3 An Introduction to Digital Signal Processing of Neural Signals

3.1 Introduction

In this chapter, we will present a short introduction to digital signal processing techniques related to neural signals that are suitable for implantable devices and neural recording embedded systems. These techniques, being *basic* processing methods, are related to finding action potentials in neural signals and classifying them according to their signal shape.

As discussed in Chapter 0, after finding the firing patterns in neural signals, we will be able to translate the neural activity (firing rate of each individual neuron) to a meaningful action, for example, moving a robotic arm. In general, an *action* that is intended by the brain is realized via changes in firing rates of different neurons. As a result, finding firing patterns of many neurons, can result in inferring the intended action [2]. In order to find firing patterns of many neurons, the fired action potential must first be detected. Then, the detected action potential must be associated with a nearby neuron according to its signal shape [36] [70]. After these two steps, the firing rate of a neuron, at a certain point in time, can be described in firings/second. In this chapter, terms *spike* and *action potential* are used interchangeably.

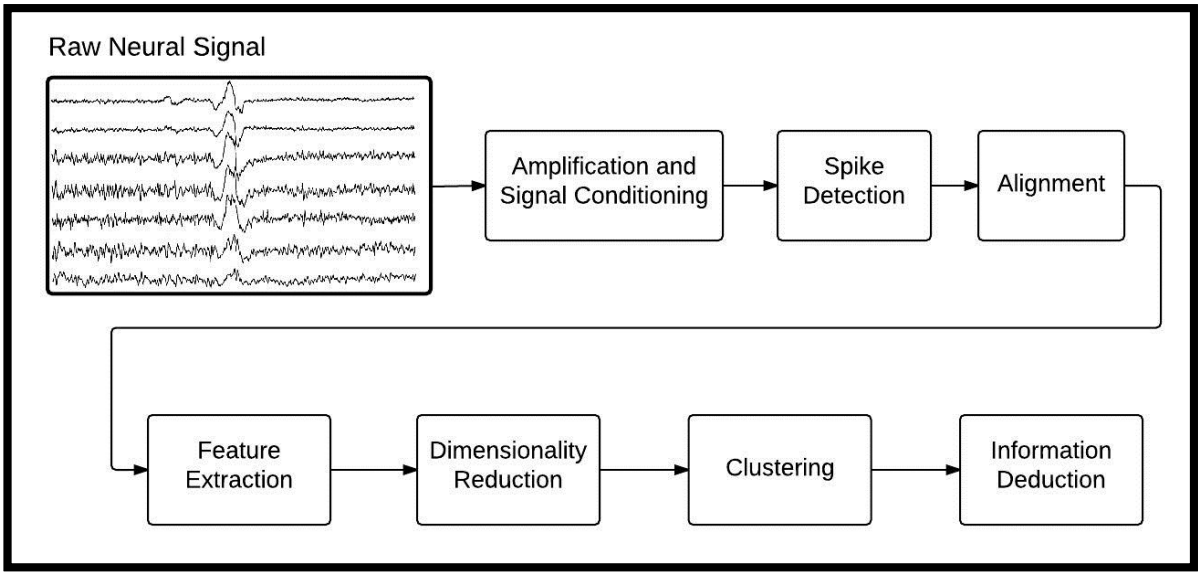

Figure 5. Overview of neural signal processing for spike sorting.

As mentioned in the previous chapter, action potentials can be recorded using electrodes. The electrodes can be placed inside the brain at proper locations to pick up the action potentials fired by one or more neurons. The (signal) shape of the action potential picked up by an electrode, depends on different parameters including

the distance between the electrode and the neuron, and also the type/shape/size of the neuron. So if there are multiple neurons in close proximity of an electrode, it can be predicted that *action potentials fired by each neuron will have a similar shape that is different from the action potentials from the other neurons* [36] [70]. The difference in action potential shapes is the basis of action potential classification, a process that is also called *spike sorting*. Figure 5 shows an overview of the required actions to sort (classify) action potentials. Spike sorting algorithms use the shape of each action potential to relate it to one neuron in the proximity of the recording electrode [36] [70].

