

**NeuroHub: Portable and Scalable Time Synchronization Instrument for Brain-
Computer Interface and Functional Neuroimaging Research**

A Thesis

Submitted to the Faculty

of

Drexel University

by

Nicholas V. Grzeczowski

in partial fulfillment of the

requirements for the degree

of

Master of Science in Biomedical Engineering

December 2014



© Copyright 2014

Nicholas V. Grzeczowski. All Rights Reserved.

Acknowledgments

I would like to acknowledge my parents, for their support. I would also like to thank everyone in the CONQUER Collaborative lab that assisted me, especially Yichuan Liu and Adrian Curtin, as well as those that provided assistance in other Drexel University labs, especially Dan Luig in the Electrical and Computer Engineering Department.

Most of all, I would like to thank my thesis advisor, Dr. Hasan Ayaz, for consistently challenging, supporting, and inspiring me, as well as easing my tensed nerves through clarification in difficult moments.

Table of Contents

List of Tables	vi
List of Figures	vii
Abstract	ix
Chapter 1: Introduction	1
1.1 Motivation	1
1.2 Specific Aim	2
1.3 Approach	2
Chapter 2: Background	4
2.1 Brain-Computer Interfaces	4
2.1.1 Locked-In Syndrome	5
2.1.2 Non-invasive BCIs	6
2.2 Electroencephalography in BCIs	7
2.2.1 EEG Principles	7
2.2.2 Sensors and Instrumentation	9
2.2.3 Examples of EEG based BCIs	10
2.3 Functional Near-Infrared Spectroscopy in BCIs	14
2.3.1 fNIR Principles	14
2.3.2 Sensors and Instrumentation	16
2.3.3 Signal Processing and Analysis	17
2.3.4 Examples of fNIR based BCIs	19
2.4 Hybrid BCIs	21
2.4.1 Unimodal Hybrid BCIs	22
2.4.2 Multimodal Hybrid BCIs	25
2.5 Need for a Solution	28
2.6 Current Approaches	28
Chapter 3: Device Design and Development	32
3.1 Device Design	32
3.1.1 System Requirements	32
3.1.2 System Specifications	34
3.2 Device Development and Implementation	38

3.2.1 Development Platform and Microcontroller Selection	40
3.2.2 Generation 1 Development	43
3.2.3 Generation 2 Development	56
Chapter 4: Testing and Validation	63
4.1 Reliability and Variability Testing Setup.....	63
4.2 Program Design and Development	64
4.3 Data Analysis	69
4.4 Results.....	71
4.4.1 Oscilloscope Measured Lag Time.....	72
4.4.2 Serial to Serial and TTL to Serial	75
4.4.3 USB and Native Serial With and Without Additional Information Influx	81
4.4.4 Different Operating Systems on the Same Hardware	92
4.5 Discussion	97
Chapter 5: Use Case Demonstrations.....	99
5.1 Use Case #1 – Multimodal Spatial Navigation BCI	100
5.1.1 Introduction.....	100
5.1.2 Background.....	100
5.1.3 Materials and Methods.....	101
5.1.4 Results and Conclusion.....	106
5.2 Use Case #2 – Synthetic Speech Perception BCI	109
5.2.1 Introduction.....	109
5.2.2 Background.....	110
5.2.3 Materials and Methods.....	111
5.2.4 Results and Conclusion.....	114
Chapter 6: Future Work and Conclusion	117
6.1 Future Work	117
6.2 Conclusion	120
List of References	122
Appendix A: Board Assembly Guide	137
Appendix B: Schematics and Board Layouts	145
Generation 1	145

Generation 2.....	148
Generation 3.....	150
Appendix C: Source Code	153
Microcontroller Code.....	153
Time Testing Code.....	159
Time Testing GUI.....	165
TTL Time Testing Code	166
TTL Time Testing GUI.....	170
MATLAB Analysis Code Example	171
Appendix D: All Testing Results	174
Serial to Serial and TTL to Serial	174
USB vs. Native Serial	214
Operating Systems	227
Appendix E: Development Platform Selection Spreadsheet.....	232

List of Tables

Table 1: Mean and standard deviation of lag time from all tests from all serial ports to other serial ports and TTL to all serial ports and the standard deviation between individual tests.	80
Table 2: Mean and standard deviation of lag time from all tests in the configurations explained in this section, and the standard deviation between individual tests.	92
Table 3: Mean and standard deviation of lag time from all tests from all serial ports to other serial ports and TTL to all serial ports and the standard deviation between individual tests.	96
Table 4: First 20 synchronization markers from use case 1 experimental protocol. Showing byte received by EEG and fNIR recording systems, as well as the timestamp (from start of recording) and time from the first marker received.	107
Table 5: First 20 synchronization markers from use case 2 experimental protocol. Showing byte received by both fNIR recording systems, as well as the timestamp (from start of recording) and time from the first marker received.	114

List of Figures

Figure 1: Picture of a male DE-9 connector and serial pinout.....	35
Figure 2: Picture of a BNC connector.....	36
Figure 3: Picture of a male DB-25 connector and parallel port pinout.....	37
Figure 4: Board bring-up cycle diagram.....	39
Figure 5: Picture of an Arduino Mega 2560.....	44
Figure 6: Atmel AVR ATmega2560 pinout.....	46
Figure 7: TTL Pulse to serial transmission of bytes with ASCII values equivalent to the changed state of the line. Left represents the line being pulled low, transmitted as "0", or 0x30 in hex. Right represents the line being pulled high, transmitted as "1", or 0x31 in hex.....	51
Figure 8: Schematic wiring of first generation NeuroHub.....	52
Figure 9: Printed circuit board design of first generation NeuroHub.....	53
Figure 10: First generation NeuroHub PCB.....	55
Figure 11: First generation NeuroHub assembled.....	56
Figure 12: Additional schematics added for second generation NeuroHub.....	58
Figure 13: Printed circuit board design of second generation NeuroHub.....	58
Figure 14: Second generation NeuroHub assembled.....	61
Figure 15: Timing program process loop diagram.....	66
Figure 16: Successful byte transmission in the tests.....	71
Figure 17: Oscilloscope screenshot of one byte transmission. The first (top) line is the input signal (receiving Rx line), the second (bottom) line is the signal being sent from NeuroHub (all other Tx lines).....	73
Figure 18: Byte transmission timing explanation, at 9,600 bits per second.....	74
Figure 19: Bar graph of serial port 1 to serial port 2 mean transmission time, as determined by the timing program, with error bars.....	76
Figure 20: Histogram of transmission times from serial port 1 to serial port 2, as determined by the timing program.....	77
Figure 21: Bar graph of TTL to serial port 1 mean transmission time, as determined by the timing program, with error bars.....	78
Figure 22: Histogram of transmission times from TTL to serial port 1, as determined by the timing program.....	79
Figure 23: Bar graph of mean serial transmission time over USB ports without input from other USB ports, as determined by the timing program, with error bars.....	82
Figure 24: Histogram of transmission times over USB ports without input from other USB ports, as determined by the timing program.....	83
Figure 25: Bar graph of mean serial transmission time over USB ports with input from other USB ports, as determined by the timing program, with error bars.....	84
Figure 26: Histogram of transmission times over USB ports with input from other USB ports, as determined by the timing program.....	85

Figure 27: Bar graph of mean serial transmission time over native serial ports, as determined by the timing program, with error bars.	86
Figure 28: Histogram of transmission times over native serial ports, as determined by the timing program.	87
Figure 29: Bar graph of mean serial transmission time over native serial ports with input from non-native serial ports (as in other tests, with USB serial ports), as determined by the timing program, with error bars.	88
Figure 30: Histogram of transmission times over native serial ports with input from non-native serial ports (as in other tests, with USB serial ports), as determined by the timing program.	89
Figure 31: Bar graph of mean serial transmission time over native serial ports without the use of NeuroHub, as determined by the timing program, with error bars.	90
Figure 32: Histogram of transmission times over native serial ports without the use of NeuroHub, as determined by the timing program.	91
Figure 33: Bar graph of mean serial transmission time using Windows XP, as determined by the timing program, with error bars.	93
Figure 34: Histogram of transmission times using Windows XP, as determined by the timing program, with error bars.	94
Figure 35: Bar graph of mean serial transmission time using Windows 7, as determined by the timing program, with error bars.	95
Figure 36: Histogram of transmission times using Windows 7, as determined by the timing program, with error bars.	96
Figure 37: Diagram describing the flow of information in use case 1 without NeuroHub implementation.	102
Figure 38: Diagram describing the flow of information in use case 1 with NeuroHub implementation.	104
Figure 39: Left – the 3x3 P300 BCI matrix used. Right – the training manual selection screen. From [118].	105
Figure 40: Time line for a run. From [118].	105
Figure 41: Histogram showing the frequencies of the difference in EEG and fNIR times from the first marker.	108
Figure 42: Diagram describing the flow of information in use case 2 without NeuroHub implementation.	112
Figure 43: Diagram describing the flow of information in use case 2 with NeuroHub implementation.	113
Figure 44: Histogram showing the frequencies of the difference in fNIR times from the systems' first markers.	115

Abstract

NeuroHub: Portable and Scalable Time Synchronization Instrument for Brain-Computer Interface and Functional Neuroimaging Research

Nicholas V. Grzeczkowski

Dr. Hasan Ayaz, PhD

The advent of new and improved brain imaging tools in recent decades has provided significant progress in understanding the physiological and neural bases of motor and cognitive processes and behavior. As neuroimaging and brain sensing technologies are further developed, they are miniaturized and become portable and wearable, allowing brain activity monitoring in ecologically-valid everyday environments. This introduces the possibility of using multiple systems concurrently on i) the same brain: multimodal/hybrid measurements for better identification of neurophysiological markers, and ii) multiple brains: hyperscanning for novel investigations of brain functions during social interactions.

In all of these new directions, seamless integration of various neuroimaging systems is required. More specifically, precise time synchronization of acquired data streams is necessary for proper analysis and interpretation of results. However, there are currently no standards for interoperability and neuroimaging systems have many different designs and interfaces. Experiment setups using multiple systems may require extensive development for a customized solution that would need reconfiguration at the expense of additional time and effort, with the risk of possibly varying precision based on the custom solution.

To address these issues, we have developed NeuroHub, a scalable device that can provide plug and play and reliable time synchronization by interfacing with common ports

in neuroimaging systems. The device consists of a custom printed circuit board that fits atop an inexpensive and readily available development board for an Atmel ATmega2560 embedded microcontroller. It is housed in a 6 x 11 x 3.5 cm durable plastic casing, smaller than most smart phones, and includes BNC, serial, and parallel communication ports located around its perimeter. The device propagates any synchronization marker it receives from one of the ports and broadcasts it to all systems connected at other ports. The device can be extended as necessary by connecting multiple NeuroHub units. Verification and validation tests indicated reliable byte transmission with 100% accuracy of transmission and a consistent 1.020 millisecond latency in its standard configuration. A program was also developed for automated testing with Monte Carlo simulation, by sending and receiving event markers in various configurations. Through these tests, it became clear how unfit the use of multiple common computer ports is for sub-millisecond precise modality recording, due to their non-embedded nature. This problem is alleviated using NeuroHub as it allows synchronization of each computer through only one port.

NeuroHub was implemented in two use cases to demonstrate its potential: i) Multimodal spatial navigation brain computer interface (BCI) that used simultaneous EEG and fNIR for enabling controlling actions within MazeSuite generated virtual environment. ii) Synthetic speech perception study which utilized two different fNIR systems simultaneously to record from a larger area. In the first use case, the naïve P300 response is used as a selection mechanism for a number of options for first-person navigation of a maze. fNIR measurements are used to assess if the person is attentive to the stimuli, which results in higher accuracy scores. For this setup to be successful, markers between the

software for stimulation presentation, EEG recording, P300 analysis, maze presentation, and fNIR recording must be synchronized. In the second use case, subjects are presented with audio recordings of 5 sentences over 4 levels of quality of speech signal, ranging from natural speech to low quality synthesized speech, and asked to rate them for naturalness and intelligibility, while fNIR measurements are recorded to provide quantitative data about how cognitively taxing the synthesized speech is that has become common in everyday devices. In this experiment, information must be synchronized between the stimulus computer and the two fNIR recording devices. Both of these use cases demonstrate NeuroHub's utility in next generation experiment setups with the goal of helping brain computer interface and functional neuroimaging research.

Chapter 1: Introduction

1.1 Motivation

In the rapidly expanding field of neuroscience and brain-computer interface (BCI) research, there is a proliferation of experimental setups that incorporate the use of more than one recording modality. With multiple modalities, it is possible to extract new features that aren't possible otherwise, resulting in a more robust BCI or experimental recording setup. This typically makes the experimental setup more complex and the synchronization of data more difficult as there are more computers that need to work in unison and more data to temporally align. Another type of experimental setup, in which the neural basis for social interaction is studied by simultaneous recording from multiple subjects, is referred to as hyperscanning. As with multimodal setups, the separate streams of data must be aligned temporally. Event markers are used to align the data in both of these types of setups, but this raises the problem of sending the markers from multiple systems to multiple other systems in order to accurately align the data. Not only does this require cumbersome setup, extra planning to get everything to work together seamlessly, but it also involves the use of information transmission protocols that are not compatible and were not designed to accommodate signals with high temporal resolution, and therefore are not precise in their transmission.

1.2 Specific Aim

This project aims to develop a portable and scalable device to alleviate the complexities that arise from hybrid multimodal and hyperscanning setups, for efficient arrangement of custom setups and reliable event marker transmission. Currently, the markers must be sent via temporally unreliable protocols and each setup requires the development of this information flow. This project also aims to review current brain-computer interface literature on hybrid setups and categorize them. With a device that is specifically designed to overcome these challenges, new setups could more easily be developed and temporal imprecision could be eliminated.

With the use of a dedicated embedded system, the envisioned device could implement common ports (serial, parallel, and digital) and protocols (RS-232, SPP and TTL) broadcast a received marker out to all other on-board ports with respective protocols. The device would act as a hub that bridges multiple independent systems including stimulus presentation, functional neuroimaging (EEG, fNIR and fMRI) and other recording systems. The result would be that all recording computers receive all event markers, which would allow simple data alignment. This would also eliminate the need for a dedicated solution that can only work with a specific experimental setup.

1.3 Approach

First, current hybrid brain-computer interfaces are reviewed and common neuroimaging modalities noted. This provided the foundation and basic understanding of the present state of the field, and the context for which the device is intended. It does not

provide an in-depth review of modalities or meta-analysis which can be found elsewhere, rather, it shows the trend of hybrid setups and why they are a growing interest. From there, the difficulty of setting up these systems is outlined, and design requirements are identified. Then, system specifications are defined to accommodate these requirements. Next step is the implementation of prototype systems and testing. The first few iterations of the devices are prototyped, implementing new features and design changes. Finally, the device is then tested for lag time and reliability.

Chapter 2: Background

This chapter aims to review the literature in the field that has necessitated the device. Brain computer interface research is especially useful for locked-in syndrome patients who have no other way of communicating with the outside world. It provides them with a method to control a computer or other machinery using their brains alone. There are various paradigms that use different techniques to extract information from one or more neuroimaging modalities. These modalities are being more and more frequently used sequentially or in parallel to gain faster, more accurate control of a BCI, a more useful BCI, or a deeper understanding of how the brain functions. With these increasingly complex setups, communication protocols are utilized to synchronize the data being recorded and analyzed on multiple computers. However, computers are not designed to be highly accurate in their timing of the protocols, which results in varying lag times that decrease the precision of timing. The proposed device is meant to both simplify the setup of these complex systems and solve the lag time problem.

2.1 Brain-Computer Interfaces

Brain-computer interfaces, or BCIs, are systems that allow the user to voluntarily control the computer using thoughts alone, in the case of active BCIs, record the response to a stimulus intentionally chosen by the user, as in the case with reactive BCIs, or record a response to understand the state the user's brain is in, as with passive BCIs [1-3]. A reactive BCI is one that is used for voluntary control, but relies on a response from an

external stimulus, such as a light blinking at a certain frequency. These interfaces can be either invasive or noninvasive. Invasive BCIs (also referred to as Brain-machine interfaces, or BMIs) involve the implantation of electrodes, for single cell recordings (which can be arranged in matrices) with high spatial frequency or implantation of electrodes directly over the cortex under the skull to record large amounts of neurons firing synchronously, known as electrocorticography (ECoG) [4]. Non-invasive imaging has many forms, some examples being electroencephalography (EEG) [5], magnetoencephalography (MEG) [6], functional magnetic resonance imaging (fMRI) [7, 8], functional near-infrared spectroscopy (fNIR)[9, 10], and positive emission tomography (PET)[11]. The most popular brain imaging techniques rely on the hemodynamic or electrophysiological responses. The following sections aim to outline a medical condition BCIs are typically intended for, two popular and portable neuroimaging methods, EEG and fNIR, their usefulness in BCIs, and a review of hybrid BCIs.

2.1.1 Locked-In Syndrome

Locked-in syndrome is a condition in which the affected is aware of their surroundings but is unable to take action in it. It can be the result of various forms of medical complications, such as amyotrophic lateral sclerosis (ALS), also known as Lou Gehrig's disease, multiple sclerosis, brain injury or hemorrhage, nerve damage, stroke, or some circulatory diseases[11]. The most common cause for locked-in syndrome, however, is ALS. ALS is a progressive neurodegenerative disease that causes the

degeneration of motor neurons, resulting in an inability for the brain to control muscle movement.[12, 13]. It starts with muscle weakness and eventually, as muscles become less and less stimulated, they atrophy (become smaller). Eventually the afflicted can become totally paralyzed and therefore categorized as having locked-in syndrome. Locked-in syndrome patients cannot move anything with the exception of their eyes and sometimes facial muscles, however cognitive activity and brain function stays intact throughout this process in majority of these patients [14, 15].

Living in such a state is understandably a difficult hardship to cope with, with no output to the ideas they wish to express. Their thoughts are absolutely suppressed and the afflicted constantly need to be cared for by others, greatly decreasing their quality of life. This is where the necessity for brain-computer interfaces is relevant. If better methods to control communication setups or physical mechanisms are developed, some of the pain of locked-in syndrome could be alleviated.

2.1.2 Non-invasive BCIs

Non-invasive brain imaging has typically been used in brain-computer interfaces to provide an output for the patient, although invasive devices have also been under development for some time and the field is currently growing rapidly [2]. These non-invasive BCIs use different modalities and rely on different techniques to come up with an appropriate method of control. Each of these modalities has benefits and disadvantages which have to be weighed carefully according to application. By using different modalities together, those benefits can be combined and new dynamics can be

understood. Because communication is so important for the wellbeing and quality of life of the patient, a lot of focus has been placed on developing speller setups and techniques to make them faster. Research has not been limited to helping the disabled, however. Much can be learned about the brain using BCIs and non-invasive imaging systems are widely used for various forms of neuroscience research.

2.2 Electroencephalography in BCIs

2.2.1 EEG Principles

Electroencephalography, or EEG, utilizes electrodes (conductive passive sensors) to capture the electrical changes due to neural activity over the scalp noninvasively. The change in voltage over the scalp is due to massive numbers of neurons firing in synch at different frequencies. This is mainly due to the interactions between neurons: a synapse from one neuron could excite or inhibit another neuron in various complex ways that are beyond the scope of this review. The electrical signals picked up by the electrodes placed on the skull are known as local field potentials (LFPs). Oscillations of LFPs due to synchronous activity are more commonly known as brain waves. Reading these brain waves allows certain insight on what is happening in the brain.

Historically four major types of continuous rhythmic sinusoidal EEG waves (rhythms) are recognized (alpha, beta, delta and theta). Delta is the frequency range up to 4 Hz and is often associated with the very young and certain encephalopathies (cerebral diseases) and underlying lesions. Theta is the frequency range from 4 Hz to 8 Hz and is

associated with drowsiness, childhood, adolescence and young adulthood. This EEG frequency can sometimes be produced by hyperventilation. Theta waves can be seen during hypnagogic states such as trances, hypnosis, deep day dreams, lucid dreaming and light sleep and the preconscious state just upon waking, and just before falling asleep. Alpha (Berger's wave) is the frequency range from 8 Hz to 12 Hz. It is characteristic of a relaxed, alert state of consciousness and is present by the age of two years. Alpha rhythms are best detected with the eyes closed. Alpha attenuates with drowsiness and open eyes, and is best seen over the occipital (visual) cortex. An alpha-like normal variant called mu is sometimes seen over the motor cortex (central scalp) and attenuates with movement, or rather with the intention to move. Beta is the frequency range above 12 Hz. Low amplitude beta with multiple and varying frequencies is often associated with active, busy or anxious thinking and active concentration. Finally, Gamma is the frequency range approximately 26–80 Hz. Gamma rhythms appear to be involved in higher mental activity, including perception, problem solving, fear, and consciousness.

EEG recording has high temporal resolution, meaning it can differentiate activity that are very close in time, as the readable signal has a fast fluctuation in voltage level. One disadvantage of this modality, however, is that it has a low spatial resolution. It is therefore difficult to identify and locate the source(s) of oscillations. Using high density electrode array such as 128 or 256, and it may be possible to estimate source locations and contributions using sophisticated techniques such as independent component analysis[16]. Although more invasive techniques have been developed, such as

electrocorticography (essentially intracranial EEG), that offer higher spatial resolution, EEG remains popular for its noninvasiveness and its many uses.

The most common use of EEG is for clinical applications specifically epilepsy and sleep studies, in which neuropathologies yield abnormalities in the EEG signals. Epilepsy results in seizures in which normally asynchronous brain oscillation behavior abnormally synchronizes and fires in excess. Surgeons that are about to perform an epilepsy surgery on epileptic patients who do not respond to medication use EEG in the process of locating the region of the brain that is the source of the seizure. EEG has also been shown able to detect the onset of a seizure. Other diagnostic uses for EEG include comas, encephalopathies (a wide variety of brain disorders), and brain death. EEG is also used in sleep studies, as brain oscillations change at different stages during the night.

2.2.2 Sensors and Instrumentation

The standardized layout of the electrodes is known as the International 10-20 system, 10 and 20 being the percentages of distance between electrodes from the anterior to posterior and medial to lateral directions, respectively. The ground electrodes are usually placed behind the ear. Most electrodes are passive, requiring amplification and the injection of a special conducting gel (or, in some setups, saline solution) to get a signal. Active electrodes are able to acquire readings without the use of gel, but the signal they pick up is noisier and the electrodes are too expensive to be considered useful in many settings. For simpler objectives, such as entertainment purposes or to provide

consumers with rough biometrics, active electrodes are used because they are simpler to set up. For academic or clinical use, passive electrodes are usually used.

In passive electrode EEG setups, amplification of the signal is necessary. This is accomplished with an amplifier that is hooked up to the electrodes on the cap which also samples the signal and sends them to the recording program on the computer. The particular model used in one of the use case setups was the NeuroScan NuAmps digital amplifier model 7181. It has 40 unipolar analog inputs, and can also be used for other types of neurophysiological signals, such as ECG, EOG, and EMG. It has a parallel port connection in the back that allows the integration of event markers from a stimulation software package to the recorded signal.

There are many software packages available for custom stimulus presentation. Among the more popular packages are E-Prime, Presentation, and BCI2000. They offer standard paradigms and the ability to customize them to suit the needs of the experiment/BCI. Mazesuite [17], a spatial navigation software developed at Drexel University, was another stimulation package that was used in one of the use case setups.

2.2.3 Examples of EEG based BCIs

Over the past 30 years EEG has been demonstrated in many settings as a successful brain sensing modality for various BCI paradigms. Among the most well studied of these paradigms is the P300-based matrix speller [18] which uses the P300 evoked potential that occurs when a user recognizes a rare target stimulus. An event-related potential (ERP) is an EEG based signal that is response to an internal or external

stimulus. Experimental psychologists and neuroscientists have discovered many different stimuli that elicit reliable ERPs from participants. The timing of these responses is thought to provide a measure of the brain's information processing and communication timings. P300 response occurs at around 300ms as a positive peak in the oddball paradigm, for example, regardless of the stimulus presented: visual or auditory. Because of this general invariance in regard to stimulus type, this ERP is understood to reflect a higher cognitive response to unexpected and/or cognitively salient stimuli.

The P300 based BCI acts on a characteristic response due to a resolution of anticipation of sorts. When an awaited stimulus presents itself, a positive response can be seen at about 300 ms after the stimulus in the EEG recording. The usual stimulus presentation is visual, although auditory [19, 20] and tactile [21, 22] based P300 BCIs have also been developed [23].

For visual stimuli, the presentation is usually a matrix of the alphabet and some other characters necessary for typing. The characters randomly blink, one at a time, and the user is instructed to watch their intended character and count the number of time it blinks within a set amount of time. By identifying a consistent P300 response for a single letter on all cycles of the run, the BCI analysis software can determine which of the characters was intended to be selected during the run and in turn select that character. This type of BCI was first developed in 1988 by Farwell and Donchin [18]. Other uses for P300 in BCIs have been developed and include other types of speller layouts and analysis techniques that have improved performance [24-27] and P300 for spatial navigation [28]. For a review of P300 based BCIs paradigms, refer to [29]. Another

notable improvement was the use of delta and beta band powers to predict when the user is paying attention to the system [30].

Another common type of BCI also makes use of the visually evoked rhythms in occipital lobe[31]. One can cause entrainment of the EEG signal to a frequency by gazing at a display flashing at that frequency. Thus, flashing icons at different frequencies provides the user with a frequency-coded selection that can be identified through EEG recordings made between the Pz and Oz locations over the visual cortex. This type of BCI works off of a response known as steady state visually evoked potential, or SSVEP. The amplitude of SSVEP greatly increases near the center of the visual field, and therefore is strongest when the user is gazing at a flashing icon. Therefore, by looking at an icon, the user can select it.

For analysis in searching for an SSVEP, first the peak frequency is detected, then it is determined if the peak is above a certain threshold. If it is above the threshold, the icon coded with the above-threshold frequency is selected. The amplitude of the response also is dependent on the frequency, with some frequencies showing a greater response than others. Therefore, it is advantages to use those frequencies that are more detectable. Lower frequencies cause flickering that can be annoying to users, so higher frequencies can be used; however, higher usable frequencies are more difficult to detect. The advantages of SSVEP are that it is fast and there is no training required, as with most other types of EEG based BCIs [32-34].

Sensorimotor rhythms have also been incorporated in BCI[35]. Motor imagery based BCIs require the user to imagine using parts of their body as a control mechanism

[36-38]. Typically this comes in the form of power asymmetry of mu (8 to 12 Hz) rhythms between the left and right hemispheres [39]. For example, a user may be asked to imagine moving their left, and alternatively their right, hand. Classifiers are identified after training the BCI and online analysis used to control the system. These systems are asynchronous and provide the user with self-paced timing of whatever the BCI is intended to control, whether that be a wheelchair [40] or a specialized speller setup[41]. Another asynchronous feature that can be extracted from EEG to for control is the regulation of slow cortical potentials, or SCPs [42, 43], although this is less commonly used for inherent difficulties the method presents.

EEG has been shown to be a useful modality for controlling BCI systems, with a few standard feature extraction methods with advantages and disadvantages of each. Current research in the field often involves improving these methods by changing stimulus presentation, using different analysis techniques, or using those features to control different parts of a system. Other passive uses include sleep research [44] and other neuroscience areas. All of these setups require event markers to align stimulus presentation with sub-millisecond EEG data being acquired. It is therefore crucial for the usefulness of the system that the event markers are precisely aligned.

2.3 Functional Near-Infrared Spectroscopy in BCIs

2.3.1 fNIR Principles

Optical brain imaging takes advantage of optical properties of hemoglobin in blood to track metabolism related changes in order to reveal information about brain function. Functional near-infrared spectroscopy, or fNIR, is the use of near infrared light to measure the cortical hemodynamic changes of a brain area by observing changes of light absorption in that specific area [45-47]. When neurons are activated, they use energy in the form of glucose. This process requires oxygen, which is transported to the cells by oxy-hemoglobin, which is then deoxygenated. Oxy-hemoglobin and deoxy-hemoglobin absorb light at different wavelengths, hence two different near infrared wavelengths can be used to spectroscopically resolve concentration changes of each chromophore (light absorbing molecule). The changes in oxy-hemoglobin (HbO), deoxy-hemoglobin (HbR), total-hemoglobin (summation of HbO + HbR) and hbD (difference in hemoglobin) changes can be used to determine the brain activity in a specific location or over time. This is also similar to functional magnetic resonance imaging (fMRI) based Blood Oxygenation Level Dependent (BOLD) signal as both measure the same underlying hemodynamic changes. However, due to large instrumentation, high cost, complicated setup and subject restrictions (no metal, need to stay motionless in supine position, high data collection noise) of fMRI, it is not as amenable to neuroergonomic studies as fNIR. Diffuse optical techniques (those used for fNIR) have been shown to

have similar results to fMRI measurements [48] and therefore could be used in situations where fMRI is impractical either by nature of experimental protocol or cost.

fNIR operates on the fact that human tissue is relatively transparent to light in the near infrared wavelength window (the optical window is around 650 to 950 nm) and light in this range is therefore able to penetrate and read the relative changes in transparency. While skin and bone are relatively invisible to near-infrared light, hemoglobin, the protein that transports oxygen in red blood cells, is responsible for most of the attenuation in tissue. Laser diode or LEDs are used to transmit light at wavelengths at which oxy-Hemoglobin and deoxy-Hemoglobin are most responsible for absorption: 850 nm and 730 nm, respectively. The light travels through the tissue in a banana shaped curve to sensors placed in the surrounding area, with the distance from the transmitter determined by the depth of tissue intended to be read, which is approximately half the separation of the light source to the sensor.

This modality is much less expensive to build and operate than fMRI, and is also much more portable, which are reasons that it is gaining in popularity in research. However, fNIR can only measure outer cortex and cannot measure deeper brain structures as an inherent limitation due to optical nature of the tissue. However, temporal resolution of fNIR can be much higher than fMRI, allowing recording more temporal characteristics. The ability of the system to measure the hemodynamic response using such a portable, even wearable low cost system has increased the popularity of the modality. It is most commonly used in passive BCIs but, as it has been shown that a user can voluntarily activate the hemodynamic response, it is also able to be used for active

BCIs. It has also been used in clinical applications such as depth of anesthesia monitor [49], and even variations of the technology has been implemented as medical devices that has received FDA approval for use in a portable handheld intracranial bleeding monitor, Infrascanner [50].

2.3.2 Sensors and Instrumentation

There are various devices available that make use of optical diffusion techniques to provide metrics about the hemodynamic response, and depending on their complexity, are able to utilize several techniques for more information, at the tradeoff of a more complex and expensive system. These measurement types are time domain, frequency domain, and continuous wave. Time domain involves very short pulses into the tissue that are measured and analyzed for temporal distribution of light by scattering and diffusion. Frequency domain involves modulating the amplitude of the light at various frequencies, where the amplitude decay and phase shift give information about the optical properties of the tissue. Continuous wave flashes light at a constant amplitude and measures the attenuation of the signal at relevant wavelengths. Each of these systems is composed of one or more transmitters, sensors, and electronic hardware. If there is only one light source, the system is a point-measurement system, while more than one source, providing a map of brain activity, the device is classified as an imaging instrument.

The portable continuous wave fNIR sensor for prefrontal cortex has been developed by the Optical Brain Imaging team at Drexel University in collaboration with Dr. Britton Chance. The system has been used in a spectrum of application areas that

require quantitative measurements of the hemodynamic response in a natural environment, such as an objective analysis of cognitive workload [9, 51-54], working memory [55-58], attention[59, 60], problem solving[61, 62], learning/training [46, 63], neuromarketing[64], brain disorders[15, 65], and rehabilitation [66-69]. For such applications, the continuous wave method is utilized, because it is portable, affordable, and easier to engineer than time domain or frequency domain methods. This makes it able to be used in a variety of situations that the other methods would be unable to.

The current fNIR imaging system consists of a headband that attaches to a base system. The headband contains a flexible circuit board inside a silicone band, with 4 LED light sources and 10 detectors, each 2.5 cm away from the light source. Each of the light sources are measured by the four surrounding detectors for a total of 16 fNIR acquisition channels, or voxels. Each source pulses 2 or 3 different wavelengths: one for oxy-hemoglobin, one for deoxy-hemoglobin, and one for dark current. The sampling rate is 2 Hz, and during a sampling period 48 measurements are made (at each voxel, at each wavelength). The measurement data is sent via USB port to a computer and recorded using COBI Studio, also developed at Drexel University [47, 70].

2.3.3 Signal Processing and Analysis

To obtain useful information from the data acquired, the Modified Beer-Lambert law is used. The Beer-Lambert law states that the amount of light that passes through a medium is proportional to the amount of light absorbing molecules in the medium. It also takes into account the absorption properties and concentration of the molecule and the

distance the light must travel through the medium. It must be modified, however, to take into account the scattering effects of tissue. Using the following equation, one can determine the amount of oxy-hemoglobin and deoxy-hemoglobin in the area between the source and the detector.

$$\log\left(\frac{I_0}{I}\right) = \alpha cLB + G$$

I_0 represents the intensity of the light entering the medium, and I is the intensity of the light measured after passing through the medium. α represents the absorption coefficient of the molecule in question, c is the concentration of the molecule, and L is the length the light must travel through the medium. B and G are the modified parts of the equation, and they represent an experimentally derived correction factor for L and a constant attenuation factor due to the optical properties of tissue, respectively. To assess the effect that a stimulus has on signal attenuation, a baseline recording is made and compared to the post-stimulus reading. The difference can be rearranged to make the following relation between the attenuation ratio and the change in concentration:

$$\Delta c = \frac{\log\left(\frac{I_{rest}}{I_{test}}\right)}{\alpha LB + G}$$

Although continuous wave systems are less susceptible to motion artifacts [71], there is still a need for motion artifact correction. Motion artifacts appear as spikes that corrupt the data. Numerous methods have been deployed to address this need, either by using an accelerometer [72] to sense movement or by using signal processing methods such as statistical filtering [73] or wavelet based algorithms [74] to reject the artifacts. Also contaminating the data are scattering and absorption changes from the superficial

layers of the scalp, another problem which correction efforts have been applied. Both principle component analysis and independent component analysis have been applied to remove changes due to the hemodynamic response in skin blood flow, as this is irrelevant to brain function [75]. Classical statistical analysis methods, such as ANOVA and t-tests, are commonly used to analyze fNIR data. A review of motion artifact removal for fNIR can be found in [76]. A review of fNIR instrumentation and methodology can be found in [77].

