Robust, Automated Methods for Filtering and Processing Neural Signals

Submitted in partial fulfillment of the requirements for

the degree of

Doctor of Philosophy

in

Electrical & Computer Engineering

John W. Kelly

B.S., Electrical Engineering, North Carolina State University
B.S., Computer Engineering, North Carolina State University
M.S., Electrical Engineering, North Carolina State University

Carnegie Mellon University
Pittsburgh, PA

May 2013

For my grandfather,
whose love of science lives on

Abstract

This dissertation presents novel tools for robust filtering and processing of neural signals. These tools improve upon existing methods and were shown to be effective under a variety of conditions. They are also simple to use, allowing researchers and clinicians to focus more time on the analysis of neural data and making many tasks accessible to non-expert personnel. The main contributions of this research were the creation of a generalized software framework for neural signal processing, the development of novel algorithms to filter common sources of noise, and an implementation of a brain-computer interface (BCI) decoder as an example application.

The framework has a modular structure and provides simple methods to incorporate neural signal processing tasks and applications. The software was found to maintain precise timing and reliable communication between components. A simple user interface allowed real-time control of all system parameters, and data was efficiently streamed to disk to allow for offline analysis.

One common source of contamination in neural signals is line noise. A method was developed for filtering this noise with a variable bandwidth filter capable of tracking a sinusoid's frequency. The method is based on the adaptive noise canceling (ANC) technique and is referred to here as the adaptive sinusoid canceler (ASC). This filter effectively eliminates sinusoidal contamination by tracking its frequency and achieving a narrow bandwidth. The ASC was found to outperform comparative methods including standard notch filters and an adaptive line enhancer (ALE).

Ocular artifacts (OAs) caused by eye movement can also present a large problem in neural recordings. Here, a wavelet-based technique was developed for efficiently removing these artifacts. The technique uses a discrete wavelet transform with an automatically selected decomposition level to localize artifacts in both time and frequency before removing them with thresholding. This method was shown to produce superior reduction of OAs when compared to regression, principal component analysis (PCA), and independent component analysis (ICA).

Finally, the removal of spatially correlated broadband noise such as electromyographic (EMG) artifacts was addressed. A method termed the adaptive common average reference (ACAR) was developed as an effective method for removing this noise. The ACAR is based on a combination of the common average reference (CAR) and an ANC filter. In a convergent process, a weighted CAR provides a reference to an ANC filter, which in turn provides feedback to enhance the reference. This method outperformed the standard CAR and ICA under most circumstances.

As an example application for the methods developed in this dissertation, a BCI decoder was implemented using linear regression with an elastic net penalty. This decoder provides automatic feature selection and a robust feature set. The software framework was found to provide reliable data for the decoder, and the filtering algorithms increased the availability of neural features that were usable for decoding.

Acknowledgments

I would like to thank the great number of people who have contributed to the completion of this work, starting with my committee members. To my advisors, Dan Siewiorek and Asim Smailagic, the freedom and encouragement you both gave me in pursuing this research was invaluable, as was your time and guidance. Wei Wang, thank you for welcoming me as a part of your lab and for providing the resources, expertise, and much of the motivation for this dissertation. Also to Richard Stern, thank you for a class that laid the foundation for much of this work and as with the rest of my committee members, for your valuable time and knowledge.

I must also thank my family, starting with my parents. Your support, sacrifices, and perhaps most importantly your patience, ensured that I always had every opportunity to pursue my goals. Without your constant love and guidance I would not be the person I am today. Patrick, you always set a high standard to shoot for that has helped me accomplish many things, and you were also there when I needed to be kept in check. To Jessi, my wonderful wife, thank you for the incredible dedication and unwavering support you have shown. I could not have made it through these years without you and I know at times it was as hard for you as it was for me.

Without my labmates at hRNEL none of this research would have been possible. I would like to especially thank Alan Degenhart and Robin Ashmore, whose collaboration and feedback paved the way for a large part of this work. Jen Collinger, Stephen Foldes, and Brian Wodlinger also provided valuable input and discussions, and although they only recently arrived John Downey and Bridget Endler helped get me through many days of writing this dissertation.

I should also thank Brian French and Scott Fisk, who provided a great office environment at CMU. I would also like to thank the professors at CMU who impacted my work, including Jelena Kovačević, Markus Püschel, Xin Li, and Vijayakumar Bhagavatula.

There are many people to thank for providing data, beginning with Gustavo Sudre, Dean Pomerleau, Rob Gaunt, and Doug Weber. Their studies provided MEG data, for which Anna Haridis and the Center for Advanced Brain Magnetic Source Imaging (CABMSI) at the University of Pittsburgh Medical Center should also be thanked. For ECoG data, many of the same people from hRNEL are to thank, but also Elizabeth Tyler-Kabara, Michael Boninger, and Aaron Batista.

Finally, I owe a debt of gratitude to those who have kept the past few years enjoyable. Many of the same people already mentioned should be thanked, but there are also many others that are too numerous to list. I should especially thank my teammates, including those from the Gigahurtz and the Incapacitors. Also, an enormous thank you must be given to Jack Hoffman and the Golden Triangle Waterski Club, who put a boatless grad student out on the river for some great skiing.

This work was supported by a National Defense Science and Engineering Graduate Fellowship sponsored by the Air Force Office of Scientific Research, an NSF Graduate Research Fellowship, and the Quality of Life Technology Center under NSF Grant No. EEEC-0540865.

Contents

List of Figures

List of Tables

List of Acronyms

ACAR	adaptive common average reference
ALE	adaptive line enhancer
ANC	adaptive noise canceling
AR	autoregressive
ASC	adaptive sinusoid canceler
BCI	brain-computer interface
BSS	blind source separation
CAE	cumulative angle error
CAR	common average reference
ECoG	electrocorticography
EEG	electroencephalography
EKG	electrocardiogram
EMG	electromyographic
EOG	electrooculographic
FFT	fast Fourier transform
FIR	finite impulse response
GUI	graphical user interface
ICA	independent component analysis
IIR	infinite impulse response
lasso	least absolute shrinkage and selection operator
LMS	least mean squares
MEG	magnetoencephalography
MSE	mean squared error
NLMS	normalized LMS
OA	ocular artifact
OLS	ordinary least squares
PCA	principal component analysis
SNR	signal-to-noise ratio
STD	standard deviation
TAE	timepoint angle error
TCP	transmission control protocol
UDP	user datagram protocol

Chapter 1

Introduction

1.1 Motivation

The study of human brain function can benefit both engineering and medicine. Clinical neural monitoring is critical in diagnosing and treating many neurological disorders such as epilepsy. Neuroscience research can help find the causes and cures for many of these same disorders. The development of brain-computer interfaces (BCIs) presents the possibility of creating a direct link between humans and their environment. This link could allow the use of brain-controlled devices to assist people with disabilities [1]. These sophisticated systems have been able to achieve real-time operation of assistive technology such as computer cursors and prosthetic arms [2], [3]. People with severe neurological or physical impairments could benefit greatly from such devices that restore even a small fraction of their lost function.

In part due to advances in neural recording techniques and computing power, there has been a steady increase in the availability and quality of recorded neural data. Analyzing these signals is a difficult process that requires extensive time, knowledge, and training. Often the raw signals must go through multiple conditioning steps including the filtering of noise and artifacts. This process can often impede researchers and clinicians, even those experienced in signal processing and software development, by taking away time that could be devoted to analysis of the data. Furthermore, the required skill set in processing neural signals hinders many potential technological advances, such as the practical implementation of BCIs in real-world settings. For BCIs both to become economically sustainable and to meet the goal of assisting those with disabilities, they must be operable by a user who has minimal training.

It is important then that tools be available for neural signal processing that can effectively perform commonly needed tasks in a simple and automated fashion. Researchers and clinicians could then minimize time lost to these tasks so that their focus could remain on producing the technology that fully harnesses the available neural data. This technology would also be accessible and beneficial to a larger population. In order to achieve this desired impact, such tools need to be packaged with a generalized framework for neural signal processing that can easily be adapted to users' specific needs.

1.2 Background

1.2.1 Software Framework

The need to create a framework to test and implement neural signal processing tasks is not new, and there are many packages available for use. These packages vary from highly complex, specialized software created for custom use, to small applications that perform a single task. Multiple software platforms for recording and viewing neural signals exist, and packages are also available for different types of analysis on these signals. BCI software even exists that can record, view, and process neural signals while presenting feedback to a user. While many of these packages do fill specific needs and provide an excellent model for a framework, there is still much room for expansion and improvement in this area.

The goal for a generalized framework would be to allow researchers to spend less time creating the software that encapsulates their experiments, and to make neural signal processing tasks more accessible to those without extensive programming experience. To meet these goals the framework should provide simple, built-in functionality for a variety of common tasks in neural signal processing. Additionally, it must be easily extendable. Finally, as with any software, data integrity and consistent system execution must be maintained.

1.2.2 Filtering Neural Signals

One common task in neural signal processing software is effective signal conditioning, including filtering noise and artifacts. Artifacts can be quite prevalent in neural data and can come from a variety of sources, including eye movement, muscle movement, cardiac rhythm, outside sources, and even neural processes other than the one of interest [4]. If the recordings are from a human subject then the best solution is sometimes to instruct the subject to avoid producing some of these artifacts. This is not always practical, though, due to the conditions of the experiment, environment, or the subject [5]. Often such noise can be detected and the contaminated data then ignored, but loss of data is not an optimal solution, especially if the recording is being used in real-time [6].

The only consistently viable solutions for dealing with these artifacts are to either remove them or, in the case of BCIs, to develop decoding algorithms that are invariant to them. The latter method is not always possible, though, if the artifact is contaminating important neural features. Also, it is difficult to prove invariance without an artifact-free signal for comparison. Thus, artifact removal is highly beneficial to both BCI and general neuroscience research [7], [8].

One other difficulty in noise removal from neural signals is that in many cases the people analyzing the signals have their expertise in areas outside of signal processing. For BCI applications the end goal is to have the user operate the system unassisted, so in that case the signals might not even be monitored. It would be useful then to be able to filter out noise with methods that are not only effective, but easy to implement and capable of operating semi-autonomously.

Line Noise

One of the most prominent sources of corruption in neural recordings is line noise, which is caused by the power line transmission frequency. Elimination of this contamination has been an active area of research, but many methods implemented still fail to effectively eliminate the interference while minimizing distortion of the signal. One common practice is to use a fixed notch filter centered around the average power line frequency. The main problem with this approach is that power line frequency varies around its mean, so the notch must be wide enough to account for this variation [9]. Increasing the notch width increases the possibility of also removing interesting physiological data. Other common approaches include low pass filtering below the power line frequency or doing a spectral analysis of the signal and ignoring those frequencies near the contamination. These techniques could also discard useful data.

Ocular Artifacts

Ocular artifacts (OAs) caused by eye movement are another common problem in many recordings and can be difficult to remove in an efficient manner [7]. An OA typically has a much higher amplitude than that of neural activity and can severely corrupt the data. Attempting to avoid blinking

introduces a cognitive process that alters the neural signals [10], [11]. Numerous methods have been attempted to filter OAs, including ones based on regression, principal component analysis (PCA), and independent component analysis (ICA) [12], [7]. Many of these methods, though, fall short in recovering the neural data and might require manual operation by an expert. The increasing dimensionality of recorded neural data also causes some methods to be computationally demanding.

Broadband Common Mode Noise

Other sources of neural contamination, such as electromyographic (EMG) artifacts caused by muscle movement, are broadband in nature. This property makes filtering the noise or using any part of the neural signal more challenging [13]. EMG artifacts are also usually spatially correlated across multiple channels of a recording. In a physiological recording, this type of multi-channel contamination is often referred to as a common mode artifact. Good referencing, blind source separation techniques, and source localization methods have all been used in attempts to remove this type of neural signal corruption. These methods are not always able to reveal the underlying neural data or operate in real-time, and again can require manual operation by an expert.

1.2.3 Analyzing Neural Signals

Once neural signals are conditioned properly, the next step is typically the identification and analysis of neural features of interest. This is the step at which the needs of each researcher or user begin to diverge. In a clinical setting, a physician might examine the signals for signs of a seizure or other abnormality. In a BCI, this step usually involves a neural decoder which, after performing feature extraction, generates a control signal from the neural features. The decoder can consist of anything from simple linear classifiers that produce an output based on weighted sums of a few features to complex machine learning algorithms.

Due to its importance in a BCI a large amount of time is often spent in setting up and training the neural decoder. Although automated decoding methods exist, the complexity of identifying

the best method to use is an example of a situation that would greatly benefit from a generalized framework with robust neural signal processing methods. By removing the need to focus on other aspects of the system such as filtering the neural signals, BCI researchers are free to devote more time directly to the decoder and to the other scientific questions they are attempting to answer. The decoder itself also has the potential to perform better with signals that have been effectively filtered.

1.3 System Overview

Although neural signal processing systems can vary in their applications and implementations, most of them contain many of the same basic system components: data acquisition, signal conditioning, and some form of decoding or analysis of the signals. In a BCI, feedback is also typically given to the user in the form of some device being controlled. The goal of this research was to provide a fully functional system that could not only provide a framework for these system components, but also perform many of the tasks that are commonly required of them. By building a general framework and automating some of the most difficult and common parts of neural signal processing, both researchers and end-users could benefit. In order to realize these benefits, the system must obtain accurate results while being usable by a non-expert. To do so the system needs to maintain precise system timing, ensure data integrity, minimize set up time, be robust to noise, and be capable of producing a reliable system output. Fig. 1.1 provides an overview of the methods developed to accomplish these goals.

1.3.1 Software Framework

The software framework is an important part of a neural signal processing system. It must provide reliable communication between system components, as well as a user interface powerful enough to control custom experiments and simple enough to be operable by non-experts. The software must also ensure data integrity and maintain precise system timing to provide smooth, accurate

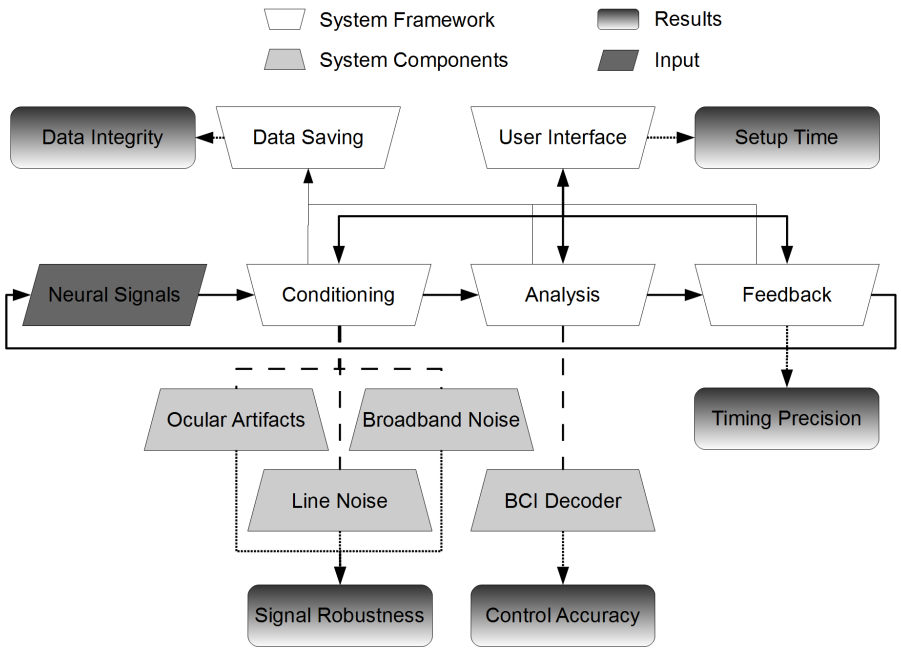

Figure 1.1: An overview of the developed neural signal processing methods. The general setup is outlined by the system framework boxes, with each of these parts broken into the individual system components that were developed. The results indicate the criteria by which the implemented methods were evaluated.

feedback to the user. A generalized framework is presented here that is capable of handling these duties.

The software platform, which is called Craniux, was developed using the LabVIEW (National Instruments) graphical programming environment. LabVIEW has a lower learning curve than many languages, and it also provides many built-in libraries for signal processing and data visualization. Craniux employs a modular system structure to allow for code reuse, extendability, and the ability to be distributed across a network. Additionally, the system internally manages all tasks related to the framework such as communication between components, data saving, and system execution. These features help to make the software reliable, efficient, and easy to use.

1.3.2 Filtering Neural Signals

Due to the susceptibility of neural recordings to noise and artifacts, one of the most common tasks of neural signal processing systems is filtering the data. Accordingly, signal filtering was a primary

focus of the methods developed here. Novel algorithms were developed that can remove OAs, line noise, and common mode broadband noise from neural signals. These removal methods are relatively simple to use, capable of real-time processing, and effective in maximizing noise removal while minimizing signal distortion. Such methods can both help the performance of neural signal processing systems and simplify the setup process.

Line Noise

For line noise removal, the adaptive sinusoid canceler (ASC) was developed. The ASC is an adaptive filter that can track the frequency of drifting sinusoidal noise and narrow its filter bandwidth around it. In this way it is able to specifically target and filter the line noise while minimizing the resulting distortion to the neural signals. This method was found to outperform both standard notch filters with fixed center frequencies and bandwidths, and an adaptive line enhancer (ALE).

Ocular Artifacts

The OA filtering method is based on a wavelet decomposition in which the goal is again to target the OA and make as small of an impact as possible on the actual neural data. This is accomplished by first selecting the best level of wavelet decomposition, then removing the OA with a threshold function in the wavelet coefficients before reconstructing the signal. Methods based on regression, PCA, and ICA were also implemented with the multi-level wavelet technique producing superior results.

Broadband Common Mode Noise

For broadband noise the adaptive common average reference (ACAR) was created, which takes advantage of the spatial correlation of the noise. This again uses an adaptive filter, and it also takes ideas from the common average reference (CAR). A weighted CAR provides a reference for the adaptive filter, and the output of the filter in each data channel is then used to adjust how the reference is created for the next timepoint. The process was found to reliably converge, and

provided better results in most cases than the CAR and ICA.

1.3.3 Analyzing Neural Signals

The analysis portion of a neural signal processing system is the application for which the other components presented in this dissertation are meant to apply. The framework and processing components provided by the methods developed in this work should save time for those who need to focus on the individual needs required at this step. In a BCI, this step usually involves a neural decoder which generates a control signal from neural features. This BCI setup was used in this dissertation as an example application to examine the impact of filtering and processing methods. In keeping with the goal of providing effective and easy to use methods, the neural decoder that was implemented used linear regression with an elastic net penalty to provide automatic feature selection and robust results.

1.4 Contributions

This dissertation provides robust, automated methods for neural signal processing and its main contributions include:

- a generalized software framework for the research and implementation of neural signal processing technologies
- algorithms for filtering common noise and artifacts found in neural signals such as:
  - line noise
  - ocular artifacts
  - common mode broadband noise such as that produced by EMG artifacts
- an implementation of a BCI decoder as an example application

1.5 Organization

The remainder of this dissertation is organized as follows. Chapter 2 presents Craniux, the software framework created for neural signal processing. Chapter 3 covers the ASC, an algorithm developed to remove line noise or other sinusoidal components that drift in frequency. The removal of OAs, including the novel multi-level wavelet method, is discussed in Chapter 4. Common mode broadband noise and its removal using the ACAR algorithm are presented in Chapter 5. Chapter 6 discusses BCI decoding as an example application for the methods developed in this dissertation. Finally, Chapter 7 offers conclusions and discusses potential future work.

Chapter 2

Software Framework

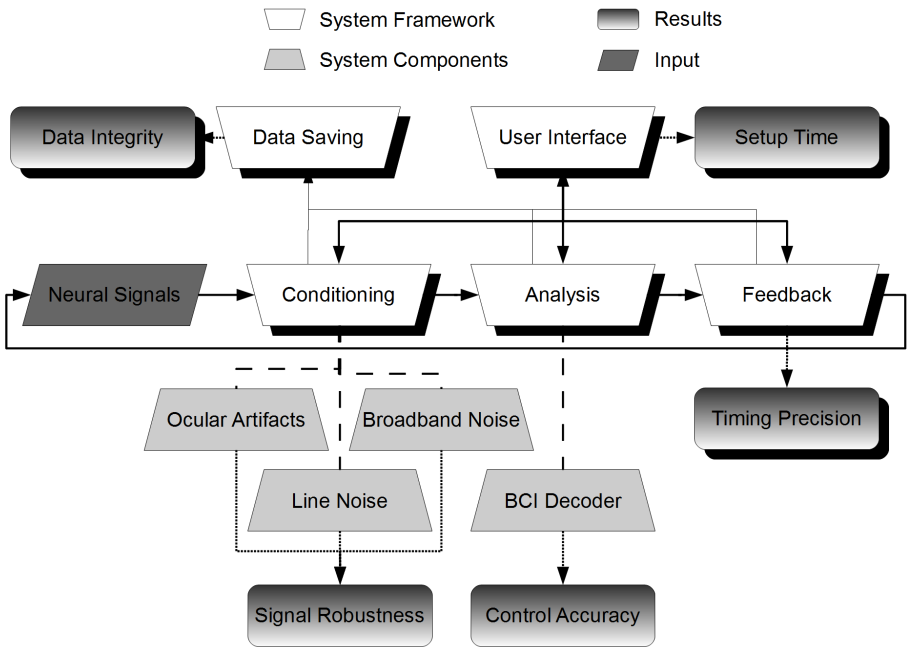

Figure 2.1: Overview diagram highlighting the system framework. *The boxes with shadows indicate the portion of the system that is discussed in this chapter.*

2.1 Introduction

This chapter presents the generalized framework for neural signal processing that is used by the algorithms discussed in later chapters. Fig. 2.1 highlights these portions of the system, including the related overall results. The framework presented here is not only a tool for the implementation of algorithms in this dissertation; it also serves a need in itself. The creation of a powerful and accessible system for processing neural signals must include the software framework.

In neural signal processing the time and difficulty involved in creating or modifying the required software can be quite large. This time sink can sometimes be due to a lack of the personnel's coding expertise, but it can also be attributed to the complexity of the systems involved. In brain-computer interface (BCI) studies, for example, researchers often need to implement different recording modalities, signal processing methods, feature extraction techniques, decoding algorithms, and applications.

To allow researchers to maximize the time spent investigating important algorithms and scientific questions, it is necessary for software to be available that can reduce the overhead involved

in experimental setup and analysis. This software must be able to adapt to users' differing needs and address their wide range of potential methods and applications. To provide such a framework a software package, called Craniux, was developed. Some of the software's goals are listed below, with a keyword that is used throughout the text when referring to each goal. These goals tie in directly with this dissertation's overall aim of providing effective and easy to use methods for neural signal processing. In relating these goals to Fig. 2.1 the first one is directly tied to data integrity, the second to timing precision, and the last two to setup time.

1. Reliability - Maintain data integrity and stream all necessary data to disk for analysis in an offline environment.

2. Performance - Provide consistent real-time execution with data visualization and parameter updates.

3. Ease of use - Be able to efficiently test common experimental paradigms and parameters with minimal setup time and training needed by the user.

4. Extendability - Allow experimenters to extend the software by developing custom modules with minimal time spent on overhead such as the system framework and user interface.

Craniux achieved these goals with a modular, distributable system coded in LabVIEW (National Instruments). This open-source software was created at the Human Rehabilitation and Neural Engineering Laboratory (hRNEL) with the primary development team consisting of Alan Degenhart, Robin Ashmore, and the author. Much of this material was earlier published in [14]. The next section discusses related software, while Section 2.3 discusses the design and implementation of Craniux. Section 2.4 presents some system performance measures and discussion of the goals listed above. Finally, Section 2.5 offers conclusions along with some of the possible implications and impacts of the software.

2.2 Related Software

Most software for neural signal processing targets specific needs without offering the flexibility to easily customize or extend existing capabilities. A large number of packages are available for monitoring, recording, and replaying neural signals. This type of software is commonly used in clinical settings and provides simple filtering and visualization tools. Some packages, such as CURRY (Neuroscan) and BESA (BESA GmbH) [15], offer more advanced analysis and brain mapping functions. Such software is typically not easily modified or extended, though. Some similar open source packages are also available, such as EEGLAB [16] and Brainstorm [17].

BCI researchers typically have a wider array of needs than other potential end users of neural signal processing software [18], [19]. In many cases these needs result in the development of highly specialized software packages [20], [21]. Some software packages, however, have attempted to serve a broader range of BCI researchers by being more adaptable to users' needs. BCI2000 is one successful package for general purpose BCI research [22]. This open-source software has seen great success in part due to its modular design, which allows the user to select one module each for acquisition, signal processing, and the application. Although the development of custom modules does require programming knowledge in C++, BCI2000 provides a user datagram protocol (UDP) interface for communicating with 3rd party software as well as the ability to create new modules in higher level languages such as Python.

Other software targeting BCI research has also seen recent success. One such platform is OpenViBE, which is again written in C++ and has a modular design [23]. One advantage of OpenViBE is the ability to use a graphical environment to arrange any number of modules into a BCI system. The TOBI Common Implementation Platform (CIP) is another effort to improve the effectiveness and reusability of BCI software [24]. The CIP provides a set of common interfaces that allows communication between individual components from different systems and software platforms through a standardized protocol.

Each of these software packages fills needs in neural signal processing. The system presented here, Craniux, builds upon their success. Its goal is to use the strengths of these platforms and add

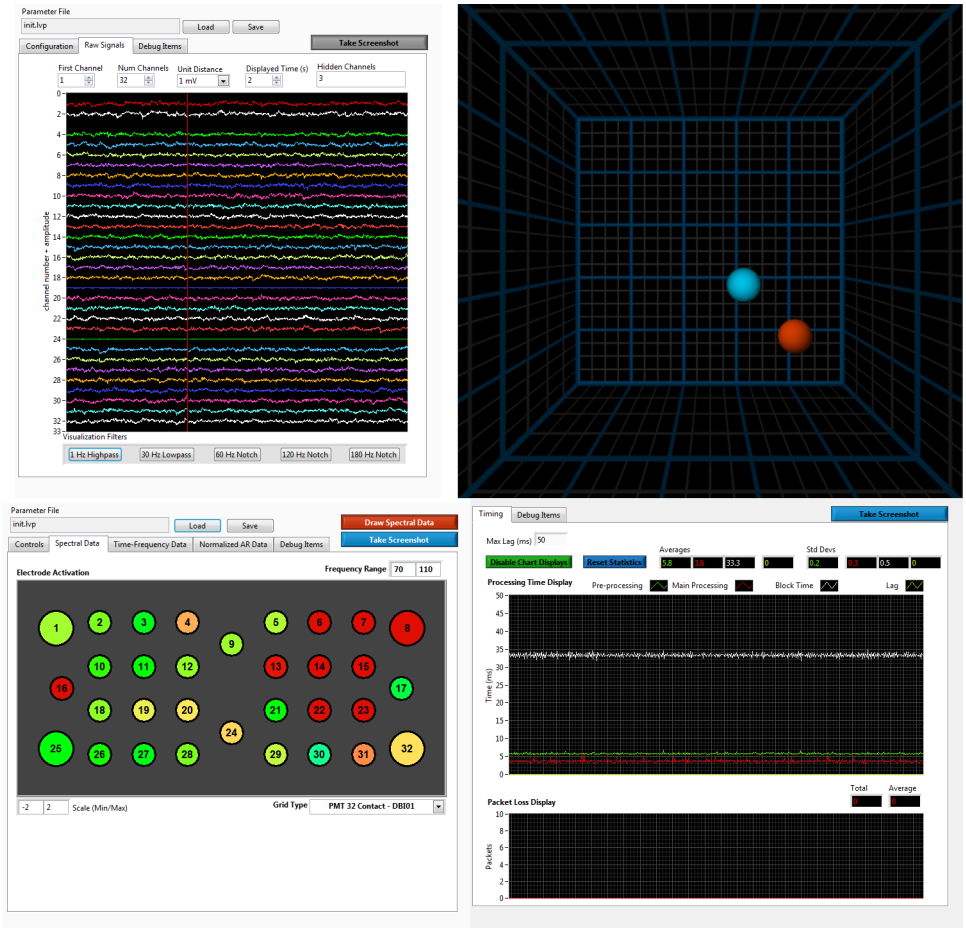

Figure 2.2: Craniux screenshots showing real-time display of (Top Left) neural signals, (Top Right) a BCI cursor task in a 3D environment, (Bottom Left) normalized high-gamma band power on a 32 channel electrode layout, and (Bottom Right) system timing and packet loss information.

additional capabilities to create a multi-use neural signal processing software suite that minimizes the time involved in experimental setup and analysis. In doing so, it would be a valuable tool for a wide array of applications for end users, researchers, and developers.

2.3 Craniux System Design

Craniux was implemented using the LabVIEW programming language. One of the main reasons for choosing LabVIEW was the high-level graphical programming environment. This environment has a lower learning curve than traditional text-based languages such as C or C++, which increases the software's accessibility to researchers who want to create custom code and do not have a

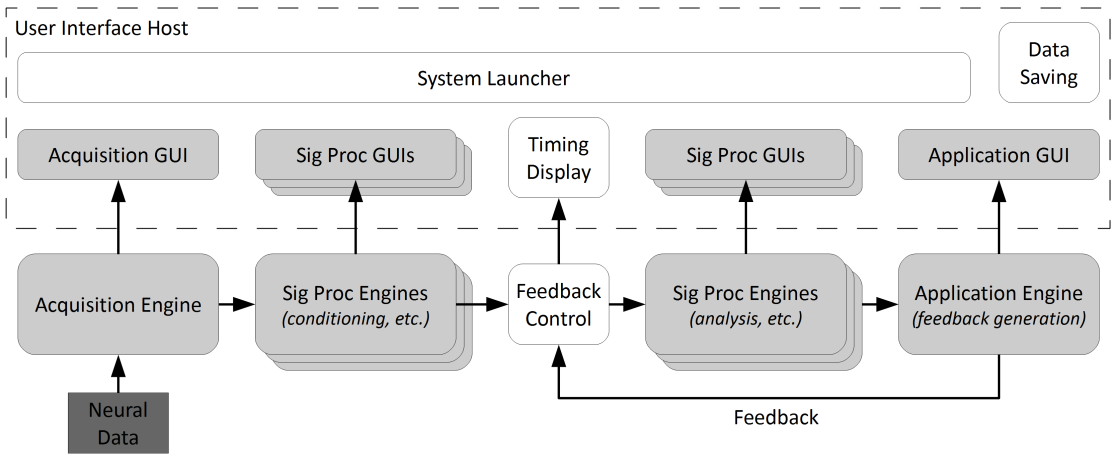

Figure 2.3: Craniux system framework. The Craniux system consists of interchangeable modules that are arranged to perform the individual tasks in an experimental paradigm. Each module (light gray) has an associated GUI for parameter updates and data visualization. System framework components (white) automatically handle the software framework including communication, data saving, and system execution. All engines have additional connections not shown, to the system launcher and to the data saving manager.

strong background in programming. The code is inherently multi-threaded, allowing complex systems with a large number of individual components to be executed more efficiently. LabVIEW also provides a large number of tools useful to even experienced programmers, such as real-time debugging and built-in libraries for data visualization, signal processing, and hardware integration. Some of these tools can be seen in Fig. 2.2, a screenshot showing various portions of the Craniux system during a real-time experiment.

2.3.1 Top Level Design

Craniux employs a modular system structure as shown in Fig. 2.3. System execution, discussed below, is controlled by the system launcher, which also loads the selected modules and contains system-level parameters. Each module consists of an engine and a graphical user interface (GUI), and modules of the same type are interchangeable. The role of modules in the system framework is discussed in this section, with details regarding the internal structure of modules provided in Section 2.3.2. Communication between components are handled by robust, automated protocols

that allow for network distribution of Craniux, as discussed later in this section. The communication channels, which are described further in Section 2.3.3, exist between each engine and its GUI, between engines, between each engine and the data saving manager, and between each engine and the system launcher. The data saving manager streams all necessary data to disk in parallel to main system execution. This process is discussed in Section 2.3.4.

System Execution

Craniux initially loads with only the system launcher. The launcher provides a simple interface with which modules and system parameters can be manually entered, or from which a parameter file with this information can be loaded. Parameter files are created by saving any created configuration. Once the selected modules are opened, system execution begins in a suspended state. In this state all system functions are operating, but no data is being saved. Parameters of any of the modules can be updated during this time, and data can be visualized as well. The user can then use the system launcher to put the system in its running state to begin an experimental trial. At this point data saving begins, with each engine and the launcher communicating with the data saving manager. From there the data is streamed to disk in parallel to ensure that the time-consuming writes do not hinder main system execution. At any time the user can put the system back into its suspended state or stop execution altogether.

Determinism and data integrity between components are maintained through the dataflow driven system design. The engines located prior to the feedback control module (shown in Fig. 2.3 as the acquisition engine and the first block of signal processing engines) are free to execute as fast as data arrives. After this chain of execution the data can be buffered if necessary in the feedback control module, which waits for feedback from the application engine. The buffer only protects against a spike in system timing, as only a user-specified amount of data is buffered before data begins to be dropped. Once both feedback and data from the previous module arrive at the feedback control module, both are passed into the second block of signal processing engines. The feedback control module also keeps track of system timing including the system's refresh rate,

Table 2.1: Currently available Craniux acquisition modules.
** indicates modules for which the author was a primary developer*

Engine Name	Description
Acquisition Template*	Generates random data, available for development and testing
FieldTrip Buffer	Reads data from a FieldTrip buffer [25]
gUSBamp*	Reads data from g.USBamp (g.tec) amplifiers
Replay Data*	Reads data stored in a MATLAB (MathWorks) structure
Ripple	Reads data from grapevine system (Ripple)
RTMA	Reads data using the Real-Time Messaging Architecture [26]
SimECoG*	Generates simulated ECoG data
UDP Binary	Reads raw data transmitted via UDP

the amount of time system processing is lagging behind data acquisition, and the processing time for both the portion of the system before and the portion after the feedback control module. This information, along with the number of dropped packets, is sent to a chart on the user interface host so that system timing information can be viewed in real-time. This chart is shown in the bottom right of Fig. 2.2.