In this chapter, we will present different steps of typical spike sorting algorithms. The algorithms reviewed in this chapter are mostly designed for small neural recording embedded systems. These systems, either based on discrete components or VLSI (very-large-scale integration) technology, usually have a limited processing power i.e., they are not high-speed as desktop computers and they have limited memory resources. The need for neural signal processing in (low-power) embedded systems and/or implantable devices comes from the fact that a real-time brain-machine interface (connected to the brain) cannot rely on offline signal processing and needs to react to neural activities in real-time.

3.2 Overview of Steps in Spike Sorting Algorithms

In order to classify (sort) the spikes according to their shapes (in the time domain), we first need to detect and extract the spikes i.e., the time-window containing the spike must be pinpointed in the neural signal [36] [3] [33] [70]. Figure 6 shows such time-window around a typical action potential. It is assumed that the neural signal is already bandpass-filtered as discussed in section 2.5.3. This bandpass-filtering removes the LFP (local field potential) component of the neural signal.

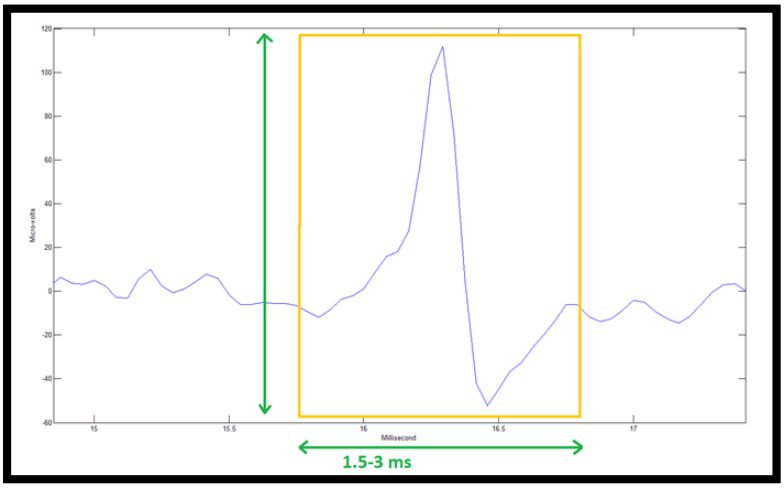

Figure 6. Time-window required to process an action potential.

After detection, spikes are usually *aligned* with respect to a certain point in their waveform. For example, spikes can be aligned in a way that their largest samples (the maximum in the waveform) would be placed at the same point if all detected spikes are overlapped [36] [70]. More details about spike detection and alignment will be provided in the subsequent sections of this chapter.

In the next step of spike sorting, the prepared (detected and aligned) spikes must be passed through a *feature extraction* algorithm where prominent characteristics of each spike are extracted. The extracted features are usually in the form of multidimensional vectors. There are various ways of extracting such vectors. The most widely used methods are reviewed in this chapter.

Finally, in the last step of spike sorting, the extracted feature vectors pointing to different points in a multidimensional space, must be *clustered* together i.e., the points that are in close proximity of each other must be assigned to the same group (i.e., the same neuron) as it is expected that they are fired from the same neuron [36] [70]. Sometimes the length of the features vectors is too long, resulting in complicated clustering (in terms of required time and memory). As a result, a *dimensionality reduction* step might precede the clustering step.

In the following sections of this chapter, different methods of spike detection, alignment and clustering are discussed in more detail.

3.3 Spike Detection

Spike detection is the process of discriminating action potentials from the background noise in a neural signal. Figure 7 shows a typical neural signal that is passed through a band-pass filter. It can be seen that the spikes can be evident as the noise level is low compared to the signal level; this is the basis of different spike detection methods [32] [36] [70].

Spike detection methods usually consist of two steps. In the first step, the neural signal is passed through some sort of mathematical operator that makes the spikes more prominent and in the second step, the output of the first step is compared with one or two thresholds [36]. In the second step, when the output of the first step is passes through the threshold(s), a recording mechanism is activated that stores the digitized samples of the action potential in some sort of memory. The number of recorded samples (i.e., the length of the time window that is saved) is usually constant [36] [70] [3] [71].

The most widely used mathematical operators for implantable devices (used in the first step of spike detection) are the absolute value and the Teager Energy Operator [72] [70]. However, the neural signal can be detected

via comparison with a threshold without any pre-emphasis on the spikes. In the following subsection, we will introduce the mentioned spike detection methods.