2.3.4 Examples of fNIR based BCIs

This section reviews various recent BCIs that utilized fNIR. A wide variety of applications have been found for fNIR. Although there have been systems developed that use the hemodynamic response as a voluntary control mechanism, the majority of fNIR based BCIs are passive studies on cognitive load. As fNIR measurements are directly related to changes in levels of oxy-hemoglobin and deoxy-hemoglobin due to the hemodynamic response, and the hemodynamic response is caused by brain activation, fNIR can be used to predict brain activation due to cognitive workload with the correct correlates [9]. It has also been used to monitor training of simulated piloting of unmanned aerial vehicles (UAVs), where Ayaz et al. found a high correlation of brain activity to the users' performance as well as their self-reported experience [9, 56, 78]. Moreover, other followup studies also confirmed that fNIR measurements can be used to predict the mental task load as well as the training effect, level of expertise in a given task[46, 52, 54, 57, 58].

In [79, 80], the assessment of cognitive neural correlates was convincing enough to suggest the use of fNIR as a BCI mind switch. In other words, the user of the BCI would be able to control a BCI by volitionally changing their mental state. A simple, one-channel fNIR BCI was developed for this purpose in [81]. This idea was taken further through implementation into a virtual environment in which the user navigated using a traditional computer keyboard but interacted with doors in the environment using fNIR. After a training phase in which the user received fNIR biofeedback, the user was then assigned the task of navigating a maze with 5 doors, each being opened by voluntary changes in brain state [82]. In another example, called Brainput, Solovey et al. demonstrate the possibility of using fNIR data to control a passive negative feedback system that in turn controls the autonomy of a system the user is controlling. They showed that with this approach they were able to improve performance metrics [83].

fNIR has also been used to study how the brain learns. In [47], fNIR was used in combination with Mazesuite [17] to show how to study the brain while it learns spatial navigation. In other learning protocols, verbal-spatial working memory was studied for increased working memory capacity in training, as well as other improvements to learning that are based on experimental data [57]. That particular study found a negative relationship between verbal working memory performance and bilateral ventrolateral prefrontal cortex (VLPFC) activation.

More uses have been found to use fNIR as a BCI besides a two state mind switch. In one study, fNIR data was used for a 4 class (left hand, right hand, left foot, and right foot) motor-imagery BCI with which all three participants of the experiment achieved

accuracies that were higher than chance (54 %, 50 %, and 33 %) [84]. It would then be logical to use this in combination with EEG based motor imagery for a more accurate system than the two modalities achieve individually. As fNIR used in the previous example reads from the prefrontal cortex only, it is understandably more difficult to predict intended movements than by measuring oxy and de-oxy hemoglobin at the motor cortex, which was done successfully in [85]. Although this study was only comparing 2 states (left hand vs. right hand) compared to the other study's 4, they achieved 73 % accuracy using Support Vector Machines (SVM) and 89 % accuracy using Hidden Markov Model (HMM) algorithms.

The number of uses for fNIR is growing constantly and are too numerous to list everything here. Other notable examples include using fNIR for developmental neuroscience [71, 86] and imaging the medial prefrontal cortex (where information about self is processed) to understand how basic psychological need satisfaction affects the difficulty of deciding while answering questions about self [87]. Following this trend, studies will continue to improve fNIR data analysis and fNIR-based BCI performance.

2.4 Hybrid BCIs

Hybrid BCI (or hBCI [88]) research has stemmed from the need for better performance systems, which is necessary if BCIs are ever to be commonly used in clinical settings [89]. A hybrid BCI is one that employs the use of more than one BCIs, or one BCI and another system, or both [90]. It has been noted in [91] that there is inconsistent language used in literature to describe the differences in hybrid BCIs and

attempts to categorize them. That group defines two BCIs being used together as a pure hybrid, a BCI that also incorporates another physiological signal as a physiological hybrid, and a system that uses another, non-physiologically based assistive technology as a mixed hybrid. In these definitions, they fail to categorize BCIs that use different features from the same modality. This is necessary as unimodal hybrid BCIs have also become a recent trend. The multiple systems can also be classified as being combined sequentially, where one system controls the other, or simultaneously, where both systems are processed in parallel. Reviews in hybrid BCI research can be found at [90] and [91].

This section aims to define the differences in unimodal and multimodal BCIs and give examples of each. The main categories can be broadly defined as follows: BCIs that utilize multiple analytical techniques from the same modality, BCIs that employ multiple modalities, and those that use a combination of both. The third category will not be covered thoroughly, however, because no systems were found that match this description. Physiological and mixed hybrids will not be separately covered.

2.4.1 Unimodal Hybrid BCIs

Unimodal Hybrid BCIs make use of multiple paradigms from the same modality at the same time. They extract different features from the same modality, but not necessarily the same set of data. One EEG feature could be extracted from the occipital lobe, which is being combined in a BCI with a motor imagery scheme that uses data from the around the sensorimotor cortex. The advantages of various analysis techniques are combined by using the strengths of each used feature in creative and appropriate ways.

Using more than one modality can combine the advantages of each modality and make a faster, more informative, and easier to use BCI setup. For example, an unimodal hybrid setup mentioned earlier used both P300 in its traditional form, but added the use of delta and beta band powers to improve accuracy by attempting to understand when the user was paying attention to the system [30]. Another type of unimodal hybrid BCI technique is known as hyperscanning. Hyperscanning typically involves measuring the same modality on more than one subject to study social interaction. For examples of this reference [92-97].

P300 ERPs and SSVEPs were used together in a P300-based speller matrix for an asynchronous BCI that used SSVEP techniques to determine control state (on or off) [98]. This is an improvement because P300-based BCIs are typically synchronous and require that the user be actively engaged at all times or the system will produce false results. Looking at the screen produced an SSVEP, which the system responded to by activating the P300 system. Because SSVEP is better suited for asynchronous control than P300, it was also used to make another P300-based BCI asynchronous, in which the user selected discrete commands for a smarthome environment in [99]. In this case, SSVEP was used as an on/off switch for the system, and resulted in high accuracy and reliability.

As P300 is a good feature to use when discretely selecting one out of many options and motor imagery is better for continuous control of fewer options, the two have been used together for various purposes. In [100], the two were combined sequentially to control a wheelchair. P300 is first used to select a location to which the wheelchair is to

travel, then for stopping the wheelchair, first a fast P300 stop command was implemented, then a motor imagery stop command was used in its stead to see the difference. Although the response time was very similar, the motor imagery command managed to attain zero false acceptances. The two features were used together for control of a virtual environment, using motor imagery for navigation and P300 to control virtual devices [101]. Although this setup did not improve accuracy, it showed P300 and motor imagery used in combination for a novel setup. The two features were also used in combination in another example to control a robot [102].

Motor imagery has also been used in combination with SSVEP in various setups. In [103], the two were used together to show that higher classification accuracy is possible using both rather than each individually. LEDs blinking at SSVEP-appropriate frequencies were used to indicate which direction the user was imagining. With motor imagery alone, 74.8 % accuracy was achieved and with SSVEP alone, the average was 76.9 % accuracy. Using the two together in a hybrid system produced higher accuracy that averaged at 81.0 %. Furthermore, the number of BCI illiterate subjects, who achieve less than 70 % accuracy, was reduced to 0 (originally 5) in the hybrid setup. This is significant as illiterate subjects are a great challenge for BCI research [104]. Motor imagery was also used as a brain switch to turn on an SSVEP controlled orthosis. SSVEP was used to control opening and closing tasks, and in combination with the motor imagery controlled mind switch, false positives were reduced by more than 50 % [105]. Another example of an SSVEP-motor imagery hybrid BCI is [106], in which the two were used in sequential combination to control functional electrical stimulation (FES), a

technique used to control limbs that can no longer be controlled by the brain but still have functioning muscles.

2.4.2 Multimodal Hybrid BCIs

Multimodal hybrid BCIs use more than one modality at a time. They work together to overcome physiological and recording limitations imposed on systems by the individual modalities, and in their combination are able to come to new findings, as well as improving accuracy and reducing error. The most common modalities record electrophysiological or hemodynamic changes. Since unimodal setups do not have high spatial *and* temporal resolutions, combining modalities allows for a new spatiotemporal resolution. Multimodal setups have also given insight to the mechanisms that govern neurovascular coupling [107] although a clear description of it remains elusive [108].

Multimodal hybrid BCI analysis methods are categorized, although it is notably difficult to do so. There are asymmetric or symmetric methods, and supervised, unsupervised or model driven methods. Supervised methods are asymmetric because they choose an independent variable from one modality, to which the other modality is fitted. Unsupervised methods do not require the predictor variable, but interpretation of results is not as easy as with supervised methods. Model driven analyses are difficult to develop because a robust realistic model of hemodynamic and electrophysiological measurements is difficult to establish [107].

Electrooculography (EOG), the measurement of eye muscle potentials, has been used as a control mechanism on its own [109-111], but has also recently been used in

parallel with EEG for a novel control scheme [112]. In it, the user controls the direction of a machine using EOG while controlling the state (stop, forward, or do nothing) of the system with EEG. To control the state, power spectral density in the alpha and beta bands were used, with different methods to control the bands instructed to the user. To stop, the user closed their eyes, which increased the alpha band. To move forward, the user thinks about moving forward, which increases beta. If both bands were under threshold, nothing was done. In this fashion, 100 % accuracy was reached for stopping completely and turning left or right, 87 % for moving forward, and 95 % for the no action state. In another example, EEG was used in an EOG controlled system to tell if the user was paying attention [113]. The combination of EOG with EEG is advantageous because there is a relatively high accuracy rate, it doesn't require many electrodes, and there is a short training time.

EEG has also been used with other modalities. It was used with electromyography (EMG), recording the electrical activity from skeletal muscles, to assist with the problem of muscle fatigue [114]. Also using EEG with EMG, [115] proposed the development of a speller using EMG to control a selection "click" on whatever letter the user was viewing. In the paper, they compare the use of EEG to EOG for letter selection.

More frequently, EEG is being used with fNIR for improved systems. In [116], the two were used together to improve accuracy in motor execution and motor imagery tasks, although the slow hemodynamic response may increase reaction time. For executed motor tasks, the mean accuracies without EEG were 71.1 % and 73.3 % for HbO and HbR, respectively, but with EEG accuracies were improved to 92.6% and 93.2%. For

motor imagery tasks, the accuracies improved from 71.7% and 65.0 % to 83.2 % and 80.6 %. In the study, it is reported that classification accuracy was able to be improved in over 90 % of the subjects using the hybrid approach. NIRS measurements were also used as a mind switch for an EEG motor imagery controlled robot in [117]. In chapter 5, a setup using P300 with fNIR is presented in more detail.

In another example, fNIR was used for a mind switch to turn on or off an SSVEP-based orthosis control BCI. With this setup, Pfurtscheller et al. were able to achieve 100 % accuracy [105]. In [118], fNIR data was used in parallel with a P300 spatial navigation BCI to predict performance rates. In another study, NIRS measurements were used with transcranial Doppler ultrasonography (TCD) to improve accuracies during a verbal fluency task in a block-stimulus paradigm [119]. Used alone, NIRS produced a mean accuracy of 76.1 ± 9.9 %, and TCD an accuracy of 79.4 ± 10.3 %. Used together, they produced a mean accuracy of 86.5 ± 6.0 %. In this study, 5 out of 9 participants were able to significantly improve accuracies ($p < 0.05$) using the hybrid setup over the two imaging techniques used individually.

As stated previously, one of the advantages of using fNIR is the ability to use it in more common situations than fMRI could for its size and nature of the signal. Also, it has been shown to improve results being used in real life situations rather than mock tasks that are common with fMRI. In one such example, the results from an fMRI and fNIR recorded mock apple peeling task on the prefrontal cortex were compared to prefrontal cortex readings on a real apple peeling task that would not have been possible in an fMRI setup [120]. The results showed increased activation in the real apple peeling sessions,

showing the potential for fNIR to be used in real situations that would not be otherwise possible with fMRI. Also using fNIR data with fMRI data, [121] found that it was possible to use fNIR to reduce variance in fMRI residual error by up to 36 %.

2.5 Need for a Solution

Because of the nature of novel experimental BCI setups, they require features specific to the experimental design. This requires the added task of making all of the involved systems work together, which is time consuming and gets in the way of the research. These systems are often complex themselves, and when the data from more than one monitor is required to be synchronized to markers that corresponds with the time of an important protocol event or stimulus, there is the added task of temporally aligning the data. Also, the imprecise timing schemes of standard computer protocols would cause imprecise recording of event marker times, increasingly so with more data arriving at another port, such as from a neuroimaging device or even a computer mouse. More information on this is provided in chapter 4.

2.6 Current Approaches

Recording can be done on an expensive, hardware dedicated data acquisition device, as long as the output is in volts. This leaves out the usefulness of transmission protocols. Other efforts to synchronize data in real time include frameworks multimodal interactions with virtual environments[122], augmented reality[123], and activity recognition such as through wearable and ambient sensors [124]. Highly precise timing is

not pertinent in these situations. There are a number of costly options at websites like <http://gridconnect.com/> that offer RS-232 to UDP options, RS-232 splitters, or similar pieces of hardware, which could be used as routing solutions for the signals. However, most previous efforts have been custom assembled for the requirements of a setup.

This issue has been addressed in a software based solution developed by the Swartz Center for Computational Neuroscience at UC San Diego called the Lab Streaming Layer, or LSL (<https://code.google.com/p/labstreaminglayer/>). It provides support for a variety of devices to work over a network and temporally aligns them. It is full featured, with a recording program, online viewers, and more, but requires programming experience to integrate into a custom experimental setup. As is described later in this document, through testing it was found that using the computers' protocols for time synchronization can be disastrous for temporal precision of data, for they are not embedded systems and this sort of timing is inherently unimportant to the operating system, and communication over these protocols is necessary for recording from devices. Because of this, there is often lag that is unknown and unaccounted for in the results, which could lead to false conclusions being reached. Effort is required if the user intends to have a minimal lag time.

In LSL, each stream of data must have an offset timing according to the nature of the stream for temporal correction. Different modalities in the streams have been tested with varying results (<http://scn.ucsd.edu/~mgrivich/Synchronization.html>). Also, the attached computers are running on different computer clocks, and a variable drift likely caused by temperature variations on the clocks speeds. This website explains that in their

testing of LSL, outliers can be explained by OS lag or drift due to a statistical error in the fit of a DAQ: http://scen.ucsd.edu/~mgrivich/LSL_Validation.html. That is with one stream of data from one computer to another, measured using a 2048 Hz sampling rate.

Time synchronization in LSL is described here:

<https://code.google.com/p/labstreaminglayer/wiki/TimeSynchronization> and is not specialized for synchronization of multiple streams. It relies on the timestamp each system/device provides via UDP, and if the device isn't reliable, neither are the timestamps. There are documented efforts to characterize devices' lag time using LSL, such as <http://wiki.neuroelectrics.com/images/a/a5/NEWP201401-WhitePaper-EventSynchronization.pdf>. Essentially, lag time can only be corrected so much.

It is those drawbacks in mind that the device being presented here aims to alleviate. Researchers ideally should be able to focus on the experiment at hand rather than devoting time to working out inevitable technical difficulties. These difficulties both distract the researcher with frustration and waste time that should be used on the experiment itself, setting back the flow of progress.

An ideal setup would allow the researcher to simply connect all event trigger output ports (whatever type of port it might be) to a single plug and play microcontroller-based device that is also connected to the computers that need to receive the markers through those same ports. The device would replicate all data flowing in to all ports flowing out for maximum simplicity in setup. Because the system would be a dedicated embedded device, all input to the device is being sent to all receiving computers at the

same time, and lag time due to OS port protocol would be negligible because all data is coming in through one port.

Chapter 3: Device Design and Development

3.1 Device Design

The selection of the microcontroller used required an understanding of both the specifications of the desired application as well as what is available on the market to fill those requirements. The first step was deciding what the most important features the device must have, and from there make a list of general requirements. After determining the general requirements necessary for the device, a microcontroller and development platform were chosen that could fulfill those needs, and development was continued from there.

3.1.1 System Requirements

The device must be able to make connecting computers in BCI setups simpler. It must be highly temporally precise and reduce the time needed to wire the system together. The first and most important two requirements decided were that the device had to be fast enough to respond in time to sub-millisecond event triggers and that the most popular communication protocols used for transmitting event triggers must be available. The timing was so important because EEG requires high temporal precision and the event triggers must be sent immediately after they are received. The popular protocols are important because the device is meant to make custom setups easier, so the popular protocols are the most important to be incorporated. That way, incorporating systems with different networking options can easily be integrated, providing an easy plug and

play setup procedure. Another advantage of having multiple commonly used ports is the ability to convert protocol types to types without having to write custom conversion scripts in the stimulus software.

Design constraints were decided on with the idea to make the device inexpensive and simple to use. For example, the device was first developed (when still in the breadboard development phase) to have switches from each of the ports to each of the other ports, to provide custom routing abilities. That way, the user could control which port is sending to which other port. Several methods were implemented and tested, but this was later decided to be omitted, as this could provide a problem to a researcher. In the nature of a simplistic design, the extra complexity was deemed unnecessary and could possibly cause more harm than good. Redundancy of information showing up at ports would not be a problem, as the researcher must intentionally be listening to the data anyway. It is likely that during the setup of an experiment, the user could become frustrated trying to troubleshoot why a marker isn't being transmitted because a switch was accidentally overlooked, and typically some software is listening to the signal, which is why information being redundant is not a problem. Compared to the complexity and cost of neuroimaging systems, the device should be simple to use and the cost of making it should be minimal. Testing must show the device to have negligible and repeatable lag time to prove efficacy.

3.1.2 System Specifications

In picking the most useful connection types to integrate, the serial port was determined to be the most important because of its popularity. Another important event marker method is through the use of a transistor-transistor logic, or TTL pulse, which was to somehow be transmitted in some fashion to the other ports. For the first generation of the device, these two were deemed necessary and other protocols were to be implemented in later generations.

Because the serial port is so commonly used in these applications and provides bidirectional information transfer, it was decided to put 4 serial ports on the board. If more ports are needed, NeuroHub could be daisy chained to other NeuroHubs by simply connecting a serial cable between them. Typically, serial communication is achieved using the RS-232 protocol, which does not operate at the same voltage levels as most microcontrollers. Microcontrollers operate at 0 V (logic 0) to 5 V (logic 1), while RS-232 operates at around 13 V (logic 0) and – 13 V (logic 1). A level conversion chip is necessary in this case.

Software serial communication is possible by bit banging. Bit banging is the process of sampling the pins at a rate sufficient to read all bits and implementing the protocol by use of software. However, bit banging is notoriously unstable and therefore not optimal for this application. Microcontrollers often have UART communication that allows the hardware to control the serial data. This method is more stable and therefore hardware serial communication was decidedly necessary.

The serial communications settings were to be set for standard, i.e. the most common, settings, and the settings of the computer needed to be adjusted if necessary. The standard RS-232 connector is the DE-9 connector, shown below. For more information on RS-232 protocol, see the development section, 3.4.

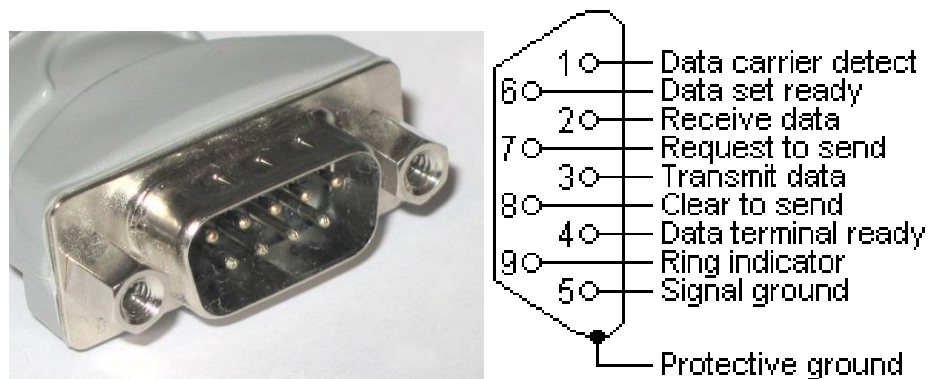


Figure 1: Picture of a male DE-9 connector and serial pinout.

TTL pulses were to be sensed by input pins and converted to a byte to be transmitted through the other ports. A popular connector used to connect TTL communicating devices, and the one on the back of the fNIR Devices fNIR Imager, is a Bayonet Neill-Concelman, or BNC, connector. Perhaps the BNC port could send the byte serially, but this would add unnecessary complexity for an uncommon procedure in this sort of setup. Serial digital interface (SDI) using a BNC connection exists in the professional video world, but is not used to send event markers in experimental setups and therefore is unnecessary. Because the bytes from other ports could not simply be

converted to a TTL pulse, the TTL port was decided to be input only. The way this was handled was sending the American Standard Code for Information Interchange (ASCII) code for the character “1” to all other ports if the TTL line was pulled high, and likewise “0” if the line was pulled low. Any other TTL cable could easily connect to the BNC connector by attaching the signal wire to the inner wire and the ground to the outer conductor and it would work the same way, as long as it operates at 5 V.



Figure 2: Picture of a BNC connector.

During development of the first generation board, more research went into popular communication connections that event markers are sent across. One such connection, the parallel port, has become somewhat outdated in the computer world and is no longer included on most computers, but is still used for event marker communication from various software and hardware packages. For example, the

NeuroScan NuAmps EEG amplifier receives event markers via parallel port and places it alongside the incoming EEG data. There are different protocols that are used over the parallel port. There is protocol for automatically detecting the type of protocol that will be used to communicate over the line, but because only basic parallel communication was found to be used on available systems, implementation of other protocols was put off until found to be necessary. The parallel port was implemented into the second generation board, and although basic parallel communication does not require all of the pins to be used, the lines were still physically connected to microcontroller ports in the case that other protocols were to be implemented. For this, the chip used needed to have enough digital in and out ports to accommodate the 25 pins on the DB-25 connector, the most common connector used for parallel ports and the one that was to be used the NeuroHub.

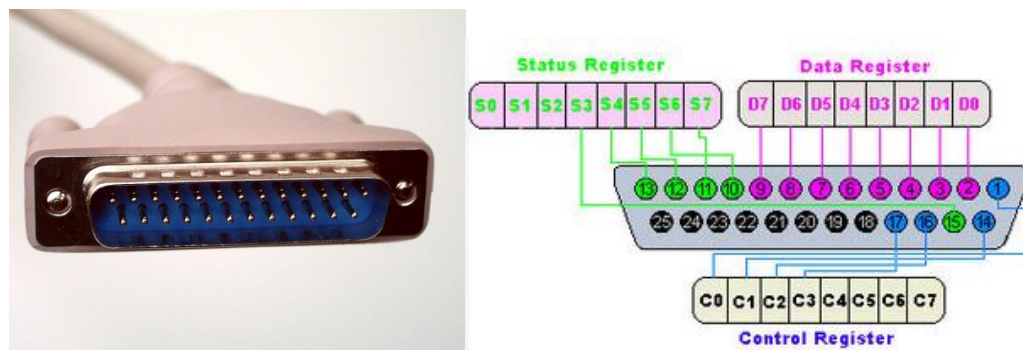


Figure 3: Picture of a male DB-25 connector and parallel port pinout.

The reasoning for the inclusion of mentioned protocols and microcontroller selection was stated above for the purpose of reasoning design decisions. The specifics of development of these features will be described in the development section that follows.

3.2 Device Development and Implementation

After making the decision as to what the development environment and microcontroller should be, the development of required features was begun. Initial implementation was completed on a breadboard before committing to a design. The printed circuit board was designed using CAD software, printed, and tested for functionality after the chip was programmed. The device was then assembled into a finished product.

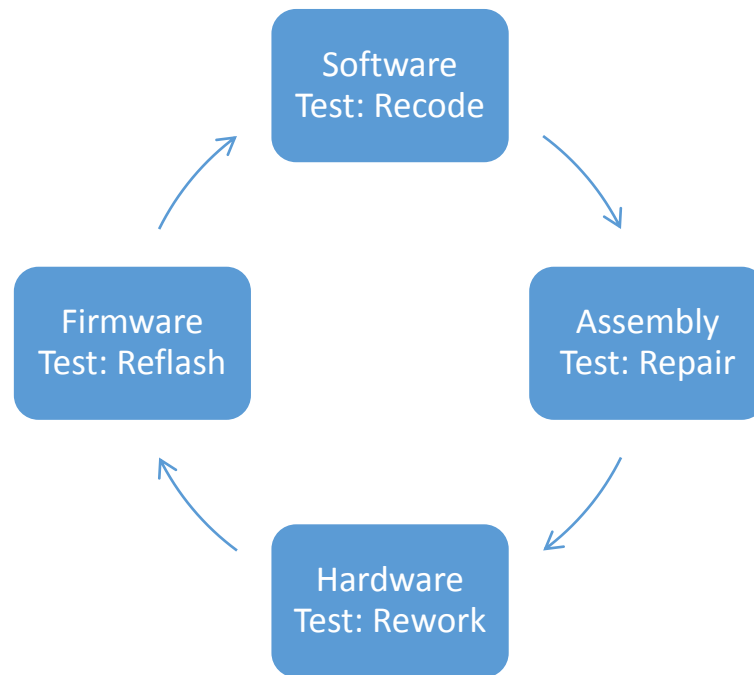


Figure 4: Board bring-up cycle diagram.

The development of an entire electronic device system is known as board bring-up. It includes validation and debugging, and the sequence is repeatable at any stage of the prototype development. It involves the assembly of the device, the hardware development, the coding of the software, and programming the firmware of the microcontroller. It must be verified that the board has been assembled correctly. The hardware is then tested for basic functional connectivity. The code is debugged, and the firmware uploaded and tested. This is repeated through the development of new prototypes.

3.2.1 Development Platform and Microcontroller Selection

During the initial stages of device development, it was difficult to understand what specifications the device needed, as microcontrollers have an overabundance of options, and it wasn't clear which of them were absolutely necessary from the design decisions made. Therefore, the necessary options were laid out, the microcontroller market was researched, and the least expensive microcontroller that fit the description was selected, because the device was to be kept at as low a cost as possible. Also important to select before development was started was the development platform. Typically, in microcontroller based device design, the microcontroller is attached to a board with the necessary components to have it up and running, while having the pins available to be attached to a breadboard or other prototyping platform. This allows the developer to eliminate microcontroller setup as a possible source of error during debugging the software testing phase of board bring-up.

For serial communications, at least 4 hardware dedicated universal asynchronous receiver/transmitters (UART) were required. As mentioned before, software implemented serial communications are reputedly less stable than their hardware counterparts. A UART basically translates data between serial and parallel communications, which is important because the microcontroller deals with bytes in parallel. An integrated chip made for this purpose could possibly have been used, but this would take up additional ports on the microcontroller which could prove to be limiting. It was decided that the microcontroller selected should have the necessary UARTs built in. This was the bottleneck requirement; most chips with 4 UARTs supplied the other options as well.

Because the device requires superior temporal precision, a high clock speed was required. A pin change interrupt pin on the chip was necessary for TTL pulses. Enough digital in and out ports were required for when the parallel port protocol was to be implemented in generation 2, and an SPI bus for Ethernet and SD logging card was necessary for when generation 3 was to be developed.

For efficient development, it was decided that a hardware debugging setup would be useful as part of the setup, whether it be on board or with an external debugger/programmer, although this was later decided against due to weighing advantages (see development section). A breadboard/prototyping area was preferred but not necessary, as most pins can be accessed via wires to a breadboard. It was to be programmed in C.

The development setup selection was important, for there are many options and the selection would have a major impact on the project workflow. The two major setups in question were using a development/evaluation board, or using a surface mount (SMD or SMT) breakout board on top a breadboard and using a programmer with debugging capabilities. A breakout board makes the pins of a surface mount chip available to use on a breadboard for development and prototyping. The breakout board would be needed when picking a specific chip because the chips with at least the specifications required are SMD which cannot be used by itself on a breadboard. Only through-hole components can be used directly on a breadboard and all of the chips with required features were not available in through-hole configuration. The reasons against using a breakout board is that it requires hardware configurations that are already available on a development

board, which would be time consuming and more risk for human error where it isn't necessary. However, the chips are very inexpensive compared to development boards, and it was possible that the hardware setup would need to be eventually transferred to a PCB, so setup knowledge would have been useful for when the PCB is to be designed. As is described later in the document, this (transferring to the breadboard) was not the case, and the added frustration of debugging hardware configurations would surely have been a setback. It was decided early on to obtain a development board rather than constructing one myself using a breakout board.

All boards examined offered a programming environment tailored for use with its respective board, and most, if not all, could be programmed in C. C was chosen as the programming language as it is currently the most popular language to program microcontrollers with, as well as being well documented with most example code being written in that language. An 8-bit chip was chosen for its relative simplicity in setup compared to 16 or 32 bit systems, which are generally more expensive and draw more power. The additional resources they offer were also not necessary for this device. FGPAs were considered but not included in the search as the extra complexity and configurability was not needed. Popular development boards marketed to hobbyists such as Arduino, Raspberry Pi, and the Intel Galileo, were also considered.

The most relevant microcontroller development boards were selected from top microcontroller producers and compiled on a spreadsheet alongside certain attributes key to selecting the most appropriate board. Where it was appropriate, a parametric search was used pointing to the options with the most available UARTs, as stated before, this

was the bottleneck feature. A wide variety of available options were included in the search to give a clear idea of what is available on the market. The spreadsheet can be found in appendix E.

3.2.2 Generation 1 Development

3.2.2.1 Serial Capability Integration

As mentioned before, during the initial stages of device development and while a chip and development environment were being selected, a basic development board and integrated development environment (IDE), Arduino, was used. The specific board used was the Arduino Mega 2560. It has the highest specification microcontroller of all of the Arduino boards, the Atmel ATmega2560 microcontroller chip. The Arduino platform was useful at this stage because of its simplicity and ease at implementing features, which comes at the expense of speed and memory.

The development phase commenced using a breadboard and the Arduino programming environment. The Arduino programming language was used in this case to make sure the hardware was wired correctly, but it was not used in the device except for initial development purposes. The integrated development environment, or IDE, for Arduino takes up too much overhead for this application when compiling the code, making the code run slower. As per the design requirements of speed, lag was to be kept at a minimum, so C was used for the programming of the device. C is typically the language used to program AVR microcontrollers. Upon comparing sizes of the .hex files

(the result of compilation and that which is programmed onto the chip) from simple code in both the Arduino IDE and its identically functioning counterpart in Atmel Studio 6 in C, we found that the Arduino compiled version takes up about 10 times the amount of space in the flash memory, although a quantitative comparison of speed was not conducted.

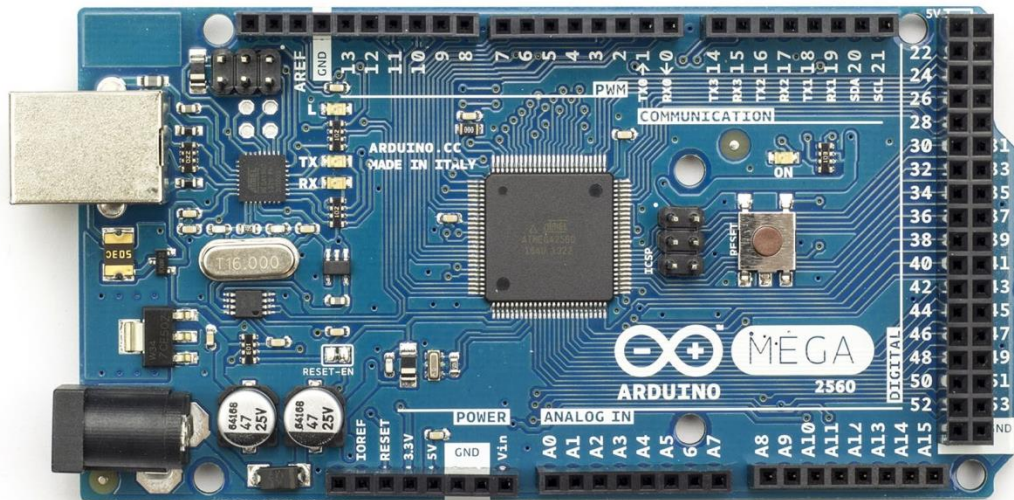


Figure 5: Picture of an Arduino Mega 2560.

The first protocol implemented was serial communication. While parallel ports send all the information in a byte across 8 data lines, serial communication uses one line for data transmitted in a string of pulses. The hardware dedicated UARTs convert the serial data to usable parallel bytes to be processed by the firmware on the chip. However,

because the microcontroller and the serial operate at different voltage levels, a level shifter IC made specifically for this application, MAX-232 by Maxim Integrated, can be implemented. The microcontroller operates at TTL levels: 0 V is logic 0 and 5 V is logic 1. RS-232 operates at a much higher voltage range: +13 V is logic 0, while -13 V is logic 1. The logic levels are flipped and voltages are dangerously high for the chip. The MAX-232 chip converts the voltage and logic levels of both transmitting and receiving lines. The actual chip used was the MAX-3232 on a breakout board. The MAX-3232 is a variation that offers conversion of 2 bidirectional serial channels. It was wired to the Arduino board on pins TX0 (transmit serial 0) and RX0 (receive serial 0) for one serial connection and TX1 and RX1 for another.

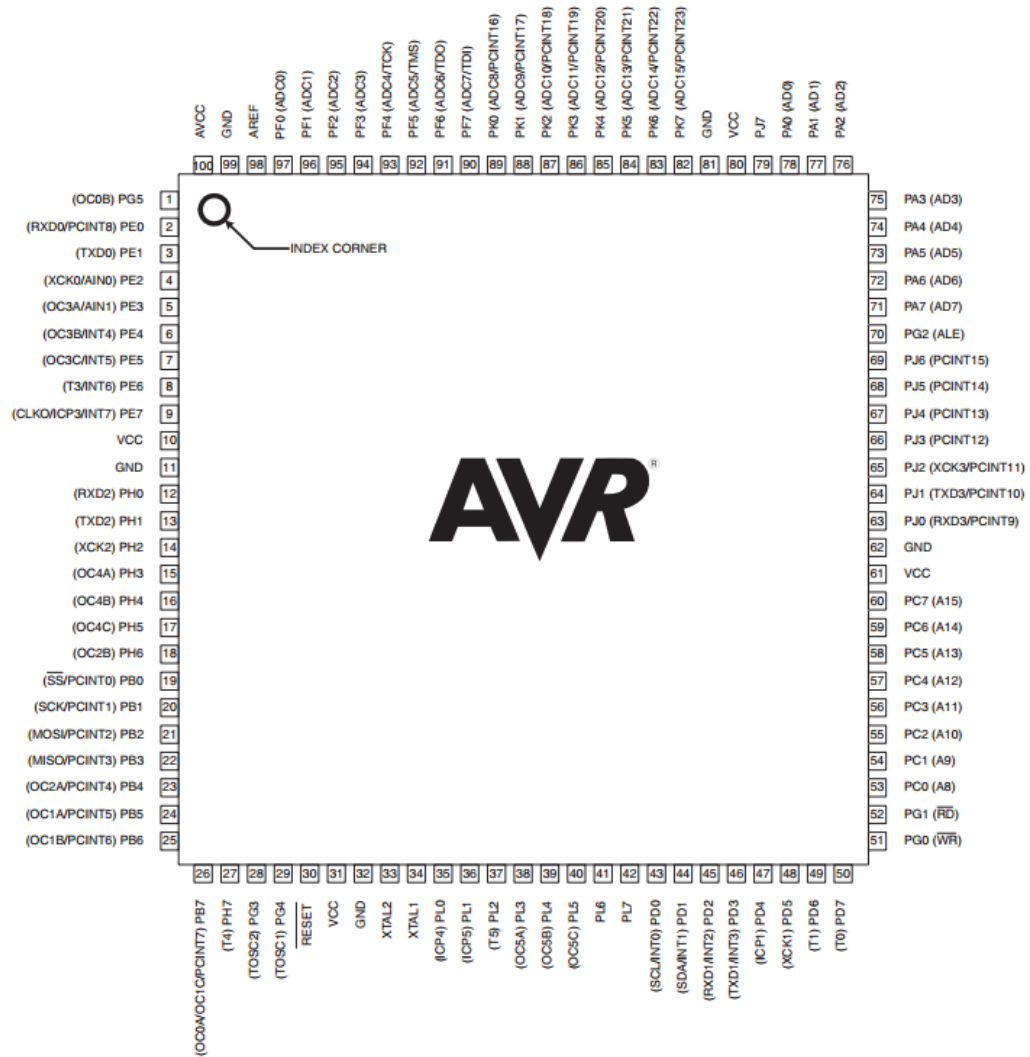


Figure 6: Atmel AVR ATmega2560 pinout.