Module Types

Craniux modules are fully interchangeable with modules of the same type. The type can be either acquisition, signal processing, or application, and the placement of these modules in the system is shown in Fig. 2.3. Each type has a template that can be used to quickly develop new modules. There are also a large number of built-in modules that implement common neural signal processing tasks. Below is a further description of the built-in modules and the responsibilities of each module type as related to the overall system.

Acquisition modules are responsible for reading and parsing neural data. If the data does not have its own external timing source then these modules must also control system timing. Currently available acquisition modules are shown in Table 2.1. These modules include interfaces with recording systems, a simulated electrocorticography (ECoG) data source for testing new signal processing or application modules, a method for replaying previously recorded data, and a UDP

Table 2.2: Currently available Craniux signal processing modules.
* *indicates modules for which the author was a primary developer*

Engine Name	Description
Add Noise*	Adds simulated noise to the data
ALE*	Adaptive line enhancer
Artifact Rejection	Flags data contaminated by detected artifacts
DAQ Code Sender	Sends data to a National Instruments DAQ device
Glove 5DT	Stores data from a 5DT data glove
Linear Classifier*	Outputs weighted combinations of input features
Neural Decoder	Trains various decoding algorithms, saves and loads parameters
Replay RTMA	Sends reply using Real-Time Messaging Architecture [26]
Signal Filtering*	Applies various filters, references, and artifact removal tools
Signal Processing Template*	Available for development and testing
Spectral Estimation*	AR spectral estimation using the Burg method [27]

connector.

Signal processing modules are the most generic, as they can perform any action that alters the data between the source and the target application. These actions typically include signal conditioning or analysis. In Fig. 2.3 these modules are split into two blocks. There are no structural differences between the modules in these two blocks, only a difference in what type of information the module receives and when it receives it. Signal processing modules in the first block are free to execute as soon as data arrives from the previous module, but do not receive feedback data from the application. In the second block feedback data is received so these modules must wait on data from two sources. Any signal processing module can be placed in either block, but the user should decide for a given experimental setup which modules need feedback data.

Currently available signal processing modules are shown in Table 2.2. As can be seen some modules perform tasks like interfacing with additional hardware or simulating data, rather than the usual signal processing duties. The neural decoder module implements some common decoding algorithms used in BCIs such as the Kalman filter [28] and ordinary least squares (OLS) regression. OLS is sometimes referred to as the optimal linear estimator (OLE) in BCI literature [29]. The signal filtering module implements some common methods used in neural signal processing, as

Table 2.3: Currently available Craniux application modules.
** indicates modules for which the author was a primary developer*

Engine Name	Description
Application Template*	Maintains dataflow, also available for development and testing
Circle Drawing	Application for drawing circles and ellipses
Cursor Control	Cursor control task with target presentation
DEKA Arm Control	Controls a DEKA prosthetic arm
FES	Controls a functional electrical stimulation system [30]
Flip Book	Uses control signal to move through a flip book
Game Control	Controls simple built-in games for 1D and 2D BCI control
MPL	Controls a Modular Prosthetic Limb [31]
Path Task	Displays a path on a monitor that can be traced
Stimulus Presentation*	Sequentially presents various stimuli on a monitor
Virtual Arm Control	Controls an arm in a Unity (Unity Technologies) environment
WoW Controller*	Allows control of a character in World of Warcraft (Blizzard)

well as many of the algorithms presented in other chapters of this dissertation such as the adaptive sinusoid canceler (ASC) and the adaptive common average reference (ACAR). In the top left, Fig. 2.2 shows data visualization in the signal filtering module.

The implementation of the Burg algorithm in the spectral estimation module used non-recursive calculations, as it was found that the fully recursive implementation carried a risk of instability due to round-off error when operating on a band-limited signal. Eliminating any chance of data corruption was a higher priority than timing improvements. Spectral estimation did benefit greatly from LabVIEW's multi-threading, though, as the calculation for each channel of data could be performed in parallel. The bottom left of Fig. 2.2 shows the layout of a 32 channel electrode grid with visualization of high-gamma band power from the spectral estimation module.

Application modules are responsible for displaying any form of feedback or stimulus to the user, whether it be a pure stimulus or something that is being controlled by the user in a BCI setup. Currently available application modules are shown in Table 2.3. Most of these modules are designed for BCI tasks, with the main exception being stimulus presentation. In a BCI, the application must send information back to any signal processing modules, such as decoding algorithms, that need to know the feedback state. The feedback from the cursor control module is shown in the top left

of Fig. 2.2. The stimulus presentation module can display any sequence of text, image, video, or audio stimuli including pseudorandom and looped sequences. For some experimental paradigms, though, it would be desired to not have any application at all. In this case the application template could be used to return blank feedback and maintain system data flow.

Network Distribution

Although Craniux is multi-threaded and can be run effectively on one computer, the system can also be distributed across a network. As shown in Fig. 2.3, all GUIs, as well as the system launcher, data saving manager, and timing display, are located on the user interface host. All engines, which handle a module's main processing, can also be on this host for a local configuration. With two hosts all engines can be placed on the second host, which ensures that visualizations and user interactions do not interfere with data processing. The system can be further distributed all the way to the point at which each engine is on a separate host. Distributing the system also allows experimental setups in which a subject and a researcher are using different computers, possibly even in remote locations.

2.3.2 Module Design

A main difficulty in Craniux is in allowing parallel control, processing, and visualization while enforcing determinism throughout the system. A large portion of this problem is handled by the system's dataflow-driven system execution discussed in Section 2.3.1. No module is free to begin the data processing portion of its execution until receiving data from the previous module. Some processes within each module, though, such as GUI events, data saving, or training a decoding algorithm, can be handled in parallel to allow system execution to continue. Preventing race conditions and data corruption for these cases must be handled at the module level.

Each module consists of an engine, which handles the actual system execution and data processing, and a corresponding GUI, which is a stand alone application that provides data visualization and allows module parameters to be updated. This separation allows GUI interactions and

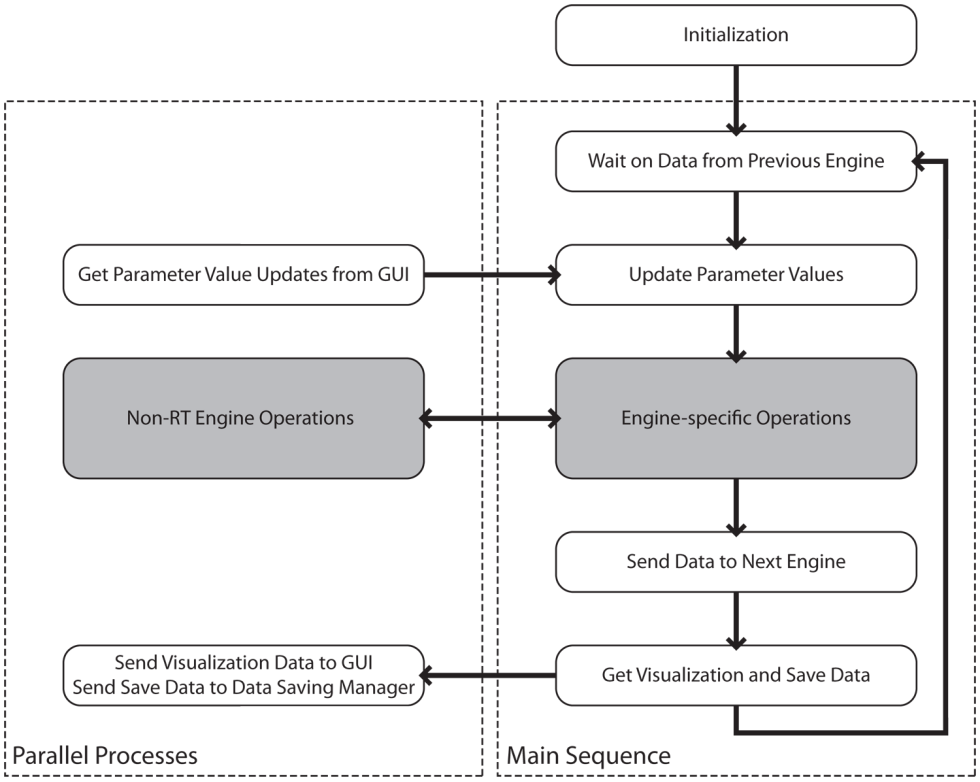

Figure 2.4: Craniux engine execution [14]. Engines in the Craniux system are structured in a way that enforces determinism and data integrity, while allowing efficient execution through multi-threading. Core functionality is indicated by the white boxes and is provided in engine templates. The researcher then needs only to implement the specific algorithm or idea of interest, indicated by the gray boxes.

processor intensive visualizations to occur without interfering with time critical system processes. The GUI displays visualization data and sends parameter updates as soon as possible. As with the system launcher, a parameter file can be loaded. The engine ensures that these events are handled in a way that maximizes efficiency and ensures determinism. The basic execution sequence of an engine is shown in Figure 2.4.

When an engine is loaded it begins execution by initializing. This process includes establishing connections to other system components, identifying those variables that need to be sent over each connection, and loading a parameter file if specified. After initialization the engine begins waiting on data from the previous engine (or acquiring the first block of data if it is an acquisition engine). Once data arrives, the engine implements any parameter changes that were sent from the GUI. Allowing updates at only this point prevents race conditions in which the parameters could be

written to at an unknown time while reads are occurring.

Once an engine has received data and updated parameters it is free to begin execution of its main task. Sometimes an engine might contain a process that only occurs occasionally but takes a long time to complete, such as training a decoding algorithm. To allow the system to continue with real-time execution, these processes are passed off onto another thread until they complete. Upon returning, these processes can write to any relevant engine variables at the step at which parameter values are updated.

When engine-specific operations are complete all necessary data is sent to the next engine so that it can begin execution. While other engines are executing, visualization data is buffered to be sent to the GUI and data that needs to be saved is buffered for the data saving manager. In a separate thread these buffers are continuously monitored and any new packets are removed and sent to their correct destinations. This design minimizes the chance that these operations could affect overall system timing. It does mean that perfect synchronization of the visualization data is not enforced, but any amount of lag would be on the order of milliseconds and imperceptible to the user. The data that is sent to the data saving manager contains all necessary information, including the sample index, so its precise timing is not important. Once all tasks are complete, the engine returns to its state of waiting for data from the previous engine.

2.3.3 System Communication

Communication between Craniux components utilizes self-establishing and self-repairing transmission control protocol (TCP) network connections. These connections provide efficient, reliable data flow and can handle any LabVIEW data type that needs to be transmitted. The establishment of all connections is handled by the system framework as shown in Fig. 2.5. A preliminary connection is first established using LabVIEW's network streams. This connection is used to inform the client of the port number that the host has automatically selected for the TCP connection. The host then waits on the connection attempt from the client. Once the TCP connection is complete, any communication failure causes the connection to be closed and re-established. This process

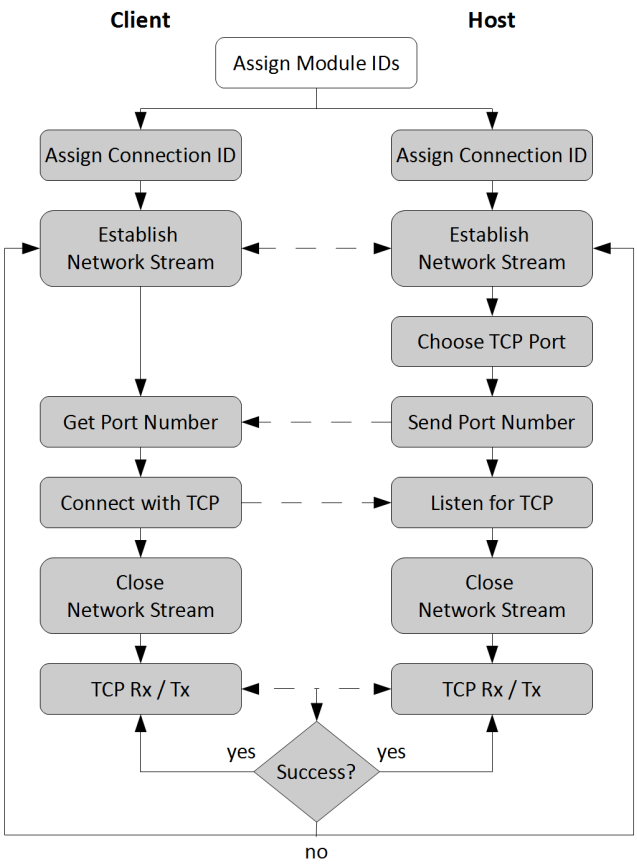

Figure 2.5: Craniux network communications. Communication in Craniux is established through automated, self-repairing TCP connections. The dotted lines represent network communications while the black lines represent process flow. The task in the unshaded box is performed by the system launcher.

requires no configuration from the user.

All network connections use the TCP protocol. TCP was chosen over UDP because its superior reliability is important in a dataflow driven program; a dropped packet between engines would break the dataflow and leave each engine waiting for data that would never arrive. It is also important to note that Nagle's algorithm [32] was disabled for all connections used in the Craniux system. Nagle's algorithm attempts to reduce TCP packet overhead and bandwidth usage by intentionally delaying transmission so that multiple packets can be combined before being sent. Here, the latency introduced by this algorithm is unacceptable, and bandwidth usage is not a concern. The concept behind Nagle's algorithm is retained, however, as all data to be sent simultaneously is combined into a single packet before transmission.

When creating a new module, the developer must implement a communications class that inherits from the base communications data class. The base class contains definitions and methods for communicating standard variables, so only additional variables and methods needed by a module must be defined. This same class can then be used on the receiving side of the connection to read, parse and write the correct values to each variable.

2.3.4 Data Saving

The Craniux data saving process, shown in Fig. 2.6, is designed to reliably stream data to disk with minimal impact on system timing. The data saving manager is a stand alone application that launches whenever Craniux enters its running state. This application establishes TCP connections with all engines as described in Section 2.3.3. When an engine is ready to save data (Fig. 2.4) it enqueues the necessary data, including a sample index, in a first in, first out buffer. In a separate thread any available data is continuously removed from this buffer and sent to the data saving manager. This design minimizes the impact of saving on system performance and also allows all data to be saved in the same file on one host computer. The maximum size of the data saving buffer is limited only by LabVIEW memory availability, but as long as the write speed of the disk is faster than the read speed of the incoming data a large backup will not occur. For example, a typical system setup that samples 32 channels of single-precision data at 1200 Hz and saves the data and 20 neural features per channel once every 40 samples would require an absolute minimum write speed of approximately 2 MB/s. This speed is far below the capabilities of modern hard drives.

The data is separated into two categories for each engine: sampled variables and controls. The first category contains data that is saved for every iteration of the system, such as the raw neural data. The second category contains parameters that only change through user input, and thus only need to be saved when these changes occur. For both categories the current sample index is saved along with the data to ensure that all data can be aligned to reconstruct the experiment in an offline setting. As with system communication, a developer creating new modules must implement a class that inherits from the base data saving class and adds the custom variables and methods necessary

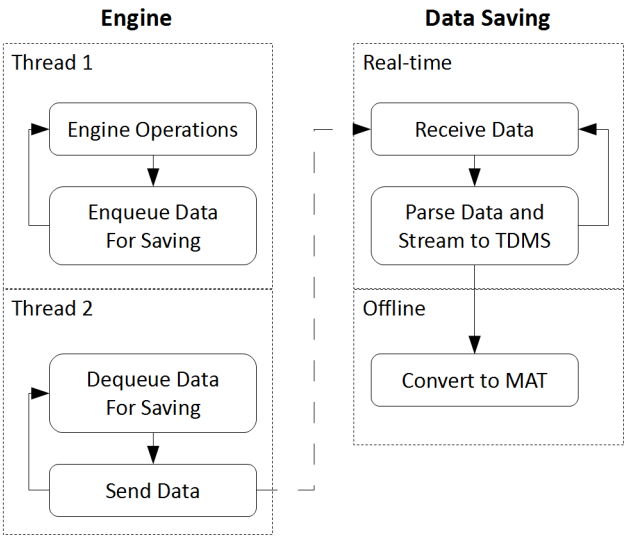

Figure 2.6: Craniux data saving process. Data saving in Craniux follows a process that reliably streams data to disk for offline analysis while minimizing the impact on system timing. The dotted lines represent network communications while the black lines represent process flow.

for that module.

Data from each experimental trial, designated by the time period during which the system is in its running state, is reliably streamed to a file in LabVIEW's TDMS format. This format was specifically created for efficiently streaming data to the hard drive [33]. Craniux also contains a separate application that can be used to convert Craniux TDMS files into the MAT-file format usable by MATLAB (The MathWorks, Inc.). For offline analysis MATLAB is the software of choice for many researchers, so the ability to convert to this format greatly improves data accessibility. Furthermore, any data that is modified or created in MATLAB can be loaded and replayed through the Craniux system by using the appropriate acquisition module.

2.4 System Evaluation

The goals laid out in Section 2.1 provide a means for measuring the effectiveness of Craniux. Of these objectives it is difficult to quantitatively assess the last two: ease of use and extendability. These goals can be assessed, though, by examining the requirements that are placed on a user or

developer. The first two criteria, reliability and performance, can be more precisely measured. Reliability is essentially a binary measure, in that it either is or it is not reliable. Any lack of data integrity at all is not desirable, so it should be the case that no data is lost and that in an offline environment the experiment can be fully reconstructed. Performance is easily examined by the timing and consistency of system execution.

2.4.1 Reliability

Reliability is the most important of the four goals, as data integrity is a top priority with nearly any potential Craniux task. An initial test on the data can be performed just by ensuring that no data packets are lost. This check is simple and can be seen in real-time during the experiment on the timing window. It is also easy to check for dropped packets by looking at the saved data offline. The dozens of Craniux data files used throughout this dissertation contained no dropped packets. The ability of the system to operate in real-time without losing data, though, depends on the system's processing time being faster than the data acquisition rate, which is examined further in the next section.

Just checking for dropped packets alone does not ensure data integrity, though. The data must be aligned between modules and saved correctly. This result can be shown by demonstrating that an experiment with meaningful data can produce good results and be reconstructed offline. To obtain this data, Craniux was used in a BCI experiment with real human ECoG data from an able-bodied subject undergoing subdural epilepsy monitoring. All data collection and procedures were approved by the University of Pittsburgh's Institutional Review Board and informed consent was obtained prior to implantation.

The experimental setup used the gUSBamp module, signal filtering, spectral estimation, a linear classifier, and cursor control. The subject attempted to control vertical cursor velocity to hit targets. The control signal was generated from the high-gamma band power (70-110 Hz) of two neighboring electrodes that showed modulation during overt movement [34]. These spectral features were normalized to zero mean and unit variance using a baseline recording. A push-pull

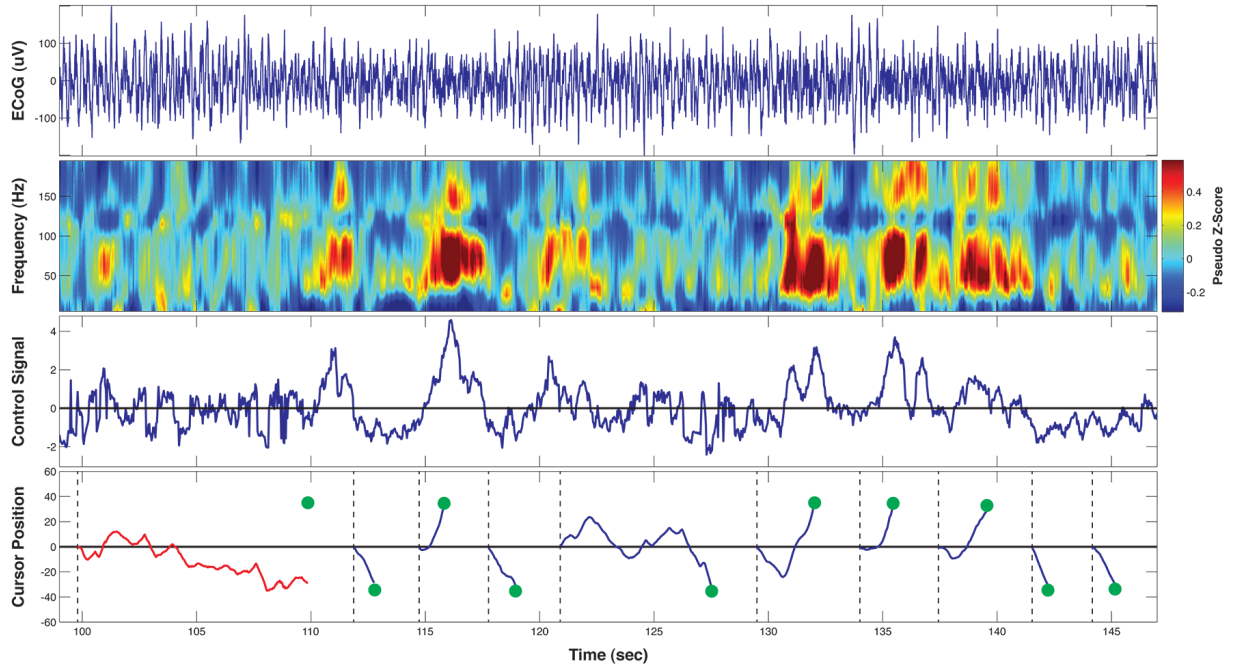

Figure 2.7: Reconstruction of a Craniux experiment [14]. This shows the data from a Craniux BCI paradigm that used real-time human subject ECoG data. The experiment was reconstructed offline in MATLAB using the data saved to disk from Craniux. Shown is (Top) one channel of the raw ECoG data, (Top Middle) the time-frequency data for the same channel, (Bottom Middle) the normalized control signal produced for vertical cursor movement, and (Bottom) the resulting vertical cursor position over time, with the target position shown by the circles and trial onsets shown by vertical dashed lines.

scheme was used for the control signal, as shown in (2.1), where c is the control signal, p_1 and p_2 are the spectral features, a is a gain factor, and θ is an offset.

$$c = a(p_1 - p_2) - \theta \tag{2.1}$$

For the cursor control task the cursor first appeared at the center of the screen. A target was then presented at a location directly above or below the cursor. A trial was successful if the subject was able to hit the target within 10 seconds. Fig. 2.7 shows the experimental results for one session of brain control, during which the subject achieved 88% success. Each step of the experimental process was reconstructed in MATLAB with the saved data, from the raw neural signal all the way to the cursor control. The reconstructed portions of the experiment lined up correctly with each other, and the results matched what was seen during the real-time experiment. This result shows

that Craniux was able to maintain data integrity throughout the experiment and successfully stream all necessary data to disk.

2.4.2 Performance

To evaluate the system performance, Craniux's timing was analyzed for a typical experimental BCI setup. Simulated data was passed into a g.USBamp amplification system from another computer. Craniux sampled this data at 1200 Hz and processed it in 40 sample blocks for a 33.3 ms expected system cycle time. The data was first passed through a band pass filter and standard notch filters for line noise removal. Spectral estimation was then performed for a 25th order autoregressive (AR) model with the Burg method and a 100 ms window. The estimate was divided into 10 Hz frequency bins, with spectral power evaluated at 10 points in each bin. These spectral features were used by a simple linear decoder. The application was a center-out cursor task rendered in a 3D environment, as shown in Fig. 2.2.

Two different timing measurements were evaluated: processing time and system frame rate. There measurements were taken on both a single computer (Windows 7 x64 operating system, Intel Core i7 CPU 920 @ 2.67GHz, 6 GB RAM, NVIDIA GeForce 9800 GT video card) and with Craniux distributed across the network. For the distributed test, the user interface host, with the portions of the system shown in Fig. 2.3, was the computer just described and all engines were located on a second computer (Windows 7 x64 operating system, Intel Core i7 CPU 870 @ 2.93 GHz, 4 GB RAM, ATI Radeon 4550 HD video card). For both configurations, tests were conducted using 8, 16, 32, 64, and 128 channels of data.

The timing measurements were evaluated on 5,000 consecutive blocks, or nearly 3 minutes, of collected data. The processing time measurement determined the amount of time between the arrival of a block of data from the amplifier, and the time when Craniux had finished all processing and feedback for that block of data. These results are shown in Fig. 2.8. As expected, processing time was found to increase with the number of processed channels but remained below the 33.3 ms time required to maintain a consistent frame rate and prevent the loss of data. The increase

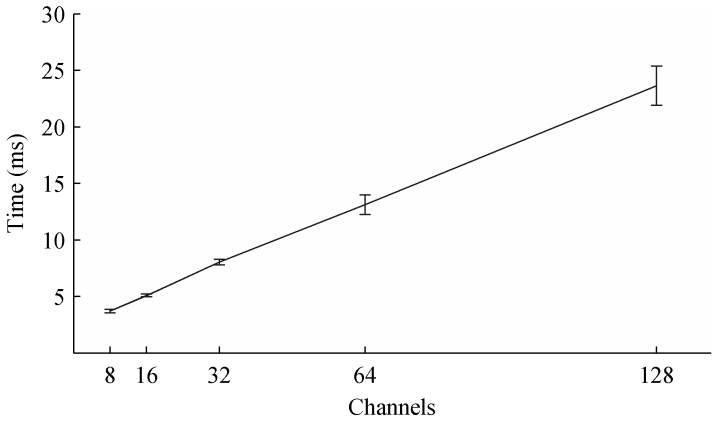

Figure 2.8: Craniux system processing times. The error bars represent standard deviation.

in processing time appears to be linear with the number of channels. It should also be noted that spectral estimation was by far the most computationally expensive portion of the tested system configuration, and any changes to the configuration of that module could greatly affect the system's processing demands.

Distributing Craniux across the network showed slight improvements in processing time for all channel configurations, with the improvement increasing substantially for the 128 channel case. Fig. 2.9 shows the speedup of the distributed system, calculated as the ratio of the processing time for the local configuration over the processing time for the distributed configuration. Since processing time is only required to remain below the frame rate, running Craniux as a distributed system is not necessary unless the system is under a heavy load. The extra processing time made available when Craniux is distributed could easily be utilized to run more complex signal processing algorithms or to decrease the frame rate, though. Network latency must also be taken into consideration when distributing the system.

The system frame rate measurement used the same 5,000 consecutive frames of data. The frame rate is the amount of time from when Craniux begins processing one block of acquired data to when it begins processing the next. The results for the local configuration are shown in Table 2.4. The distributed configuration had no significant effect on this measurement. For all configurations the frame rate was found to be 33.3 ms, precisely what would be expected given the system configuration. Furthermore, the low variability of this timing indicates that the user would

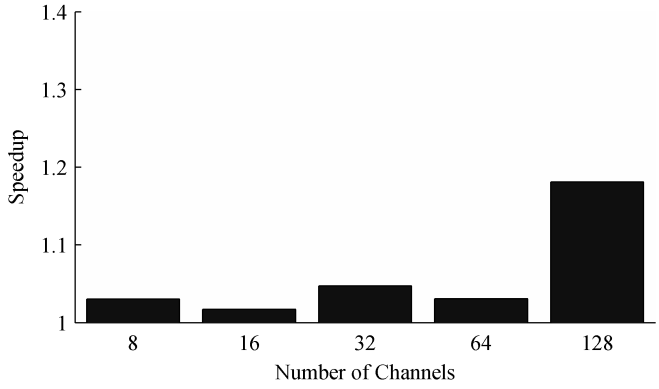

Figure 2.9: Speedup obtained from distributing Craniux across two computers

Table 2.4: Craniux system frame rates. The values shown are mean $\pm$ standard deviation.

Channels	Frame rate (ms)
8	33.3 ± 0.2
16	33.3 ± 0.1
32	33.3 ± 0.2
64	33.3 ± 0.6
128	33.3 ± 1.2

experience a consistent feedback update with no noticeable jitter.

2.4.3 Ease of Use

Craniux's ease of use can be assessed by the requirements placed on a Craniux user in running common experimental paradigms. The interchangeable modules and the straight forward interface provided by the system launcher (Fig. 2.10) and the GUIs greatly help to minimize these requirements. By providing common modules that can be configured and connected according to the user's needs, many experimental setups can be created without any programming knowledge. For example, common acquisition and signal processing methods can be easily tested with different application modules.

Although it is not feasible to provide every existing paradigm as a built-in feature of Craniux, dozens of common experiments can be implemented using modules that are already developed and shown in Tables 2.1, 2.2, and 2.3. With a parameter file, any paradigm with existing modules

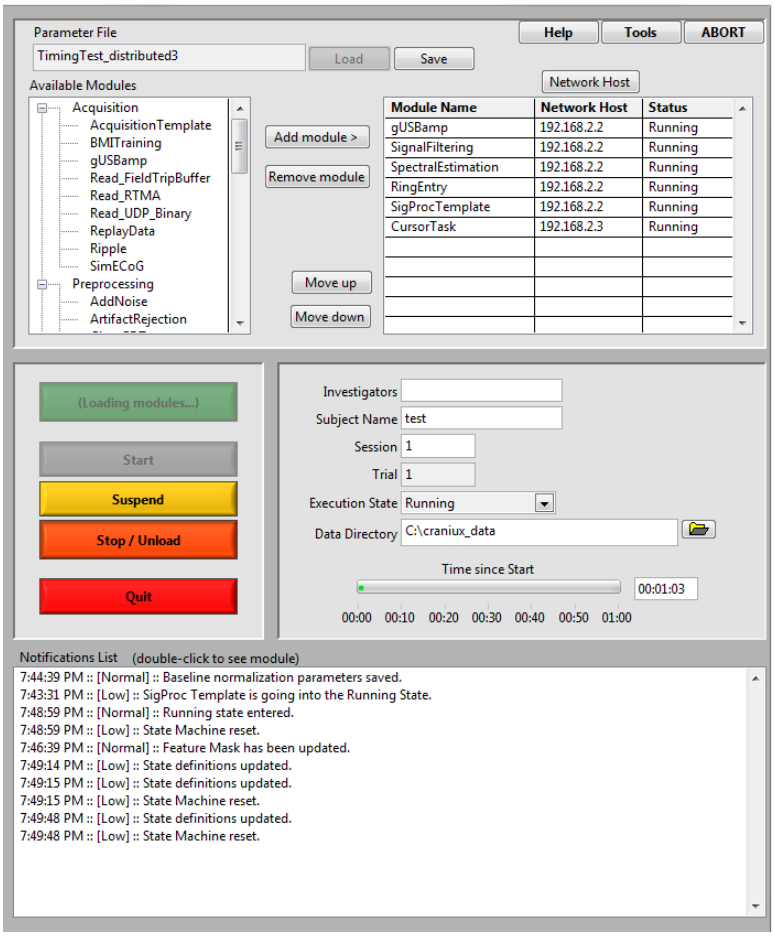

Figure 2.10: Craniux system launcher

could be run simply by selecting the correct file and then clicking two buttons. Manually selecting modules without a parameter file would require slightly more knowledge and time from the user, but with instructions is still easily performed in seconds.

Customization of module parameters takes little additional effort as well, as the parameters can be changed even during run-time through the module's GUI. Updating parameters in this way is highly useful, especially in BCI research where settings might need to be adjusted in response to the feedback being generated by the user. Determining the correct adjustments is also made easier by providing real-time visualization of all important data and system output.

The simplest method of running an experiment would only require one executable that could open Craniux, load the modules, set the parameters, and begin execution. This method would require a separate executable for every possible configuration, though, and would thus limit the

number of experiments that could easily be run through combining different modules and setting custom parameters. As Craniux currently stands it allows a fairly wide variety of paradigms to be tested and customized with its built-in modules, but still requires minimal time and expertise to run. Ultimately, though, the amount of skill required to correctly configure any module depends on the task of the module itself and the design of the algorithm it implements. The main focus of other chapters in this dissertation is in creating algorithms that effectively perform some of the most common tasks in neural signal processing with simple, automated methods.

2.4.4 Extendability

Extendability can be examined by considering the requirements placed on a developer when creating a new module. This goal is highly related to the ease of use criteria, as many of the same design considerations impact extendability. The modular design, along with the availability of common paradigms, allows extensive code reuse. If a developer wants to test, for example, a new signal filtering algorithm, then it would be possible to reuse acquisition, application, and other signal processing modules as part of the experimental setup. The developer then only needs to work on the new module.

Module layout minimizes the effort necessary in the creation of new modules. As can be seen in Fig. 2.3.2, all portions of an engine relating to system framework (indicated by the white boxes) are provided in templates. The GUI design is similar in that regard. The only framework-related work that must be done when creating a new module is implementing classes for saving and system communication. These classes inherit from the base classes that provide common functionality. The developer is then free to focus on coding the algorithms of interest.

The task of developing modules is further simplified by the high-level programming environment provided by LabVIEW. Some programming knowledge and familiarity with LabVIEW are still required, but the learning curve is fairly low compared to a language such as C++. The numerous visualization and signal processing tools available also help to ensure that no additional coding beyond the module-specific task needs to be done. Additionally, DLL files compiled from other

languages and even nodes containing MATLAB script can easily be used within the framework. Developers then have these options if they are more familiar with other languages or the task requires another language. The gUSBamp module, for example, communicates with the amplifiers through custom DLLs that were written and compiled in C++. It should be noted that a small amount of overhead is involved in calling MATLAB scripts so these can not be used for tasks that require optimal execution speed.

