Imperial College London
Department of Electrical and Electronic Engineering

Resource Efficient On-Node Spike Sorting

Deren Y. Barsakcioglu

December 11, 2015

Supervised by Dr. Timothy G. Constandinou

Submitted in part fulfilment of the requirements for the degree of
Doctor of Philosophy in Electrical and Electronic Engineering of Imperial College London
and the Diploma of Imperial College London

Declaration

I herewith certify that all material in this dissertation which is not my own work has been properly acknowledged.

Deren Y. Barsakcioglu

Copyright

The copyright of this thesis rests with the author and is made available under a Creative Commons Attribution Non-Commercial No Derivatives licence. Researchers are free to copy, distribute or transmit the thesis on the condition that they attribute it, that they do not use it for commercial purposes and that they do not alter, transform or build upon it. For any reuse or redistribution, researchers must make clear to others the licence terms of this work

Abstract

Current implantable brain-machine interfaces are recording multi-neuron activity by utilising multi-channel, multi-electrode micro-electrodes. With the rapid increase in recording capability has come more stringent constraints on implantable system power consumption and size. This is even more so with the increasing demand for wireless systems to increase the number of channels being monitored whilst overcoming the communication bottleneck (in transmitting raw data) via transcutaneous bio-telemetries. For systems observing unit activity, real-time spike sorting within an implantable device offers a unique solution to this problem.

However, achieving such data compression prior to transmission via an on-node spike sorting system has several challenges. The inherent complexity of the spike sorting problem arising from various factors (such as signal variability, local field potentials, background and multi-unit activity) have required computationally intensive algorithms (e.g. PCA, wavelet transform, superparamagnetic clustering). Hence spike sorting systems have traditionally been implemented off-line, usually run on work-stations. Owing to their complexity and not-so-well scalability, these algorithms cannot be simply transformed into a resource efficient hardware. On the contrary, although there have been several attempts in implantable hardware, an implementation to match comparable accuracy to off-line within the required power and area requirements for future BMIs have yet to be proposed.

Within this context, this research aims to fill in the gaps in the design towards a resource efficient implantable real-time spike sorter which achieves performance comparable to off-line methods. The research covered in this thesis target:

- Identifying and quantifying the trade-offs on subsequent signal processing performance and hardware resource utilisation of the parameters associated with analogue-front-end. Following the development of a behavioural model of the analogue-front-end and an optimisation tool, the sensitivity of the spike sorting accuracy to different front-end parameters are quantified.

- Identifying and quantifying the trade-offs associated with a two-stage hybrid solution to realising real-time on-node spike sorting. Initial part of the work focuses from the perspective of template matching only, while the second part of the work considers these parameters from the point of whole system including detection, sorting, and off-line training (template building). A set of minimum requirements are established which ensure robust, accurate and resource efficient operation.

- Developing new feature extraction and spike sorting algorithms towards highly scalable

systems. Based on waveform dynamics of the observed action potentials, a derivative based feature extraction and a spike sorting algorithm are proposed. These are compared with most commonly used methods of spike sorting under varying noise levels using realistic datasets to confirm their merits. The latter is implemented and demonstrated in real-time through an MCU based platform.

Dedicated to my great grandmother...

Acknowledgement

"...a ship [...] makes it through the journey not only because it can withstand the waves, but also because it is guided carefully by the captain..." were the exact words I wrote in acknowledgements of my masters thesis. Four years later, I complete a journey yet again. I am extremely grateful to Dr. Timothy Constandinou for his guidence, patience and support over all these years. He has been an inspirational role model for me.

I would also like to thank Dr. Amir Eftekhar for his support, encouragement and valued contributions. I would also like to thank Dr. Sivylla Paraskevopoulou, Dr. Joaquin Navajas and Dr. Yan Liu with whom I had the privilege to collaborate during my PhD. Moreover, I would like to thank Ms. Pooja Bunjun and Ms. Helen Root for their contributions.

A special thanks to my dear colleagues and friends in Neural Interfaces Group — Onur, Lieuwe, Song, Ian, Yan, Satoshi, Sivylla, Nicoletta, Francesca, Timo and Adrien — with whom I enjoyed very insightful discussions and shared wonderful memories which I will never forget. I would also like to thank Ermis, Yuanqi, Tatiana, Mohammedreza, Mohamed, Melina, Claudio, Ben, Pau, Andrea and all other people in Centre for Bio-inspired technology who are the friendliest and most supportive group of people one can ask for! In particular thanks to Peter Pesl and Dr. Pantelis Georgiou with whom I share the passion for Arsenal F.C. Our Champions League moments will forever stay with me.

Most importantly, I would like to express my gratitude for my dear family. They have always supported and encouraged me throughout my whole life. It has been their love and support that has kept me going for the last 26 years, even when the seas were rough. I will never be able to repay them...

Last but not the least, I would like to thank Ms. Meryem Ozgencil for making life more beautiful each and every day...

Contents

1. Introduction **18**

1.1. Motivation 18

1.2. Research Objectives 20

1.3. Outline of Thesis 21

2. Background and State-of-the-Art **23**

2.1. Brain-Machine Interfaces 23

2.1.1. Levels of Interfacing 24

2.1.2. Nature of Neurons and Action Potentials 25

2.1.3. Cortical Implants and Neural Recordings 28

2.2. Spike Sorting 30

2.2.1. To sort, or not to sort, that is the question 31

2.2.2. Challenges of Spike Sorting 32

2.2.3. State-of-the-Art Spike Sorting 33

2.3. Spike Detection 40

2.4. Summary 41

3. Optimisation of Analogue Front-End for On-node Spike Sorting **43**

3.1. Front-End Neural Interface Architectures 44

3.2. Front-End Modelling and Behavioural Optimisation 46

3.2.1. Front-end Behavioural Model 46

3.2.2. Testing and Evaluation 49

3.2.3. Test Data 51

3.3. Behavioural Model Realisation 52

3.4. Behavioural Model Validation 53

3.4.1. Circuit Implementation 54

3.4.2. Comparison between integrated circuit measurements and behavioural results 55

3.5. Behavioural Model Demonstration 57

3.5.1. State-of-the-art Front-Ends 57

3.5.2. Spike Processing Accuracy and Hardware Requirements 62

3.6. Parametric Design Optimisation 65

3.6.1. Methodology 65

3.6.2. Design Optimisation . . . 66
3.7. Conclusion . . . 72

4. Minimum Requirements for Template-based Real-time On-Node Spike Sorting 74
4.1. Template Matching . . . 75
4.1.1. Methods . . . 77
4.1.2. Results and Discussion . . . 83
4.2. Two-Stage Online Spike Sorting System . . . 99
4.2.1. Methods . . . 99
4.2.2. Results . . . 105
4.2.3. Discussion . . . 116
4.3. Conclusion . . . 119

5. Hardware Efficient Real-time Spike Sorting using Derivative Features 121
5.1. Derivative-Based Spike Sorting . . . 122
5.1.1. Materials and Methods . . . 123
5.1.2. Results and Discussion . . . 126
5.2. Improved Feature Space for Derivative Based Sorting Algorithm . . . 132
5.2.1. Methodology . . . 133
5.2.2. Extracellular Action Potential Dynamics and Waveform Features . . . 133
5.2.3. Results and Discussion . . . 135
5.3. Hardware-efficient spike sorting method based on waveform and derivative features 142
5.3.1. Methodology . . . 142
5.3.2. Proposed Spike Sorting Method . . . 143
5.3.3. Results and Discussion . . . 145
5.4. Real-Time Spike Sorting Implementation . . . 149
5.4.1. C Implementation . . . 150
5.4.2. Microcontroller Implementation . . . 153
5.4.3. Complete System Implementation . . . 155
5.5. Conclusion . . . 157

6. Conclusions 161
6.1. Original Contributions . . . 161
6.2. Future Directions . . . 163
6.2.1. On-node Spike Sorting . . . 163
6.2.2. Prosthetic applications: near to mid-term solutions . . . 164
6.3. Concluding Remarks . . . 165

A. Supplementary Figures for Chapter 4 167

B. Test Datasets **176**
B.1. Synthetic extracellular recordings: 1 176
B.2. Synthetic extracellular recordings: 2 176

C. Matlab Code for Automated Test Platform for Real-time Spike Sort Implementation **179**

D. C-Code for WDF implemented on KL43Z for tests involving RHD2216 **190**

E. Publications **199**

List of Figures

2.1. Various neural interface modalities with different levels of spatio-temporal resolution and invasiveness. . . . 24
2.2. Overview of neuron structure for different types of neurons. . . . 26
2.3. Action potential profile of a firing neuron. . . . 27
2.4. Breakdown of a typical extracellular recording detailing the contributions of single-unit activity (Zone I), multi-unit activity (Zone II), and local field potentials and background noise (Zone III). . . . 28
2.5. Variation in the observed extracellular action potential profile with spatial position. Shown is an example in the vicinity of a spiking pyramidal cell. . . . 30
2.6. WaveClus graphical user interface. . . . 35
2.7. Clustering performance comparison of PCA, Wavelets, and Graph-Laplacian features on Difficult1 dataset at 0.05 noise level. . . . 36
2.8. Exponential growth in the number of recorded neurons. . . . 38

3.1. General architecture of a neural interface. . . . 45
3.2. Basic system behavioural macromodel, including input parameters for each functional block. . . . 47
3.3. Illustration of 5 neurons being detected with: (a) a single positive threshold; (b) an absolute value threshold; and (c) a single threshold after NEO processing. . . . 49
3.4. Behavioural model system architecture showing different I/O nodes. . . . 52
3.5. Graphical user interface for the front-end behavioural model. . . . 53
3.6. Channel architecture of the front-end neural interface IC. . . . 54
3.7. Circuit implementation of the front-end neural interface. . . . 55
3.8. Microphotograph of the circuit implemented in a 0.18 μm CMOS technology. . . . 56
3.9. Comparison of neural front-end IC and the behavioural model frequency responses at various configurations. . . . 57
3.10. Distribution of error between neural front-end IC and the behavioural model frequency responses at various configurations. . . . 59
3.11. Comparison of simulated (behavioural) and measured (chip) response to a test stimulus (spike waveform). . . . 60
3.12. Effect of analogue front-end on spike waveform. . . . 62
3.13. Spike sorting performance plotted against hardware resource requirements for all front-ends. . . . 64

3.14. Spike detection performance plotted against hardware resource requirements for all front-end configurations. 65
3.15. Effect of SNR on spike sorting accuracy of the overall system. 67
3.16. Effect of high pass and low pass filtering on spike sorting accuracy. 68
3.17. The effect of a 4^{th} order high-pass Butterworth filter on the original spike shape. 68
3.18. Effect of data conversion on spike sorting accuracy. 69

4.1. A typical hardware-based spike sorting system encompassing signal amplification and conditioning, spike sorting (including feature extraction) and any subsequent processing (either offline for analysis or online for rehabilitation). 76
4.2. Process of creating a template based on mean and/or median of a known cluster. 78
4.3. Distribution plot for one particular template. 80
4.4. Accuracy for the different template matching quantifiers through all interpolation frequencies and methods. 84
4.5. Effect of interpolation frequency on template matching accuracy for SED and SAD quantifiers. 85
4.6. Histograms of spike sorting accuracy differences between Test2 and Test1 results. 86
4.7. Spike sorting accuracy vs. execution time of: (a) template matching methods, and (b) interpolation methods. 86
4.8. Template Matching (SAD) results for various window sizes. 88
4.9. Template Window performance for four datasets at 24kHz sampling rate and 0.1 noise level. 89
4.10. Change in accuracy for each data for different resolutions and different template matching methods. 90
4.11. The mean and max/min (errorbars) across all template matching quantifiers for a non-causal implementation of low pass filters (Butterworth, Elliptic and Bessel), for each data set 1-6 (a-b). 91
4.12. The mean across all template matching quantifiers for a non-causal implementation of high pass filters (Butterworth, Elliptic and Bessel), for each data set 1-6 (a-b). 92
4.13. Example of the effect of misalignment on the separation of clusters. 92
4.14. Sensitivity of template matching quantifies to varying SNR, averaged over all datasets (with min/max error bars indicating variation over frequency of interpolation for alignment). 94
4.15. Noise performance of the different interpolation methods (with min/max error bars indicating variation over alignment interpolation frequency) using SED as the template matching method. 95
4.16. Effect of varying interpolation frequency (for alignment) on SNR performance. . 96
4.17. Effect of varying resolution on SNR performance for different template matching quantifiers. 96

4.18. Effect of LPF cut-off frequency and filter type on SNR performance. 97
4.19. Effect of HPF cut-off frequency and filter type on SNR performance. 97
4.20. Monte Carlo simulation of the spread in accuracy for $N = 100$ randomly selected 10% and 20% training segments using the near optimal parameter set determined. 98
4.21. Example training/testing results for datasets 1-4 over all four SNR values using a 20% training set. 99
4.22. Extracellular recording creation and data processing. 101
4.23. Impact of filtering on spike detection. 106
4.24. The sensitivity of spike detection performance to sampling rate and resolution. . 107
4.25. The sensitivity of spike sorting and template building to sampling rate and signal resolution. 108
4.26. Illustration of two-stage hybrid strategy for real-time on-node spike sorting. . . . 110
4.27. The sensitivity of real-time template matching performance to sampling rate and resolution. 111
4.28. The sensitivity of template matching performance to misalignment and template window size. 113
4.29. Validation of established minimum requirements with real data. 115

5.1. Typical action potential waveforms with their first and second derivatives (FD and SD). 124
5.2. Test dataset 2 showing: (a) a 1 s segment at noise levels of 0.05 to 0.2 (from top to bottom), and (b) a two dimensional representation of the clustering space when using three principle components. 125
5.3. Comparison of the classification error between commonly used feature extraction methods (across all datasets and noise levels). 127
5.4. Observing the effect of dimensionality on classification error for the different feature extraction methods used in this work. 129
5.5. Spike classification utilising FSDE features. 130
5.6. Comparing classification error (averaged over all datasets and noise-levels) with computational complexity for the different feature extraction methods. 132
5.7. General structure of generated action potential. 134
5.8. Normalized amplitude distributions of local min/max and inflection points of spike waveform and its derivatives (up to third order). (a) EAP waveforms (b) First Derivative (c) Second Derivative (d) Third Derivative. 137
5.9. Spike sorting performance at varying background and white noise levels of different feature subsets (original waveforms and derivatives up to third order). (a) Easy1 dataset (b) Easy2 dataset. 139
5.10. Spike sorting performance at varying background and thermal noise levels of different feature subsets (original waveforms and derivatives up to third order). (a) Difficult1 dataset (b) Difficult2 dataset. 140

5.11. Performance comparison of the selected methods for proposed reduce feature space versus utilizing the whole. . . . 141
5.12. A high level description of WDF. . . . 144
5.13. Illustration of the threshold choice ($T_{i,j}$) based on distributions of features i and j, and the mechanics of comparison based classification. . . . 145
5.14. Comparison of the spike sorting performance for different methods across varying background and AWGN noise levels. . . . 146
5.15. Comparison of the spike sorting performance for different methods across varying background and AWGN noise levels. . . . 147
5.16. Assessment of classification error (averaged across all datasets and noise levels) from complexity and dimensionality perspective. . . . 149
5.17. Spike sorting accuracy difference between Matlab and C implementations at various background activity and AWGN levels of four datasets. . . . 151
5.18. Analysis of computational requirements for different sections of C-implementation. 152
5.19. Test platform for verifying real-time spike sorting implemented on Freescale FRDM-KL25Z board. . . . 153
5.20. Spike sorting accuracy difference between Matlab and MCU implementations at various background activity and AWGN levels of four datasets. . . . 154
5.21. Power consumption analysis of real-time spike sorting implementation on FRDM KL25Z Board. . . . 155
5.22. Complete system implementation comprising of analogue front-end (Intan RHD2216), real-time on-node spike sorting (FRDM KL43Z), and Graphical User Interface (on PC). . . . 156
5.23. Graphical User Interface description detailing functionality. . . . 158
5.24. Sample screenshot from GUI in operation. . . . 159

A.1. Illustration of segments of simulated data at various signal-to-noise ratios (SNR). 167
A.2. Spike detection performance as a function of threshold value, defined as in Equation 1. . . . 168
A.3. Impact of filtering on spike detection accuracy. . . . 169
A.4. Extended example of Figure 4.25C. Illustrated is the incorrect merging of separate clusters due to limitations in sampling rate or signal resolution. . . . 170
A.5. Sensitivity of real-time template matching performance (with different metrics) to sampling rate and signal resolution. . . . 171
A.6. Sensitivity of real-time template matching performance (with different metrics) to sampling rate and signal resolution. . . . 172
A.7. Template matching performance against relative complexity at different sampling rate and resolution specifications. . . . 173
A.8. Sensitivity of template matching performance to misalignment and window size. 174

B.1. Mean spike profiles for the 6 datasets with corresponding Bray-Curtis similarity measures applied between each neuron cluster within each dataset. 177
B.2. Mean spike profiles for the 5 datasets with corresponding Bray-Curtis similarity measures applied between each neuron cluster within each dataset. 178

List of Tables

2.1. Specifications of different physiological signals used by BCIs. 26
2.2. Comparison Between Similarity Measurements used in template matching. 37
2.3. Summary of spike detection methods along with varying pre-processing and noise estimations techniques. 40

3.1. Behavioural model input design parameters 52
3.2. Technical specifications of the front-end neural interface IC 55
3.3. Spike Detection Performance difference between the modelled and measured output 58
3.4. Spike Sorting Performance difference between the modelled and measured output 58
3.5. Review of state-of-the-art neural interfaces showing key design parameters 61
3.6. Spike Detection Performance of state-of-the-art front-end circuits 63
3.7. Spike Sorting Performance of state-of-the-art front-end circuits 63
3.8. Review of state-of-the-art neural interfaces comparing key design parameters and resource utilisation 71

4.1. Summary of parameters varied to assess the performance of template matching for spike sorting. 82
4.2. Effect of HPF phase distortion on alignment for three different methods across all datasets. 93
4.3. Computational complexity in number of operations. 112
4.4. Power consumption estimations for different modes of data transmission 116

5.1. Comparison of the classification error for different combinations of derivative features. 126
5.2. Computational complexity of each feature extraction method. 129
5.3. Variation in amplitude and temporal domain of the critical features of the EAP waveform. 136

A.1. Median number of misclassified spikes during spike sorting for signals with different sampling rates, signal resolutions, and SNRs. 175

List of Abbreviations

SCI – Spinal Cord Injury
BMI – Brain Machine Interface
SoC – System-on-a-Chip
CNS – Central Nervous System
LNA – Low Noise Amplifier
ADC – Analogue-to-Digital Converter
SPC – Superparamagnetic Clustering
DHT – Discrete Haar Transformation
IID – Independent and Identically Distributed
DBVI – Davies-Bouldin Validation Indices
SSDA – Sequential Similarity Detection Algorithms
OSAD – Optimized Sum of Difference
PCDC – Pairwise Cluster Dividing and Combining
NN – Neural Network
MTF – Matched Template Filtering
ZCF – Zero-Crossing-Features
DWT – Discrete Wavelet Transform
PCA – Principle Component Analysis
ICA – Independent Component Analysis
EM – Expectation-Maximization
AFE – Analogue-Front-End
SAD – Sum of Absolute Difference
SSD – Sum of Squared Difference
WED – Weighted Euclidean Distance
SUA – Single Unit Activity
EAP – Extracellular Action Potential
LFP – Local Field Potentials
EEG – Electroencephalography
ECoG – Electrocorticography
EMG – Electromyographic
EOG – Electrooculagoraphic
NN – Neural Network
DD – Discrete Derivatives

1. Introduction

Interfacing the nervous system, studying and understanding its secrets is key in harnessing its power to provide solutions for neural diseases, disability and even the design of future computing devices. Chronic implants which monitor the individual activity of large neuronal populations have been pivotal in deciphering some of these secrets. As the technological advances continue to increase the number of neurons observed, the limits for safe thermal dissipation (without damaging neuronal tissue) places fundamental limitations on the amount of data transmission. In this regard all discussions, investigations, and algorithms developed in this thesis centres around one main theme: improving information transfer rate in a chronic implant.

1.1. Motivation

Every day thousands of people around the world are paralysed or diagnosed with neuro-degenerative diseases [1–3] which not only result in negative emotional and social consequences for patients and their families, but also result in a large economic impact. For example, the lifelong costs for the care (together with initial treatment costs) of a person living with SCI can mount up to $4 million (over his/her lifetime) in the U.S. [4], which places huge economical burdens on both the families and the state. In the U.K., the annual cost to the nation of caring only for SCI patients is estimated to be around £500 million [2]. On the other hand, there are around 650,000 people suffering from Alzheimer's or Parkinson's disease in the U.K. alone, with a projection of 157% increase in numbers over the next 38 years [5]. In other words, SCI and neuro-degenerative diseases combined, is a problem on a societal scale with psychological, societal and economical consequences.

Hence, understanding how the interactions of billions of neurons in the brain produce our thoughts, perceptions, and actions is one of the greatest challenges of 21st century science. The ability to interface to these neurons using electronics is presenting new opportunities for neural rehabilitation with prosthetic devices. Commonly referred as neuroprosthetics, such devices aim to restore the lost sensory and/or motor abilities, and has the potential to significantly impact the quality of life of millions of people around the world. For example, sensory cochlear implants, are already impacting the quality of life of around 300,000 individuals with profound deafness [6]. More recently, owing to the developments in robotics, neuroscience and microelectronics, emerging motor prosthetics have already demonstrated that mobility, lost due to spinal cord injury or neural diseases, could be restored [7].

In general, such devices have an underlying neural interface which either interacts with the sensory or motor pathways. Sensory prosthesis involve translating an external sensory stimuli

into appropriate stimulation patterns. For example, cochlear implants utilise electrical neural stimulation to modulate the firing rate of neurons and communicate sound to the brain, via the sensory pathway. Motor prosthetics, on the other hand, use recording electronics to monitor the activity of neural circuits, and either control assistive devices or stimulate existing motor pathways. In addition to sensory and motor function restoration, there exists efforts in restoring lost cognitive functionality such as hippocampal prosthetics which aim to restore ability to form new long-term memories (due to damage to hippocampus) [8].

Moreover, establishing a direct communication pathway with such BMIs offer significant contributions to neuroscience research as well. With these tools, scientists can monitor the brain-functionality down to individual neuron level in order to study underlying network dynamics, build experimental models (e.g. cellular models), investigate contributions of neural populations to control parameters (e.g. tuning and selectivity), and establish connectivity maps of neuronal ensembles. These are only few of the numerous benefits of such BMIs.

Neural interfaces monitoring the brain vary in their level of invasiveness. For example, (1) electroencephalography (EEG) monitor "global" activity non-invasively at the surface of the skull, (2) electrocortiography (ECoG) monitors more localized (although still network) activity semi-invasively on the surface of the brain, and (3) penetrating microelectrodes monitor local activity of individual cells (neurons) invasively. It is generally accepted that neural activity at the highest resolution (i.e. as in (3)) is most useful in deciphering signals of high dimensionality.

Therefore recording of large numbers of neurons is thus an extremely important task, but one that requires overcoming several technical challenges. Recent years have seen the development of micro-fabricated neural probes such as the Utah and Michigan arrays, now commonplace in experimental labs [9, 10]. These neural recordings obtained from micro-electrodes, not only contain the activity of individual cells called extracellular action potentials (EAP), but also include other signals sources such as local field potentials (LFP), and, multi-unit and background activity. As a result, for any portable or implantable device such probes require miniature electronics locally, commonly referred as Analogue-Front-End (AFE) to amplify the weak neural signals, filter out noise and out-of-band interference, and digitise for transmission. With recent advances in modern semiconductor technology, this is now possible and has sparked significant research activity in the community, particularly in this last decade [11–25].

These advances in recording technology, coupled with those in micro-fabrication techniques, have significantly increased the number of recording sites (hence the number of monitored neurons) over the years. In fact, [26] estimates that the number of simultaneously recorded neurons double every 7.4 years. This increase in the amount of data to be transmitted, increases the bandwidth and power requirements for the hardware. Wireless transcutaneous telemetries that are crucial for chronic implants, however, have fundamental limitations as to the amount of data that can be transmitted within safe limits for thermal dissipation.

In order to overcome this bandwidth and power consumption bottleneck, reduction of the data prior to transmission is necessary, and this can be achieved by on-node spike sorting. Typically done off-node post-transmission, spike sorting is the identification of different neurons

around the recording electrode tip. This process involves establishing the number of and the attributes of available neurons, and subsequently classifying the incoming spikes. However, this data reduction via on-node spike sorting has several challenges. The major challenge posed for the implantable realisation of spike sorting systems is due to the inherent complexity of the spike sorting problem, which is discussed in Section 2.2.2. In order to achieve reliable spike sorting, traditionally offline systems employ computationally demanding algorithms such as Principal Component Analysis (PCA), wavelet transform, and, clustering algorithms such as K-means. As a result of their complexity and not-so-well scalability with increasing number of recording channels, these algorithms cannot be implemented in a resource efficient way in hardware. On the contrary, although there have been several attempts in implantable hardware, an implementation to match comparable accuracy to off-line within the required power and area requirements for future BMIs have yet to be proposed.

1.2. Research Objectives

Within this context, this research aims to fill in the gaps in the design towards resource efficient implantable real-time on-node spike sorting. There exists two different specifications for real-time operation depending on the BMI application. For BMIs that incorporate closed-loop feedback stimulation (e.g. to study plasticity), the latency required for signal processing is in the order of sub-milliseconds. On the other hand, the signal processing latency required for real-time operation is in the order of hundreds of milliseconds for open-loop control applications which track populations of neurons over such time scale. The work presented in this thesis is applicable to both applications. In addition to being real-time and hardware resource efficient, achieving comparable spike sorting performance to the established off-line methods (e.g. PCA) is also crucial.

In order to achieve such an optimum realisation of the on-node spike sorting system, one of the key objectives of this research was to identify and analyse all parameters along the signal processing chain that affect spike sorting process. One of the key stages of this processing is the analogue-front-end which performs the initial signal processing (amplification, filtering, etc.) and digitises the recording. Not only the selection of specifications have direct effects on the performance of subsequent processing stages, but also in overall power consumption and area. When I started my research, there had not been a comprehensive study in analysing these parameters from the perspective of spike sorting performance and resource utilisation, and a survey of literature revealed a wide range of specifications. Therefore, one of the initial aims of the conducted research was to identify and investigate some of the design trade-offs associated with analogue-front-end parameters, and demonstrate these for various spike sorting algorithms.

The second research objective dealt with investigating the feasibility of a two-stage hybrid strategy employing both offline and online processing for robust and hardware efficient real-time sorting implementation. The offline processing involves clustering the detected spikes in order to establish the number of existing neurons (in the recording) and their associated waveforms, i.e.

templates. On the other hand, the online processing is responsible for real-time spike detection and classification.

This solution targets to mitigate computationally demanding parts of spike sorting (i.e. clustering) to an external device and utilise a simple template matching algorithm to achieve real-time classification. Although template matching is a commonly used off-line classification method, to the best of my knowledge, there is only one previous study (pre-dating this work) that proposed real-time on-node classification with downloaded templates, and the reported resource utilisation is far from desired. However, to date, there haven't been any studies investigating to which extent the specifications of such systems can be reduced (hence efficient resource utilisation) without sacrificing performance in order to facilitate real-time on-node classification. In this regard, key research question involved identifying and quantifying parameters that have impact (on performance and resource) on various parts of the system.

The last research objective aimed developing novel and efficient feature extraction and spike classification algorithms that outperforms available methods (especially those established), and yet still are extremely efficient to implement in any hardware platform. This motivation relies on the fact that together with the reduction of features (hence the computational resources and area spend), the potential to monitor even larger groups of neurons increases. However, the most important research question was: which features to use? During the last stage of the research novel feature extraction and classification methods are developed drawing inspiration from the dynamics of extracellular activity.

1.3. Outline of Thesis

The structure of the thesis is as follows:

- Chapter 1 has introduced a high-level description of the stated problem, briefly identifying the motivation and main objectives of the research.
- Chapter 2 provides background information on the subject. Emerging BMIs and neural interfaces are discussed. In addition, some physiological background regarding the nature of neural signals are given. Spike sorting is defined and its importance within the context of BMIs and prosthetics is highlighted. Moreover, challenges associated with spike sorting are discussed in order to emphasize why this is not such a trivial signal processing task. The discussion follows on with a generic description of the spike sorting systems, and then reviews both off-node and on-node implementations, highlighting state-of-the-art. During the review process, the need for shifting from off-line to on-line sorting are once more emphasized.
- Chapter 3 aims to identify and quantify the front-end parameters in terms of their effects on spike sorting performance and hardware resource. After introducing front-end neural interface architectures, a behavioural model of the analogue-front-end and an optimisation tool, which provides the designer a platform to investigate effects of parameters in a fast

way, is developed. Using the proposed model, the sensitivity of spike sorting accuracy to different front-end parameters are quantified.

- Chapter 4 investigates the effects of spike sorting relevant parameters (focusing on template matching) for accuracy and complexity trade-off, and aims to achieve a near optimum set of parameters. The hypotheses behind each method, along with methodology, results and findings are detailed and analysed in depth. A feasibility study of a two-stage hybrid real-time spike sorting strategy — which combines off-line and on-line processing — is conducted. The results of these reveal a set of minimum requirements to achieve robust, real-time, accurate and hardware efficient solution.

- Chapter 5 proposes a new feature extraction and a new spike sorting method based on derivative features. First, hypotheses behind the use of derivatives is explained, and it is compared and quantified with established spike sorting methods. Then, a spike sorting method incorporating elements from derivative based spike sorting and raw waveform is proposed and quantified, followed by the real-time implementation and testing of the algorithms.

- Chapter 6 concludes this thesis highlighting original contributions and discusses possible future directions.

2. Background and State-of-the-Art

2.1. Brain-Machine Interfaces

Interfacing a machine with the human brain, reading the mind as well as controlling with it, have inspired countless science fiction novels and films over the past. Started in 1790s with Luigi Galvani by twitching dead frogs' legs with electric sparks, and following decades of research paralleled by technological advances in both the micro-electrode technology and micro-electronics, the dawn of "Bionic Era" is finally upon us.

Establishing a direct communication pathway between external devices and the brain, brain-machine interfaces comprise of assistive devices and neural prosthetics. BMIs concerned with assistive technologies to patients mainly target establishing basic communication and a means of acting on the environment [27]. These technologies particularly target locked-in patients who suffer from total paralysis following brainstem stroke or degenerative diseases such as amytotrophic lateral sclerosis. Examples include spelling and speaking aides such as P300 BCI system. On the other hand, neural prosthetics aim to restore the lost sensory, motor and cognitive abilities. In fact, over the past decades such devices have already changed thousands of lives. Cochlear implants that have been implanted to around 40000 patients to date, deep brain stimulators (DBS), numerous prosthetic limbs (such as Deka Arm [28]), exoskeletons (such as Hybrid Assistive Limb (HAL) [29]), and, emerging visual [30] and cognitive prosthetic research to replace lost brain functionality [8, 31] are some examples which have helped or have the potential to help millions of patients.

Moreover, BMIs can provide powerful research tools for neuroscience as well. Influences on experimental models in fields such as cellular, computational and cognitive neuroscience; investigative use of such systems to study contributions of different neuron populations to motor parameters; in-vitro hybrid biological and artificial networks to study neural circuits' cellular properties are a few examples of what BMIs can provide for the future of basic brain research [32].

From an operational point of view, BMIs can be divided into two categories [33]. First category comprises of devices that stimulate brain via artificially generated electrical signals to relay sensory information or to mimic a neurological function. In such systems, external stimuli, such as sound (auditory prostheses) or image (visual prosthesis) is processed in real-time, and relevant regions of brain are stimulated with appropriate stimulation patterns. In addition, methods of direct brain stimulation to lessen pain, to control motor disorders (e.g. Parkinson's), and to reduce epileptic activity are also in this category. The second group of

BMIs performs real-time sampling and processing of brain activity in order to control artificial devices or perform stimulations. In these devices, neural signals are recorded from brain (regions relevant to the application), processed and analysed, and then relayed (usually wireless) to an external device which could be an actuator to control a prosthetic limb or to perform stimulation (if the limb is still intact). Often it is desirable to make these systems closed loop by providing feedback, either visual or tactile/proprioceptive, in order for the prosthetic to behave and feel more naturally [34].

2.1.1. Levels of Interfacing

Neural interfaces enable the communication between the nervous system and the machine by tapping into sensory and motor pathways. There exists multiple levels of invasiveness employed by BMIs depending on the needs of the application (Figure 2.1). Some of the main factors that affect the choice of interface modality include risk-involved, ease of conducting experiments and recruiting participants, amount of information required for particular application (e.g. controlling a computer cursor vs a prosthetic arm) [35, 36].

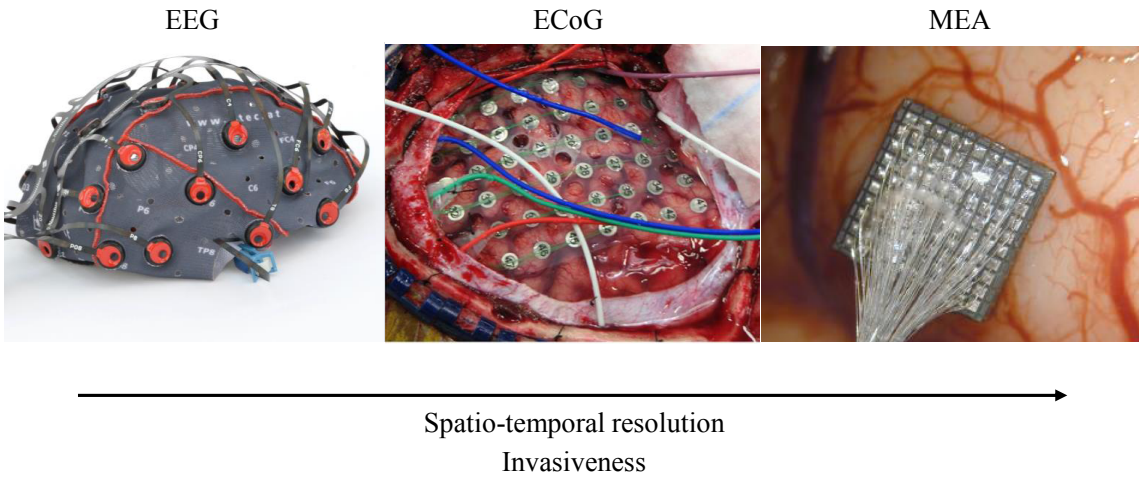


Figure 2.1.: Various neural interface modalities with different levels of spatio-temporal resolution and invasiveness. Adapted from [37–39].

Traditionally, BMIs have used specific components of EEG which is the least invasive (minimal risks to the user) and has relative low cost [35,40]. Acquired by a set of surface electrodes placed on the scalp, EEG have provided numerous successful BMIs which include spelling devices [41, 42], controlling computer cursors [43] and driving a wheelchair [44]. However, EEG is a limited method in terms of spatial and temporal resolution (i.e. information transfer rate) [45] due to overlapping electrical activity of different cortical areas, low-pass filtering and attenuation of signals through brain tissue, bone and layers of the scalp [34, 40]. In addition, EEG signals are prone to mechanical artefacts, electromyographic (EMG), and electrooculagoraphic (EOG) signals. These render EEG suboptimal for investigating underlying brain processes and their short-lived spatio-temporal dynamics [46], and control of assistive devices that require higher

data dimensionality such as movement restoration through prosthetics with multiple degrees of freedom [35, 40].

In an attempt to increase recording resolution, more invasive methods such as electrocorticography (ECoG) have been introduced [34]. This method involves surface electrodes placed on the brain to record relatively localised (compared to EEG) neuronal ensemble activity. Since the electrodes record activity directly from cerebral cortex surface, ECoG bypasses the skull and scalp tissue which play a major role in attenuating and distorting the signals, while artefacts such as EOG and EMG are minimised [46]. Therefore, both the temporal and spatial resolution of the acquired signal can be significantly improved [40, 46, 47]. As a result, there have been successful demonstrations of ECoG in analysis of brain functionality revealing information about task-related activites (such as motor execution and planning) [46], as well as BMI applications (e.g. cursor control) [35]. Due to increased monitoring resolution, it is argued that these ECoG-based BCIs have better accuracy and shorter training times than their EEG counterparts [34, 40].

However, the highest spatial and temporal resolution is obtained by employing micro-electrodes implanted into brain. BMIs utilizing micro-electrode arrays have been gaining huge interest in the research community due to advances in the micro-electrode technology [45]. In addition to allowing to explore neural activity at deeper locations in the brain, intra-cortical recordings provides the highest spatial and temporal resolution (among all recording modalities) revealing the spiking activity (at single neuron resolution) of the neural populations within a small volume of tissue [40, 47]. In addition to single neural events, these electrodes also capture local field potentials (LFPs) which represent the summed electrical activity of underlying neural network.

Such low-level direct interfacing is not only invaluable to understanding underlying dynamics of the brain (from neuroscience perspective), but also in providing a host of precise assistive devices with more capabilities (e.g. prosthetics to restore movements with multiple degrees of freedom) [40,48]. In fact, many research groups around the world have demonstrated controlling assistive devices in both animal and human subjects [49, 50]. Despite the shortcomings of this method due to bio-compatibility, scar-tissue formation (which degrades signals recorded overtime) and loss/death of neurons [32, 34], cases for stable recordings for months and even years in few cases have been demonstrated [34]. The work presented here is within the context of such BMIs that monitor neural activity via intra-cortical micro-electrodes.

2.1.2. Nature of Neurons and Action Potentials

With their trillions of connections, 100 billion neurons form one the most power-efficient and elegant computing machines, the brain, which gives us ability to think, to act, and to regulate our bodily functions and mechanisms. The mysteries and wonders of the brain have intrigued many scientists over hundreds of years. Starting with then 49 year-old Czech anatomist Jan Purkinje over 174 years ago, countless research time has been spent to identify and unlock the secrets of this enormous communication and data processing network.

Neurons, nerve cells, are specialised cells that form the nervous system which coordinates

Table 2.1.: Specifications of different physiological signals used by BCIs [35, 40, 48, 51–54].

Signal	**Amplitude** (peak to peak)	**Bandwidth**	**Temporal Resolution**	**Spatial Resolution**	**Invasiveness**
EEG	1 - 100 μV	$\leq$50 Hz	$\sim$50 ms	$>$cm^2	Surface (Scalp)
ECoG	1 - 500 μV	$\leq$200 Hz	$\sim$5 ms	$\sim$ cm^2	Surface (Brain)
	1 μV - 2 mV*	$\leq$200 Hz	$\sim$5 ms	$\sim$mm^2	Surface (Brain)
LFP	500 μV - 5 mV	$\leq$300 Hz	$\sim$3 ms	$\sim$mm^2	Intra-cortical
SUA	50 μV - 500 μV	250 Hz - 10 kHz	$\leq$0.2 ms	sub-mm^2	Intra-cortical

*Values given are for μECoG with smaller electrode pitch and surface area than conventional ECoG.

voluntary and involuntary actions, and are responsible for communicating this information to relevant parts of the organism. Although many types of neurons exist in many shapes (depending on their location and functionality) [55] , the typical structure of a nerve cell consists of soma, dendrites and axon. Soma is the main body of the neuron containing cell nucleus, while dendrites and axons are extrusions, former extending a few hundred micrometers and latter extending far more (Figure 2.2).

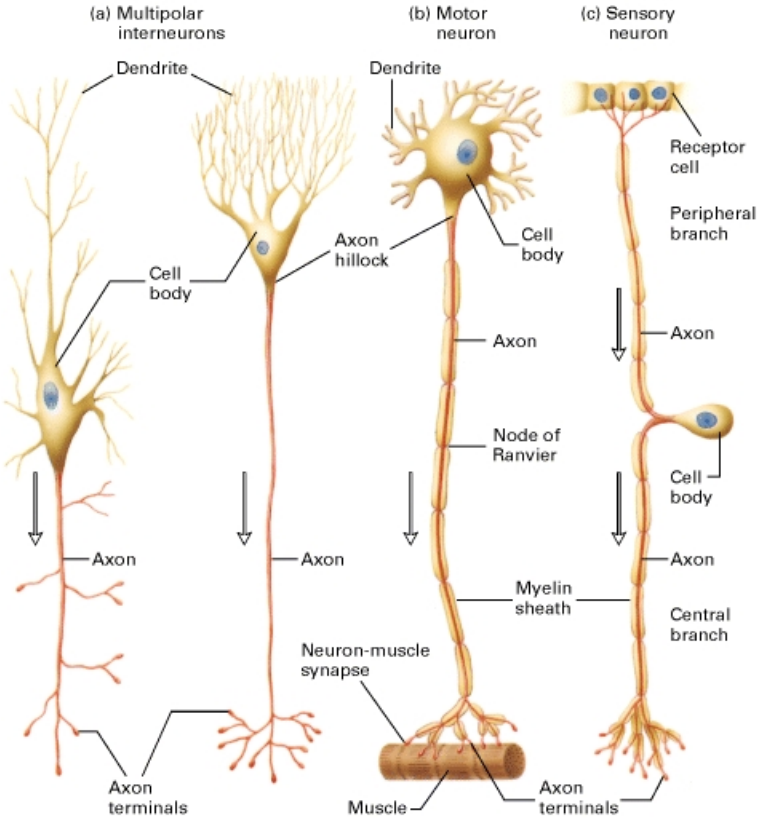


Figure 2.2.: Overview of neuron structure for different types of neurons. Extracted from [56].

As electrically excitable cells, neurons communicate with each other via electrical signals known as "Action Potentials" (Figure 2.3). Nerve cells are electrically polarized due to the

extracellular and intracellular molecule concentrations such as potassium, sodium, chloride, and calcium ions. The resting potential of the neurons is calculated using the Nernst equation, and the typical resting potential for a neuron (measured intracellularly) at body temperature is around -70mV [57].

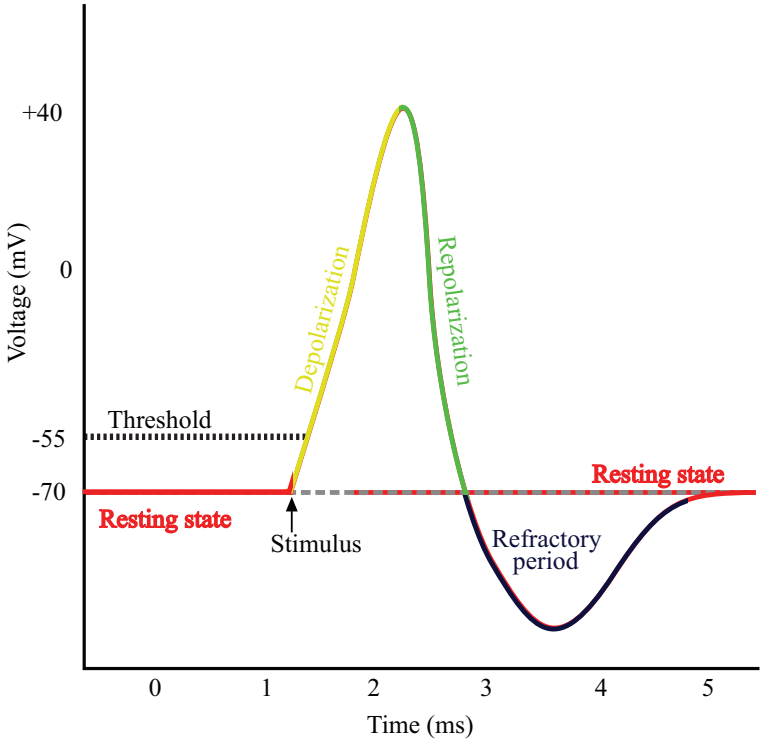


Figure 2.3.: Action potential profile of a firing neuron. Adapted from [58].

Whenever there is an excitatory input to the nerve cells, this creates a flow of Na^+ ions, which causes the membrane potential to increase. If this flow results in membrane potential exceeding a certain threshold limit, more Na^+ channels open. The flow of Na^+ ions further increases the membrane potential, and this period is known as depolarization. Increase in membrane potential continues until a peak value of about +55mV is reached[1].

Once this peak value is reached, the raised voltage also causes the K^+ channels to open while Na^+ channels close, and the membrane potential starts decreasing. The phase during which the membrane potential decreases after the peak value is called repolarisation. Since K^+ channels are slow to close during re-polarisation, an undershoot called hyperpolarisation occurs before the membrane potential is settled at resting potential. During this period, also known as refractory period, no further action potential can be initiated. It should be noted that described spike characteristics define the intra-cellular action potential. As far as neural signal acquisition is concerned, the recorded signals exhibit differences than the intra-cellular spike due to various factors (such as relative position of electrode and composition of ionic currents and different cells) [59], and are discussed in Section 2.1.3.

[1]This value is just an general average, and might change from neuron to neuron.

2.1.3. Cortical Implants and Neural Recordings

As previously mentioned, the major advantage of observing neural activity through cortical implants is the inherent high temporal and spatial resolution data obtained, which is extremely useful for BMI applications (whether for studying underlying dynamics of the neural network or being utilised by assistive devices) [40, 48].

Typically recorded through micro-electrode arrays, the signals observed in the extracellular space consist of three main components: (1) Single-unit Activity, (2) Multi-unit activity, and (3) Local field potentials (see Figure 2.4).

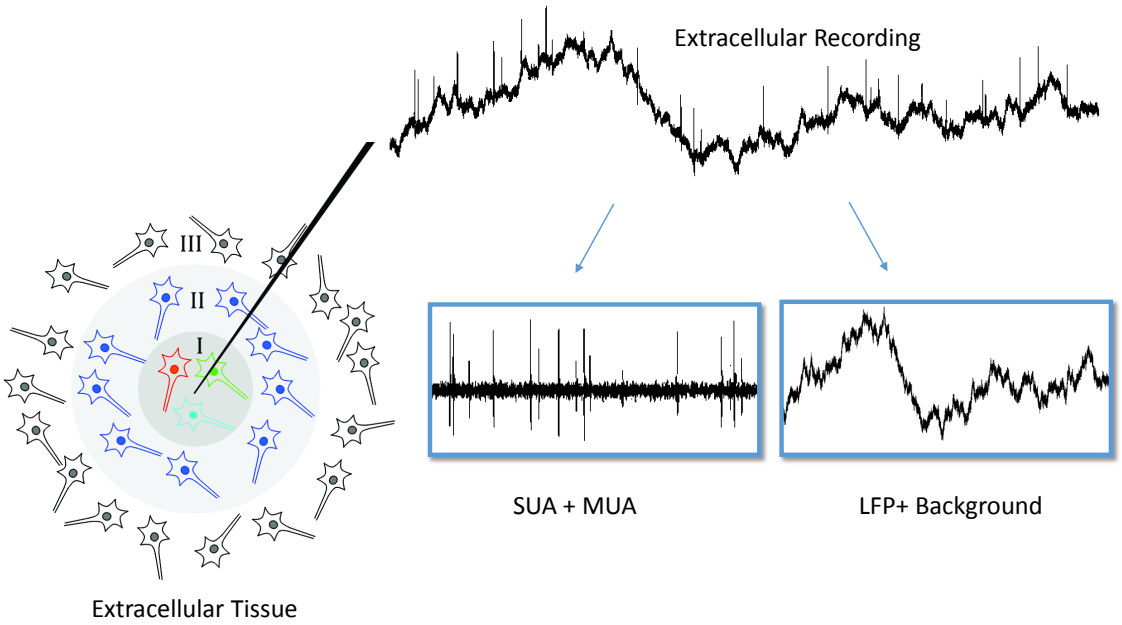


Figure 2.4.: Breakdown of a typical extracellular recording detailing the contributions of single-unit activity (Zone I), multi-unit activity (Zone II), and local field potentials and background noise (Zone III).

2.1.3.1. Single-unit Activity

The single-unit activity (SUA), also known as extracellular action potential, refers to the voltage gradients created in the extracellular space by the strong currents (Na^+) induced by the individual spiking activity of the neurons [47]. Captured within 50 μm — the maximum distance single units can be differentiated — of the electrode tip [60], the typical observed amplitudes of EAPs range from 60 μV to 500 μV depending on electrode position [60]. In fact, this spatial variation (see Figure 2.5) observed in EAPs can be described by a simple model (Eq. 2.1) treating equivalent current source of an individual neuron as an electric dipole moment ($\overrightarrow{p}$) [61], and factoring in the neuron-to-electrode distance ($\overrightarrow{r}$) and the cosine of the angle between the two ($\cos\theta$) [61].

$$V(\overrightarrow{r}) = \frac{\overrightarrow{p}\,\overrightarrow{r}}{4\pi\epsilon_0 r^3} = \frac{pr \cdot cos\theta}{4\pi\epsilon_0 r^3} \tag{2.1}$$

2.1.3.2. Multi-unit Activity and Background Activity

Multi-unit activity (MUA) refers to neurons that lie in the region between 50 μm and 140 μm from the electrode tip. These fire spikes with larger amplitudes than the background to be detected, but are too small to be sorted [47, 62]. Beyond 140 μm away from electrode tip, the spikes generated by the neurons are too small to be detected. Contribution of these distant neurons — called background activity — can be as high as 20 μV_{rms} [62], and is one of the systematic noise sources in neural recordings [60].

2.1.3.3. Local field potentials

In addition to aforementioned signals, the extracellular recordings also exhibit low-frequency oscillations, called local field potentials (LFPs), which reflect the dynamics of underlying neural network. Reaching as high as 5 mV_{pp}, these signals are the superimposition of the summed dendritic and synaptic currents within the surrounding volume of tissue [47]. LFPs occupy the frequency band up to 300 Hz [40].

2.1.3.4. Variability in extracellular action potentials

The observed shape of the EAP waveform is the function of the total ionic current exchange across the cell membrane and the relative position of the recording electrode [59]. As previously described, the spatial variation of the EAP waveform can be modelled with a dipole model which describe the amplitude scaling in relation to electrode and source distance [61].

Besides amplitude scaling, the position of the electrode relative to different sections of the neuronal structure play a major role in the expression of critical features in the EAP [47]. This is mainly the result of different current components that occur along the neuronal structure. Variations in current components are direct manifestations of differences in conduction densities, and (together with electrode position) play a major role in variability observed in main negative peak and re-polarisation phases [63].

For example, while capacitive phase peak — initial positive peak — is not observed at all when the electrode is closer to soma, the capacitive peak can reach to almost the same amplitude as the main negative peak when recorded near locations closer to apical trunk [59]. This is mainly due to the fact that near soma Na^+ current is the driving factor in membrane potential, and the capacitive current is not significant until Na^+ current is already active. Since Na^+ current is larger than the capacitive current, the EAP lacks the initial positive phase. On the other hand, due to lower Na^+ conductance density at dendrites and the brief interval between initial depolarisation and action potential regeneration, the capacitive current is much larger in distant dendrites.

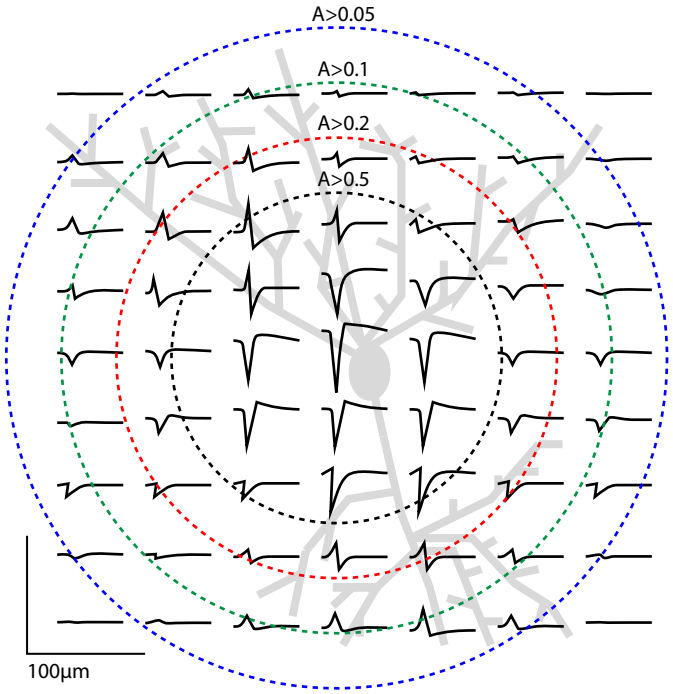


Figure 2.5.: Variation in the observed extracellular action potential profile with spatial position. Shown is an example in the vicinity of a spiking pyramidal cell. Adapted from [47]. Dotted lines illustrate the regions of relative spike amplitudes larger than 0.5 (black), 0.2 (red), 0.1 (green) and 0.05 (blue).

Another example for such variability is the duration of the Na^+-dominant phase of the waveform. If the electrode is placed away from the soma, both the somatic initiation and the back-propagation of action potential into dendrites contribute to the EAP waveform (both have similar scaling due to distance). Since back-propagation along the main dendrite takes a large fraction of a millisecond, the superposition of multiple moving current sources appear as a long-duration single static current source [59]. Thus, Na^+-dominant main peak will have a longer duration.

2.2. Spike Sorting

Spike sorting is a signal processing step which aims to find the number of neurons in a neural recording, establish specific attributes of each neuron, and subsequently classifying to identify the source of each incoming extracellular action potential (EAP). This separation in signal sources is made possible since EAP observed for each neuron varies depending on many factors. Neuron's dendritic tree topology, ionic channel distribution, inhomogeniety of extracellular medium, and topological placement of the micro-electrodes (i.e. in orientation and proximity) results in different signal dynamics, which in principle resolves which spike corresponds to which neuron [47, 64].

2.2.1. To sort, or not to sort, that is the question

The importance of spike sorting as a useful neuroscience tool is due to the fact that neurons (recorded over the same channel) can exhibit selectivity and fire in response to particular stimulus [65, 66]. For example, studies on human and rat medial temporal lobe — which is involved in visual memory retention, language and emotion processing — have shown that neurons in nearby areas fire to specific and unrelated concepts (hence referred as "concept cells") [66]. Another example to selectivity of individual neurons is shown in a study conducted by [67] on encoding characteristics of hippocampal cells. The results revealed that the encoding characteristics of hippocampal pyramidal cells are independent, and hence enable the hippocampus to store arbitrary associations (increasing its capacity) [67].

Even in situations where nearby neurons show increased levels of correlation to stimulus patterns, observing their individual firing characteristics and tuning properties can reveal insightful information about their interactions and roles within the neural network. An example to this is the ability of a neuron to inhibit the activity of nearby neurons (i.e. lateral inhibition) [65]. In addition, spike sorting is essential in detecting the sparsely firing neurons (with very low firing rates) which otherwise would have been completely missed due to neurons with higher firing rates. This further facilitates studying the organisational structure and connectivity of nearby neurons.

However, not only performing spike sorting is crucial, but also performing it in real-time is as important. Requiring computationally demanding signal processing due to inherent challenges (described below), spike sorting have been traditionally implemented off-line on computers. Nevertheless, over the past decade there has been a tremendous demand from both neuroscience research and prosthetic device perspective, and significant research efforts have been dedicated in realising real-time spike sorting (see Section 2.2.3.2). For example, real-time sorting is crucial in closed-loop experiments, which depend on the neural activity and behavioural performance [65].

On the other hand, prosthetic applications (e.g. artificial limbs) make use of the real-time sorting results for high decoding performance in controlling devices with multiple-degrees of freedom [68, 69]. Although there have been demonstrations of using BMIs with fewer samples [70–72] and some argue that future BMIs won't need large populations of neurons, the BMIs controlled by fewer neurons led to limited experimental demonstrations of motor control [32]. Moreover, functionality is distributed over cortex so including large number of neurons will reduce the errors associated with variability [33, 34].

Furthermore, any post-surgical disruption in electrode properties that could prevent recording from full original sample, or natural loss or death of neurons could lead to catastrophic results in systems using fewer neuronal samples [32]. Since the goal is to provide life-long operation (while maintaining a high-level performance), it is projected that sampling of around 1000 neurons from different cortical regions (each cortical area contributing 100-200 neurons) will render useful BMI applications [32].

In addition to being real-time, performing spike sorting on-node in the implant site is also crucial for BMI systems. The main drawback of current BMI systems is the use of percutaneous

wires connecting the recording system to external device for analysis (e.g. spike sort). There are two major disadvantages in such approach. First, wires passing through the skin barrier exposes the subject to risk of infections. Second, tethered connections imposes limitations on the nature of experiments preventing researchers to perform these in more ecological conditions [65]. Therefore, the concept of BMIs with wireless link transmitting signals to an external receiver have been proposed [73–75]. However, wireless streaming of raw data place severe restrictions on the number of channels observed due to bandwidth and power limitations. For example, considering commonly used parameters in neural acquisition — 25 kHz sampling at 16 bits — the required transmission bandwidth is 400 kbits/s per channel. In order to overcome this bottleneck, several dimensionality reduction techniques such as feature extraction prior to transmission have been proposed, and are discussed in detail in Section 2.2.3. Within this context, on-node spike sorting achieves even higher data reduction than transmitting features since the firing of the neurons are reduced to binary events. Considering that the number of neurons recorded from single electrodes may reach up to 6 (i.e. 3-bit representation of neuron information) [60, 76] with an average firing rate of 10 Hz, the estimated bandwidth required for transmitting sorted data is 180 bits/s per channel.

2.2.2. Challenges of Spike Sorting

There exists numerous methods for exploiting differences in spike shape to classify signals into their sources. Traditionally all have been computationally intensive off-line algorithms. Some prominent examples include principle component analysis (PCA), independent component analysis (ICA), and wavelets [77, 78]. Accompanying these are multitude of clustering algorithms such as k-means, expectation maximisation, subtractive clustering, Bayesian clustering [79] and super-paramagnetic clustering [77]. The choice of such algorithms were mainly driven by the inherent challenges of spike sorting problem, together with the hardware constraints on power and area resource for online implementation. It is crucial to understand the fundamental limitations, posed by inherent sorting challenges, in trying to achieve on-node sorting performance and resource efficiency. Therefore, an overview of these limitations are discussed in this section.

The entire concept of spike sorting, is built upon the idea that each neuron's spike has distinct features/morphology which enables the separation between different neurons [80]. However, the question arises: what is the degree of distinctiveness that enables separation? There have been numerous works which investigate the nature of the spike signal (intracellular (IC) membrane potentials), as well as the relationship between the IC and the extracellular waveform observed [81,82]. In addition to the effects of electrode-neural tissue interface [83], it has been shown that the signal observed show variations as a function of electrode distance as well as from where the signal is observed (e.g. soma, axon, etc.) [84]. Although, these factors are generally accepted as sources of separation, it is possible to have cases where these factors do not yield enough distinctions in EAPs recorded over a single channel for accurate spike sorting, such as the recordings from hippocampus where the neurons are tightly packed. For such circumstances, use of tetrodes (four-wire electrodes) in conjunction with sorting algorithms, that takes into

account the spatial and amplitude variations of observed EAPs across four-wires, have been proposed [84]. In other words, spike sorting of EAPs recorded over single channel is most effective when the density of the neurons is low such as in motor cortex — which is the main target application of the work presented in this thesis. Nevertheless, the degree of waveform similarity plays a significant role in all sorting applications.

Another issue that impacts sorting performance are variations in the spike waveforms over time. It has been found that the amplitude and waveform of spikes show substantial variation as a result of pre-synaptic network activity. Thus, the extracellular waveform changes over somatodendritic axis, while its characteristics are modulated by network behaviour [84]. Such an example is the bursting neurons which the first and the last spikes of the burst are not the same due to intracellular reasons. Moreover, encapsulation of the electrode by the scar tissue formation — wound healing and inflammatory immune response due to factors such as mechanical insertion trauma and bio-compatibility — results in degradation and change of signals observed over time [85, 86]. In addition, death and loss of neurons over time could also be observed [32].

Another sorting challenge which does not arise from physiological causes, but rather temporal conflict is the issue of overlapping spikes. This is the partial or complete superposition of multiple spikes in time [87], and can be the result of either synchronous firing or closing firing by chance [80]. The importance of resolving these overlapping spike waveforms not only lies with the fact that they may contain useful neural information, but also failure to do so will degrade the clustering performance since these create outliers in the cluster space [84].

All these challenges are further exasperated by the noisy nature of recordings. Additional noises sources to those originating from the electrode-tissue interface [83] and instrumentation (electronics), are the background activity of distant neurons [88] and noise contributions from axons, dendrites and synaptic currents [89–91].

These are the prime reasons why these algorithms have traditionally been computationally demanding and run off-line on PCs or dedicated computing devices. Therefore, mitigating these systems to implant site for real-time on-node recording in a resource efficient manner (thus also suitable for chronic implantation), is a significant engineering challenge.

2.2.3. State-of-the-Art Spike Sorting

For years, there have been numerous algorithms and systems proposed to either completely or partially solve the challenges of spike sorting. Owing to the computationally demanding nature of the problem posed, spike sorting has traditionally been implemented off-node. However, due to the rapid increase in the number of neurons monitored by implantable brain-machine interfaces, the increasing amount of data to be transmitted results in a power and bandwidth bottleneck. Hence, on-node spike sorting has become an alternative method to achieve data reduction (without sacrificing information quality) prior to transmission.

Whether off or on-node, there are two main strategies used to achieve spike sorting: (1) To measure or calculate features that distinctively identify the spike shapes (e.g. spike height,

principal components etc.), or (2) To use model spike shapes to classify the action potentials (template based approach) [92]. Template-based methods rely on the comparison of the incoming signals with pre-determined templates, and spike sorting is achieved by assigning the spike to the template with smallest difference. Template creation can be achieved by performing clustering on either the spike waveforms or other features. In the systems that employ the latter strategy, spikes are clustered via selected features, and template matching is used as the classification method.

In feature based approach, specific features of the spike which allow the most differentiability are extracted. This leads to dimensionality reduction; instead of comparing n variables, where n is the number of sample points, m variables ($m < n$) are compared, where m is the number of selected features. For example, some basic features include the amplitude, width, power, or phase of the spike [93]. Typical analytical methods to obtain features are: principal components analysis (PCA) [94], discrete wavelet transform (DWT) [95], discrete derivatives (DD) [96], zero-crossing features (ZCF) [97] and integral transform (IT) [98].

Once the features have been extracted they can be projected into their m-dimensional space, usually referred to as classification (or clustering). There exist several clustering algorithms (based on different partitioning methods) including k-means clustering [99], Bayesian clustering [100], valley detection [101, 102], superparamagnetic clustering [77], expectation maximization [84, 103] and artificial neural networks (ANN). A comprehensive review of clustering methods is provided in [104].

The objective of this section is to present the reader various spike sorting solutions proposed in literature and discuss the state-of-the-art.

2.2.3.1. Off-node Spike Sorting Systems

Since power consumption and computational complexity is not an issue for off-node systems, an abundance of algorithms have been published, often implemented using powerful simulation software (such as MATLAB) on workstations.

These algorithms are often complex, involving highly computational feature extraction, clustering and statistical analysis tools. Once the signals are recorded and stored, a selected feature extraction algorithm establishes the features that provide the best separation amongst the spikes, and spike sorting is performed via clustering the extracted features.

Among the off-line systems, the two most widely adopted automatic spike sorting algorithms are KlustaKwik [84] and WaveClus [77]. KlustaKwik achieves automatic spike sorting by utilising mixture of Gaussians and conditional expectation maximization (CEM) algorithm. On the other hand, WaveClus do not make any assumptions regarding the distribution of spikes and utilises an automatic threshold selection method (which is argued to improve the yield of detected spikes) for spike detection, Eq. 2.2. Once the spikes are detected via amplitude thresholding, wavelet transform is applied, a time-frequency representation of the signals, and the optimal coefficients for separating the spike classes are automatically selected. Then, these coefficients are clustered via super-paramagnetic clustering (SPC) algorithm.

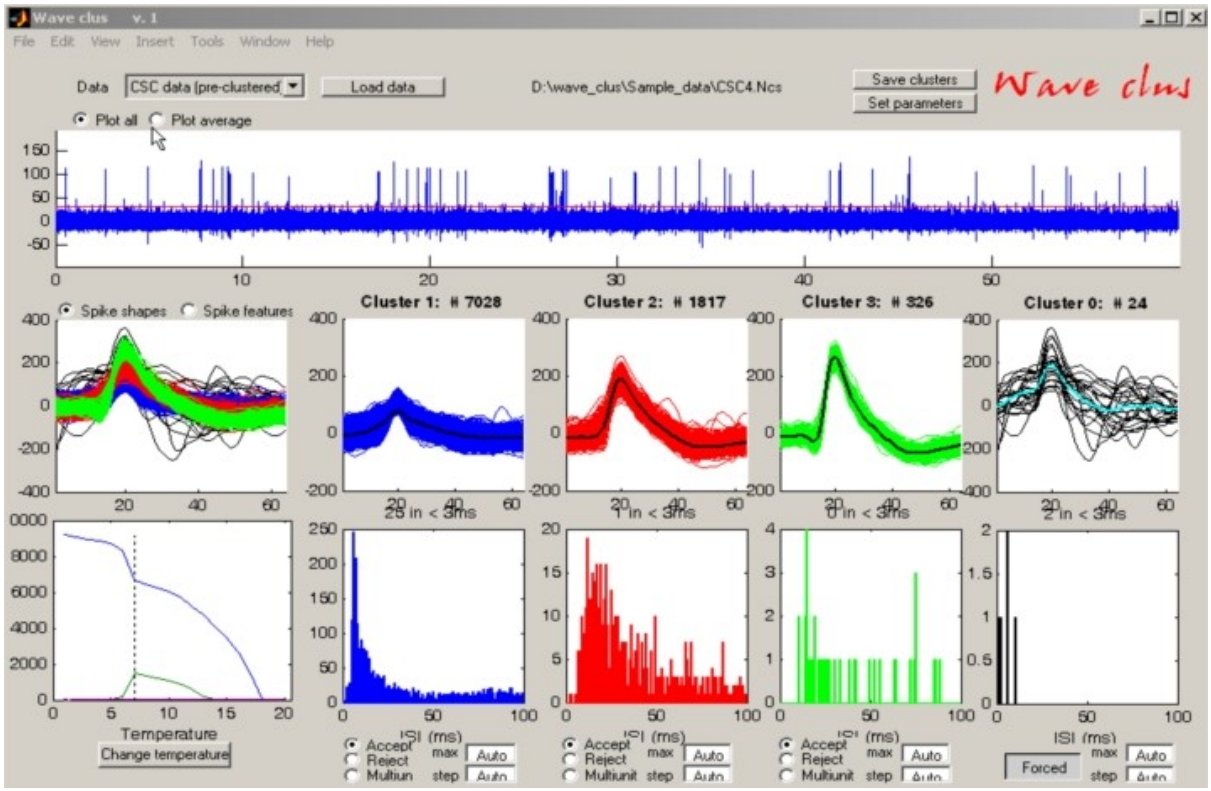


Figure 2.6.: WaveClus graphical user interface showing filtered spike train and detection threshold, sorted spikes and their interspike interval (ISI) [105].

However, as previously mentioned, these algorithms cannot be applied directly to on-node systems due to complexity and scalability issues. For example, WaveClus requires neural signals to be recorded and stored, since it performs statistical analysis on the whole data set to determine the threshold values, wavelet coefficients, and clusters. Adapting such a system without optimisation into real-time hardware implementation would require 221,312 flip-flops only for the storage of the signals to be processed. This would mean 15.27 mm^2 memory area in 0.18 μm technology and a significant power consumption. It would take at least 9.2 seconds to gather enough data for analysis (excluding the processing time of the data)[2], which would not serve as a real-time system.

Apart from the methods above, Independent Component Analysis (ICA) and Principal Component Analysis (PCA) are also popular methods which have been used often in conjunction with different techniques. PCA converts a group of correlated variables into linearly uncorrelated variables, called principal components, using orthogonal transformations [106]. On the other hand, ICA might be seen as an extension to PCA, and it aims to break a multivariable signal into subcomponents [107]. Majority of the off-line algorithms are either based on these methods or their derivatives. An example of such work which employs ICA is [108], while [78, 109, 110] incorporate PCA in their algorithms. [109] presents an algorithm in which

[2]Calculations mentioned are based on the average number of spikes (3458) calculated from data presented in Table 1 of [77], the sampling frequency (24 kHz) , and number of samples (64) used in the work.

Artificial Neural Networks (ANN) and X-means clustering is used to obtain a real-time PCA. On the other hand, [111] utilizes hierarchical clustering, while subtractive clustering and χ^2 test is used in [78].

In addition to these methods, there are plenty of other feature sets which have been used. For example, [112] proposes a linear feature extraction technique called 'Graph Laplacian', and claims that it performs better than PCA and wavelet based methods due to its relatively compact and well separated clusters (Figure 2.7). In another work, features related to spike repolarisation slopes are used together with Mahalanobis distance and k-means algorithm [113], while [114] proposes a "fully-automated" system based on Discreet Haar Transformation (DHT), and, peak-to-peak amplitude and intervals. There are also statistical approaches such as Expectation-Maximization based competitive decomposition algorithm [88] and probabilistic approaches such as Hidden Markov Models as described in [87].

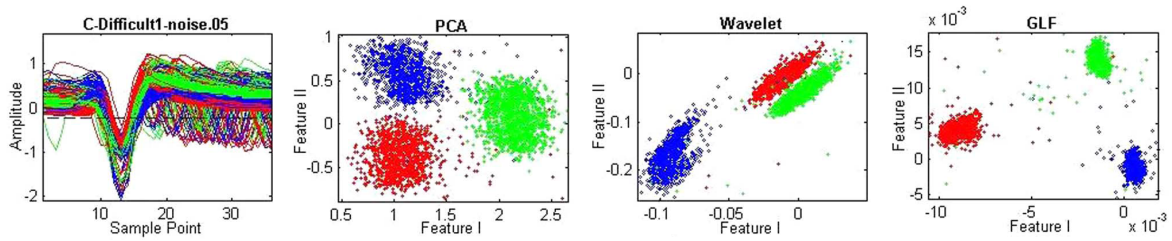


Figure 2.7.: Clustering performance comparison of PCA, Wavelets, and Graph-Laplacian features on Difficult1 dataset at 0.05 noise level. Extracted from [112]

Besides using features and clustering, another method employed by off-line algorithms is template matching. Simply using a model waveform to match the detected spikes, perhaps, template matching is one of the more suitable, among off-line methods described above, for on-node implementation. In fact, it has been suggested that template matching is preferred due to its versatility (since it can be combined with any clustering offline) [110,115] and comparable performance to other techniques [111]. For example, some of the works mentioned above [78, 110,111] incorporate template matching into their PCA based algorithms.

However, for template matching to be feasible for hardware, its computational complexity arising from template creation (e.g. PCA and clustering based) and distance calculation between template and spike samples need to be solved [110]. These computational costs can indeed be reduced significantly if supplementary techniques are used. One such technique, Davies-Bouldin Validation Indices (DBVI)[3], is proposed by [110]. The proposed index is used in conjunction with a Sequential Similarity Detection Algorithm (SSDA)[4], and an 83% reduction in distance calculations is reported. Besides these methods, amplitude-ordering and time-ordering are other alternative techniques that could also be used to reduce number of operations [110].

Furthermore, several different template matching quantifiers could also be utilised (instead

[3]DVBI is a function of the ratio of the sum of within-cluster scatter to between-cluster separation.