The setup was assembled on a breadboard and the code tested for functionality. Once the setup was validated as functioning as intended, we switched programming environments to Atmel Studio 6, a free compiler used to program Atmel chips in C. A USBtinyISP programmer which was soldered together and assembled was used in combination with AVRdude, an AVR utility used to edit the contents of the ROM and

EEPROM memory of the AVR line of microcontrollers, to write the .hex files to the chip's flash memory. The .hex files are the output files of the compile process, formatted for use in microcontrollers, and burned to the chip with the programmer. In this method of development, the in-circuit serial programming (ICSP) pins on the Arduino board are utilized instead of the USB port to program the board. Every time the chip is rewritten, the old contents are erased, which leads to the Arduino bootloader being erased. The bootloader is there originally to allow the Arduino user to program the device without any additional hardware, as in with the USB port. Programming the chip via the ICSP chips erases this bootloader as well, and unfortunately the programmer that we were using came out before the ATmega2560, and was not compatible with it. This was not of consequence, because the future iterations of the software were to be uploaded to the board in the same way and the Arduino IDE was not to be used.

As for the programming of the chip to deal with serial data as specified by the design, two methods of routing the data from these ports were initially considered. The first code to be written used a method known as polling, which involved checking all the ports, one at a time, for information, and if it found information there, it would output that same information at all of the other ports.

The next method to be implemented was an interrupt based system. An interrupt driven system is advantageous in this sort of situation because the program doesn't manually check each port, but has dedicated triggers that initiate a segment of code. In this way, a flag is posted which breaks the program off from wherever it is and completes the interrupt code. Essentially the microcontroller is automatically constantly checking

for certain actions to take place if it is set up properly. These actions stop the main portion of the firmware code and run another section of code specially designed for that action. Because interrupts can be fired in the middle of running other interrupt code, at the beginning of each interrupt service routine (ISR), global interrupts are turned off, then turned back on after the code is complete. The microcontroller then looks to the other interrupt addresses and if a flag was set it jumps to the appropriate ISR. This was added in the case that two event markers are sent simultaneously, which otherwise might have resulted in partial bytes being sent.

Each serial port had its own ISR programmed. First, global interrupts were turned off, as mentioned previously. Second, the byte was stored to a variable, then that variable was written to all other ports. Interrupts were then turned back on and the flag was cleared.

A method (a custom function) was created to initialize all four of the UARTs and set them to the most standard serial communication settings by changing the registers for those options. It would be run whenever the program was started, i.e. when the device was plugged in. The desired baudrate was used to calculate the settings of two registers used to regulate baudrate by using a timing crystal attached to the Arduino board. The baudrate could then be easily changed in the code if for some reason it was needed in communicating with other devices or computers in the setup with a different baudrate. It was set to default at 9,600 bits per second, what serial ports are typically set to as default. It was decided that the UARTs would not require stop bits or parity, and

that characters received and transmitted would be 8 bits long (which can be changed, if necessary).

It was at this point that it was realized that the ATmega2560 is an ideal chip for the project, as it contains all of the necessities for the project and the features that we intended to implement. It was decided that development would continue on the Arduino board for a few reasons. The boards are inexpensive and readily available, making the device easier to replicate and daisy chain if needed. Also, if absolutely needed, the firmware code can be customized as necessary for a project. We were familiar with the development setup by this time so staying with the Arduino saved development time as well. Also, rather than having to design a board that catered to the microcontroller's needs, focus would be placed on functionality and a board would be designed to be used as a "shield," or custom board made for Arduino. The board has a need for external physical components, which would have been time consuming to debug and transfer to a PCB design after the features had been developed. There were no notable disadvantages (with the exception of no debugging capabilities) to this approach, and the advantages were numerous enough to come to the conclusion that it was the most reasonable method.

3.2.2.2 TTL Pulse Sensing Capability Integration

In the development section, it was mentioned that an input pin would be used to detect TTL pulses and convert the pin changes to bytes, "1" for when the pin went high and "0" for when the pin went low. There are a few pins on the ATmega2560 that can be attached to interrupt routines for level changes. Second in interrupt priority only to the

RESET pin is the INT1 pin, the External Interrupt Request 1, so when the pin changes the response is immediate. The INT pins can be configured to interrupt in one of four conditions (low, high, rising, or falling) and are viewed as independent interrupts, rather than triggering an entire port's interrupts with any pin change like with PCINT. The pin can detect a rising pin or a falling pin as long as the pulses are longer than 50 nanoseconds. Because the microcontroller operates at 5 V, no level conversion was necessary.

Like with the serial initiation, a custom function was created to run on device startup that set the registers for options required for TTL pulse functionality. Pin 1 on port D is where the INT1 interrupt is located. INT1 was configured to sense any level change edge, and the interrupt was activated. More information can be found in Appendix C.

The code proceeded as follows. If a pin change was detected, as in the case of receiving an event trigger, the interrupt is fired. It first turns off global interrupts, then checks if the pin is high or low (different interrupts cannot be set up for each, unfortunately). If the pin is high, it stores the character "1", or 0x31 in hex, to a variable, and if the pin is low, it stores the character "0", or 0x30 in hex, to a variable. It then sends the variable to all other ports as it did with the serial ports.

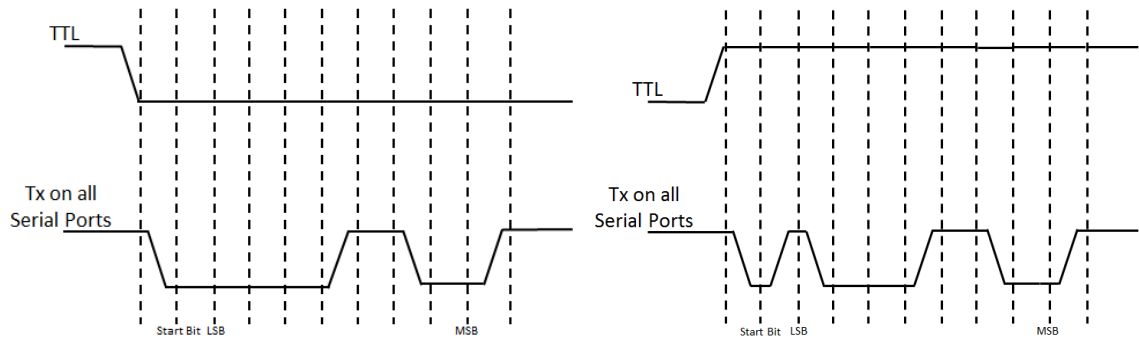


Figure 7: TTL Pulse to serial transmission of bytes with ASCII values equivalent to the changed state of the line. Left represents the line being pulled low, transmitted as "0", or 0x30 in hex. Right represents the line being pulled high, transmitted as "1", or 0x31 in hex.

The code was tested by creating a breadboard friendly BNC connector, with the signal line connected to the interrupt pin and the outside shield connected to ground. It was tested using the fNIR system, which sends a TTL pulse event marker when various actions are completed. Every time one of these actions was completed, a "1" then a "0" would appear on serial monitoring software connected to a serial port, which showed that the port was functioning as intended.

3.2.2.3 PCB Design

After testing the code out on a circuit board setup, a more complete product, a printed circuit board, was developed. The boards were first printed then assembled, in different methods for different versions of the board. The board was to fit over an Arduino board as a "shield." A plastic case meant to encase an Arduino with a shield was used for the device case. The board was improved in a second version.

A common PCB CAD program, Eagle CAD, was selected because it is one that is commonly used and well documented so it was relatively easy to set up the board. The device parts to be used were either found in existing parts libraries or created from scratch. A MAX238 through-hole IC chip was used, and the necessary capacitors were attached accordingly. The serial ports were wired to their spots on the chip, and the chip was attached to the headers corresponding to the correct pins on the Arduino. The BNC port was also attached to its necessary pin and to ground. The parts were wired together as shown in figure 8.

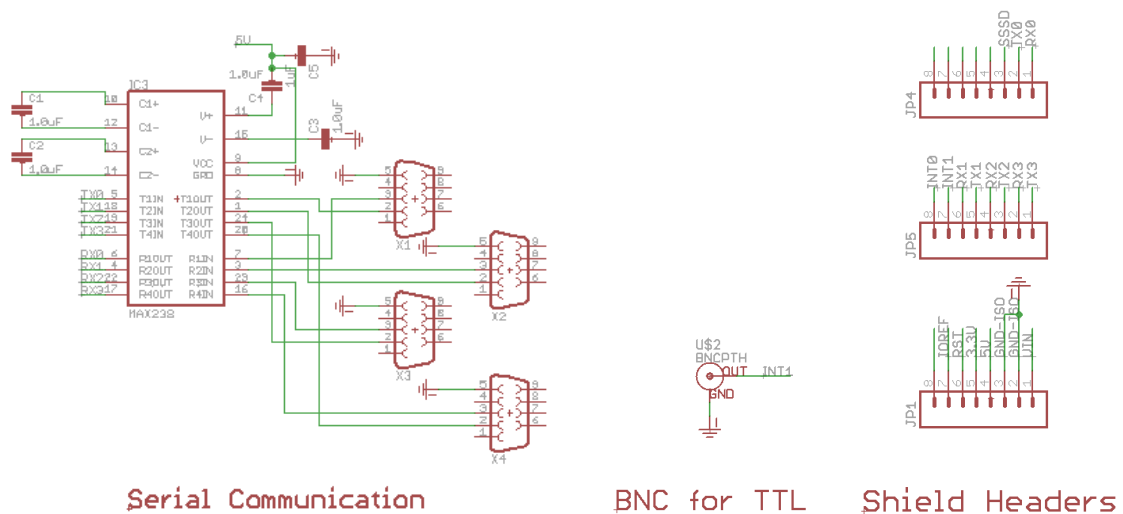


Figure 8: Schematic wiring of first generation NeuroHub.

The PCB was then arranged and routes created. The first generation board was to be printed at Drexel's Electrical Engineering facilities, but problems with the machine-printed board slowed development for a brief period of time. The machine could only print two layers, and the through holes weren't plated. This resulted in the necessity to solder only the side of the board from which a trace emerges, which was often under the plastic of the piece being soldered on. Routing everything on the opposite side of the board from the part on the next print fixed the problem. Vias, holes that connect different sides of the board, could be added, but those available were the wrong size for most components and had to be placed at a free spot on the board rather than the component hole.

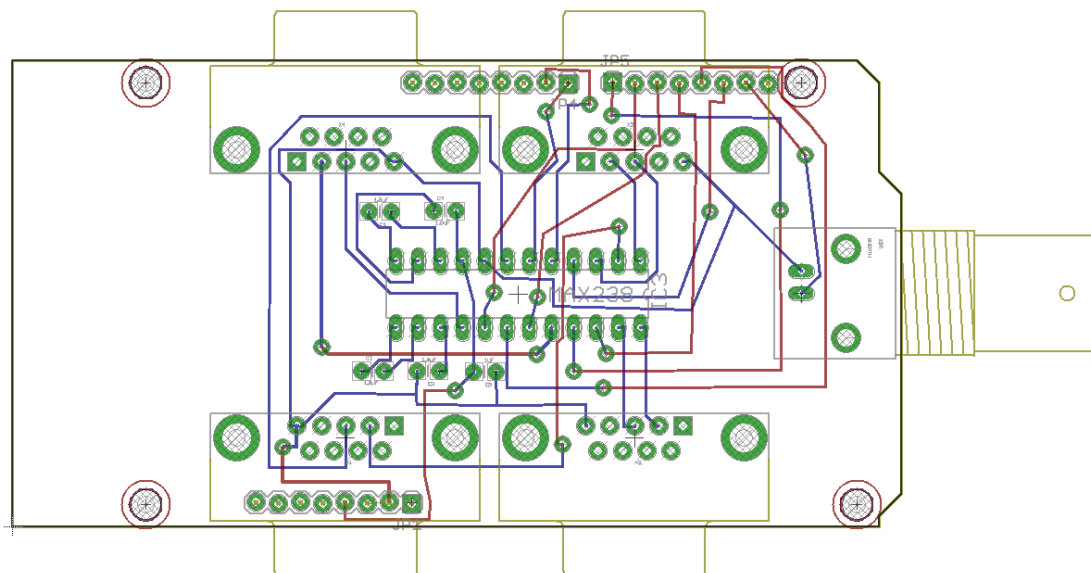


Figure 9: Printed circuit board design of first generation NeuroHub.

Another problem encountered with the board was the ability of the traces to stick to the board. The PCB printer was an LPKF ProtoMat S62, a prototype PCB milling machine. Much like a protoboard, the copper traces were relatively lightly attached. In the first design, the RS-232 connectors were to stick out of the side of the case through holes, attached to the PCB at a right angle. Therefore, whenever the serial ports were attached or detached, there was strain on the traces that were attached to the pins from the right angle connector, which caused the traces to break connection on a few occasions. In retrospect, the traces could have been made larger, which could have helped, but it was also not clear what the problem was at the time. The board would work, then would stop working without anything haven been intentionally changed. For this reason, the plans for the second generation of the board included a setup where the ports are attached to the side of the case with wires to the board. That way, the strain goes to the strong plastic case and freely moving flat ribbon cables.

The milling machine caused other development setbacks as well. On one printing, the printer's alignment was off so the top traces and outline were not correctly aligned. On the final board of generation one, a few wires were used to bridge connections that had broken.

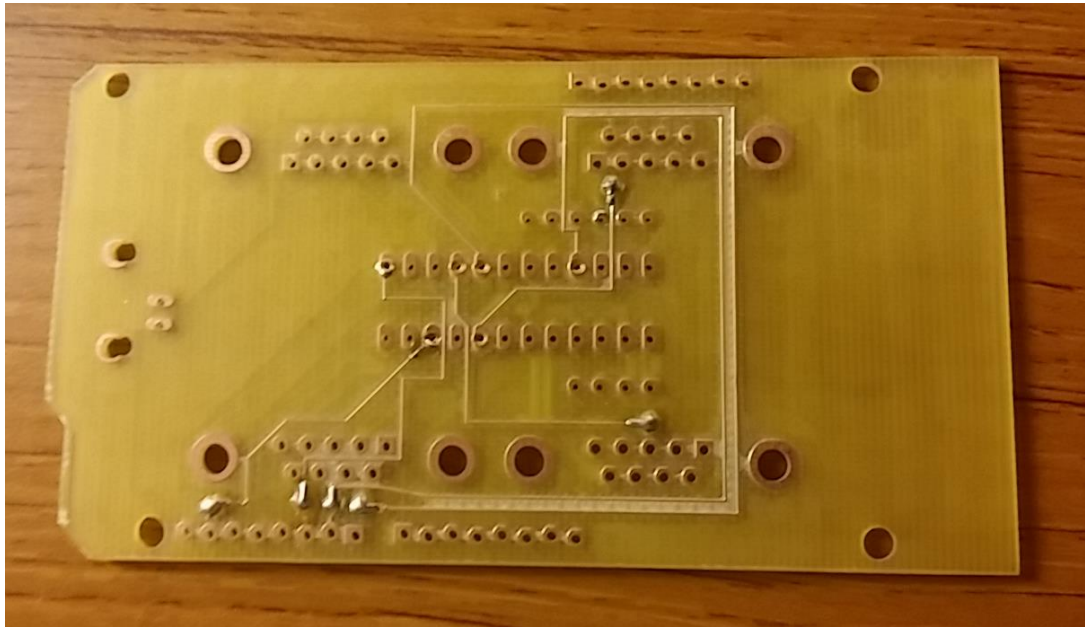


Figure 10: First generation NeuroHub PCB.

Because the board was going to fit over an Arduino board, the board was made the same outline as the Arduino Mega 2560. The BNC port was put on the opposite side of the power and USB connections, and 2 serial ports were put on each side. As mentioned before, they were to be soldered on using right angle connectors, as was the BNC connector.

Initially, the case was to be printed with a 3D printer, but typically prototype cases are selected from the large variety of premade electronics cases on the internet. One was found that suited the needs of the project. To access the ports with the case on, adjustments needed to be made to the case. This was done using a Dremel tool. The initial cuts were made to the size of the port, then additional material was grinded away

to fit the top of the case on correctly and provide space for the metal casing that fits around the ports. Unfortunately, this left an unsightly gap from the bottom of the connectors to where the bottom part of the case met the top part. The previously mentioned idea to have the connectors detached from the board would solve this. The case was labeled, and with that NeuroHub generation 1 was completed.



Figure 11: First generation NeuroHub assembled.

3.2.3 Generation 2 Development

The second generation board was to both add features and improve the setup from lessons learned in developing the first generation. The original idea of having a parallel port for the second generation was the most important improvement, while a

professionally printed board, a new layout, and a different assembly approach were additional improvements to be made.

3.4.3.1 PCB Design

Parallel port development was started on a solderless breadboard but was not completed until after the PCB was finished and files sent to a printing house in an effort to save time. In Eagle CAD, the BNC port was moved to the opposite side of the board to fit into a convenient hole in the plastic casing meant for an Ethernet connection that is very similar in size. Where the BNC port was, two parallel lines of header holes were placed to connect to the parallel port. These header holes were attached to appropriate ports on the microcontroller. The entire range of pins on the parallel port was attached to microcontroller ports, in case other protocols were to be implemented later in development.

The most important 8 bits of the parallel port, the data line, were attached to port A of the ATmega2560. Ports E and G were to be used for control and status lines, respectively. All 3 of these ports were located directly underneath the location of the parallel port connector. This proximity made routing the board more simple for a two-sided design.

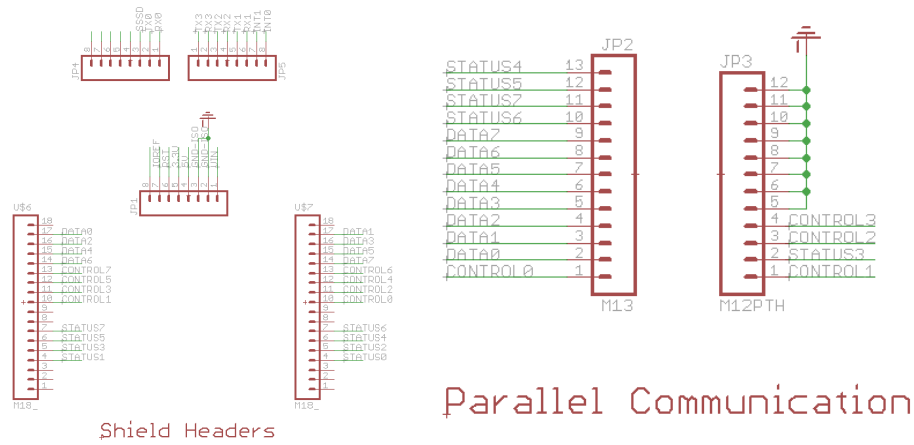


Figure 12: Additional schematics added for second generation NeuroHub.

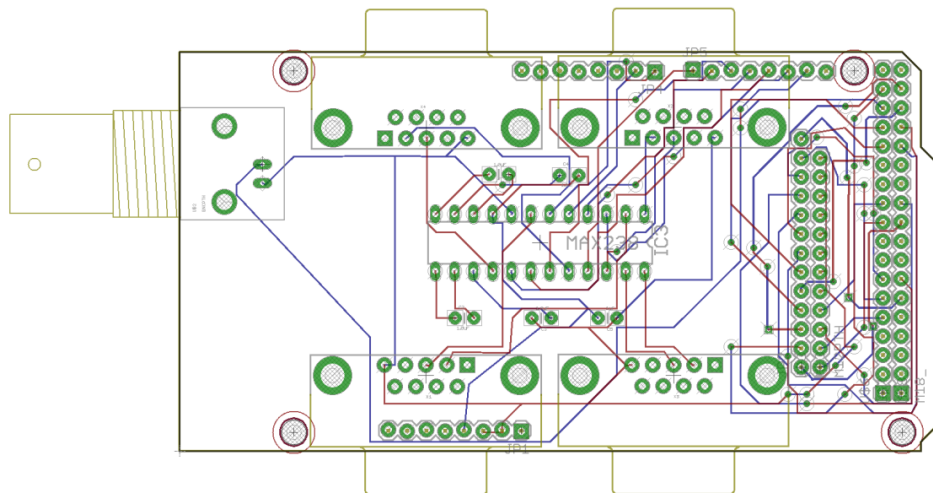


Figure 13: Printed circuit board design of second generation NeuroHub.

3.4.3.1 Parallel Port Integration

The board was then sent to a PCB printing house, OSH Park (<https://oshpark.com/>), for a durable, quality printed board, which was then assembled.

The serial and parallel port connectors were attached to one end of ribbon cables with the other end being attached to a line of pin headers to fit in designated holes on the board. Rather than carving up the side of the plastic casing, holes were drilled to the size of the ports (with the exception of the BNC connector). The idea is that the headers can be slid through the port holes, put in their designated holes, and soldered from the other side of the board. The ports were screwed into place. The assembly process is explained in more detail in appendix A.

There are two roles played in parallel port protocol, and usually the computer plays the “master” role, controlling a “slave” attached device and requesting information from it. When there is a parallel port available on a computer, it is usually a female connector.

As the board was to be implemented first with setups in the Drexel University labs, it seemed an appropriate starting place to research the type of parallel port protocol to be used. The EEG amplifier available was the NeuroScan NuAmps digital amplifier model 7181. A specially designed stim-to-scan cable is used to connect the male DB-25 parallel port at the back of a computer to a female DE-9 connector on the amplifier. Typically, a stimulation protocol software package sends event triggers over the parallel port, which is received by the amplifier and stored alongside the EEG data sent to a recording computer. It uses no special protocol, it just sets the data line to the byte to be sent for a brief duration and returns all of the lines to low. Further research showed that this is often the case, and checking for responses from a receiving device (such as part of standard parallel protocol) is normally overlooked to save time. For this reason, the

parallel port was developed into a female unidirectional port that takes incoming bytes from the other ports and puts them in parallel over the data line.

Once again, a function was created to initialize the parallel port. All lines of the standard parallel port protocol were appropriately set to input and output lines, although this wasn't completely necessary for the setup described. In fact, all serial and TTL ISRs had been programmed to send the received byte to the parallel port using SPP protocol, but because there was no acknowledgement from the amplifier, the protocol implementation added no further function.

The idea was to further develop the port to act as a computer would, automatically detecting the type of protocol the receiving device was capable of. However, this had the problem of not being able to be the slave from another computer. This could be accomplished by switching the state of the port with a mechanical switch or possibly adding another port that is male, but such physical alterations would have to wait for the next generation of the device, and the current implementation was sufficient for the available setups.

To test the code, an event marker diagnostic that is part of the NeuroAmps Acquire software (which is used for EEG data collection). Bytes were sent from a computer to one of the serial ports that relayed the byte to the parallel port. At the end of the other ISRs (including whichever was sending the byte), the data line would be set to all zeroes, to terminate sending of the byte. The code runs at a speed that, with the brief time that the byte is on the data line, the amplifier being used to test the port rarely saw the byte. To solve this, code was added that pulled the data line low every 5 milliseconds.

That way, nothing would be affected if the code was interrupted while counting to 5, but the parallel port would remain high for 5 milliseconds, and if the event marker isn't picked up, it is due to the amplifier's sampling frequency of the event markers and not the fault of NeuroHub. When tested alongside EEG data recording, the markers show up as the decimal equivalent of the hexadecimal value that represents the byte sent.



Figure 14: Second generation NeuroHub assembled.

Development of the third generation of NeuroHub was started, and the plan is to implement Ethernet capabilities as well as SD card data logging capabilities. For future plans for the device, please see the future development section in chapter 6.

Chapter 4: Testing and Validation

The first tests to assess system characteristics were completed using the first generation board and custom timing software. Upon reviewing the initial results, it was realized that they have much more to do with how the computer handles protocols than lag resulting from the device. To acquire actual lag time produced by the device, an oscilloscope was used. Those results are presented first. A full set of tests was then completed to address the protocol lag problem and expose its weakness, which the results are then presented afterwards. First, however, the testing setups are presented.

4.1 Reliability and Variability Testing Setup

During the development of the device, performance tests were devised to determine the properties of the system and how it performs in a setup, specifically studying lag time. As defined by the design requirements, one of the most crucial features is extreme precision of the timing of the system. This equates to minimal lag time. It is important to test the device over many iterations, for experimental protocols can be lengthy and the device cannot lose precision over time. As doing this testing manually would prove to take an unreasonable amount of time, applications were developed to automate the testing and record the results. The data was then statistically analyzed to produce metrics that represent the system. As it turned out, the information acquired about lag time had more to do with variability of the computer hardware/software setup than of the device. This, along with the knowledge that the device has a very consistent

1.020 millisecond lag time as measured with the oscilloscope, further proves the need for the device in these setups.

4.2 Program Design and Development

The initial purpose of testing was to determine the reliability and speed of the setup. It was to be tested to be sure that the bytes being transmitted were the same as those being received and in a reasonable amount of time. The program was to run in every possible configuration, then compare the variations in outcome. The various setups included from each port to other ports. The tests were to be run over many iterations, to show the mean lag time and standard deviation over many tests. While testing, it became obvious that the differences in the computer's setup had a great effect on lag time, so these differences were tested as well. To fulfill these testing requirements, specialized timing programs were developed in C#.

The language chosen for the development of the testing program was C#. This was chosen for its simplicity in implementing the necessary features, for development time was intended to be minimal and C# embodied the low-level/complexity tradeoff required for such a program. It contains the necessary functions for sub-millisecond timing accuracy and easy access to serial ports. Microsoft Visual Studio was the chosen development environment. To make the testing setup as simple as possible, a basic graphical user interface (GUI) was developed. When the program is started, it checks for available COM ports on the computer and presents them in two drop down selection boxes, one to send and one to receive. The user also defines the length of the test in

iterations of bytes to send and receive. Two buttons were included: one to start recording, and one to exit the program. When the “Start “button is pressed, the selected COM ports are opened with a baud rate of 9,600, that which the device operates at, and all buttons and selection controls disabled.

To prevent other processes from interrupting during testing, the `ThreadPriority` and `ProcessPriorityClass` properties were set to the highest value. To stabilize the CPU cache and pipeline, a warmup of 1,500 milliseconds was used. This significantly improved the results of the first few iterations, which otherwise were unusually higher than the rest of the data. A system diagnostic tool, the `Stopwatch` class, contains methods to provide sub-millisecond timing metrics, and is part of the `System.Diagnostics` namespace. This was chosen for its robust feature set and ability to count at a high temporal resolution. It has the ability to measure in clock ticks, therefore providing the most accurate resolution possible. At the start of the warmup, the stopwatch object is reset, then started and run until its `ElapsedMilliseconds` property reaches 1,500. The actual testing is then begun.

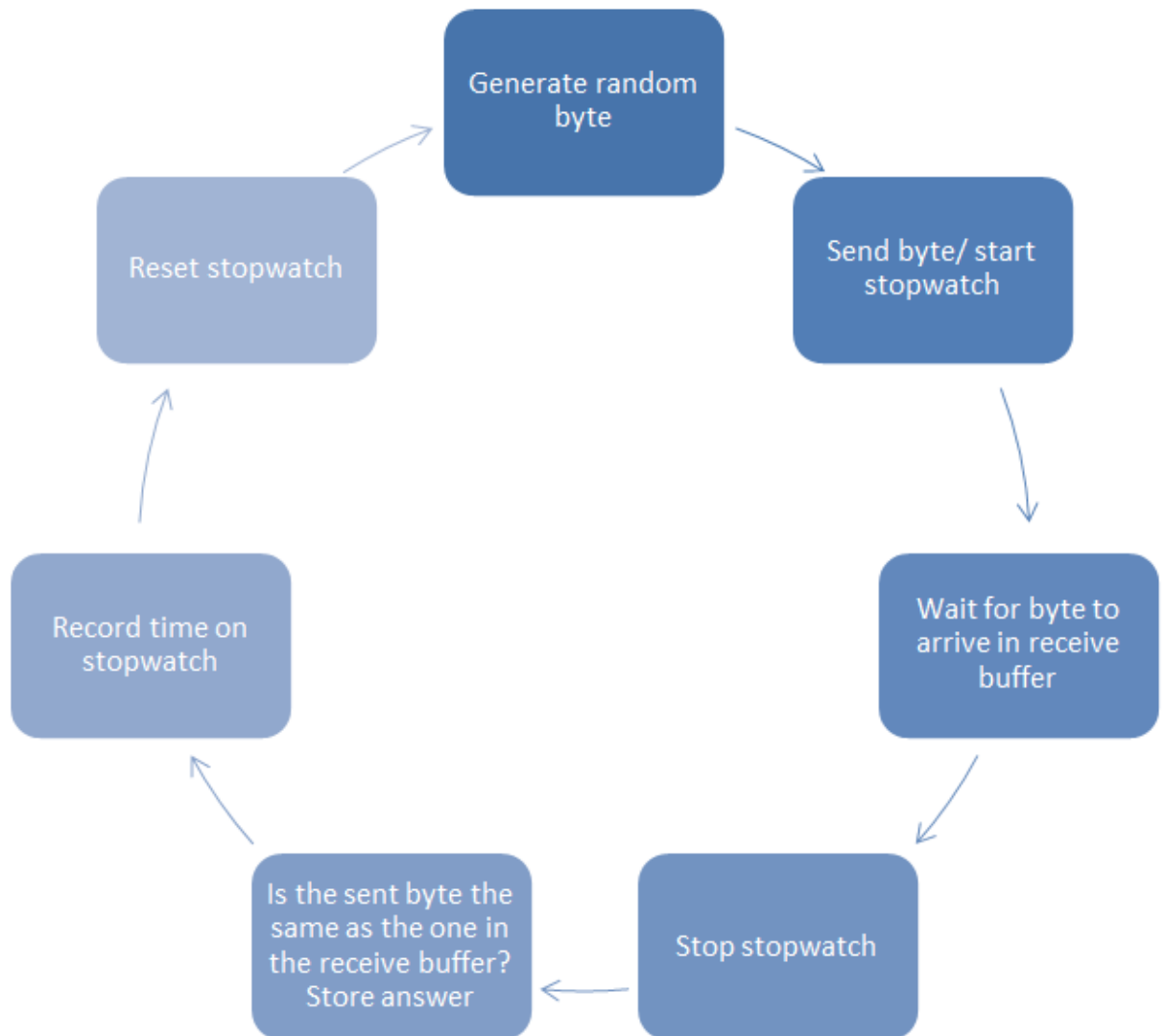


Figure 15: Timing program process loop diagram.

Whatever value was set in the “Iterations” input box is compared to a counter in a for loop. Inside the loop, a random byte is placed in an array, after which the stopwatch is reset and started, then the byte is send out over the selected “COM Out” port. The program then waits until there is an incoming byte to read on the “COM In” port, stops

the stopwatch, and checks to see if the random byte that was sent out was properly received. StringBuilder was used to store the data as it is being recorded. This was preferable to string concatenation because it is an intrinsically mutable string class, whereas string concatenation is actually making a new string every time something is added to it, which takes significantly longer. The data being recorded is also displayed in the GUI's textbox while it is recording for debugging purposes.

Because there is not a simple TTL out setup on most PCs, a custom setup needed to be devised to test the BNC port on the device. The first attempt at a solution used automated triggers to control COBI studio, the program used to interact with and record data from the fNIR system, which has the ability to be controlled by triggers sent via a serial port. It also has the ability to record other incoming bytes to a marker file with timestamps attached. These features were exploited for the custom timing setup. One such trigger to control the program is byte 251, which initiates baseline. There are two other triggers, 253 for starting recording, and 254 for ending recording, but when recording has ended, the program also stops the current experiment, which would not work in this case, as 1,000 measurements needed to be made at a time. Typically, as soon as baseline is complete, recording begins, but there is a feature in COBI studio to turn that off.

A C# program was written to send the baseline byte every 11 seconds, to allow time for the baseline reading to be taken and restarted. The TTL pulses from the fNIR device goes high when the baseline is started, then low again 150 milliseconds later, then high again when the baseline is completed, and low, once again, 150 milliseconds later.

Since the NeuroHub device was designed to send the character “1” (byte 45) when the line goes high to 5V, and character “0” (byte 48) when the line goes low to 0V, each baseline recording supplied 4 marker timestamps. As such, 250 baseline measurements were taken per testing session, for a total of 1,000 measurements.

These markers were routed back to COBI studio to be stored in the marker file. Because COBI studio listens to only one COM port, both the baseline trigger and the TTL high and low markers were sent through the NeuroHub device, through serial port and BNC connector, respectively. A serial cable was also connected to the computer running COBI studio, so it was able to listen to both the testing program running on one computer and the TTL pulses coming from the back of the fNIR device. This shows how NeuroHub makes setups simpler by overcoming the limitations of available testing software. As in previous tests, the resulting marker file was imported into MATLAB to be analyzed.

After the data had been acquired, there was found to be a much higher standard deviation for the TTL pulse setup than the serial setup. Because there were so many points in the testing setup that could have added lag time that were out of my control, simplification of the setup was required to remove other variability.

The 5 V digital pins on an Arduino are the right level and relatively easy to control, so a script was written in the Arduino IDE to fulfill this purpose. It pulls the pin high for 25 ms, then pulls it low for 25 ms. The C# timing program was modified to be used to with this setup. It listens to the COM port, and makes sure that it is receiving “1”

then “0” byte, as well as recording the time between them (which should be 25 milliseconds).

