AN UNSUPERVISED AND REFRACTORINESS-SUPPORTED ALGORITHM DESIGN FOR REAL-TIME SPIKE SORTING

by

Alparslan Önder

B.S., in Biomedical Engineering, Izmir Katip Çelebi University, 2020

Submitted to the Institute of Biomedical Engineering
in partial fulfillment of the requirements
for the degree of
Master of Science
in
Biomedical Engineering

Boğaziçi University
2023

AN UNSUPERVISED AND
REFRACTORINESS-SUPPORTED
ALGORITHM DESIGN FOR REAL-TIME SPIKE SORTING

APPROVED BY:

Prof. Dr. Ahmet Ademoğlu (Thesis Advisor)	.....
Prof. Dr. Burak Güçlü (Thesis Co-advisor)	.....
Prof. Dr. Albert Güveniş	.....
Assoc. Prof. Dr. Esin Öztürk Işık	.....
Assoc. Prof. Dr. Murat Okatan	.....

DATE OF APPROVAL: 30 May 2023

ACKNOWLEDGMENTS

I would like to express my deepest gratitude to Prof. Dr. Ahmet Ademoğlu and Prof. Dr. Burak Güçlü for their invaluable guidance, patience, support, and encouragement throughout the entire process. Their expert feedback and insights have been instrumental in shaping this thesis.

I am also grateful to my loving family and my girlfriend Süreyya for their unwavering support and encouragement during the ups and downs of my academic journey. Their love and belief in me have been a constant source of motivation and inspiration.

Finally, I would like to take a moment to remember and acknowledge the lives lost in the devastating earthquake that struck our country during the course of my studies. My thoughts and condolences are with their families and loved ones during this difficult time.

ACADEMIC ETHICS AND INTEGRITY STATEMENT

I, Alparslan Önder, hereby certify that I am aware of the Academic Ethics and Integrity Policy issued by the Council of Higher Education (YÖK), and I fully acknowledge all the consequences due to its violation by plagiarism or any other way.

Name :

Signature:

Date:

ABSTRACT

AN UNSUPERVISED AND REFRACTORINESS-SUPPORTED ALGORITHM DESIGN FOR REAL-TIME SPIKE SORTING

Neural spike sorting algorithms have been used to group the action potentials in electrophysiological signals according to their characteristics to use them in neuroscience studies. In this study, we developed an unsupervised and online spike sorting algorithm that performs better than existing online and unsupervised sorting algorithms, calculates thresholds variably, and can be used for real-time studies when applied on FPGA boards. A template matching algorithm OSort, based on Euclidean distance, functioned as the base for this algorithm. We used the NEO method to detect the spikes and the Windowed Sinc Interpolation method to upsample the detected spikes four times. We developed a refractoriness control mechanism that works according to the peak points of the spikes to prevent assigning a spike to the wrong cluster. We designed a second block that considers the probability of a spike belonging to the second closest cluster to itself. It controls this issue to prevent assigning a spike to the wrong cluster if the spike waveforms of the different neurons are similar. We avoided using more complex mathematical operations like calculating standard deviation with the aim of future real-time studies on cheaper FPGA boards. When we compared the performance of our algorithm with the well-known algorithms in the literature, we saw that ours performed significantly better than the online ones and insignificantly worse than offline ones. The presented thesis shows that more accurate online and unsupervised spike sorting is possible without complex algorithms. It is expected that this kind of algorithms will support the increment in the number of neuroscience studies that use spike sorting.

Keywords: Spike Sorting, MATLAB, Action Potentials, Detection, Clustering, Online, Unsupervised, Varying, Real-time

ÖZET

AKSİYON POTANSİYELLERİNİN GERÇEK ZAMANLI KÜMELENMESİ İÇİN GÖZETİMSİZ VE REFRAKTERLİK DESTEKLİ BİR ALGORİTMA TASARIMI

Elektrofizyolojik sinyallerin içerisindeki aksiyon potansiyellerini, sinirbilim çalışmalarında kullanılabilmeleri için kümeleyen algoritmalara spayk kümeleme algoritmaları denir. Bu çalışmada, mevcut çevrimiçi ve gözetimsiz algoritmalardan daha iyi performans gösteren, eşikleri uyarlamalı olarak hesaplayan ve FPGA kartlara uygulandığında gerçek zamanlı çalışmalarda kullanılabilecek gözetimsiz ve çevrimiçi bir spayk kümeleme algoritması geliştirilmiştir. Bu algoritma geliştirilirken, Öklid mesafesine dayalı, şekil eşleştirme tipi bir algoritma olan OSort temel alınmıştır. Spaykların yakalanması için NEO metodu; yakalananların da dört defa üst örneklenmesi için Pencereli Sinc İnterpolasyon yöntemi kullanılmıştır. Spaykların yanlış kümelenmesini önlemek için spaykların tepe noktalarını baz alan bir refrakterlik kontrol mekanizması geliştirilmiştir. Farklı nöronlara ait spayk şekillerinin benzer olduğu durumlarda spaykların yanlış kümelenmesini önlemek için, bir spaykın kendisine ikinci en yakın kümeye ait olma ihtimalini kontrol eden bir blok tasarlanmıştır. Algoritmanın gelecekte, uygun fiyatlı FPGA kartlarla, gerçek zamanlı çalışmalarda kullanılabilmesi için matematiksel olarak kompleks işlemlerin kullanımından kaçınılmıştır. Algoritmanın, literatürdeki popüler çevrimiçi algoritmalardan istatistiksel olarak anlamlı ölçüde iyi performans gösterdiği ve çevrimdışı olanlardan istatistiksel olarak anlamsız ölçüde kötü performans gösterdiği görülmüştür. Sunulan bu tez, kompleks algoritmalar kullanmadan da spaykları yüksek doğrulukla, çevrimiçi ve gözetimsiz şekilde kümelemenin mümkün olduğunu göstermiştir. Bu tür algoritmaların, spayk kümeleme uygulamalarının kullanıldığı sinirbilim çalışmalarının yaygınlaşmasını desteklemesi beklenmektedir.

Anahtar Sözcükler: Spayk Kümeleme, MATLAB, Aksiyon Potansiyelleri, Yakalama, Sıralama, Çevrimiçi, Gözetimsiz, Uyarlamalı, Gerçek Zamanlı

TABLE OF CONTENTS

ACKNOWLEDGMENTS . . . iii
ACADEMIC ETHICS AND INTEGRITY STATEMENT . . . iv
ABSTRACT . . . v
ÖZET . . . vi
LIST OF FIGURES . . . ix
LIST OF TABLES . . . x
LIST OF SYMBOLS . . . xii
LIST OF ABBREVIATIONS . . . xiii
1. INTRODUCTION . . . 1
1.1 Motivation and Aim . . . 1
1.2 Outline . . . 3
2. BACKGROUND . . . 4
2.1 Nervous System and The Brain . . . 4
2.2 Neural Spike Signals . . . 5
2.3 Spike Sorting Algorithms . . . 7
2.4 Algorithm Design Platforms for the Spike Sorting . . . 9
2.5 Hardware for the Spike Sorting . . . 9
3. MATERIALS AND METHODS . . . 11
3.1 Osort Algorithm . . . 11
3.2 Algorithm Used in This Study . . . 13
3.2.1 Spike Detection . . . 13
3.2.2 Spike Realignment . . . 15
3.2.3 Spike Sorting . . . 15
3.2.4 Cluster Merging . . . 19
3.2.5 Cluster Pruning . . . 20
3.2.6 Control Block . . . 20
4. RESULTS . . . 22
4.1 Performance Comparison . . . 22
4.1.1 Dataset I . . . 23

4.1.2 Dataset II 24
4.1.3 Dataset III 25
4.2 Comparison with Other Algorithms 26
5. DISCUSSION 30
5.1 Previous Studies 30
5.2 Technical Limitations 32
5.3 Conclusion and Future Prospects 33
APPENDIX A. ADDITIONAL ALGORITHMS 35
APPENDIX B. ADDITIONAL TABLES 40
REFERENCES 56

LIST OF FIGURES

Figure 2.1	General anatomical view of the human nervous system.	4
Figure 2.2	Lobes of the cerebrum of the human brain [1].	5
Figure 2.3	Bipolar neuron and its parts [1].	6
Figure 2.4	Reproduction drawing of the first published intracellular action potential waveform by Hodgkin and Huxley in 1939 [1].	7
Figure 3.1	Reproduction of flow chart drawing of the sorting part of the *OSort* [2].	12
Figure 3.2	Demonstration of the absolute and relative refractory periods [3].	17
Figure 3.3	Flowchart of the sorting block of the algorithm designed in this study.	19
Figure 3.4	The control part of the flowchart of the algorithm designed in this study is shown with dashed lines.	21
Figure 4.1	Spike waveforms used in the first dataset.	23
Figure 4.2	Spike waveforms used in the second dataset.	25
Figure 4.3	Spike waveforms used in the third dataset.	26
Figure 4.4	The means of the $F1$ scores of different algorithms for simulated probabilistic datasets with varying signal-to-noise ratios. Solid fills show the online, and dotted fills show the offline algorithms.	28

LIST OF TABLES

Table 4.1	$F1$ score comparisons of OSort [2] and the algorithm designed in this study for the first dataset.	24
Table 4.2	$F1$ Score comparisons of OSort [2] and the algorithm designed in this study for the second dataset.	25
Table 4.3	$F1$ score comparisons of OSort [2] and the algorithm designed in this study for the third dataset.	26
Table 4.4	Results of the two way ANOVA test.	29
Table B.1	The spike sorting performance of the OSort [2] algorithm and algorithm designed in this study for the first dataset.	40
Table B.2	The spike sorting performance of the OSort [2] algorithm and algorithm designed in this study for the second dataset.	41
Table B.3	The spike sorting performance of the OSort [2] algorithm and algorithm designed in this study for the third dataset.	42
Table B.4	Results of the two way ANOVA test for dataset of OSort.	43
Table B.5	The multiple comparison table from the two-way ANOVA's Tukey's HSD post-hoc test for OSort's dataset [2].	43
Table B.6	The multiple comparison tables from the two-way ANOVA's Tukey's HSD post-hoc test for comparison with other algorithms.	44
Table B.7	The number of spikes of each waveform in every simulated signal trace.	45
Table B.8	The sorting results of all algorithms for simulation 1.3_3.	46
Table B.9	The sorting results of all algorithms for simulation 1.7_4.	47
Table B.10	The sorting results of all algorithms for simulation 2.6_5.	48
Table B.11	F1 score, precision, and recall values for Wave Clus [4] based spike sorting algorithm.	49
Table B.12	F1 score, precision, and recall values for OSort [2] based spike sorting algorithm.	50
Table B.13	F1 score, precision, and recall values for our algorithm.	51

Table B.14 F1 score, precision, and recall values for Skew-t Distribution [5] based spike sorting algorithm. 52
Table B.15 F1 score, precision, and recall values for Gaussian Mixture Models [6] based spike sorting algorithm. 53
Table B.16 F1 score, precision, and recall values for K-Means [7] based spike sorting algorithm. 54
Table B.17 F1 score, precision, and recall values for T-distribution [8] based spike sorting algorithm. 55

LIST OF SYMBOLS

t	Time
$f(t)$	Filtered signal
$\bar{f}(t)$	Running average of f
$P(t)$	Local energy or power of the filtered signal
$\bar{P}(t)$	Running average of P
C_{th}	Threshold constant of the OSort
$Th_{det}(t)$	Detection threshold of the OSort
$Th_{cont}(t)$	Detection control threshold of the OSort
$Th_{sort}(t)$	Spike sorting threshold of the OSort
$\overrightarrow{S}$	Spike waveform
$\overrightarrow{R}$	Upsampled and Realigned Spike
D	Euclidean distance between two spike waveforms
DD	Euclidean distance between two clusters' average waveforms
Th_{neo}	NEO spike detection threshold
C_{neo}	NEO constant
N	Number of the data points in a spike waveform
$Thr_{sort}(t)$	Spike sorting threshold of our algorithm
$Thr_{cls}(t)$	Closeness control threshold of our algorithm
t_{peak}	Time of the peak of the spike waveform
t_{diff}	Difference of (t_{peak}) values of two spike waveforms
D_{diff}	Difference between the two smallest D values of a spike
M	Average waveform of a cluster

LIST OF ABBREVIATIONS

FPGA	Field Programmable Gate Array
MATLAB	Matrix Laboratory
OSort	Online Sorting
NEO	Non-linear Energy Operator
Wave Clus	Wavelets Clustering
GMM	Gaussian Mixture Models
PCA	Principle Component Analysis
SNR	Signal to Noise Ratio
ASIC	Application Specific Integrated Circuit
HDL	Hardware Description Language
BNN	Binarized Neural Networks
CL	Competitive Learning
CNN	Convolutional Neural Network
ANOVA	Analysis of Variance

1. INTRODUCTION

1.1 Motivation and Aim

Spike sorting refers to processing the raw neurophysiological signals to capture the action potentials coming from different neurons and assigning them to distinct classes. The spikes of different neurons have unique shapes due to the branching and connections of neurons. However, the position of the recording electrode is the most essential factor to consider. The difference in spike waveforms between them becomes apparent only when two different neurons are placed at an equal distance from the electrode. If the distances are different, the most noticeable difference between the waveform shapes will be that one neuron's amplitude is greater than the other, and only the amplitude difference can be used as the basis for sorting [9]. Using the spike waveforms, researchers extract information about interactions among neurons and use this information in several situations, such as diagnosing neuromuscular or neurological disorders.

Most spike sorting algorithms consist of four main steps. i) Filtering is the first step in which the raw signal is filtered through 300 Hz and 3000 Hz to eliminate local field potentials and other signals to isolate the action potentials. ii) The action potentials are detected in the filtered signal. iii) Feature extraction follows the detection, which is about determining the features of waveforms for sorting if needed. iv) Finally, action potentials are sorted using their extracted features or waveform properties [10].

Spike sorting algorithms vary depending on features such as supervised or unsupervised, automatic or manual, online or offline, and real-time or non-real-time. Researchers who do not have any programming skills may use automatic and unsupervised algorithms during their neuroscience evaluations. Even though offline sorting algorithms generally perform better than online ones, the latter are much faster than the former. The real-time algorithms also provide an immense data reduction, which

is important for neuroscience studies that perform recordings extending to hours.

In this study, we aim to design an online, unsupervised, and computationally cost-effective algorithm that calculates all thresholds automatically and variably, which performs better than current online algorithms and can be used for real-time applications when applied to the FPGA boards in the market. However, this design has not been implemented on an FPGA in this thesis. We design the algorithm in the MATLAB platform. The algorithm is based on a well-known online and unsupervised sorting method called OSort [2], which is a Euclidean distance based template matching algorithm. We replace its detection procedure with a non-linear energy operator (NEO) [11] since it is easier to apply on FPGAs, computationally cost-effective, and provides better detection. To upsample detected spikes, we prefer a Windowed Sinc Interpolation [12, 13] based upsampling method, which increases sorting accuracy and is usable on FPGAs for real-time applications [14]. In order to reduce the misclassification, we check the refractoriness of a spike whether it is formed in the absolute refractory period of the previous spike assigned to a particular cluster. As a second feature, we propose a unit that checks the difference between the Euclidean distances of a spike to its two closest clusters. If the difference is smaller than a threshold, the algorithm applies a second check to this spike after a period. This feature decreases the probability of assigning a spike to the wrong cluster if the spike waveforms of different neurons are similar. Unlike the original OSort [2], which uses the standard deviation of the streaming signal for spike sorting threshold calculation, we use the variance of the streaming signal. This way, we can eliminate the standard deviation calculations, which make the algorithm computationally expensive. We also improve the cluster merging and pruning stages for better sorting.

We simulate three datasets with different average signal-to-noise ratios, each with twenty probabilistic signal traces. With those datasets, we simulate both online and offline algorithms in an offline platform. When we analyze those signals with our algorithm and compare the results with online and offline algorithms, our results are significantly better than a well-known online sorting algorithm OSort [2] and a well-known offline sorting algorithm Wave Clus [4]. On the other hand, it seems slightly

inferior to $K-$Means sorting [7], t-distributions sorting [8], and Gaussian mixture models (GMM) based sorting [6] offline algorithms.

The thesis shows that it is possible to sort neural spikes accurately with an online, unsupervised, and adaptive algorithm based on a simple method enhanced with basic features. Hopefully, it will provide easy usage for users without programming knowledge.

1.2 Outline

The outline of the thesis is as follows; in the first chapter, we describe our motivation and aim in this study. In Chapter 2, we offer background information about the nervous system and its relation to neural spikes, spike sorting algorithms, design platforms for software and hardware. The details of the spike sorting algorithm proposed in this study are described in Chapter 3. In Chapter 4, a comparison of the proposed algorithm with some other online and offline algorithms available is presented. In Chapter 5, we offer conclusions of our results with respect to previous studies, technical limitations, and future prospects.