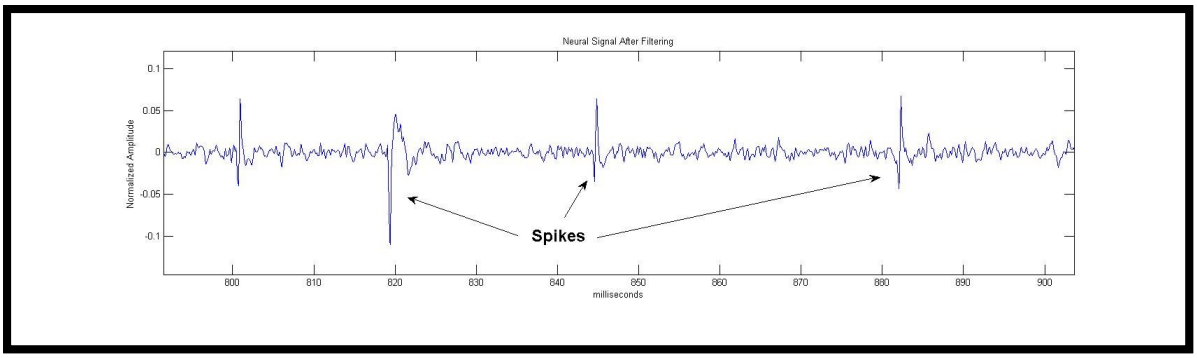

Figure 7. Typical band-pass-filtered neural signal.

3.3.1 Direct Comparison with Threshold

In this technique, the neural signal is directly compared with one or two thresholds [3] [36] [70]. This method is the simplest method of spike detection and, depending on the noise level of the recorded signal, can be the most efficient method too.

The threshold used in this method can be chosen manually or can be derived from the signal statistics [36] [70]. A widely used threshold for raw neural signals is derived from the median of the signal [33] [36] [70]:

$$Threshold = 4 * median\left\{\frac{|x(n)|}{0.6745}\right\}$$

This threshold can only be valid when all or most of the bandpass-filtered neural signal is available; as a result, real-time systems that employ this threshold should have a separate memory for storing the signal calculating the thresholds.

3.3.2 Absolute Value

As can be seen in Figure 7, depending on the relative position of the electrodes in the tissue, the action potentials might have a larger positive or negative peak. So it is more appropriate to compare the absolute value of the neural signal with a threshold [36] [73] [70]. This technique has been proven to have a better quality for spike detection than comparing the raw neural signal with a threshold [73]. The threshold used in this method can be the same threshold used for raw neural signals [36] [70].

3.3.3 Teager Energy Operator (TEO)

TEO (also known as the Nonlinear Energy Operator) is a nonlinear mathematical operator that has been devised for tone detection and FM demodulation [72] [70].

It is defined as

$$TEO\{x(n)\} = x(n)^2 - x(n+1) \times x(n-1)$$

TEO has the property of amplifying the high-frequency components of the signal. So, action potentials having higher frequency components than the background noise can result in higher values at the output of the TEO [36] [70].