4.3 Data Analysis

To describe the characteristics of the device in a clearer way, a MATLAB script was used to perform calculations and display the information. Because the testing program exported a tab-delimited text file with ticks in the first column, a Boolean value representing if the byte received was the same as the one that was sent in the second, and milliseconds calculated (ticks/frequency times 1,000, frequency is in ticks per second) in the third, the values were easily imported into MATLAB. It then created relevant graphs and calculations to describe the results of each test.

All ten tests in the session were stored in one folder with the script. The script started by clearing all data in the workspace and provides a location to enter the identification of the test session in the same folder (for example, computer 1, running on Windows 7, using USB ports, serial port 1 to serial port 2). The workspace was then saved in case it was to be used later. All data was loaded into easy to read matrices, separating all tests in the session into different columns. Two matrices were created, one for the lag time calculated in milliseconds, and one with all of the Boolean success values.

The success values were first checked to be all true, as any incorrect byte would constitute a failure and would not be acceptable. 2 arrays were then created to find the mean of each test and the standard deviation of each test.

The next part of the script created plots of the data. It started by plotting each of the tests in the session on subplots, the x-axis being the iteration, and the y-axis being the lag time of that iteration. The title of the plot, as well as all generated graphs, was created using the identification string at the beginning of the program for ease of labeling. This plot was stored, then another plot was created using all data points in the session.

After the plots were completed, the script created histograms. The histogram is meant to show the probability that the lag time would fall within a certain window after the byte was sent. This gives a clear picture as to what the typical response time is. Like with the plots, the histograms were made first on a subplot for each test, then in one big histogram for all tests in the session. Also like the plots, the histograms were automatically saved in the same folder. A bar graph was then created which showed the mean lag time of all the tests in the session as different bars, with error bars showing standard deviation of each test. The standard deviation between the all tests in the session was found, as well as the mean of all iterations in all tests in the session.

The script was copied to each folder with data and the identification of the test was changed accordingly. All tests were run and their results were stored, either automatically for the figures, or copied to an Excel spreadsheet for all other calculated values, for easy reference. The data was analyzed the same way.

4.4 Results

All tests were determined to be successful, as the success rate for correct byte transmission was 100%. This is necessary, as any unsuccessful transmission would result in inaccurate event marking, potentially ruining the data.

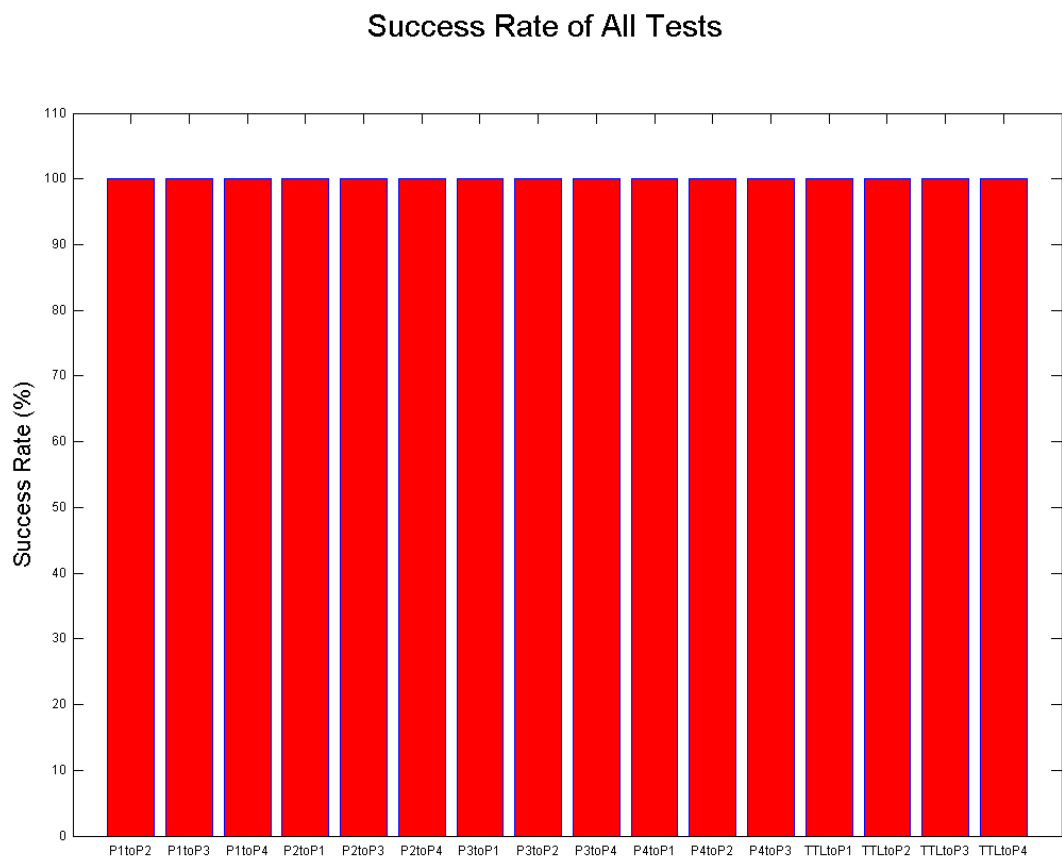


Figure 16: Successful byte transmission in the tests.

The rest of the results will be arranged by testing purpose, according to comparisons that seemed most necessary. The specifications for each computer are also included. The most important results are included here; please refer to appendix D for all results.

4.4.1 Oscilloscope Measured Lag Time

By the time the testing of the first generation by the testing program was completed (again, presented after this section), the second generation of the board was ready. All testing was immediately switched to generation two. The first tests' results showed little about how the device responded and more about the flaws of using non-embedded systems for high temporal precision brain computer interfaces and experimental protocols. Because of this, testing was moved from custom timing programs to the oscilloscope. The setup for testing each port was conducted and the results were very consistent.

On the oscilloscope used, there were two BNC connections for the two available channels. One computer was used to send repeated bytes over a serial connection, while another was used to acquire a screenshot of the oscilloscope. To measure the lines going in and out of the device, only the appropriate lines were wired to both the data stream and the oscilloscope channels, i.e. the Tx and Rx lines, as well as ground. The following is a screenshot of the oscilloscope measuring lag time of serial data transmission. The lag time was found to be negligible.

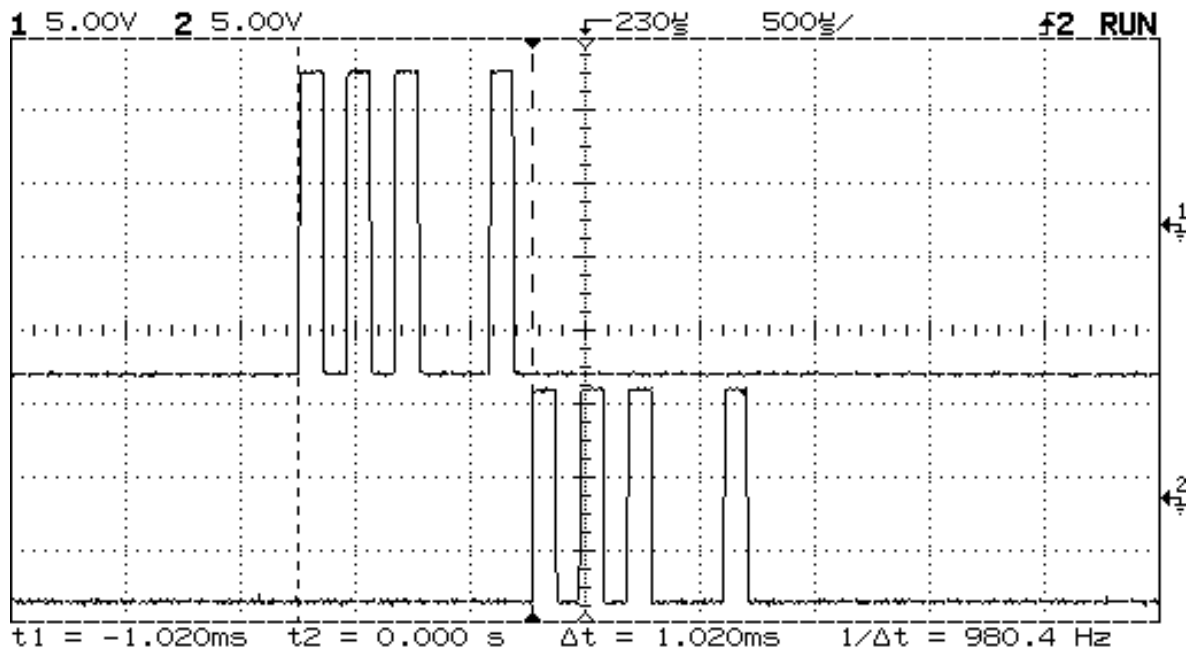


Figure 17: Oscilloscope screenshot of one byte transmission. The first (top) line is the input signal (receiving Rx line), the second (bottom) line is the signal being sent from NeuroHub (all other Tx lines).

The results were very precise, with a consistent delay of 1.020 milliseconds, for the entire duration of testing and at all ports. Even after a considerable length of time (about 30 minutes) of constantly receiving and transmitting bytes, the lag time remained the same. As is seen in figure 17, the device waits until it has received the entire byte before sending it to the other ports. There is negligible delay from after having received the byte to sending it out. In the firmware, writing the data to all of the other ports comes after the parallel port, so the fact that testing the parallel port on the oscilloscope isn't

possible (there are 8 parallel lines and only 2 inputs on the oscilloscope) was of no negative consequence because it is guaranteed to be faster.

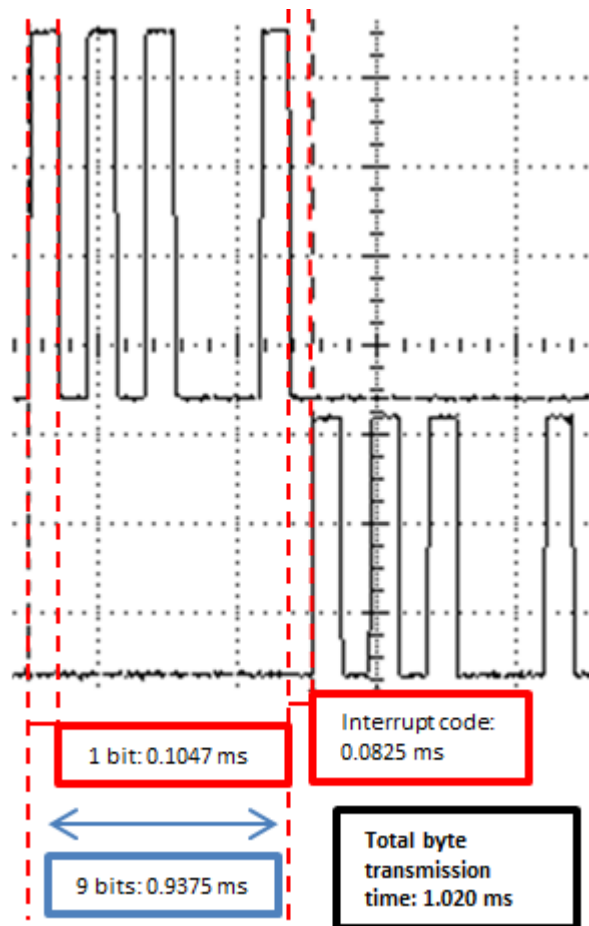


Figure 18: Byte transmission timing explanation, at 9,600 bits per second.

The lag time is mostly the time it takes to send a byte serially at a baud rate of 9,600 bits per second. This was the baud rate used because it is the standard default on

most systems, but if needed the code for NeuroHub could easily be changed to accommodate another baud rate. Also, because the parameters of the UARTs can be individually set, they could easily be set for their own settings and NeuroHub act as a baud rate converter. With a higher baud rate, the lag time would be shorter, but this increases risk of faulty byte transmission. At a rate of 9,600 bits per second, each bit is held on the line for 0.1047 milliseconds. With the start bit, there are 9 bits to send, totaling 0.9375 milliseconds. After the USART completes reception of the byte, it causes the microcontroller to jump to that particular USART's interrupt code, which then takes time to run. The lag time was measured to be 1.020 milliseconds and 0.9375 milliseconds were needed to receive the byte, which leaves 0.0825 milliseconds to carry out the interrupt code. The microcontroller's clock is running at 16 MHz (or 62.5 nanoseconds per cycle), so the interrupt code takes 1,320 cycles to complete.

4.4.2 Serial to Serial and TTL to Serial

The first computer tested on was a Sony Vaio laptop. Under the hood, it has a 64 bit Intel i5 CPU running at 2.53 GHz, with 4 GB RAM. This computer was used to test all serial ports to each other, to assure consistency across all ports. The computer had no native serial ports, so serial to USB converters were used. The results across all ports were very similar, and so only port 1 to port 2 and TTL to port 1 are included here. See appendix D for the rest of the testing results.

First shown is serial transmission from serial port 1 to port 2. The first graph shows the mean lag time of each test with standard deviation bars. The tests were

completed at different times, and the results were fairly consistent. The second graph shows a histogram of all iterations of all tests from port 1 to port 2 for a clear visualization of the data. All charts show consistent results, as do the results from the other ports. The mean lag time was 2.788 ms, with a standard deviation of 0.250 ms, and a standard deviation between tests of 0.033 ms.

Mean Lag Time Across All Tests, Computer 1, Windows 7, USB, P1 to P2

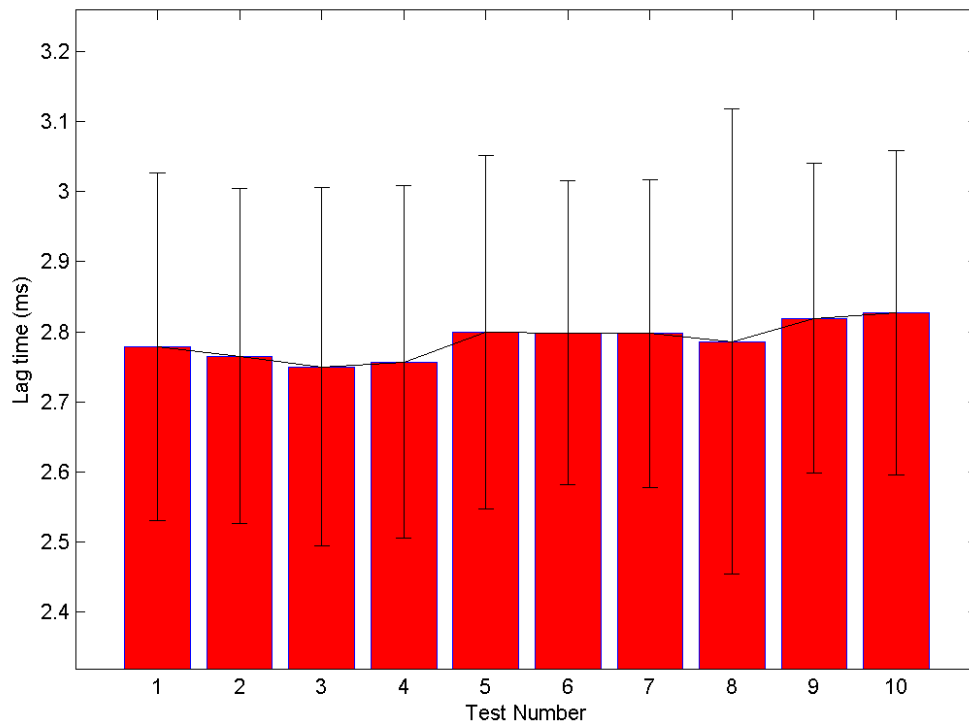


Figure 19: Bar graph of serial port 1 to serial port 2 mean transmission time, as determined by the timing program, with error bars.

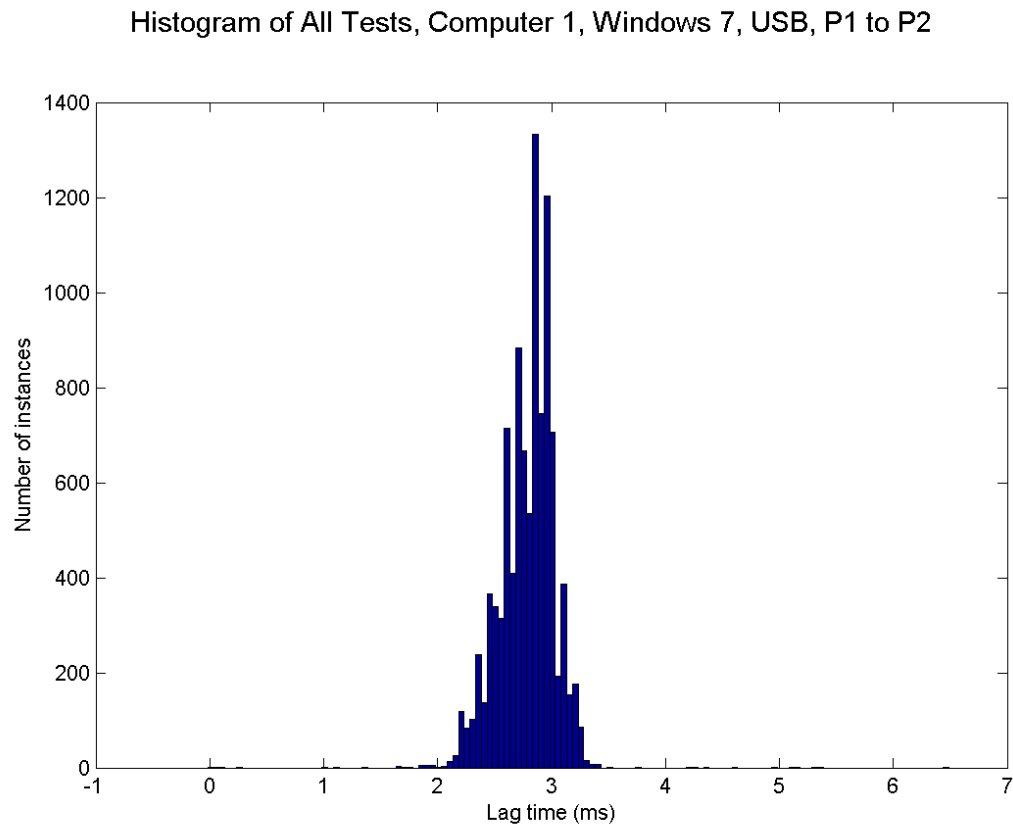


Figure 20: Histogram of transmission times from serial port 1 to serial port 2, as determined by the timing program.

The following results are from the BNC port to port 1. It has a consistently slightly higher lag time, which was unexpected, but once again is likely due to the testing setup rather than the device itself. The mean lag time was 3.525 ms, with a standard deviation of 1.449 ms, and a standard deviation between tests of 0.329 ms.

Mean Lag Time Across All Tests, Computer 1, Windows 7, USB, TTL to P1

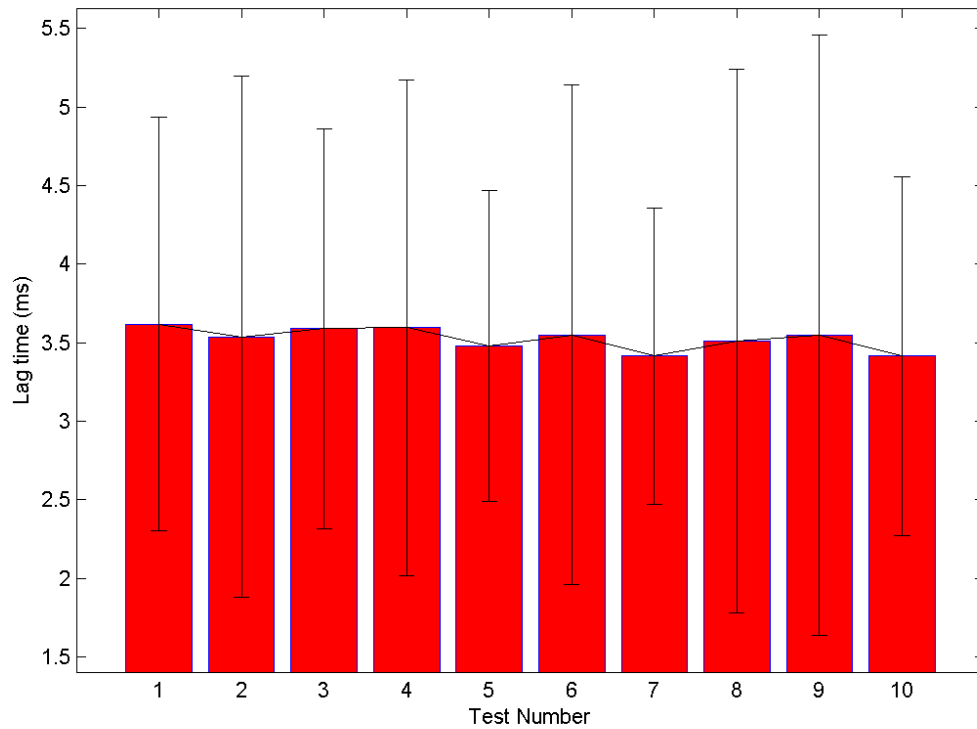


Figure 21: Bar graph of TTL to serial port 1 mean transmission time, as determined by the timing program, with error bars.

Histogram of All Tests, Computer 1, Windows 7, USB, TTL to P1

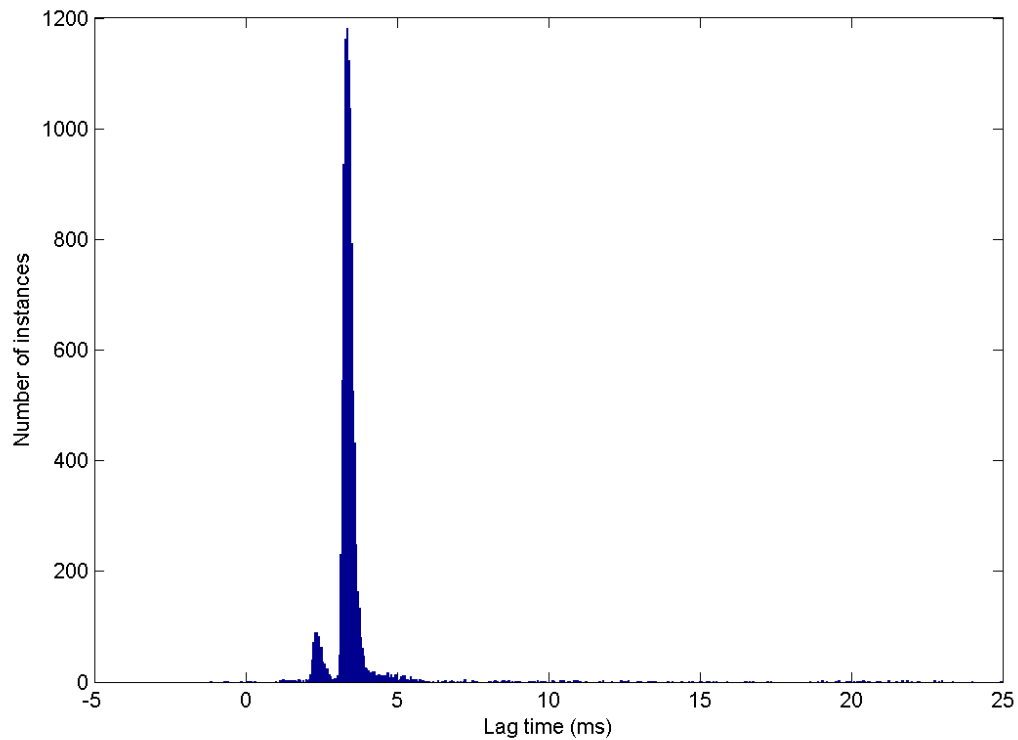


Figure 22: Histogram of transmission times from TTL to serial port 1, as determined by the timing program.

In conclusion, the test results from TTL to serial port 1 were much more varied than those from serial port 1 to port 2. The original TTL to serial test results are not included because they were entirely too variable. This testing setup removed most of that variance, but it is suspected that much of the inconsistency is again due to the testing setup, as an Arduino and a different timing program was used. That is not to say that the

Arduino is or the timing program itself were inconsistent, rather that the introduction of different testing variables could create unknown timing inconsistencies.

Table 1: Mean and standard deviation of lag time from all tests from all serial ports to other serial ports and TTL to all serial ports and the standard deviation between individual tests.

	Mean (ms)	Standard Deviation (ms)	Standard Deviation between Tests (ms)
P1 to P2	2.788	0.250	0.033
P1 to P3	2.681	0.316	0.028
P1 to P4	2.812	0.252	0.042
P2 to P1	2.828	0.248	0.045
P2 to P3	2.782	0.295	0.051
P2 to P4	2.826	0.242	0.041
P3 to P1	2.817	0.270	0.053
P3 to P2	2.814	0.244	0.039
P3 to P4	2.816	0.251	0.043
P4 to P1	2.794	0.274	0.062
P4 to P2	2.819	0.270	0.066
P4 to P3	2.590	0.323	0.018
TTL to P1	3.525	1.449	0.329

TTL to P2	3.420	1.298	0.253
TTL to P3	3.375	1.175	0.281
TTL to P4	3.412	1.252	0.354

4.4.3 USB and Native Serial With and Without Additional Information Influx

These tests were intended to compare the use of a USB to serial converter to a native serial port, as the first computer didn't have a native serial port. Also, it is running Windows 7 as computer 1, but on different hardware. The tests on this computer had the most interesting results which further clarified the need for the device.

In this section, there are results of 5 different types of tests. While testing the USB port, it was noted that the lag times are significantly higher when the mouse was being moved. The mouse was a USB mouse, and was therefore inputting information to the USB port on the computer when it was moving. As such, comparing the tests moving the mouse and not moving the mouse would yield interesting results. To test if the lag was due to the fact that the mouse was also using a USB port, the two types of tests, moving mouse and still mouse, were conducted when the native serial ports were being tested. First examined is the serial to USB port with the mouse still. It appears that there are times that are preferred by the fact that there are histogram bins with high amounts of instances next to bins with nothing. The fact that the distribution isn't even close to being even is a result of the protocol and not random chance. The mean lag time was 5.860 ms,

with a standard deviation of 1.201 ms, and a standard deviation between tests of 0.172 ms.

Mean Lag Time Across All Tests, Computer 2, Windows 7, USB, Mouse Still

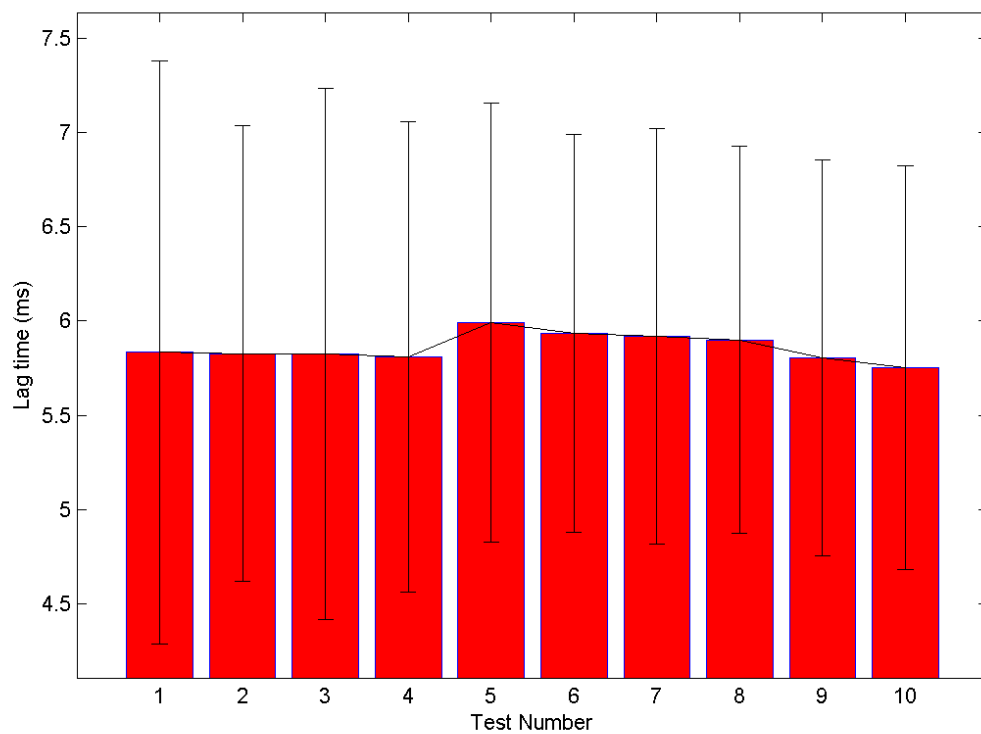


Figure 23: Bar graph of mean serial transmission time over USB ports without input from other USB ports, as determined by the timing program, with error bars.

Histogram of All Tests, Computer 2, Windows 7, USB, Mouse Still

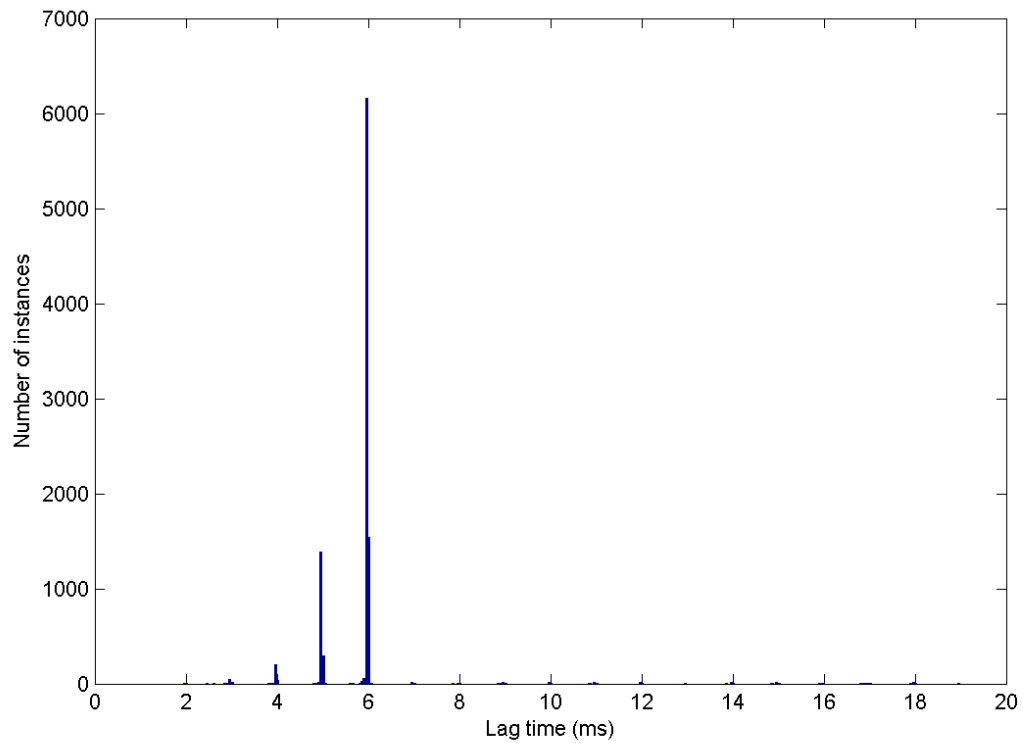


Figure 24: Histogram of transmission times over USB ports without input from other USB ports, as determined by the timing program.

The same tests were repeated, but this time while moving the mouse. There is significantly higher mean lag time and standard deviation. The variability in between tests could have been due to moving the mouse at different rates, but the conclusions drawn are the same nonetheless. The mean lag time was 13.27 ms, with a standard deviation of 5.768 ms, and a standard deviation between tests of 0.284 ms.

Mean Lag Time Across All Tests, Computer 2, Windows 7, USB, Moving Mouse

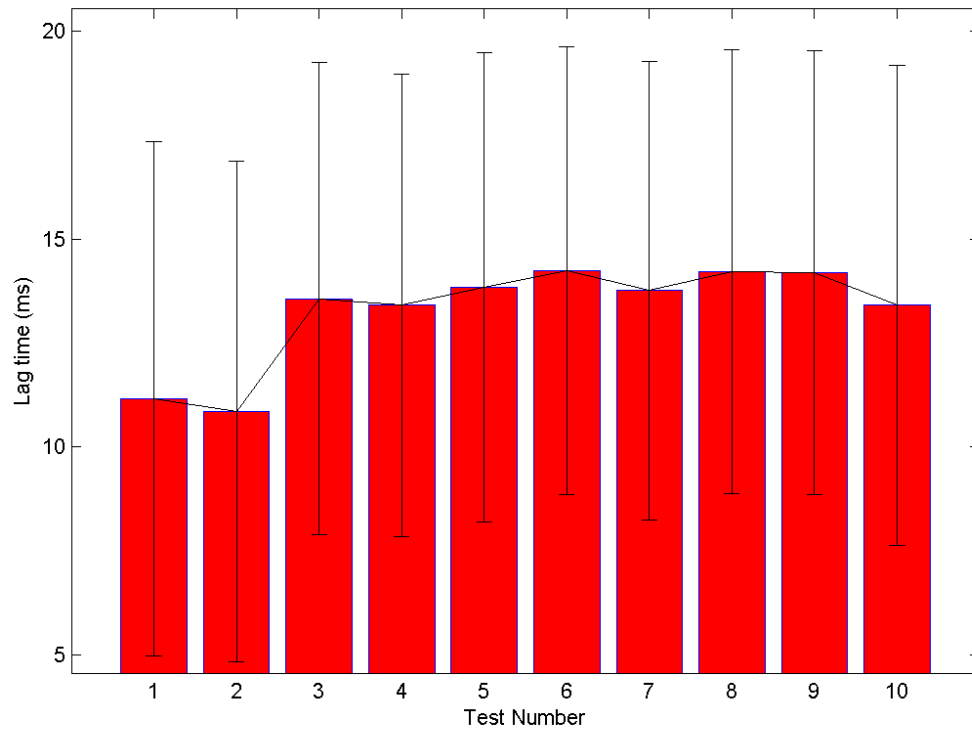


Figure 25: Bar graph of mean serial transmission time over USB ports with input from other USB ports, as determined by the timing program, with error bars.

In the following histogram, it appears that there are lag times that were preferred, specifically at 17 ms, which is unacceptable. This presumably had something to do with the other information being received at the USB port where the mouse was plugged in. This is the most glaringly obvious reason why USB ports should not be used in high temporal precision neuroimaging setups.

