the original specification and however sophisticated and exhaustive the original requirement analysis was, changes will inevitably be needed later. Third, is the correct operation of the software *verifiable* to the required degree? The extent of the verification depends on the importance (and immediacy) of the real world problem being addressed, particularly in safety critical applications. Fourth, but most fundamentally, does the software *solve the problem* that it

is supposed to solve? If it does not then it is by definition a failure.

There are also several secondary considerations that cannot be ignored without, at best, compromising the quality of the finished software. First, how *fast* is the software? If the speed is inadequate on the proposed hardware then it does not matter whether the solution is elegant (or whatever else). The software will not be usable. Second, how *clear* is the source code? Clearly written code is more likely to be well written, (McConnell, 1993) and it is certainly easier to maintain. Third, how *standardised* is the software both internally and externally. Code which is written using a variety of coding standards is not clear, and software which does not employ the current metaphors for user interactions without good reason is less easily used. Worse still is software in which different parts use different metaphors or require different assumptions on the part of the user, and this extends even to ways of laying out information on the display. In this respect a single bad standard can be better than two good ones.

2.7.1 The trade-offs.

Any design has to balance the merits of a particular solution against the drawbacks. In some instances, the advantages of a particular solution are compelling, but more frequently some sort of reckoning must be made in terms of the fulfilment of the underlying goals. This is not to suggest that fulfilling the software's basic requirements is something to be traded against elegant design (although this is sometimes done), rather how this is done is a legitimate, even necessary, consideration.

In software, the most common trade-off is against execution speed. The more compact the code the faster it will run. The more highly customised functions that are used the faster, but harder to write, debug and test the code becomes. Special knowledge of object structures can allow a quick fixup for a special case. These, and similar features enhance the speed of the program at the expense of stability, maintainability, verifiability and testability.

However, speed is not the only currency in software design. Program stability

(i.e. the capacity of the program to continue running without error when stressed) is also a common currency for trade-offs. Standardisation of software usually assists stability by making the code easier to write, debug, test and maintain. Some external "standards", particularly enhanced features of operating systems such as *Windows* may however compromise stability. Among other things *Windows* encourages the use of dynamic link libraries (DLLs) with the idea of making incremental upgrades possible. The result is often confusion about which version of a particular DLL is correct for a particular package, and may result in multiple copies of a DLL being required. *Windows* also offers dynamic data exchange (DDE) and object linking and embedding (OLE) mechanisms to permit supposedly "user friendly" standardised inter-task data exchange facilities. These are good in principle, but are complex to implement and complexity always implies potential stability problems, particularly when the ideas are not clearly documented, or like DDE / OLE, when the basic specification of the facilities is repeatedly revised.

2.7.2 The testing and debugging of software.

A bug can loosely be defined as any unexpected outcome of a particular input, and it is important to realise that all software contains bugs, which manifest themselves with differing frequencies (McConnell, 1993). Most bugs rapidly manifest under ordinary testing, however a few will only appear after years of normal operation - and can be correspondingly catastrophic (Neumann, 1995). Bugs in executable software are of two general types: defects of logic and defects of implementation (McConnell, 1993), syntax errors having already been detected by the compiler. Defects of implementation themselves divide into boundary value defects and general case defects.

Consistent testing with boundary values and a range of other inputs will allow the identification of many bugs, however a large proportion will require the use of a symbolic debugger (a piece of software which takes the debugging information built into the application executable and allows the programmer to step through the execution of the application line by line, viewing the source as it is executed and allowing variables to

be inspected). The debuggers available will be described later, however it must be noted that the debugger shares the memory space available for the application being debugged. For DOS applications this is a highly significant limitation.

2.8 The process of generating a functioning program.

In practise the process of creating software is an iterative process, a new section of code will be written, it will be compiled, linked, bound, tested, debugging will be undertaken, the code will be revised and so on until that section is operating. The next section will then be written, and this process continues until the entire program is finished. The importance of considering this is that the efficiency of the compile cycle largely determines the productivity of the entire process of code construction (as distinct from design), and the efficiency of the compile cycle is itself largely determined by the IDE - or for that matter the lack of an IDE. The quality of the IDE used may seem to be relatively trivial in some respects, but it does have a significant impact on the ultimate software quality and, in a time limited project, whether and to what extent there is a working piece of software at all.

2.8.1 Compiling under Microsoft C version 6.

One of the development systems used in the course of this project was *Microsoft C* version 6 (C6), and this section will describe the capabilities and limitations of this system as they relate to the process of developing software for this project. The development system itself is well documented (Microsoft, 1990), and books exist about the language variations in C6 and the capabilities of the development tools, however there is sufficient scope for variation within the system setup that it is appropriate to detail the actual installation used.

C6 in the configuration used is a DOS development system and it does not incorporate C++ compiler facilities. The Programmers workbench IDE was tested and discarded on the basis that it provided no significant operational benefits. Editing was

performed using Borland Brief version 3.1. This is designed to be a configurable programming editor and limited IDE. It allows source files to be compiled and lets error and warning messages be displayed along with the relevant lines of source code. Linking has to be performed at the DOS prompt and was usually accomplished using custom written batch files, the make utility being rejected on the basis that it was relatively hard to use and provided little or no real advantage over a few well structured batch files. Debugging was accomplished initially using Microsoft CodeView version 3.0 and later using Borland TDConvert version 2.0 to convert the executables to be debugged using Borland Turbo Debugger version 3.1, either as a cross-platform debugger or in a remote debugging configuration.

Overall C6 is totally inadequate for large scale development work without significant additional tools. Subsequent versions addressed these concerns, but none were available for this project.

2.8.2 Compiling under Borland C++.

The other development system used in the course of this project was *Borland C++* (BC) versions 3.1 (initially) and 4.0 (later), and this section will describe the capabilities and limitations of this system as they relate to the process of developing software for this project. BC provides an IDE, resource editor a C compiler, and a debugger, which can generate applications for both DOS and *Windows*. BC version 4 (BC4) incorporates a *Windows* version of the Brief editor, and an integrated debugger.

Since this system was used for the development of *Windows* applications, only this part will be described. The BC IDE provides excellent project management capabilities, including the capability to rebuild object files and libraries based on the date and time of the source files. Resources can be created and edited using the extensive Resource Workshop facilities. Debugging can be accomplished either using the IDE debugger or the Turbo Debugger for *Windows*. This debugger also readily allows dual monitor debugging - which by permitting the debugger to display on one screen while the normal *Windows* display is on the other aids the overall debugging process.

2.8.3 A few definitions.

A source file contains the mechanism of the program expressed in the programming language used. An object file is produced by the action of a compiler on a source file. A collection of object files constitutes a library file. A resource file contains bitmaps and other resources used by the software. An executable file is produced by the action of a linker on the object and library files. Alternately a DLL may be produced, which links to the executable when the program is executed. These are all either parts of the operational program or intermediates in the production process. A glossary of other technical terms is provided as an appendix.

2.9 The materials available.

2.9.1 Additional development tools and libraries.

The only additional development tool made available for this project was the Phar-Lap Software Inc. 286|DOS-Extender version 2.5. This version is a sixteen bit virtual control program interface (VCPI) DOS extender capable of linking with C6 object files to give the application access to up to 16 MB of RAM.

The only commercially produced libraries made available for this project were the the Cambridge Electronic Design (CED) CFS and SON data storage libraries (see chapter 3).

2.9.2 Signal capture and analysis facilities.

Signal capture was effected using a CED 1401 and, when available, a CED 1401+ analog to digital interface controlled by CED's *Spike2* software version 3.1 for DOS, and in the final stages also *Spike2* version 3.1 for *Windows*. Despite having the same version number these two packages have different capabilities, and neither data files nor macro scripts from one can readily be used with the other. *Spike2* version 3.1

for *Windows* incorporates spike identification capabilities.

2.9.3 Calculation aides used.

Many of the complex calculations implemented in the project software were checked against specimen calculations implemented on several packages. These were the spreadsheets *Lotus 123* version 2.1 and *Microsoft Excel* version 4.0 and the mathematical drafting package *MathCad* version 4.0 produced by Math Soft.

The spreadsheets used provided all of the required calculation capabilities, however the implementation of many complex equations in a spreadsheet leaves much room for error since the cell based formulae bear little resemblance to the mathematical equations from which they are derived. A mathematical drafting tool such as *MathCad* provides a means for implementing equations in a more "natural" fashion.

2.9.4 The computer hardware available.

IBM PCs were the only hardware platform available for this project. Initially, a 25 MHz 386 with 2 MB of RAM, a VGA display, 40 MB of disk storage, and no floating point coprocessor was made available, and this was eventually replaced by a 33 MHz 486 with 8 MB of RAM, an SVGA display, 200 MB of disk storage and a 120 MB tape drive.

2.9.5 The use of the systems available.

The original hardware and development tools made available for the project were inadequate for the task in hand, and were progressively replaced by more powerful facilities, although the timescale for this could have been more advantageous. However, certain of the libraries and facilities required the continued use of C6 rather than BC and this, rather than any philosophical objection to C++ mandated the use of C. The possibility of using C++ for some of the development, particularly for *Windows*

specific facilities, was considered but rejected on the basis that continual swapping between C and C++ was likely to lead to a reduction in coding efficiency in both languages rather than provide benefits in either. This did not however exclude the use of OOP techniques in C. The Phar-Lap DOS extender became available much later than was expected, and consequently the focus of development (and the experience) had already moved to the production of *Windows* software. Consequently the use of the DOS extender was somewhat limited, as will be discussed more fully in chapter 3.

The large memory model was used exclusively for the software developed here. None of the other programming models offered any real advantage, but did offer significant problems.

3 Biological Materials and Methods.

Signals have been recorded, manipulated and analysed since the development of suitable electronics - if not before (Bures *et al.*, 1967). The development of computers has however provided for considerable automation of the data capture and analysis process and many, if not most, data capture devices (such as oscilloscopes or chart recorders) are today based on digital electronics with interfaces permitting data to be transferred to computer. Additionally, it is possible to use general purpose analog to digital (A/D) converters linked to PCs or Macs and driven directly by software running on the computer.

3.1 The experimental setup.

It is useful to note some of the details of the experiments conducted by Dr. Maria Denheen (1992) from which the data used here are derived. In summary, extracellular recordings were made of the third root of the superficial flexor nerve (Fig 3.1) of the Norway lobster (*Nephrops norvegicus*). This root innervates a thin muscle sheet (the slow or superficial flexor muscle) which is involved in abdominal positioning. The third root itself consists of six motor neurons (F1-F6), five of which are excitatory, the remaining one being inhibitory (Knox and Neil, 1991; Denheen, 1992). In this and other crayfish and lobster systems the motor neurons have been numbered sequentially on the basis of neuron diameters and spike amplitude, the smallest being F1 and the largest F6 (Kennedy and Takeda, 1965; Wine *et al.*, 1974; Thompson and Page, 1982; Denheen, 1992). In this scheme F5 is the inhibitor neuron.

Data was initially recorded onto an analog tape recorder, and subsequently transferred to PC using the 1401 A/D converter interface (manufactured by CED), controlled by the *Spike2* software package (written by CED). Dr. Denheen's analysis of this was based on the use of the facilities within *Spike2* to generate timed event markers for specific classes of nerve spike (corresponding to individual neurons). This was achieved by using an appropriate set of threshold voltages to generate sets of event

markers which could be compared to give the events corresponding to each of the spike categories (see Fig 3.2 for an example).

3.2 The technology of signal capture.

Before considering the digitisation process it is first necessary to understand how analog and digital signals differ. An analog signal is a continuously varying voltage, while a digital signal uses a stream of numbers to represent (in some way) the waveform under consideration. Also a general analog signal contains no timing pulses or other artificial information.

3.2.1 The sampling process.

In the current context the sampling process and its consequences are crucial, and so particularly relevant aspects need to be pointed out (for a general review of the process see Stremler (1990)). An analog signal is continuous, and digitisation is accomplished by sampling, most commonly to 12 bit accuracy, at a specified interval - with the potential consequence of aliasing the signal. This is a relevant concern only when the signal contains components with frequencies greater than one half of the sampling frequency.

Of more specific concern in the recording of electrophysiological events is the potential for the loss of instantaneous extreme values (see Fig 3.3A), and the consequent distortion of crucial parts of the waveform under consideration. This can be avoided by oversampling the signal (see Fig 3.3B), or alternately by using the Fourier transform to reconstruct the waveform (see section 3.2.2) though this approach is not used by *Spike2*. However, this facility is not available in the CED 1401 A/D interface, and as controlled by the CED *Spike2* program (version 3.1) the hardware is not capable of sampling at a sufficiently high rate to permit software to perform the task. The capabilities of the CED 1401+ interface (a version of the 1401 based on a more powerful CPU) are greatly enhanced, however a 1401+ was only available for the final data capture session.

3.2.2 Waveform representation and data compression.

The sampling process is, however, only part of the story. The shape of the analog waveform can only be represented approximately by a series of instantaneous voltage measurements, and the actual voltage of the signal in between two such measurements most certainly cannot be represented accurately by a linear combination of the measured values. However, any waveform can be represented to any degree of accuracy required if it is reconstructed from its Fourier transform or the related Fast Fourier Transform (FFT) (Stremler, 1990). In this approach the sample values of the digitised signal are used to calculate the coefficients of the corresponding Fourier series and this is used to generate intermediate values. This can be extended by using the instantaneous frequency breakdown of the signal (given by performing a FFT) to identify "unimportant" frequency components (probably relative to the responses of the human auditory system) which are then discarded (giving a form of "lossy" data compression). This gives a simpler signal, which can be stored using fewer bits per sample or correspondingly a greater number of samples per time interval, at the expense of losing some information. One major drawback of this method is that the FFT calculation is too complex to implement in real time using only software - special hardware must therefore be available. The method of waveform reconstruction (and to a lesser extent compression) is however the basis of some digital audio devices.

The alternative approach to obtaining a more faithful representation of the analog signal is to step up the sampling rate. Taken to its limits, this permits a signal to be represented to any level of accuracy needed, provided only that the sample interval is small with respect to the time course of events of interest. Lossless data compression methods such as the Lempel-Ziv algorithm (Ziv and Lempel, 1977; 1978) can then be applied to the storage of the digital signal, without any concern that the simplifying assumptions used in lossy data compression methods are in some way altering the data.

3.2.3 The signal capture facilities available.

Having outlined the principles of signal acquisition, it is appropriate to consider the capabilities of the hardware and software available. As mentioned earlier, the A/D converter used was the CED 1401, a PC controlled unit providing up to 32 analog input channels and a small amount of onboard memory. This was controlled by CED's *Spike2* software (version 3.1) and this combination provided for a maximum data rate of 20 KHz **in total**, which had to be shared between all the active channels (compared with the data rate of 44.1 KHz per channel used by digital audio systems). This implies for a single channel recording that the maximum bandwidth of the signal is 10 KHz. In any consideration of the identification of nerve spikes by shape, sub-millisecond information is of importance, since the time course of the entire spike is of the order of one millisecond (Abeles and Goldstein, 1977). Even for single channel recordings of nerve spike event data, the sampling rate is therefore close to the low end of the acceptable scale. The use of a device such as the CED 1401+, which is capable of capturing short waveform events with high resolution as well as higher sustained data rates, would eliminate any concern about the quality of the recording. Indeed the data ultimately captured with a 1401+ was sampled at 25 KHz on each of two channels.

The output of the 1401/*Spike2* system is a data file containing the digitised values of the signal (to 12 bit accuracy). There are no inbuilt data compression mechanisms, with the consequence that data files (sampled at 20 KHz) are around 2.4 MB per minute of recording. This has the practical effect of encouraging the user to operate the system at lower sampling rates, with consequently lower bandwidth recordings.

3.2.4 Alternative matching software.

Many of the matching systems described in chapter 1 are software based, and a number of investigators indicated that the source code for their systems was available. Considering the extensive modification that would be required, the merits of re-using portions of source code from other projects are debatable, particularly when the code

was not designed to be used in this fashion. Indeed when source code is generated in an incompatible language, or is intended to run on different hardware or use different operating systems, the benefits of attempting to re-use it are largely outweighed by the practical problems of integrating it with new material.

With the standardisation of computer equipment, and its widespread availability in recent years, it is not surprising that commercial interest would develop in the provision of spike classification techniques. CED have taken an interest in such techniques and have incorporated a degree of spike classification capability into their *Spike2* software, but only for use with the 1401+. This is primarily an on-line solution (hence the requirement for the 1401+), using a version of the variable envelope template method described here, based upon templates generated using a clustering technique.

3.3 The problem expressed in terms of nerve spikes.

The signal data available was recorded on a four channel analog tape recorder, with up to three channels being used for nerve spike recordings. Each channel contains the electrical activity for all the neurons in one fibre and, in terms of the information content, these consist of discrete events of six categories occurring in any sequence. Events may overlap (see Fig 3.4) and the signals are subject to a degree of background noise, as well as alterations in the baseline voltage and the amplitude of the events (see Fig 3.5). The absolute amplitude of the spikes thus depends on the experimental conditions prevailing at any particular time. However the ordering of the amplitude of spike classes F1 to F6 remains the same. All of these features must be addressed to at least some degree by any matching process.

The window discriminator method used by Dr. Denheen and many others (Littauer and Walcott, 1959; Hermann *et al.*, 1962; Bradley *et al.*, 1967; Freeman, 1971) depends on setting threshold voltages to trigger events. This works well for the spike categories F3 to F6 but the two smallest spike categories F1 and F2 are difficult to distinguish on this basis. Depending on the properties of the electrode in each particular instance, the amount of noise, and the variations of voltage, it is often impossible to

separate F1 events from F2 events by this means (Harris-Warwick and Kravitz, 1984). Unfortunately the separation of F1 from F2 is the most significant classification of all, since the F1 neuron originates in the ganglion posterior to the ganglion (Wine *et al.*, 1974) from which the others originate (see Fig 3.6).

Clearly if the F1 and F2 spike waveforms or any other waveforms being analysed have the same shape with the same peak voltages from the same baseline on the same timecourse (within the margins due to background noise), then it will be impossible to distinguish between them on the basis of a single recording (although two or more recordings at separate parts of the nerve would add the possibility of separating events based on propagation speed (Schmidt and Stromberg, 1969; Roberts and Hartline, 1975)). Assuming that there **is** a difference between the spike waveforms, and that this difference cannot readily be established by using a threshold voltage, the shape of the waveforms must be considered.

3.4 The matching methods used.

This section will describe the mathematical and computational aspects of the matching methods employed by the software for this part of the project. The theoretical justification for each method and its advantages will be discussed in chapter 5, while the actual implementation will be discussed in section 3.6.2.

3.4.1 The variable envelope template method.

As a first approximation it is possible to automate an intrinsically manual approach to shape identification. The manual process under consideration is that of tracing the shape of the waveform of interest and adding an outline representing the acceptable level of deviation from that shape, comparing this with each waveform, and accepting or rejecting matches on the basis of the degree of overlap. Following the terminology of Schmidt (1984b), this approach is actually a contour matching process, since it is baseline independent, but considering the diversity of template methods it is a

somewhat artificial distinction. The automated equivalent of this is to select a section of waveform and define the maximum and minimum range curves based on the formulae:

$$R_{max} = V + (k_1 + k_2 D)$$

$$R_{min} = V - (k_1 + k_2 D)$$

where R_{max} and R_{min} are the range values, V is the value of the datapoint, D is the value of the derivative of the curve at the datapoint, k_1 and k_2 are user defined constants. The search is then performed by finding the minimum difference for each of the possible alignments between the curve defined by the maximum range and the curve of the waveform to be matched. Then the curve of the minimum range (offset by the minimum difference) is compared with the waveform curve, and the match is accepted if the range curve lies below or touches (but does not cross) the waveform curve (see Fig 3.7).

3.4.2 The minimum merit distance method.

A second, and more theoretically interesting, method was used as an alternative approach to the identification of waveforms. This uses the calculation of a "minimum merit distance" (MMD) for each template/waveform comparison according to the methods described by Kruskal (1983). The templates were defined as described above, but the search did not take account of the relative alignment of the template and waveform. The assumption was made that the MMD calculation would absorb slight variations from the default curve alignment (i.e. the alignment on which the template was defined). One difference with the standard MMD calculation was the use of a user defined range of perfect match values. This allows for noise effects to be discounted, as well as providing a consistency of behaviour between the two template methods.

The MMD measures the degree of similarity between the information contained in the template and the waveform, and this could be treated as a distance measure in substantially the same fashion as any other. However it was considered to be more useful to embed the match distance into a Boolean choice of the available templates, with the most similar (i.e. the comparison with the lowest MMD value) being selected. In these terms the templates are not useful as independent entities, there being an implicit

requirement that most if not all the waveform classes are represented by templates.

3.5 The process of creating a spike database.

As with the spot analysis system, the fundamental approach in the design of the signal analysis system (or Signal Viewer) was to split the process of data acquisition (via the 1401/*Spike2* system), data translation, template generation and data analysis into separate sections, each controlled by a separate program. Having described the data acquisition stage, this section will consider the generation of a database of nerve spike waveforms.

The use of a database of spike waveforms was selected in preference to direct operation on continuous waveform recordings for three reasons. First, it removed the dependence on a single data source. Nerve spikes are discrete events, and thus capable of being treated as separate entities, so this approach does not alter the key information contained in a continuous recording. Indeed much analysis is done on packets of waveform information, whether gathered by a digital storage oscilloscope or by a software package such as CED's *Sigavg* program. Thus restricting the range of data to continuous recordings would reduce the flexibility (and hence the utility) of the analysis software without providing corresponding benefits. Second, when this part of the program was designed it was hoped that other data acquisition methods capable of sampling at higher rates than the 1401 controlled by *Spike2*, such as use of the 1401+ or digital oscilloscopes or even direct control of the 1401 by custom software, would be available at some stage. These would not produce continuous waveform information, and consequently would not be compatible with an approach that required data in this form. Third, at the time the software for this part of the project was designed and coded, the only library available for accessing data files was for CED's "CFS" data file format. *Spike2* generates data in "SON" format, and it was necessary to use a utility program (supplied as a standard part of *Spike2*) to convert it to CFS format. The CFS access library proved to be less reliable than was desirable for a key component of any piece of software, and good defensive programming practices dictated that it should be removed

from the program kernel. It was therefore replaced by a set of custom extensions to the Ashton-Tate dBase 4 database file format, implemented through a customised library. This was the only other available framework for handling data, and was primarily suited to the handling of discrete data items.

The spike database creation utility thus takes in a CFS format data file together with user defined information about the channels to be captured, a list of trigger voltages to use, and the pre- and post-trigger timing intervals. When an event is detected, the relevant sections of all the channels are stored, together with the time and the identity of the channel causing the capture event. Once created, this database is passed to the next stage of analysis as described in the next section.

The actual spike event recognition mechanism is simply a positive going threshold event, i.e. any situation where one datapoint which lies below the threshold is followed by one on or above the threshold. This differs from the event definition mechanisms used by CED (Greg Smith, CED Ltd., personal communication), in that there is neither a second negative going threshold crossing required within a specified time, nor is there a "dead" interval before the next possible positive crossing. The latter is a more sophisticated approach, but in the context of the data available this mechanism was considered to be over engineered. This, of course, is much less true for CED's software requirements, hence the difference in approach. Additionally, the mechanism as used by *Spike2* version 4 for the capture of isolated spike events is understood to depend on processing occurring within the 1401+ rather than in the PC, with surplus waveform data being discarded. Hence the performance and reliability requirements are significantly greater for *Spike2*.

3.6 The implementation of the classification methods.

This section will consider how the final version of the Signal Viewer program operates, consideration of design and development issues being given in section 3.8.

3.6.1 The signal analysis program described.

As ultimately implemented the Signal Viewer consisted of two separate but closely related programs, the Template Generator and the Result Viewer, both of which operate on the spike waveform database generated by the import utility. The Template Generator allows individual spike waveforms to be viewed (see Fig 3.8A) and templates to be defined (in accordance with the formula in section 3.4.1), edited or deleted. These are then stored in a template database. The Result Viewer allows the search method to be selected, performs the search, generates the match database and displays portions of the spike database with the spikes in their correct temporal alignments (see Fig 3.8B). Individual channels can be copied, and logical filtering of the display for each channel can be effected. Information about the timing and numbering of each filtered event can be generated in a form suitable for importation into a spreadsheet such as *Microsoft Excel.*

3.6.2 The implementation of the match methods.

Both search methods use the same format of spike and template databases, and generate a match database of the same form. All of the routines are coded in C, and the calculations make extensive use of floating point arithmetic (with consequent performance implications). In principle both methods rely on off-line analysis, that is analysis which is not performed in real time, either because it cannot be performed in chronological sequence or because it cannot be performed in the interspike intervals.

3.6.3 The variable envelope template method in detail.

In this section the principals and implementation of the variable envelope (VE) template method will be described.

The VE template method is a contour fitting algorithm in which the shape of the data waveform is compared with two contours (see Fig 3.9), corresponding to the upper

and lower bounds for the data waveform, which can be synchronously offset in both dimensions. In this implementation a series of comparisons are performed to find the optimal temporal alignment between the data waveform and the upper bound, resulting in an array of voltage differences corresponding to the possible alignments. The **largest** of these corresponds to the near-optimal temporal alignment which allows both upper and lower bounds to be offset by a value which will tend (but not in all cases be guaranteed) to maximise the distance between the data waveform and the lower bound without permitting the data waveform to cross the upper bound. (see Fig 3.9B). If the data waveform lies entirely above the lower bound then this is classed as a match, otherwise it is rejected. The search is then repeated on the next (valid) template / waveform comparison. A flow diagram of this is given in Fig 3.10.

This method is computationally cheap, requiring few calculations per datapoint, and operates fastest on templates which are close to the data waveform in length. It is in principal sufficiently fast to be implemented in real time.

3.6.4 The minimum merit distance method in detail.

This section will consider the principles and implementation of the minimum merit distance (MMD) template method.

This method is a modified implementation of the algorithm described by Kruskal (1983), in which two sequences are compared by determining the "cost" of inserting, deleting or substituting sequence elements in order to make the two sequences equal. The scheme of sequence alterations corresponding to the minimum cost is the optimal transformation, and the cost represents the distance between the information contained in the two sequences. This is therefore an intrinsically baseline dependent method. Its operation is perhaps best understood by considering a simple example as illustrated in Fig 3.11, and its implementation by considering a flow diagram (Fig 3.12).