[4]SSDA is a method of successively comparing windowing pairs in a certain order and discarding the data when the distance between the template and signal exceeds a threshold

of widely used Euclidean distance) to improve matching quality and/or reduce computational costs. For example, [115] uses weighted Euclidean distance in conjunction with Levenberg-Marquardt method (to estimate variance). Another work, [116], from the field of image processing, proposes a new method called "Optimized Sum of Difference (OSAD)" in order to solve a face localization problem. The algorithm calculates the absolute value of the difference between two samples, chooses the largest, and divide the absolute value of the sum with it. Apart from the OSAD method, other methods, which might have potential applications in spike sorting, are also compared in their work. Table 2.2 presents the performance of other techniques used in template matching.

Table 2.2.: Comparison Between Similarity Measurements used in template matching. The accuracy figures are extracted from [116].

Similarity Measure	**Accuracy(%)**
Optimized Sum of Absolute Difference(OSAD)	100
Sum of Absolute Differences(SAD)	98
Zero-mean Sum of Absolute Differences(ZSAD)	98
Locally scaled Sum of Absolute Differences(LSAD)	98
Sum of Squared Differences(SSD)	95
Zero-mean Sum of Squared Differences(ZSSD)	95
Locally scaled Sum of Squared Differences(LSSD)	95
Normalized Cross Correlation(NCC)	80
Zero-mean Normalized Cross Correlation(ZNCC)	80
Sum of Hamming Distances(SHD)	43

2.2.3.2. On-node Spike Sorting Systems

On-node spike sorting implementations have been a hot topic for research for past two decades, as the implantable neural recording systems are struggling to cope with the power and bandwidth demands due to ever increasing number of neurons recorded. In fact, dubbed as the Moore's law of Neuroscience, [26] demonstrates that the number of neurons recorded since 1960s has exponentially grown. The study reveals that the number of neurons recorded double every 7 years, and projects physiologists will be able to record from 1000 neurons by year 2025 (see Figure 2.8).

This growth, enabled by the advances in micro-electrode technology, is driven by the fact that BMIs need to monitor large samples of neurons in order to study the functionality of the brain and provide control for assistive devices such as predicting the intended movement for a prosthetic limb (discussed in detail in Section 2.2.1). Combined with the need for wireless telemetries for BMIs — to eliminate infection risk due to percutaneous wires and to allow researchers conduct experiments with subjects (untethered) in natural surroundings—, wireless transmission of such large quantities of data is not a feasible option for implantable prosthetics due to power and bandwidth constraints posed by safe thermal dissipation (in order to prevent

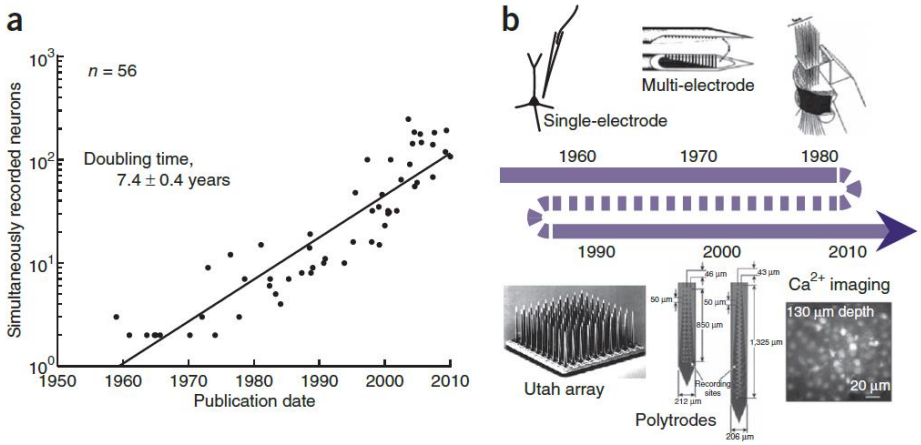


Figure 2.8.: Exponential growth in the number of recorded neurons. (a) Number of simultaneously recorded neurons, (b) Timeline of recording technologies. Extracted from [26].

tissue damage) and battery life. Thus, due to its potential of data reduction, real-time on-node spike sorting has gained significant importance in the last decade.

One of the biggest challenges for on-node spike sorting implementation is to achieve a comparable accuracy to their off-node counterparts. Considering the computational demands to overcome spike sorting challenges (such as spike variability, noise sources, overlapping spikes etc.), achieving such high performance while keeping hardware resources at a minimum is not a trivial task.

Hence, in order to meet the trade-off between accuracy and hardware resource utilisation, numerous approaches have been implemented for on-node spike sorters. While some propose a fully-analogue approach [117], others introduce custom digital implementations (excluding the front-end analogue electronics) [118–123]. On the other hand, off-the-shelf implementations are also proposed such as systems that use DSP, FPGA or other off-the-shelf systems [124–126].

One such example of utilising commercially available off-the-shelf products is proposed by [125]. In this work, on-node real-time spike sorting is implemented on smartdust wireless sensor networks. It is argued that using available platforms is easier, cheaper and more risk free considering constant changes in spike sorting algorithms. Based on the same argument, [124] proposes an unsupervised system which was implemented on commercially available Cyberkinetics Neuroport® System. The system proposed uses log-likelihood and squared Mahalanobis Distance as its method to classify the spikes, and can adapt to changes in the waveform due to electrode movement by changing the boundaries for clusters. In terms of forming the clusters, Pairwise Cluster Dividing and Combining (PCDC) — which the histogram built is examined and combined/divided depending on the available peaks of the collected data— is used. However, although the system proposed is claimed to be autonomous, the algorithm uses a built model with a priori information about the spikes. In other words, their system is input a typical neural spike waveform that is expected at the tip of the electrode [124].

On the other hand, [126] presents a system which runs on a DSP, mounted in an IBM Pen-

tium 5 computer. The proposed system implements a Neural Network (NN) in order to do the classification. In addition to the implementability of the algorithm on high-speed processors, it is argued that NNs are capable of resolving super-positions of units in a multi-unit channel recording. NN based algorithm is compared with a matched template filtering (MTF) to evaluate the performance. According to the results presented, NN and MTF perform almost the same in terms of correct classification without super-positions in the dataset, while NN outperforms MTF by 20% for classifications with super-positions. Additionally, another variety of commercially available solutions is implementing these algorithms on FPGAs. As a low cost and easy hardware implementation, [127] uses FPGA to realise a Hebian Eigen Filter based PCA algorithm.

The third approach to design optimum on-node spike sorting systems is custom analogue or digital approach (or mixed). In order to reduce the computational and memory requirements of the subsequent classification, custom implementations generally employ simpler features and distance metrics. One such system which uses the peak and trough amplitudes, and peak width is described in [117]. As a fully analogue system, it is implemented in 0.35 μm CMOS technology with a power consumption of 70 μW per channel (including the pre-amplifier). It is claimed that single-units were assigned correctly with a percentage over 90% [117].

Another set of features that have been proposed are Discrete-Wavelet Transforms (DWT) [119, 121, 128]. [128] proposes a system that uses DWT and PCA for feature extraction, and K-means algorithm for classification. A cubic spline interpolator prior to alignment is also proposed by the authors to increase system performance. Implemented in 90 nm CMOS technology, the proposed neural signal processor achieves 0.68 μW per channel, with an area of 8.89 mm^2. In another study, a neural processor chip which extract features based on a method called "Discrete Derivatives" (DD) is proposed [129].

On the contrary, [120] presents an alternative simpler feature extraction algorithm called Zero-Crossing-Features(ZCF). ZCF is based on different neurons having spikes with "different area after the first zero-crossing" [120]. These features are compared to PCA, and found to be equally good performing while using 5% of the resources [97]. Furthermore, another study proposes "Self-organising Maps" —an unsupervised learning neural network algorithm— in conjunction with ZCF to achieve real-time spike classification [130]. Alternatively, [131] introduces a feature denoising filter and a new metric, called integral of repolarisation, along with maximum and minimum features as a low-resource and robust feature set for spike sorting.

In another study, a decision tree based classification is proposed as an alternative to commonly used L1-norm distance [132]. The authors show that the proposed method achieves feature classification as accurate as L1-norm but more efficiently. The synthesized implementation in 0.13 μm technology reveals 32 nW power and 5115 μm^2 area per channel.

There have also been approaches which rely on template-based operation. For example, unsupervised classification algorithms developed by [118] and [123] employ time-domain waveforms. [118] introduces a novel architecture called "Neural Cache", and perform classification based on Euclidean distance between the incoming spikes and the templates. On the other

Table 2.3.: Summary of spike detection methods along with varying pre-processing and noise estimations techniques [12, 134–138].

Threshold crossing	**Pre-processing**	**Noise estimation**
Single	Absolute	Root Mean Squared
Double	Square	Median of Absolute Deviation
Matched Filter Based	Negation	Cap-fitting
	NEO†	Local Averaging
	SNEO‡	Envelope Tracking

† Nonlinear Energy Operator
‡ Smoothed Nonlinear Energy Operator

hand, [123] argues that Euclidean distance is sensitive to biological noise and proposes the use of L_1 norm metric which is more robust to noise. The authors implement a modified hardware implementation of an on-line clustering algorithm developed by [80]. The algorithm clusters the data serially, calculates cluster means and then the spikes are mapped to the available clusters using the minimum L_1 norm. The proposed architecture achieves a clustering accuracy of around 75% with a power consumption of 75 μW at 270 mV, and an area of 2.45 mm^2 in a 65 nm CMOS technology [123].

2.3. Spike Detection

Preceding spike sorting, spike detection is one of the most crucial steps in neural signal processing chain, as it has direct impact on system performance (both accuracy and resource utilisation). It can be either implemented prior to digitisation stage (in analogue circuitry) or performed post-digitisation (in digital) [133].

In its most basic form, spike detection makes use of voltage threshold crossings to differentiate EAPs from background noise [92]. These thresholds are typically calculated based on estimated background noise, and can either be fixed or adaptive. In fixed threshold scenarios, a pre-programmed threshold is used throughout the recording. On the other hand, adaptive threshold approach monitors the changes in the background noise and adjust the threshold value accordingly. Calibration for adaptive threshold can either be done by off-line at certain intervals (re-training) or can be integrated on-node.

Although single thresholding on raw or pre-processed signals are most widely used forms of spike detection, nevertheless, there exists multitude of spike detection methods with varying pre-processing and noise estimation algorithms. Some of these are summarised in Table 2.3.

Among these methods, the two most widely used are single threshold: (1) on raw signal, and (2) single threshold on NEO processed data. An example of simple threshold on raw data is employed by WaveClus algorithm in conjunction with Median of Absolute Deviation (Eq. 2.2) noise estimation technique [77]. The noise is estimated based on the median of the signal in order to reduce the effect of spikes in the estimation. They argue that for large spike amplitudes

and high firing rates, estimating the noise based on standard deviation of the signal lead to high threshold values (hence increase error rate) [77]. The threshold calculation assumes that noise is Gaussian in nature, and is given by:

$$\begin{aligned} \sigma_n &= median\left\{\frac{|x|}{0.6745}\right\} \\ Thr &= 4\sigma_n \end{aligned} \tag{2.2}$$

where 0.6745 in the denominator is the value of cumulative distribution function of the standard normal distribution evaluated at 0.75. Factor of 4, multiplying standard deviation, is calculated empirically.

For NEO-based spike detection, on the other hand, thresholding is performed on the energy estimate of the signal given by NEO — also known as Teager Energy Operator [133, 134]. NEO emphasizes signal instances where both frequency and amplitude is high [139]. It is given by:

$$\psi[x(n)] = x^2(n) - x(n+1) \times x(n-1) \tag{2.3}$$

The detection threshold is calculated by taking the mean of the NEO signal and scaling it with a constant defined empirically.

2.4. Summary

Ability to interface with the brain in order to understand and harness its powers have been one of the biggest challenges over the past decades. Together with advances in technology, our ability to interface and establish a communication with the nervous system have dramatically increased. Today, Brain-machine interfaces provide an invaluable platform to study the brain, and use our understanding to treat many neurological diseases and disabilities. Over the past decades, thousands of patients have benefited from assistive BMIs such as deep brain stimulators, cochlear implants and spelling devices. Depending on the information required for particular application, BMIs can employ different modalities of interfacing at varying levels of invasiveness.

Advances in electronics and micro-electrode technology has led to the development of BMIs that can interface intra-cortically to the brain monitor hundreds of neurons. Coupled with spike sorting algorithms — which aim to differentiate recorded signals into their individual sources—, these interfaces are instrumental in deciphering the underlying network dynamics, connectivity of neurons, and biological processes. Hence, these developments have contributed significantly to fields such as cognitive and computational neuroscience. Moreover, these techniques can directly be translated in controlling assistive devices such as prosthetic limbs with multiple degrees of freedom. In fact, the most current research have already demonstrated cortically controlled devices with which subjects used for feeding, locomotion, and handling objects.

However, extending capabilities of neuroscience research and clinical translation of BMI systems require developing real-time spike sorting algorithms that can be performed on-node in the implant site. The motivation for this demand from neuroscience community can be summarised

by:

- Real-time operation:
  - Performing closed-loop experiments which depend on neural activity and behavioural feedback in real-time.
  - Decoding algorithms used in assistive devices such as prosthetics rely on real-time spike sorting results for high performance.
- On-node operation:
  - In order to ensure safety of the experimental subjects and patients, traditional method of connecting intra-cortical electrodes via wires through skin barrier have been superseded by wireless telemetries. However, considering future demands (in parallel with advances in recording technology), transmitting raw information to be processed externally is not sustainable for a multi-channel systems monitoring thousands of neurons. On-node spike sorting achieves data reduction by representing detected spikes as labelled binary events (i.e. only a tag resenting neuron identity needs to be transmitted).

Traditionally implemented off-line due its inherent challenges, implementing real-time on-node spike sorting is a much harder problem to tackle. The solution not only needs to overcome these challenges and perform comparable to off-line counterparts, but also it has to achieve such operation in a resource efficient manner (in terms of power and area) in order to ensure: (1) Long-term operation (battery life), (2) Safe operation within biological limits (i.e. not cause tissue damage), and (3) efficient operation to justify its on-node implementation (i.e. wireless link power savings should be much higher than energy spent for on-node spike sorting).

In addition to resource efficiency, another critical and desired feature of spike sorting implementations is adaptability. As the background noise levels and spike shapes change over time, the proposed on-node solutions must also adapt to these in order to ensure long-term stable and accurate operation. Although this feature is very important for experimental use of BMIs as well, it is vital for clinical translation of the BMI systems in the future.

In order to achieve resource efficient on-ship sorting numerous approaches investigating different features, clustering and classification methods, and implementation methods have been proposed over the past decade. These have helped and inspired numerous works, including the work presented in this thesis. However, despite all efforts and reported systems, there hasn't been, to date, a system to overcome all the challenges associated with real-time on-node spike sorting.

3. Optimisation of Analogue Front-End for On-node Spike Sorting

Analogue front-end (AFE) is a critical pre-processing stage in all neural recording systems. It directly impacts the quality of the recording since AFE involves amplification, signal conditioning, and digitisation of recorded EAPs. Therefore, selection of front-end specifications may directly affect the performance of the subsequent post-processing stages, namely spike detection and sorting. There exists multitude front-end designs in literature with varying specifications (see Section 3.5.1). Since the quality of the recording may affect the accuracy with which spikes are detected and sorted during post-processing, these designs often employ high front-end specifications.

Although such approach ensures accurate post-processing performance, it is not compatible with the future trends in monitoring large number of recording channel since higher front-end specifications result in more power and area utilisation in hardware. Therefore, it is imperative that the impact of front-end design parameters on post-processing performance is considered during design time to ensure minimum hardware resource utilisation. In other words, the underlying hypothesis is that the current state-of-the-art front-ends are over-engineered since they do not take into account the following spike detection and sorting stages during design time. In the light of the this discussion, the primary research question that motivated this work centres around identifying front-end specific parameters and quantifying the sensitivity of the post-processing performance to these.

In order to achieve such optimal design, the objectives of the work presented in this chapter[1] are:

1. Developing an optimisation framework for the parameters related to analogue front-end design.

2. Quantifying the sensitivity of the spike sorting accuracy to identified front-end parameters using the developed framework.

Therefore, a behavioural model for the front-end is developed. By modelling the AFE using a behavioural tool, the designer can investigate the effect of different topologies in a fast way, and before committing to a specific circuit implementation, such as to optimise hardware resources

[1]The work presented in Section 3.2 was done in collaboration with Dr. Yan Liu. Dr. Yan Liu designed the front-end integrated circuit used in this study, and performed analysis of simulated and measured circuit response. Miss Helen Root and Pooja Bunjun have contributed to graphical user interface in the past.

[145]. Thus, a good balance between resource efficiency and outright performance can therefore be struck at design time.

Following an overview of analogue front-end architectures in Section 3.1, Section 3.2 introduces the behavioural front-end model that allows the optimisation of system parameters (including signal-to-noise ratio, filter type/order, bandwidth, converter resolution/rate) at design time. In Section 3.3, an optimisation tool based on the developed behavioural model is presented. The proposed behavioural model is then validated, in Section 3.4, by applying a test stimulus to a hardware platform and comparing the measured circuit response to the expected from the behavioural model. This is then used to demonstrate the effect of the AFE on subsequent spike processing by testing established spike detection and sorting methods on a selection of different state-of-the-art configurations in Section 3.5.

It is revealed that although these designs have a wide variation in design parameters (and thus also circuit complexity), the ultimate impact on spike processing performance is relatively low (10-15%). This confirms the initial hypothesis that many of these front-ends are over-engineered, hence hardware resource utilisation can be significantly reduced (by up to an order of magnitude) through optimising front-end parameters with minimal compromise on spike processing performance. Consequently, in Section 3.6, a parametric design optimisation is presented (utilising the AFE model) that investigates each specification of all the different front-end blocks in order to reveal the sensitivity of spike sorting performance to various AFE parameters, and a set of near optimal AFE parameters is established. This is followed by conclusion in Section 3.7.

3.1. Front-End Neural Interface Architectures

As previously mentioned, front-end electronics is a crucial pre-processing step in spike sorting systems that not only has a direct impact on detection and sorting accuracy, but also on hardware resources, namely power and silicon area.

The specifics of the electrode material, the electrode/tissue interface as well as the nature of the bio-potential signal itself pose challenges on the front-end microelectronics [86]. The signals observed contain an electrode offset potential (due to the electrode-electrolyte interface) as well as both the extracellular action potentials (EAPs) and local field potentials (LFPs). The EAPs typically have amplitudes of $25\,\mu\text{V}-1\,\text{mV}$ and are recorded with a signal band of $300\,\text{Hz}-5\,\text{kHz}$, whereas the LFPs have amplitudes up to $10\,\text{mV}$ recorded with a signal band of $1-300\,\text{Hz}$ [74]. Additionally, the electrode-electrolyte interface introduces an offset that can be several 100's of mV, with the microelectrodes themselves contributing thermal noise due to their relatively high impedance. All these factors dictate the minimum requirements for the front-end electronics, that are additionally limited by resource constraints (power, size and bandwidth). In particular, the desire to make such systems implantable poses limits on size and thermal dissipation (i.e. to prevent tissue damage) [140], as well as requiring wireless transmission (i.e. thus limiting communication channel capacity) [141, 142].

Regardless of the choice of sorting method, sorting accuracy directly correlates with the performance of front-end electronics. While the demand for sorting performance enforces minimum requirements on the front-end, the resource constraints (power and area) limit the scalability but also strike a trade-off with the front-end specifications. Therefore, in order to achieve both a good resource efficiency and spike sorting accuracy, careful front-end optimisation is crucial. In order to ensure the optimal trade-off between accuracy and hardware requirements, one must identify all parameters associated with each stage of front-end, and analyse their effect on accuracy and hardware resources.

A typical architecture for a front-end neural interface (for applications using EAPs) is illustrated in Figure 3.1. This consists of three main blocks: (1) the bio-potential amplifier, (2) bandpass filter, and (3) data converter. This section describes the requirements for each of these blocks:

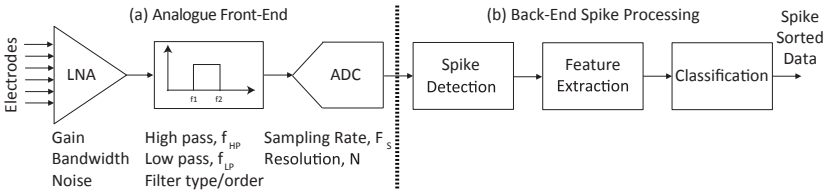


Figure 3.1.: General architecture of a neural interface, showing: (a) Analogue Front-End (AFE), consisting of a low noise amplifier (LNA), filter and analog-to-digital converter (ADC); and (b) back-end spike processing.

A low noise preamplifier is required to boost the signal level from sub-mV to 10's of mV, such that the subsequent stages can have relaxed noise performance. This is often designed to be AC-coupled such as to remove any DC levels due to electrode offset, and typically provides a gain of $50-200$, with a bandwidth of $3-10\,\mathrm{kHz}$, and input-referred noise of $1-10\,\mu\mathrm{V_{rms}}$. Key parameters for subsequent spike sorting are thus the Gain-Bandwidth Product (GBW) and input-referred noise (IRN).

Following pre-amplification, a bandpass filter is required to: (1) reject the out-of-band LFPs (high pass), and (2) prevent aliasing (low pass). The high-pass cut-off frequency is typically set between $100-300\,\mathrm{Hz}$, and low-pass between $3-10\,\mathrm{kHz}$. Due to the close proximity between the high- and low-pass cut-off frequencies, a sharp response is required to avoid in-band attenuation and it is thus desirable to use a high order filter. A key challenge is however, to minimise the effect of phase distortion as this will impact subsequent signal processing. Key parameters for the filter design are therefore the filter type, order, and low-/high-pass cut-offs.

The final stage of the front-end processing is the analogue-to-digital converter (ADC) which digitises the processed neural recording. The main design specifications for the analogue-to-digital converter (ADC) are the resolution and sampling rate (typically 8-10 bit, and $16-32\,\mathrm{kS/s}$). Although these set the numerical accuracy in subsequent spike sorting computation, this is fundamentally limited by the signal-to-noise ratio (SNR) and bandwidth of the signal. The resolution and sampling rate must therefore be optimised to minimise the raw data throughput.

3.2. Front-End Modelling and Behavioural Optimisation

In this section, the proposed front-end behavioural model is discussed in detail. In addition, the specifics of testing and evaluation strategy — used in model validation and demonstration in Sections 3.4 & 3.5 — are presented.

3.2.1. Front-end Behavioural Model

To accurately characterise front-end architectures and circuits, one must consider not only ideal behaviour, but also the non-idealities that a realistic circuit implementation introduces [146]. This can significantly affect the signal fidelity and as such may impact downstream spike processing.

3.2.1.1. Electrode Model

The electrode is the conduit between acquisition electronics and neural tissue. Over the last several years, with the drive of microtechnology and fabrication techniques the number of simultaneously recorded single neurons has greatly increased and projected to double every seven years [147]. Commercially available multi-electrode arrays today can interface with 10s to 100s of electrodes [148].

Each electrode is typically characterised by its charge density (for stimulation) and impedance characteristics [149], the latter a vital parameter for recording. In-depth characterisation of some of the state-of-the-art in electrodes can be found in [148]. The impedance characteristics of the electrode play a vital role in deriving the noise added to the signal prior to amplification [89,149]. This non-ideality is attributed to electrochemical effects at the tissue-electrode interface, scar tissue formation, and inherent electrical properties of the electrode (i.e. material, area) [149]. When in contact with tissue, the electrode forms an electrical double layer capacitance.

This capacitance depends on the electrode surface texture and area [74, 149], and it is calculated as a series combination of double layer and diffusion layer capacitance. It is typically modelled as as constant phase element (CPE) [150], which is highly dependant on electrode area.

One can also consider impedance changes due to injury related mechanisms, possibly as a result of electrode insertion into the tissue. For simplicity, the electrode can simply be modelled as a frequency independent model [149,151], where the dominating restive contribution includes spreading or seal resistance (resistance between electrode and medium, i.e. neural tissue). The noise is then defined as:

$$V_{rms} = \sqrt{4kTR_{eq}B} \tag{3.1}$$

where k is Boltzmann's contestant, T is temperature in Kelvin, R_{eq} is the electrode resistance and B the signal bandwidth. In a number of studies R_{eq} is measured at a specific frequency [147, 148].

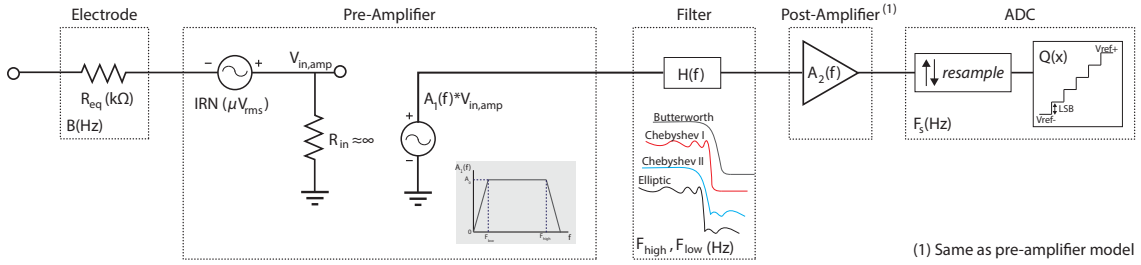


Figure 3.2.: Basic system behavioural macromodel, including input parameters for each functional block. This include the electrode (equivalent resistance and bandwidth), pre-amplifier (gain, IRN and frequency cut-offs), filter (type and cut-off frequencies) and ADC (sampling frequency, resolution and references).

3.2.1.2. Pre and Post Amplifier Model

After the electrode a low noise pre-amplifier is required to increase the signal level from sub-mVs to 10s of mVs with minimal additional noise. These are typically designed to be AC-coupled to remove DC electrode offset which can be in the order of 100s of mV depending on electrode material. Gains of $50-200$, bandwidths of $3-10\,\text{kHz}$, and input-referred noise (IRN) of $2-10\,\mu\text{V}_{\text{rms}}$ are typical specifications for these amplifiers [12, 21].

Since both the pre- and post-amplifiers are the same in terms of functionality, the model described below applies to both. Due to the pole introduced by the device parasitic capacitances, the gain of the amplifier starts to roll-off (20dB/dec) at high frequencies, which can be characterised by a low-pass response. In addition, since the DC offset introduced by the electrodes need to be removed [14], often front-end amplifiers are built with a high-pass response introduced via a feedback loop [152]. Therefore, a 2^{nd} order Butterworth filter with a mid-band gain is used to model the amplifier. Although amplifiers may contain a second pole, this is typically placed at frequencies higher than the ones of interest for phase stability.

The IRN of the amplifier is a combination of thermal and flicker noise and is normally measured in $nV/\sqrt{(Hz)}$. In a model, this is a key target specification, and it can be combined with the amplifier bandwidth to give an indication of the added V_{rms} noise. In the implementation, Matlab's *randn* function is used to generate this noise.

The non-idealities of the system are therefore: input/output offsets, non-linearities, among other noise sources. However, it is typically the gain, bandwidth and noise that are the target specifications in the design of these amplifiers.

3.2.1.3. Bandpass Filter Model

Following pre-amplification, a bandpass filter is required to: (1) reject out-of-band LFPs (high pass), and (2) prevent aliasing (low pass). The high-pass cut-off frequency is typically set between $100-300\,\text{Hz}$, and low-pass between $3-10\,\text{kHz}$. Due to the close proximity between the high- and low-pass cut-off frequencies, a sharp response is required to avoid in-band attenuation and it is thus desirable to use high order filters. A key challenge is however, to minimise the

effect of phase distortion as this will impact subsequent signal processing.

The most important parameters associated with filtering are filter type, order, cut-off frequencies, passband and stopband ripple. The filter stage is thus modelled based on these parameters, where a bandpass filter transfer function is utilised in conjunction with the built-in Matlab function *filter*.

The filter model preserves the phase information of the filter, as opposed to other filtering methods such as *filtfilt* (a Matlab built-in function). Preserving this attribute of analogue filters is of utmost importance, since it has been shown that non-linear phase dependence with frequency, may cause significant distortions in the shape of the observed spikes, thus affecting the spike detection and sorting performance [153].

In the developed model, the user can input any filter order and cut-off frequencies, and is given the choice of four different filter implementations: Butterworth, elliptic and Chebyshev Type I & II.

3.2.1.4. Analogue-to-Digital Converter Model

The main design specifications for the analogue-to-digital converter (ADC) are the resolution and sampling rate (typically 8-12 bit, and $16-32$ kS/s). Although these set the numerical accuracy in subsequent spike computation (detection and sorting), this is fundamentally limited by the signal-to-noise ratio (SNR), dynamic range and bandwidth of the signal.

In the model, the primary parameters for ADC are sampling rate, resolution (ENOB) and reference voltages for the ADC. Based on these parameters, first the signal is resampled (using the built-in Matlab function *resample*) and then the resampled signal is quantised using Eq. 3.2 and Eq. 3.3.

$$LSB = \frac{V_{\text{ref+}} - V_{\text{ref-}}}{2^N - 1} \tag{3.2}$$

$$Q = \text{sgn}(x).\left\lfloor \frac{x}{LSB} + \frac{1}{2} \right\rfloor \tag{3.3}$$

where Least Significant Bit (LSB) is the ADC step size, $V_{\text{ref+}}$ and $V_{\text{ref-}}$, are the positive and negative voltage references, N is the number of bits (resolution), x is the sample to be quantised, and Q the quantised signal.

It should be noted that the behaviour modelled is of an ideal ADC, and there are numerous non-idealities that impacts ADC output. These include offset and gain errors, integral and differential non-linearities, aliasing and quantisation effects. These non-linear effects and non-idealities will be minimised according to the performance of the prior analogue stages, and resolution requirements for the spike detection and sorting, so that the equivalent non-ideal effects of the ADC referred to the input will be smaller than the input referred noise of the analogue circut and electrodes.

On the other hand, quantisation effects are quantified in the definition of resolution and reference levels. In fact, instead of using the ADC resolution, the model uses the effective

number of bits (ENOB) that is typically effective in encompassing ADC non-idealities, especially for the following detection and sorting stages.

3.2.2. Testing and Evaluation

As mentioned, the aim is to: (1) establish a behavioural front-end model, (2) develop a front-end optimisation tool based on this, (3) validate the model using an application specific integrated circuit implementation, and finally (4) demonstrate the tool's usefulness in establishing a good balance between spike processing performance and hardware efficiency during design time. In the following sections the test methodologies and accuracy quantifiers associated with detection and sorting, are defined.

3.2.2.1. Spike Detection

Spike detection is the process of identifying that an EAP has occurred and has been recorded by the system. Three common spike detection methods utilised here are: single positive thresholding, absolute value thresholding, and single positive thresholding with the Non-linear Energy Operator (NEO) (Figure 3.3).

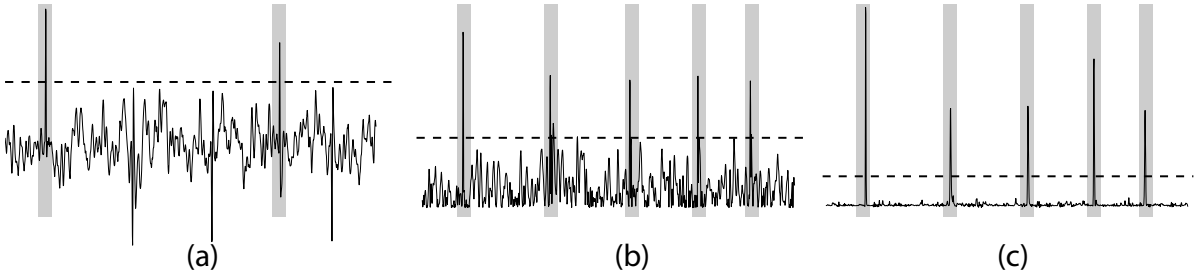


Figure 3.3.: Illustration of 5 neurons being detected with: (a) a single positive threshold; (b) an absolute value threshold; and (c) a single threshold after NEO processing. Note the shaded regions indicate successfully detected spikes.

Single Threshold Single positive thresholding applies an amplitude threshold to the signal, whereby a spike is detected upon crossing it. The threshold level is set as described in [77]:

$$thr = 4\sigma_n = 4 \times median\left\{\frac{|x|}{0.6745}\right\} \tag{3.4}$$

where σ_n is the estimation of the background noise standard deviation [135].

Absolute Value This method applies the amplitude threshold of Eq. 3.4 to the absolute value of the signal, i.e. $|x|$.

NEO In this detection method, the amplitude threshold is applied after processing the neural signal with a Non-linear Energy Operator (NEO) ψ given by Eq. 3.5. The NEO, also known

as the Teager Energy Operator [134], estimates energy by taking the square of the product of amplitude [154], and is defined as [133]:

$$\psi[x(n)] = x^2(n) - x(n+1) \times x(n-1) \tag{3.5}$$

As described similarly in [155] and [156], threshold Th_{NEO} is the mean of the NEO scaled by a constant, C, which is defined empirically to be 7.5.

$$Th_{NEO} = C\frac{1}{N}\sum_{n=1}^{N}\psi[x(n)] \tag{3.6}$$

3.2.2.2. Spike Sorting

Once a spike is detected, the process of identifying to which of the detectable neurons in the vicinity of the electrode it belongs to, is referred to as spike sorting. For the results reported herein, spike sorting is carried out with three different methods.

Template Matching (TM) This method involves aligning the maximum peak of the signal with a spike template and using the Squared Euclidean Distance (3.7) as a similarity/distance measure.

$$ED = \sqrt{\sum_{i=1}^{n}(x_i - T_i)^2} \tag{3.7}$$

where n is the number of data points in the spike template, x is the detected spike and T the template. The templates are created by taking the mean of the spikes (within each cluster), aligned to their individual maximum peaks. To create templates, a training dataset is used, and TM performance is assessed using a separate testing dataset.

Principle Component Analysis (PCA) PCA is a well established method for extracting orthogonal components of a signal and is typically used as a benchmark for spike sorting systems. Here, the first two principle components (for each spike) and k-means for clustering (50 iterations) are used. Matlab's in-built function *princomp* is used to calculate principle components.

First and Second Derivative Features (FSDE) This method is based on taking the minimum and maximum values of the 2^{nd} derivative and the maximum value of the 1^{st} derivative (within each spike) which has been shown to provide good performance in resource constrained hardware [143]. Matlab function *gradient* is used to calculate the derivatives.

For both methods Matlab's in-built clustering function *K-means* is used.

3.2.2.3. Evaluation

Here the quantifiers for accuracy in both spike detection and sorting are defined. These will be used as the metric for evaluating the effect of different neural interface architectures on detection and sorting.

Spike detection accuracy (SD_a) is given by:

$$SD_a = P_d \times \theta(P_d) \tag{3.8a}$$

and

$$P_d = (1 - \frac{N_e}{N_{su}}) \tag{3.8b}$$

where N_e is the total number of missed spikes and false positives, and N_{su} is the number of spikes. $\theta(x)$ is the unit step function which ensures that detection performance is zero when the number of errors are higher or equal to number of spikes.

$$\theta(x) = \begin{cases} 1, & \text{for } x \geq 0 \\ 0, & \text{for } x < 0 \end{cases} \tag{3.9}$$

Spike sorting accuracy is defined as:

$$SS_a = \frac{N_{cs}}{N_{det} - N_e} \tag{3.10}$$

where N_{cs} is the number of correctly identified spikes, N_{det} is the number of detected spikes, and N_e is the number of detection errors as described above. Eq. 3.10 only reflects the accuracy of spike sorting algorithm and does not include errors associated with detection. Furthermore, a combined accuracy figure-of-merit (FOM) for spike detection and sorting accuracy SDS_a is defined as:

$$FOM_a = SD_a \times SS_a \times 100\% \tag{3.11}$$

3.2.3. Test Data

The methods were tested using a total of 30 synthetic datasets that were created according to methods described in [157]. These contain five different groups (each using 3 single units) that are simulated at three different SNR levels. For training and testing, separate recordings are used. For more detailed information on dataset generation, please refer to Appendix B.2.

3.3. Behavioural Model Realisation

Based on the behavioural model established in Section 3.2.1, a graphical user interface (GUI) was developed in MATLAB, and is available online[2]. The tool provides a front-end modelled as a single recording channel (Figure 3.4), in which the signals can be analysed at each node (1-6) to study the behaviour of each front-end block. Through the GUI, the user can input all the critical front-end design parameters, and observe the input and output signals in a simple and user-friendly graphical user interface.

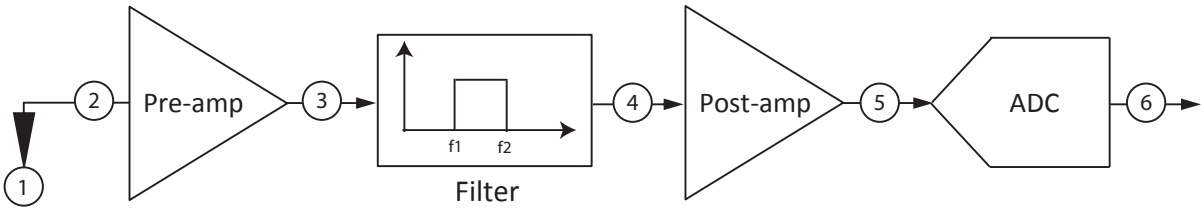


Figure 3.4.: Behavioural model system architecture showing different I/O nodes.

Table 3.1.: Behavioural model input design parameters

Electrode	**Amplifiers**	**Filter**	**ADC**
Impedance	Gain	High-pass cut-off	Sampling Rate
Bandwidth	Bandwidth	Low-pass cut-off	Resolution
	IRN	Filter Type	$V_{\text{ref+}}$
		Filter Order	$V_{\text{ref-}}$
		Passband Ripple	
		Stopband Ripple	

The GUI has been designed to operate within a single window in four main panels, as presented in Figure 3.5.

- The *Node Select* panel shows a graphical representation of the model with check boxes for the user to define the desired input and output nodes. Based on the node selection, the corresponding blocks are activated to facilitate parameter input. No more than two nodes can be selected at the same time.
- The *Function Select* panel provides the user with the option to run the simulations, reset the system, input the data to be processed and save the signals at every (activated) node in the defined signal processing chain.
- The *Stage Select* panel allows the user to configure each activated front-end stage. Furthermore, amplifier and filter tabs have additional response plots, so that user can visually see the magnitude and frequency response of the corresponding blocks. In addition to the

[2] www.imperial.ac.uk/bioinspiredtechnology/research/neuralinterfaces/tools

front-end parameters, the user is also prompted to enter the original sampling frequency of the input signal.

- The *Node Display* panel displays time and frequency responses of both the input and output nodes, after each simulation.

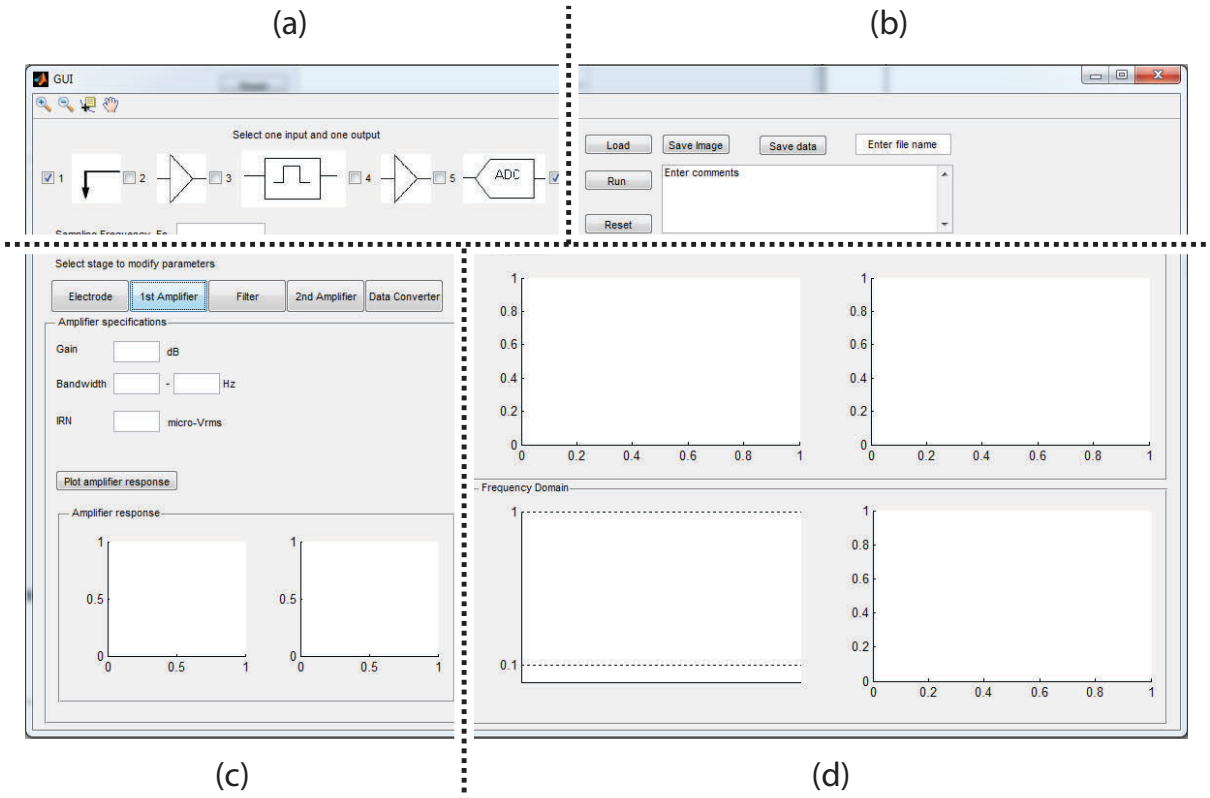


Figure 3.5.: Graphical user interface for the front-end behavioural model. Annotated are the four main panels: (a) Node Select; (b) Function Select; (c) Stage Select; and (d) Node Display.

3.4. Behavioural Model Validation

To demonstrate the validity the behavioural model proposed herein, a neural recording integrated circuit (IC) is used. The neural recording IC and the behavioural model are configured with the same parameters, and their frequency responses (as well as responses to different test stimulus) are compared. These comparisons are done for several different configurations and are further discussed in Section 3.4.2.

The implemented channel architecture is shown in Figure 3.6 including the front-end amplifier (FEA), the analogue signal processing, (i.e. filters), and the data converter. The total gain of the system is set to 65.8 dB for EAPs and 46 dB for LFPs, such that the input signal was amplified/mapped onto the input range of the data converter. To compare the real system

with the proposed model, the low and high pass filters have been implemented separately with individually tunable corner frequencies.

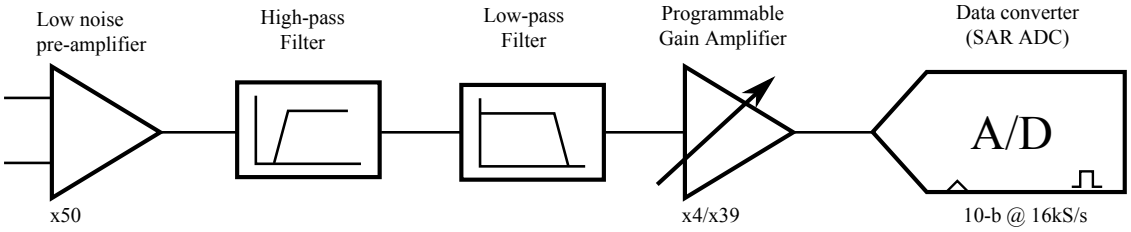


Figure 3.6.: Channel architecture of the front-end neural interface IC.

3.4.1. Circuit Implementation

The FEA is based on the established Harrison topology with a symmetric operational transconductance amplifier (OTA) [74], shown in Figure 3.7(a). A gain of 50 (33dB) is set by the capacitance ratio to avoid saturation and reduce the distortion by further filtering. The capacitors are implemented by an array of Metal-Insulator-Metal (MIM) capacitors with unit capacitance of 150 fF selected for good noise and matching performance [74]. The current consumption of the OTA is 3 μA with a 600/1.5 μm input differential pair to minimise flicker noise.

This is then followed by a second order high pass filter (HPF) based on a Bessel function for removing the LFP before further amplification (if only action potentials are required). This filter is realised using a gm-c topology arranged as 2$^{\text{nd}}$ order ladder configuration, shown in Fig. 3.7(b). To increase the dynamic range, bump linearisation was applied to the gm cell [158]. A second order low pass filter (LPF) is implemented in a similar manner with a cross coupled input differential pair to further increase the linearity, also shown in Fig. 3.7(b). The cut-off frequency of both the HPF and LPF is tunable by switching in capacitance, providing corner frequency settings at 120 Hz, 240 Hz, 300 Hz for HPF, and 3 kHz, 4 kHz, 6 kHz for LPF. Bypass switches are applied to both filters, to allow the inputs and outputs to be shorted thus bypassing the filtering. This is followed by a programmable gain amplifier based on capacitive feedback to further boost the gain, shown in Fig. 3.7(d). This uses a flyback capacitor configuration [17] to provide either a gain of 4 for LFPs, or 39 for EAPs. A 2-stage miller amplifier with open loop gain of 72dB, and gain-bandwidth (GBW) of 1 MHz is used to drive the ADC capacitive input. A standard charge redistribution Successive Approximation (SAR) analogue digital converter (ADC) with a 16 kHz sampling rate and 10-bit resolution is used (not detailed herein). The capacitor array is implemented by MIM capacitors with 33 fF unit capacitance and 9:1 split configuration [22] to reduce the total active area while maintaining good linearity. The specifications of the circuit used for comparison with the proposed model is summarised in Table3.2.

The chip microphotograph together with overlaid floorplan is shown in Fig. 3.8. This includes 16 recording channels with each channel occupying a footprint of 400 μm $\times$ 400 μm including all components. For test purposes, a single channel has been implemented separately and connected

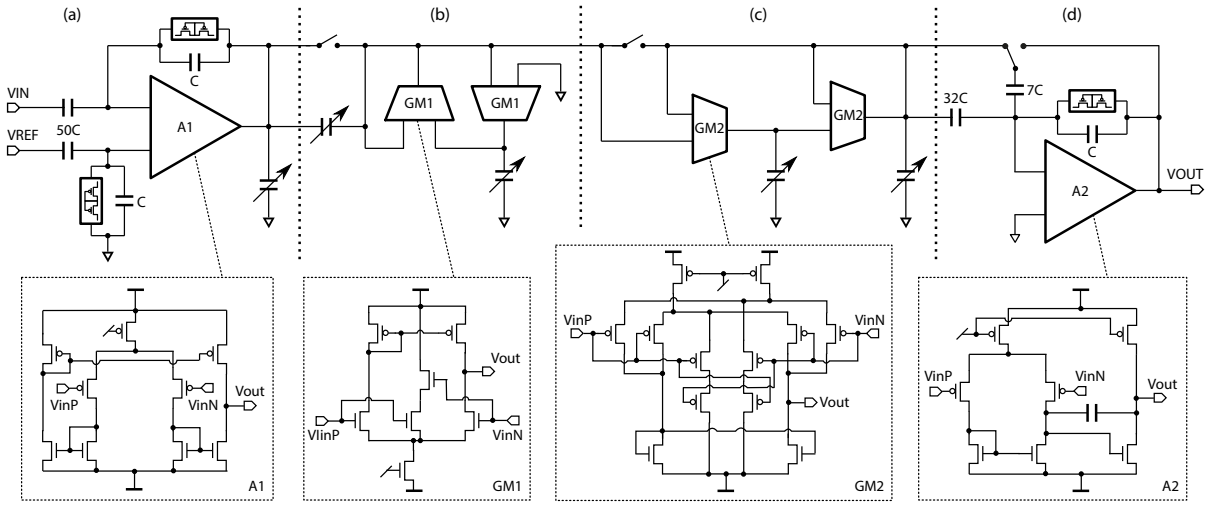


Figure 3.7.: Circuit implementation of the front-end neural interface. Schematics shown for: (a) low noise pre-amplifier; (b) high-pass filter; (c) low-pass filter; and (d) programmable gain amplifier that directly drives the SAR ADC (schematic not shown).

Table 3.2.: Technical specifications of the front-end neural interface IC

Parameter	Simulated	Measured
	Signal Conditioning	
Gain	66dB	62dB
Noise (300Hz-3kHz)	$4.2\,\mu V_{rms}$	$5\,\mu V_{rms}$
HP corner	$\{0.1, 180, 240, 320\}$Hz	$\{0.5, 190, 240, 360\}$Hz
LP corner	$\{3, 5, 6, 7\}$kHz	$\{3.2, 4.2, 5.1, 5.6\}$kHz
Power consumption	$9.1\,\mu$W	$10\,\mu$W
	Data Conversion	
Sampling frequency	16 kHz	15.5 kHz
ENOB	9.2b	9.1b
Power consumption	$3.4\,\mu$W	$5\,\mu$W

to a buffer to allow for direct analogue signal recording.

3.4.2. Comparison between integrated circuit measurements and behavioural results

As previously mentioned, in order to confirm the validity of the proposed model, the neural recording IC described in Section 3.4.1 and the behavioural model are configured with the same parameters, and their frequency responses are compared for various configurations.

Frequency responses of the behavioural model, as well as the simulated and measured frequency responses of the corresponding implemented system are presented in Figure 3.9. It should be noted that the implemented design uses capacitor and transconductance values derived directly from the model without accurate modification regarding to the parasitics and

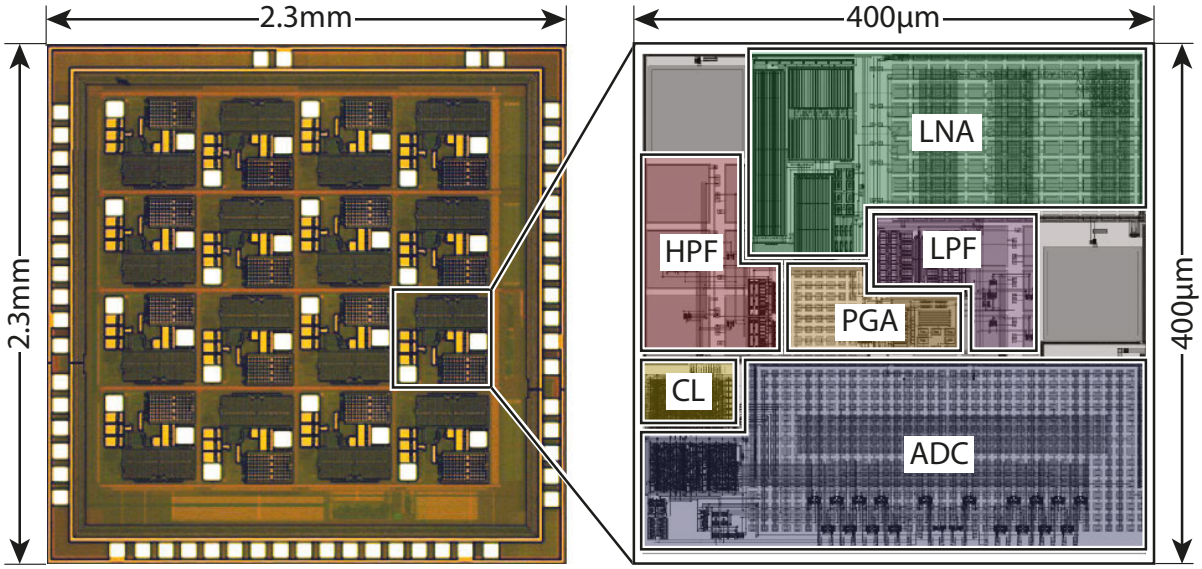


Figure 3.8.: Microphotograph of the circuit implemented in a 0.18 μm CMOS technology showing: (a) the entire 16-channel system; and (b) floorplan of a single recording channel (AFE).

non-linearity. In other words, a model oriented design approach rather than a circuit oriented approach has been followed. This helps the realistic validation and comparison of the behavioural model results.

As shown in Figures 3.9 and 3.10, the frequency responses of the behavioural model and the hardware system closely match, with small distortion around 3dB due to device mismatch and parasitic capacitances in cascading stages. These non-ideal influences depend on device sizes and circuit topologies, and are beyond the main scope of this work.

Figure 3.11 shows the time domain input and output from the neural IC and the front-end model for comparison, which reveal a close match between the two outputs. It should be noted that the measured data exhibits higher noise levels than modelled. Although the measured IRN level is $5\mu V_{rms}$ (hence within expectations), an estimated $18\mu V_{rms}$ is additionally introduced by test setup. More specifically, this noise is due to external signal generator circuits, which output attenuated datasets of different noise levels, and also include line frequency harmonics and environmental noise. Moreover, the neural recording IC implements a Bessel filter, while behavioural model uses a Butterworth filter configuration (since Bessel option is not available), thus additional mismatch between the results are introduced.

For further validation of the behavioural model, all neural test data (with different noise levels) are used as the input to both the integrated circuit and the proposed model. The system (as described previously) is configured as follows: gain = 66dB, high-pass f_c =210 Hz, low-pass f_c = 4.2 kHz. Spike detection and sorting performances of modelled and measured outputs are compared in Table 3.3 and Table 3.4, respectively. It should be noted that data presented in these tables represent the difference between the modelled and measured spike processing performance.

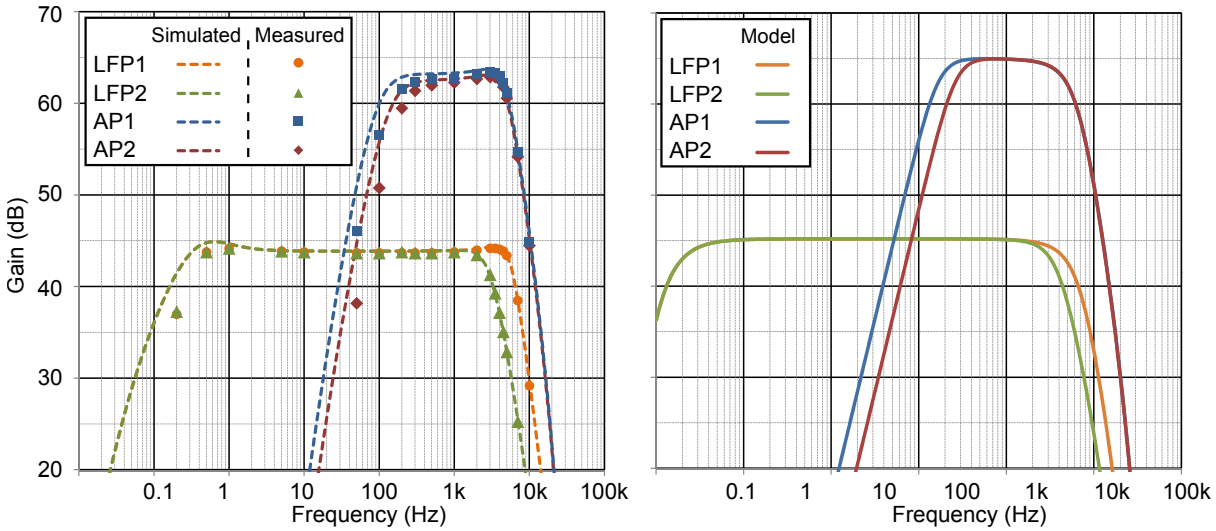


Figure 3.9.: Comparison of neural front-end IC and the behavioural model frequency responses at various configurations. (a) Measured (dashed lines) and simulated in a EDA tool (marks) frequency response of neural IC (b) Simulated frequency response of the behavioural model.

It can be seen from Table 3.3 and Table 3.4 that spike detection and sorting performances of the model and measured output closely match. Across all detection methods, datasets and different noise levels, the mean difference in spike detection is 1.34% with standard deviation of 4.97%. On the other hand, average sorting performance difference (Table 3.4) between the model and the measurement is 0.27% with standard deviation of 7.67%. However, it should be noted that there are a few exceptions in which larger differences compared to the majority of results are observed. These are due to additional noise introduced in the experimental setup as discussed above. These relatively larger differences are especially observed across several datasets for FSDE which has a higher sensitivity to white noise.

3.5. Behavioural Model Demonstration

Here, the AFE parameters are investigated (through the proposed behavioural model) in terms of their impact on subsequent spike processing (detection and sorting), as well as their impact on hardware. This is achieved by configuring the proposed behavioural model with the design parameters of relevant state-of-the art front-end architectures reported in literature.

3.5.1. State-of-the-art Front-Ends

Here the state-of-the-art front ends [12–14,16,18,19,21–24] are considered by applying their design parameters to the proposed model. Whenever possible, the measured system specifications such as gain, bandwidth, and effective number of bits were chosen as input parameters (instead of design targets). For any missing information regarding each stage, reasonable approximations were made. For example, for any system with missing electrode information, a typical

Table 3.3.: Spike Detection Performance difference between the modelled and measured output

Dataset	**SNR**	**Detection Difference %**		
		ST	AT	NEO
D1	High	11.11	7.94	28.57
	Med.	0	0	1.27
	Low	-1.39	-1.39	2.78
D2	High	0	-1.47	0
	Med.	-1.43	0	1.43
	Low	1.28	1.28	-1.28
D3	High	0	1.45	-5.80
	Med.	3.03	3.03	0
	Low	-2.90	-2.90	-1.45
D4	High	1.69	1.69	0
	Med.	1.47	1.47	-1.47
	Low	0	0	-1.33
D5	High	1.30	1.30	0
	Med.	-1.52	0	1.52
	Low	4.11	5.48	1.37

Table 3.4.: Spike Sorting Performance difference between the modelled and measured output

Dataset	**SNR**	**Sorting Difference %**		
		TM	PCA	FSDE
D1	High	0	0	11.11
	Med.	0	0	-1.27
	Low	0	1.39	13.89
D2	High	0	0	-8.82
	Med.	0	0	22.86
	Low	0	0	-24.36
D3	High	0	0	-4.35
	Med.	0	0	-1.52
	Low	1.45	0	0
D4	High	0	0	-1.69
	Med.	0	26.47	4.41
	Low	0	0	-2.67
D5	High	0	0	-5.19
	Med.	0	0	-16.67
	Low	0	1.37	-4.11

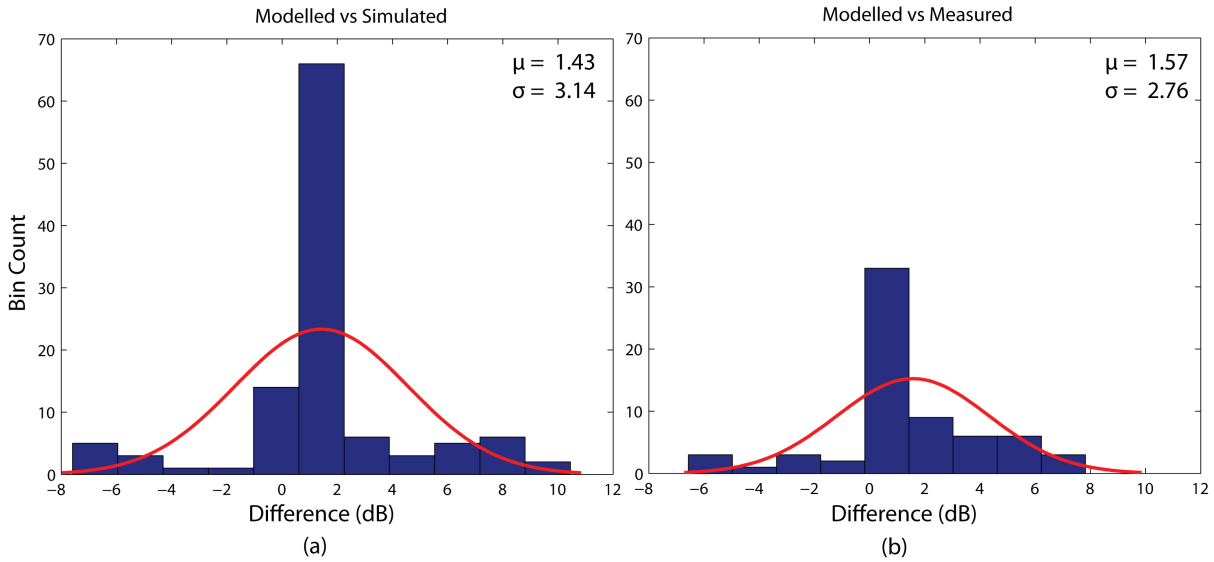


Figure 3.10.: Distribution of error between neural front-end IC and the behavioural model frequency responses at various configurations. (a) Difference between simulated frequency response of the behavioural model and frequency response of neural IC simulated in a EDA tool (b) Difference between simulated frequency response of the behavioural model and measured frequency response of neural IC. Both plots have been fitted with a normal distribution (red coloured line) and statistics of each distribution shown.

impedance value of 100 kΩ was assigned, with the electrode bandwidth assumed to cover the entire signal band.

For the behavioural model to accurately represent the front-ends being investigated, adjustments were required in order to ensure that the overall transfer function of the model matches that of the front-end architectures. Considering the fact that the presented model is a fixed five stage system and that the front-ends investigated have varying stages of amplification and filtering, not all of the parameters for each stage required by the model were available. Therefore, whenever the architecture differed from the expected (in the model), the parameters of interest had to be adjusted to have minimal impact on the overall transfer function. This typically only affected a number of the amplification and filtering stages, and involved adjusting gain, bandwidth, filter order and type, and input referred noise parameters. For any architecture with a "missing" stage, gain and IRN were always set to 0dB and $0\mu V_{rms}$, and filter was configured as a second order Butterworth filter. Most crucially, the bandwidth was set at the entire frequency band (i.e. half the sampling rate of the original input signal to the system) in order to ensure that the configuration of these missing stages had minimal impact, if any, on the input signal.

On the other hand, whenever the number of stages exceeded those specified in the model, parameters were re-distributed to ensure that the overall transfer function of the original circuit was accurately represented. For example, [23] consists of four bandpass amplifier/filter stages, which cannot be represented with the proposed model. However, this can be easily resolved by combining the two middle stages together as a 4^{th} order filter stage with (unity gain), and with

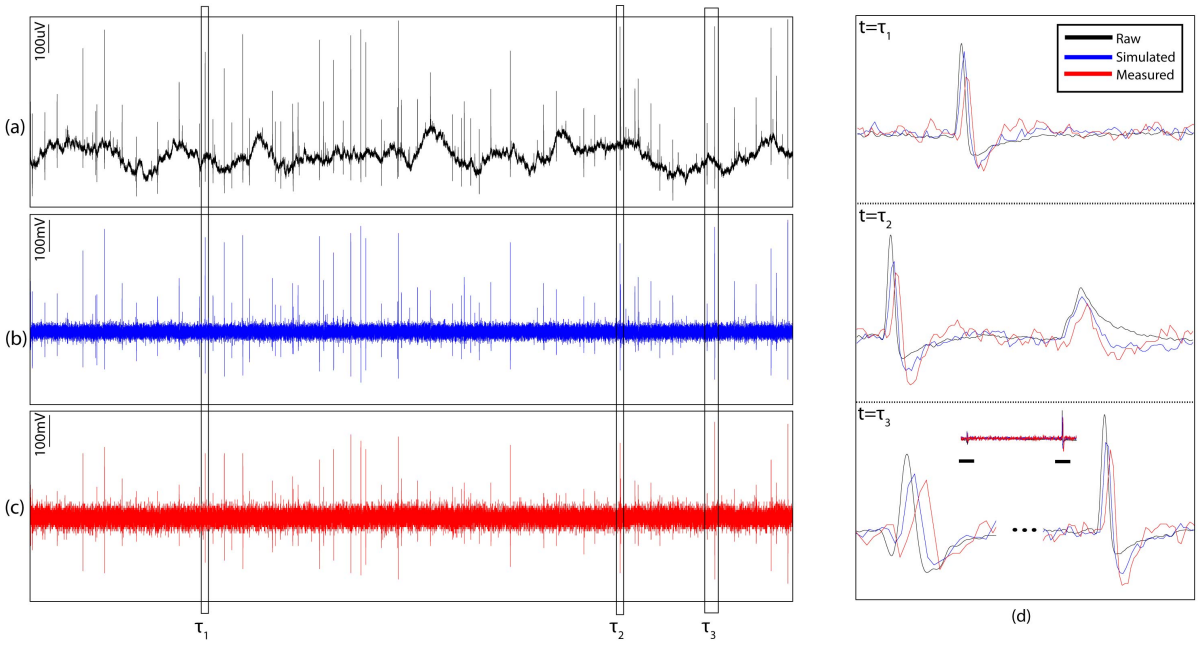


Figure 3.11.: Comparison of simulated (behavioural) and measured (chip) response to a test stimulus (spike waveform). Shown are: (a) raw neural input; (b) behavioural model output; (c) measured output; and (d) magnified sample spikes (with normalised amplitude and added time increment). Please note that for t=τ_3, smaller spike on the left have been scaled-up for illustrative purposes (relative scale is shown above the spikes).

the gain of the middle stages being transferred to the pre and post amplifiers such that the overall gain of the model is equivalent to that of the real circuit. The complete list of specifications (modified/redistributed where necessary to fit the model) is presented in Table 3.5.

Table 3.5.: Review of state-of-the-art neural interfaces showing key design parameters

			Electrode			**Pre-Amplifier**				**Filter**			**Post-Amplifier**				**Data Converter**				**Resource**			
Ref.	Vdd (V)	Tech (μm)	Z ($k\Omega$)	fc1 (Hz)	fc2 (Hz)	Gain (dB)	fc1 (Hz)	fc2 (kHz)	IRN (μV_{rms})	Type*, Order	fc1 (Hz)	fc2 (kHz)	Gain (dB)	fc1 (Hz)	fc2 (kHz)	IRN (μV_{rms})	Fs (kS/s)	Res. (bit)	Vref (V)	Vref (V)	Power (mW)	Area (mm^2)	Power/ch.‡ (μW)	Area/ch.‡ (mm^2)
[14]	3.3	0.5	400	0	fso/2	40	0	5	5.1	BW,2	300	fso/2	20	0	5	5.1	15	10	2.475	0	13.5	27.73	67.5	0.056
[16]	3.3	0.5	1000	0	fso/2	40	94	8.2	1.94	BW,2	0	fso/2	0	0	fso/2	0	16	7	0.125	-0.125	1.8	9	112.5	0.563
[18]	1.65	0.35	190	0	fso/2	40	100	10	4.9	BW,2	0	fso/2	20	0	fso/2	0	40	9	0.5	-0.5	6	63.36	33.5	0.449
[19]	0.8	0.13	100	0	fso/2	49	100	6.2	14	BW,2	0	fso/2	0	0	fso/2	0	80	8	1	0	0.02	1.54	20	0.080
[21]	1.4	0.18	1000	0	fso/2	40	1	100	11.2	BW,2	350	12	26	0	fso/2	0	31.25	7.5	1	0	0.325	9.92	10.1	0.310
[22]	1.2	0.13	100	0	fso/2	56	0.1	100	3	BW,2	280	10	0	0	fso/2	0	31.25	10	0.6	-0.6	6.5	25	68	0.26
[23]	3.3	0.35	100	0	fso/2	56	200	6	2.9	BW,4	200	6	20	0	fso/2	0	12.5	12**	1	-1	5.94	25.2	231	1.575
[24]	1.2	0.13	100	0	fso/2	47.5	167	6.9	2.84	BW,2	0	fso/2	18	0	fso/2	0	22.5	7.62	0.6	-0.6	0.0028	0.16	2.8	0.16
[12]†	3.3	N/A	100	0	fso/2	74	0	fso/2	1	BW,10	250	7.5k	0	0	fso/2	0	30	12	1	-1	N/A	N/A	N/A	N/A

*BW=Butterworth, **uses an off-the-shelf ADC, † only considered for accuracy comparisons (discrete implementation), ‡area/power per channel only taken for systems quoting these for AFE.

3.5.2. Spike Processing Accuracy and Hardware Requirements

Having extracted all relevant parameters and applied them to the proposed model, all datasets were processed with each front-end configuration. Figure 3.12 illustrates the output of each front-end and its effect on spike shapes. All outputs have been re-scaled for illustration purposes, i.e. to account for different gains and ADC resolutions.

As stated earlier, front-end performances were quantified in terms of the spike detection and sorting methods, and the overall accuracy was compared to the power and silicon area utilisation of each design. Tables 3.6 and 3.7 present the detection and sorting accuracies across all state-of-the-art front-end designs.

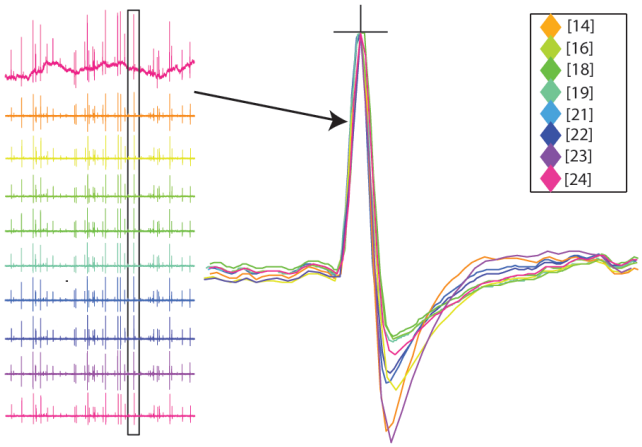


Figure 3.12.: Effect of analogue front-end on spike waveform. Shown is a test signal passed through all front-end configurations reviewed using the behavioural model developed. Spike outputs have been peak aligned and normalised for different gain and quantisation levels.

3.5.2.1. Spike Detection

It was observed that the relative accuracy of the front-ends were constant (across all methods), although the detection performance varied depending on the threshold method. Although the variation in spike detection performance is around 18% to 20% on average (depending on the detection method), it should also be noted that 88% of the AFEs had variations of 10% to 13% on average. Both simple and absolute threshold methods performed with 80% and above accuracy in general.

3.5.2.2. Spike Sorting

Spike sorting performance of front-ends showed significantly less variation when compared with spike detection performance. It can be observed that variations in performance for TM, PCA and FSDE were within 1.28%, 2.17% and 11.5% on average. More specifically, for TM and PCA the worst case variation amongst the front-end performances were 5.45% and 9.43%. It should

Table 3.6.: Spike Detection Performance of state-of-the-art front-end circuits

Dataset	**SNR**	**Detection Accuracy %**					
		ST		AT		NEO	
		mean	std.	mean	std.	mean	std.
D1	High	92.43	11.74	91.81	11.45	92.48	9.18
	Med.	90.67	16.72	90.11	16.50	86.60	15.75
	Low	82.13	22.91	82.28	23.01	69.60	22.23
D2	High	92.99	10.24	92.74	10.19	86.18	20.10
	Med.	88.01	18.21	87.88	18.20	78.28	25.16
	Low	72.58	28.99	72.44	29.04	58.24	28.37
D3	High	92.63	10.73	92.33	10.61	85.26	13.06
	Med.	89.80	17.72	89.50	17.72	77.79	14.53
	Low	77.16	28.17	76.87	28.20	58.33	17.29
D4	High	95.92	7.38	95.67	7.29	95.28	7.67
	Med.	91.46	17.66	91.18	17.58	91.05	13.91
	Low	83.33	30.06	83.17	30	74.12	23.57
D5	High	88.58	21.49	88.04	21.34	78.89	22.97
	Med.	82.31	29.26	82.04	29.14	66.16	25.83
	Low	67.44	32.55	67.35	32.61	44.55	22.67

be noted again that these spike sorting results do not include the spike detection and since the "ground truth" to the datasets are known the spike detection accuracy is essentially 100%.