Histogram of All Tests, Computer 2, Windows 7, USB, Moving Mouse

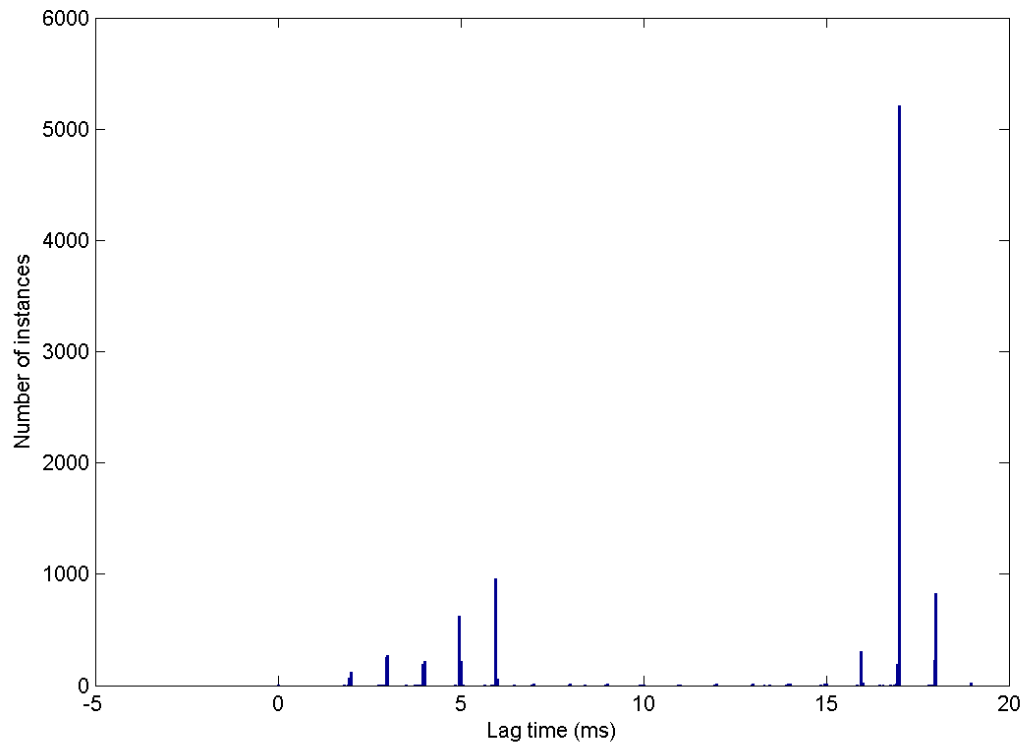


Figure 26: Histogram of transmission times over USB ports with input from other USB ports, as determined by the timing program.

The native serial port was much more consistent, even when information is being inputted into the USB port, pinpointing the extreme lag time problem to the USB port usage. The mean lag time was 7.054 ms, with a standard deviation of 0.023 ms, and a standard deviation between tests of 0.012 ms.

Mean Lag Time Across All Tests, Computer 2, Windows 7, Serial, Not Moving Mouse

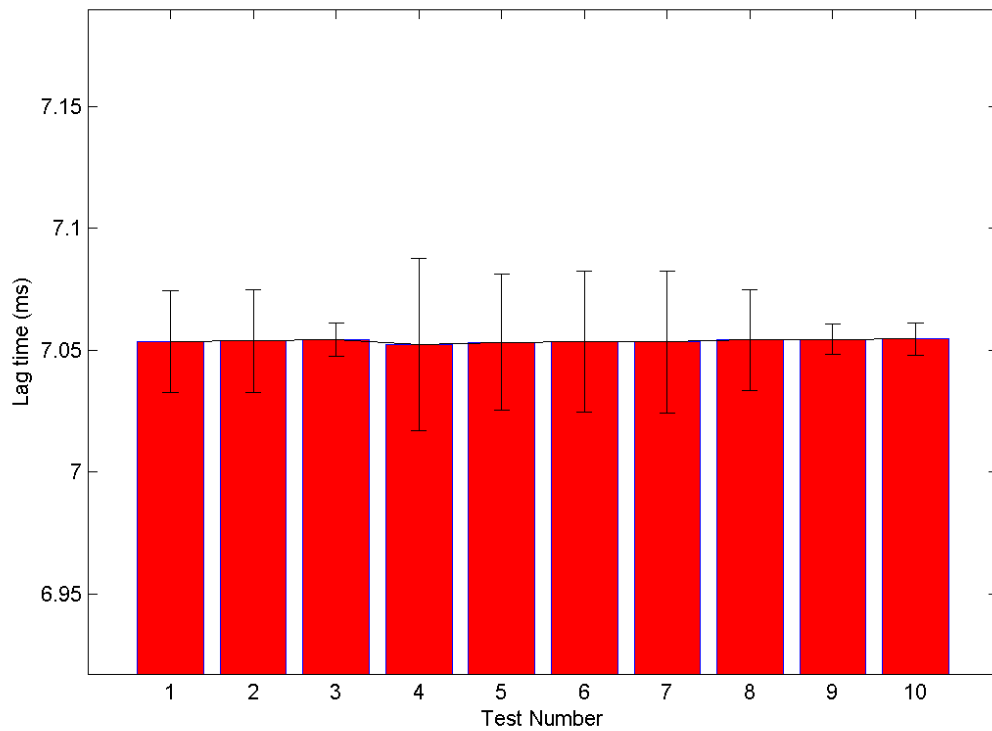


Figure 27: Bar graph of mean serial transmission time over native serial ports, as determined by the timing program, with error bars.

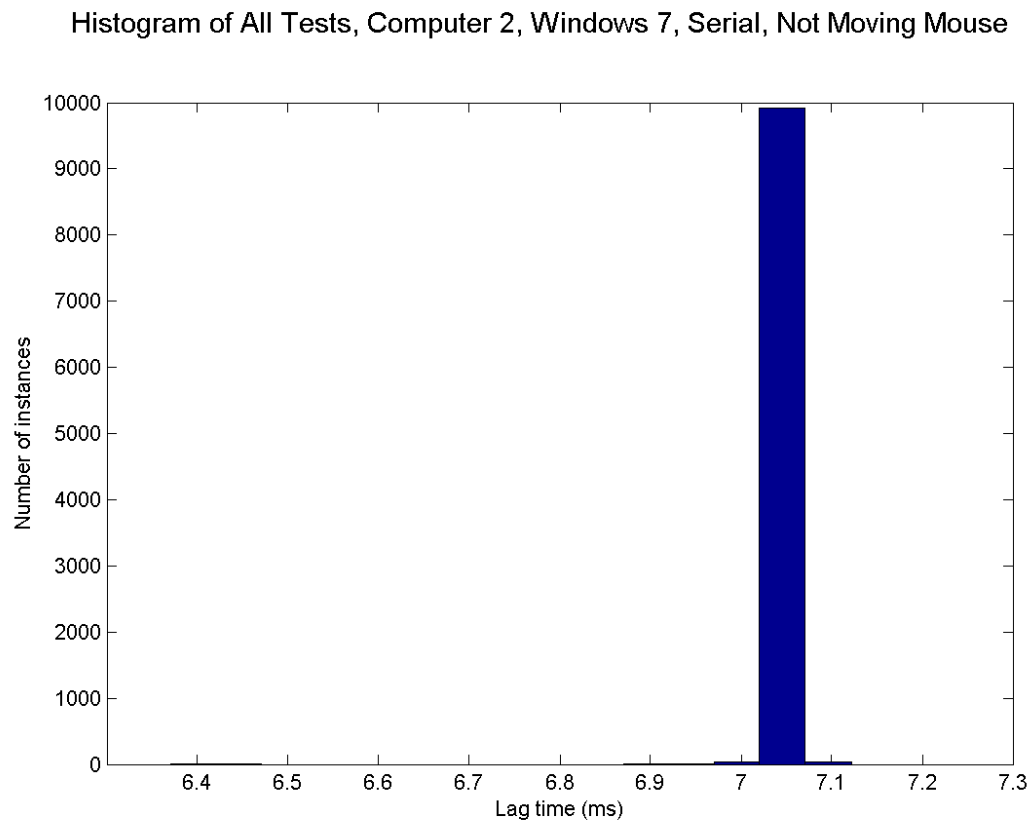


Figure 28: Histogram of transmission times over native serial ports, as determined by the timing program.

Very similar results were found even when the mouse was moving, unlike with the USB port. The mean lag time was 7.048 ms, with a standard deviation of 0.028 ms, and a standard deviation between tests of 0.006 ms.

Mean Lag Time Across All Tests, Computer 2, Windows 7, Serial, Moving Mouse

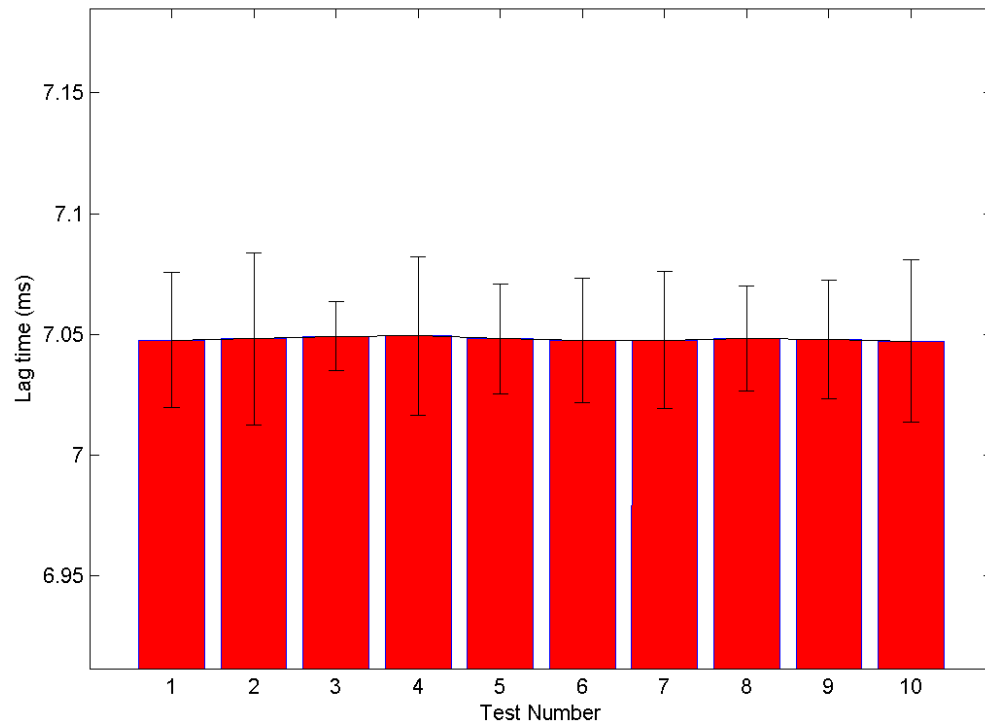


Figure 29: Bar graph of mean serial transmission time over native serial ports with input from non-native serial ports (as in other tests, with USB serial ports), as determined by the timing program, with error bars.

Histogram of All Tests, Computer 2, Windows 7, Serial, Moving Mouse

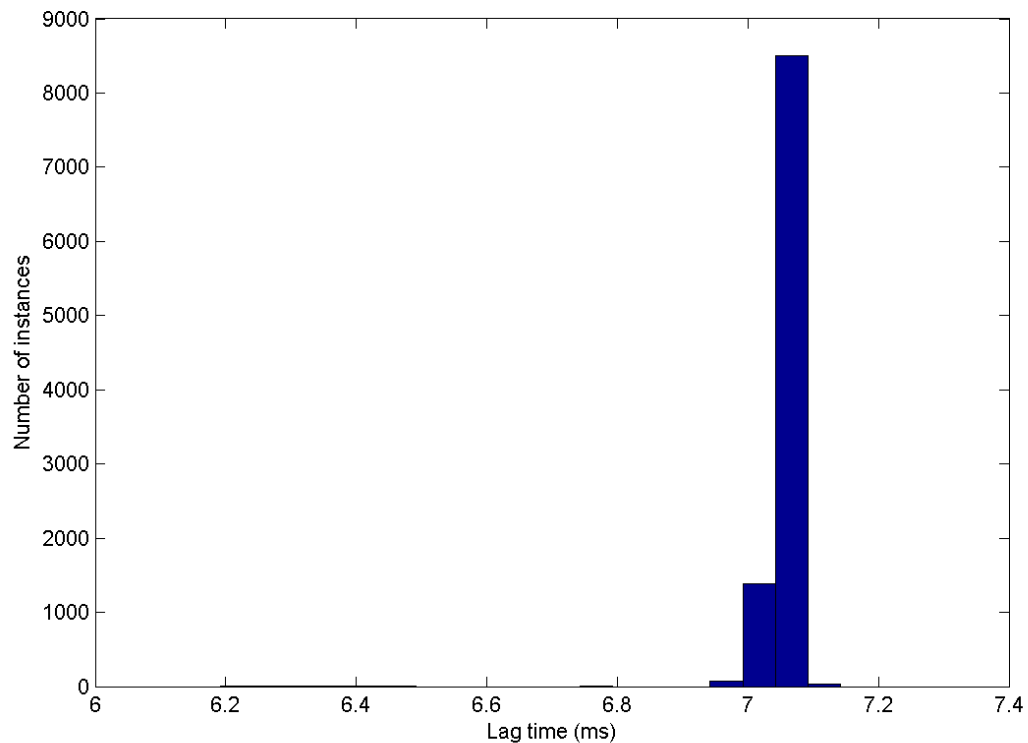


Figure 30: Histogram of transmission times over native serial ports with input from non-native serial ports (as in other tests, with USB serial ports), as determined by the timing program.

Here, the tests were completed on the native serial ports *without* the device. These results were also very consistent, at about 1 ms faster than the same port *with* the device. These tests were completed before oscilloscope testing, which also showed a consistent 1 ms lag time. This is when it was understood that the variance in the tests is due to the computer and not the device. The mean lag time was 6.044 ms, with a standard deviation of 0.023 ms, and a standard deviation between tests of 0.009 ms.

Mean Lag Time Across All Tests, Computer 2, Windows 7, Serial, No Device

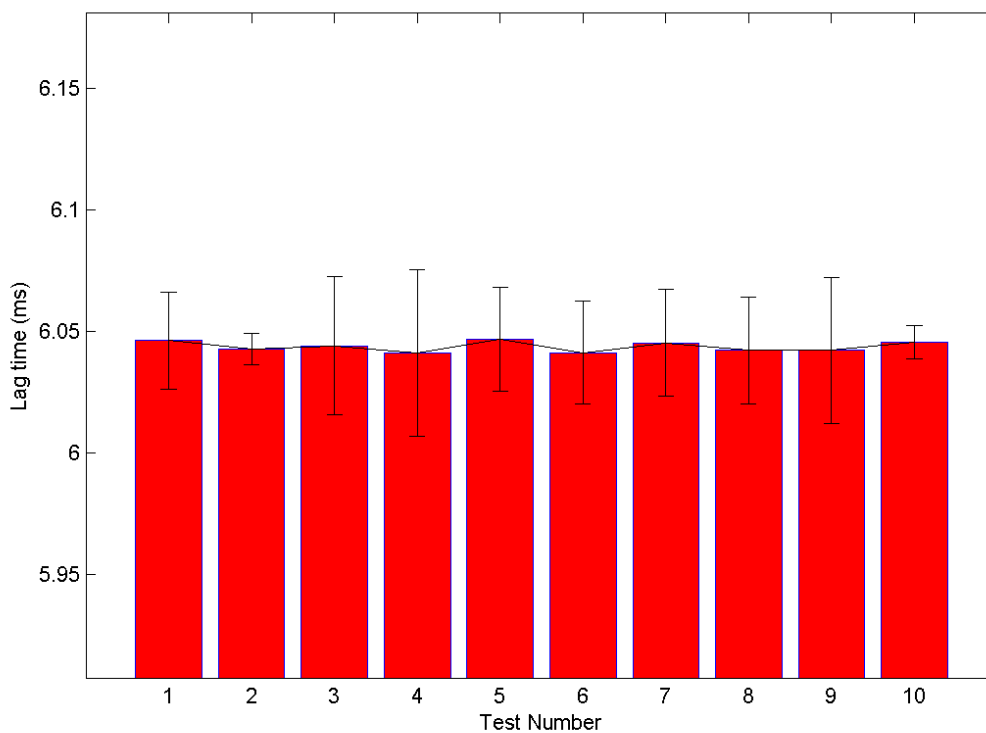


Figure 31: Bar graph of mean serial transmission time over native serial ports without the use of NeuroHub, as determined by the timing program, with error bars.

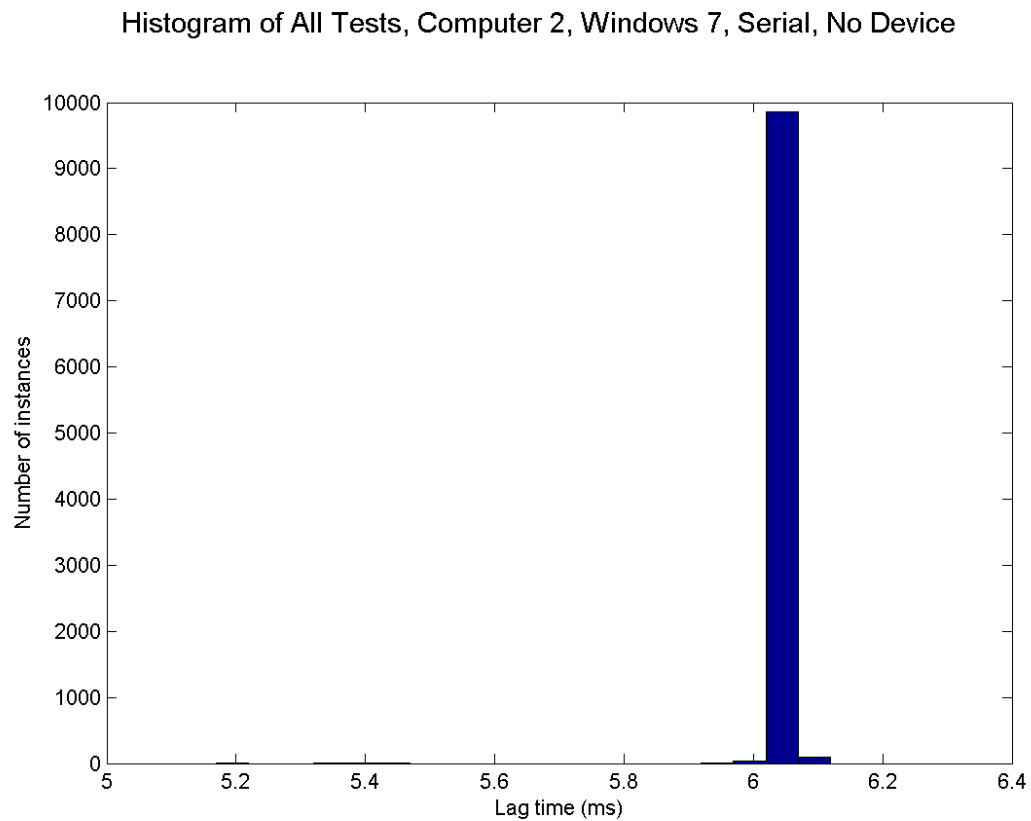


Figure 32: Histogram of transmission times over native serial ports without the use of NeuroHub, as determined by the timing program.

The tests performed on this computer demonstrated the unacceptable variability of lag time between protocols, especially USB. Testing without the device also yielded relevant results, as it showed that the device had little to do with the lag times that were being experienced in the tests.

Table 2: Mean and standard deviation of lag time from all tests in the configurations explained in this section, and the standard deviation between individual tests.

	Mean (ms)	Standard Deviation (ms)	Standard Deviation between tests
USB, Mouse Still	5.860	1.201	
USB, Mouse Moving	13.27	5.768	
Serial Port, Mouse Still	7.054	0.023	
Serial Port, Mouse Moving	7.048	0.028	
Serial Port, No Device	6.044	0.023	

4.4.4 Different Operating Systems on the Same Hardware

The purpose of these tests was to see the effect of the operating system used. The computer was running Windows XP when it was first tested, then the operating system was replaced with Windows 7 and tested again. The mean lag time for XP was 6.594 ms, with a standard deviation of 0.823 ms, and a standard deviation between tests of 0.219 ms.

Mean Lag Time Across All Tests, Computer 3, Windows XP, USB

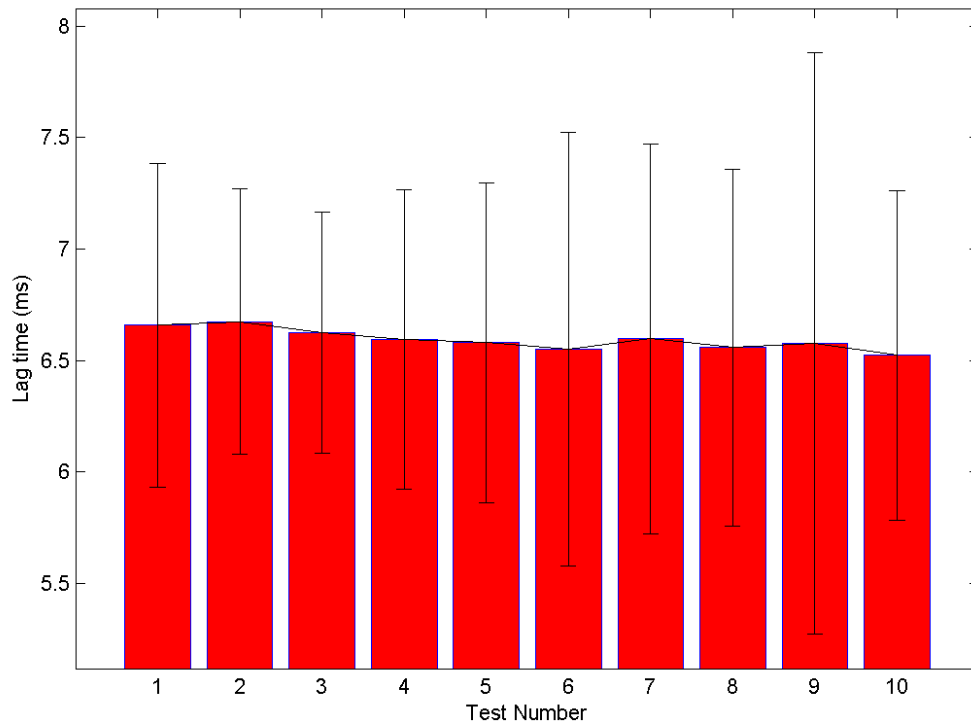


Figure 33: Bar graph of mean serial transmission time using Windows XP, as determined by the timing program, with error bars.

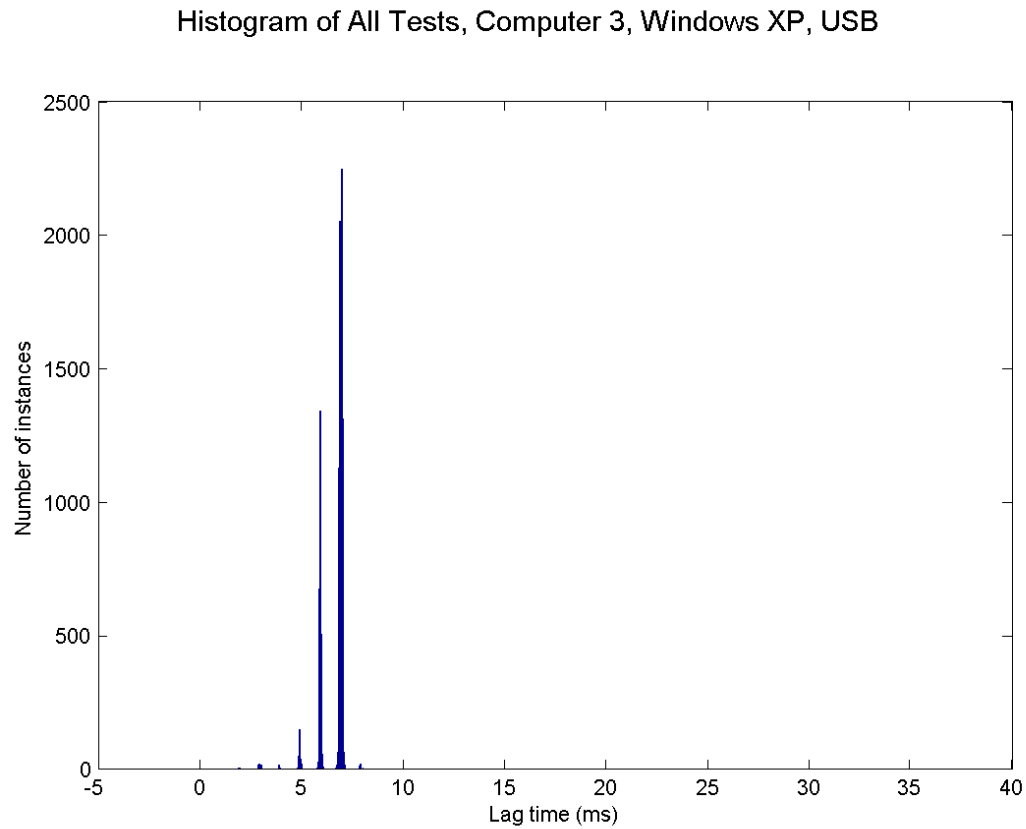


Figure 34: Histogram of transmission times using Windows XP, as determined by the timing program, with error bars.

Here, the operating system was changed to Windows 7 and tested again. The mean lag time was consistently higher using Windows 7, which was likely due to the fact that the same hardware was being used to run a more taxing operating system. It also appears that there was a greater difference between tests, indicating a greater variance from using the ports at different times. The mean lag time was 7.660 ms, with a standard deviation of 0.845 ms, and a standard deviation between tests of 0.1621 ms.

Mean Lag Time Across All Tests, Computer 3, Windows 7, USB

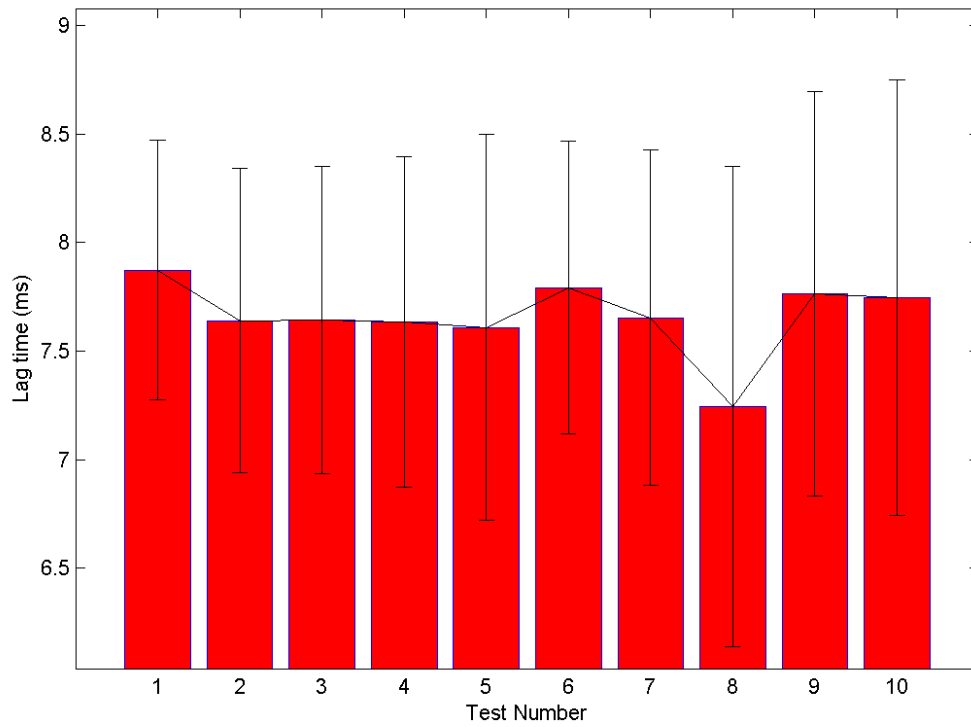


Figure 35: Bar graph of mean serial transmission time using Windows 7, as determined by the timing program, with error bars

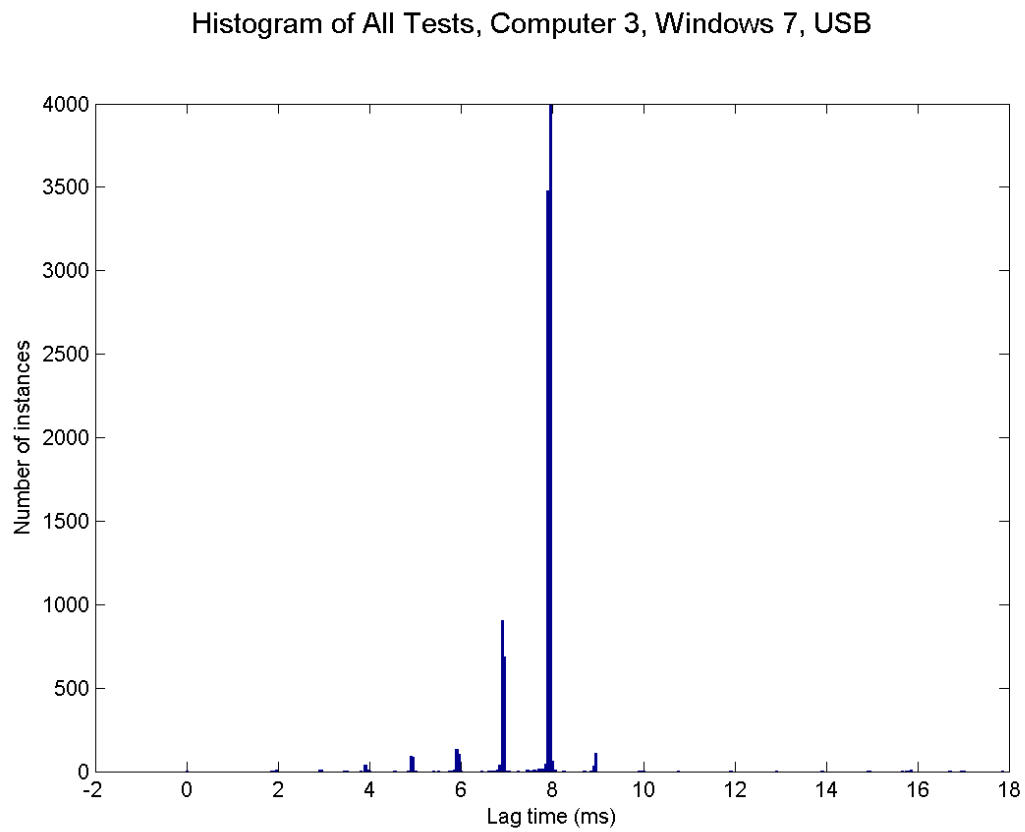


Figure 36: Histogram of transmission times using Windows 7, as determined by the timing program, with error bars

Table 3: Mean and standard deviation of lag time from all tests from all serial ports to other serial ports and TTL to all serial ports and the standard deviation between individual tests.

	Mean (ms)	Standard Deviation (ms)	Standard Deviation between Tests (ms)
Windows XP	6.594	0.823	0.219
Windows 7	7.660	0.845	0.162

4.5 Discussion

One of the main design objectives for this device was to alleviate the problem of different computers running on their separate clocks. The results from these tests opens discussion on using standard computer ports for scientific research.

The differences in testing results from system to system and the relatively large variances found in the data sets appear to show less about the device itself and more about the systems on which it was running. The device is an embedded system with a precise clock running at a high speed, and the computation needed for this program was not taxing the system heavily. With the exception of Windows Embedded Compact, made for use in embedded systems, no version of Windows is a real-time operating system. A real-time operating system operates as an embedded system does: timing is critical so it is put as top priority. All other versions of Windows are general-purpose operating systems, meaning they take their time to complete the time at task, and the time they take to finish the task is not important. For example, the time it takes for a program to open or complete an action in Windows is not important, but the timing of a brake system in a car is critical.

Serial ports are going the way of the dinosaur and are seen on less and less computers each year as newer protocols such as USB are on every computer. As these tests have showed, USB is unfit for temporally reliable event marker routing. Part of the USB protocol is a Start of Frame (SOF) packet identifier sent from the host every 1 ms to provide a time base. This explains why the lag times are on the millisecond, but not why

there would be so much more lag when one USB port was receiving information. At a sub-millisecond resolution modality, such as EEG, a lag time of 17 or 18 ms is a serious problem, and the variance in lag times is unacceptable. Native serial ports have a reasonable lag time, but faking one with USB converters is not advised. A better solution would be a custom embedded BCI system.

Chapter 5: Use Case Demonstrations

To demonstrate the utility of NeuroHub and its use in experimental setups, this chapter describes two different scenarios. Using the equipment in Drexel University's CONQUER Collaborative lab, multimodal and hybrid setups were tested with the device, and data was gathered from the devices for online classification as well as post analysis. These entirely different two experiment setups also highlight the flexibility of NeuroHub. Although future work will expand possibilities, this section aims to show that NeuroHub is useful in various setups and does in fact simplify and improve the experiment setup procedures.

The first scenario used to demonstrate the application of NeuroHub is a hybrid fNIR and EEG BCI setup. It aims to find a correlation between attention state and fNIR recorded hemodynamic changes to improve performance of a spatial navigation P300-response based EEG BCI [118]. NeuroHub coordinates event markers between BCI2000, the P300 stimulus software, NeuroScan Acquire, the EEG recording software, MazeSuite, the maze navigation software, and COBI Studio, the fNIR recording software. All but one of the ports were used making the otherwise difficult setup much easier to assemble and have the markers go where they need to go.

The second scenario was an unimodal setup that monitored different parts of the brain with multiple fNIR instruments to quantitatively compare the unconscious effort of the brain to understand various qualities of synthesized speech to natural speech [125]. This is particularly important for user experience research because synthesized speech is

more and more frequently being used on devices we use daily. Different devices were useful to use to measure a greater portion of the brain and define stronger neural correlates. The information from these devices needed to be aligned through a setup that sent different types of markers from the stimulus software to the two fNIR recording systems through two serial ports. With NeuroHub, the setup is made simpler by sending one marker through one port to both systems.

5.1 Use Case #1 – Multimodal Spatial Navigation BCI

5.1.1 Introduction

The first setup used demonstrated the application of NeuroHub to a hybrid fNIR and EEG setup. The experiment itself aims to find a correlation between attention state and fNIR recorded hemodynamic changes to improve performance of a spatial navigation P300-response based EEG BCI. NeuroHub coordinates event markers between BCI2000, the P300 stimulus software, NeuroScan Acquire, the EEG recording software, MazeSuite [17], the maze navigation software, and COBI Studio, the fNIR recording software. Here the experiment is presented in a simplistic fashion, in the context of shedding light on the usefulness of NeuroHub. Refer to the original paper for further details.