Craniux is designed to allow for fully custom algorithms and paradigms to be implemented as new modules, while minimizing the time and programming skill it takes to do so. The design was successful in allowing a developer to focus on the task of interest. Furthermore, the effort required to implement those tasks was reduced. Few tasks that involve developing neural signal processing methods are trivial, but Craniux provides a framework to make programming these methods easier and more accessible.

2.5 Conclusions

It has been shown that Craniux is an effective software package for neural signal processing. It maintains data integrity while reliably streaming data and experimental parameters to disk. System execution is efficient, benefitting from the software's design and from LabVIEW's multi-threading capabilities. The software also has a high ease of use; it allows common experimental paradigms to be efficiently tested, customized, and visualized in real-time. Finally, due to the software's design, the provided templates, and the high-level graphical programming, it can be easily extended to test novel algorithms and experimental paradigms. These traits make Craniux an excellent tool for researchers, programmers, and non-expert end users.

The creation of tools such as Craniux is an aim of the work done for this dissertation. Indeed, the goals for the software that were laid out in Section 2.1 reflect the overall goal of providing accessible, yet powerful methods for neural signal processing. In order for the full potential of Craniux to be realized, though, robust algorithms must be developed and implemented that fill

common needs. The following chapters explore such algorithms, with a focus on filtering and artifact removal methods. Craniux was used as the main software platform for the research, development, and implementation of the novel methods that are presented.

Chapter 3

Line Noise

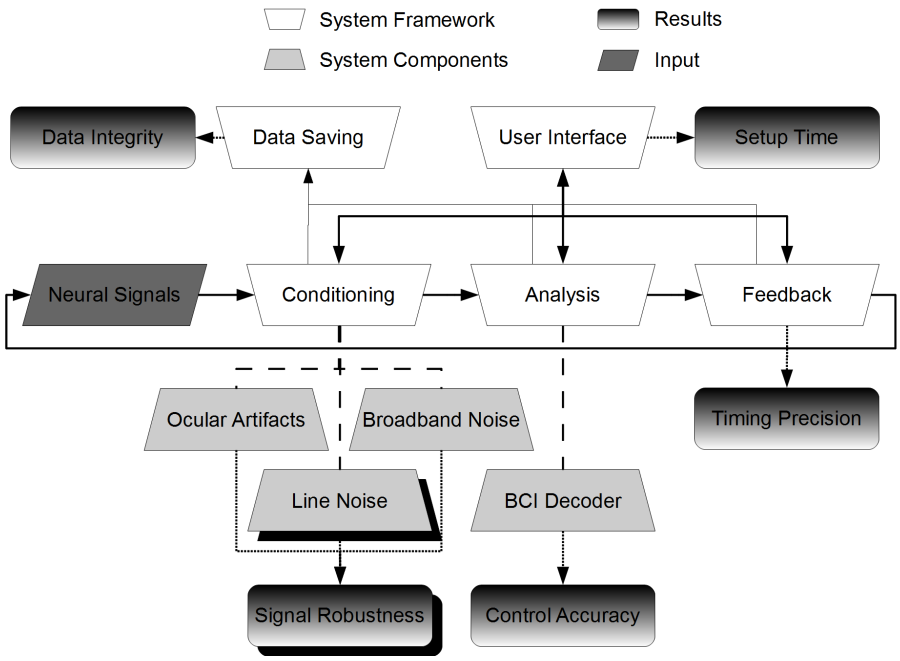

Figure 3.1: Overview diagram highlighting line noise removal. *The boxes with shadows indicate the portion of the system that is discussed in this chapter.*

3.1 Introduction

The software presented in the previous chapter is an excellent framework, but for a system to truly be a valuable tool it must include methods to overcome common obstacles faced by users. One such obstacle is the corruption of neural signals from various sources of noise. The next few chapters develop algorithms that minimize the effects of some of the most common sources of noise. In relation to the overall system, these algorithms are included in the conditioning block in Fig. 3.1. This particular chapter focuses on line noise, as indicated in the same diagram.

Sinusoidal contamination can be a problem in many signal recordings. One of the most common sources of sinusoidal noise is the power line frequency at 60 Hz or 50 Hz. Interference from line noise can especially be a problem in physiological recordings where the signal-to-noise ratio (SNR) is typically low [35], [36]. This type of contamination can be a major hindrance in the research or use of neural signals. Elimination of line noise has been an active area of research, but many methods commonly implemented still leave room for improvement in eliminating the interference while minimizing distortion of the underlying signal.

This chapter discusses a method for removing sinusoidal contamination that will be referred to as the adaptive sinusoid canceler (ASC). The ASC is capable of frequency tracking and a variable bandwidth in the absence of a noise reference. This method adds minimal complexity to the standard adaptive noise canceling (ANC) filter configuration and using a generalized setup proved effective in eliminating sinusoidal noise and avoiding signal distortion. In keeping with the goals of this dissertation the filter achieved high performance while remaining accessible to a non-expert user.

Relevant background methods and material are discussed in Section 3.2. The full implementation and details of the ASC are covered in Section 3.3. In order to obtain quantitative results, much of the data used for analysis was simulated. For further verification real neural data was used, but these results can only be visually presented since the true signal and noise are not known. The performance of the ASC, standard notch filters, and an adaptive line enhancer (ALE) are compared in Section 3.4. The conclusion is presented in Section 3.5. It should also be noted that the ASC was first presented and initially validated in [37]. The work in this chapter improves upon the filter, and offers greatly expanded analysis and results.

3.2 Background

3.2.1 Line Noise

For removing line noise in neural signals, it is common practice to use fixed notch filters centered at the average power line frequency and its harmonics [38], [39]. The problem with this approach is that power line frequency varies around its mean, so the notch must be wide enough to account for the variation [40]. Increasing the notch width helps ensure the noise is removed, but also increases the amount of the signal that is removed. Other common approaches include low pass filtering below the power line frequency or doing a spectral analysis of the signal and ignoring frequencies near the contamination. These techniques, while normally effective in eliminating the contamination, could also discard useful data.

The main objective of any noise removal problem is to eliminate the interference while causing minimum distortion of the signal. For a sinusoidal noise component that can drift in frequency this goal is best achieved by implementing a filter that is able to track the drifting frequency [41]. In doing so it can then also maintain a narrower filter bandwidth to decrease the distortion caused to the underlying signal [42], [9]. Adaptive filters and phase-locked loops (PLLs) are two common solutions to accomplish this task, but these typically require an external reference signal and possibly additional hardware as in [43].

3.2.2 Adaptive Noise Canceler

Overview

Adaptive filters have time-varying weights that adjust to minimize the mean squared value of an error signal. In most cases the goal of the filter is to converge to a state in which it imitates an unknown system. The ANC filter is a form of adaptive filter based largely on work done by Widrow [44] and has proven to be an effective means of removing noise that is correlated with a known reference signal [45]. In [46], an ANC infinite impulse response (IIR) notch filter with varying poles and zeros was developed to effectively remove sinusoidal noise, but this filter required manual adjustment of parameters and a reference input that was correlated with the noise.

In some experimental setups, it might be difficult to record an accurate reference for the sinusoidal interference. This difficulty could be due to a number of factors including available hardware, safety precautions and regulations, or the knowledge and experience of the personnel involved. If a reference cannot be recorded then one must be artificially generated and it becomes difficult for the ANC filter to outperform a standard notch filter. In order to maximize the potential of using an ANC filter in this situation it is vital to be able to generate as accurate of a reference as possible.

The method presented in [47] is an excellent example of an ANC filter that has been modified to eliminate the need for an external reference signal for sinusoidal noise. That method produced good results, but its bandwidth remained stationary and it also required an additional notch filter

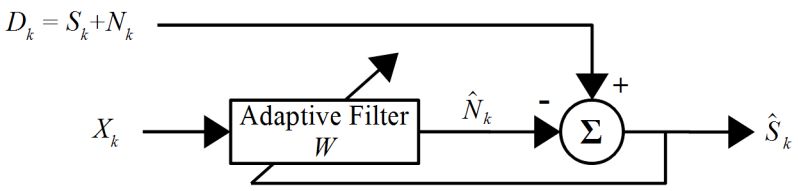

Figure 3.2: System design for an adaptive noise canceling (ANC) filter. D is the recorded signal that consists of the true signal S plus noise N. A reference correlated with N is given by X, and the filter coefficients W are adapted by minimizing the output $\hat{S}$ so that $\hat{N}$ most closely resembles N before being subtracted.

that did not adapt with the detected noise frequency. For effective removal of sinusoidal noise it is necessary to be able to track the noise's frequency, and it is also useful to be able to vary the bandwidth of the filter in order to more quickly and accurately adjust to changes in the noise frequency.

Theory

A typical ANC filter is given in Fig. 3.2. The recorded signal is given by D, and a reference for the noise is given by X. The reference must be temporally correlated with N, although a small amount of time shift is acceptable as long as it does not exceed the length of the finite impulse response (FIR) filter shown in the diagram with coefficients W. The filter minimizes $\hat{S}_k$, which is the difference between D and $\hat{N}$. Since X is correlated with N and not S, this process should also minimize the difference between N and $\hat{N}$ so that $\hat{S}$ converges towards S. For good performance the correlation between N and S should be minimal. The sample index is denoted by k.

One of the most commonly used convergence methods for adaptive filters is the least mean squares (LMS) algorithm [48]. The LMS filter update equation is given in (3.1), where L is the filter length and σ_x^2 is the signal power of X. For real-time applications σ_x^2 must typically be estimated. The bound on μ, the step-size parameter, is required for stability of the filter and was derived in [49]. Many other factors are important, though, in ensuring the filter's stability. In general, smaller values of μ or larger values of L result in longer convergence times and smaller steady state error, and vice versa.

$$W_{k+1} = W_k + 2\mu\hat{S}_k X_k, \qquad 0 < \mu < \frac{1}{L\sigma_x^2} \tag{3.1}$$

Sinusoidal Noise

For the case of sinusoidal noise, X is a sinusoid of the form in (3.2) where f_{samp} is the sampling frequency, and B and ϕ are the amplitude and phase of the sinusoid, respectively. In this special case, the ANC filter actually implements a 2nd order IIR notch filter centered at f_x, the frequency of the reference sinusoid. In [50] it was shown that the transfer function for an ANC filter in the case of a pure sinusoidal reference is given by (3.3), where z is from the Z-transform. This approximates a notch filter centered at f_x with bandwidth approximated by (3.4).

$$x_k = B\cos(2\pi f_x k/f_{samp} + \phi) \tag{3.2}$$

$$W(z) = \frac{z^2 - 2z\cos(2\pi f_x/f_{samp}) + 1}{z^2 - 2\big(1 - \frac{L\mu B^2}{4}\big)z\cos(2\pi f_x/f_{samp}) + \big(1 - \frac{L\mu B^2}{2}\big)} \tag{3.3}$$

$$\text{BW} = \frac{L\mu B^2 f_{samp}}{2\pi}\ \text{Hz} \tag{3.4}$$

The accuracy of this approximation depends on the strength of the time-invariant components of the filter's transfer function relative to the time-varying components. This ratio was also shown in [50] to be given by (3.5). If f_x is known beforehand to not drift by a large amount then it is easy to minimize β by choosing an L to make the numerator of (3.5) close to zero. L should then be the nearest integer to any value that produces an integer when multiplied by the normalized frequency (f_x/f_{samp}). As the range of possible values for f_x increases or as the normalized frequency approaches 0 or 0.5 (values at which the denominator of β is close to zero), L must be increased to help ensure β is small.

$$\beta = \frac{\sin(2\pi L f_x/f_{samp})}{L\sin(2\pi f_x/f_{samp})} \tag{3.5}$$

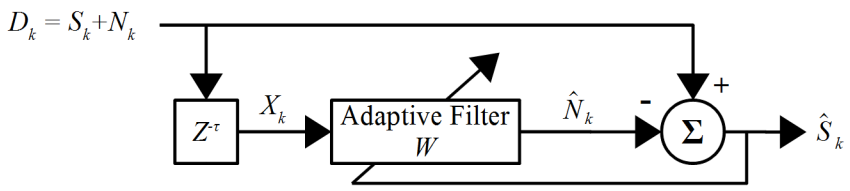

Figure 3.3: System design for an adaptive line enhancer (ALE). This filter operates the same way as the ANC shown in Fig. 3.2, but a delayed version of the recorded signal is used as the reference.

When given a sinusoidal reference and an appropriate value of L, it is then known that the ANC technique with the LMS algorithm converges to a notch filter centered at the reference's frequency with a bandwidth that depends on μ. If the reference is an accurate model for the noise, then the ANC filter is able to easily track drifts in the frequency of the sinusoidal interference. For situations where the frequency of the sinusoidal noise has any amount of uncertainty, such as with power line noise, this method is a far superior solution to a fixed notch filter. The challenge then becomes maintaining this performance in the absence of an accurate reference.

3.2.3 Adaptive Line Enhancer

The ALE is a version of the ANC filter that was designed for cases where a reference is unavailable and either the noise or the signal is known to be periodic [51]. It has been found to be useful for many other tasks, though, and can also be used to eliminate correlated portions of a signal [52]. For the purposes here, it is known that the noise is periodic and elimination of any other correlated signal portions should be avoided. The ALE typically assumes then that the signal is not periodic, although if configured carefully can still be used as long as the signal period is not an integer multiple of the noise period and vice versa. The ALE design is shown in Fig. 3.3. Although prior work has shown convergence algorithms and other parameters to have an effect on its performance [53], it is not a difficult method to implement.

There are a few practical considerations to keep in mind with the ALE. The key to most of these is remembering that the reference is the recorded signal shifted by τ, which the adaptive filter can then shift an additional L in either direction. This means that the range from $\tau - L$ to

$\tau + L$ should contain minimal values in the autocorrelation function of S. It also means that same range should contain an integer multiple of the period of N. These two considerations allow the filter to match $\hat{N}$ to the periodic N while minimizing the effect of S that remains in $\hat{N}$. If L is greater than one half-cycle of N then the second condition is met no matter the value of τ.

The ALE is effective at removing periodic noise, but it can add a significant amount of signal distortion. Even if the filter is able to perfectly remove N, it still contaminates the signal with a delayed and filtered version of S. So based on a number of factors such as the original SNR of the recording, the autocorrelation of S, the spectral shape of S near f_N, and how precisely f_N is known, an ALE may or may not outperform a notch filter for the removal of sinusoidal noise.

3.3 Methods

3.3.1 Adaptive Sinusoid Canceler

It is well known that an ANC filter is effective if given an accurate reference signal, and the difficulty here lies in implementing an ANC scheme that does not rely on an external reference. This can be accomplished by taking advantage of the inherent structure of the ANC filter and making a few small changes to the algorithms presented in Section 3.2.2. The method discussed in this section is diagrammed in Fig. 3.4.

Frequency Tracking

A useful property of the ANC filter can be utilized to track the frequency of sinusoidal noise. In Section 3.2.2 it was seen that in Fig. 3.2, the path from D to $\hat{S}$ when X is sinusoidal is a notch filter centered at f_x with a bandwidth given by (3.7). A corollary to this property is that the path from D to $\hat{N}$ is a bandpass filter matching the notch filter in center frequency and bandwidth [50].

This property means that if the true line noise frequency (f_N) drifts then a sinusoid correlated to N, although attenuated, is still present in $\hat{N}$. Since $\hat{N}$ is bandpassed the estimated frequency, $f_{\hat{N}}$, can be calculated with a method as simple as measuring time between zero crossings (ZC). The

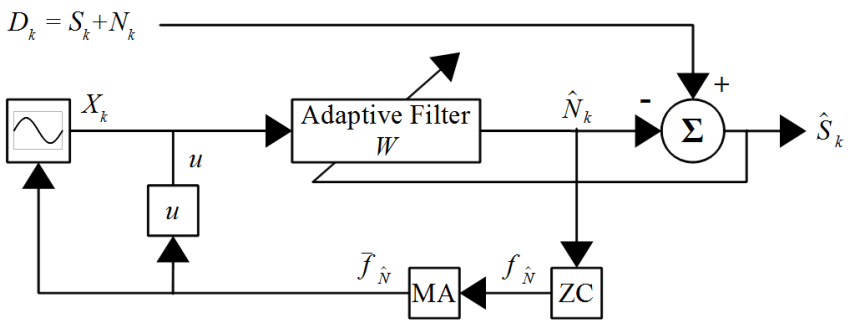

Figure 3.4: System design for the adaptive sinusoid canceler (ASC). The same basic structure as the ANC in Fig. 3.2 is present, but here the reference is a digitally generated sinusoid with a frequency of $\bar{f}_{\hat{N}}$. This frequency is calculated by taking a moving average of the distances between zero crossings in $\hat{N}$. The learning rate of the filter is also a function of $\bar{f}_{\hat{N}}$, as shown by (3.9).

time of a zero crossing can be estimated using interpolation. Due to noise introduced by broadband signal components, a moving average (MA) of length T is applied to $f_{\hat{N}}$ to produce $\bar{f}_{\hat{N}}$. Higher values of T produce smoother, less responsive estimates. Since the goal is to demonstrate a generic filter without fine-tuning parameters, $T = 120$ was used for all analysis. New samples of $f_{\hat{N}}$ occur at each zero crossing of $\hat{N}$. So for $f_N = 60$ Hz this window corresponds to one second of data.

A reasonable estimate of f_N is used to initialize the ASC. If no estimate for f_N is known, then the filter might need to be initialized by using a method such as spectral peak detection to determine the initial reference frequency. Once the filter is initialized it begins calculating $\bar{f}_{\hat{N}}$, which tracks and converges towards the true f_N. The speed and accuracy of this convergence depend on the bandwidth of the filter, but this value is adapted through an automated method as well.

Variable Bandwidth

To simplify the bandwidth calculation, the normalized u given in (3.6) is used in place of μ [48]. Any further reference to the learning rate is referring to u. This substitution is sometimes referred to as normalized LMS (NLMS). In addition to making the learning rate bounds easier to remember, u allows the filter bandwidth given in (3.4) to be represented by (3.7), since for a sinusoid the signal power (σ_x^2) is known to be $B^2/2$. The dependence on L and B has been removed.

$$\mu = \frac{u}{L\sigma_x^2}, \quad 0 < u < 1 \tag{3.6}$$

$$\text{BW} = \frac{u f_{samp}}{\pi} \text{ Hz} \tag{3.7}$$

The bandwidth of the notch filter from D to $\hat{S}$, and also the bandwidth of the corresponding bandpass filter, is then controlled by u. This value can be automatically adjusted based on the behavior of $\bar{f}_{\hat{N}}$. If the estimate is not consistent, demonstrating that the ASC has a poor confidence in its estimated reference frequency, then the bandwidth should be increased. This increase helps maintain the elimination of the sinusoidal noise in $\hat{S}$, and at the same time helps to decrease the attenuation of the line noise in $\hat{N}$, causing the measurement of the zero crossings to be more accurate. As $\bar{f}_{\hat{N}}$ approaches f_N and becomes more consistent, u decreases and the filter narrows around $\bar{f}_{\hat{N}}$. This decrease reduces the amount of the broadband signal eliminated in $\hat{S}$ and passed through to $\hat{N}$, both improving the output and increasing the accuracy of $\bar{f}_{\hat{N}}$ (which in turn allows u to decrease further). The process repeats in an iterative fashion as $\bar{f}_{\hat{N}}$ tracks f_N. Bounds were placed on u so that the filter's bandwidth remained between 0.2 and 4 Hz.

The relation between the changes in $\bar{f}_{\hat{N}}$ and the filter bandwidth is shown in (3.8), where the function $G_T(\bar{f}_{\hat{N}})$ calculates the difference between the maximum and minimum of $\bar{f}_{\hat{N}}$ during the window T. Various other measures were experimentally examined, and G_T proved to be a good tradeoff between stability and responsiveness. Measures such as the derivative that use the rate of change of $\bar{f}_{\hat{N}}$ were not as consistent. Measures such as the standard deviation of $\bar{f}_{\hat{N}}$, or even a linear fit to all the points in the window, were fairly consistent but did not respond as quickly to changes in f_N. The results of (3.8) were not highly susceptible to small fluctuations in $\bar{f}_{\hat{N}}$, but still quickly responded when $\bar{f}_{\hat{N}}$ began moving towards a new value of f_N.

$$\text{BW} = c * G_T(\bar{f}_{\hat{N}}) \text{ Hz} \tag{3.8}$$

The value of c is a free parameter that determines how much the filter's bandwidth responds to

changes in $\bar{f}_{\hat{N}}$, and is referred to as the bandwidth sensitivity. The effect of bandwidth sensitivity does depend somewhat on T, but this relationship is not as strong as would be expected. Higher values of T calculate G_T over a greater distance, but they also add more smoothing to $\bar{f}_{\hat{N}}$. These two behaviors to a large extent cancel each other out, and a single value of c was experimentally found to be effective for various values of T, f_{samp}, and f_N. Again, though, the goal here is to produce a generalized filter rather than demonstrating the effect of fine-tuning parameters, so $c = 20$ was used for all results and analysis of the ASC filter.

Finally, combining (3.7) and (3.8) gives the ASC's learning rate u shown in (3.9). The end result is a notch filter that tracks f_N while minimizing its bandwidth through a process that first increases bandwidth to locate a new f_N, then narrows its bandwidth around the new value.

$$u = \frac{c\pi G_T(\bar{f}_{\hat{N}})}{f_{samp}} \tag{3.9}$$

3.3.2 Data Collection

Simulated Data

An electrocorticography (ECoG) signal simulator built in to Craniux, the software framework discussed in Chapter 2, was used to generate 8 channels of signals [14]. These signals had a sampling frequency of 1200 Hz and contained pink noise with a $1/f$ power falloff to simulate ECoG baseline signals [54]. Simulated power line noise N was added to S using (3.10), where m is the harmonic number and f_N is the fundamental line noise frequency. Two harmonics were added ($M = 2$) in addition to the fundamental frequency and each was given half the amplitude of the previous one.

$$N = \sum_{m=0}^{M} \frac{A}{2^m} \cos(2\pi m f_N + \phi) \tag{3.10}$$

A was calculated to create a specified SNR between S and N. Since the harmonics are non-interfering sinusoids their total power is equal to the sum of their individual powers as shown in (3.11). For a specific SNR, A is then calculated with (3.12). P_s is the average power of S.

$$P_N = \sum_{m=0}^{M} \frac{(A/2^m)^2}{2} = \frac{2A^2}{3}(1-(1/4)^{M+1}) \tag{3.11}$$

$$A = \sqrt{\frac{3P_s}{2 * 10^{SNR_{db}/10}(1-(1/4)^{M+1})}} \tag{3.12}$$

At times the line noise fundamental frequency, f_N, was controlled deterministically. For further analysis, it was also sometimes varied according to the Gauss-Markov process in (3.13) as was done in [9]. η_{Δ_k} is a random sample from a zero-mean Gaussian distribution with variance σ_η^2.

$$f_{N,\Delta_{k+1}} = f_{N,\Delta_k} + \eta_{\Delta_k} \tag{3.13}$$

Real Data

A small amount of ECoG data was also used to qualitatively show the removal of line noise from real data. This data also had a sampling frequency of 1200 Hz and was collected from a human subject who was subdurally implanted with a 32 channel ECoG grid over primary motor and sensorimotor areas. All data collection and procedures were approved by the University of Pittsburgh's Institutional Review Board and informed consent was obtained prior to implantation.

3.3.3 Experimental Parameters

Data was collected in the method described in Section 3.3.2 and the ASC was validated through comparison to more traditional removal methods for sinusoidal noise. The error after filtering, or the remaining noise, was calculated as the difference between the true signal and the filtered signal. In this way all mean squared error (MSE) and SNR calculations took into account both the removal of the sinusoidal noise and any distortion that occurred to the signal. Unless otherwise noted, the

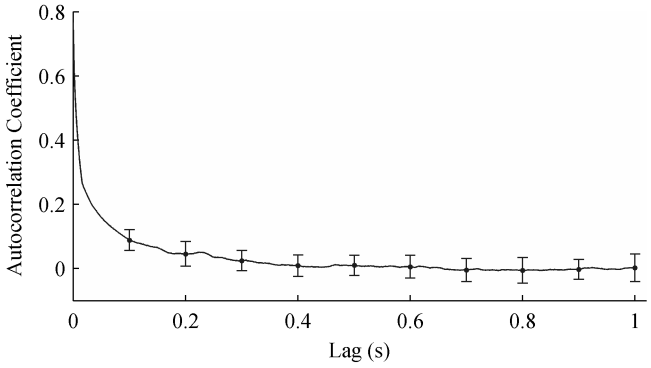

Figure 3.5: Average autocorrelation of 10 second windows of S. The error bars represent the standard deviation across all windows.

SNR before filtering was always 0 dB. All results presented for the simulated data took the average across the 8 channels of data and all filtering methods operated on the same sets of data. The ASC tracked the noise's fundamental frequency and for the ASC, as well as all notch filter methods, corresponding filters were added for the harmonics.

As stated previously, the ASC used $c = 20$ and $T = 120$ for all conditions. For both the ALE and the ASC, $L = 20$ was used. According to (3.5), this length should allow the ASC to perform optimally by minimizing β. A length of 20 is also longer than one half-cycle of N, which means the only consideration for the ALE was to select τ to minimize the autocorrelation of S, which is shown in Fig. 3.5. From this figure, $\tau = 0.5$ seconds was chosen as a sufficient delay.

3.4 Results and Discussion

3.4.1 Simulated Data

Internal State of the ASC

The overall performance of the ASC depends on the accuracy of its internal adjustments, so the behavior of its frequency tracking and variable bandwidth was examined first. To do so, the frequency of the additive sinusoidal noise was adjusted at 30 second intervals by increasingly larger amounts. Fig. 3.6 shows the internal behavior of the ASC as these changes occurred. As can be

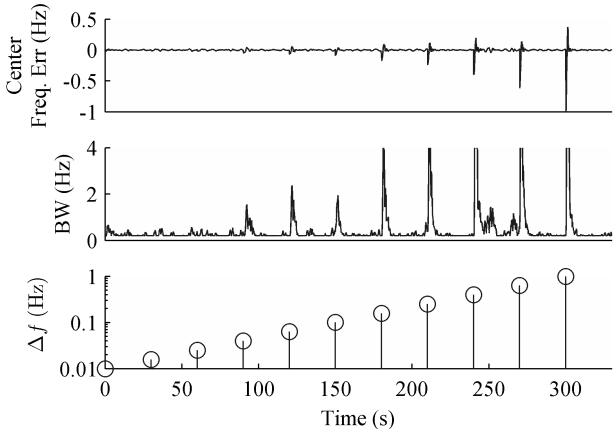

Figure 3.6: ASC frequency tracking and bandwidth as changes in noise frequency occur. Shown is (Top) the error in the ASC's frequency estimate, (Middle) the ASC's bandwidth as the frequency changes occur, and (Bottom) the log scaled magnitude of changes in the noise frequency.

seen, the filter was able to effectively track changes in the noise frequency while adjusting its filter bandwidth accordingly. As a reminder, the ASC's bandwidth was bounded by 0.2 and 4 Hz.

The spikes in the frequency error correspond to the convergence period of the filter, and during these periods the bandwidth of the filter increased to help keep the noise attenuated and to allow the filter to more quickly find the new frequency. As the changes in frequency get progressively larger so do the size and then duration of the increases in bandwidth. This process is susceptible to noise, as evidenced partly by the small unexpected increases in bandwidth seen between the two frequency changes near 250 seconds.

ASC Frequency Tracking Performance

The frequency tracking of the ASC needed a baseline to compare against, so a spectral peak detection method was implemented and analyzed. This method simply took a fast Fourier transform (FFT) of the signal and looked for the frequency between 55 and 65 Hz where a peak occurred. To examine the effectiveness of this method against the ASC's frequency tracking the stochastic model for noise frequency given in (3.13) was employed. The model was set up with $\Delta_k = 2$ seconds and $\sigma_\eta = 0$, 0.01, and 0.1. For $\sigma_\eta = 0$ the frequency remained at 60 Hz.

It is well known that the accuracy of an FFT's frequency estimate increases with the length

Table 3.1: MSE between the actual and estimated ASC frequency

	$\sigma_n = 0$	$\sigma_n = 0.01$	$\sigma_n = 0.1$
ASC	$5.0 * 10^{-5}$	$9.8 * 10^{-3}$	$2.9 * 10^{-1}$
FFT	$3.4 * 10^{-3}$	$8.3 * 10^{-3}$	$2.9 * 10^{-1}$

of its window, which corresponds to the resulting number of frequency bins. When the goal is to track frequency changes, though, longer windows could result in missing quick changes. With this in mind, the spectral peak detection method was tested using window sizes from 1 second up to 10 seconds in 1 second increments. An increase in performance was seen from 1 second to 2 seconds, but after that the average results did not vary significantly.

The results for a 2 second window are given in Table 3.1. Both methods perform similarly, with the ASC coming out ahead at $\sigma_\eta = 0$ and spectral peak detection slightly better at $\sigma_\eta = 0.01$. It is expected that there are times in which spectral peak detection or other frequency estimation methods could produce better results than the one employed here by the ASC. A main benefit of the ASC in its current form is that it does not add much complexity or computational cost to the standard ANC configuration.

ASC Variable Bandwidth Performance

Next, the effect of the ASC's variable bandwidth on performance was analyzed. Since the variable bandwidth was bounded by 0.2 and 4 Hz, for comparison the filter was set up to first have a fixed bandwidth of 0.2 Hz, and then of 4 Hz. The resulting performance is shown in Fig. 3.7. This figure was created using the same data as Fig. 3.6, so these two figures can be examined together to better see the relationship between speed of convergence and the ASC's variable bandwidth.

With a fixed 0.2 Hz bandwidth the filter could converge to a low MSE, but convergence time significantly increased as the magnitude of the changes in frequency increased. With a fixed 4 Hz bandwidth, the filter was very consistent and converged quickly even at the larger frequency changes, but it was not able to produce as low of an MSE. With the variable bandwidth the filter still had more variance in its MSE than the fixed 4 Hz bandwidth, but it was able to converge

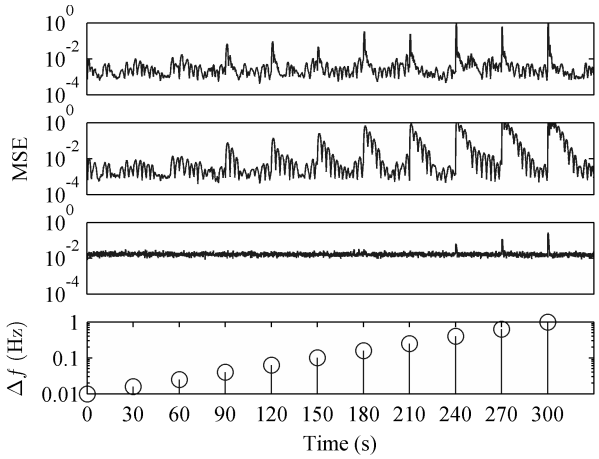

Figure 3.7: Performance of the ASC with and without variable bandwidth. The top three graphs show the MSE between the true signal and filtered signal. The MSE was smoothed with a 100 ms long moving average filter to make the plots more visibly clear. Shown is (Top) variable bandwidth between 0.2 and 4 Hz, (Top Middle) bandwidth set at 0.2 Hz, (Bottom Middle) bandwidth set at 4 Hz, and (Bottom) the log scaled magnitude of changes in the line noise frequency.

quickly and produce a lower average MSE. The SNRs over the whole trial for the variable, 0.2 Hz, and 4 Hz bandwidth were 20.2 dB, 11.5 dB, and 17.0 dB, respectively. The results of the fixed 0.2 Hz and 4 Hz bandwidths do not necessarily indicate the performance that would be seen from bandwidths in between these values, but the boundary conditions were used to illustrate that the variable bandwidth method is able to retain some of the advantages of both the extreme cases.

Deterministic Noise Frequency

For filters with a fixed center frequency, their effectiveness depends only on the distance of the noise frequency from that center. Using the stochastic model given by (3.13) with a limited number of samples could give inconsistent results for these methods based on how far the sample mean drifts from the true mean. So for comparison to a fixed 4 Hz notch filter the frequency of the additive noise was increased from 60 Hz by 0.1 Hz every 2 minutes to measure the resulting SNR at specific frequencies. The SNR after adding the noise was 0 dB.

As shown by Fig. 3.8, the performance of the standard notch filter degraded as the frequency increased to 61 Hz even though its bandwidth was 4 Hz. The ASC and ALE were able to maintain a steady SNR. The ASC produced the highest SNR even at 60 Hz, the ideal condition for the

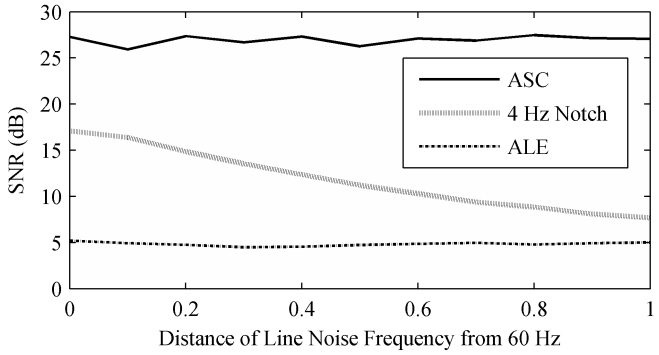

Figure 3.8: Performance of filters on a deterministically drifting sinusoid. The x-axis shows the distance of the noise frequency from 60 Hz and the y-axis shows the filtered SNR.

fixed notch filter. Note that the SNR here also takes into account the time period during which the ASC is adjusting to the new frequency, and the small variance in the SNR is due to this convergent process and finite data sample lengths.