2. BACKGROUND

2.1 Nervous System and The Brain

The nervous system works to coordinate the senses and actions of the body by transmitting electrical and chemical signals. Functionally, the nervous system has two parts: the sensory system, which collects and processes the information in the environment, and the motor system, which generates responses to the information collected by the sensory system [15]. On the other hand, it is anatomically divided into two parts; the central and the peripheral nervous systems. The brain and spinal cord are parts of the central nervous system; the cranial and spinal nerves comprise the peripheral nervous system [16]. The cranial nerves enter the CNS at the level of the brainstem and act as a relay for the signals coming from different body parts [17]. Secondly, spinal nerves extend from the vertebrae to the organs to receive sensory messages and transfer the brain response to these messages [18].

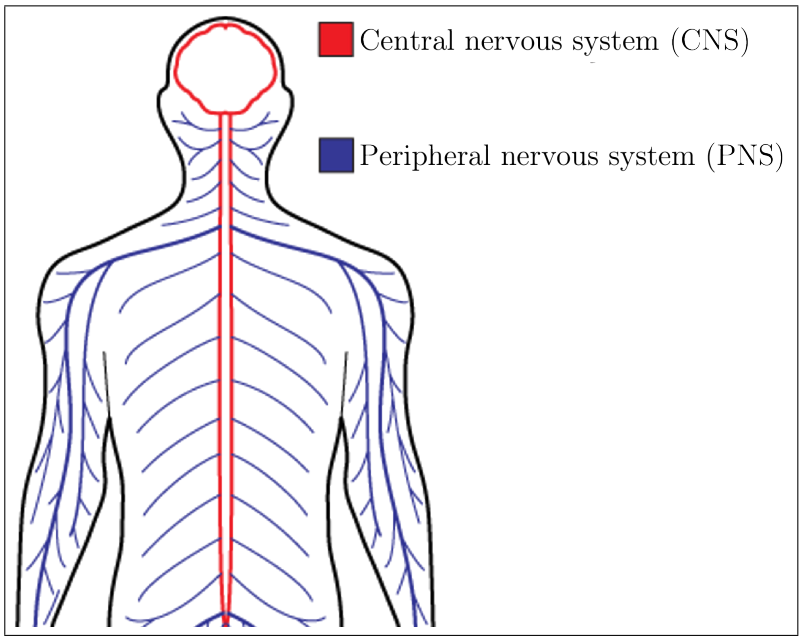

Figure 2.1 General anatomical view of the human nervous system.

The transfer of motor and sensory impulses between the brain and body, as well as the production of some reflexes, is carried out via the spinal cord, a long link made

up of nerve tissues that runs from the brain stem to the lumbar area of the body in the vertebral bones [19].

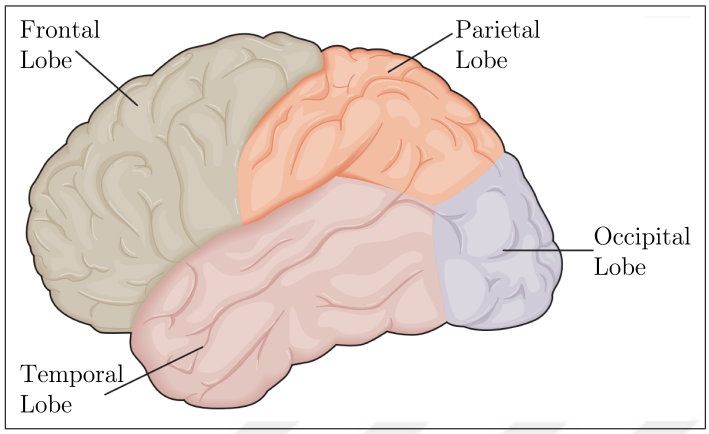

Figure 2.2 Lobes of the cerebrum of the human brain [1].

Neurons are the building blocks of the brain, both functionally and structurally [20]. The number of neurons in an adult human body may reach over 100 billion, weighing around 1300 grams [15]. Even though the spinal cord has three different types of neurons (sensory, motor, and interneurons), the brain has a wide variety of neurons that vary according to their properties, like shape and function. In general, most neurons are formed from three different parts; the cell body, the axon, and the dendrites. The cell body is the place where the DNA and RNA are located and proteins are produced. Just like any other system, neurons should also work in synchrony, in order to generate signals to communicate with others. Those signals are micro-level electrical fluctuations that are called action potentials. Generally, the axon of one neuron produces and carries action potentials to another neuron's dendrite, which is the signal receiver site of a neuron [1].

2.2 Neural Spike Signals

Action potentials - also called neural spikes - are the signals of communication for the neurons. The membrane potentials of neurons' cell bodies are the source of neural spikes. The axon of a neuron generates spike signals at its end towards its cell body - a point called the initial segment- and is able to carry spike signals from 0.1 mm

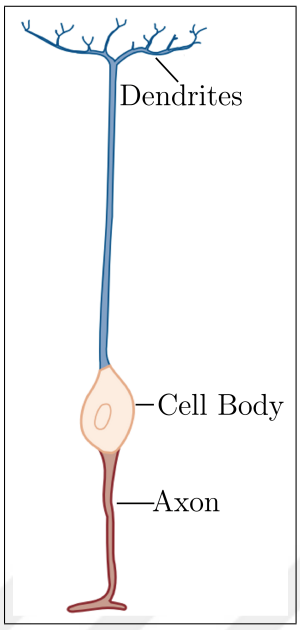

Figure 2.3 Bipolar neuron and its parts [1].

to 2 m without any signal reduction thanks to its physical property. The lipid-based sheath that surrounds some parts of the axons, called the myelin, contributes to this feature which enables the axons to transmit action potentials faster. The amplitude of a spike signal may vary from 70 mV to 110 mV, nearly ten times higher than receptor and synaptic potentials. Its overall duration may vary from 1 ms to 10 ms.

Another unique feature of action potentials is the all-or-none rule. If the membrane of a neuron's cell body becomes electrochemically more negative, which is called hyperpolarization, the neuron's cell body will not produce any spike signal. A spike can be generated only if the membrane gets electrochemically more positive, called depolarization. However, depolarization should reach to a level called threshold potential to enable the cell body to produce a spike signal. No matter how much depolarized the membrane is over the threshold potential, action potentials have the same amplitude for the same neuron, and this principle is called the all-or-none rule [1].

It is possible to detect and measure the spike signals with electrophysiological recordings. The best way to record an action potential is the extracellular recording by placing an electrode close to the region under investigation. Those recordings have been

used in studies for studying different kinds of cognitive disorders, such as Alzheimer's Disease [21].

Luigi Galvani discovered in 1791 that the nervous system could be excited by electrical currents. He applied a tiny current to a frog's leg muscle and observed muscle contractions [22]. Hodgkin and Huxley were the first researchers who published a paper about recorded intracellular action potential waveforms in 1939 [1]. Hubel and Wiesel analyzed the action potentials to investigate the activity of the visual signal-processing mechanism of the brain in 1977 [23]. Generally, single-channel electrodes have been used to record action potentials. However, with the current technology, we can record thousands of neurons simultaneously using high-density multi-electrode arrays, which is constantly increasing with the advent of new technologies. On the other hand, processing these recordings is becoming more demanding as well.

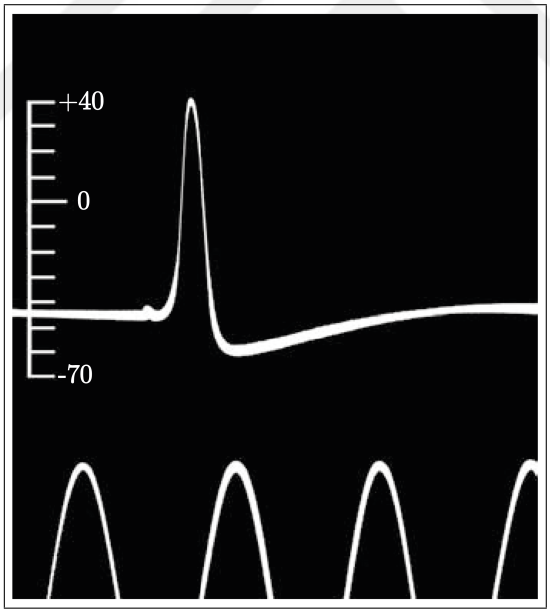

Figure 2.4 Reproduction drawing of the first published intracellular action potential waveform by Hodgkin and Huxley in 1939 [1].

2.3 Spike Sorting Algorithms

Spike sorting is identifying the particular neuron to which a spike waveform belongs by analyzing the raw neurophysiological signal. Shapes of the action potentials

are specific to the neurons from which they originate [9]. Typically, a spike sorting process starts with signal filtering. Most of the current data acquisition systems record the extracellular signals and filter the signal with analog causal infinite impulse response filters between 0.3 Hz and 7500 Hz. An additional digital bandpass filtering is applied with a 300Hz-3000Hz frequency band, which retains action potentials and eliminates the other irrelevant components [24].

The second step is to detect the spikes which are embedded in noise. Generally, spike signals can be easily seen by the eye in the signal trace due to their high amplitude peaks. However, detecting spike signals by visual inspection is time-consuming. The easiest way to threshold the signal is using a constant amplitude threshold, but it may have a high number of wrong detections. For that reason, varying thresholds are used in solutions employing different mathematical approaches. Some methods determine the threshold using the standard deviation of the noise signal in a time interval, while others use that of the filtered signal. Still, some other applications use standard deviations of both noise and filtered signal [25]. Continuous wavelet transform thresholding [26] and thresholding with fuzzy decision-making algorithms [27] are other approaches that give better results, but they are computationally more expensive. Non-linear energy operator based thresholding is another alternative with low mathematical complexity and high efficiency [11].

Feature extraction comes after the spike detection, for selecting features to sort spikes. Several applications use waveform features of spikes signals such as their width, peak time point, and peak amplitude, but they may not be sufficient [28]. Wavelet analysis is another feature extraction approach that produces the bandpass-filtered versions of spike signals using the wavelets [29]. Principle component analysis (PCA) is a common feature extraction method that uses principle components as features for spike sorting [30]. Template matching is a simple and computationally cost-effective method. It measures the shape of the spike waveform using different metrics as the Euclidean distance metric [31]. The last step of spike sorting is clustering which assigns spikes individually to different neurons using their extracted features. Sometimes extra steps can help increase the process's efficiency, such as pruning the irrelevant clusters.

While experts are developing new technologies for electrophysiological signal detection systems, advances in sorting algorithms are emerging in parallel. There are studies focusing on spike sorting techniques for higher accuracy which often come with higher computational complexity [32]. However, another major focus is to design automatic, online, real-time, and easy-to-use algorithms to perform faster experiments with reduced memory for saving the recordings. Even professionals unfamiliar with sorting techniques may use this kind of algorithms in their studies. The OSort [2] is one of the most known online spike-sorting algorithms; it uses the Euclidean distance to sort spikes, but its accuracy is not as high as offline algorithms.

2.4 Algorithm Design Platforms for the Spike Sorting

There are several coding platforms available for an algorithm to be designed. The *OSort* [2] algorithm was designed in MATLAB. On the other hand, Python programming language is a free alternative to MATLAB. Although there are no predefined blocks of codes that make its users to run faster, there are various free code libraries in Python for neural spike sorting like SpySort [33].

2.5 Hardware for the Spike Sorting

In order to use spike sorting algorithms during experiments, we should implement them to hardware platforms and optimize them for efficiency and accuracy. There are several hardware options to run these algorithms, and the most common are microprocessors. However, they have fixed architecture that cannot be trimmed for different applications. Performing simple operations sometimes may take many clock cycles, and only a few calculations are performed simultaneously [34].

On the other hand, Application-Specific Integrated Circuits (ASIC) and Field Programmable Gate Arrays (FPGA) have architectural designs that can be customized

to the algorithm and can operate 10 to 100 calculations in parallel [35]. These properties make FPGAs and ASICs better hardware solutions for spike-sorting algorithms. As its very name signifies, ASICs are designed for specific applications, and their designs cannot be changed. Therefore, they are not reasonable solutions for algorithm reconfigurations. Designing ASICs is also expensive. Prototyping an algorithm on ASICs is not possible since they are not reconfigurable [36].

FPGAs are reconfigurable hardware with predefined logic units which are cost-effective. FPGAs may be the best alternative for algorithm design and prototyping on hardware. A programming language called HDL (Hardware Description Language) is specific to FPGA programming. The algorithms designed in different platforms or programming languages like MATLAB or Python are easily applicable to FPGAs with different softwares designed to convert those codes written in different languages to HDL [36]. Even without knowing HDL, algorithms can be designed on platforms like MATLAB Simulink [37], or Labview [38].

Many different spike sorting algorithms are applied on FPGA boards in the literature [39, 40]. Due to the flexibility property of FPGAs, many different combinations of algorithms have been developed. For instance, some researchers used the FPGAs for real-time spike sorting operations [40, 41], and some used them to shorten the need for processing time for offline algorithms that process recordings longer than a few hours [39].

3. MATERIALS AND METHODS

3.1 Osort Algorithm

The OSort [2] is an online algorithm designed in MATLAB programming platform and can be applied to hardware for real-time signal processing due to its low computational requirements. Like other methods in the literature, this algorithm starts with bandpass filtering the signal using a fourth-order Butterworth filter between 300Hz and 3000Hz.

For the detection of neural spikes, a varying threshold is calculated. If the filtered signal is $f(t)$, its moving average calculated over a window of length n samples usually corresponding to one minute is $\bar{f}(t)$ (Eq. 3.3) and its power signal is $P(t)$ (Eq. 3.2). The varying threshold for spike detection $Th_{det}(t)$ is calculated by using the mean power, the standard deviation of $P(t)$ and a constant C_{th} as given in Eq. 3.4. Users may change C_{th} between 3 to 5 depending on the noise level in the signal.

$$\bar{f}(t) = \frac{1}{n}\sum_{i=1}^{n} f(t-i) \tag{3.1}$$

$$P(t) = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(f(t-i) - \bar{f}(t))^2} \tag{3.2}$$

$$\bar{P}(t) = \frac{1}{n}\sum_{i=1}^{n} P(t-i) \tag{3.3}$$

$$Th_{det}(t) = \bar{P}(t) + C_{th} * std(P(t)) \tag{3.4}$$

If local energy exceeds a precalculated varying threshold at a certain time, a signal trace containing 24 before and 39 after the marking is extracted, yielding a 64-point waveform in total. A second check is applied to determine the exact peak point

of the potential spike by using a threshold $Th_{cont}(t)$. If the absolute maximum value of the spike is higher than the threshold, it is selected or otherwise discarded.

$$Th_{cont}(t) = 2 \cdot std(f(t)) \tag{3.5}$$

Before the realignment process, the potential spike signal trace is upsampled four times to 256 data points for a more accurate Euclidean distance comparison. Detected and checked signals are realigned according to their absolute maximum point. If spikes are not realigned successfully, their Euclidean distances can not be calculated accurately, and the sorting process may fail. After realignment, the spike is ready for sorting.

The spike sorting and cluster merging thresholds used in OSort are equal. This threshold is multiplied by a burst correction factor as given in Eq. 3.6.

$$Th_{sort}(t) = 1.2 \cdot (std(f(t)))^2 \tag{3.6}$$

At the start, the algorithm assigns the first spike to an empty cluster. If the spike

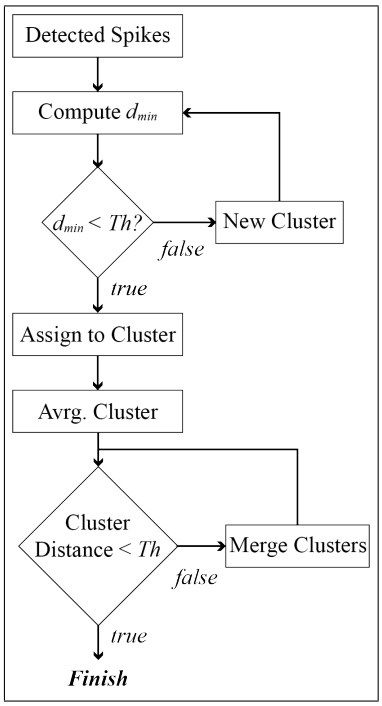

Figure 3.1 Reproduction of flow chart drawing of the sorting part of the *OSort* [2].

under investigation is $\overrightarrow{S}_i(k)$ and the mean waveform of a cluster is $M_j(k)$, N being

the number of the data points, its squared Euclidean distance to class j is D_{ij} and is calculated as in Eq. 3.7. If the minimum of the calculated distances is less than the threshold, the algorithm assigns the spike to the cluster that has minimum distance; else, it assigns it to a new cluster. After every spike assignment, the mean waveform of the cluster is recalculated using the last hundred assigned spikes added to this cluster.

$$D_{ij} = \sum_{k=1}^{N} (\overrightarrow{S}_i(k) - M_j(k))^2 \tag{3.7}$$

After assigning each spike, the algorithm updates the distance between the assigned cluster and other clusters using Eq. 3.7, which is similar to calculating the Euclidean distance between two spike waveforms.

If the distance of a cluster to another one is less than the same threshold without the burst factor, the algorithm merges them. Clusters with few spikes are merged with their closest clusters. The clusters with less than two spikes are pruned, and their spikes are assigned as noise signals for every thousand spikes processed. The process terminates after exhausting the last detected spike signal [2].

3.2 Algorithm Used in This Study

The algorithm used in this study was based on OSort [2]. The accuracy was enhanced by introducing several checking mechanisms. The algorithm was divided into six major blocks: detection, realignment, sorting, merging, pruning, and control block.