5.1.2 Background

As outlined in chapter 2, P300 based BCIs show the user a sequence of stimuli and the user is asked to attend to the stimuli and wait for a particular desired stimulus. If

the user is attentive, a noticeable positive response can be measured in the EEG signal about 300 ms after the stimulus. Typically, this involves a number of selections on a screen individually flashing randomly. As such, the user can select an object onscreen by counting the number of times it flashes during a set amount of time. By performing online analysis, the computer can understand which of the objects the user wished to select. This is dependent on the user being attentive, however, as lower performance is achieved if not. Performance results are determined by asking the user which icon they intended to select.

Also explained in chapter 2 is fNIR, an optical brain imaging technique used to measure changes in oxygenation of hemoglobin in the prefrontal cortex. Activity in the brain is linked with an increase in oxygenation, and therefore this technique is used to measure localized brain activity. This has been shown to be useful in a number of BCIs, and here is used to measure how attentive the user is to the stimuli being presented to them. In this experimental protocol, it is used passively to examine if there is a link between high performance accuracy of the P300 and oxygenation changes in hemoglobin in the frontal lobe. Detection of attention shift for increasing P300 BCI performance has also been demonstrated previously using EEG only [30].

5.1.3 Materials and Methods

In this experiment, the subject is seated in front of two computer monitors connected to two computers. One computer is used to display the P300 stimulus, record EEG data from the EEG amplifier, and provide online analysis of the data to determine

the P300 icon selected. The second was used to display a first person view of the maze and record the fNIR data.

The following diagram is representative of the setup without using NeuroHub.

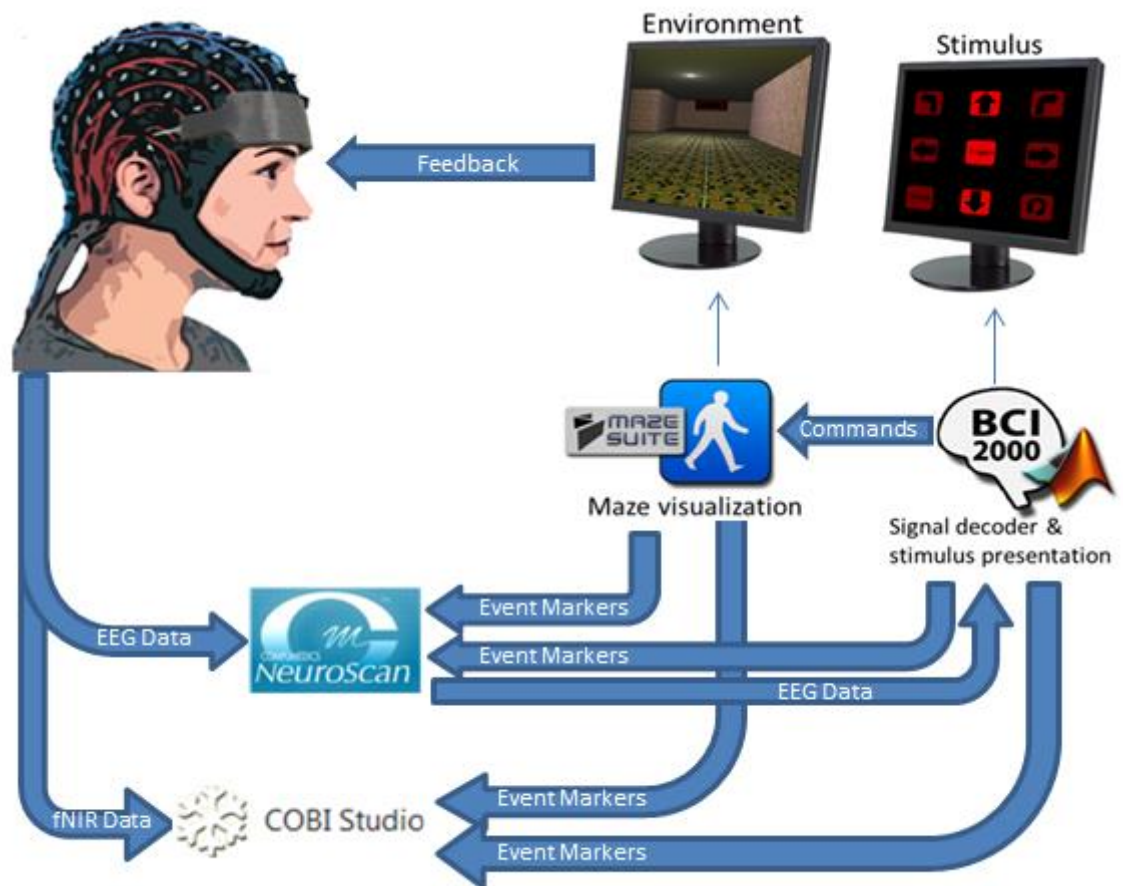


Figure 37: Diagram describing the flow of information in use case 1 without NeuroHub implementation.

Coordination of event markers between both computers and their software takes considerable effort. For example, MazeSuite only accepts event markers to control the maze by serial or TCP/IP, COBI Studio only accepts serial or parallel markers and sends only serial markers or TTL pulses for certain events, and NeuroScan Acquire only

receives data markers via a custom parallel port cable. In this experiment, all event markers should be available for all recording software for post-hoc analysis as well as controlling the MazeSuite software.

To accomplish this, NeuroHub was used. The EEG recording computer was connected to one of the serial ports on NeuroHub, while the EEG amplifier was attached to the parallel port. The second computer was connected to two serial ports, and the BNC connector on the back of the fNIR Imager was connected as well (although, in hindsight, this was unnecessary as fNIR was being recorded the entire time and therefore there were no TTL pulses on the line). In this way, all stimulus presentation markers were sent to both EEG and fNIR data streams, and control markers were sent to MazeSuite and COBI Studio. All event data was transferred to all locations using this very simple setup. EEG data was recorded from 9 locations at 250 Hz: FCz, Cz, CP3, Cpz, CP3, P3, Pz, P4, and Oz. fNIR data was recorded at 2 Hz.

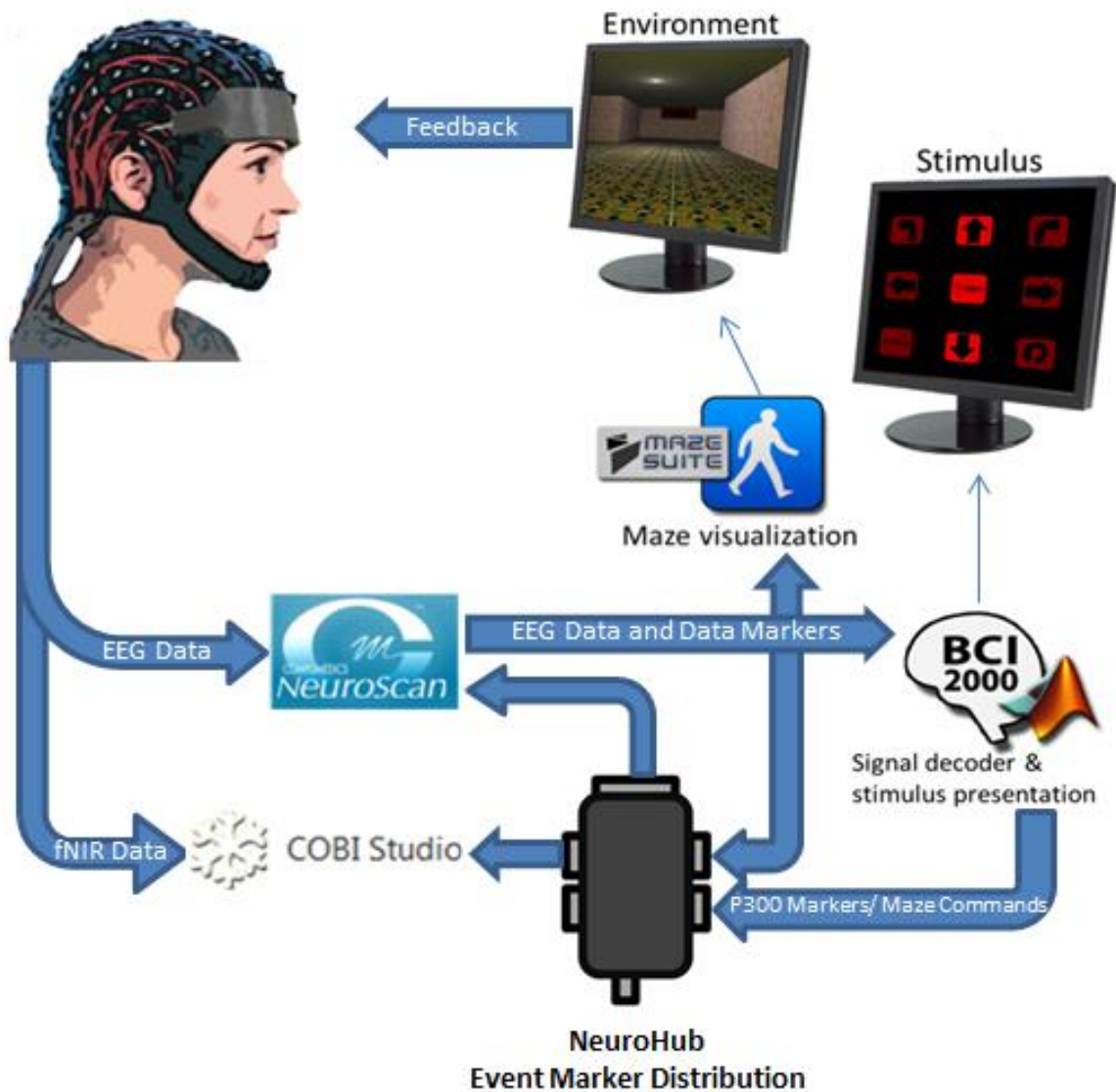


Figure 38: Diagram describing the flow of information in use case 1 with NeuroHub implementation.

The experimental protocol consisted of two parts. In the first part, EEG data was recorded to train the P300 BCI system. The subject was instructed to select a specific icon, and count how many times it flashes, then click, when instructed, on the icon that

they intended to select. This was repeated for 24 runs. In the second part of the experiment, the subject is asked to navigate to the exit of a series of mazes. The subjects were instructed when to look at the matrix and when to look at the maze navigation screen.



Figure 39: Left – the 3x3 P300 BCI matrix used. Right – the training manual selection screen. From [118].

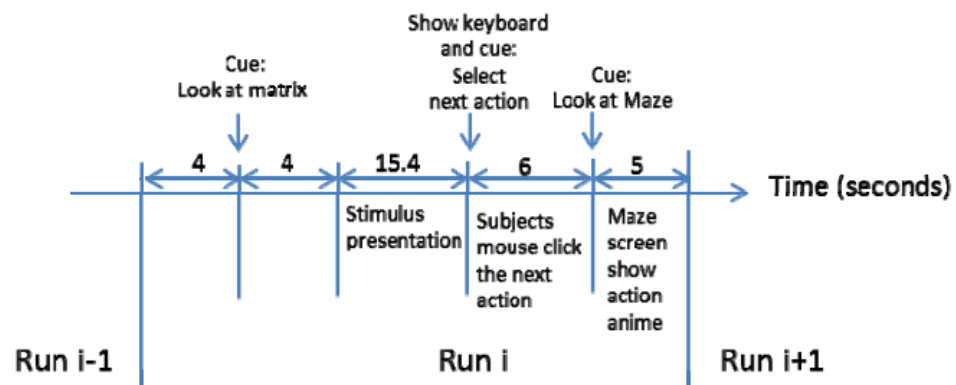


Figure 40: Time line for a run. From [118].

The EEG signals were bandpass filtered from 0.5 to 12 Hz and stepwise linear discriminant analysis, or SWLDA, was used to classify weights to predict the intended

target during on line analysis. fNIR data was lowpass filtered at 0.1 Hz for each run from 0 to 15 seconds, as well as using an artifact detection system (sliding window motion artifact rejection). Changes in oxygenated and deoxygenated hemoglobin were calculated from this data.

5.1.4 Results and Conclusion

The experiment was determined to be a success, because the system worked seamlessly and markers made it to all necessary data, control, and online analysis locations. The fNIR and EEG marker files show all spatial navigation markers. The user was able to successfully train the system and navigate the maze, but because the results from this particular setup were from only one subject, only a brief description of results from the original publication will be presented here for conclusiveness.

The following table shows the first 20 synchronization markers received, in decimal format and the timestamp (in seconds, since recording was started) that the device marked as the time the marker was received.

Table 4: First 20 synchronization markers from use case 1 experimental protocol. Showing byte received by EEG and fNIR recording systems, as well as the timestamp (from start of recording) and time from the first marker received.

EEG Data			fNIR Data		
Byte Value	Timestamp (s)	Time from first marker (s)	Byte Value	Timestamp (s)	Time from first marker (s)
95	0.376	0	95	90.089	0
77	5.316	4.94	77	95.019	4.93
67	38.012	37.636	67	127.714	37.625
77	41.724	41.348	77	131.432	41.343
68	74.432	74.056	68	164.139	74.05
77	78.136	77.76	77	167.841	77.752
66	110.832	110.456	66	200.53	110.441
77	114.536	114.16	77	204.248	114.159
65	147.228	146.852	65	236.929	146.84
77	150.964	150.588	77	240.662	150.573
72	183.604	183.228	72	273.304	183.215
77	187.332	186.956	77	277.037	186.948
66	220	219.624	66	309.706	219.617
77	223.728	223.352	77	313.43	223.341
67	256.432	256.056	67	346.144	256.055
13	260.124	259.748	77	349.828	259.739
66	292.796	292.42	66	382.502	292.413
77	296.54	296.164	77	386.236	296.147
67	329.22	328.844	67	418.927	328.838
77	332.924	332.548	77	422.629	332.54

To find the discrepancies in the timestamps, the difference was found between the amounts of time recorded from the first timestamps of each device. The mean discrepancy in time recorded by the recording devices was found to be 0.0136 seconds,

with a standard deviation of 0.0073 seconds. The following histogram shows the distribution of differences from the fNIR timestamp to the EEG timestamp.

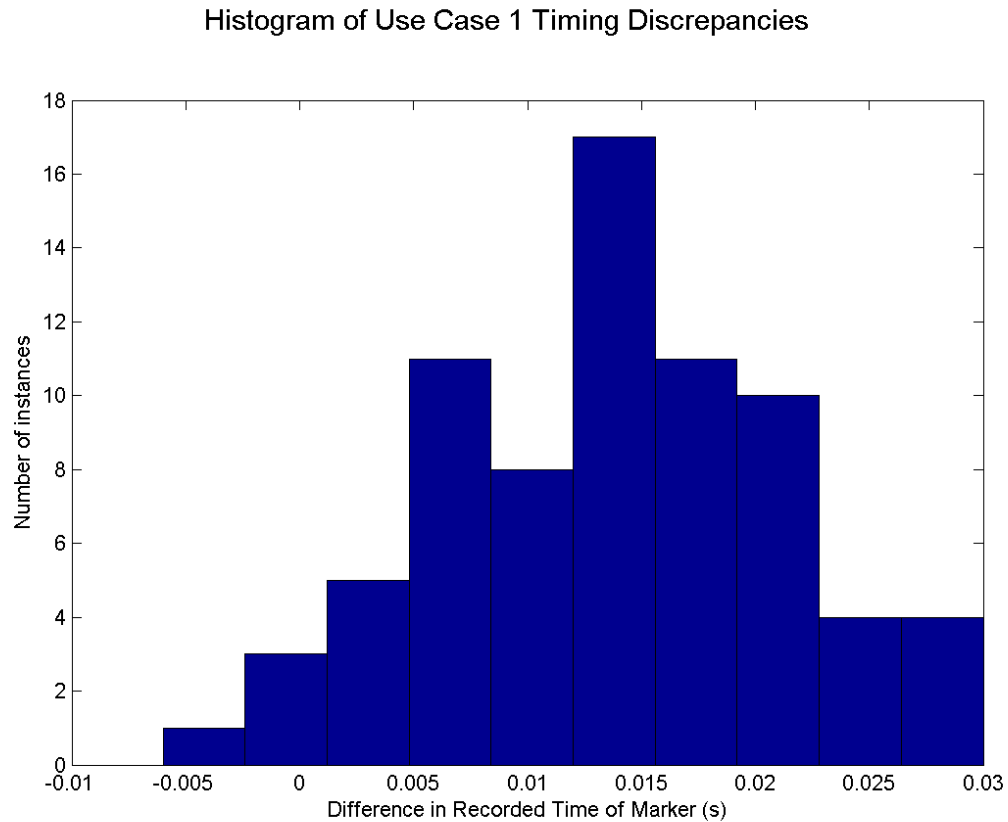


Figure 41: Histogram showing the frequencies of the difference in EEG and fNIR times from the first marker.

For both left and right hemispheres, oxygenated hemoglobin increases and deoxygenated hemoglobin decreases more for high performance runs than for low performance runs over the course of the stimulus presentation. The effect is stronger in

the left hemisphere. These results show that fNIR can be used as a predictor for attention in a P300 based BCI system. This could further be implemented into a sort of switch that turns off control if the user is not paying attention, rather than guessing at what is essentially noise for the desired target.

The purpose of this experiment was to demonstrate if fNIR data can be used to make a better P300 BCI. In the original publication, it is mentioned that more subjects and larger sample sizes would be needed for validation. NeuroHub made setup and accurate alignment of event markers simple as it was intended to do. If the experiment is repeated in future studies, NeuroHub should be used in the setup.

5.2 Use Case #2 – Synthetic Speech Perception BCI

5.2.1 Introduction

The second use case demonstration involved the implementation of NeuroHub into an experimental setup that aims to measure cognitive processing costs associated with synthetic speech perception. The setup is unimodal and involves event markers from a stimulus program presenting the sounds to 2 fNIR recording systems. The two systems have different requirements for receiving event markers, which originally were dealt with by sending the markers in their required formats over two separate serial ports. With the NeuroHub, they were able to be sent from the same serial port on the stimulus computer to two different fNIR recording systems that image different parts of the brain. Although precise event marker timing is not as crucial for fNIR as it is for EEG because of the slow

nature of the hemodynamic response, this is an example where NeuroHub makes the setup simpler.

5.2.2 Background

As synthesized speech more commonly becomes a part of daily life through the use of devices that need to speak a more widely varying vocabulary than would be practical to use all natural recordings, it unconsciously causes the brain to fill in gaps while trying to understand what is being said. This can lead to an inaccurate understanding of the information being transmitted, as well as longer reaction times and eventually fatigue. Typically, tests that assess these negative features are self-reported and therefore are difficult to obtain quantitative results from. Brain imaging modalities such as fMRI [126] and PET [127] have been used for this purpose, but introduce noise and are unable to place the subject in a realistic situation. These studies showed that the prefrontal cortex, the area of the brain monitored by the fNIR system used here, is significantly activated in response to this type of task.

The purpose of this experimental setup was to identify neural correlates using a prefrontal cortex monitoring fNIR system in combination with an fNIR system that is capable of imaging other parts of the brain than the prefrontal cortex. Using information from different parts of the brain, a more complete understanding of the effects synthesized speech has on cognitive processing can be gained.

5.2.3 Materials and Methods

In this experiment, the subjects listen to a series of statements in varying qualities: 1 being natural speech, the other 3 being synthesized speech. There were 5 different 5 second long sentences that were repeated in these differing qualities, to which the user then had to give a self-reported metric on intelligibility, naturalness, and overall quality of sound on a scale of 1 to 5, 5 being the best.

Event markers were sent from the audio stimulus program via serial to both the computer connected to the fNIR Devices system and to the Hitachi ETG-4000 fNIR system. The difficulty with this setup, however, is that the Hitachi receives the event markers in packets of three bytes, whereas the computer setup recording from the fNIR Devices system receives only one byte. The three bytes received by the Hitachi are a start byte, the event marker byte, and a stop byte. Therefore, the stimulus presentation computer setup was customized to send the separate formats over two separate serial ports. To simplify the setup, they only need to be sent over one serial port to both systems.

The following diagram is representative of the setup without using NeuroHub.

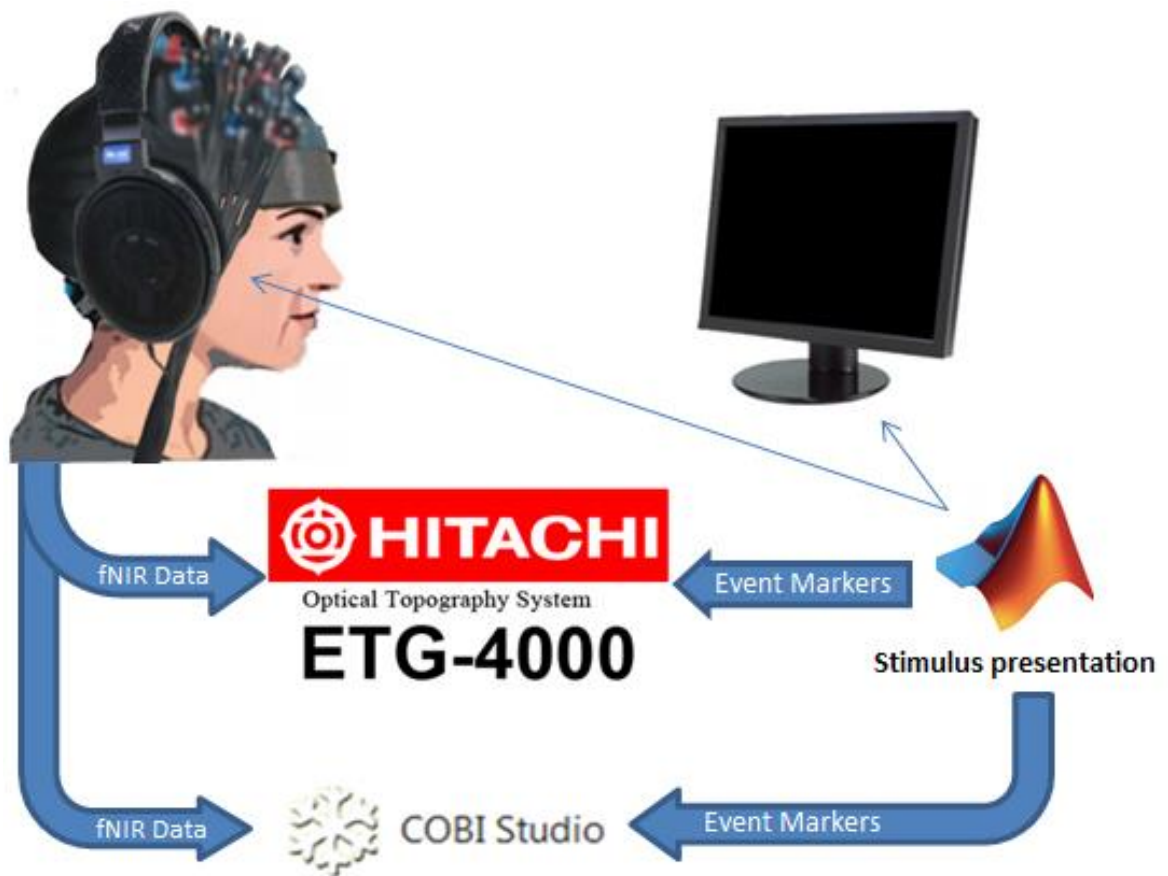


Figure 42: Diagram describing the flow of information in use case 2 without NeuroHub implementation.

The following diagram is representative of the setup with the NeuroHub.

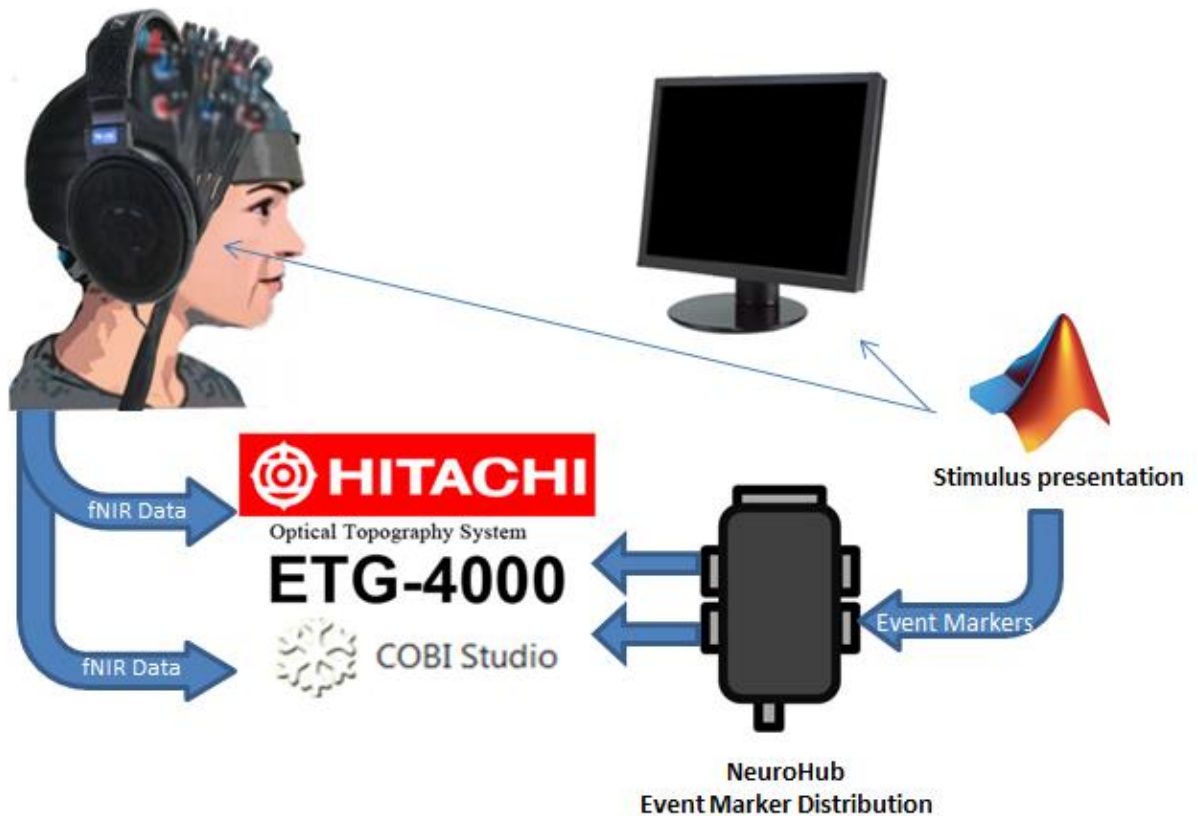


Figure 43: Diagram describing the flow of information in use case 2 with NeuroHub implementation.

The fNIR signals were low-pass filtered at a cutoff of 0.1 Hz to remove high frequency noise. The stimulus package sent event markers which were used to extract the fNIR data from 5 seconds pre-audio playing to 5 seconds after the audio was completed. The rest periods before the audio playing were compared to the periods while audio was playing to obtain average oxygenation change for each run.

5.2.4 Results and Conclusion

The discrepancies in timestamps was found in the same way it was for use case 1.

The following are the first 20 markers.

Table 5: First 20 synchronization markers from use case 2 experimental protocol. Showing byte received by both fNIR recording systems, as well as the timestamp (from start of recording) and time from the first marker received.

Hitachi fNIR Data			fNIR Devices Data		
Byte Value	Timestamp (s)	Time from first marker (s)	Byte Value	Timestamp (s)	Time from first marker (s)
1	61287.35	0	1	403.198	0
8	61288.56	1.21	8	404.331	1.133
2	61289.95	2.6	2	405.8	2.602
3	61294.95	7.6	3	410.812	7.614
4	61295.75	8.4	4	411.649	8.451
8	61303.56	16.21	8	419.421	16.223
8	61303.65	16.3	8	419.442	16.244
3	61307.56	20.21	3	423.418	20.22
4	61308.25	20.9	4	424.103	20.905
8	61316.06	28.71	8	431.868	28.67
3	61318.95	31.6	3	434.778	31.58
4	61319.65	32.3	4	435.488	32.29
8	61327.45	40.1	8	443.294	40.096
3	61328.85	41.5	3	444.677	41.479
4	61330.25	42.9	4	446.066	42.868
8	61341.65	54.3	8	457.456	54.258
3	61343.54	56.19	3	459.344	56.146
4	61344.85	57.5	4	460.729	57.531
8	61356.25	68.9	8	472.164	68.966
3	61358.04	70.69	3	473.847	70.649

The mean of all discrepancies is 0.0081 seconds, and the standard deviation is 0.0466. The following is a histogram showing the distribution of discrepancies:

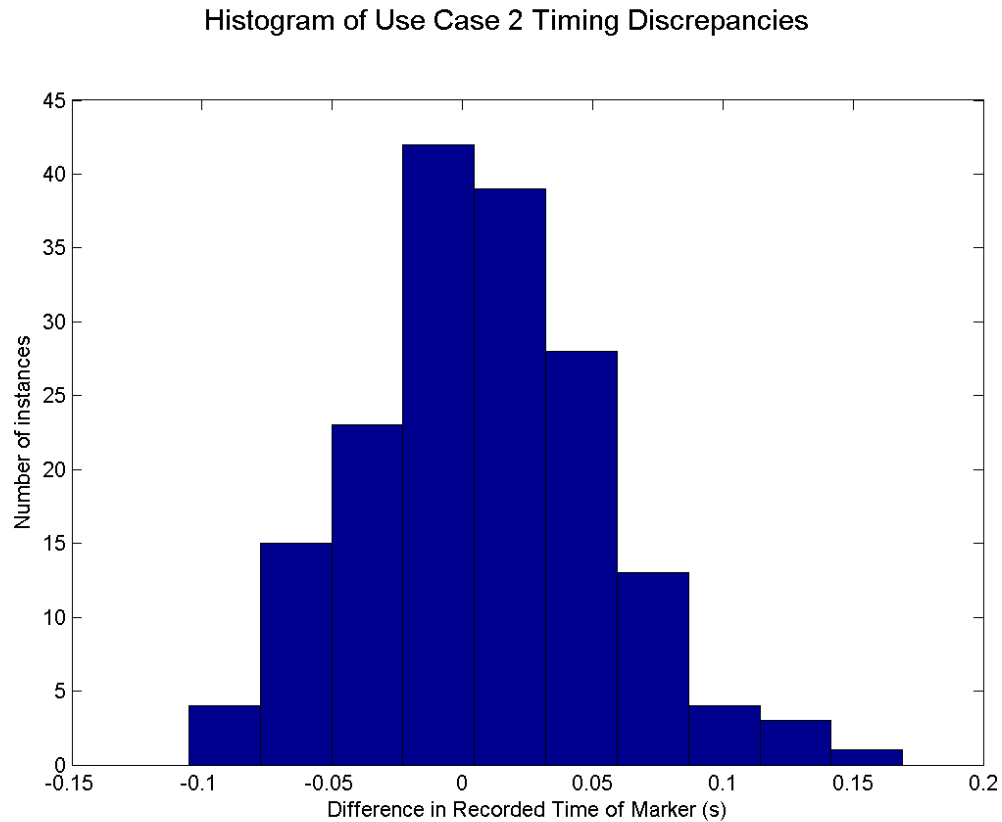


Figure 44: Histogram showing the frequencies of the difference in fNIR times from the systems' first markers.

All synchronization markers were successfully received by both systems so NeuroHub was successful. In this second setup, the arrangement of marker information flow was not necessarily simpler, but would save the researcher time in developing the setup, as they would not have to send two different formats of markers out from separate

serial ports. Instead, only one marker was necessary, and if other devices needed to be added, they would simply need to be attached to the NeuroHub that is already in place.

Chapter 6: Future Work and Conclusion

6.1 Future Work

NeuroHub currently supports serial, parallel, and TTL pulse event marker transmission. These are the first set of communication protocols to be implemented because they were the most common for setups. Development has already been started on a third generation device that would offer even more features. This sections describes present the work that has been completed on it as well as ideas and direction for future improvements.

The next major improvement for the device is the implementation of Ethernet capabilities and an SD card based data logging feature. The device would be able to send and receive information being sent over a UDP or TCP/IP stream. This would be useful for a number of reasons. First, as with the protocols already implemented, the device would be able to send the markers received there to other ports and send information arriving at other ports to a computer on the network. Secondly, it could be used as an extension to lab streaming layer to benefit from the advantages of both systems, in the same way that hybrid systems benefit from the advantages of different features or modalities. Although LSL has its drawbacks which this device aims to remedy, the usefulness of its features cannot be dismissed, as it offers many options which would be useful in custom setups. NeuroHub, being a microcontroller based embedded system dedicated to eliminating lag time due to non-embedded operating systems and un-complicating event marker connection setups, could be implemented with lab streaming

layer, with its full featured customizability, over Ethernet, providing the best of both worlds.

A chip was selected, the ENC28J60 by Microchip, that is designed to handle Ethernet protocol over serial peripheral interface (SPI) protocol, which the ATmega2560 supports. SPI operates in master/slave mode in which multiple slaves can operate on the same line sequentially, selected over individual slave select lines controlled by the master. SD card protocol also operates over SPI. So, if the microcontroller (the master) wished to address the ENC28J60, it would first notify the chip over the slave select line then interact with the chip over the master out slave in (MOSI) and master in slave out (MISO) lines. If the chip then wanted to address the SD card, it would stop communication with the ENC28J60, select the SD card as slave, then interact with the card over the same MOSI and MISO lines. Because the Arduino programming language is much simpler to implement new features, the wiring of the chip to the microcontroller has been tested on a breadboard using Arduino first. A few C based AVR libraries for the ENC28J60 exist, which would be useful in speeding development time. Further development would result in the Ethernet port functioning first as the other ports function, duplicating information, then a solution would be developed to include LSL, as mentioned previously.

The idea of developing an SD card data logging setup is to provide an accurate backup, in case the computers somehow fail to correctly record the markers. As mentioned previously, SD card protocol is also SPI, but the SD card and the microcontroller operate at different voltage levels. The microcontroller operates at 5

volts, while the card operates at 3.3 V. Directly connecting the card to the chip would likely cause damage to either or both. Therefore, a level conversion IC would be needed. The HEF4050BP chip by NXP Semiconductors was selected as a suitable option to meet this need. Development would proceed much like for Ethernet: first Arduino language on a breadboard, then a suitable C library would be used.

These chips, especially the ENC28J60, require other external components to function. The schematics including these components and how the chips are to be connected to the Arduino pins have been started, but it is likely that the board would need 4 sided printing, as arranging the components and routing the wires on the board layout proved difficult if not impossible with 2 layers, and the parallel port hasn't been added yet. With 4 layers, the design would be much simpler.

The components would also need to be rearranged, as presently the Ethernet jack is where the BNC port is on the second generation board and an appropriate spot for either the BNC port or parallel port has not been decided on. No additional code, except initial attempts to use the ENC28J60 with the ATmega2560, has been generated. It would also be useful to further develop the parallel port code. There is no suitable library available to handle more advanced protocols, and because they are master/slave based protocols, a solution would need to be developed which dealt with this decision, possibly with a switch that designated the port as master or slave, or the addition of a male parallel port.