The performance of ALE on this data was well below that of the ASC. This was most likely due to the delayed portions of S used by the filter impacting the output. Even if the autocorrelation is zero at the chosen lag, the delayed signal is still part of $\hat{N}$ and is subtracted out. This essentially adds random noise to the signal that has the same shape as S.

Stochastic Noise Frequency

To measure the ASC's performance on drifting sinusoidal noise, the stochastic model for noise frequency given in (3.13) was once again employed with $\Delta_k = 2$ seconds and $\sigma_\eta = 0$, 0.01, and 0.1. The case of $\sigma_\eta = 0$ was included so that a baseline comparison to a standard notch filter could be made, and also since an adaptive filter such as the ASC might be used in the case where it is unknown if a sinusoidal noise frequency will actually drift or not. For each value of σ_η, 5 minutes of data were generated and analyzed. The outcomes of these experiments are given in Table 3.2.

These results are consistent with both Fig. 3.7 and Fig. 3.8. The first column of Table 3.2 again indicates that the ASC outperformed a traditional 4 Hz notch filter even under ideal circumstances for the notch filter. This is because the frequency estimate is able to converge to 60 Hz and then use its confidence in the noise frequency to narrow its bandwidth and minimize distortion of the

Table 3.2: Filtered SNR (dB) for different standard deviations in the noise frequency model.

	$\sigma_n = 0$	$\sigma_n = 0.01$	$\sigma_n = 0.1$
Var. f_c, BW (ASC)	25.3	22.8	17.2
Var. f_c, BW = 0.2	25.1	22.0	6.9
Var. f_c, BW = 4	17.2	17.2	17.0
4 Hz Notch	17.2	-	-
ALE	4.6	4.5	4.3

signal. The ALE once again maintains consistent performance, but not at the level of the ASC.

Of the frequency tracking methods, the variable bandwidth produced the highest SNR across all 3 tested conditions. It is interesting to note that, although by an insignificant amount, the variable bandwidth ASC outperformed the variable frequency, 0.2 Hz bandwidth filter even at $\sigma_\eta = 0$. Since the frequency estimate is subject to noise, it is possible that the variable bandwidth is an advantage even for a fixed unknown noise frequency.

At $\sigma_\eta = 0.1$, the 0.2 Hz bandwidth filter's performance dropped significantly while the variable bandwidth ASC was able to keep performance at the level of the 4 Hz bandwidth filter. The 4 Hz bandwidth filter performed well in all tested conditions, but was unable to take advantage of the lower variances to converge more tightly around the line noise. The variable bandwidth method was able to increase performance at the lower variances and still maintain a good SNR at the highest variance.

A careful reader might note that results similar to those in Table 3.2 and Fig. 3.8 were presented in [37], but that the numbers are slightly different. For some of the results this is simply due to modifications made to the ASC. It is also because all methods were tested on new data sets that were generated with an initial SNR that included all noise harmonics, while in [37] the initial SNR was only measured between the signal and the noise at the fundamental frequency.

Variable Initial SNR

With SNR in mind, it can be observed that the accuracy of the ASC's frequency tracking and bandwidth adjustments might depend heavily on the initial SNR of the recording. To help determine

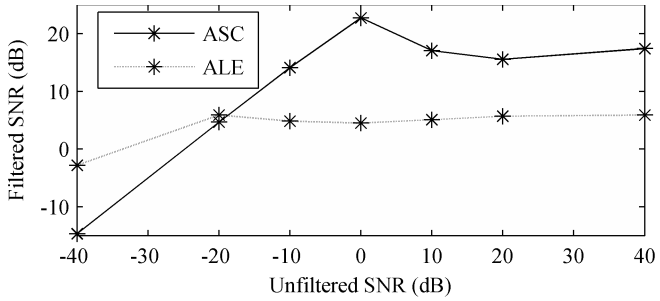

Figure 3.9: Performance of the ASC and the ALE as the initial SNR changes.

this effect, 5 minute segments of data were generated with SNRs ranging from -40 to 40 dB. The stochastic noise frequency model was used with $\sigma_\eta = 0.01$. The results for both the ASC and the ALE for this data are shown in Fig. 3.9.

The ASC was able to improve the signal quality for any initial SNR below 10 dB, with its best performance coming at 0 dB. Once the initial SNR got to 20 dB the ASC was unable to locate the noise due to the broadband noise components in $\hat{N}$ being significant enough to make the frequency estimate unreliable. As with a standard notch filter, though, even if the frequency estimate were accurate it is doubtful that the SNR could be improved at that point. The distortion caused by the filter would most likely outweigh the noise removal.

For lower initial SNRs, the frequency estimate accuracy improved but the resulting SNR from the ASC still dropped sharply. The penalty for the accuracy being off even a small amount was much more severe in this case because the high power noise was then not fully attenuated by the filter. The variance in the noise frequency model was enough to cause momentary inaccuracies in the frequency estimate that let some of the noise slip through. Refer back to Table 3.1 to see that even with an initial SNR of 0 dB the frequency estimate MSE dropped by almost 2 orders of magnitude from $\sigma_\eta = 0$ to $\sigma_\eta = 0.01$. For example, with a -40 dB initial SNR and $\sigma_\eta = 0$, the ASC was still able to converge to output an SNR of about 16 dB.

The ALE's performance on the variable SNR data was more consistent than that of the ASC, but still overall lower. Unsurprisingly, the ALE was not able to improve its performance above about 5 dB for the higher initial SNRs, resulting in degraded signal qualities. At lower SNRs, the ALE's performance did not drop off as sharply as the ASC's, with the ALE actually having far

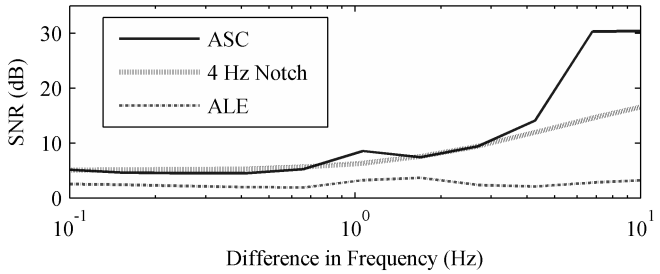

Figure 3.10: SNR resulting from filtering a signal with a sinusoidal component near the sinusoidal noise component. The difference in frequency between the components is given on a log scale.

superior results at an initial SNR of -40 dB. Since the ALE is using the signal itself as a reference rather than generating its own, it always has the precise frequency of the noise in its reference. By the very way in which it's designed, the ALE places a much higher value on removing the noise than it does on preserving the signal. This trait causes the relative strength of the ALE to increase as the initial SNR decreases.

Sinusoidal Signal Components

As a last test with simulated data, the ability of the ASC to discriminate between sinusoidal noise and a neighboring sinusoidal signal component was examined. In this test the assumption was made that the initial frequency estimate is closer to the noise frequency than to the sinusoidal signal component, otherwise the ASC would latch on to the wrong sinusoid. The noise component was set to a frequency of 60 Hz, while the signal component began at a frequency of 70 Hz and every 2 minutes was moved closer to 60. The results of this test are shown in Fig. 3.10. The initial SNR was again set at 0 dB, which means that the sinusoidal noise component did have a slightly higher amplitude than the sinusoidal signal component.

For the ASC, the main transition in performance in Fig. 3.10 occurred between about a 2 and 5 Hz frequency difference. Above 6 Hz the performance leveled out, indicating that the sinusoidal signal component was left intact by the ASC. At the top the performance was even slightly higher than in Fig. 3.8, which was most likely because the sinusoidal signal component resulted in a larger portion of the signal power being outside of and unaffected by the notch filter. Below a 1

Hz difference the performance again leveled out, indicating that the filter could no longer discern between the two components. Until the ASC was able to discriminate between the 2 frequencies and maximize its performance, it behaved similarly to a 4 Hz fixed notch filter.

The results of the ALE are also presented in Fig. 3.10, although this test was not something for which the ALE was designed. The ALE does not target a specific frequency, so it removed both components. Although τ and L can be manipulated to get the ALE to favor one component, this would in general not be a worthwhile practice if that much information is known beforehand about the relevant frequencies.

3.4.2 Real Data

Finally, the effectiveness of the ASC in removing line noise from ECoG signals was qualitatively shown on a small amount of real data. These results are meant to offer initial evidence for the feasibility of the filter on real data. Before any additional processing was performed, the signals were bandpass filtered from 1 to 250 Hz. Fig. 3.11 shows the FFT on 5 minutes of this data before and after being filtered by both the ASC and the ALE. Standard notch filters are not shown here since visibly, there would not be much difference between them and the ASC.

In the original signal clear spikes are visible at 60 Hz and 180 Hz, with only a small one at 120 Hz. The ASC removed all of these spikes, and the visible lower frequency portions of the signal were not noticeably affected. The ALE greatly diminished, but did not remove, all of the spikes. It also affected other portions of the signal.

To get a closer look at the effect of the filtering on the real data's spectrum, the coherence of the data over the same 5 minute interval was calculated. The results are shown in Fig. 3.12. The ASC's coherence was what would be expected for a typical notch filter, with sharp dips to 0 at each noise harmonic, and 1 everywhere else (the small dip below 1 Hz is a result of the signal being bandpass filtered and having no real content below 1 Hz).

The ALE's coherence showed significant effects on portions of the spectrum outside of the noise harmonics. Surrounding 60 Hz and 180 Hz, the coherence could be seen to drop. Addition-

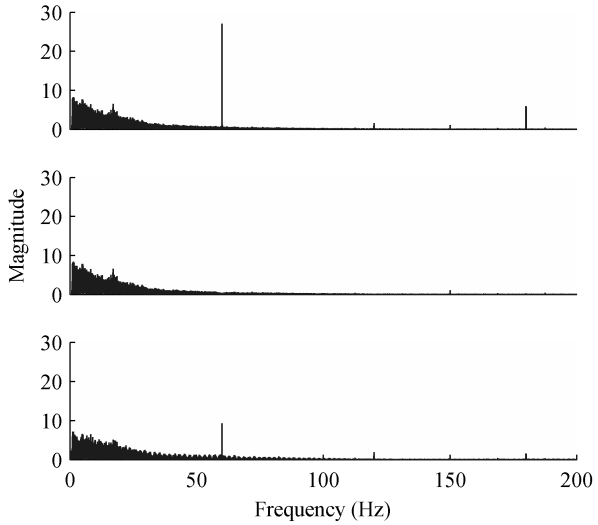

Figure 3.11: Spectral power of real ECoG data with line noise contamination. Shown is (Top) the original signal, (Middle) the signal after filtering with the ASC, and (Bottom) the signal after filtering with the ALE.

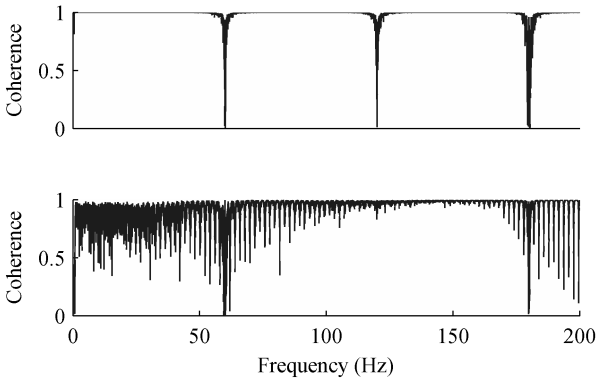

Figure 3.12: Coherence of the ECoG data with the output of (Top) the ASC and (Bottom) the ALE

ally, the lower frequency portion of the coherence resembled the inverse of the original signal's spectrum. This result can be expected since the ALE subtracts out a shifted and filtered version of the original signal.

3.5 Conclusions

This chapter presented a filter, termed the adaptive sinusoid canceler (ASC), designed to effectively remove drifting sinusoidal noise. The ASC operates in a computationally efficient manner with

minimal user input and without the use of a reference signal. The results from the ASC were superior to those from traditional notch filters and an adaptive line enhancer (ALE).

The frequency tracking and variable bandwidth portion of the ASC were shown to work well. The filtering performance was shown to be excellent under varying conditions for the noise frequency. Circumstances that could cause the performance of the ASC to drop were also explored, including the initial SNR being too low or too high or the signal having a sinusoidal component close in frequency to the noise component. The ASC performed within expectations for these situations and comparable to or better than alternative methods. Finally, it was demonstrated that the ASC was effective on real data.

For a specific situation in which parameters such as the SNR, the precise noise frequency, and maybe even a reference signal are known beforehand, a better filter could probably be tailored to meet that need. Indeed the ASC's parameters could even be fine-tuned to better suit some of the situations tested in this chapter. The ASC was shown to still be effective with the same parameters under a variety of conditions, though, and this versatility and ease of use is part of what makes it an attractive option for the removal of sinusoidal noise. The computational cost of the ASC also allows it to be used in most real-time systems.

The results of the ASC in this chapter demonstrate that it outperforms traditional methods for line noise removal from neural signals. It is also easy to use; it requires no expert knowledge to configure it in such a way as to achieve the performance shown here. The ASC is another valuable method for effectively processing neural signals and allowing researchers and end users alike to focus on other tasks.

Line noise is far from the only source of contamination in neural data, though, and for a system to be effective in filtering these signals it must also address other common noise. The following chapters present methods for some of these other types of noise. In the next chapter ocular artifacts (OAs) are examined and in Chapter 5 broadband contamination, usually caused by electromyographic (EMG) activity, is addressed.

Chapter 4

Ocular Artifacts

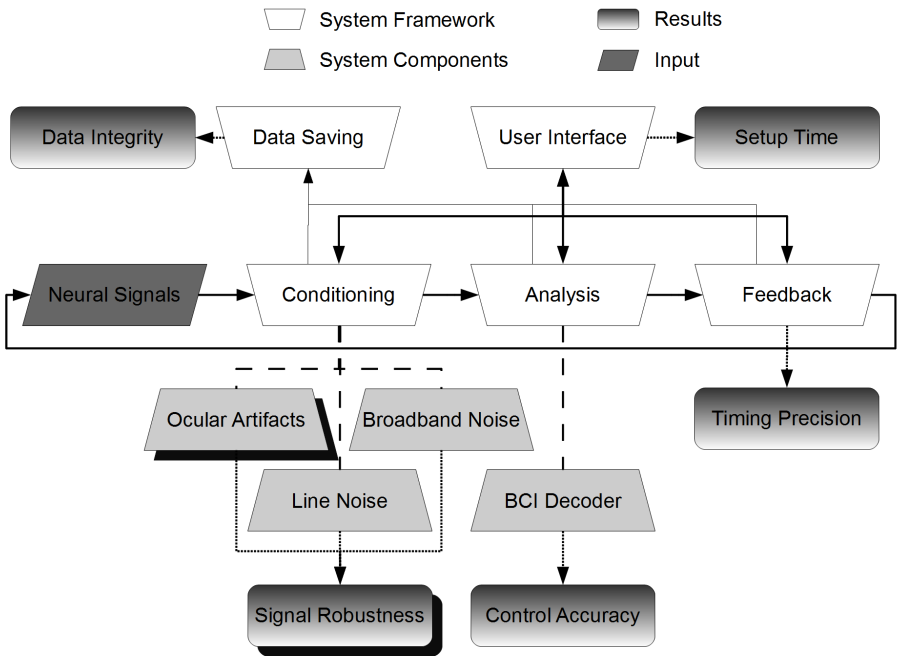

Figure 4.1: Overview diagram highlighting OA removal. *The boxes with shadows indicate the portion of the system that is discussed in this chapter.*

4.1 Introduction

Artifacts produced by eye movement and blinks, commonly referred to as ocular artifacts (OAs) or electrooculographic (EOG) artifacts, are another common source of noise in neural signals. OAs are often dominant over other electrophysiological artifacts and can make neural data unusable. The methods developed for this chapter attempt to remove these artifacts in order to prevent the loss of data contaminated by OAs. Fig. 4.1 indicates this goal, as related to the entire system presented in this dissertation.

In this chapter, OA removal techniques are evaluated on magnetoencephalography (MEG) data. Although OAs do occur in intracranial recordings [55], they are usually a larger problem in recordings from outside the skull such as MEG. This high-dimensional data covers nearly the entire surface of the brain, accounting for the changes in OAs across different recording sites. The feasibility of removal techniques on high-dimensional data is also an important factor to determine. The goal of this chapter is to evaluate novel OA removal algorithms on this data through a quantitative comparison to traditional removal methods. This material was earlier published in [56].

The first novel technique uses a discrete wavelet transform with a Haar basis function to localize artifacts in time and frequency before removing them with thresholding. A method was developed in order to automatically select the level of decomposition for optimal time-frequency isolation of artifacts. This method is based on the smoothness of the artifactual approximation coefficients.

The second novel method separates the signal into independent components and labels some as artifactual by a method termed distribution offset, which measures the difference between the mean and median of each component. The correct number of components is removed based on distribution offset and the power of the reconstructed signal.

A major challenge with OA reduction in neural data is evaluating the results [57] on real data. A method for that purpose was developed here based on correlation, Euclidean distance, and difference in power between the signal before and after OA removal, as well as the number of detected artifacts before and after removal. These metrics attempt to measure the effectiveness of the techniques in removing OAs while preserving neural data.

For comparison to the novel OA removal techniques, a few traditional methods based on regression, principal component analysis (PCA), and independent component analysis (ICA) are also analyzed. The background for methods used in this chapter is presented in Section 4.2 and the implementation of these methods is given in Section 4.3. The results and discussion are in Section 4.4 and the conclusion in Section 4.5.

4.2 Background

4.2.1 Ocular Artifacts & Neural Recordings

OA removal is a difficult task due to the unpredictable changes in the artifacts between channels and the challenges in obtaining a clean reference signal. EOG recordings near the eyes can provide a good model for an artifact, but even these recordings are contaminated by neural data and other external signals. It is also difficult to avoid OAs, since attempting to not blink introduces a cognitive process in the brain that alters the underlying neural data [10], [11].

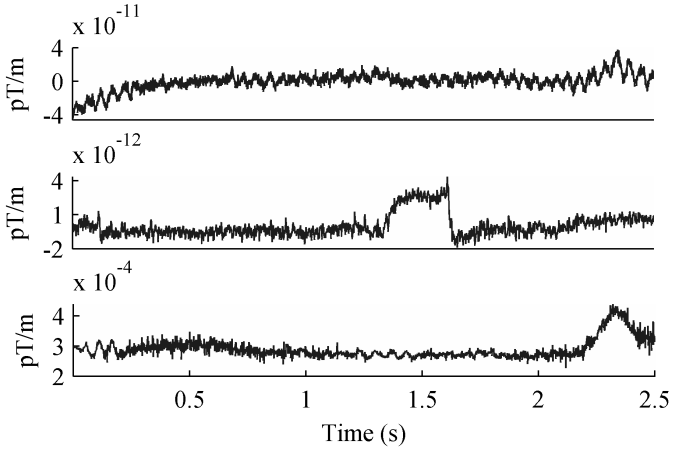

Figure 4.2: Types of OAs. Shown is (Top) eye movement, (Middle) saccadic movement, and (Bottom) an eye blink.

Neural recordings can be modeled as in (4.1), where D is the recorded signal channel m at timepoint k, S is the actual neural activity at the sites of the recordings, N contains each noise source v, and A determines the contribution of each noise source to each channel of neural data. Here, it is assumed that the source of noise is eye movement so A is null for the columns of all other noise sources. Three types of OAs must be accounted for, though: 1) normal eye movement resembles a low frequency drift, 2) blink artifacts produce a sharp spike, and 3) saccadic artifacts produce a near box-shaped waveform (Fig. 4.2). OAs occupy a fairly wide frequency band due to these different types, but they are generally strongest at under 4 Hz [58].

$$D_{m,k} = S_{m,k} + A_{m,v} * N_{v,k} \tag{4.1}$$

It is often possible to obtain a model for an OA through EOG reference channels. OAs occur because the cornea and retina are oppositely charged, causing the eye to be a dipole. Movement of this dipole creates a large change in potential that propagates across the scalp and contaminates neural recordings, especially those with electrodes or sensors located outside the skull. EOG reference channels attempt to measure this contamination at the source with electrodes placed above and below and to either side of the eye (to account for both horizontal and vertical eye movement). Sometimes electrodes are also used to measure radial eye movement.

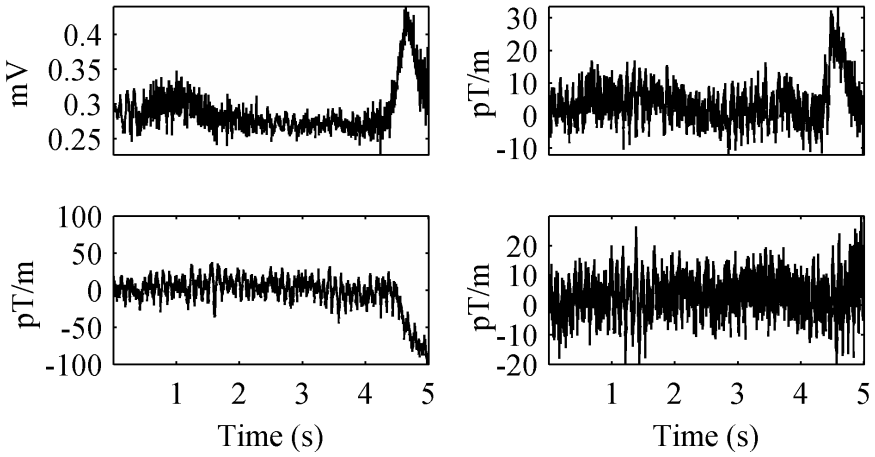

Figure 4.3: Effect of an OA on different data channels. Shown is (Top Left) the EOG reference channel, (Top Right) a neural data channel in which the artifact is nearly duplicated, (Bottom Left) a neural data channel in which the OA's effect has opposite polarity and appears slightly delayed, and (Bottom Right) a neural data channel in which no obvious effect is visible.

4.2.2 Ocular Artifact Removal Techniques

Regression

Since it is possible to obtain a model for the noise through EOG recordings, (4.1) leads naturally to the use of regression to remove the artifacts from the recorded signal. N is approximated by the EOG channels, and then it is only necessary to calculate A to solve for S in (4.1). The problem with the regression method, though, is that it assumes the EOG recordings are clean models for the noise. In reality the contamination is bi-directional and a small amount of neural data propagates to the sites of the EOG recordings. In subtracting out the EOG signals some neural information is then lost. Also, neural data that propagates to the reference channels is introduced into other recording sites [59].

With correlation between the clean neural signal and the EOG recording, it is actually impossible to solve for an exact value of A in the regression model. Even methods that utilize a topographic map of electrical propagation from the eyes across the scalp fail due to the inherent variance in such a map caused by skin and environmental conditions. An OA can also affect neural channels in different ways, as shown by Fig. 4.3.

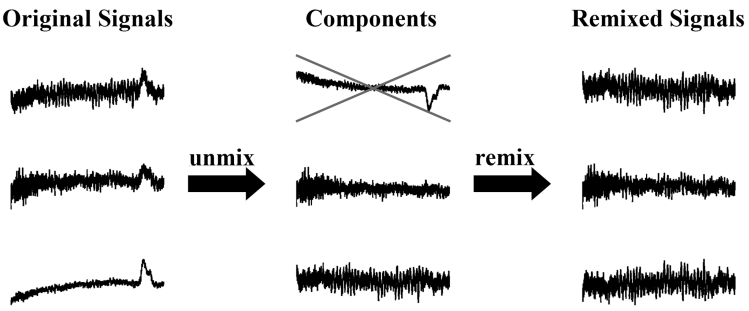

Figure 4.4: Illustration of component-based artifact removal process. Here, the eye blink artifact seen near the end of the signals is isolated to the top component. The columns are then nulled that correspond to this component in the mixing matrix, so that the component is removed from the original signals.

Component-based Removal

Another approach for OA reduction uses component-based methods such as PCA and ICA. The goal in these methods is to 1) transform the original signal into a component space, 2) identify components that correspond to artifacts, and then 3) transform back to the original data space using only non-artifactual components. This process is shown in Fig. 4.4.

PCA transforms a dataset into uncorrelated components and sorts them in order of the amount of variance each component contributes to the dataset. In the context of noise removal, PCA can be useful in multi-channel datasets in which the same source of noise is contaminating all channels, especially if the noise has a higher amplitude than the signal. This has been shown to be effective in OA removal [60]. For this process, the somewhat arbitrary assumption must be made that the signal sources are spatially orthogonal. Also, while PCA does decorrelate the signals, it does not guarantee independence.

ICA is able to go beyond decorrelation and achieve independence between components. ICA is one of the most highly studied and successful techniques for artifact removal in neural signals [61] - [64]. Neural recordings (other than single-neuron recordings) consist of a mixture of signal sources at each recording site, so the problem of removing OAs can be modeled as a blind source separation (BSS) problem. ICA separates the sources by maximizing independence based on one of a number of possible metrics. A few assumptions are made when using standard ICA, the most

important here being that at most one of the sources is Gaussian and that there is negligible signal propagation delay. ICA is also unable to determine the correct order, scale, or polarity of the sources, making artifact identification difficult.

Wavelet Methods

The final method examined in this chapter uses wavelet decomposition for OA removal. This technique has received much less attention in the research community, but some work has been done and wavelets have been applied in other areas of neural signal processing [65] - [68].

Wavelet decomposition involves recursively passing the signal through a pair of quadrature mirror filters and downsampling, resulting in the coefficients at each level of decomposition having higher frequency resolution and lower temporal resolution than the previous level. The goal in removing artifacts with wavelet decomposition is normally to isolate the artifact so that it can be removed with a thresholding function. This can be done by attempting to match the shape of the wavelet to the transient of interest, which is an OA in this case, or by trying to isolate an artifact based on its time-frequency localization. In either case, the isolated artifact can then be removed with thresholding before reconstructing the signal from the wavelet coefficients. Typically, this technique does not need a template for the noise channel and it is also fairly easy to automate, but its performance depends largely on the choice of threshold, basis function, and decomposition level.

4.2.3 Evaluation of Artifact Reduction

Quantitatively evaluating artifact removal in real data is a difficult problem in itself. Since the true, artifact-free neural signal is never known it is difficult to quantitatively assess the performance of OA removal techniques. With recorded neural data it is difficult to even estimate the artifact-free signal since neural signals are non-stationary. There is then no available ground truth to use for comparison with the processed signal. In simulated data this truth is known, but simulations in this case do not capture the true nature of OA contamination in neural recordings.

If the ground truth is known, such as in simulated data, there are methods to evaluate the results of OA removal techniques. Some of these measures are the correlation coefficient, the ratio of the standard deviation (STD), and the Euclidean distance between the processed signal and the true, artifact-free signal [69]. The correlation coefficient determines how well the shape of the true signal is retained, the STD ratio determines how much the power is affected, and the Euclidean distance helps measure both shape and amplitude. The closer the correlation coefficient and STD ratio are to 1 and the closer the Euclidean distance is to 0, the better the results of the OA removal. These methods must be adapted for use on real data, though.

Another common evaluation criteria is to look at frequency correlation [65]. In (4.2), $\tilde{x}$ and $\tilde{y}$ are the Fourier coefficients of the two signals, and w_1 and w_2 are bounds of the frequency window. This measure is just a windowed version of coherence. The goal is to show that the processed signal is nearly perfectly correlated to the original signal at all frequencies except the band containing the artifact.

$$c_{x,y} = \frac{0.5 * \sum_{w_1}^{w_2} \tilde{x}^* \tilde{y} + \tilde{x}\tilde{y}^*}{\sqrt{\sum_{w_1}^{w_2} \tilde{x}\tilde{x}^* * \sum_{w_1}^{w_2} \tilde{y}\tilde{y}^*}} \tag{4.2}$$

4.3 Methods

As stated before, high-dimensional neural data allows the OA removal techniques to be tested on a wide range of recording sites. This type of data can also be extremely useful in clinical studies and in both brain-computer interface (BCI) and neuroscience research, so it is important that the computational burden can be handled. Some of the methods presented here are adapted to overcome the unique problems encountered in removing OAs from high-dimensional data. Both novel and traditional OA removal methods based on regression, component analysis, and wavelets were implemented. The performance of these methods was analyzed using automated, quantitative metrics that are also presented here. All methods were fully automated and computation time was measured, as speed and automation are important in high-dimensional data.

4.3.1 Regression-Based Removal

The first removal method examined was regression. As stated in Section 4.2.2, regression attempts to calculate A in (4.1) in order to solve for S. Many algorithms have been used in solving for A, and the one used here is given by (4.3). This equation can be mathematically proven under the false, but necessary assumption that the correlation between the EOG reference channel (N) and the neural signal (S) is 0 [69]. Note that in (4.3) X_m and N_p are zero-mean and the ratio is the estimations at zero lag of the cross-covariance between X_m and N_p to the auto-covariance of N_p.

$$A_{m,p} = \frac{X_m * N_p^T}{N_p * N_p^T} \tag{4.3}$$

Here it was assumed that A was the same for multiple EOG channels (i.e. $A_{m,i} = A_{m,j}$ for all i, j), so in (4.3) $p = 1$ and N_1 equaled the sum of the EOG channels. This was a reasonable assumption since the source, and thus the propagation path, of the EOG channels was the same. Since A was calculated algorithmically, this removal method was fully automated.

4.3.2 Component-Based Removal

PCA

With high-dimensional data, PCA and ICA became difficult since they perform computations on all channels at once. With PCA the computations were still possible, but it was at times difficult to load the necessary data into memory at once.

Like regression, PCA was simple to automate. This was mostly due to the high amplitude of OAs relative to neural signals. Since the artifact was distributed throughout the neural data at various scales, the high amplitude caused the artifact to contribute a large amount of variance to the data. With the absence of other high amplitude artifacts such as interictal spikes in subjects with epilepsy, it was fairly safe to assume then that any artifactual components would contribute the highest amount of variance to the data. Since PCA orders components by variance, the first p principal components should then contain the artifacts, where p is the number of EOG channels.

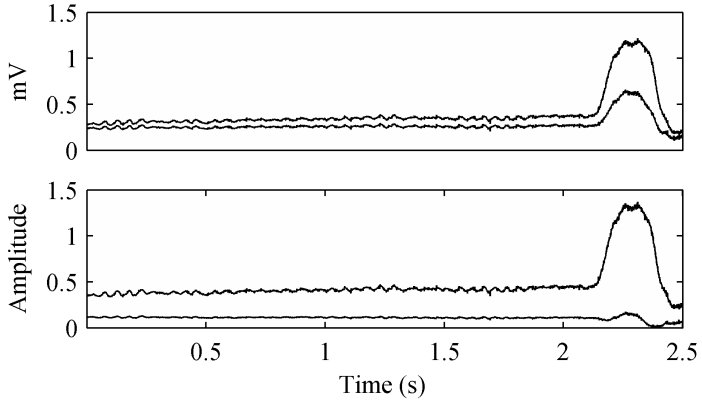

Figure 4.5: EOG reference channels and corresponding principal components. Shown is (Top) the EOG reference channels and (Bottom) the corresponding principal components.

Fig. 4.5 shows this result from running PCA on the full set of neural data and EOG reference channels. In transforming the components back to the original signal space, the first p columns of the transformation matrix were nulled.

ICA

ICA faced the most difficulties with high-dimensional data. For ICA to converge, the number of time points usually needs to be at least several times the square of the number of data channels [61]. For high-dimensional data, the number of time points needed often exceeds trial length.

Two solutions were considered for allowing ICA to converge. First, multiple trials could be concatenated to achieve enough time points. This method assumes that the neural sources and the linear mixture model are stationary across trials. Second, a separate epoch of ICA could be run for each neural data channel paired with the EOG reference channels, as opposed to doing one epoch of ICA containing all the data channels. This is a very time-consuming process, but it also has a few beneficial side effects such as less of a need to worry about the mixtures of distinct neural processes at each electrode: i.e. spatial stationarity of the underlying neural sources, linearity of the mixtures, and the number of sources. Here, the latter of the two methods was used. This is because different trials in the data could contain different stimuli and thus it would be unknown if the neural sources would remain stationary across trials. To somewhat alleviate the problem of

computation time, the FastICA algorithm was used [70].

A further difficulty in using ICA on high-dimensional data was automating the process of identifying artifactual components. Many studies using ICA manually identify artifacts through visual inspection [62], but for a large dataset that would be impractical. Multiple methods for automation have been examined, such as the Hurst exponent [63], kurtosis, Shannon's entropy, and Renyi's entropy [64]. Unfortunately Renyi's entropy proved too computationally costly for high-dimensional data due to the kernel density estimation necessary for each component. The Hurst exponent was used, and the results were compared to a novel method for identifying artifactual components. For the Hurst exponent method, removing components with Hurst values in the eye blink range of 0.58-0.64 removed very few artifacts, so only components with values of 0.70-0.76 that correspond to data from actual neural processes were kept while all others were marked as artifacts. These values are described further in [63].