3.2.1 Spike Detection

Like in all spike sorting algorithms, we started by filtering the raw signal between 300Hz and 3000Hz using a fourth-order Butterworth filter. In OSort, a varying threshold was to be determined using Eq. 3.3, 3.2, and 3.4; which involves computing

the standard deviation of the filtered signal, which is hard to apply in hardware since square rooting is not available in every FPGA board [42]. Even though it is possible to compute the square root on FPGAs, it increases the computational cost.

In the original algorithm, if the local energy signal $P(t)$ exceeds the detection threshold $Th_{det}(t)$, a segment including 24 data points before and 39 after is extracted as a candidate spike signal. The algorithm checks this candidate spike signal against another threshold $Th_{cont}(t)$ calculated as given in Eq. 3.5. If the absolute maximum point of the potential spike signal exceeds this threshold, this point is determined as the peak point of the spike signal, and the detected signal is rearranged to be 24 data points before the peak and 39 data points after. Else, the candidate signal trace is classified as a noise signal. This second threshold is used to find the exact peak point of the spike waveform to realign it accurately. However, Double-checking increases the computational cost and reduces hardware efficiency.

Due to these reasons, we preferred to use a non-linear energy operator(NEO) based spike detection algorithm [11]. Many spike sorting algorithms use a NEO-based detection algorithm due to its simplicity and high efficiency [43]. We calculated the detection threshold in a six-second sliding window. If $x(t)$ is the signal, the non-linear energy operator is given in Eq. 3.8.

$$\psi(x(t)) = (\frac{dx(t)}{dt})^2 - x(t)(\frac{d^2x(t)}{dt^2}) \tag{3.8}$$

We can convert Eq. 3.8 into discrete-time signal as in Eq. 3.9.

$$\psi(x[n]) = x^2[n] - x[n+1] \cdot x[n-1] \tag{3.9}$$

The threshold Th_{neo} was calculated as in Eq. 3.10. C_{neo} was a constant for the NEO-based detection threshold, which was chosen as 8 in our case. N was the number of the NEO points used for calculation. In our application, 150,000 data points of a signal trace, simulated at the 25 kHz sampling frequency, were used with 6 seconds

long sliding windows.

$$Th_{neo} = \frac{C_{neo}}{N} \sum_{n=1}^{N} (\psi(x[n])) \tag{3.10}$$

If the calculated NEO value exceeded the NEO threshold at some time point, 24 data points before and 39 data points after this time point (64 data points total) were extracted as a spike waveform. The pseudo-code of the NEO-based detection block and its demonstration is given in Appendix A (Algorithm 1).

3.2.2 Spike Realignment

The spike realignment is one of the crucial steps, especially for template matching-based neural spike sorting algorithms. Since the OSort is also a template matching-based algorithm, it is a critical stage in our method as well. When there is a need for upsampling the data, it should be done before realignment. We upsampled the detected spike signal four times to enhance the accuracy of the Euclidean distance metric. Upsampling is followed by spike realignment. In order to compare the spike signals according to their wave shapes, the waveforms should be faithfully realigned for template-based spike sorting. The OSort algorithm takes the 95th datapoint as the absolute maximum of the upsampled spike signal trace. If this point is not the maximum, the spike signal is shifted so that the 95th data point corresponds to the maximum value. The pseudo-code of the upsampling and realignment function of OSort algorithm is shown in Appendix A (Algorithm 2).

3.2.3 Spike Sorting

The sorting block is the main part of the spike sorting algorithm. As defined in Section 3.1, OSort is a Template-matching based algorithm in which the Euclidean distance between the data points of two different spike waveforms is calculated with the residual-sum-of-squares method as given in Eq. 3.7. The algorithm developed in this study was based on the same principle. The sorting threshold $Th_{sort}(t)$ of OSort was

calculated by taking the square of the filtered signal's standard deviation in a sliding window, as in Eq. 3.6. Due to the reasons explained in Section 3.1, we do not prefer to calculate the standard deviation because of its complication on FPGA boards. For every detected spike, the OSort algorithm calculates the distances of the spike signal to the mean waveforms of all clusters. If the minimum of those signals is more than the threshold $Th_{sort}(t)$, it generates a new cluster and assigns spikes to it; else assigns the spike signal to the closest cluster. The mean waveform of the cluster that the last spike assigned is recalculated with the last hundred assigned spikes in this cluster. Appendix A (Algorithm 3) shows the pseudo-code of the sorting part of the OSort algorithm.

The sorting algorithm that we propose uses two additional features for improved sorting. The first one is related to the refractoriness of the action potentials. A refractory period is defined as the time that elapses for a neuron to be capable of producing a second action potential [44]. For neurons, there are two different refractory periods, one is the absolute refractory period, and the other one is the relative refractory period. The absolute refractory period takes about 1 to 2 ms during which a second action potential signal cannot be produced in this period. The relative refractory period comes after the absolute one and generally takes about 4ms or more, depending on the neuron. In this stage, the neuron can fire another action potential, but it is less likely than firing at the resting state [3]. Considering the absolute refractory period while sorting the spikes signal increases the sorting accuracy. Due to this reason, we preferred to use refractoriness as an additional feature in sorting.

In general, the clustering methods like the OSort are called hard clustering methods because they consider a spike waveform can belong to only one neuron [45]. Especially at the beginning of the sorting process, if the algorithm assigns a spike to a wrong cluster, the mean waveform of this multiunit cluster will lead to other wrong sortings. For instance, a spike may be assigned to the closest cluster even if it belongs to the second closest cluster due to the template of a multiunit cluster. In the end, this snowball effect may cause the failure of the whole algorithm. To deal with this problem, we also considered the second closest cluster for the spike while sorting.

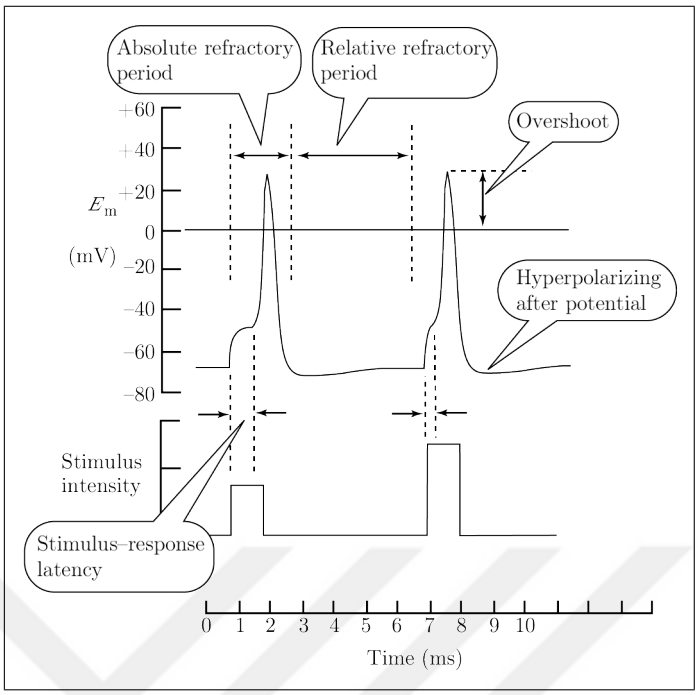

Figure 3.2 Demonstration of the absolute and relative refractory periods [3].

.

We used two different thresholds calculated in the same way in the sorting block of our algorithm. The first one Thr_{sort} was calculated with the filtered signal $f(t)$ variance in a one-minute moving window and multiplied by 1.2 as a burst factor, as given in Eq. 3.11. This was the main threshold used for spike sorting.

$$Thr_{sort}(t) = 1.2 \cdot var(f(t)) \tag{3.11}$$

However, a second threshold was used as in Eq. 3.12 while comparing the difference between the Euclidean distances of a spike to the first and second closest clusters.

$$Thr_{cls}(t) = \frac{Thr_{sort}(t)}{10} \tag{3.12}$$

Additionally, we used the refractory period as 1 ms for a second check after determining the time difference between the peaks of consecutive spikes.

The sorting block of our algorithm assigned the first coming spike to an empty cluster. For the second spike, it calculated the Euclidean distance D between the mean waveform of the first cluster -which was equal to the first spike in this case- and the

second spike, with the formula in Eq. 3.7. If calculated D was more than the Thr_{sort}, this spike was assigned to a new cluster. Else, the algorithm calculated the difference between the times of the peak points of the spike signal in progress and the spike in the first cluster, t_{diff}. If t_{diff} was less than 1 ms, the spike in progress could not belong to the first cluster and was classified as noise. Else, the spike in progress was assigned to the first cluster.

For the incoming spikes, the sorting block computed the Euclidean distances between the mean waveforms of all clusters and those spikes using Eq. 3.7. If the minimum distance was greater than Thr_{sort}, the spike was assigned to a new cluster. Else if the minimum distance was not greater than Thr_{sort} and if the second minimum distance was greater than Thr_{sort}, the spike was classified as noise if the time difference with the last assigned spike in the cluster with minimum distance was less than 1 ms, else it was assigned to the cluster with the minimum distance.

If the two smallest distance values were less than Thr_{sort}, D_{diff} was calculated by subtracting the first smallest distance from the second smallest. If D_{diff} was greater than or equal to $Thr_{cls}(t)$, the spike was assigned to the cluster with the first smallest distance if the time difference with the last assigned spike in the cluster with minimum distance was more than 1 ms, else to the second closest cluster. If D_{diff} was less than $Thr_{cls}(t)$, and if the time difference with the last assigned spike in the cluster with minimum distance was more than 1 ms, the spike was assigned to the first closest cluster with a flag and was not considered while calculating the cluster's mean waveform. Else it was assigned to the second closest cluster with a flag. We flagged those spikes to control them in the following steps of the sorting process because they are likely to belong to more than one cluster.

We developed this control mechanism because hard clustering algorithms like OSort directly assign a spike to a cluster and do not change this assignment. Especially if the waveforms of the spikes of different neurons are similar, this issue may cause errors. At the beginning of the sorting process, if a spike is assigned to the wrong cluster, it affects the mean waveform of the cluster. This may cause a snowball effect

and end with the failure of the whole process. With this process, we provided a control mechanism for the spikes classified as a spike close to more than one cluster, and this mechanism positively affected the final results of the sorting process.

Finally, the mean waveform was calculated using the last hundred unflagged spikes assigned to the same cluster. The pseudo-code of the sorting block is shown in Appendix A (Algorithm 4 and 5). The flowchart of this block is shown in Figure 3.3.

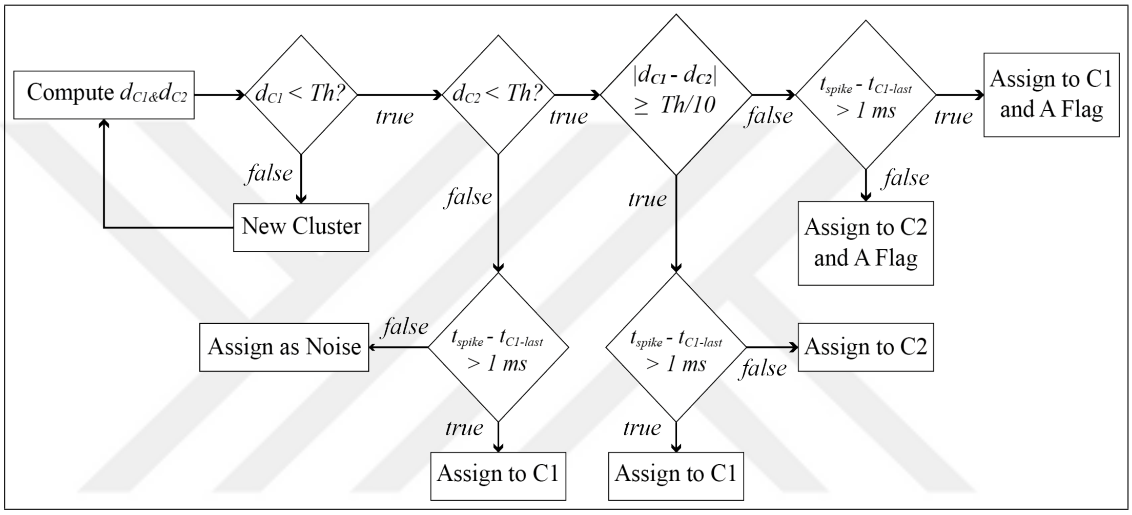

Figure 3.3 Flowchart of the sorting block of the algorithm designed in this study.

.

3.2.4 Cluster Merging

Clustering algorithms use different similarity metrics for merging clusters. The OSort uses the Euclidean distance and residual-sum-of-squares to calculate similarity based on mean waveforms of clusters, using the formula in Eq. 3.13, where DD_{ij} is the distance between clusters with mean waveforms M_1 and M_2.

$$DD_{1-2} = \sum_{k=1}^{N} (M_1(k) - M_2(k))^2 \tag{3.13}$$

The merging threshold Th_{merg} is taken to be the same as the sorting threshold

Th_{sort} Eq. 3.6. Once a spike is assigned to a cluster and its mean waveform is computed, distances to other clusters DD are also computed. If the minimum DD value is below Th_{merg}, the two clusters are merged, spikes of the newer cluster are transferred to the older one, and the emptied cluster is removed.

We improved the OSort to merge clusters only when they reached a certain number of spikes to ensure the reliability of their mean waveforms. Our algorithm computed distances between clusters DD after each spike assignment, stored them in memory, and compared them to the merging threshold Thr_{merg} (Eq. 3.12). When merging candidate clusters reached this certain number of spikes, the merging process began with the closest clusters having DD values less than Thr_{sort} and continued until all candidate clusters were merged. Our algorithm carried the newly created cluster's spikes from the old cluster, removed old ones from memory, then recalculated DD values. This process was repeated until there were no clusters to merge. Appendix A (Algorithm 6) shows the merging block's pseudo-code.

3.2.5 Cluster Pruning

Cluster pruning is a process of cleaning unnecessary clusters from memory for efficiency. In the original OSort, pruning is applied after every thousand sorted spikes, where clusters with less than two assigned spikes are deleted, and their spikes are labeled as noise. Our study changed the pruning mechanism by applying it after every cluster merging step. We cleaned clusters with less than 0.5% of all sorted spikes and labeled their spikes as noise. The pseudo-code of the merging block and pruning process is shown in Appendix A (Algorithm 7).

3.2.6 Control Block

A control block was added to the OSort to perform a second check on flagged spikes by calculating their Euclidean distance to all clusters and comparing them to

the sorting threshold Thr_{sort} following the cluster merging and pruning steps. If the minimum of the Euclidean distance values was equal to or more than the threshold, it was carried to a new cluster.

Else if the minimum of the Euclidean distance values was less than the threshold and if the minimum distance belongs to the cluster that already has this spike, there was no need for a change. However, If the cluster with the minimum Euclidean distance value was different from the cluster that already owns this spike, the spike was carried to the cluster with the minimum Euclidean distance value. After the control of flagged spikes, their flags were removed. This checking mechanism improved spike classification and enhanced the final results. The pseudo-code for this block is shown in Appendix A (Algorithm 8). The control part from the algorithm's flowchart is shown with dashed lines in Figure 3.4.

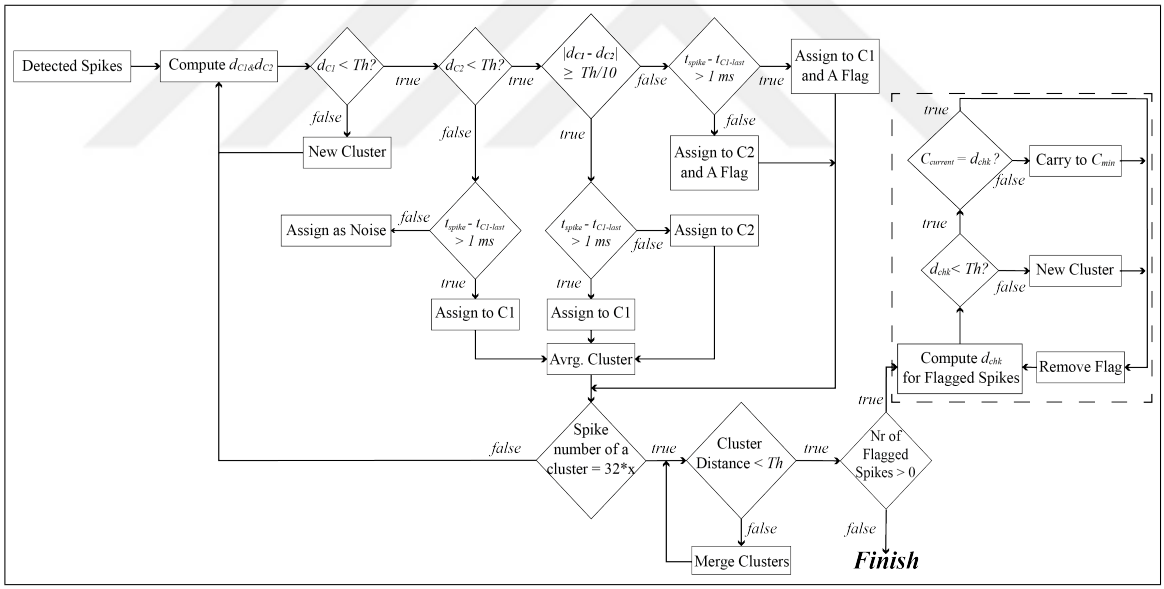

Figure 3.4 The control part of the flowchart of the algorithm designed in this study is shown with dashed lines.