Table 3.7.: Spike Sorting Performance of state-of-the-art front-end circuits

Dataset	**SNR**	**Sorting Accuracy %**					
		TM		PCA		FSDE	
		mean	std.	mean	std.	mean	std.
D1	High	99.59	0.10	99.47	0.32	85.87	8.65
	Med.	99.54	0.11	99.43	0.38	79.86	8.34
	Low	98.49	1.14	97.77	2.96	71.86	10.40
D2	High	99.59	0.15	99.56	0.26	65.47	20.00
	Med.	99.05	1.13	99.15	1.08	51.60	12.19
	Low	97.07	3.51	97.48	3.07	47.41	12.65
D3	High	99.36	0.22	99.35	0.20	78.24	9.52
	Med.	99.19	0.68	99.44	0.53	74.00	8.24
	Low	96.05	2.74	97.40	2.90	66.30	10.54
D4	High	99.53	0.15	96.16	9.43	86.43	10.40
	Med.	99.36	0.23	98.87	0.79	81.44	13.38
	Low	98.11	1.72	97.45	2.80	66.89	15.20
D5	High	99.40	0.40	99.30	0.42	63.21	15.99
	Med.	98.16	1.50	98.32	1.57	54.15	7.45
	Low	94.63	5.45	94.83	5.84	53.59	9.31

3.5.2.3. Power and Silicon Area Requirements

When considering the spike processing performance, i.e. both detection and sorting, the variation in accuracy between different front-end interfaces was relatively low. However, these minor differences were accompanied by much larger differences in power and silicon area specifications. Although some designs achieve higher specifications (in terms of individual component performance), their overall spike processing performances barely exceed others. It is therefore common that individual component specifications are over-engineered due to the fact that impact on system performance is unknown. The choice of the FE parameters is thus a critical element of the design that can be informed using a behavioural model.

More specifically, when sorting performances are compared (Figure 3.13) the spread of the majority of works fall within 5% of each other while there exists orders of magnitude difference in power and silicon area per channel. On the other hand, when detection results (Figure 3.14) are compared, one can observe again that among the works with similar performances, there exists large differences in silicon area and power. Note these illustrate the absolute power and silicon requirements and does not take the effect of technology scaling into account. For example, transistor area generally scales with feature size and dynamic power consumption scales proportionally to Vdd^2. However, as the ratios of static to dynamic power and passive to active device area are generally not reported, a more general technology-independent FOM cannot be established.

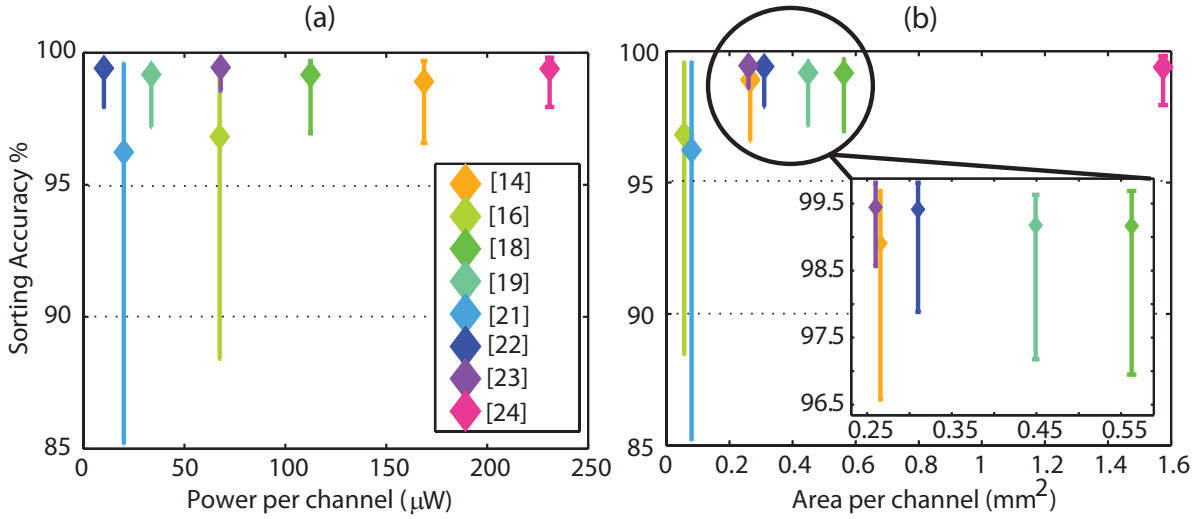


Figure 3.13.: Spike sorting performance plotted against hardware resource requirements for all front-ends. Shown are: (a) Power consumption per channel; and (b) Silicon area per channel, for a single spike sorting method (across all test datasets at all noise levels)

.

The results, however, clearly demonstrates that it is possible to achieve good detection/accuracy while making further savings by not over-designing some aspects of the front-end. The proposed tool presented herein therefore has more impact in minimising resource requirements

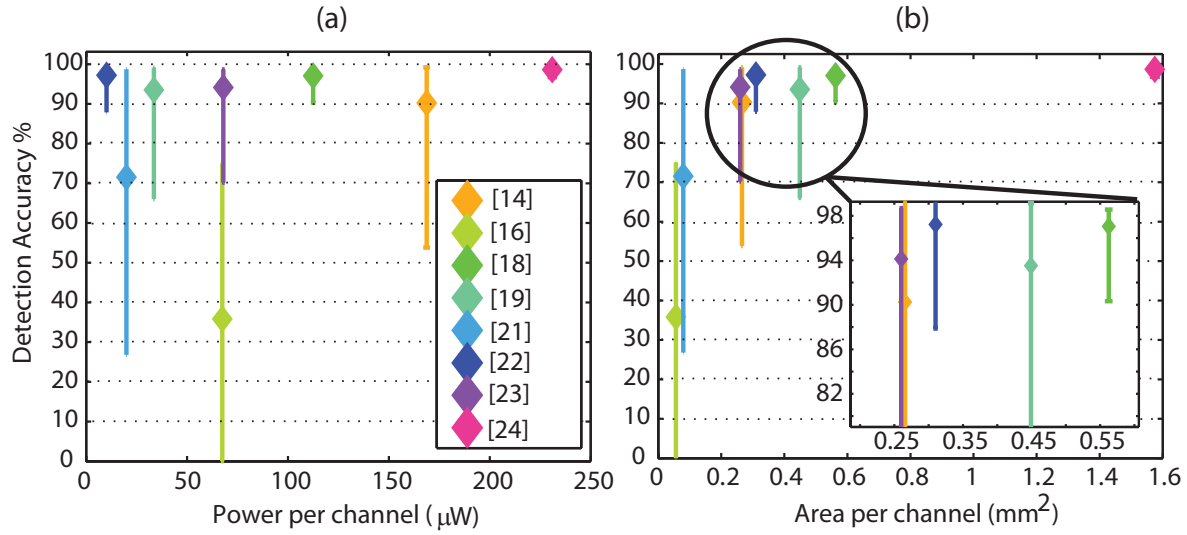


Figure 3.14.: Spike detection performance plotted against hardware resource requirements for all front-end configurations. Shown are: (a)Power consumption per channel; and (b) Silicon area per channel, for a single spike sorting method (across all test datasets at all noise levels)

.

rather than maximising performance. This essentially helps designers to make hardware efficient design choices that do not significantly degrade spike processing.

3.6. Parametric Design Optimisation

Previous sections have demonstrated the initial hypothesis that the front-ends are often over engineered by not taking into account their impact on subsequent spike processing during design. Therefore, it is possible to relax design specifications such that minimum hardware resource utilisation is achieved with minimum trade-off in spike processing accuracy. Having developed and verified a behavioural front-end model to achieve such optimisation, it is thus the aim of this section to quantify the sensitivity of spike sorting to parameters relating to each stage of the AFE.

These parameters are: (1) Gain-Bandwidth Product and input-referred noise of the low noise pre-amplifier (i.e. SNR), (2) filter type, order, and low-/high-pass cut-offs of the bandpass filter, and (3) resolution and sampling rate of the ADC. The results are tested against three different spike sorting algorithms. It should be noted, however, that the following analysis only reveals single dimensional trends for each AFE stage whereas the front-end optimisation is essentially a multi-dimensional problem. Therefore, the resulting set of parameters are near-optimal.

3.6.1. Methodology

The optimisation is achieved by investigating the effect of each design parameter (i.e. those relevant to the front-end) on spike sorting performance within a simulation environment using

Mathworks Matlab^{TM} R2011b v7.13. The effect of each identified front-end parameter is tested by three spike sorting methods with 24 data sets based on simulated extra-cellular recordings.

3.6.1.1. Spike Sorting

For the results reported herein, the spike sorting performance is quantified using the effective accuracy, calculated by: % total spikes classified × % spikes correctly classified (excluding spike detection). This has been benchmarked for 3 different sorting methods. One of the methods, Principle Component Analysis (PCA) is a commonly used and benchmarked method, while Template Matching and Derivative Features are relatively computationally-efficient methods for spike sorting.

- Template Matching (TM): Aligning the maximum peak of the signal with a template and using the Squared Euclidean Distance as a similarity/distance measure. The templates are created by taking the mean of the spikes (within each cluster), aligned to their individual maximum peaks.
- Principle Component Analysis (PCA): Taking the first two principle components (for each spike) and using k-means for clustering (50 iterations). Uses the Matlab^{TM} in-built functions (*princomp* and *kmeans*).
- 2^{nd} Derivative Features (DER): Taking the minimum and maximum values of the 2^{nd} derivative (within each spike) and using k-means for clustering (50 iterations). Uses the Matlab^{TM} in-built functions (*gradient* and *kmeans*) [143].

3.6.1.2. Test Datasets

The methods were applied to a total of 24 datasets based on simulated extracellular recordings [77, 144]. These contain 6 different groups (each using 3 single units). Each group is then used to generate 4 datasets at varying SNR levels (26, 20, 16.5, and 14 dB). For further details on dataset generation, please refer to Appendix B.1.

3.6.2. Design Optimisation

The front-end design parameters tested are: (1) SNR, (2) high-pass response, (3) low-pass response, (4) resolution, and (5) sampling rate. Each test is repeated on all 6 dataset groups with results showing the mean and spread. In all tests (with exception of high-pass response), the LFPs have been removed using a high order, zero-phase filter. In tests not involving noise variations, an SNR of 20 dB is used as it is representative of the trends observed.

3.6.2.1. Signal-to-Noise Ratio

The overall effect of SNR on spike sorting accuracy is tested for all 6 dataset groups at the 4 SNR levels (as described previously). The results (Figure 3.15) show that by reducing the SNR

(over the range 14−26 dB), the spike sorting accuracy decreases by up to 30% (depending on spike sorting method used). The results also show that the algorithm based on second derivative features has the highest noise immunity, with template matching being the most sensitive to noise.

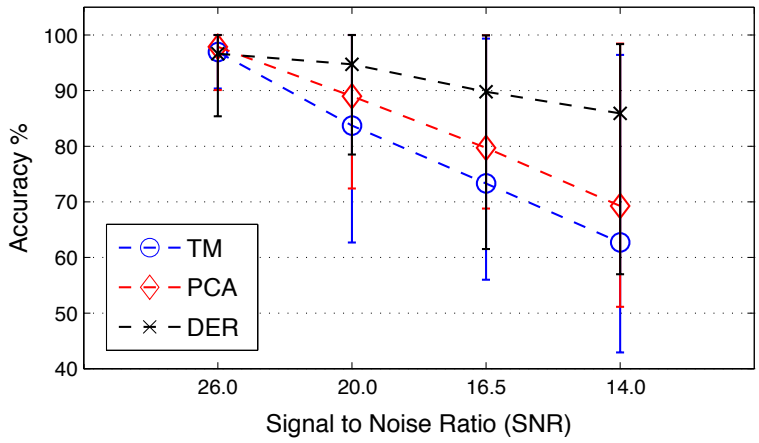


Figure 3.15.: Effect of SNR on spike sorting accuracy of the overall system (mean and spread shown over all data sets).

3.6.2.2. High-pass Response

This test examines the effect of the high-pass filter response on spike sorting accuracy. Key parameters in the high-pass filter design are: filter type, order and cut-off frequency (i.e. f_{3dB}). To emulate the effect of LFPs, these were extracted (200 Hz low-pass filtered and normalised for a 5:1 LFP to EAP amplitude ratio) from a hippocampus data set from the CRCNS database and superimposed to datasets. These were tested by pre-filtering the datasets using 2^{nd} and 4^{th} order Elliptic, Butterworth and Bessel filter types at f_{3dB}={300,400,500,600,700}Hz.

The results (Figure 3.16) reveal several trends. When the results of Butterworth and Bessel filters are investigated for TM and PCA, it is observed that spike sorting performance improves by ∼7% for both PCA and TM (both filters). As expected, this is mainly due to rejection LFPs and noise. Although similar improvements are observed for elliptic filter, performance of all spike sorting methods degrade beyond 500 Hz. Reduction in performance for higher cut-offs frequencies can be traced to phase distortion introduced due to analogue filtering. Figure 3.17 illustrates the effect of phase distortion introduced by a realistic analogue HPF. Here, it can be observed that the distortion increases with increasing cut-off frequency. This particularly impacts the spike alignment, hence reduces the sorting accuracy. This effect is particularly increased for higher order filters.

Although 500 Hz strike a good balance between rejecting LFPs (and background noise) and phase distortion effects for elliptic (for TM and PCA), for the remaining filter types the frequency range tested does not reveal such trends. On the other hand, the effects of filtering on DER is negligible (except for ≥500 Elliptic). This is attributed to the fact that differenti-

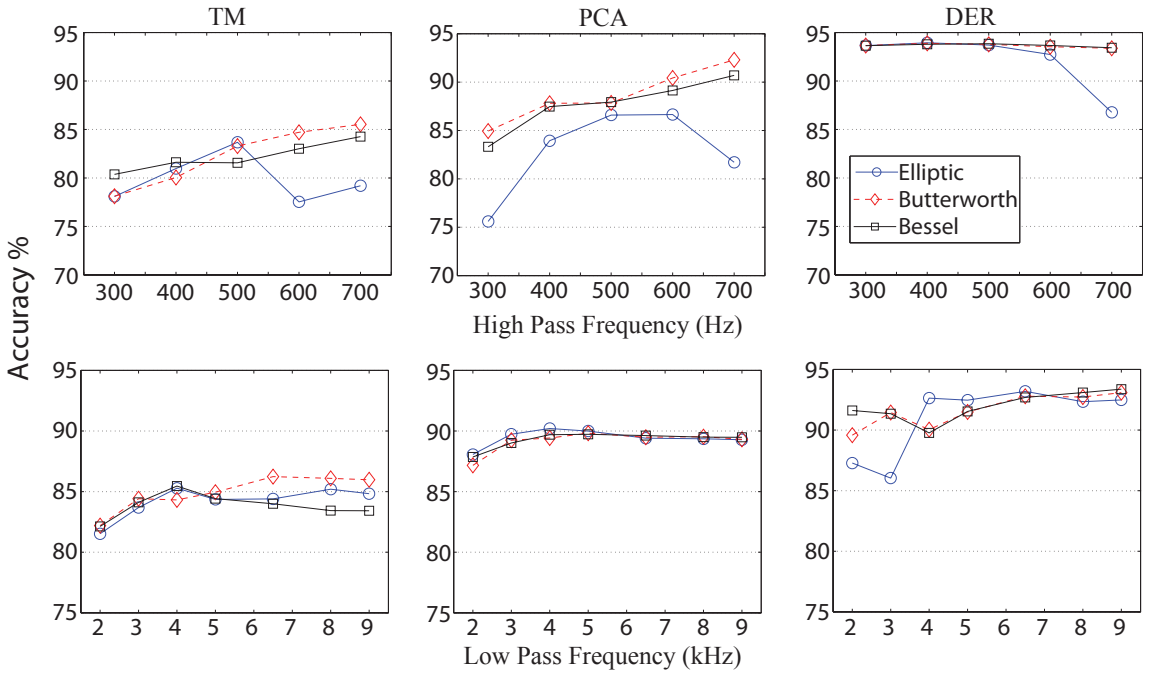


Figure 3.16.: Effect of high pass (top) and low pass (bottom) filtering on spike sorting accuracy (mean over all datasets for a noise level of 0.1, i.e. SNR = 20 dB). This is shown for 2^{nd} order Elliptic, Butterworth and Bessel filters.

ation can be considered as applying a filter where the higher frequency components are more emphasized, hence the low frequency noise sources such as background activity and LFPs are attenuated (compared to higher frequency components).

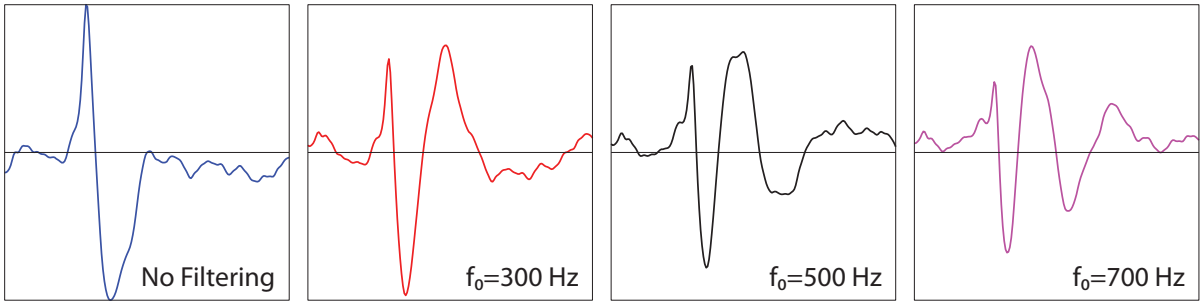


Figure 3.17.: The effect of a 4^{th} order high-pass Butterworth filter on the original spike shape.

3.6.2.3. Low-pass Response

This test examines the effect of the low-pass filter response on spike sorting accuracy. Key parameters in the low-pass filter design are as those previously identified for the high-pass. Again, these were tested by pre-filtering the datasets (using SNR=20 dB) using 2^{nd} order Elliptic, Butterworth and Bessel filter types at f_{3dB}={2,3,4,5,6.5,8,9,10,11.5}kHz. The results (Figure 3.16) show that increasing the low pass cut-off improves the spike sorting accuracy as

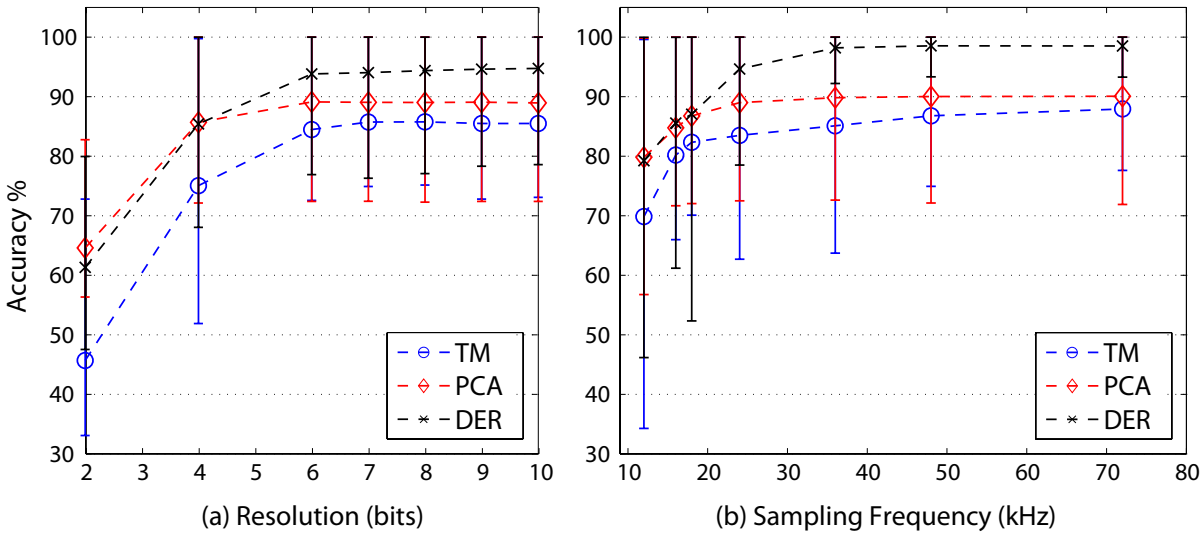


Figure 3.18.: Effect of data conversion (a) resolution, and (b) sampling rate on spike sorting accuracy (mean and spread shown over all datasets for a noise level of 0.1, i.e. SNR = 20 dB).

no phase distortion is introduced. However, it can be observed that beyond 4-6 kHz, there is moderate improvement in performance. Also the filter type appears to have negligible impact on performance (1.5% difference between types). In both low- and high-pass filtering, 2^{nd} vs. 4^{th} order filter implementations yielded negligible accuracy differences.

3.6.2.4. Sampling Rate

This test examines the effect of data converter sampling rate on spike sorting accuracy. The results are shown in Figure 3.18(b). It can be observed that increasing the sampling rate can significantly increase spike sorting accuracy, however saturating beyond 24 kHz (for all sorting methods tested). Increasing sampling rate is expected to improve performance, since this captures finer features which may help further differentiate the signals.

3.6.2.5. Resolution

This test examines the effect of data converter resolution (i.e. quantisation) on spike sorting accuracy (with scaling of the dataset to optimally fill the input range of the converter). The results are shown in Figure 3.18(a). It can be observed that although the performance does increase with resolution, it saturates beyond 5−6 bit resolution. This maximum useful resolution is ultimately limited by the SNR. However, since the amplitude of the observed EAPs can vary, typically, by one order of magnitude, additional resolution is needed (i.e. 2−3 bit), given the fixed amplification gain.

3.6.2.6. Hardware Resource

Hardware resource requirements rely on key parameters that include power and area. Table 3.8 reviews recently reported state-of-the-art neural interfaces, listing the key design parameters and achieved specifications for the pre-amplifier, filter and ADC (using reported data). In general for the pre-amplifier and data converter, the resource requirements are approximately directly proportional to the specifications. Specifically, the noise/gain for the pre-amplifier, and the resolution/sampling rate for the data converter are proportional to power consumption and area (depending upon technology used). The resource requirements for the filter are however more variable, due to the number of different applicable design methodologies. The filter order is typically the only parameter that is directly related to resource requirements.

Table 3.8.: Review of state-of-the-art neural interfaces comparing key design parameters and resource utilisation

		Amplifier				Filter			Data Converter			
Ref.	Tech. (μm)	Noise (μV_{rms})	Gain (dB)	Area (mm^2)	Power (μW)	Order, Type	Low-pass (kHz)	High-pass (Hz)	ENOB (bits)	Samp. Rate (kS/s)	Area (mm^2)	Power (μW)
[14]	0.5	5.1	60	0.16	42	1, G_mC	5	300, 800	10	15	0.15‡	1‡
[15]	0.13	1.95	38.3	-	12.5	-	-	-	8	10−100	0.27†	1
[17]	0.35	2.5	39.2	0.4†	0.03	1, G_mC	0.03−0.3	0.005−3.6	12	-	0.3†	0.2
[18]	0.35	4.9	40	0.07†	6.6	-	2−20	0.1−200	9	640	0.6†	-
[19]	0.13	14	40	0.3†	0.6	-	-	-	8	-	0.1†	-
[20]	0.35	6.1	54−73	-	12.8	-	5	0.5−50	8	111	-	2.8
[21]	0.18	5.4	49−66	0.03	16.7	G_mC	11.7	350*	8	125	0.02	1.9
[22]	0.13	2.2	-	-	-	2, G_mC + SC	10	280	10	31.3	0.3	1.1
[23]	0.35	2.3−2.9	40−75	0.9	371‡	4	0.2−6.2	2.6−572	-	-	-	-
[24]	0.13	3.8	47.5	0.06†	1.9	-	4.8−9.8	11.5-167	8	22.5−90	0.05†	0.5−1.8

Notes: †area estimated from microphotograph, ‡reported for multiple channels (value scaled per channel)

3.7. Conclusion

With next generation neural interfaces targeting hundreds to thousands of channels, the power and silicon area budgets on the front-end electronics are becoming increasingly stringent. While the demand for spike detection and sorting performance enforces minimum front-end requirements, the limited power and silicon area resources, in addition to fundamental limitations posed (such as maximum power dissipation not to damage neural tissue), necessitate careful design of front-end specifications as well.

In Section 3.2 a behavioural front-end model is developed to investigate its impact on back-end processing. Based on the proposed model, a behavioural optimisation tool is further proposed with which the designer can investigate the effect of different topologies during design time so that a good balance between resource efficiency and sorting performance can be achieved.

The developed behavioural model and optimisation tool was validated by comparing the predictions of the model with a real front-end implementation. This is achieved by: (1) applying a test stimulus to a hardware platform to compare the frequency responses at different configurations, and (2) comparing the spike processing (i.e. detection and sorting) performance for different datasets at varying noise levels. Following verification, the optimisation tool is then used to demonstrate the effect of the Analogue Front-End (AFE) on subsequent spike processing (detection and sorting) by testing established methods on a selection of different state-of-the-art configurations. The impact of FE parameters have been discussed in terms of spike processing performance and hardware requirements.

The reported results show that while the variation in the observed spike processing accuracy between different front-end interfaces is relatively low and comparable (10-15%), the designs show significant spread in specifications for individual components which translate into the large deviations in power and silicon area requirements (up to orders of magnitude). In other words, despite some designs achieving higher specifications, their overall spike processing performance barely exceed each other. Hence, by not over-engineering some aspects of the front-end, power and silicon area can be minimised while maintaining the spike processing performance. The proposed behavioural model provides the designer a platform to investigate the effects of different parameters in a fast way. Thus, a good balance between resource efficiency and performance can be achieved during design time.

The critical parameters associated with front-end electronics are identified as: (1) Gain-Bandwidth and noise for the amplifier, (2) Filter order, type, and cut-off frequencies (both high and low pass) for filtering stage, and, (3) sampling frequency and resolution for analogue-to-digital conversion stage. In Section 3.6, these parameters have been investigated over 24 datasets using three different sorting methods (Principle Component Analysis, Template Matching and Second Derivative Features). The trends observed for investigated parameters have been consistent across all tested sorting methods.

The results of the parametric optimisation have revealed that a high-pass frequency of 500 Hz seems to be a good balance between rejecting LFPs and phase distortion effects (caused by

analogue filtering) for elliptic filter; however, Bessel and Butterworth filters do not reveal such trends over tested frequency range due to their better group delay (hence less phase distortion). In addition, it is revealed that derivative features attenuate low-pass frequency components (hence LFPs and background activity) and thus the analogue filter parameters can be relaxed for this method.

On the other hand, low pass results reveal a marginal effect on the results suggesting 5-8 kHz low pass cut-off would suffice. Filter type and order has shown marginal differences across all datasets and sorting methods, therefore a second order filter does not compromise performance and is more resource efficient. When ADC parameters are investigated, it can be seen that the sorting performance saturates beyond 24 kHz for sampling frequency and 4-6 (6-8 with added resolution for variations in amplitude for fixed amplification gain) bits for resolution. It should be noted once again that the analysis for parametric optimisation provides single dimensional trends for each parameter. Since the front-end optimisation is a multi-dimensional problem, the provided set of results can only be taken as near optimal solutions. Nevertheless, the provided behavioural model and tool provides the user a platform for pursuing such a holistic optimisation approach to find an optimum set of solutions.

4. Minimum Requirements for Template-based Real-time On-Node Spike Sorting

Real-time on-node spike sorting, which reduces the amount of data to be transmitted to binary events indicating the firing of the neurons, is the most effective data reduction strategy without sacrificing information quality prior to transmission in neural recording systems. However, as previously mentioned, successful implementation of this strategy relies on realising on-node spike sorting in an efficient manner such that power savings from wireless transmission (due to data reduction) is greater than power consumed for spike sorting. However, considering the inherent complexity of the spike sorting, this is not a trivial task. This chapter focuses on template matching within the context of on-node real-time spike sorting.

Template matching, as a spike classification method, have been a widely utilised technique in many off-line spike sorting algorithms (see Section 4.1 for more detail). This has been primarily attributed to its versatility — any clustering method can be used to create templates — and its comparable performance to other techniques while being relatively computationally efficient [110, 111, 115].

Despite its merits, being not resource efficient enough is one of the major criticism towards template matching. The primary factor in such argument relies on the fact that the templates (i.e. representative waveforms) of the recorded neurons need to be stored on-node, hence significantly increasing memory requirements. On the other hand, it may also be argued that operating on the complete neural waveform (through methods such as Euclidean Distance) is expensive in terms of computational requirements. However, there have been no previous study, to the best of my knowledge, through which template matching have been systematically investigated to:

1. identify template matching related parameters and quantify the sensitivity of template matching accuracy to these.
2. quantify the sensitivity of template matching accuracy to AFE parameters.
3. identify how robust and real-time operation can be achieved using template matching (i.e. implementation strategy).
4. test the feasibility and establish minimum requirements for this implementation strategy to achieve such resource efficient implementation.

Following Chapter 3 which investigated the parameters of the front-end and established its trade-offs on sorting performance and hardware resources, Section 4.1 extends this study into template matching specific parameters. In addition to front-end parameters, on-node resource utilisation is also directly linked with the power consumption and area of the back-end digital electronics. Particularly, any operations requiring multiplication and/or division significantly contribute to power consumption. Any storage required for signal waveforms or features will contribute to overall area utilised. Primary method of interest being template matching, it is therefore the aim of Section 4.1 to investigate effects and establish the trade-off of back-end processing parameters (e.g. matching metric, template creation, interpolation etc.) on resource utilisation and sorting performance.

Section 4.2[1], on the other hand, tests (computationally) the feasibility of a two-stage hybrid strategy for robust and hardware-efficient real-time sorting implementation. The first stage, "template building", involves the implant wirelessly streaming detected spikes to an external computer to perform computationally intensive steps of spike sorting (i.e. defining templates by feature extraction and clustering). The spike templates (for each channel) are then sent back to the implant. During the second stage, "template matching", the detected spikes are compared with these templates and assigned to the one with the least distance (classification). Considering the low complexity and robustness of template matching, this approach provides low-power processing while preserving the quality of the signal.

To the best of my knowledge, there is only one previous study (pre-dating this work) which proposed real-time spike sorting using downloaded templates following a training session [159]. The proposed a 96-channel implantable platform wirelessly stream detected spikes and perform real-time template matching in an external device. However, one major drawback of the implementation (as argued by the authors in that study) is that the power consumption of chip is far from ideal. Hence, the device does not have enough resources to perform template matching inside the body. Section 4.2, therefore, investigates the extent which the specifications of a hypothetical chip (similar to [159]) can be reduced without degrading performance significantly.

4.1. Template Matching

As discussed previously, low-power hardware architectures make use of a mixed-signal approach in order to balance computation of analogue and digital signal processing [160]. System described in Figure 4.1 is a typical configuration of such an approach, where front-end analogue circuits amplify, filter and sample the incoming spikes, while back-end is a digital signal processing unit. At the front end, the specifications of the analogue circuitry directly affects area and power requirements, forcing many trade-offs [142]. Therefore, in Sections 3.6 and 3.2 each stage of the front-end have been investigated in order to optimise the circuits so to achieve low power consumption with high channel count.

[1]The work presented in Section 4.2 was done in collaboration with Dr. Joaquin Navajas. We have co-designed the test methodology of the hybrid sorting strategy, and I have contributed to real-time template matching investigations.

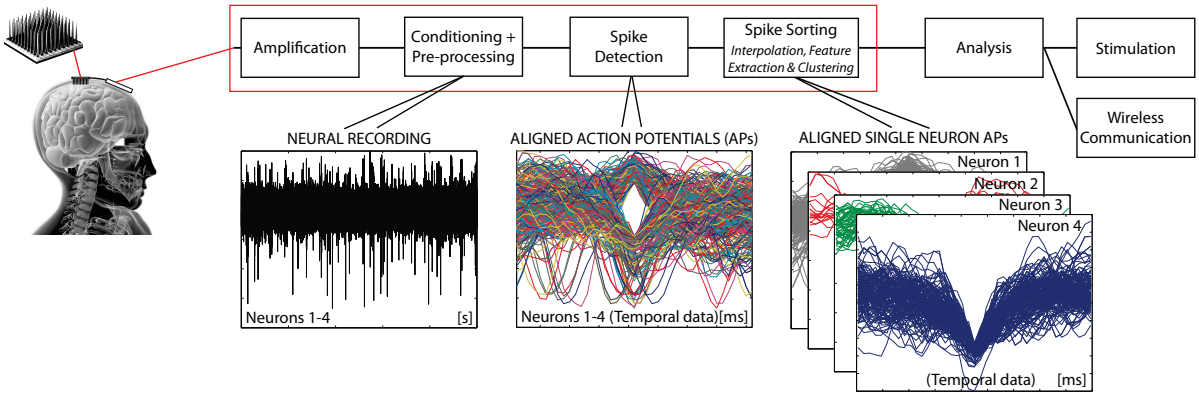


Figure 4.1.: A typical hardware-based spike sorting system encompassing signal amplification and conditioning, spike sorting (including feature extraction) and any subsequent processing (either offline for analysis or online for rehabilitation).

Although many feature-based spike sorting (including wavelets [119, 161], integral transform, discrete derivatives, blind source separation [162]) and clustering methods (k-means [163], superparamagnetic [77] and many more) exist [104], less aimed to use the raw neural time-domain signal as the base template in the context of hardware feasibility. [162–165].

Template matching was initially used in the early 1980s, where templates were automatically generated based on feature-clustering [92, 114, 166]. Templates can then be generated from the mean of these clusters [92, 166] and using standard comparative measures such as Euclidean distance [110, 159, 167]. From the discussion of the various systems proposed, it is observed that template matching have been employed either alone or with another method to achieve spike sorting. This was attributed to its consistency [110, 115] and comparable performance to other techniques [111], including traditional PCA [166]. Although it has been suggested that template matching does not perform well in noisy conditions [167] and may be computationally expensive [110], these can be ameliorated by using signal processing techniques to cope with noise, and by incorporating additional techniques (such as in [110]) to reduce computational complexity. Moreover, performance of template matching can be dependent on many key parameters (such as metric used for matching), hence its computational requirements can be further be minimised with new methods and careful design considerations. Therefore, template matching can offer a unique solution due to its accuracy and computational feasibility. Specifically, templates generated using offline and complex but highly accurate methods can be fed back into a hardware template matching system with a potentially better balance of this trade-off. This is assuming that template matching can achieve comparable accuracy compared to state-of-the-art implementations. In fact, some attempts have been made at integrating template matching into such on-line systems [168].

In addition to encouraging suggestions from literature on accuracy, consistency and versatility of template matching, work presented in Sections 3.6 and 3.2 support the fact that template matching performs as well as PCA and has relatively low computational cost. However, to the best of my knowledge, to date; there have been no studies to identify and quantify the sensi-

tivity of such key parameters for template matching regarding sorting accuracy and hardware implementability. In that regard, work presented in this section attempts to establish and quantify these parameters such as matching quantifiers, alignment, template creation, and template window size.

4.1.1. Methods

This section identifies the quantifiers and methods that are relevant to template matching. These are template matching methods, alignment methods, clustering, statistical measures, bandwidth, resolution and noise.

4.1.1.1. Alignment

For template matching systems, it is crucial to align the signals to be compared, to a common reference so as to improve matching and clustering. This is typically done by defining a fiducial marker such as peak, and aligning the incoming signals to this marker [77]. Therefore, in order to improve the alignment, thus the accuracy, numerous methods have been proposed. One such is interpolating the neural spike to an effective higher sampling frequency to enhance peak point detection [169, 170].

The following are the interpolation methods investigated in the preliminary work:

- *Linear*: Uses a linear equation, $y_i = y_0 + (t_i - t_0)\frac{y_1 - y_0}{t_1 - t_0}$, where t_0 & y_0 and t_1 & y_1 are the points being interpolated between and t_i & y_i is the new interpolation point. It can be translated into an interpolation filter.
- *Quadratic*: Has a quadratic as the representation equation. This method produces a more realistic curve.
- *Cubic Spline*: Utilises a 3^{rd} order polynomial interpolant with smoothness criteria to ensure continuity to the second derivative of the interpolating function. To do so, a triangular matrix with predetermined end point criteria needs to be constructed.
- 5^{th} *Order Cubic*: This method uses a 5^{th} order polynomial, and ensures smoothness to the second derivative.
- *Sinc*: This method is based on the Nyquist-Shannon sampling theory that any signal can be reconstructed from *sinc* basis functions, $s(t) = \sum_{-\infty}^{\infty} a_n sinc(t - nT)$. In the simulations, this interpolation is implemented by taking the Fourier Transform of the signal, and applying inverse Fourier Transform to obtain the signal back with more sample points

Please note that the reference point used to align the spikes is the maximum peak of the signal. However, other markers could also be used and will be discussed in the following sections.

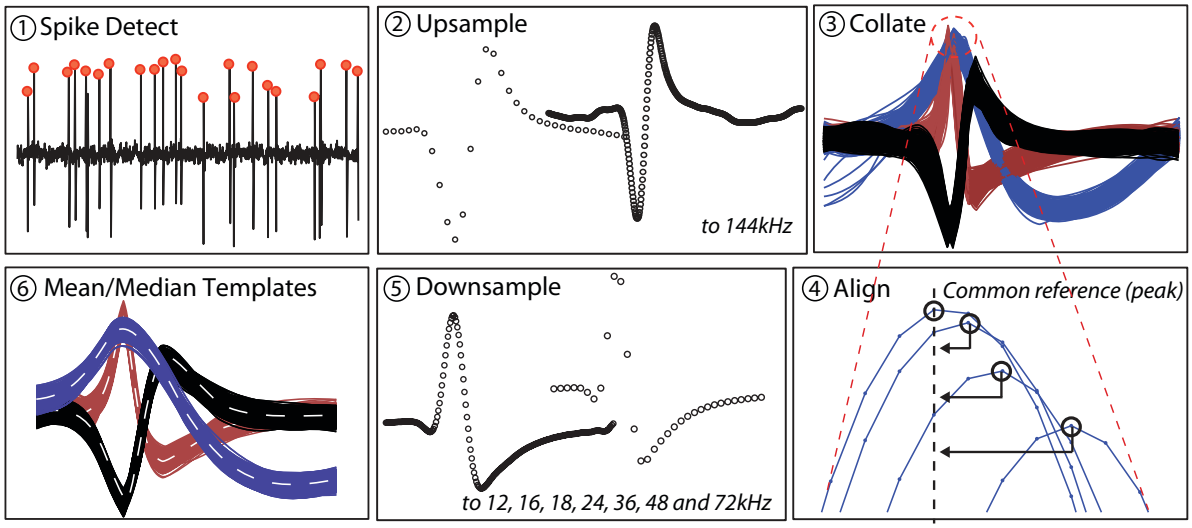


Figure 4.2.: Process of creating a template based on mean and/or median of a known cluster: (1) spikes are extracted; (2) upsampled using interpolating to $144kHz$; (3) separated into clusters using *a priori* information; (4) aligned to the peak; (5) downsampled to the relevant frequency; (6) mean or median is computed.

4.1.1.2. Template Creation

In order to create the templates associated with each cluster, the following procedure is followed for each test (Figure 4.2). Once the spikes are obtained and clustered, templates are formed using the mean and median of 144kHz interpolated signals for each different clusters. Interpolation is done via the interpolation method of interest, and once templates are formed, they are downsampled holding peak as a reference to the frequency of interest depending on tests.

4.1.1.3. Spike Classification

Template matching is achieved by taking a difference measure between the template and the incoming spike, after they are aligned to the reference point. The following are the template matching methods tested:

- Euclidean Distance (ED): One of the most commonly used measure [104] given by:

$$ED_k = \sqrt{\sum_{i=1}^{n}(y_i - T_i)^2} \tag{4.1}$$

- Squared Euclidean Distance (SED) [171]:

$$SED_k = \sum_{i=1}^{n}(y_i - T_i)^2 \tag{4.2}$$

- Weighted Euclidean Distance (WED):

$$WED = \sqrt{\sum_{i=1}^{n} w_i(y_i - T_i)} \tag{4.3}$$

  where weights w_i are the inverse of the variance of the i^{th} sample point [172]. This is calculated for each sample position during the construction of the templates.

- Sum of Absolute Differences (SAD): This is the simplest metric used which sums the absolute difference between signals and the template

$$SAD_k = \sum_{i=1}^{n} |(y_i - T_i)| \tag{4.4}$$

- Optimized Sum of Absolute Differences (OSAD): This is originally used in [116] as a template matching method for image processing problem. Since image values are all positive, Eq. 4.5 is a slightly modified version of the original equation. In this case, absolute value of the scaling values are taken.

$$OSAD_k = \frac{\sum_{i=1}^{n} |(y_i - T_i)|}{max(|y_i|, |T_i|)} \tag{4.5}$$

where y is the detected spike and T the template, of both length n.

Traditionally, template matching systems compare the detected spike with all available templates and assign the detected spike to the one with the smallest error (i.e. difference). However, this approach is not computationally efficient and the number of operations is directly proportional to the number of templates, resolution (bits/sample), and sampling frequency combined with template window (samples/template). One of the ways to reduce the computational complexity is the use of comparison thresholds to stop template matching process if a match is found rather than going through all available templates.

For example, given three templates, the traditional template matching systems compute the error between the detected spike and all three templates, and assign it to the one with minimum error. On the other hand, when comparison thresholds are used, as soon as the difference of the detected spike with one particular template is calculated, it is checked against corresponding comparison threshold. If it is below the threshold, template matching process is terminated and the detected spike is assigned that particular neuron. Therefore, this strategy eliminates the computations required for comparisons with subsequent templates.

Comparison thresholds are calculated following template creation that usually involves a clustering algorithm during a training stage. After establishing templates, the distribution of errors between recorded spikes and created templates are obtained. Figure 4.3) illustrates the distribution of error between recorded spikes and one particular template. The cluster with the lowest index belongs to the template of interest. Therefore, the comparison threshold must be

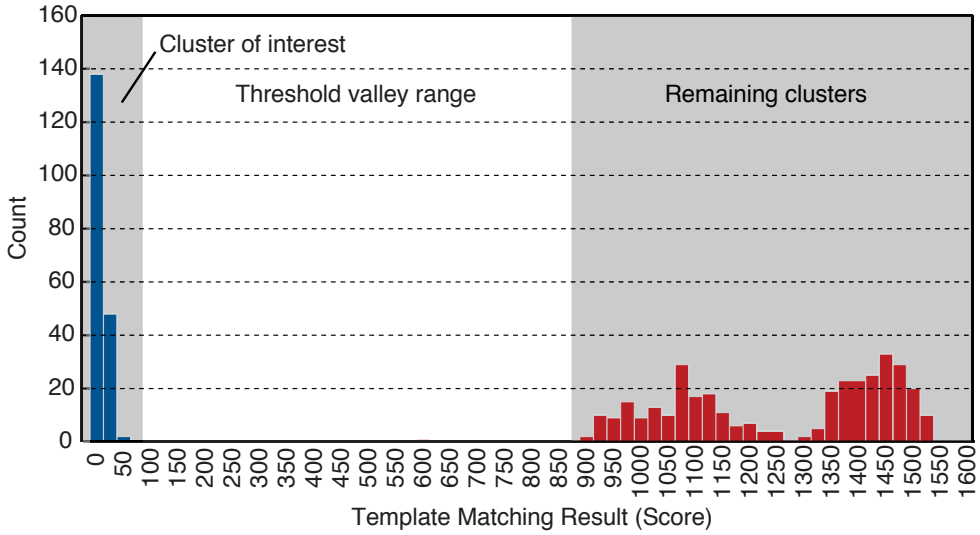


Figure 4.3.: Distribution plot for one particular template. The y-axis indicates spike count for a particular matching result on the x-axis.

set between 100 and 900 for this particular template. This may be automatically done using a first valley detector during training stage.

As the noise levels and similarity of neural classes increases, the clusters will become more difficult to separate. Also, different valley seeking algorithms have degrees of accuracy depending on the metric they employ. To eliminate the effect of these non-idealities *a-priori* information regarding spike classes are incorporated to find the optimum threshold setting. This is defined as the threshold setting that maximises the accuracy of the separation (Eqn. 4.6). This ensures that the threshold provides the maximum possible accuracy for given template matching methods and no bias is introduced as a result of choosing different valley seeking methods.

4.1.1.4. Statistical Measures

Since this study does not involve spike detection the accuracy quantifier simply measures spikes correctly classified. Since some spikes may not be classified at all, and of those classified, not all may be correctly done so, these measures are combined into one (4.6).

$$Accuracy, \% = (\%C) * (\%CC) \tag{4.6}$$

where C is the spikes classified and CC the correctly classified. For example, if only 90% of spikes are classified of which 80% are correct, then the accuracy is 72%. Where necessary both the individual classification and correct classification results are reported.

4.1.1.5. Other Signal Attributes

In this section, hardware implementation parameters that affect the accuracy of spike sorting are discussed. Although these are parameters related specifically to front-end electronics, and

are investigated (and trade-offs established) in Sections 3.6, these are repeated here to verify once more the results of the previous findings, and assess the behaviour of specific template matching methods under these parametric conditions. These are:

- Bandwidth: As described previously, typical signal bandwidths range between of 300-700 Hz (to remove local field potentials (LFP)) for high pass, and 5-15 kHz for low-pass filtering (to avoid aliasing). In addition, the typical filter roll-offs in literature are 2^{nd} or 4^{th} order. With use of filtering,the temporal morphology of the signal is expected to be altered (due to phase distortions introduced in analogue filtering), which may reduce spike sorting accuracy.

  In the tests, three filter types (previously investigated) are tested: Elliptic, Bessel, and Butterworth. Each of these filters is implemented as a 2^{nd} and 4^{th} order filter with high-pass cut-offs of 300, 500 and 700 Hz, and low pass cut-offs of 4, 5, 6, 8, 9, 10, 12 kHz. These filters are additionally implemented as non-causal, i.e. zero-phase, filters (Using Matlab's *filtfilt* function) to compare the morphological changes due to only bandlimiting (without phase distortion).

- Resolution: Since the signal processing occurs in digital domain, detected signals are quantised by analogue-to-digital converter. Typically, spike sorting systems use a resolution of 6 to 10 bits to represent the signals. However, these quantised signals include the quantisation noise, and resolution tends to be limited by the SNR of the signals and the system. Previously, it has been established that 7-8 bits would be a good compromise between hardware complexity and sorting performance. Here, this hypothesis is verified once more for specific template matching methods.

- Signal to Noise Ratio (SNR): Signal-to-noise ratio is an important signal attribute which may have significant effects on other parameters. Therefore, the goal in these simulations is to test all template matching relevant parameters under realistic SNR levels.

4.1.1.6. Test Datasets

In order to test the parameters of interest under different signal morphologies and noise levels, six data sets were applied (based on simulated extracellular recordings described in [77, 144]). Each of these data sets are simulated at different SNRs (26, 20, 16.5, and 14 dB), bringing the total number of data sets tested to 24. For more detailed information on datasets, please refere to Appendix B.1.

4.1.1.7. Tests

In this section, tests conducted to analyse the parameters of interest are described. These tests were designed mutually exclusive to isolate the key parameters' effect on spike sorting. Not only these methods are assessed in terms of their accuracy, but also complexity and practical implementation based on the required computation time. All tests are carried in Matlab v7.13,

Table 4.1.: Summary of parameters varied to assess the performance of template matching for spike sorting.

Interpolation Methods	Linear, Quadratic, Cubic, 5^{th} Order Spline, sinc
Interpolation Frequency	12, 16, 18, 24, 36, 48 and 72 kHz
Noise Levels	0.05-0.4 (14-26 dB)
Template Matching	ED, SED, WED, SAD, OSAD
Template Creation	Mean and Median
Filter Types	Butterworth, Elliptic, Bessel
Filter Order	2^{nd}, 4^{th}
Low-pass cut-off	4, 5, 6.5, 8, 9, 10 and 12 kHz
High-pass cut-off	300, 500, 700 Hz
Resolution	2-10 bit

and all parameters being tested are listed in Table 4.1.

Test 1: Interpolation for Alignment

Improving the spike alignment such that alignment features correlate in time with the template of interest, should improve the accuracy. In this test, the objective is to find out how accuracy changes when signals are aligned at different interpolated frequencies. Therefore, using each interpolation method listed in Section 4.1.1.1, signals are aligned to their peak at various frequencies. Once aligned, they are then downsampled back to 24 kHz, using the peak at the interpolated frequency as the common reference for template matching. It should be noted that templates are generated as described in Section 4.1.1.2, and signals aligned via specific interpolation method are matched with templates constructed using the same interpolation method.

Test 2: Interpolation for Template Matching

The aim of this test is to assess whether including more sample points in the matching process increases the accuracy or not. Therefore, in this test, both the alignment and matching are performed at the interpolated frequencies. By comparing Test 2 with Test 1, one can observe any additional improvements in interpolation for template matching. For Test 1 and Test 2, signals are downsampled from 144 kHz to 12 kHz, and then interpolated to the frequency of interest.

Test 3: Template Window Size

This test is an extension to the hypothesis investigated in "Test 2: Interpolation for Template Matching". In Test 2, the primary objective is to assess whether more samples enhance sorting performance. However, in doing so a fixed template window is used. In this test, the goal is to investigate the effects of window size on sorting process. Moreover, whether a subset of samples could perform as well as the whole template window will be analysed. This is particularly important from resource utilisation perspective since having a larger window size (at a fixed sampling frequency) results in more samples, hence more computations for template matching, and, additional storage for template and detected spikes. Therefore template matching performance of various window sizes (at different sampling frequencies) are investigated by varying

the number of samples before and after spike peak.

Test 4: Resolution
This test investigates the effect of different resolution levels on template matching accuracy.

Test 5: Bandwidth
In this test, the effect of filtering on template matching is assessed by varying filtering parameters (cut-off frequencies, filter order and type).

Test 6: Noise Performance
Using data sets with different levels of SNR, (1) Accuracy of TM methods with noise, and (2)Performance of interpolation and other matching metrics with varying levels of noise are investigated.

Test 7: Training
In all previous tests, templates and thresholds are determined by using the full data set in order to reduce the errors due to non-idealities. In this test, a random percentage of the data sets is chosen as training set to find the optimum thresholds and create the templates.

4.1.2. Results and Discussion

This section provides the findings, as well as the discussion of their significance in terms of accuracy and hardware implementability.

4.1.2.1. Interpolation for Alignment

Up-sampling the data allows a more accurate estimate of the peaks to be generated, hence improving alignment and accuracy. However, the interpolation method may not have suitable control of curve shapes to estimate the signal accurately.

In this test there are three dimensions to explore: template matching quantifier, interpolation method, and interpolation frequency. Given the amount of data and potential parameters to explore, first template matching method is investigated. First, to estimate which template matching quantifier performs better (throughout all interpolation frequencies and methods), the accuracy for the different datasets are quantified, shown in Fig. 4.4. It is observed here that in general, the template matching quantifies based on Euclidean metrics are the best performers. Mean and median templates do not have large enough variations to opt for one over the other. Mean templates are used for analysis henceforth since it is relatively easier to calculate mean (compared to median) from implementation point-of-view.

Looking more closely at the results of SED and SAD across interpolation methods and frequencies (see Fig. 4.5), a number of interesting observations can be made. Firstly, the SED

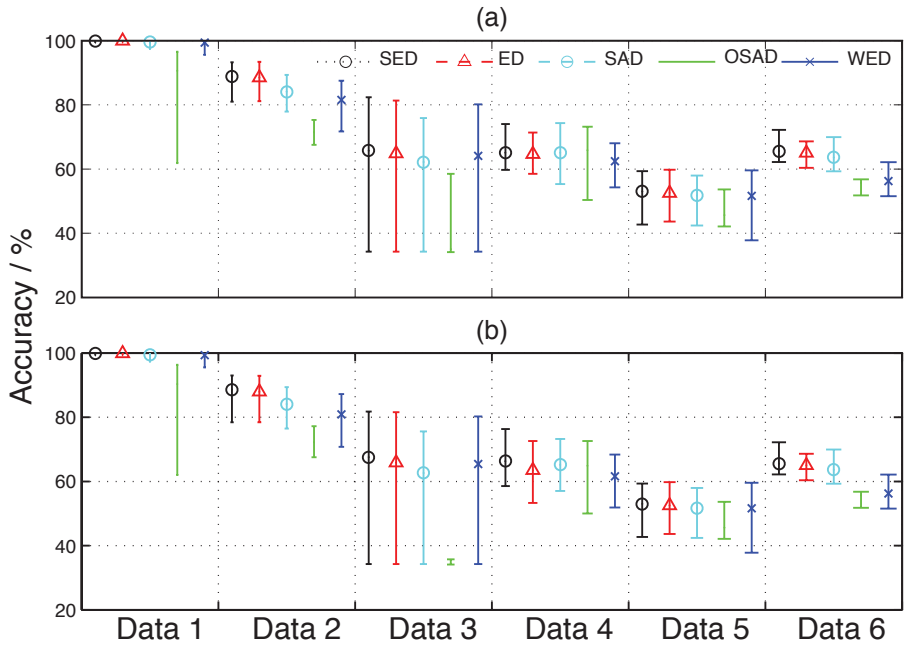


Figure 4.4.: Accuracy for the different template matching quantifiers through all interpolation frequencies and methods. Shown are results using: (a) mean, and (b) median templates. All the datasets tested at a noise level of 0.1, i.e. SNR=20 dB.

vs. SAD separation in accuracy previously observed is correct but with SED reaching higher accuracy values as interpolation frequency increases. In fact the results show that after 24 kHz interpolation frequency the accuracy tapers to fixed value.

Another observation is that both cubic and sinc interpolation methods are the best performers. Quadratic is by far the most variable. One wouldn't expect linear to change the peak position but given the sampling rates are not of integer multiples of each other when down-sampled from 144 kHz template the sample points chosen will vary, i.e. samples chosen from 144 kHz to 24 kHz will differ from those selected for 18 kHz. Hence this is indicative of some sampling point bias introduced in the results. This also accounts for some of the other apparent random variations (e.g. Figure 4.5 (d) 5^{th} order cubic spline).

However, neglecting these outliers, in alignment alone, cubic or sinc interpolation for alignment can improve accuracy by 2-25%. The point at which saturation of this change occurs is at 24 kHz in all data sets (which is the same sampling frequency obtained in front-end optimisation simulations).

It is important to note that in hardware, up-sampling the data would result in an increase in power consumption due either to an increased ADC sampling frequency [173], or to the additional complexity required for implementing the interpolation filter.

4.1.2.2. Interpolation for Template Matching

This test measures if there is any iterative improvement on accuracy by matching the signal at the interpolated frequency (i.e. including more samples). Therefore, the template matching

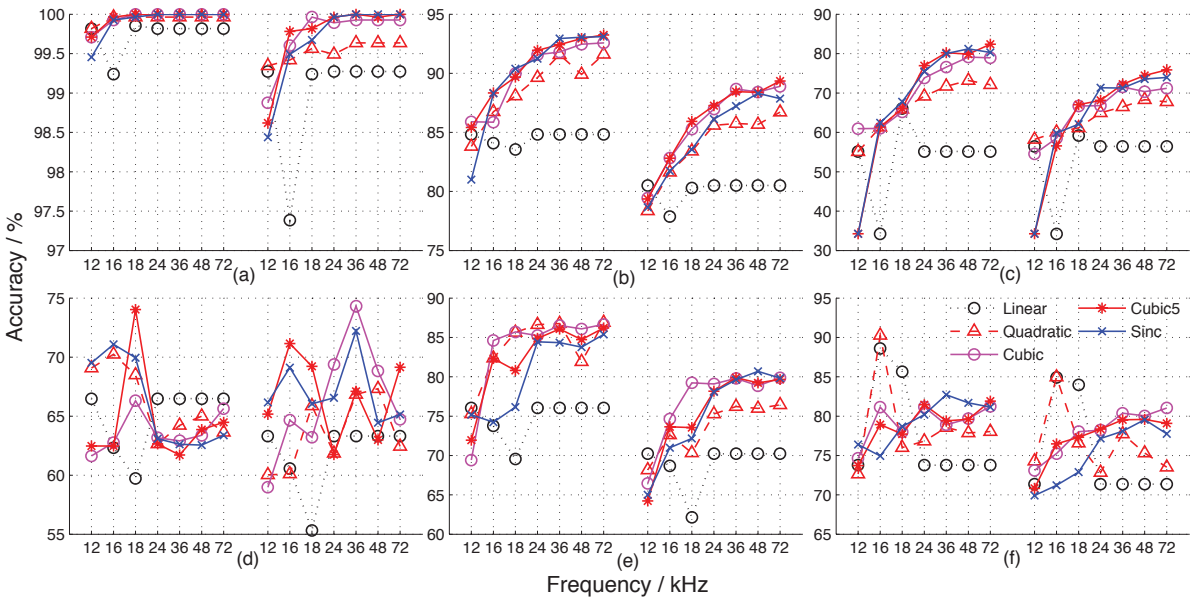


Figure 4.5.: Effect of interpolation frequency on template matching accuracy for SED and SAD quantifiers. Data shown is for each dataset 1-6 (a-f).

accuracy obtained in Test 2 and Test 1 are compared. Figure 4.6 presents the distribution of differences between Test2 and Test2 accuracy. The results clearly reveal that there are no extra gains in accuracy by including more samples in template matching (i.e. matching at interpolated frequency). As mentioned in Test 1, the slight variability observed is due to the specific sample points selected when constructing the resampled signal from 144 kHz . SED and OSAD shows the least accuracy difference between Test2 and Test1, while ED and SAD are the least variable among template matching methods. In addition, cubic interpolation shows the least variability and accuracy difference compared to sinc interpolation.

From hardware perspective, both the interpolation and template matching methods will have an impact on chip area and power consumption. Therefore, mean computation time over all data sets and noise levels are quantified using Matlab's timing functions. Computation time is then compared to average accuracy to illustrate accuracy vs. computation trade-off (see Figure 4.7). It is observed that SED and ED provides the highest accuracies with efficient computation time, while they are closely followed by SAD which is more efficient. In terms of interpolation, cubic and sinc has the highest accuracy but sinc is the most computationally efficient. This is due to the fact that this method is implemented using an FFT filter-based methodology which is very efficient. However, both sinc and cubic methods can be implemented as filter topologies [174–176], and since there are various implementation possibilities, it is difficult to quantify the true trade-off between sinc and cubic interpolation.

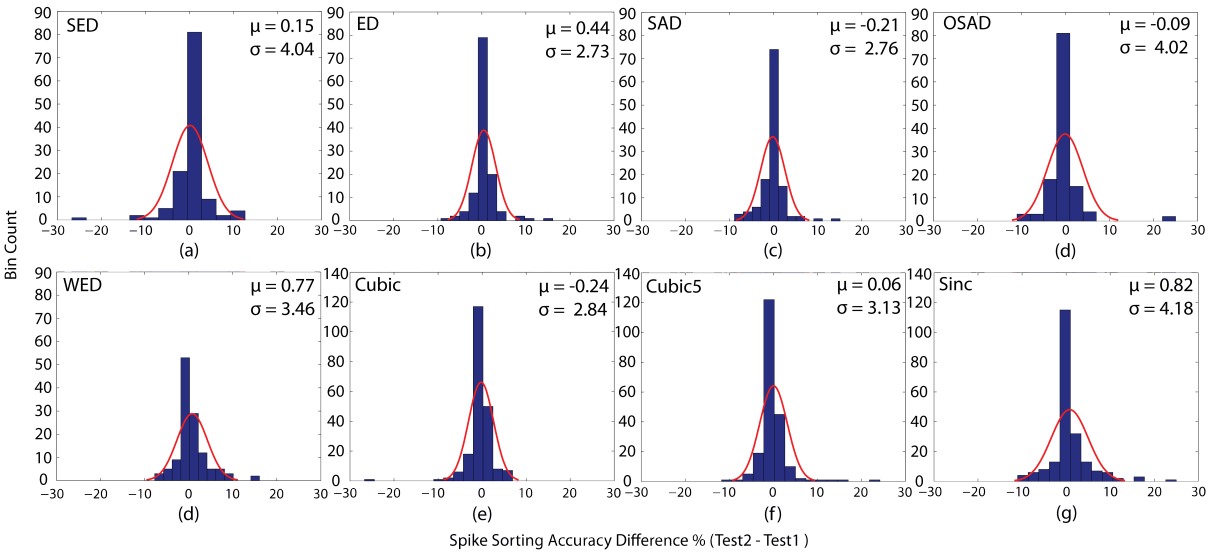


Figure 4.6.: Histograms of spike sorting accuracy differences between Test2 and Test1 results. Presented in plots are: (a-d) different template matching metrics across interpolation methods (3^{rd} and 5^{th} cubic spline and sinc) and all datasets, (e-g) cubic spline (3^{rd} and 5^{th}) and sinc across all template matching metrics and datasets. A Gaussian distribution fit for each histogram is also presented.

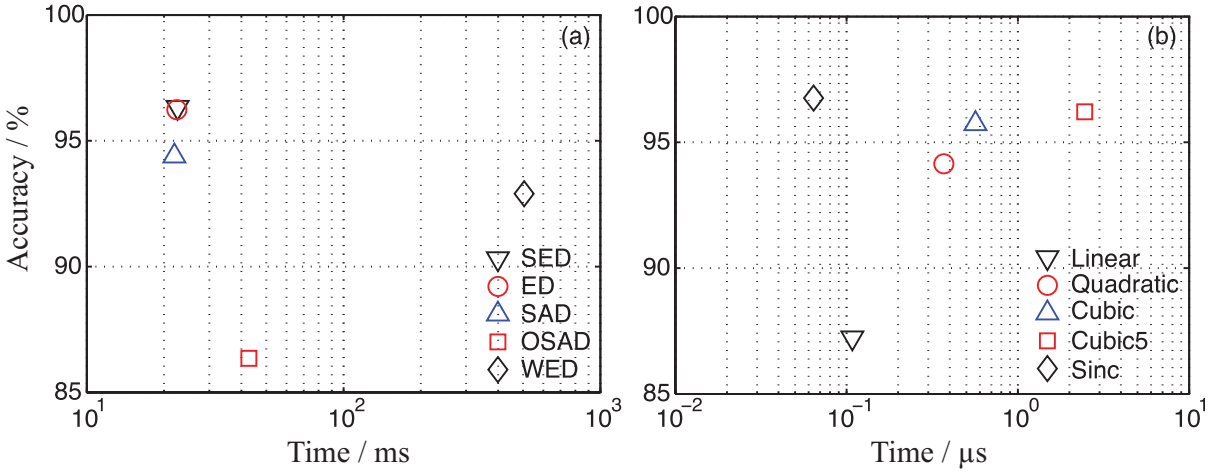


Figure 4.7.: Spike sorting accuracy vs. execution time of: (a) template matching methods, and (b) interpolation methods. Evaluated by measuring the computation time of the various in-built functions in Matlab.

4.1.2.3. Template Window Size

Typically identified around 2.7 ms (action potential duration), template window size not only might have effects on sorting performance, but its effects on hardware resources are two fold. First, a larger window size at a fixed sampling frequency means more samples to match, hence increased number of computations. Second, a larger window size would also mean more memory elements are required in order to store both the templates and the incoming spikes. Especially, as far as scalability is concerned, template memory becomes one of the bottlenecks in terms of system area. Therefore, any possible reduction in the window size would make a template-based spike sorter more feasible and scalable.

In "Test 2: Interpolation for Template Matching", the aim is to investigate if there are any iterative improvements on performance by including more samples (via interpolation). Although by up-sampling more samples are included, the window size (in milliseconds) remain fixed through all interpolated frequencies. The goal of this section is to test whether increasing the window size has an effect on performance. Moreover, the goal is to establish if any subset of samples in a given template window can match the performance of the whole template window. This will be iterated at various sampling frequencies to assess if there are trends different than observed in previous tests. The parameters considered for this test are as follows: Window Size (samples before and after peak), template matching quantifier, sampling rate, data sets and noise levels. Therefore, the investigation methodology has been designed to fix those parameters along the way to converge to a solution.

One of the first parameters to fix is the template matching quantifier. In previous investigations, three quantifiers have provided a good balance between accuracy and computational requirements: SED, ED and SAD. Among these, SED and ED provide the highest sorting performance while SAD trades accuracy for more efficient performance. However, since ED and SED are almost identical in performance and computational complexity (as well as showing similar trends), only ED is investigated along with SAD in the analysis.

First, the effects of noise and sampling are investigated across all data sets. Figure 4.8 presents one of the data sets for illustration purposes. Results show that as the noise levels and sampling frequency is changed, there is an absolute change in the performance which parallel the trends observed in previous sections when sampling frequency and the SNR were changed. However, despite changes in absolute performance in these two dimensions, the relative performance of different window size combinations (samples before and after the peak) stay constant. Based on this observation, for the remaining analysis, sampling frequency and noise level is fixed. Sampling frequency is chosen to be 24 kHz since topographic plots could be investigated at a higher resolution, while a noise level of 0.1 standard deviation represents a realistic noise level for neural recordings. The plots for all data sets at fixed sampling frequency and noise level are illustrated in Figure 4.9.

One of the main observations is that the larger template window does not guarantee higher performance. Although they generally perform much better than the smallest window sizes, the regions of high performance show variations across different combinations of samples before

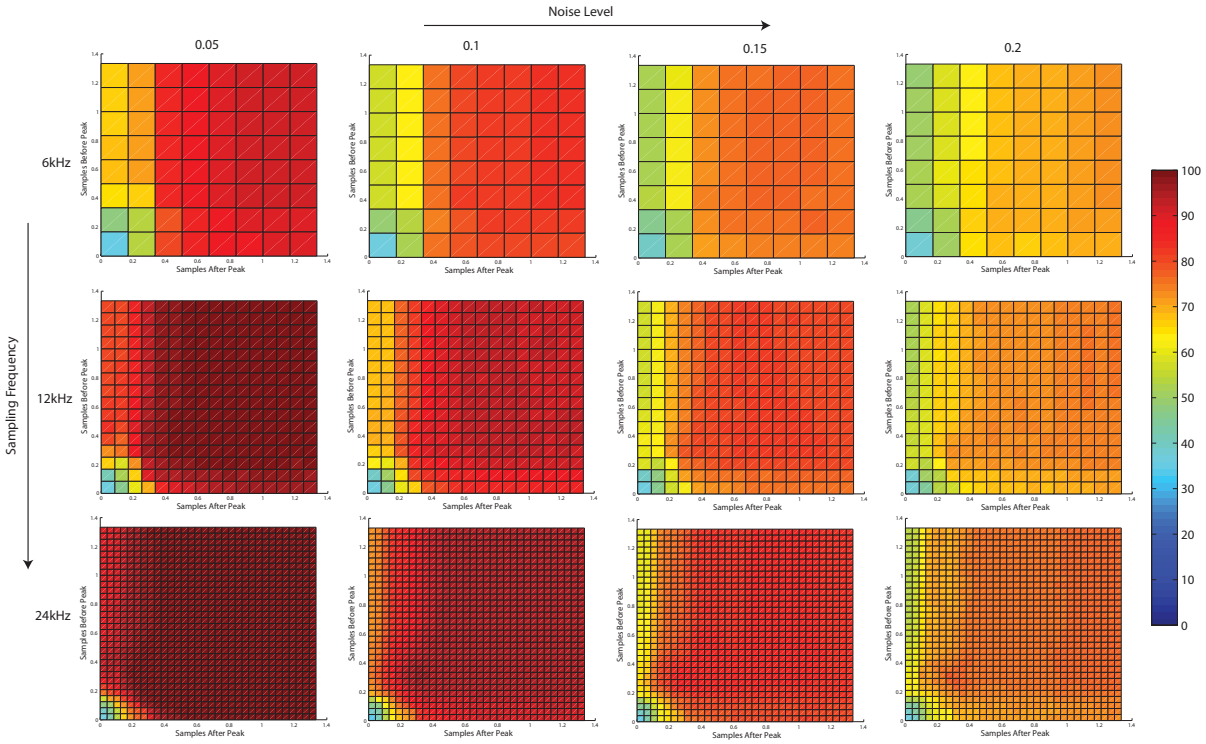


Figure 4.8.: Template Matching (SAD) results for various window sizes. Difficult1 Dataset at varying noise levels and sampling frequency.

and after peak. Moreover, the location of "regions of high performance" also vary depending on dataset. This is simply because each dataset contains different groups of spikes which may differ at different locations compared to those in other datasets. Hence, different combinations of samples (before and after the peak) may be needed to include those differentiating features. Likewise, decrease in performance observed in some instances — when more samples are included in template window — may be attributed to the fact that those samples do not present any valuable information content to distinguish spikes, and only injects noise into distance calculation.

Nevertheless, the key observation here is that template windows that span 0.4 ms before the peak and 0.3 ms after the peak is among the highest performing regions in all datasets. Although the exact value of template window shows variations based on sampling frequency, for 24 kHz — which has been previously shown to be a near optimum operation point for accurate and efficient spike sorting— 0.7 ms template window is found to provide high accuracy while saving computations and memory.

4.1.2.4. Resolution

Having established the trends and trade-offs of ADC resolution in previous sections dealing with front-end optimisation, this test aims to verify previous conclusions once more, and to observe the behaviour of each template matching quantifier.

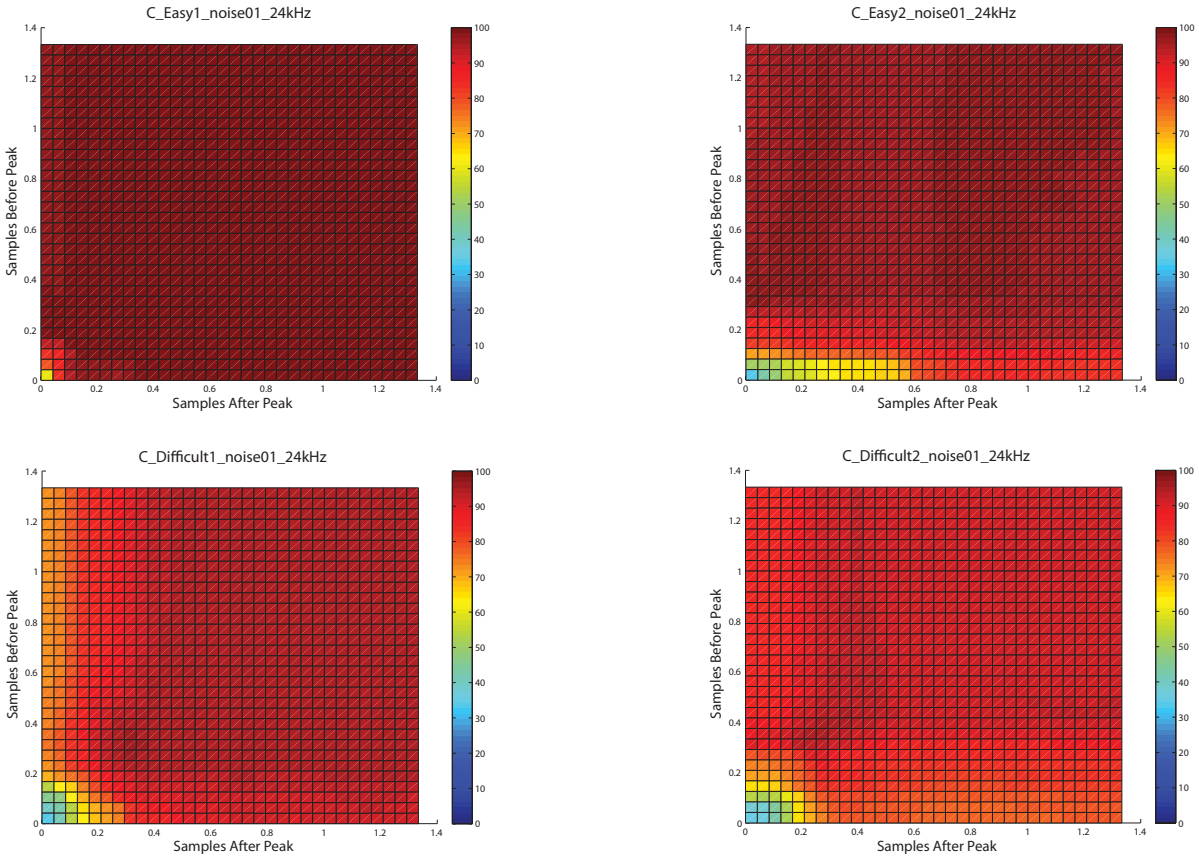


Figure 4.9.: Template Window performance for four datasets at 24kHz sampling rate and 0.1 noise level.