In short, there are many directions this project could go. Earlier in the thesis, it was mentioned that this device would not be able to completely solve the problem but

only alleviate some of its symptoms. To completely solve the problem, an embedded recording system would need to be developed. It would be optimal if existing software and drivers could be ported to the specialized recording device. As for the device itself, suggestions have been made for its further development, but this list is not complete and other options should be explored.

6.2 Conclusion

This project set out as an attempt to shed light on problems arising from complications due to the use of multiple computers in a BCI system and propose and develop a solution. The problem of setup complexity was relatively simple to illustrate, whereas the problem of needing millisecond precise timing from computers and protocols that are not embedded or intended for millisecond precise timing at all was less straightforward. The incredible amount of variation in lag times in USB data transmission is unfit for use in BCIs. Also, the incompatibility of event marker transmission protocols used by different software or recording devices leaves the researcher with the task of finding the way to make the system work together. NeuroHub was designed for remedying these issues in a simple, inexpensive tool that can be assembled by anyone with little to no experience in electronic devices.

With future improvements, the design and plans of NeuroHub could accomplish these goals even more completely, handling more protocols and offering more features to give the user more time to do research by taking away time spent on attempting to make everything work together. The possibility of using NeuroHub alongside a software-based

data handling system such as the lab streaming layer opens the door to even more options and useful setups. Although this project focused on its intended use, BCIs, the device could also be used in any situation where precise timing of event markers to coordinate data between computers is needed.

What was initially intended to test the device attached to a real computer turned into an exhibition of just how bad the problem of variable lag time actually is, and provided more than adequate data to show beyond a doubt that the issue is real and open discussion on the adequacy of personal computers for high temporal resolution recording and the possibility of developing a true solution to the problem. As such, NeuroHub is only a crutch for this problem which researchers can easily utilize, but the problem still remains and thus should be more thoroughly addressed. NeuroHub on its own performed consistently and reliably with an 100 % transmission accuracy rate and virtually no deviation from a 1.020 millisecond lag time.

NeuroHub reliably broadcasts event markers over a few standard protocols which simplifies the setup of BCI systems and ensures markers are being sent to all recording devices at the same time. The intent is that this will aid BCI research and in turn help alleviate some of the psychological pain that comes with not being able to communicate with loved ones or do anything for oneself at all due to locked-in syndrome, as well as further our understanding on how the brain works and other useful findings resulting from neuroimaging research.

List of References

- [1] J. R. Wolpaw, N. Birbaumer, D. J. McFarland, G. Pfurtscheller, and T. M. Vaughan, "Brain-computer interfaces for communication and control," *Clinical Neurophysiology*, vol. 113, pp. 767-91, Jun 2002.
- [2] M. Lebedev, "Brain-machine interfaces: an overview," *Translational Neuroscience*, vol. 5, pp. 99-110, 2014.
- [3] T. Zander, C. Kothe, S. Jatzev, and M. Gaertner, "Enhancing Human-Computer Interaction with Input from Active and Passive Brain-Computer Interfaces," in *Brain-Computer Interfaces*, D. S. Tan and A. Nijholt, Eds., ed: Springer London, 2010, pp. 181-199.
- [4] G. Schalk, E. C. Leuthardt, P. Brunner, J. G. Ojemann, L. A. Gerhardt, and J. R. Wolpaw, "Real-time detection of event-related brain activity," *Neuroimage*, vol. 43, pp. 245-9, Nov 1 2008.
- [5] N. V. Thakor, "Translating the Brain-Machine Interface," *Science Translational Medicine*, vol. 5, pp. 210-17, 2013.
- [6] J. Mellinger, G. Schalk, C. Braun, H. Preissl, W. Rosenstiel, N. Birbaumer, *et al.*, "An MEG-based brain-computer interface (BCI)," *Neuroimage*, vol. 36, pp. 581-593, 2007.
- [7] C. R. deCharms, "Applications of real-time fMRI," *Nat Rev Neurosci*, vol. 9, pp. 720-729, 2008.
- [8] R. Sitaram, N. Weiskopf, A. Caria, R. Veit, M. Erb, and N. Birbaumer, "fMRI brain-computer interfaces," *Signal Processing Magazine, IEEE*, vol. 25, pp. 95-106, 2007.
- [9] H. Ayaz, P. A. Shewokis, S. Bunce, K. Izzetoglu, B. Willems, and B. Onaral, "Optical brain monitoring for operator training and mental workload assessment," *NeuroImage*, vol. 59, pp. 36-47, 2012.

-
- [10] S. M. Coyle, T. E. Ward, and C. M. Markham, "Brain-computer interface using a simplified functional near-infrared spectroscopy system," *Journal of neural engineering*, vol. 4, pp. 219-226, 2007.
- [11] J. J. Daly and J. R. Wolpaw, "Brain-computer interfaces in neurological rehabilitation," *The Lancet Neurology*, vol. 7, pp. 1032-1043, 2008.
- [12] G. Gallegos-Ayala, A. Furdea, K. Takano, C. A. Ruf, H. Flor, and N. Birbaumer, "Brain communication in a completely locked-in patient using bedside near-infrared spectroscopy," *Neurology*, vol. 82, pp. 1930-1932, May 27, 2014.
- [13] M. R. Turner, O. Hardiman, M. Benatar, B. R. Brooks, A. Chio, M. de Carvalho, *et al.*, "Controversies and priorities in amyotrophic lateral sclerosis," *The Lancet Neurology*, vol. 12, pp. 310-322, 2013.
- [14] L. H. Goldstein and S. Abrahams, "Changes in cognition and behaviour in amyotrophic lateral sclerosis: nature of impairment and implications for assessment," *The Lancet Neurology*, vol. 12, pp. 368-380, 2013.
- [15] H. Ayaz, P. A. Shewokis, L. Scull, D. Libon, J. , S. Feldman, J. Eppig, *et al.*, "Assessment of Prefrontal Cortex Activity in Amyotrophic Lateral Sclerosis Patients with Functional Near Infrared Spectroscopy," *Journal of Neuroscience and Neuroengineering*, vol. 3, pp. 41-51, 2014.
- [16] S. Makeig, S. Debener, J. Onton, and A. Delorme, "Mining event-related brain dynamics," *Trends in cognitive sciences*, vol. 8, pp. 204-210, 2004.
- [17] H. Ayaz, S. L. Allen, S. M. Platek, and B. Onaral, "Maze Suite 1.0: A complete set of tools to prepare, present, and analyze navigational and spatial cognitive neuroscience experiments," *Behavior Research Methods*, vol. 40, pp. 353-359, 2008.
- [18] L. A. Farwell and E. Donchin, "Talking off the top of your head: toward a mental prosthesis utilizing event-related brain potentials," *Electroencephalography and clinical neurophysiology*, vol. 70, pp. 510-523, 1988.

-
- [19] I. Käthner, C. A. Ruf, E. Pasqualotto, C. Braun, N. Birbaumer, and S. Halder, "A portable auditory P300 brain-computer interface with directional cues," *Clinical neurophysiology : official journal of the International Federation of Clinical Neurophysiology*, vol. 124, p. 327, 2013.
- [20] H. Nojo, M. Kawasaki, T. Jyo, A. Ishiyama, N. Kasai, and Y. Ono, "Appropriate auditory stimuli for P300 brain-computer interface," *Neuroscience research*, vol. 68, p. e327, 2010.
- [21] A.-M. Brouwer and J. B. F. van Erp, "A tactile P300 brain-computer interface," *Frontiers in neuroscience*, vol. 4, p. 19, 2010.
- [22] R. Ortner, Z. Lugo, Q. Noirhomme, S. Laureys, and C. Guger, "A tactile Brain-Computer Interface for severely disabled patients," in *Haptics Symposium (HAPTICS), 2014 IEEE*, 2014, pp. 235-237.
- [23] T. M. Rutkowski and H. Mori, "Tactile and bone-conduction auditory brain computer interface for vision and hearing impaired users," *J Neurosci Methods*, Apr 21 2014.
- [24] D. J. Krusienski, E. W. Sellers, F. Cabestaing, S. Bayouth, D. J. McFarland, T. M. Vaughan, *et al.*, "A comparison of classification techniques for the P300 Speller," *Journal of Neural Engineering*, vol. 3, pp. 299-305, 2006.
- [25] D. J. Krusienski, E. W. Sellers, D. J. McFarland, T. M. Vaughan, and J. R. Wolpaw, "Toward enhanced P300 speller performance," *Journal of neuroscience methods*, vol. 167, pp. 15-21, 2008.
- [26] W. Speier, I. Fried, and N. Pouratian, "Improved P300 speller performance using electrocorticography, spectral features, and natural language processing," *Clinical neurophysiology : official journal of the International Federation of Clinical Neurophysiology*, vol. 124, pp. 1321-1328, 2013.
- [27] C. S. Throckmorton, K. A. Colwell, D. B. Ryan, E. W. Sellers, and L. M. Collins, "Bayesian Approach to Dynamically Controlling Data Collection in P300 Spellers," *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, vol. 21, pp. 508-517, 2013.

-
- [28] A. Curtin, H. Ayaz, Y. Liu, P. A. Shewokis, and B. Onaral, "A P300-based EEG-BCI for spatial navigation control," United States, 2012, pp. 3841-3844.
- [29] R. Fazel-Rezai, S. Gavett, W. Ahmad, A. Rabbi, and E. Schneider, "A comparison among several P300 brain-computer interface speller paradigms," *Clinical EEG and neuroscience*, vol. 42, p. 209, 2011.
- [30] Y. Liu, H. Ayaz, A. Curtin, P. A. Shewokis, and B. Onaral, "Detection of attention shift for asynchronous P300-based BCI," United States, 2012, pp. 3850-3853.
- [31] N. Galloway, "Human brain electrophysiology: Evoked potentials and evoked magnetic fields in science and medicine," *The British journal of ophthalmology*, vol. 74, p. 255, 1990.
- [32] E. E. Sutter, "The brain response interface: communication through visually-induced electrical brain responses," *Journal of Microcomputer Applications*, vol. 15, pp. 31-45, 1992.
- [33] M. Middendorf, G. McMillan, G. Calhoun, and K. S. Jones, "Brain-computer interfaces based on the steady-state visual-evoked response," *IEEE Transactions on Rehabilitation Engineering*, vol. 8, pp. 211-214, 2000.
- [34] Y. Wang, Y. T. Wang, and T. P. Jung, "Visual stimulus design for high-rate SSVEP BCI," *Electronics Letters*, vol. 46, p. 1057, 2010.
- [35] A. Kübler, F. Nijboer, J. Mellinger, T. M. Vaughan, H. Pawelzik, G. Schalk, *et al.*, "Patients with ALS can use sensorimotor rhythms to operate a brain-computer interface," *Neurology*, vol. 64, pp. 1775-1777, 2005.
- [36] W.-P. Teo and E. Chew, "Is Motor-Imagery Brain-Computer Interface Feasible in Stroke Rehabilitation?," *PM&R*, vol. 6, pp. 723-8, Aug 2014.
- [37] G. Pfurtscheller and C. Neuper, "Motor imagery and direct brain-computer communication," *Proceedings of the IEEE*, vol. 89, pp. 1123-1134, 2001.

-
- [38] D. Wang, D. Miao, and G. Blohm, "Multi-class motor imagery EEG decoding for brain-computer interfaces," *Frontiers in neuroscience*, vol. 6, p. 151, 2012.
- [39] H. Ehrlichman and M. S. Wiener, "EEG asymmetry during covert mental activity," *Psychophysiology*, vol. 17, pp. 228-235, 1980.
- [40] C. S. L. Tsui, J. Q. Gan, and H. Hu, "A self-paced motor imagery based brain-computer interface for robotic wheelchair control," *Clinical EEG and neuroscience*, vol. 42, p. 225, 2011.
- [41] B. Blankertz, G. Dornhege, M. Krauledat, M. Schröder, J. Williamson, R. Murray-Smith, *et al.*, "The Berlin Brain-Computer Interface presents the novel mental typewriter Hex-o-Spell," 2006.
- [42] T. Hinterberger, S. Schmidt, N. Neumann, J. Mellinger, B. Blankertz, G. Curio, *et al.*, "Brain-computer communication and slow cortical potentials," *IEEE Transactions on Biomedical Engineering*, vol. 51, pp. 1011-1018, 2004.
- [43] A. Kbler, N. Neumann, J. Kaiser, B. Kotchoubey, T. Hinterberger, and N. P. Birbaumer, "Brain-computer communication: Self-regulation of slow cortical potentials for verbal communication," *Archives of Physical Medicine and Rehabilitation*, vol. 82, pp. 1533-1539, 2001.
- [44] I. G. Campbell, "EEG recording and analysis for sleep research," *Curr Protoc Neurosci*, vol. Chapter 10, p. Unit 10.2, Oct 2009.
- [45] M. Izzetoglu, K. Izzetoglu, S. Bunce, H. Ayaz, A. Devaraj, B. Onaral, *et al.*, "Functional near-infrared neuroimaging," *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, vol. 13, pp. 153-159, 2005.
- [46] H. Ayaz, B. Onaral, K. Izzetoglu, P. A. Shewokis, R. McKendrick, and R. Parasuraman, "Continuous monitoring of brain dynamics with functional near infrared spectroscopy as a tool for neuroergonomic research: Empirical examples and a technological development," *Frontiers in Human Neuroscience*, vol. 7, pp. 1-13, 2013.

-
- [47] H. Ayaz, P. A. Shewokis, A. Curtin, M. Izzetoglu, K. Izzetoglu, and B. Onaral, "Using MazeSuite and Functional Near Infrared Spectroscopy to Study Learning in Spatial Navigation," *J Vis Exp*, p. e3443, 2011.
- [48] G. Strangman, D. A. Boas, and J. P. Sutton, "Non-invasive neuroimaging using near-infrared light," *Biological psychiatry*, vol. 52, pp. 679-693, 2002.
- [49] K. Izzetoglu, "Neural correlates of cognitive workload and anesthetic depth: fNIR spectroscopy investigation in humans," Dissertation/Thesis, ProQuest, UMI Dissertations Publishing, 2008.
- [50] H. Ayaz, B. Ben Dor, D. Solt, and B. Onaral, "Infrascanner: Cost Effective, Mobile Medical Imaging System for Detecting Hemotomas," *Journal of Medical Devices*, vol. 5, p. 27540, 2011.
- [51] D. Afergan, E. M. Peck, E. T. Solovey, A. Jenkins, S. W. Hincks, E. T. Brown, *et al.*, "Dynamic Difficulty Using Brain Metrics of Workload," presented at the CHI2014, 2014.
- [52] G. Derosièrè, S. Dalhoumi, S. Perrey, G. Dray, and T. Ward, "Towards a Near Infrared Spectroscopy-Based Estimation of Operator Attentional State," *PLoS ONE*, vol. 9, p. e92045, 2014.
- [53] F. A. Fishburn, M. E. Norr, A. V. Medvedev, and C. J. Vaidya, "Sensitivity of fNIRS to cognitive state and load," *Frontiers in Human Neuroscience*, vol. 8, 2014.
- [54] G. Derosièrè, K. Mandrick, G. Dray, T. E. Ward, and S. Perrey, "NIRS-measured prefrontal cortex activity in neuroergonomics: strengths and weaknesses," *Frontiers in Human Neuroscience*, vol. 7, September 19 2013.
- [55] Y. Hoshi, B. H. Tsou, V. A. Billock, M. Tanosaki, Y. Iguchi, M. Shimada, *et al.*, "Spatiotemporal characteristics of hemodynamic changes in the human lateral prefrontal cortex during working memory tasks," *NeuroImage*, vol. 20, pp. 1493-1504, Nov 2003.

-
- [56] H. Ayaz, B. Willems, B. Bunce, P. A. Shewokis, K. Izzetoglu, S. Hah, *et al.*, "Cognitive Workload Assessment of Air Traffic Controllers Using Optical Brain Imaging Sensors," in *Advances in Understanding Human Performance: Neuroergonomics, Human Factors Design, and Special Populations*, T. Marek, W. Karwowski, and V. Rice, Eds., ed: CRC Press Taylor & Francis Group, 2010, pp. 21-31.
- [57] R. McKendrick, H. Ayaz, R. Olmstead, and R. Parasuraman, "Enhancing dual-task performance with verbal and spatial working memory training: continuous monitoring of cerebral hemodynamics with NIRS," *NeuroImage*, vol. 85 Pt 3, pp. 1014-1026, 2014.
- [58] C. Herff, D. Heger, O. Fortmann, J. Hennrich, F. Putze, and T. Schultz, "Mental workload during n-back task—quantified in the prefrontal cortex using fNIRS," *Frontiers in Human Neuroscience*, vol. 7, p. 935, January 16 2014.
- [59] C. Bogler, J. Mehnert, J. Steinbrink, and J. D. Haynes, "Decoding Vigilance with NIRS," *PLoS One*, vol. 9, p. e101729, 2014.
- [60] A. R. Harrivel, D. H. Weissman, D. C. Noll, and S. J. Peltier, "Monitoring attentional state with fNIRS," *Frontiers in Human Neuroscience*, vol. 7, December 13 2013.
- [61] A. C. Ruocco, A. H. Rodrigo, J. Lam, S. Di Domenico, B. Graves, and H. Ayaz, "A Problem-Solving Task Specialized for Functional Neuroimaging: Validation of the Scarborough adaptation of the Tower of London (S-TOL) using Near-Infrared Spectroscopy," *Frontiers in Human Neuroscience*, vol. 8, 2014.
- [62] H. Ayaz, P. A. Shewokis, M. İzzetoğlu, M. P. Çakır, and B. Onaral, "Tangram solved? Prefrontal cortex activation analysis during geometric problem solving," in *34th Annual International IEEE EMBS Conference*, San Diego, CA, 2012, pp. 4724 - 4727
- [63] H. Ayaz, M. P. Cakir, K. Izzetoglu, A. Curtin, P. A. Shewokis, S. Bunce, *et al.*, "Monitoring Expertise Development during Simulated UAV Piloting Tasks using Optical Brain Imaging," presented at the IEEE Aerospace Conference, BigSky, MN, USA, 2012.

-
- [64] D. Gefen, H. Ayaz, and B. Onaral, "Applying Functional Near Infrared (fNIR) Spectroscopy to Enhance MIS Research," *AIS Transactions on Human-Computer Interaction*, vol. 6, pp. 55-73, 2014.
- [65] F. Irani, S. M. Platek, S. Bunce, A. C. Ruocco, and D. Chute, "Functional near infrared spectroscopy (fNIRS): an emerging neuroimaging technology with important applications for the study of brain disorders," *The Clinical Neuropsychologist*, vol. 21, pp. 9-37, 2007.
- [66] P. M. Arenth, J. H. Ricker, and M. T. Schultheis, "Applications of functional near-infrared spectroscopy (fNIRS) to neurorehabilitation of cognitive disabilities," *The Clinical Neuropsychologist*, vol. 21, pp. 38-57, 2007.
- [67] H. Saitou, H. Yanagi, S. Hara, S. Tsuchiya, and S. Tomura, "Cerebral blood volume and oxygenation among poststroke hemiplegic patients: Effects of 13 rehabilitation tasks measured by near-Infrared spectroscopy," *Archives of Physical Medicine and Rehabilitation*, vol. 81, pp. 1348-1356, Dec 2000.
- [68] S. Bunce, K. Izzetoglu, M. Izzetoglu, H. Ayaz, K. Pourrezaei, and B. Onaral, "Treatment Status Predicts Differential Prefrontal Cortical Responses to Alcohol and Natural Reinforcer Cues among Alcohol Dependent Individuals," in *Advances in Brain Inspired Cognitive Systems*. vol. 7366, H. Zhang, A. Hussain, D. Liu, and Z. Wang, Eds., ed: Springer Berlin / Heidelberg, 2012, pp. 183-191.
- [69] A. C. Merzagora, M. T. Schultheis, B. Onaral, and M. Izzetoglu, "Functional near-infrared spectroscopy-based assessment of attention impairments after traumatic brain injury," *Journal of Innovative Optical Health Sciences*, vol. 4, pp. 251-260, 2011.
- [70] H. Ayaz and B. Onaral, "Analytical software and stimulus-presentation platform to utilize, visualize and analyze near-infrared spectroscopy measures," Masters Degree Thesis (MS), Drexel University, Philadelphia, PA, 2005.
- [71] S. Lloyd-Fox, A. Blasi, and C. E. Elwell, "Illuminating the developing brain: The past, present and future of functional near infrared spectroscopy," *Neuroscience & Biobehavioral Reviews*, vol. 34, pp. 269-284, Feb 2010.

-
- [72] A. Blasi, D. Phillips, S. Lloyd-Fox, P. H. Koh, and C. E. Elwell, "Automatic detection of motion artifacts in infant functional optical topography studies," in *Oxygen Transport to Tissue XXXI*, ed: Springer, 2010, pp. 279-284.
- [73] H. Ayaz, M. Izzetoglu, P. A. Shewokis, and B. Onaral, "Sliding-window Motion Artifact Rejection for Functional Near-Infrared Spectroscopy," *Conf Proc IEEE Eng Med Biol Soc*, pp. 6567-70, 2010.
- [74] B. Molavi and G. A. Dumont, "Wavelet based motion artifact removal for Functional Near Infrared Spectroscopy," in *Engineering in Medicine and Biology Society (EMBC), 2010 Annual International Conference of the IEEE*, Buenos Aires, Argentina, 2010, pp. 5-8.
- [75] J. Virtanen, T. Noponen, and P. Meriläinen, "Comparison of principal and independent component analysis in removing extracerebral interference from near-infrared spectroscopy signals," *Journal of Biomedical Optics*, vol. 14, p. 054032, 2009.
- [76] F. C. Robertson, T. S. Douglas, and E. M. Meintjes, "Motion Artifact Removal for Functional Near Infrared Spectroscopy: A Comparison of Methods," *IEEE Transactions on Biomedical Engineering*, vol. 57, pp. 1377-1387, 2010.
- [77] F. Scholkmann, S. Kleiser, A. J. Metz, R. Zimmermann, J. Mata Pavia, U. Wolf, *et al.*, "A review on continuous wave functional near-infrared spectroscopy and imaging instrumentation and methodology," *NeuroImage*, vol. 85, Part 1, pp. 6-27, Jan 15 2014.
- [78] H. Ayaz, S. Bunce, P. Shewokis, K. Izzetoglu, B. Willems, and B. Onaral, "Using Brain Activity to Predict Task Performance and Operator Efficiency," in *Advances in Brain Inspired Cognitive Systems*. vol. 7366, H. Zhang, A. Hussain, D. Liu, and Z. Wang, Eds., ed: Springer Berlin / Heidelberg, 2012, pp. 147-155.
- [79] H. Ayaz, M. Izzetoglu, S. Bunce, T. Heiman-Patterson, and B. Onaral, "Detecting cognitive activity related hemodynamic signal for brain computer interface using functional near infrared spectroscopy," *Conf Proc 3rd IEEE/EMBS on Neural Eng*, pp. 342-345, 2007.

-
- [80] H. Ayaz, P. A. Shewokis, S. Bunce, M. Schultheis, and B. Onaral, "Assessment of Cognitive Neural Correlates for a Functional Near Infrared-Based Brain Computer Interface System," in *Foundations of Augmented Cognition. Neuroergonomics and Operational Neuroscience*, D. Schmorow, Ed., ed, 2009, pp. 699-708.
- [81] S. M. Coyle, T. E. Ward, and C. M. Markham, "Brain-computer interface using a simplified functional near-infrared spectroscopy system," *J Neural Eng*, vol. 4, pp. 219-26, Sep 2007.
- [82] H. Ayaz, P. A. Shewokis, S. Bunce, and B. Onaral, "An optical brain computer interface for environmental control," *Conf Proc IEEE Eng Med Biol Soc*, vol. 2011, pp. 6327-30, 2011.
- [83] E. Solovey, P. Schermerhorn, M. Scheutz, A. Sassaroli, S. Fantini, and R. Jacob, "Brainput: enhancing interactive systems with streaming fnirs brain input," 2012, pp. 2193-2202.
- [84] A. M. Batula, H. Ayaz, and Y. E. Kim, "Evaluating a Four-Class Motor-Imagery-Based Optical Brain-Computer Interface," in *IEEE Engineering in Medicine Conf.*, Chicago, USA, 2014.
- [85] R. Sitaram, H. Zhang, C. Guan, M. Thulasidas, Y. Hoshi, A. Ishikawa, *et al.*, "Temporal classification of multichannel near-infrared spectroscopy signals of motor imagery for developing a brain-computer interface," *Neuroimage*, vol. 34, pp. 1416-27, Feb 15 2007.
- [86] V. Quaresima, S. Bisconti, and M. Ferrari, "A brief review on the use of functional near-infrared spectroscopy (fNIRS) for language imaging studies in human newborns and adults," *Brain and Language*, vol. 121, pp. 79-89, May 2012.
- [87] S. I. Di Domenico, M. A. Fournier, H. Ayaz, and A. C. Ruocco, "In search of integrative processes: basic psychological need satisfaction predicts medial prefrontal activation during decisional conflict," *Journal of experimental psychology.General*, vol. 142, p. 967, 2013.

-
- [88] J. D. R. Millán, R. Rupp, G. R. Müller-Putz, R. Murray-Smith, C. Giugliemma, M. Tangermann, *et al.*, "Combining Brain-Computer Interfaces and Assistive Technologies: State-of-the-Art and Challenges," *Frontiers in neuroscience*, vol. 4, 2010.
- [89] P. Brunner, L. Bianchi, C. Guger, F. Cincotti, and G. Schalk, "Current trends in hardware and software for brain-computer interfaces (BCIs)," *Journal of neural engineering*, vol. 8, p. 025001, 2011.
- [90] G. Pfurtscheller, B. Z. Allison, C. Brunner, G. Bauernfeind, T. Solis-Escalante, R. Scherer, *et al.*, "The hybrid BCI," *Frontiers in Neuroscience* vol. 4, p. 30, 2010.
- [91] S. Amiri, R. Fazel-Rezai, and V. Asadpour, "A Review of Hybrid Brain-Computer Interface Systems," *Advances in Human-Computer Interaction*, vol. 2013, pp. 1-8, 2013.
- [92] L. Astolfi, J. Toppi, F. De Vico Fallani, G. Vecchiato, S. Salinari, D. Mattia, *et al.*, "Neuroelectrical hyperscanning measures simultaneous brain activity in humans," *Brain topography*, vol. 23, pp. 243-256, 2010.
- [93] F. Babiloni, F. Cincotti, D. Mattia, M. Mattiocco, F. De Vico Fallani, A. Tocci, *et al.*, "Hypermethods for EEG hyperscanning," *Conf Proc IEEE Eng Med Biol Soc*, vol. 1, pp. 3666-3669, 2006 2006.
- [94] M. Hirata, T. Ikeda, M. Kikuchi, T. Kimura, H. Hiraishi, Y. Yoshimura, *et al.*, "Hyperscanning MEG for understanding mother-child cerebral interactions," *Frontiers in human neuroscience*, vol. 8, p. 118, 2014.
- [95] N. Osaka, T. Minamoto, K. Yaoi, M. Azuma, and M. Osaka, "Neural Synchronization During Cooperated Humming: A Hyperscanning Study Using fNIRS," *Procedia - Social and Behavioral Sciences*, vol. 126, pp. 241-243, 2014.
- [96] P. R. Montague, G. S. Berns, J. D. Cohen, S. M. McClure, G. Pagnoni, M. Dhamala, *et al.*, "Hyperscanning: Simultaneous fMRI during Linked Social Interactions," *NeuroImage*, vol. 16, pp. 1159-1159, 2002.

-
- [97] L. Holper, F. Scholkmann, and M. Wolf, "Between-brain connectivity during imitation measured by fNIRS," *NeuroImage*, vol. 63, p. 212, 2012.
- [98] R. C. Panicker, S. Puthusserypady, and Y. Sun, "An Asynchronous P300 BCI With SSVEP-Based Control State Detection," *IEEE Transactions on Biomedical Engineering*, vol. 59, pp. 1781-1788, 2011.
- [99] G. Edlinger, C. Holzner, and C. Guger, "A Hybrid Brain-Computer Interface for Smart Home Control," in *Human-Computer Interaction. Interaction Techniques and Environments*. vol. 6762, J. Jacko, Ed., ed: Springer Berlin Heidelberg, 2011, pp. 417-426.
- [100] B. Rebsamen, E. Burdet, Q. Zeng, H. Zhang, M. Ang, C. L. Teo, *et al.*, "Hybrid P300 and mu-beta brain computer interface to operate a brain controlled wheelchair," in *Proceedings of the 2nd International Convention on Rehabilitation Engineering & Assistive Technology*, Bangkok, Thailand, 2008, pp. 51-55.
- [101] Y. Su, Y. Qi, J.-x. Luo, B. Wu, F. Yang, Y. Li, *et al.*, "A hybrid brain-computer interface control strategy in a virtual environment," *Journal of Zhejiang University SCIENCE C*, vol. 12, pp. 351-361, 2011.
- [102] H. Riechmann, N. Hachmeister, H. Ritter, and A. Finke, "Asynchronous, parallel on-line classification of P300 and ERD for an efficient hybrid BCI," in *Neural Engineering (NER), 2011 5th International IEEE/EMBS Conference on*, 2011, pp. 412-415.
- [103] B. Z. Allison, C. Brunner, V. Kaiser, G. R. Müller-Putz, C. Neuper, and G. Pfurtscheller, "Toward a hybrid brain-computer interface based on imagined movement and visual attention," *Journal of Neural Engineering*, vol. 7, p. 026007, 2010.
- [104] C. Vidaurre and B. Blankertz, "Towards a cure for BCI illiteracy," *Brain topography*, vol. 23, pp. 194-198, 2010.
- [105] G. Pfurtscheller, T. Solis-Escalante, R. Ortner, P. Linortner, and G. R. Müller-Putz, "Self-paced operation of an SSVEP-Based orthosis with and without an

imagery-based "brain switch:" a feasibility study towards a hybrid BCI," *IEEE Trans Neural Syst Rehabil Eng*, vol. 18, pp. 409-14, Aug 2010.

- [106] A. Savić, U. Kisić, and M. B. Popović, "Toward a Hybrid BCI for Grasp Rehabilitation," in *5th European Conference of the International Federation for Medical and Biological Engineering*. vol. 37, Á. Jobbágy, Ed., ed: Springer Berlin Heidelberg, 2012, pp. 806-809.
- [107] F. Biessmann, S. Plis, F. C. Meinecke, T. Eichele, and K.-R. Müller, "Analysis of Multimodal Neuroimaging Data," *IEEE Reviews in Biomedical Engineering*, vol. 1, pp. 26-58, 2011.
- [108] N. K. Logothetis, "What we can do and what we cannot do with fMRI," *Nature*, vol. 453, pp. 869-878, 2008.
- [109] S. Aungsakul, A. Phinyomark, P. Phukpattaranont, and C. Limsakul, "Evaluating Feature Extraction Methods of Electrooculography (EOG) Signal for Human-Computer Interface," *Procedia Engineering*, vol. 32, pp. 246-252, 2012.
- [110] R. Barea, L. Boquete, M. Mazo, and E. López, "Wheelchair Guidance Strategies Using EOG," *Journal of Intelligent and Robotic Systems*, vol. 34, pp. 279-299, 2002.
- [111] C.-C. Postelnicu, F. Girbacia, and D. Talaba, "EOG-based visual navigation interface development," *Expert Systems with Applications*, vol. 39, p. 10857, 2012.
- [112] Y. Punsawad, Y. Punsawad, Y. Wongsawat, Y. Wongsawat, M. Parnichkun, and M. Parnichkun, "Hybrid EEG-EOG brain-computer interface system for practical machine control," United States, 2010, pp. 1360-1363.
- [113] T. O. Zander, M. Gaertner, C. Kothe, and R. Vilimek, "Combining Eye Gaze Input With a Brain-Computer Interface for Touchless Human-Computer Interaction," *International Journal of Human - Computer Interaction*, vol. 27, pp. 38-51, 2011.

-
- [114] R. Leeb, H. Sagha, R. Chavarriaga, and J. d. R. Millán, "A hybrid brain–computer interface based on the fusion of electroencephalographic and electromyographic activities," *Journal of Neural Engineering*, vol. 8, p. 025011, 2011.
- [115] X. Yong, M. Fatourech, R. K. Ward, and G. E. Birch, "The Design of a Point-and-Click System by Integrating a Self-Paced Brain-Computer Interface With an Eye-Tracker," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 1, pp. 590-602, 2011.
- [116] S. Fazli, J. Mehnert, J. Steinbrink, G. Curio, A. Villringer, K.-R. Müller, *et al.*, "Enhanced performance by a hybrid NIRS-EEG brain computer interface," *NeuroImage*, vol. 59, pp. 519-529, 2012.
- [117] T. Tsubone, T. Muroga, and Y. Wada, "Application to robot control using brain function measurement by near-infrared spectroscopy," *Conf Proc IEEE Eng Med Biol Soc*, vol. 2007, pp. 5342-5345, 2007 2007.
- [118] Y. Liu, H. Ayaz, A. Curtin, B. Onaral, and P. A. Shewokis, "Towards a Hybrid P300-Based BCI Using Simultaneous fNIR and EEG," in *Foundations of Augmented Cognition*, ed: Springer, 2013, pp. 335-344.
- [119] A. Faress and T. Chau, "Towards a multimodal brain-computer interface: combining fNIRS and fTCD measurements to enable higher classification accuracy," *NeuroImage*, vol. 77, pp. 186-194, 2013.
- [120] M. Okamoto, H. Dan, I. Dan, K. Shimizu, K. Takeo, T. Amita, *et al.*, "Multimodal assessment of cortical activation during apple peeling by NIRS and fMRI," *NeuroImage*, vol. 21, pp. 1275-1288, 2004.
- [121] R. J. Cooper, L. Gagnon, D. M. Goldenholz, D. A. Boas, and D. N. Greve, "The utility of near-infrared spectroscopy in the regression of low-frequency physiological noise from functional magnetic resonance imaging data," *NeuroImage*, vol. 59, pp. 3128-3138, 2012.
- [122] D. Touraine, P. Bourdot, Y. Bellik, and L. Bolot, "A framework to manage multimodal fusion of events for advanced interactions within virtual environments," presented at the Proceedings of the workshop on Virtual environments 2002, Barcelona, Spain, 2002.