The novel artifactual component identification method, which will be referred to as the distribution offset, ranks each component by its chances of being an artifactual component. The high amplitude of an artifact causes the mean of the component's distribution to be offset from its median. The mean of a clean neural signal, even one with the presence of a strong event-related potential, is not offset nearly as much. To calculate the distribution offset, the component was centered, and then the difference between the number of samples on the same side of zero and half the number of time points was calculated. This is shown in (4.4), where k is the number of samples and C is the independent component. This measure is similar to skewness, but skewness also factors in the distances of points from the mean.

$$abs\frac{k}{2} - \sum_{i=1}^{k} y_i, \quad y_i = \begin{cases} 1, & C_i - E[C] > 0 \\ 0, & otherwise \end{cases} \tag{4.4}$$

The distribution offset was used to initially mark artifactual components as those where the value in (4.4) was more than 3% of the number of samples. As is the case in Fig. 4.6, the distribution offset normally made a good distinction between artifactual components and neural compo-

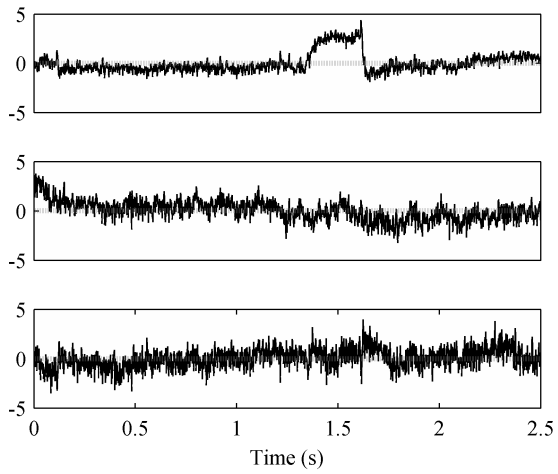

Figure 4.6: Distribution offset values for components of ICA for one channel. Shown is (Top) an artifactual component as detected by its distribution offset of 370.5, (Middle) a component with a distribution offset of 1.5, and (Bottom) a component with a distribution offset of 6.5. The gray dotted line represents the signal mean.

nents, but an additional step was taken to ensure removal of the correct number of components.

After re-mixing, if the signal's power was far above the power of a normal neural signal then the component with the next highest distribution offset was removed. Likewise, if the resulting signal power was far below the power of a normal neural signal the removed component with the lowest distribution offset was added back in to create the final signal. Acceptable power levels for the MEG data that was used corresponded to an STD between 1E-15 and 1E-11. These thresholds could vary with neural recording method. This procedure was also used with the Hurst exponent method, with the distance of the exponent from 0.73 as the ranking criteria.

4.3.3 Wavelet-Based Removal

The wavelet approach used here was a novel method that took advantage of OAs being well localized in time and frequency. The traditional discrete wavelet transform was used with the goal of isolating the artifact in both time and frequency in order to minimize the impact of the artifact removal process on the neural signal. The wavelet coefficients were thresholded to remove the artifact before reconstructing the signal from the thresholded coefficients.

For the wavelet basis function, the Haar wavelet was chosen as it is the simplest wavelet and computation time is important on high-dimensional data. The Haar wavelet provides accurate decomposition and reconstruction with minimum distortions and data redundance [71]. Also, many of its limitations, such as non-differentiability and a chance for detail coefficients to miss sudden changes, were not a concern here as only approximation coefficients are examined before simply thresholding and re-constructing the data. The strategy of selecting a wavelet that matches the shape of the transient of interest was not used since the three types of OAs have different shapes.

To localize the artifact, it is important to zoom in the right amount in time and frequency. This was done by selecting the proper level of wavelet decomposition. Each level of decomposition increases frequency resolution and decreases temporal resolution. Decompose too far and the artifact becomes diluted across frequency bands, making it difficult to remove with thresholding and difficult to isolate in time. Decompose too little and the artifact will not be isolated in frequency, causing neural data to be unnecessarily lost in the thresholding process. This is shown in Fig. 4.7, where the level 3 decomposition failed to isolate the artifact in frequency and the level 6 decomposition began to stretch the temporal bounds of the artifact. The proper level of decomposition depends in large part on whether the artifact is the result of a blink, or of saccadic or regular eye movement, and also on the sampling frequency of the data.

Multi-level Wavelet Decomposition

To choose the proper decomposition level for each artifact a strategy of multi-level wavelet decomposition was used. If an artifact has been fully isolated its wavelet coefficients should be smooth, but cross-contamination with neural signals can make it appear to have higher frequency components (Fig. 4.7). The multi-level process attempts to continue the wavelet decomposition to a depth that is sufficient to remove these higher frequency components.

In this process the signal first underwent a minimum level wavelet decomposition, which was set at level 3. The bounds of any OAs were then marked by finding the first local extrema on the outside of threshold crossings. The threshold was $\pm(5\text{E-}11 + |\text{median}(R)|)$, where R is the

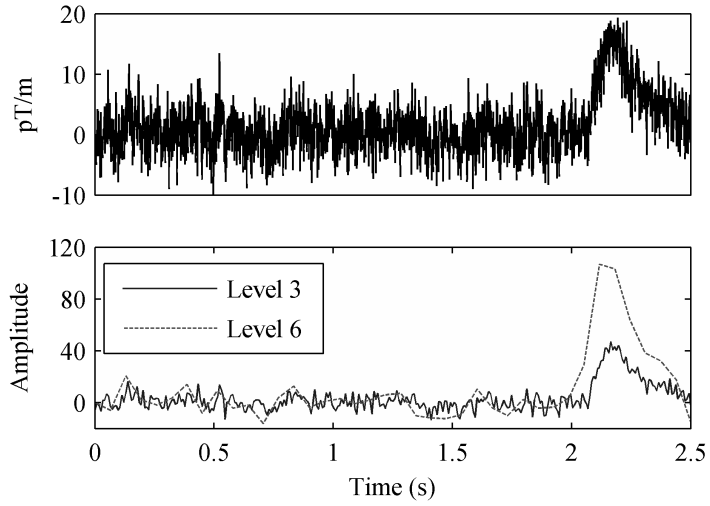

Figure 4.7: Wavelet approximation coefficients for an OA at two decomposition levels. Shown is (Top) the original signals and (Bottom) the wavelet approximation coefficients at two decomposition levels. Level 3 and level 6 were used to accentuate the differences between levels of decomposition.

vector of wavelet coefficients, although the optimal value of this threshold could again vary with recording method. To determine smoothness, it was checked if the derivative of the artifactual coefficients ever changed sign on either side of the peak. If it did, neural data was assumed to still be present and the wavelet decomposition went a level deeper. The artifactual coefficients with an absolute value above the threshold were set to the median of the set of coefficients outside the artifact. This differs from typical wavelet denoising (hard, soft, or soft-like thresholding) in that it is concerned with eliminating the values above the threshold rather than below. The multi-level wavelet decomposition technique is illustrated below in Fig. 4.8. For comparison to this process, wavelet removal was also done with the level held constant at each value from 3 through 9.

4.3.4 Quantitative Analysis

Analysis of OA removal is a difficult process in itself. The seemingly two most obvious methods of evaluation are visual inspection and in the case of a BCI, decoding results. Visual inspection is effective for quick verification or with small datasets, but it is subject to human bias and error and is impractical for use on high-dimensional data. BCI decoding results are not a good criteria because

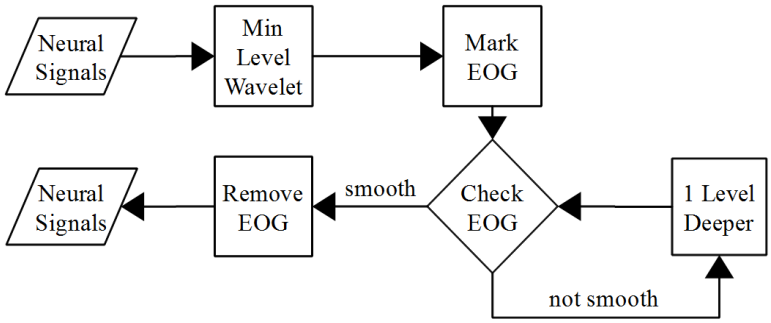

Figure 4.8: Illustration of the process for determining the optimal level of wavelet decomposition. The minimum level decomposition was set at level 3. The 'Check OA' step checks the smoothness of the coefficients marked as artifactual as described in the text.

the decoding algorithm could be invariant to OAs or eye movement could be biased towards a certain class, thereby improving decoding results and giving the impression that the BCI is effective when the system is actually controlled by eye movement.

There are two main criteria that should be used to evaluate an OA removal technique: 1) how well the artifact was removed and 2) how well the neural data was preserved. The methods used here to measure these criteria were in large part taken from previous methods given in Section 4.2.3 [65], [69], but modifications were made to improve performance and to adapt to using real neural data rather than simulated. This is key since the main difficulties in evaluating OA removal arise with real data where the ground truth is not known.

To determine how well the neural data was preserved a number of measures were used. For many of these measures, only those portions of the signals that were originally artifact free were used. These portions should remain the same after OA removal. OAs were marked by low pass filtering at 10 Hz and then detecting threshold crossings. On the uncontaminated portion of the signal, correlation coefficient and Euclidean distance were used as discussed in Section 4.2.3. The STD ratio was replaced due to the possibility that a denominator near 0 for any trial would be a large enough outlier to ruin the overall average. Instead of the STD ratio, the mean squared difference between the STD of the original and processed signals was used. This will be referred to as the exterior STD difference.

To determine how well the artifact was removed, the main metric was the percentage of con-

taminated trials where an OA was no longer detected after the reduction process (using the filtering and thresholding method discussed in the previous paragraph). It should be noted that the absolute percentage of removed artifacts is not as important as the relative percentages between methods since the artifact detection itself is not perfect. Also, the difference between the STD of the entire processed signal and that of the artifact-free portion of the original signal was calculated. This will be referred to as the total STD difference. Using the entire signal should reduce the impact of the signal's non-stationarity and since the power of a portion of the signal can deviate to either side of the mean, taking the mean over a large number of trials should produce a value as close to 0 as possible. Finally, the frequency correlation was examined to determine the effect of the removal process on the signal's spectrum.

4.4 Results and Discussion

The datasets used here contained 306-channel MEG neural recordings, sampled at 1 kHz, of both language processes and motor functions. In the language sets, subjects were observing various words and images. Trials are 5 seconds long, and there were a total of 540 trials. The motor datasets contain 2.5 second trials of both overt (775 trials) and imagined (640 trials) wrist movement. All subjects had normal brain function and data collection was approved by the Institutional Review Boards of the University of Pittsburgh and Carnegie Mellon University. The different datasets were used to test OA removal methods on data of different trial lengths and with different frequencies of each type of OA. EOG recordings were made above and lateral to the eye. It has been shown that having vertical and horizontal EOG channels produces better results than one channel [72].

4.4.1 Overall Evaluation

Table 4.1 shows the results of the quantitative performance metrics discussed in Section 4.3.4 after performing OA removal on the datasets above. The running time of each removal process relative to the fastest method (regression) is also given. To evaluate the effectiveness of the multi-level

Table 4.1: Evaluation of OA Removal Techniques

			ICA		Wavelet		
Metric	Regression	PCA	Dist. Offset	Hurst	Level 5	Level 4	Multi-Level
Corr. Coeff. (E-2)	99 ± 1.9	98 ± 2.4	96 ± 14	72 ± 35	98 ± 5.8	98 ± 5.2	98 ± 5.2
Euclidean Dist. (E-11)	2.6 ± 3.8	3.3 ± 4.3	3.5 ± 8.2	8.0 ± 11	2.4 ± 6.7	2.1 ± 6.1	2.4 ± 6.8
Ext. STD Diff. (E-25)	0.3 ± 2.5	0.5 ± 4.8	20 ± 140	62 ± 190	3.0 ± 30	2.6 ± 28	2.9 ± 29
Total STD Diff. (E-14)	-10 ± 54	-5 ± 53	18 ± 140	110 ± 220	0.7 ± 65	-2.6 ± 61	-0.5 ± 65
Removal Percentage	20	29	74	65	89	91	91
Relative Comp. Time	1.0	2.2	21	20	2.8	2.5	2.7

wavelet technique, its results were compared to the results of using wavelets with optimal constant decomposition levels, which were found to be levels 4 and 5 through the same metrics presented in Table 4.1.

Surprisingly, regression performed well in measures of preservation of the neural data. The poor performance in removing artifacts nulls any value associated with retention of neural data, though. The removal percentage was extremely low, and the negative total STD difference indicates that not enough power was removed from the contaminated portion of the signal. These results indicate that the regression coefficients (A in (4.1)) were too small. Using a regression method in which the regression coefficients are calculated separately for each EOG channel might improve results, but it probably could not increase removal percentage to a satisfactory level while maintaining high preservation of neural data due to the neural contamination in EOG channels.

PCA was also not very effective at removing artifacts. This result is most likely an inherent limitation of using PCA for OA removal in that PCA was unable to fully separate the artifacts from the neural data. PCA only decorrelates the data, and its requirement of spatial orthogonality of the signal components is an additional restriction that might prevent separation of artifacts from the neural components. Methods of detecting additional artifactual components or residuals distributed throughout the remaining principal components could improve results, but any improvement with this method would most likely come at the cost of preservation of neural data and PCA would probably still not match the results from other removal techniques.

ICA showed a large improvement over PCA and regression in removing artifacts. The novel distribution offset method outperformed the Hurst exponent method in identification of artifactual components. Distribution offset had a higher removal percentage, but its main advantage was that

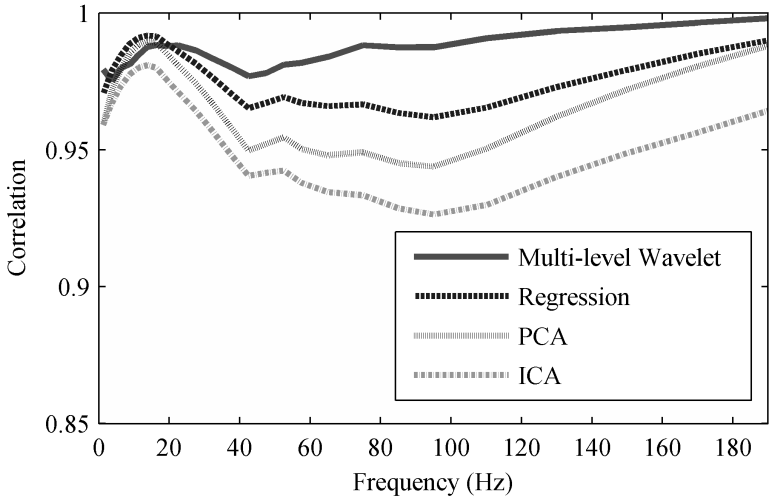

Figure 4.9: Frequency correlation between original and filtered signals. Here, ICA used distribution offset for identification of artifactual components.

it far exceeded the Hurst exponent in preserving neural data. Since the Hurst values eliminated were associated with any non-interesting data rather than just OAs, as in [63], it is possible that the Hurst method removed additional noise, such as line noise or electromyographic (EMG) artifacts, rather than vital neural data, but it is doubtful that the large amount of information removed could be fully accounted for by noise.

Even using distribution offset, ICA was not as effective as other methods in preserving neural data. Fig. 4.9 reinforces this result. This tradeoff was expected, and the overall results showed ICA to be a much more viable option for OA removal than regression and PCA. Based on the neural preservation metrics and the positive total STD difference, ICA removed too much data, which was most likely a result of neural data not being fully separated from artifactual components. The frequency correlation graph also shows this as ICA (distribution offset method) had the smallest correlation of any removal technique.

A method that computes ICA on the full data matrix might be able to better separate the data, but on this dataset such a method would require concatenation of trials to obtain enough data points, which means the assumption must be made of spatial stationarity of the neural sources across trials. This would be a difficult assumption to make given that the trials here contained different stimuli, although it should be noted that some studies have found it satisfactory to only compute the ICA

Table 4.2: Removal percentages for different datasets

Dataset	Regression	PCA	ICA (Dist. Offset)	Wavelet (Multi-Level)
Language	56	62	86	96
Overt Wrist	12	23	73	90
Imagined Wrist	14	21	73	90

unmixing matrix once with as little as 10 s of data [59]. That technique also helps alleviate ICA's other downside, which is its computation time.

The most effective method examined was the wavelet method. Not only did the wavelet technique produce the best results in removing OAs, it was far superior to ICA in retaining neural data. It was also much more consistent than ICA in all measures as indicated by the standard deviations in Table 4.1. Additionally, the wavelet method produced results that had the highest frequency correlation with the original signal above 20 Hz, as shown by Fig. 4.9.

The multi-level method also had superior performance compared to the fixed-level results shown in Table 4.1. Although multi-level wavelets were not the best in every metric, the overall results were slightly better than any single level. As expected, the multi-level technique removed artifacts as well as level 4 decompositions as indicated by removal percentage, while still leaving behind the proper amount of power in the contaminated portion of the signal as shown by total STD difference. The multi-level technique also saved the time of manually finding the optimal level. If the optimal level for nearly all artifacts is the same, though, then the multi-level method adds unnecessary computation time.

4.4.2 Ocular Artifact Removal by Dataset

Wavelets, along with ICA, were also more robust to the different datasets (Table 4.2). Regression and PCA performed over 40% worse at removal in the motor datasets compared to the language set. There was no significant difference between the imagined and overt motor sets for any removal method, though, which would seem to indicate that the large drop in removal percentage from the language set was due to the change in trial length from 5 to 2.5 seconds.

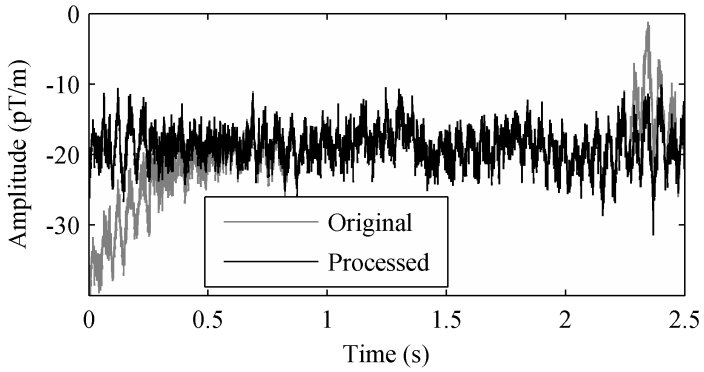

Figure 4.10: Example of OA removal with multi-level wavelets. At the beginning of the signal there is an eye movement artifact, and at the end of the signal an eye blink occurs. Both artifacts were removed while retaining neural data.

4.4.3 Visual Results

In visual inspection of a sample of signals, the multi-level wavelet technique again showed the best results. Fig. 4.10 shows typical results from multi-level wavelets in which it is clear that the non-contaminated portions of the signal were left untouched, and two types of OAs appear to have been removed while still retaining the neural data.

4.5 Conclusions

This chapter presented the development of a novel wavelet technique for removal of ocular artifacts from neural data, a novel method for automatic identification of ocular artifact components in ICA, and a set of quantitative metrics for automatic evaluation of the effectiveness of OA removal on real neural data. As discussed in Section 4.4, multi-level wavelets were most effective in terms of OA removal and preservation of neural data. This conclusion is supported by the quantitative metrics (Table 4.1, Fig. 4.9) as well as by visual inspection of a sample of signals (such as Fig. 4.10). This method also did not require EOG reference channels.

Although ICA encountered complications with the high-dimensional data, it has been highly used in low-dimensional datasets. Here, distribution offset, the novel method that was developed for automatic artifact identification, outperformed the Hurst exponent technique in both artifact

removal and preservation of neural data. Neither ICA method performed as well as wavelets, but were both far superior to regression and PCA.

With its performance and full automation, the multi-level wavelet method is a valuable tool for processing neural signals. It provides a means of removing a common artifact that can heavily corrupt neural data, but it requires no expertise to use. Also, the wavelet technique has the distinct advantage of not needing the EOG reference channels, thereby decreasing experimental complexity and possible sources of error while increasing the subject's comfort. These traits make it an important step towards meeting this dissertation's overall goal of providing effective and accessible methods for processing neural signals.

As with the line noise discussed in the previous chapter, the algorithms developed for OAs were able to take advantage of known characteristics of the noise in order to effectively remove it. Line noise is sinusoidal near a known frequency, and OAs produce a relatively low frequency, high amplitude burst that can be localized in time. Some noise in neural data, though, can come from an unknown source or is not as well defined. Noise from EMG activity, for example, is broadband in nature and can have a wide, varying amplitude. This type of noise is discussed in the next chapter.

Chapter 5

Broadband Common Mode Noise

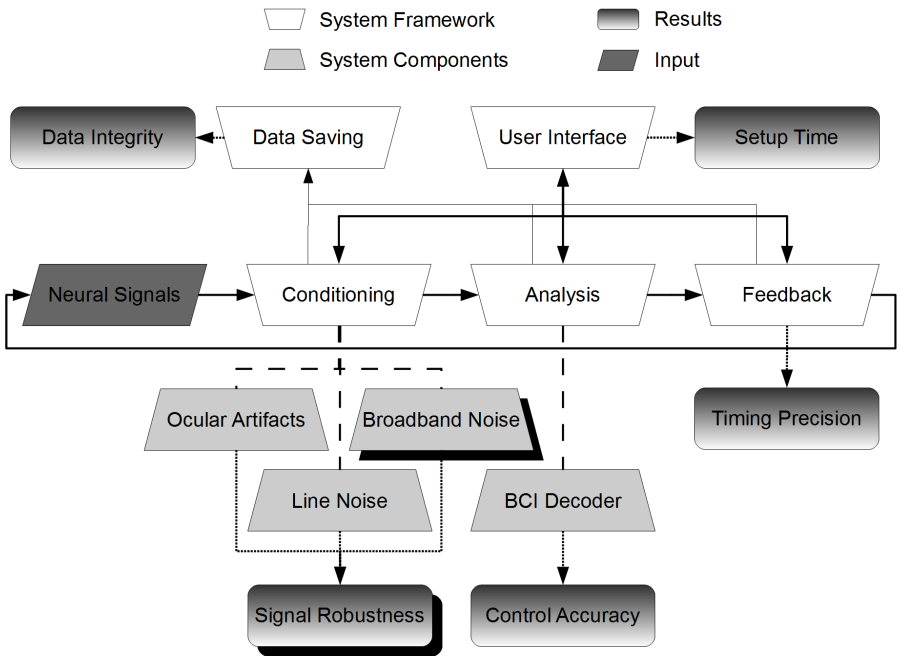

Figure 5.1: Overview diagram highlighting broadband noise removal. *The boxes with shadows indicate the portion of the system that is discussed in this chapter.*

5.1 Introduction

Sometimes the source and characteristics of noise in neural data are known, such as with power line noise, cardiac rhythm, or ocular artifacts (OAs), and it can be removed by filters specifically aimed at its characteristics such as those presented in Chapters 3 and 4. In many cases, though, contamination is present that is more difficult to isolate. Some noise in neural signals is broadband in nature, highly variable in power, and not always well-isolated in time. Fig. 5.1 indicates this noise in the main system diagram.

Electromyographic (EMG) activity from muscle movement is an excellent example of this type of noise [7]. The broadband nature and large amplitude of EMG artifacts makes filtering the noise or using any part of the signal more challenging [13]. A few likely characteristics of this noise do exist that can be utilized in filters, though, such as spatial correlation. Due to the relatively low signal-to-noise ratios (SNRs), small recording areas, and conductivity of the surrounding tissues, most contamination in neural recordings affects multiple channels. In any multi-channel physiological data contamination that affects a large number of channels is often referred to as a common

mode artifact.

The goal of this chapter is to demonstrate a robust and automated method for removing common mode artifacts from neural signals. The method presented is based on a combination of the popular common average reference (CAR) [73] and an adaptive noise canceling (ANC) filter [44], and as a result it is referred to as the adaptive common average reference (ACAR). The basic idea of the ACAR is that for each iteration a weighted CAR is used as a reference signal in an ANC filter for each channel, and the correlation between the reference and the ANC output is then used as a weight for each channel in calculating the reference on the next iteration.

Relevant background methods and material are discussed in Section 5.2. The full implementation and details of the ACAR are covered in Section 5.3. In order to obtain quantitative results, much of the data used for analysis was simulated. For further verification real neural data was used, but these results can only be visually presented since the true signal and noise are not known. The performance of the ACAR, independent component analysis (ICA), and the CAR is compared in Section 5.4. The conclusion is presented in Section 5.5.

5.2 Background

5.2.1 Multi-Channel Physiological Recordings

A generalized model for a multi-channel recording contaminated by noise can be given as follows:

$$D_{m,k} = \sum_{U} \alpha_{m,u} * S_{u,k-\tau_{u,m}} + \sum_{V} \beta_{m,v} * Q_{v,k-\eta_{v,m}} \tag{5.1}$$

where D is the recorded signal channel m at time point k, S is the actual uncontaminated signal from source u with a propagation delay of τ to each recorded channel, α determines the contribution of each S to each channel of D, Q is the noise source v with a propagation delay η and a contribution of β to each channel.

For common mode artifact removal in physiological signals a simplified approach is often used.

The propagation delays in (5.1) are assumed to be zero, and the signal processing and analysis can be done at the sensor level instead of the source level [34]. By treating the recording at each sensor as its own signal, (5.1) can then be simplified by the following: $U = M$ and α is an identity matrix. The assumption of no propagation delay also means that $\tau = 0$ for all u, m and $\eta = 0$ for all v, m. These simplifications are shown in (5.2).

$$D_{m,k} = S_{m,k} + \sum_{V} \beta_{m,v} * Q_{v,k} \tag{5.2}$$

Physiological noise is usually the summation of many sources (for example muscle fibers or neurons firing together) as represented in (5.2). It is sometimes assumed, though, that the relative contribution of each source of common mode noise to each sensor is constant, meaning the noise portion of (5.2) can be simplified as shown in (5.3). So (5.1) simplifies to (5.4), where γ_m and R_k are defined in (5.3). Finally, this is often simplified further to (5.5).

$$\begin{aligned} \sum_{V} \beta_{m,v} * Q_{v,k} &= \sum_{V} \gamma_m * \delta_v * Q_{v,k} \\ &= \gamma_m * R_k \end{aligned} \tag{5.3}$$

$$D_{m,k} = S_{m,k} + \gamma_m * R_k \tag{5.4}$$

$$D_{m,k} = S_{m,k} + N_{m,k} \tag{5.5}$$

5.2.2 Common Average Reference

The model in (5.4) gives rise to the popular CAR used in physiological recordings, which has been proven effective under the right conditions but does have many limitations [74]. The CAR simply subtracts from each channel the average across all channels, and is given below in (5.6), where the

$E_m[D_k]$ operator represents the expected value of D_k across all M channels. It is assumed that the signals of interest are uncorrelated. The idea is that the reference then mostly contains the common mode noise since the signal portions of the recording should average out. If in (5.4), $\gamma = 1$ for all m and $E_m[S_k] = 0$, then the CAR would provide perfect removal of R.

$$\hat{S}_{m,k} = D_{m,k} - E_m[D_k] \tag{5.6}$$

One problem with the CAR comes from the fact that differing channel characteristics can cause the noise to have unpredictable amplitudes and even polarity in each channel ($\gamma \neq 1$ for all m in (5.4)). With these difficulties the CAR could actually be harmful if some channels were not originally affected by the noise or if amplitude and polarity differences caused an inaccurate reference to be generated.

5.2.3 Independent Component Analysis

The shortcomings of the CAR have led to other algorithms being used for removal of spatially correlated noise in multi-channel physiological recordings, with blind source separation (BSS) techniques being among the most common [5]. One of the most well-known methods is ICA. ICA has seen widespread use in numerous problems involving identifying individual signal and noise sources in multi-channel recordings, and as a result is a good standard to measure against [75].

In the context of noise removal, ICA is used in an attempt to isolate the noise to its own independent component(s). Any noise component must then be identified, and its corresponding row in the mixing matrix is set to zero before re-mixing the components back into the original signal space (Fig. 4.4). At times identification of the noise components can be done algorithmically. ICA guarantees nothing about the order, amplitude, or polarity of the separated components, though, so at times this process can be complex and possibly require manual identification [62]. This task is made more difficult if the noise and signal characteristics are similar or unknown.

Also, ICA can fail to separate the signals in many situations depending on the convergence

algorithm, including the cases where there are more sources than sensors or where the sources are not spatially stationary. Even if the sources are spatially stationary, ICA cannot be used in real-time unless a recorded segment of data is available beforehand with which the mixing matrix can be calculated and the noisy components determined.

5.3 Methods

5.3.1 Adaptive Common Average Reference

The method presented here, referred to as the adaptive common average reference (ACAR), attempts to combine the strengths of the CAR and of an ANC filter. The ANC filter, which was described in Section 3.2.2, has been shown to be extremely effective in removing noise if an accurate reference is available [45].

As in Chapter 3, the ANC filters here used the normalized LMS (NLMS) convergence algorithm, which is given again in (5.7). For stability, a step size u between 0 and 1 is required. σ_X^2 can be estimated from a segment of X. In this implementation a one second segment of data was used, although the exact value is not critical as long as it is long enough to provide a stable estimate and short enough to adapt to changes in X. The filter length is represented by L. Decreasing L has similar effects to increasing the step size and in general should be as small as possible to model the necessary system. Since the desire here was to show that the ACAR works with a generalized setup for an unknown system rather than fine-tuning multiple parameters for a specific data set, L was set to 10 for all data.

$$W_{k+1} = W_k + \frac{2u\hat{S}_k X_k}{L\sigma_X^2}, \qquad 0 < u < 1 \tag{5.7}$$

The CAR is in general able to produce a usable reference signal for spatially correlated noise, which is needed for X. The CAR might not provide a reference good enough for an ANC filter to be fully effective, but it is good enough for the ANC filter to begin converging. The filter output

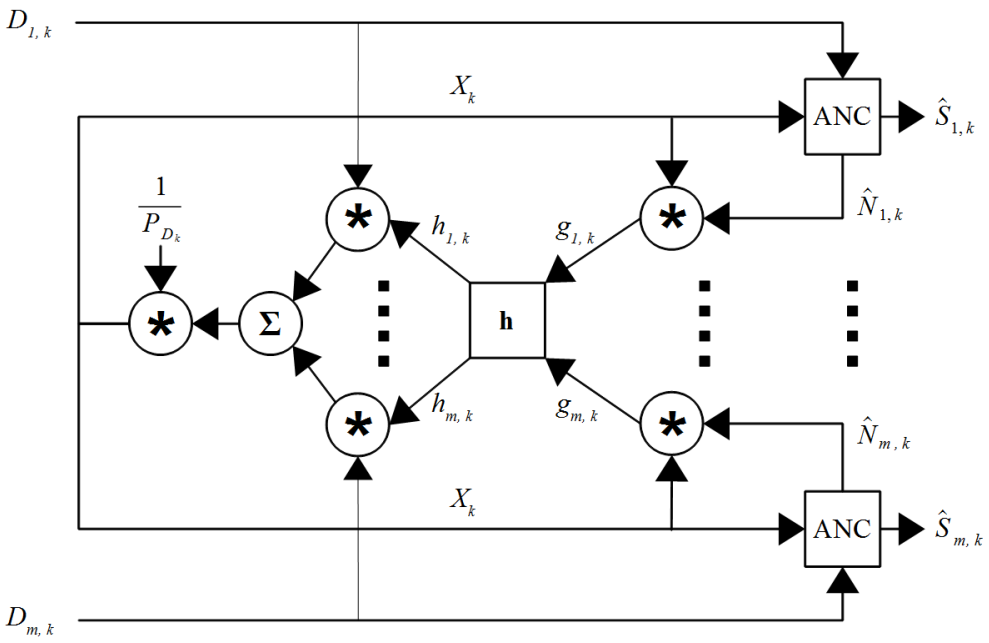

Figure 5.2: Block diagram of the ACAR. D is the recorded data channels and $\hat{S}$ is the filtered output. The ANC block contains the contents of Fig. 3.2 and the h block performs a moving average on g and normalizes the weights used for each channel in generating the reference as given in (5.9). P_{D_k} is the average signal power of all channels of D in a moving window as given by (5.10). *Note:* Until the window used by h is filled, g is calculated as the product of X and D as explained in the text.

can then be used to improve upon the reference, which in turn makes the ANC filter more effective. These two convergent processes build upon each other to effectively reduce the spatially correlated noise. A diagram of this process is shown in Fig. 5.2.

In the diagram, D is the original recorded signal as in (5.1), and each ANC block contains the contents of Fig. 3.2. On initialization, X is the standard CAR. For each channel of the recording, if X is correlated with N then the scale and polarity of N relative to X can be estimated by the product of X with $\hat{N}$, which produces g in Fig. 5.2. If the noise changes polarity across channels, then the polarity information provided by g can alone greatly enhance X. The magnitude is also useful as it allows the channels to be weighted by the relative strength of the noise that is present in each channel.

It should be noted that g is a rudimentary measure of correlation between X and $\hat{N}$. Calculation of correlation usually also involves z-scoring the data, but the mean is not removed since the noise

could have a DC component. Also, the ANC filter adapts $\hat{N}$ based on the noise in each channel and normalizing by the standard deviation of $\hat{N}$ would penalize noisier channels. Normalizing by the standard deviation of X is also not necessary since normalization of g and the reference signal is performed later and thus only the relative values of g are important at this step.

The values of h are calculated by smoothing g with a moving average and then rescaling it as shown in (5.8) and (5.9). As with all windows used in this chapter, any reasonable length should work and so T was set to be one second of data. The $\max_m(|\bar{g}_k|)$ operator represents the maximum of $|\bar{g}_k|$ across all M channels.

$$\bar{g}_{m,k} = \frac{1}{T} * \sum_{i=1}^{T} g_{m,k-i+1} \tag{5.8}$$

$$h_{m,k} = \frac{\bar{g}_{m,k}}{\max_m(|\bar{g}_k|)} \tag{5.9}$$

It should be noted that the initial samples of $\hat{N}$ before the ANC filter begins converging are inaccurate and h cannot provide much smoothing until its moving average window is filled. Due to this concern, the system implemented here calculates g as the product of X and D until the window used by h is filled. This modification had no discernible affect on the steady state error of the system, but did improve the speed and consistency of convergence.