For each test, the resolution is distributed evenly over the input range of the dataset. Thus many of the signals will be subjected to an effective number of bits (ENOB) less than what is presented here. For example, for a dataset with a maximum input range of 500 μV (=$V_{max} - V_{min}$) that is sampled at 10-bit resolution, a spike of amplitude 100 μV_{pp} is represented by an ENOB of 7.68 (=$log_2[(100\mu/500\mu) \times 2^{10}]$. However, in practice spikes can vary by orders of magnitude so will be subjected to a variable resolution. The results for all 6 data sets are shown in Fig. 4.10. It can be clearly observed that for a resolution of between 4 and 6-bits (depending on dataset), the signals are still separated with similar ($< 1\%$) accuracy. This effectively means that the raw spike signal, if captured over the full scale, could be captured with a resolution lower than 6 bits. Allowing further 2-4 bits for spike amplitude variations, the results here confirms the previous ones. In terms of template matching quantifiers, it can be observed that SED, ED an SAD are the most consistent and highest performers.

4.1.2.5. Bandwidth

Here, once more, the effect of band-limiting the neural signals using a high and low pass filter is investigated. Impact of band-limiting is investigated for every template matching quantifier, and, as before, the parameters of interest are cut-off frequencies (high and low), filter type, and

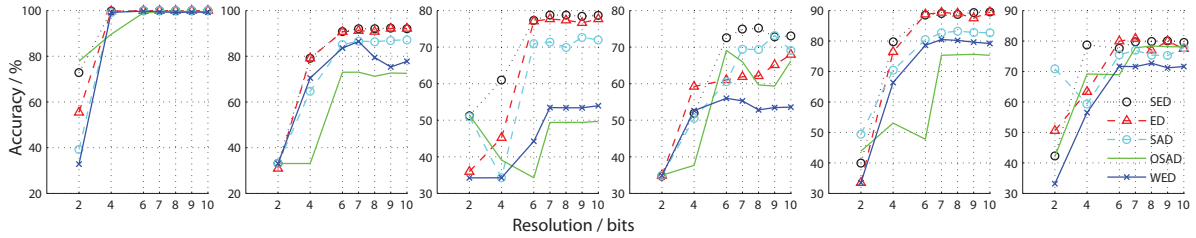


Figure 4.10.: Change in accuracy for each data for different resolutions and different template matching methods.

filter order. Moreover, the impact of phase distortion effects of analogue filtering, especially on alignment, is explored. Therefore, filtering is performed both with non-causal (zero-phase) and causal filtering to compare two different implementations in terms of sorting performance.

The results showing the effect of filtering on template matching accuracy are shown in Figs. 4.11 and 4.12 respectively for low- and high-pass filters (non-causal). What can be observed in Figure 4.11, is that the low pass filter (LPF) has little effect on the overall accuracy of spike sorting. The optimum cut-off frequency appears to be in the 5-8 kHz range where it can be argued that sufficient noise has been removed to improve the matching and not so much as to distort the low frequency components of the spike shape. This is true for all three filter types, although it would appear Elliptic and Bessel have the least variation among template matching methods. Similarly, with the results of the high pass filter (HPF) shown in Fig. 4.12, only a small ($< 5\%$) accuracy variation can be observed, with mixed results between the filter implementation. At 300-500 Hz, it would appear overall that Elliptic then Bessel are better performing implementations. These results are for 2^{nd} order filter implementations. Little difference was found for both LPF and HPF 4^{th} order implementations.

The next set of results considers the issue of phase distortion that a front-end analogue filter would suffer from. The signal templates shown in Figure B.1 follow a typical profile for action potentials. Figure 3.17 illustrates the effect of phase distortion introduced by a realistic analogue HPF, while Figure 4.13 illustrates the effect of misalignment on sorting performance. Here, it can be observed that the distortion increases with increasing cut-off frequency. This particularly impacts the spike alignment, as quantified in Table 4.2, over all the datasets. Here, it can be observed that introducing a HPF can cause up to a 27% misalignment (in the worst case scenario), depending on the shape of the spike. The methods used here for alignment include: (1) the first peak [80], (2) absolute maximum peak [177], and (3) the maximum peak. Other methods that can also be used include threshold crossings and applying techniques commonly used in spike detection enhancement (such as non-linear energy operators). The distortion effects of filtering need to be carefully considered when choosing filter topology and cut-off frequency. For the datasets tested herein, the best results are achieved with a Bessel filter with high pass cut-off frequency of 300-500 Hz, using maximum peak alignment.

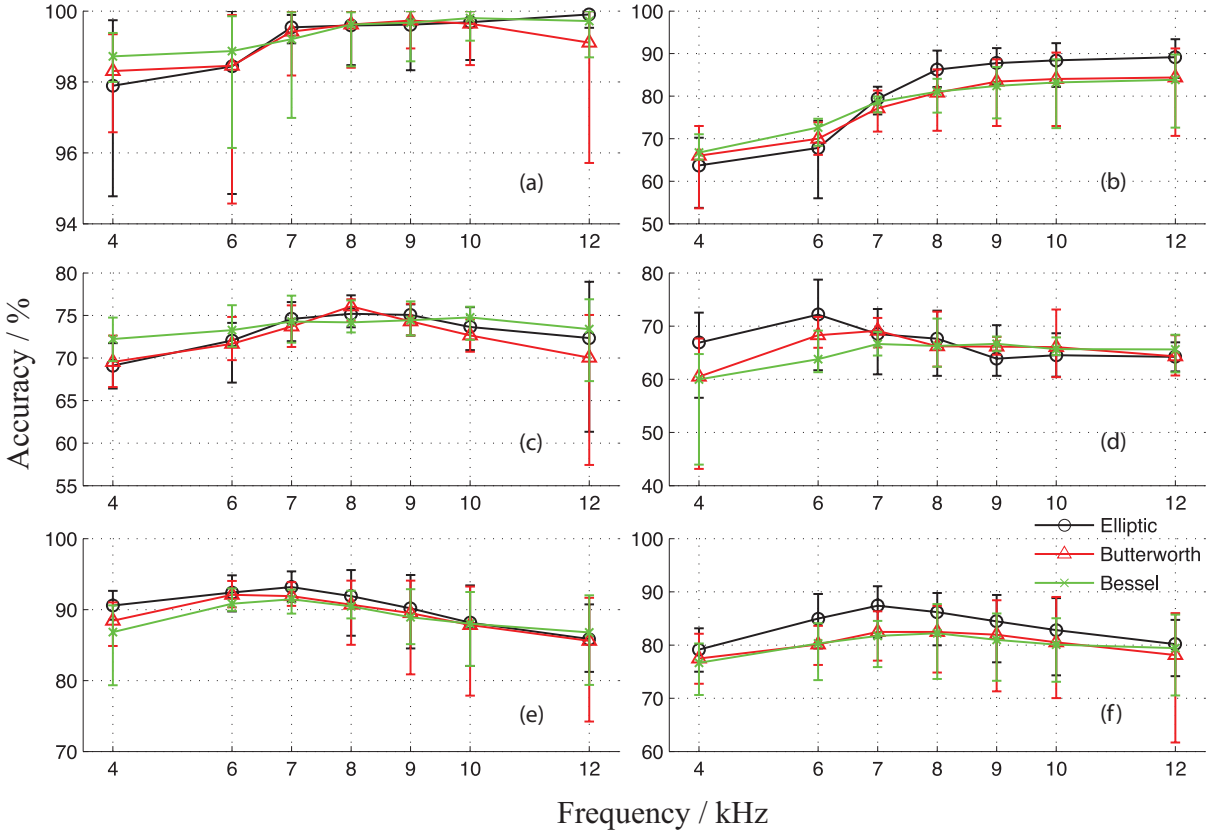


Figure 4.11.: The mean and max/min (errorbars) across all template matching quantifiers for a non-causal implementation of low pass filters (Butterworth, Elliptic and Bessel), for each data set 1-6 (a-b).

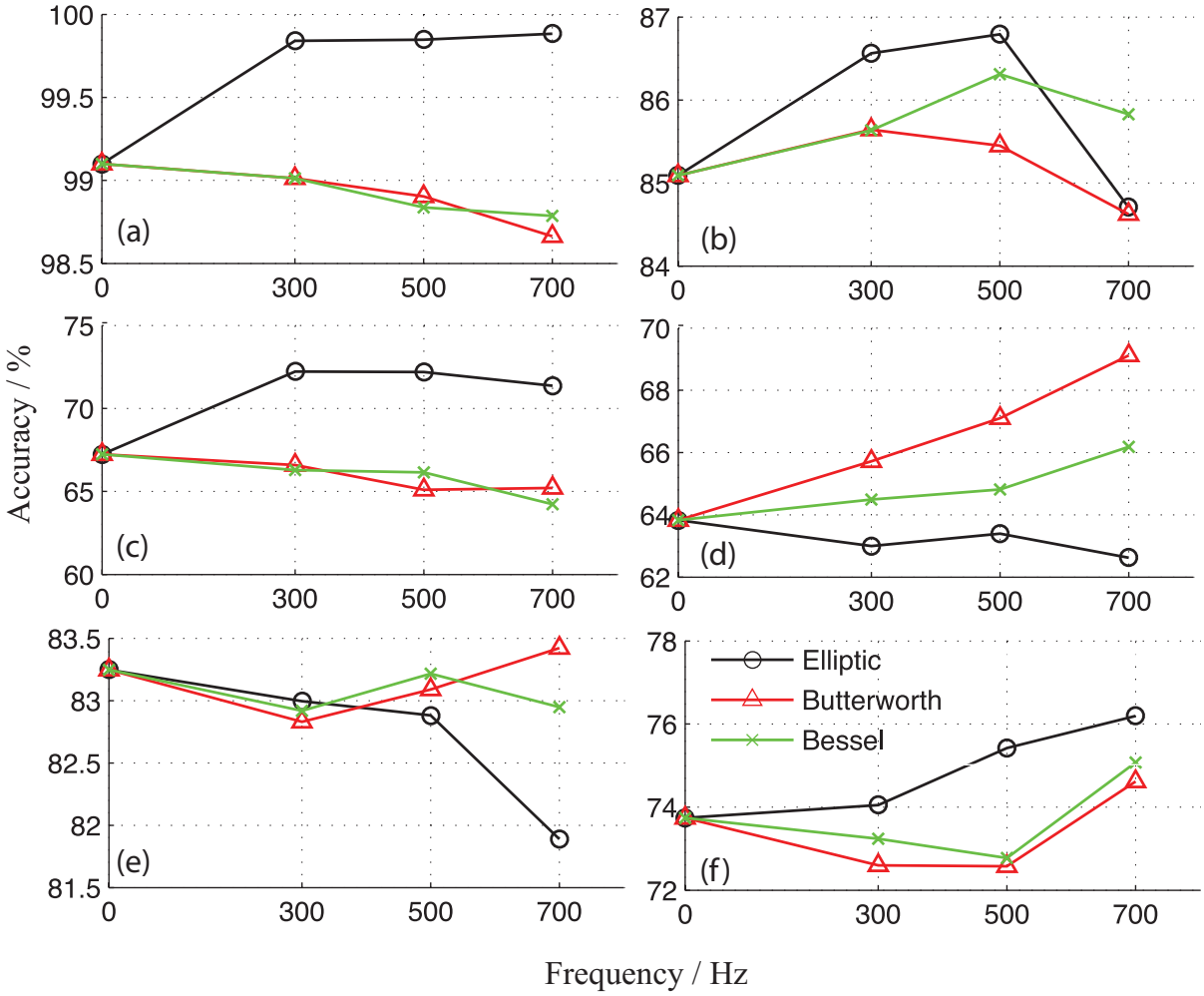


Figure 4.12.: The mean across all template matching quantifiers for a non-causal implementation of high pass filters (Butterworth, Elliptic and Bessel), for each data set 1-6 (a-b).

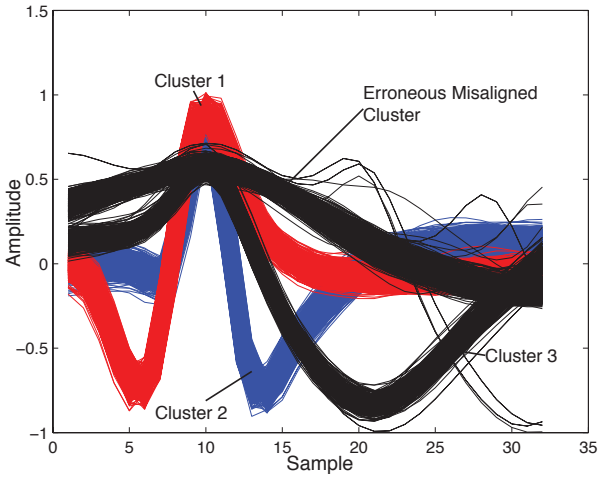


Figure 4.13.: Example of the effect of misalignment on the separation of clusters. In this case a 4^{th} cluster is formed which counts as an unclassified cluster.

Table 4.2.: Effect of HPF phase distortion on alignment for three different methods across all datasets.

			2nd Order		4th Order	
Alignment Method	Cut-off Freq/Hz	Filter Type	Range min-max	Mean/Std.	Range min-max	Mean/Std.
First Peak	300	Elliptic	0 - 5.39	0.98/2.17	0 - 3.15	1.20/1.55
		Butterworth	0 - 9.2	1.57/3.74	0 - 16.71	5.60/ 8.61
		Bessel	0 - 16.15	2.75/6.57	0 - 1.32	0.59/ 0.66
	500	Elliptic	0 - 12.09	4.07/6.21	0 - 13.48	2.50/ 5.39
		Butterworth	0 - 1.03	0.43/0.51	0 - 4.96	1.51/ 1.97
		Bessel	0 - 2.44	0.43/0.99	0 - 12.01	4.09/ 6.14
	700	Elliptic	0 - 14.91	5.12/7.59	0 - 25.76	7.51/ 10.82
		Butterworth	0 - 3.81	1.28/1.96	0 - 9.74	1.90/ 3.86
		Bessel	0 - 0.33	0.15/0.17	0 - 8.5	3.98/4.40
Abs. Max	300	Elliptic	0 - 16.86	4.29/6.49	0.08 - 28.36	10.58/9.51
		Butterworth	0 - 14.78	4.28/5.77	0 - 17.11	9.68/6.21
		Bessel	0 - 16.79	5.05/6.61	0 - 17.19	7.91/ 8.14
	500	Elliptic	0 - 16.71	8.63/8.19	0.12 - 23.7	9.10/9.20
		Butterworth	0 - 15.06	7.19/7.20	0.11 - 27.75	9.61/11.13
		Bessel	0 - 14.25	5.99/6.88	0 - 13.72	7.30/5.07
	700	Elliptic	0 - 17	6.23/5.73	1.58 - 15.43	7.10/5.67
		Butterworth	0 - 21.97	7.53/7.48	0.08 - 7.76	4.37/3.15
		Bessel	0 - 16.51	6.85/5.74	0 - 21.87	8.10/9.11
Maximum	300	Elliptic	0 - 0	0/0	0 - 2.7	0.90/1.39
		Butterworth	0 - 0	0/0	0 - 16.6	5.53/8.57
		Bessel	0 - 0	0/0	0 - 1.34	0.45/0.69
	500	Elliptic	0 - 11.98	3.99/6.19	0 - 0.67	0.24/0.24
		Butterworth	0 - 1.02	0.34/0.53	0 - 1.95	0.65/1.01
		Bessel	0 - 0.07	0.02/0.04	0 - 12.9	4.30/6.66
	700	Elliptic	0 - 15.39	5.13/7.95	0 - 15.98	5.49/8.05
		Butterworth	0 - 3.73	1.24/1.93	0 - 1.18	0.29/0.44
		Bessel	0 - 0.33	0.11/ 0.17	0 - 8.32	2.77/4.30

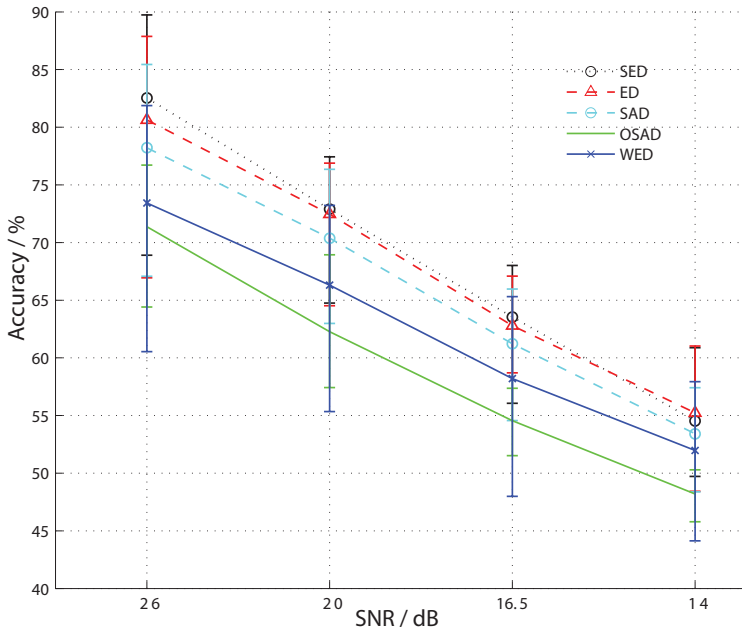


Figure 4.14.: Sensitivity of template matching quantifies to varying SNR, averaged over all datasets (with min/max error bars indicating variation over frequency of interpolation for alignment).

4.1.2.6. SNR

In this section, the general performance of each parameter is assessed on its sensitivity to noise. It should be noted that in these tests, the average over all data sets is presented in an effort to observe trends in performance in relation to varying the noise level and thus the SNR.

Template Matching and Interpolation: Firstly, the sensitivity of the different template matching quantifiers to noise is tested, with the results shown in Fig. 4.14. What is clear is that as the SNR is reduced, the performance of template matching methods degrades significantly. Although note that this is averaged over all data sets, since values of 100% accuracy are observed in dataset1 . When matching quantifiers are compared, it can be observed that SED and ED still achieve the highest accuracies. However, average relative performance of SAD is within 5% of SED and ED making it the third highest performer. Moreover, this difference gets smaller with increasing noise levels.

Similarly, by testing the effect of varying the SNR on the different interpolation methods (see Fig. 4.15), it can be observed that both cubic and 5^{th} order cubic achieve the highest accuracies. In general, the methods for interpolation (for both alignment and template matching) scale with the SNR, in a similar fashion as to the template matching quantifier. This indicates that it is the matching method, more than interpolating for alignment that affects the noise performance.

Interpolation frequency, as has been previously observed, shows little improvement beyond 24 kHz, regardless of noise level (see Fig. 4.16). This again confirms the general increasing accuracy with interpolation frequency, but that the general noise performance is affected mostly

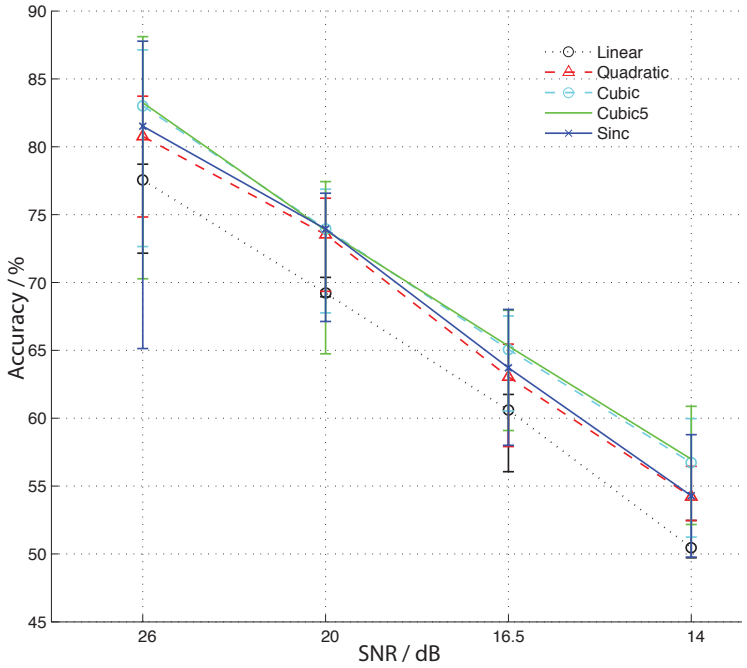


Figure 4.15.: Noise performance of the different interpolation methods (with min/max error bars indicating variation over alignment interpolation frequency) using SED as the template matching method.

by the matching quantifier.

Resolution: As observed previously by varying the resolution, for 6 bits and higher the highest accuracies can be achieved for all noise levels. This is illustrated in Fig. 4.17, with SED and ED showing the best performance, followed by SAD at these quantisation levels (Fig. 4.17(a)).

Filtering: The effect of LPF cut-off frequency on the template matching accuracy at different SNRs is shown in Fig. 4.18. This shows that 5-8 kHz is maintained as the optimal frequency for all filter types and on average there is only a small variation ($< 5\%$) between different filter types.

The HPF should in theory remove more noise and therefore improve accuracy for reduced SNRs. The results shown in Fig. 4.19 reveal this is generally true, however, it can be observed that the Bessel filter type is the most consistent at maintaining this trend.

What these results so far reveal is that template matching performance degrades significantly with noise, and although methods such as interpolation for both alignment and template matching can improve the accuracy, the fundamental bottleneck in robustness to noise is the template matching methodology (i.e. quantifier). The resolution and filter specifications do play a role in optimising the system performance as established previously, but again rely on the matching method to be resilient to noise.

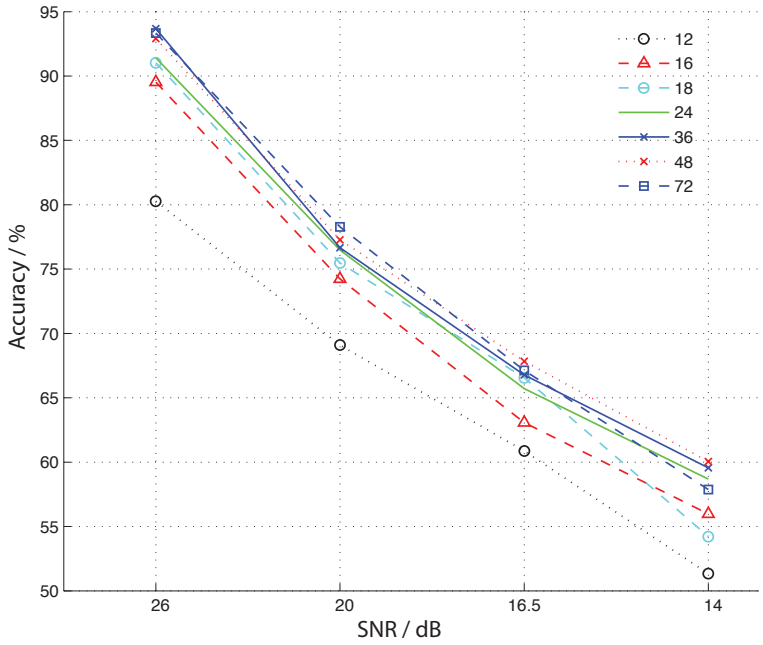


Figure 4.16.: Effect of varying interpolation frequency (for alignment) on SNR performance. Results shown for interpolation frequencies of 12, 16, 18, 24, 36, 48 and 72 kHz.

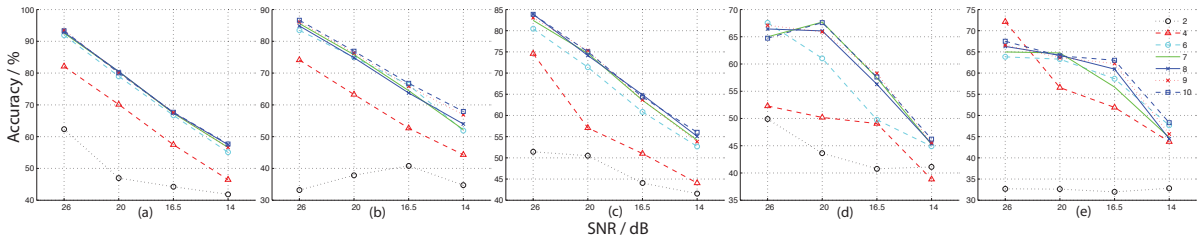


Figure 4.17.: Effect of varying resolution on SNR performance for different template matching quantifiers: (a) SED, (b) ED, (c) SAD, (d) OSAD and (e) WED.

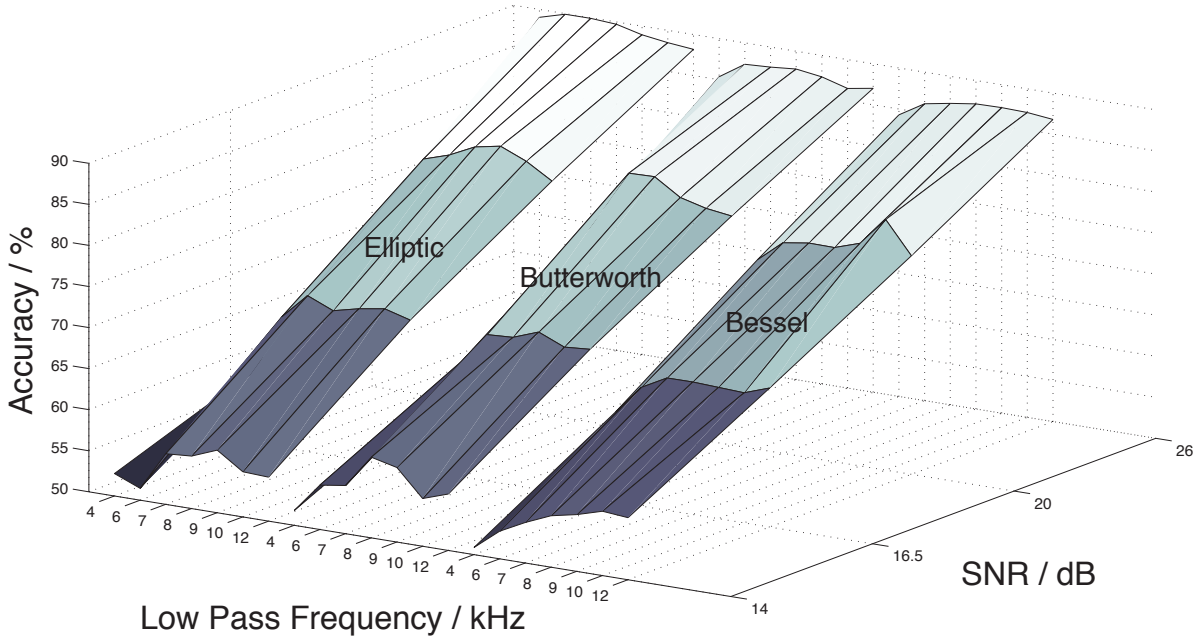


Figure 4.18.: Effect of LPF cut-off frequency and filter type on SNR performance.

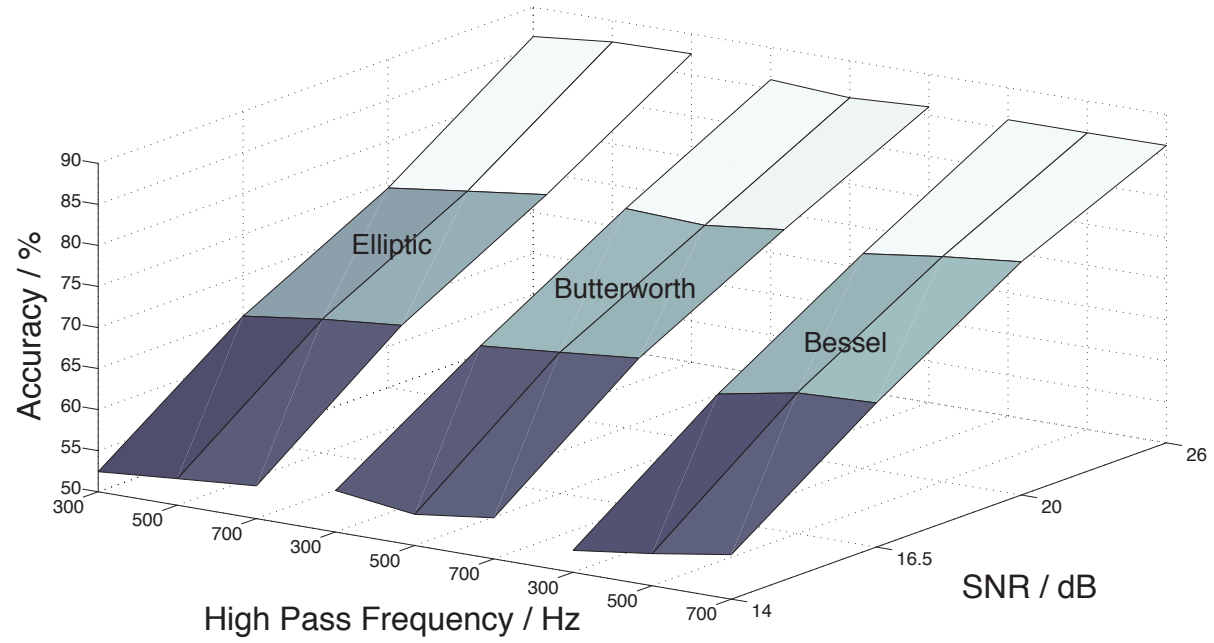


Figure 4.19.: Effect of HPF cut-off frequency and filter type on SNR performance.

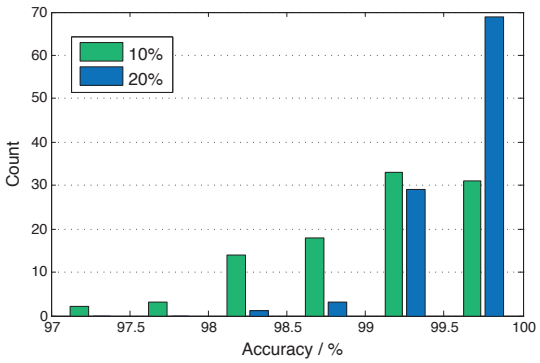


Figure 4.20.: Monte Carlo simulation of the spread in accuracy for $N = 100$ randomly selected 10% and 20% training segments using the near optimal parameter set determined herein.

4.1.2.7. Training Test

Previous tests were all carried out by using the whole data set for template creation and optimum threshold calculation. However, in a real situation, the training data will not be the whole data set to be analysed. Therefore, in order to realistically assess the performance of optimum parameters determined in the previous tests, a subset of signals are used as the training set to create templates and to find thresholds. For these simulations, the signals are sampled at 7-bit resolution, filtered using 2^{nd} order Bessel filters, with high pass cut-off of 300 Hz and low-pass cut-off of 8 kHz. The alignment is achieved using maximum peak, and SED is employed as the matching method.

Using a Monte Carlo-based statistical method, 100 training datasets were generated using 10% and 20% randomly selected segments (each containing approximately 200-300 spikes). *A priori* knowledge of the signals is used, assuming that a spike sorting classifier correctly separates this training cluster, to determine the mean templates. The results are shown in Fig. 4.20. As expected, this shows that the 20% training set achieves a higher mean (accuracy) with a lower standard deviation.

Then using a 20% dataset for template training and determining the valley threshold level for the clustering, the methodology is applied to the dataset. The results are shown for datasets 1-4 in Fig. 4.21. This shows similar to results to what has been previously reported with good performance in low noise environments, but with reduced performance at lower SNRs.

A limiting factor of the analysis performed here is that the analysis is limited to three clusters. In high noise and phase distortion situations errors introduced are a result of not only template matching classification but also due to misalignment of the peak, as has been described. The result can be that spikes are defined as unclassified in the simulations where in fact these are the same clusters generated due to the misalignment. In an automated system these would likely be classified as separate clusters.

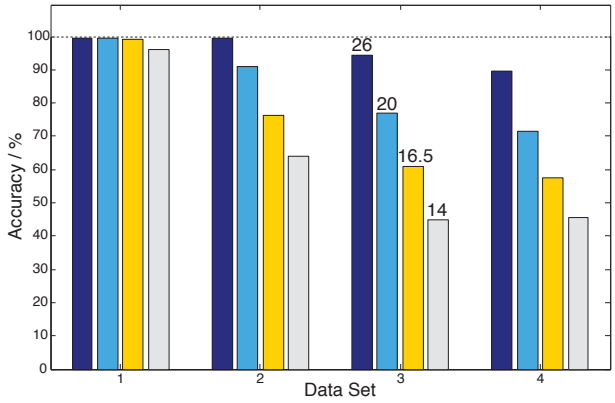


Figure 4.21.: Example training/testing results for datasets 1-4 over all four SNR values using a 20% training set.

4.2. Two-Stage Online Spike Sorting System

As previously mentioned, an alternative strategy in overcoming bandwidth and power limitations which restrict monitoring large number of channels, is performing spike sorting in the implanted device. In this section, a hybrid two-stage approach combining both offline and online data processing is evaluated as a low-power long-term solution to wireless BMIs or closed loop stimulation designs.

To assess the feasibility of such platform which wirelessly transmit the neural signals and perform real-time on-site spike sorting, the aforementioned two-stage implementation is computationally modelled. The first stage involves detecting the spikes on-node and streaming them to an external computer where mean templates are created and sent back to the chip. During the second stage, the spikes are sorted through template matching in real-time.

This procedure is evaluated using realistic simulations of extracellular recordings (synthetic data mimicking real extracellular recordings) as described in [89, 157]. The minimum data and processing requirements (for both offline and online components) that reduces hardware resource (energy and complexity), while maintaining reliable spike detection and sorting performance, are systematically investigated in terms of filtering, sampling frequency, signal resolution, etc.

4.2.1. Methods

4.2.1.1. Synthetic extracellular recordings

The realistic extracellular recordings were simulated by modelling the contributions of the local field potentials (LFPs), background noise, multi-unit activity and single-unit activity. As described in Figure 4.22, synthetic data generation procedure models the tissue surrounding the electrode into three zones. In Zone I, the neurons located close to the tip of the electrode generate single-unit activity, while neurons in Zone II produce the multi-unit activity — spikes that can be detected but cannot be clustered into different single units due to their relatively small amplitude. Zone III models the further away neurons that give rise to the background

noise and local field potentials. For detailed dataset generation methodology, please refer to Appendix B.2.

Total of 90 different simulated datasets (each 2-min long), containing three single units in addition to the multi-unit activity, are created. Of these, 30 are used in Section 4.2.2.1 for spike detection test, another 30 are used in Section 4.2.2.2 for spike sorting (template building) test, as well as being used as training set in Section 4.2.2.3 for the real-time spike sorting test. The remaining 30 simulations are used as testing set for the real-time spike sorting test (Section 4.2.2.3). Each group of 30 datasets mentioned above, consists of 10 simulations at each SNR (low, medium, high). For each of these 10 simulations, the spike waveforms are randomly chosen, but are kept constant across SNR levels (to only vary the SNR). In order to simulate varying SNR conditions, spikes are added with different amplitudes while the noise remained constant. The single-unit amplitudes range between 50 μV and 200 μV which are consistent with spikes observed in real recordings (e.g. [178]).

Since spike detection and sorting are different problems with different breakpoints (i.e. spike sorting is more difficult than spike detection), different SNR values are used for each test. For the spike detection test (Section 4.2.2.1) the amplitudes of the spikes used are 50 μV, 75 μV, and 100 μV yielding SNR values of 7.1 (low SNR), 10.7 (medium SNR), and 14.3 (high SNR) respectively. Supp. Figure A.1 presents a simulated dataset at three different SNR levels. It should be noted that simulated datasets with spike amplitudes larger than 100 μV are not included in Section 4.2.2.1, since for all tested specifications these datasets yielded 100% spike detection performance.

For the spike sorting (template building) test (Section 4.2.2.2), the spike amplitudes between 100 μV and 200 μV are used. In order to achieve simulations with medium and low SNR, the amplitudes of the spikes are reduced by 25% and 50%, respectively. The resulting SNR values are: 10.7 (low SNR), 16.0 (medium SNR), and 21.4 (high SNR). In Section 4.2.2.3, these simulations are used as training sets for template building, while a separate set of simulations with same parameters are created for template matching stage.

4.2.1.2. Filtering and virtual analogue-to-digital conversion (ADC)

Following the approach used by many hardware-efficient platforms previously proposed (e.g. [179, 180]), simulated extracellular recordings are filtered prior to data conversion. Analogue Filter Design library (in Matlab) is used to simulate three standard filter types: Elliptic, Butterworth, and Bessel. The low pass cut-off frequencies range from 1-9 kHz, while the high pass cut-off frequencies are set to 100, 200, . . ., 900 Hz. The number of poles tested for the filter implementations range from 1-6. The data is filtered using causal filter implementations in order to simulate analogue filtering performed realtime on-node. In addition, the signals are also filtered non-causally (zero-phase response) to compare with offline analysis. The non-causal filtering is implemented using 'filtfilt' function in Matlab.

Different sampling rates are simulated by downsampling the filtered data. The sampling frequencies are set to 28 kHz, 14 kHz, 7 kHz, 4 kHz, and 2 kHz. For the cases where the filtered

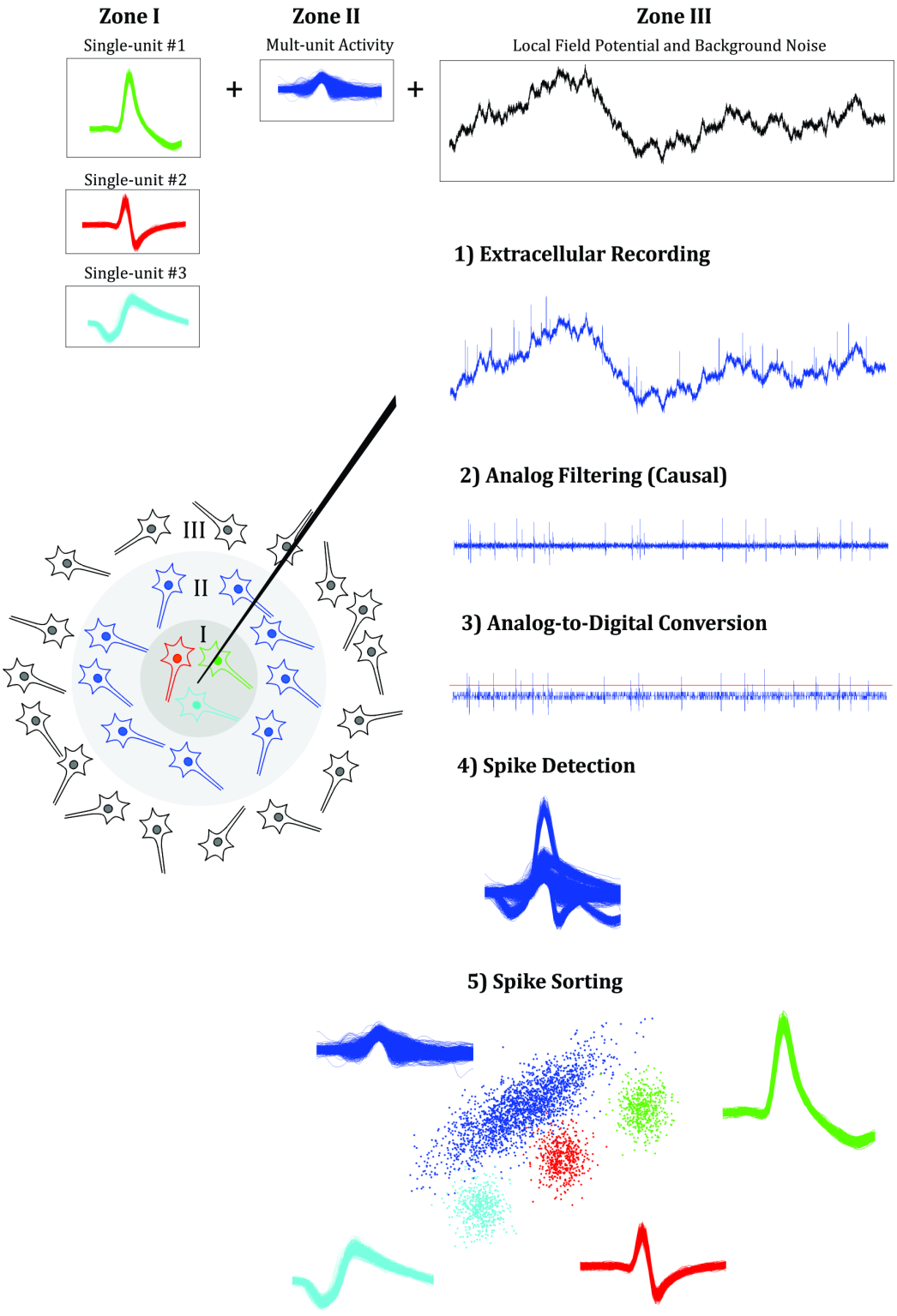


Figure 4.22.: Extracellular recording creation and data processing. Recordings are created (Step 1) by modelling and adding the contributions of single units (Zone I), multi-units (Zone II), and local field potentials and background activity (Zone III). Recordings are then filtered under various specifications (Step 2), while analogue-to-digital conversion is simulated by downsampling and quantizing the signal (Step 3). This is then followed by spike detection and sorting (Steps 4 and 5).

data is downsampled below Nyquist frequency (sampling frequencies of 4, 2 and 1 kHz), an additional Elliptic lowpass filter (2^{nd} order) is used to prevent aliasing. The resolution of the virtual ADC is set to 4, 6, . . ., 16 bits with a fixed signal range of $\pm 500\,\mu$V.

4.2.1.3. Spike detection

The spikes are detected using the unsupervised amplitude thresholding method proposed by [77]. The threshold (Thr) is automatically set to:

$$Thr = 4\sigma_n \text{ where } \sigma_n = median\left\{\frac{|x|}{0.6745}\right\} \tag{4.7}$$

Slightly lower or higher thresholds (from 3 to 5 times σ_n) yielded similar results (see Section 4.2.2.1). Spike detection performance is calculated by:

$$Performance = P_d \times \theta(P_d) \tag{4.8a}$$

and

$$P_d = (1 - \frac{N_e}{N_{su}}) \tag{4.8b}$$

where N_{su} is the number of single-units, $\theta(x)$ is the unit step function, and N_e is the number of errors. N_e represents the total number of missed single-units and false positives. The spike detection performance is 100% if and only if the number of errors is zero, and 0% if the N_e is equal or higher than N_{su}.

4.2.1.4. Evaluation of clustering and template building

Clustering is performed using "Wave Clus", an unsupervised[2] method described in [77]. In order to prevent biases introduced by user subjectivity, the completely unsupervised solutions provided by "Wave Clus" are used. Each detected spike is aligned to its peak and stored (64-samples per spike). Then, features of the spike shapes are extracted using wavelet transform, and Kolmogorov-Smirnoff test of Normality selects the coefficients that separate the spike waveforms best. Ten coefficients with the least Normal distribution (i.e. most likely to represent more than one cluster of spike shapes) are chosen. Finally, the spikes are classified using super-paramagnetic clustering [77, 181, 182].

Clustering performance is evaluated according to the criteria proposed in [62], and quantified in terms of number of *hits*, *misses* and *false positives*. *Hits* refer to the correctly identified clusters available in the recording, while *misses* refer to the available clusters which are not identified by "Wave Clus". The *false positives*, on the other hand, refers to identified clusters which do not correspond to any actual clusters. These may arise from false partitioning of the

[2] Within the context of spike sorting, the term "unsupervised" indicates that no a-priori information regarding data and the number of neurons is available to the clustering method.

units present in the recording [62]. A perfect spike detection is assumed during evaluation. The criteria that defines *hits*, *misses* and *false positives* are as follows [62]:

- Hits: A cluster is identified as a hit if it fulfills two conditions: (1) at least 50% of the spikes within identified cluster correspond to the same neuron; and (2) the number of detected spikes for identified cluster is, at least, 50% of the total number of spikes generated for this particular neuron.
- Misses: The number of misses is quantified as the total number of generated neurons (i.e. three for all the simulations in this study) minus the number of hits.
- False positives: All detected clusters that are not hits are considered as false positives.

Clustering performance is given by:

$$Performance = P_s \times \theta(P_s) \tag{4.9a}$$

and

$$P_s = (1 - \frac{N_e}{N_{units}}) \tag{4.9b}$$

where N_{units} is the number number of single-units present in the recording (i.e. three for all simulations), $\theta(x)$ is the unit step function, and N_e is the number of errors. N_e represents the total number of misses and false positives. The clustering performance is 100% if neither false positives nor misses are present in the clustering results. In the circumstances where the number of errors (either false positives or misses) is equal or greater than the number of neurons simulated, the clustering performance is 0%.

4.2.1.5. Real-time template matching

Template matching is investigated under 5 different metrics for real-time classification. Given y_i is the spike and T_i is the mean template (both of length n), these are:

- Squared Euclidean Distance: $d = \sum_{i=1}^{n}(y_i - T_i)^2$
- Norm 1: $d = \sum_{i=1}^{n}|y_i - T_i|$
- Norm Infinite: $d = max_i(|y_i - T_i|)$
- Mahalanobis: $d = (\vec{y} - \vec{T})^t S^{-1}(\vec{y} - \vec{T})$ where S is the covariance matrix (size $n \times n$) of a given template, defined as:

$$S = \frac{1}{m-1}\sum_{j=1}^{m}(T^j - \vec{T})(T^j - \vec{T})^t \tag{4.10}$$

where Tj is a column vector of length n representing the jth out of the m observations made to create the mean template T. In other words, Tj is the jth out of the m spikes in the training set for this particular cluster (mean template T is created with spikes coming from a training set, see last paragraph of Section 4.2.1.1).

- Nearest neighbours: Given a spike detected in the testing set, its Squared Distance to all spikes detected in the training set is calculated. The 5 training spikes with closest distance (i.e., the 5 nearest neighbours) provides one vote, and the test spike is assigned to the most common class among those 5 votes. Since there are four clusters per simulation (one multi-unit and three single-units) and five neighbours (hence 5 votes), it is possible for two different clusters to have the same votes (i.e. 2 votes each, and the remaining vote to a third cluster). Under such circumstances, two clusters (with 2 votes) are randomly chosen to assign the spike.

In order to assess the template matching performance only, perfect template creation is assumed. Accuracy for each template matching metric is evaluated by calculating the proportion of spikes that are classified correctly. In addition to assessing accuracy performance, computational demands for each metric is estimated by measuring the machine time required to classify all spikes in a given dataset[3]. However, it should be noted that the absolute time measured depends on computer specifications, hence the relative times — which are informative of relative complexity of the metrics— are reported. It should also be emphasized that these are first order approximations as the measured time for Matlab implementations may differ from those of the actual on-node implementations.

4.2.1.6. Validation of the method using real data

Minimum requirements established (in Section 4.2) are tested and validated through real extracelluar recordings from human medial temporal lobe. These are recorded from five patients implanted with ten electrodes for clinical reasons [183]. The equipment used and the signal characteristics are same as those described in Appendix B.2. From each electrode, three recordings (two-minutes long) are obtained. One of the traces is used to test spike detection (as in Section 4.2.2.1), another one for testing template building (as in Section 4.2.2.2), and the last one is used for testing template matching (as in Section 4.2.2.3). The recordings consist of two single-unit and one multi-unit activity. The mean single-unit firing rate is 4.2 ± 0.9 Hz.

Initially, highest ADC specifications (28 kHz sampling rate and 16-bit resolution) and non-causal filtering (i.e. offline data processing) are used to carry out spike detection, spike sorting (template building), and template matching. Then, following the procedure described in Section 4.2.1.2, the sampling rate and resolution are set to 7 kHz and 10-bit, while a causal Elliptic filter (2^{nd} order) with a passband of 300-3000 Hz is implemented. Using the results obtained with highest specifications as the "ground truth", the spike detection (Eq. 4.8) and sorting (Eq.

[3]Machine time using implementations in Matlab with a PC (3.47 GHz clock and 48 Gb RAM)

4.9) performances are quantified. Squared Euclidean Distance is used as the template matching metric (described in Section 4.2.1.5).

4.2.2. Results

4.2.2.1. Effect of basic signal features on spike detection

First, the effect basic signal features on spike detection performance is assessed. As previously described, a positive threshold is used on amplitude (see Eq. 4.7). For spike detection, several threshold settings (slightly higher or lower) are tested and found out to be not affecting the results. In fact, threshold values between 3 and 5 times σ_n reveal similar detection performance (see Supp. Fig. A.2).

Prior to passing through virtual ADC, the simulated data are filtered under various specifications. These are implemented as causal filters in order to mimic analogue filters used in realtime processing, and the results of these are compared to ones obtained via non-causal implementations (see Section 4.2.1). Figure 4.23 presents the spike detection performance (for the simulations with lowest SNR) of all three filter types under varying high-pass cut-off frequencies (causal and non-causal). For investigating the effect of high-pass cut-off frequency, the low-pass cut-off frequency is kept constant at 3 kHz while the filter order is set to 2. The results reveal that non-causal filtering has better spike detection performance (Figure 4.23A). This difference in detection performance is due to the distortion of the spike shapes by causal filtering. In fact, there has been previous research which has shown that causal filters create spurious distortions in the spike waveforms [64]. The results clearly show that this phase distortion (see Figure 4.23B for an example) impairs detection performance, and it gets worse with increasing high-pass cut-off frequency (Figure 4.23C). The observations are valid for all SNRs and for the Bessel, Butterworth, and to a lesser degree, Elliptic filters.

To investigate the effect of low-pass cut-off frequency, the high-pass cut-off is fixed to a value that provided the best performance for each filter type (i.e. 300 Hz for the Elliptic filter, 200 Hz for the Butterworth filter, and 100 Hz for the Bessel filter). In the meantime, the filter order is set to 2. The results reveal that low-pass cut-off frequencies equal or larger than 3 kHz achieved nearly optimal performance under all SNRs (Supp. Figure A.3A). Then, following a similar approach, the low-pass cut-off frequency is fixed to 3 kHz and the number poles are varied. It is observed that spike detection performance deteriorated as the filter order is increased (Supp. Figure A.3B). This attributed to larger 'ringing' effects of higher order causal filters [64], accentuating the artificial rebound observed in Figure 4.23B, and hence lowering the amplitude of spike waveform further.

In addition to effects of filtering, the impact of sampling parameters on detection performance is also investigated. For this test, an Elliptic filter (2^{nd} order) with a passband of 300 Hz-3 kHz — that achieves 100% spike detection performance at 28 kHz sampling rate and 16-bit resolution — is used. The original datasets are downsampled and quantised according to procedure described in Section 4.2.1.2. The results reveal that 7 kHz is the minimum sampling rate at which the

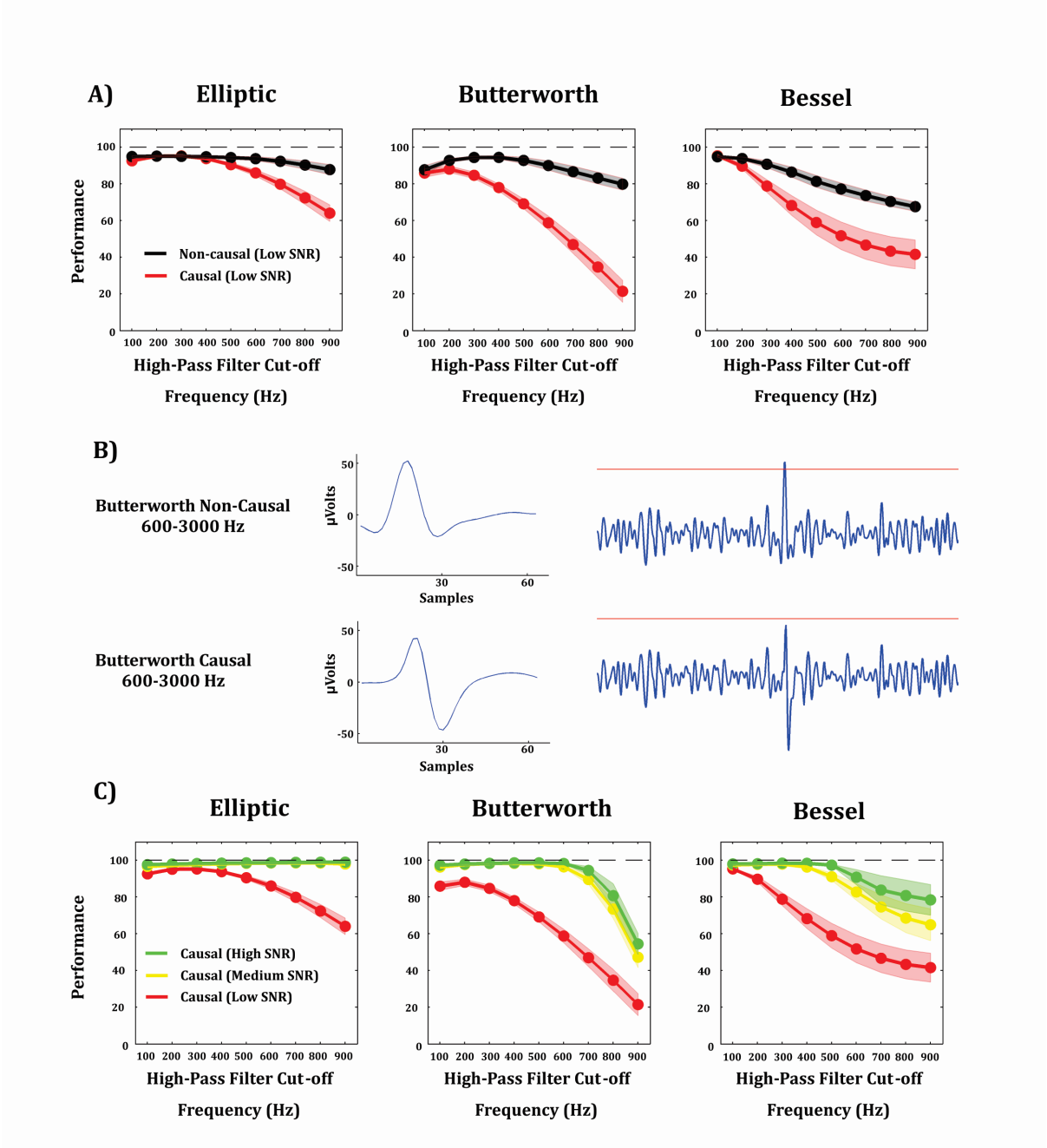


Figure 4.23.: Impact of filtering on spike detection. (A) Spike detection performance of non-causal filters (black line) and causal filters (red line) under various high-pass cut-off frequencies for the Elliptic, Butterworth, and Bessel filters (simulations with low SNR). (B) An illustration of causal filters introducing phase distortion effects. For a given detection threshold, the spike acquired with non-causal filtering is detected (top-right panel) while the spike acquired with causal filtering is not detected (low-right panel). (C) Spike detection performance with causal Elliptic, Butterworth, and Bessel filters for all SNR levels.

detection performance is not significantly degraded for all SNRs. On the other hand, the minimum number of bits that allows accurate spike detection is 6 bits which is equivalent to a signal resolution of 15.6 μV/bit (Figure 4.24B). This finding is not explained by the fact of setting the threshold using high-precision calculations in Matlab, given that implementing the same approach but with limited precision yielded exactly the same results.

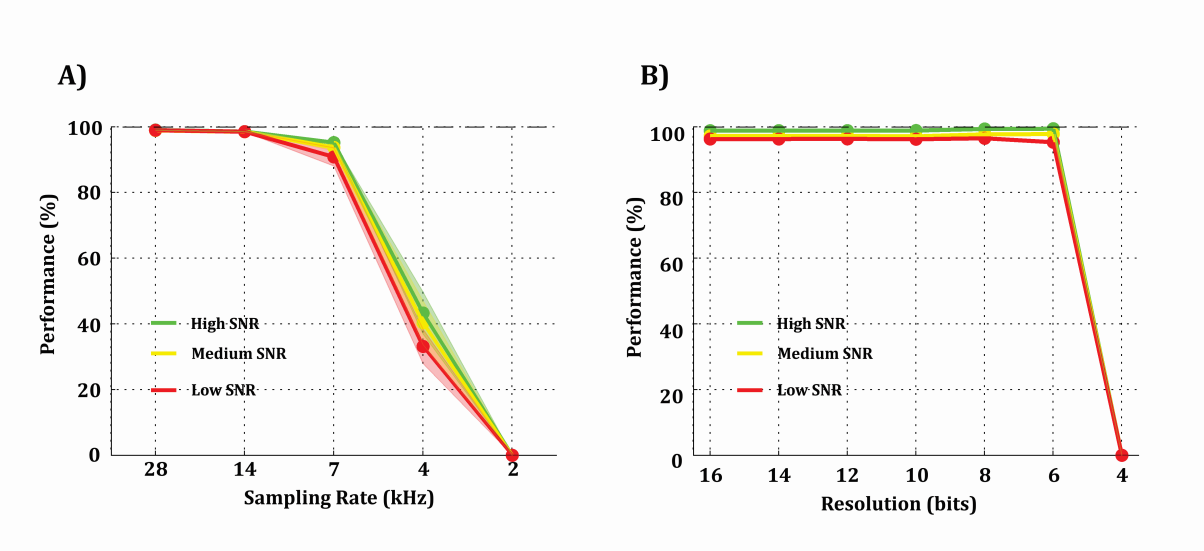


Figure 4.24.: The sensitivity of spike detection performance to sampling rate and resolution. Spike detection performance at: (A) various sampling rates, and (B) various signal resolutions. These are tested for high (green), medium (yellow), and low (red) SNRs.

4.2.2.2. Effect of basic signal features on spike sorting and template building

Spike sorting is the process of clustering spikes coming from different neurons according to their shape or features [184]. The two most complex tasks of any clustering algorithm are: (1) Estimating the number of neurons in the recording [62], and (2) defining the proper templates to search for. Here, the performance of a widely used off-line spike sorting method, "Wave Clus" [77], is assessed under different specifications. This test aims to establish minimum requirements — minimising power consumption — that neural recording circuitry must have in order to ensure high spike sorting performance.

During this test, datasets are filtered using an Elliptic filter (2^{nd} order) with a passband of 300 Hz-3 kHz[4]. The spike sorting performance under various sampling rates and resolutions is presented in Figure 4.25. Although the spike sorting performance is dependent on SNR, the results reveal similar trends. In fact, one can observe that all spike sorting performances are qualitatively similar for sampling rates above 7 kHz, while below 7 kHz the performance significantly degrades (Figure 4.25A). Similarly, sorting performance deteriorates for signal resolutions below 0.97 μV/bit (i.e. 10 bits over $\pm$500 μV input range), while higher values result in (prac-

[4]Identical results are obtained with different filter specifications

tically) unaltered sorting accuracy (Figure 4.25B). Figure 4.25C illustrates how spike shapes from different neurons end up being mixed due to limitations in sampling(for more details see Supp. Figure A.4).

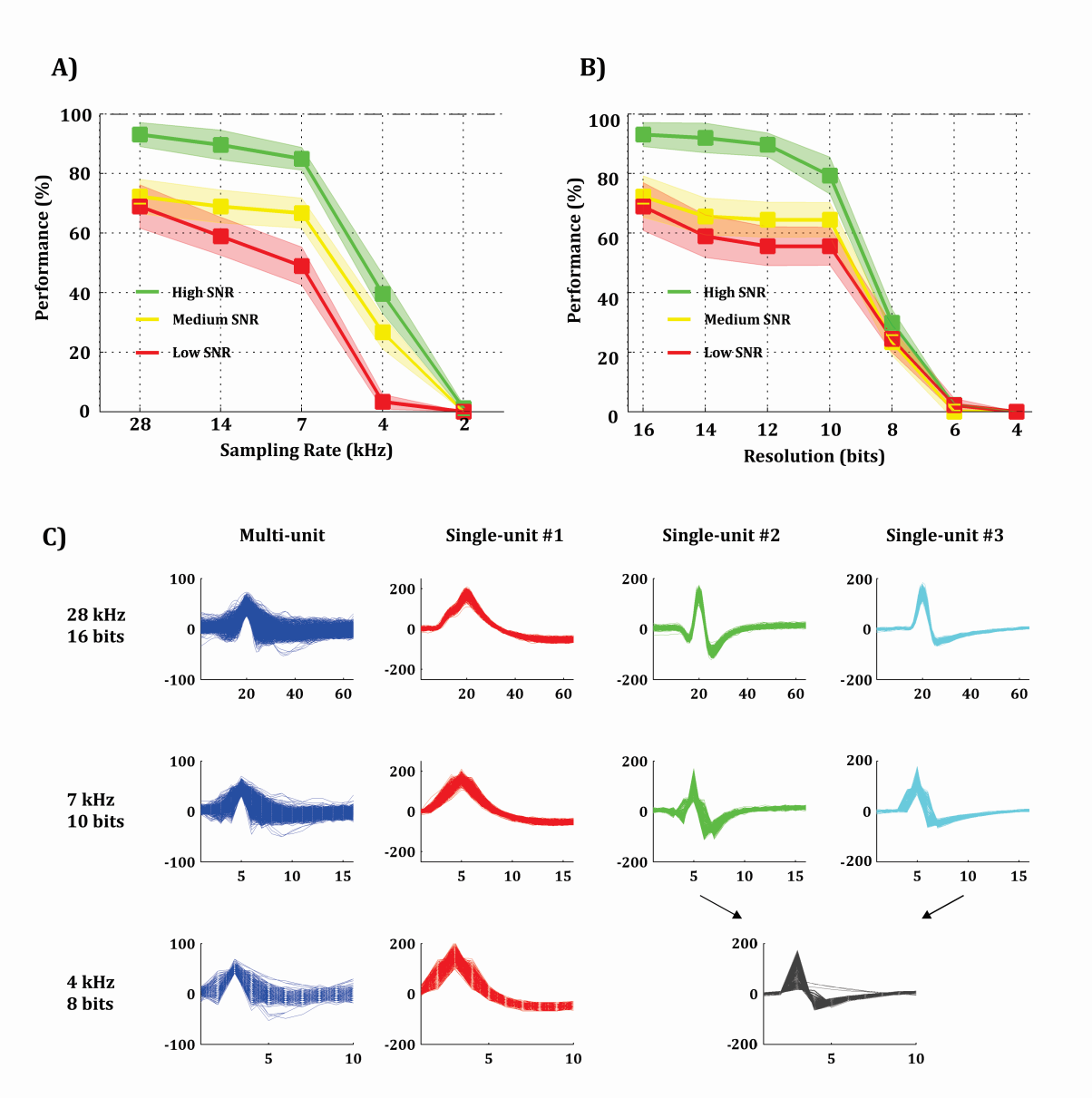


Figure 4.25.: The sensitivity of spike sorting and template building to sampling rate and signal resolution. Spike sorting performance at various: (A) sampling rates, and (B) signal resolutions. These are tested for high (green), medium (yellow), and low (red) SNRs. (C) An illustration of two separate clusters (single-unit #2 and #3) being incorrectly merged together due to limitations in sampling rate and resolution. In the given example, sampling rate of 7 kHz and 10-bits signal resolution ensures correct identification of all available clusters.

Based on these observations, it is argued that sampling signals at 7 kHz with 10-bit resolution (signal range of $\pm 500\,\mu$V) can provide sorting performance almost as good as those obtained

with (commonly used and power demanding) higher specifications. In fact, the proportion of the misclassified spikes obtained at 28 kHz with 16-bit resolution (typically 6 out of 3797 for the highest SNR, see Supp. Table A.1) are similar to those sampled at 7 kHz with 10-bit resolution (typically 20 out of 3797). (Please refer to Supp. Table A.1 for more information on misclassified spikes under varying values of SNR, signal resolution and sampling rates.)

4.2.2.3. A hybrid strategy for real-time spike sorting

To perform real-time on-node spike sorting, a hybrid approach consisting of two-stages is modelled (Figure 4.26). During the initial stage, i.e. template building, a low-power platform performs filtering, analogue-to-digital conversion, and detects the incoming spikes in real-time. As the spikes are detected, they are wirelessly transmitted to an external receiver to extract spike features and perform clustering in order to build the spike templates. The second stage follows transmitting the templates back to the low-power platform, and involves performing real-time spike sorting via template matching (Figure 4.26B). During this stage, similarly to first, the low-power implantable platform performs signal conditioning (i.e. filtering and ADC) and real-time spike detection. Each detected spike is then compared to templates of the relevant channel, and assigned to the most similar one (template matching).

The feasibility of this approach is tested by assessing the sorting performance and computational requirements of various template matching metrics under varying sampling conditions (Figure 4.27). Templates are constructed using two-minute long recordings described in Section 4.2.1.4 (training set), while the template matching performance is evaluated using a different set of simulations with identical parameters (testing set, see Section 4.2.1). Template matching performance is defined as the fraction of correctly classified spikes in a given dataset.

First, the impacts of sampling rate on different template matching metrics are investigated. Figure 4.27A presents the template matching performance of all tested metrics using simulations with medium SNR. The results show a smooth decay in performance for all, as the sampling rate is decreased. Especially, it is observed that template matching performance equal or larger than 80% can be achieved (for all metrics) at 7 kHz — the minimum sampling frequency which is previously shown to provide a good spike detection and sorting accuracy (see Section 4.2.2.1 and 4.2.2.2). In a similar analysis, the impacts of signal resolution are also studied. Figure 4.27B reveal that 6 or more bits (signal resolution of $\geq$15.6 μV/bit) achieve almost identical performance for template matching results. The same behaviour is observed for all three levels of noise for the different sampling rates and resolutions (see Figure 4.27C and D for Squared Euclidean Distance, and Supp. Figures A.5 and A.6 for other metrics).

It is clear that some of the tested template matching metrics are more difficult to implement efficiently in hardware. For example, Nearest Neighbours require computing distances with all the spikes detected in the training set, while Mahalanobis distance involves multiplication by $n \times n$ covariance matrix. Therefore, the trade-off between computational requirements and template matching performance need to be carefully assessed. Figure 4.27E presents the template matching performance for different metrics against their computational complexity simulated at

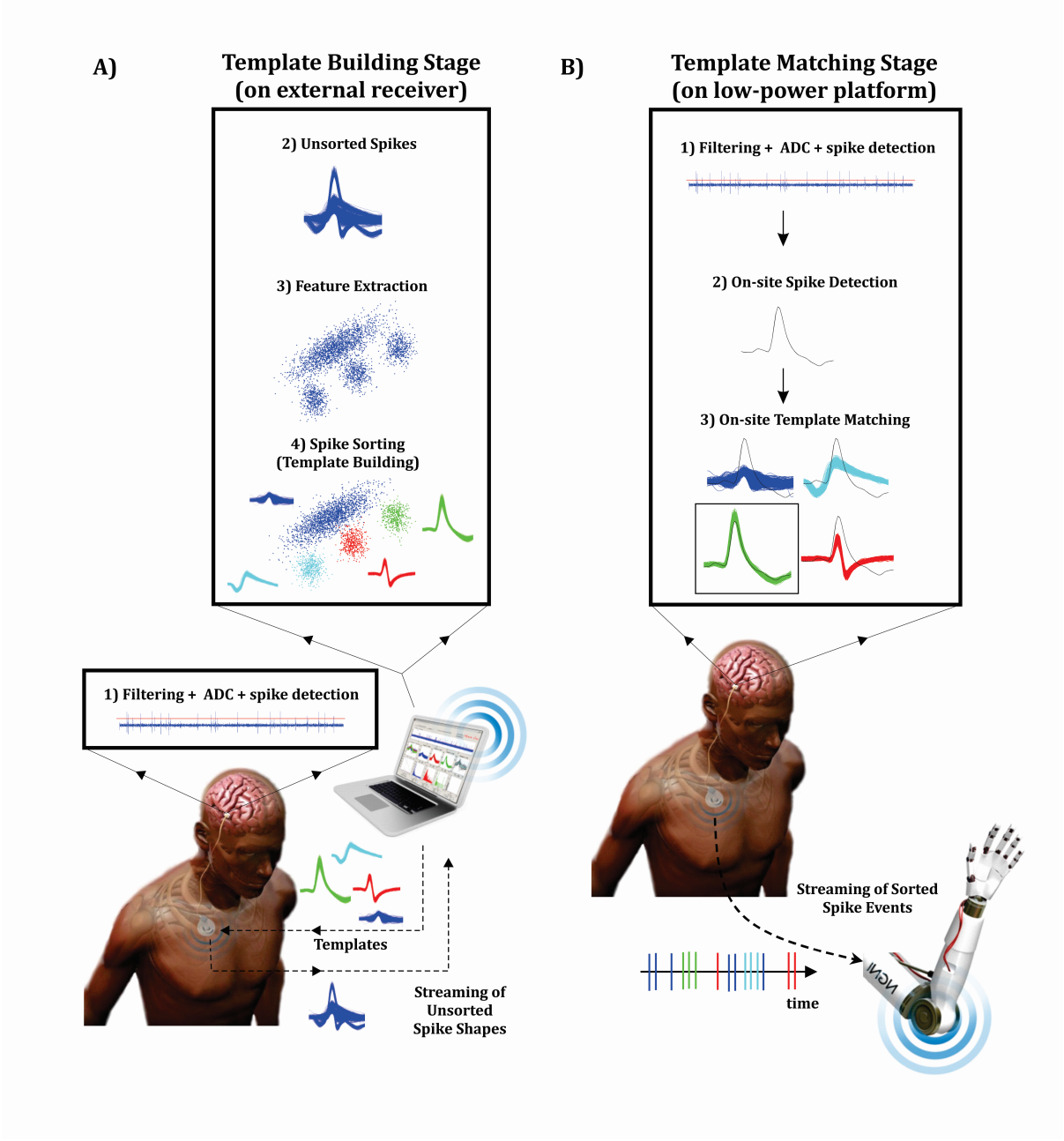


Figure 4.26.: Illustration of two-stage hybrid strategy for real-time on-node spike sorting. (A) Template building stage (off-node) and (B) Template Matching stage (on-node). During the template building stage, the spikes are detected in real-time (on-node) following signal conditioning (i.e. filtering and ADC), and then transmitted to a computer where templates are built and sent back to the low-power platform. During the template matching stage, the detected spikes are compared with these templates to achieve real-time classification.