.

4. RESULTS

The fundamental aim of spike sorting algorithms is to perform more accurate sorting. Our study presents an unsupervised and online spike sorting algorithm that incorporates varying threshold calculations, outperforming existing online and unsupervised sorting methods. Furthermore, it is suitable for real-time applications when implemented on a user-class FPGA board. In this chapter, we compared our sorting results with existing online and offline algorithms obtained from the software. We used the $F1$ score given in Eq. 4.3 to compare with other algorithms.

$$Precision = \frac{TP}{TP + FP} \tag{4.1}$$

$$Recall = \frac{TP}{TP + FN} \tag{4.2}$$

$$F1 = \frac{2 * Precision * Recall}{Precision + Recall} \tag{4.3}$$

4.1 Performance Comparison

We first compared the sorting results of our algorithm with those of OSort [2]. We compared our results with and without the implementation of refractoriness control to evaluate the effectiveness of the algorithm's component responsible for controlling spikes that could potentially belong to the second closest cluster.

They provided a simulated dataset selected from a database, including 150 different spike waveforms. They simulated Poisson distributed three recordings at 25 kHz sampling rate, with 100 seconds duration. Additionally, they generated different background noise signals by randomly selecting the waveforms from the same database and changing their frequencies, amplitudes, and waveforms. The standard deviations of the noise signals were adjusted to 0.5, 0.10, 0.15, and 0.20 to generate different

signal-to-noise ratio database waveforms. Noise signals were added to three simulated recordings.

4.1.1 Dataset I

With the first dataset, the sorting capability of the algorithms was demonstrated [2]. Figure 4.1 shows three spike waveforms were used to simulate the signal trace. The peak levels were equalized to provide the same signal-to-noise ratio for every spike waveform at different noise levels. Four different simulated signal traces were embedded in this dataset with signal-to-noise ratios of 6.7, 3.4, 2.2, and 1.1 with respect to noise standard deviations 0.05, 0.1, 0.15, and 0.2, respectively. Those signal-to-noise ratios were calculated using the formula in Eq. 4.4.

$$SNR = 10\log_{10}\left(\frac{\sum_{i=1}^{N}|x_i|^2}{\sum_{i=1}^{N}|e_i|^2}\right) \tag{4.4}$$

where x_i and the e_i are the signal and noise signals, respectively.

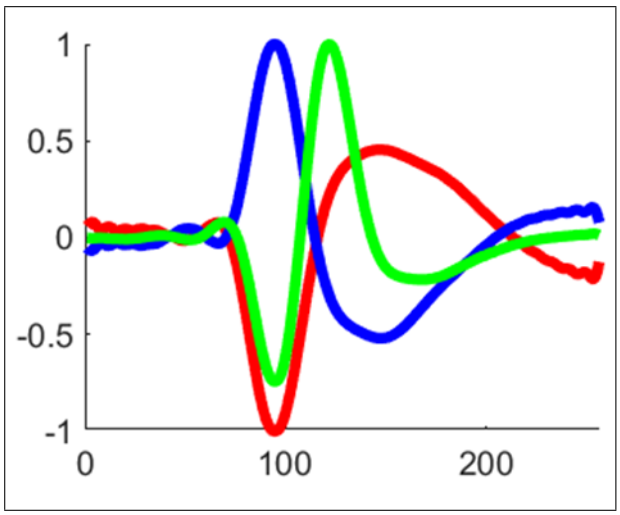

Figure 4.1 Spike waveforms used in the first dataset.

The sorting results of the original algorithm and our development are given in Appendix B (Table B.1). As seen in Table 4.1, OSort and our algorithm yielded the same $F1$ scores for signal traces with noise levels of 0.05 and 0.1. Even though

our algorithm performed better for signals under noise levels of 0.15 and 0.20, the refractoriness control's effect was insignificant. In contrast, the effect of controlling spikes that could potentially belong to the second closest cluster was more according to Table B.4-B.5 that show the results of two-way ANOVA test and Tukey's HSD post-hoc test.

Table 4.1
$F1$ score comparisons of OSort [2] and the algorithm designed in this study for the first dataset.

F1 Scores For Data Set # 1				
Algorithm	**Noise STD 0.05**	**Noise STD 0.10**	**Noise STD 0.15**	**Noise STD 0.20**
OSort [2]	0.99	0.99	0.96	0.81
Our Algorithm (without Refractoriness Control)	0.99	0.99	0.98	0.89
Our Algorithm (with Refractoriness Control)	0.99	0.99	0.98	0.89

4.1.2 Dataset II

The second dataset was used to show the spike detection performance of the algorithm [2]. Again, there were three different spike waveforms in this dataset. However, their peak points were not equalized this time, implying that the signal-to-noise ratio for each spike waveform was different. Figure 4.2 shows the spike waveforms in this dataset. Four different simulated signal traces were included in this dataset with average signal-to-noise levels of 5.17, 2.57, 1.70, and 1.30 and calculated with the Eq. 4.4 for noise standard deviations of 0.05, 0.1, 0.15, and 0.2, respectively. The sorting results of the OSort [2] and our algorithm on this dataset are shown in Appendix B (Table B.2). As seen in Table 4.2, OSort showed similar $F1$ scores as our algorithm's for 0.05 noise. Our algorithm performed better for noise standard deviations of 0.10, 0.15, and 0.20, while controlling spikes that could potentially belong to the second closest cluster affected the results more according to Table B.4-B.5 which show the results of two-way ANOVA test and Tukey's HSD post-hoc test.

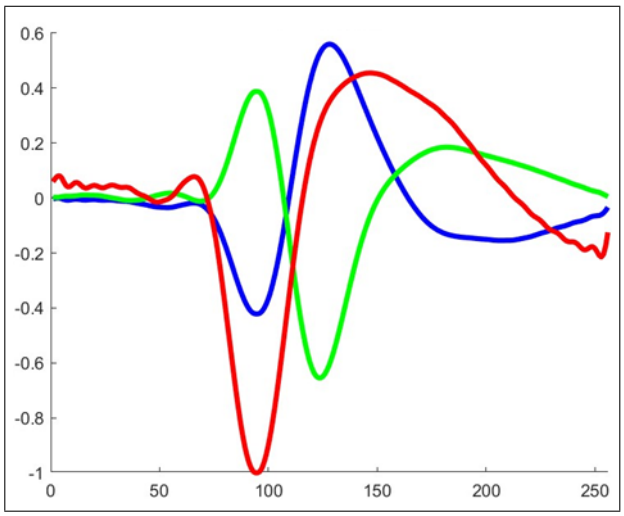

Figure 4.2 Spike waveforms used in the second dataset.

Table 4.2
$F1$ Score comparisons of OSort [2] and the algorithm designed in this study for the second dataset.

F1 Scores For Data Set # 2				
Algorithm	**Noise STD 0.05**	**Noise STD 0.10**	**Noise STD 0.15**	**Noise STD 0.20**
OSort [2]	0.99	0.96	0.69	0.44
Our Algorithm (without Refractoriness Control)	0.99	0.97	0.77	0.54
Our Algorithm (with Refractoriness Control)	0.99	0.97	0.78	0.55

4.1.3 Dataset III

The third dataset was generated to show the algorithm's detection and sorting performance limits. Figure 4.3 shows five different spike waveforms in this dataset; each one is a scaled version of the same spike waveform. Especially for higher signal-to-noise ratios, such waveforms become hard to be sorted due to their similarity. Four noise traces were added to the signal, as in the other two datasets. Their standard deviations were 0.05, 0.10, 0.15, and 0.2, corresponding to 4.74, 2.34, 1.56, and 1.16 signal-to-noise ratios calculated with the Eq. 4.4, respectively. The sorting results of the original algorithm and our algorithm on this dataset are shown in Appendix B (Table B.3). As seen in Table 4.3, our algorithm had higher $F1$ scores for all noise levels.

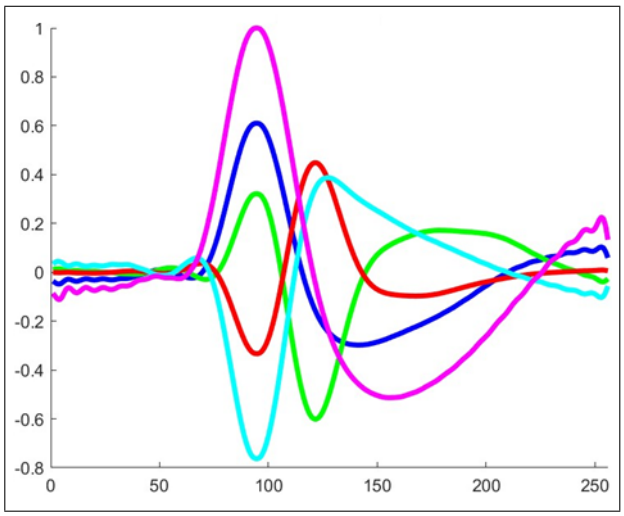

Figure 4.3 Spike waveforms used in the third dataset.

Additionally, when we check Table B.4-B.5, we saw that the refractoriness control had a minor effect than controlling the second closest cluster according to Table B.4-B.5 which show the results of two-way ANOVA test and Tukey's HSD post-hoc test.

Table 4.3
$F1$ score comparisons of OSort [2] and the algorithm designed in this study for the third dataset.

F1 Scores For Data Set # 3				
Algorithm	**Noise STD 0.05**	**Noise STD 0.10**	**Noise STD 0.15**	**Noise STD 0.20**
OSort [2]	0.93	0.77	0.60	0.43
Our Algorithm (without Refractoriness Control)	0.97	0.88	0.72	0.51
Our Algorithm (with Refractoriness Control)	0.97	0.89	0.73	0.53

4.2 Comparison with Other Algorithms

We prepared a set of simulated signals to compare our algorithm with many other algorithms in the literature. We used a MATLAB add-in named SpikeSimulationTool, developed by Soto-Breceda et al. [46], for the spike signal simulations. We

set the sampling frequency as 25 kHz and the duration as 100 seconds. We simulated three groups of signals with different average signal-to-noise ratios as 1.3, 1.7, and 2.6. Twenty probabilistic signal traces with three different spike waveforms were generated for each noise level. In total, we had 60 probabilistic signal traces. Id's of those signal traces and the number of spikes of each waveform in every signal trace are shown in Appendix B (Table B.7).

We compared the spike sorting performance of our algorithm with one online and five offline clustering algorithms from the literature. The online algorithm we used for comparison was Osort [2]. Many studies were based on the OSort algorithm in the literature; the spike sorting algorithm we designed in this study was also based on it.

On the other hand, we compared the performance of our algorithm with the most common offline spike algorithms to study the difference between the online and offline algorithms. The first algorithm was the $t-$distribution spike sorting technique [8], which sorted spike signals automatically using mixtures of multivariate $t-$distributions as one of the most popular offline spike sorting techniques. $K-$means was another popular offline algorithm, even if it was a template-matching based technique like the online algorithms. Su et al. [7] applied the $K-$means technique for spike sorting. The third offline technique we chose was Gaussian Mixture Models (GMM) [6]. For the fourth one, we used the skew-t distribution-based automatic spike sorting algorithm designed in [5], one of the most recent algorithms proposed in the literature. The last one was Wave Clus 3 [4], the latest version of a well-known offline sorting algorithm; the first version was developed by Quiroga et al. [47]. All offline spikes sorting algorithms except Wave Clus were added into a MATLAB add-in named ROSS, developed by Toosi et al. [5].

For OSort and Wave Clus, the default detection mechanisms of the algorithms were used. A median calculation based spike detection technique was used for other offline methods. Before detection, signals were filtered with a fourth-order Butterworth filter from 300 Hz to 3000 Hz. While running, all parameter settings were chosen as default. The sorting results of the algorithms for signal traces 1.3_3, 1.7_4, and 2.6_5

are shared as examples in Table B.8-B.10, demonstrating a sorting outcome for every different dataset with varying average signal-to-noise ratios.

$F1$ scores were calculated as shown in Eq. 4.3 for statistical testing. The precision, recall, and F1 scores of algorithms for every simulation are shown in Appendix B (Tables B.11-B.17). The means of the F1 scores are represented in Figure 4.4.

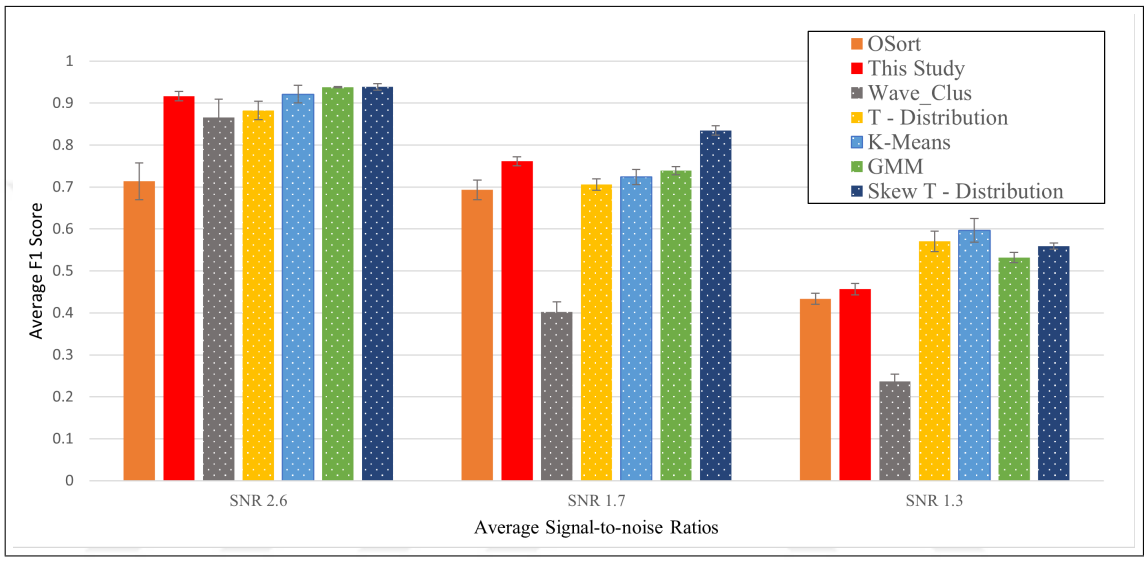

Figure 4.4 The means of the $F1$ scores of different algorithms for simulated probabilistic datasets with varying signal-to-noise ratios. Solid fills show the online, and dotted fills show the offline algorithms.

As seen in Figure 4.4, for the dataset with the average signal-to-noise ratio of 2.6, the sorting performance of our algorithm was better than those of Wave Clus 3 [4], Osort [2] and $t-$distribution sorting [8], but worse than the those of $K-$means [7], GMM [6] and skew$-t$ distribution [5].

For the second dataset, which has a 1.7 signal-to-noise ratio, the performance of Wave Clus 3 [4] was much lower than the results of other algorithms. While the skew$-t$ distribution based sorting algorithm [5] had the best performance among all algorithms, our algorithm had the second-best performance.

For the last dataset, which had a 1.3 average signal-to-noise ratio, Wave Clus 3 [4] had the worst result among all algorithms again. K-means spike sorting algorithm

[7] had the best results, with an average $F1$ score of nearly 0.6. Our algorithm had better results than Wave Clus 3 [4] and Osort [2] but was far from other algorithms.

Table 4.4
Results of the two way ANOVA test.

Source of Variation	SS	df	MS	F	P-Value
Algorithm	3.338	6	0.556	63.309	0.000
Average SNR	11.200	2	5.600	637.207	0.000
Interaction	1.497	12	0.125	14.198	0.000
Within	3.507	399			
Total	19.542	419			

Looking at the graph in Figure 4.4, which compares the means of $F1$ scores, the offline algorithms performed better than the online algorithms, especially at the lower signal-to-noise ratios. However, our algorithm had the best results among online algorithms for all noise levels. We controlled the null hypothesis to support those comments about sorting algorithms from Table 4.4, which is the output of two-way ANOVA. When we looked at p values calculated according to sorting algorithms, noise levels, and interaction of algorithms and noise levels, all p values were less than 0.05; that means the difference between the results of the algorithms was significant. We rejected the null hypothesis for both fixed factors and their combination.

According to multiple comparison tables, which is the outcome of Tukey's HSD post-hoc test, our algorithm indicates a $p < 0.05$ for Wave Clus 3 [4] and OSort [2], showing a significantly better performance than these algorithms. However, the p values for other offline algorithms were higher than 0.05, except for the skew$-t$ distribution [5], the only offline algorithm that performed significantly better than ours. Since offline algorithms use complex mathematical methods to sort spikes, most of them perform better than online algorithms. The features we added positively affected the sorting results and achieved closer performances to offline ones.