In detail, two arrays are created - one containing a total of T template waveform elements, and the other containing D data waveform elements, but this will normally be restricted to the sub-sequence corresponding to the one on which the template was

defined (giving T=D). A two dimensional matrix of size T x D is created to hold the intermediate cost values, along with necessary control arrays. The matrix is filled with the minimum cost values according to the rules given by Kruskal (1983), with the exception that two elements are deemed to be equal (i.e. have a substitution cost of zero) if they differ by less than a user specified amount. The matrix entries represent the comparison of the sub-sequences up to the current positions and should increase from left to right and from top to bottom (see Kruskal (1983) for examples of this). The entry in the final column of the final row represents the cost of transforming the template into the data waveform (or vice versa), and is the significant value. This process is repeated for each valid template, resulting in an array of cost values. The **minimum** value in this array corresponds to the template which best matches the data waveform, and this correspondence is recorded in the match database as the preferred match - all other templates being rejected. As with the VE template, the search is then repeated for the next available data waveform. A flow diagram of the overall process is given in Fig 3.13.

Even from this brief account three points should be obvious. First, the cost increases in proportion to T x D. Second, the use of templates of varying lengths will influence the cost of the transformation (small templates necessarily have a lower transformation cost than large ones in this implementation). Consequently templates of the same length should be used. A normalisation of the cost could be performed, but this was not implemented, preference being given to the use of standard length templates. Third, several calculations are required per matrix entry, with the consequence that the whole process is computationally expensive and correspondingly slow. The first and third points taken together imply that as the waveform resolution improves, the speed of the method will decline in a non-linear fashion, with consequent performance implications.

3.6.5 Data flow and operation of the matching kernel.

The previous two sections outline the operation of the match methods. However,

this is only part of the story. The kernel of the matching system also contains data management facilities, and it is impossible to understand the operation of the Result Viewer without some broad explanation of the way in which data flows through the kernel.

Waveform data is held in a waveform database, the templates are held in a template database, and the results of matching the templates to the waveform data are held in a match database (which has as many records as the waveform database and as many fields as the template database). The waveform and template databases are opened by the user prior to initiating a search. When a search is started, a match database is created and initialised. A search is performed on all non-deleted data waveforms using the current selection of non- deleted templates. The selection mechanism normally restricts the template search to meaningful combinations (templates and data derived from the same data channel) but can be altered by the user to permit a search of all the data channels for all the templates. Matches are determined as described in the two previous sections and logged to the match database as a three state Boolean variable (match / no match / no search).

Following the template search, the user is able to copy the display of any data channel and apply a display filter consisting of any logical combination of match events to the channel. At its simplest this would allow a channel containing two waveform classes to be copied so that there were three separate display channels, corresponding to template one, template two and neither template. This information can then be outputted (in the form of columns of times of events) along with summary statistics to an ASCII file, which can then be imported into another package (such as *Spike2*).

3.7 Validation.

Validation and operational testing of software is a major part of the process in any development (Adrion *et al.*, 1982). Some software is effectively self validating, insofar as it either operates or it fails to operate, and everything else is a reliability issue rather than a correct operation issue. For this type of software, where failure will not

result in significant adverse consequences (such as loss of life) and where the anticipated user base is small, this level of verification is appropriate (McConnell, 1993) - and applicable to parts of the Signal Viewer. However, the data analysis part of the Signal Viewer requires specific validation. The Signal Viewer has adequate amounts of real and synthetic data available for analysis, and more significantly has data analysed by alternative methods for comparison, as well as data from experiments providing a physical basis for separating F1 and F2 spike waveforms. Various validation tests were carried out as detailed in the following sections.

3.7.1 Application to comparative data.

Data of a suitable nature was captured as described above, and analysed by Dr. Denheen in accordance with the methods described by her (Denheen 1992). The same data were independently analysed by myself using a variety of templates for both search methods. The results of the two analyses were then compared by overlaying the event output from the Signal Viewer back onto the original data file (as supplementary event channels), and new events were generated to mark differences between the two analyses. This was performed using the macro language facilities provided within *Spike2*.

3.7.2 Application to F1 censored data.

In experiments previously conducted by Dr. Denheen, the ventral nerve cord was cut posterior to the ganglion from which the relevant third root nerve originated (see Fig 3.6). This had the effect of cutting the F1 neuron (and hence removing F1 activity) while leaving F2 to F6 intact. Therefore, the accuracy of the template search for F2 could be established by considering the number of false F1 matches, assuming that the cutting process did not itself have an effect on the spike waveforms.

3.7.3 Application to tagged data from the crayfish cuticular stress detector.

In experiments previously conducted by Dr. Cornelia Leibrock (1993) on the sensory system of the crayfish *Procambarus clarkii*, recordings were made extracellularly from the anterior distal root (adr) nerve and intracellularly from a sensory terminal (CSD1-t) of the cuticular stress detector one (CSD1) nerve. The CSD1 axons form a part of the adr, hence the adr recording will contain an extracellular record of the same activity recorded from the CSD1-t. This gives a means for the independent verification of template matched CSD1 spikes.

A section of the adr recording was therefore digitised, suitable MMD templates were defined, and the results of the search compared event by event with the activity of the CSD1-t.

3.7.4 Application to tagged data from the crayfish coxo-basal chordotonal organ.

A further data set containing both an extracellular and an intracellular channel was obtained from Dr Daniel Cattaert of the Centre Nationale de la Recherche Scientifique (CNRS) in Marseille. The intracellular recording is from the fifth thoracic ganglion of the crayfish *Procambarus clarkii*, and comes from the central terminals of one of the sensory axons of the coxo-basal chordotonal organ in the fifth walking leg, while the extracellular recording is from a tungsten wire electrode pressed against the sensory nerve coming from the corresponding chordotonal organ, and sealed in place by vaseline. Again the intracellular recording provides for independent verification of template matched spikes.

A section of the extracellular recording was therefore digitised, suitable MMD templates were defined, and the results of the search compared event by event with the activity recorded by the intracellular electrode.

The same section was also searched using the spike identification facilities in *Spike2* for *Windows*. Templates were generated using the inbuilt facilities and these were used to search the extracellular channel. Search results which failed to identify the key spike categories were discarded.

3.7.5 Sensitivity testing.

Sets of variable envelope templates were defined using different k1 and k2 parameters (as defined in section 3.4.1) and the number of matches for each combination was established. This identifies the range of acceptable margins of error.

3.7.6 Application to synthetic data.

Three data sets were prepared using waveform averaged F1, F2 and F3 data (as identified) from one of Dr. Denheen's experiments. These spike classes were then used to generate synthetic data sets in which the actual spike identities were known. Three different protocols were used to generate these data sets.

In the first protocol, each spike event was chosen randomly from one of the three classes, and a predetermined amount of normally distributed noise (generated using an implementation of the Box-Muller algorithm (Press *et al.*, 1992)) was added to the spike waveform. The standard deviation (s.d.) of the noise was varied over a range from zero (i.e. an infinite signal to noise (S/N) ratio) to a level where the signal was badly degraded (approximately zero S/N ratio). Templates for each spike class were generated on the zero s.d. data and the MMD search method was used to identify matches. These were then compared with the actual events to give absolute performance comparisons. The three basic spikes are shown in Fig 3.14A, and typical examples of the noise degraded F1 spike are shown in Fig 3.14B.

The second protocol was intended to test the MMD search method's ability to distinguish overlapping spike events. Spikes were selected randomly as before, and noise was added as before (though only four different noise levels were used), and a second (randomly selected) spike of one of the two remaining classes was added. The second spike could occur prior to the first, at the same time, or subsequent to it. This data set was searched as before, and the results again compared to the known events. Examples of the overlap events are shown in Fig 3.14C.

The third protocol was intended to test the MMD search method's capabilities in

an artificial situation, namely its performance when presented with spikes of intermediate shape (that is linear combinations of F1 and F2). Spikes were selected as before, with noise as before but the F1 and F2 spikes were combined using the formula rF1 + (1-r)F2 according to predetermined ratios (r). Events were described in terms of the dominant component (i.e. F1 if r > 0.5). F3 spikes were left as single entities. Again the data set was searched and the results compared to the known events. Examples of the hybrids are shown in Fig 3.14D.

3.8 Design and implementation issues.

As outlined in chapter 2 the design and implementation process should move from a consideration of the goals of the software, through a process of formally specifying the structure and functions of the software, to implementation on the specified development system. This is followed by integration with third-party components, and an iterative testing and debugging phase. The smooth operation of this process depends on two factors, there being no fundamental design changes (such as major changes of target operating system) and on the expected facilities being available when anticipated (McConnell, 1993). The overall effort obviously also depends on the amount of pre-existing material available for re-use, the less that has to be designed and coded the smaller the development - and the faster it can be realised (McConnel, 1993). In this context there is a major difference between the effort required to add an extra module to an already existing framework program such as CED's *Spike2* and the effort required to design and implement the framework as well as the analysis module. The only possibly available framework program was in fact CED's *Spike2*, but as access was not provided to either *Spike2* source code or object modules, a framework program had to be developed prior to any implementation of the matching algorithms.

3.8.1 The design of the signal viewer.

The design of the framework program as an entirely new piece of software

allowed a relatively free consideration of its design. The program had to be able to display spike waveforms and permit user definition of templates, and consequently had to use a graphic display. The use of a mouse was also considered to be indispensable for an adequate user interface which handled graphic objects. The general design selected was an event driven system using a simple *Windows* like GUI. Internally the program was structured to use event handlers similar to those used in *Windows* applications, and in other respects to rely on similar API functionality. The intention of this approach was to leave the way open for the program to be turned into a full-fledged *Windows* application if suitable facilities became available and if the programming (or other) requirements justified this development. Use was made of existing libraries where possible, but new facilities were developed using OOP techniques. It would, however, be inaccurate to suggest that the new facilities were completely object oriented.

3.8.2 The development facilities initially available.

The development system initially available was the Microsoft C6 system described in section 2.8.1. This was a DOS real mode development system, which had significant consequences for the design and implementation of the software. As discussed in section 2.2.1, DOS does not provide the high level API functions required for even a simple GUI, but more significantly limits memory availability to 640 KB. The API constraints were dealt with by keeping the requirements for API functions to a minimum, and providing these in as simple a way as possible.

From an early stage it was apparent that memory availability would be crucial in determining the operational status of the finished program. However, introducing the high level API functions only aggravated the existing problems of limited memory availability. The use of a DOS extender (as discussed in section 2.3.3) would have alleviated these problems without requiring recoding. However, this only became available after coding was completed, resulting in major structural and debugging problems.

3.8.3 Additional development facilities.

Given that the program had significant stability and usability problems, it was necessary to undertake a further stage of debugging and code restructuring to generate an operational program. There were two possible approaches to this problem.

First, the DOS extender could be used with the C6 development system to undertake debugging. Final code modifications and enhancements could then be implemented as appropriate. The advantage of this was the direct use of existing code without modifications required for a change of operating system. The drawback was the limitation of a "home made" GUI without multiple parent windows, child windows, message or dialog boxes, or any form of text input or other standard interface controls. Additionally, the Phar-Lap API and development tools were relatively unfamiliar, which necessarily adds to the development time required.

The second possible approach was to use the *Borland C++* development system to generate a *Windows* based program. This approach had the advantage that the system was already familiar, and necessary hardware and software facilities were available. The standard interface objects for *Windows* are also much more extensive than anything available to a "home made" GUI. Also the quality of the IDE and the utility of both the IDE debugger and the Turbo Debugger enabled levels of productivity (in terms of work per unit time) which would be impossible with the C6 / Phar-Lap combination. The drawback was the requirement to sort out various differences between the Signal Viewer dialog interfaces and message handling system and the standard methods used by *Windows*.

Initial attempts to debug the Signal Viewer using the C6/Phar-Lap combination indicated that the extent of the unresolved problems was such that the time saved by using the Borland system would more than offset the conversion time required, and a better user interface would result (almost) as a by-product of this conversion process. This was therefore the approach ultimately used, and the consequence was a reasonably stable system with a good user interface.

Figure 3.1

Diagrammatic representation of a dorsal view of the ventral nerve cord and superficial flexor muscles of a Norway lobster.

VNC - ventral nerve cord.

r1 - the first ganglionic root which innervates the swimmerets and contains motor and sensory axons.

r2 - the second ganglionic root which innervates the slow and fast extensor muscles, stretch receptors and sensory hairs of the body wall.

Sr3 - the superficial branch of the third ganglionic root which innervates the superficial flexor muscles.

SFM - superficial flexor muscles.

G - ganglion.

Redrawn after Denheen (1992).

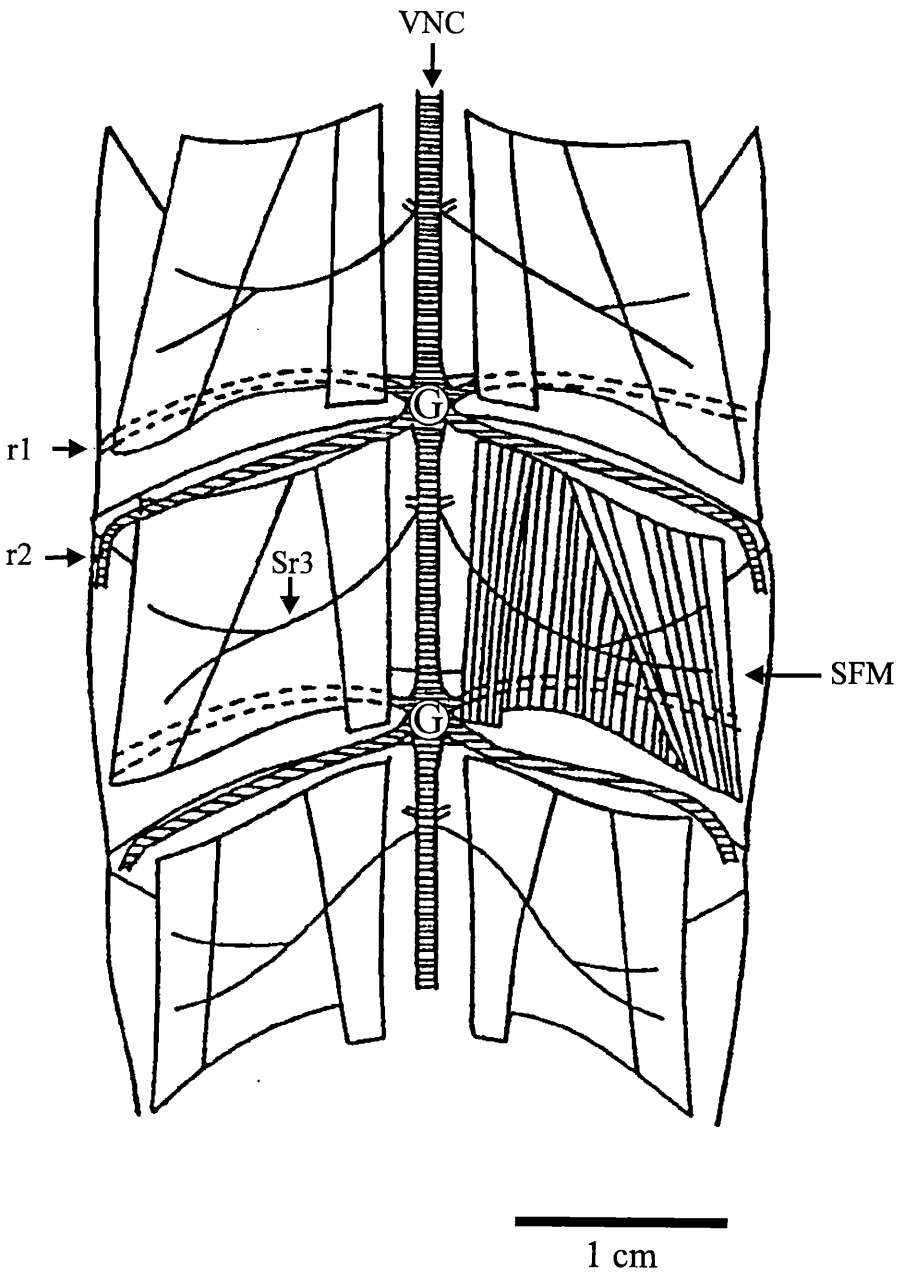

VNC
G
r1
r2
Sr3
SFM
G
1 cm

Figure 3.2

Production of event markers by single window discrimination performed using CED's Spike2 software. Event channels were generated by voltage thresholding for the voltage thresholds representing the lower and upper limits of voltage window w1 and the upper limit of voltage window w2. These were then logically subtracted to give the event marker channels 2 and 6, the residual events being in channel 7. This analysis although simple requires the use of a custom written Spike2 macro language program to perform the subtraction which is outwith the basic capability of Spike2.

Channel 1 - analog waveform channel.

Channel 2 - events exceeding upper threshold of event window w2.

Channel 6 - events lying between lower and upper thresholds of event window w1.

Channel 7 - events lying between lower and upper thresholds of event window w2.

w1, w2 - approximate bounds of the voltage windows used.

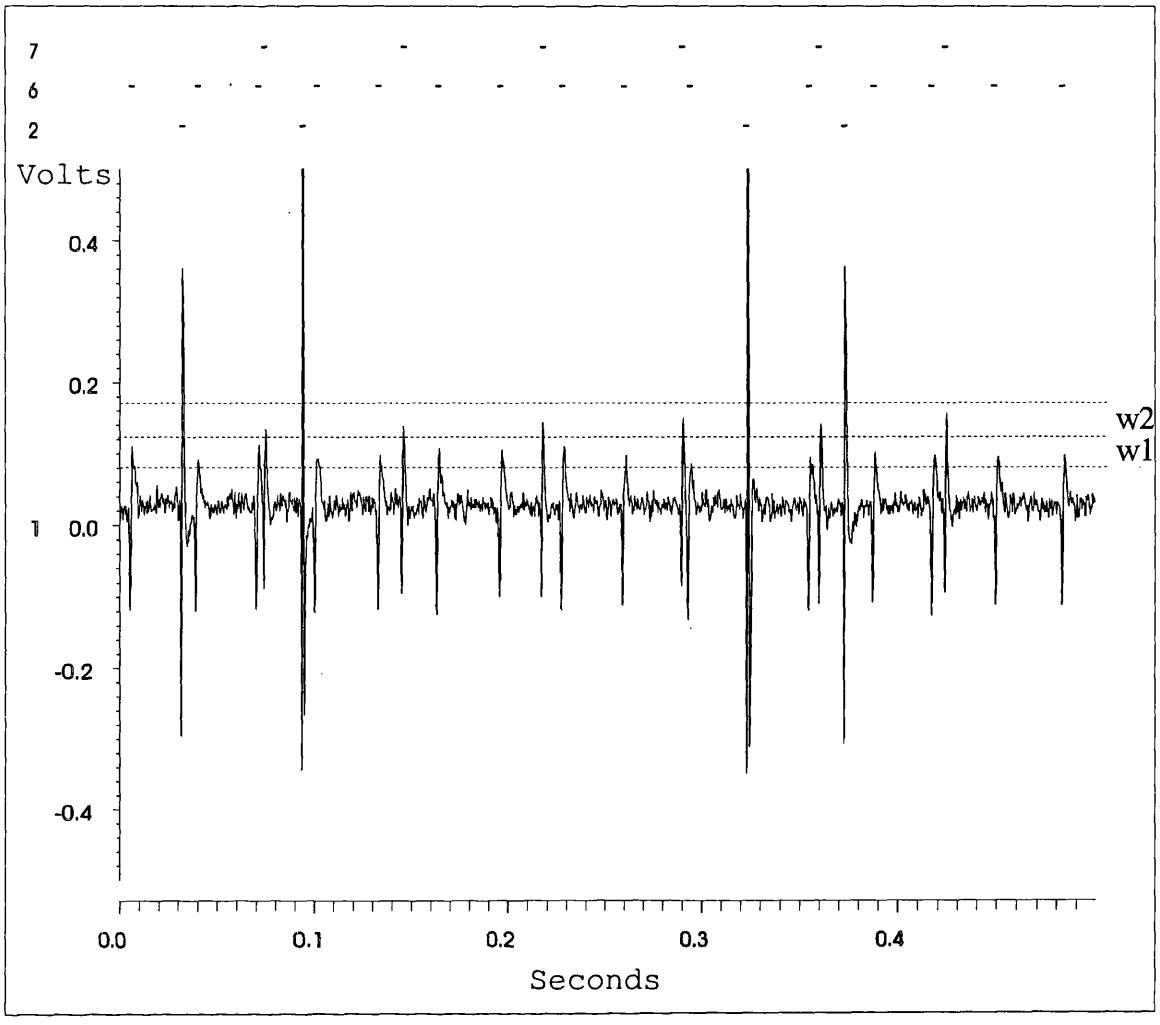

7
6
2
Volts
0.4
0.2
1 0.0
-0.2
-0.4
w2
w1
0.0
0.1
0.2
0.3
0.4
Seconds

Figure 3.3

Sampling at twice the bandwidth provides the capability of reconstructing a waveform accurately from its frequency components. However, performing a linear interpolation on the sample values is not an accurate way of generating intermediate values.

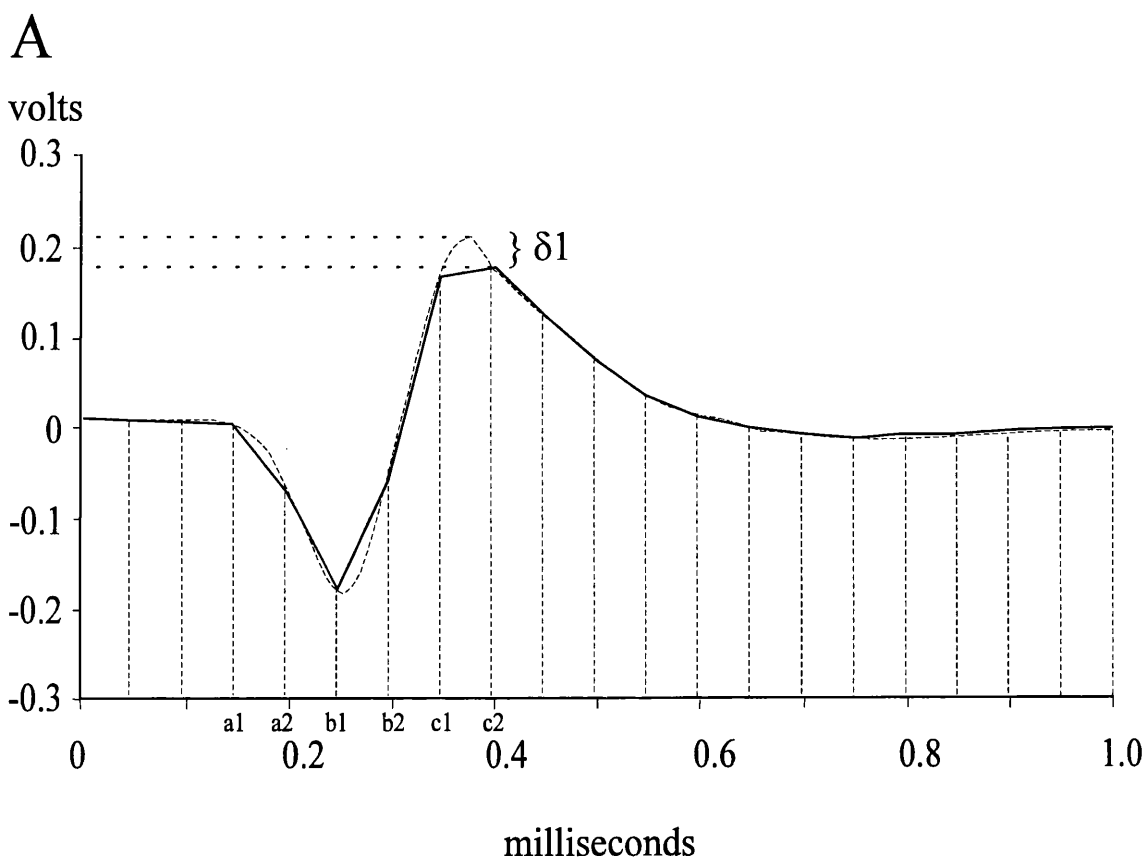
A. The effect of linear interpolation on the instantaneous peak value of a spike waveform. There is a displacement of the peak in both voltage and time, even with an adequate sampling rate, as well as significant deviation between the sample and reconstructed waveforms on sections a1 to a2, b1 to b2, and c1 to c2. Note however that the rest of the waveform is adequately represented.

B. Oversampling to prevent loss of instantaneous peak values. The introduction of sample points s1 and s2 between c1 and c2 results in a much better approximation of the waveform, but triples the sampling rate. For a system in which the sampling rate is already at the limit of the hardware capacity this is not a feasible option, and in any case wastes data storage capacity.

Solid line - linear interpolated waveform.

Dashed line - sample waveform.

Vertical dashed lines - sample points.


A
volts
0.3
0.2
0.1
0
-0.1
-0.2
-0.3
} δ1
a1 a2 b1 b2 c1 c2
0 0.2 0.4 0.6 0.8 1.0
milliseconds

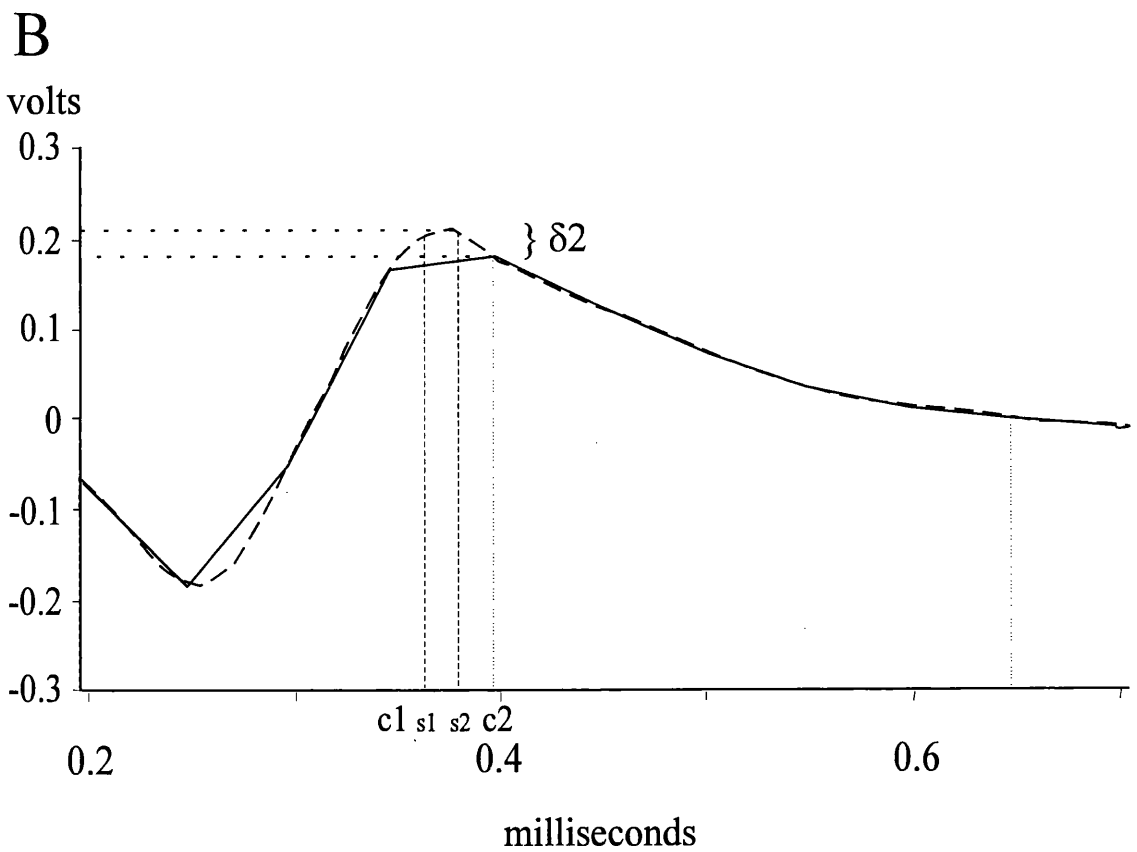

B
volts
0.3
0.2
0.1
0
-0.1
-0.2
-0.3
} δ2
c1 s1 s2 c2
0.2 0.4 0.6
milliseconds

Figure 3.4

A typical spike overlap event, in this case probably between a F1 and a F2 spike.