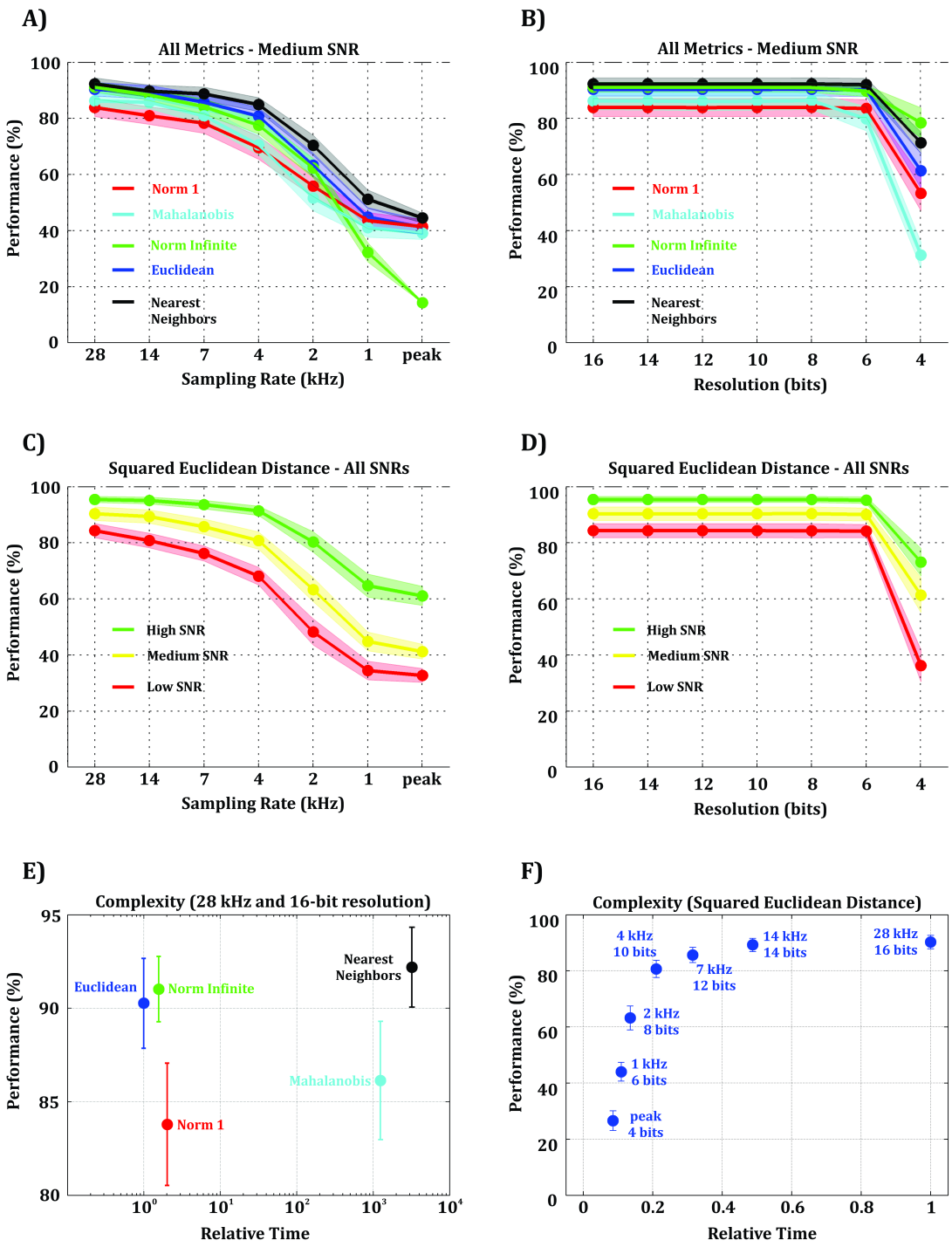


Figure 4.27.: The sensitivity of real-time template matching performance to sampling rate and resolution. Performance of all metrics at various (A) sampling rates, and (B) signal resolutions. Performance of Squared Euclidean Distance at various (C) sampling rates, and (D) signal resolutions. (E) Relative complexity of each metric for signals with 28 kHz and 16-bit resolution. Values are normalised to the metric with least complexity (Squared Euclidean Distance). (F) Relative complexity of the Squared Euclidean Distance for different sampling rates and signal resolutions. Values are normalised to the specifications with highest complexity (28 kHz and 16-bit resolution).

Table 4.3.: Computational complexity in number of operations.

TM Method	Additions	Multiplications	Total Cost**
Euclidean	$[2n-1]\cdot K$	$n\cdot K$	2013
Norm1	$[3n-1]\cdot K$	-	501
Norm Infinite	$[2n-1]\cdot K$	-	333
Mahalanobis*	$[n^2-1]\cdot K$	$[n^2+n]\cdot K$	105×10^3
Nearest Neighbours	$[2n-1]\cdot T + Tlog(T)$	nT	4×10^5

*The computations required for calculating covariance matrix, S, are excluded.
**Total cost is calculated as: Total Cost = # of Add. + 10 × # of Mult.
K: # of neurons (3), n: # of samples per spike (56), T: # of spikes in training set

the highest specifications (sampling rate of 28 kHz and 16-bit resolution). As described in Section 4.2.1, the computational complexity is estimated by measuring the machine time required to classify spikes. Since absolute time measured depends on computer specifications, the results are given in terms of "relative time" which is the machine times normalised to the method with least complexity (i.e. relative time of least complex method is 1). It should be emphasized once again that these are first order approximations since Matlab and actual hardware implementations may differ. Machine time measurements can be influenced by the specifics of the function implementations in Matlab as well as current state of the computing device.

The results reveal similar performances in template matching accuracy for all metrics, however, Nearest Neighbours and Mahalanobis distances are at least three orders of magnitude more computationally demanding than the others (Figure 4.27E). However, given the limitations of computing time measurements discussed above, it is difficult to reach a conclusive comparison among Squared Euclidean distance and Norm methods. Therefore, in addition to computation time, computational demand for each method is also assessed in terms of number of operations (i.e. additions and multiplications) required to compute them. Table 4.3 confirms previous results, showing that Nearest Neighbours and Mahalanobis distances are at least three orders of magnitude more demanding than others. On the other hand, Squared Euclidean is approximately four times more costly[5] to implement than Norm1, while Norm Infinite is the least computationally demanding method.

Furthermore, the computational complexity of different metrics are estimated (in terms of computation time) for signals with varying sampling conditions as well (Figure 4.27F and Supp. Figure A.7). It is observed that computational complexity can be reduced by at least five times — compared to the most power-demanding specifications (i.e., 16-bit resolution and 28 kHz of sampling rate) — without a noticeable reduction in performance by selecting optimal parameters.

A crucial step prior to template matching, that ensures an accurate point-wise comparison with the templates, is spike alignment (e.g. to their peak). Considering that the proposed method is inspired by its possible application to low-power architectures, it is possible that a

[5]Cost of computation time is calculated based on number of additions and multiplications at K=3 and n=56 (See Table 4.3 for details). Each multiplication is considered to be ten times more costly to implement. Cost function is calculated as # of additions + 10 × # of multiplications.

certain degree of error might be introduced by the alignment method. In order to assess the sensitivity of the proposed approach to varying levels of template-spike misalignment, a jitter of s is introduced in the peak alignment. This is achieved via artificially shifting the spikes by a number of samples from a uniform distribution U(-s,s). For example, a jitter value of 4 shifts the spikes — either left or right — by a number of samples equal or lower than 4, drawn from a uniform distribution. In other words, this corresponds to peak-alignment method having an error of no more than 4 samples. The results show that template matching performance is very sensitive to alignment errors, particularly the Norm Infinite and Mahalanobis distance (Figure 4.28A). For the rest, a small degree of misalignment (∼0.125 ms) can still provide a reasonable accuracy (80%), however, larger alignment errors degrade the performance by 30% or more. Especially, a 4 sample-error at 28 kHz (Figure 4.28A), a 2-sample error at 14 kHz (Supp. Figure A.8A), or a 1-sample error at 7 kHz (Supp. Figure A.8C) deteriorated the accuracy by approximately 20%. These results indicate that a good peak alignment technique is essential for accurate on-line spike sorting via template matching.

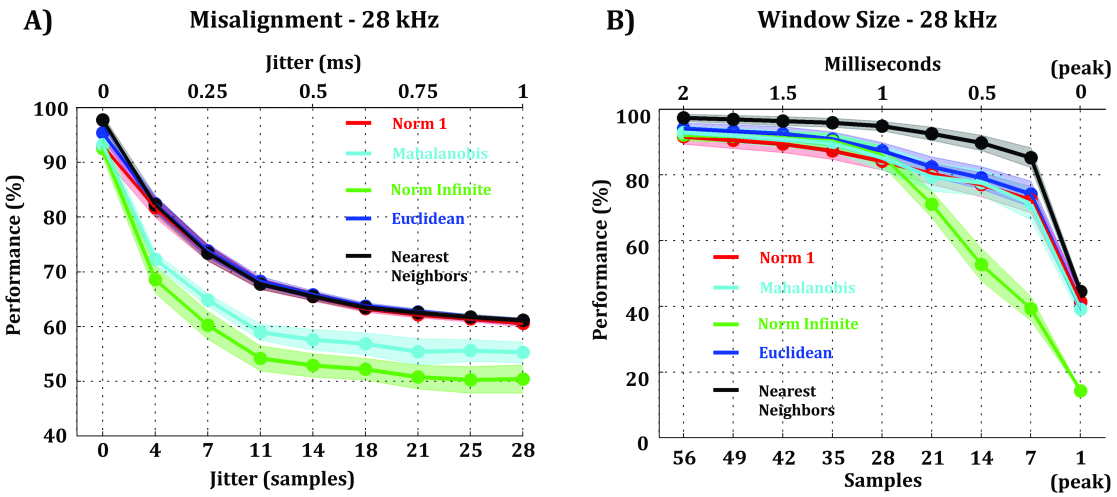


Figure 4.28.: The sensitivity of template matching performance to misalignment and template window size. Template matching performance of different metrics at various: (A) jitter values in the peak alignment of the spikes, and (B) template window sizes. For both (A) and (B), data was sampled at 28 kHz with 16-bit resolution.

The last investigation explores the optimal window size for accurate template matching. Spike shapes are symmetrically shortened by reducing the number of data points used, and then the template matching performance for each time window is calculated. Figure 4.28B presents the performance of different metrics as the window size is decreased. It is observed that, a 0.5 ms window size corresponds to a performance loss of typically 20% (10% for Nearest Neighbours) while it reduced the amount of data needed to be stored (and perform template matching on) by at least four times. This corresponds to a window size of 14 samples at a sampling rate of 28 kHz (Figure 4.28B), 7 samples at 14 kHz (Supp. Figure A.8B), or 4 samples at 7 kHz (Supp.

Figure A.8D). Compared to other metrics, Norm Infinite is more sensitive to window size (performance loss of 20% at 1 ms window size). These results, combined with the misalignment analysis (Figure 4.28A) indicate that the Norm Infinite is a less reliable metric than Squared Euclidean distance.

4.2.2.4. Assessing the feasibility of the method using real data

Having established the minimum requirements, this section aims to validate the proposed method using real extracellular recordings, recorded from the human medial temporal lobe (see Section 4.2.1.6). To perform this test, ten datasets are obtained to test spike detection (as in Section 4.2.2.1) and template building (training set, as in Section 4.2.2.2). Another ten datasets (obtained from different times in the same recordings) are used to test real-time template matching (testing set, as in Section 4.2.2.3).

An example of the clusters, obtained in one of the datasets under highest specifications (sampling rate of 28 kHz with 16-bit resolution) and offline data processing (i.e. non-causal filtering), is shown in Figure 4.29A. Figure 4.29B, on the other hand, presents the same but utilising the minimum signal requirements established previously (7 kHz and 10-bit resolution). In order to simulate real-time processing, a causal filter (i.e. 2^{nd} order Elliptic filter with passband of 300-3000 Hz) is used. The performance of the method is quantified by assuming that the results of highest specifications are correct, and comparing them with the ones obtained using minimum requirements (Figure 4.26). The results (obtained using minimum requirements) show that the spike detection accuracy is (80 ± 3)%, spike sorting accuracy is (81 ± 11)%, and template matching accuracy is (88 ± 4)% (Figure 4.29C). Although a ∼20% performance loss is observed for minimum requirements, ∼85% savings in the amount of data to be processed is achieved (64 data-points × 16 bits = 1024 bits per spike, vs. 16 data-points × 10 bits = 160 bits per spike).

4.2.2.5. Power Consumption

The actual applicability of the proposed strategy relies on the fact that power requirements of this hypothetical chip are reduced compared to existing devices. For example, if the reduction in the amount of transmitted data comes at a cost of performing computationally demanding calculations (hence increasing overall power consumption of the chip), then such a method will very likely fail to provide a useful hardware implementation.

For the spike sorting approach simulated here, the simplicity of the calculations indicate that power saving due to data reduction overcomes the power spent for performing those calculations. Based on the power requirements reported in low-power neural interfaces [21, 23, 185, 186], the power consumption of the simulated hypothetical chip can be estimated to assess its feasibility. Assuming an analogue front-end (AFE) performing amplification, filtering, and ADC at 7 kHz and 10-bit resolution, the power consumption of this step is estimated to be 10 μW per channel [21]. Considering 100 channels, 100 neurons (firing at 10 Hz), and a communication energy of

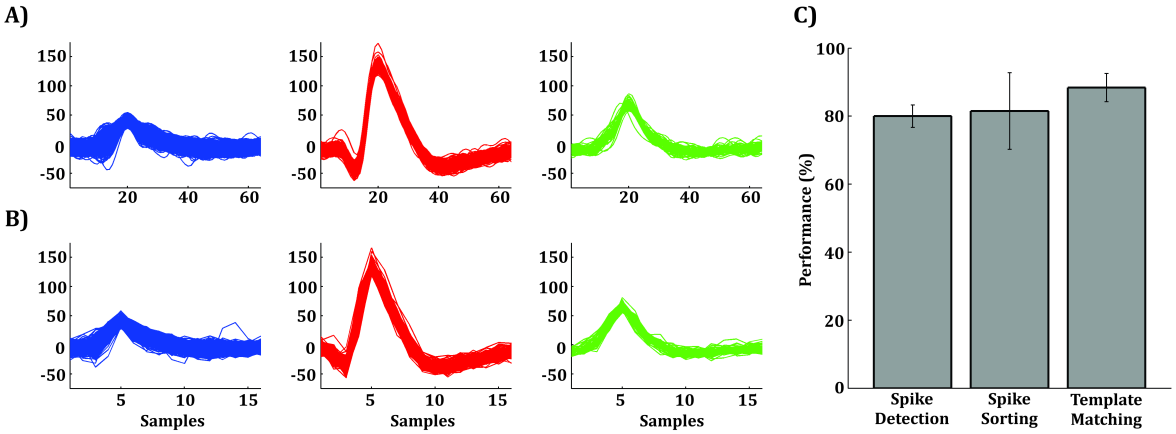


Figure 4.29.: Validation of established minimum requirements with real data, obtained from the human medial temporal lobe. (A) Clusters are detected using the highest specifications (28 kHz and 16-bit resolution) and offline data processing (i.e. non-causal filters). (B) Same as (A) but using the minimum signal requirements established (7 kHz and 10-bit resolution) and real-time data processing (i.e. causal filters). Detected clusters represent multi-unit activity (blue cluster) and two single units (red and green clusters). (C) Spike detection, spike sorting, and template matching performance obtained with real extracellular recordings.

20 nJ/bit[6] [187], streaming the raw signals without any data reduction consumes ∼141 mW. This power consumption is significantly decreased to ∼4.9 mW, if realtime spike detection is included (spending an extra 2 μW per channel [188]), thus only streaming detected spikes.

Further including the final step described in this work, i.e. spike sorting via template matching — for which an extra 8 μW per channel is estimated [188] —, the overall power consumption of the device is estimated to be 2.5 mW. Hence, in principle, real-time template matching reduces the power requirements by a factor of ∼60 compared to sending raw data with optimal specifications (7 kHz and 10-bit resolution), whereas it shows even more significant decrease in power consumption if it is compared with the most power-demanding — yet typically used — specifications (28 kHz and 16-bit resolution). Detailed breakdown of estimated power consumption for different modalities of data transmission is presented in Table 4.4[7][8][9][10]

[6]This figure is estimated in between WiFi and Bluetooth Low Energy (BLE). WiFi energy efficiency is ∼5.25 nJ/bit, while BLE is 150 nJ/bit.

[7]Power estimations use minimum signal requirements established in this work: 7 kHz and 10-bit resolution.

[8]Raw Data Transmission Power = AFE Power + Communication Power = (10 μW × 100 channels) + (10 bits × 7 kHz × 100 channels) × 20 nJ/bit.

[9]Detected Spike Transmission Power = AFE Power + Spike Detection Power + Communication Power = (10 μW × 100 channels) + (2 μW × 100 channels) + [(10 bits × 16 samples/spike + 8 bits/channel id. + 16 bits/spike timestamp) × 100 neurons × 10 Hz] × 20 nJ/bit.

[10]Sorted Data Transmission Power = AFE Power + Spike Detection Power + Spike Sorting Power + Communication Power = (10 μW × 100 channels) + (2 μW × 100 channels) + (8 μW × 100 channels) + [(8 bits/channel id. + 16 bits/spike timestamp) × 100 neurons × 10 Hz] × 20 nJ/bit.

Table 4.4.: Power consumption estimations for different modes of data transmission

Transmission Mode	**AFE Power**	**Spike Detection Power**	**Spike Sort Power**	**Communication Power**	**Total Power**
Raw Data	1 mW	-	-	140 mW	141 mW
Detected Spikes	1 mW	0.2 mW	-	3.7 mW	4.9 mW
Sorted Spikes	1 mW	0.2 mW	0.8 mW	0.48 mW	2.5 mW

4.2.3. Discussion

4.2.3.1. Minimum signal requirements for spike detection and sorting

Over the past years, many unsupervised spike sorting algorithms — that hardly require any user intervention — have been proposed [77,78,80,95,103,168]. Due to their computational demands, most of these algorithms are run on power-demanding platforms (such as PCs). However, recent research efforts have pursued to develop hardware efficient spike sorting methods for implantable low-power platforms [78, 141, 165, 179]. Within this context, Section 4.2 have explored the impacts of basic signal features (such as filtering, sampling rate, and resolution) on the accuracy of real-time spike sorting.

First, it was observed that causal filtering introduce phase distortions that may deteriorate spike detection using an amplitude threshold. This finding is in line with previous research that has shown phase distortions distort the waveform of extracellular action potentials, causing an artificial bi-phasic shape [64]. In the same study, it is also shown that zero-phase response (i.e. non-causal filters) avoids such distortions. However, non-causal filtering is not always an option, in particular for real-time on-node applications. Implementing a transform function to cancel non-linear phase response of the filter could be considered as an alternative approach. However, this strategy places increased computational demands which might not be desirable for low-power applications. In this study, it has been shown that the performance loss due to causal filtering could be minimised by careful selection of filter parameters. The results revealed that a second order filter with lower high-pass cut-off frequency reduced the distortions (hence maintained performance).

The results also showed that a good spike detection and sorting performance ($\geqslant$80% for high SNRs) can be achieved using a sampling rate of 7 kHz and a signal resolution of 10 bits (in a fixed signal input range of $\pm 500\,\mu$V). Current data acquisition systems employ sampling and resolution parameters that exceed (by far) these minimum requirements. This could possibly be explained by the fact that computational power was not considered a bottleneck until very recently.

Typically, extracellular spikes have been recorded over short sessions lasting a few hours. However, recent improvements in neural recording technology and chronic electrode arrays now enable recording sessions over days, or even months [189, 190]. These developments, together with the prospect of recording from hundreds of channels, result in significant amount of data which is challenging to store and process. For example, recording from hundred channels in

one day at a sampling rate of 28 kHz and 16-bit resolution results in approximately 500 GB of data. The results of this work show that amount of data can be reduced by a factor of 6.4 (e.g. sampling at 7 kHz with 10 bits) without a noticeable information loss about the spiking activity of individual neurons (Figure 4.25).

4.2.3.2. Real-time hardware-efficient spike sorting

Traditionally, neuroscience experiments employed wires passing through the scalp and the skin in order to relay the recorded spiking activity to an external receiver. However, this approach exposes subjects to risks of infection. In addition, such tethered connection limits the free movement of the subject thus hindering long-term studies in natural surroundings. Therefore, implantable recording devices with wireless telemetries have been proposed by several groups [179]. However, bandwidth and power limitations restrict the number of channels that can be monitored. Therefore, optimal specifications that can minimise power consumption need to be explored. In this work, it has been shown that sampling requirements (i.e. sampling rate and resolution) can be significantly reduced increasing the number of channels monitored.

An important step in minimising hardware requirements is the analogue filtering of the recordings (prior to ADC) in order to achieve optimal bandwidth for spike detection and sorting. This decreases the power requirements due to reduced dynamic range of the signal, but at the cost of removing the information carried by low-frequency local field potentials. Alternatively, digital filtering (for band separation) could be implemented after ADC (hence increased dynamic range) in order to enable recording and transmission of local field potentials.

There have been various approaches for reducing the amount of data transmitted to allow more channels to be monitored. One strategy involves transmitting the spikes that are being detected in real-time [180], while other works perform a feature extraction step prior to transmission in order to achieve further dimensionality reductions [191]. On the other hand, there have been more ambitious strategies that proposed on-node spike sorting to reduce transmitted data into a stream of binary spike events [119, 179, 192]. For example, [179] presented an implantable chip that performs (i) amplification and analogue filtering, (ii) ADC at 12 kHz,(iii) spike detection with a programmable micro-controller, and (iv) spike sorting with a window discriminator before data streaming or closed-loop stimulation. Such neural interfaces, despite being supervised and based on a single channel, has been shown to induce long-term plasticity in the motor cortex of freely behaving non-human primates [189]. Therefore, this study aimed to evaluate the feasibility of a method that is capable of monitoring multiple channels utilising an unsupervised sorting algorithm [77]. The presented method aims to achieve real-time on-node spike sorting in two steps. First, spikes are detected in real-time and transmitted to an external receiver where mean template waveforms are created using WaveClus (template building stage). In the second stage, real-time spike sorting is achieved through template matching. The key strategy of the proposed hybrid approach is to mitigate all computationally demanding calculations outside the chip, while real-time processing is done via template matching using the templates created in the first step. Considering that template matching is computation-

ally inexpensive compared to defining templates, this mixed strategy exploits the advantages of combining offline and online processing.

Having computationally modelled this hypothetical chip, its performance and complexity was evaluated under various specifications using realistic simulations of extracellular recordings [89, 157]. This analysis was carried out for different template matching metrics, and the results revealed that Squared Euclidean Distance strikes a good balance between computational requirements and accuracy. The remaining template matching metrics tested were either less robust to noise levels (i.e. Norm 1, see Supp. Figure A.4A and B), or less robust to the selection of small window sizes (i.e. Norm Infinite, see Figure 4.28B), or too complex to be implemented in hardware (i.e. Mahalanobis and Nearest Neighbours, see Fig. 4.27E). In addition, sensitivity to spike alignment was also investigated, and the results revealed that good peak alignment is necessary prior to template matching. This is due to the fact that template matching is a point-wise comparison of the mean template and the detected spike.

4.2.3.3. Hardware implementation

As discussed previously, in addition to minimising data specifications and developing integrated circuits with less power requirements, there are several other issues arising in real recordings that need to be addressed for the realisation of this approach. For example, observed spike waveforms can exhibit large variations and even suddenly disappear due to microcscopic electrode shifts [179, 189, 190]. For the proposed hybrid strategy, this would require to perform re-calibration (as in Figure 4.26A) in order to update the available templates for real-time sorting (Figure 4.26B). This drawback is nevertheless common to most implantable systems developed to date [128, 188, 193, 194], but see [188] for one exception.

A possible solution to this issue is to actively track (online) the stability of the recordings utilising one or more indicators of performance loss. For example, in BMI applications involving control of a device (or a prosthetic), decoding performance could be an obvious indicator [34]. However, for exploratory neuroscience and other applications such as research in freely behaving animals [189, 195, 196], performance indicators should be independent of the existence of a decoder. Under such circumstances, a threshold on template matching distance could be used. This threshold could be set in way that if the distance between a spike and all templates exceeds such value, then the spike is considered as misclassified. By tracking the evolution of misclassified spikes, the system could be set to request re-calibration if the proportion misclassified spikes exceeds a certain number. Exact values for these thresholds will certainly depend on the stability of the recording and the specific application.

The results obtained in this study clearly showed that a good alignment method is crucial to achieve accurate real-time template matching (Figure 4.28A). Previous spike sorting methods dealing with this problem have proposed to upsample the spike waveforms after detection in order to improve peak alignment [77]. However, upsampling introduces more complexity, thus increasing the power requirements of the implantable platform. Hence, alternative approaches have been proposed to achieve spike alignment in an efficient manner [197, 198]. For example,

a low-power ($\leqslant 1\,\mu$W) integrated circuit that performs spike alignment have been presented by [198]. Therefore, such low-power circuits will be required for hardware application of the method described in this study.

The minimum requirements established for spike detection can be extended to other systems using amplitude thresholding as their spike detection method [74, 134, 199, 200]. In addition, it has also been shown that the established minimum requirements are robust to several widely used template matching metrics (Figure 4.28A and B), regardless of their relative complexities (Figure 4.28E). It is argued that minimum specifications required for the template building stage would be similar to ones obtained with other algorithms; however, a systemic comparison with other methods is beyond the scope of this study.

The method introduced in this work has many potential applications. For example, it can be used in closed-loop stimulation designs [201,202], as well as being well-suited for animal research considering the proposed method would allow long-lasting recordings without the necessity of percutaneous wires (hence reducing risk of infections and increasing mobility for the subjects) [179, 189, 195, 196]. Similarly, current research towards next-generation BMIs is focused on implementing implantable low-power platforms with wireless telemetries [74,75,119]. In addition to possible applications for basic neuroscience and clinical applications, the proposed method can also be applied to all low-power systems employing template matching [203, 204], such as ECG implantable systems [205].

It has been shown that using hardware specifications that are more power efficient than typical ones, only $\sim$20% of the performance is lost (see Section 4.2.2.5). However, it should be noted that acceptability of this degradation in performance depends on particular application of the method. For example, performance may be prioritised over power reduction within the context of neuroscience experiments on non-human primates [189, 190]. On the other hand, decoding approaches for controlling assistive devices (or prosthetics) [34] might prioritise low power implementations to monitor larger numbers of channels. Furthermore, the allowable performance loss will depend on the specific decoding algorithm used. Nevertheless, the performance profiles for all tested parameters are presented in the analysis so that designers can choose a compromise that would suit the needs of the particular application.

4.3. Conclusion

Within the context of finding a viable solution for real-time on-node spike sorting for wireless BMIs, this chapter has investigated the merits of a two-stage hybrid approach which combine both off-line and on-line data processing. The first stage involves on-node detection and transmission of spikes to an external computer for template creation. Upon downloading calculated templates back on-node, the second stage aims to spike sort incoming spikes via template matching.

Template matching has been suggested to be consistent [110, 115] and having comparable performance to other techniques [111], including traditional PCA [166]. On the other hand,

some of the criticisms towards hardware implementability of the method involved its computational complexity [110] and poor performance in noisy conditions [167]. In response, there have been many works which has demonstrated that by employing various techniques computational demands of template matching can be significantly reduced [110]. However, performance (both computational and accuracy) and memory requirements — which is the biggest concern for template based operations — depend on many parameters.

Therefore, Section 4.1 quantified the key metrics and parameters associated with template-based spike sorting to provide a comprehensive assessment on the trade-offs between accuracy and hardware resources (computation and memory). These parameters are interpolation and alignment, template creation, template matching quantifiers, and template window size. Additionally, the effects resolution and bandwidth (parameters associated with front-end) on different template matching quantifiers are also investigated to re-confirm the trends observed in previous sections, and to assess performance of different template matching quantifiers under these parameters.

In Section 4.2, aforementioned two-stage hybrid platform is modelled and its feasibility is assessed. Using realistic simulations of extracellular recordings (synthetic data mimicking real extracellular recordings) as described in [89,157], this work aimed at computationally testing its practicability and explored specifications that optimise performance for minimising the signal requirements for future hardware implementation. More specifically, while the previous section investigated template-matching only, here the system is analysed as a whole quantifying parameters from the perspective of spike detection and off-line template building stage (i.e WaveClus) as well.

The results of both studies (discussed in detail previously, hence not included in this section) quantified and validated a set of minimum signal requirements that minimised hardware resources to allow real-time on-node spike sorting applications.

5. Hardware Efficient Real-time Spike Sorting using Derivative Features

For achieving on-node spike sorting efficiently in hardware, there exists two approaches: template-based and feature-based spike sorting. In the previous chapter, the feasibility of a template-based two-stage hybrid approach have been established through quantifying parameters relevant to template matching and various stages of the proposed hybrid approach (such as detection and clustering).

In feature based spike sorting approach, the main trade-off involves introducing extra processing to reduce the dimensionality of the spikes, thus minimising storage requirements. This extra processing comprises selection of a subset of critical features — that differentiate the neural signals — usually preceded by an initial mathematical transformation. In addition to reducing storage requirements, such approach may also reduce overall power consumption. It has been shown that, for neural recording systems that utilise complete neural waveforms, power consumption is dominated by memory requirements [123]. However, successful realisation of feature based approach depends on the complexity of the feature extraction method. Traditional approaches such PCA and wavelet transform demand too much processing power which outweigh the possible reductions in area and power consumption arising from lower memory requirements. Within this context, the key research questions that this chapter focuses on are:

1. What is the simplest mathematical transformation that would yield useful features for hardware-efficient real-time spike sorting?
2. How can features be efficiently extracted?
3. How can efficient real-time classification be achieved by these features?

In the light of the first and second research questions, a simple feature extraction method based on first and second derivative extrema of the spike waveforms is proposed in Section 5.1[1]. The computational requirements and spike sorting accuracy of the proposed method are quantified and compared against commonly used feature extraction methods. The proposed method achieves a relatively high spike sorting accuracy with very low computational requirements, hence is suitable for low power hardware implementation. In Section 5.2, focusing on the first and second research question, an improved set of features hypothesized based on EAP

[1]The work presented in Section 5.1 was done in collaboration with Dr. Sivylla Paraskevopoulou and Mr. Mohammed R. Saberi. Mr. Saberi and I did the initial concept and work together, and later collaborated with Dr. Paraskevopoulou to take the initial study further and test it.

waveform dynamics is proposed. It is found that a subset of features given by local extrema and inflection points of EAP waveform and its derivatives provide the majority of separation between neurons. Based on these features, it is established that a combination of EAP waveform and second derivative features provide the highest immunity against varying noise levels. Building on Section 5.2, a new hardware-efficient spike sorting method is proposed in Section 5.3, addressing the three key research questions mentioned above. The proposed method incorporates raw waveform and second derivative features, and employs channel-specific selection of near-optimal subset of features. These features are determined through off-line training, while real-time feature extraction is simply done by using the index positions (relative to EAP peak) of the feature subset determined during training. Real-time classification, on the other hand, is achieved through simple comparisons between extracted features and pre-determined (during offline training) thresholds. The performance, as well as computational complexity, of the proposed method is quantified and compared to commonly used feature extraction methods, and shown to provide the best spike sorting accuracy and computational complexity trade-off. Finally, it is implemented and demonstrated on an ARM Cortex M0+ based embedded platform.

5.1. Derivative-Based Spike Sorting

The initial motivation for using derivative features is based on the dipole model of an action potential [61]. Eq. 5.1 describes the potential generated by a dipole of charge $\overrightarrow{p}$ at location $\overrightarrow{r}$ from the centre of the dipole.

$$V(\overrightarrow{r}) = \frac{\overrightarrow{p}\,\overrightarrow{r}}{4\pi\epsilon_0 r^3} = \frac{pr \cdot cos\theta}{4\pi\epsilon_0 r^3} \tag{5.1}$$

where p is the dipole operator, r is the electrode-to-neuron-distance, $\cos\theta$ is the cosine of the angle between the two vectors $\overrightarrow{p}$ and $\overrightarrow{r}$, and ϵ_0 is the dielectric constant. This dipole is formed by the extracellular current balancing the influx of Na^+ cations from the neuronal extracellular to the intracellular space [47], thus maintaining equilibrium at the membrane.

The extracellular potentials generated by a spiking pyramidal cell and measured at different locations in it's vicinity are illustrated in Fig. 2.5 (see Chapter 2). An initial observation is that the orientation of the electrode relative to the spiking neuron changes the spike shape, and its distance reduces the amplitude of the spikes.

Based on this observation, the paradigm that a recording electrode and two identical spiking neurons (i.e. with identical morphology) are aligned in the x-direction, but the distance between the electrode and the first neuron is smaller than the distance between the electrode and the second neuron, is examined. In that case, the recorded spikes from the distinct neurons will exhibit 'identical' spike shapes but differ in amplitude. In fact, the spike amplitude will be decreasing inversely proportionally to r^2, according to Eq. 5.1. As the two recorded spikes have the same expansion in time and different amplitude, the spike amplitude can be used as a feature to separate the two spikes. However, this difference will also be evident in their first derivatives,

additionally providing some indication to the spike morphology. Therefore, intuitively the first derivative appears to be an eligible feature for spike sorting.

5.1.1. Materials and Methods

The previous section provided an intuitive rationale behind using derivatives to extract spike features for subsequent classification. The use of such features has been shown analytically [206], and through simulations [129, 206]. Derivatives have also been proven an efficient filter for noise shaping (highlighting the recorded spiking activity against background spiking activity) [206]. In this section, the efficiency of a new feature extraction method based on using both the first and second derivatives will be explored. All the methods reported herein are tested using Mathworks Matlab v7.12.

5.1.1.1. Proposed method for featured extraction

The derivatives are computed as the difference between the current and previous sample points of the spike waveform (s), according to Eq. 5.2 for the first derivative (FD) and Eq. 5.3 for the second derivative (SD).

$$FD(n) = s(n) - s(n-1) \tag{5.2}$$

$$SD(n) = FD(n) - FD(n-1) \tag{5.3}$$

Fig. 5.1 illustrates the effect of taking the first and second derivatives of typical action potential signals. Here, onto the spike waveform are annotated the occurrences of the following features: (a) negative, and (b) positive peaks of the first derivative (FD_{min} and FD_{max}), and (c) negative, and (d) positive peaks of the second derivative (SD_{min} and SD_{max}). In this example, it can be clearly observed that these features together can be used to distinguish between these three spike profiles.

The first derivative of any function (or geometric object) can be described by its rate of change (or slope). Therefore, the second derivative is essentially the rate of change of the slope, representing the curvature of the signal. Spikes are characterised by several morphological changes including slopes, curvature and amplitude, all of which are dependent on the observation point relative to the neuron. The first and second derivative can therefore provide, directly and indirectly, information on all three of these expected morphological variations.

5.1.1.2. Other feature extraction methods (for comparison)

To validate the viability of the proposed method for hardware spike sorting, its accuracy and computational complexity is compared in Section 5.1.2 to other low complexity and derivative-based feature extraction methods, in addition to Principal Component Analysis (PCA), which is widely regarded as a 'gold standard' for spike sorting.

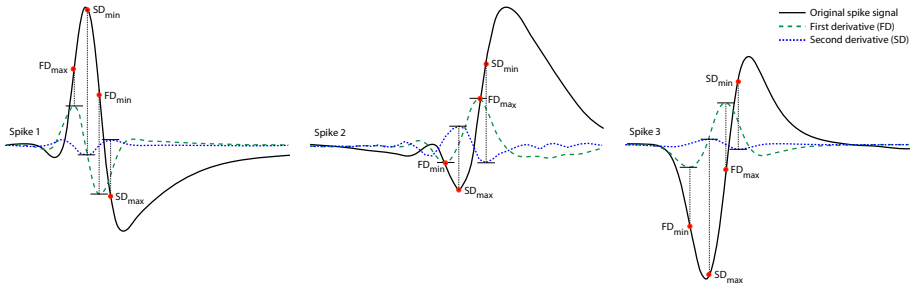


Figure 5.1.: Typical action potential waveforms with their first and second derivatives (FD and SD). Annotated onto the original spike waveforms are the points corresponding to the extrema (i.e. minimum and maximum) of the first and second derivatives.

PCA PCA is a technique that extracts linearly uncorrelated components via orthogonal transformations from a set of observations (spike waveforms in this case). These principal components are the eigenvectors of the autocorrelation matrix of the original signal, and their number is equal to the number of samples. However, the most significant information is contained in the first few components.

Discrete derivatives (DD) Based on computing the derivatives at each sample point of the spike waveform, Discrete Derivatives are given by Eq. 5.4 [96]

$$dd_\delta(n) = s(n) - s(n-\delta) \tag{5.4}$$

where s is the spike signal, n the sample point, and δ the time delay. In [129], three different values for the delay $\delta = 1, 3, 7$ are used. Therefore, the dimensionality of the feature space increases by a factor of three compared to the original number of samples of the spikes. To decrease the dimensionality, the most significant DD coefficients are selected. This method has already been shown to be hardware implementable and demonstrated on a 64-channel DSP [129].

First derivative (FD) [207] proposes a method based on the derivation of the spike waveform in the discrete domain with selected features being the positive and negative peak of the first derivative of spike waveform along with the peak of the spike itself. In the following sections, this work will be referred as the FD method.

Autoregressive model (AR) [208] has shown that by modelling spike signals using an autoregressive (AR) model of p-th order (the coefficients are computed using the Burg algorithm), signals can be separated from background activity . Here, the coefficients of the AR model for each spike is used as features for spike sorting. The AR model is described by Eq. 5.5

$$x_t = c + \sum_{(i=1)}^{p} \phi_i x_{t-1} + \epsilon_t \tag{5.5}$$

where $\phi_1, ..., phi_p$ are the parameters of the model, c is a constant and ϵ_t is white noise.

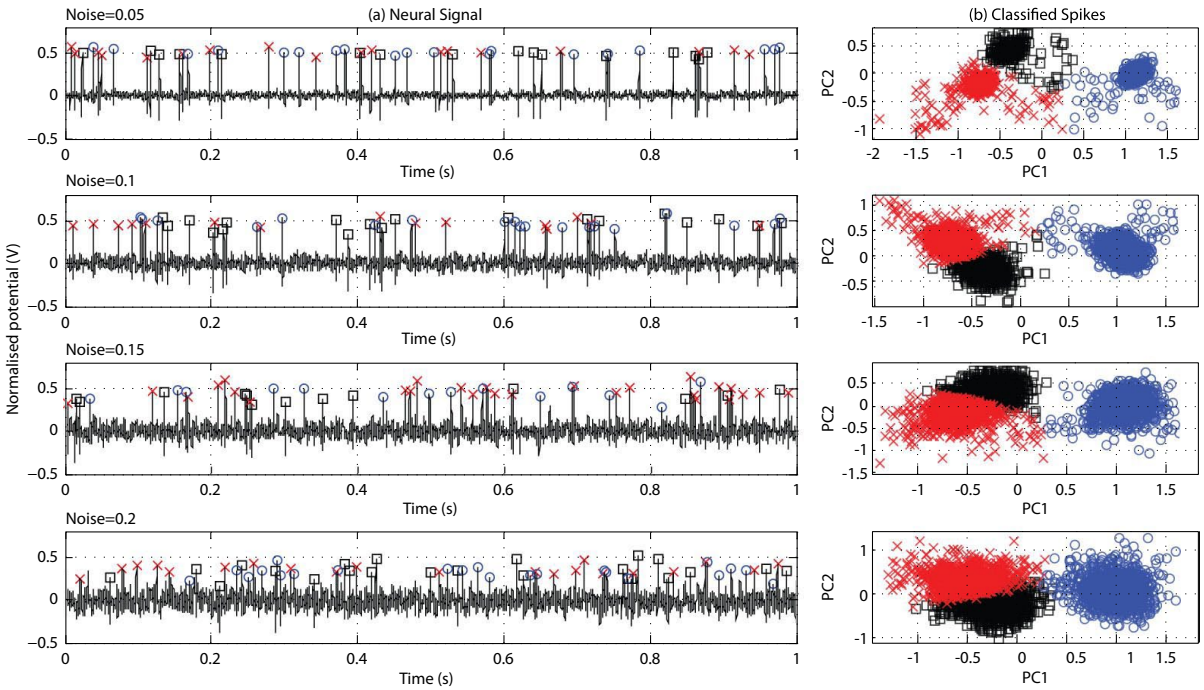


Figure 5.2.: Test dataset 2 showing: (a) a 1 s segment at noise levels of 0.05 to 0.2 (from top to bottom), and (b) a two dimensional representation of the clustering space when using three principle components (z-axis is 3rd principle component - not shown).

Point-to-point (PP) comparison This method utilises all of the samples of the spike waveform (i.e. with no dimensionality reduction). This method is herein referred to as point-to-point comparison (PP).

5.1.1.3. Classification (using k-means clustering)

Following feature extraction, k-means clustering is used to differentiate the features. The objective is to partition the spikes into k clusters (within the selected feature space), in which each spike is assigned to the cluster with the nearest mean, as described by Eq. 5.6.

$$J = \sum_{j=1}^{K} \sum_{n \in S_j} \|x_n - \mu_j\|^2 \tag{5.6}$$

where J is the objective function (i.e. the squared error function), n is the number of spikes, (x_1, x_2.. x_n) is the set of spike features (each of m-dimension), K is the number of clusters ($K \leq n$) and S_j are the different sets (i.e. spike classes).

Generally implemented as an iterative algorithm that converges towards a solution, k-means aims to determine the clusters such as to minimise the objective function J. The number of iterations required for convergence varies depending on the feature types and their dimensionality. For all classifications presented, the Matlab ('kmeans') function was used. In order to ensure that all methods converge to a near-optimum classification accuracy, the number of iterations

Table 5.1.: Comparison of the classification error for different combinations of derivative features. Results given across all the tested datasets and for all noise levels.

Dataset	**Noise†**	**Classification Error**						
		Method 1 (m=3)*	Method 2 (m=3)*	Method 3 (m=3)*	Method 4 (m=3)*	Method 5 (m=2)*	Method 6 (m=2)*	Method 7 (m=4)*
Dataset 1	0.05	0.0544	0.0561	0.0546	0.0538	0.0544	0.0572	0.0544
	0.10	0.0443	0.0468	0.0451	0.0446	0.0457	0.0477	0.0446
	0.15	0.0552	0.0581	0.0546	0.0555	0.0555	0.0601	0.0552
	0.20	0.0489	0.0535	0.0498	0.0492	0.0507	0.0596	0.0492
Dataset 2	0.05	0.0531	0.3457	0.0531	0.0519	0.0522	0.0575	0.0528
	0.10	0.0528	0.1247	0.0585	0.0517	0.0699	0.0622	0.0526
	0.15	0.0630	0.1850	0.0718	0.0504	0.2987	0.0988	0.0627
	0.20	0.0757	0.2283	0.0993	0.0729	0.3369	0.1469	0.0822
Dataset 3	0.05	0.0562	0.0437	0.0553	0.0550	0.3491	0.1339	0.0559
	0.10	0.0525	0.0415	0.0519	0.0522	0.3457	0.1180	0.0522
	0.15	0.0628	0.0441	0.0576	0.0619	0.3491	0.1489	0.0565
	0.20	0.0682	0.0539	0.0665	0.0940	0.3453	0.2293	0.0636
Dataset 4	0.05	0.3517	0.0386	0.3496	0.0562	0.3505	0.0612	0.0552
	0.10	0.0477	0.0576	0.0511	0.0552	0.3463	0.0661	0.0474
	0.15	0.0974	0.1180	0.3488	0.1282	0.3535	0.1308	0.0968
	0.20	0.1970	0.3458	0.2906	0.1829	0.3561	0.2030	0.1575
Total		**0.0863**	**0.1151**	**0.1099**	**0.0697**	**0.2350**	**0.1045**	**0.0649**

†Noise level relative to a normalised signal (i.e. noise=1/SNR), *m=number of features/dimensions

are set to 10.

5.1.1.4. Test datasets

To evaluate the spike sorting accuracy for feature extraction methods, four simulated neural datasets each containing three single-units at different SNR levels are used [105]. For details on datasets, please refer to Appendix B.1.

Figure 5.2 illustrates a sample portion of Dataset 2 with varying noise levels (from top to bottom). Raw datasets (on left) and cluster plots (on right) using the first three principal components are shown for each noise level. In addition, different spike classes (known *a priori*) are also annotated (in red, blue and black). It should be noted that although only a 1 s portion of the inputs is shown, the cluster plots include all the spikes from the respective datasets. It can be observed that increasing the noise level (i.e. decreasing the SNR) adversely affects the cluster separation.

5.1.2. Results and Discussion

In this section, the proposed method is compared to other hardware implementable methods (previously described) in terms of its performance (accuracy) and computational complexity.

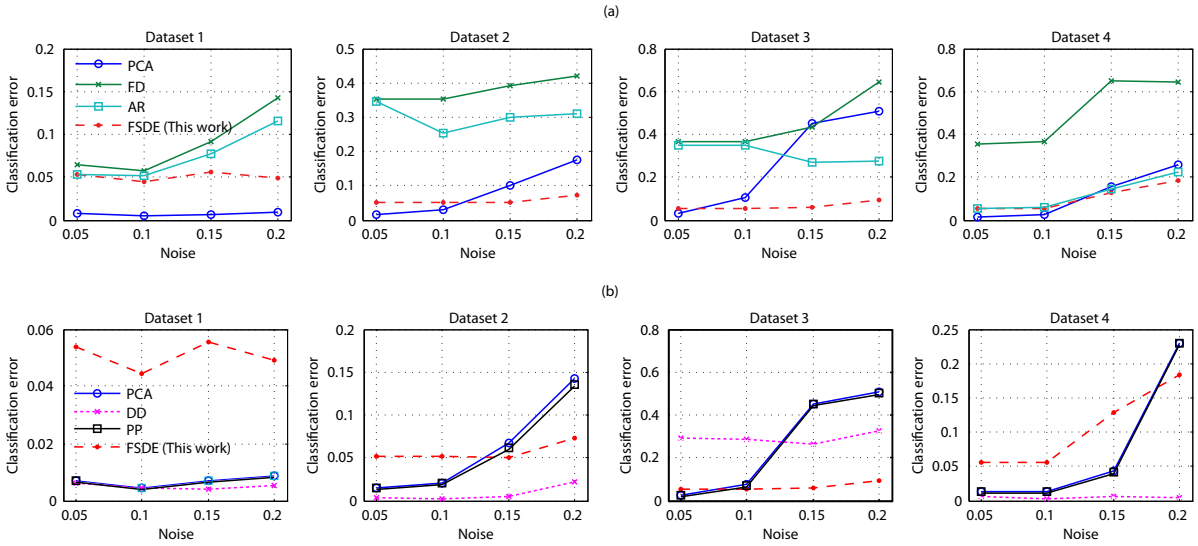


Figure 5.3.: Comparison of the classification error between commonly used feature extraction methods (across all datasets and noise levels). Shown are comparison to methods with: (a) the same feature space dimensionality, and (b) different (higher) feature space dimensionality.

5.1.2.1. Classification accuracy for different derivative feature combinations

To validate the efficiency of using first and second derivative features (derived as described in Section 5.1.1.1), the classification accuracies of six different feature combinations are compared. These are as follows:

- Method 1: The negative and positive peaks of the first derivative (FD_{min} and FD_{max}), along with the negative peak of the second derivative (SD_{min}).
- Method 2: The negative and positive peaks of the first derivative (FD_{min} and FD_{max}), along with the positive peak of the second derivative (SD_{max}).
- Method 3: The negative peak of the first derivative (FD_{min}), along with both the negative and positive peaks of the second derivative (SD_{min} and SD_{max}).
- Method 4: Positive peak of the first derivative (FD_{max}), along with both the negative and positive peaks of the second derivative (SD_{min} and SD_{max}).
- Method 5: Range of the first derivative (i.e. (FD_{max}-FD_{min}), along with range of the second derivative (i.e. SD_{max}-SD_{min}).
- Method 6: Average of the first derivative (i.e. (FD_{min}+FD_{max})/2) along with average of the second derivative (i.e. (SD_{min}+SD_{max})/2).
- Method 7: All first and second derivative features, i.e. both the negative and positive peaks of the first and second derivatives (FD_{min}, FD_{max}, SD_{min} and SD_{max}).

As previously mentioned, the main criteria quantified is the classification error. This is given by the number of misclassified spikes over the total number of spikes. However, together with misclassification error, one should also consider the relative complexity of each method, i.e. the number of features taken. The results are presented in Table 5.1. The total error stated has been quantified as the average classification error across all datasets and noise levels.

Table 5.1 shows that Methods 6, 4, and 7 achieve the lowest classification errors for 2, 3 and 4 features respectively. As expected, the more features are used, the lower the classification error. However, assuming that the relative complexity between these methods is proportional to the number of features, and then comparing the relative classification errors, a much greater improvement in accuracy is observed when moving from 2 to 3 features than from 3 to 4.

Therefore, Method 4 is chosen as it provides a good trade-off between the number of features used and the classification error, in addition to exhibiting good noise immunity. From here on, the combination of features used in Method 4 will be referred to as the First and Second Derivative Extrema (FSDE) method.

5.1.2.2. Classification accuracy using FSDE features compared to other feature extraction methods

As mentioned in Section 5.1, the size of the feature space (m) directly impacts the computational complexity of spike sorting. Therefore, the classification accuracy of FSDE feature extraction with k-means has been considered separately for methods with the same and different m, shown in Figures 5.3(a) and 5.3(b) respectively. The results have been averaged across all noise levels.

- Methods with the same feature space as FSDE (i.e. $m \leq 3$): The selected methods are FD, PCA_3, and AR_3 (i.e. subscript denoting the number of coefficients).
- Methods with a higher feature space than FSDE (i.e. $m > 3$): The selected methods are PCA_{10}, DD_{21}, and PP_{64}. For PCA, the number of coefficients are limited to 10, as there is a negligible improvement in classification error for more coefficients. For DD, 21 coefficients, which has been reported to be the optimum [129], are used. The coefficients are selected through a blind training of 300 spikes per dataset (roughly 10% of the dataset) to identify the coefficients with the largest variance. Finally, for PP all the sample points are used (i.e. m=64).

The results in Figure 5.3 reveal that FSDE generally achieves higher classification accuracy than other methods of the same feature space. On the other hand, methods with a higher feature space generally achieves a better classification accuracy. Another important observation is that the FSDE and DD methods (across all feature space) exhibit the best noise immunity amongst all methods.

Moreover, to directly establish the relationship between dimensionality and classification error, dimensionality factor is defined as the number of features over the number of samples in the spike waveform (i.e. m/n). This is illustrated in Figure 5.4, showing the classification

Table 5.2.: Computational complexity of each feature extraction method (in terms of the number of arithmetic operations).

Method	Additions	Multiplications	Dimensionality Reduction
FD	N-1	-	-
SD	2N-3	-	-
DD†	3N - 11	-	(3N-11)(1-5T) add., T(6N-22) mult.‡
PP	-	-	-
AR	$[N-1]+\sum_{i=1}^{p}5(N-i)+i+1$	$[N+1]+\sum_{i=1}^{p}5(N-i)+i+3$	-
PCA	N^2+2N+1	N^2+N	-

N is the number of samples in each spike, T is the number of spikes for training, and p is the model order for the Autoregressive model.
†Additional complexity associated for dimensionality reduction, ‡variance calculation

error averaged across all datasets and noise levels. It is observed that the classification error of FSDE is comparable to DD (i.e. 21 coefficients with maximum variance) and PCA (despite a significantly lower complexity), and is only larger than that of DD, when using all coefficients (m=181).

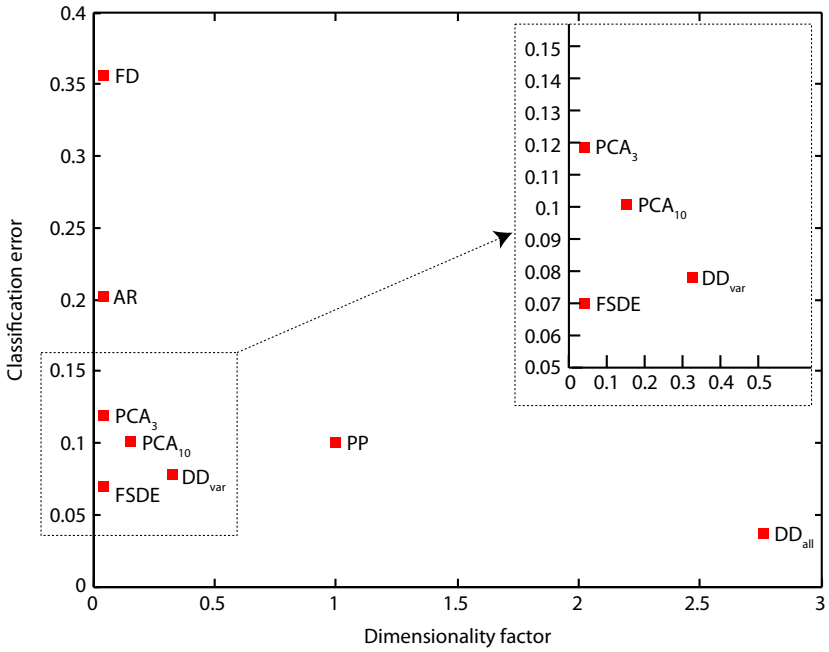


Figure 5.4.: Observing the effect of dimensionality on classification error for the different feature extraction methods used in this work.

5.1.2.3. Clustering FSDE features using k-means

To demonstrate the effectiveness of spike sorting using FSDE features (with 10-iteration k-means), the actual clusters are compared to those determined. This is shown in Figure 5.5, also illustrating the windowed spike waveforms (and corresponding mean templates) of the actual and determined classes.

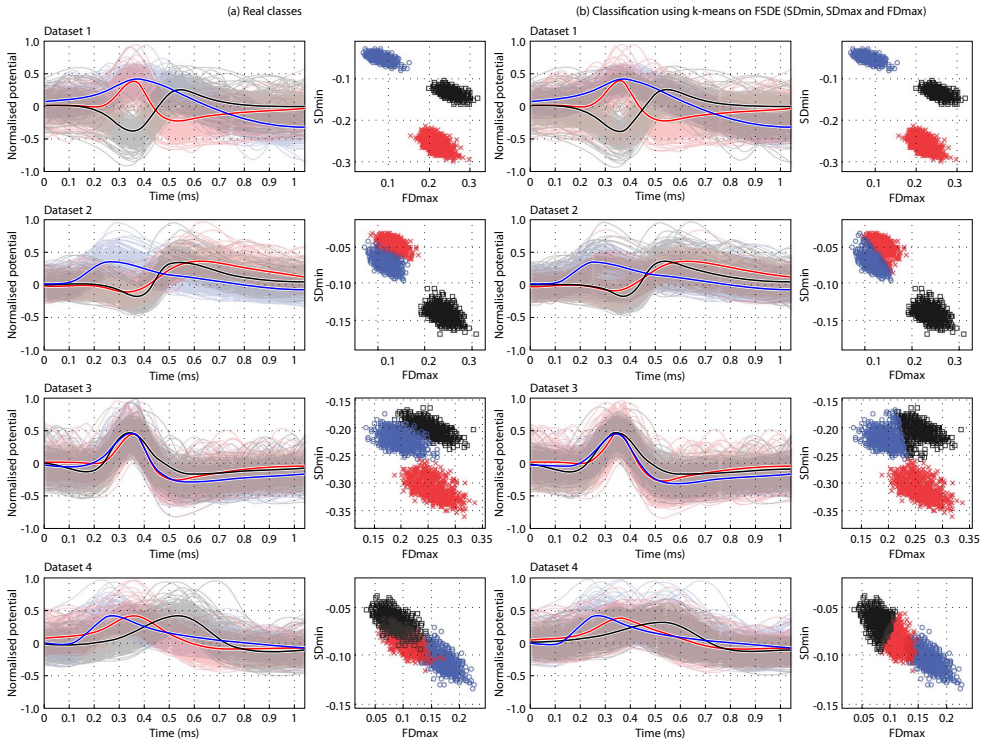


Figure 5.5.: Spike classification utilising FSDE features. Shown are: (a) the actual classes (as specified in each dataset at a noise level of 0.2), and (b) classes as generated using FSDE features with k-means clustering. Shown for both sets are the spike waveforms (with mean templates overlaid) and clusters in FSDE feature space (z-axis is SD_{max} - not shown). In each case, the 3 different clusters are coloured in red, blue and black.

5.1.2.4. Computational complexity

Within the context of implantable spike sorting hardware, computational complexity is as important as the classification accuracy of the spike sorting method. This is due to previously mentioned resource constraints (i.e. power, bandwidth and volume). Here, the computational complexity is defined in terms of the required number of arithmetic operations (i.e. additions, multiplications, etc.) to compute each feature. Table 5.2 presents an estimation of relative complexity of each feature extraction method, in addition to any extra computation required for selecting a subset of features (i.e. in dimensionality reduction).

As mentioned above, k-means is the chosen clustering algorithm used. Since the k-means algorithm is mainly dominated by calculating point to cluster mean distances (using Squared Euclidean metric), its computational complexity is estimated to be $k(2m-1)$ additions and km multiplications per spike, where m is the number of features per spike and k is the number of clusters.

In order to assess the accuracy and computational complexity of different feature extraction methods, the aforementioned factors contributing to computational demands has been combined to produce a single complexity figure-of-merit (CFOM) as in [96, 98]:

$$C = N_{add} + 10 \times N_{mult} \tag{5.7}$$

where N_{add} is the number of additions, and N_{mult} is the number of multiplications required. It should be noted that, only the computations associated with feature extraction and clustering are considered in CFOM. For any computation associated with choosing a subset of features extracted (such as variance calculations as in DD) are only calculated once, and therefore not included. CFOM is given (per spike) assuming 64-sample spikes (n=64) of three different clusters (k=3), and a fourth order autoregressive model (p=4).

The trade-off between classification error and computational complexity is shown in Figure 5.6. It is clearly observed that the FSDE method achieves both a good classification accuracy and low computational complexity. In fact, the only method (from those compared) that outperforms FSDE is D_{all}, which requires 35× more computations. On the other hand, when compared to DD_{var} (considered to be both accurate and computationally efficient), the FSDE method has a 10% lower classification error with 4× lower complexity. In other words, amongst all the feature extraction methods considered, FSDE provides the best trade-off between accuracy and computational complexity.

In addition, the number of features (i.e. dimensions) selected impact the complexity due to added resource required for the increased clustering computation. Hence, methods with fewer features are preferable. Since FSDE uses only three, this further reflects in reduced computational complexity as opposed to methods with higher dimensionality.

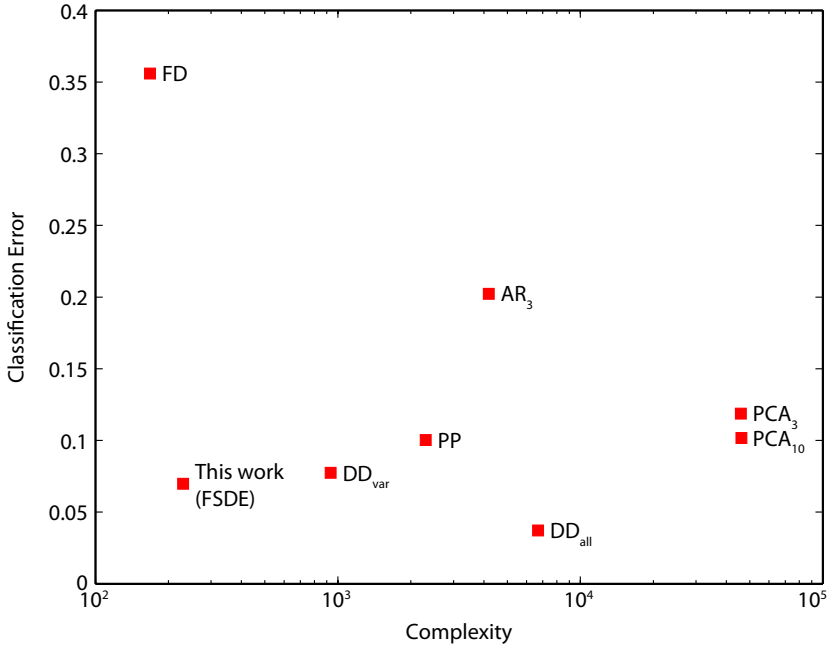


Figure 5.6.: Comparing classification error (averaged over all datasets and noise-levels) with computational complexity for the different feature extraction methods considered herein.

5.2. Improved Feature Space for Derivative Based Sorting Algorithm

The ease of computation and potential to reduce dimensionality while extracting useful features of a spike waveform proves derivative based spike sorting as an attractive solution. From an implantable system perspective, the reduction in computational and storage requirements is critical for the scalability of such systems to monitor large neuronal populations.

Furthermore, it has been shown that derivative features of extracellular action potentials (EAP) provide immunity regarding the background activity [129, 153, 206], hence improve differentiability of the spikes in neural recordings. In fact, in Section 5.1, it has been proven that a simple method based on first and second derivative extrema provides a good balance between the classification error, number of features (i.e. dimensions), and the computational requirements.

In this section, the derivative based feature spaces for spike sorting are further investigated — within the context of implementation complexity and classification accuracy — , but with emphasis on (1) low-SNR conditions, and (2) features other than those described in Section 5.1. More specifically, effects of varying levels of additive white Gaussian noise (AWGN) are of interest given the sensitivity of derivative operation under such circumstances. The white noise sources in a neural recording include electronic noise, axons, dendrites and synaptic currents [89–91].

It has been shown that within 50 μm of the electrode tip (maximum distance single units

can be differentiated), the minimum spike amplitude can be as low as 60 μV [60], while the background noise (activity of neurons beyond 140 μm from the electrode tip) can be as high as 20 μV_{rms} [62]. In addition, white noise contributions observed can also be significant. Even only considering the contribution from electronics, the survey of state-of-the-art recording systems could introduce up to 11 μV_{rms} alone (Section 3.2.1).

First, a set of critical features — that aim to maximise separability of neurons — of EAP waveform are hypothesized based on the EAP waveform dynamics and underlying physiological processes relating to action potential generation. These critical features — spanning EAP waveform and its derivatives up to third order — are then empirically assessed using a database of real neuron templates and a subset of these are identified as potential features to be used for spike sorting. These are then systematically tested under varying background and AWGN noise levels. Based on these results, an improved feature space — that provides best immunity against both background and AWGN — is proposed for spike sorting applications.

5.2.1. Methodology

All investigations are performed using Mathworks Matlab^{TM} R2011b v7.13. The spike sorting accuracies of different feature spaces are evaluated using L1-distance to calculate minimum error (i.e. similar to template matching but with reduced number of samples) for four different simulated neural recordings (each with three single-units) at varying background activity levels (standard deviations of the added noise are 0.05, 0.01, 0.15 and 0.2) [77,209]. Different levels of background activity are simulated by superimposing 594 waveforms recorded from the human neocortex and basal ganglia. For details on simulated datasets, please refer to Appendix B.1.

As mentioned earlier, AWGN arising from axons, dendrites, synaptic currents and electronics can be significant [89–91]. In fact, AWGN contributions from electronics alone can be as high as 11 μV_{rms} (see Section 3.2.1). Therefore, in addition to background activity, varying levels of additive white Gaussian noise (AWGN) are also added to the datasets above. These four datasets are referred as "Easy1", "Easy2", "Difficult1", and "Difficult2" in the following discussions.

5.2.2. Extracellular Action Potential Dynamics and Waveform Features

Extracellular action potentials comprise three distinct phases: (1) Initial brief positive peak, (2) much larger negative peak, and (3) a longer duration of positive period with slowly decaying amplitude. The brief initial peak is due to the positive capacitive current; the main negative peak is attributed to the influx of Na^+ current driving the action potential; and the final positive phase results from re-polarizing K^+ current flowing out of the neuron (see Figure 5.7) [59].

The shape of the observed EAP waveform is proportional to the time profile of total current exchange across the cell membrane and the position of the electrode relative the the source [59]. As previously described (in Section 5.1), the EAP can be modelled as a dipole charge $\overrightarrow{p}$ at location $\overrightarrow{r}$ from the center of dipole, and the amplitude of the EAP scales with the distance

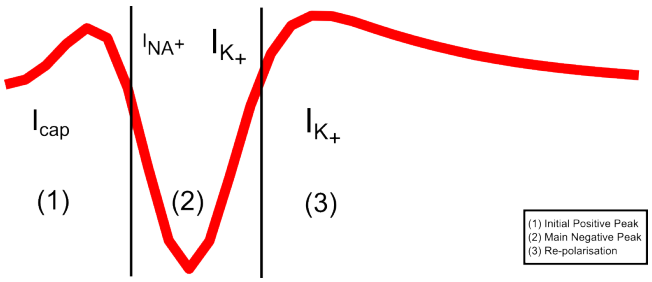


Figure 5.7.: General structure of generated action potential. (1) Brief positive peak due to capacitive current (2) Main negative peak due to the influx of Na^+ ions (3) Re-polarisation phase as a result of K^+ outflow

from the source [61]. In addition to this scaling effect of the EAP amplitude, [47] also shows that the position of the electrode, with respect to different sections of the neuronal structure, also affects the expression of the critical features described above due to variations observed in different current components across the neuronal structure.

For example, the EAP may exhibit different degrees of capacitive phase peak depending on the electrode position. These can range anywhere from none (closer to soma) to almost as same amplitude as the Na^+ dominated negative peak (closer to locations along the apical trunk). Near soma, the membrane potential change is driven by local Na^+ current, and there is no significant capacitive current until after the Na^+ current is already active [59]. Since the amplitude of the Na^+ current is larger than the capacitive current, the total current lacks an initial positive peak. On the other hand, in the more distant dendrites the initial capacitive current is relatively larger due to the fact that there is a brief interval between the action potential regeneration (through local Na^+ channel openings) and the initial (passive) depolarisation. Moreover, the Na^+ conductance density is lower in the dendrite than at the soma, hence the capacitive current is relatively larger [59].

Differences in conduction densities is also the major source of variability observed in the main negative peak and re-polarisation phases [59, 63]. The main contributor to the main negative peak of an AP is the inflow of Na^+ which causes depolarisation of the membrane potential, while the following re-polarisation phase is dominated by outflow of K^+. Analysis of gene expression profiles provide evidence for varying levels of expression of ionic channels [63]. Such differences in conductance densities result in variations of the:

- width of Na^+ -dominant phase and the inflection points in the transition to the K^+ -dominant phase
- amplitude and shape of the K^+ current.

In addition to the effects of conduction density variabilities, the position of the electrode also affects the duration of the Na^+ -dominant phase of the waveform [47, 59, 61]. The further away the electrode from the cell, the longer the duration of the Na^+-dominant phase. This is due to the fact that action potentials take time to back-propagate and voltage amplitudes decay with

distance from the source.

When placed close to soma, a recording electrode will observe only the somatic and most proximal dendritic current sources since the distribution from distant sources are scaled. If the electrode is moved away from the soma, the observed EAP at the electrode will be due to both from the back-propagation of action potential into dendrites and the somatic initiation since they will have similar scaling due to distance. Back-propagation along the main dendrites takes a large fraction of a millisecond, hence superposition of multiple moving current sources appear as a long-duration single static current source [59]. In addition to all these factors, variabilities in the size of the soma, biophysical properties, action potential initiation and channel behaviours (i.e. kinetics) further contribute to the differences observed in the EAP waveform.

Following the discussion above, variations in the EAP waveform can be summarised in three categories: (1) Variations in amplitude and width (thus the rate of change) of the positive capacitive peak preceding the main negative peak; (2) variations in the amplitude and width (thus the rate of change) of the main negative peak; (3) Variations in the amplitude, rate of decay of re-polarisation phase and existence of hyper-polarisation. Hence the following hypotheses are proposed:

- Hypothesis 1: The main features affected by the EAP variations are predominantly the local minima, maxima and the inflection points of the EAP waveform, hence these features are responsible for the most of the differentiation among the recorded spikes. In other words, the characterisation of the recorded signals can be done using such positions of inflection points and local minima/maxima; amplitudes of the signals at these positions as well as the magnitudes of rates of change.

- Hypothesis 2: Following "Hypothesis 1", since the aforementioned features provide the most differentiation between the spikes and the rest contribute little (if any) information, then either all of these features or a subset of these should provide a comparable spike sorting accuracy to using all of the features available.

5.2.3. Results and Discussion

5.2.3.1. Empirical assessment of hypothesized potential features for spike sorting

Derivatives provide a relatively computationally efficient method in order to extract the features mentioned in the above hypotheses. For example, the local minima/maxima and inflection points can easily be calculated from the zero-crossings of the first derivative of the waveform, while the rates of change at these local minima/maxima can be extracted using the zero-crossings of the second derivative of the signal. Similar relationship holds on for further higher order derivatives.

In order to establish which of these local extrema (proposed by hypotheses) could potentially provide the best separation, EAPs (from a database of 594 averaged EAP waveforms) are investigated. These EAP waveforms are from real recordings of monkey neo-cortex and basal

Table 5.3.: Variation in amplitude and temporal domain of the critical features of the EAP waveform. The values represented are the standard deviations of feature amplitudes and time positions. Waveforms are aligned w.r.t. main negative peak and spikes have been normalised across different derivatives.

	Feature No	**1**	**2**	**3**	**4**	**5**	**6**
Amplitude	0	0.0392	0.1009	0.0587			
	1	0.0218	0.0743	0.0919	0.0202		
	2	0.0169	0.0592	0.0950	0.0514	0.0082	
	3	0.0203	0.0485	0.0880	0.0672	0.0237	0.0086
Temporal	0	15	0	56			
	1	9	3	5	23		
	2	7	4	2	5	13	
	3	7	5	4	4	5	8

ganglia, and are sampled at 96kHz [157]. The primary criteria for selection of a subset of these features is based on their spread in amplitude and temporal domain. The higher the spread is, the more differentiable are the spikes.

Distributions of these features (given by local min/max) are presented in Figure 5.8. More specifically, Figure 5.8 (a) shows the distribution of three distinct features of action potentials: positive capacitive current peak, Na^+ -dominant phase main negative peak, and repolarisation peak (K^+ dominated). Figure 5.8 (b) presents the maximum rates at which these phases of action potential change while Figure 5.8 (c) provides information regarding the curvature (rate of change of rate of change) and Figure 5.8 (d) is the rate of change of curvature.

The variations in amplitude and time position are given in Table 5.3. The majority of the spread of the original waveform is observed at the Na^+ -dominant main negative peak, while it is followed by variations in re-polarisation and initial capacitive phases. Moreover, these features show a large variation regarding positions increasing potential differentiability. Note that since spikes are aligned with respect to their main peak, there is no temporal spread at peak position. Among the first derivative features, features 2 and 3 (Na^+ gradients) show the largest variation in amplitude (not so much for temporal), while features 2, 3, and 4 are the ones with highest variability among second and third derivatives. Hence, the primary focus is on these subset of features as potential candidates to use in the proposed algorithm.

5.2.3.2. Feature selection and verification

As previously discussed, it is one of the primary objectives of Section 5.2 to quantify the sensitivity of derivatives in the presence of increasing AWGN. Having identified — through the proposed hypotheses and empirical assessment — four different feature spaces (spanning EAP waveforms and its derivatives up to third order) and specific features representing local extrema within each feature space, here the spike sorting accuracy is quantified for these features under varying background and AWGN noise levels. It has been shown in previous sections that derivatives have a clear advantage over rejecting background activity, while EAP waveforms, in contrast, is expected to provide a much better choice for increased levels of AWGN.