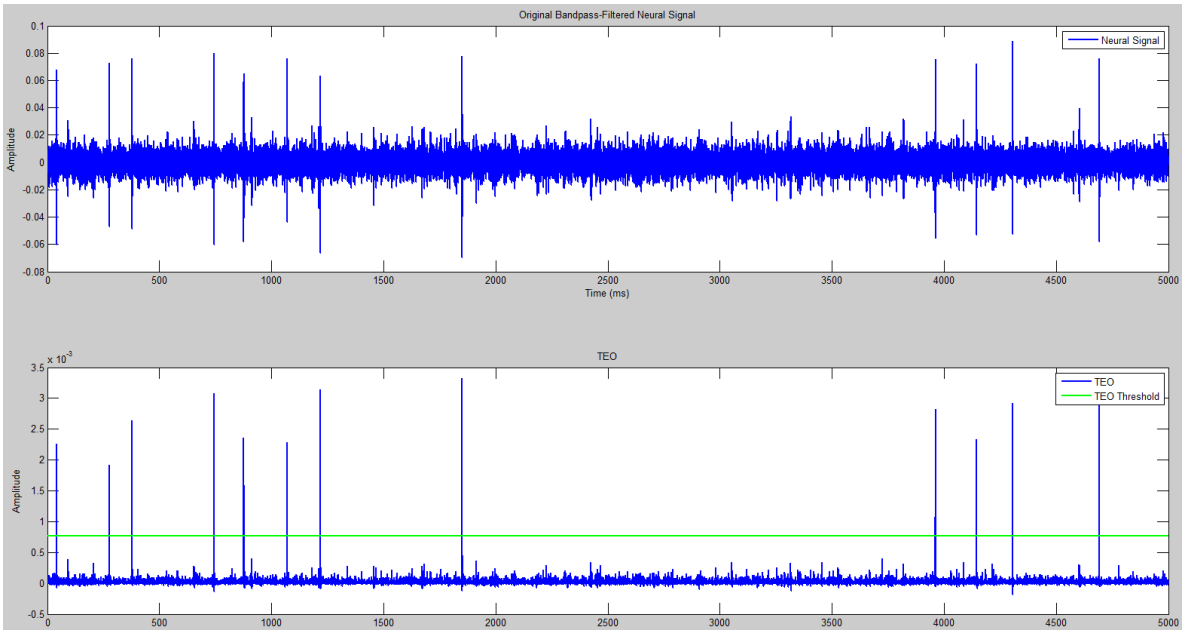

Figure 8. Neural signal passed through TEO.

Figure 8 shows a typical bandpass-filtered neural signal fed into the TEO and the output of the TEO. It can be seen that TEO emphasizes action potentials. Similar to previous methods, in order to detect spikes, the outputs of the TEO are compared with a threshold. This threshold can be derived from the average value of the TEO output because TEO emphasizes the spikes and deemphasizes the background noise—making their weight in averaging almost negligible [36] [70]. More formally, the TEO threshold can be stated as:

$$TEO\ Threshold = C \times \frac{1}{N} \sum_{i=1}^{N} TEO\{x(n)\}$$

where *C* is constant.

The fact that TEO threshold can be derived from averaging rather than calculating a median, makes threshold calculation much easier than the case of absolute value as calculating the mean value does not need large memories.

Besides the classical TEO, there is also a variant of the TEO called the k-TEO [74]. This operator is similar to the TEO, however, instead of delays of one sample, delays of *k* samples are used. K-TEO is defined as:

$$k\text{-}TEO\{x(n)\} = x(n)^2 - x(n+k) \times x(n-k)$$

It can be shown that k-TEO can be tuned, using the value of *k*, to detect action potentials that are wider. Also, [74] shows the application of a bank of k-TEO operators for spike detection.

3.3.4 Comparison of Spike Detection Algorithms

Different spike detection algorithms have been introduced in the previous subsections. Since these algorithms can be considered as binary classification tests, we can measure their performance using ROC (receiver operating characteristic) curves [36] [70]. The ROC curve shows the true positive rate as a function of false positive rate [75]. In general, it is desirable to have more true positive rate and less false positive rate. A comparison presented in [36] shows that the absolute value and the TEO have very close characteristics in the ROC curve. However the authors of this paper mention that, by considering the choice probabilities and the resilience to noise, TEO outperforms the absolute value. It should also be mentioned that the implementation cost in terms of operations or chip area of TEO is higher than the absolute value method [36] [70].

3.4 Spike Alignment

After detection, the spikes need to be aligned in a way that their differences/similarities would be as prominent as possible. This task is called alignment and it is assumed that all detected spikes occupy a constant number of samples i.e., constant temporal length.

Two spike alignment methods are mostly used [36] [70]:

1) Alignment with respect to sample with the maximum value: in this method, all spike waveforms are arranged in a way that their sample, having the maximum value, is at a certain point in time when all spike waveforms are overlapped.
2) Alignment with respect to the maximum derivative: in this method, all spike waveforms are arranged in a way that their sample, having the maximum slope (with respect to the previous sample), is at a constant point in time when all spike waveforms are overlapped.

Figure 9 and Figure 10 show spike alignment using the maximum value and maximum slope, respectively. In Figure 9, there are two different spike groups (having different shapes) and it can be seen that all spikes are *pinned* at the maximum value. On the other hand in Figure 10, three different groups of spikes are pinned at the point where they have the maximum slope (positive or negative).