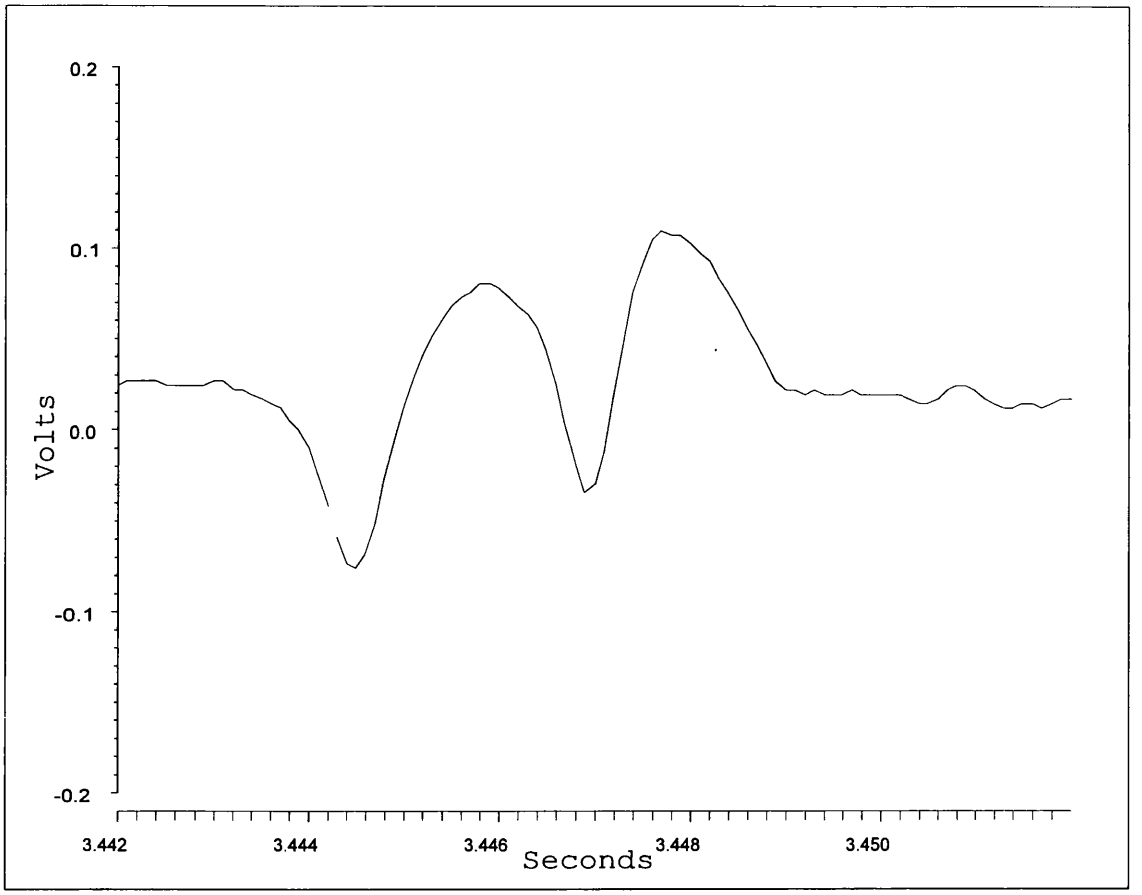

0.2
0.1
0.0
-0.1
-0.2
Volts
3.442
3.444
3.446
3.448
3.450
Seconds

Figure 3.5

Natural variation of the amplitude of spikes during a recording of the *Nephrops* system. No deliberate alteration of the electrodes or the saline was effected during the course of this recording.

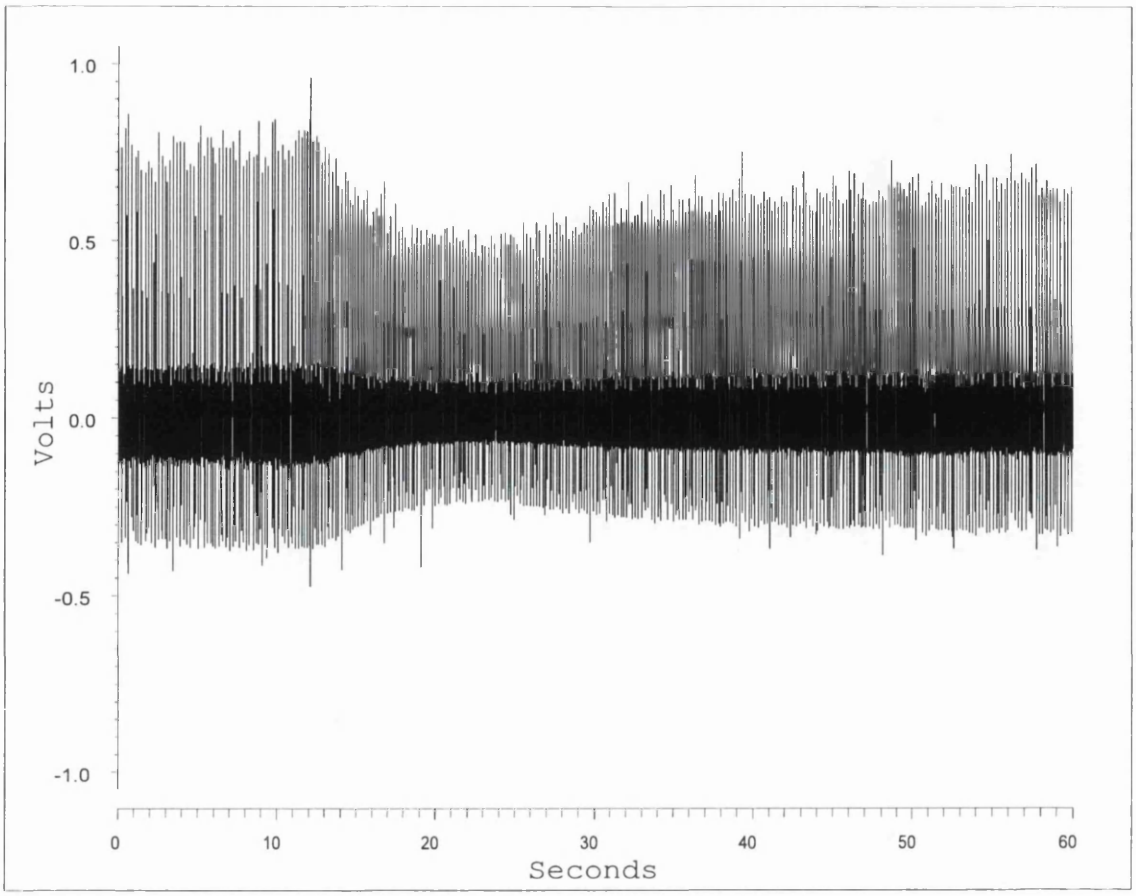

1.0
0.5
0.0
-0.5
-1.0
Volts
0
10
20
30
40
50
60
Seconds

Figure 3.6

Camera lucida drawing of a cobalt backfill of one superficial flexor root. The backfill shows staining in six motor neurons, labelled according to the size of soma. Five of these are located in the ganglion anterior to the superficial flexor root (F2-F6), while the remaining motor neuron, F1, has its cell body located within the posterior ganglion. Cutting at the location indicated selectively inactivates F1.

Scale bar = 200 μm.

Redrawn after Denheen (1992).

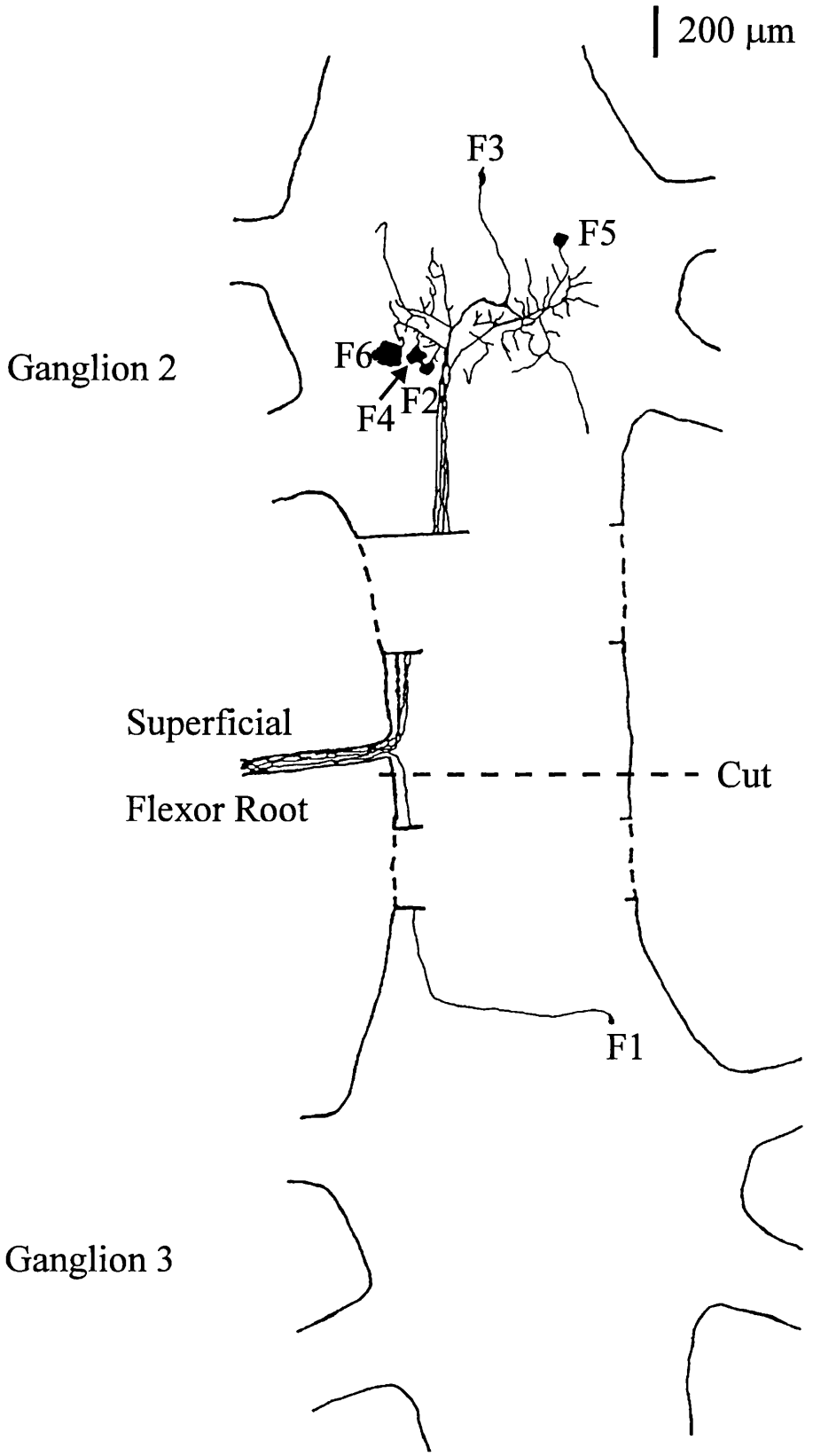
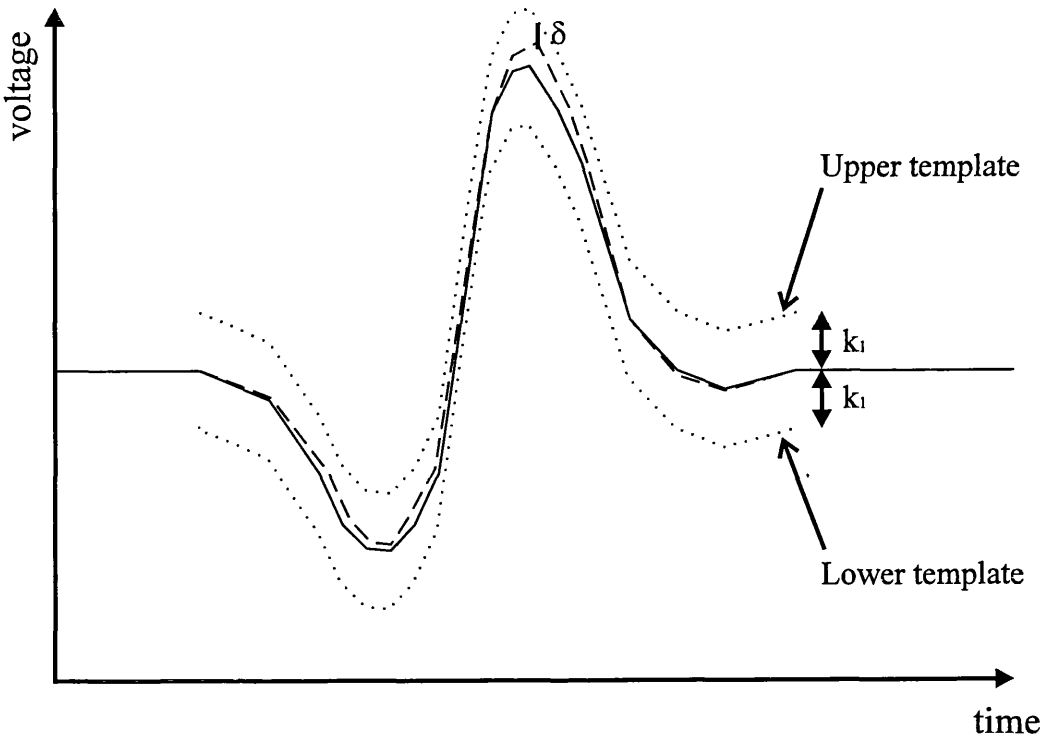

200 µm
F3
F5
F6
F4
F2
Ganglion 2
Superficial
Flexor Root
Cut
F1
Ganglion 3

Figure 3.7

The definition and operation of a variable envelope template. The upper and lower templates (dotted lines) are defined with respect to the template waveform (solid line) by respectively adding and subtracting a constant (k1) plus a second constant (k2) times the derivative of the template waveform. In this diagram k2 = 0 and is not shown.

The value δ is the minimum difference between the upper template and a data waveform (dashed line), here represented at a diagrammatically convenient point. A δ value is found for each possible alignment of upper template and data waveform, the optimal alignment between template and data waveforms being the one yielding the maximum δ.



Template waveform

Data waveform

Template

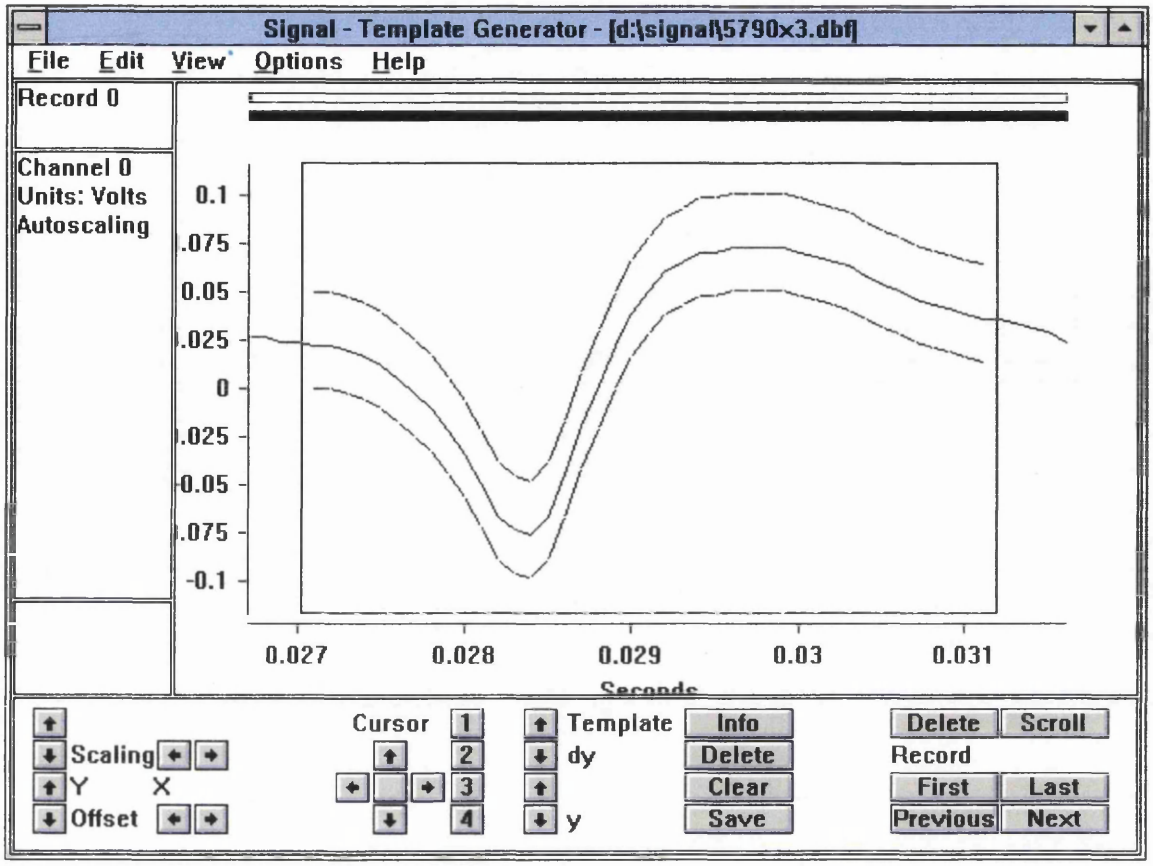
Figure 3.8

A. Screenshot of the template generation module, showing a template in the process of being created. The application follows normal *Windows* style guidelines for a single document interface window with the exception of the enhanced toolbar along the bottom of the window. This is designed more along the lines of an oscilloscope control panel than a conventional toolbar and, in addition to the display and template generation facilities, provides controls for displaying on-screen markers giving the voltage and time for the specified point.

B. Screenshot of the result view module showing the result of a template search. The data file contains one channel of data (channel 0), and this has been duplicated (channel 1). The data displayed in channel 0 has then been filtered according to the results of the template search, in this case to show the spike events matching the F1 template. This process of channel duplication and filtering can be extended until all of the template classes are displayed.

The inset shows a magnified version of one of the nerve spikes.

A


Signal - Template Generator - [d:\signal\5790x3.dbf]
File Edit View Options Help
Record 0
Channel 0
Units: Volts
Autoscaling
0.1
.075
0.05
.025
0
.025
-0.05
.075
-0.1
0.027
0.028
0.029
0.03
0.031
Seconds
Scaling
Y X
Offset
Cursor 1 2 3 4
Template
dy
y
Info
Delete
Clear
Save
Delete Scroll
Record
First Last
Previous Next

B

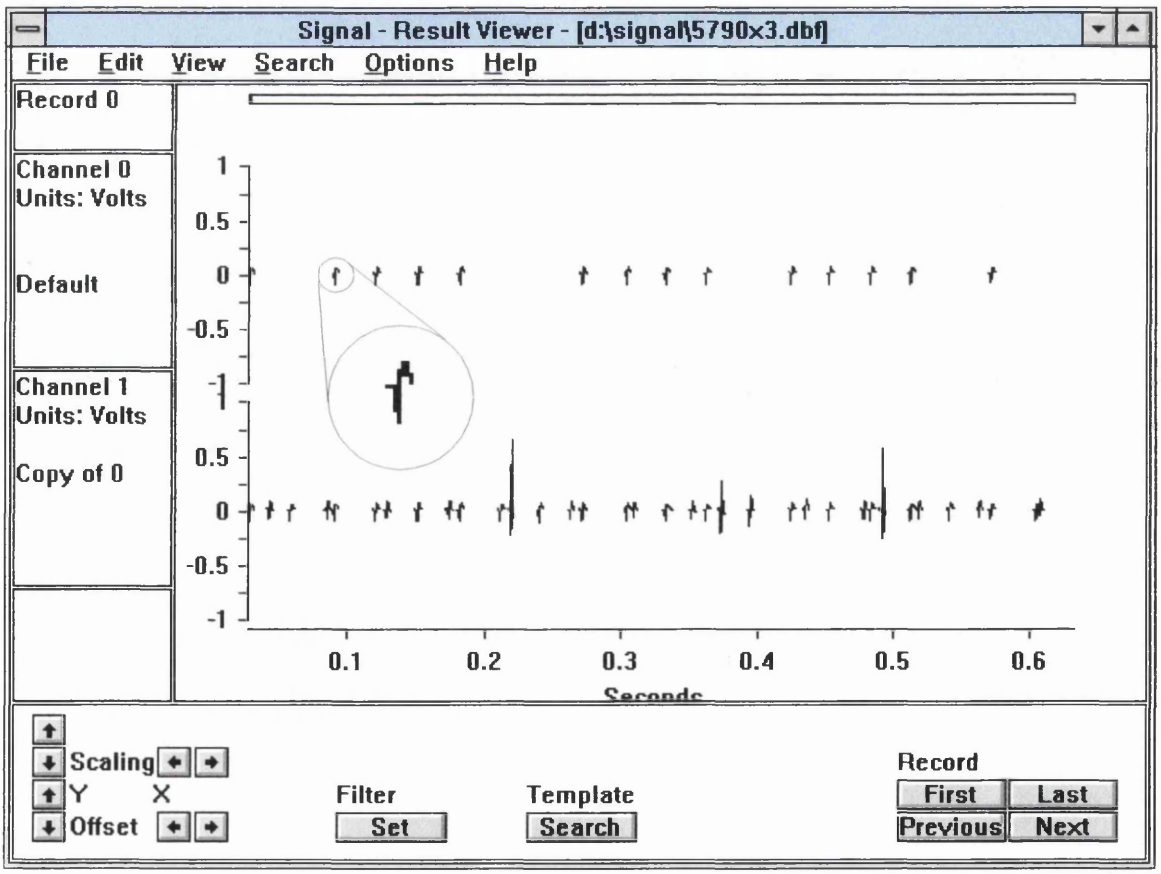
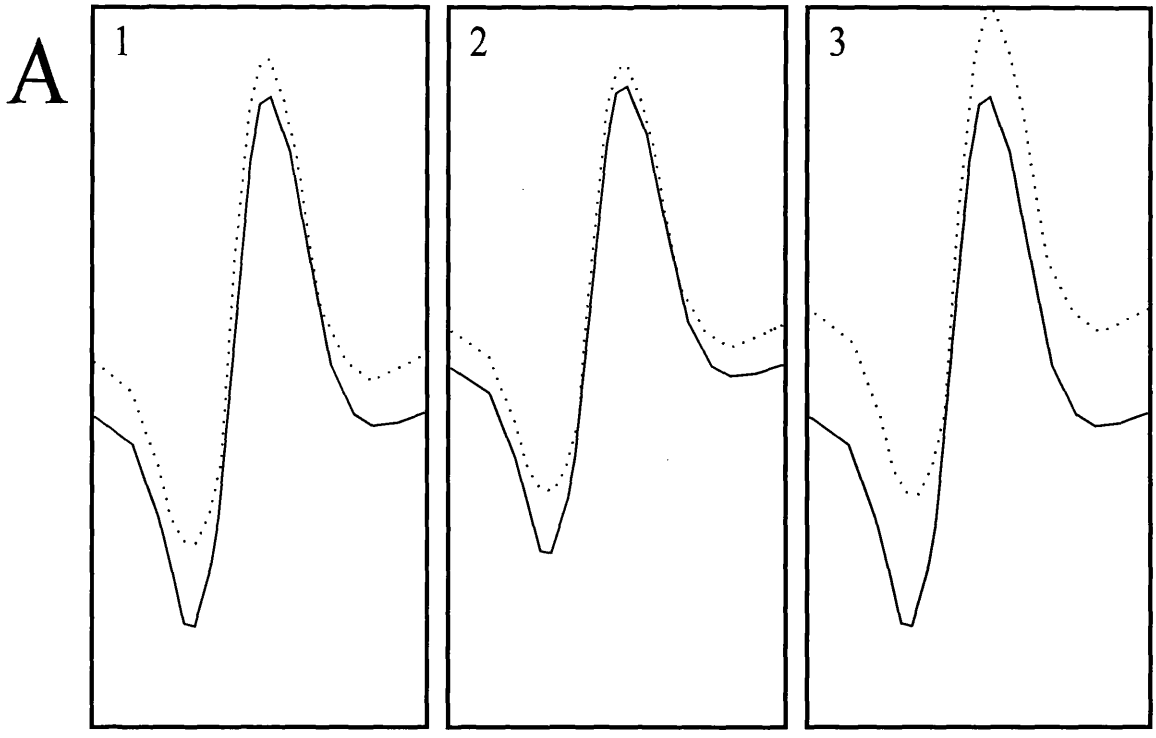

Signal - Result Viewer - [d:\signal\5790x3.dbf]
File Edit View Search Options Help
Record 0
Channel 0
Units: Volts
Default
Channel 1
Units: Volts
Copy of 0
1
0.5
0
-0.5
-1
0.1
0.2
0.3
0.4
0.5
0.6
Seconds
Scaling
Y X
Offset
Filter
Set
Template
Search
Record
First Last
Previous Next

Figure 3.9

The variable envelope template matching process in a diagrammatic form.

A. The upper template (dotted line) and a data waveform (solid line) for the optimal alignment (2) and the alignments one datapoint left (1) and right (3) of the optimal. The δ value (as defined in the previous figure) has been subtracted from the upper template in each instance.