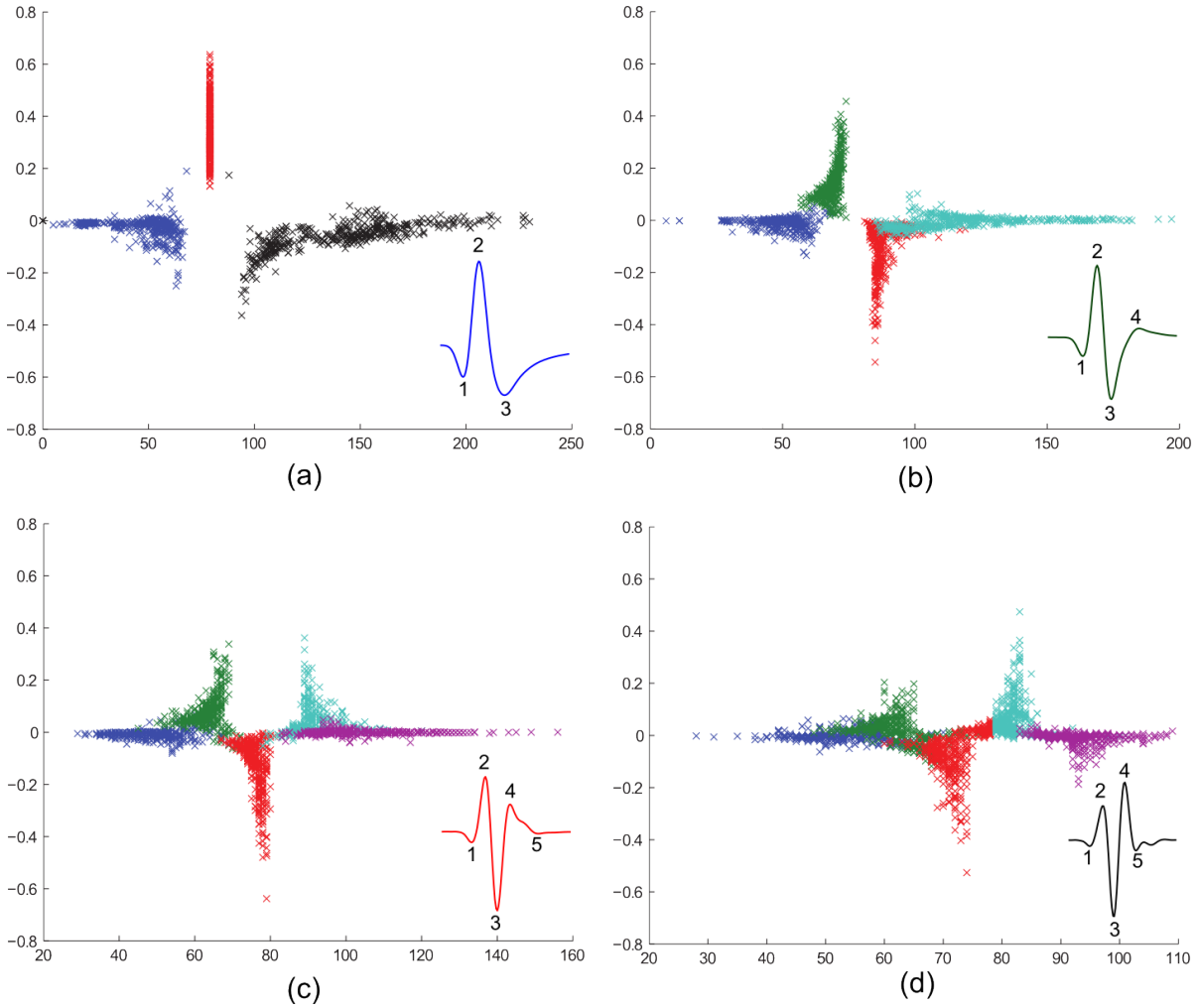


Figure 5.8.: Normalized amplitude distributions (y-axis) of local min/max and inflection points of spike waveform and its derivatives (up to third order). X-axis corresponds to sample number (signals sampled at 96 kHz). (a) EAP waveforms (b) First Derivative (c) Second Derivative (d) Third Derivative. Each subfigure contains a representative waveform with numbered local min/max and extrema. These correspond to feature numbers given in Table 5.3.

As mentioned in Section 5.2.1, L1-distance in conjunction with a similar approach to template matching is used to assess the sorting accuracy for different feature spaces. The distance between the extracted features from each EAP and "feature templates" (consisting of cluster centres, i.e. mean, of selected features) is computed, and the EAP is then assigned to the neuron class with minimum distance. The extracted features for each feature space are selected among positions of local extrema (determined in the empirical study) which provide the largest variation.

For example, for the analysis of second derivative feature space, the features to be used are selected among feature positions denoted by {2, 3, and 4} on each neuron. Considering 3 neurons are available in the recordings[2], this corresponds to a maximum of 9 possible feature locations[3] of largest separation between neuron classes. Three features are selected for feature extraction (out of 9 possible locations) to ensure at least one feature is dedicated for separation among different pairs of available neurons[4].

The spike sorting accuracy of all feature spaces, under varying background activity and AWGN, are presented in Figures 5.9 and 5.10. While the accuracy of the feature spaces do not show a significant difference in Easy1 dataset due to relatively distinct features of the involved action potentials (hence easier to spike sort), there is a clear differentiation among them for the remaining datasets where neuron shapes are much similar[5]. In fact, for Easy1 dataset, the maximum difference in accuracy occurs at the worst case SNR, and for the remaining SNR values the spike sorting accuracy of all feature spaces are within a few percent of each other.

The results once more confirm the immunity of the derivative features to the increasing background activity. It can generally be concluded that the higher the derivative, the better the spike sorting accuracy. In fact, the sorting accuracies of the second and third derivatives are significantly better than the first derivatives for increasing background activity. Averaged across all datasets, their performance differences over first derivatives are 4.6% and 3.6% respectively. Considering performance over "Difficult1" and "Difficult 2" datasets, the accuracy difference is further increased to 7.57% and 6.20%, respectively. From an increasing AWGN perspective, on the other hand, the results reveal the sensitivity of higher order derivatives to increasing AWGN. For example, while the second and third derivatives perform 6.5% and 9.21% better than first derivatives for higher SNR values[6](11.57% and 11.46% for difficult datasets), these accuracy differences are only 2.18% and -0.43% (2.63% and -0.3% for difficult datasets) for lower SNR values[7]. In light of these results one can clearly say that second derivatives are the

[2]The average number of neurons observed in real extracellular recordings is 2.5 neurons per channel.

[3]Some of these positions might coincide resulting in less than 9 possible locations.

[4]Three extracted features aim to capture maximum L1-separation between pairs of: Neuron 1 & Neuron 2, Neuron 1 & Neuron 3, and Neuron 2 & Neuron 3. The selection is based on identifying these 9 features on mean EAP waveforms (or derivative waveforms), and choosing the ones that yield maximum L1-separation. On the other hand, created "feature templates" consist of two features per template. This is due to the fact that, for example, feature separating Neuron 2 & Neuron 3 does not contribute any useful information regarding Neuron1 hence excluded from "feature template 1".

[5]Similarity measures are previously shown in Figure B.1.

[6]Within the context of AWGN, the term "high SNR values" refers to those less than 20 dB decrease in SNR in AWGN (x-axis in plots).

[7]Within the context of AWGN, the term "low SNR values" refers to those more than 20 dB decrease in SNR in

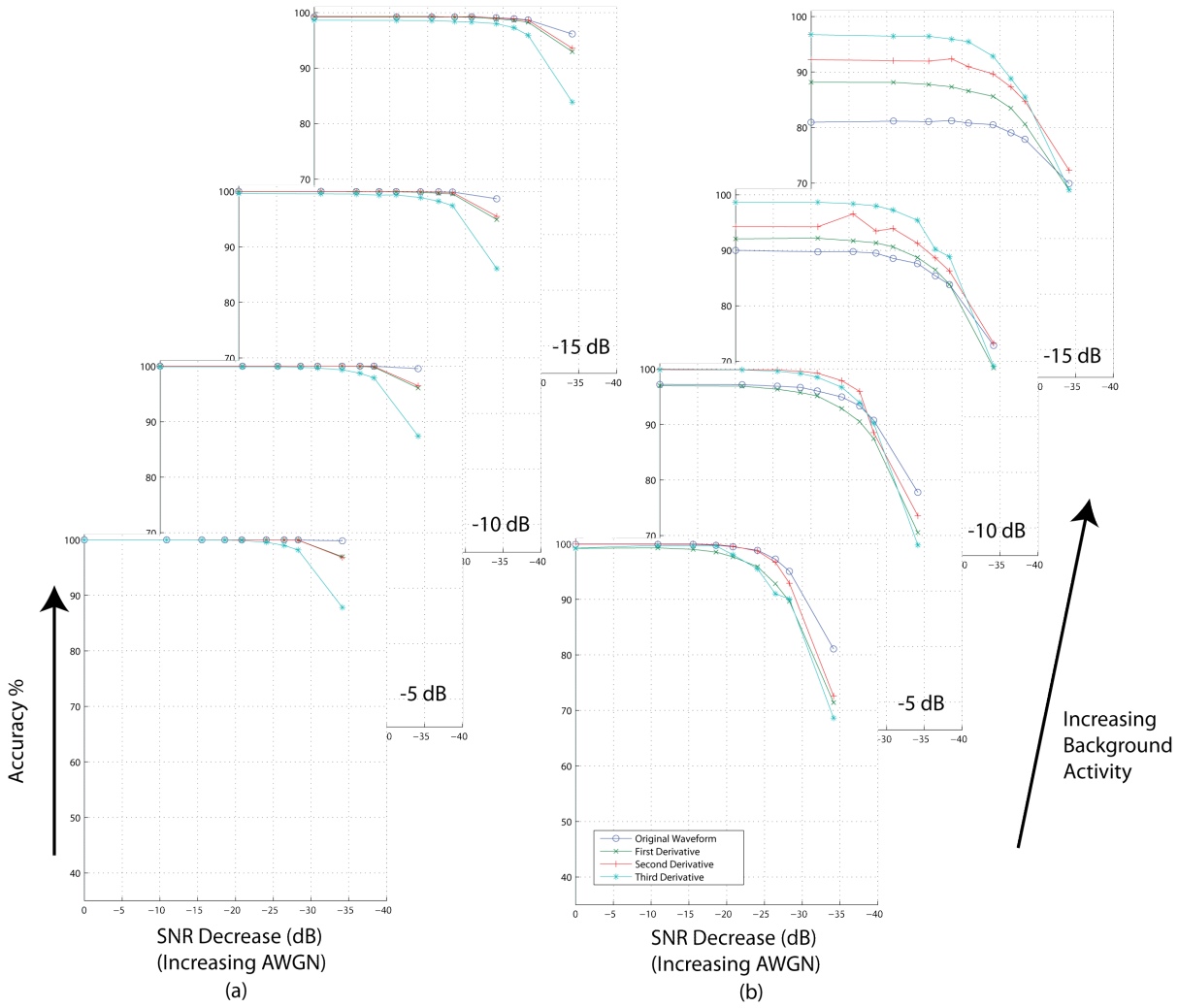


Figure 5.9.: Spike sorting performance at varying background and white noise levels of different feature subsets (original waveforms and derivatives up to third order). (a) Easy1 dataset (b) Easy2 dataset. The starting SNR from AWGN perspective is 47 dB, while from background activity perspective it is 0.05 standard deviation (spike amplitudes normalised to 1) corresponding to 26 dB SNR value.

best performers among derivative space (closely followed by third derivatives) striking a good balance between immunity to AWGN and rejecting background activity.

On the other hand, as expected, the features extracted from the EAP waveform are more immune to increasing AWGN than derivatives. While the accuracy loss of all derivatives for high SNR values are within -3.3% (compared to no AWGN), reductions in sorting accuracy up to 47% are observed for lower SNR values (increasing AWGN). In fact, the average rate of accuracy loss[8] of derivative features are 0.44%, 0.60% and 0.72% per dB (increasing order of derivatives). The original waveform features, in contrast, perform much better at increased AWGN values having half the accuracy loss rate (0.25% accuracy per dB) of the nearest derivative feature.

AWGN (x-axis in plots).

[8]first order approximation

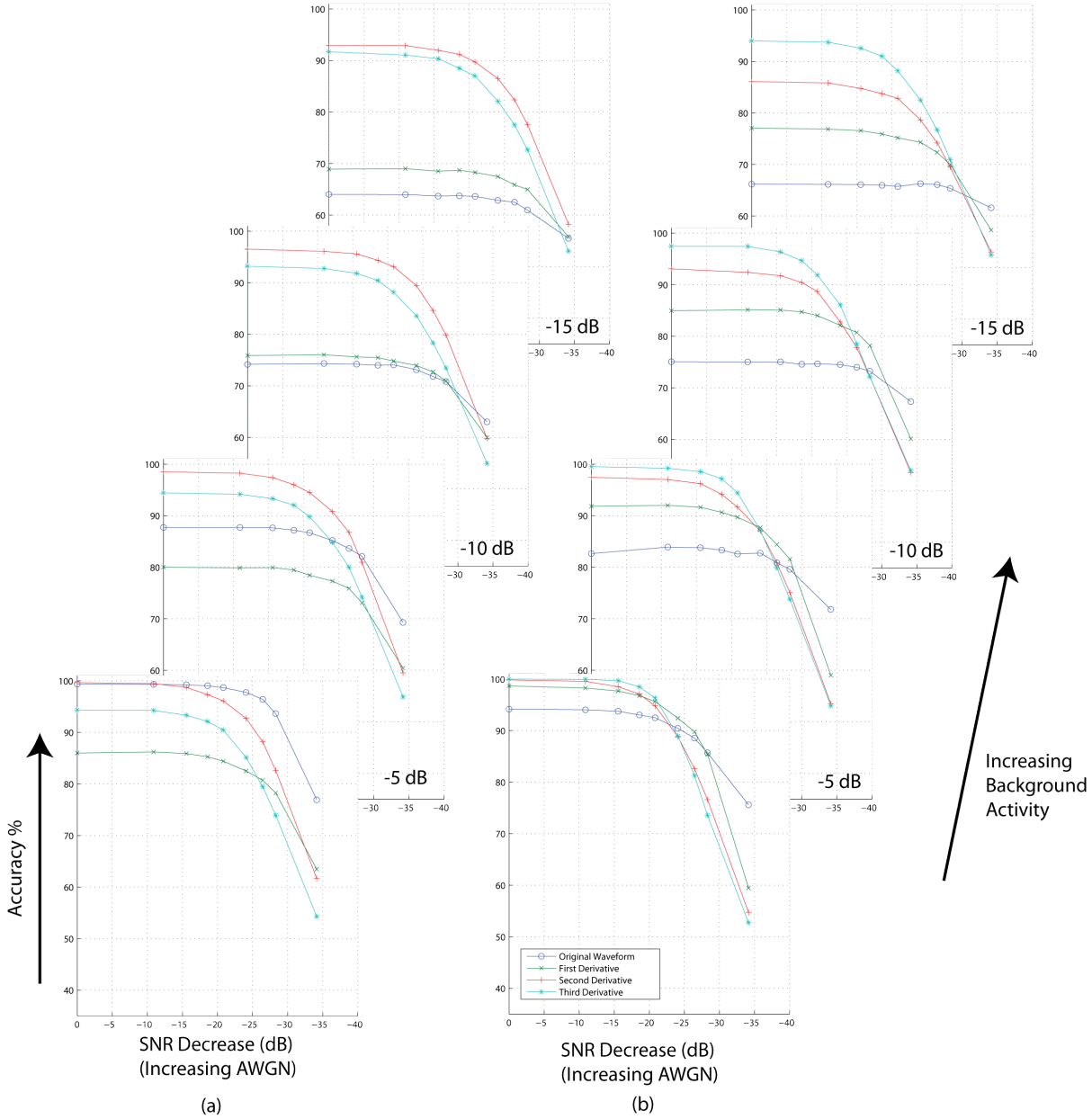


Figure 5.10.: Spike sorting performance at varying background and thermal noise levels of different feature subsets (original waveforms and derivatives up to third order). (a) Difficult1 dataset (b) Difficult2 dataset. The starting SNR from AWGN perspective is 47 dB, while from background activity perspective it is 0.05 standard deviation (spike amplitudes normalised to 1) corresponding to 26 dB SNR value.

When EAP waveform features are compared with second and third derivatives, there is a clear sorting accuracy difference owing to higher order derivatives' ability to reject background activity. In fact, the overall accuracy of the waveform features are 4.62% and 3.63% worse (compared to second and third order derivatives respectively), while first derivative features are only 0.18% better in overall performance. For low SNR values, on the other hand, waveform features perform 1.79% and 2.23% better compared to first and third derivatives whereas the

accuracy is -0.39% lower compared to second derivatives.

For the lowest SNR cases (beyond 26 dB reduction), however, the sorting accuracy of EAP waveform features are 3.7%, 3.2% and 6.8% better compared to derivative feature space (increasing order of derivatives). This accuracy difference becomes more emphasized for difficult datasets resulting in 4.7%, 5.26% and 8.7% better performance achieved by EAP waveform features.

In the light of these results and discussion, the two main conclusions are: (1) Second derivative feature space offers the best trade-off in rejecting background activity and AWGN immunity and (2) EAP waveform features are the most accurate under low SNR due to increasing AWGN. Considering that neural recordings are also dominated by high levels of AWGN besides background activity, it is suggested that EAP waveform features and second derivative features are both used together to improve the second derivative performance at the lowest SNR spectrum. However, this inevitably comes at a cost of extra computational complexity — albeit a minimal one considering no extra processing is required for EAP waveform features — and a potential slight reduction in sorting accuracy for the cases of high background activity with almost no AWGN (which is highly unlikely).

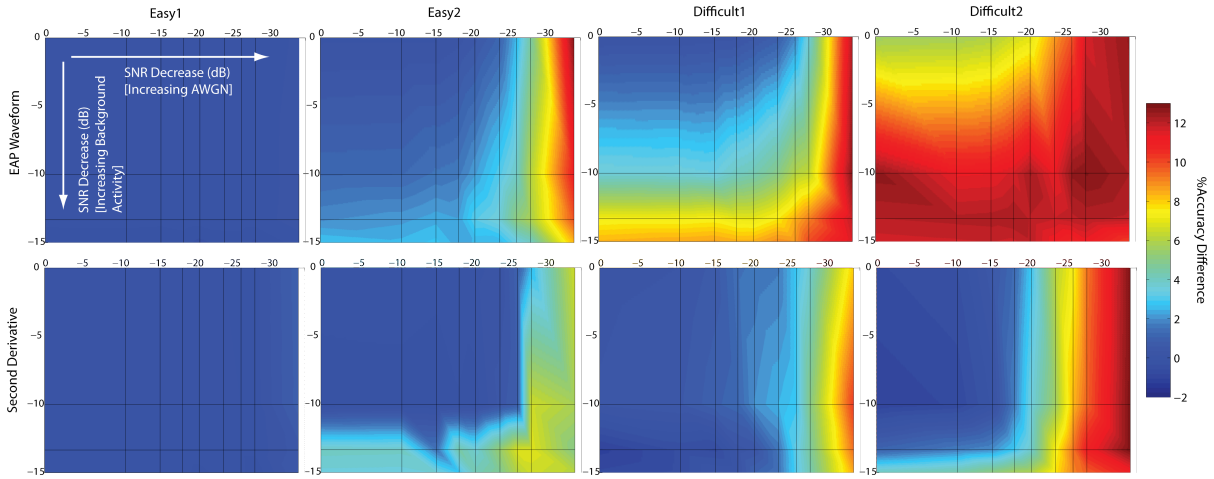


Figure 5.11.: Performance comparison of the selected methods for proposed reduce feature space versus utilizing the whole.

The main motivation behind this work centred around the hypotheses based on waveform dynamics which proposed that a few critical features capture the most information regarding differentiability of neurons. Therefore, the proposed feature spaces are compared to using complete set of samples to achieve spike sorting. The results of this comparison under varying background activity and AWGN is presented in Figure 5.11. The plots reveal that the largest degradation in performance (12%) occurs only in the most difficult cases to separate at the very lowest end of the SNR spectrum. In fact, the average spike sorting accuracy loss is only 3.9%.

As previously mentioned, the extracted features for each tested feature space are selected among positions of local extrema (determined in the empirical study) which provide the largest variation. This selection of features to be extracted involved choosing the ones that ensured

at least one feature is dedicated for separation among different pairs of available neurons. As a result, three features per neuron have been extracted for each feature space. This is a 21× reduction in both computational (impact on feature extraction and classification) and memory requirements considering neuron waveforms consist of 64-samples[9]. Considering the small loss in accuracy for the complexity trade-off, the proposed number of features to extract prove to be both accurate and efficient. This result once again confirms the underlying hypothesis that most of the information is contained in few defining features given by local min/max as discussed previously.

5.3. Hardware-efficient spike sorting method based on waveform and derivative features

In this section, a novel hardware-efficient spike sorting method — developed based on EAP waveform and second derivative feature spaces proposed in Section 5.2 — is presented. The proposed method consists of off-line training and real-time classification stages (similar to two-stage hybrid approach proposed in Chapter 4). The off-line training performs channel-specific selection of near-optimal subset of features (for each feature space) to be extracted, and the real-time classification is achieved through simple comparisons between extracted features and pre-determined (during offline training) comparison thresholds. While the use of feature spaces proposed in Section 5.2 ensures a good sorting accuracy across varying background activity and AWGN noise levels, performing real-time classification through simple comparisons ensure minimal resources for hardware implementation. Compared to other low-complexity methods, the proposed spike sorting method achieves the best trade-off in sorting accuracy and computational complexity. From here on, the proposed method is referred as waveform and derivative features (WDF).

5.3.1. Methodology

The verification of WDF is performed using Mathworks Matlab™ R2011b v7.13. Its sorting accuracy and computational requirements are compared to those of other low-complexity methods under several datasets with varying levels of background activity and AWGN.

5.3.1.1. Test Datasets

The same datasets in Section 5.2 are used to test WDF. These are four simulated neural recordings (each with three single-units) at varying background activity and AWGN levels. Please refer to Section 5.2 and Appendix B.1 for more information on datasets. These four datasets are referred as “Easy1”, “Easy2”, “Difficult1”, and “Difficult2” in the following discussions.

[9] 2.5 ms window sampled at 24 kHz.

5.3.1.2. Method Verification

To establish the viability of WDF within the context of spike sorting accuracy and hardware efficiency, it is compared to other low-complexity methods. These are First and Second Derivative Extrema (FSDE, see Section 5.1) and Discrete Derivatives (DD; 21 coefficients) [96]. In addition, WDF is also compared to Template Matching (TM) and Principal Component Analysis (PCA) which is widely regarded as the "gold standard". Only the first six principle components have been used in comparisons. For the feature extraction methods — FSDE, DD and PCA — compared herein, K-means is used as the clustering/classification method.

The accuracy of the compared methods are quantified by Eq. 5.8. Their computational requirements are assessed by a single complexity figure-of-merit (CFOM) as in [96, 98], and given by Eq. 5.7 (see Section 5.1.2). Computational requirements are analysed in terms of feature extraction and overall computational complexity, as well as in terms of dimensionality reduction. The computational complexity of the training algorithms have not been considered in CFOM for any of the methods since these are one-off events.

$$Accuracy = \frac{No.\ of\ correctly\ sorted\ spikes}{Total\ no.\ of\ spikes} \times \% \tag{5.8}$$

It should be noted that while WaveClus [77] is used for clustering purposes (during off-node training) in real-time hardware implementation and demonstration of WDF (please refer to Section 5.4.3 for system implementation), throughout the validation procedure a-priori class information is used during training stage instead of clustering. This is to isolate the performance of WaveClus in order to only quantify WDF. It should also be noted that the training algorithm and the feature extraction for the proposed method uses half the sample space (1.6ms spike window). This decision is taken in the light of template window size discussions in Chapter 4.

5.3.2. Proposed Spike Sorting Method

The main motivation behind the proposed sorting method is the fact that local extrema and inflection points on EAP waveforms and its derivatives describe majority of the difference among recorded EAPs. This has been hypothesized based on waveform dynamics and demonstrated in previous sections. In Section 5.2, it has been established that original waveform features have the most immunity with respect to increased levels of AWGN and perform best at the worst case SNR levels, while the second derivative features have the greatest immunity to increased background activity. Therefore, these two feature spaces are chosen for WDF.

A high level description of the proposed spike sorting method is shown in Figure 5.12. It consists of two stages: off-node training and on-node real-time classification. During off-node training, neural data is streamed out to an external device where available neurons (N) and their associated waveforms (i.e. templates) are established through initial clustering of the data. The training algorithm then locates the positions of features, i.e. local extrema established in Section 5.2.3.1, for both the waveform and second derivative templates. Recalling that these positions of local extrema — S — were shown (in Section 5.2.3.1) to be the regions of possible

maximum separation between neurons, the training algorithm then selects a subset of features — at sample positions P — within S that maximises separation of all N. For example, $P_{1,2}$ is the sample index of the feature that maximally separates neuron #1 and neuron #2 (see Figure 5.12). Such approach ensures channel-specific selection of near-optimal features. In addition, a corresponding comparison threshold, $T_{i,j}$ is computed for each feature at position $P_{i,j}$.

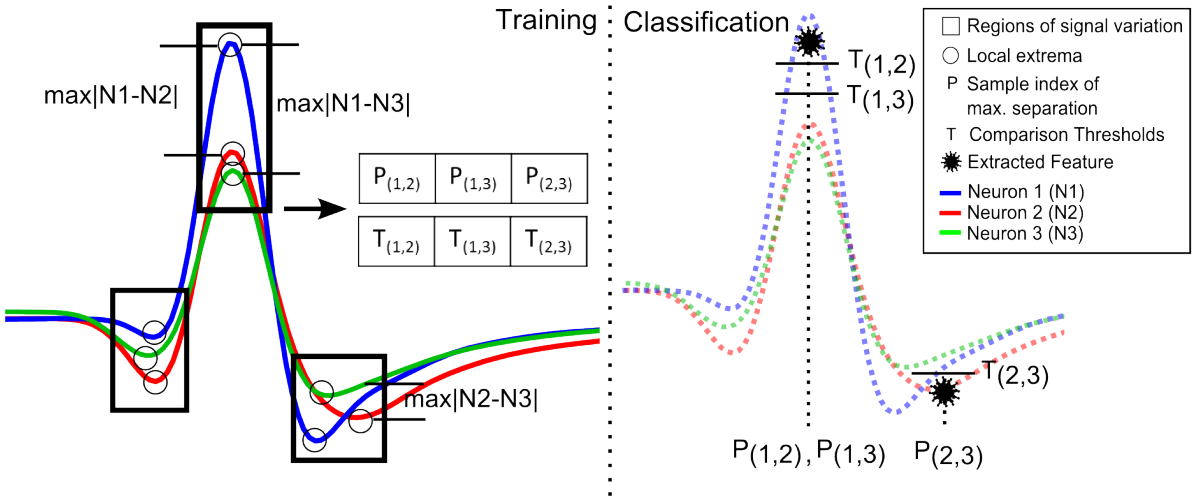


Figure 5.12.: A high level description of WDF.

Following the training, feature positions (P) and comparison thresholds (T) are transmitted back on-node and stored for real-time classification. During the classification stage, features are extracted using feature positions P, and compared to corresponding T to cast a vote for either of the relevant neurons. This classification process is illustrated in Figure 5.12. Consider $P_{1,2}$, the feature position at which the maximum separation between neurons #1 and #2 occur. Since the extracted feature at position $P_{1,2}$ is above $T_{1,2}$, it is likely to have been extracted from neuron #1. Therefore, a vote is cast for neuron #1. The same is repeated for all features extracted at positions P, and the detected spike is assigned to the neuron class with maximum votes. It should be noted that, although not shown in Figure 5.12, the same procedure is performed on second derivative feature space as well.

5.3.2.1. Off-node Training

As mentioned previously, the off-node training stage involves initial clustering of the detected spikes during which the templates of the EAP waveform and second derivative space are determined. Using these templates, the positions of local extrema (established in Section 5.2.3.1) are identified, and among these the training algorithm chooses features at sample positions $P_{i,j}$ for each available neuron pair — where $i, j \in N$, $i \neq j$ — such that features at $P_{i,j}$ has the largest L1-distance (hence maximises separation of N_i and N_j). Considering n(N) = 3, three feature positions — $P_{1,2}, P_{1,3}, P_{2,3}$ — are selected from both the EAP waveform and second derivative space (total of 6 features).

In addition, off-node training also determines a corresponding comparison threshold $T_{i,j}$ at

each $P_{i,j}$. $T_{i,j}$ represents the mid-point of the features of N_j and N_i at $P_{i,j}$, scaled according to the spread (i.e. standard deviation) of the features. Without such scaling, $T_{i,j}$ would be placed exactly at the midpoint, and may cause errors in classification if there exists a significant difference in distribution of the features of N_j and N_i (at $P_{i,j}$). The proposed concept of comparison threshold computation is illustrated in Figure 5.13.

5.3.2.2. Feature Extraction and classification method

As mentioned previously, the feature extraction relies on the feature positions, $P_{i,j}$, selected during off-node training. As the spikes are detected, features are extracted at $P_{i,j}$ and compared to $T_{i,j}$. This procedure is illustrated in Figure 5.13. Given X is the extracted feature at $P_{i,j}$, it is compared to $T_{i,j}$. As X is below $T_{i,j}$, it is highly likely that detected spike belongs to neuron $\#j$. Therefore, neuron $\#j$ (N_j) receives a vote. The same procedure is applied for all $P_{i,j}$, and the detected spike is assigned to the neuron class with vote majority. Such simple feature extraction and classification method — based on comparisons and voting — can be efficiently implemented in hardware for real-time spike sorting.

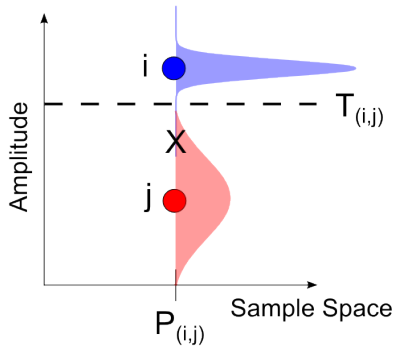


Figure 5.13.: Illustration of the threshold choice ($T_{i,j}$) based on distributions of features i and j, and the mechanics of comparison based classification.

In order to improve the real-time classification accuracy, a weighted voting scheme — taking into account the standard deviation of the noise — is used. If the extracted feature, X, is within one standard deviation of $T_{i,j}$, the weighting of the vote is 1. If X is beyond one standard deviation of $T_{i,j}$, on the other hand, the weight of the vote is 2. Despite such approach slightly increasing the computational complexity by one comparison per $P_{i,j}$, it improves the classification accuracy by 4.73% on average (8.19% for difficult datasets). At some instances, the improvement in classification accuracy up to 11% have been observed. The standard deviations of noise are calculated (during off-node training) separately for EAP waveform and second derivative space.

5.3.3. Results and Discussion

In this section, WDF is compared with other established low-complexity methods in terms of spike sorting accuracy and computational requirements.

5.3.3.1. Comparison of classification accuracy with other methods

The comparison of classification the accuracy of WDF with other commonly used feature extraction and sorting methods — across varying background and AWGN noise levels — is presented in Figures 5.14 and 5.15. It is observed that the proposed method is on par in terms of classification accuracy when compared to methods such as PCA and TM, and in fact it performs much better at increasing background activity due to its second derivative component. The observed average accuracy loss of the proposed method is 8.36% as the background activity is increased[10], while the average accuracy loss of TM and PC are 11.2% and 15.4% respectively. As a result, under higher background activity, WDF performs 0.4% and 4.6% better than TM and PCA respectively. It should also be noted that WDF has an error rate of only 2.46% more than TM and PCA for lower background activity simulations. The average accuracy loss of DD and FSDE, on the other hand, were similar to WDF (roughly 7% for both) as expected. DD and FSDE are also derivative based methods hence are immune to the background activity. Nevertheless, WDF still outperforms both methods by 5.91% and 7.85% for lower background activity, and by 4.2% and 6.8% for higher background activity.

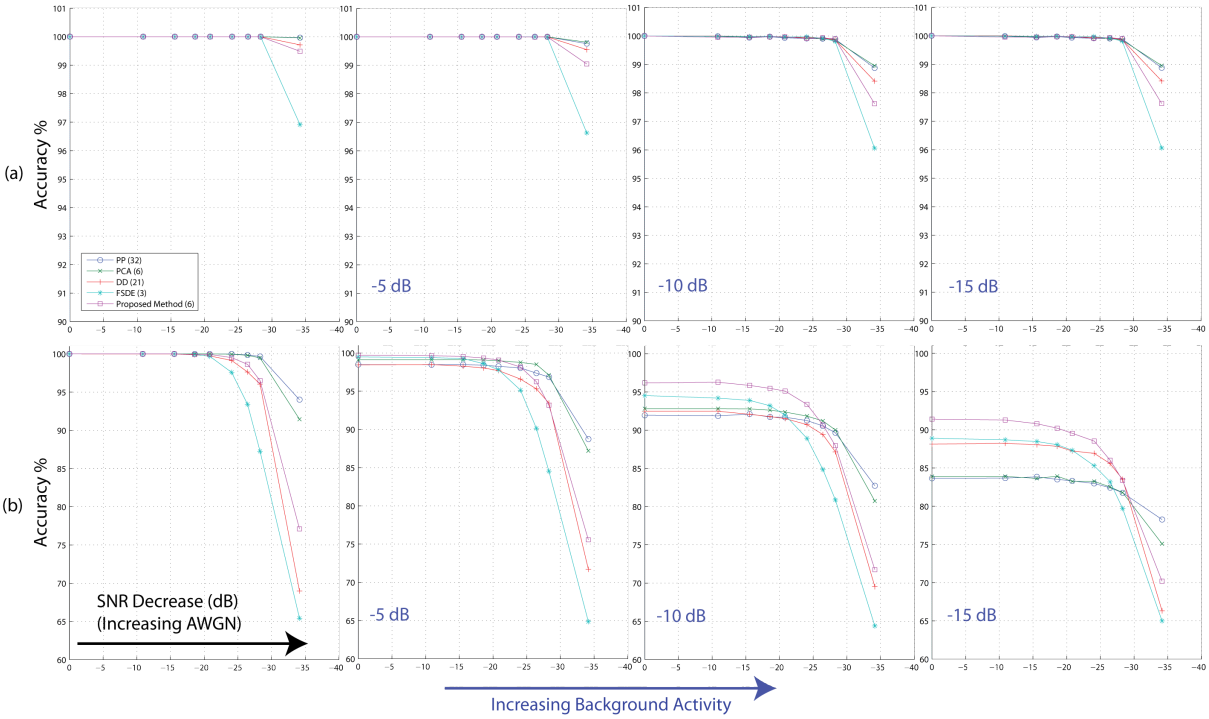


Figure 5.14.: Comparison of the spike sorting performance for different methods across varying background and AWGN noise levels. X-axis for each plot reflects increasing levels of AWGN, while plots from left to right represent increasing levels of background activity. Shown are the results for (a) Easy1 (b) Easy2 datasets.

From an increasing AWGN perspective, on the other hand, WDF performs comparable to both PCA and TM, while outperforming other derivative based methods (DD and FSDE).

[10]The sorting accuracy loss is averaged across all AWGN levels and datasets, and reflect the difference between background activity levels denoted by 0 dB & -5 dB and -15 dB & -20 dB

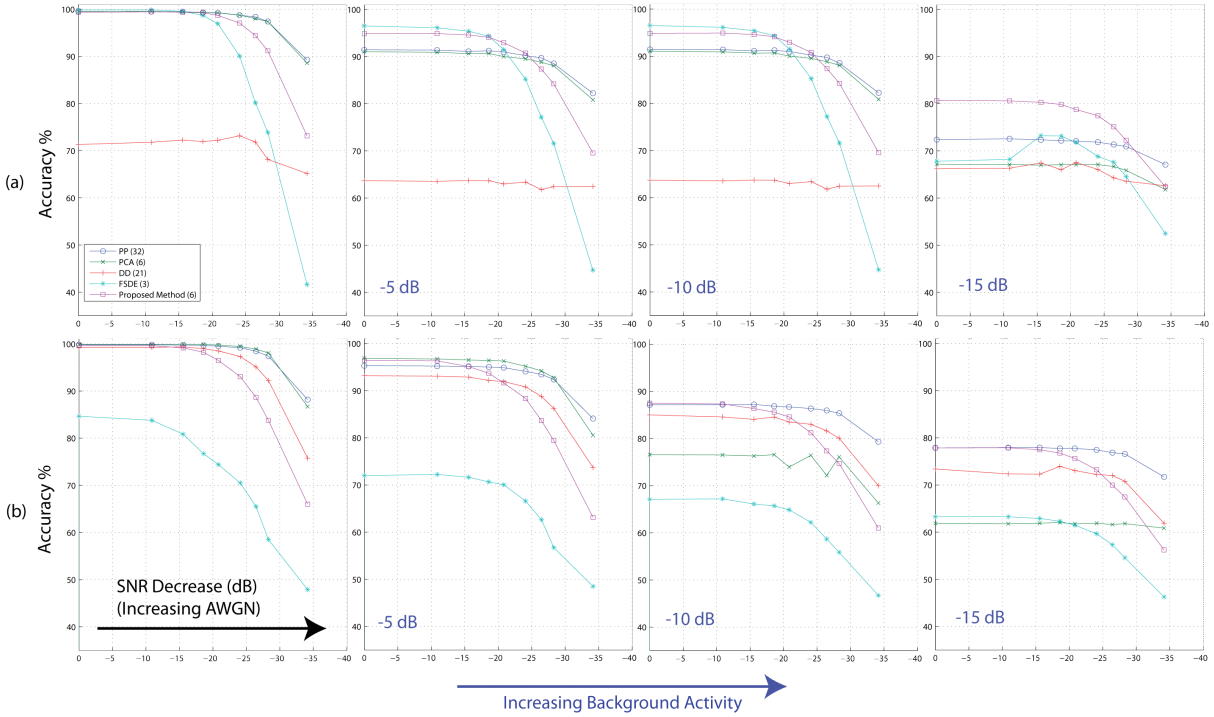


Figure 5.15.: Comparison of the spike sorting performance for different methods across varying background and AWGN noise levels. X-axis for each plot reflects increasing levels of AWGN, while plots from left to right represent increasing levels of background activity. Shown are the results for (a) Difficult1 (b) Difficult2 datasets.

TM and PCA are the most robust to increasing AWGN levels showing the least degradation in performance. As the AWGN is increased, the average sorting accuracy loss[11] is 2% for both TM and PCA. The sorting accuracy of TM and PCA are 92% and 90% for lower AWGN[12], while these are 90% and 88% for higher AWGN. On the contrary, the derivative based methods (as expected) reveal the highest loss in performance between lower and higher AWGN levels. DD is the most resilient with an average accuracy loss of 4%, followed by WDF (7% loss) and FSDE (10% loss). However, WDF is still the best performing method with 94% accuracy compared to 87% (DD) and 88% (FSDE) for lower AWGN noise, and with 87% accuracy compared to 83% (DD) and 78% (FSDE).

There exists two main reasons regarding the comparable performance — under AWGN — of WDF to PCA and TM, while it outperforms other derivative methods. The first reason is the use of EAP waveform features. As shown in earlier sections, EAP waveform features perform better and show less degradation than derivative features in increasing AWGN. This is primarily due to the fact that differentiation degrades the SNR under AWGN, hence the inclusion of EAP waveform features in the algorithm to improve sorting performance at higher AWGN levels. The second advantage of WDF is due to the nature of feature extraction itself.

WDF relies on given sample positions, P, and alignment with respect to EAP waveform peak to perform feature extraction. As previously mentioned, EAP waveform is more immune to AWGN, therefore using EAP waveform peak as a reference for feature extraction positions of second derivative space reduces possible errors. In addition, feature extraction positions are determined using template waveforms (calculated through initial clustering) during off-node training. Since these templates are created by averaging, the possible incorrect selection of useful local extrema on second derivative space are eliminated. For example, one of the compared methods — FSDE — relies on calculating max/min on the derivative waveforms which with increasing AWGN may introduce artificial spikes that can be picked up as the max/min (when they are not) resulting in incorrect feature extraction. Therefore, a combination of off-node training on mean waveforms and feature extraction based on pre-determined index positions with respect to EAP waveform eliminates such errors resulting in WDF achieving better performance under high AWGN.

5.3.3.2. Computational Complexity

In addition to sorting accuracy, computational complexity is also a crucial factor within the context of implantable spike sorting hardware due to resource utilisation constraints (i.e. power, bandwidth and volume). Here, the computational complexity is defined in terms of the number of additions and multiplications. For detailed estimation of relative complexity for each feature extraction and clustering method, please refer to Table 5.2 (in Section 5.1.2). To assess the accuracy and computational demands, a single complexity figure-of-merit (CFOM) as in [96,98]

[11] The sorting accuracy is averaged across all background noise levels and datasets.

[12] "Low AWGN" is defined as less than 20dB SNR decrease due to AWGN, while "high AWGN" represents more than 20dB decrease in SNR due to AWGN.

is produced and given by Eq. 5.7 (Section 5.1.2). It should be noted that any comparison operations required are counted as an addition operation for the CFOM.

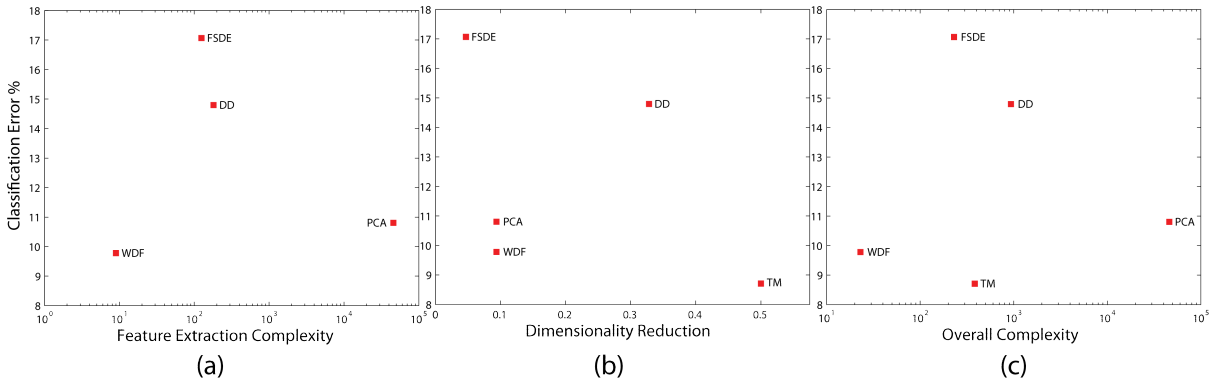


Figure 5.16.: Assessment of classification error (averaged across all datasets and noise levels) from complexity and dimensionality perspective. Shown are the results for classification error with respect to: (a) feature extraction complexity only, (b) dimensionality reduction, and (c) overall complexity including both the feature extraction and sorting performance.

The spike sorting accuracy and computational complexity trade-off (averaged across all datasets and noise levels) is shown in Figure 5.16. This trade-off is presented in three plots, each illustrating the resource utilisation from a different perspective. Figure 5.16 (a) is based only on the feature extraction complexity (excluding the classification), while Figure 5.16 (c) includes the computational requirements of both feature extraction and classification/clustering methods (e.g. K-means for FSDE). Figure 5.16 (b), on the other hand, measures the complexity in relation to the number of features used in the sorting process which is reflected in dimensionality reduction (i.e. m/n; m number of extracted features, n number of samples).

It is clearly observed from all three plots that the proposed method achieves the best trade-off between sorting accuracy and computational demands. In fact, the only method that outperforms the proposed algorithm is TM (by 1.07%) , which requires 17x more computations. FSDE, in contrast, is the closest sorting method in terms of computational requirements; however, on average it performs 7.23% worse than the proposed algorithm.

5.4. Real-Time Spike Sorting Implementation

Having verified the proposed algorithm as a robust and highly efficient method through Matlab modelling and simulations, this section aims to demonstrate and prove the merits of WDF through a real-time spike sorting implementation. Here a microcontroller based prototype is presented and the performance of the algorithm is discussed in terms of sorting accuracy, computational requirements (instructions per sample), and power consumption. The main motivations for choosing an MCU-based platform are reduced development time (e.g. familiarity of tools, ease of translating algorithms into C), cost (£13 for the development board used), and most importantly the I/O capabilites allowing flexibility in interfacing within the context of

automated testing (specific details of the testbench and testing are discussed below). Although the chosen implementation method do not present the most power efficient implementation (e.g. compared to ASIC), nevertheless the emphasis here is on the computational efficiency due to simplicity of the algorithm, not the implementation. Hence, even on a generic MCU-based solution the WDF is demonstrated to be highly efficient and accurate. For the real-time demonstration of WDF, Keil μVision 5 embedded development environment is used.

This section is organised as follows. First, in Section 5.4.1, the WDF is translated into C programming language to assess its computational requirements (instructions per sample) and compare its accuracy to MATLAB simulations. In Section 5.4.2, WDF is implemented on a microcontroller platform which is then followed by hardware specific validation and measurements. In Section 5.4.3, WDF is demonstrated within a full signal-processing chain paired with an Intan RHD2216 (analogue-front end) and an off-node training component together with a graphical user interface (GUI).

5.4.1. C Implementation

Prior to MCU implementation, the WDF have been written and tested in C programming language using 'Microsoft Visual C++ Express 2010' in order to assess the sorting accuracy difference with MATLAB models, conduct a preliminary assessment of computational complexity of the algorithm in terms of instructions required per sample, and optimise the algorithm where necessary.

The sorting accuracy difference between the Matlab and C implementation (across all noise levels and datasets) are demonstrated in Figure 5.17. In general, one can observe that the accuracy differences lie within a few percent of each other. In fact, the average sorting accuracy difference between two implementations is 1.06% with standard deviation of 1.43%. The largest difference is observed to be 7.7% at the highest noise level for Easy1 dataset, while for the remaining datasets the C-implementation have slightly improved performance (maximum of 2.5% difference). This slight difference in performance is attributed to the effect of quantisation due to different data types used by methods (integer for C-implementation and double for Matlab simulations).

In order to estimate the computational complexity and highlight the most executed areas of the C-code (hence require optimisations), "GCC Profiler" is used to analyse the number of instructions required per sample and code coverage. The results reveal an estimated average instructions per sample value of 30[13] while the maximum instantaneous value observed is 104 (including spike detection, alignment, feature extraction and voting). Figure 5.18 presents the complexity (in terms of instructions executed) for specific sections of the C-implementation.

[13]This value is based on: Total number of instructions executed/Total number of test samples.

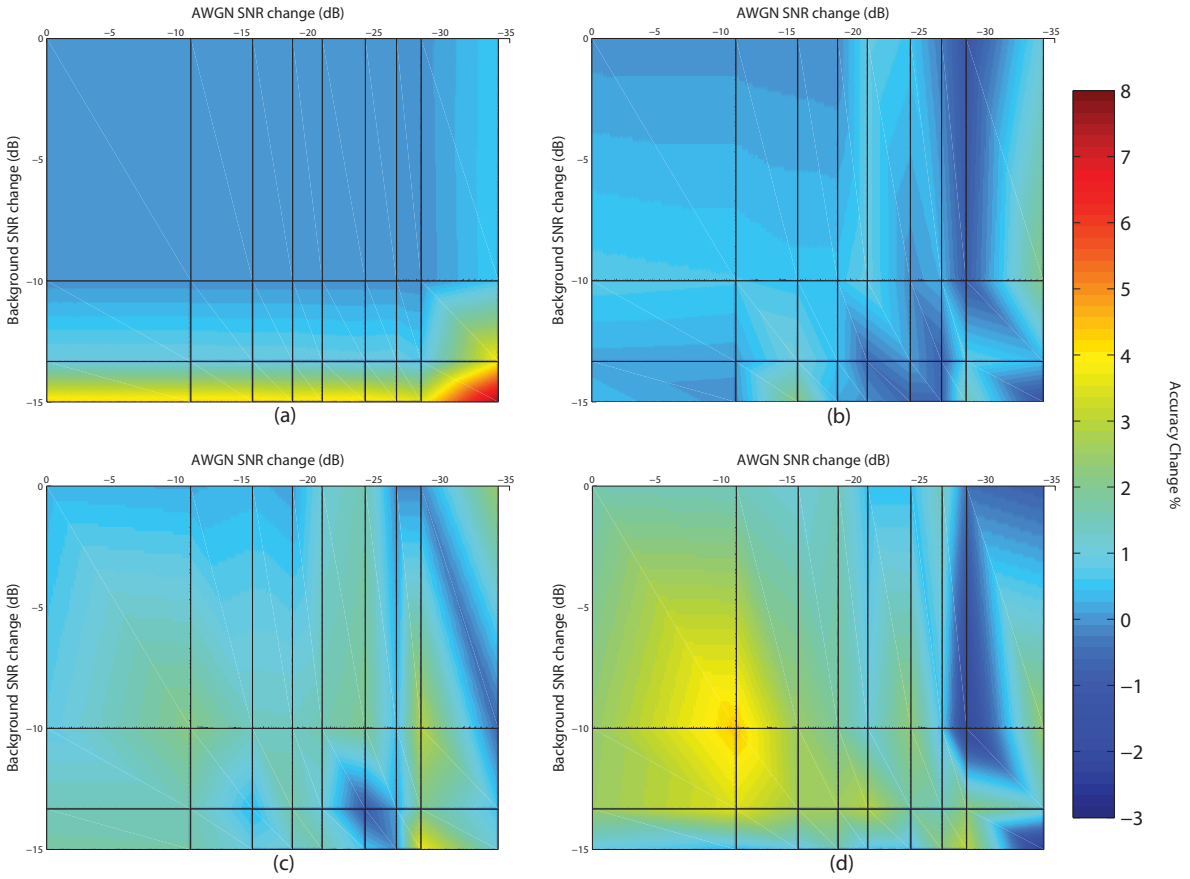


Figure 5.17.: Spike sorting accuracy difference between Matlab and C implementations at various background activity and AWGN levels of four datasets(a-d). (a) Easy1 (b)Easy2 (c)Difficult1 (d) Difficult2

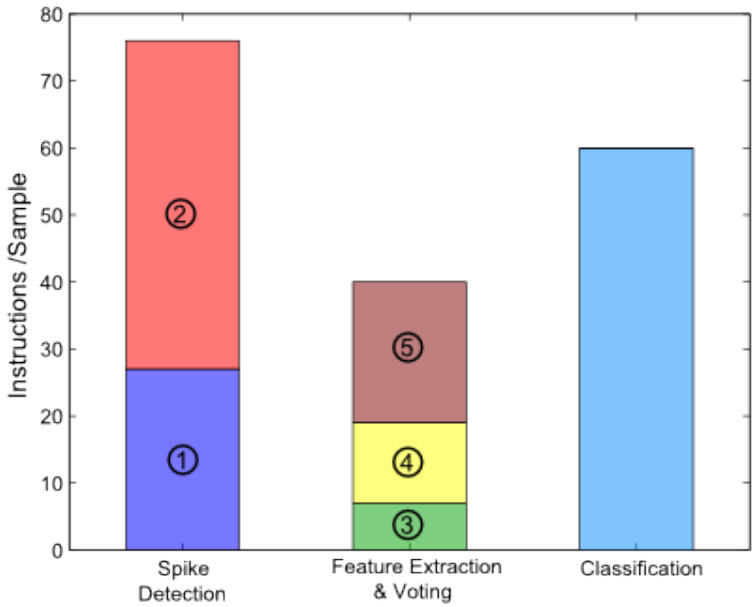


Figure 5.18.: Analysis of computational requirements for different sections of C-implementation tested on 'Microsoft Visual C++ Express 2010' (prior to MCU implementation). The numbered regions represent: (1) Running spike detection algorithm. This is the minimum number of instructions, 27, for each sample, (2) Spike alignment to the maximum peak value, (3) Minimum instructions required for feature extraction (i.e. a single waveform feature), (4) Maximum instructions required for feature extraction (i.e. both waveform and second derivatives features are extracted at the same sample position), and (5) voting based on extracted features.

5.4.2. Microcontroller Implementation

The microcontroller of choice for the implementation is Kinetis KL25Z provided by Freescale Freedom Development Platform (FRDM-KL25Z). Running an ARM Cortex-M0+ processor, KL25Z MCU is one of the most power efficient and low-cost platforms that provides necessary communication interfaces (such as SPI, UART and USB) and system peripherals (e.g. DMA controllers) for efficient real-time implementation, test and verification of the proposed WDF algorithm. Keil uVision 5 is used as the embedded development environment.

An automated test platform is built in order to verify the performance of the real-time operation (Figure 5.19). Central to the testing platform is the workstation which performs the training on spike data, uploads the parameters (required for online sorting) to the MCU, receives the sorted data from the MCU via UART link, reads the time information recorded by the oscilloscope over network, controls 8PI Control Panel to load and run datasets on WaveXpress Waveform Generator, and assesses the spike sorting accuracy.

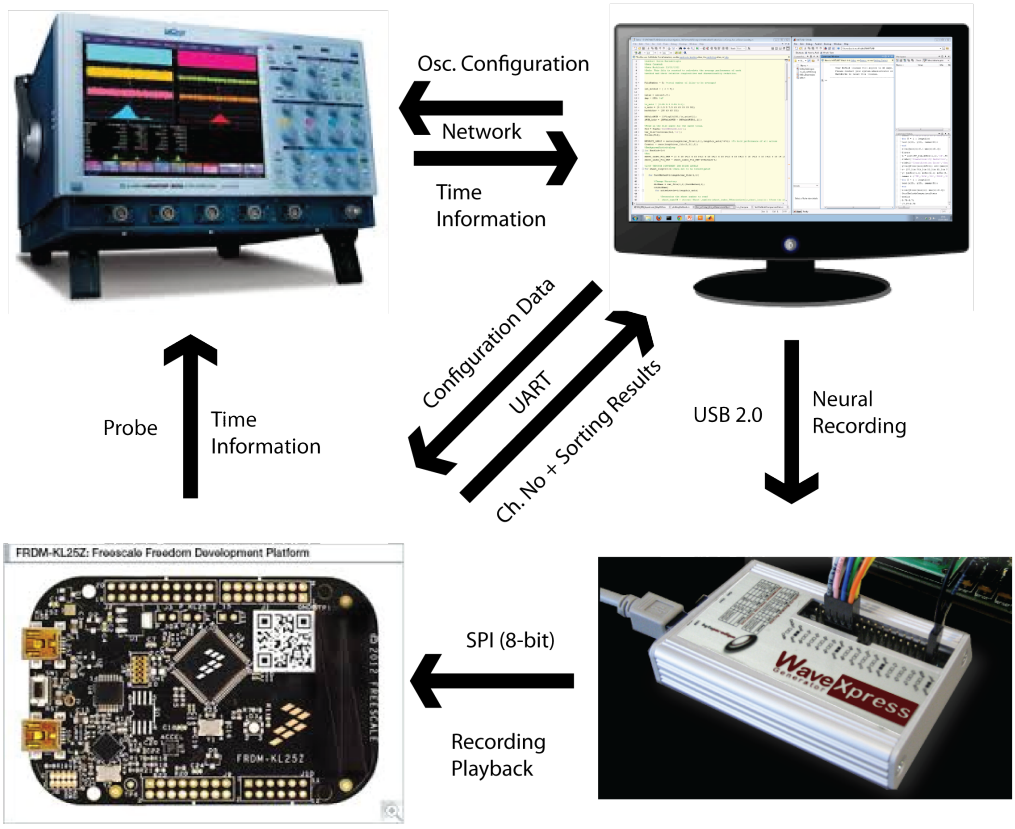


Figure 5.19.: Test platform for verifying real-time spike sorting implemented on Freescale FRDM-KL25Z board. Test platform include a PC (running MATLAB R2011B and 8PI Control Panel), WaveXpress Waveform Generator (by ByteParadigm), FRDM-KL25Z, and LeCroy WavePro 7300A.

The sorting accuracy comparison of MATLAB simulations and real-time implementation is given in Figure 5.20. The performance loss compared to Matlab is 5.52% (standard deviation of 1.74%), while the maximum difference observed is 9.32%. Considering that the sample

resolution for real-time implementation is 8-bits (compared to double-precision floating point in MATLAB), loss of 5.52% accuracy on average is an expected trade-off between performance and hardware resource utilisation. The choice of 8-bit data resolution have been made in the light of the results and discussions in Chapters 3 and 4.

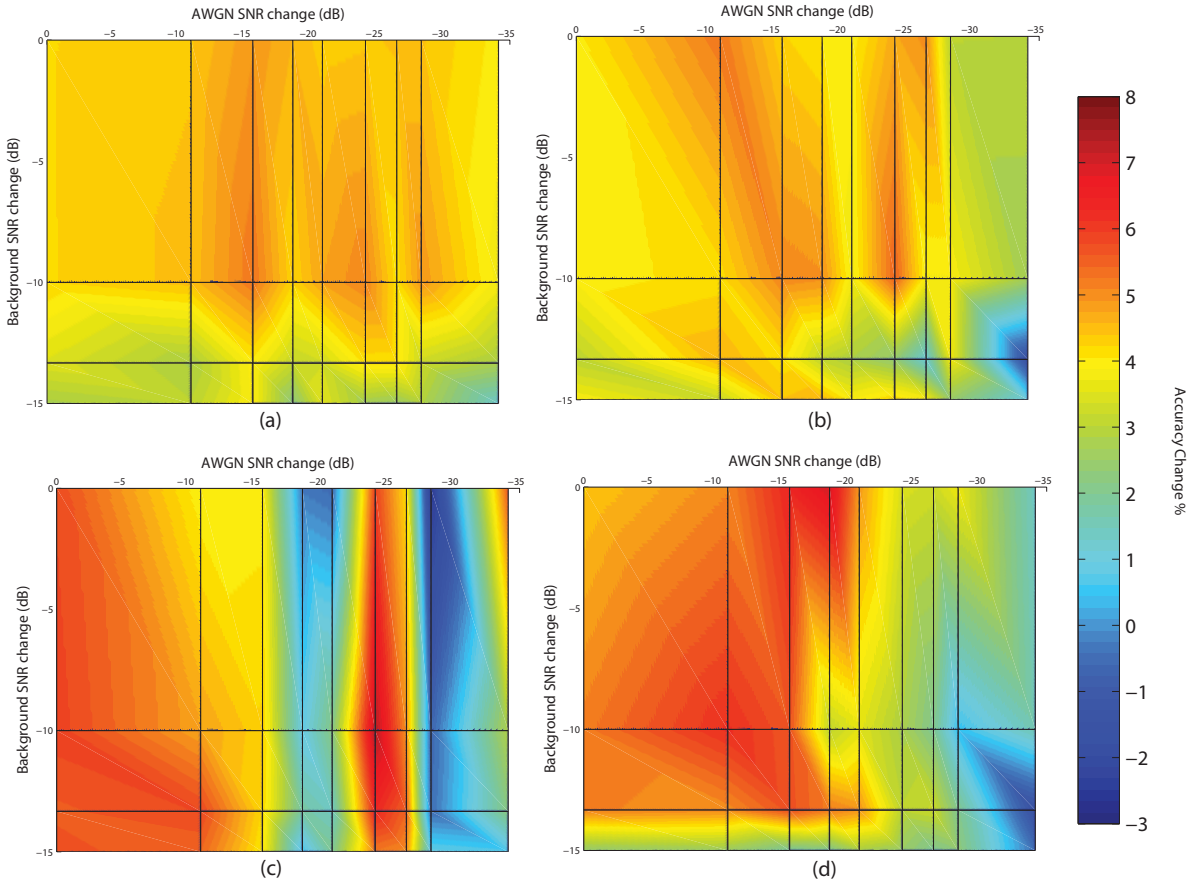


Figure 5.20.: Spike sorting accuracy difference between Matlab and MCU implementations at various background activity and AWGN levels of test datasets(a-d). (a) Easy1 (b) Easy2 (c) Difficult1 (d) Difficult2

Besides quantifying (hence confirming) the performance of the proposed algorithm, the second objective of real-time implementation is to demonstrate the efficiency and scalability of WDF. Therefore, the average power consumption of the KL25Z MCU have been measured across various aspects of its operation and summarised in Figure 5.21. The power consumption is presented as per channel basis and the power analysis is broken down into three major components: Idle, Communication, and Detection and Sorting. The results have been measured across a 120-ohm resistor connected in series to KL25Z (via the header provided by FRDM board) and the current consumption is recorded via the oscilloscope set up. Three measurements have been made and the power consumption quoted for each component is inferred from these. These measurements are: (1) Wait Mode active, while communication, detection and sorting are deactivated; (2) Communication is activated and data is streamed in, while sorting and detection is disabled;

and (3) Total power consumption (during real-time sorting).

The idle component is the power consumed by KL25Z while it is in 'Wait Mode'. It represents the power used by having the device on, and it is measured in the absence of communication, spike detection and sorting. Despite the system clock to Cortex M0+ being disabled during this mode, a considerable energy is being spent to have the MCU on (and this is the dominant source of power consumption beyond 5-channels). On the other hand, power consumption associated with communication include having the SPI module on and actively streaming in data. It should be noted, however, that the power consumed for transmitting the sorting results is not included in communications figure but are incorporated together with detection and sorting results.

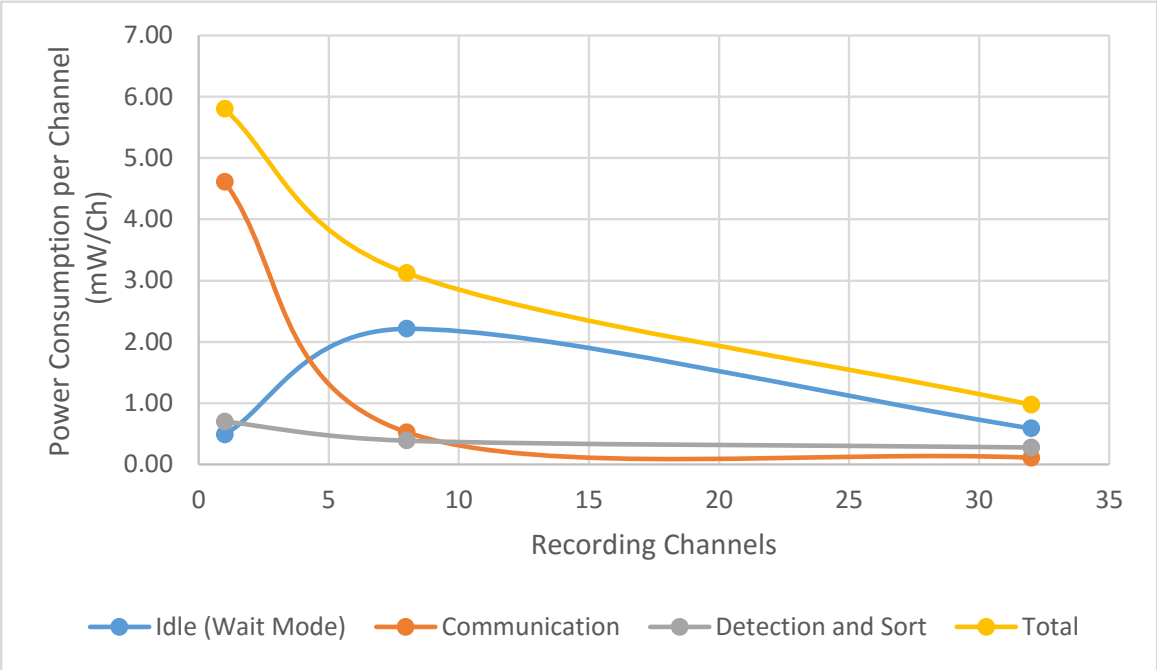


Figure 5.21.: Power consumption analysis of real-time spike sorting implementation on FRDM KL25Z Board.

When the power breakdown is analysed, one can clearly see that the power consumption required for spike sorting and detection stay constant (around 268 μW per channel) as the number of channels increased, hence showing that the proposed algorithm is very efficient (considering a generic MCU platform implementation). However, the most important observation is the fact that the WDF algorithm is linearly scalable with increasing number of channels.

5.4.3. Complete System Implementation

Following confirmation of the sorting accuracy and resource efficiency of WDF (and its implementation), a complete system demonstrator is proposed incorporating an off-the-shelf analogue front end, an MCU (real-time sorting developed above), and Graphical User Interface (GUI). Neural activity is emulated by playing neural recordings (stored on a USB stick) via Agilent

33522A Arbitrary Waveform Generator. Since the lowest amplitude resolution of the waveform generator is 1 mV, an attenuator followed by a buffer is implemented to provide realistic neural signal amplitudes. The details of the system set-up and components are presented in Figure 5.22.

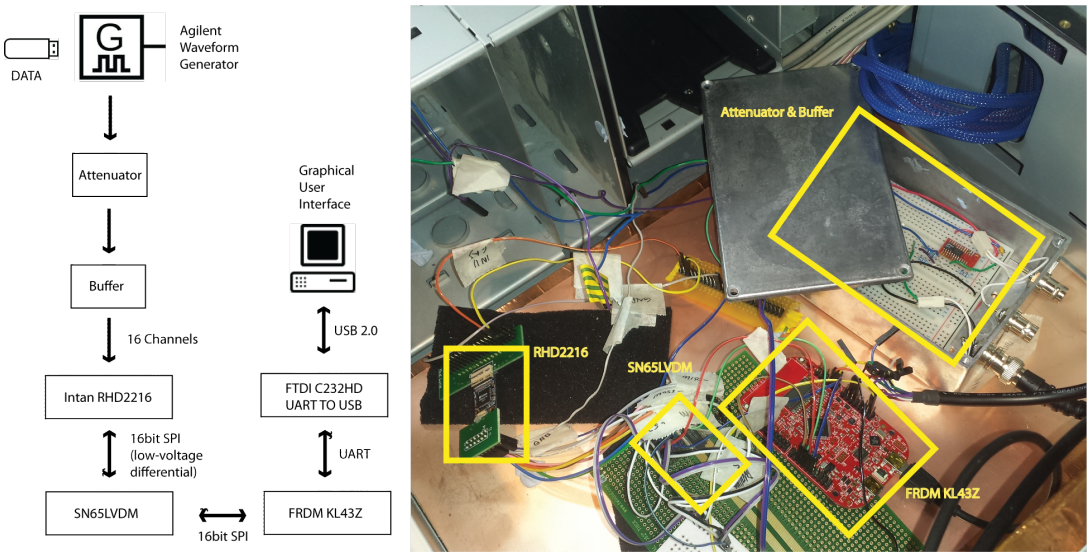


Figure 5.22.: Complete system implementation comprising of analogue front-end (Intan RHD2216), real-time on-node spike sorting (FRDM KL43Z), and Graphical User Interface (on PC).

5.4.3.1. Intan RHD2216

RHD2216 Amplifier board (by Intan Technologies) is chosen as the front-end module implementation. It provides 16-amplifier channels with software configurable amplifier bandwidth (0.1 Hz to 20 kHz) and sampling rates (1kS to 30kS), as well as an SPI connection (standard or low-voltage differential signalling (LVDS) option) which provides an easy interface for the MCU. It should be noted that since RHD2216 board provided was set by default to LVDS, SN65LVDM is used to convert from LVDS to standard SPI signalling.

5.4.3.2. Microcontroller

For the complete system implementation, Freescale FRDM-KL43Z board is chosen instead of FRDM-KL25Z. This is due to the fact that RHD2216 requires 16-bit SPI connection to communicate instructions and data, and KL25Z MCU only has 8-bit SPI option. Besides providing 16-bit SPI, KL43Z MCU also has 32KB of SRAM (compared to 16KB of KL25Z). Despite some differences with KL252Z (e.g. pin configurations, peripherals, clock distrubtion and sources), KL43Z MCU still incorporates an ARM Cortex M0+ and the communication peripherals used previously, hence the developed C-code is easily portable with minor modifications (e.g. changing I/O pin assignments to SPI and UART).

5.4.3.3. Graphical User Interface

Graphical User Interface have been developed using "GUIDE Tool" provided by MATLAB R2013b, and is the main hub allowing users to:

1. stream raw neural recording, perform training, and load the parameters required for real-time spike sorting to MCU.
2. monitor the real-time spike sorting output.
3. display neural recording statistics such as ISI diagrams and spike count rates.
4. save a new recording session or load a previous one for further analysis.

Figure 5.23 presents a general overview of the GUI. There are three main panels: "Settings", "Mode", and "Plot". Settings panel allows the user to enter both the "Channel" and "Uart" settings. Channel settings include parameters such as number of channels and sampling rate, while the "Uart Settings" establishes the communication protocol's parameters such as the port number and baud rate, as well as sample resolution (currently chosen between 8 or 16 bits).

"Mode Panel", on the otherhand, provides user a selection of data visualisation methods (or modes) which are "Neuron", "Event", and "Statistics" modes. The options under "Neuron Mode" allow the user to observe both the waveforms associated with recorded neurons ("Template View") and the raw neural recording used for training ("Raw Data"). "TRAIN" button initiates the raw data streaming from the MCU, performs clustering on the dataset ("WaveClus") to create template and then runs the training algorithm to identify sorting parameters described previously. Following training, user can display the real-time sorting results via "Event Mode". Neural recording statistics such as ISI diagrams and spike count rates can also be visualised under "Statistics Mode". Besides these, the GUI also allows user to save the current session or load a previous one. Figure 5.24 demonstrates a subset of functionalities of the GUI in operation.

5.5. Conclusion

In this chapter, the potential of derivative features as an alternative hardware implantable spike sorting method have been investigated. The proposed methods have been tested against commonly used sorting algorithms for numerous datasets under varying noise conditions, and found to strike the best balance between resource efficiency and accuracy.

In Section 5.1, a new feature extraction method, not requiring any calibration, based on first and second derivative features of the spike waveform is proposed. Its accuracy and computational complexity are quantified and compared against commonly used feature extraction methods across four datasets at various background noise levels.

The results reveal that the proposed methods strikes a good balance between the achieved classification error and the number of features (i.e. dimensions) used. Moreover, when computational requirements are compared, FSDE provides the best trade-off in terms of the number of

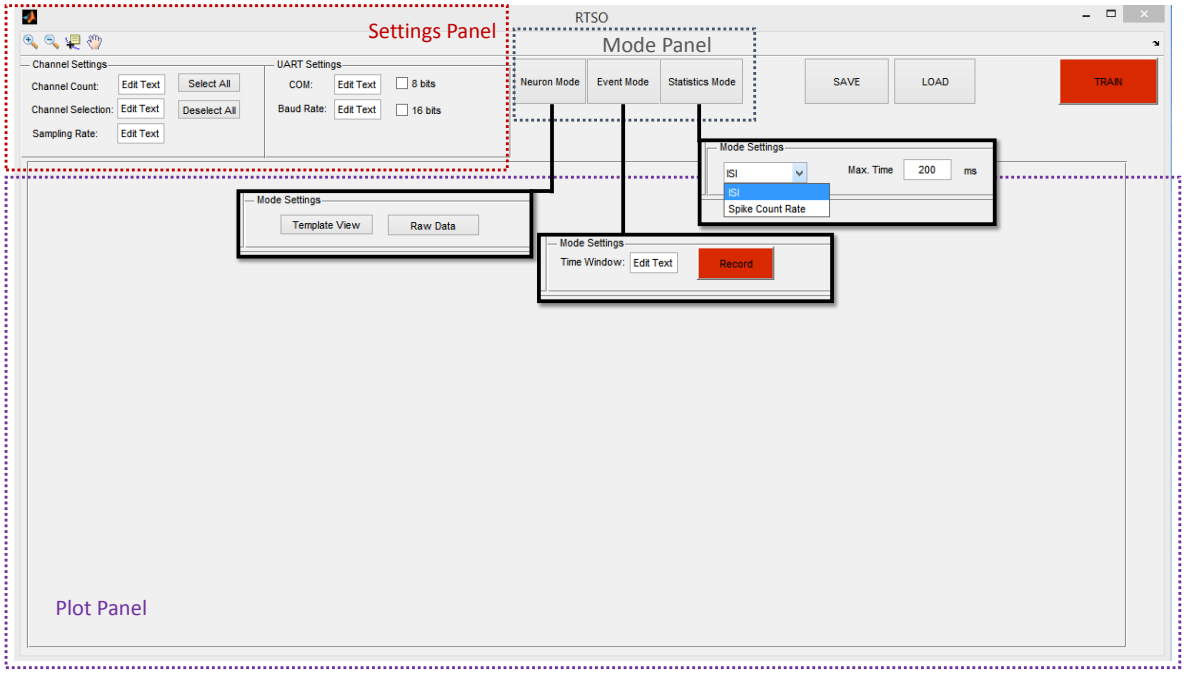


Figure 5.23.: Graphical User Interface description detailing functionality.

computations and the achieved sorting accuracy. In fact, from Figure 4.7 one can observe that on average, FSDE is more accurate and 4× more efficient than the closest contender (DD_{var}), while the only better performing method D_{all} requires 35× more computations.