After h is calculated, it is used to weight each channel before summing and producing X for the next iteration. As a final step, the reference is scaled by the average power across all channels of D given by (5.10). This scaling ensures that the reference signal remains stable and does not undergo rapid changes in power as the ANC filters adapt. As a fail-safe, the ANC filter coefficients are continuously monitored by computing the reflection coefficients with Levinson recursion and checking stability with the Schur-Cohn algorithm [48]. If instability is detected the ANC filter maintains its previous coefficients and D is passed through to the output unchanged.

$$P_{D_k} = \frac{1}{T * M} * \sum_{j=1}^{M} \sum_{i=1}^{T} D_{j,k-i+1}^2 \tag{5.10}$$

5.3.2 Independent Component Analysis

In order to give ICA optimal results and not have its performance depend on an algorithm for detecting the noise component, an oracle was used to determine which independent components were noise. Each component was individually removed and the one was selected that, when removed, resulted in the highest SNR in the reconstructed signals. This process was repeated iteratively until the SNR could not be further improved. In this way, if ICA did not achieve full separation of the data then multiple components containing noise could be eliminated. For real data, the noise components were manually identified by two experts in the analysis of neural recordings. In practice, automatically selecting the correct component can be difficult and time-consuming [62].

ICA was implemented using RobustICA, which has shown excellent results for the kind of data used here [76]. RobustICA attempts to maximize the non-Gaussianity of the sources. The algorithm iteratively calculates the normalized kurtosis contrast function and can separate any component that has non-zero kurtosis. It should then be capable of separating any data set that has at most one Gaussian source. The use of higher order moments such as kurtosis is common amongst ICA methods. In some circumstances, though, such as when multiple signal sources are Gaussian, BSS algorithms that rely on other measures can produce superior results [77].

5.3.3 Data Collection

Simulated Data

Most of the data used for analysis was simulated to ensure that the target signal was known. Signals were generated at 1200 Hz using the Craniux software suite presented in Chapter 2 [14]. S in (5.4) again consisted of pink noise with a $1/f$ power falloff to simulate a baseline electrocorticography (ECoG) recording [54]. This pink noise was created by generating uniform white noise and passing it through a digital filter with a $1/f$ response. Additional uniform white noise was added to the filtered signal, resulting in data that was sub-Gaussian with a kurtosis of about 2.4.

The noise, R, consisted of Gaussian white noise. For each trial, the mixing vector γ was

generated with each element as a random number between -1 and 1. This vector was then scaled to provide a specified average SNR as calculated in (5.11). The average SNR was calculated as the mean signal power over the mean noise power, which ensured that a constant SNR also provided a constant mean squared error (MSE).

$$\text{SNR}_{average} = 10\log_{10}\left(\frac{E_m[E_k[S^2]]}{E_m[E_k[(\gamma * R)^2]]}\right) \tag{5.11}$$

Real Data

The ACAR filter was also tested on real ECoG data. Although the true, noise-free signal was not known for these recordings, it is useful to at least qualitatively show that the method performs well on real data. All data was collected with g.USBamp (Guger Technologies) amplifiers and the raw signals were sampled at 1200 Hz. A standard 4 Hz wide notch filter was used to remove line noise and its harmonics. The data was recorded with subdural ECoG while the subjects attempted to use high gamma band modulation to control cursor movement in a 2D space.

The first data set was collected from a human subject who was implanted with a 32 channel grid over primary motor and sensorimotor areas. The subject was a 30 year old male who suffered a complete C4 spinal cord injury 7 years prior. All data collection and procedures were approved by the University of Pittsburgh's Institutional Review Board and informed consent was obtained prior to implantation. The second data set was obtained from a non-human primate who had 12 electrodes over primary motor and pre-motor areas. All data collection and procedures were approved by the University of Pittsburgh's Institutional Animal Care and Use Committee.

5.3.4 Analysis

ICA and the CAR were implemented for comparison to the ACAR. The CAR was not used when the noise had polarity changes across channels due to its poor performance in this situation. Unless otherwise noted, each result was averaged over 50 trials that were each 20 seconds long with 16 channels of data and an average SNR before filtering of 0 dB. As in Chapter 3, the error after

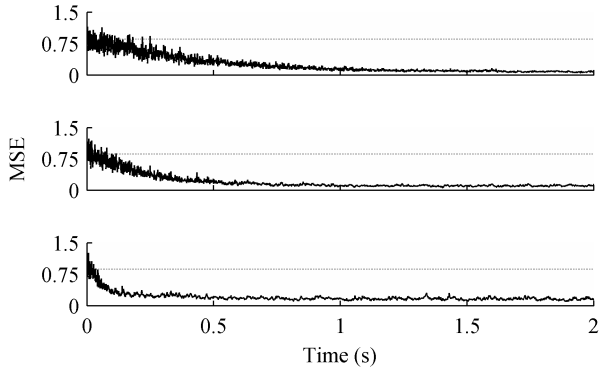

Figure 5.3: MSE for the ACAR with variable step sizes. The dotted line is the average unfiltered MSE. Step sizes were (Top to Bottom) 0.005, 0.01, and 0.05.

Table 5.1: SNR (mean $\pm$ standard deviation) for variable ACAR step sizes

Step Size	SNR (dB)
0.005	10.0 ± 0.8
0.01	9.2 ± 0.9
0.05	7.1 ± 0.7

filtering was calculated as the difference between the true signal and the filtered signal. The MSE after filtering was used to examine convergence speed. The SNR was used to measure steady state error, so its calculation ignored the first 5 seconds of each trial.

5.4 Results and Discussion

5.4.1 Simulated Data

ACAR Learning Rate

The ACAR was first tested with different step sizes, u in (5.7). As expected Fig. 5.3 shows that smaller step sizes take longer to converge, but result in a smaller steady state error as confirmed by Table 5.1. The optimal learning rate varies depending on many factors, but the goal here, is to show that a single ACAR setup works well under a variety of circumstances. So for the remainder of the analysis a step size of 0.01 was used. Fig. 5.4 shows a histogram of the filtered SNR for the 50 trials with a 0.01 learning rate. The ACAR improved the signal quality for every trial.

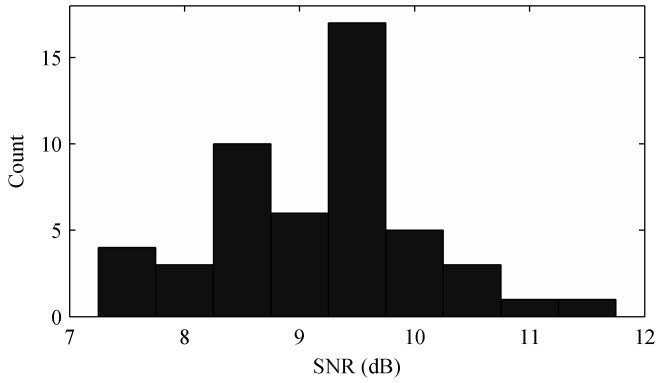

Figure 5.4: Histogram of SNR for the ACAR with a 0.01 step size.

Table 5.2: SNR (mean $\pm$ standard deviation) for variable initial SNRs

Unfiltered	ACAR	ICA
-20	11.5 ± 0.5	1.2 ± 0.5
-10	11.2 ± 0.5	3.1 ± 0.7
-5	10.5 ± 1.0	5.2 ± 1.0
0	9.2 ± 0.9	5.7 ± 1.0
5	6.5 ± 1.0	7.1 ± 0.9
10	3.1 ± 0.5	8.1 ± 0.6
20	3.6 ± 0.3	10.5 ± 1.3

Variable Initial Noise Power

Next, the behavior of the ACAR with different initial SNRs was examined by varying the SNR for each session. Fig. 5.5 shows the convergence of the ACAR for different initial SNRs. Although it is difficult to tell due to the scales, the ACAR did take slightly longer to fully converge at the lower SNRs. Table 5.2 shows the resulting SNR after filtering this data using both the ACAR and ICA. The ACAR remains consistent and exceeds the performance of ICA up until an initial SNR of 5 dB. The lower the initial SNR, the more accurate of a reference the ACAR was able to generate, which compensated for the problem of removing higher levels of noise. At high input SNRs, the ACAR was unable to converge upon an accurate reference.

In further testing the ACAR remained consistent down to around -280 dB, after which point it continued to attenuate about 290 dB of noise but could not maintain the output at over 10 dB as seen with the lower SNRs in Table 5.2. At -280 dB the filter took approximately 20 seconds to

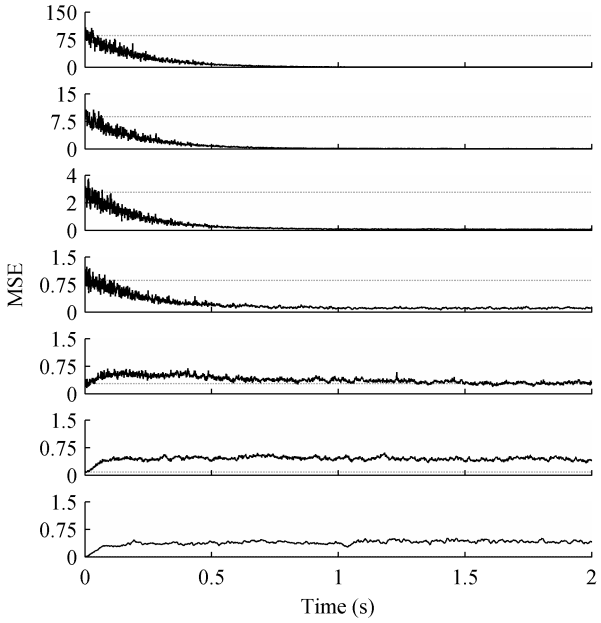

Figure 5.5: MSE for the ACAR with variable initial SNRs. The dotted line represents the average unfiltered MSE. The initial SNRs in dB were (Top to Bottom) -20, -10, -5, 0, 5, 10, 20.

fully converge. For arbitrarily high input SNRs, the ACAR maintained an output SNR of 3 to 5 dB. ICA was also unable to improve its performance for higher SNRs and maintained an output SNR of about 10 to 12 dB as the input SNR increased beyond 20 dB.

So the improvement in signal quality from the ACAR increased with decreasing input SNRs. At about 5 dB the ACAR began to struggle to converge and at higher input SNRs the signal quality was made worse by the ACAR, although an output SNR of over 3 dB was maintained. ICA outperformed the ACAR at these higher input SNR levels, but it also hurt the signal quality for tested input SNRs greater than 5 dB. With high SNRs it becomes difficult for many filters to remain effective since signal distortion caused by the filter begins to outweigh the benefit of noise removal. Most physiological recordings that the ACAR would target would have fairly low SNRs.

Variable Number of Data Channels

The performance of the ACAR as the number of data channels changed was also tested. Fig. 5.6 shows the convergence of the filter for each number of channels, and Table 5.3 shows the converged

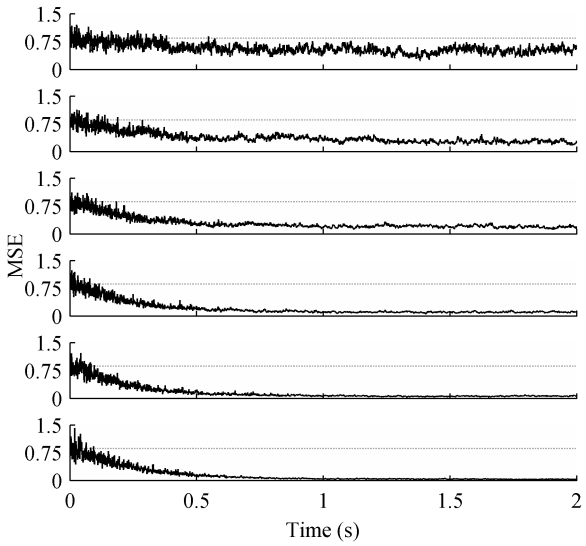

Figure 5.6: MSE for the ACAR with a variable number of data channels. The dotted line represents the average unfiltered MSE. The number of data channels were (Top to Bottom) 2, 4, 8, 16, 32, and 64.

Table 5.3: SNR (mean $\pm$ standard deviation) for a variable number of data channels

Channels	ACAR	ICA
2	2.4 ± 0.3	1.9 ± 1.1
4	4.7 ± 0.5	3.7 ± 1.3
8	7.2 ± 0.8	4.7 ± 1.1
16	9.2 ± 0.9	5.7 ± 1.0
32	11.6 ± 0.9	6.4 ± 0.8
64	13.8 ± 1.0	7.1 ± 0.5

SNR. As can be seen, the filter converged more quickly and smoothly as the number of channels increased, and the converged SNR improved as well. As the number of channels increase the signal itself is more likely to average out to zero when generating the reference signal, resulting in a cleaner reference for the noise. ICA also improved as the number of channels increased, but in all cases was outperformed by the ACAR in both mean and in consistency.

Variable Noise Source Conditions

The ACAR was also tested under conditions in which the noise did not have consistent power or spatial stationarity. Both the initial SNR and the mixing vector were adjusted according to

Table 5.4: SNR for variable noise conditions, calculated over 20 minutes of data.

Condition	Unfiltered	ACAR	ICA
Var Mix	0.0	9.2	1.9
Var SNR	-2.1	7.8	11.9
Var Mix, SNR	-5.4	7.6	0.9

the Gauss-Markov process shown in (5.12). In this process a is the value being updated, Δ is the timepoint at which the update occurs, and η is a value drawn from a zero-mean Gaussian distribution with a specified standard deviation.

$$a_{\Delta_{k+1}} = a_{\Delta_k} + \eta_{\Delta_k} \tag{5.12}$$

For the first trial, the SNR was held constant at 0 dB and every 2 seconds each value in the mixing vector was changed by η with a standard deviation of 0.1. The values were clipped between -1 and 1 and re-normalized. For the second trial, the mixing vector was held constant and the SNR was adjusted every 2 seconds by η with a standard deviation of 1. The SNR was clipped if it went outside the bounds of -10 to 10 dB. For the third trial, both the mixing vector and the SNR were varied. Each trial was 20 minutes long.

Table 5.4 shows the SNR across each trial for both the ACAR and ICA. The ACAR improved the signal quality across all trials and maintained performance that is consistent with what was seen in earlier trials. ICA performed poorly on the two trials in which the mixing vector varied, which is expected since ICA expects spatially stationary sources. On the trial where only SNR varied, though, ICA performed very well. The noise power changing most likely caused the distribution of the noise over the entire trial to be non-Gaussian, allowing ICA to better separate it into its own component.

Signal and Noise Gaussianity

The Gaussianity of the signals and the noise can greatly contribute to the performance of an ICA algorithm that relies on kurtosis. As stated earlier, the simulated signals were sub-Gaussian while the

Table 5.5: SNR for different signal and noise distributions

Signal Distribution	Noise Distribution	ACAR	ICA
Uniform	Gaussian	11.4	7.1
Sub Gaussian pink noise	Uniform	9.2	10.2
Uniform	Uniform	11.4	12.1

noise itself was Gaussian. In order to compare results to ICA under more optimal (i.e. less Gaussian) conditions, tests were done with signal and noise sources that were uniformly distributed. Table 5.5 shows these results.

Making the signals uniform, causing them to be even more sub Gaussian, improved ICA's results slightly. The biggest advantage, though, was seen in making the noise uniformly distributed, despite it being expected that RobustICA could separate the data when only one Gaussian source was present. ICA outperformed the ACAR when the noise was uniform, but the ACAR did stay within 1 dB. Interestingly, the ACAR's results were slightly improved by having uniformly distributed source signals. This could be explained by lower frequency signals having better odds of being correlated by chance, and the $1/f$ signals are skewed more towards the lower end of the frequency spectrum.

Signal Correlation

In general the ACAR makes the assumption that the source signals are uncorrelated, and thus anything spatially correlated across the recorded channels is considered noise. This is not always the case in practice, especially in physiological recordings where the signal sources can often propagate to multiple recording sites. To examine the performance of the ACAR in this non-ideal, but realistic situation, a simulated data set was created in which the signals were created from mixing the independently generated sources.

For one condition a square matrix, with height and width equal to the number of data channels, was created with the elements being uniformly distributed between 0 and 1. The original sources were then multiplied by this matrix to create the actual signal channels seen by the filters. For

Table 5.6: SNR for correlated signals

Condition	ACAR	ICA
Fully random	4.5	7.7
Spatially normalized	8.0	7.7

the second condition the mixing matrix was symmetric with all ones on the diagonal. The off-diagonal elements were chosen in the same manner as the previous condition except this time they were scaled by the difference in the row and column index. This means that each source was treated as local to one channel, but it spread to neighboring channels with a gain proportional to distance. This behavior is similar to what can be expected in real physiological recordings. The results for these data sets are shown in Table 5.6.

The ACAR was still able to improve the signal quality under the fully random condition. ICA outperformed the ACAR in that case, and ICA received a modest boost in performance compared to previous results in which the sources were not mixed. For the second condition, in which the strength of each source was scaled by the distance to a channel, the ACAR performed well and was on par with the results of ICA.

ACAR was still effective in these situations because the noise remained the dominant source in the recordings and the single dominant source is what the filter is designed to find. In the fully random condition an SNR of 0 dB means that the noise still on average contributes as much power to each channel as the independent sources combined. In the spatially normalized condition the noise is even more dominant due to the attenuation of the independent source by distance. This condition also makes it less likely that the spatial distribution of any signal source would overlap with the distribution of the noise and thus get removed along with the noise.

Changes in Noise Polarity

Finally, the effect of the polarity of the noise across channels was examined. In the 'uniform' condition the noise was added to each channel with the same polarity and scaling factor. In the 'monopolar' condition the noise was given the same polarity across all channels, but still had a

Table 5.7: SNR for different noise polarities

Noise	ACAR	ICA	CAR
Uniform	9.3 ± 0.8	5.7 ± 1.0	11.8 ± 0.5
Monopolar	9.3 ± 0.8	5.6 ± 0.8	4.7 ± 0.9
Bipolar	9.2 ± 0.9	5.7 ± 1.0	-0.4 ± 0.6

random scaling factor between 0 and 1. For reference, the 'bipolar' condition used in previous tests is also included. This data allowed the results for the CAR to be included for comparison. The results are shown in Table 5.7.

Table 5.7 shows that the ACAR and ICA performed consistently across all conditions. As expected the CAR performed poorly on the bipolar condition, better on the monopolar condition, and on the uniform condition exceeded even the performance of the ACAR. The uniform condition is the ideal situation for the CAR and in this case provides an upper bound on how well the ACAR could be expected to perform under those circumstances. The ACAR was unable to perfectly converge to the CAR for the uniform condition, but as the characteristics of the noise began to move away from the uniform condition the performance of the CAR dropped steeply while the ACAR maintained consistent performance.

5.4.2 Real Data

Human Data

The results of the ACAR on real data were also promising. Fig. 5.7 shows the 32 channels of human-subject ECoG data after high-pass filtering at 0.1 Hz, and after application of the CAR, ICA, and the ACAR. The raw data contained a highly periodic common mode artifact from an unknown source. It is unlikely this noise was physiological in nature due to its timing. A much larger artifact was also present in the recording at about 100 seconds.

As shown, the CAR managed to clean the artifact much of the time for most channels. It struggled with some channels, though, such as 4 and 19, and it added the artifact to channels where it was not initially present, such as 8 and 9. It also failed to provide much improvement to

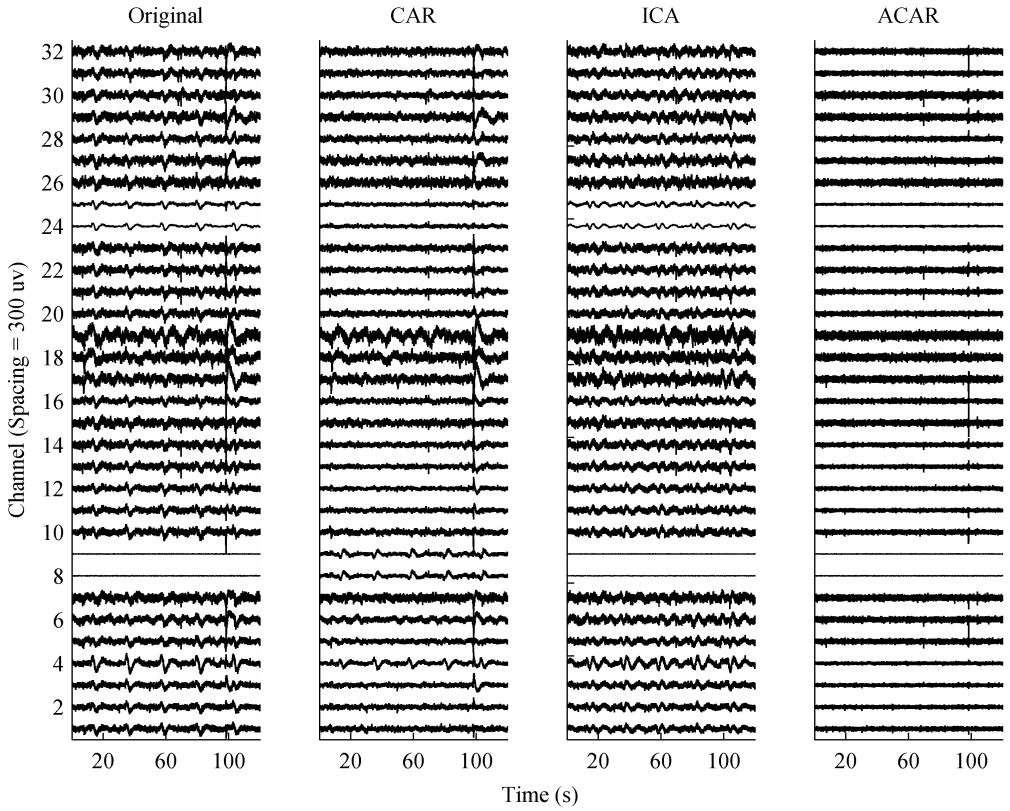

Figure 5.7: ECoG data with common mode artifacts. The data was processed by (Left) high-pass filtering at 0.1 Hz, (Left Middle) high-pass filtering and then using a CAR, (Right Middle) high-pass filtering and then using ICA, and (Right) applying the ACAR.

the large artifact near the end of the recording, which is probably partly due to the fact that there seems to be some polarity changes in that artifact across channels. ICA removed the large artifact better than the CAR and it did not adversely affect clean channels, but it still failed to remove much of the contamination.

The ACAR consistently removed the periodic artifact, and avoided disturbing the channels where the artifact was not present. Most of the the large artifact was also removed, although it is still noticeable in most of the channels. This remaining spike is most likely the result of the ACAR not adapting quickly enough to such a large artifact. It should be noted that due to amplifier characteristics the data had a large DC offset and low frequency drift, which is normally eliminated with a high pass filter. The ACAR was able to remove this offset and correlated drift on its own in addition to the periodic noise.

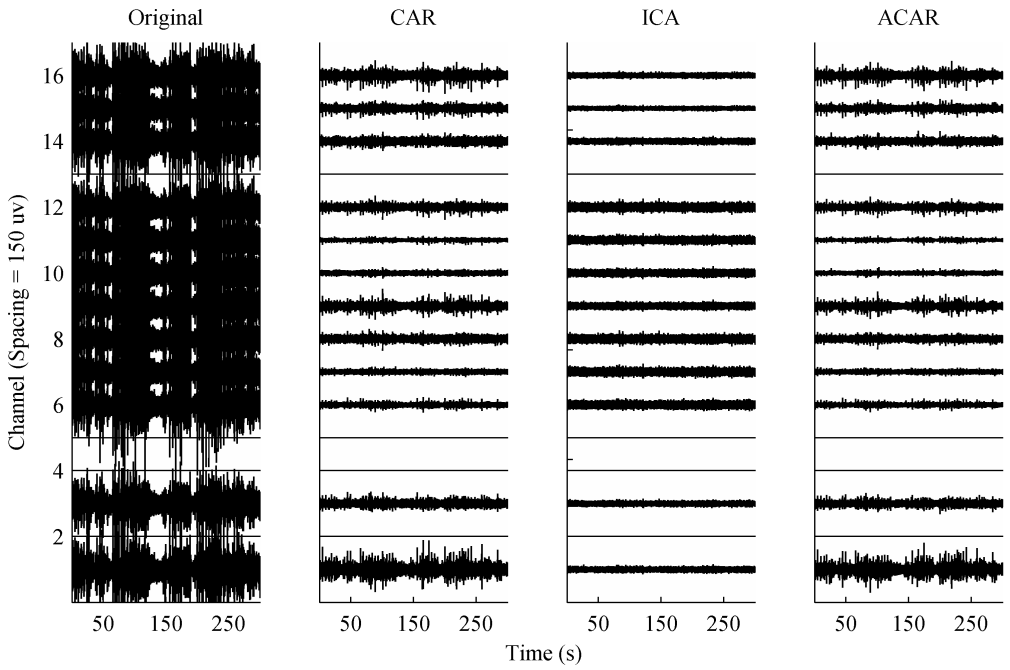

Figure 5.8: ECoG data with heavy EMG contamination. The data was processed by high-pass filtering at 1 Hz and then (Left) no additional filtering was used, (Left Middle) a CAR was applied, (Right Middle) ICA was used, and (Right) the ACAR was applied.

Non-human Primate Data

The 12 channel non-human primate data was more heavily contaminated due to constant jaw and tongue movement by the subject. As can be seen in Fig. 5.8, the original signals were nearly completely masked by the EMG artifacts. The recording was also affected more strongly by low frequency drift and to achieve optimal results a high-pass filter at 1 Hz was applied before all methods, including the ACAR. The ACAR was capable of removing the drift on its own, but if allowed to do so its performance removing the EMG noise decreased as in this case the drift had a much different spatial distribution than the targeted noise. In this data set the recording contained 16 channels, but channels 2, 4, 5, and 13 contained no data and were ignored during filtering. For easier visualization the open channels have been set to zero in Fig. 5.8.

As can be seen, the CAR performed well on this data. The result of the ACAR highly resembles that of the CAR, and only through close examination or by overlaying the plots could it be seen that the ACAR reduced the artifacts by an insignificant amount more. This result is not surprising

for this data given that the contamination appeared to be monopolar and fairly uniform across channels. These conditions are ideal for the CAR, and it is good that the ACAR converged to nearly the same result. ICA performed extremely well in removing the noise, but this was at the expense of eliminating 5 out of the 12 components. It is unknown how much neural data was lost in the process. Without a knowledge of the true underlying signal, it is difficult to tell with certainty in this case whether the ICA result is more or less desirable than the result obtained by the CAR and the ACAR.

5.5 Conclusions

The ACAR is a method for effectively removing common mode artifacts from multi-channel physiological recordings. The technique works even with polarity changes in the noise between channels. It was found to be effective with as few as 2 channels of data, with performance improving further as the number of channels increased. It also showed consistent results in improving signal quality to around 10 dB on data with an average SNR in the range of about -280 to 5 dB. At higher SNRs the ACAR could not generate an accurate noise reference and degraded the signal quality, but consistently kept its output between 3 and 5 dB. At these higher input SNR levels a filter would not be needed in most physiological recordings. A noise detector could also be added that only triggered the ACAR when the noise power reached a certain level.

In addition to reducing constant, spatially stationary noise, the ACAR was found to produce consistent results under variable noise conditions. This includes situations in which the noise power, mixing vector, or both were changing over time. This is important since most sources of real noise will drift slightly in power or spatial location, although probably not to the extent that was tested here. Real multi-channel physiological data also often contains correlation between the signals recorded by each channel, so this situation was tested as well. Although the ACAR's performance did decrease in some of these conditions, it still consistently improved signal quality.

Through visual inspection the ACAR was found to be effective in removing various artifacts

from real data. Most conditions tested with both real and simulated data showed that for removing spatially correlated noise from multi-channel recordings, the ACAR was superior to a standard CAR and in many cases better than the RobustICA algorithm. This was with an implementation of ICA in which an oracle determined the noise component that should be removed before reconstructing the signals. Additionally, the ACAR was able to improve the quality of the underlying neural signals of interest in real data that was corrupted by simulated noise.

The ACAR was used in a generalized form without changing any parameters across multiple conditions for simulated and real data, and so it should be easily usable in an automated fashion without configuration by expert personnel. Unlike ICA it can be easily implemented in real-time, which is a significant advantage for applications that need online analysis and processing, such as brain-computer interfaces (BCIs). The filter presented in this chapter showed potential for reducing common mode artifacts in both offline and online recordings of physiological data, and its performance here justifies further investigation and development.

There are situations, though, where the ACAR might not be the best choice for common mode artifact removal. Some of these examples were presented in this chapter. When the noise had no polarity or amplitude changes across channels the standard CAR performed better. As the signals and the noise became less Gaussian, ICA improved to a point where it exceeded the ACAR. Last, it was shown that correlation between the signals could decrease the performance of the ACAR. Although this result was expected, it is a highly realistic condition for multi-channel physiological recordings.

Additionally, the ability of the ACAR to uncover the underlying signals of interest needs to be further studied. Spatial filters, such as the CAR, have been shown to improve the performance of electroencephalography (EEG) BCIs [78], [58]. This topic is examined in the next chapter where a BCI decoder is presented as an example application for the methods developed in this dissertation. Examining the effect of the ACAR on such experiments is an important step in determining the level of impact it can have on neural recordings.

Chapter 6

Application - BCI Decoding

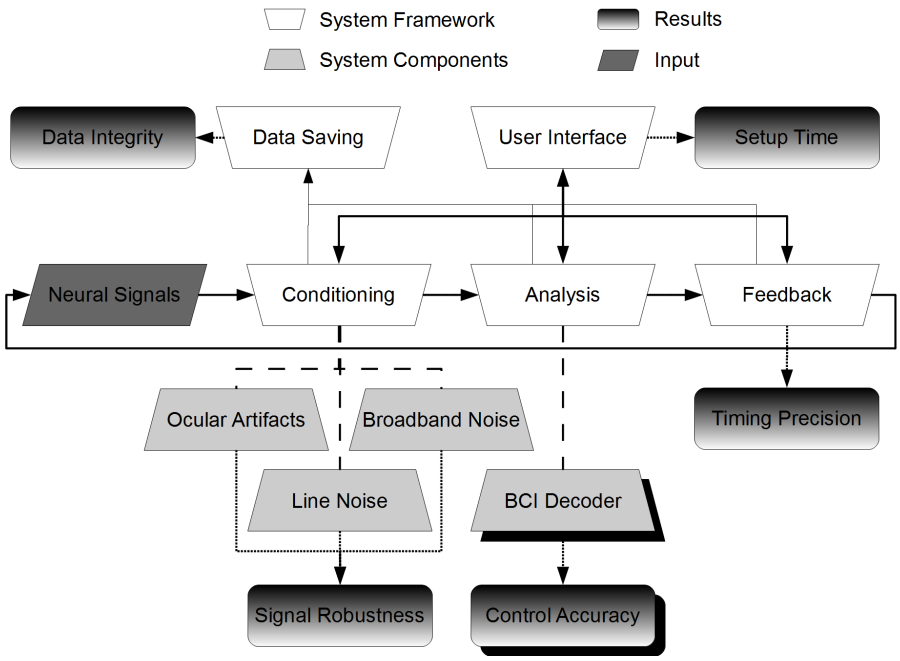

Figure 6.1: Overview diagram highlighting BCI decoding. *The boxes with shadows indicate the portion of the system that is discussed in this chapter.*

6.1 Introduction

The system framework presented in Chapter 2, as well as the filters presented in Chapters 3-5, all have the goal of increasing the accessibility and efficiency of the development, implementation, and use of technologies that harness neural signals. The effectiveness of these novel methods in simplifying and improving neural signal processing has been demonstrated, but the question still remains of the impact that these techniques could have on actual technologies that use neural signals. An example of such technology is brain-computer interfaces (BCIs), which decode neural signals in order to control a computer or device. This chapter presents a BCI as an example of an application for the work in the previous chapters. The BCI was implemented using elastic net regularization for linear regression, which was chosen based on its performance and the ability to automatically eliminate irrelevant features while retaining a robust feature set.

The goal of this chapter is not only to demonstrate and discuss the merits of this decoding method; it is also to show the potential impact of the algorithms and techniques presented in previous chapters when applied to a real application of neural signal processing. Fig. 6.1 highlights

the decoding portion of the system in the main diagram. Much of the material related to the decoding methods and data used in this chapter can be found in [79] and [2].

Section 6.2 covers the relevant background information for the decoder and Section 6.3 discusses its implementation and validation. Section 6.4 shows some results demonstrating the impact of the methods from this dissertation on BCI decoding. Finally, Section 6.5 discusses the conclusions and implications of this chapter.

6.2 Background

6.2.1 Brain-Computer Interface Decoding

BCIs have progressed greatly in recent years, and continue to move towards the goal of offering neural control of assistive devices. This progress is due in part to improvements in computing power, as well as recording and processing methods, that allow increasingly larger amounts of neural data to be processed and analyzed in real time. This increase in data generally increases the likelihood that useful signals are present, but it also causes an increase in the amount of irrelevant or noisy data.

In the case of a BCI the goal is typically to decode the neural data to produce a control signal for an external device such as a computer cursor, robotic arm, or wheelchair [80]. The decoding algorithm could be anything from a simple linear classifier to complex methods such as support vector machines (SVMs) or Kalman filters [81], [82]. Two popular methods for BCIs that decode movement are the population vector and the optimal linear estimator (OLE) [83], [29]. These two methods assume that every feature has a preferred direction of movement and solve for a set of weights that best reproduce the observed movement.

If not handled properly, extraneous or contaminated neural features can translate into noise in the output. The objective then should be to implement a neural decoding method that is invariant to irrelevant features. One strategy to choose the best features is to observe the modulation of the neural signals and then choose appropriate parameters for the decoder [34]. Neural plasticity then

typically allows the brain to further adapt to the selection [1]. This strategy has even been taken to the extreme in non-human primates, where it was shown that the brain could eventually adapt to randomly chosen features [84]. This strategy is time-consuming, though, and requires operation by highly trained personnel. For BCIs to be usable by non-experts, as is the goal for all methods in this dissertation, the decoding algorithm needs to be highly automated and robust.