B. Comparison of the data waveform with the lower template (lower dotted line), from which δ has again been subtracted, for a waveform match (1) and non-match (2). A match is found when the data waveform touches or lies between the template waveforms but does not cross either.


A
1
2
3

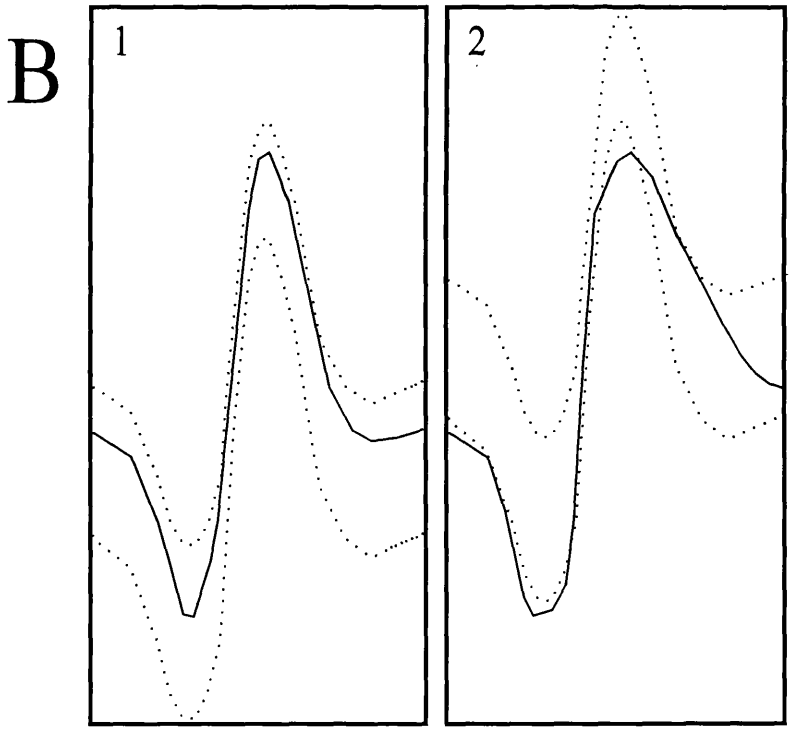

B
1
2

Figure 3.10

Flow diagram showing the principle stages the user would pass through in the course of a typical analysis session for either template comparison mechanism. Due to the event driven nature of this software, multiple execution paths are actually possible, but these secondary paths have been omitted for clarity.

This diagram also omits the program logic, which involves many processing stages and considerable branching, and which could not be adequately represented in a simple flow diagram.

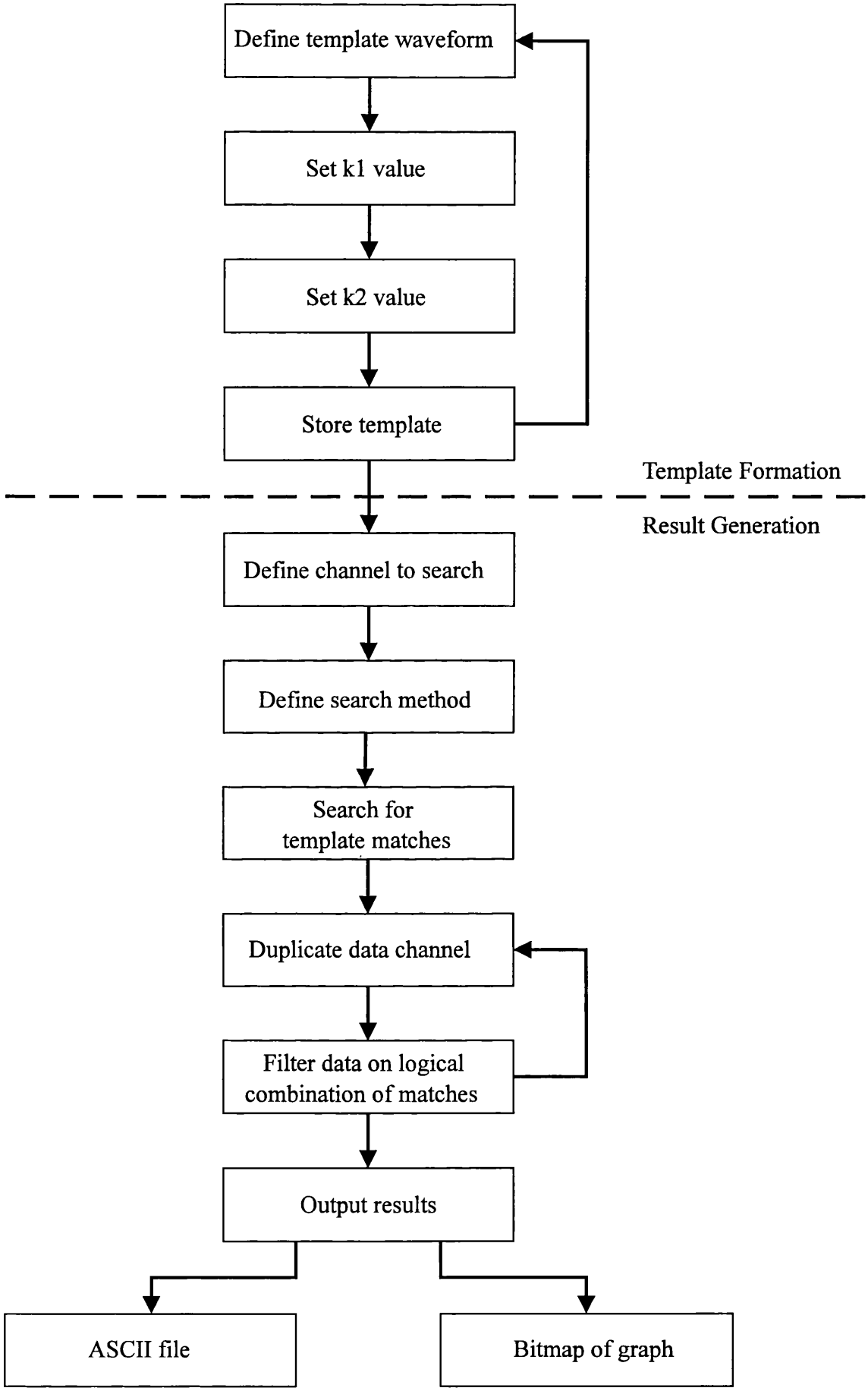

Define template waveform
Set k1 value
Set k2 value
Store template
Template Formation
Result Generation
Define channel to search
Define search method
Search for
template matches
Duplicate data channel
Filter data on logical
combination of matches
Output results
ASCII file
Bitmap of graph

Figure 3.11

Diagram showing the basic operation of the minimum merit distance comparison method on two arbitrary strings of characters (α and β). This is intended to show conceptually how the algorithm works rather than how it is implemented computationally.

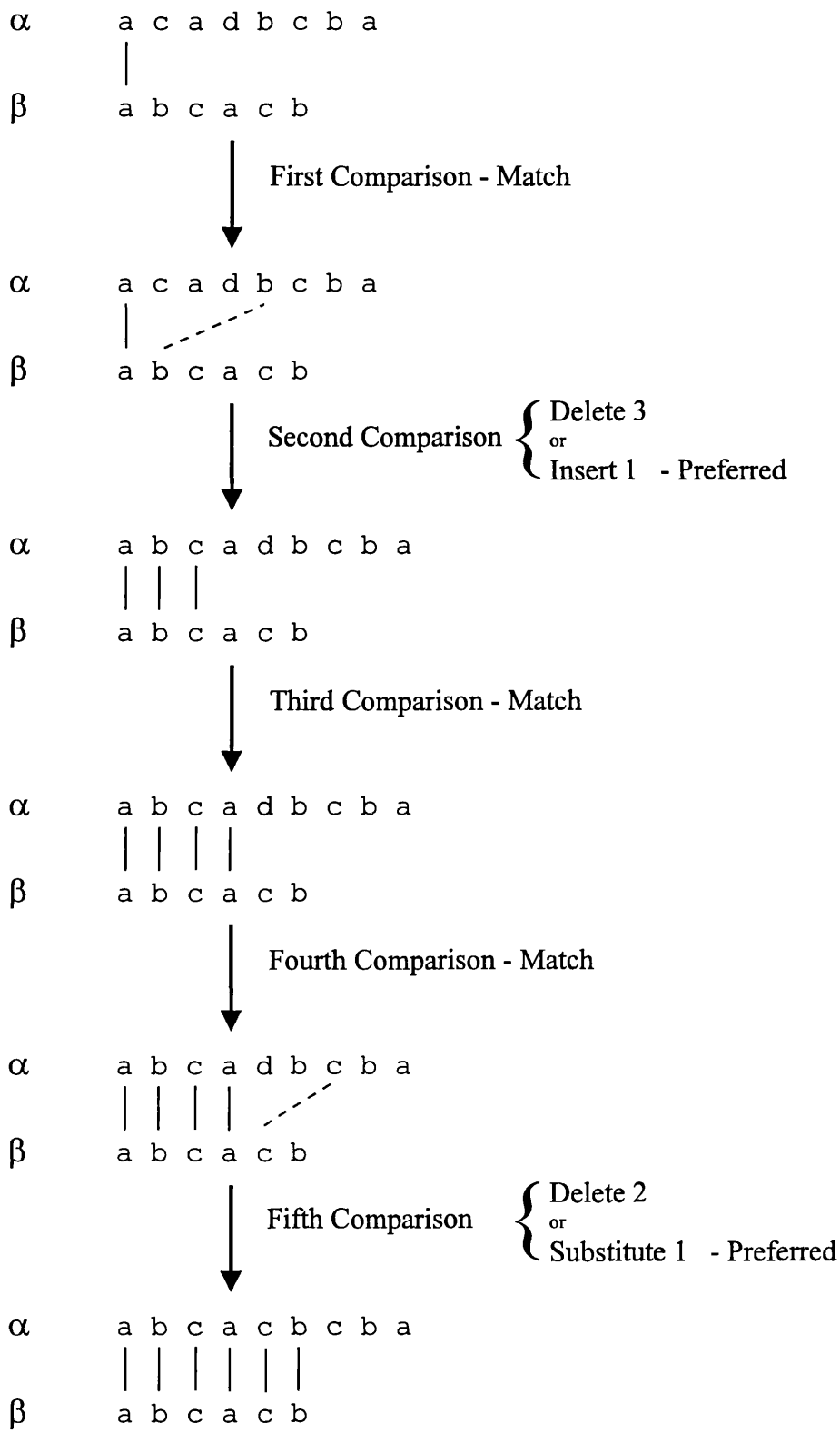

α a c a d b c b a
β a b c a c b
First Comparison - Match
α a c a d b c b a
β a b c a c b
Second Comparison
Delete 3
or
Insert 1 - Preferred
α a b c a d b c b a
β a b c a c b
Third Comparison - Match
α a b c a d b c b a
β a b c a c b
Fourth Comparison - Match
α a b c a d b c b a
β a b c a c b
Fifth Comparison
Delete 2
or
Substitute 1 - Preferred
α a b c a c b c b a
β a b c a c b

Figure 3.12

Flow diagram showing in detail the stages of the minimum merit distance algorithm. The level of detail is equivalent to pseudocode, however this diagram omits code level details of program operation, such as memory allocation, initialisation, error trapping, and resource clean-up stages.

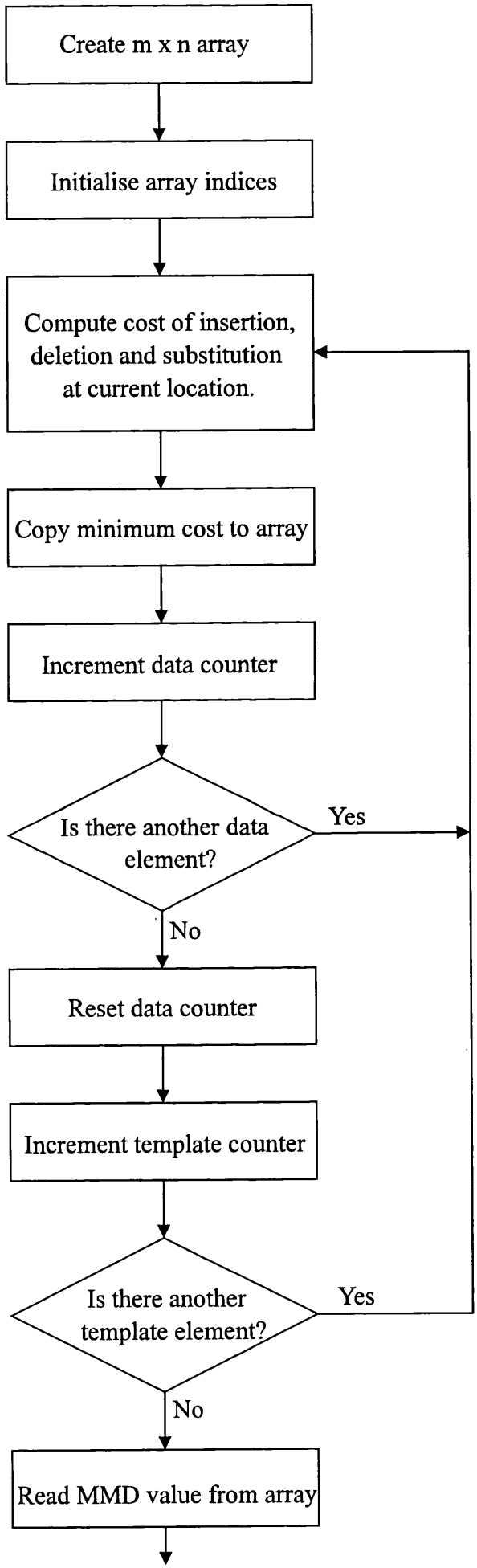

Create m x n array
Initialise array indices
Compute cost of insertion, deletion and substitution at current location.
Copy minimum cost to array
Increment data counter
Is there another data element?
Yes
No
Reset data counter
Increment template counter
Is there another template element?
Yes
No
Read MMD value from array

Figure 3.13

Flow diagram showing an overview of the operation of the minimum merit distance comparison program. This corresponds to the sequence of events which follows the initiation of a search.

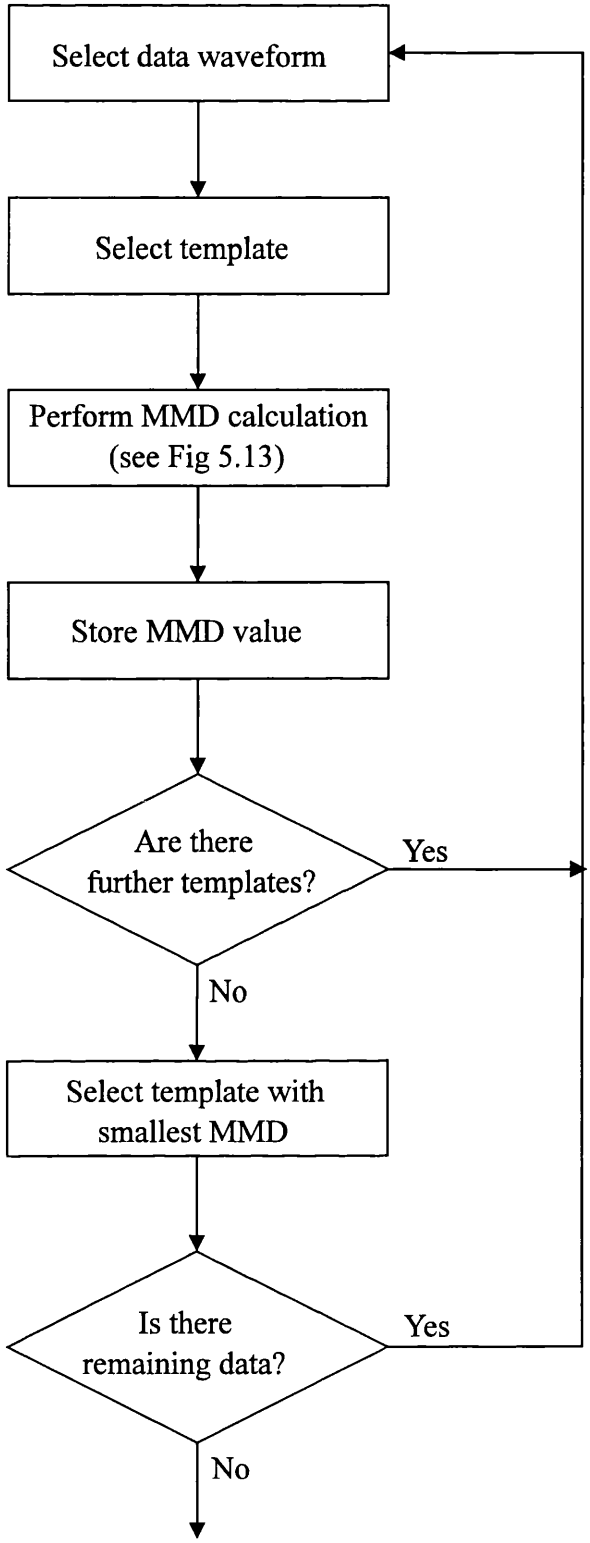

Select data waveform
Select template
Perform MMD calculation (see Fig 5.13)
Store MMD value
Are there further templates?
Yes
No
Select template with smallest MMD
Is there remaining data?
Yes
No

Figure 3.14

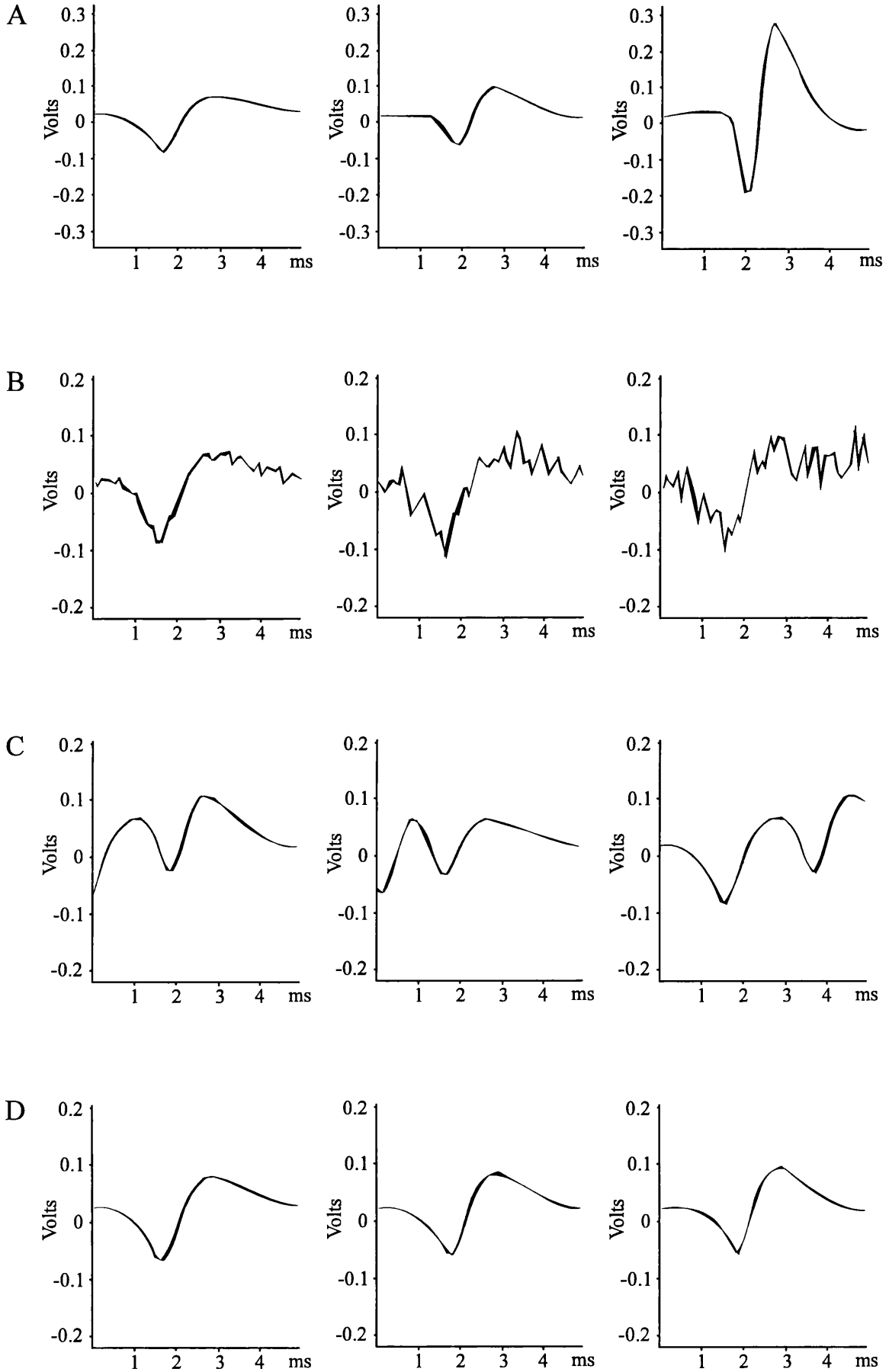
A. From left to right, the F1, F2 and F3 synthetic waveforms used for the construction of the synthetic data sets. These waveforms were obtained by averaging several selected examples from one of the real data sets available.

B. From left to right, the F1 waveform with normally distributed noise added at 0.01, 0.02 and 0.03 standard deviations. These correspond to S/N ratios of 13.9 dB, 7.9 dB and 3.0 dB respectively.

C. From left to right, synthetic double event waveforms corresponding to F2 with F1 prior, F1 with F2 prior and F1 with F2 subsequent. In each case the offset of the notional trigger point of the secondary event is 1.8ms.

D. From left to right, F1/F2 hybrid waveforms with F1 contribution of 20%, 50% and 80%.

The voltage scales in all of the graphs on this figure are comparable.


A
B
C
D
Volts
ms
0.3
0.2
0.1
0
-0.1
-0.2
-0.3
1
2
3
4

4 Results.

4.1 The test data available.

The original analog recordings of most, if not all, of Dr. Denheen's experiments conducted from 1988 to 1991 were available for use. This archive represented the results of many hours of electrophysiological experiments as described by her (Denheen 1992). Recordings of some of Dr. Leibrock's and Dr Cattaert's experiments were also available.

From these, the recordings of four particular experiments were selected. The first was chosen (by Dr. Denheen and myself) as a representative sample of the complete system described in section 3.1. The raw data clearly showed the presence of the F1, F2, F3, F4 and F6 spike classes with a 20mV (approximately) difference in the peak amplitude of the F1 and the F2 spike classes (see Fig 4.1). This data set was therefore amenable to F1/F2 separation by the methods previously used by Dr. Denheen (see section 3.1), and (after digitisation) was analysed by her in this fashion. In the second experiment, the F1 spike class was removed by cutting of the ventral nerve cord as described in section 3.7.2. The third and fourth experiments were those described in section 3.7.3 and 3.7.4 and contained both a difficult spike separation problem and a separate means of verifying the classification.

4.2 Template sensitivity analysis.

The sensitivity of the VE (variable envelope) template method to variations in the error tolerances described in section 3.4.1 was assessed by selecting sections of spike waveforms representative of the F1 and F2 categories. Holding the waveform constant in each case, the template error parameter (k1) and the derivative error parameter (k2) were varied over suitable ranges (0 to 30mV for k1 and 0 to 200μV for k2) and the number of events matching the template were recorded. As expected, the number of matches increased as the error parameters increased (see Fig 4.2). Interestingly the shape

of the curves for the various k2 values are very similar (for both F1 and F2), indicating that the effect of k2 on the sensitivity of the template is minor compared with that of k1.

Since the MMD (minimum merit distance) templates are not independent of each other and MMD templates are intended to operate with zero error tolerances, no corresponding analysis was performed using them.

4.3 Comparative analysis of spike data.

The comparative analysis was based on a section of data previously analysed using voltage thresholds, and for which F1 and F2 events had thus already been identified (as described in section 3.3). This was used as a control to establish the reliability of both the VE and the MMD template identification methods as applied to the separation of F1 and F2 spike classes.

For the VE method, templates were defined as before with a range of k_1 values (10 to 25mV) and a fixed k_2 value (100μV) and a search was performed for both F1 and F2. The results are presented in tables 4.1A and 4.1B. Again as expected this shows that as the k1 value increases the percentage of false negatives decreases (i.e. those template comparisons which failed to give a match to the spike class where a match is given by the corresponding voltage threshold method) while the percentage of false positives increases.

For the MMD method, templates were defined for all available spike classes, with zero k1 and k2 parameters, and a search was performed. This was repeated excluding the template for F3 to establish the extent to which false positive matches for F3 were generating false negative matches for F2. The results are presented in Table 4.1C.

4.4 Application to F1 censored data.

Having performed comparisons to the voltage threshold method, both VE and MMD search methods were applied to the pre- and post-cut data available from the

experiment during which the ventral nerve cord was cut posterior to the ganglion, and hence the F1 axon was severed. The data were visually inspected for the occurrence of F1 and F2 events prior to the cut and for F2 events subsequent to the cut (there being no F1 events in this part of the data). Separate sets of suitable VE and MMD templates were defined on the pre-cut data segment and both the pre-cut and post-cut data segments were searched. The number of matches found is presented in Table 4.2. As can be seen, the VE method fails almost totally to identify post-cut F2 events, while the MMD method identifies most post-cut F2 events. However, a VE template defined on a post-cut F2 event will pick up all the F2 events in that data segment (see Table 4.2). Both methods however find no post-cut F1 events.

A comparison of the pre-cut F1 and F2 waveforms and the post-cut F2 waveform (see Fig 4.3) give a good indication of the likely cause of the failure of the VE method. The amplitude of the post-cut F2 is intermediate between the pre-cut F1 and F2, and there is some change in the shape of the F2 waveform. This is unlikely to be due to the preparation deteriorating with time, since the two samples are only a few minutes apart, it is also unlikely to be an electrophysiological consequence of the severing of the ventral nerve cord - this does alter the firing patterns, but should not influence the individual spike waveforms since these are entirely dependent on the properties of the axon. However, slight movement of the electrode would probably occur during the cutting process as would contraction of any attached muscle fibres, and this is probably the explanation for the observed changes in the F2 waveform.

The entire firing pattern is disrupted by the cutting of the ventral nerve cord (see Fig 4.4), and consequently it is not possible to derive post-cut F2 firing rates and compare them with pre-cut firing rates as a measure of the accuracy of the template matching process. Nor is it possible to derive the probability of false F1 identification for either the VE or the MMD template method, since the shape of the F2 waveform is sufficiently altered by the cutting process to prevent effective matching without the generation of a new F2 template. Likewise this basic difference in the data set renders any consideration of the VE template sensitivity meaningless over a sensible range of sensitivity values.