The average classification error is shown to be below 7% with a computational complexity of $2N - 3$, where N is the number of sample points of each spike. Overall, this method presents a good trade-off between accuracy and computational complexity and is thus particularly well-suited for hardware-efficient implementation.

In Section 5.2, an improved set of features, which provide high sorting accuracy under varying background activity and AWGN noise levels, is proposed. Following initial hypotheses based on EAP waveform dynamics, it is shown through an empirical study that a subset of features given by local extrema and inflection points of the EAP waveform and its derivatives (up to third order) provide the majority of separation between neurons. These features are then tested under varying background activity and AWGN levels, and a combination of EAP waveform and second derivative features are proposed to provide high accuracy under varying noise conditions. Building on these results, in Section 5.3, a new spike sorting method (WDF) — based on a simple comparison and voting algorithm utilising original spike and second derivative features — is proposed. Its computational requirements and sorting performance are quantified against commonly used spike sorting methods across four datasets and varying noise levels (both background and AWGN). The results verify that WDF provides the best trade-off in terms of computational efficiency and sorting accuracy. In fact, it has been shown that the closest sorting method in terms of computational requirements, FSDE, performs 7.23% worse on average; while the only method (TM) that outperforms WDF requires 17x more computations. As a

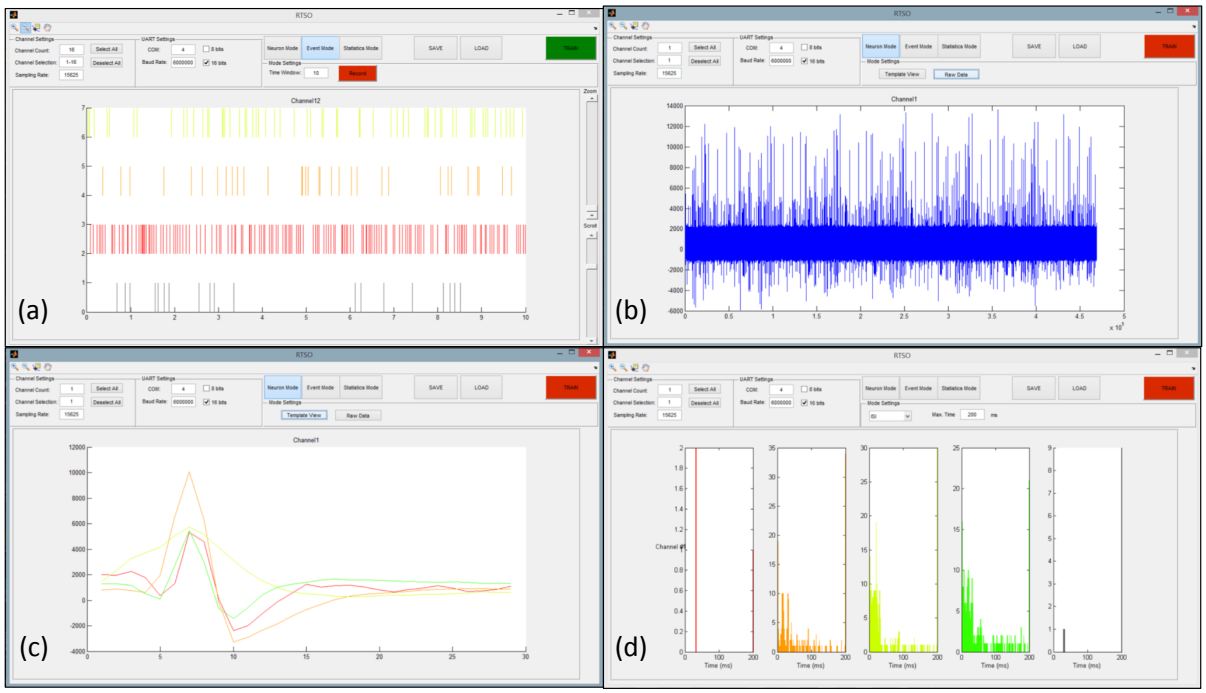


Figure 5.24.: Sample screenshot from GUI in operation. (a) Illustrates the "Event Mode" while the sorted data is recorded. Note the "TRAIN" button change to green (from red) indicating training is complete. (b) "Raw Data" visualisation in "Neuron Mode". This option allows the user to observe the neural recording with which the training is performed. (c) "Template View" in "Neuron Mode" illustrating the user the waveforms associated with each neuron observed. (d) "Statistics Mode" displaying the ISI diagrams for real-time sorting.

proof-of-concept for efficient on-node real-time sorting, WDF is implemented on Freescale KL series MCU running an ARM Cortex M0+ processor and paired with Intan RHD2216 (front-end recording) and a custom graphical user interface.

The key in success of the proposed algorithm is rooted in the initial hypotheses that led to this work. It has been shown that there exists features based on waveform dynamics which are defined by the local minima/maxima and inflection points of the original waveform and its derivatives up to third order; and a subset of these contain majority of the information to discriminate the action potentials. Such reduction in the feature space not only enabled a real-time method that performs well and efficiently even in the lowest SNR levels, but also contributes greatly to the simplicity of the training algorithm. Therefore, limiting the number of features (as well as from which feature domains) to use, and only looking for the most differentiating features at specific locations in relation to EAP waveform dynamics (as described previously) are the two major decisions also made for a simpler training algorithm.

Although the proposed algorithm is demonstrated successfully, only making it into a complete product (with small form factor) and demonstrating through in-vivo experiments will realise its full potential. As described earlier, the implementation of the demonstrator system followed

the two-stage hybrid strategy studied in Chapter 4.2 with slight modification to the training algorithm and replacement of template matching with WDF. However, unlike the two-stage hybrid system proposed, the training stage included transmitting raw signals instead of detected spikes. As shown in Chapter 4.2, transmitting raw signals for training is highly un-efficient for large scale monitoring of neurons, and was only implemented due to small number of channels targetted for testing. In addition to this, active noise tracking and detection threshold calibration could be further valuable improvements on the system.

Another critical decision involves choosing the platform to implement the algorithm through. As mentioned earlier, the main reasons for choosing an MCU-based (with ARM Cortex M0+) platform are reduced development time (e.g. familiarity with the tools), cost and abundant I/O capabilities allowing flexibility within the context of automated testing. However, there are several different strategies that could be investigated within the context of a compact portable platform. One such possible strategy involves distributing the processing on multiple MCUs (perhaps separating detection and sorting) and taking advantage of low power modes at reduced clock frequencies. Another possible solution involves evaluating some of the low-power FPGA platforms that have emerged over the past years. For example, IGLOO nano series provide extremely low power FPGAs that could be a more viable solution. In fact, the prospect of increased parallelism — hence reduced latency — better suits real-time operations. Last but not the least, the custom graphical user interface needs further improvements. As mentioned earlier, GUI was developed through Matlab's own "GUIDE" tool. Despite working for the purposes of demonstration, the real-time performance of Matlab based systems are far from ideal. Therefore, C/C# based GUI is suggested for future work.

Therefore, the next stage in taking this work further must include careful considerations of implementation mentioned above to translate the algorithms and methods developed into a tool used in neuroscience labs.

6. Conclusions

Deciphering the secrets of the brain is one of the greatest challenges of the 21st century. So close to us yet still unknown, accessing and understanding the brain holds the key to a better future for humanity. Hidden among the quadrillion electro-chemical connections are the cures for neural diseases and the information for countless assistive devices for the disabled. Moreover, inspiring a host of new approaches to computation (such as neuromorphic computing), our understanding of the interactions of the neural circuitry in the brain is likely to cause a paradigm shift for future computing.

To realise all the above and beyond, there have been a surge of researchers trying to tackle challenges (at various levels) of interfacing the brain for the past decades. The recording capabilities have improved tremendously, and Brain-Machine Interfaces can now record large groups of neurons by multi-channel, multi-electrode micro-electrodes. These developments have given neuroscientists invaluable opportunities to conduct experiments (both animal and human) to advance our understanding of diseases and harnessing brain power to help disabled through assistive devices. Initially conducted via external recording systems connected to electrodes via wires, there have been a huge demand for implantable recording systems with wireless telemetries. The two major driving factors that have seen many implantable recording systems emerge over the years are: the risk of infection due to wires passing through skin and scalp, and the limitations of the wired connection in terms of subject mobility. Therefore, implantable recording systems are crucial in both advancing the research in neuroscience labs and translating these technologies into real-life situations (e.g.control of a prosthetic limb). However, an ever increasing demand for monitoring large groups of neurons puts stringent constraints on the implantable system resources in terms of bandwidth and energy consumption. Hence reducing data dimensionality is of utmost importance for future scalability of neural recording systems.

The work presented here aimed at investigating the real-time on-node spike sorting as method to reduce dimensionality of the data prior to transmission. Therefore, this thesis explored various parameters at different levels of the signal processing chain and developed (as well as demonstrating) new spike sorting algorithms.

6.1. Original Contributions

This thesis has made original contributions in the following main areas: (1) developing a behavioural analogue front-end model and an optimisation tool to provide designers a platform for investigating the effects of front-end parameters on spike sorting accuracy, (2) quantifying the

sensitivity of spike sorting accuracy to front-end parameters (using developed optimisation tool), (3) quantifying the sensitivity of accuracies of both the template matching and the proposed two-stage stage hybrid spike sorting strategy to parameters related spike sorting and analogue front-end, (4) developing a novel hardware-efficient feature extraction method based on first and second derivative extrema — FSDE, (5) developing a novel hardware-efficient feature extraction and classification method — WDF — that employs a selection of channel-specific near optimal set of features.

Chapter 3 focused on the optimisation of analogue front-end (AFE) for realising accurate and efficient on-node spike sorting. As previously mentioned, AFE is a crucial pre-processing stage which not only affects the performance of the sub-sequent stages (hence overall spike sorting accuracy), but also impacts the power and area utilisation in hardware. Therefore, AFE designs need to be carefully optimised to minimise specifications of AFE parameters (hence reducing resource requirements) while maintaining spike sorting accuracy. To achieve such optimal design, the primary research question centred around identifying analogue front-end parameters and quantifying the sensitivity of spike sorting performance to these parameters. As the first objective, a behavioural model and a tool is developed (in Section 3.2) to serve as an optimisation framework. The developed model and tool allow the designers to investigate the effect of different front-end parameters during design time so that a good balance between resource efficiency and sorting accuracy can be achieved. Having developed the behavioural model and tool, the sensitivity of spike sorting accuracy to AFE paremeters are quantified in Section 3.6. These parameters are: (1) Gain-Bandwidth and noise for the amplifier, (2) Filter order, type, and cut-off frequencies (both high and low pass) for filtering stage, and, sampling frequency and resolution for analogue-to-digital conversion stage. These are investigated for three different spike sorting methods (Principle Component Analysis, Template Matching and Second Derivative Features) over 24 datasets, and a set of near optimal solutions are proposed.

Considering the on-node resource utilisation is also directly linked with the back-end digital electronics, Chapter 4 focused on establishing minimum requirements from the spike sorting perspective. Section 4.1 investigated key metrics and parameters associated with on-node template matching which includes (but not limited to) interpolation and alignment, template creation, template matching quantifiers, and template window size. Moreover, these are further investigated in terms of some of the parameters explored in Chapter 3.

In Section 4.2, on the other hand, the feasibility of a two-stage hybrid strategy combining both off-line and on-line data processing was evaluated as low-power long term solution to wireless BMIs or closed loop stimulation designs. The first stage involves detecting the spikes on-node and transmitting them to an external computer to create and send back mean templates. The second stage involves running a real-time spike sorting through template matching. Section 4.2 explored the impact of aforementioned parameters (extending Section 4.1) for both off-line and on-line components. Using realistic simulations of extracellular recordings, these parameters are systematically investigated for robust and accurate, but at the same efficient and hardware implementable real-time spike sorting solution.

Chapter 5 proposed novel derivative based feature extraction and spike sorting methods that can be efficiently implemented in hardware. Having shown the immunity of derivative features against background activity, Section 5.1 proposed a feature extraction method based on first and second derivative extrema of the spike waveforms. Compared against commonly used feature extraction methods, it is shown to be 4 × more efficient (as well as being more accurate) than the closest contender while the only better performing method (by only ∼3%) required 35× more computations.

In Section 5.3, a novel hardware-efficient spike sorting method — WDF — is proposed. The proposed method combines an off-node training stage, and a real-time on-node feature extraction and classification stage. During the off-node training, a subset-of features from both the EAP waveform and second derivative feature spaces is selected for each channel. These subset features are selected among local extrema that were hypothesized (based on EAP waveform dynamics) and verified in Section 5.2.

WDF relies on stored feature positions (computed during off-node training) to extract features, and classification comprises simple comparisons and voting. These allow the method to be computationally very efficient, while the combination of second derivative and EAP waveform features provide high spike sorting accuracy under varying noise conditions. Having tested WDF under various datasets and noise levels (both background activity and AWGN), the results reveal that WDF achieves the best spike sorting accuracy and computational complexity trade-off compared to other low-complexity methods. In fact, the only method that outperforms the proposed algorithm (by 1.07%) requires 17× more computations, while the method with closest computational complexity (but less efficient) has 7.23% lower spike sorting accuracy than WDF. Having quantified accuracy and computational complexity, the proposed algorithm was demonstrated in real-time with an MCU based implementation, paired with an Intan RHD2216 AFE and a custom graphical user interface.

6.2. Future Directions

Despite tremendous advances in technology, there are still numerous challenges to overcome in order to improve the capabilities of neural recording systems to monitor large groups of neurons (whether for neuroscience experiments or neural prostheses). Many of these challenges are discussed in Chapter 2 (as well as throughout the thesis), while the specific improvements on presented work are included in conclusions drawn at the end of each chapter. Therefore, these will not be discussed in this section. Instead, the focus will be on future role of cortical implants integrated with spike sorting.

6.2.1. On-node Spike Sorting

As the advances in recording technology provides ever increasing number of neuronal populations to monitor, on-node spike sorting will continue to be the determining factor in the amount information we can extract from the brain. In other words, on-node realtime spike sorting will

even be more critical for the future BMI systems that target monitoring 1000s of recording channels whether for neuroscience research or control of assistive devices. Therefore, the efforts in developing resource efficient algorithms and implementations need to be continued for the foreseeable future.

Besides developing efficient algorithms and implementations, one of the most important factors is making these adaptive and self-calibrated. As the EAP waveforms and the background noise show variations over time (see Chapter 2), ability to adapt to varying conditions provides longer-term and accurate monitoring of neuronal activity. Not only this enables neuroscientists to perform continuous longer-term experiments with subjects (e.g. in natural surroundings), but also it is essential in clinical translation of BMIs for a life-time use by the patients.

6.2.2. Prosthetic applications: near to mid-term solutions

One of the requirements (among many) for implantable systems required for prosthetic applications (compared to experimental configurations) is the need for long-term (i.e. lifetime) stable and accurate of operation. One of the biggest challenges for devices relying on spike sorting for life-time chronic operation, is the loss of recorded signal integrity over time due to scar-tissue formation at electrode-tissue interface (see Chapter 2). There exists various approaches to alleviate these chronic effects. Improved electrode insertion techniques, manipulations of electrode geometry, and using anti-inflammatory coating agents are some of the many methods that are currently under research. While waiting for a solution from the materials researchers and biologists, alternative near to mid-term solutions that centres purely around engineering and signal processing can be produced.

One such approach is making use of local field potentials (LFPs) in conjunction with spike sorting systems. As described earlier, local field potentials are summed electrical activity of the local neuronal network. Occupying frequency bands less than 300 Hz, these low frequency oscillations do not degrade over time (i.e. stable) due to scar tissue formation. Moreover, being low frequency requires lower sampling rates by the front leading to potential hardware resource reductions. Recently, our collaborators from Movement Lab in Newcastle University have demonstrated that low-frequency LFPs (lf-LFP) contain considerable information regarding upper limb kinematics, and hence can be used to decipher the activity of local neuronal populations. They have demonstrated that given the individual activity of the neurons, a model can be constructed around the lf-LFP with which can later be used to decipher individual neural activity from lf-LFPs (to be used in decoding). Pairing efficient on-node spike sorting systems (such as those investigated in this thesis) with lf-LFPs has a great potential to achieve long-term stable operation, and enable clinical translation of brain-machine interfaces (to control prosthetics, assistive devices etc).

6.3. Concluding Remarks

It is of no doubt that over the past decades our ability to interface nervous system have paved way for great successes in neuroscience and healthcare. We understand the brain better and have devices which have been implanted to thousands of patients successfully such as cochlear implants (for deaf) and deep brain stimulators (for Parkinson's patients). Researchers have even demonstrated patients controlling robotic arms via intra-cortical electrodes which gives hope to millions of people suffering from disabilities (whether due to spinal cord injury or neurological diseases such as multiple sclerosis and amyotrophic lateral sclerosis). However, there is still a long way to go in every aspect of the field from neuroscience to engineering (electrodes, implants, battery life etc.) until these devices are safe, robust, stable and accurate for lifetime, and easily available to public.

Thus, it was the aim of this project to contribute to the efforts in overcoming one of the most challenging obstacles in the field — real-time accurate, robust and efficient on-node spike sorting. I hope that the work presented here — analogue front-end optimisation (Chapter 3); minimum requirements for real-time sorting (Chapter 4); and resource-efficient derivative based algorithms developed (Chapter 5) — will bring us one step closer to better understand and interface the brain.

Appendices

A. Supplementary Figures for Chapter 4

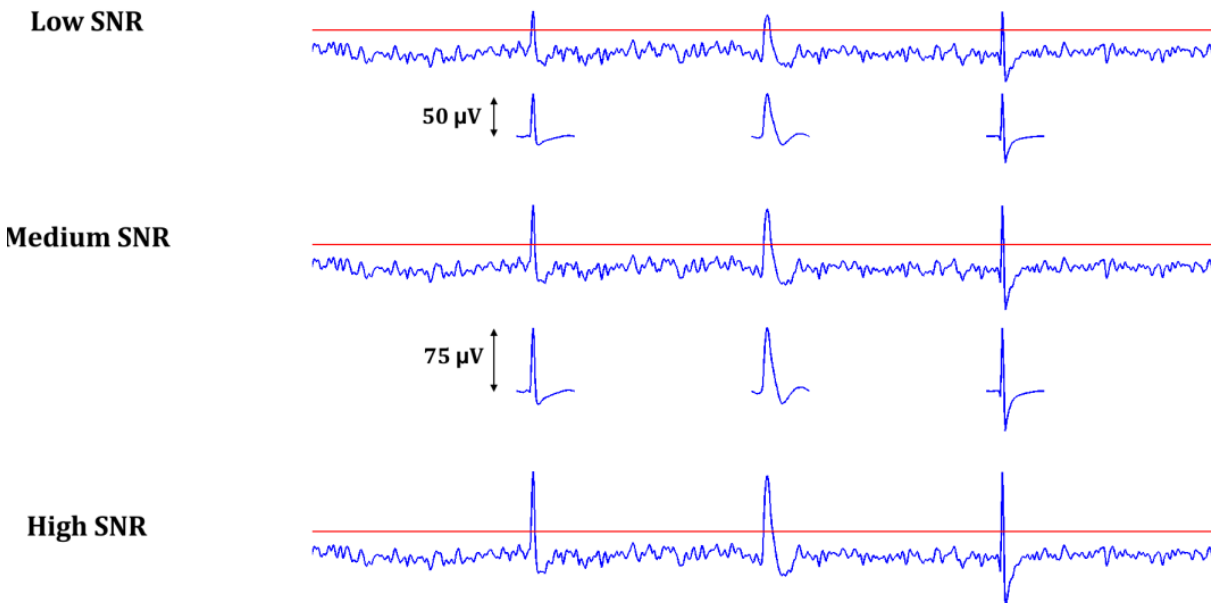


Figure A.1.: Illustration of segments of simulated data at various signal-to-noise ratios (SNR). Shown segments are filtered with a second order causal Elliptic filter between 300 Hz and 3 kHz.

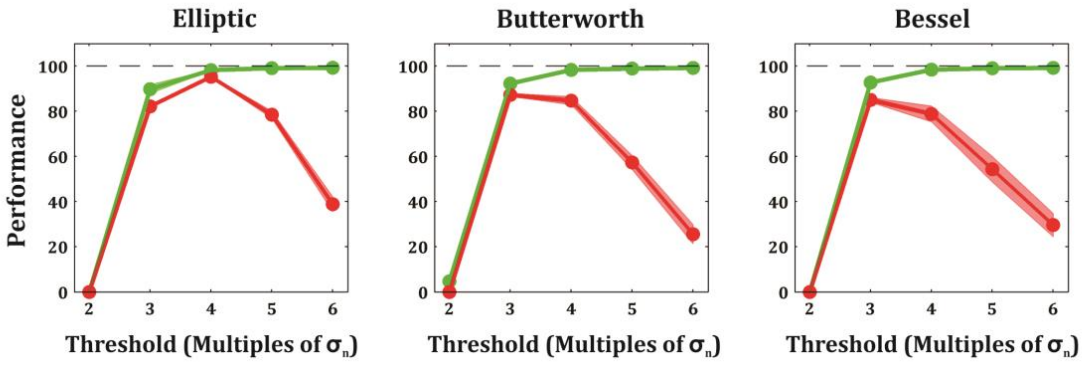


Figure A.2.: Spike detection performance as a function of threshold value, defined as in Equation 1. The green (red) line depicts performance for simulations with high (low) SNR. When the threshold is set to $2\sigma_n$, spike detection performance is too low due to a high number of false positives (as performance is low for simulations with both high and low SNR). When the threshold is set to 6 , performance is very high for simulations with high SNR, but very low for simulations with low SNR, indicating the presence of too many misses. For threshold values between 3 and 5 times σ_n, spike detection performance seems to be acceptably good (>75% lumping together simulations with high and low SNR). The set of parameters that leads to highest spike detection performance are the use of an Elliptic filter and a threshold value of $4\sigma_n$.

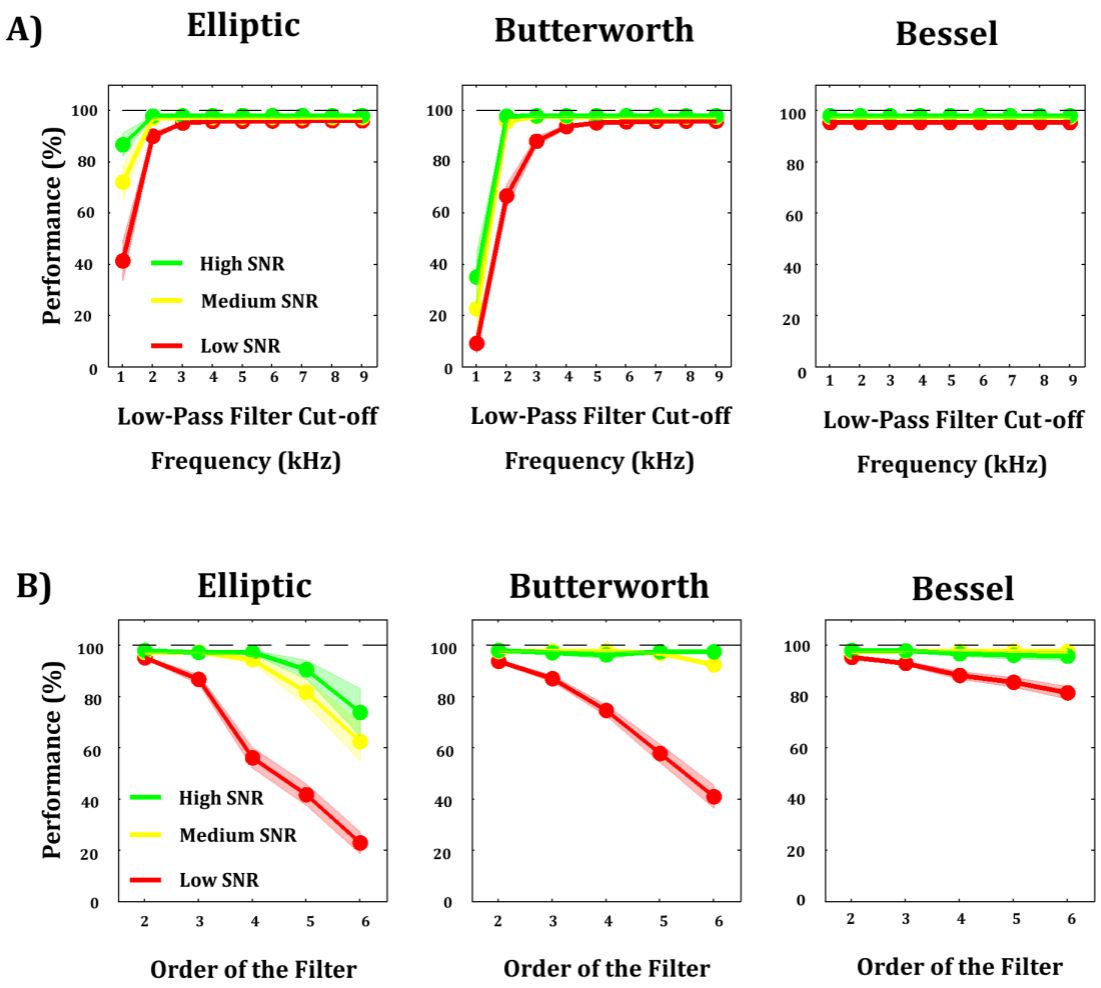


Figure A.3.: Impact of filtering on spike detection accuracy. Spike detection performance for Elliptic, Butterworth and Bessel filters (under different SNRs) at various: (A) low-pass cut-off frequencies, and (B) filter orders.

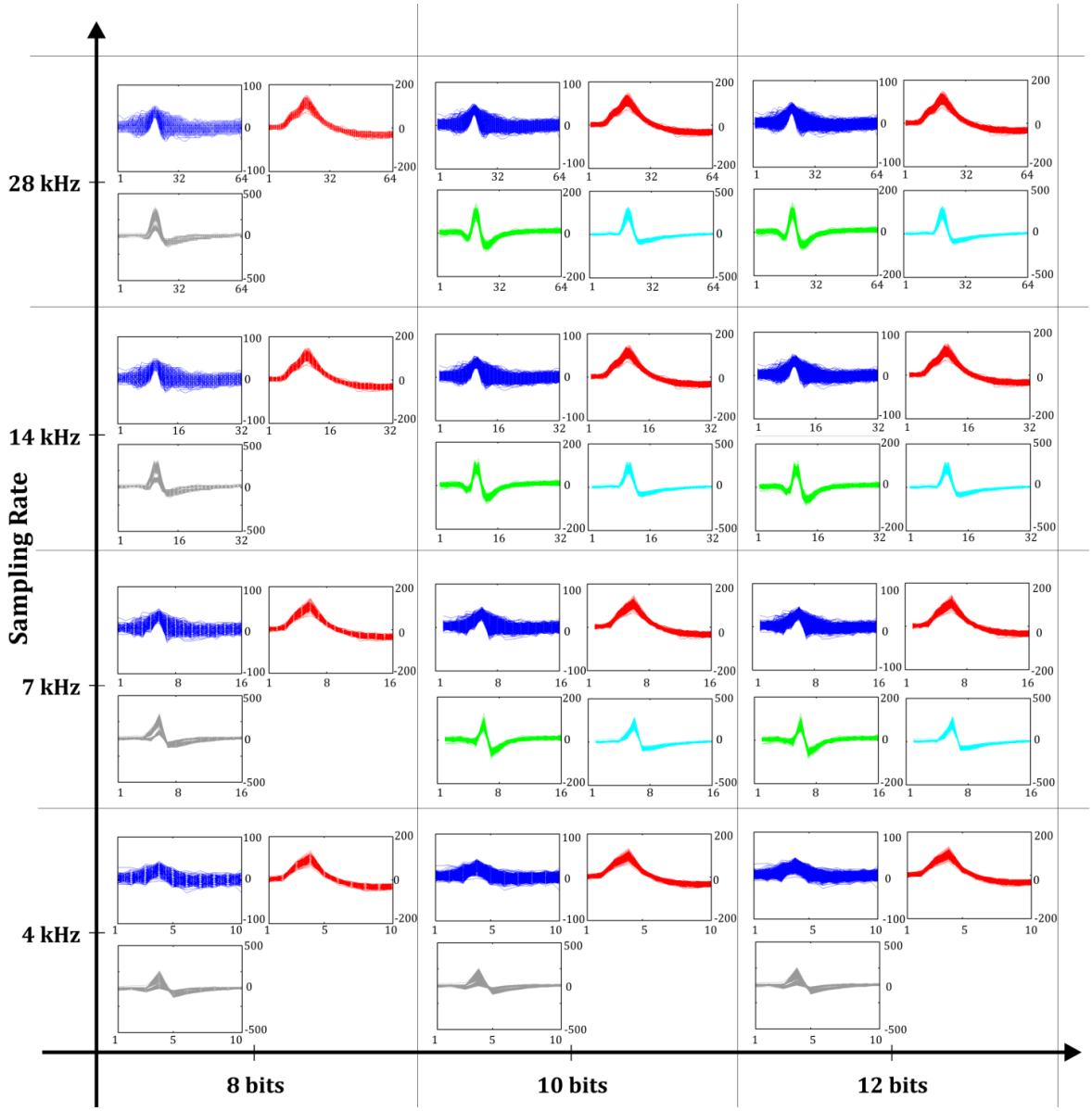


Figure A.4.: Extended example of Figure 4.25C. Illustrated is the incorrect merging of separate clusters due to limitations in sampling rate or signal resolution.

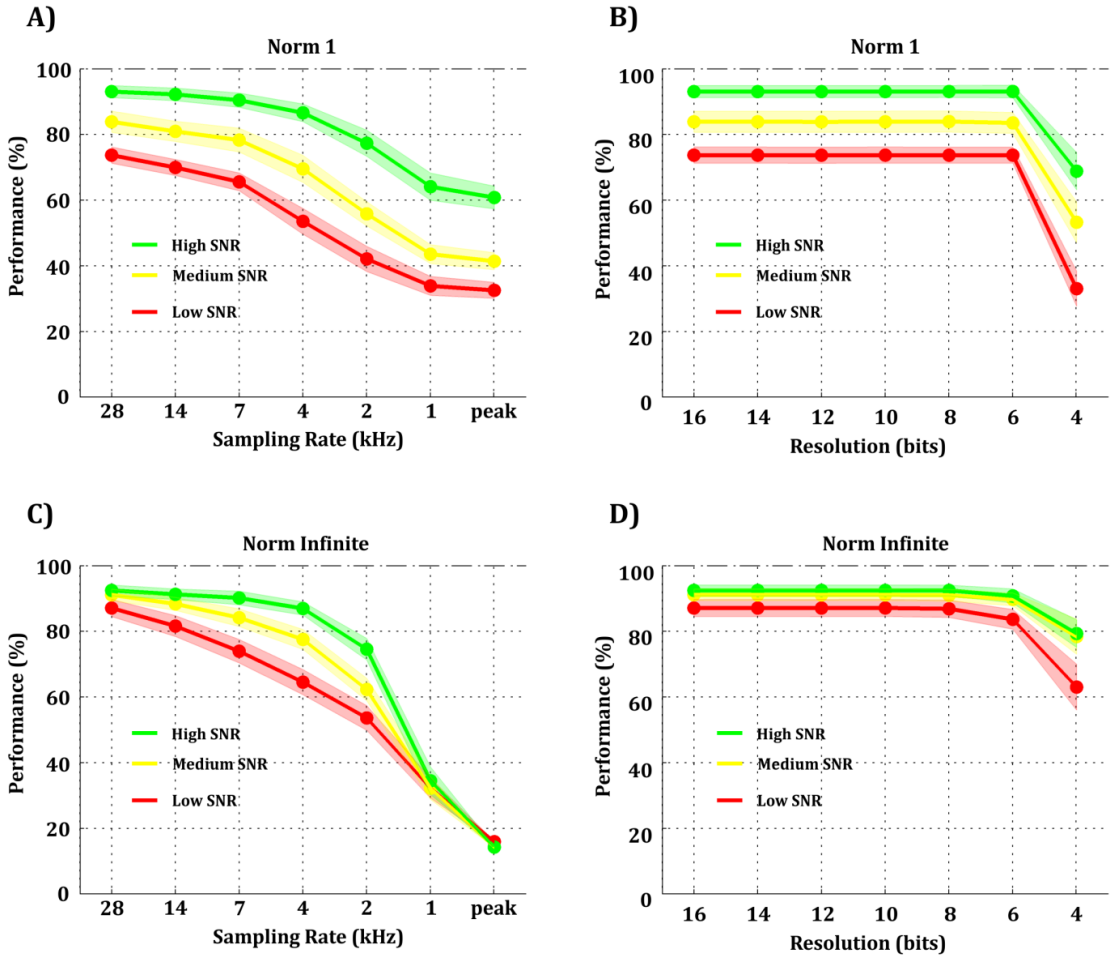


Figure A.5.: Sensitivity of real-time template matching performance (with different metrics) to sampling rate and signal resolution. Performance of Norm1 distance at various (A) sampling rates, and (B) signal resolutions. Performance of Norm Infinite distance at various (C) sampling rates, and (D) signal resolutions. The results are given for high (green), medium (yellow), and low (red) SNRs.

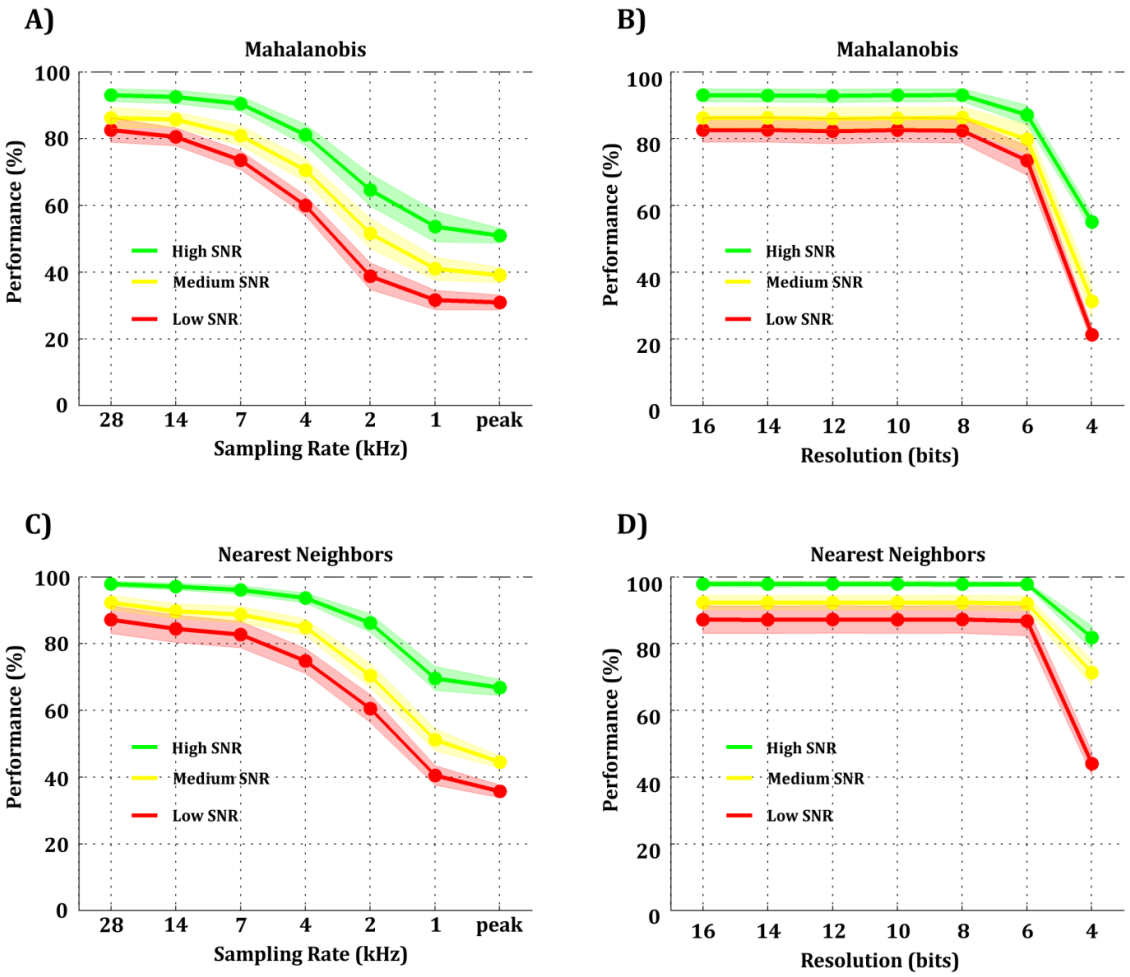


Figure A.6.: Sensitivity of real-time template matching performance (with different metrics) to sampling rate and signal resolution. Performance of Mahalanobis distance at various (A) sampling rates, and (B) signal resolutions. Performance of Nearest Neighbours distance at various (C) sampling rates, and (D) signal resolutions. The results are given for high (green), medium (yellow), and low (red) SNRs.

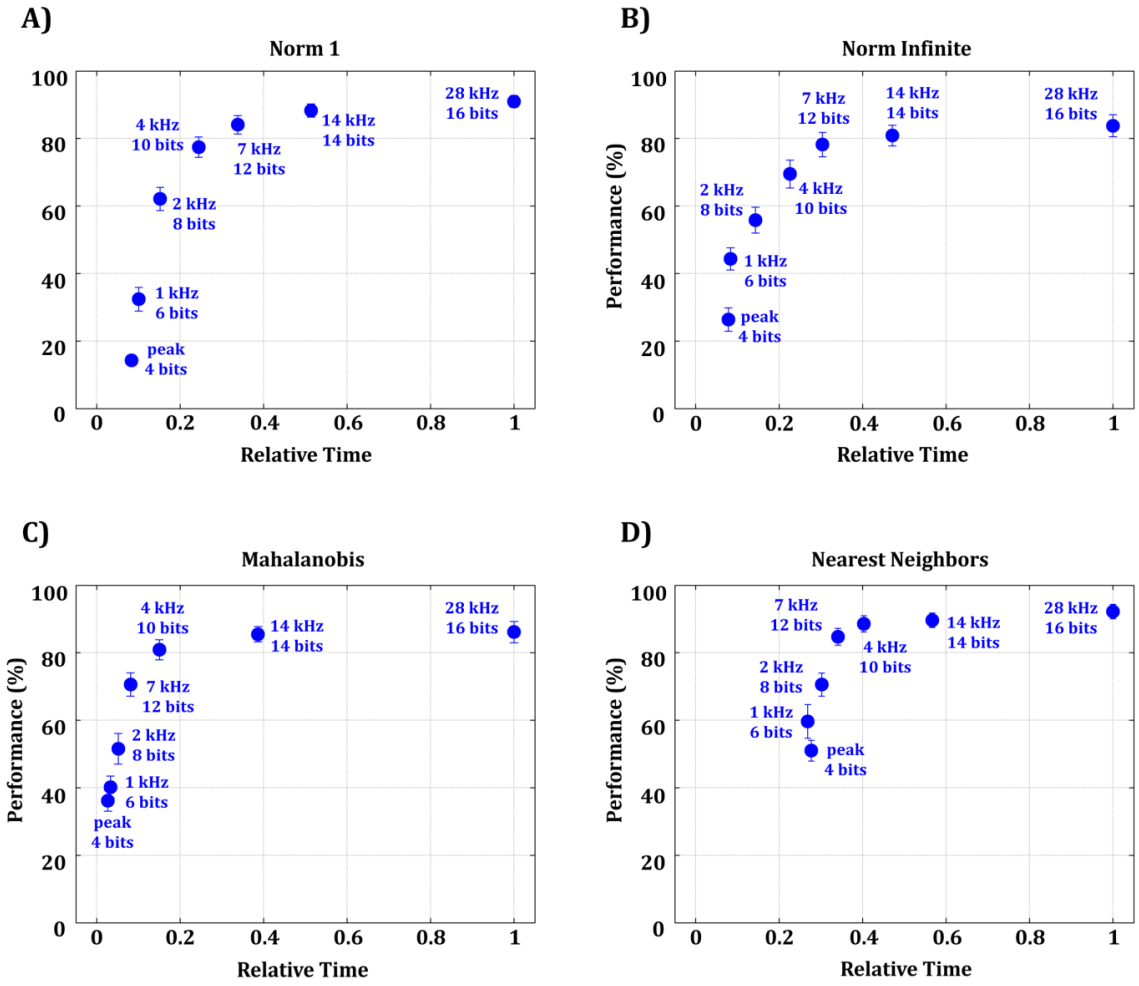


Figure A.7.: Template matching performance against relative complexity at different sampling rate and resolution specifications for: (A) Norm 1, (B) Norm Infinite, (C) Mahalanobis, and (D) Nearest Neighbours. Values are normalized to the specifications with highest complexity (28 kHz and 16-bit resolution).

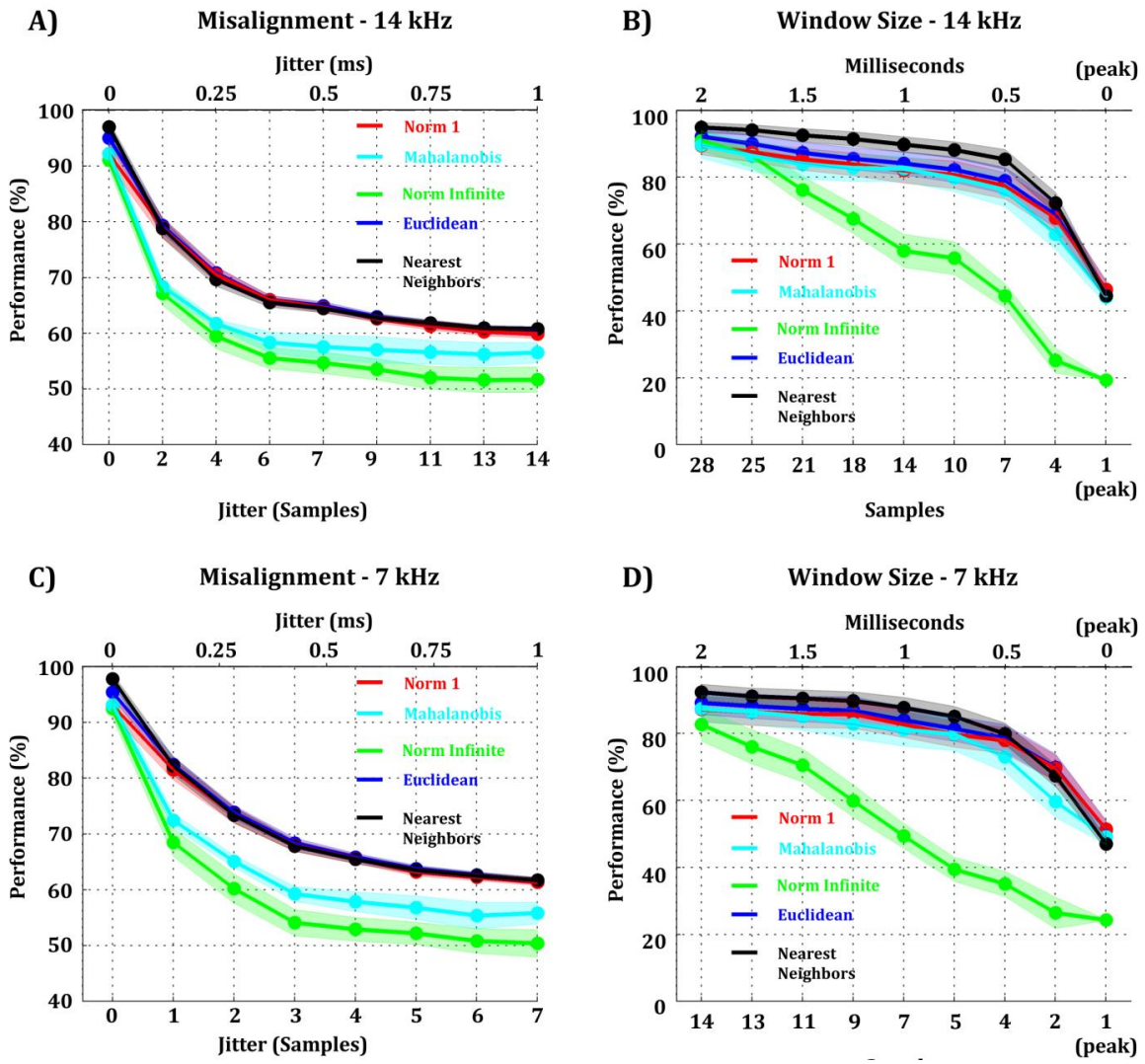


Figure A.8.: Sensitivity of template matching performance to misalignment and window size. Template matching performance of different metrics at various: (A-C) jitter values in the peak alignment of the spikes, and (B-D) spike window sizes. For (A) and (B) data is sampled at 14 kHz with 16-bit resolution, while for (C) and (D) data is sampled at 7 kHz with 16-bit resolution.

Table A.1.: Median number of misclassified spikes during spike sorting for signals with different sampling rates, signal resolutions, and SNRs. Given between the brackets are the median number of clusters detected in each case.

		28 kHz	14 kHz	7 kHz	4 kHz	2 kHz
High SNR	16 bits	6(3)	8(3)	17(3)	1153(1)	3797(0)
	14 bits	6(3)	8(3)	17(3)	1153(1)	3797(0)
	12 bits	7(3)	9(3)	19(3)	1154(1)	3797(0)
	10 bit	7(3)	11(3)	20(3)	1155(1)	3797(0)
	8 bits	1053(1)	1061(1)	1078(1)	1157(1)	3797(0)
	6 bits	3797(0)	3797(0)	3797(0)	3797(0)	3797(0)
	4 bits	3797(0)	3797(0)	3797(0)	3797(0)	3797(0)
Medium SNR	16 bits	527(2)	529(2)	543(2)	1161(1)	3797(0)
	14 bits	527(2)	530(2)	543(2)	1161(1)	3797(0)
	12 bits	527(2)	531(2)	544(2)	1162(1)	3797(0)
	10 bit	528(2)	531(2)	547(2)	1171(1)	3797(0)
	8 bits	1054(1)	1060(1)	1077(1)	1177(1)	3797(0)
	6 bits	3797(0)	3797(0)	3797(0)	3797(0)	3797(0)
	4 bits	3797(0)	3797(0)	3797(0)	3797(0)	3797(0)
Low SNR	16 bits	529 (2)	530(2)	547(2)	1163(1)	3797(0)
	14 bits	529(2)	531(2)	548(2)	1164(1)	3797(0)
	12 bits	529(2)	533(2)	550(2)	1164(1)	3797(0)
	10 bits	529(2)	537(2)	552(2)	1181(1)	3797(0)
	8 bits	1056(1)	1063(1)	1080(1)	1182(1)	3797(0)
	6 bits	3797(0)	3797(0)	3797(0)	3797(0)	3797(0)
	4 bits	3797(0)	3797(0)	3797(0)	3797(0)	3797(0)

B. Test Datasets

This section provides the detailed description of the datasets used throughout this thesis. As previously mentioned similarity of the signals play a major role in the assessed spike sorting performance. The more similar the signals are, the harder it is to separate them. Therefore, it is essential to quantify the degree of similarity. Bray-Curtis similarity index is chosen for quantifying similarity of neuron classes within each dataset [210], and it is given as:

$$S_{x,y} = 1 - \frac{\sum_{i=1}^{N} |x(i) - y(i)|}{\sum_{i=1}^{N} |x(i)| + |y(i)|} \tag{B.1}$$

where x and y are the two spike waveforms being compared and N is the number of sample points. $S_{x,y}$ is in the range (0 – 1), with 1 corresponding to identical signals.

B.1. Synthetic extracellular recordings: 1

Originally used in [77] to evaluate spike sorting with wavelets and superparamagnetic clustering, these data sets were constructed by randomly selecting spike signals from a database of over 594 waveforms recorded from the neocortex and basal ganglia in humans. There are 4 datasets each containing 3 single units at varying SNR levels. Each of these datasets are simulated at four different SNR levels bringing the total number of datasets to 24.

The different SNRs have been obtained by superimposing attenuated spike waveforms onto the signal, in order to simulate the background activity of the distant neurons.The amplitudes and timing of both the spike and noise signals have been randomly distributed within present ranges, such as to achieve the desired SNR and firing rate (with a target of 50 spikes/sec). The standard deviations of the added noise are 0.05, 0.01, 0.15 and 0.2. These datasets can be found at [105]

In addition, two more datasets (each containing 3 single units) have been simulated at the same noise levels using the synthetic recording simulator proposed by [144]. The simulator is based on the methods described above. Fig. B.1 illustrates the mean ('spikes') of each dataset together with corresponding similarity measures.

B.2. Synthetic extracellular recordings: 2

The realistic extracellular recordings were simulated by modelling the contributions of the local field potentials (LFPs), background noise, multi-unit activity and single-unit activity. The LFPs and the background noise were simulated using surrogates of one real extracellular recording

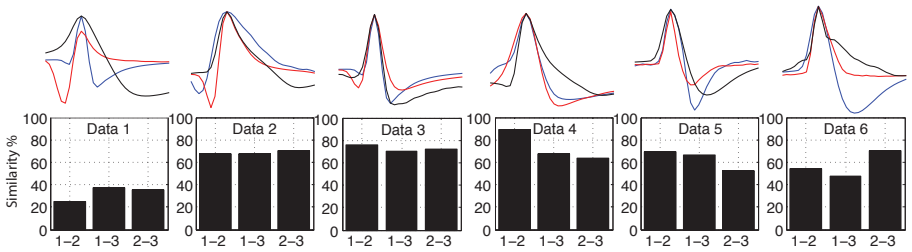


Figure B.1.: Mean spike profiles for the 6 datasets with corresponding Bray-Curtis similarity measures applied between each neuron cluster within each dataset.

from the human medial temporal lobe (MTL). The patient, with pharmacologically intractable epilepsy, was implanted with intracranial electrodes for clinical reasons [183]. To record single neuron activity, the intracranial probe had a total of 9 micro-wires with 8 active recording channels and 1 reference. Following amplification, the differential signals from the micro-wires were sampled at 28 kHz with 16-bit resolution (signal input range of $\pm$1mV). 40 μm-sized electrodes were used, and the signal bandwidth was set to 0.1-9000 Hz. The recordings were performed with a Neuralynx "Cheetah-32" system. "Lynx-8" amplifiers, which work in conjunction with "Cheetah-32", have an input noise of 15 μV_{pp} and an output noise of 10 mV_{pp}. (For the full list of specifications about the Lynx-8 amplifier, please refer to http://neuralynx.com/manuals/Lynx-8 Manual.pdf.)

The channel used in surrogate creation did not contain any single or multi-unit activity, but exhibited the same amplitude and power spectrum characteristics of neighbouring channels with both types of neural activity. The surrogates were constructed following the implementation proposed by [211], which uses a constrained randomisation to preserve both the power spectrum and amplitude distribution. Each simulation was built using a different surrogate that was created by applying the Fourier transform, shuffling the phases, and then applying the inverse transform.

Synthetic multi-unit and single-unit activity were added to the LFP and background noise following the implementation described in [157]. The simulations were created using a database of 594 different average spike shapes recorded from the monkey neocortex and basal ganglia. Multi-unit activity was generated by mixing the activity of the 594 spike waveforms in the database. The multi-unit amplitude was uniformly distributed between 20 μV and 40 μV, and the firing rate was set at 20 Hz. Single-unit activity was created by adding spike shapes with varying amplitudes to the background noise and LFP. In order to avoid introducing edge artefacts, the spike waveforms (92 samples long) had a smooth decay to zero.

Spike shapes of varying amplitudes were superimposed on the background noise and LFP to generate the SUA. Single-unit spike amplitudes were set to 50 μV (low SNR), 75 μV (medium SNR), and 100 μV (high SNR), in order to create datasets with different SNRs. Each spike train followed a Poisson process, with a mean firing rate of 5 Hz. Spikes that fell within a 2 ms window of each other were removed so as to introduce a refractory period and delete overlapping spikes. The mean waveforms of the spikes for each cluster in each dataset, and their similarity

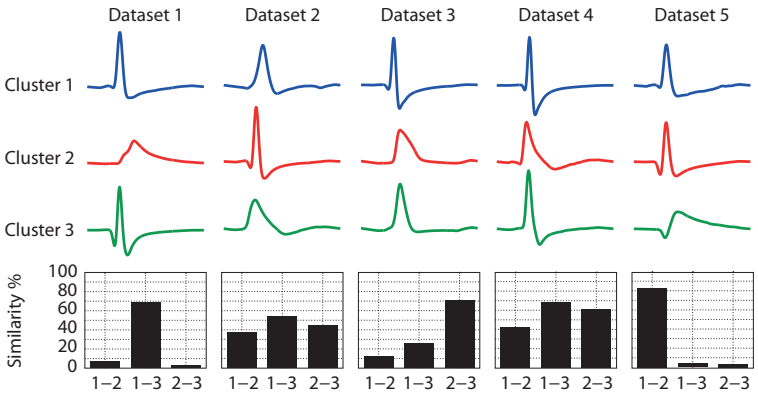


Figure B.2.: Mean spike profiles for the 5 datasets with corresponding Bray-Curtis similarity measures applied between each neuron cluster within each dataset.

measures are presented in Figure B.2.

C. Matlab Code for Automated Test Platform for Real-time Spike Sort Implementation

```
1  %<>>>>>>>>>>>>>Automated MCU Test<<<<<<<<<<<<<
2  %Written by: Deren Y. Barsakcioglu
3  %Created:  01/10/2014
4  %Last Modified: 17/10/2014
5
6  %Import java class to control the mouse pointer
7  import java.awt.Robot;
8  import java.awt.event.*;
9  mouse = Robot;
10 %ChannelCount
11 ChNo=16; %Choose from 1-32
12 DataLength = 1000000;%0;
13 %Quantization Level
14 B=8;
15 Process = str2num(input('Choose action: (1)Generate Data (2)Run Data &
       Process Result','s'));
16 %Configure Serial Communication Settings
17 %COM4:
18 com4 = serial('COM4','BaudRate',6e6,'DataBits',8,'StopBits',1,'Parity','
       none','FlowControl','none','Terminator','');
19 com4.OutputBufferSize = 2000000;
20 com4.InputBufferSize = 100000;
21 %com4.TimeOut = 500; %extend time-out to 200 seconds since data is quite
       long
22 fopen(com4); %Open serial connection
23
24 %% Enable communication with Lecroy
25 errflag = 0;
26 % Create a TCPIP object.
27 interfaceObj = instrfind('Type', 'tcpip', 'RemoteHost', '10.0.0.206', '
       RemotePort', 1861, 'Tag', '');
28 % Create the TCPIP object if it does not exist
29 % otherwise use the object that was found.
30 if isempty(interfaceObj)
```

```
31      interfaceObj = tcpip('10.0.0.206', 1861);
32  else
33      fclose(interfaceObj);
34      interfaceObj = interfaceObj(1);
35  end
36  % Create a device object.
37  lecroyObj = icdevice('lecroy_basic_driver.mdd', interfaceObj);
38  % Input buffersize needs to be edited from the driver
39  % Connect device object to hardware.
40  connect(lecroyObj);
41  %Set the time base
42  lecroy_Delay = -2.45;
43  set(lecroyObj.Acquisition(1), 'Timebase', 500e-3);
44  set(lecroyObj.Acquisition(1), 'Delay', lecroy_Delay);
45  for iteration=1:1%:10
46      mkdir(strcat('Iteration_',num2str(iteration)))
47
48  %Read in the file names for the spike train.
49  fid2 = fopen('BW_Experiment.txt');
50  var_file2=textscan(fid2,'%s');
51  fid4 = fopen('Names_BW.txt');
52  var_file4=textscan(fid4,'%s');
53  fclose(fid2);
54  fclose(fid4);
55
56  %Cluster number info.
57  Cluster_no = 3;
58  %Noise test paramters
59  sa = 250e-6; %assumed amplitude for the normalized signal in volts
60  noiseLevels = [0 2.5 5 7.5 10 15 20 25 50]*1e-6; %noise levels
61
62  %Statistics on which samples are being picked.
63  SampleHistogram = zeros(1,length(noiseLevels)*length(var_file2{1,1})*
        Cluster_no*(Cluster_no-1)  );
64  SHistCount = 1;
65  MaxPt = 7;%10;
66  WaveFeat = zeros(Cluster_no,3);
67  WaveFeatTime = zeros(Cluster_no,3);
68  SDFeat = zeros(Cluster_no,3);
69  SDFeatTime = zeros(Cluster_no,3);
70  DetStat = zeros(16,length(noiseLevels));
71  SortStat = zeros(16,length(noiseLevels));
72  TpStat = zeros(16,length(noiseLevels));
73  FpStat = zeros(16,length(noiseLevels));
74
```

```
75  for noise_loop=1:1:length(noiseLevels)
76
77  %****** REMOVED SECTION OF THE CODE DUE TO SPACE CONCERNS: Declaration of
        variables*******
78
79  for data_loop=1:1:length(var_file2{1,1}) %FOR LOOP: Open different data
        files.
80  s = var_file2{1,1}{data_loop,1};
81  data_file_name = var_file4{1,1}{data_loop,1};
82  %Load the data set structure to be processed
83  DCELL = load(s);
84  cd('D:\Deren\uC_TestSetup\')
85  %Prepare output file
86  Rows = length(filter_type) * length(filter_order) * length(cut_off_f_lp) *
        length(cut_off_f_hp)+4;
87  Columns = 33; %(5 methods,each 3 temp creation and two metrics) 5*6+1->for
        the spacing
88  OUTPUT_FILE = cell(Rows,Columns);
89  %Name of the data set
90  OUTPUT_FILE{1,1} = data_file_name;
91
92  %****** REMOVED SECTION OF THE CODE DUE TO SPACE CONCERNS: Output file
        formatting*******
93
94  %END OF PREPARING OUTPUT TABLE
95  new_data = resample(DCELL.data,2,3);
96  new_spkTime = DCELL.spike_times{1,1};
97  temp_spkTime = ((new_spkTime - 1) .* 2) + 1; %New positions after
        upsampling
98  new_spkTime = floor(((temp_spkTime - 1))./3)+1; %New positions after
        downsampling
99
100 %Add Noise
101 whiteGaussianNoise = randn(1,length(new_data));
102 generatedNoise = (noiseLevels(1,noise_loop)/sa)*whiteGaussianNoise;
103 new_data = new_data + generatedNoise;
104
105 %Determine the window size to be taken for each spike. Original
106 %spikes 64 samples.
107 sample_window = 43;
108
109 %Plot frequency spectrum
110 %plotContent( new_data,Fs )
111 %Set up a loop to do different filtering techniqes.
112 for filter_index=1:length(filter_type)
```

```
113 for hp_filter_index=1:length(cut_off_f_hp)
114 for frequency_index=1:length(cut_off_f_lp)
115 for filter_order_index = 1:length(filter_order)
116 %Check for HP
117 if cut_off_f_hp(hp_filter_index)~=0 %hp filter signal before low pass
118         %****** REMOVED SECTION OF THE CODE DUE TO SPACE CONCERNS*******
119 else %no need for HP
120         %****** REMOVED SECTION OF THE CODE DUE TO SPACE CONCERNS*******
121 end
122 cd('D:\Deren\uC_TestSetup\')
123 %QUANTIZE SIGNALS TO B BITS
124 MaxVal = max(filtered_spike_train);
125 MinVal = min(filtered_spike_train);
126 LSB = (MaxVal - MinVal)/(2^B-1);
127 filtered_spike_train = sign(filtered_spike_train).* floor(abs(
        filtered_spike_train)./LSB + 0.5);
128 %Perform spike extraction.
129 [ signals class_info Threshold ] = return_spikes( filtered_spike_train,
        new_spkTime, DCELL.spike_class,filtering_option,sample_window );
130 MisalignTest_Size = size(signals);
131 MisalignTest_Size(1,1)
132 [aligned_spikes alignable max_point] = new_alignment( signals,time_window,
        fs,BaseFreq,class_info,data_loop,Threshold );
133
134 %Combine the signal and its 1st,2nd and 3rd
135 %derivatives into one signal
136 FD_alSpk =  diff(aligned_spikes,1,2); %diff(X,n,dim)
137 SD_alSpk =  diff(aligned_spikes,2,2); %diff(X,n,dim)
138 TD_alSpk =  diff(aligned_spikes,3,2); %diff(X,n,dim)
139
140 %****** REMOVED SECTION OF THE CODE DUE TO SPACE CONCERNS:
141 %Calculating mean templates for EAP waveforms and derivatives*******
142
143 %Determine the standard deviation of noise
144 TmW = round(median(abs(filtered_spike_train)/0.6745));
145 TmSD = round(median(abs(diff(filtered_spike_train,2,2))/0.6745));
146 [ WaveFeatTemp, WaveFeatTimeTemp,SDFeatTemp,SDFeatTimeTemp ] = ExtractFeat
        ( aligned_spikes(~alignable,:), FD_alSpk(~alignable,:), SD_alSpk(~
        alignable,:), TD_alSpk(~alignable,:),MaxPt);
147 [ SampleIndicesW ] = FeatClusIndexV2( WaveFeatTimeTemp, WaveFeatTemp,
        Cluster_no, TmW,0 );
148 [ SampleIndicesSD ] = FeatClusIndexV2( SDFeatTimeTemp, SDFeatTemp,
        Cluster_no, TmW,0 );
149
150 %****** REMOVED SECTION OF THE CODE DUE TO SPACE CONCERNS:*******
```

```
151 %****** Finding local extrema and among these choosing indices which give
        largest separation*******
152
153 if totalCheckSum==0 %All spikes are detected no NaNs.
154         %Samples of interest: [ T1,2 T1,3 T2,3]
155          indexTM = [ templatesIndex(1,1) templatesIndex(1,2) templatesIndex
                (2,2)];
156         %Samples of interest: [ T1,2 T1,3 T2,3]
157          indexSD = [ SDIndex(1,1) SDIndex(1,2) SDIndex(2,2)];
158         %Only use the mid-points
159         MidIndicator = zeros(1,3); %This is important since in midpoint I
                have to know which side is above or below the midpoint
160         MidIndicatorSD = zeros(1,3); %This is important since in midpoint I
                 have to know which side is above or below the midpoint
161         %Find the ratios of standard deviations to
162         %scale and adjust threshold properly. This
163         %takes into account the distribution. NOTE THE
164         %INVERSE RELATIONSHIP!!! HIGHER THE STD THE
165         %FURTHER AWAY IT IS FROM THE MEAN
166         Sc1 =  stdTtemplates(1,templatesIndex(1,1))/(stdTemplates(1,
                templatesIndex(1,1))+stdTemplates(2,templatesIndex(1,1)));
167         Sc2 = stdTemplates(2,templatesIndex(1,1))/(stdTemplates(1,
                templatesIndex(1,1))+stdTemplates(2,templatesIndex(1,1)));
168
169         FirstMidTM = Sc2*templates(1,templatesIndex(1,1)) + Sc1*templates
                (2,templatesIndex(1,1)); %T1 vs T2
170         if templates(1,templatesIndex(1,1))>templates(2,templatesIndex(1,1)
                )
171                 MidIndicator(1,1) = 0; %T1 is higher than T2
172         else
173                 MidIndicator(1,1) = 1; %T2 is higher than T1
174         end
175
176         %****** REMOVED SECTION OF THE CODE DUE TO SPACE CONCERNS*******
177         %****** More calculations of comparison thresholds*******
178
179         ThirdMidSD = Sc2*SDtemplates(2,SDIndex(2,2)) + Sc1*SDtemplates(3,
                SDIndex(2,2)); %T2 vs T3
180         if SDtemplates(2,SDIndex(2,2))>SDtemplates(3,SDIndex(2,2))
181                 MidIndicatorSD(1,3) = 0; %T2 is higher than T3
182         else
183                 MidIndicatorSD(1,3) = 1; %T3 is higher than T2
184         end
185
186         midPoint =  [FirstMidTM SecondMidTM ThirdMidTM ];
```

```
187         midPointSD = [  FirstMidSD SecondMidSD ThirdMidSD];
188         %Sort according to extraction indices
189         [Y I] = sort(indexTM);
190         [Y2 I2] = sort(indexSD);
191         Thresholds = midPoint(I);
192         Thresholds_SD = midPointSD(I2);
193
194         %NOTE: Convention I have here for mid-indicator
195         %is not the same as what I have implemented in
196         %C
197         ThreshInd = ~MidIndicator(I);
198         ThreshInd_SD = ~MidIndicatorSD(I2);
199
200         DimensionArray = 0:1:(Cluster_no*(Cluster_no-1)/2)-1;
201         IndDim = DimensionArray(I);
202         IndDimSD = DimensionArray(I2);
203
204         if (Process==1) %Generate Data
                                                                    %
            Write the spike data samples
205             filtered_spike_train = int8(filtered_spike_train);
206             filtered_spike_trainX =  zeros(1,length(
                    filtered_spike_train)*ChNo);
207
208             countCh=1;
209             for ChCnt=1:ChNo:length(filtered_spike_trainX)
210                 filtered_spike_trainX(1,ChCnt:ChCnt+ChNo-1)=
                        filtered_spike_train(1,countCh);
211                 countCh=countCh+1;
212             end
213             %Write text file of SPI format
214
215             formatSPI( filtered_spike_trainX(1:DataLength),8,strcat('
                    WaveformGenerator','_',num2str(data_loop),'_',num2str(
                    noise_loop),'.txt') )
216         elseif (Process==2) %Send training & Run Data Through & Assess the
                result
217             %Set the LeCroy trigger
218             if (~strcmp(get(lecroyObj.Trigger(1), 'Mode'),'single'))
219                 set(lecroyObj.Trigger(1), 'Mode', 'single');
220             end
221
222             %Replace the data source file in the
223             %configuration files of WaveXpress
224             %Generator
```

```
225                 delete('DerenTest.cfg');
226                 fin = fopen('deren.cfg','r');
227                 fout = fopen('DerenTest.cfg','w');
228                 NametoReplace = '@file txt_rawbin "D:\Deren\uC_TestSetup\"'
                        ;
229                 ReplaceWith = strcat('@file txt_rawbin "D:\Deren\
                        uC_TestSetup\',strcat('WaveformGenerator','_',num2str(
                        data_loop),'_',num2str(noise_loop),'.txt') );
230
231                 while ~feof(fin)
232                     s = fgetl(fin);
233                     s = strrep(s, NametoReplace, ReplaceWith);
234                     fprintf(fout,'%s\n',s);
235                     disp(s)
236                 end
237
238                 fclose(fin);
239                 fclose(fout);
240
241                 %Load the updated configuration file to
242                 %WaveXpress GUI
243                 mouse.mouseMove(1060, 205); %move pointer to where load
                        button is
244                 mouse.mousePress(InputEvent.BUTTON1_MASK);
245                 mouse.mouseRelease(InputEvent.BUTTON1_MASK);
246                 pause on
247                 pause(120) %Wait for 2 mins for data to load. I would
248                                 %have preferred to listen the
249                                 %busy status of the mouse and
250                                 %continue upon that condition
251                                 %but couldn't figure it out.
252
253                 %Send Training Data
254                 filler = [0 0 0];
255                 TrainData = round([3 6 (Y-1) filler (Y2+2-1) filler
                        Thresholds filler Thresholds_SD filler ThreshInd filler
                        ThreshInd_SD filler IndDim filler IndDimSD filler TmW
                        TmSD]);
256
257                 %Convert to 8 bit signed format
258                 TrainData = int8(TrainData);
259                 TrainDataX = zeros(1,length(TrainData)*ChNo);
260                 countCh=1;
261                 for ChCnt=1:ChNo:length(TrainDataX)
262                         TrainDataX(1,ChCnt:ChCnt+ChNo-1)=TrainData(1,
```

```
                countCh);
263             countCh=countCh+1;
264         end
265
266         %Append minmax
267         TrainDataX = [TrainDataX 1];
268         %Write numerical data to serial. Since
269         %train function can't handle the continous
270         %streams of 6MHz baud and above give it
271         %some time.
272         pause on
273         for TwrInd=1:1:length(TrainDataX)
274             fwrite(com4,TrainDataX(1,TwrInd),'int8');
275             pause(0.0001)
276         end
277
278         while ~(strcmp(get(com4,'TransferStatus'),'idle'))
279             %Wait until transmission is complete.
280         end
281
282         %Run the data through Waveform Generator
283         %Xpress
284         %input('Press Enter when the data is processed','s');
285
286         mouse.mouseMove(1120, 315); %move pointer to where start
                button is
287         mouse.mousePress(InputEvent.BUTTON1_MASK);
288         mouse.mouseRelease(InputEvent.BUTTON1_MASK);
289         pause on
290         pause(60) %Pause for a minute to allow all acquisition to
                settle
291
292         %Read the sorting results from serial input
293         Result = fread(com4,100000,'uint8');%Sanity check
294
295         c_start=clock;
296         while(1)
297         try
298             if (strcmp(get(lecroyObj.Trigger(1), 'Mode'),'stop'
                    ))
299                 break;
300             end
301             % Capture Oscilloscope Time Out! 1s.
302             if etime(clock,c_start) > 1
303                 ME = MException('Lecroy:TimeOut', ...
```

```
304                                 'Lecroy has not capture anything in 1s');
305                                 throw(ME);
306                         end
307                 catch err
308                         % manually set to stop, the charge_cal should catch
                                the error
309                         % and rerun for this target charge set.
310 %               set(lecroyObj.Trigger(1), 'Mode', 'stop');
311                         %% Diplay error as usual
312                         display(sprintf('Error Message: %s\n',err.
                                identifier,err.message));
313                         errflag = 1;
314                         break;
315                 end
316                 end
317
318                 if (errflag == 1)
319                         continue;
320                 end;
321                 %Query LeCroy for completion of measurement
322                 %and read in the data
323                 groupObj = get (lecroyObj, 'Waveform');
324                 groupObj = groupObj(1);
325                 [Y2,X2,YUNIT1,XUNIT1,HEADER1] = invoke(groupObj, '
                        readwaveform', 'channel2'); %Output Toggle
326                 [Y3,X3,YUNIT2,XUNIT2,HEADER2] = invoke(groupObj, '
                        readwaveform', 'channel3'); %Spi Select
327                 [Y4,X4,YUNIT3,XUNIT3,HEADER3] = invoke(groupObj, '
                        readwaveform', 'channel4'); %Sample Toggle
328                 OutputInd = [X2' Y2'];
329                 SPISelect = [X3' Y3'];
330                 SampInd = [X4' Y4'];
331
332                 %return timestamps
333                 [ timeStamps] = decodeOutputInd(OutputInd,SPISelect,SampInd
                        ,fs,ChNo);
334                 %Check if number of timestamps & Results
335                 %match
336                 if (length(Result)~=length(timeStamps))
337                         error('TimeStamps and Results do not match')
338                 end
339                 ResultBin = dec2bin(Result,8);
340                 ChannelInfo=bin2dec(ResultBin(:,1:5));
341                 SortInfo= bin2dec(ResultBin(:,6:8));
342                 %Extract sorted results
```

```
343                     sortedResults = zeros(ChNo,size(Result,1));
344                     sortedTimeStamps = zeros(ChNo,size(Result,1));
345                     spikeCount=zeros(ChNo,1);
346                     for resIndex=1:1:ChNo
347                             sortedResults(resIndex,1:length(SortInfo(
                                    ChannelInfo==(resIndex-1)))) = SortInfo(
                                    ChannelInfo==(resIndex-1));
348                             sortedTimeStamps(resIndex,1:length(SortInfo(
                                    ChannelInfo==(resIndex-1))))=timeStamps(
                                    ChannelInfo==(resIndex-1));
349                             spikeCount(resIndex,1) = sum(ChannelInfo==(resIndex
                                    -1));
350                     end
351                     %Data Check: Which section of the data is being
352                     %input to test.
353                      timeWindow=(DataLength/ChNo)/fs ;
354                      newSpkTimeSec = new_spkTime/fs;
355                     OriginalTime = newSpkTimeSec(newSpkTimeSec<timeWindow);
356                     DetResult=zeros(1,ChNo);
357                     SortResult=zeros(1,ChNo);
358                     TpResult=zeros(1,ChNo);
359                     FpResult = zeros(1,ChNo);
360
361                     for chLoop=1:1:ChNo
362                             [DetResult(1,chLoop) SortResult(1,chLoop) TpResult
                                    (1,chLoop) FpResult(1,chLoop)] =
                                    DetSortCompareV2( OriginalTime, sortedTimeStamps
                                    (chLoop,1:spikeCount(chLoop)), DCELL.spike_class
                                    ,sortedResults(chLoop,1:spikeCount(chLoop)),2.7e
                                    -3 );
363                     end
364
365                     DetStat(data_loop,noise_loop) = mean(DetResult);
366                     SortStat(data_loop,noise_loop) = mean(SortResult);
367                     TpStat(data_loop,noise_loop) = mean(TpResult);
368                     FpStat(data_loop,noise_loop) = mean(FpResult);
369             end
370     else
371             error('Some spikes could not be differentiated')
372     end
373     end
374     end
375     end
376     end
377             %Record the statistics
```

```
378 end %END - different data files
379 end %End of sweeping noise levels
380 end
381 save('DetStat.mat', 'DetStat')
382 save('SortStat.mat', 'SortStat')
383 %% Close communication with Lecroy
384 disconnect(lecroyObj);
385 delete([lecroyObj interfaceObj]);
386 %Close the communication
387 fclose(com4);
388 delete(com4);
389 clear com4
390 emailme(sprintf('Acquisition done at %s\n',num2str(fix(clock))));
```