5. DISCUSSION

5.1 Previous Studies

There are several studies in the literature that sort neural spikes in real-time. From a general point of view, spike sorting algorithms can be divided into three main blocks; spike detection, feature extraction, and sorting. Also, offline or online training phases can be incorporated into the clustering block. Every spike sorting starts with detecting neural spikes from the signal trace, separating them from background noise and multiunit activity. A good detection algorithm should catch spikes even when the signal is very noisy. On the other hand, detection should be quick and with minimum delay, especially for real-time applications. That is why basic detection techniques are generally preferred for real-time applications. The absolute thresholding [48, 49], the amplitude thresholding with the standard deviation of background noise [50, 51], or the signal itself [52], and the non-linear (Teager) energy operator (NEO) based thresholding [40, 41, 53, 54, 55] are the three most widely used spike detection methods in the literature. Among them, NEO is the one that can provide a better detection percentage because it has been developed to catch the points where the signal shows a large change in both frequency and amplitude. Additionally, it does not need any standard deviation operation, which makes it easy to implement in hardware platforms and computationally cost-effective. Due to these reasons, we used a NEO-based spike detection method. Further, some novel applications took the spike sorting process as a single piece and did not divide it into blocks [56].

The second step of a common spike sorting process is feature extraction. Before sorting spike signals, some techniques calculate different features of the spike in progress for a better sorting performance. There are a large variety of feature extraction methods; the principle component analysis (PCA) [50, 52], and the Haar Wavelets [57, 58] are two of the most common ones that have been used for real-time applications. While some studies use those features directly in the spike sorting stage [59, 60],

others use them during online and offline learning phases [61, 62, 63]. On the other hand, some examples of applications with learning phases without a feature extraction process exist. While some of them extract spike templates for clustering with common online sorting algorithms [48, 64, 65, 66], others use methods like PCA by Hebbian Eigenfilter [67] and supervised learning [68, 69]. Some novel methods also exist in the literature, such as spike-timing-dependent plasticity learning [70]. Even though both feature extraction and learning increase the spike sorting accuracy, they also increase the computational complexity and cost. Especially, strategies in offline learning algorithms make them far from reaching the aims of real-time spike sorting. In this study, we did not use any feature extraction or learning method with the aim of low computational cost. To increase the sorting accuracy, we added some extra features directly to the sorting phase, which could be implemented in FPGA for real-time applications.

The spike sorting stage is a clustering algorithm's last but most crucial part. Template matching based spike sorting methods are proper for real-time applications due to their low complexity. Even though they are not as accurate as offline sorting methods, their low computational cost makes them preferred mostly [24]. The Euclidean distance based techniques [71, 72], L1 norm distance based algorithms [73, 74], and versions of K-Means [59, 72] are the most commonly preferred template matching based spikes sorting algorithms for real-time usage. However, in recent years, different neural network techniques have been tried in different hardware platforms for real-time sorting. The binarized neural networks [75], the competitive learning (CL) neural networks [68], and the convolutional neural networks (CNN) [69] are successful examples of them. Although they provide high accuracy, they have higher computational costs due to their complexity.

In this study, we took OSort [2] as a base and built our algorithm on it because of its high classification accuracy, low computational complexity, and memory requirement [76]. By adding some features to the base of OSort, such as checking the second closest cluster, we achieved better $F1$ scores. We upsampled the signals by a factor of four for a more accurate Euclidean distance comparison. We determined detection, sorting, and merging thresholds variably. Even though most algorithms do

not use varying thresholding, it is an effective way to detect the changes in the signal in real time. We also optimized the merging and pruning process. As an algorithm with higher accuracy, it can be applied and used on the user-class FPGAs for real-time applications with lower computational requirements.

5.2 Technical Limitations

While performing spike sorting, simulated signals were used to examine the algorithms. Most simulated signals in the literature have standard background noise, which is assumed as stationary. However, background noise may change while processing real signals due to environmental conditions, which may fail the methods that do not calculate the detection threshold variably from the streaming signal. With varying thresholding, a system can be adapted to slow changes in levels of noise.

Neural bursts, which are a train of spikes generated by a neuron in a short period with decreasing amplitudes and different waveshapes are another technical limitation. Those bursts do not obey the model of normal neural spikes and may cause problems for sorting.

Overlapping is another problem when two close neurons fire in a short time interval, thereby generating overlapped spikes. Furthermore, if those neurons synchronize, they keep generating overlapped spikes, causing them to act like a single neuron. Those waveforms may sometimes cancel each other causing no detection.

For template matching based spike sorting algorithms, realignment of the detected spike waveforms plays a crucial role. If algorithms fail about realignment, an exact template comparison can not be made, and the whole process fails. Due to this reason, the algorithm's realignment units should be designed carefully.

Some currently developed algorithms have high computational complexity, which means high computational cost and the need for FPGA boards with high computa-

tional performance. On the other hand, some applications employ user-class FPGA boards [57]. The low computational cost of an algorithm enables real-time spike sorting in electrophysiological studies, even with user-class FPGA boards.

While comparing the performances of different spike sorting algorithms, there must be gold standard data for a fair comparison accepted by the community.

In this study, we used the NEO-based method as the detection mechanism for the algorithm. Since the NEO method catches significant signal changes in both frequency and amplitude, potentially leading to the detection of multiple points for a spike waveform which may lead to processing a single spike more than once. An additional rule should be developed to deal with this issue.

We merged clusters when they were closer than a threshold and reached specific spike numbers or multiples, ensuring the accuracy of their mean waveforms. Although this approach reduces the number of merging operations and improves performance, it may be worth revising the rule since merging is not a frequent operation. Merging can be applied if both clusters have a specific number of spikes and can be merged without spike number control in further steps.

5.3 Conclusion and Future Prospects

In this study, we aimed to create an unsupervised algorithm for sorting neural spikes. We incorporated variable mechanisms for spike detection and sorting threshold calculations to achieve this. These thresholds are calculated based on the changing properties of the signal over time. Our approach involved enhancing existing template matching-based spike sorting algorithms by introducing new features. Additionally, we considered the algorithm's computational complexity, as our ultimate goal is to implement it on a user-class FPGA board for real-time processing of neurophysiological signals in the future. With all these, we successfully developed an algorithm that can be used by researchers, even those without programming experience.

We tested the algorithm in this study with simulated probabilistic datasets with varying average signal-to-noise ratios. We compared the performance of our algorithm with the other algorithms in the literature. According to the two-way ANOVA test we applied, our algorithm performed significantly better than one of the most popular online spike sorting algorithms OSort [2] and one of the most popular offline spike sorting algorithms Wave Clus [4]. Among the offline algorithms we compared, the skew-t distribution sorting [5] performed significantly better than ours. However, K-Means sorting [7], t-distribution sorting [8], and GMM sorting [6] showed better performance than ours but not significantly.

For further studies, we aim to apply our spike sorting algorithm to a user-class FPGA. Wang et al. showed a good example of real-time spike sorting using a user-class FPGA board, and they used Xilinx Artix-7 35T FPGA board [57]. We will apply our algorithm to low-price user-class FPGAs. This issue is important for disseminating the use of real-time spike sorting in neuroscience studies since high-class industrial FPGA boars are expensive and not affordable in most countries in the Middle East, North Asia, and Africa.

We plan to generate simulated signals on a computer in real-time as input to an FPGA board and collect the outputs in a second computer to perform evaluations. After the algorithm performance on a user-class FPGA board for real-time signals from a single channel is established, modifications may be done for processing signals from multiple channels.

APPENDIX A. ADDITIONAL ALGORITHMS

Algorithm 1: NEO-based spike detection block of our algorithm.

f = Filtered signal$x$

$\overrightarrow{S}$ = Detected Spike

for $every(n)$ **do**

  calculate $\psi(f[n]) = f^2[n] - f[n+1] \cdot f[n-1]$

  calculate $Th_{neo} = \frac{C_{neo}}{N} \sum_{n=1}^{N} (\psi(f[n]))$

  if $\psi(f[n]) > Th_{neo}$ **then**

    save S

  end

end

Algorithm 2: Upsampling and realignment block of the *Osort* algorithm.

$\overrightarrow{S}$ = Detected Spike

$\overrightarrow{R}$ = Upsampled and Realigned Spike

for $every(\overrightarrow{S})$ **do**

  $\overrightarrow{A}(k) = interpft(\overrightarrow{S}(k), 256);$

  $b = find(max\,\overrightarrow{A}(k)));$

  if $b = 95$ **then**

    $\overrightarrow{R}(k) = \overrightarrow{A}(k);$

  else

    realign $\overrightarrow{A}(k)$ as $b = 95;$

    $\overrightarrow{R}(k) = \overrightarrow{A}(k);$

  end

end

Algorithm 3: Sorting block of the *OSort* algorithm.

$\overrightarrow{R}$ = Upsampled and Realigned Spike

D = Euclidean Distance

M = Cluster Mean Waveform

for *Every Cluster* **do**

  $D = \sum_{k=1}^{N}(\overrightarrow{R}(k) - M(k))^2$;

end

if $min(D) > Th_{sort}(t)$ **then**

  Generate a new cluster and assign R to it.

else

  Assign R to cluster with $min(D)$

  Calculate cluster mean of cluster with new spike $\overrightarrow{R}$

end

Algorithm 4: Difference comparison mechanism for Euclidean distances in our algorithm.

$\overrightarrow{R}$ = Upsampled and Realigned Spike

D_{diff} = Difference of Two Euclidean Distance Value

$D_{diff} = min(D)_2 - min(D)_1$;

$t_{diff} = t_{peak-spike} - t_{peak-1}$;

if $D_{diff} \geq Thr_{cls}(t)$ **then**

  if t_{diff} *> 1 ms* **then**

    Assign $\overrightarrow{R}$ to the cluster with $min(D)_1$

  else

    Assign $\overrightarrow{R}$ to the cluster with $min(D)_2$

  end

else

  if t_{diff} *> 1 ms* **then**

    Assign $\overrightarrow{R}$ to the cluster with $min(D)_1$, and flag it

  else

    Assign $\overrightarrow{R}$ to the cluster with $min(D)_2$, and flag it

  end

end

Algorithm 5: General design of the sorting block of algorithm in this study.

$\overrightarrow{R}$ = Upsampled and Realigned Spike

M = Cluster Mean Waveform

for *Every* $\overrightarrow{R}$ **do**

  for *Every Cluster* **do**

    $D = \sum_{k=1}^{N}(\overrightarrow{R}(k) - M(k))^2$;

  end

  if R *is the first spike* **then**

    Generate a new cluster and assign $\overrightarrow{R}$ into it.

  else

    if $min(D)_1 > Thr_{sort}$ **then**

      Generate a new cluster and assign $\overrightarrow{R}$ to it.

    else

      if $min(D)_2 > Thr_{sort}$ **then**

        $t_{diff} = t_{peak-spike} - t_{peak-1}$

        if t_{diff} *> 1 ms* **then**

          Assign $\overrightarrow{R}$ to the cluster with $min(D)_1$

        else

          Assign $\overrightarrow{R}$ as a noise signal

        end

      else

        Call the Algorithm 4

      end

    end

  end

end

Algorithm 6: Merging block of our algorithm.

M = Cluster Mean Waveform

$M_{current}$ = Cluster of $\overrightarrow{R}$

Z = Spike Numbers of the Cluster

Q = Limit Number for Merging Control

for *Every* $\overrightarrow{R}$ *assigned to a cluster* **do**

  for *Every Cluster* **do**

    $DD = \sum_{k=1}^{N}(M_{current}(k) - M(k))^2$;

  end

end

if *Any* $Z = Q$ *or its multiples* **then**

  if $min(DD) < Thr_{merg}$ **then**

    Merge the cluster that has Z spikes with the cluster that has $min(DD)$

    for *Every Cluster* **do**

      $DD = \sum_{k=1}^{N}(M_Z(k) - M(k))^2$;

    end

  end

end

Algorithm 7: Pruning block of algorithm in this study.

Z = Spike Numbers of the Cluster

for *Every cluster* **do**

  if $Z <$ *Number of Sorted Spikes*/200 **then**

    Remove the cluster

    Assign its spikes as noise signals

  end

end

Algorithm 8: Control block of our algorithm.

$\overrightarrow{R}_{flag}$ = Flagged Spike Signals

Cls_{crnt} = Current Cluster of the Flagged Spike Signal

Cls_{new} = New Cluster of the Flagged Spike Signal

for *Every* R_{flag} **do**

  $D_{chk} = \sum_{k=1}^{N}(M(k) - \overrightarrow{R}_{flag}(k))^2$;

  if $min(D_{chk}) > Thr_{sort}$ **then**

    Generate a new cluster and carry $\overrightarrow{R}_{flag}$ into it.

  else

    if Cls_{crnt} *is the cluster with* $min(D_{chk})$ **then**

      Break

    else

      Carry $\overrightarrow{R}_{flag}$ to Cls_{new} with $min(D_{chk})$

    end

  end

end

APPENDIX B. ADDITIONAL TABLES

Table B.1
The spike sorting performance of the OSort [2] algorithm and algorithm designed in this study for the first dataset.

Data Set #1													
OSort Results [2]													
		Noise STD 0.05			**Noise STD 0.10**			**Noise STD 0.15**			**Noise STD 0.20**		
Clusters	**# Spikes**	**# Detected**	**#TPs**	**#FPs**	**# Detected**	**#TPs**	**#FPs**	**# Detected**	**#TPs**	**#FPs**	**# Detected**	**#TPs**	**#FPs**
First	475	475	465	0	475	465	1	452	431	16	376	336	96
Second	718	712	695	0	712	697	1	695	669	34	585	547	135
Third	383	368	364	0	368	357	1	367	330	3	317	266	8
Total	1576	1555	1524	0	1555	1519	3	1514	1430	53	1278	1149	239
F1 Scores		0.99			0.99			0.96			0.81		
Results of Our Algorithm (without Refractoriness Comparison)													
		Noise STD 0.05			**Noise STD 0.10**			**Noise STD 0.15**			**Noise STD 0.20**		
Clusters	**# Spikes**	**# Detected**	**#TPs**	**#FPs**	**# Detected**	**#TPs**	**#FPs**	**# Detected**	**#TPs**	**#FPs**	**# Detected**	**#TPs**	**#FPs**
First	475	474	458	0	472	447	0	469	426	3	439	360	81
Second	718	705	691	0	705	666	0	699	653	11	670	588	110
Third	383	360	359	0	360	332	1	361	308	3	359	291	22
Total	1576	1539	1508	0	1537	1445	1	1529	1387	17	1468	1239	213
F1 Scores		0.99			0.99			0.98			0.89		
Results of Our Algorithm (with Refractoriness Comparison)													
		Noise STD 0.05			**Noise STD 0.10**			**Noise STD 0.15**			**Noise STD 0.20**		
Clusters	**# Spikes**	**# Detected**	**#TPs**	**#FPs**	**# Detected**	**#TPs**	**#FPs**	**# Detected**	**#TPs**	**#FPs**	**# Detected**	**#TPs**	**#FPs**
First	475	474	462	0	472	451	0	469	429	1	439	366	87
Second	718	705	689	0	705	669	0	699	652	19	670	584	106
Third	383	360	357	0	360	329	1	361	324	3	359	281	13
Total	1576	1539	1508	0	1537	1449	1	1529	1405	23	1468	1231	206
F1 Scores		0.99			0.99			0.98			0.89		

Table B.2
The spike sorting performance of the OSort [2] algorithm and algorithm designed in this study for the second dataset.

Data Set #2													
OSort Results [2]													
		Noise STD 0.05			**Noise STD 0.10**			**Noise STD 0.15**			**Noise STD 0.20**		
Clusters	**# Spikes**	**# Detected**	**#TPs**	**#FPs**	**# Detected**	**#TPs**	**#FPs**	**# Detected**	**#TPs**	**#FPs**	**# Detected**	**#TPs**	**#FPs**
First	470	470	446	0	446	376	10	297	218	83	166	102	187
Second	706	697	643	0	667	513	3	433	275	24	218	120	43
Third	392	379	377	0	378	358	4	372	337	178	324	252	135
Total	1568	1546	1466	0	1491	1247	17	1102	830	285	708	474	365
F1 Scores		0.99			0.96			0.69			0.44		
Results of Our Algorithm (without Refractoriness Comparison)													
		Noise STD 0.05			**Noise STD 0.10**			**Noise STD 0.15**			**Noise STD 0.20**		
Clusters	**# Spikes**	**# Detected**	**#TPs**	**#FPs**	**# Detected**	**#TPs**	**#FPs**	**# Detected**	**#TPs**	**#FPs**	**# Detected**	**#TPs**	**#FPs**
First	470	467	438	0	451	370	6	358	230	96	179	96	108
Second	706	691	638	0	689	512	2	640	334	31	445	221	68
Third	392	370	364	0	371	342	7	371	301	186	353	237	181
Total	1568	1528	1440	0	1511	1224	15	1369	865	313	977	554	357
F1 Scores		0.99			0.97			0.77			0.54		
Results of Our Algorithm (with Refractoriness Comparison)													
		Noise STD 0.05			**Noise STD 0.10**			**Noise STD 0.15**			**Noise STD 0.20**		
Clusters	**# Spikes**	**# Detected**	**#TPs**	**#FPs**	**# Detected**	**#TPs**	**#FPs**	**# Detected**	**#TPs**	**#FPs**	**# Detected**	**#TPs**	**#FPs**
First	470	467	438	0	451	370	4	358	235	84	179	101	101
Second	706	691	638	0	689	512	2	640	334	31	445	221	68
Third	392	370	364	0	371	342	4	371	305	186	353	258	177
Total	1568	1528	1440	0	1511	1224	10	1369	874	301	977	580	346
F1 Scores		0.99			0.97			0.78			0.55		

Table B.3
The spike sorting performance of the OSort [2] algorithm and algorithm designed in this study for the third dataset.