Overall however, this experiment indicates that for suitably defined sets of templates the MMD method is the more robust in coping with experimental variations.

4.5.1 Application to tagged data from the crayfish cuticular stress detector.

Using data from one of Dr. Leibrock's experiments which contained both an extracellular (adr) and an intracellular (CSD1-t) channel, templates were defined as before and an MMD search was carried out on the adr channel. The results of this were imported into *Spike2* as event channels (the adr event channels), and compared with an event channel (the CSD1-t event channel) created by performing a voltage threshold search on the CSD1-t channel. It was verified by inspection that all of the events on the CSD1-t event channel corresponded to events in one of the adr event channels. The CSD1-t events were then compared with the adr events on the adr event channel showing correspondence (the template event channel).

Out of a total of 912 adr events, 215 corresponded to CSD1-t events. Of these 198 (92.1% of the CSD1-t events) were identified as template events (i.e. true positive events). A further 127 template events represented false positive events (39.1% of the total template events). This gives a probability for a template event being a CSD1-t event of 60.9% and a probability for a non- template event being a CSD1-t event of 2.9%. Figure 4.5 shows a representative section of the data and the event markers for the template events. From this figure it can readily be seen that there is no possibility of effecting the key separation using a voltage thresholding technique.

Application of the waveform identification method implemented in recent versions of *Spike2* resulted in 172 (79.6%) of the CSD1-t events being identified correctly (for the same section of data), but with no false positive events being recorded. This reflects differences in the template generation mechanisms and a different balance between the acceptable error rates, so without further comparative tests it is difficult to form meaningful conclusions about the relative accuracy of the two algorithms. It is however fair to point out that the more sophisticated template generation mechanisms in *Spike2* should, if anything, give better end results than the method for template

generation employed here.

4.5.2 Application to tagged data from the crayfish coxo-basal chordotonal organ.

This data set is particularly interesting due to the similarity between the spike category (designated R0) corresponding to the intracellular event, which always precedes intracellular events by an average of 4.26 ms, and another category (designated R1) which occurs at uncorrelated times. A section of this data is shown in Fig 4.6A and representative samples of R0 and R1 are shown in Fig 4.6B. Templates were defined as before and an MMD search was carried out on the extracellular channel.

Due to the size of the data set (12,156 spike events) it was not possible to use the script facilities of *Spike2* to perform the analysis. Accordingly, the results of the search were tabulated, and integrated (by means of a specially written utility) with events generated by setting a voltage threshold on the intracellular channel. The correspondence between template matches and intracellular events was then examined. The results are summarised in Table 4.3A.

If the MMD template method was performing no better than chance the distribution of the intracellular events between categories R0 and R1 (identification as any other category being expected to be the result of overlap events) should occur in proportion to the total numbers of R0 and R1 events. Applying a chi-square test to the search results provides a probability value of less than 0.01, indicating that this result differs significantly from a random distribution, a significantly higher number of intracellular events being classified as R0 (see Appendix A).

All of the other template categories also generated matches, most of which will have been due to overlap events. However the number of matches for template R4 is excessive, and an inspection of the spike events indicates that it is indicative of a systematic problem of misidentification.

This set of data was also filtered using a voltage threshold to remove spikes with peak voltages significantly larger or smaller than the classes of interest. The resulting data set was presumed to contain only the two classes of interest, together with a small

number of overlap events, and to exclude a small number of relevant overlap events. The MMD templates for the classes of interest were then used to search this data set. The results are summarised in Table 4.3B. Applying a chi-square test to the search results again provides a probability value of less than 0.01, indicating that the number of intracellular events classified as R0 was significantly greater than expected by chance (see Appendix A).

Using only a voltage threshold to separate spikes gave the results summarised in Table 4.3C, which may be used as a benchmark for comparing the templating methods with a method inherently incapable of separating R0 and R1 events (this data set being selected precisely for this property).

The template search mechanism of *Spike2* was also used as a comparison. A number of separate template generation and search runs were undertaken, the results from the usable searches (excluding searches which failed to identify R0 and R1 as spikes, and searches using very crude templates) are summarised in Table 4.4. Figure 4.7 shows a representative section of the templates generated by this means.

The voltage thresholding suggests that there are of the order of 3000 spike events in the R0 and R1 categories, and that approximately 100 (out of 1429) R0 events are subject to spike overlap. The MMD method identifies 3212 events as either R0 or R1, including 1281 out of a possible 1329 R0 events (allowing for overlaps). The segregation between R0 and R1 is better than chance, but is not sufficiently accurate (both because of false positive and false negative classifications) to be of use in most circumstances. However, the rate of false positive matches for *Spike2* searches varies between 33% and 75%, when *Spike2* is able to identify R0 or R1 as events at all, with slight variations in the template generation parameters giving wildly differing results (including complete non-recognition of R0 and R1 as events). Overall the R0/R1 separation is extremely difficult, and no mechanism is likely to perform well - at least not to levels which would be useful in practise. The interesting feature of the MMD classification results is that the MMD method could do anything with data of this nature.

4.6 Application to synthetic data.

The data described in section 3.7.6 was analysed in accordance with the procedures described there. The single event search results are shown in Fig 4.8, which shows the expected increasing failure rate with increasing noise levels, though with the F2 identification failing more dramatically than might perhaps be expected.

The double event search results are presented in Table 4.5. This again shows the expected decline in matching performance with increasing levels of noise and, as expected, that simultaneous events completely destroy reliable discrimination. It also shows that the presence of a non-simultaneous secondary event significantly reduces the discriminating ability of the MMD search method, suggesting that it would in general require to be coupled with either the use of compound templates (representing specific overlap combinations) or identified event subtraction followed by re-matching of identified overlap situations.

The hybrid event search results are presented in Fig 4.9, except for the zero noise data which showed 100% reliability for all comparisons except for the F1/F2 50/50 hybrid which was identified as 100% F1. As expected, these show a decline in the reliability of identification both as the noise and the proportion of the minor hybrid component increased. Perhaps surprisingly this had an effect on the identification of the un-hybridised F3, which was included for comparison.

Table 4.1

A. Variable envelope template results for F1.

B. Variable envelope template results for F2.

Comparison of the results of the variable envelope method with the voltage window technique (Denheen, 1992) when applied to the same data set. For a template search performed with the specified k1 and k2 parameters (see text for definitions), the events identified as being F1 and F2 were compared with the F1 and F2 events identified by voltage windowing. False positive events were defined as being template events which did not match window events, while false negative events were defined as being window events which did not match template events. False positive events are also noted as a percentage of the total template events.

As expected, as the k1 value increases so the false negative events decline while the false positive events increase.

C. Minimum merit distance results for F1 and F2 compared with the voltage window technique. Series **a** (F1a and F2a) show the results using templates for F1, F2, F3, F4, F5/F6 (combined), while series **b** (F1b and F2b) show the results using templates for F1, F2, F4, F5/F6 (combined). This should reduce the probability of incorrectly classifying an F2 event as an F3 event (at the expense of incorrect classification of F3 events), which is in fact the case.

For the purposes of these tables, the voltage window method is assumed to identify F1 and F2 events with complete accuracy, an assumption which may not in fact be valid.

A

k1 value	k2 value	actual matches	template matches	number (%) of false positive	number (%) of false negative	% of template matches false
10	100	159	100	8 (5.03)	67 (42.14)	8.00
15	100	159	128	12 (7.55)	43 (27.04)	9.38
20	100	159	138	16 (10.06)	37 (23.27)	11.59
25	100	159	165	31 (19.50)	25 (15.72)	18.79

B

k1 value	k2 value	actual matches	template matches	number (%) of false positive	number (%) of false negative	% of template matches false
10	100	118	83	6 (5.85)	40 (33.90)	7.23
15	100	118	100	7 (5.93)	24 (20.34)	7.00
20	100	118	105	11 (9.32)	23 (19.49)	10.48
25	100	118	112	15 (12.71)	20 (16.95)	13.39

C

template	actual matches	template matches	number (%) of false positive	number (%) of false negative	% of template matches false
F1a	159	160	15 (9.43)	14 (8.81)	9.38
F2a	118	104	9 (7.63)	23 (19.49)	8.66
F1b	159	165	17 (10.69)	14 (8.81)	10.30
F2b	118	112	12 (10.17)	18 (15.25)	10.71

Table 4.2

Results of template searches for F1 and F2 waveforms before and after the cutting of the ventral nerve cord posterior to the ganglion from which the third root originated. This abolished all F1 activity. Figures are absolute numbers of events for each template, when searches were performed on the same two sections of data.

The templates are as follows:

F1a VE template for F1 defined on the pre-cut data segment.

F2a VE template for F2 defined on the pre-cut data segment.

F1b MMD template for F1 defined on the pre-cut data segment.

F2b MMD template for F2 defined on the pre-cut data segment.

F2c VE template for F2 defined on the post-cut data segment.

template	pre-cut events	post-cut events
F1a	266	0
F2a	68	3
F1b	267	0
F2b	71	44
F2c	**N/A**	50

Table 4.3

Results of template searches of tagged data from the crayfish coxo-basal chordotonal organ.

A. Results of searching the entire data set for templates (designated R0 to R5) corresponding to the various classes of spike present in the data set.

B. Results of searching the filtered data set for templates R0 and R1, corresponding to the class separation of interest. The spikes corresponding to classes R2 to R5 were removed by eliminating all events with an amplified peak amplitude of less than 0.5 volts or more than 0.8 volts. This will also remove some overlap events which would potentially be of interest, as well as some events previously classified as R0 or R1, and will also include representatives of other classes of spike, resulting in the total number of events not corresponding to the previous search (which is supported by the 34 intracellular events which did not correspond to events included in the filtered data).

C. Results of searching the entire data set using a voltage window. The spikes corresponding to category W0 have a peak amplitude of between 0.55 and 0.8 volts, the spikes corresponding to category W1 constitute the rest of the file.

A

Class	Total Events	Intracellular Matches
R0	1549	767
R1	1663	514
R2	1005	9
R3	2979	24
R4	2960	101
R5	2000	19

B

Class	Total Events	Intracellular Matches
R0	1606	821
R1	1634	579

C

Class	Total Events	Intracellular Matches
W0	2896	1332
W1	9260	97

Table 4.4

Results of using *Spike2* to generate templates and search the entire set of tagged data from the crayfish coxo-basal chordotonal organ. In each run a new set of templates was generated, and a search performed using these. The template categories designated S0 to S9 differ between runs, the number of categories differs between sessions, and the number of detected spike events differs.

Class	First Run		Second Run		Third Run	
	Total	Matches	Total	Matches	Total	Matches
S0	1865	1241	2859	1363	2683	735
S1	703	68	3079	24	855	322
S2	527	52	286	15	261	16
S3	540	35	1628	10	2211	9
S4	2260	10	369	10	266	8
S5	716	7	360	6	1268	4
S6	1538	6	-	-	-	-
S7	168	5	-	-	-	-
S8	821	4	-	-	-	-
S9	36	1	-	-	-	-
Total	9174	1429	8581	1428	7544	1094

Table 4.5

Results of MMD (minimum merit distance) template searches for F1, F2 and F3 waveforms on synthetic data representing instances where two spike events overlap. In each instance the primary spike event is the one which is correctly aligned with respect to the template and the secondary event occurs at the time indicated (with the timings being relative to the notional voltage threshold trigger event).

Figures given are percentages of primary events correctly identified for each template while * denotes combinations not attempted. Each column of tables corresponds to the specified secondary event, while each row of tables corresponds to the specified S/N ratios.

The S/N ratios presented here and elsewhere are derived from the standard deviation of the noise relative to the F1 spike. The S/N ratios are given with $[S/N]_{db} = 10\log_{10}(S^2/N^2)$ where S^2 and N^2 are $\Sigma S^2/n$ and $\Sigma P^2/n$ respectively (S = sample, N = noise, n = number of datapoints).

Secondary event (1.8ms prior)

Secondary event (at same time)

Secondary event (1.8ms after)

Prinmary event $[S/N]_{db} = \infty$

	F1	F2	F3
F1	*	100.0	100.0
F2	100.0	*	0.0
F3	100.0	100.0	*

	F1	F2	F3
F1	*	0.0	0.0
F2	100.0	*	0.0
F3	100.0	100.0	*

	F1	F2	F3
F1	*	100.0	100.0
F2	100.0	*	0.0
F3	100.0	100.0	*

Prinmary event $[S/N]_{db} = 13.9$

	F1	F2	F3
F1	*	100.0	100.0
F2	100.0	*	0.0
F3	95.0	85.0	*

	F1	F2	F3
F1	*	0.0	0.0
F2	87.0	*	0.0
F3	100.0	100.0	*

	F1	F2	F3
F1	*	43.0	8.0
F2	6.0	*	0.0
F3	66.0	94.0	*

Prinmary event $[S/N]_{db} = 7.9$

	F1	F2	F3
F1	*	94.0	84.0
F2	62.0	*	6.0
F3	77.0	57.0	*

	F1	F2	F3
F1	*	0.0	0.0
F2	90.0	*	11.0
F3	100.0	100.0	*

	F1	F2	F3
F1	*	20.0	0.0
F2	5.0	*	25.0
F3	55.0	61.0	*

Prinmary event $[S/N]_{db} = 3.0$

	F1	F2	F3
F1	*	85.0	88.0
F2	30.0	*	7.0
F3	75.0	56.0	*

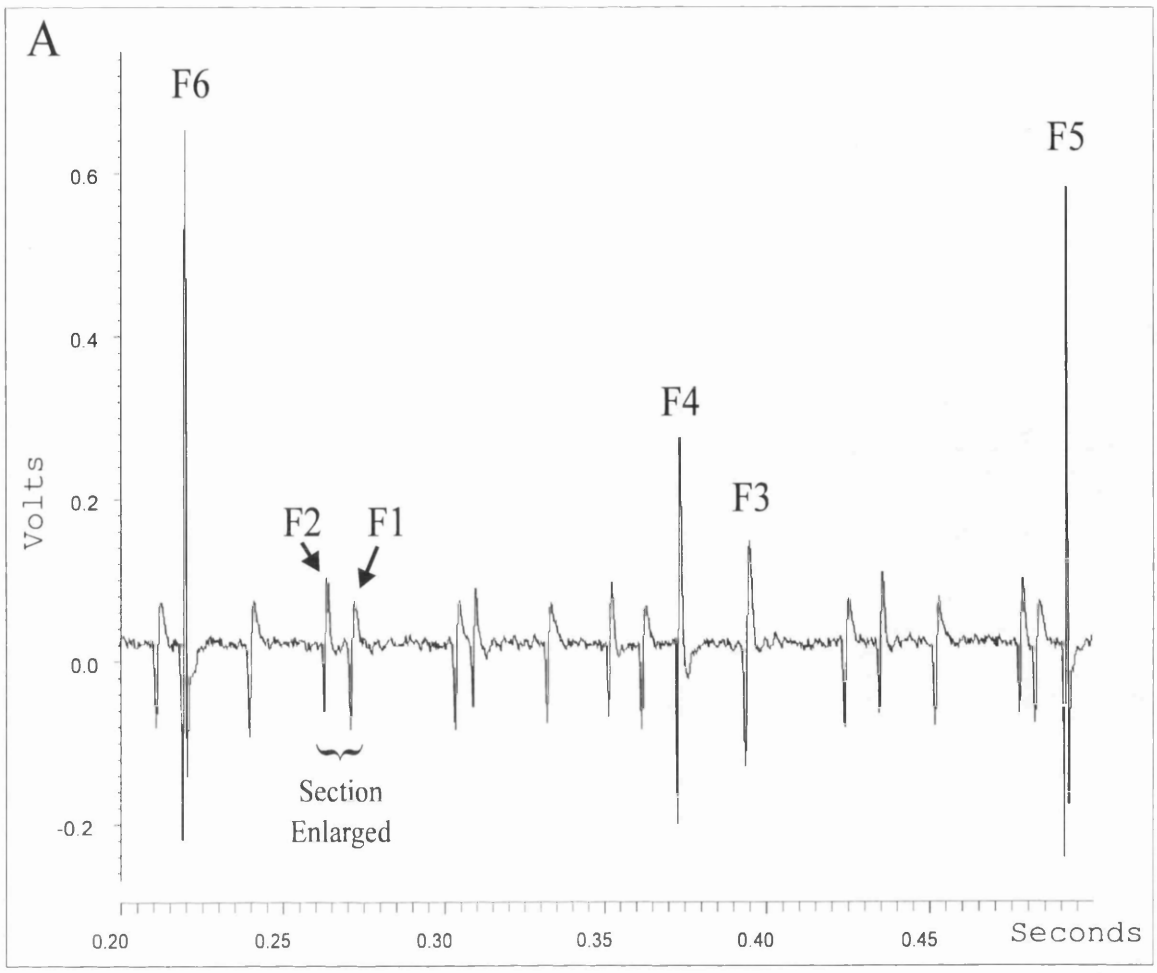
	F1	F2	F3
F1	*	0.0	0.0
F2	52.0	*	0.0
F3	100.0	95.0	*

	F1	F2	F3
F1	*	40.0	45.0
F2	0.0	*	0.0
F3	23.0	44.0	*

Figure 4.1

A. Single channel recording of the activity on the third root of the superficial flexor nerve, showing spike events identified as belonging to the spike classes F1 to F6.

B. The F1 and F2 spike waveforms in greater detail. Overlaid onto the F2 waveform is a copy of the F1 waveform (F1') to show the difference in shape.


A
F6
F5
F4
F3
F2
F1
0.6
0.4
0.2
0.0
-0.2
Volts
Section
Enlarged
0.20
0.25
0.30
0.35
0.40
0.45
Seconds

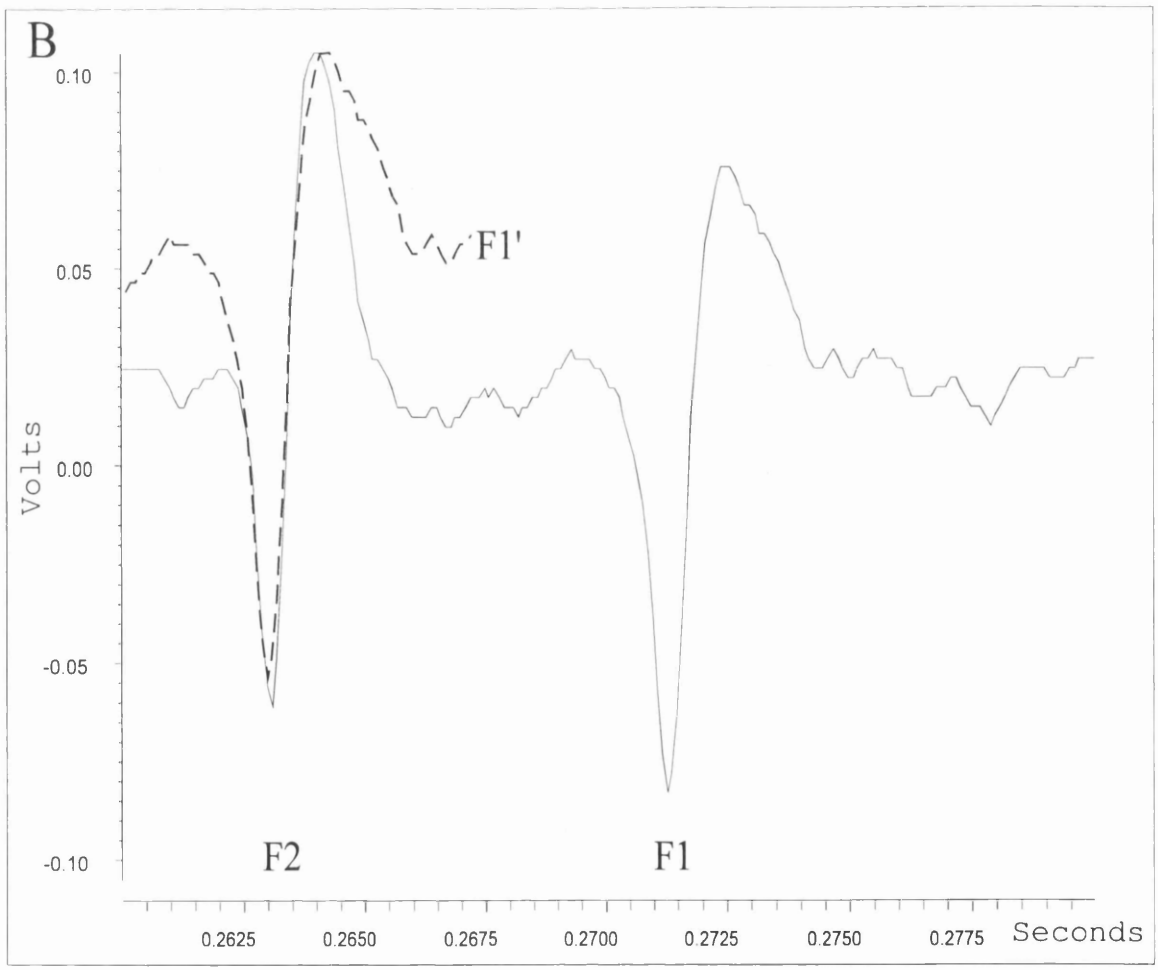

B
0.10
0.05
0.00
-0.05
-0.10
Volts
F1'
F2
F1
0.2625
0.2650
0.2675
0.2700
0.2725
0.2750
0.2775
Seconds

Figure 4.2

Results of sensitivity testing of the variable envelope template. The number of spike matches is shown for each template error tolerance (k1) value for a range of template derivative error tolerance (k2) values from 0 to 200 μV. Increasing the k1 value should result in an increased number of matches, until all of the available spike events are matched for some k1 value. As expected the shape of the curve is only slightly influenced by the k2 value, with increased k2 mostly resulting in an increased number of matches for any given k1 value.

A. Result for F1 template search (k1 range 0 to 28 mV).

B. Result for F2 template search (k1 range 0 to 30 mV).

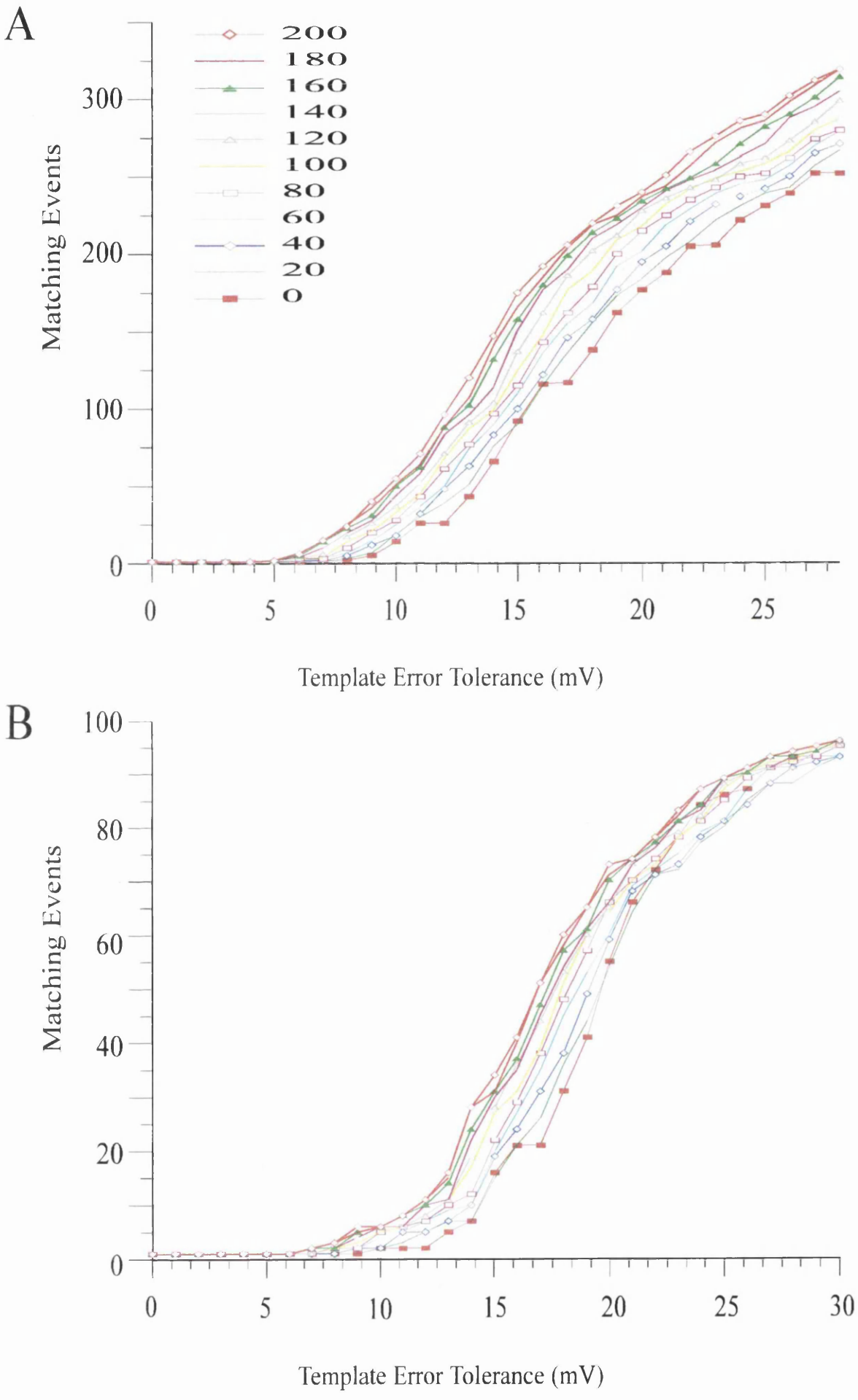

A
200
180
160
140
120
100
80
60
40
20
0
Matching Events
300
200
100
0
0
5
10
15
20
25
Template Error Tolerance (mV)
B
Matching Events
100
80
60
40
20
0
0
5
10
15
20
25
30
Template Error Tolerance (mV)

Figure 4.3

Representative F1 and F2 spike events from the experiment in which the ventral nerve cord was cut to sever the F1 neuron.