6.2.2 The Curse of Dimensionality

The task of training a classifier with a large number of irrelevant features and a small number of observations is not unique to BCIs. Overfitting and the contributions of noisy features both become major concerns in these types of problems. Dimensionality reduction, feature selection, and regularization are methods often employed in high-dimensional decoding problems. Dimensionality reduction methods such as principal component analysis (PCA) and linear discriminant analysis (LDA) have been used in BCIs [75], but these techniques transform the features to a new basis, making it more difficult to interpret the real-world significance of the raw features. Feature selection retains the original basis, but might not capture as much of the original information as dimensionality reduction.

A number of BCI studies have used 'pure' feature selection methods. These methods, such as forward stepwise regression, only choose features and then separately solve for weights using a standard method such as ordinary least squares (OLS). For feature selection, forward stepwise regression adds the feature at each step that eliminates the most residual error (backward stepwise removes features at each step that eliminate the least amount of residual error). These techniques can be biased, since the best set of $M + 1$ features does not necessarily contain the best set of M features [85]. They can also be unstable, in that a small change in the data could result in a large change in the selected features. Stagewise regression attempts to minimize the problems associated with stepwise regression by increasing a feature's weight by a small amount at each step rather than all the way to the least squares solution. The adjusted feature could remain the same for multiple steps.

6.2.3 Regularized Linear Regression

Linear regression, which has been used frequently in BCIs, takes the form of the optimization problem given by (6.1). Y is a vector containing M observations, $\mathbf{X}$ is an $M \times P$ matrix containing P features for each observation, β is a vector of P weights that map the features to the observations, and θ is the bias, or offset, term. OLS is the simplest method for computing β based on this model.

$$\hat{\beta} = \underset{\beta}{\operatorname{argmin}} \|Y - \mathbf{X}\beta + \theta\|_2^2 \tag{6.1}$$

Regularization can help address overfitting and other problems with high-dimensional feature spaces. This technique adds a penalty term, represented by $\mathrm{c}(\beta)$ in (6.2). λ is a free parameter that determines the magnitude of the penalty. In the case of ℓ_2 regularization, sometimes referred to as ridge regression, the penalty is the ℓ_2-norm of β. In ℓ_1-regularization, also known as lasso (least absolute shrinkage and selection operator), the penalty is the ℓ_1-norm. The lasso penalty is more computationally challenging since it is non-differentiable, but it also performs feature selection by reducing some values of β to zero [86]. It has actually been shown that in stagewise regression as the feature weight step size optimally approaches zero, the result approaches the lasso [85].

$$\hat{\beta} = \underset{\beta}{\operatorname{argmin}} (\|Y - \mathbf{X}\beta\|_2^2 + \lambda * \mathrm{c}(\beta)) \tag{6.2}$$

6.2.4 Elastic Net

Lasso has proven to be be highly effective in classification problems with a large number of irrelevant features and has been used on neural data, however, it is not without drawbacks. In a situation where multiple features are useful but highly correlated, lasso tends to keep one and drop the rest. Stability then becomes a concern, and robustness could also be an issue in situations where not all features remain reliable over time due to noise or other events.

Elastic net blends the ℓ_1 and ℓ_2 penalties, as shown in (6.3). The goal is to produce a sparse feature space with the ℓ_1 penalty, but improve stability and retain correlated features with the ℓ_2

penalty. Like lasso this penalty is not a computationally simple problem, but efficient methods for solving it have been developed. There is also an additional free parameter, α, which determines the relative strength of the penalties. Previous studies have shown this technique to be effective in classification of functional magnetic resonance imaging (fMRI) data [87], [88].

$$\hat{\beta} = \underset{\beta}{\operatorname{argmin}}(\|Y - \mathbf{X}\beta\|_2^2 + \lambda(\alpha\|\beta\|_1 + (1-\alpha)\|\beta\|_2)) \tag{6.3}$$

6.3 Elastic Net Validation

6.3.1 Data

The datasets used for validation of the elastic net decoder consisted of electrocorticography (ECoG) signals recorded from two subjects undergoing monitoring for intractable epilepsy. Informed consent was obtained from both subjects prior to implantation, and all data collection and experimental procedures were approved by the Institutional Review Board of the University of Pittsburgh.

The signals were sampled at 1200 Hz with g.USBamp amplifiers and bandpass filtered from 0.1 to 200 Hz. The data signals were acquired using BCI2000 and were then sent to Craniux, the software framework discussed in Chapter 2 [22], [14]. Spectral estimation was performed for an autoregressive (AR) model with 10 Hz frequency bins, 300 ms windows, and a 33 ms step size. Subjects were observed to ensure eye and facial movements were not used to control the BCI.

In choosing the data for offline analysis, it was attempted to use experimental paradigms that have minimal online error correction by the user. Paradigms with error correction, such as a 2D cursor task in which the subject might not move along the ideal path to the target, present problems in offline analysis of a decoder. It can become difficult to determine the subject's exact intent and to incorporate the neural adaptation that is occurring as a result of error correction.

Subject A performed a simple hand grasp screening task. The subject was presented with a visual cue in the form of a gray box on a black screen, and was instructed to continually open and close the hand that was contralateral to grid placement while the cue was present. For Subject B,

the experimental paradigm was a 1D center-out cursor task. A cursor would appear on the screen along with a target to the right or the left of the cursor. The subject was instructed to perform hand grasps to move the cursor to the right, and to move their elbow to send the cursor to the left. The cursor was constrained to horizontal movement and the trial ended when the cursor touched the target. It is then assured that the subject was always attempting to move the cursor in the same direction for the duration of each trial. The hand and elbow used for movement were again contralateral to grid placement.

Subject A had 64 recorded channels: 48 from a standard clinical ECoG grid, and 16 from a high-density ECoG research grid. Subject B had 128 recorded channels: 62 from clinical grids, 32 from research grids, 2 electrocardiogram (EKG) channels, and 32 open channels. The EKG and open channels were left in the data to show the feasibility of an automated decoder with no supervision on channel selection. Data from Subject A consisted of 5 sessions with 24 trials each. For Subject B, data contained 4 sessions with 42 to 90 trials each for a total of 234 trials.

6.3.2 Classification

Decoding of the neural signals was done in an offline analysis using four different methods: elastic net, lasso, ridge regression, and OLS. The solutions for the first three methods were calculated using a modified version of glmnet, a freely available software package developed at Stanford University. Glmnet uses cyclical coordinate descent in a pathwise fashion and has previously shown excellent results and convergence speed [89], [90].

To determine the best value of λ for elastic net, lasso, and ridge regression, 10-fold cross-validation was performed for each training of the classifier across 20 different values of λ. A similar scheme was originally adopted to determine the best value of α, but it was found that this method generally caused the result to closely mirror the lasso solution. While this solution may indeed be the best fit for a particular set of training data, it fails to produce the robustness and stability that were earlier discussed as motivations for using the elastic net penalty. For this reason, α was set at 0.1 ($\alpha = 1$ is equivalent to lasso and $\alpha = 0$ is ridge regression).

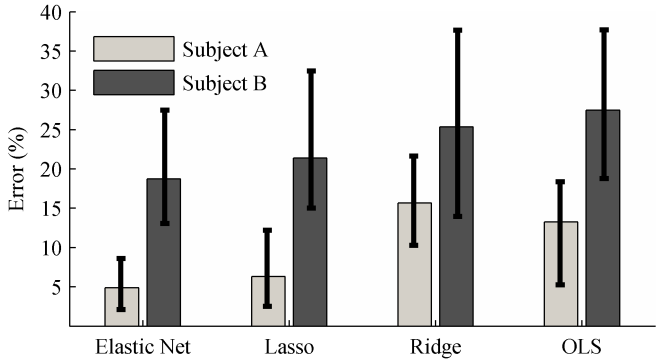

Figure 6.2: Percentage of timepoints classified incorrectly. The value shown indicates the percentage across all timepoints in all sessions. The error bars indicate the minimum and maximum percentage of timepoints classified incorrectly for an individual session.

Results for each session were calculated using 10-fold cross-validation. In the training set, the time-average of each feature over each trial was used, but in the testing set the decoding was done on each timepoint of spectral data as it would be in a real-time BCI. The main metric calculated was the percent of timepoints in which the decoder was correct. For Subject A this means determining whether the subject was grasping or not, and for Subject B this means determining if the cursor would move in the correct direction. Since this metric only determines the accuracy of the direction of movement and not magnitude, the change in distance to target was also measured for Subject B. For both subjects, timepoints that were within 500 ms of stimulus onset were ignored. This was to ensure that the spectral estimation window consisted of neural data produced after the subject had reacted to the stimulus.

6.3.3 Results

Fig. 6.2 shows the percentage of timepoints that were classified incorrectly for both subjects using each decoder. Elastic net had a lower error than the other decoders across all sessions for both subjects. The advantage over lasso in the average error is small, although elastic net did appear to be more dependable across sessions as indicated by the maximum session error for both subjects. The range of error across sessions was quite large for all decoders with Subject B, but as expected the error and consistency for the simpler task performed by Subject A was much better.

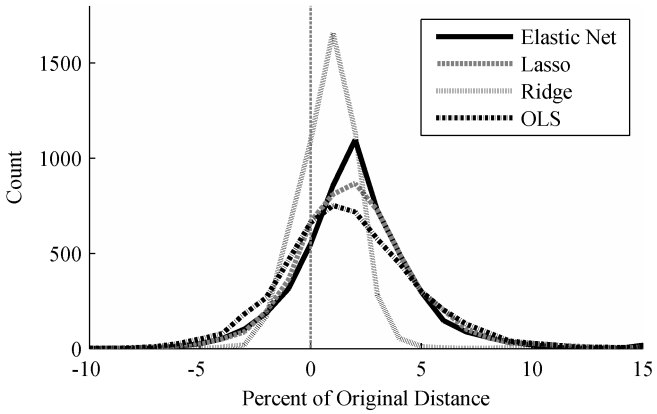

Figure 6.3: Change in distance to target for 1D cursor control. The distances were normalized by the original distance to the target, and then binned with a bin width of 1. The values represent the count of timepoints in each bin.

The better consistency of Subject A helped highlight the significant improvement of elastic net and lasso over ridge regression and OLS ($p < 0.05$ for all cases). It should also be noted that for similar tasks results are often reported for testing on time-averaged features for each trial rather than individual timepoints, which generally results in lower errors. For Subject B this method resulted in errors of 10%, 13%, 22%, and 20% (elastic net, lasso, ridge regression, OLS).

The results in Fig. 6.3 reinforce those given by Fig. 6.2. Additionally, they show that ridge regression produced a control signal that, although not as accurate on average, was much more stable than the other decoders in that it never moved the cursor a great distance in either direction. This could be desirable in operating physical devices such as robotic arms where sudden jerks and unpredictability could present a danger.

The sparsity of the weights used by the decoders is also important in their discussion. Fig. 6.4 shows the weights calculated by each decoder when trained on one session of data from Subject B. As expected OLS had no sparsity in its results and ridge regression, while having many weights that were close to zero, also did not give a sparse set of weights. Some banding can even be seen in these weights near 120 Hz and 180 Hz, which was most likely the result of line noise harmonics in the data. Lasso, on the other hand, produced a set of weights in which only 35 of the 2,560 weights were non-zero. For the elastic net decoder, 114 features had non-zero weights. Elastic net and lasso also chose no features from the 32 open channels at the end, although there were a few

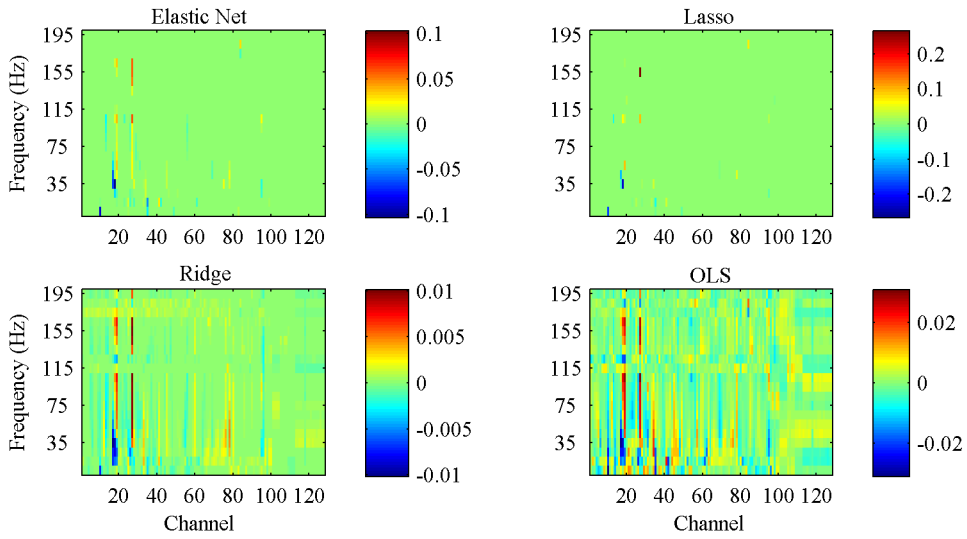

Figure 6.4: Decoder weights calculated from one session of data from Subject B

small non-zero weights on one of the EKG channels.

Many of the features eliminated by lasso but retained by elastic net closely neighbor a non-zero lasso weight both spatially and in frequency. For example, lasso used the 100-110 Hz bin on channel 27. Elastic net used this feature as well as the rest of the gamma band on channel 27 and channel 19, which was spatially adjacent. This extra redundancy in the decoder could be useful in a BCI where the features are subject to noise and are themselves adapting due to neural plasticity. The extra stability in the feature set would also be desired when re-training so that the decoding weights are not as much of a moving target for the BCI user. When trained on each session of Subject B data, not a single feature was common across all sessions for lasso.

6.4 Impact of Methods on BCI Decoding

6.4.1 Data and Methods

Additional data was collected from a human subject who was implanted with a 32 channel ECoG grid over primary motor and sensorimotor areas. The subject was a 30 year old male who suffered a complete C4 spinal cord injury 7 years prior. All data collection and procedures were approved

by the University of Pittsburgh's Institutional Review Board and informed consent was obtained prior to implantation. Data from this subject was also used in Chapters 3 and 5. In this chapter this subject will be referred to as Subject C.

For offline analysis, the signals were decoded using the elastic net algorithm presented in this chapter [79]. As in Section 6.3, training features were averaged over the presentation time of each cursor target and for testing each timepoint was decoded individually as it would be in a real-time BCI. Results were again calculated using 10-fold cross validation. To minimize the effect of online error correction attempted by the subject only the first one second of cursor movement data for each target was used (or until the cursor hit the target). In the real-time experiment the targets were presented 500 ms before movement could begun to ensure the subject had time to react to the target.

6.4.2 Craniux

Craniux was used with this subject for real-time data collection and BCI control. The software provided an excellent platform for efficiently conducting the experiments. Parameters could be updated during run-time, modules could be interchanged, system timing was consistent, and BCI decoding weights could be saved and loaded whenever desired. During real-time operation the system was configured with a standard notch filter for line noise removal, AR spectral estimation, and an OLS decoder. As with data presented earlier in this chapter, the spectral power in 10 Hz bins from 0 to 200 Hz was used as features for the decoder. With this Craniux setup the subject was able to achieve a success rate of nearly 90% in hitting targets with 2D cursor movement and 80% with 3D cursor movement, the first time such effective control has been demonstrated with ECoG in a subject with tetraplegia [2]. The data was also streamed to disk with important experimental parameters, allowing for further analysis in an offline environment.

For offline analysis the data could be processed by loading it back into Craniux or by using other software such as MATLAB (The MathWorks, Inc.). In the following sections, which examine the effect of noise and filtering on Subject C's data, a combination of these approaches was

used. The raw neural data was first loaded and replayed through Craniux with different system configurations. Modules were used that performed tasks such as adding simulated noise and applying the filters under investigation, including the adaptive common average reference (ACAR) and the adaptive sinusoid canceler (ASC). The newly processed data was again streamed to disk by Craniux. In MATLAB, the data was then decoded using the elastic net algorithm to analyze the effects of the filters on the decoding results.

6.4.3 Broadband Noise

Methods

The data from Subject C in which good BCI performance was obtained was relatively free of broadband common mode noise. It was desired, though, to examine the impact of the ACAR from Chapter 5 on removing such noise to reveal the underlying neural activity that can be used for BCI decoding. To do so, simulated noise was added to the real data. The use of simulated noise also ensured that the contamination would not be correlated with the desired decoder output, so the decoder performance should decrease from the corruption of the neural data. To allow a comparison to the common average reference (CAR), monopolar noise was used as in Table 5.7. To simulate the realistic power fluctuations in physiological noise and to minimize any bias resulting from the use of a single constant mixture, the noise model with a variable mixture and signal-to-noise ratio (SNR) presented in Table 5.4 was used. This model precludes the use of independent component analysis (ICA) as a viable method, but in real-time BCI operations ICA would typically be difficult to use.

The best available neural data was used as the original signals, which consisted of 3 consecutive sessions of data that contained 176 targets and over 16,000 timepoints. To measure the effect of the corruption the original signals were first decoded, and then the noise was added and the decoding performed once again. The noise was then filtered back out with the ACAR before decoding once again to measure the ability of the ACAR to recover the original neural information buried under the added noise. For comparison, a standard CAR was separately applied to the same corrupted

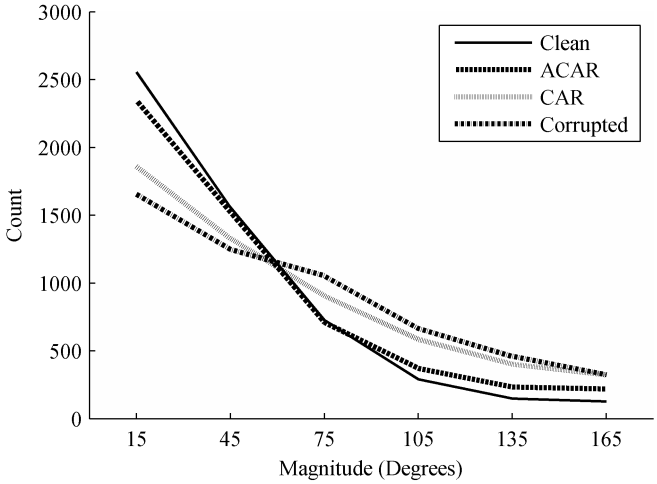

Figure 6.5: Distribution of TAE magnitude with simulated broadband noise. The errors were binned into 30 degree wide bins with the first bin center at 15 degrees. The values shown are the count in each bin.

signals before decoding a final time. In all cases the signals were first passed through a 1 Hz high-pass filter and a standard 4 Hz notch filter at each line noise harmonic.

Evaluation

The offline decoding of 2D cursor control data was first evaluated by measuring the angle error for each timepoint between the direction to the desired target (the target vector) and the direction of movement determined by the decoder (the movement vector). This metric will be referred to as the timepoint angle error (TAE). The magnitude of this error can range from 0 to 180 degrees. For each signal condition, the distribution of this error is shown in Fig. 6.5. The distribution was calculated by placing all errors in 30 degree wide bins, with the first bin centered at 15 degrees. As expected, the performance of the decoder on the original signals was the best. The distribution for the noisy signals was much flatter with more timepoints having larger errors. The CAR improved the results of the noisy signals and the ACAR showed an even further improvement. The results from the ACAR more closely match the performance of the original signals, indicating that it was best able to recover the underlying neural information.

The angle error is a valuable metric in examining the performance of 2D cursor control, but it

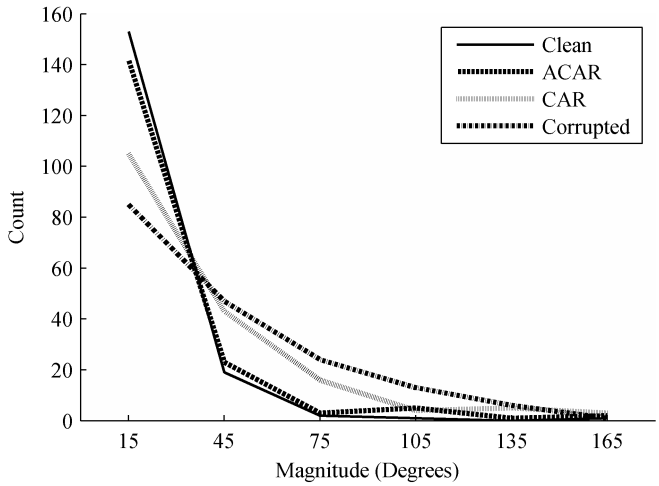

Figure 6.6: Distribution of CAE magnitude with simulated broadband noise. The errors were binned into 30 degree wide bins with the first bin center at 15 degrees. The values shown are the count in each bin.

does not necessarily tell the entire story. Moving far away from the target on one timepoint and slightly towards it on another is clearly not as good as doing the opposite, even though the average angle error might be the same. For this reason a metric was conceived that could account for the magnitude of movement as well as the angle. For each timepoint, the target vector was rotated to 0 degrees. The movement vector was then rotated by the same amount. The rotated movement vectors for each timepoint were then added to produce one movement vector for all timepoints relative to the same target vector. The angle of this summed vector was measured to provide an angle error that took into account the magnitude of movement for each timepoint. This procedure was followed for each target presentation in the data, resulting in one error measurement for each target. This measure will be referred to as the cumulative angle error (CAE).

The distributions of the CAE magnitude are shown in Fig. 6.6. As with the TAE, the results for the original signals are the best, followed closely by the signals filtered by the ACAR. By factoring in the magnitude of movement, the drop-off from the ACAR to the CAR appears to have grown. The distributions for the clean signals and for the ACAR signals approach 0 much more quickly, while the distributions for the CAR and for the corrupted signals decrease more gradually. For additional reference, the average magnitude of both the TAE and the CAE are given in Table 6.1.

In addition to the measures of error in the decoding results, it can be useful to look at the

Table 6.1: BCI decoding mean error magnitudes for real data with simulated broadband noise.

	Clean	ACAR	CAR	Corrupted
TAE	42	48	59	63
CAE	17	22	33	41

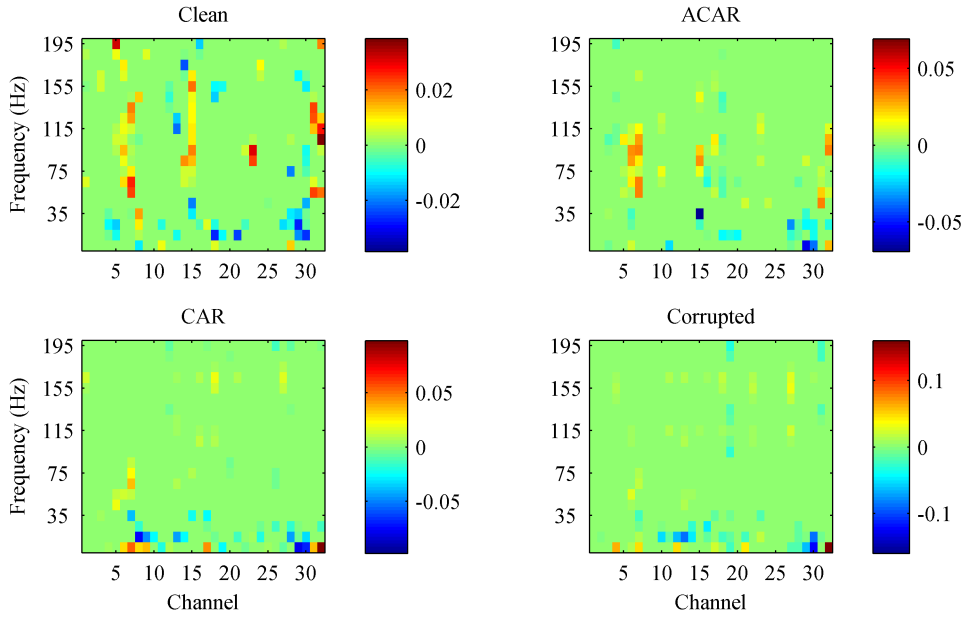

Figure 6.7: Elastic net decoder weights for vertical movement in 2D cursor control with real data and simulated broadband noise.

decoding weights for each feature as was done during the validation of the elastic net decoder. For 2D cursor control a separate set of weights are generated for each dimension. Fig. 6.7 shows these weights for the vertical movement dimension. The interesting thing to note is that what was shown to be an improvement in the decoder performance corresponds to an increase in the magnitude of the decoder weights at higher frequency features. For the clean signals the decoder heavily weighted features all the way up to 200 Hz. For the ACAR strong features were found mostly below about 120 Hz while the prominent CAR weights were mostly below 70 Hz and the decoder for the corrupted data was forced to choose a large portion of its features from below 30 Hz.

This result can be explained by the $1/f$ falloff in the power spectrum for ECoG signals. The higher the frequency, the more sensitive a feature is to noise. So as the power of corrupting white noise increases, the maximum frequency for which useful neural information can be recovered

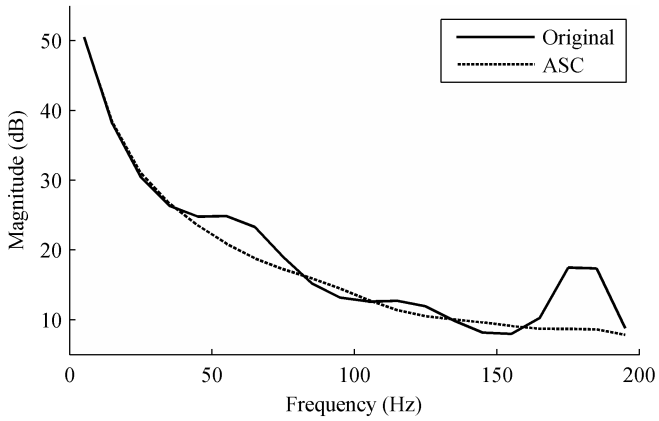

Figure 6.8: Spectral estimate with light line noise contamination.

decreases. This is not to say that the noise does not affect lower frequencies as well, but there is a higher SNR at the lower frequencies. Retaining useful features at higher frequencies is important for BCI decoding, though, as these features typically reflect more localized neural activity. The low frequency information is more widely distributed and thus might not be capable of specifically targeting a BCI task. The increased decoding performance offered by the ACAR over the CAR and the corrupted signals might then be attributable to the availability of higher frequency features.

6.4.4 Line Noise

Methods

As with most physiological recordings, line noise was present in the data from Subject C. The amount of contamination varied between sessions, though, as shown in Fig. 6.8 and Fig. 6.9. Light line noise contamination is shown in Fig. 6.8, which is from the same session of data used in Section 6.4.3. This data will be referred to as Session 1. The heavier contamination in Fig. 6.9 comes from a separate session of data that had inferior real-time decoding results and will be referred to as Session 2.

The figures contain the AR spectral estimates, binned every 10 Hz, that are commonly used in BCI decoding as has been done in this chapter. Compared to the fast Fourier transform (FFT) the AR model has a low error variance and an increased resolution with short time records, but it is still

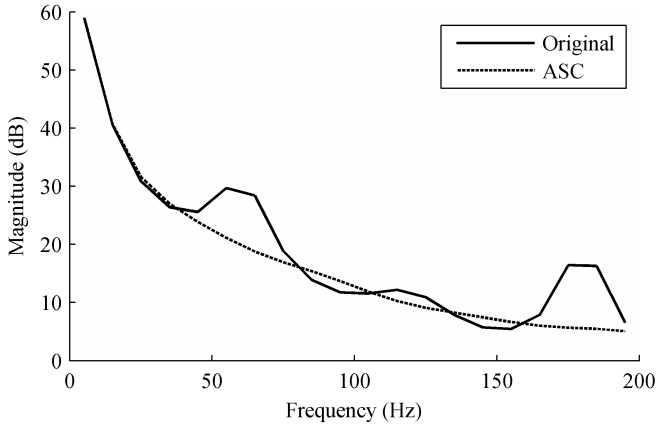

Figure 6.9: Spectral estimate with heavy line noise contamination.

susceptible to spectral leakage and some anomalous effects can occur when the data is dominated by sinusoidal components [27]. As shown by both figures, the contamination from the line noise at its fundamental frequency and harmonics caused a ripple effect in the spectral estimate that altered everything over 40 Hz.

In Section 6.4.3 it was suggested that the higher frequency neural features play a large role in the effectiveness of BCI decoding. Given the apparent effect of line noise on these upper frequencies, it might be expected that the noise would greatly impact decoding results. Many BCI studies, though, do not have access to the higher frequency content that ECoG does. Electroencephalography (EEG), which is similar in frequency content to magnetoencephalography (MEG), typically only uses spectral content up to 70 Hz since the SNR at higher frequencies makes the neural information difficult to use [91]. In the presence of line noise at 60 Hz a low-pass filter at 50 Hz is often used, discarding a potentially valuable and relatively large portion of the already limited neural data.

To evaluate the impact of line noise filtering on decoding results data was decoded in an offline environment without line noise removal, and then after filtering by standard notch filters and by the ASC. It was known that the line noise would not make any large, sudden changes in frequency so for quicker convergence the ASC was adjusted to a maximum bandwidth of 2 Hz and for Session 1 a bandwidth sensitivity of 10. In Chapter 3 these values were 4 Hz and 20, respectively. The same methods used in Section 6.4.3 were used for decoding, with a high-pass filter at 1 Hz, 10

Table 6.2: Average CAE magnitude for Session 1 data with line noise filtering

Features	No Filter	Notch Filter	ASC
All	16	17	16
< 70 Hz	25	26	22
< 50 Hz	26	26	26
$50 - 70$ Hz	40	38	33

Hz frequency bins for features, and the elastic net decoding algorithm. The impact of the filtering when the frequency content of the neural data had a more limited range, as might be seen in EEG or MEG, was also examined.

Evaluation

The CAE presented in the previous section was used to measure the offline decoding results for line noise removal. These results are shown in Table 6.2 for Session 1 and Table 6.3 for Session 2. Only features from those frequencies indicated were used. As would be done in EEG and MEG studies that only use content below 50 Hz, a low-pass filter with a cutoff at 50 Hz was used in that situation to ensure the removal of any line noise influence.

Although the differences in results from the different filtering methods are not as large as in Section 6.4.3, there are a few interesting things to note. First, in both sessions it was found that eliminating data never increased performance, even if no line noise filtering at all was implemented. This would indicate that even if data is contaminated it is better to let the decoder try to make use of it than to completely discard it. In the presence of a good decoding algorithm with automatic feature selection, such as elastic net, bad features are better than no features. Completely useless or harmful features are ignored.

Second, the impact of line noise was more harmful on Session 2. Given the stronger performance of Session 1 in general, though, it is difficult to tell how much of this impact was due to the stronger line noise contamination and how much was due to Session 1 merely having strong, redundant features that eliminated the need for those features contaminated by line noise. In Session 1 the effect of the standard notch filter was negligible at best, which again was most likely due to a

Table 6.3: Average CAE magnitude for Session 2 data with line noise filtering

Features	No Filter	Notch Filter	ASC
All	56	54	52
< 70 Hz	69	60	58
< 50 Hz	76	76	76
$50 - 70$ Hz	76	66	62

combination of the weak line noise and the strong neural features that were eliminated along with the line noise. In Session 2 forgoing a line noise filter had a more detrimental effect, especially when features above 70 Hz were not available.

Last, on this limited sample size the ASC did show improved performance over the standard notch filter. This improvement was most pronounced when only features between 50 Hz and 70 Hz were used. Session 1 also showed the advantage of the ASC in situations with weak line noise contamination and/or strong neural features. The ASC was designed to minimize signal distortion while removing sinusoidal noise, and the effect of preserving the neural data can be seen. While the standard notch filter had trouble outperforming the unfiltered signals in this situation, the ASC produced the best results when features above 70 Hz were unavailable. In Session 2 the advantage of ASC over the standard notch filter was smaller, which is most likely due to a combination of the stronger line noise and the weaker neural features. In such a situation the tradeoff between removing noise and preserving the signal is pushed more towards the side of noise removal, which is what the standard notch filter places the most value on.

So while without further data the overall improvement the ASC offers over the standard notch filter is marginal, it did appear to do a better job of removing the line noise while preserving any neighboring neural features. The resulting impact on decoding performance most likely depends on how harmful the line noise is and on how valuable those neural features are, but with this data it appeared that having extra neural features was negligible at worst. The situation might occur then, where better neural features are lost and the robustness and redundancy offered by those extra features would at that point become more valuable.

It is also important to remember that decoding accuracy depends on the relative changes in the

spectral features during BCI-related tasks, not on the absolute magnitude of the spectral features. As long as the modulation induced by these tasks is enough to cause significant changes in the spectral estimate, decoding results might not suffer significantly from constant noise. Unlike most broadband contamination, line noise power does not greatly fluctuate in short time windows. As shown by Fig. 6.8 and Fig. 6.9, though, it can vary significantly over longer time periods or due to changing environmental conditions. For long-term BCI use then, in which decoder training is infrequent, the effective removal of line noise could play a greater role.

6.4.5 Ocular Artifacts

Good BCI decoding data contaminated with ocular artifacts was not available, and as indicated in Chapter 4 ocular artifacts (OAs) are difficult to accurately simulate. It is also difficult to accurately assess the impact of OA removal from decoding results since the artifacts themselves are often correlated with the BCI's task. Effective removal of the artifacts then would not necessarily improve decoding results. In most BCI studies that are susceptible to OA, trials that are contaminated by OA are simply discarded. This practice results in a loss of valuable data, and in a real-time BCI would not be practical. For example, in wrist movement data presented in Chapter 4 anywhere from 5% to 20% of trials in each session were contaminated by OA and originally discarded. The multi-level wavelet method was then on average able to increase the amount of available data by about 13%.