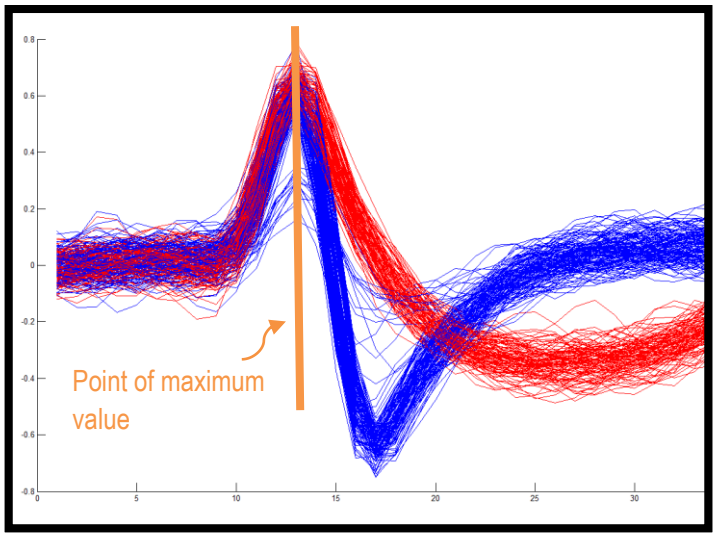

Figure 9. Spike alignment using maximum value.

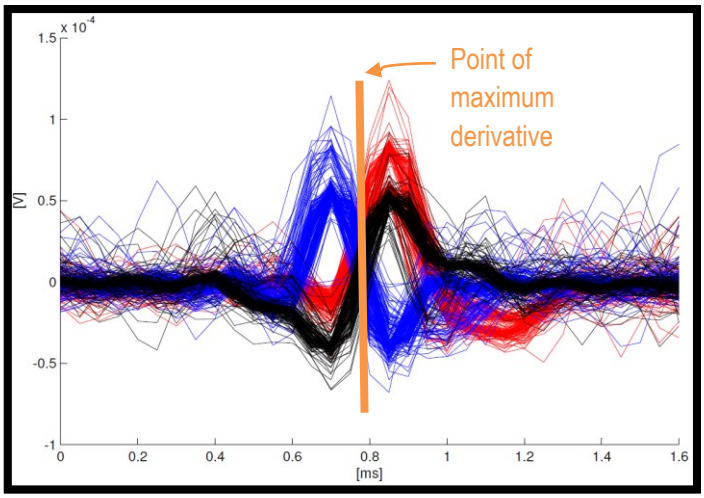

Figure 10. Spike alignment using the maximum slope.

3.5 Feature Extraction

As evident in Figure 9 and Figure 10, spike alignment maximizes the differences between the action potentials. After alignment, a *feature extraction* algorithm is needed to convert the shape of each spike to an *n*-dimensional vector. It should be noted that action potentials, being vectors of real numbers, can be already

considered as points in the space. However feature extraction algorithms tends to elaborate the prominent spike features using less number of samples.

The most widely used feature extraction methods are Principal Component Analysis (PCA), Discrete Wavelet Transform (DWT), Discrete Derivatives (DD) and Integral Transform (IT) [36] [70]. In the following sections, each feature extraction method is elaborated.

3.5.1 Principal Component Analysis

PCA is a widely used method that has been used numerous times in spike sorting applications. PCA tries to find an orthogonal basis for the data (set of all detected spikes) whose variations are maximal. This orthogonal basis can be derived via eigenvalue decomposition of the covariance matrix [36] [70] [76].

After finding the principal components, each spike can be expressed in the new basis as a set a values c_i:

$$c_i = \sum_{n=1}^{N} PC_i(n) \times s(n)$$

where $s(n)$, the spike, is a vector, $PC_i(n)$ is the i^{th} principal component and N is the number of elements in the spike vector [70].

The c_i vector or a subset of its elements can be used for *clustering* as most of the data variations are represented in the first few principal components [36] [70].

Although PCA has been widely used as a feature extraction method, it has the disadvantage that it requires relatively many multiplications and additions making it inappropriate for real-time feature extraction in embedded systems with limits on computation power.