A. Pre-cut F1 spike.

B. Pre-cut F2 spike.

C. Post-cut F2 spike. Note the difference in the waveform from B to C.

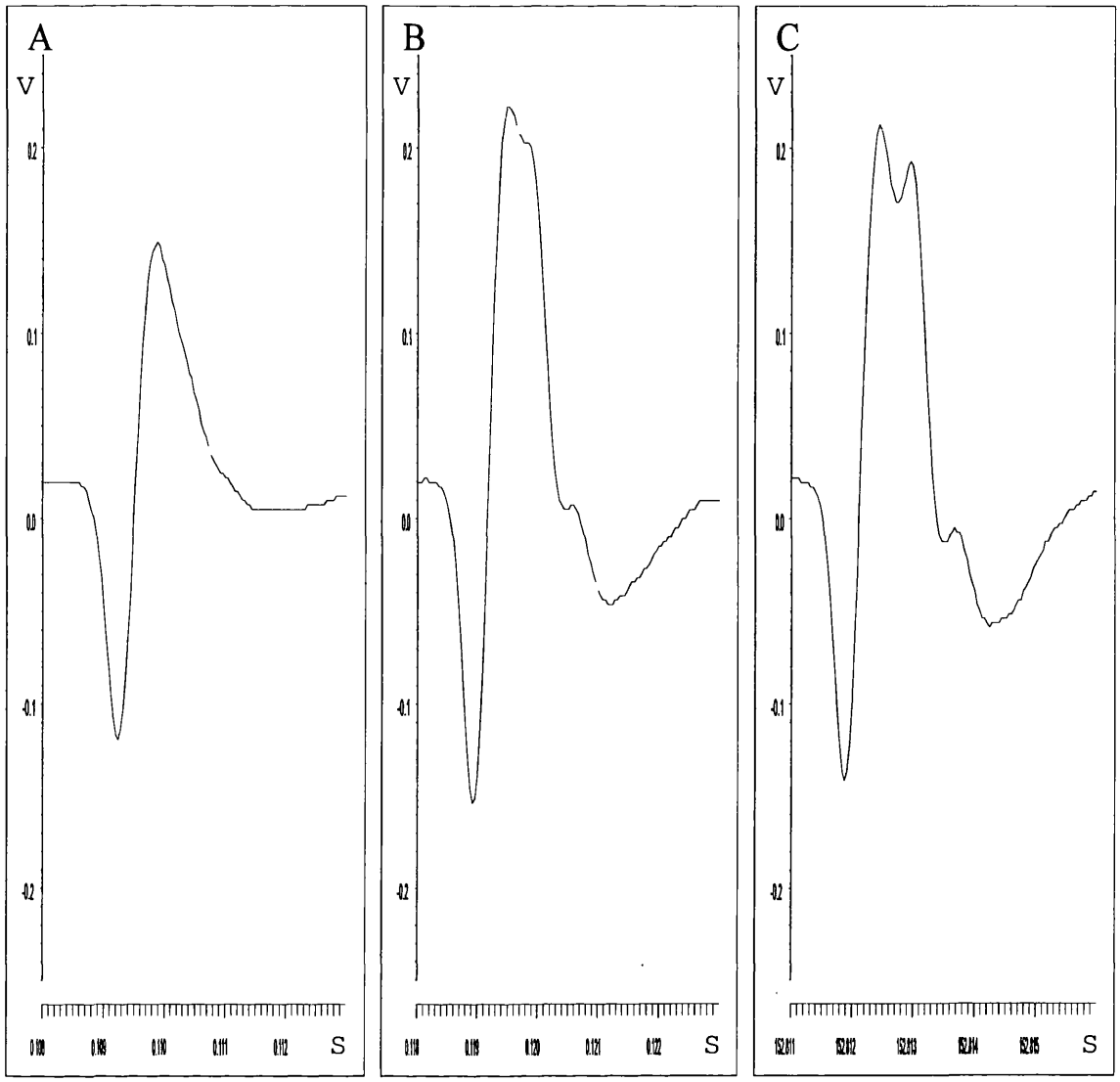

A
V
0.2
0.1
0.0
-0.1
-0.2
0.108
0.109
0.110
0.111
0.112
S
B
V
0.2
0.1
0.0
-0.1
-0.2
0.118
0.119
0.120
0.121
0.122
S
C
V
0.2
0.1
0.0
-0.1
-0.2
152.811
152.812
152.813
152.814
152.815
S

Figure 4.4

The effect of cutting the ventral nerve cord on the activity of the superficial flexor nerve.

A. Activity during an eight second period prior to cutting.

B. Activity during an eight second period immediately following cutting.

C. Activity during a ten second period several minutes after cutting.

Note that the timing of the three sections as marked on the traces does not correspond to the actual intervals between the three recordings.

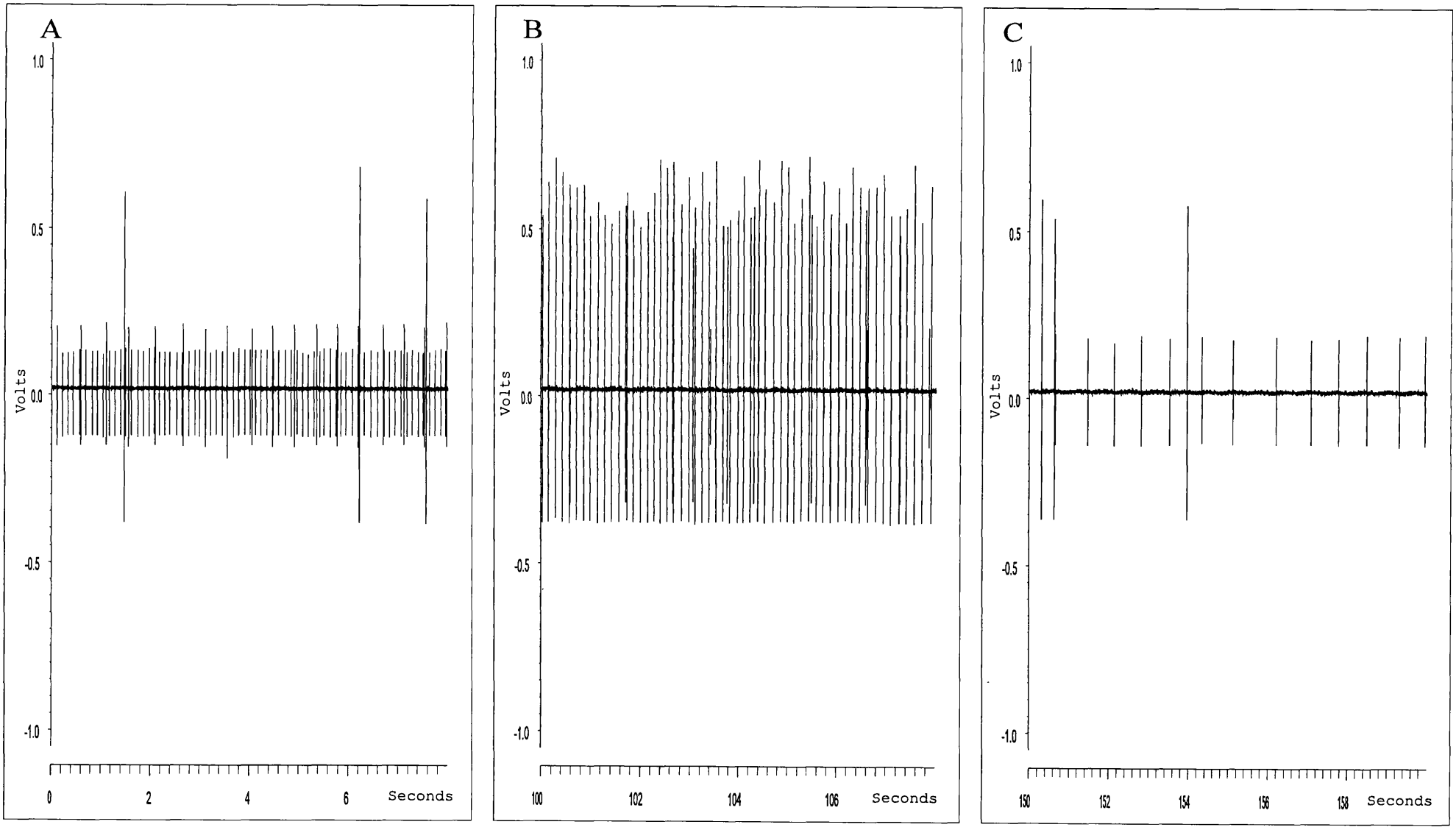

A
1.0
0.5
0.0
-0.5
-1.0
Volts
0
2
4
6
Seconds
B
1.0
0.5
0.0
-0.5
-1.0
Volts
100
102
104
106
Seconds
C
1.0
0.5
0.0
-0.5
-1.0
Volts
150
152
154
156
158
Seconds

Figure 4.5

A representative segment of the tagged data from the crayfish cuticular stress detector, showing various possible outcomes for the template match.

Channel 1 shows the adr activity.

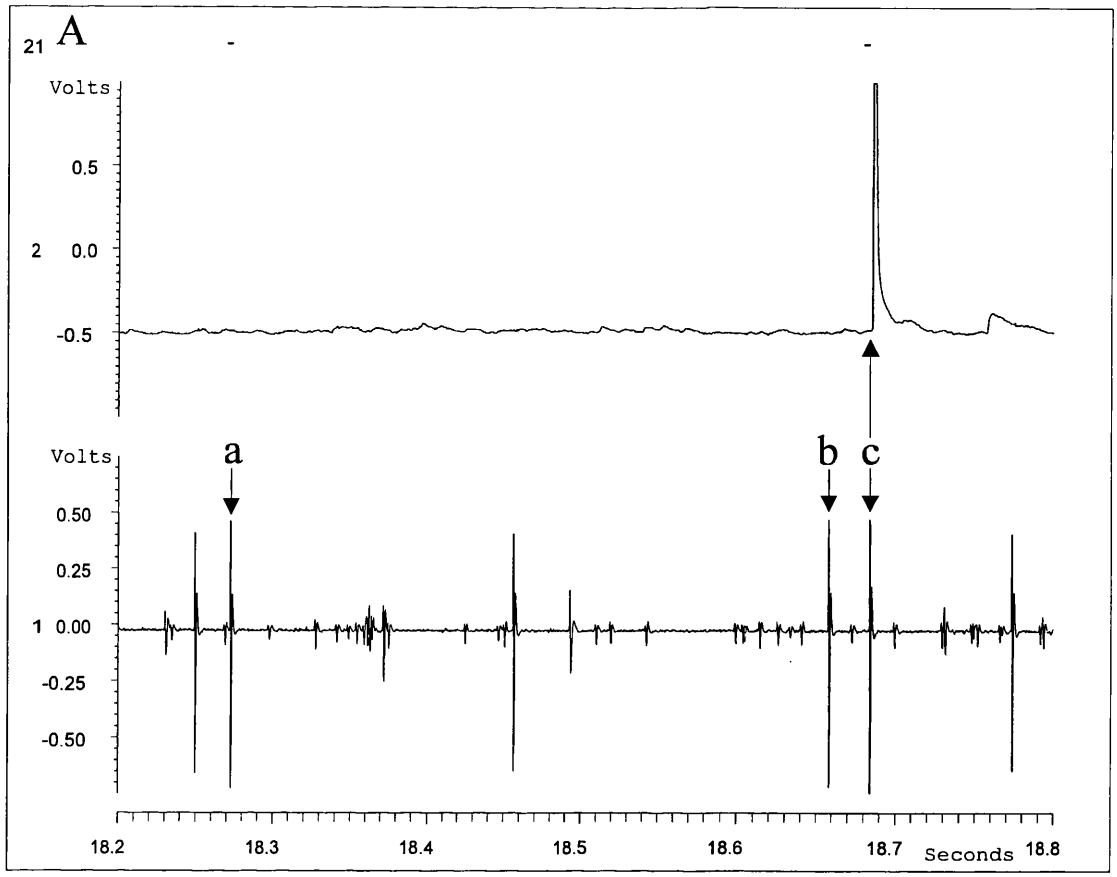
Channel 2 shows the CSD1-t activity.

Channel 21 shows the template events.

A. A short section showing:

- **a** a false positive template event
- **b** a true negative event
- **c** a true positive event

B. A longer section giving a picture of the overall activity.


A
21
Volts
0.5
2 0.0
-0.5
Volts
a
b
c
0.50
0.25
1 0.00
-0.25
-0.50
18.2
18.3
18.4
18.5
18.6
18.7
Seconds
18.8

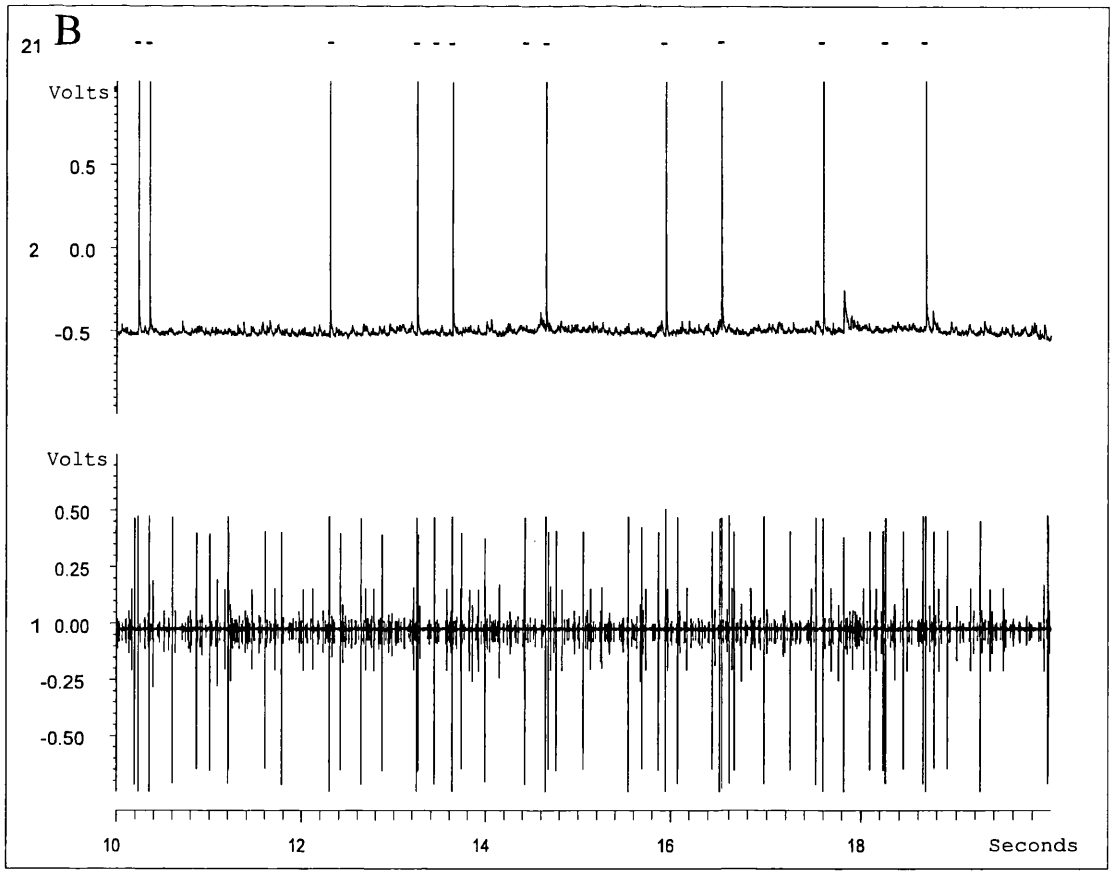
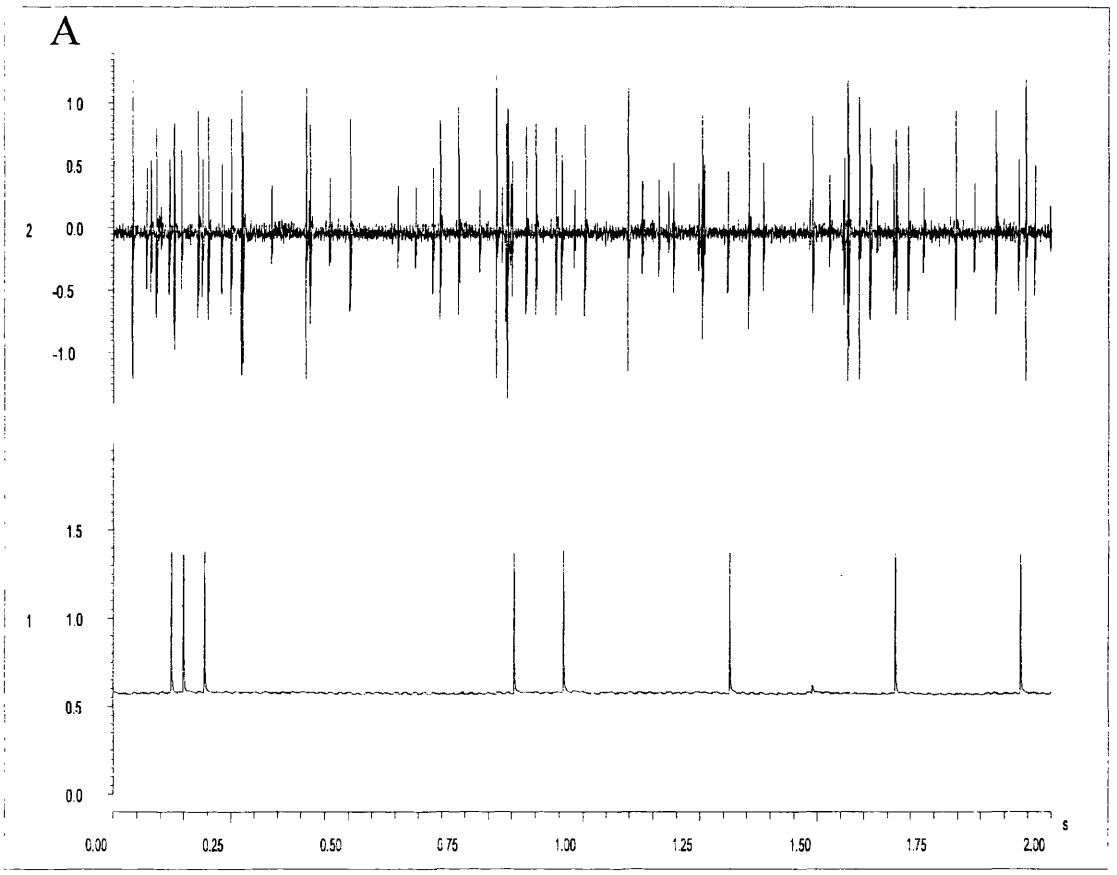

B
21
Volts
0.5
2 0.0
-0.5
Volts
0.50
0.25
1 0.00
-0.25
-0.50
10
12
14
16
18
Seconds

Figure 4.6

A representative segment of the tagged data from the crayfish coxo-basal chordotonal organ, showing various possible outcomes for the template match.

A. A section showing the intracellular activity on channel 1 and the extracellular activity on channel 2.

B. Representative R0 (red) and R1 (blue) waveforms superimposed to show the similarity in shape and amplitude.


A
1.0
0.5
2 0.0
-0.5
-1.0
1.5
1 1.0
0.5
0.0
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
s

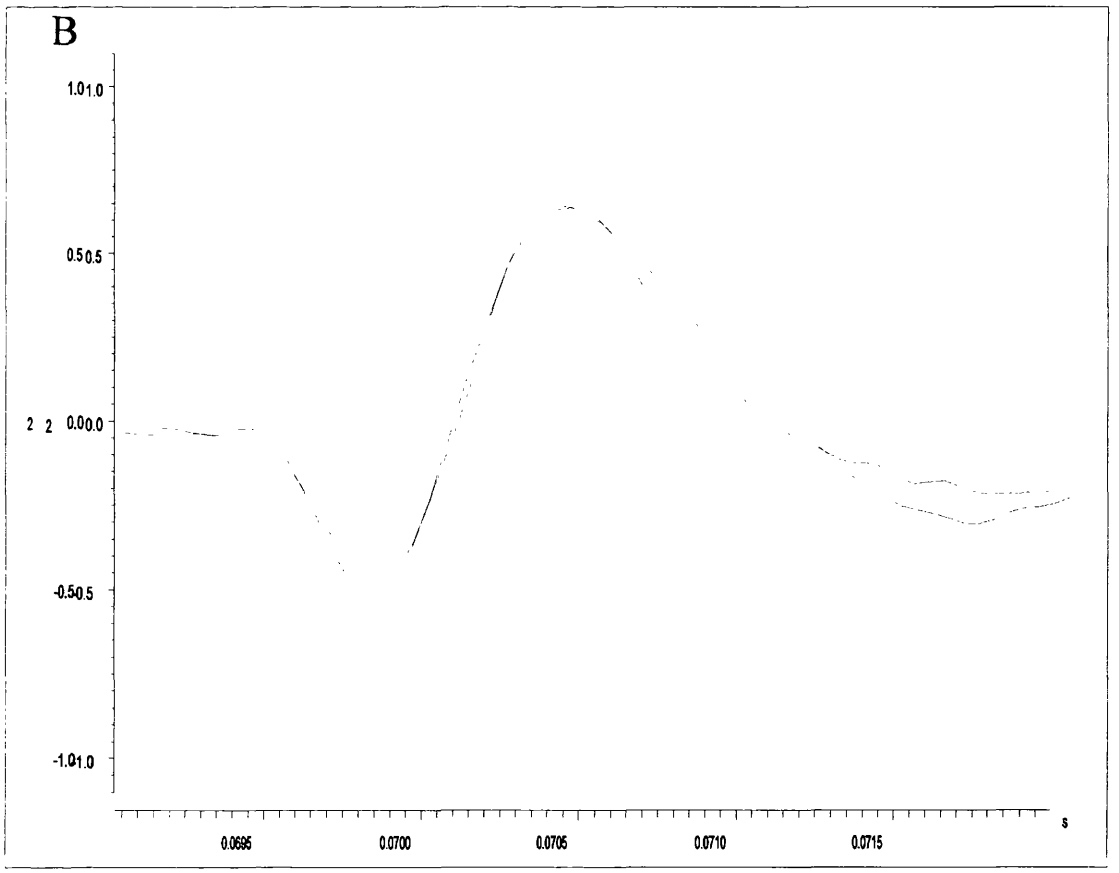

B
1.01.0
0.50.5
2 2 0.00.0
-0.50.5
-1.01.0
0.0695
0.0700
0.0705
0.0710
0.0715
s

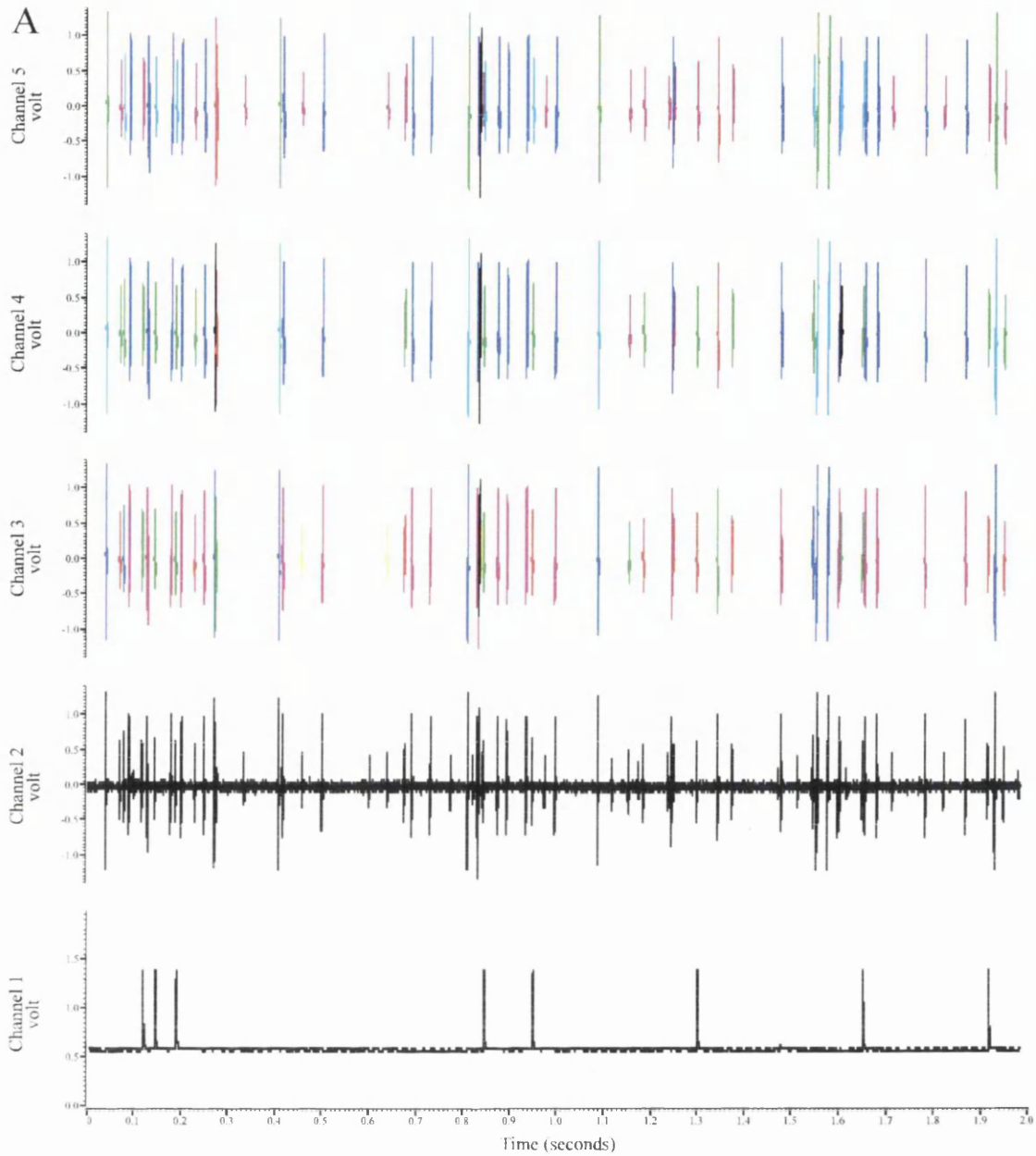
Figure 4.7

An illustrative segment of the tagged data from the crayfish coxo-basal chordotonal organ showing the results of template classification using *Spike2*. **A** shows a two second segment, while **B** shows detail from part of that segment. There does not appear to be any consistency in the classification of R0 and R1 events.

Channel 1 shows the intracellular activity.

Channel 2 shows the extracellular activity.

Channels 3 to 5 show the template classification results of searches from separate sessions. Within each channel the spikes are colour coded to show the template category to which they belong. The colour coding is not consistent between channels.