6.5 Conclusions

This chapter first demonstrated the feasibility of sparse linear regression using the elastic net penalty for BCIs. The decoding accuracy of this method was better than ridge regression and OLS, but not much of an improvement over lasso. The feature set the elastic net chose appeared to retain more correlated features than lasso, though, resulting in a more stable set of feature weights across training sessions.

Some level of sparsity is typically desired in any decoding problem with a high-dimensional feature set in order to eliminate noisy and irrelevant features, but the proper level of sparsity in a BCI remains an open problem. Having fewer features may allow the BCI user to more easily adapt to the decoding weights. Eliminating features that are only moderately useful could allow those features to be used to control an additional degree of freedom. As discussed here, though, a feature space that is too sparse could result in loss of robustness and stability. A further advantage of the elastic net penalty is that the level of sparsity can be scaled all the way from lasso to the ridge regression solution.

The elastic net decoder was also used as an application example to show the effect that some of the methods presented in previous chapters could have on a real neural signal processing task. The results from these examples showed that the methods have potential to positively impact BCI decoding. Craniux allowed a real-time BCI experiment to operate smoothly and achieve effective control of 3D cursor movement, something that had not been achieved before with ECoG in a subject with tetraplegia. In offline analysis, the filtering methods in many cases improved the decoding results in the presence of noise. In the very least they made more features or trials available for use by the decoder, which improves robustness and reliability.

The amount of available data was limited, so it should be remembered that the presented analysis was meant to offer an example from a real application of neural signal processing. This analysis is not sufficient for conclusive evidence, but does provide solid initial results to show the impact some of the methods in this dissertation could have on a BCI. To fully measure their effectiveness, though, further studies should be done with additional real data. Real-time application of the methods also needs to be examined, so that the BCI user's adaptation is also a factor. It could be possible that results would improve even further in that situation, as the BCI user might be able to learn to make use of the improved, filtered data.

Chapter 7

Conclusion

7.1 Overview

The outcome of this research was an increase in the effectiveness and accessibility of some common neural signal processing tools. It is hoped that through these tools clinicians and researchers will be able to better focus on the neural analysis and scientific questions that drive their work. The same tools should also help many basic tasks in neural signal processing, such as operation of a brain-computer interface (BCI), to become possible for non-expert users. This result would increase the target population for advanced technologies that are based on neural signal processing, and also further ease the burden on researchers and clinicians by allowing more tasks to be delegated to non-expert personnel.

The software presented in Chapter 2, Craniux, proved to be an excellent system framework for neural signal processing. It reliably maintained data integrity and performed well with consistent system timing. The user interface, as well as the automation of many system components, make it easy to use the system for real-time control and visualization of experiments. Its highly modular design, as well as the use of a high-level language with powerful built-in libraries, simplifies the process of extending the software by adding new components or algorithms to the system. The combination of these features gives users with a wide range of skill sets the flexibility to perform both real-time experiments and thorough offline analysis.

Results also showed that the novel filtering algorithms have the potential to effectively perform and automate some of the common steps that are necessary in neural signal processing systems. In Chapter 3 the adaptive sinusoid canceler (ASC) was presented as a filter that could track and isolate sinusoidal components in a signal. It does so by using an adaptive noise canceling (ANC) filter setup and internally tracking the line noise frequency in order to generate its own reference. This filter was applied to the prevalent problem of line noise in neural recordings, and results showed the method to be superior to other methods commonly used for this problem such as standard notch filters and an adaptive line enhancer (ALE). The ASC is able to ensure removal of the sinusoidal component, while minimizing the distortion to the underlying signal.

Chapter 4 discussed the multi-level wavelet technique for removal of ocular artifacts (OAs) in

neural data. This method relies on localizing the artifacts in both time and frequency before removing them with a threshold function. The localization is done by identifying an artifact, determining the optimal level of wavelet decomposition, and then locating its temporal bounds. After removing the artifact in the wavelet coefficients, the remaining signal can be reconstructed. This technique was found to be more effective than traditional techniques such as regression, principal component analysis (PCA), and independent component analysis (ICA), and it is also fully automated and computationally efficient on large data sets.

The adaptive common average reference (ACAR), a method for removing common mode artifacts, was presented in Chapter 5. Like the ASC, this algorithm uses an ANC filter and internally generates its own reference. The reference is first produced by a common average reference (CAR), after which the output of the ANC filter is used as feedback to enhance the reference on the next timepoint. Results showed the ACAR to be effective at removing spatially correlated noise, and under most circumstances was superior to a CAR and to ICA.

Finally, Chapter 6 presented BCI decoding, an example application for the methods developed in this dissertation. Craniux was found to be an effective framework for real-time BCI decoding. The data the software streamed to disk also allowed offline analysis to be performed using linear regression with elastic net regularization. This decoder was shown to provide automatic feature selection and robust results. The novel filtering algorithms were demonstrated to increase the amount of usable data that was available for the decoding process by improving the quality of contaminated neural signals. Given the typically limited amount and quality of neural data available for training BCIs, any extra data is always desired. Whether an increase in data comes in the form of additional trials or additional features, it is beneficial to the accuracy and robustness of the decoder.

7.2 Directions for Future Work

The methods in this dissertation should provide a stepping stone to further research in the expanding field of neural signal processing. This final section will discuss ways in which the presented

algorithms and ideas can be further improved or studied. There are numerous needs for the future development of advanced tools that assist with the processing of neural data, but here the discussion will be limited to topics directly related to the tools created for this dissertation. These topics include improvements that could be made, as well as additional analysis that could be done to further validate or gain new insight into the methods.

7.2.1 Craniux Development

Multi-tasking. Craniux currently provides an excellent means for conducting a single experiment or task, but it could at times be useful to run multiple real-time tasks at once. For example, in a BCI it might be useful to simultaneously look at the output produced by different decoders or feature sets. To accomplish this, Craniux would need to be capable of branching between modules rather than relying on the single input and output from each module. The main difficulty would lie in still ensuring that determinism was maintained between modules.

Executables Craniux is currently freely available as open-source software and runs in the LabVIEW environment. It would be accessible to an even larger number of people, though, if it were compiled down to executables. These executables could clearly not be modified as the source code can, but would allow its built-in functionality to be used without access to LabVIEW. One roadblock to creating these executables is that some current aspects of communication between system components rely on LabVIEW features that cannot be compiled. Removing the need for these features would most likely require the creation of a stand alone communications manager or server that could help initialize the communication channels between system components.

7.2.2 ASC Improvements

Frequency identification. The lowest effectiveness of the ASC is at high signal-to-noise ratios (SNRs), where it is at times unable to initially locate and then track the frequency of the noise. This problem can create a situation in which, instead of the frequency estimate being improved by

a narrowing bandwidth, the bandwidth broadens to search for the frequency and creates an even higher SNR within the bandpassed portion of the signal. The need for the filter is minimized at high SNRs, but a more advanced method for determining the noise frequency could help alleviate this problem. It was shown that the current method outperformed spectral peak detection at an SNR of 0 dB, but at higher SNRs some form of spectral peak detection would most likely be less susceptible to interference from the surrounding signal.

Noise detection. The ASC might also be improved by including or calculating an estimate of the SNR. The filter could then adjust its bandwidth sensitivity to close more tightly and prevent signal distortion in the high SNR case and open more widely to ensure noise attenuation in the low SNR case. This adjustment would also help optimize the frequency estimate. At the least, a mechanism could be implemented in which the filter is automatically bypassed if the SNR is too high.

7.2.3 OA Removal

Wavelet function. The selection of the Haar wavelet for OA removal was made mostly for computational efficiency and due to the fact that the differing types of OA would make it difficult to select a single optimal wavelet basis function. A more thorough investigation into the effect of different wavelets could either confirm the Haar as the most practical choice, or determine a better basis function that could be used to further improve the method. If a real-time implementation is needed, then computation time would have to be considered in addition to effectiveness.

Thresholding method. The performance of the multi-level wavelet method lies in its ability to localize the artifact in both time and frequency. Once localized, though, the artifact is still removed with a simple threshold. A more advanced algorithm that could take into consideration the shape of the OA might better preserve the neural data present at that time/frequency. This information might be obtained by locating the same time/frequency window on the electrooculographic (EOG) reference channel, but this would also eliminate the advantage of the removal method not requiring

the reference. Since a multi-channel recording would probably be in use and most OAs qualify as common mode, it might be possible to generate a reference using a method similar to the ACAR. As with any engineering question, though, it must be considered if the resulting improvement in that case would be worth the increased system complexity.

7.2.4 ACAR Considerations

Channel selection. One of the benefits of the ACAR over the CAR is its ability to weight channels when creating a reference based on the strength and polarity of the common mode artifact in each channel. If a channel contains a small amount of the artifact then it receives a small weight. The weight is not zero, though, and if the channel has a much larger amplitude than other channels then it can adversely affect the reference. An example of this is an open channel that is just acting as an antenna. It would be useful if such channels could be completely ignored (i.e. set their weights to zero) when generating the reference. Simply normalizing the weight by the channel's power would help, but this would be a major detriment when a channel has high power due to heavy contamination by the artifact. A combination of looking at the channel's power and its potential weight could probably arrive at a solution that ignored these problematic channels. Also, it would be useful if the channel selection could extend to not filtering any channels in the case where no common mode noise is detected.

Multiple common mode sources. The ACAR converges to a reference and spatial map for the most dominant source in the recording. This property limits the ACAR to only removing noise over one spatial distribution. Multiple noise sources could be removed, but only if they had similar spatial distributions or if they were temporally isolated so that the filter could adapt between them. As long as the noise sources had varying powers so that an ACAR could converge to the strongest remaining source, then multiple ACARs could be cascaded to remove all the noise. This might not be possible, though. If not then the only other current option would be to present each noise source individually, allow an ACAR to adapt, and then lock its values into place. It could be beneficial to

develop a method that could allow the ACAR to isolate multiple spatially distinct sources of noise.

Source localization and connectivity analysis. Many source localization and connectivity analysis techniques rely on the correlation between signals to find the dominant sources (although some actually look for the differences between channels [92]). It is important then that any filtering performed before source localization does not affect the signal correlation structure. The effect that the ACAR has on this structure could vary greatly depending on factors such as SNR, the spatial layout of the noise and sources of interest, and the number of recorded channels. The same property mentioned above that currently only allows the ACAR to converge to one noise source should also prevent it from damaging the correlation of signal sources, though. In the case where a source of interest is dominant, or where it has a spatial distribution similar to the dominant source, it would likely be corrupted by the ACAR. A full understanding of this interaction between the ACAR and source localization requires further investigation and would make an interesting study.

7.2.5 Additional Analysis

Real-time BCI decoding. All of the results in this dissertation that examine the impact of the filtering methods on BCI decoding use offline data. To truly understand the effect of these filters they must be implemented in real-time decoding. BCI adaptation is a two-way process: as the decoder learns the neural features the BCI user is also learning the decoder. Improved filtering can make more features usable by the decoder, but if the filtering is done in an offline environment then the user is never able to adapt to the availability or improvement of those features. In a real-time environment, the subject might be able to make better use of the features to further improve the results. It would likely take a large amount of data to conclusively determine the impact of the filters, though, due to the learning that occurs as the subject uses the BCI.

Other applications. BCI decoding was presented as an example application for the novel neural signal processing methods that were developed. The impact of these methods is not meant to be

limited to BCIs, though, and it would be desired to examine their effect on other applications. Neuroscience studies, medical research, and clinical monitoring are just a few other areas where some of this work might be applicable. Some of the filters might even be useful in completely unrelated areas in which similar noise is a problem. The end goal of this research was to create effective and accessible tools that could assist with neural signal processing, but that should not be considered a limit to its applications.

Bibliography

[1] W. Wang, J. Collinger, M. Perez, E. Tyler-Kabara, L. Cohen, N. Birbaumer, S. Brose, A. Schwartz, M. Boninger, and D. J. Weber, "Neural interface technology for rehabilitation: exploiting and promoting neuroplasticity," *Phys. Medi. and Rehab. Clinics of N. Amer.*, vol. 21, no. 1, pp. 157–178, 2010.

[2] W. Wang, J. Collinger, A. Degenhart, E. Tyler-Kabara, A. Schwartz, D. Moran, D. Weber, B. Wodlinger, R. Vinjamuri, R. Ashmore, J. Kelly, and M. Boninger, "An electrocorticographic brain interface in an individual with tetraplegia," *PloS One*, vol. 8, no. 2, p. e55344, Feb. 2013.

[3] J. L. Collinger, B. Wodlinger, J. E. Downey, W. Wang, E. C. Tyler-Kabara, D. J. Weber, A. J. McMorland, M. Velliste, M. L. Boninger, and A. B. Schwartz, "High-performance neuroprosthetic control by an individual with tetraplegia," *Lancet*, vol. 6736, no. 12, pp. 1–8, Dec. 2012.

[4] T. Vaughan and W. Heetderks, "Brain-computer interface technology: a review of the Second International Meeting." *IEEE T. on Neural Sys. and Rehab. Eng.*, vol. 11, no. 2, pp. 94–109, Jun. 2003.

[5] M. De Vos, D. M. Vos, S. Riès, K. Vanderperren, B. Vanrumste, F.-X. Alario, S. Van Huffel, V. S. Huffel, and B. Burle, "Removal of muscle artifacts from EEG recordings of spoken language production," *Neuroinform.*, vol. 8, no. 2, pp. 135–50, Jun. 2010.

[6] J. P. Donoghue, "Bridging the brain to the world: a perspective on neural interface systems," *Neuron*, vol. 60, no. 3, pp. 511–21, Nov. 2008.

[7] M. Fatourechi, A. Bashashati, R. K. Ward, and G. E. Birch, "EMG and EOG artifacts in brain computer interface systems: A survey," *Clin. Neurophys.*, vol. 118, no. 3, pp. 480–94, Mar. 2007.

[8] R. J. Croft and R. J. Barry, "EOG correction: which regression should we use?" *Psychophys.*, vol. 37, no. 1, pp. 123–5, Jan. 2000.

[9] D. Olguin, F. Bouchereau, and S. Martinez, "Adaptive Notch Filter for EEG Signals Based on the LMS Algorithm with Variable Step-Size Parameter," in *Conf. on Inf. Sciences and Sys.*, 2005.

[10] C. J. Ochoa and J. Polich, "P300 and blink instructions," *Clin. Neurophys.*, vol. 111, no. 1, pp. 93–8, Jan. 2000.

[11] R. Verleger, "The instruction to refrain from blinking affects auditory P3 and N1 amplitudes," *Electroenceph. and Clin. Neurophys.*, vol. 78, no. 3, pp. 240–51, Mar. 1991.

[12] G. L. Wallstrom, R. E. Kass, A. Miller, J. F. Cohn, and N. A. Fox, "Automatic correction of ocular artifacts in the EEG: a comparison of regression-based and component-based methods," *Int. J. of Psychophys.*, vol. 53, no. 2, pp. 105–19, Jul. 2004.

[13] I. I. Goncharova, D. J. McFarland, T. M. Vaughan, and J. R. Wolpaw, "EMG contamination of EEG: spectral and topographical characteristics," *Clin. Neurophys.*, vol. 114, no. 9, pp. 1580–1593, 2003.

[14] A. D. Degenhart, J. W. Kelly, R. C. Ashmore, J. L. Collinger, E. C. Tyler-Kabara, D. J. Weber, and W. Wang, "Craniux: a LabVIEW-based modular software framework for brain-machine interface research," *Comp. Intell. and Neuro.*, vol. 2011, Jan. 2011.

[15] W. Miltner, C. Braun, and R. Johnson, "A test of brain electrical source analysis (BESA): a simulation study," *Electroenceph. and Clin. Neurophys.*, 1994.

[16] A. Delorme and S. Makeig, "EEGLAB: an open source toolbox for analysis of single-trial EEG dynamics including independent component analysis," *J. of Neuro. Meth.*, vol. 134, no. 1, pp. 9–21, Mar. 2004.

[17] F. Tadel, S. Baillet, J. C. Mosher, D. Pantazis, and R. M. Leahy, "Brainstorm: a user-friendly application for MEG/EEG analysis," *Comp. Intell. and Neuro.*, vol. 2011, Jan. 2011.

[18] C. Brunner, G. Andreoni, and L. Bianchi, "BCI Software Platforms," in *Towards Practical Brain-Computer Interfaces*. Biological and Medical Physics, 2013, pp. 303–331.

[19] P. Brunner, L. Bianchi, C. Guger, F. Cincotti, and G. Schalk, "Current trends in hardware and software for brain-computer interfaces (BCIs)," *J. of Neural Eng.*, vol. 8, no. 2, p. 025001, Apr. 2011.

[20] M. Velliste, S. Perel, M. C. Spalding, A. S. Whitford, and A. B. Schwartz, "Cortical control of a prosthetic arm for self-feeding." *Nature*, vol. 453, no. 7198, pp. 1098–101, Jun. 2008.

[21] D. M. Taylor, S. I. H. Tillery, and A. B. Schwartz, "Direct cortical control of 3D neuroprosthetic devices," *Science*, vol. 296, no. 5574, pp. 1829–32, Jun. 2002.

[22] G. Schalk, D. J. McFarland, T. Hinterberger, N. Birbaumer, and J. R. Wolpaw, "BCI2000: a general-purpose brain-computer interface (BCI) system," *IEEE T. on Biomed. Eng.*, vol. 51, no. 6, pp. 1034–43, Jun. 2004.

[23] Y. Renard, F. Lotte, G. Gibert, M. Congedo, E. Maby, V. Delannoy, O. Bertrand, and A. Lécuyer, "OpenViBE: an open-source software platform to design, test, and use braincomputer interfaces in real and virtual environments," *Presence: Teleoperators and Virtual Environments*, vol. 19, no. 1, pp. 35–53, Feb. 2010.

[24] C. Breitwieser, I. Daly, C. Neuper, and G. R. Müller-Putz, "Proposing a standardized protocol for raw biosignal transmission," *IEEE T. on Biomed. Eng.*, vol. 59, no. 3, pp. 852–9, Mar. 2012.

[25] R. Oostenveld and P. Fries, "FieldTrip: open source software for advanced analysis of MEG, EEG, and invasive electrophysiological data," *Comp. Intell. and Neuro.*, vol. 2011, Jan. 2011.

[26] S. Clanton, "Brain-computer interface control of an anthropomorphic robotic arm," Ph.D. dissertation, Carnegie Mellon University, 2011.

[27] S. M. Kay and S. L. Marple, "Spectrum analysis - a modern perspective," *Spectrum*, vol. 69, no. 11, 1981.

[28] R. Kalman, "A new approach to linear filtering and prediction problems," *J. of Basic Eng.*, vol. 82, no. Series D, pp. 35–45, 1960.

[29] E. Salinas and L. F. Abbott, "Vector reconstruction from firing rates," *J. of Comp. Neuroscience*, vol. 1, no. 1-2, pp. 89–107, Jun. 1994.

[30] D. Rushton, "Functional electrical stimulation," *Phys. Meas.*, vol. 241, 1999.

[31] C. Moran, "Revolutionizing Prosthetics 2009 Modular Prosthetic Limb - body interface: overview of the prosthetic socket development," *Johns Hopkins APL Technical Digest*, vol. 30, no. 3, pp. 240–249, 2011.

[32] J. Nagle, "Congestion control in IP/TCP internetworks," Ford Aerospace and Communications Corporation, Tech. Rep. January, 1984.

[33] NI, "The NI TDMS File Format," National Instruments, Tech. Rep., 2012.

[34] W. Wang, A. D. Degenhart, J. L. Collinger, R. Vinjamuri, G. P. Sudre, P. D. Adelson, D. L. Holder, E. C. Leuthardt, D. W. Moran, M. L. Boninger, A. B. Schwartz, D. J. Crammond, E. C. Tyler-Kabara, and D. J. Weber, "Human motor cortical activity recorded with micro-eCoG electrodes, during individual finger movements," in *Conf. of the IEEE Eng. in Med. and Bio. Soc.*, vol. 2009, Jan. 2009, pp. 586–9.

[35] P. Jiruska, R. Cmejla, A. D. Powell, W.-C. Chang, M. Vreugdenhil, and J. G. R. Jefferys, "Reference noise method of removing powerline noise from recorded signals," *J. of Neuro. Meth.*, vol. 184, no. 1, pp. 110–4, Oct. 2009.

[36] A. K. Ziarani and A. Konrad, "A nonlinear adaptive method of elimination of power line interference in ECG signals," *IEEE T. on Biomed. Eng.*, vol. 49, no. 6, pp. 540–7, Jun. 2002.

[37] J. W. Kelly, J. L. Collinger, A. D. Degenhart, D. P. Siewiorek, A. Smailagic, and W. Wang, "Frequency Tracking and Variable Bandwidth for Line Noise Filtering without a Reference," in *Conf. of the IEEE Eng. in Med. and Bio. Soc.*, 2011.

[38] N. R. Anderson, T. Blakely, G. Schalk, E. C. Leuthardt, and D. W. Moran, "Electrocorticographic (ECoG) correlates of human arm movements," *Exp. Brain Research*, vol. 223, no. 1, pp. 1–10, Nov. 2012.

[39] T. Blakely, K. J. Miller, R. P. N. Rao, M. D. Holmes, and J. G. Ojemann, "Localization and classification of phonemes using high spatial resolution electrocorticography (ECoG) grids." in *Conf. of the IEEE Eng. in Med. and Bio. Soc.*, vol. 2008, Jan. 2008, pp. 4964–7.

[40] T. V. Baak, "60 Hz AC mains frequency accuracy measurement," *http://www.leapsecond.com/pages/mains/*, 2004.

[41] M. Ta and V. DeBrunner, "Adaptive Notch Filter with time-frequency tracking of continuously changing frequencies," in *IEEE Conf. on Acoustics, Speech and Sig. Proc.*, Mar. 2008, pp. 3557–3560.

[42] D. W. Mortara, "Digital filters for ECG signals," *Comp. in Cardiology*, 1977.

[43] M. M. Z. Zadeh, S. Niketeghad, and R. Amirfattahi, "A PLL based adaptive power line interference filtering from ECG signals," in *CSI Int. Symp. on Art. Intell. and Sig. Proc.*, no. Aisp. Ieee, May 2012, pp. 490–496.

[44] B. Widrow, J. Glover, J. McCool, J. Kaunitz, C. Williams, R. Hearn, J. Zeidler, J. Eugene Dong, and R. Goodlin, "Adaptive noise cancelling: principles and applications," *Proc. of the IEEE*, vol. 63, no. 12, pp. 1692–1716, 1975.

[45] S. Haykin, *Adaptive Filter Theory*, 3rd ed. Upper Saddle River, NJ: Prentice Hall, 1996.

[46] M. Ferdjallah and R. E. Barr, "Adaptive digital notch filter design on the unit circle for the removal of powerline noise from biomedical signals," *IEEE T. on Biomed. Eng.*, vol. 41, no. 6, pp. 529–536, 1994.

[47] I. S. Badreldin, D. S. El-Kholy, and A. A. El-Wakil, "A modified adaptive noise canceler for electrocardiography with no power-line reference," in *Cairo Int. Biomed. Eng. Conf.*, vol. 13, no. 2, 2010, pp. 5–8.

[48] J. S. Lim and A. V. Oppenheim, *Advanced Topics in Signal Processing*. Englewood Cliffs, NJ: Prentice Hall, 1988.

[49] B. Widrow, P. E. Mantey, L. J. Griffiths, and B. B. Goode, "Adaptive antenna systems," *Proc. of the IEEE*, vol. 55, no. 12, pp. 2143–2159, 1967.

[50] J. R. Glover, "Adaptive noise canceling applied to sinusoidal interferences," *IEEE T. on Acoustics, Speech, and Sig. Proc.*, vol. 25, no. 6, pp. 484–491, 1977.

[51] J. Treichler, "Transient and convergent behavior of the adaptive line enhancer," *IEEE T. on Acoustics, Speech, and Sig. Proc.*, no. 1, 1979.

[52] R. Ramli, A. Noor, and S. Samad, "A review of adaptive line enhancers for noise cancellation," *Australian J. of Basic and App. Sci.*, vol. 6, no. 6, pp. 337–352, 2012.

[53] S. Dhull, S. Arya, and O. P. Sahu, "Performance comparison of adaptive algorithms for adaptive line enhancer," *Int. J. of Comp. Sci.*, vol. 8, no. 3, pp. 553–558, 2011.

[54] M. Keshner, "1/f noise," *Proc. of the IEEE*, vol. 70, no. 3, pp. 212–218, 1982.

[55] C. K. Kovach, N. Tsuchiya, H. Kawasaki, H. Oya, M. a. Howard, and R. Adolphs, "Manifestation of ocular-muscle EMG contamination in human intracranial recordings," *NeuroImage*, vol. 54, no. 1, pp. 213–33, Jan. 2011.

[56] J. W. Kelly, D. P. Siewiorek, A. Smailagic, J. L. Collinger, D. J. Weber, and W. Wang, "Fully automated reduction of ocular artifacts in high-dimensional neural data," *IEEE T. on Biomed. Eng.*, vol. 58, no. 3, pp. 598–606, Mar. 2011.

[57] B. Noureddin, P. D. Lawrence, and G. E. Birch, "Quantitative evaluation of ocular artifact removal methods based on real and estimated EOG signals," *Conf. of the IEEE Eng. in Med. and Bio. Soc.*, vol. 2008, pp. 5041–4, Jan. 2008.

[58] D. J. McFarland, L. M. McCane, S. V. David, and J. R. Wolpaw, "Spatial filter selection for EEG-based communication," *Electroenceph. and Clin. Neurophys.*, vol. 103, no. 3, pp. 386–94, Sep. 1997.

[59] T. P. Jung, S. Makeig, C. Humphries, T. W. Lee, M. J. McKeown, V. Iragui, and T. J. Sejnowski, "Removing electroencephalographic artifacts by blind source separation." *Psychophysiology*, vol. 37, no. 2, pp. 163–78, Mar. 2000.

[60] T. Lagerlund, F. Sharbrough, and N. Busacker, "Spatial filtering of multichannel electroencephalographic recordings through principal component analysis by singular value decomposition," *Clinical Neurophysiology*, vol. 14, no. 1, p. 73, 1997.

[61] S. Makeig, "Frequently asked questions about ICA applied to EEG and MEG data," *http://sccn.ucsd.edu/eeglab sccn.ucsd.edu/˜scott/icafaq.html*, 2013.

[62] T. P. Jung, S. Makeig, M. Westerfield, J. Townsend, E. Courchesne, and T. J. Sejnowski, "Removal of eye activity artifacts from visual event-related potentials in normal and clinical subjects," *Clin. Neurophys.*, vol. 111, no. 10, pp. 1745–58, Oct. 2000.

[63] S. Vorobyov and A. Cichocki, "Blind noise reduction for multisensory signals using ICA and subspace filtering, with application to EEG analysis," *Bio. Cybernetics*, vol. 86, no. 4, pp. 293–303, Apr. 2002.

[64] N. Mammone and F. C. Morabito, "Enhanced automatic artifact detection based on independent component analysis and Renyi's entropy," *Neural Networks*, vol. 21, no. 7, pp. 1029–40, Sep. 2008.

[65] V. Krishnaveni, S. Jayaraman, L. Anitha, and K. Ramadoss, "Removal of ocular artifacts from EEG using adaptive thresholding of wavelet coefficients," *J. of Neural Eng.*, vol. 3, no. 4, pp. 338–46, Dec. 2006.

[66] T. Zikov, S. Bibian, G. A. Dumont, M. Huzmezan, and C. R. Ries, "A wavelet based de-noising technique for ocular artifact correction of the electroencephalogram," in *Proc. of the 2nd joint IEEE EMBS/BMES Conf.*, 2002, pp. 98–105.

[67] G. Ouyang, X. Li, Y. Li, and X. Guan, "Application of wavelet-based similarity analysis to epileptic seizures prediction," *Comp. in Bio. and Med.*, vol. 37, no. 4, pp. 430–7, Apr. 2007.

[68] X. Li, X. Yao, J. Fox, and J. G. Jefferys, "Interaction dynamics of neuronal oscillations analysed using wavelet transforms," *J. of Neuro. Meth.*, vol. 160, no. 1, pp. 178–85, Feb. 2007.

[69] L. Vigon, M. Saatchi, J. Mayhew, and R. Fernandes, "Quantitative evaluation of techniques for ocular artefact filtering of EEG waveforms," *IEE Proceedings - Science, Meas. and Tech.*, vol. 147, no. 5, p. 219, 2000.

[70] A. Hyvärinen, "Fast and robust fixed-point algorithms for independent component analysis," *IEEE T. on Neural Networks*, vol. 10, no. 3, pp. 626–34, Jan. 1999.

[71] H. Cui and G. Song, "Study of the wavelet basis selections," in *Int. Conf. on Comp. Intell. and Sec.*, no. x, 2006, pp. 1833–1836.

[72] T. Elbert, W. Lutzenberger, B. Rockstroh, and N. Birbaumer, "Removal of ocular artifacts from the EEG - a biophysical approach to the EOG," *Electroenceph. and Clin. Neurophys.*, vol. 60, pp. 455–463, 1985.

[73] F. Offner, "The EEG as potential mapping: the value of the average monopolar reference," *Electroenceph. and Clin. Neurophys.*, 1950.

[74] J. Dien, "Issues in the application of the average reference: review, critiques, and recommendations," *Behavior Research Meth., Instr., & Comp.*, vol. 30, no. 1, pp. 34–43, Mar. 1998.

[75] A. Bashashati, M. Fatourechi, R. K. Ward, and G. E. Birch, "A survey of signal processing algorithms in brain-computer interfaces based on electrical brain signals," *J. of Neural Eng.*, vol. 4, no. 2, pp. R32–57, Jun. 2007.

[76] V. Zarzoso and P. Comon, "Robust independent component analysis by iterative maximization of the kurtosis contrast with algebraic optimal step size," *IEEE T. on Neural Networks*, vol. 21, no. 2, pp. 248–261, 2010.

[77] A. Belouchrani, K. Abed-Meraim, J.-F. Cardoso, and E. Moulines, "A blind source separation technique using second-order statistics," *IEEE T. on Sig. Proc.*, vol. 45, no. 2, pp. 434–444, 1997.

[78] S. T. Foldes and D. M. Taylor, "Offline comparison of spatial filters for two-dimensional movement control with noninvasive field potentials," *J. of Neural Eng.*, vol. 8, no. 4, p. 046022, Aug. 2011.

[79] J. W. Kelly, A. D. Degenhart, D. P. Siewiorek, A. Smailagic, and W. Wang, "Sparse linear regression with elastic net regularization for brain-computer interfaces," in *Conf. of the IEEE Eng. in Med. and Bio. Soc.*, vol. 2012, Aug. 2012, pp. 4275–8.

[80] J. R. Wolpaw, N. Birbaumer, D. J. McFarland, G. Pfurtscheller, and T. M. Vaughan, "Brain-computer interfaces for communication and control," *Clin. Neurophys.*, vol. 113, no. 6, pp. 767–91, Jun. 2002.

[81] F. Lotte, M. Congedo, A. Lécuyer, F. Lamarche, and B. Arnaldi, "A review of classification algorithms for EEG-based brain-computer interfaces," *J. of Neural Eng.*, vol. 4, no. 2, pp. R1–R13, Jun. 2007.

[82] W. Wu and N. G. Hatsopoulos, "Real-time decoding of nonstationary neural activity in motor cortex," *IEEE T. on Neural Sys. and Rehab. Eng.*, vol. 16, no. 3, pp. 213–22, Jun. 2008.

[83] A. Georgopoulos, A. Schwartz, and R. Kettner, "Neuronal population coding of movement direction," *Science*, vol. 233, no. 4771, pp. 1416–1419, Sep. 1986.

[84] A. G. Rouse and D. W. Moran, "Neural adaptation of epidural electrocorticographic (EECoG) signals during closed-loop brain computer interface (BCI) tasks." in *Conf. of the IEEE Eng. in Med. and Bio. Soc.*, vol. 2009, Jan. 2009, pp. 5514–7.

[85] T. Hesterberg, N. H. Choi, L. Meier, and C. Fraley, "Least angle and L1 penalized regression: a review," *Stat. Surveys*, vol. 2, pp. 61–93, 2008.

[86] R. Tibshirani, "Regression shrinkage and selection via the lasso," *J. of the Royal Stat. Society*, pp. 267–288, 1996.

[87] M. K. Carroll, G. a. Cecchi, I. Rish, R. Garg, and a. R. Rao, "Prediction and interpretation of distributed neural activity with sparse models," *NeuroImage*, vol. 44, no. 1, pp. 112–22, Jan. 2009.

[88] S. Ryali, K. Supekar, D. a. Abrams, and V. Menon, "Sparse logistic regression for whole-brain classification of fMRI data," *NeuroImage*, vol. 51, no. 2, pp. 752–64, Jun. 2010.

[89] J. Friedman, T. Hastie, and R. Tibshirani, "Regularization Paths for Generalized Linear Models via Coordinate Descent." *J. of Stat. Software*, vol. 33, no. 1, pp. 1–22, Jan. 2010.

[90] N. Simon, J. Friedman, T. Hastie, and R. Tibshirani, "Regularization paths for Cox's proportional hazards model via coordinate descent," *J. of Stat. Software*, vol. 39, no. 5, 2011.

[91] E. Niedermeyer and F. da Silva, *Electroencephalography: basic principles, clinical applications, and related fields*, 5th ed. Philadelphia, PA: Lippincott Williams & Wilkins, 2012.

[92] J. Zhang, G. Sudre, X. Li, W. Wang, D. J. Weber, and A. Bagic, "Task-related MEG source localization via discriminant analysis," *Conf. of the IEEE Eng. in Med. and Bio. Soc.*, vol. 2011, pp. 2351–4, Jan. 2011.