3.5.2 Discrete Wavelet Transform

DWT, similar to PCA, is used to derive a set of coefficients from the data that represent the differences in the data more clearly [36] [70]. DWT coefficients are derived using

$$DWT(u,j) = \sum_{n=-\infty}^{+\infty} s(n) \times \frac{\Psi(\frac{n-u}{2^j})}{2^{j/2}}$$

where u is the *translation* parameter and 2^j is the *scaling* parameter with j an integer.

The Ψ function is the mother wavelet function that is different depending on the wavelet family used for the transform. The DWT operation can be implemented as a filter bank and depending on the wavelet family, the DWT might involve many multiplication/addition operations.

3.5.3 Discrete Derivatives

Discrete Derivatives (DD) is a low-complexity method similar to a simplified version of DWT [36] [70]. It calculates the slope between the n^{th} samples and the $(n-d)^{th}$ sample:

$$DD_d(n) = x(n) - x(n-d)$$

3.5.4 Integral Transform

The Integral Transform (IT) is another low-complexity method useful for real-time feature extraction that produces a feature vector of two elements. It calculates the average nonnegative phase and the average negative phase of the action potential waveform:

$$\begin{cases} IT_P = \dfrac{1}{N_P}\sum s(n) & \text{where } s(n) \geq 0 \\ IT_N = \dfrac{1}{N_N}\sum s(n) & \text{where } s(n) < 0 \end{cases}$$

where N_P and N_N are the number of nonnegative and negative samples of the action potential.

The feature vector will be $[IT_P\ IT_N]$, resulting in simple two-dimensional clustering.

3.5.5 Comparison of Feature Extraction Algorithms

In terms of implementation costs, the PCA and DWT methods are an order of magnitude more complex than the IT and DD methods. This is due to their memory and computation (multiplications and additions) requirements [36]. In [36] and [70], the accuracy of these feature extraction algorithms has been plotted versus their computational cost and it has been deduced that since the DD algorithm lies at the knee point of the curve it has the best accuracy-complexity tradeoff for small embedded or implantable systems.

3.6 Data Clustering Algorithms

The last step of spike sorting, also being the most complex one, is clustering the feature vectors. It should be noted that the clustering algorithms presented in this section consider the feature vectors as points in a multi-dimensional space.

Before talking about the details of clustering algorithms, it should be mentioned that there is another step, usually taken in spike sorting algorithms, that is not introduced in this chapter. This step consists of reducing the number of elements in feature vectors also known as *dimensionality reduction* [36] [70]. There are various ways to reduce the number of elements in features vectors [70] including uniform resampling, Lilliefors Test [77] and Hartigan's Dip Test [78].

Uniform resampling consists of keeping only one element out of each *k* elements in the feature vectors. However, the other two tests try to find multimodality in the elements of the feature vectors and as a result, they are more efficient. Uniform resampling can be easily implemented in low-resource embedded systems. However, the two other tests require large memories and high processing speeds, which is not suitable for low-power neural recording applications.

In the following subsections, we will present the most widely-used data clustering algorithms. It should be noted that these algorithms are more power/memory-hungry than the algorithms involved in spike detection, alignment and feature extraction. Also, they are usually iterative algorithms with an unknown number of iterations.

3.6.1 K-Means Algorithm

The k-Means algorithm [79] classifies the data points into *k* different groups where *k* is provided by the user. So, it is a supervised method (i.e., it requires manually selected parameters). This algorithm starts by randomly defining *k* cluster centroids and then, through many iterations, it recalculates the centroids by measuring the Euclidean distance between the centroids and the data points.

3.6.2 Valley Seeking Algorithm

This algorithm is based on calculating the normalized density derivatives (NDD) and finding the peaks of these functions [80]. It has the advantage of being nonparametric and unsupervised. However, its disadvantage is requiring relatively large processing power making it unsuitable for implantable applications [70].

3.6.3 Superparamagnetic Clustering

Superparamagnetic Clustering (SPC) is a clustering method inspired from the physical properties of the inhomogeneous ferromagnetic model [33] [81]. This clustering method, being unsupervised and accurate, has been used by [33] along with DWT for spike sorting. However, similar to many other clustering algorithms, this algorithm requires a large processing power.