A
Channel 5 volt
Channel 4 volt
Channel 3 volt
Channel 2 volt
Channel 1 volt
Time (seconds)

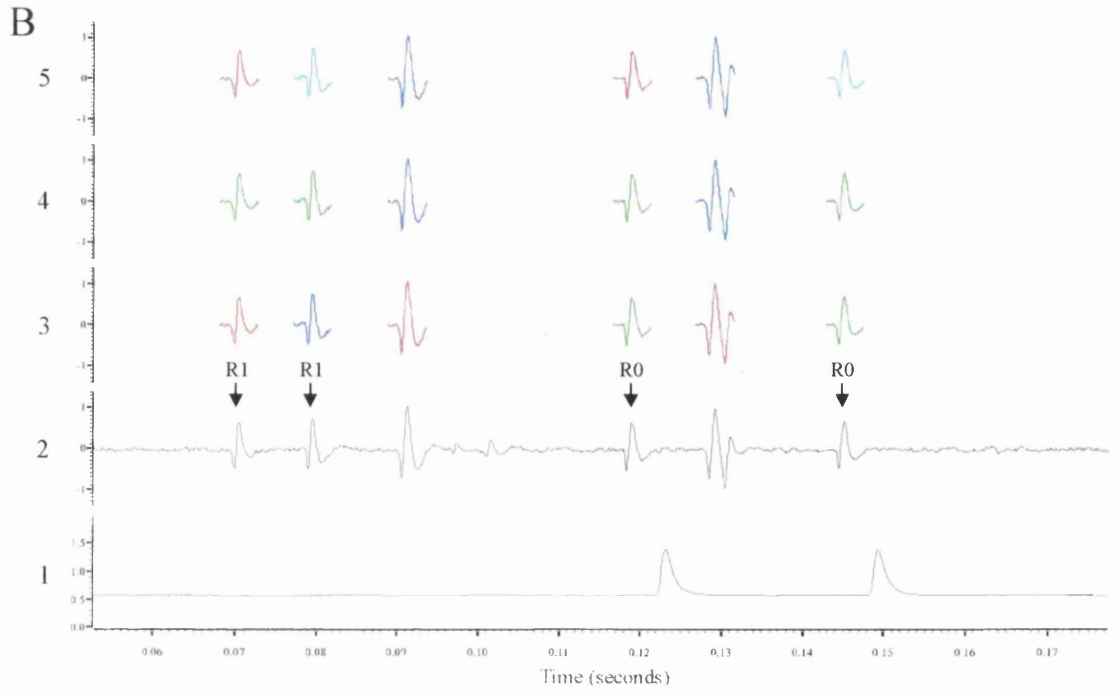

B
5
4
3
2
1
R1
R1
R0
R0
Time (seconds)

Figure 4.8

The effect of increasing background noise on the reliability of the MMD template method when applied to single spike synthetic data. The failure rates are given as percentages of the known events which are correctly identified at each of the noise levels. The S/N ratios given here are derived from the standard deviation of the noise relative to the F1 spike.

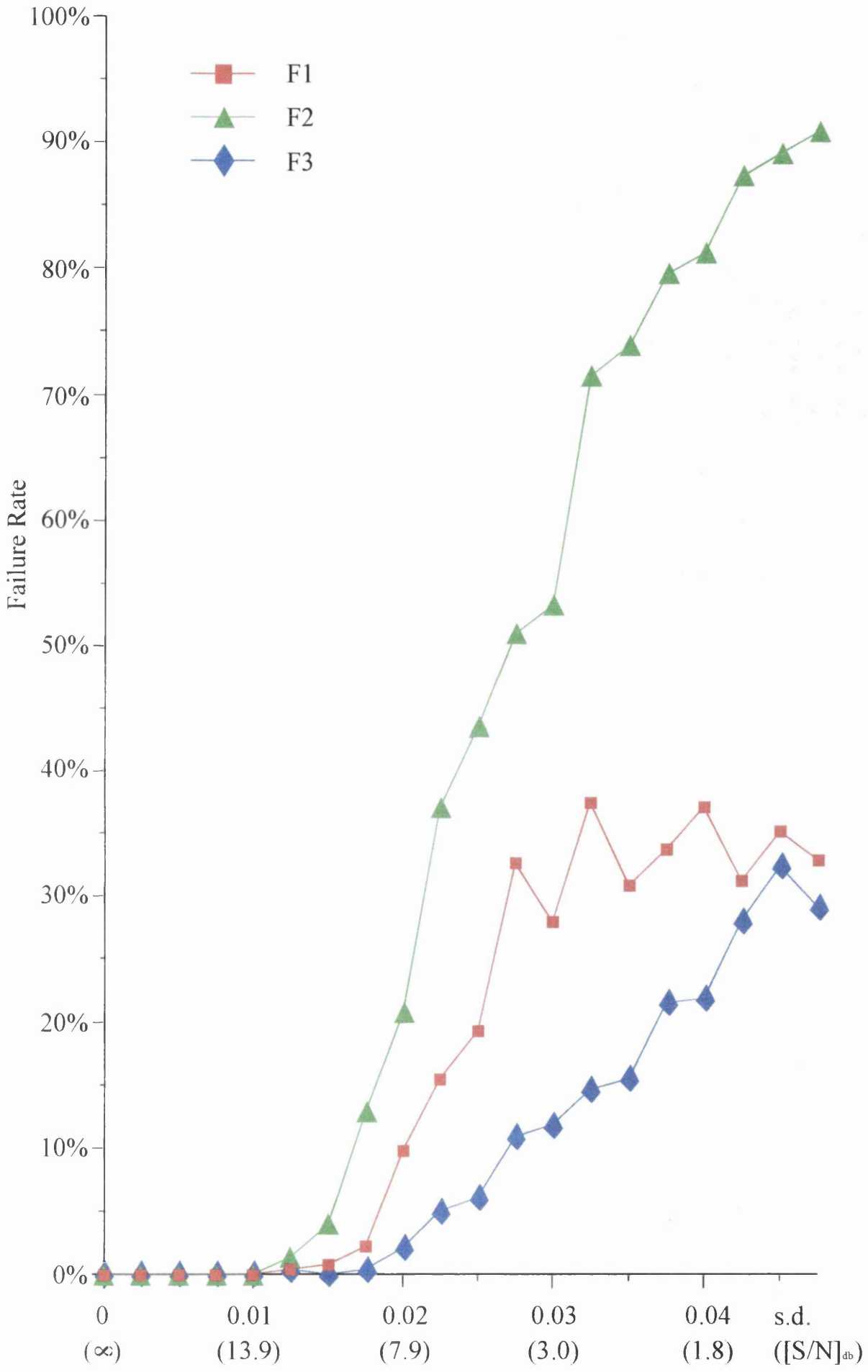

100%
90%
80%
70%
60%
50%
40%
30%
20%
10%
0%
Failure Rate
F1
F2
F3
0
(∞)
0.01
(13.9)
0.02
(7.9)
0.03
(3.0)
0.04
(1.8)
s.d.
([S/N]db)

Figure 4.9

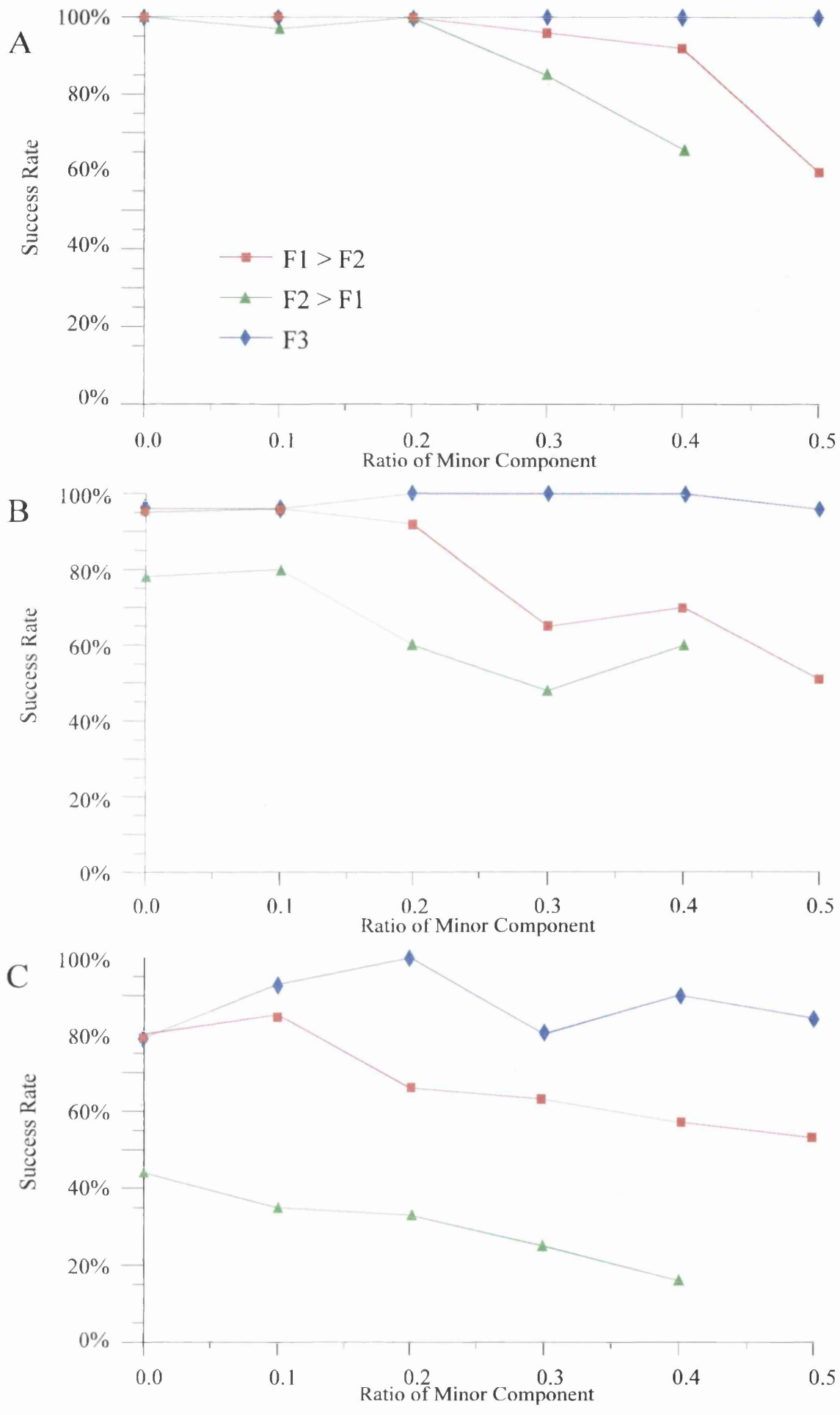
The matching success of the MMD template method when applied to spikes intermediate in shape between F1 and F2. Each graph shows the effect of the ratio of the minor component on the matching success. F3 is not hybridised in this data set.

A. The result at 0.01 of standard deviation (S/N = 13.9dB).

B. The result at 0.02 of standard deviation (S/N = 7.9dB).

C. The result at 0.03 of standard deviation (S/N = 3.0dB).

The equivalent graph for zero noise would show 100% success at all ratios tested, except for the F1/F2 50/50 hybrid which was identified as 100% F1.


A
100%
80%
60%
40%
20%
0%
Success Rate
F1 > F2
F2 > F1
F3
0.0
0.1
0.2
0.3
0.4
0.5
Ratio of Minor Component
B
100%
80%
60%
40%
20%
0%
Success Rate
0.0
0.1
0.2
0.3
0.4
0.5
Ratio of Minor Component
C
100%
80%
60%
40%
20%
0%
Success Rate
0.0
0.1
0.2
0.3
0.4
0.5
Ratio of Minor Component

5 Discussion.

5.1 Event identification methods compared.

As discussed in chapter 1 there are a number of possible approaches to the identification of nerve spikes. At the simplest level, events can be identified on the basis of the waveform crossing a threshold level, while at a somewhat more complex level waveforms can be compared (by some means) to templates. These range conceptually from an extension of the voltage threshold to form a continuous series (giving a closed, bounded region of a poly-dimensional space) to those utilising some measure of "distance" between waveforms. Various distance measures (or metrics) are used, most commonly the sum of the absolute values of the differences of the component values (D'Hollander and Orban, 1979; Jansen and Maat, 1992) or the Euclidian distance (Salganicoff *et al.*, 1988), using either the entire waveform or a subset of the component datapoints.

On a slightly different track it is possible to consider the voltage values making up each waveform as the coordinates of a point in a poly-dimensional space. These points will form clusters which can be identified (by considering the distance from each point to its neighbours, as is done in two or three dimensions) and the space can then be partitioned to separate these clusters. The means of accomplishing this partitioning becomes the key to the success of this technique. The difference between this approach and the consideration of a template as describing a bounded region of the same poly-dimensional space lies in the means used to define the region. In a template-based approach the region is defined simply, for example as the analogue of a sphere or ellipsoid surrounding a point which represents the average of all currently matched waveforms, and which may vary depending on data ordering. A true clustering method instead partitions the data with reference to **all** of the waveforms. This can be accomplished by the use of an algorithm such as the "k-means" algorithm (Hartigan, 1975) employed by various investigators (Salganicoff *et al.*, 1988; Kreiter *et al.*, 1989) or alternately by use of a "genetic" algorithm (Forrest, 1993), which operates on the

principle of swapping elements between clusters until an optimal distribution is achieved.

It would also be possible to consider a waveform as being a form of stochastic process with each data point having a different probability distribution depending upon which event class the waveform belonged to. The probability that a given waveform could take the particular shape given its **actual** identity could then be calculated and the class giving the highest probability could then be assigned as the match. However it seems that this approach has not been discussed in the literature.

In a more deterministic fashion, a curve fitting algorithm could be used to determine an approximate equation for each waveform. Curve fitting is a standard capability for "off-the-shelf" mathematical analysis packages such as MathSoft's *MathCad* 5 plus, and this approach could be applied to give the class, and with suitable restrictions the parameters, of the equation for each waveform. This has the advantage of transforming a set of voltage values into a set of equation parameters which relate to the information content of the entire waveform. If in addition the waveform equations were constrained to be either of the same class, or of highly distinct classes (i.e. those producing waveforms of substantially differing shapes), then the parameter information could be directly compared. This is the approach followed by Remmel (1983).

Approaches considering some aspects of the information content of waveforms were among the first to be considered (Mishelevich, 1970; McCann, 1973), probably due to the suitability of their highly reductive nature to implementations on early computer systems. Aside from principal component analysis (Abeles and Goldstein, 1977; Eggermont *et al.*, 1983), the more extensive consideration of the information content which more powerful computing facilities have made possible appears to have been neglected in favour of high speed templating systems which could be implemented in real time. Indeed, apart from the Fourier or principal component analyses, no consideration has been made of systems using a distance metric based on information content rather than raw shape.

5.1.1 Identification of events using neural networks.

One of the most interesting and most generally applicable techniques is that of neural net classification. The principles of neural net techniques are well known (Hart, 1992), and in short utilise a form of "training" to establish a classification mechanism for (potentially) highly complex data. This approach is next to useless when there is only a small quantity of data available. However, the classification of large numbers of elements is ideally suited to this approach.

However, this should not be taken as suggesting that neural net methods are necessarily the last word in terms of classification and matching. There are objections on both theoretical and practical levels to the blanket application of any technique, and neural networks are no exception in this respect. On the theoretical level there are several reasons to avoid neural nets, chief amongst which must be its distinct "black-box" quality. With most matching techniques the exact process by which a comparison is effected is likely to be understood by its original developers. However, the basic concept of the neural net is that the training period influences the weights of intermediate connections between known inputs and outputs. Therefore it is difficult, if not impossible, to know what actually determines the classification scheme used, giving rise to the possibility of training the net using data in which secondary characteristics are present but not readily appreciated resulting in an unreliable classification of real data (Hart, 1992). On a practical level, most pattern matching algorithms are based directly on mathematical or statistical procedures and a particular implementation can therefore be verified both by analysis of the code for correctness and by comparison of the results of analysing the same data via the new code and by some other method (such as comparing the behaviour of mathematical equations coded in a programming language with the same equations handled through a symbolic mathematics system such as Wolfram's *Mathematica*). While the kind of code produced for an implementation of a neural net can obviously be verified by routine code checking and verification procedures, these are far from being highly reliable (McConnell, 1993), and unfortunately neither the algebraic nor comparative techniques are appropriate for this

kind of code. All this leaves the problem of adequately verifying a system when the code cannot be verified and the results cannot be checked. None of this is by any means insurmountable, indeed similar comments could once have been made about many other areas, but it does contribute to the use of neural nets without a proper understanding of the issues involved. Just because a technique has potential drawbacks, or could be implemented improperly is no reason to reject its use. The volume of nerve spike data is perfectly adequate for the neural net approach to be applied. There are alternate methods which could be applied to generate matched sets of data (either from voltage thresholding or from templating - see the following sections) for training purposes, and there is also the possibility that a neural net approach could be applied "blind" to new data sets based on previous training. This summary, however, contains the seeds of the reasons for rejecting this approach. In the first place a net system has to be trained, and for data which cannot be classified by voltage thresholding this implies that clustering or templating has to be performed first. In the second place there are two approaches to the addition of neural net capabilities. Either a net facility could be developed from scratch, which is too substantial a task to have been realistically tackled as part of this project, or alternately an outside system (either commercially available or developed as part of another research project) could have been grafted onto the data management systems. This approach avoids the direct development problems. However, there remain the usual problems of integrating what is necessarily a complex system with the necessary support facilities. This task is significantly easier when the neural net facilities are already available (in the form of a set of libraries) at design time. This approach was not even considered at that stage, and did not prove possible later.

However, this does not remove the possibility of extending the signal analysis system to use neural net mechanisms in the future. Indeed the data management systems for the search and result display facilities are designed to be primarily "black-box" control facilities. Addition of a training stage would require only the routing of the results of a template search to the net training facility. Overall the programming tasks necessary to integrate a suitable neural net processing facility into the system are relatively minor.

5.1.2 Comparison of template and voltage threshold techniques.

This section considers the threshold voltage method as an alternate means of classifying spike events. The results described in section 4.5.2 illustrate the limitations of voltage thresholding techniques. In this data set there are two distinct categories of spikes (identifiable on the basis of intracellular recording) which have identical minimum and maximum amplitudes. As expected voltage thresholding completely fails to separate this data.

It is clear that the simplest method which reliably identifies events of interest is likely to be the most satisfactory in a given situation. Thus it goes almost without saying that if the events of interest (spike waveforms corresponding to the activity of particular neurons) can be identified reliably on the basis of, say, peak voltage then more sophisticated methods are inappropriate. This is normally the case in the *Nephrops* system when considering spike classes F3 to F6.

It is also clear that when there is no significant difference between two waveforms (i.e. when the difference between the waveforms is less than the level of random noise), then it will not be possible to separate them on the basis of the electrical activity at a single point on the axon (Sarna *et al.*, 1988; Kreiter *et al.*, 1989). This in general leaves three main possibilities for the separation of waveform events.

First is the option of separating spike waveforms on the basis of the electrical activity at two distinct points on the axon bundle. Unless the axons have precisely the same electrical properties, the conduction velocities will differ (Schmidt and Stromberg, 1969; Roberts and Hartline, 1975), and consequently once the conduction velocities for the various axons have been established, the events can be separated by predicting the activity at the second recording point based on the activity at the first. This is a highly reliable method which has been used for this purpose (Schmidt and Stromberg, 1969; Roberts and Hartline, 1975), however it does require specific experimental procedures. It at least doubles the data recording requirements and, however desirable it might be to experiment in this fashion, there will be times when it may be impractical or impossible to do so.

The second approach is to use principal component analysis to derive the contributions (or eigenvalues) of the basis waveforms (i.e. those waveforms which can be linearly combined to give each actual spike), and then to use a clustering algorithm to identify spikes on the grouping of their eigenvalues (Abeles and Goldstein, 1977; Eggermont *et al.*, 1983). In mathematical terms this corresponds to performing a template or cluster search on the transform of the spike waveforms.

The third approach is to perform a template or cluster analysis directly on the untransformed waveform. Taking the data from a single electrode it is probable that the individual axons will generate waveforms which differ in shape over at least some part of their timecourse. On the assumption that the waveforms do not differ sufficiently in peak voltage that they can be distinguished this way, the use of a method considering the shape is a logical next approach.

The development of an approach like this cannot however be undertaken in isolation. It is necessary in the first instance to apply a shape based approach to data which can (albeit marginally) be analysed in another fashion. As described in section 5.1.8, the approach to shape matching used here is the generation of specific template waveforms which are then compared with all of the data waveforms. As a control (described in section 3.7.1) the data was suitable for event separation based on peak voltages and this was carried out separately. The critical separation was that of F1 and F2, in which the difference in recorded peak voltages was of the order of 20mV (approximately 12% of the overall spike height) in the data used, and this permitted the comparison of the template search methods with an established mechanism. As the results show, the template methods provide similar results to voltage thresholding when applied to real data. This does not of itself say anything about the reliability of the templating methods, since there is no independent check of the **actual** identity of the events.

However the use of tagged data (described in section 3.7.3) provides just such a check. The results of that analysis are promising though, depending on the requirements, somewhat mixed. Exclusion errors (false negative events) are rare, but at the expense of an undesirable inclusion error rate. To what extent this represents a limitation of the

MMD search method rather than specific difficulty in generating usable templates remains unanswered. A partial answer to this is given by the use of synthetic data, in which the search results can be compared with the known event identities using templates of a known quality. In particular the use of hybrid waveforms, built from linear combinations of F1 and F2 waveforms together with random noise, enables the discrimination capabilities of the MMD method to be established with much greater accuracy and, as the results show, with a fair degree of success. The key difference between the hybrid waveform data and single or overlapping event data is the duality of the waveform. It is neither F1 nor F2, and so tests the key feature of the MMD algorithm - the comparison of information content. Of course, to be strictly accurate, this comparison should use a linear combination weighted by the waveforms being combined, otherwise the cut-off point will not be at a 50/50 combination. One waveform will necessarily be predominant at this ratio. However, the test still shows discrimination capability in circumstances where neither voltage threshold nor voltage windowing techniques would be applicable.

Overall though, adequate testing to establish the limits of the template methods involves application to **multiple** real and synthetic data sets in order for the operational limits to be established with confidence.

5.1.3 Comparison of the techniques used with other template techniques.

The results described in section 4.5.2 illustrate the difficulty which faces any technique based upon a single point recording. The template mechanism employed by *Spike2* is a variation of the VE template, with a defined proportion of the data points making up an individual spike having to lie within the envelope for a match to be accepted. The templates are generated in the first instance by a clustering mechanism, with a limited amount of user input. As a commercial product, this system has presumably been refined and enhanced based upon the experience of users. However it still fails to perform well when challenged with difficult data.

The template matching techniques which are discussed in the literature are

primarily variations of the VE template method, mostly differing in the manner in which the templates are defined or the distance measurement used for the comparison. Many approaches utilise a small number of template datapoints, and much effort has been devoted to strategies for reducing the number of relevant datapoints per template, particularly in on-line or hardware based spike sorters. Baseline independent templates have been considered (Marion-Poll and Tobin, 1991), generated by differentiating the data waveforms prior to template generation and classification process. This study provides one of the few examples of the use of a mathematical as opposed to a statistical transform of the data waveforms prior to classification. Overall however, a number of common themes quickly become apparent from a literature review (see chapter 1). First is the reduction in data processing and storage requirements by reducing the number of datapoints in each spike waveform. This is usually carried out either by reducing an oversampled waveform (Jansen and Maat, 1992) or by feature extraction (Dinning and Sanderson, 1981; Wörgötter *et al.*, 1986; Salganicoff *et al.*, 1988; Kreiter *et al.*, 1989). Second, templates are defined during a "learning phase", usually a representative subset of the data (Salganicoff *et al.*, 1988; Kreiter *et al.*, 1989; Bergman and DeLong, 1992), but sometimes on a first pass through the entire data set (Jansen and Maat, 1992). Third, on-line systems have difficulty processing the data within expected minimum interspike intervals (Mishelevich, 1970; D'Hollander and Orban, 1979; Kreiter *et al.*, 1989; Bergman and DeLong, 1992), particularly when coupled with data storage. Fourth, the prevailing interest is in on-line systems (Mishelevich, 1970; D'Hollander and Orban, 1979; Dinning and Sanderson, 1981; Wörgötter *et al.*, 1986; Kreiter *et al.*, 1989; Salganicoff *et al.*, 1989) and these tend to involve various degrees of hardware implementation and dedicated computer facilities.

None of these template comparison methods consider the actual information content of the waveform, beyond defining the match distance or the voltage window in terms of the standard deviation of the background variation in the signal (sometimes, as with Bergman and DeLong (1992), allowing up to 5 standard deviations in routine operation). The VE template method takes limited account of the information content by allowing a derivative contribution to the template envelope as well as by permitting

templates to be defined on specific waveform features. However the MMD template expressly compares the information content of template and data, and is thus conceptually (though not in its operation) more akin to the principal component analysis approach.

5.1.4 Comparison of the variable envelope and merit distance techniques.

As described in chapter 4, both methods display the ability to separate similar waveforms such as F1 and F2 with broadly similar levels of effectiveness. The VE template method allows data to be searched for a single category of event, and this would be of particular use when applied to data containing many classes of events for which it would be impractical, or even impossible, to identify reliably templates for each category. This kind of data poses substantial problems even for systems using automated template generation facilities, due to the sheer number of resultant templates. Further, since the templates are not mutually exclusive, it is possible to define templates which are applicable to characteristic portions of the event waveform and then identify events based on matches to combinations of characteristic patterns.

The VE templates (as implemented) are very susceptible to problems of partial event overlap, though no more so than other template systems. This has definite implications in situations where event overlap is common and, while it is possible to use combinations of templates to classify events, there are obvious advantages to a system which does not require high levels of user intervention.

The MMD template method implicitly has the capability to "best guess" in situations where events do overlap. There will necessarily be a degree of incorrect assignment, but this is implicit in any method which attempts to resolve overlapping events in a single pass (see section 5.2). As the results of applying the MMD method to double event synthetic data show, this method is surprisingly effective in such situations. This is particularly apparent when considering that any variant of multiple voltage windowing technique would be completely inapplicable to such data. The MMD method is therefore a considerable improvement on systems which simply discard overlap

events.

One of the key questions that must be considered in respect of any software which attempts to automate an analysis process is whether it is easier to use the software or some other means to perform the analysis. This generalises to the question of whether it is worth performing the analysis at all. The answer to the second question obviously depends to a considerable extent on the value of the data. However, the first question is highly relevant to any consideration of working software. It is not enough to be able to perform a complex analysis if the processes of inputting the data and outputting the results are so cumbersome or error prone that an alternative approach is preferable. Correspondingly, the capability of handling the data with ease could in some instances simplify the analysis process to such a degree that complex analysis techniques would not actually be necessary for the system to have advantages over other approaches. This is clearly an extreme position, best suited to cases where the data retrieval is the obstacle rather than the analysis. However, it does reinforce the point that ease of use is vital in anything other than early prototype versions of the software.

To this extent, the signal matching system is necessarily a prototype and ease of use in respect of the interface could be said to be irrelevant except where the design does not allow for modification based on users' experience. However, the idea of using many combinations of partial templates is inherently complex and is therefore correspondingly weak. This does not mean that it is inappropriate on occasion, simply that it should not be a method of first choice.