Data Set #3													
OSort Results [2]													
		Noise STD 0.05			**Noise STD 0.10**			**Noise STD 0.15**			**Noise STD 0.20**		
Clusters	**# Spikes**	**# Detected**	**#TPs**	**#FPs**	**# Detected**	**#TPs**	**#FPs**	**# Detected**	**#TPs**	**#FPs**	**# Detected**	**#TPs**	**#FPs**
First	509	506	465	1	370	323	38	218	8	9	126	0	9
Second	672	639	438	0	394	207	4	207	83	11	117	39	106
Third	375	131	117	0	60	37	47	31	1	9	27	0	6
Fourth	591	568	550	0	546	494	150	422	396	131	250	217	184
Fifth	839	793	793	0	793	776	5	776	776	211	647	606	127
Total	2986	2637	2363	1	2163	1837	244	1654	1264	371	1167	862	432
F1 Scores		0.93			0.77			0.60			0.43		
Results of Our Algorithm (without Refractoriness Comparison)													
		Noise STD 0.05			**Noise STD 0.10**			**Noise STD 0.15**			**Noise STD 0.20**		
Clusters	**# Spikes**	**# Detected**	**#TPs**	**#FPs**	**# Detected**	**#TPs**	**#FPs**	**# Detected**	**#TPs**	**#FPs**	**# Detected**	**#TPs**	**#FPs**
First	509	492	450	1	472	382	116	349	219	119	181	0	72
Second	672	642	439	0	633	329	17	568	238	68	361	117	143
Third	375	350	329	0	319	217	60	190	0	53	96	0	36
Fourth	591	553	541	0	554	456	126	530	430	358	443	375	365
Fifth	839	781	767	0	781	692	19	766	677	43	630	572	129
Total	2986	2818	2526	1	2759	2076	338	2403	1564	641	1711	1064	745
F1 Scores		0.97			0.88			0.72			0.51		
Results of Our Algorithm (with Refractoriness Comparison)													
		Noise STD 0.05			**Noise STD 0.10**			**Noise STD 0.15**			**Noise STD 0.20**		
Clusters	**# Spikes**	**# Detected**	**#TPs**	**#FPs**	**# Detected**	**#TPs**	**#FPs**	**# Detected**	**#TPs**	**#FPs**	**# Detected**	**#TPs**	**#FPs**
First	509	492	452	1	472	392	108	349	232	126	181	10	46
Second	672	642	442	0	633	340	15	568	247	49	361	134	137
Third	375	350	328	0	319	229	60	190	0	21	96	2	36
Fourth	591	553	543	0	554	470	118	530	429	366	443	375	352
Fifth	839	781	781	0	781	705	8	766	675	45	630	567	109
Total	2986	2818	2546	1	2759	2136	309	2403	1583	607	1711	1088	680
F1 Scores		0.97			0.89			0.73			0.53		

Table B.4
Results of the two way ANOVA test for dataset of OSort.

Source of Variation	SS	df	MS	F	P-Value
Algorithm	0.023	2	0.011	0.604	0.555
Noise STD	0.699	3	0.233	12.492	0.000
Interaction	0.008	6	0.001	0.075	0.998
Within	0.448	24			
Total	1.178	35			

Table B.5
The multiple comparison table from the two-way ANOVA's Tukey's HSD post-hoc test for OSort's dataset [2].

Algorithms		Mean Difference	Std. Error	P- Value	95% Confidence Interval	
					Lower Bound	Upper Bound
Our Algorithm (without Refractoriness Control)	OSort [2]	0.05079	0.05402	0.78534	-0.13666	0.23824
Our Algorithm (with Refractoriness Control)	OSort [2]	0.05515	0.05402	0.75239	-0.13230	0.24260
Our Algorithm (with Refractoriness Control)	Our Algorithm (without Refractoriness Control)	0.00436	0.05402	0.99821	-0.18309	0.19181

Table B.6
The multiple comparison tables from the two-way ANOVA's Tukey's HSD post-hoc test for comparison with other algorithms.

Algorithms		Mean Difference	Std. Error	P- Value	95% Confidence Interval	
					Lower Bound	Upper Bound
Wave Clus [4]	OSort [2]	-0.11191	0.01712	0.00000	-0.16263	-0.06119
	Our Algorithm	-0.21007	0.01712	0.00000	-0.26079	-0.15935
	T-distribution sorting [8]	-0.21544	0.01712	0.00000	-0.26617	-0.16472
	K-Means sorting [7]	-0.24582	0.01712	0.00000	-0.29654	-0.19510
	GMM sorting [6]	-0.23503	0.01712	0.00000	-0.28575	-0.18431
	Skew-t distribution sorting [5]	-0.27618	0.01712	0.00000	-0.32690	-0.22546
OSort [2]	Wave Clus [4]	0.11191	0.01712	0.00000	0.06119	0.16263
	Our Algorithm	-0.09817	0.01712	0.00000	-0.14889	-0.04745
	T-distribution sorting [8]	-0.10354	0.01712	0.00000	-0.15426	-0.05282
	K-Means sorting [6]	-0.13391	0.01712	0.00000	-0.18463	-0.08319
	GMM sorting [6]	-0.12312	0.01712	0.00000	-0.17384	-0.07240
	Skew-t distribution sorting [5]	-0.16427	0.01712	0.00000	-0.21499	-0.11355
Our Algorithm	Wave Clus [4]	0.21007	0.01712	0.00000	0.15935	0.26079
	OSort [2]	0.09817	0.01712	0.00000	0.04745	0.14889
	T-distribution sorting [8]	-0.00537	0.01712	0.99992	-0.05609	0.04535
	K-Means sorting [7]	-0.03574	0.01712	0.36158	-0.08646	0.01498
	GMM sorting [6]	-0.02495	0.01712	0.76949	-0.07568	0.02577
	Skew-t distribution sorting [5]	-0.06611	0.01712	0.00249	-0.11683	-0.01539
T-distribution sorting [8]	Wave Clus [4]	0.21544	0.01712	0.00000	0.16472	0.26617
	OSort [2]	0.10354	0.01712	0.00000	0.05282	0.15426
	Our Algorithm	0.00537	0.01712	0.99992	-0.04535	0.05609
	K-Means sorting [7]	-0.03037	0.01712	0.56609	-0.08109	0.02035
	GMM sorting [6]	-0.01958	0.01712	0.91395	-0.07030	0.03114
	Skew-t distribution sorting [5]	-0.06074	0.01712	0.00784	-0.11146	-0.01001
K-Means sorting [7]	Wave Clus [4]	0.24582	0.01712	0.00000	0.19510	0.29654
	OSort [2]	0.13391	0.01712	0.00000	0.08319	0.18463
	Our Algorithm	0.03574	0.01712	0.36158	-0.01498	0.08646
	T-distribution sorting [8]	0.03037	0.01712	0.56609	-0.02035	0.08109
	GMM sorting [6]	0.01079	0.01712	0.99579	-0.03993	0.06151
	Skew-t distribution sorting [5]	-0.03036	0.01712	0.56647	-0.08108	0.02036
GMM sorting [6]	Wave Clus [4]	0.23503	0.01712	0.00000	0.18431	0.28575
	OSort [2]	0.12312	0.01712	0.00000	0.07240	0.17384
	Our Algorithm	0.02495	0.01712	0.76949	-0.02577	0.07568
	T-distribution sorting [8]	0.01958	0.01712	0.91395	-0.03114	0.07030
	K-Means sorting [7]	-0.01079	0.01712	0.99579	-0.06151	0.03993
	Skew-t distribution sorting [5]	-0.04115	0.01712	0.19938	-0.09187	0.00957
Skew-t distribution sorting [5]	Wave Clus [4]	0.27618	0.01712	0.00000	0.22546	0.32690
	OSort [2]	0.16427	0.01712	0.00000	0.11355	0.21499
	Our Algorithm	0.06611	0.01712	0.00249	0.01539	0.11683
	T-distribution sorting [8]	0.06074	0.01712	0.00784	0.01001	0.11146
	K-Means sorting [7]	0.03036	0.01712	0.56647	-0.02036	0.08108
	GMM sorting [6]	0.04115	0.01712	0.19938	-0.00957	0.09187

Table B.7
The number of spikes of each waveform in every simulated signal trace.

Dataset with Average SNR 1.3					Dataset with Average SNR 1.7					Dataset with Average SNR 2.6				
Simulation ID	**WF #1**	**WF #2**	**WF #3**	**# Total Spikes**	**Simulation ID**	**WF #1**	**WF #2**	**WF #3**	**# Total Spikes**	**Simulation ID**	**WF #1**	**WF #2**	**WF #3**	**# Total Spikes**
1.3_1	536	471	313	1320	1.7_1	445	299	169	913	2.6_1	267	379	414	1060
1.3_2	24	329	186	539	1.7_2	358	127	471	956	2.6_2	26	243	475	744
1.3_3	356	205	274	835	1.7_3	106	329	472	907	2.6_3	220	336	299	855
1.3_4	419	297	139	855	1.7_4	254	459	353	1066	2.6_4	455	412	148	1015
1.3_5	78	359	132	569	1.7_5	493	390	261	1144	2.6_5	479	365	197	1041
1.3_6	381	321	252	954	1.7_6	403	443	232	1078	2.6_6	65	303	468	836
1.3_7	632	210	290	1132	1.7_7	725	175	200	1100	2.6_7	175	340	341	856
1.3_8	428	361	201	990	1.7_8	534	147	278	959	2.6_8	259	326	238	823
1.3_9	288	349	153	790	1.7_9	353	224	373	950	2.6_9	225	350	448	1023
1.3_10	572	450	198	1220	1.7_10	345	448	294	1087	2.6_10	252	410	255	917
1.3_11	448	451	219	1118	1.7_11	302	373	446	1121	2.6_11	203	266	461	930
1.3_12	168	235	234	637	1.7_12	537	139	447	1123	2.6_12	132	245	475	852
1.3_13	149	294	284	727	1.7_13	536	277	331	1144	2.6_13	247	293	288	828
1.3_14	117	230	306	653	1.7_14	145	416	451	1012	2.6_14	522	269	200	991
1.3_15	77	274	254	605	1.7_15	216	277	448	941	2.6_15	328	392	168	888
1.3_16	40	327	195	562	1.7_16	332	345	393	1070	2.6_16	435	362	167	964
1.3_17	513	228	176	917	1.7_17	420	331	219	970	2.6_17	379	314	335	1028
1.3_18	334	288	293	915	1.7_18	445	203	318	966	2.6_18	398	295	196	889
1.3_19	474	369	302	1145	1.7_19	145	399	393	937	2.6_19	332	281	181	794
1.3_20	451	222	240	913	1.7_20	429	382	175	986	2.6_20	96	256	392	744

Table B.8
The sorting results of all algorithms for simulation 1.3_3.

Simulation 1.3_3					
Algorithms	**Waveforms**	**# Spikes**	**# Detection**	**# TPs**	**# FPs**
Our Algorithm	WF #1	356	356	145	0
	WF #2	205	38	6	103
	WF #3	274	128	79	1
	Total	835	522	230	104
OSort [2]	WF #1	356	356	124	0
	WF #2	205	3	0	184
	WF #3	274	238	113	0
	Total	835	597	237	184
Wave Clus [4]	WF #1	356	356	285	0
	WF #2	205	45	25	1463
	WF #3	274	273	203	2
	Total	835	674	513	1465
Skew-t distribution [5]	WF #1	356	356	115	0
	WF #2	205	65	45	44
	WF #3	274	102	44	34
	Total	835	523	204	78
Gaussian Mixture Models [6]	WF #1	356	356	354	0
	WF #2	205	0	0	67
	WF #3	274	102	44	34
	Total	835	458	398	101
K-Means [7]	WF #1	356	356	354	0
	WF #2	205	2	0	0
	WF #3	274	247	247	93
	Total	835	605	601	93
T-distribution [8]	WF #1	356	356	356	0
	WF #2	205	2	0	0
	WF #3	274	247	247	91
	Total	835	605	603	91

Table B.9
The sorting results of all algorithms for simulation 1.7_4.

Simulation 1.7_4					
Algorithms	**Waveforms**	**# Spikes**	**# Detection**	**# TPs**	**# FPs**
Our Algorithm	WF #1	254	254	253	0
	WF #2	459	457	262	0
	WF #3	353	216	207	104
	Total	1066	927	722	104
OSort [2]	WF #1	254	254	242	0
	WF #2	459	459	253	0
	WF #3	353	104	98	2
	Total	1066	817	593	2
Wave Clus [4]	WF #1	254	253	228	1
	WF #2	459	459	392	0
	WF #3	353	326	260	1509
	Total	1066	1038	880	1510
Skew-t distribution [5]	WF #1	254	245	126	0
	WF #2	459	459	352	0
	WF #3	353	73	60	28
	Total	1066	777	538	28
Gaussian Mixture Models [6]	WF #1	254	245	245	112
	WF #2	459	459	459	0
	WF #3	353	73	0	0
	Total	1066	777	704	112
K-Means [7]	WF #1	254	253	250	3
	WF #2	459	459	459	0
	WF #3	353	228	225	89
	Total	1066	940	934	92
T-distribution [8]	WF #1	254	244	244	110
	WF #2	459	459	453	0
	WF #3	353	73	0	6
	Total	1066	776	697	116

Table B.10
The sorting results of all algorithms for simulation 2.6_5.

Simulation 2.6_5					
Algorithms	**Waveforms**	**# Spikes**	**# Detection**	**# TPs**	**# FPs**
Our Algorithm	WF #1	479	400	372	94
	WF #2	365	365	339	0
	WF #3	197	196	168	0
	Total	1041	961	879	94
OSort [2]	WF #1	479	407	396	0
	WF #2	365	365	362	0
	WF #3	197	197	188	0
	Total	1041	969	946	0
Wave Clus [4]	WF #1	479	477	215	1
	WF #2	365	363	250	0
	WF #3	197	197	63	0
	Total	1041	1037	528	1
Skew-t distribution [5]	WF #1	479	406	203	4
	WF #2	365	365	153	0
	WF #3	197	197	163	0
	Total	1041	968	519	4
Gaussian Mixture Models [6]	WF #1	479	463	447	81
	WF #2	365	365	365	0
	WF #3	197	197	197	22
	Total	1041	1025	1009	103
K-Means [7]	WF #1	479	463	462	87
	WF #2	365	365	365	1
	WF #3	197	197	196	1
	Total	1041	1025	1023	89
T-distribution [8]	WF #1	479	463	462	87
	WF #2	365	365	365	0
	WF #3	197	197	195	1
	Total	1041	1025	1022	88

Table B.11
F1 score, precision, and recall values for Wave Clus [4] based spike sorting algorithm.

Wave Clus [4] - F1 Score, Precision and Recall											
Dataset with Average SNR 1.3				Dataset with Average SNR 1.7				Dataset with Average SNR 2.6			
Simulation ID	F1 Score	Precision	Recall	Simulation ID	F1 Score	Precision	Recall	Simulation ID	F1 Score	Precision	Recall
1.3_1	0.306	0.197	0.677	1.7_1	0.374	0.230	0.995	2.6_1	0.098	0.093	0.408
1.3_2	0.144	0.078	0.940	1.7_2	0.244	0.145	0.781	2.6_2	0.991	0.988	0.993
1.3_3	0.387	0.259	0.761	1.7_3	0.379	0.235	0.976	2.6_3	0.998	0.996	1.000
1.3_4	0.121	0.071	0.401	1.7_4	0.534	0.368	0.969	2.6_4	0.994	0.998	0.989
1.3_5	0.152	0.093	0.408	1.7_5	0.507	0.345	0.961	2.6_5	0.995	0.998	0.992
1.3_6	0.239	0.190	0.322	1.7_6	0.476	0.313	0.994	2.6_6	0.798	0.988	0.670
1.3_7	0.179	0.136	0.261	1.7_7	0.370	0.328	0.423	2.6_7	0.822	0.994	0.701
1.3_8	0.340	0.232	0.633	1.7_8	0.374	0.315	0.460	2.6_8	0.858	0.996	0.753
1.3_9	0.189	0.179	0.199	1.7_9	0.330	0.296	0.372	2.6_9	0.908	0.990	0.839
1.3_10	0.200	0.152	0.292	1.7_10	0.367	0.305	0.463	2.6_10	0.992	0.989	0.995
1.3_11	0.230	0.254	0.210	1.7_11	0.275	0.319	0.241	2.6_11	0.856	0.992	0.753
1.3_12	0.338	0.243	0.556	1.7_12	0.271	0.344	0.224	2.6_12	0.861	0.997	0.758
1.3_13	0.238	0.211	0.273	1.7_13	0.494	0.333	0.955	2.6_13	0.869	0.994	0.772
1.3_14	0.212	0.160	0.316	1.7_14	0.478	0.324	0.915	2.6_14	0.982	0.994	0.971
1.3_15	0.153	0.217	0.118	1.7_15	0.250	0.243	0.257	2.6_15	0.964	0.994	0.937
1.3_16	0.242	0.158	0.519	1.7_16	0.522	0.357	0.975	2.6_16	0.824	0.991	0.706
1.3_17	0.362	0.259	0.601	1.7_17	0.247	0.336	0.196	2.6_17	0.877	0.994	0.784
1.3_18	0.221	0.216	0.226	1.7_18	0.530	0.419	0.721	2.6_18	0.952	0.993	0.913
1.3_19	0.244	0.181	0.372	1.7_19	0.512	0.354	0.925	2.6_19	0.824	0.990	0.706
1.3_20	0.238	0.175	0.369	1.7_20	0.507	0.368	0.816	2.6_20	0.850	0.990	0.744

Table B.12
F1 score, precision, and recall values for OSort [2] based spike sorting algorithm.