3.6.4 Osort

This method [82], devised by neuroscientists, is both unsupervised and can be implemented on relatively small embedded systems. In Osort, the first data point (the first spike waveform) becomes its own cluster. For the following spikes, the distance (usually Euclidean) to all cluster centroids is computed and the minimum distance is considered. If this minimum distance is less than the merging threshold, the spike is added to the nearest cluster and the cluster centroid is recalculated, otherwise a new cluster is generated. While processing the incoming spikes, if the distance between two clusters becomes less than a certain value, the two clusters are merged.

3.6.5 Comparison of Clustering Algorithms

Table 8 (derived from [70]) shows the tradeoffs between different clustering algorithms. It can be seen that Osort provides the best tradeoff between implementation complexity, speed and need for supervision.

In this chapter, only the clustering *algorithms* were discussed. However, it should be mentioned that manual spike sorting, based on the visual feedback from the extracted features, can also be a viable solution. But, manual clustering cannot be real-time, hence not implantable in neural recording embedded systems.

Table 8. Comparison of clustering algorithms.

	Manual	**k-Means**	**Valley Seeking**	**SPC**	**Osort**
Nonparametric	NO	NO	YES	YES	NO
Unsupervised	NO	NO	YES	YES	YES
Real-Time	NO	NO	NO	NO	YES
Adaptive	NO	NO	NO	NO	YES
Complexity	-	LOW	HIGH	HIGH	LOW

3.7 Some Notes on Implementing DSP Algorithms

Some of the algorithms that were introduced in this chapter can be easily ported to low-power microcontrollers or FPGAs for real-time implementations. In order to implement such algorithms in digital systems the following methodology is suggested:

1) A suitable algorithm targeting low-power applications must be chosen. This choice can be affected by many factors including the digital processor (i.e., an FPGA or a microcontroller) that is going to be used, the available clock frequencies, the power budget, etc… Not all algorithms can be implemented in all signal processing systems.

2) The algorithm must be implemented and tested using floating-point calculations (for example in MATLAB) on signal obtained from signal banks. This allows the designer to verify the correctness of the algorithms rather than the implementation. Also, testing algorithms in environments like MATLAB gives the designer some insight into the inner-working of the algorithm. At this stage, the designer must make sure that the chosen algorithm is capable of processing the signal as desired on many different signals of the same family (for example on many different ECG signals from different test subjects).

3) Based on the architecture of the processor or the limitations of the FPGA that is going to be used, the designer must now simulate the algorithm with limited precision i.e., in fixed-point. This allows the designer to investigate the effects of limited precision on the algorithm. The effects of fixed-point calculations can show themselves in many different ways including the frequency response of filters or the quality of the signal after processing. This step can also be done in MATLAB.

4) After making sure that the algorithm work perfectly in fixed-point simulations, one can obtain real-life signals from the system (that is going to be designed) and test them using the fixed-point algorithms that were developed. For example, if ECG signals are to be compressed in the target systems, the designer can obtain real ECG signals using the same A/D that is going to be used later and test the compression algorithm using fixed-point calculations in MATLAB.

5) After the mentioned tests, the designer must implemented the algorithm in real-time in the target system. The way this step is done can vary from one system to another. Basically, the engineer must make sure that the delays caused by processing the signals/samples are lower than the required system response times. In the case of microcontroller, DMAs and interrupts can help a lot with reducing the delays of signal processing and in the case of FPGAs, pipelined architectures can be used.

In [83] authors present an implementation of a spike detection algorithm based on the absolute value on the hardware that has been developed in this project. This paper is an example of the methodology that is mentioned above.

3.8 Conclusion

In this chapter, a short review on one class of signal processing techniques for neural signals was presented. This class of neural signal processing is about detecting action potentials and associating them to neurons when multiple neurons are in close proximity of a recording electrode.

Other types of signal processing can also be carried on recorded neural signals, most notably action potential compression. Action potential compression allows the neural headstage or the implantable device to postpone the spike sorting to a later time in a high-speed computer while keeping the spike waveform. This results in relatively less complexity in the headstage firmware. However, even in the case of action potential compression, the need for spike detection exists.

In the next chapter, we will present the details of the headstage design.