Finally, the likely effects of data quality on the operation of both template methods must be considered. The VE templates are (as implemented) absolute, there are no degrees of match. Either the waveform matches the template or it does not, and therefore random noise must be accommodated within the levels of permitted error. However, the VE templates are base-line independent and, in consequence, drift in the base voltage will not have any effect. In contrast, the application of the MMD template method results in a "best" match under all circumstances. The MMD calculation should reduce the influence of random noise precisely because it is based on common subsequence identification rather than comparison of the whole sequence. However, the

assignment of a match to all events will necessarily result in some spurious assignments due to event overlap or ambiguity. This could be overcome by replacing the Boolean choice with a class specific threshold. However, this would require an additional threshold determining step with a consequent added requirement for user interaction.

As previously mentioned, there will inevitably be a certain degree of random noise present in the data. Also there will be a degree of variation between spike events of the same category. Obviously two spike classes can be separated by means of templates only if the normal variation within each class plus the normal level of noise leaves a measurable difference between classes over some portion of the waveform. In this respect, as in others, sophisticated analysis techniques are no substitute for good quality data. Rather, good data and sophisticated analysis should work together to produce a better end result. The alternative is to produce increasing quantities of poor quality results.

5.2 The effectiveness of the software.

The effectiveness of any piece of software can and indeed must be measured both in terms of the extent to which it performs its allotted task and the degree to which it is preferable to the available alternatives.

To consider then the signal analysis system in terms of its effectiveness in separating the key spike classes, it is necessary to look at the results from the single spike synthetic data. This shows an identification rate better than 75% for all spike classes even under conditions of significant noise degradation. Of course, given sufficiently low S/N ratios, the performance of any system degrades to unacceptable levels. However, it is reasonable to say that useful levels of identification are achieved with data reflecting realistic S/N ratios, and this using data which is inherently difficult to classify. In these terms the software must be regarded as being effective. By itself however this is an incomplete summary of a complex situation. The template matching methods employed and the strategy for their use may not in fact prove to be as effective as could be achieved with suitable refinement. As discussed earlier there are many

different approaches to the matching problem, and the best approach in terms of designing a piece of software to be used as an analytical tool is undoubtedly to allow for the addition of other capabilities. To this extent, the signal matching system is built around the template approach rather than an ad-hoc classification mechanism such as multi-dimensional clustering, and is correspondingly limited. In theory of course it could be modified to do anything, but the point of any software is that it should be possible to build on what exists rather than starting from the beginning each time and in these terms the signal analysis system does not provide a substantial framework for the implementation of a classification scheme which is not based on a comparison of data with templates. However to the extent that modified templating schemes could readily be implemented (see section 5.3) the design of the system must be considered a success.

Overall, however, the limit to the range of matching methods which could be applied with a reasonable degree of effort is only one part of the question. Automated data handling methods can enhance the process of data analysis by making it possible to manipulate sufficiently large data sets with a reasonable degree of effort. This is not to say that bigger is better. Indeed, in many instances, a small amount of data analysed by a skilled individual may be more valuable than a large amount classified automatically. However, in biological research statistical analysis is frequently performed on data sets which are either of insufficient size or which have been selected from the population in a non-random fashion. Many statistical techniques were originally devised for purposes such as agricultural research where large sample sizes are usual. Any means of simplifying the data handling process or performing even crude separation has the capability of yielding data sets of adequate size. Thus the signal analysis system has to be viewed also in terms of the other data gathering, analysis, and statistical software available, as well as the presentation software which will be used for the preparation of the end results. All too often there are such big gaps between the various steps from raw data to end results that analysing even one data set becomes an ordeal for the user. No one link in the chain can cure all the problems. However, as well as permitting the internal operations to proceed in a straightforward manner, a well designed piece of software has to move the data in and get the results out painlessly.

No novel software design is likely to resolve all the issues involved in the production of a good interface without going through two stages. First, those parts of the interface which provide the user with facilities akin to those commonly available should follow the normal metaphors for user interaction, and those parts which are novel should extend the normal metaphors as far as possible. Second, the design as initially implemented should be further refined to reflect the user's experience with the system. It may be that one of several different approaches to the interface design turns out to be much more satisfactory than the others. This is strictly analogous to the design of the underlying software, in which the problem can ultimately be expressed fully only when the solution is already known. As software becomes ubiquitous over the coming years, the number of situations in which novel kinds of operations are to be performed will decline. Also the current metaphors for interaction will be refined, some change may occur for purely commercial reasons but hopefully there will be an overall improvement.

The signal analysis system was unfortunately unable to pass through the second stage of interface refinement within the time available for the project, and the standard metaphors for interaction were themselves radically altered by the impact of *Microsoft Windows* during the course of the project. This left the software rather less refined than would have been desirable, however, the software is event driven and consequently the addition of new on-screen buttons or the reorganisation of the menus has little or no impact on the general design or its current implementation. Of more concern is what should actually be done with the processed output. The current provision of facilities to allow results to be imported into *Spike2* (together with the necessary *Spike2* macro script to accomplish this) and the corresponding capacity to format the results for importation into *Microsoft Excel* or other packages is not altogether adequate. These packages require their own scripts in order to perform further formatting and basic processing without necessitating substantial user intervention. Once again, time has proved to be the limiting factor in this, with such post-processing as has been required in the analysis of the data described in chapter 4 being undertaken partially manually.

5.3 Further development of the software.

Any good software design leaves room for additions and amendments. Indeed this is part of the definition of good design. Equally any design, however good, closes off some avenues of development. A good design places as little restriction on the flexibility of the system as is consistent with keeping the work required to implement that design within reasonable bounds. In other words some possibilities are designed in while others are designed out. This section will consider the types of operations which have been allowed for but not actually implemented in the signal analysis system, and the work which would be involved in pursuing each.

The most obvious deficiency in the entire system is the lack of any mechanism in the Result Viewer allowing the user to intervene and determine the classification of a specific event although this can be remedied by the use of *Spike2* at a post-processing stage. This would be most useful when combined with a search for events which remained unclassified after a template search of the database, and which thus represent overlapping or otherwise ambiguous events. The generation of unclassified events presently occurs with the VE template method and could be introduced with the MMD method by the definition of a maximum acceptable error for each template class. A user intervention facility would also require that the templates could be visualised within the Result Viewer, perhaps in a summary window showing a portion of the template database at one time. Also a mechanism for inputting the new match information would be required, a dialog box or (on a technically more complex level) a drag and drop facility would perform this task. The information would then be written back to the match database and thereafter processed in the usual way. At a similar level, the capability to load the match database corresponding to a previous search would be useful. Technically this is simple, requiring only that the match database is loaded at startup and internally flagged as existing rather than being created at search time. Searches would be unaffected, since they presently handle both the cases of no prior match database and existing databases. This facility was not needed for the purposes of the analysis described in chapter 4 and was accordingly not implemented.

Another consideration would be the addition of alternate result display formats. Specifically, waveform views become less useful when displaying long sections of data. Thus an approach using overlaid views of the waveforms for each spike class (as is common in the literature), or a bar chart summary might be useful supplements to the current display.

Finally, as regards data management issues, is the question of selecting the template file separately from the data file. This was not originally perceived as being an important facility, since judicious renaming of template files allows this to be implemented indirectly. However, it would be easier if the template file name was selectable rather than simply being derived from the data filename as at present.

A consideration of the template generation and analysis process leads to the possibility of a number of other facilities being developed. First, based on events matched it would be readily possible to generate average waveforms for each template category, together with the standard deviation of each datapoint. This information could be used to generate a new set of VE or MMD templates which could then be reapplied to the entire database. This is analogous to the use of templates defined on the basis of a "training period" covering part of the waveform database. Second, since long duration extracellular recordings involve both deterioration of the preparation and local variations in the sensitivity of the recording (perhaps due to the experimental regime itself) it is possible that templates which match well during the early portion of the database will be inappropriate at a later stage. This leads to the idea of modifying the templates based on the shapes of the waveforms currently being matched. This could take the form of an average of the matches up to the current position, a rolling average or weighted average of the last so many matches. Because this process is interactive, it is technically somewhat more involved than performing a second pass on the data. However, there is no major internal obstacle to its satisfactory implementation.

Some waveform events will either be unclassifiable or unreliably classified because they are actually composed of two overlapping spike events. In this case it is likely that the first event will be closely aligned to the expected position for a single event, since the alignment depends entirely on the voltage threshold used for

distinguishing spikes from background noise. This leaves the possibility that the waveform template for each spike class could be subtracted from the data waveform, and the result compared to each of the other template classes (using an appropriate temporal shift to compensate for the second spike not occurring simultaneously). If, as seems likely, the data waveform is simply the sum of the two spike events, then (on the assumption that there is one template per spike class) there will be one comparison which will yield a match, and consequently identify both waveforms. A second-pass search operating on these lines could fairly readily be added. All the support facilities already exist. However, the second-pass search routines would require to be coded separately as a modified form of the standard search to enable this mechanism to be integrated in a straightforward fashion. This could be extended to seek events occurring at user defined frequencies, on the basis that where the occurrence of regular firing patterns is detected there must be the suggestion that irregularities could be due to event overlap.

The event filtering facilities of the result viewer presently provide for event selection based on the occurrence of identified events on two (or more) data channels. However these events must be exactly concurrent, which in general is an unreasonable assumption when considering the extracellular activity of either two nerve fibres or two locations on the same fibre. The introduction of conditional event filters would permit the separation of correlated events. However this could not be readily achieved using the current event filter mechanism. Either a new facility could search for conditional events and write them to the event database or the existing facility for building events could be extended by the introduction of a restricted search on a second event channel. Of these two approaches the first has the merits of being conceptually simple and not interfering with operational code, while the second has the advantage that it does not alter the match database and is hence reversible.

No consideration of enhancements to the matching system would be complete without at least mentioning the possibility of performing the search on a convolved or transformed data set rather than on the original. The nature of the convolution and the templates would of course depend on the data, but the process of template generation and

searching is independent of the shape of the waveform, and consequently the convolution search could be implemented readily by means of a freestanding filter of the waveform database. Only slightly more complex would be a system generating a convolved waveform database on demand prior to template generation or searching operations. Either approach has the potential to add a powerful additional tool for the segregation of spike waveforms.

Finally, the implementation of additional non-template matching facilities should be considered. This is particularly important in respect of neural net methods which, of course, require extensive training on pre-identified data, and which are consequently well suited to a two pass system with the learning stage being a template matching algorithm. As with any other non-template method, the standard data management facilities are available to load data waveforms, store match information, and display filtered results as necessary. However the entire matching mechanism, together with the code to support any user interface requirements would have to be written. This could then be hooked into the parent window menu and the message handler as an extra freestanding item.

5.4 Summary of main conclusions.

In summary therefore, a *Windows* based database management system has been implemented on the IBM-PC as the basis of a system for the classification of nerve spike events in terms of their shape. The classification process has two stages, first the manual selection of template waveforms, and second the automated search for the templates using either of two identification algorithms. The first of these is a variation of a previously described contour matching algorithm (Kent, 1971; Akker *et al.*, 1982) while the second is an entirely novel application of an information comparison algorithm. The system has been successfully tested both with real and synthetic data. However, further refinement on the basis of the experience gained has not been undertaken. The system has the underlying flexibility to be extended in the future to cope with other matching methods, resulting in a potentially extremely versatile waveform classifier.

References.

Abeles, M. and Goldstein, M.H., Multispike train analysis. *Proc. IEEE*, **65**:762-773.

Adrion, W.R., Branstad, M.A. and Cherniavsky, J.C (1982) Validation, verification, and testing of computer software. *Computing Surveys*, **14**:159-192.

Akker, T.J. van den, Ros, H.H., Koelman, S.M. and Dekker, C. (1982) An on-line method for reliable detection of waveforms and subsequent estimation of events in physiological signals. *Comp. Biomed. Res.*, **15**:405-417.

Andreassen, S., Stein, R.B. and Oguztöreli, M.N. (1979) Application of optimal multichannel filtering to simulated nerve signals. *Biol. Cybern.*, **32**:25-33.

Augarten, S. (1984) Bit by bit. An illustrated history of computers. George Allen and Unwin, London.

Bak, M.J. and Schmidt, E.M. (1977) An improved time-amplitude window discriminator. *IEEE Trans. Biomed. Eng.*, **BME-24**:486-489.

Bergman, H. and DeLong, M. (1992) A personal computer-based spike detector and sorter: implementation and evaluation. *J. Neurosci. Methods*, **41**:187- 197.

Bessou, P. and Perl, E.R. (1969) Response of cutaneous sensory units with unmyelinated fibres to noxious stimuli. *J. Neurophysiol.*, **32**:1025-1043.

Bradley, W., Conway, C., Glover, E. and McCormick, S. (1967) Discriminator and integrator instrument for an on line frequency analysis of single unit discharges. *Electroenceph. clin. Neurophysiol.*, **22**:177-179.

Bures, J., Petrán, M. and Zachar, J. (1967) Electrophysiological methods in biological research. Academic Press, New York and London.

Camp, C. and Pinsker, H. (1979) Computer separation of unitary spikes from whole-nerve recordings. *Brain. Res.*, **169**:455-479. Caplan, R.M. (1990) How fingerprints came into use for personal identification. *J. Am. Acad. Dermatol.*, **23**:109-14.

Cohen, A. and Landsberg, D. (1983) Adaptive real-time wavelet detection. *IEEE Trans. Biomed. Eng.*, **BME-30**:332-340.

Denheen, M.T. (1992) Central and peripheral actions of the neuropeptide proctolin on a postural neuromuscular muscle system in the Norway lobster *Nephrops norvegicus* (L.). Ph.D. thesis, Glasgow University.

Dettman, T. (1989) DOS programmers reference. Que, Carmel, California.

Dinning, G.J. and Sanderson, A.C. (1981) Real-time classification of multiunit neural signals using reduced feature sets. *IEEE Trans. Biomed. Eng.*, **BME-28**:804-811.

D'Hollander, E.H. and Orban, G.A. (1979) Spike recognition and on-line classification by unsupervised learning systems. *IEEE Trans. Biomed. Eng.*, **BME-26**:279-284.

Eggermont, J.J., Epping, W.J.M. and Aertsen, A.M.H.J. (1983) Stimulus dependent neural correlations in the auditory midbrain of the grassfrog (*Rana temporaria* L.). *Biol. Cybern.*, **47**:103-117.

Forney, J. (1989) MS-DOS beyond 640k: working with extended and expanded memory. Windcrest, Blue Ridge Summit, Pennsylvania.

Forrest, S. (1993) Genetic algorithms: principles of natural selection applied to computation. *Science*, **261**:872-878.

Freeman, J.A. (1971) A simple multichannel spike height discriminator. *J. App. Physiol.*, **31**:939-941.

Gerstein, G.L. and Clark, W.A. (1964) Simultaneous studies of firing patterns in several neurons. *Science*, **143**:1325-1327.

Glaser, E.M. and Ruchkin, D.S. (1976) Principles of neurobiological signal analysis. Academic Press, New York and London.

Harris-Warwick, R.M. and Kravitz, E.A. (1984) Cellular mechanisms for modulation of posture by octopamine and serotonin in the lobster. *J Neurosci.*, **4**:1976-1993.

Hart, A. (1992) Using neural networks for classification tasks - some experiments on datasets and practical advice. *J. Opl. Res. Soc.*, **43**:215- 226.

Hartigan, J.A. (1975) Clustering algorithms. Wiley, New York.

Hasegawa, A. and Kodama, Y. (1981) Signal transmission by optical solitons in optical fibre communications. *Proc. IEEE*, **69**:1145-1150.

Heetderks, W.J. and Williams, W.J. (1975) Partition of gross peripheral nerve activity into single unit responses by correlation techniques. *Science*, **188**:373-375.

Hermann, H.T., Stark, L. and Willis, P.A. (1962) Instrumentation for processing neural signals. *Electroenceph. clin. Neurophysiol.*, **14**:557- 560.

Jansen, R.F. (1990) The reconstruction of individual spike trains from extracellular multineuron recordings using a neural network emulation program. *J. Neurosci. Methods*, **35**:203-213.

Jansen, R.F. and Maat, A.T. (1992) Automatic waveform classification of extracellular multineuron recordings. *J. Neurosci. Methods*, **42**:123-132.

Kanz, J., Camp, C. and Pinsker, H. (1978) Computer separation of unitary spikes from whole nerve recordings. *Fed. Proc.*, **37**:219.

Kennedy, D. and Takeda, K. (1965) Reflex control of abdominal flexor muscles in the crayfish. II. The tonic system. *J. exp Biol.*, **43**:229-246.

Kent, E.W. (1971) An educable wave form recognition program suitable for on-line discrimination of extracellular single units. *Electroenceph. clin. Neurophysiol.*, **31**:618-620.

King, A. (1994) Windows the next generation: an advance look at the architecture of chicago. *Microsoft Systems J.*, **3(1)**:7-19.

Knox, P.C. and Neil, D.M. (1991) The coordinated action of abdominal postural and swimmeret motor systems in relation to body tilt in the pitch plane in the Norway lobster *Nephrops norvegicus*. *J. exp. Biol.*, **155**:605-627.

Kreiter, A.K. Aertsen, A.M.H.J. and Gerstein, G.L. (1989) A low-cost single-board solution for real-time, unsupervised waveform classification of multineuron recordings. *J. Neurosci. Methods*, **30**:59-69.

Kruskal, J.B. (1983) An overview of sequence comparison. In: *Time warps, string edits and macromolecules: the theory and practise of sequence comparison.* (Sankoff, D. and Kruskal, J.B. eds.) Addison-Wesley, Reading, Massachusetts.

Landolt, J.P. and Milliken, W.E. (1970) A discriminator and integrator device for multi-unit neural activity analysis. *Electroenceph. clin. Neurophysiol.*, **28**:83-84.

Leibrock, C.S. (1993) Processing of mechanoreceptive input in crayfish: an in vivo and in vitro study of the role of the cuticular stress detectors in motor control. Ph.D. thesis, Glasgow University.

Littauer, R.M. and Walcott, C. (1959) Pulse-height analyser for neuro-physiological applications. *Rev. Sci. Instr.*, **30**:1102-1106.

MacNichol, E.F. and Jacobs, J.A.H. (1955) Electronic device for measuring reciprocal time intervals. *Rev. Sci. Instr.*, **26**:1176-1180.

Marion-Poll, F. and Tobin, T.R. (1991) Software filter for detecting spikes superimposed on a fluctuating baseline. *J. Neurosci. Methods*, **37**:1-6.

McCann, G.D. (1973) Interactive computer strategies for living nervous system research. *IEEE Trans. Biomed. Eng.*, **BME-20**:1-11.

McConnell, S. (1993) Code complete: a practical handbook of software construction. Microsoft Press,

Redmond, Washington.

McNaughton, B.L., O'Keefe, J. and Barnes, C.A. (1983) The stereotrode: A new technique for simultaneous isolation of several single units in the central nervous system from multiple unit records. *J. Neurosci. Methods*, **8**:391-397.

Metcalf, M. (1992) Still programming after all these years. *New Scientist*, **135(1838)**:30-33.

Microsoft (1990) Microsoft C advanced programming techniques. Microsoft Corporation, Redmond, Washington.

Millar, J. (1983) A 'wavegate' spike discriminator for sorting extracellular nerve action potentials. *J. Neurosci. Methods*, **7**:157-164.

Millecchia, R. and McIntyre, T. (1978) Automatic nerve impulse identification and separation. *Comp. Biomed. Res.*, **11**:459-468.

Mishelevich, D.J. (1970) On-line real-time digital computer separation of extracellular neuroelectric signals. *IEEE Trans. Biomed. Eng.*, **BME- 17**:147-150.

Neumann, P.G. (1995) Computer related risks. Addison-Wesley, Reading, Masachusetts.

Oguztöreli, M.N. and Stein, R.B. (1977) Optimal filtering of nerve signals. *Biol. Cybern.*, **27**:41-48.

O'Connell, R.J. and Schoenfeld, R.L. (1973) Minicomputer identification and timing of nerve impulses mixed in a single channel recording. *Proc. IEEE*, **61**:1615-1621.

Petzold, C. (1990) Programming windows. Microsoft Press, Redmond, Washington.

Press, W.H., Teukolsky, S.A., Vetterling, W.T. and Flannery B.P. (1992) Numerical recipes in C: the art of scientific computing. Cambridge University Press, Cambridge.

Prochazka, V.J., Conrad, B. and Sindermann, F. (1972) A neuroelectric signal recognition system. *Electroenceph. clin. Neurophysiol.*, **32**:95-97.

Prochazka, V.J. and Kornhuber, H. (1973) On-line multi-unit sorting with resolution of superposition potentials. *Electroenceph. clin. Neurophysiol.*, **34**:91-93.

Remmel, R.S. (1983) A computerised discriminator for action potentials. *Electroenceph. clin. Neurophysiol.*, **56**:528-530.

Roberts, W.M. and Hartline, D.K. (1975) Separation of multi-unit nerve impulse trains by a multi-channel linear filter algorithm. *Brain. Res.*, **94**:141-149.

Roberts, W.M. (1979) Optimal recognition of neuronal waveforms. *Biol. Cybern.*, **35**:73-80.

Salganicoff, M. Sarna, M.F., Sax, L. and Gerstein, G.L. (1988) Unsupervised waveform classification

for multi-neuron recordings: a real-time, software based system. I Algorithms and implementation. *J. Neurosci. Methods*, **25**:189-196.

Sarna, M.F., Gochin, P., Kaltenbach, J., Salganicoff, M. and Gerstein, G.L. (1988) Unsupervised waveform classification for multi-neuron recordings: a real-time, software based system. II Performance comparison to other sorters. *J. Neurosci. Methods*, **25**:189-196.

Schmidt, E.M. (1971) An instrument for separation of multiple-unit neuroelectric signals. *IEEE Trans. Biomed. Eng.*, **BME-18**:155-157.

Schmidt, E.M. (1984a) Instruments for sorting neuroelectric data: a review. *J. Neurosci. Methods*, **12**:1-24.

Schmidt, E.M. (1984b) Computer separation of multi-unit neuroelectric data: a review. *J. Neurosci. Methods*, **12**:95-111.

Schmidt, E.M. and Stromberg, M.W. (1969) Computer dissection of peripheral nerve bundle activity. *Comp. Biomed. Res.*, **2**:446-455.

Simon, W. (1965) The real-time sorting of meuro-electric action potentials in multiple unit studies. *Electroenceph. clin. Neurophysiol.*, **18**:192-195.

Sokolove, P.G. and Tatton, W.G. (1975) Analysis of postural motorneuron activity in crayfish abdomen. I. Coordination by premotoneuron connections. *J. Neurophysiol.*, **38**:313-331.

Stein, R.B., Andreassen, S. and Oguztöreli, M.N. (1979) Mathematical analysis of optimal multichannel filtering for nerve signals. *Biol. Cybern.*, **32**:19- 24.

Stremler, F.G. (1990) Introduction to communication systems. Addison-Wesley, Reading, Massachusetts.

Studer, R.M., de Figueiredo, R.J.P., Moschytz, G.S. (1984) An algorithm for sequential signal estimation and system identification for EMG signals. *IEEE Trans. Biomed. Eng.*, **BME-31**:285-294.

Tatton, W.G. and Sokolove, P.G. (1975) Analysis of postural motorneuron activity in crayfish abdomen. II. Coordination by excitatory and inhibitory connections between motoneurons. *J. Neurophysiol.*, **38**:332- 346.

Thompson, C.S. and Page, C.H. (1982) Command fibre activation of superficial flexor motoneurons in the lobster abdomen. *J. comp. Physiol.*, **148**:515-527.

Wine, J.J., Mittenhal, J.E. and Kennedy, D. (1974) The structure of tonic flexor motoneurons in

crayfish abdominal ganglia. *J. comp. Physiol.*, **93**:315-335.

Wörgötter, F., Daunicht, W.J. and Eckmiller, R. (1986) An on-line spike form discriminator for extracellular recordings based on an analog correlation technique. *J. Neurosci. Methods*, **17**:141-151.

Yamada, S., Kage, H., Nakashima, M., Shiono, S. and Maeda, M. (1992) Data processing for multi-channel optical recording: action potential detection by neural network. *J. Neurosci. Methods*, **43**:23-33.

Ziv, J. and Lempel A. (1977) A universal algorithm for sequential data compression. *IEEE Trans. Inform. Theory*, **IT-23**:337-343.

Ziv, J. and Lempel A. (1978) Compression of individual sequences via variable- rate coding. *IEEE Trans. Inform. Theory*, **IT-24**:530-536.

Appendix A.

χ^2 calculation for the results given in table 4.3A.

Observed values

	R0	R1	R2	R3	R4	R5	Totals
No of events	1549	1663	1005	2979	2960	2000	12156
MMD templates	767	514	9	24	101	19	1434
Intracellular totals	2316	2177	1014	3003	3061	2019	13590

Expected values

	R0	R1	R2	R3	R4	R5	Totals
No of events	2071.6	1947.3	907.0	2686.1	2738.0	1806.0	12156
MMD templates	244.4	229.7	107.0	316.9	323.0	213.0	1434
Intracellular totals	2316	2177	1014	3003	3061	2019	13590

$\chi^2 = (1549\text{-}2071.6)^2/2071.6 + (1663\text{-}1947.3)^2/1947.3 + (1005\text{-}907.0)^2/907.0 + (2979\text{-}2686.1)^2/2686.1 + (2960\text{-}2738.0)^2/2738.0 + (2000\text{-}1806.0)^2/1806.0 + (767\text{-}244.4)^2/244.4 + (514\text{-}229.7)^2/229.7 + (9\text{-}107.0)^2/107.0 + (24\text{-}316.9)^2/316.9 + (101\text{-}323.0)^2/323.0 + (19\text{-}213.0)^2/213.0$

$= 2413.8$

χ^2 for $p < 0.01$ (5 degrees of freedom) $= 15.09$

i.e. $p << 0.01$

χ^2 calculation for the results given in table 4.3B.

Observed values

	R0	R1	Totals
No of events	1606	821	2427
MMD templates	1634	579	2213
Intracellular totals	3240	1400	4640

Expected values

	R0	R1	Totals
No of events	1694.7	732.3	2427
MMD templates	1545.3	667.7	2213
Intracellular totals	3240	1400	4640

$\chi^2 = (1606-1694.7)^2/1694.7 + (821-732.3)^2/732.3 + (1634-1545.3)^2/1545.3 + (579-667.7)^2/667.7$

$= 32.25$

χ^2 for $p < 0.01$ (1 degree of freedom) = 6.63

i.e. $p << 0.01$