OSort [2] - F1 Score, Precision and Recall											
Dataset with Average SNR 1.3				Dataset with Average SNR 1.7				Dataset with Average SNR 2.6			
Simulation ID	F1 Score	Precision	Recall	Simulation ID	F1 Score	Precision	Recall	Simulation ID	F1 Score	Precision	Recall
1.3_1	0.381	0.583	0.283	1.7_1	0.759	0.644	0.924	2.6_1	0.473	0.868	0.325
1.3_2	0.504	0.632	0.419	1.7_2	0.531	0.741	0.414	2.6_2	0.994	0.989	1.000
1.3_3	0.529	0.563	0.499	1.7_3	0.837	0.885	0.794	2.6_3	0.998	0.996	1.000
1.3_4	0.450	0.793	0.314	1.7_4	0.825	0.997	0.704	2.6_4	0.922	0.999	0.855
1.3_5	0.344	0.517	0.258	1.7_5	0.660	0.657	0.662	2.6_5	0.963	1.000	0.929
1.3_6	0.518	0.543	0.494	1.7_6	0.611	0.888	0.466	2.6_6	0.605	0.933	0.448
1.3_7	0.501	0.735	0.381	1.7_7	0.568	0.973	0.401	2.6_7	0.996	0.993	0.998
1.3_8	0.480	0.660	0.377	1.7_8	0.605	0.684	0.543	2.6_8	0.598	0.808	0.474
1.3_9	0.350	0.613	0.245	1.7_9	0.605	0.646	0.569	2.6_9	0.798	0.672	0.983
1.3_10	0.395	0.611	0.292	1.7_10	0.831	0.903	0.770	2.6_10	0.491	0.897	0.338
1.3_11	0.464	0.675	0.353	1.7_11	0.661	0.969	0.502	2.6_11	0.880	0.908	0.853
1.3_12	0.394	0.661	0.281	1.7_12	0.588	0.678	0.520	2.6_12	0.535	0.735	0.420
1.3_13	0.502	0.720	0.385	1.7_13	0.640	0.741	0.564	2.6_13	0.682	0.933	0.538
1.3_14	0.377	0.785	0.248	1.7_14	0.830	0.816	0.844	2.6_14	0.656	0.688	0.627
1.3_15	0.392	0.675	0.276	1.7_15	0.783	0.764	0.804	2.6_15	0.888	0.830	0.955
1.3_16	0.432	0.752	0.303	1.7_16	0.615	0.915	0.464	2.6_16	0.697	0.785	0.626
1.3_17	0.473	0.594	0.393	1.7_17	0.770	0.943	0.651	2.6_17	0.542	0.713	0.437
1.3_18	0.385	0.565	0.292	1.7_18	0.759	0.812	0.713	2.6_18	0.484	0.686	0.374
1.3_19	0.415	0.788	0.281	1.7_19	0.578	0.747	0.471	2.6_19	0.526	0.986	0.358
1.3_20	0.385	0.556	0.294	1.7_20	0.805	0.956	0.695	2.6_20	0.542	0.649	0.466

Table B.13
F1 score, precision, and recall values for our algorithm.

Our Algorithm - F1 Score, Precision and Recall											
Dataset with Average SNR 1.3				Dataset with Average SNR 1.7				Dataset with Average SNR 2.6			
Simulation ID	F1 Score	Precision	Recall	Simulation ID	F1 Score	Precision	Recall	Simulation ID	F1 Score	Precision	Recall
1.3_1	0.542	0.673	0.454	1.7_1	0.729	0.591	0.951	2.6_1	0.883	0.902	0.865
1.3_2	0.539	0.670	0.451	1.7_2	0.789	0.854	0.732	2.6_2	0.998	0.996	1.000
1.3_3	0.525	0.689	0.424	1.7_3	0.766	0.703	0.843	2.6_3	0.888	1.000	0.977
1.3_4	0.498	0.736	0.377	1.7_4	0.856	0.874	0.839	2.6_4	0.914	0.999	0.842
1.3_5	0.430	0.450	0.413	1.7_5	0.769	0.827	0.718	2.6_5	0.910	0.903	0.917
1.3_6	0.457	0.485	0.433	1.7_6	0.691	0.769	0.627	2.6_6	0.953	0.994	0.915
1.3_7	0.408	0.622	0.304	1.7_7	0.773	0.746	0.801	2.6_7	0.822	0.976	0.710
1.3_8	0.500	0.555	0.455	1.7_8	0.737	0.721	0.754	2.6_8	0.889	0.989	0.807
1.3_9	0.449	0.520	0.395	1.7_9	0.843	0.837	0.848	2.6_9	0.881	0.919	0.846
1.3_10	0.334	0.693	0.220	1.7_10	0.694	0.708	0.682	2.6_10	0.911	0.917	0.904
1.3_11	0.476	0.673	0.369	1.7_11	0.730	0.696	0.767	2.6_11	0.927	0.913	0.941
1.3_12	0.485	0.485	0.486	1.7_12	0.788	0.755	0.824	2.6_12	0.925	0.950	0.901
1.3_13	0.391	0.585	0.294	1.7_13	0.745	0.862	0.656	2.6_13	0.854	0.988	0.751
1.3_14	0.390	0.551	0.301	1.7_14	0.691	0.640	0.750	2.6_14	0.987	0.976	0.998
1.3_15	0.513	0.643	0.426	1.7_15	0.704	0.716	0.692	2.6_15	0.914	0.910	0.918
1.3_16	0.485	0.719	0.366	1.7_16	0.823	0.828	0.818	2.6_16	0.943	0.982	0.906
1.3_17	0.437	0.723	0.313	1.7_17	0.732	0.821	0.660	2.6_17	0.845	0.985	0.740
1.3_18	0.375	0.488	0.304	1.7_18	0.807	0.719	0.918	2.6_18	0.908	0.938	0.879
1.3_19	0.387	0.462	0.334	1.7_19	0.774	0.810	0.742	2.6_19	0.987	0.986	0.987
1.3_20	0.516	0.470	0.573	1.7_20	0.787	0.847	0.735	2.6_20	0.990	0.984	0.996

Table B.14
F1 score, precision, and recall values for Skew-t Distribution [5] based spike sorting algorithm.

Skew-t Distribution [5] - F1 Score, Precision and Recall											
Dataset with Average SNR 1.3				Dataset with Average SNR 1.7				Dataset with Average SNR 2.6			
Simulation ID	F1 Score	Precision	Recall	Simulation ID	F1 Score	Precision	Recall	Simulation ID	F1 Score	Precision	Recall
1.3_1	0.585	0.959	0.420	1.7_1	0.894	0.844	0.949	2.6_1	0.936	0.960	0.914
1.3_2	0.612	0.694	0.547	1.7_2	0.764	0.814	0.720	2.6_2	0.952	0.911	0.998
1.3_3	0.511	0.723	0.395	1.7_3	0.832	0.806	0.859	2.6_3	0.997	1.000	0.994
1.3_4	0.544	0.822	0.407	1.7_4	0.772	0.951	0.651	2.6_4	0.879	0.920	0.843
1.3_5	0.512	0.731	0.394	1.7_5	0.738	0.846	0.655	2.6_5	0.931	0.992	0.877
1.3_6	0.567	0.747	0.457	1.7_6	0.828	0.865	0.793	2.6_6	0.949	0.959	0.939
1.3_7	0.598	0.763	0.492	1.7_7	0.887	0.928	0.849	2.6_7	0.952	0.954	0.950
1.3_8	0.565	0.911	0.410	1.7_8	0.792	0.875	0.723	2.6_8	0.936	0.954	0.918
1.3_9	0.606	0.911	0.454	1.7_9	0.870	0.902	0.841	2.6_9	0.973	0.950	0.998
1.3_10	0.591	0.802	0.468	1.7_10	0.889	0.889	0.889	2.6_10	0.997	0.998	0.996
1.3_11	0.580	0.866	0.436	1.7_11	0.785	0.898	0.697	2.6_11	0.899	0.934	0.867
1.3_12	0.535	0.840	0.393	1.7_12	0.883	0.832	0.941	2.6_12	0.910	0.913	0.906
1.3_13	0.598	0.930	0.441	1.7_13	0.790	0.931	0.686	2.6_13	0.905	0.986	0.836
1.3_14	0.534	0.929	0.374	1.7_14	0.833	0.808	0.859	2.6_14	0.884	0.931	0.841
1.3_15	0.523	0.911	0.367	1.7_15	0.873	0.901	0.847	2.6_15	0.940	0.997	0.889
1.3_16	0.551	0.794	0.422	1.7_16	0.879	0.846	0.916	2.6_16	0.970	0.982	0.959
1.3_17	0.525	0.946	0.364	1.7_17	0.867	0.814	0.928	2.6_17	0.907	0.970	0.852
1.3_18	0.539	0.899	0.385	1.7_18	0.802	0.915	0.714	2.6_18	0.973	0.970	0.975
1.3_19	0.590	0.736	0.493	1.7_19	0.831	0.915	0.761	2.6_19	0.970	0.987	0.954
1.3_20	0.520	0.903	0.366	1.7_20	0.888	0.906	0.870	2.6_20	0.910	0.938	0.884

Table B.15
F1 score, precision, and recall values for Gaussian Mixture Models [6] based spike sorting algorithm.

Gaussian Mixture Models [6] - F1 Score, Precision and Recall											
Dataset with Average SNR 1.3				Dataset with Average SNR 1.7				Dataset with Average SNR 2.6			
Simulation ID	F1 Score	Precision	Recall	Simulation ID	F1 Score	Precision	Recall	Simulation ID	F1 Score	Precision	Recall
1.3_1	0.575	0.903	0.422	1.7_1	0.661	0.581	0.768	2.6_1	0.942	0.930	0.953
1.3_2	0.454	0.440	0.470	1.7_2	0.761	0.920	0.649	2.6_2	0.937	0.884	0.997
1.3_3	0.625	0.798	0.514	1.7_3	0.805	0.837	0.776	2.6_3	0.942	0.912	0.974
1.3_4	0.477	0.814	0.337	1.7_4	0.778	0.863	0.709	2.6_4	0.929	0.925	0.933
1.3_5	0.471	0.762	0.341	1.7_5	0.700	0.650	0.758	2.6_5	0.944	0.907	0.984
1.3_6	0.496	0.680	0.390	1.7_6	0.753	0.913	0.641	2.6_6	0.940	0.918	0.965
1.3_7	0.583	0.862	0.440	1.7_7	0.726	0.615	0.887	2.6_7	0.943	0.898	0.994
1.3_8	0.560	0.680	0.477	1.7_8	0.678	0.746	0.622	2.6_8	0.939	0.886	0.997
1.3_9	0.601	0.623	0.580	1.7_9	0.713	0.754	0.676	2.6_9	0.937	0.914	0.961
1.3_10	0.580	0.632	0.535	1.7_10	0.710	0.901	0.585	2.6_10	0.939	0.924	0.954
1.3_11	0.556	0.724	0.451	1.7_11	0.798	0.680	0.965	2.6_11	0.941	0.918	0.966
1.3_12	0.592	0.700	0.512	1.7_12	0.769	0.730	0.813	2.6_12	0.941	0.893	0.995
1.3_13	0.583	0.503	0.692	1.7_13	0.695	0.603	0.819	2.6_13	0.931	0.913	0.950
1.3_14	0.489	0.520	0.461	1.7_14	0.678	0.795	0.592	2.6_14	0.944	0.918	0.972
1.3_15	0.541	0.751	0.423	1.7_15	0.756	0.659	0.888	2.6_15	0.931	0.916	0.948
1.3_16	0.470	0.628	0.375	1.7_16	0.760	0.648	0.919	2.6_16	0.942	0.928	0.956
1.3_17	0.469	0.487	0.451	1.7_17	0.758	0.738	0.780	2.6_17	0.941	0.908	0.977
1.3_18	0.482	0.564	0.421	1.7_18	0.734	0.864	0.638	2.6_18	0.929	0.897	0.964
1.3_19	0.529	0.842	0.386	1.7_19	0.800	0.670	0.992	2.6_19	0.931	0.918	0.945
1.3_20	0.511	0.727	0.394	1.7_20	0.747	0.865	0.657	2.6_20	0.938	0.900	0.979

Table B.16
F1 score, precision, and recall values for K-Means [7] based spike sorting algorithm.

K-Means [7] - F1 Score, Precision and Recall											
Dataset with Average SNR 1.3				Dataset with Average SNR 1.7				Dataset with Average SNR 2.6			
Simulation ID	F1 Score	Precision	Recall	Simulation ID	F1 Score	Precision	Recall	Simulation ID	F1 Score	Precision	Recall
1.3_1	0.613	0.838	0.483	1.7_1	0.752	0.613	0.972	2.6_1	0.546	0.906	0.391
1.3_2	0.756	0.756	0.756	1.7_2	0.679	0.782	0.600	2.6_2	0.935	0.877	1.000
1.3_3	0.788	0.866	0.723	1.7_3	0.823	0.789	0.860	2.6_3	0.998	1.000	0.995
1.3_4	0.430	0.730	0.305	1.7_4	0.895	0.910	0.881	2.6_4	0.890	0.852	0.932
1.3_5	0.418	0.595	0.322	1.7_5	0.609	0.525	0.725	2.6_5	0.951	0.920	0.985
1.3_6	0.686	0.616	0.773	1.7_6	0.769	0.906	0.668	2.6_6	0.988	0.993	0.983
1.3_7	0.772	0.689	0.878	1.7_7	0.643	0.605	0.687	2.6_7	0.911	0.891	0.931
1.3_8	0.553	0.654	0.479	1.7_8	0.800	0.836	0.767	2.6_8	0.934	0.932	0.936
1.3_9	0.466	0.653	0.363	1.7_9	0.715	0.591	0.905	2.6_9	0.962	0.927	0.999
1.3_10	0.449	0.733	0.324	1.7_10	0.626	0.838	0.500	2.6_10	0.891	0.900	0.883
1.3_11	0.721	0.689	0.756	1.7_11	0.623	0.664	0.586	2.6_11	0.987	0.991	0.982
1.3_12	0.746	0.700	0.799	1.7_12	0.649	0.602	0.703	2.6_12	0.951	0.937	0.966
1.3_13	0.532	0.787	0.402	1.7_13	0.709	0.811	0.630	2.6_13	0.995	0.993	0.998
1.3_14	0.579	0.745	0.473	1.7_14	0.685	0.735	0.642	2.6_14	0.902	0.934	0.872
1.3_15	0.636	0.862	0.504	1.7_15	0.688	0.611	0.787	2.6_15	0.952	0.935	0.971
1.3_16	0.604	0.721	0.519	1.7_16	0.814	0.864	0.769	2.6_16	0.963	0.992	0.936
1.3_17	0.537	0.631	0.468	1.7_17	0.751	0.846	0.676	2.6_17	0.941	0.954	0.928
1.3_18	0.716	0.672	0.766	1.7_18	0.720	0.899	0.600	2.6_18	0.922	0.946	0.899
1.3_19	0.443	0.859	0.299	1.7_19	0.844	0.855	0.832	2.6_19	0.891	0.931	0.854
1.3_20	0.484	0.645	0.388	1.7_20	0.686	0.690	0.682	2.6_20	0.916	0.879	0.957

Table B.17
F1 score, precision, and recall values for T-distribution [8] based spike sorting algorithm.