D. C-Code for WDF implemented on KL43Z for tests involving RHD2216

```
1  //Author: Deren Barsakcioglu
2  //Description: Microcontroller (FRDM KL43) Code written for experiments
       with RHD2216 Analogue Front-End
3  //Note:  Please note that several sections of the code (such as variable
       declarations,
4  //RHD2216 initialisation, functions) are removed from appendix due to space
        concerns. Full code is available upon
5  //request from the author.
6
7  #include "MKL43Z4.h"                    // Device header
8  #include <stdio.h>
9  #include <stdlib.h>
10 #include <stdbool.h>
11 /*******Global Integers*******/
12 uint8_t ChNoS=0;
13 uint16_t testNumber;
14 uint16_t timeVal;
15 volatile int TrainDone;
16 volatile int TrainReceive;
17 int TrainStart ;
18 /*******End of Global Integers*******/
19 /*******FIFO DESCRIPTON*******/
20 #define FIFO_size 8192 //8192 512
21 #define FIFO_mask (FIFO_size-1)
22 #define ChSz 16   //number of channels to be used 16
23 #define ChSzMask (ChSz-1)     //number of channels to be used
24 #define SpLn 33 // 43 64 spike length.
25 #define MaxPt 7//10 point where the maximum point of the spike is
26 #define NeuronCapacity 4
27 #define MemAlloc (NeuronCapacity*NeuronCapacity - NeuronCapacity)/2
28 //BufferDeclaration
29 volatile uint16_t wrInd; //for reading via interrupt SPI
30 volatile uint16_t rdInd;
31 volatile int16_t data[FIFO_size];
32 volatile uint16_t TimeDebug;
33 int p; //pointer to initial address
```

```
34 /*******END FIFO DESCRIPTON*******/
35 /*******Spike Sort Variables*******/
36 //Spike sort variables are not shown in appendix due to space concerns.
       Full code is available upon request.
37 /*******END Spike Sort Variables*******/
38 /******* Function declarations *******/
39 void uart_init(void);
40 void DMA_init(void);
41 void DMA_init2(void);
42 void uart_send(uint8_t spiMsg);
43 //void spi_init_slave(void);
44 void spi_init_master(void);
45 void spi_init_master_DMA(uint8_t BaudRate);
46 void spi_send(uint16_t spiMsg);
47 void lptmr_init_sort(void); //Setting timer module
48 //void spikeSort(int8_t sample,int8_t ChNo);
49 void ChTrain(int8_t sample);
50 void vote1(int8_t index, int8_t ChNo);
51 void vote2(int8_t index, int8_t ChNo);
52 /*******END Function declarations*******/
53 /*-------------------------------------------------------------------------

54   main
55  *-------------------------------------------------------------------------*/

56 int main(void) {
57         /*Initialise variables*/
58         int StatusField;
59         TrainDone=0;
60         TrainReceive=0;
61         rdInd = 0;
62         wrInd = 0;
63         p = (uint32_t) &data; //initial address
64         /*Setup SPI*/
65         spi_init_master();
66         /*Setup UART*/
67         uart_init(); //Initialize UART
68         timeVal=0;//reset time value
69         lptmr_init_sort(); // Resolution: 0.24414ms
70         /*************RHD2216 Initial Setup*************/
71         //This section of the code have been excluded from Appendix due to
               space concerns. The full code is available upon request from the
                author.
72         /*************End of RHD2216 Initial Setup*************/
73         ConvertADC = 0x0100;
```

```
74          Count=0;
75          /*Setup DMA: To query AFE for samples and stream out samples*/
76          DMA_init();
77          spi_init_master_DMA(0x23); //0x22 0x23
78          while (!(SPI1->S & SPI_S_SPTEF_MASK)){  //Wait until SPRF is ready
79          }
80          spi_send(0x0000) ;  //Make sure you discard the first two samples
                in MATLAB since ADC result appears at the third clock period.
81          SPI1->C2 |= (SPI_C2_RXDMAE_MASK); //Only Enable DMA receive, 16-bit
                mode
82          NVIC_EnableIRQ(DMA2_IRQn);  //Sample Stream
83          NVIC_EnableIRQ(LPUART0_IRQn); // Train Receive
84          /*Stream data until training starts, then disable sampling and
                streaming*/
85          while(TrainReceive==0 ){ //Manual data request. Enables the
                efficient DMA later during spike sorting
86                  //Request Intan to send samples
87                  while (!(SPI1->S & SPI_S_SPTEF_MASK)){  //Wait until SPRF
                        is ready
88                  }
89                  spi_send(ConvertADC) ;
90                  /*Adjust the channel no*/
91                  if (Count<(3500000-2)){ //937500
92                          ConvertADC = ((ConvertADC+0x0100) & 0x0300);
93                          Count++;
94                  }else if(Count<7500000-2){
95                          ConvertADC = (((ConvertADC+0x0100) & 0x0700) | (0
                                x0400));
96                          Count++;
97                  }else if(Count<11250000-2){
98                          ConvertADC = (((ConvertADC+0x0100) & 0x0B00) | (0
                                x0800));
99                          Count++;
100                 }else if(Count<(15000000-2)){
101                         ConvertADC = (((ConvertADC+0x0100) & 0x0F00) | (0
                                x0C00));
102                         Count++;
103                 }else{
104                         ConvertADC = (((ConvertADC+0x0100) & 0x0300));
105                         Count=-1;
106                 }
107         }
108         //DMAMUX0->CHCFG[0] = 0;  //Disable Sampling
109         DMAMUX0->CHCFG[2] = 0;  //Disable datastreaming
110         //NVIC_DisableIRQ(DMA0_IRQn);
```

```
111         NVIC_DisableIRQ(DMA2_IRQn);
112         SPI1->C1 = 0x00; //Disable SPI
113         /*Until Training is done and all registers are initialized wait*/
114         while(TrainDone==0){
115         }
116         //DMAMUX0->CHCFG[2] = 0;  //Disable Sampling
117         NVIC_DisableIRQ(LPUART0_IRQn); //Once all training samples are
                received disable
118         /*Send two initial commands. Sample result is available at the
                third instruction. So, when
119         DMA starts sending first sample to be sent over uart is Channel 0
                sample.*/
120         ConvertADC = 0x3F00;
121         DMA_init2();
122         spi_init_master_DMA(0x21);
123         while (!(SPI1->S & SPI_S_SPTEF_MASK)){  //Wait until SPRF is ready
124         }
125         spi_send(0x0000) ;
126         SPI1->C2 |= (SPI_C2_RXDMAE_MASK|SPI_C2_TXDMAE_MASK); //Enable DMA
                receive and transmit, 16-bit mode
127         NVIC_EnableIRQ(DMA0_IRQn);       //Convert(X) commands to Front End
128         NVIC_EnableIRQ(DMA1_IRQn);
129         NVIC_EnableIRQ(DMA3_IRQn);  //Sample Store to Buffer
130         LPTMR0->CSR &= ~LPTMR_CSR_TEN_MASK;   //Turn off LPT and start
                counting
131         LPTMR0->CSR |= LPTMR_CSR_TEN_MASK;   //Turn on LPT and start
                counting   Reset counter
132         /*Due to the way DMA and SPI is set first two samples will be void.
                 Therefore wait until wrInd>1. You know for sure that 3rd  ie.
                Buffer[2] the first valid
133         response corresponding to Channel is stored*/
134         while (wrInd<2){
135         }
136         rdInd=2;//pffset to start from valid channel 0
137
138
139         /************Spike Sorting Code************/
140         /*Loop Forever: Detection and Sorting*/
141   while(1){
142 /*Check buffer: If empty don't do anything*/
143 if (!(rdInd == wrInd)){  // if (!(UartFifo.rdInd == (DMA0->DMA[3].DAR - p))
      ) or  (!(UartFifo.rdInd == UartFifo.wrInd))
144 //Check if all votes are calculated. If not, continue det, extr and voting.
      Else, go to spike sorting.
145 if(AllFeat[ChNoS]==true){
```

```
146         //If count has reached the last sample, then sort the spikes. This
                is being checked so that accidental spike
147         //detection due to high repolarization peak is prevented. [FOR
                IMPROVED SPIKE DETECTION]
148         if(CntS[ChNoS]==SpLn){
149                 //do sorting
150                 CntN=0;
151                 //all features extracted evaluate
152                 while (CntN<N[ChNoS]){
153                         //Check for maximum votes
154                         if (Vote[ChNoS][CntN] > MaxVote){
155                                 MaxVote = Vote[ChNoS][CntN];
156                                 MaxVoteInd = CntN;
157                                 MvFlag = false;
158                         }else if(Vote[ChNoS][CntN] == MaxVote){
159                                 MvFlag = true;
160                         }
161                         CntN++;
162                 }
163                 if (MvFlag==true){ // multiple votes, so unresolved
164                         //When processing of the particular spike has
                                finished, reset variables.
165                         MaxMinVal[ChNoS] = DetThrsh[ChNoS];
166                         MaxVote = 0;
167                         featCnt[ChNoS]=0;
168                         MaxVoteInd=0;
169                         FlDet[ChNoS] = false;
170                         AllFeat[ChNoS]=false;
171                         LPTMR0->CNR = 0x0000; //Write to CNR with any value
                                to synchronize
172                         uart_send((LPTMR0->CNR & 0x00FF)); //Send time info
173                         uart_send(ChNoS<<3 | 0x80);  //Ch+Sort has a
                                leading 1
174                         LPTMR0->CSR &= ~LPTMR_CSR_TEN_MASK;    //Turn off
                                LPT and start counting
175                         LPTMR0->CSR |= LPTMR_CSR_TEN_MASK;    //Turn on LPT
                                and start counting   Reset counter
176                         goto RETURN1;
177                 }else{
178                         //When processing of the particular spike has
                                finished, reset variables.
179                         MaxMinVal[ChNoS] = DetThrsh[ChNoS];
180                         MaxVote = 0;
181                         featCnt[ChNoS]=0;
182                         FlDet[ChNoS] = false;
```

```
183                             AllFeat[ChNoS]=false;
184                             //Send data over UART0 via DMA
185                             LPTMR0->CNR = 0x0000; //Write to CNR with any value
                                    to synchronize
186                             uart_send((LPTMR0->CNR & 0x00FF)); //Send time info
187                             uart_send((MaxVoteInd+1)|(ChNoS<<3) | 0x80); //Ch+
                                    Sort has a leading 1
188                             if (LPTMR0->CNR>TimeDebug){
189                                     TimeDebug=LPTMR0->CNR;
190                             }
191                             LPTMR0->CSR &= ~LPTMR_CSR_TEN_MASK;    //Turn off
                                    LPT and start counting
192                             LPTMR0->CSR |= LPTMR_CSR_TEN_MASK;    //Turn on LPT
                                    and start counting   Reset counter
193                             goto RETURN1;
194                     }
195             }else{
196                     //Still not the end of spike
197                     CntS[ChNoS]++;
198                     goto RETURN1;
199             }
200     }
201     //detection and finding max/min
202     if (data[rdInd] > MaxMinVal[ChNoS]){
203             FlDet[ChNoS] = true;
204             MaxMinVal[ChNoS] = data[rdInd];
205             //reset all counters and votes
206             CntS[ChNoS]=0;Cntl[ChNoS]=0;Cntl_SD[ChNoS]=0;
207             featCnt[ChNoS]=0;
208             CntVW[ChNoS]=0;
209             CntVS[ChNoS]=0;
210             Vote[ChNoS][0] = 0;
211             Vote[ChNoS][1] = 0;
212             Vote[ChNoS][2] = 0;
213             Vote[ChNoS][3] = 0;
214     }
215     //If spike is not detected, don't even bother comparing it.
216     if(FlDet[ChNoS]==true){
217             ExtFlg = false;
218             // Check if the counter matches the indices of the original
                    waveform
219             if(CntS[ChNoS]==ExtrIndex[ChNoS][Cntl[ChNoS]]){ //extraction
220                     //set extraction flag
221                     ExtFlg = true;
222                     //Get the sample of interest.
```

```
223                     BufOut = data[(rdInd-(MaxPt-1)*ChSz) & FIFO_mask];
224 LOOP1:             ExtSample[ChNoS][Cntl[ChNoS]] = BufOut;
225                     Cntl[ChNoS]++;
226                     if(CntS[ChNoS]==ExtrIndex[ChNoS][Cntl[ChNoS]]){
227                             goto LOOP1;
228                     }
229             }
230             // Check if the counter matches the indices of the original
                    waveform
231             if(CntS[ChNoS] == ExtrIndex_SD[ChNoS][Cntl_SD[ChNoS]]){
232                     //set extraction flag
233                     ExtFlg = true;
234                     //Calculate Second Derivative of the Sample
235                     BufOut = data[(rdInd-(MaxPt-1)*ChSz) & FIFO_mask] - 2*data
                            [(rdInd-MaxPt*ChSz) & FIFO_mask]+data[(rdInd-(MaxPt+1)*
                            ChSz) & FIFO_mask];
236 LOOP2:             ExtSampleSD[ChNoS][Cntl_SD[ChNoS]] = BufOut;
237                     Cntl_SD[ChNoS]++;
238                     if(CntS[ChNoS] == ExtrIndex_SD[ChNoS][Cntl_SD[ChNoS]]){
239                             goto LOOP2;
240                     }
241             }
242             //no match check if voting can be done
243             if (ExtFlg==false){ //If no extraction is done, then proceed with
                    voting.
244                     //Perform Comparison and Voting: Original Waveform.
245                     if (CntVW[ChNoS]<Cntl[ChNoS]){ //Check if any samples have
                            been extracted. If Cntl is larger than CntVW, this
                            indicates unprocessed new samples.
246                             if (ExtSample[ChNoS][CntVW[ChNoS]]>Thresholds[ChNoS
                                    ][CntVW[ChNoS]]){
247                                     //Determine weight
248                                     if (ExtSample[ChNoS][CntVW[ChNoS]] <= (
                                            Thresholds[ChNoS][CntVW[ChNoS]]+stdDev[
                                            ChNoS]) ) {
249                                             Weight=1;
250                                     }else{
251                                             Weight=2;
252                                     }
253                                     if (ThreshInd[ChNoS][CntVW[ChNoS]]== true){
254                                             vote1(IndDim[ChNoS][CntVW[ChNoS]],
                                                    ChNoS);
255                                     }else{
256                                             vote2(IndDim[ChNoS][CntVW[ChNoS]],
                                                    ChNoS);
```

```
257                         }
258                     }else if (ExtSample[ChNoS][CntVW[ChNoS]]<Thresholds
                            [ChNoS][CntVW[ChNoS]]){
259                         if (ExtSample[ChNoS][CntVW[ChNoS]] >=(
                                Thresholds[ChNoS][CntVW[ChNoS]]-stdDev[
                                ChNoS])){
260                             Weight=1;
261                         }else{
262                             Weight=2;
263                         }
264                         if (ThreshInd[ChNoS][CntVW[ChNoS]]== true){
265                             vote2(IndDim[ChNoS][CntVW[ChNoS]],
                                    ChNoS);
266                         }else{
267                             vote1(IndDim[ChNoS][CntVW[ChNoS]],
                                    ChNoS);
268                         }
269                     }
270                     CntVW[ChNoS]++; //increment index
271                     featCnt[ChNoS]++; //increment feat. count index.
272                     if (featCnt[ChNoS] == TM[ChNoS]){
273                         AllFeat[ChNoS]=true;
274                     }
275                 }else if (CntVS[ChNoS]<Cntl_SD[ChNoS]){//Check if any
                        samples have been extracted. If Cntl_SD is larger than
                        CntVS, this indicates unprocessed new samples.
276                     //Perform Comparison and Voting: Second Derivative.
277                     if (ExtSampleSD[ChNoS][CntVS[ChNoS]]>Thresholds_SD[
                            ChNoS][CntVS[ChNoS]]){
278                         //Determine weight
279                         if (ExtSampleSD[ChNoS][CntVS[ChNoS]] <= (
                                Thresholds_SD[ChNoS][CntVS[ChNoS]]+
                                stdDevSD[ChNoS]) ) {
280                             Weight=1;
281                         }else{
282                             Weight=2;
283                         }
284                         if (ThreshInd_SD[ChNoS][CntVS[ChNoS]]==
                                true){
285                             vote1(IndDim_SD[ChNoS][CntVS[ChNoS
                                    ]],ChNoS);
286                         }else{
287                             vote2(IndDim_SD[ChNoS][CntVS[ChNoS
                                    ]],ChNoS);
288                         }
```

```
289                             }else if (ExtSampleSD[ChNoS][CntVS[ChNoS]]<
                                    Thresholds_SD[ChNoS][CntVS[ChNoS]]){
290                                     if (ExtSampleSD[ChNoS][CntVS[ChNoS]] >=(
                                            Thresholds_SD[ChNoS][CntVS[ChNoS]]-
                                            stdDevSD[ChNoS])){
291                                             Weight=1;
292                                     }else{
293                                             Weight=2;
294                                     }
295                                     if (ThreshInd_SD[ChNoS][CntVS[ChNoS]]==
                                            true){
296                                             vote2(IndDim_SD[ChNoS][CntVS[ChNoS
                                                ]],ChNoS);
297                                     }else{
298                                             vote1(IndDim_SD[ChNoS][CntVS[ChNoS
                                                ]],ChNoS);
299                                     }
300                             }
301                             CntVS[ChNoS]++; //increment counter
302                             featCnt[ChNoS]++;//increment feat. count index.
303                             if (featCnt[ChNoS] == TM[ChNoS]){
304                                     AllFeat[ChNoS]=true;
305                             }
306                     }
307             }
308             //Increment sample index
309             CntS[ChNoS]++;
310     }
311     /*END of spikesort function*/
312     //Increment the channel number and read index
313     /*************End of Spike Sorting Code*************/
314     RETURN1:
315     rdInd = (++rdInd & FIFO_mask);//updateIndex  //UartFifo.rdInd = (++UartFifo
            .rdInd & FIFO_mask);//updateI
316     ChNoS++;
317     ChNoS = ChNoS & ChSzMask;
318     }
319     }
320     }
```

E. Publications

Parts of the presented thesis were previously published as conference and journals papers for which I am the lead or second author. The list of chapters and relevant publications are listed as follows:

Chapter 3 contains content from:

- Deren Y. Barsakcioglu, Amir Eftekhar, and Timothy G. Constandinou, "Design optimisation of front-end neural interfaces for spike sorting systems," Proc. IEEE International Symposium on Circuits and Systems, 2013, pp. 2501–2504.
- Deren Y. Barsakcioglu, Yan Liu, Pooja Bhunjun, Joaquin Navajas, Amir Eftekhar, Andrew Jackson, R. Quian Quiroga, and Timothy G. Constandinou. "An Analogue Front-End Model for Developing Neural Spike Sorting Systems," IEEE Trans. Biomedical Circuits and Systems, vol. 8, no. 2, pp. 216–217, 2014.

Chapter 4 contains content from:

- Joaquin Navajas, Deren Y. Barsakcioglu, Amir Eftekhar, Andrew Jackson, Timothy G. Constandinou, and Rodrigo Quian Quiroga. "Minimum requirements for accurate and efficient real-time on-chip spike sorting," Journal of neuroscience methods, vol. 230, pp. 51–64, 2014.

Chapter 5 contains content from:

- Sivylla E. Paraskevopoulou, Deren Y. Barsakcioglu, Mohammed R. Saberi, Amir Eftekhar, and Timothy G. Constandinou. "Feature Extraction using First and Second Derivative Extrema (FSDE), for Real-time and Hardware-Efficient Spike Sorting," Journal of neuroscience methods, vol. 215, no.1, pp. 29–37, 2013.

Bibliography

[1] Christopher & Reeve Foundation. (2013, Sep.) Paralysis facts and figures. [Online]. Available: http://www.christopherreeve.org/site/c.mtKZKgMWKwG/b.5184255/k.6D74/Prevalence_of_Paralysis.htm

[2] Apparelyzed. (2013, Sep.) Spinal cord injury statistics. [Online]. Available: http://www.apparelyzed.com/statistics.html

[3] Action for Blind People. (2013, Sep.) Facts and figures about issues around sight loss. [Online]. Available: https://www.actionforblindpeople.org.uk/about-us/media-centre/facts-and-figures-about-issues-around-sight-loss/

[4] Christopher & Reeve Foundation. (2013, Sep.) The costs of living with sci. [Online]. Available: http://www.christopherreeve.org/site/c.mtKZKgMWKwG/b.5193227/k.FC2/The_Costs_of_Living_with_Spinal_Cord_Injury.htm

[5] A. Society, "What is alzheimer's disease?" June 2015. [Online]. Available: http://www.alzheimers.org.uk/site/scripts/documents_info.php?documentID=100

[6] "Cochlear implants," 2011. [Online]. Available: http://www.nidcd.nih.gov/health/hearing/pages/coch.aspx

[7] L. R. Hochberg, D. Bacher, B. Jarosiewicz, N. Y. Masse, J. D. Simeral, J. Vogel, S. Haddadin, J. Liu, S. S. Cash, P. van der Smagt, and J. P. Donoghue, "Reach and grasp by people with tetraplegia using a neurally controlled robotic arm," *Nature*, vol. 485, no. 7398, pp. 372–375, 2012.

[8] T. W. Berger, D. Song, R. H. Chan, V. Z. Marmarelis, J. LaCoss, J. Wills, R. E. Hampson, S. Deadwyler, J. J. Granacki *et al.*, "A hippocampal cognitive prosthesis: multi-input, multi-output nonlinear modeling and vlsi implementation," *IEEE Trans. Neural Systems and Rehabilitation Engineering*, vol. 20, no. 2, pp. 198–211, 2012.

[9] Velliste, Meel and Perel, Sagi and Spalding, M Chance and Whitford, Andrew S and Schwartz, Andrew B, "Cortical control of a prosthetic arm for self-feeding," *Nature*, vol. 453, no. 7198, pp. 1098–1101, 2008.

[10] Simeral, JD and Kim, Sung-Phil and Black, MJ and Donoghue, JP and Hochberg, LR, "Neural control of cursor trajectory and click by a human with tetraplegia 1000 days after

implant of an intracortical microelectrode array," *Journal of neural engineering*, vol. 8, p. 025027, 2011.

[11] K. Najafi and K. D. Wise, "An implantable multielectrode array with on-chip signal processing," *IEEE Journal of Solid-State Circuits*, vol. 21, no. 6, pp. 1035–1044, 1986.

[12] K. Guillory and R. Normann, "A 100-channel system for real time detection and storage of extracellular spike waveforms," *J. Neuroscience Methods*, vol. 91, no. 1, pp. 21–29, 1999.

[13] R. H. Olsson III and K. D. Wise, "A three-dimensional neural recording microsystem with implantable data compression circuitry," *IEEE Journal of Solid-State Circuits*, vol. 40, no. 12, pp. 2796–2804, 2005.

[14] R. R. Harrison, P. T. Watkins, R. J. Kier, R. O. Lovejoy, D. J. Black, B. Greger, and F. Solzbacher, "A low-power integrated circuit for a wireless 100-electrode neural recording system," *IEEE Journal of Solid-State Circuits*, vol. 42, no. 1, pp. 123–133, 2007.

[15] S. Rai, J. Holleman, J. N. Pandey, F. Zhang, and B. Otis, "A 500μw neural tag with 2μv rms afe and frequency-multiplying mics/ism fsk transmitter," *Proc. IEEE International Solid-State Circuits Conference*, pp. 212–213, 2009.

[16] M. Mollazadeh, K. Murari, G. Cauwenberghs, and N. Thakor, "Micropower cmos integrated low-noise amplification, filtering, and digitization of multimodal neuropotentials," *IEEE Trans. Biomedical Circuits and Systems*, vol. 3, no. 1, pp. 1–10, 2009.

[17] X. Zou, X. Xu, L. Yao, and Y. Lian, "A 1-v 450-nw fully integrated programmable biomedical sensor interface chip," *IEEE Journal of Solid-State Circuits*, vol. 44, no. 4, pp. 1067–1077, 2009.

[18] M. S. Chae, Z. Yang, M. R. Yuce, L. Hoang, and W. Liu, "A 128-channel 6 mw wireless neural recording ic with spike feature extraction and uwb transmitter," *IEEE Trans. Neural Systems and Rehabilitation Engineering*, vol. 17, no. 4, pp. 312–321, 2009.

[19] Z. Xiao, C.-M. Tang, and D. Bashirullah, "A 20μw neural recording tag with supply-current-modulated afe in 0.13μm cmos," *Proc. IEEE International Solid-State Circuits Conference*, pp. 122–123, 2010.

[20] F. Shahrokhi, K. Abdelhalim, D. Serletis, P. L. Carlen, and R. Genov, "The 128-channel fully differential digital integrated neural recording and stimulation interface," *IEEE Trans. Biomedical Circuits and Systems*, vol. 4, no. 3, pp. 149–161, 2010.

[21] W. Wattanapanitch and R. Sarpeshkar, "A low-power 32-channel digitally programmable neural recording integrated circuit," *IEEE Trans. Biomedical Circuits and Systems*, vol. 5, no. 6, pp. 592–602, 2011.

[22] H. Gao, R. M. Walker, P. Nuyujukian, K. A. Makinwa, K. V. Shenoy, B. Murmann, and T. H. Meng, "Hermese: 96-channel full datarate direct neural interface in 0.13μm cmos," *IEEE Journal of Solid-State Circuits*, vol. 47, no. 4, pp. 1043–1055, 2012.

[23] C. M. Lopez, D. Prodanov, D. Braeken, I. Gligorijevic, W. Eberle, C. Bartic, R. Puers, and G. Gielen, "A multichannel integrated circuit for electrical recording of neural activity, with independent channel programmability," *IEEE Trans. Biomedical Circuits and Systems*, vol. 6, no. 2, pp. 101–110, 2012.

[24] A. Rodriguez-Perez, J. Ruiz-Amaya, M. Delgado-Restituto, and A. Rodriguez-Vazquez, "A low-power programmable neural spike detection channel with embedded calibration and data compression," *IEEE Trans. Biomedical Circuits and Systems*, vol. 6, no. 2, pp. 87–100, 2012.

[25] J. Lee, M. Johnson, and D. Kipke, "A tunable biquad switched-capacitor amplifier-filter for neural recording," *IEEE Trans. Biomedical Circuits and Systems*, vol. 4, no. 5, pp. 295–300, 2010.

[26] I. H. Stevenson and K. P. Kording, "How advances in neural recording affect data analysis," *Nature neuroscience*, vol. 14, no. 2, pp. 139–142, 2011.

[27] F. A. Mussa-Ivaldi and L. E. Miller, "Brain–machine interfaces: computational demands and clinical needs meet basic neuroscience," *Trends in Neurosciences*, vol. 26, no. 6, pp. 329–334, 2003.

[28] L. Resnik, S. L. Klinger, and K. Etter, "The DEKA Arm: Its features, functionality, and evolution during the Veterans Affairs Study to optimize the DEKA Arm," *Prosthetics and orthotics international*, vol. 38, no. 6, pp. 492–504, 2014.

[29] Y. Sankai, "Hal: Hybrid assistive limb based on cybernics," in *Robotics Research*. Springer, 2011, pp. 25–34.

[30] P. M. Lewis, H. M. Ackland, A. J. Lowery, and J. V. Rosenfeld, "Restoration of vision in blind individuals using bionic devices: A review with a focus on cortical visual prostheses," *Brain research*, vol. 1595, pp. 51–73, 2015.

[31] T. W. Berger, A. Ahuja, S. H. Courellis, S. A. Deadwyler, G. Erinjippurath, G. A. Gerhardt, G. Gholmieh, J. J. Granacki, R. Hampson, M. C. Hsaio *et al.*, "Restoring lost cognitive function," *Engineering in Medicine and Biology Magazine, IEEE*, vol. 24, no. 5, pp. 30–44, 2005.

[32] M. A. Nicolelis, "Brain–machine interfaces to restore motor function and probe neural circuits," *Nature Reviews Neuroscience*, vol. 4, no. 5, pp. 417–422, 2003.

[33] M. A. Nicolelis, "Actions from thoughts," *Nature*, vol. 409, no. 6818, pp. 403–407, 2001.

[34] M. A. Lebedev and M. A. Nicolelis, "Brain–machine interfaces: past, present and future," *Trends in Neurosciences*, vol. 29, no. 9, pp. 536–546, 2006.

[35] J. J. Shih, D. J. Krusienski, and J. R. Wolpaw, "Brain-computer interfaces in medicine," in *Mayo Clinic Proceedings*, vol. 87, no. 3. Elsevier, 2012, pp. 268–279.

[36] S. Waldert, T. Pistohl, C. Braun, T. Ball, A. Aertsen, and C. Mehring, "A review on directional information in neural signals for brain-machine interfaces," *Journal of Physiology-Paris*, vol. 103, no. 3, pp. 244–254, 2009.

[37] "Nautilus eeg cap," http://www.gtec.at/Products/Hardware-and-Accessories/g.Nautilus-Specs-Features.

[38] J. L. Roland, C. D. Hacker, J. D. Breshears, C. M. Gaona, R. E. Hogan, H. Burton, M. Corbetta, and E. C. Leuthardt, "Brain mapping in a patient with congenital blindness–a case for multimodal approaches," *Frontiers in human neuroscience*, vol. 7, 2013.

[39] E. Fernández, B. Greger, P. A. House, I. Aranda, C. Botella, J. Albisua, C. Soto-Sánchez, A. Alfaro, and R. A. Normann, "Acute human brain responses to intracortical microelectrode arrays: challenges and future prospects," *Frontiers in neuroengineering*, vol. 7, 2014.

[40] L. F. Nicolas-Alonso and J. Gomez-Gil, "Brain computer interfaces, a review," *Sensors*, vol. 12, no. 2, pp. 1211–1279, 2012.

[41] C. Neuper, G. R. Müller-Putz, R. Scherer, and G. Pfurtscheller, "Motor imagery and eeg-based control of spelling devices and neuroprostheses," *Progress in brain research*, vol. 159, pp. 393–409, 2006.

[42] L. A. Farwell and E. Donchin, "Talking off the top of your head: toward a mental prosthesis utilizing event-related brain potentials," *Electroencephalography and clinical Neurophysiology*, vol. 70, no. 6, pp. 510–523, 1988.

[43] J. R. Wolpaw and D. J. McFarland, "Control of a two-dimensional movement signal by a noninvasive brain-computer interface in humans," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 101, no. 51, pp. 17 849–17 854, 2004.

[44] F. Galán, M. Nuttin, E. Lew, P. W. Ferrez, G. Vanacker, J. Philips, and J. d. R. Millán, "A brain-actuated wheelchair: asynchronous and non-invasive brain–computer interfaces for continuous control of robots," *Clinical Neurophysiology*, vol. 119, no. 9, pp. 2159–2169, 2008.

[45] G. Mclaughlin and M. Imran, "Set it and forget it: Innovations in implantable medical technology," *IEEE SSC Magazine*, vol. 4, no. 2, pp. 30–33, 2012.

[46] N. J. Hill, D. Gupta, P. Brunner, A. Gunduz, M. A. Adamo, A. Ritaccio, and G. Schalk, "Recording human electrocorticographic (ECoG) signals for neuroscientific research and real-time functional cortical mapping," *Journal of visualized experiments: JoVE*, no. 64, 2012.

[47] G. Buzsáki, C. A. Anastassiou, and C. Koch, "The origin of extracellular fields and currents—eeg, ecog, lfp and spikes," *Nature Reviews Neuroscience*, vol. 13, no. 6, pp. 407–420, 2012.

[48] B. Gosselin, "Recent advances in neural recording microsystems," *Sensors*, vol. 11, no. 5, pp. 4572–4597, 2011.

[49] L. R. Hochberg, M. D. Serruya, G. M. Friehs, J. A. Mukand, M. Saleh, A. H. Caplan, A. Branner, D. Chen, R. D. Penn, and J. P. Donoghue, "Neuronal ensemble control of prosthetic devices by a human with tetraplegia," *Nature*, vol. 442, no. 7099, pp. 164–171, 2006.

[50] E. Strickland, "Telekinesis made simple," *IEEE Spectrum*, vol. 52, no. 7, pp. 11–12, 2015.

[51] E. C. Leuthardt, G. Schalk, J. R. Wolpaw, J. G. Ojemann, and D. W. Moran, "A brain–computer interface using electrocorticographic signals in humans," *Journal of neural engineering*, vol. 1, no. 2, p. 63, 2004.

[52] W. Wang, A. Degenhart, J. Collinger, R. Vinjamuri, G. Sudre, P. Adelson, D. Holder, E. Leuthardt, D. Moran, M. Boninger, A. Schwartz, D. Crammond, E. Tyler-Kabara, and D. Weber, "Human motor cortical activity recorded with micro-ecog electrodes, during individual finger movements," in *Proc. IEEE International Conference of Engineering in Medicine and Biology Society*, Sept 2009, pp. 586–589.

[53] R. Muller, H.-P. Le, W. Li, P. Ledochowitsch, S. Gambini, T. Bjorninen, A. Koralek, J. Carmena, M. Maharbiz, E. Alon, and J. Rabaey, "A minimally invasive 64-channel wireless μecog implant," *IEEE Journal of Solid-State Circuits*, vol. 50, no. 1, pp. 344–359, Jan 2015.

[54] J. Viventi, D.-H. Kim, L. Vigeland, E. S. Frechette, J. A. Blanco, Y.-S. Kim, A. E. Avrin, V. R. Tiruvadi, S.-W. Hwang, A. C. Vanleer *et al.*, "Flexible, foldable, actively multiplexed, high-density electrode array for mapping brain activity in vivo," *Nature neuroscience*, vol. 14, no. 12, pp. 1599–1605, 2011.

[55] C. Briar, *Nervous System*. Mosby, 2003.

[56] H. Lodish, A. Berk, S. L. Zipursky, P. Matsudaira, D. Baltimore, and J. Darnell, Eds., *Molecular Cell Biology*. W. H. Freeman, 2000, vol. 2.

[57] M. F. Bear, B. W. Connors, and M. A. Paradiso, *Neuroscience: Exploring the Brain*. Lippincott Williams and Wilkins, 2006.

[58] "Action potential," Creative Commons, Licensed under CC BY-SA 3.0 via Wikimedia Commons. Original created by en:User:Chris 73, updated by en:User:Diberri, converted to SVG by tiZom - Own work. [Online]. Available: https://commons.wikimedia.org/wiki/File:Action_potential.svg#/media/File:Action_potential.svg

[59] C. Gold, D. A. Henze, C. Koch, and G. Buzsáki, "On the origin of the extracellular action potential waveform: a modeling study," *Journal of neurophysiology*, vol. 95, no. 5, pp. 3113–3128, 2006.

[60] G. Buzsáki, "Large-scale recording of neuronal ensembles," *Nature neuroscience*, vol. 7, no. 5, pp. 446–451, 2004.

[61] F. Mechler and J. Victor, "Dipole characterization of single neurons from their extracellular action potentials," *Journal of computational neuroscience*, vol. 32, no. 1, pp. 73–100, 2012.

[62] C. Pedreira, J. Martinez, M. J. Ison, and R. Q. Quiroga, "How many neurons can we see with current spike sorting algorithms?" *Journal of neuroscience methods*, 2012.

[63] M. Toledo-Rodriguez, B. Blumenfeld, C. Wu, J. Luo, B. Attali, P. Goodman, and H. Markram, "Correlation maps allow neuronal electrical properties to be predicted from single-cell gene expression profiles in rat neocortex," *Cerebral Cortex*, vol. 14, no. 12, pp. 1310–1327, 2004.

[64] R. Quian Quiroga, "What is the real shape of extracellular spikes?" *Journal of neuroscience methods*, vol. 177, no. 1, pp. 194–198, 2009.

[65] H. G. Rey, C. Pedreira, and R. Q. Quiroga, "Past, present and future of spike sorting techniques," *Brain research bulletin*, 2015.

[66] H. G. Rey, M. J. Ison, C. Pedreira, A. Valentin, G. Alarcon, R. Selway, M. P. Richardson, and R. Quian Quiroga, "Single-cell recordings in the human medial temporal lobe," *Journal of anatomy*, 2014.

[67] A. D. Redish, F. P. Battaglia, M. K. Chawla, A. D. Ekstrom, J. L. Gerrard, P. Lipa, E. S. Rosenzweig, P. F. Worley, J. F. Guzowski, B. L. McNaughton *et al.*, "Independence of firing correlates of anatomically proximate hippocampal pyramidal cells," *Journal of neuroscience*, vol. 21, no. 5, p. RC134, 2001.

[68] S. Todorova, P. Sadtler, A. Batista, S. Chase, and V. Ventura, "To sort or not to sort: the impact of spike-sorting on neural decoding performance," *Journal of neural engineering*, vol. 11, no. 5, p. 056005, 2014.

[69] K. Ganguly, D. F. Dimitrov, J. D. Wallis, and J. M. Carmena, "Reversible large-scale modification of cortical networks during neuroprosthetic control," *Nature neuroscience*, vol. 14, no. 5, pp. 662–667, 2011.

[70] M. D. Serruya, N. G. Hatsopoulos, L. Paninski, M. R. Fellows, and J. P. Donoghue, "Brain-machine interface: Instant neural control of a movement signal," *Nature*, vol. 416, no. 6877, pp. 141–142, 2002.

[71] D. M. Taylor, S. I. H. Tillery, and A. B. Schwartz, "Direct cortical control of 3d neuroprosthetic devices," *Science*, vol. 296, no. 5574, pp. 1829–1832, 2002.

[72] P. R. Kennedy and R. A. Bakay, "Restoration of neural output from a paralyzed patient by a direct brain connection," *Neuroreport*, vol. 9, no. 8, pp. 1707–1711, 1998.

[73] M. Aghagolzadeh and K. Oweiss, "An adaptive wireless communication protocol for neural data transmission in freely behaving subjects," in *Proc. IEEE Neural Engineering*, 2011, pp. 404–407.

[74] R. Harrison, "The design of integrated circuits to observe brain activity," *Proc. of the IEEE*, vol. 96, no. 7, pp. 1203–1216, 2008.

[75] B. I. Rapoport, L. Turicchia, W. Wattanapanitch, T. J. Davidson, and R. Sarpeshkar, "Efficient universal computing architectures for decoding neural activity," *PloS one*, vol. 7, no. 9, p. e42492, 2012.

[76] S. Shoham, D. H. O'Connor, and R. Segev, "How silent is the brain: is there a dark matter problem in neuroscience?" *Journal of Comparative Physiology A*, vol. 192, no. 8, pp. 777–784, 2006.

[77] R. Q. Quiroga, Z. Nadasdy, and Y. Ben-Shaul, "Unsupervised spike detection and sorting with wavelets and superparamagnetic clustering," *Neural Computation*, vol. 16, no. 8, pp. 1661–87, 2004.

[78] P.-M. Zhang, J.-Y. Wu, Y. Zhou, P.-J. Liang, and J.-Q. Yuan, "Spike sorting based on automatic template reconstruction with a partial solution to the overlapping problem," *Journal of neuroscience methods*, vol. 135, no. 12, pp. 55 – 65, 2004.

[79] A. Bar-Hillel, A. Spiro, and E. Stark, "Spike sorting: Bayesian clustering of non-stationary data," *Journal of neuroscience methods*, vol. 157, no. 2, pp. 303–316, 2006.

[80] U. Rutishauser, E. Schuman, and A. Mamelak, "Online detection and sorting of extracellularly recorded action potentials in human medial temporal lobe recordings, in vivo," *Journal of neuroscience methods*, vol. 154, no. 1, pp. 204–224, 2006.

[81] B. Naundorf, F. Wolf, and M. Volgushev, "Unique features of action potential initiation in cortical neurons," *Nature*, vol. 440, no. 7087, pp. 1060–1063, 2006.

[82] L. Smith and N. Mtetwa, "A tool for synthesizing spike trains with realistic interference," *Journal of neuroscience methods*, vol. 159, no. 1, pp. 170–180, 2007.

[83] J. Guo, J. Yuan, and M. Chan, "Modeling of the cell-electrode interface noise for microelectrode arrays," *IEEE Trans. Biomedical Circuits and Systems*, vol. 6, no. 6, pp. 605–613, 2012.

[84] K. Harris, D. Henze, J. Csicsvari, H. Hirase, and G. Buzsáki, "Accuracy of tetrode spike separation as determined by simultaneous intracellular and extracellular measurements," *Journal of Neurophysiology*, vol. 84, no. 1, pp. 401–414, 2000.

[85] J. Turner, W. Shain, D. Szarowski, M. Andersen, S. Martins, M. Isaacson, and H. Craighead, "Cerebral astrocyte response to micromachined silicon implants," *Experimental neurology*, vol. 156, no. 1, pp. 33–49, 1999.

[86] V. S. Polikov, P. A. Tresco, and W. M. Reichert, "Response of brain tissue to chronically implanted neural electrodes," *Journal of neuroscience methods*, vol. 148, no. 1, pp. 1–18, 2005.

[87] J. Herbst, S. Gammeter, D. Ferrero, and R. Hahnloser, "Spike sorting with hidden markov models," *Journal of neuroscience methods*, vol. 174, no. 1, pp. 126–134, 2008.

[88] S. Shoham, M. Fellows, and R. Normann, "Robust, automatic spike sorting using mixtures of multivariate t-distributions," *Journal of neuroscience methods*, vol. 127, no. 2, pp. 111–122, 2003.

[89] L. Camuna-Mesa and R. Q. Quiroga, "A detailed and fast model of extracellular recordings," *Neural Computation*, vol. 25, pp. 1191–1212, 2013.

[90] R. R. Llinás, "The intrinsic electrophysiological properties of mammalian neurons: insights into central nervous system function," *Science*, vol. 242, no. 4886, pp. 1654–1664, 1988.

[91] M. Farrant, D. Feldmeyer, T. Takahashi, and S. G. Cull-Candy, "NMDA-receptor channel diversity in the developing cerebellum," 1994.

[92] M. Lewicki, "A review of methods for spike sorting: the detection and classification of neural action potentials," *Network: Computation in Neural Systems*, vol. 9, no. 4, pp. 53–78, 1998.

[93] K. Balasubramanian and I. Obeid, "Fuzzy logic-based spike sorting system," *Journal of neuroscience methods*, vol. 198, no. 1, pp. 125–134, 2011.

[94] M. Abeles and M. Goldstein Jr, "Multispike train analysis," *Proceedings of the IEEE*, vol. 65, no. 5, pp. 762–773, 1977.

[95] J. Letelier and P. Weber, "Spike sorting based on discrete wavelet transform coefficients," *Journal of neuroscience methods*, vol. 101, no. 2, pp. 93–106, 2000.

[96] S. Gibson, J. Judy, and D. Markovic, "Comparison of spike-sorting algorithms for future hardware implementation," in *Proc. IEEE International Conference of the Engineering in Medicine and Biology Society*, 2008, pp. 5015–5020.

[97] A. Kamboh and A. Mason, "Computationally efficient neural feature extraction for spike sorting in implantable high-density recording systems," *IEEE Trans. on Neural Systems and Rehabilitation Engineering*, 2012.

[98] A. Zviagintsev, Y. Perelman, and R. Ginosar, "Low power architectures for spike sorting," in *Proc. IEEE International Conference of the Engineering in Medicine and Biology Society*, 2005, pp. 162–165.

[99] H. Chan, T. Wu, S. Lee, S. Fang, P. Chao, and M. Lin, "Classification of neuronal spikes over the reconstructed phase space," *Journal of neuroscience methods*, vol. 168, no. 1, pp. 203–211, 2008.

[100] J. Dai, X. Liu, Y. Yi, H. Zhang, J. Wang, S. Zhang, and X. Zheng, "Experimental study on neuronal spike sorting methods," in *Proc. IEEE International Conference on Future Generation Communication and Networking*, vol. 2, 2008, pp. 230–233.

[101] S. Kim and J. McNames, "Automatic spike detection based on adaptive template matching for extracellular neural recordings," *Journal of neuroscience methods*, vol. 165, no. 2, pp. 165–174, 2007.

[102] M. Mahmud, A. Bertoldo, S. Girardi, M. Maschietto, and S. Vassanelli, "Sigmate: A matlab–based automated tool for extracellular neuronal signal processing and analysis," *Journal of neuroscience methods*, vol. 207, no. 1, pp. 97–112, 2012.

[103] C. Pouzat, O. Mazor, and G. Laurent, "Using noise signature to optimize spike-sorting and to assess neuronal classification quality," *Journal of neuroscience methods*, vol. 122, no. 1, pp. 43–57, 2002.

[104] R. Xu and D. Wunsch, "Clustering algorithms in biomedical research: A review," *IEEE Reviews in Biomedical Engineering*, vol. 3, pp. 120 –154, 2010.

[105] R. Q. Quiroga, "Unsupervised detection and sorting," January 2009. [Online]. Available: http://www2.le.ac.uk/departments/engineering/research/bioengineering/neuroengineering-lab/spike-sorting

[106] H. Abdi and L. Williams, "Principal component analysis," *Wiley Interdisciplinary Reviews: Computational Statistics*, vol. 2, no. 4, pp. 433–459, 2010.

[107] A. Hyvärinen, J. Karhunen, and E. Oja, *Independent component analysis*. Wiley-interscience, 2001, vol. 26.

[108] Z. Tiganj and M. Mboup, "Deflation technique for neural spike sorting in multi-channel recordings," in *Proc. IEEE MLSP*, 2011, pp. 1–6.

[109] G. Bausch and H. Beikirch, "An unsupervised method for realtime spike sorting," in *Proc. IEEE IDAACS*, vol. 2. IEEE, 2011, pp. 536–539.

[110] T. Sato, T. Suzuki, and K. Mabuchi, "Fast automatic template matching for spike sorting based on davies-bouldin validation indices," 2007, pp. 3200–3203.

[111] R. Yadav, A. Shah, J. Loeb, M. Swamy, and R. Agarwal, "A novel unsupervised spike sorting algorithm for intracranial EEG," in *Proc. IEEE International Conference of Engineering in Medicine and Biology Society*, 2011, pp. 7545–7548.

[112] Y. Ghanbari, L. Spence, and P. Papamichalis, "A graph-laplacian-based feature extraction algorithm for neural spike sorting," in *Proc. IEEE International Conference of Engineering in Medicine and Biology Society*, 2009, pp. 3142 –3145.

[113] D. Schwarz, M. Zilany, M. Skevington, N. Huang, B. Flynn, and L. Carney, "Semi-supervised spike sorting using pattern matching and a scaled mahalanobis distance metric," *Journal of neuroscience methods*, vol. 206, no. 2, pp. 120–131, 2012.

[114] X. Yang and S. Shamma, "A totally automated system for the detection and classification of neural spikes," *IEEE Trans. Biomedical Engineering*, vol. 35, no. 10, pp. 806 –816, oct. 1988.

[115] H. Cho, D. Corina, J. Brinkley, G. Ojemann, and L. Shapiro, "A new template matching method using variance estimation for spike sorting." in *Proc. IEEE International Conference on Neural Engineering*, 2005, pp. 225 –228.

[116] J. Josefina Janier and N. Nadir, "Fast template matching method based optimized sum of absolute difference algorithm for face localization," *International Journal of Computer Applications (IJCA)*, 2011.

[117] A. Bonfanti, T. Borghi, R. Gusmeroli, G. Zambra, A. Oliyink, L. Fadiga, A. Spinelli, and G. Baranauskas, "A low-power integrated circuit for analog spike detection and sorting in neural prosthesis systems," in *Proc. IEEE Biomedical Circuits and Systems Conference*, 2008, pp. 257 –260.

[118] C.-C. Peng and R. Bashirullah, "An autonomous real-time neural signal processor," in *Proc. IEEE International Conference on Neural Engineering*, 29 2009-may 2 2009, pp. 665 –668.

[119] M. Aghagolzadeh, F. Zhang, and K. Oweiss, "An implantable VLSI architecture for real time spike sorting in cortically controlled Brain Machine Interfaces," in *Proc. IEEE International Conference of Engineering in Medicine and Biology Society*, 2010, pp. 1569 –1572.

[120] K. Awais and M. Andrew, "On-chip feature extraction for spike sorting in high density implantable neural recording systems," in *Proc. IEEE Biomedical Circuits and Systems Conference*, 2010, pp. 13–16.

[121] F. Zhang, M. Aghagolzadeh, and K. Oweiss, "An implantable neuroprocessor for multi-channel compressive neural recording and on-the-fly spike sorting with wireless telemetry," in *Proc. IEEE Biomedical Circuits and Systems Conference*, 2010, pp. 1 –4.

[122] M. Chae, W. Liu, Z. Yang, T. Chen, J. Kim, M. Sivaprakasam, and M. Yuce, "A 128-Channel 6mW Wireless Neural Recording IC with On-the-Fly Spike Sorting and UWB Tansmitter," in *Proc. IEEE International Solid-State Circuits Conference*, feb. 2008, pp. 146 –603.

[123] V. Karkare, S. Gibson, C.-H. Yang, H. Chen, and D. Markovic, "A 75 uw, 16-channel neural spike-sorting processor with unsupervised clustering," in *Symposium on VLSI Circuits*, 2011, pp. 252 –253.

[124] D. Sebald and A. Branner, "Automatic spike sorting for real-time applications," in *Proc. IEEE International Conference on Neural Engineering*, 2007, pp. 670 –674.

[125] S. Huang, J. Oresko, Y. Sun, and A. Cheng, "Programmable neural processing on a smartdust," in *Proc. IEEE DAC*, 2009, pp. 619 –620.

[126] R. Chandra and L. Optican, "Detection, classification, and superposition resolution of action potentials in multiunit single-channel recordings by an on-line real-time neural network," *IEEE Trans. Biomedical Engineering*, vol. 44, no. 5, pp. 403 –412, 1997.

[127] B. Yu, T. Mak, X. Li, F. Xia, A. Yakovlev, Y. Sun, and C. Poon, "Real-time fpga-based multichannel spike sorting using hebbian eigenfilters," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, no. 99, pp. 1–1, 2011.

[128] T.-C. Chen, T.-C. Ma, Y.-Y. Chen, and L.-G. Chen, "Low power and high accuracy spike sorting microprocessor with on-line interpolation and re-alignment in 90nm cmos process," in *Proc. IEEE International Conference of Engineering in Medicine and Biology Society*, 2012, pp. 4485–4488.

[129] V. Karkare, S. Gibson, and D. Markovic, "A 130-μw, 64-channel neural spike-sorting dsp chip," *IEEE Journal of Solid-State Circuits*, vol. 46, no. 5, pp. 1214–1222, 2011.

[130] Y. Yang and A. Mason, "On-chip spike clustering & classification using self organizing map for neural recording implants," in *Proc. Biomedical Circuits and Systems Conference*. IEEE, 2011, pp. 145–148.

[131] Y. Yang, S. Boling, A. Eftekhar, S. E. Paraskevopoulou, T. G. Constandinou, and A. J. Mason, "Computationally efficient feature denoising filter and selection of optimal features

for noise insensitive spike sorting," in *Proc. IEEE International Conference of Engineering in Medicine and Biology Society*, 2014, pp. 1251–1254.

[132] Y. Yang, C. S. Boling, and A. J. Mason, "Power-area efficient vlsi implementation of decision tree based spike classification for neural recording implants," in *Proc. IEEE Biomedical Circuits and Systems Conference*. IEEE, 2014, pp. 380–383.

[133] S. Gibson, R. Chandler, V. Karkare, D. Marković, and J. W. Judy, "An efficiency comparison of analog and digital spike detection," *Proc. IEEE International Neural Conference on Neural Engineering*, pp. 423–428, 2009.

[134] I. Obeid and P. Wolf, "Evaluation of spike-detection algorithms fora brain-machine interface application," *IEEE Trans. Biomedical Engineering*, vol. 51, no. 6, pp. 905–911, 2004.

[135] D. L. Donoho and J. M. Johnstone, "Ideal spatial adaptation by wavelet shrinkage," *Biometrika*, vol. 81, no. 3, pp. 425–455, 1994.

[136] P. H. Thakur, H. Lu, S. S. Hsiao, and K. O. Johnson, "Automated optimal detection and classification of neural action potentials in extra-cellular recordings," *Journal of Neuroscience Methods*, vol. 162, no. 1, pp. 364–376, 2007.

[137] L. Traver, C. Tarin, P. Martí, and N. Cardona, "Adaptive-threshold neural spike detection by noise-envelope tracking," *Electronics Letters*, vol. 43, no. 24, pp. 1333–1335, 2007.

[138] B. Gosselin and M. Sawan, "Adaptive detection of action potentials using ultra low-power cmos circuits," in *Proc. IEEE Biomedical Circuits and Systems Conference*. IEEE, 2008, pp. 209–212.

[139] S. Mukhopadhyay and G. Ray, "A new interpretation of nonlinear energy operator and its efficacy in spike detection," *IEEE Trans. Biomedical Engineering*, vol. 45, no. 2, pp. 180–187, 1998.

[140] W. M. Reichert, *Indwelling neural implants: strategies for contending with the in vivo environment*. CRC Press, 2010.

[141] Z. Zumsteg, C. Kemere, S. O'Driscoll, G. Santhanam, R. Ahmed, K. Shenoy, and T. Meng, "Power feasibility of implantable digital spike sorting circuits for neural prosthetic systems," *IEEE Trans. Neural Systems and Rehabilitation Engineering*, vol. 13, no. 3, pp. 272–279, 2005.

[142] A. Eftekhar, S. Paraskevopoulou, and T. Constandinou, "Towards a next generation neural interface: Optimizing power, bandwidth and data quality," *Proc. IEEE Biomedical Circuits and Systems Conference*, pp. 122–125, 2010.

[143] S. E. Paraskevopoulou, D. Y. Barsakcioglu, M. R. Saberi, A. Eftekhar, and T. G. Constandinou, "Feature extraction using first and second derivative extrema (fsde) for real-time and hardware-efficient spike sorting," *Journal of neuroscience methods.*, vol. 215, no. 1, pp. 29–37, 2013.

[144] P. T. Thorbergsson, H. Jorntell, F. Bengtsson, M. Garwicz, J. Schouenborg, and A. Johansson, "Spike library based simulator for extracellular single unit neuronal signals," in *Proc. IEEE Engineering in Medicine and Biology Society.* IEEE, 2009, pp. 6998–7001.

[145] M. S. Chae, W. Liu, and M. Sivaprakasam, "Design optimization for integrated neural recording systems," *IEEE Journal of Solid-State Circuits*, vol. 43, no. 9, 2008.

[146] J. Xu, T. Wu, and Z. Yang, "A new system architecture for future long-term high-density neural recording," *IEEE Trans. Circuits and Systems II: Express Briefs*, vol. 60, no. 7, pp. 402–406, 2013.

[147] I. H. Stevenson and K. P. Kording, "How advances in neural recording affect data analysis," *Nature neuroscience*, vol. 14, no. 2, 2011.

[148] M. P. Ward, P. Rajdev, C. Ellison, and P. P. Irazoqui, "Toward a comparison of microelectrodes for acute and chronic recordings," *Brain Research*, vol. 1282, pp. 183–200, 2009.

[149] J. Guo, J. Yuan, and M. Chan, "Modeling of the cell-electrode interface noise for modeling of the cell-electrode interface noise for microelectrode arrays," *IEEE Trans. Biomedical Circuits and Systems*, vol. 6, no. 6, pp. 605–613, 2012.

[150] W. Franks, I. Schenker, P. Schmutz, and A. Hierlemann, "Impedance characterization and modeling of electrodes for biomedical applications," *IEEE. Trans. Biomedical Engineering*, vol. 52, no. 7, pp. 1295–1302, 2005.

[151] K. W. Horch and G. S. Dhillon, Eds., *Neuroprosthetics Theory and Practice.* World Scientific Publishing, 2004, vol. 2.

[152] Y. Perelman and R. Ginosar, *The Neuroprocessor.* Springer, 2008.

[153] D. Y. Barsakcioglu, A. Eftekhar, and T. G. Constandinou, "Design optimisation of front-end neural interfaces for spike sorting systems," *Proc. IEEE International Symposium on Circuits and Systems*, pp. 2501–2504, 2013.

[154] J. F. Kaiser, "On a simple algorithm to calculate theenergy'of a signal," *Proc. IEEE International Conference on Acoustics, Speech and Signal Processing*, pp. 381–384, 1990.

[155] S. Gibson, J. Judy, and D. Markovic, "Comparison of spike-sorting algorithms for future hardware implementation," *Proc. IEEE International Conference Engineering in Medicine and Biology Society*, pp. 5015–5020, 2008.

[156] S. Mukhopadhyay and G. C. Ray, "A new interpretation of nonlinear energy operator and its efficacy in spike detection," *IEEE Trans. Biomedical Engineering*, vol. 45, no. 2, pp. 180–187, 1998.

[157] J. Martinez, C. Pedreira, M. J. Ison, and R. Q. Quiroga, "Realistic simulation of extracellular recordings," *Journal of neuroscience methods*, vol. 184, no. 2, pp. 285–293, 2009.

[158] T. Delbrucki, "Bump circuits for computing similarity and dissimilarity of analog voltages," *Proc. IJCNN*, vol. 1, pp. 475–479, 1991.

[159] M. Rizk, C. A. Bossetti, T. A. Jochum, S. H. Callender, M. A. L. Nicolelis, D. A. Turner, and P. D. Wolf, "A fully implantable 96-channel neural data acquisition system," *Journal of neural engineering*, vol. 6, 2009.

[160] R. Sarpeshkar, "Universal principles for ultra low power and energy efficient design," *IEEE Trans. Circuits and Systems II: Express Briefs*, vol. 59, no. 4, pp. 193–198, 2012.

[161] E. De Benedetti, S. Lew, and B. Zanutto, "A wavelet approach for on-line spike sorting in tetrode recordings," in *Proc. IEEE International Conference of Engineering in Medicine and Biology Society*, 2010, pp. 6662–6665.

[162] S. Gibson, J. W. Judy, and D. Markovic, "Technology-aware algorithm design for neural spike detection, feature extraction, and dimensionality reduction," *IEEE Trans. on Neural Systesm and Rehabilitation Engineering*, vol. 18, pp. 469–478, 2010.

[163] Z. S. Zumsteg, R. E. Ahmed, G. Santhanam, K. V. Shenoy, and T. H. Meng, "Power feasibility of implantable digital spike-sorting circuits for neural prosthetic systems," *Proc. IEEE International Conference of Engineering in Medicine and Biology Society*, pp. 4237–4240, 2004.

[164] L.-F. Cheng, T.-C. Chen, N.-F. Chang, and L.-G. Chen, "Density-based hardware-oriented classification for spike sorting microsystems," in *Proc. IEEE International Conference on Neural Engineering*, 2011, pp. 170–173.

[165] A. Zviagintsev, Y. Perelman, and R. Ginosar, "Low power architectures for spike sorting," in *Proc. IEEE International Conference on Neural Engineering*, 2005, pp. 162–165.

[166] B. C. Wheeler and W. J. Heetderks, "A comparison of techniques for classification of multiple neural signals," *IEEE Trans. on Biomedical Engineering*, vol. BME-29, no. 12, pp. 752 –759, 1982.

[167] P.-M. Zhang, J.-Y. Wu, Y. Zhou, P.-J. Liang, and J.-Q. Yuan, "Spike sorting in multichannel extracellular recordings of retinas," in *Proc. IEEE International Conference on Neural Networks and Signal Processing*, vol. 1, 2003, pp. 712 –715 Vol.1.

[168] C. Vargos-Irwin and J. P. Donoghue, "Automated spike sorting using density grid contour clustering and subtractive waveform decomposition," *Journal of neuroscience methods*, vol. 164, pp. 1–18, 2007.

[169] Y. Chen, T. Chen, and L. Chen, "Accuracy and power tradeoff in spike sorting microsystems with cubic spline interpolation," in *Proc. IEEE International Symposium on Circuits and Systems*. IEEE, 2010, pp. 1508–1511.

[170] T. Blanche and N. Swindale, "Nyquist interpolation improves neuron yield in multiunit recordings," *Journal of neuroscience methods*, vol. 155, no. 1, pp. 81–91, 2006.

[171] I. N. Bankman, K. O. Johnson, and W. Schneider, "Optimal detection, classification, and superposition resolution in neural waveform recordings," *IEEE Trans. on Biomedical Engineering*, vol. 40, pp. 836–841, 1993.

[172] Y. Zhu, T. Tan, and Y. Wang, "Biometric personal identification based on iris patterns," in *Proc. IEEE International Conference on Pattern Recognition*, vol. 2, 2000, pp. 801–804.

[173] R. Dlugosz and K. Iniewski, "Ultra low power current-mode algorithmic analog-to-digital converter implemented in 0.18/spl mu/m cmos technology for wireless sensor network," in *Proc. IEEE International Conference on Mixed Design of Integrated Circuits and System*, 2006, pp. 401–406.

[174] I. Bankman, *Handbook of medical imaging: processing and analysis*. Academic Press, 2000.

[175] L. Erup, F. Gardner, and R. Harris, "Interpolation in digital modems. ii. implementation and performance," *IEEE Trans. Communications*, vol. 41, no. 6, pp. 998–1008, 1993.

[176] J. Kim, "Efficient polynomial interpolation filters with symmetric coefficients," in *Proc. IEEE Vehicular Technology Conference*, vol. 5, 2006, pp. 2276–2279.

[177] Y. Ghanbari, P. Papamichalis, and L. Spence, "Graph-laplacian features for neural waveform classification," *IEEE Trans. on Biomedical Engineering*, vol. 58, no. 5, pp. 1365–1372, 2011.

[178] R. Q. Quiroga, R. Mukamel, E. A. Isham, R. Malach, and I. Fried, "Human single-neuron responses at the threshold of conscious recognition," *Proceedings of the National Academy of Sciences*, vol. 105, no. 9, pp. 3599–3604, 2008.

[179] J. Mavoori, A. Jackson, C. Diorio, and E. Fetz, "An autonomous implantable computer for neural recording and stimulation in unrestrained primates," *Journal of neuroscience methods*, vol. 148, no. 1, pp. 71–77, 2005.

[180] B. Gosselin and M. Sawan, "A low-power integrated neural interface with digital spike detection and extraction," *Analog Integrated Circuits and Signal Processing*, vol. 64, no. 1, pp. 3–11, 2010.

[181] M. Blatt, S. Wiseman, and E. Domany, "Superparamagnetic clustering of data," *Physical review letters*, vol. 76, no. 18, p. 3251, 1996.

[182] E. Domany, "Superparamagnetic clustering of datathe definitive solution of an ill-posed problem," *Physica A: Statistical Mechanics and its Applications*, vol. 263, no. 1, pp. 158–169, 1999.

[183] I. Fried, K. A. MacDonald, and C. L. Wilson, "Single neuron activity in human hippocampus and amygdala during recognition of faces and objects," *Neuron*, vol. 18, no. 5, pp. 753–765, 1997.

[184] R. Q. Quiroga, "Spike sorting," *Current Biology*, vol. 22, no. 2, pp. R45–R46, 2012.

[185] S. Gibson, J. W. Judy, and D. Marković, "An FPGA-based platform for accelerated offline spike sorting," *Journal of neuroscience methods*, vol. 215, no. 1, pp. 1–11, 2013.

[186] A. D. Rush and P. R. Troyk, "A power and data link for a wireless-implanted neural recording system," *IEEE Trans. Biomedical Engineering*, vol. 59, no. 11, pp. 3255–3262, 2012.

[187] P. Smith, "Comparing low-power wireless technologies," Digi-Key Corporation, Tech. Rep., August 2011.

[188] V. Karkare, S. Gibson, and D. Markovic, "A 75-μw, 16-channel neural spike-sorting processor with unsupervised clustering," *IEEE Journal of Solid-State Circuits*, vol. 48, no. 9, pp. 2230–2238, 2013.

[189] A. Jackson, J. Mavoori, and E. E. Fetz, "Long-term motor cortex plasticity induced by an electronic neural implant," *Nature*, vol. 444, no. 7115, pp. 56–60, 2006.

[190] ——, "Correlations between the same motor cortex cells and arm muscles during a trained task, free behavior, and natural sleep in the macaque monkey," *Journal of neurophysiology*, vol. 97, no. 1, pp. 360–374, 2007.

[191] K. G. Oweiss, A. Mason, Y. Suhail, A. M. Kamboh, and K. E. Thomson, "A scalable wavelet transform vlsi architecture for real-time signal processing in high-density intracortical implants," *IEEE Trans. Circuits and Systems I: Regular Papers*, vol. 54, no. 6, pp. 1266–1278, 2007.

[192] F. Abu-Nimeh, M. Aghagolzadeh, and K. Oweiss, "Real-time, on-chip spike sorting and firing rate estimation for cortically-controlled brain machine interface applications," *In: Society for Neuroscience Meeting*, p. 39, 2009.

[193] A. M. Kamboh and A. J. Mason, "Computationally efficient neural feature extraction for spike sorting in implantable high-density recording systems," *IEEE Trans. Neural Systems and Rehabilitation Engineering*, vol. 21, no. 1, pp. 1–9, 2013.

[194] M. Saeed and A. M. Kamboh, "Hardware architecture for on-chip unsupervised online neural spike sorting," in *Proc. IEEE International Conference on Neural Engineering*. IEEE, 2013, pp. 1319–1322.

[195] C. Chestek, V. Gilja, P. Nuyujukian, R. J. Kier, F. Solzbacher, S. Ryu, R. R. Harrison, and K. V. Shenoy, "Hermesc: low-power wireless neural recording system for freely moving primates," *IEEE Trans. Neural Systems and Rehabilitation Engineering*, vol. 17, no. 4, pp. 330–338, 2009.

[196] R. R. Harrison, H. Fotowat, R. Chan, R. J. Kier, R. Olberg, A. Leonardo, and F. Gabbiani, "Wireless neural/emg telemetry systems for small freely moving animals," *IEEE Trans. Biomedical Circuits and Systems*, vol. 5, no. 2, pp. 103–111, 2011.

[197] T. Horiuchi, T. Swindell, D. Sander, and P. Abshier, "A low-power cmos neural amplifier with amplitude measurements for spike sorting," in *Proc. IEEE International Symposium on Circuits and Systems*, vol. 4, 2004, pp. IV–29.

[198] S. E. Paraskevopoulou and T. G. Constandinou, "A sub-1μw neural spike-peak detection and spike-count rate encoding circuit," in *Proc. IEEE Biomedical Circuits and Systems Conference*. IEEE, 2011, pp. 29–32.

[199] E. Koutsos, S. E. Paraskevopoulou, and T. G. Constandinou, "A 1.5 μw neo-based spike detector with adaptive-threshold for calibration-free multichannel neural interfaces," in *Proc. IEEE International Symposium on Circuits and Systems*. IEEE, 2013, pp. 1922–1925.

[200] M. Rizk, I. Obeid, S. H. Callender, and P. D. Wolf, "A single-chip signal processing and telemetry engine for an implantable 96-channel neural data acquisition system," *Journal of neural engineering*, vol. 4, no. 3, p. 309, 2007.

[201] P. Gorzelic, S. Schiff, and A. Sinha, "Model-based rational feedback controller design for closed-loop deep brain stimulation of parkinson's disease," *Journal of neural engineering*, vol. 10, no. 2, p. 026016, 2013.

[202] J. Müller, D. J. Bakkum, and A. Hierlemann, "Sub-millisecond closed-loop feedback stimulation between arbitrary sets of individual neurons," *Frontiers in Neural Circuits*, vol. 6, p. 121, 2013.

[203] G. Erten and F. Salam, "Hybrid analog/digital vlsi chip for template matching," in *Proc. IEEE Midwest Symposium on Circuits and Systems*, vol. 2, 1994, pp. 868–871.

[204] N. Ranganathan and S. Venugopal, "A vlsi chip for template matching," in *Proc. IEEE International Conference on Computer Design: VLSI in Computers and Processors*. IEEE, 1994, pp. 542–545.

[205] O. Takahiro, H. Morita, and A. van Wijngaarden, "Real-time and memory-efficient arrhythmia detection in ecg monitors using antidictionary coding," *IEICE Trans. Fundamentals of Electronics, Communications and Computer Sciences*, vol. 96, no. 12, pp. 2343–2350, 2013.

[206] Z. Yang, Q. Zhao, and W. Liu, "Improving spike separation using waveform derivatives," *Journal of neural engineering*, vol. 6, no. 4, pp. 2–12, 2009.

[207] Z. Yang, T. Chen, and W. Liu, "A neuron signature based spike feature extraction algorithm for on-chip implementation," in *Proc. IEEE International Conference of the Engineering in Medicine and Biology Society*, 2008, pp. 1716–1719.

[208] H. Chan, M. Lin, Y. Wu, H. Lai, S. Lee, J. Yang, and S. Fang, "Detection and characterization of neural spikes," in *Proc. IET Conference on Advances in Medical, Signal and Information Processing*, 2006, pp. 1–4.

[209] R. Quiroga, "last accessed: November 2012," *www.vis.caltech.edu/rodri/Wave_clus/Wave_clus_home.htm*, 2006.

[210] J. Lian, G. Garner, D. Muessig, and V. Lang, "A simple method to quantify the morphological similarity between signals," *Signal Processing*, vol. 90, no. 2, pp. 684–688, 2010.

[211] T. Schreiber and A. Schmitz, "Surrogate time series," *Physica D: Nonlinear Phenomena*, vol. 142, no. 3, pp. 346–382, 2000.