T-distribution [8] - F1 Score, Precision and Recall											
Dataset with Average SNR 1.3				Dataset with Average SNR 1.7				Dataset with Average SNR 2.6			
Simulation ID	F1 Score	Precision	Recall	Simulation ID	F1 Score	Precision	Recall	Simulation ID	F1 Score	Precision	Recall
1.3_1	0.627	0.867	0.492	1.7_1	0.619	0.456	0.963	2.6_1	0.537	0.808	0.402
1.3_2	0.588	0.518	0.680	1.7_2	0.661	0.751	0.590	2.6_2	0.727	0.572	1.000
1.3_3	0.790	0.869	0.724	1.7_3	0.822	0.787	0.860	2.6_3	0.950	0.909	0.995
1.3_4	0.429	0.727	0.304	1.7_4	0.774	0.857	0.706	2.6_4	0.890	0.852	0.932
1.3_5	0.532	0.708	0.426	1.7_5	0.698	0.648	0.757	2.6_5	0.952	0.921	0.985
1.3_6	0.775	0.781	0.769	1.7_6	0.630	0.703	0.571	2.6_6	0.832	0.851	0.814
1.3_7	0.491	0.866	0.342	1.7_7	0.652	0.764	0.568	2.6_7	0.890	0.811	0.985
1.3_8	0.451	0.568	0.374	1.7_8	0.801	0.848	0.759	2.6_8	0.901	0.847	0.963
1.3_9	0.585	0.590	0.579	1.7_9	0.652	0.485	0.997	2.6_9	0.852	0.908	0.803
1.3_10	0.533	0.595	0.483	1.7_10	0.730	0.790	0.679	2.6_10	0.853	0.843	0.864
1.3_11	0.484	0.531	0.444	1.7_11	0.719	0.574	0.961	2.6_11	0.846	0.841	0.851
1.3_12	0.663	0.798	0.568	1.7_12	0.742	0.697	0.792	2.6_12	0.941	0.917	0.967
1.3_13	0.545	0.474	0.642	1.7_13	0.750	0.740	0.760	2.6_13	0.949	0.904	0.999
1.3_14	0.642	0.693	0.599	1.7_14	0.776	0.821	0.736	2.6_14	0.932	0.917	0.947
1.3_15	0.471	0.628	0.377	1.7_15	0.657	0.835	0.542	2.6_15	0.932	0.877	0.995
1.3_16	0.445	0.603	0.352	1.7_16	0.668	0.653	0.684	2.6_16	0.941	0.902	0.983
1.3_17	0.730	0.599	0.935	1.7_17	0.681	0.810	0.588	2.6_17	0.940	0.917	0.964
1.3_18	0.513	0.703	0.404	1.7_18	0.627	0.571	0.695	2.6_18	0.940	0.905	0.978
1.3_19	0.627	0.735	0.547	1.7_19	0.720	0.830	0.635	2.6_19	0.936	0.893	0.983
1.3_20	0.484	0.645	0.388	1.7_20	0.737	0.736	0.737	2.6_20	0.904	0.917	0.891

REFERENCES

1. Kandel, E. R., J. H. Schwartz, T. M. Jessell, S. Siegelbaum, *et al.*, *Principles of Neural Science*, McGraw-hill New York, 4 ed., 2000.

2. Rutishauser, U., E. M. Schuman, and A. N. 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. 1-2, pp. 204–224, 2006.

3. Feher, J. J., *Quantitative Human Physiology: an Introduction*, Academic Press, 2017.

4. Chaure, F. J., H. G. Rey, and R. Q. Quiroga, "A novel and fully automatic spike-sorting implementation with variable number of features," *Journal of Neurophysiology*, Vol. 120, no. 4, pp. 1859–1871, 2018.

5. Toosi, R., M. A. Akhaee, and M.-R. A. Dehaqani, "An automatic spike sorting algorithm based on adaptive spike detection and a mixture of skew-t distributions," *Scientific Reports*, Vol. 11, no. 1, pp. 1–18, 2021.

6. Souza, B. C., V. L. dos Santos, J. Bacelo, and A. B. Tort, "Spike sorting with gaussian mixture models," *Scientific Reports*, Vol. 9, no. 1, pp. 1–14, 2019.

7. Su, C.-K., C.-H. Chiang, C.-M. Lee, Y.-P. Fan, *et al.*, "Computational solution of spike overlapping using data-based subtraction algorithms to resolve synchronous sympathetic nerve discharge," *Frontiers in Computational Neuroscience*, Vol. 7, p. 149, 2013.

8. Shoham, S., M. R. Fellows, and R. A. Normann, "Robust, automatic spike sorting using mixtures of multivariate t-distributions," *Journal of Neuroscience Methods*, Vol. 127, no. 2, pp. 111–122, 2003.

9. Quiroga, R. Q., "Spike sorting," *Current Biology*, Vol. 22, no. 2, pp. R45–R46, 2012.

10. Quiroga, R. Q., Z. Nadasdy, and B. Yoram, "Unsupervised spike detection and sorting with wavelets and superparamagnetic clustering," *Neural Computation*, Vol. 16, no. 8, pp. 1661–1687, 2004.

11. Kaiser, J. F., "On a simple algorithm to calculate the energy of a signal," in *International Conference on Acoustics, Speech, and Signal Processing*, pp. 381–384, IEEE, 1990.

12. Crochiere, R. E., and L. R. Rabiner, *Multirate Digital Signal Processing*, New Jersey: Prentice-hall Englewood Cliffs, NJ, 1983.

13. Schafer, R. W., and L. R. Rabiner, "A digital signal processing approach to interpolation," *Proceedings of the IEEE*, Vol. 61, no. 6, pp. 692–702, 1973.

14. Richard, W. D., "Efficient parallel real-time upsampling with xilinx fpgas," *Xcell Journal*, Vol. 89, pp. 38–44, 2014.

15. Tortora, G. J., and B. H. Derrickson, *Principles of Anatomy and Physiology*, John Wiley & Sons, 2018.

16. Purves, D., G. J. Augustine, D. Fitzpatrick, W. Hall, *et al.*, *Neurosciences*, De Boeck Supérieur, 2019.

17. Hirsch, B., "Gray's anatomy: the anatomical basis of clinical practice," *JAMA*, Vol. 301, no. 17, pp. 1825–1831, 2009.

18. Kaiser, J. T., and J. G. Lugo-Pico, "Neuroanatomy, spinal nerves," in *StatPearls [Internet]*, StatPearls Publishing, 2021.

19. Hopkins, J., A. Maton, W. M. Charles, J. Susan, *et al.*, "Human biology and health," *New Jersey: Englewood Cliffs*, 1993.

20. Kandel, E. R., *In Search of Memory: The Emergence of a New Science of Mind*, WW Norton and Company, 2007.

21. Kocagoncu, E., D. Nesbitt, T. Emery, L. E. Hughes, *et al.*, "Neurophysiological and brain structural markers of cognitive frailty differ from alzheimer's disease," *Journal of Neuroscience*, Vol. 42, no. 7, 2022.

22. Kipnis, N., "Luigi Galvani and the debate on animal electricity, 1791–1800," *Annals of Science*, Vol. 44, no. 2, pp. 107–142, 1987.

23. Hubel, D. H., and T. N. Wiesel, "Ferrier lecture-functional architecture of macaque monkey visual cortex," *Proceedings of the Royal Society of London. Series B. Biological Sciences*, Vol. 198, no. 1130, pp. 1–59, 1977.

24. Rey, H. G., C. Pedreira, and R. Q. Quiroga, "Past, present and future of spike sorting techniques," *Brain Research Bulletin*, Vol. 119, pp. 106–117, 2015.

25. Buccino, A. P., S. Garcia, and P. Yger, "Spike sorting: new trends and challenges of the era of high-density probes," *Progress in Biomedical Engineering*, 2022.

26. Nenadic, Z., and J. W. Burdick, "Spike detection using the continuous wavelet transform," *IEEE Transactions on Biomedical Engineering*, Vol. 52, no. 1, pp. 74–87, 2004.

27. Azami, H., J. Escudero, A. Darzi, and S. Sanei, "Extracellular spike detection from multiple electrode array using novel intelligent filter and ensemble fuzzy decision making," *Journal of Neuroscience Methods*, Vol. 239, pp. 129–138, 2015.

28. Lewicki, M. S., "A review of methods for spike sorting: the detection and classification of neural action potentials," *Network: Computation in Neural Systems*, Vol. 9, no. 4, p. R53, 1998.

29. Mallat, S., *A Wavelet Tour of Signal Processing*, Elsevier, 1999.

30. Abeles, M., and M. H. Goldstein, "Multispike train analysis," *Proceedings of the IEEE*, Vol. 65, no. 5, pp. 762–773, 1977.

31. D'Hollander, E. H., and G. A. Orban, "Spike recognition and on-line classification by unsupervised learning system," *IEEE Transactions on Biomedical Engineering*, no. 5, pp. 279–284, 1979.

32. Buccino, A., C. Hurwitz, S. Garcia, J. Magland, *et al.*, "Spikeinterface, a unified framework for spike sorting," *eLife*, Vol. 9, 2020.

33. Pouzat, C., and G. I. Detorakis, "Spysort: neuronal spike sorting with python," *arXiv Preprint arXiv:1412.6383*, 2014.

34. Haseeb, A., M. Faizan, M. F. Khan, and M. T. Iqrar, "Microprocessors and microcontrollers: Past, present, and future," in *Functional Reverse Engineering of Machine Tools*, pp. 3–28, CRC Press, 2019.

35. Marković, D., and R. W. Brodersen, *DSP Architecture Design Essentials*, Springer Science & Business Media, 2012.

36. Kuon, I., and J. Rose, "Measuring the gap between FPGAs and ASICs," in *Proceedings of the 2006 ACM/SIGDA 14th International Symposium on Field Programmable Gate Arrays*, pp. 21–30, 2006.

37. Siwakoti, Y. P., and G. E. Town, "Design of FPGA-controlled power electronics and drives using matlab simulink," in *2013 IEEE ECCE Asia Downunder*, pp. 571–577, IEEE, 2013.

38. Ursutiu, D., M. Ghercioiu, C. Samoila, and P. Cotfas, "FPGA labview programming, monitoring and remote control," *International Journal of Online Engineering*, Vol. 5, no. 2, 2009.

39. Gibson, S. P., *Neural Spike sorting in hardware: From theory to practice.* PhD thesis, University of California, 2012.

40. Yu, B., T. Mak, X. Li, F. Xia, *et al.*, "Real-time FPGA-based multichannel spike sorting using Hebbian Eigenfilters," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, Vol. 1, no. 4, pp. 502–515, 2011.

41. Schaffer, L., Z. Nagy, Z. Kincses, R. Fiath, *et al.*, "Spatial information based osort for real-time spike sorting using FPGA," *IEEE Transactions on Biomedical Engineering*, Vol. 68, no. 1, pp. 99–108, 2021.

42. Zhou, Z., and J. Hu, "A novel square root algorithm and its FPGA simulation," in *Journal of Physics: Conference Series*, Vol. 1314, IOP Publishing, 2019.

43. Koutsos, E., 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 *2013 IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 1922–1925, IEEE, 2013.

44. Biederman-Thorson, M. A., R. F. Schmidt, and G. Thews, *Human Physiology*, Springer Science & Business Media, 2013.

45. Bora, D. J., D. Gupta, and A. Kumar, "A comparative study between fuzzy clustering algorithm and hard clustering algorithm," *arXiv Preprint arXiv:1404.6059*, 2014.

46. Soto-Breceda, A., C. Davey, and M. Stebbing, "SpikeSimulationTool." https://www.github.com/srarty/SpikeSimulationTool, 2019.

47. Quiroga, R. Q., Z. Nadasdy, and Y. Ben-Shaul, "Unsupervised spike detection and sorting with wavelets and superparamagnetic clustering," *Neural Computation*, Vol. 16, no. 8, pp. 1661–1687, 2004.

48. Luan, S., I. Williams, M. Maslik, Y. Liu, *et al.*, "Compact standalone platform for neural recording with real-time spike sorting and data logging," *Journal of Neural Engineering*, Vol. 15, no. 4, 2018.

49. Kalantari, F., H. Hosseini-Nejad, and A. M. Sodagar, "Hardware-efficient, on-the-fly, on-implant spike sorter dedicated to brain-implantable microsystems," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, Vol. 30, no. 8, pp. 1098–1106, 2022.

50. Yu, B., T. Mak, X. Li, F. Xia, *et al.*, "A reconfigurable Hebbian Eigenfilter for neurophysiological spike train analysis," in *2010 International Conference on Field Programmable Logic and Applications*, pp. 556–561, IEEE, 2010.

51. Liu, Y., J. L. Pereira, and T. G. Constandinou, "Event-driven processing for hardware-efficient neural spike sorting," *Journal of Neural Engineering*, Vol. 15, no. 1, 2018.

52. Mohammadi, Z., D. Denman, A. Klug, and T. C. Lei, "Multichannel neural spike sorting with spike reduction and positional feature," *bioRxiv*, 2022.

53. Carta, N., C. Sau, D. Pani, F. Palumbo, *et al.*, "A coarse-grained reconfigurable approach for low-power spike sorting architectures," in *2013 6th International IEEE/EMBS Conference on Neural Engineering (NER)*, pp. 439–442, IEEE, 2013.

54. Schaffer, L., Z. Nagy, Z. Kineses, and R. Fiath, "FPGA-based neural probe positioning to improve spike sorting with osort algorithm," in *2017 IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 1–4, IEEE, 2017.

55. Balasubramanian, K., and I. Obeid, "Massively parallel neural signal processing: System-on-chip design with FPGAs," in *2011 Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, pp. 4609–4612, IEEE, 2011.

56. Bernert, M., and B. Yvert, "Fully unsupervised online spike sorting based on an artificial spiking neural network," *BioRxiv*, 2017.

57. Wang, P. K., S. H. Pun, C. H. Chen, E. A. McCullagh, *et al.*, "Low-latency single channel real-time neural spike sorting system based on template matching," *Plos One*, Vol. 14, no. 11, 2019.

58. Yang, Y., S. Boling, and A. J. Mason, "A hardware-efficient scalable spike sorting neural signal processor module for implantable high-channel-count brain machine interfaces," *IEEE Transactions on Biomedical Circuits and Systems*, Vol. 11, no. 4, pp. 743–754, 2017.

59. Oh, S., S. Han, and I. Youn, "Real-time neural signal sensing and spike sorting system using a modified zero-crossing feature with highly efficient data computation and transmission," *Sensors Mater*, Vol. 29, no. 7, pp. 1031–1042, 2017.

60. Jun, J. J., C. Mitelut, C. Lai, S. L. Gratiy, *et al.*, "Real-time spike sorting platform for high-density extracellular probes with ground-truth validation and drift correction," *BioRxiv*, 2017.

61. Karkare, V., S. Gibson, and D. Marković, "A 130-μW, 64-channel spike-sorting DSP chip," *IEEE Journal of Solid-State Circuits*, Vol. 46, no. 5, pp. 1214–1222, 2011.

62. Hwang, W.-J., W.-H. Lee, S.-J. Lin, and S.-Y. Lai, "Efficient architecture for spike sorting in reconfigurable hardware," *Sensors*, Vol. 13, no. 11, pp. 14860–14887, 2013.

63. Shi, Y., A. Ananthakrishnan, S. Oh, X. Liu, *et al.*, "A neuromorphic brain interface based on rram crossbar arrays for high throughput real-time spike sorting," *IEEE Transactions on Electron Devices*, Vol. 69, no. 4, pp. 2137–2144, 2021.

64. Williams, I., S. Luan, A. Jackson, and T. G. Constandinou, "Live demonstration: A scalable 32-channel neural recording and real-time FPGA based spike sorting system," in *2015 IEEE Biomedical Circuits and Systems Conference (BioCAS)*, pp. 1–5, IEEE, 2015.

65. Valencia, D., and A. J. Alimohammad, "An efficient hardware architecture for template matching-based spike sorting," *IEEE Transactions on Biomedical Circuits and Systems*, Vol. 13, no. 3, pp. 481–492, 2019.

66. Xu, H., Y. Han, X. Han, J. Xu, *et al.*, "Unsupervised and real-time spike sorting chip for neural signal processing in hippocampal prosthesis," *Journal of Neuroscience Methods*, Vol. 311, pp. 111–121, 2019.

67. Yu, B., T. Mak, X. Li, F. Xia, *et al.*, "A stream-based hebbian eigenfilter for real-time neurophysiological signal processing," in *2010 Biomedical Circuits and Systems Conference (BioCAS)*, pp. 90–93, IEEE, 2010.

68. Chen, H.-Y., C.-C. Chen, and W.-J. Hwang, "An efficient hardware circuit for spike sorting based on competitive learning networks," *Sensors*, Vol. 17, no. 10, pp. 22–32, 2017.

69. Yi, J., J. Xu, E. Chen, M. Chamanzar, *et al.*, "Multichannel many-class real-time neural spike sorting with convolutional neural networks," *IEEE Open Journal of Circuits and Systems*, Vol. 3, pp. 168–179, 2022.

70. Bernert, M., and B. Yvert, "An attention-based spiking neural network for unsupervised spike-sorting," *International Journal of Neural Systems*, Vol. 29, no. 8, 2019.

71. Leone, G., L. Raffo, and P. Meloni, "ZyON: Enabling spike sorting on apsoc-based signal processors for high-density microelectrode arrays," *IEEE Access*, Vol. 8, pp. 218145–218160, 2020.

72. Do, A. T., S. M. A. Zeinolabedin, D. Jeon, D. Sylvester, *et al.*, "An area-efficient 128-channel spike sorting processor for real-time neural recording with 0.175 μ W/channel in 65-nm CMOS," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, Vol. 27, no. 1, pp. 126–137, 2018.

73. Navajas, J., D. Y. Barsakcioglu, A. Eftekhar, A. Jackson, *et al.*, "Minimum requirements for accurate and efficient real-time on-chip spike sorting," *Journal of Neuroscience Methods*, Vol. 230, pp. 51–64, 2014.

74. Karkare, V., S. Gibson, and D. Marković, "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.

75. Valencia, D., and A. Alimohammad, "Neural spike sorting using binarized neural networks," *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, Vol. 29, pp. 206–214, 2020.

76. Saeed, M., A. A. Khan, and A. M. Kamboh, "Comparison of classifier architectures for online neural spike sorting," *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, Vol. 25, no. 4, pp. 334–344, 2016.