-
- [123] M. Dellisanti, M. Fiorentino, G. Monno, and A. Uva, "Flexible Architecture for Multimodal Augmented Reality Engineering Applications."
- [124] D. Bannach, O. Amft, and P. Lukowicz, "Automatic Event-Based Synchronization of Multimodal Data Streams from Wearable and Ambient Sensors," in *Smart Sensing and Context*. vol. 5741, P. Barnaghi, K. Moessner, M. Presser, and S. Meissner, Eds., ed: Springer Berlin Heidelberg, 2009, pp. 135-148.
- [125] H. Ayaz, P. Crawford, A. Curtin, M. Syed, B. Onaral, W. M. Beltman, *et al.*, "Differential Prefrontal Response during Natural and Synthetic Speech Perception: An fNIR Based Neuroergonomics Study," in *Foundations of Augmented Cognition*. vol. 8027, D. Schmorow and C. Fidopiastis, Eds., ed: Springer Berlin Heidelberg, 2013, pp. 241-249.
- [126] R. R. Benson, D. H. Whalen, M. Richardson, B. Swainson, V. P. Clark, S. Lai, *et al.*, "Parametrically dissociating speech and nonspeech perception in the brain using fMRI," *Brain and language*, vol. 78, pp. 364-364, 2001.
- [127] D. J. Sharp, S. K. Scott, and R. J. S. Wise, "Monitoring and the controlled processing of meaning: distinct prefrontal systems," *Cerebral cortex (New York, N.Y.: 1991)*, vol. 14, pp. 1-10, 2004.

Appendix A: Board Assembly Guide

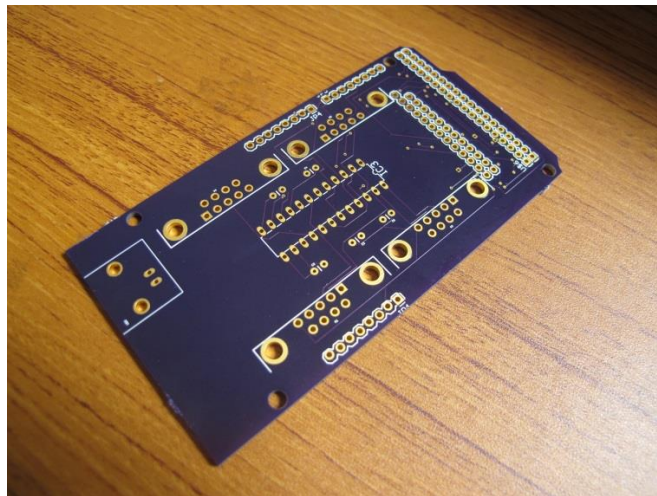
NeuroHub was designed and developed to be made by readily available components and easily assembled so it can be duplicated and expanded on. This guide intends to provide the reader with step-by-step instructions on obtaining the necessary parts, assembling the board, and programming the microcontroller.

All of the parts are relatively common and certainly not difficult to obtain. They can be found at Mouser Electronics or Digikey Electronics. The Arduino Mega 2560 can be purchased for much less from other online stores. Here is a parts list and, if a specific part is required, their corresponding Digikey part number:

- MAX238 IC – Serial logic conversion: MAX238CNG+-ND
- AC to DC Wall Adapter – Arduino power supply: T983-P5P-ND
- Plastic case: 1050-1003-ND
- BNC Mount: A97553-ND
- 5x 1.0 uF Capacitors (any will work)
- 4x Female Serial Connectors (any will work)
- Female Parallel Port Connector (any will work)
- Header pins
- 24 pin IC socket adapter
- Ribbon cable

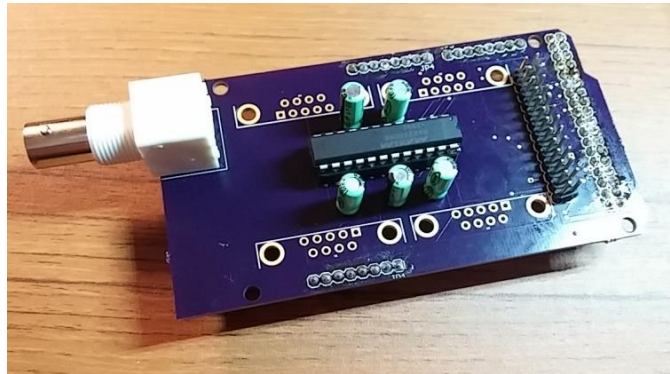
Some basic tools are also needed. Basic soldering supplies such as a soldering iron, flux, solder, helping hands, etc. are required, as well as a wire stripper and a Dremel tool. If you are building this device and don't have access to the original schematics, layout, or code, you can refer to appendix B for schematics and board layouts, and appendix C for the code.

The board layout then needs to be printed, either by an in-house PCB milling machine or by being sent to a fabrication house. Getting the board sent to a fabrication house is preferable as, in my experience, in-house milled boards can be problematic. The board pictured here was printed at OSH Park (<https://oshpark.com/>).



First, solder the IC socket in place, as well as the capacitors. Be sure they are the correct polarity: refer to appendix B. Solder the headers into place on the opposite side of

the board. The BNC connector can also be attached at this point. After soldering, the MAX238 IC can be placed in the socket adapter. The following image shows the board with all components soldered in place.



Next, ribbon cable should be used to attach the ports to the board. For the serial ports, cut a short strip of 5 wires and strip both ends of the wires. You can either separate the wires using a blade and individually strip them or, if you have one available, a wire stripper that can handle multiple wires at once can be used. One end should be soldered to the top row of pins on the DE-9 connector, while the other end should be soldered to either a single row of headers or the headers can be soldered to the board first and a press connector can be attached to the other side of the ribbon cable. The press connector option allows for the device to be taken apart if necessary. The press connector is attached by setting the connector, with the unstripped cable inside, in a vice, which is then slowly closed on the connector and cable inside. Repeat the process with the DB-25

connector for the parallel port. Electrical tape is used to separate the rows of wires and electrically isolate them from each other. Heat shrink wrap can also be used. The following is a parallel port connector being assembled.



Here is a picture of a DE-9 connector attached to a row of headers. The bottom line of pins on the connector is not attached to anything because they are not used.



The Arduino board then needs to be programmed. This can be done in a number of ways. The way used during development of NeuroHub was using a simple in circuit serial programmer (ICSP), the USBtinyISP. It is an option that is inexpensive, easy to assemble, and simple to use. The kit can be purchased from <http://www.adafruit.com/> and is compatible with AVRDUDE (<http://www.nongnu.org/avrdude/>), a utility used to download, upload, or otherwise manipulate the ROM and EEPROM contents of AVR microcontrollers. The following steps will outline the programming process using these tools, although many other options are available.

After assembling the programmer as the instructions provided outline and setting up AVRDUDE, the code must be compiled and uploaded to the board. Open the Atmel Studio solution or, if the code is not available, start a new one in C for the Atmel ATmega2560 and copy the code in appendix C. After compiling the project, connect the programmer to the board, open a Windows command line and change the directory to where the .hex file was compiled (should be in the Release folder in the folder where the

project is saved). Check the configuration by using the command “`avrdude -c usbtiny -p m2560`”. The response should not indicate an initialization fail or not being able to find the USB device, USBtinyISP. If there are no errors (consult the AVRDUDE and USPTinyISP online information if there is), using the command “`avrdude -c usbtiny -p m2560 -U flash:w:filename.hex`” where *filename* is the name of the project. This will tell the programmer to write the .hex file (the compiled code) to flash memory on the microcontroller.

Alternatively, the board can be programmed using the Arduino IDE, although the code has not been developed or tested. This would avoid requiring the programmer, but could possibly introduce lag time due to the overhead required by Arduino.

The plastic case must be modified to fit the electronics inside. This can be done using a Dremel tool. Mark the locations where the headers are and trace the back (behind the plating) of the connector on the casing. Starting holes can then be drilled, and the hole cleared with an appropriate Dremel bit. The pictures below show the plastic case after modification.



Make sure the connectors fit inside their locations, then mark the screw locations and drill pilot holes. If the headers for the connectors are attached to the ribbon cable, put them through their holes and put them in their positions. They can then be soldered in place, then the connectors can be screwed in as well. If push connectors were used, the connectors can be screwed to the plastic case, and the connectors hooked up to the headers that were previously soldered to the board. The Arduino can be fit into the bottom half of the casing and the top half fit in place, but only after the plastic posts that are in the way of the top portion of the device are cut in half.

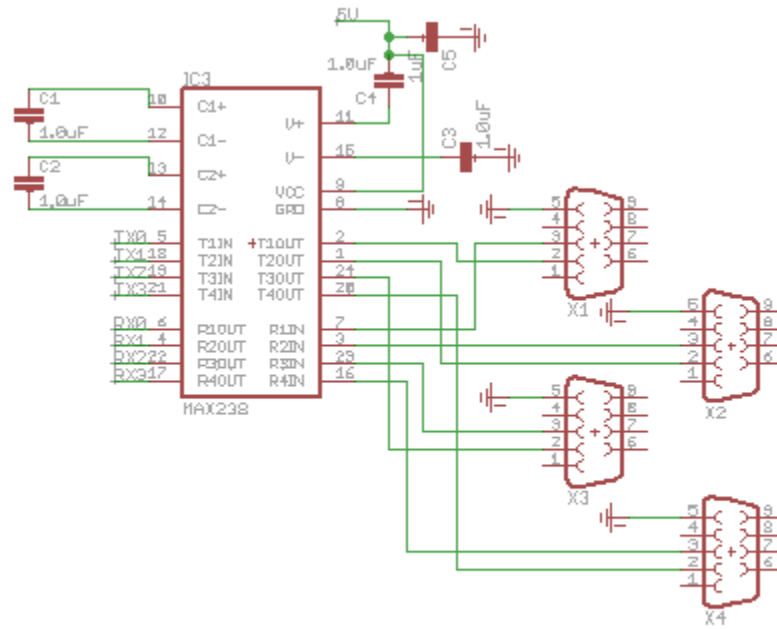


During assembly of the second generation of NeuroHub, possible improvements were noted. Header spacing should replace the DE-9 connectors that were still used from the first generation board, and an “on” LED light should be used. See chapter 6 for more discussion on possible future directions of the project. With more capabilities, the NeuroHub could be an even more useful tool for new hybrid BCI setups.

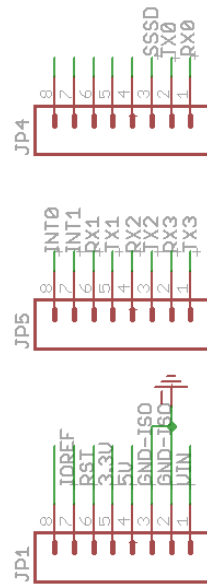


Appendix B: Schematics and Board Layouts

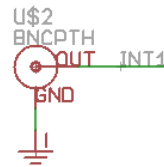
Generation 1



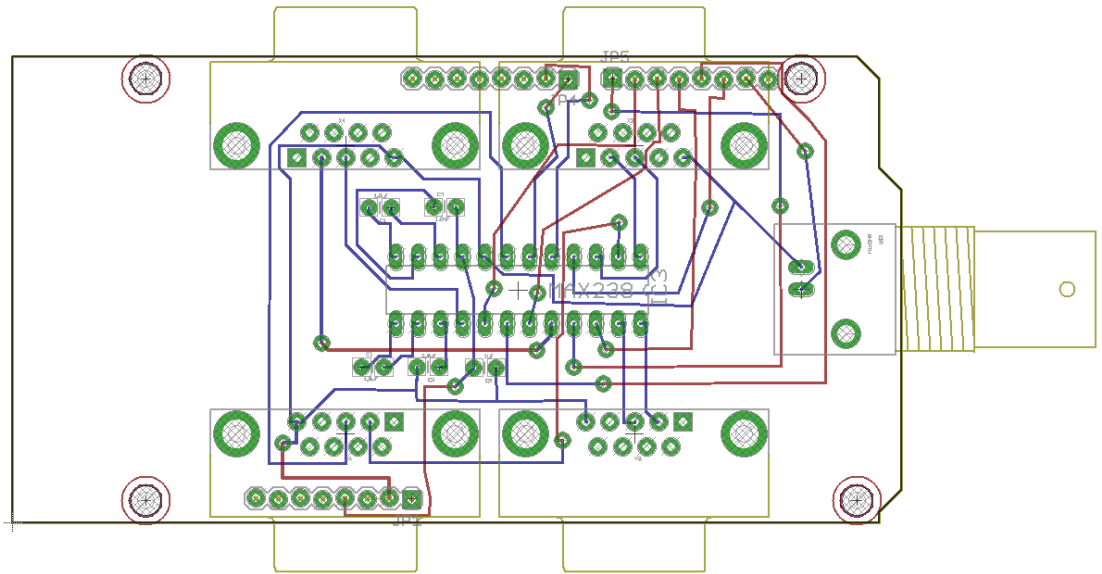
Serial Communication



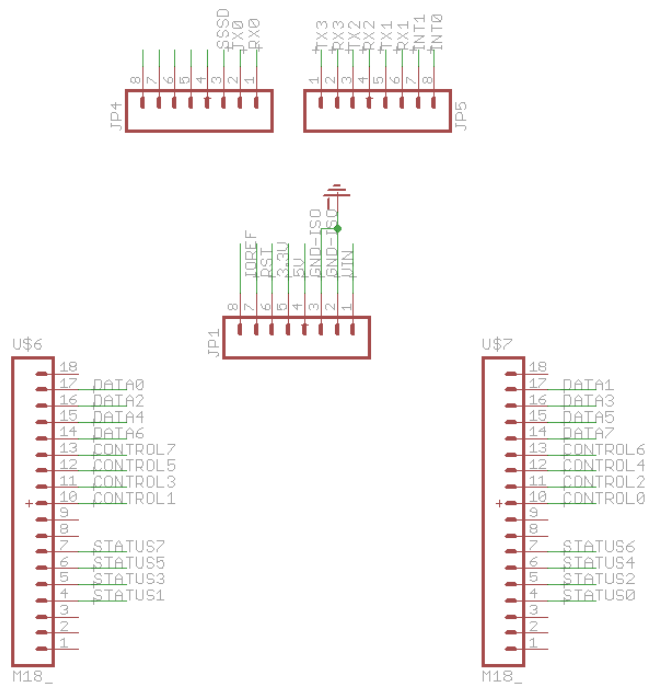
Shield Headers



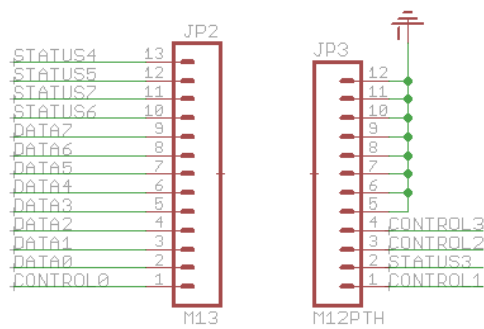
BNC for TTL



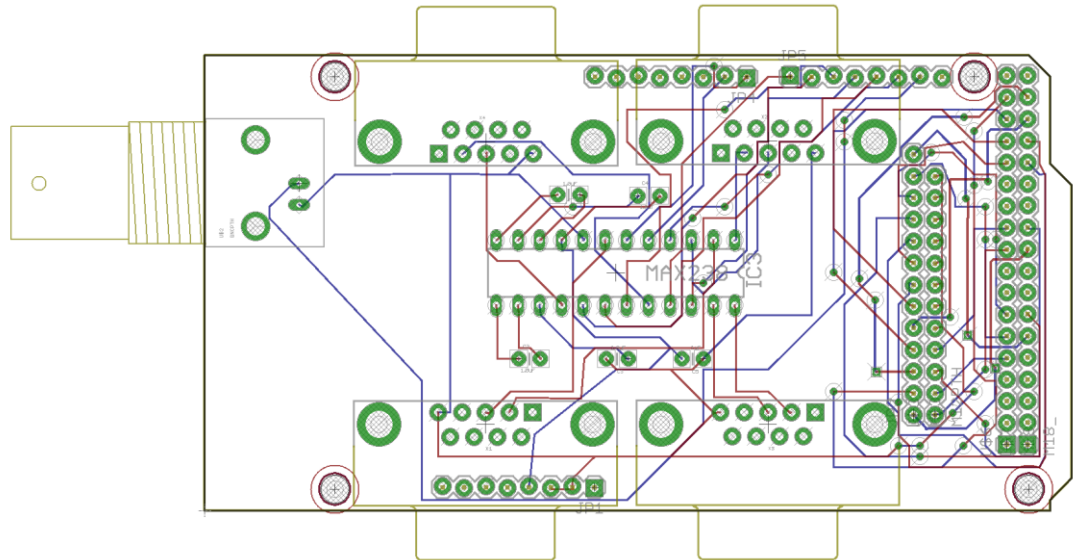
Generation 2



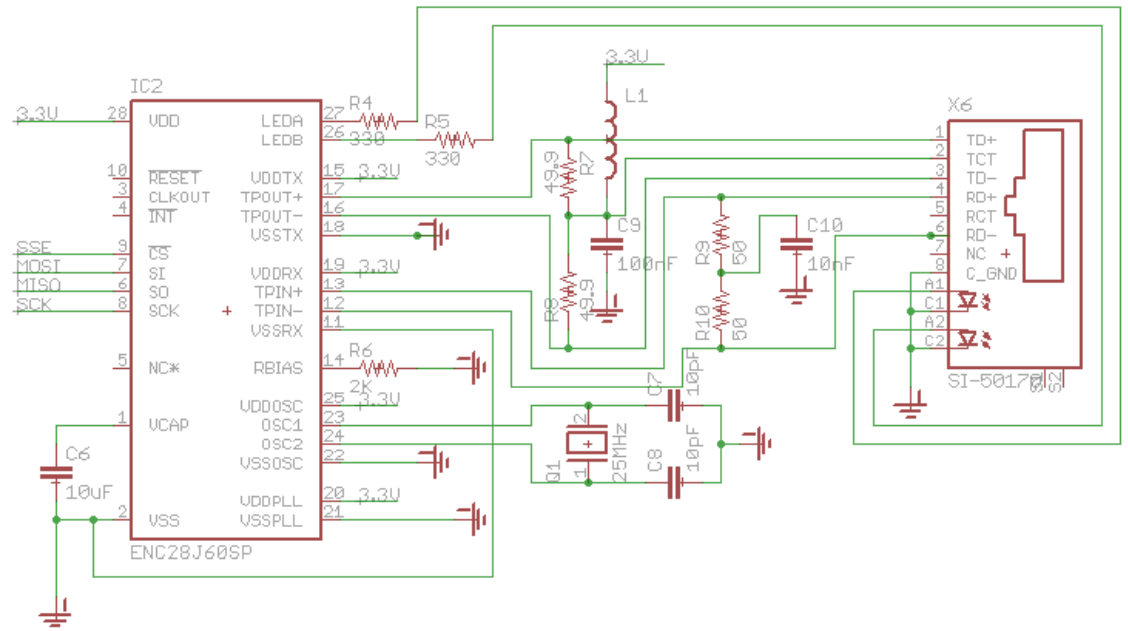
Shield Headers



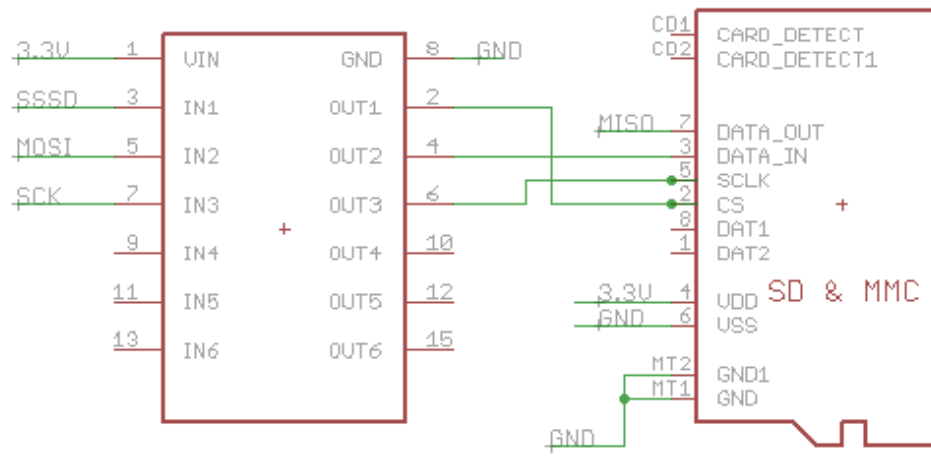
Parallel Communication



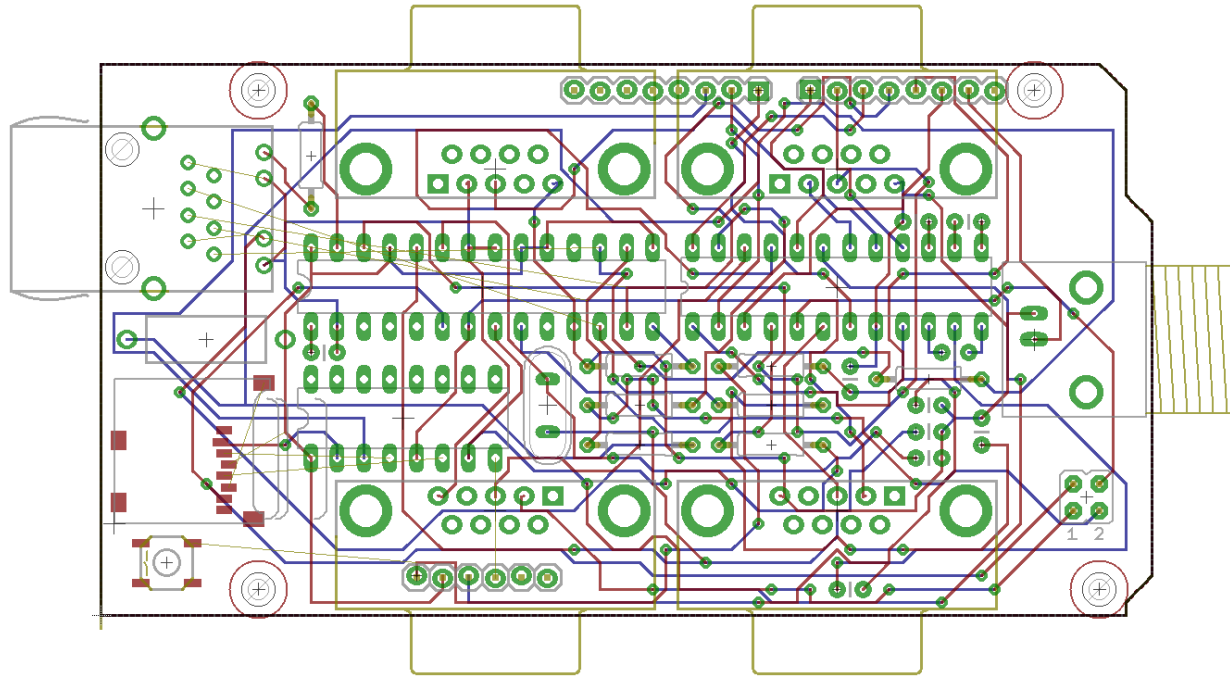
Generation 3



Ethernet



Micro SD slot



Appendix C: Source Code

Microcontroller Code

```
#define F_CPU 16000000UL
#include <avr/io.h>
#include <util/delay.h>
#include <avr/interrupt.h>

#define BAUDRATE 9600
#define BAUD_PRESCALLER (((F_CPU / (BAUDRATE * 16UL))) - 1)

//Declaration of our functions
void USART_init(void);
void parallel_init(void);
void TTL_init(void);

int main(void){
    USART_init();    //Call the USART initialization
    parallel_init();
    TTL_init();

    sei();

    while(1)
    {
        _delay_ms(5);
        PORTA = 0x00;
    }

    return 0;
}

void parallel_init(void){
    // Parallel Port setup
    DDRA = 0xFF; // Make all of PORTA output, this is the data line
    DDRE |= (1<<PE5); // Make PE pin5, or pin 3 on Arduino, output, this is strobe
line
    DDRE &= ~(1<<PE4); // make PE4 pin4, or pin 2 on Arduino, input, this is busy
line
    PORTE |= (1<<PE4); // Activate pull-ups in PORTE pin 4
```

```

}

void TTL_init(void){
    //BNC setup
    DDRD &= ~(1<<PD1); //Configure PORTD pin 1 (pin 20 or INT1) as an input
to check the level
    PORTD |= (1<<PD1); //enable pullup resistor
    EICRA=(EICRA&(~(0<<ISC11|1<<ISC10))|(0<<ISC11|1<<ISC10));
//Configure INT1 to sense any edge
    EIMSK|=(1<<INT1); //Enable INT1 interrupt
}

void USART_init(void){

    UBRR0H = (uint8_t)(BAUD_PRESCALLER>>8); // Register where we set the
baudrate
    UBRR0L = (uint8_t)(BAUD_PRESCALLER);
    UCSR0B = (1<<RXEN0)|(1<<TXEN0)| (1<<RXCIE0); // Activate the RX and
TX pins, and enable UART interrupts
    UCSR0C = (3<<UCSZ00); // sets USCZ00 to 1 and USCZ01 to 1 as well,

    UBRR1H = (uint8_t)(BAUD_PRESCALLER>>8);
    UBRR1L = (uint8_t)(BAUD_PRESCALLER);
    UCSR1B = (1<<RXEN1)|(1<<TXEN1)| (1<<RXCIE1);
    UCSR1C = (3<<UCSZ00);

    UBRR2H = (uint8_t)(BAUD_PRESCALLER>>8);
    UBRR2L = (uint8_t)(BAUD_PRESCALLER);
    UCSR2B = (1<<RXEN2)|(1<<TXEN2)| (1<<RXCIE2);
    UCSR2C = (3<<UCSZ00);

    UBRR3H = (uint8_t)(BAUD_PRESCALLER>>8);
    UBRR3L = (uint8_t)(BAUD_PRESCALLER);
    UCSR3B = (1<<RXEN3)|(1<<TXEN3)| (1<<RXCIE3);
    UCSR3C = (3<<UCSZ00);

}

ISR(INT0_vect)
{
    cli();
    char ReceivedByte;

    if((PIND & (1<<PD1)) == 0)

```

```
{
    ReceivedByte = 0x30; // Send '0'
}
else
{
    ReceivedByte = 0x31; // Send '1'
}

// Write to parallel port
PORTA = ReceivedByte; // write byte to the data port

// For SPP protocol
while(PE4 != 1){} // is the busy line low? if its high, wait
PORTE &= ~(0<<PB5); //when it is low, pull strobe low
//

// Serial send
UDR0 = ReceivedByte;
UDR1 = ReceivedByte;
UDR2 = ReceivedByte;
UDR3 = ReceivedByte;
//

// Finishing SPP protocol
PORTE &= ~(1<<PE5); // pull strobe high
// PORTA = 0x00;
//

sei();
}

ISR(USART0_RX_vect) // USART0 Received Byte Interrupt
{
    cli();
    char ReceivedByte;
    ReceivedByte = UDR0;

    // Write to parallel port
    PORTA = ReceivedByte; // write byte to the data port

    // For SPP protocol
    // while(PE4 != 1){} // is the busy line low? if its high, wait
    // PORTE &= ~(0<<PB5); //when it is low, pull strobe low
    //
```



```
// Serial send
UDR1 = ReceivedByte;
UDR2 = ReceivedByte;
UDR3 = ReceivedByte;
//

// Finishing SPP protocol
// PORTE &= ~(1<<PE5);// pull strobe high
// PORTA = 0x00;
//

sei();
}

ISR(USART1_RX_vect)
{
cli();
char ReceivedByte;
ReceivedByte = UDR1;

// Write to parallel port
PORTA = ReceivedByte;// write byte to the data port

// For SPP protocol
// while(PE4 != 1){} // is the busy line low? if its high, wait
// PORTE &= ~(0<<PB5); //when it is low, pull strobe low
//

// Serial send
UDR0 = ReceivedByte;
UDR2 = ReceivedByte;
UDR3 = ReceivedByte;
//

// Finishing SPP protocol
// PORTE &= ~(1<<PE5);// pull strobe high
// PORTA = 0x00;
//

sei();
}
```

```
ISR(USART2_RX_vect)
{
    cli();
    char ReceivedByte;
    ReceivedByte = UDR2;

    // Write to parallel port
    PORTA = ReceivedByte;// write byte to the data port

    // For SPP protocol
    // while(PE4 != 1){} // is the busy line low? if its high, wait
    // PORTE &= ~(0<<PB5); //when it is low, pull strobe low
    //

    // Serial send
    UDR0 = ReceivedByte;
    UDR1 = ReceivedByte;
    UDR3 = ReceivedByte;
    //

    // Finishing SPP protocol
    // PORTE &= ~(1<<PE5);// pull strobe high
    // PORTA = 0x00;
    //

    sei();
}

ISR(USART3_RX_vect)
{
    cli();
    char ReceivedByte;
    ReceivedByte = UDR3;

    // Write to parallel port
    PORTA = ReceivedByte;// write byte to the data port

    // For SPP protocol
    //while(PE4 != 1){} // is the busy line low? if its high, wait
    //PORTE &= ~(0<<PB5); //when it is low, pull strobe low
    //

    // Serial send
```

```
UDR0 = ReceivedByte;
UDR1 = ReceivedByte;
UDR2 = ReceivedByte;
//

//Finishing SPP protocol
//PORTE &= ~(1<<PE5);// pull strobe high
//PORTA = 0x00;
//

sei();
}

ISR(INT1_vect)
{
    /*
    THIS IS HERE FOR WHEN EEP IS DEVELOPED
    */
}
```

Time Testing Code

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.IO.Ports;
using System.IO;
using System.Diagnostics;
using System.Threading;

namespace DeviceTest
{
    public partial class Form1 : Form
    {
        ///Declare variables

        // Start stopwatch
        Stopwatch stopwatch = new Stopwatch();

        decimal dataPoints;
        bool correctByte;

        //Open a stringbuilder
        StringBuilder sb = new StringBuilder();

        Random random = new Random();

        // Declare byte to send
        byte[] sentByte = new byte[1];
        byte[] receivedByte = new byte[1]; // read byte allocation

        public Form1()
        {
            InitializeComponent();
        }

        private void Form1_Load(object sender, EventArgs e)
        {
```

```
{
    /// Load up COM port settings
    List<String> tList = new List<String>();
    comPortOut.Items.Clear();
    foreach (string s in SerialPort.GetPortNames())
    {
        tList.Add(s);
    }
    tList.Sort();

    //comPortOut.Items.Add("Select COM port"); // COM OUT loadup
    comPortOut.Items.AddRange(tList.ToArray());
    comPortOut.SelectedIndex = 0;
    //comPortIn.Items.Add("Select COM port"); // COM IN loadup
    comPortIn.Items.AddRange(tList.ToArray());
    comPortIn.SelectedIndex = 1;
}
}

private void startButton_Click(object sender, EventArgs e)
{
    /// Disable all options
    comPortOut.Enabled = false;
    comPortIn.Enabled = false;
    startButton.Enabled = false;
    exitButton.Enabled = false;
    /// Enable the stop button
    stopButton.Enabled = true;

    /// Set the serial ports to selected values
    serialPort1.PortName = comPortOut.SelectedItem.ToString();
    serialPort1.BaudRate = 9600;
    serialPort1.Open();
    serialPort2.PortName = comPortIn.SelectedItem.ToString();
    serialPort2.BaudRate = 9600;
    serialPort2.Open();

    /// Testing prep and warmup
    Process.GetCurrentProcess().ProcessorAffinity = new IntPtr(2); // Uses the
second core
    Process.GetCurrentProcess().PriorityClass = ProcessPriorityClass.High; //
Prevents other processes from interrupting Threads
```

```
Thread.CurrentThread.Priority = ThreadPriority.Highest; // Prevents other threads
from interrupting this thread

textBox1.AppendText(Environment.NewLine);

textBox1.AppendText("Warmup");
textBox1.AppendText(Environment.NewLine);
stopwatch.Reset();
stopwatch.Start();

// Clear the stringbuilder (for multiple testing in one session)
sb.Clear();

while (stopwatch.ElapsedMilliseconds < 1500) // A warmup of 1500ms to
stabilize the CPU cache and pipeline
{ }
stopwatch.Stop();

textBox1.AppendText("Beginning Testing");
textBox1.AppendText(Environment.NewLine);

// Testing begins here
serialPort2.DiscardInBuffer();
serialPort1.DiscardOutBuffer();
serialPort1.DiscardInBuffer();
serialPort2.DiscardOutBuffer();

for (int repeat = 0; repeat <= dataPoints; ++repeat)
{

    //sentByte = Convert.ToChar(Convert.ToInt32(Math.Floor(26 *
random.NextDouble() + 65)));
    sentByte[0] = (Byte)random.Next(255);
    textBox1.AppendText("Write byte: " + sentByte.ToString());
    textBox1.AppendText(Environment.NewLine);

    stopwatch.Reset();

    // Test Start
    serialPort1.Write(sentByte, 0, 1); //Send the byte
    stopwatch.Start();
    while (serialPort2.BytesToRead == 0)
    { }
    stopwatch.Stop();
```

```
serialPort2.Read(receivedByte, 0, 1);
// Test Stop

textBox1.AppendText("Read byte: " + receivedByte.ToString());
textBox1.AppendText(Environment.NewLine);

// Is the byte received the same as the one sent out?
if (receivedByte[0] == sentByte[0])
    correctByte = true;
else
    correctByte = false;

// Display information in real time
textBox1.AppendText(Environment.NewLine);
textBox1.AppendText("Elapsed ticks: " + stopwatch.ElapsedTicks + "
Frequency: " + ((stopwatch.ElapsedTicks * 1000.0) / Stopwatch.Frequency) + " Correct?
" + correctByte + stopwatch.ElapsedMilliseconds);
textBox1.AppendText(Environment.NewLine);

// Add to string builder for storing in .txt file
sb.AppendLine(stopwatch.ElapsedTicks + "\t" + correctByte + "\t" +
((stopwatch.ElapsedTicks * 1000.0) / Stopwatch.Frequency)); // Add the new data points
to the StringBuilder
decimal dataLeft = dataPoints - repeat;

// clear serial port buffers
serialPort2.DiscardInBuffer();
serialPort1.DiscardOutBuffer();
}

textBox1.AppendText("Done");
// Testing is complete

textBox1.AppendText(sb.ToString()); // Shows results in the test box for a quick
lookover
StopRecording();
this.Invoke(new EventHandler(SaveDialog));
}

private void exitButton_Click(object sender, EventArgs e)
{
    StopRecording();
    Application.Exit();
}
```

```
private void stopButton_Click(object sender, EventArgs e)
{
    dataPoints = 0;
    StopRecording();
}

public void SaveDialog(object sender, EventArgs e)
{
    /// When the timer runs out or STOP is pressed, a Save Dialog appears
    SaveFileDialog saveFileDialog1 = new SaveFileDialog();

    saveFileDialog1.Filter = "txt files (*.txt)|*.txt|All files (*.*)|*.*";
    saveFileDialog1.FilterIndex = 1;
    saveFileDialog1.RestoreDirectory = true;
    saveFileDialog1.FileName = "DeviceTest0";
    if (saveFileDialog1.ShowDialog(this) == DialogResult.OK)
    {
        File.WriteAllText(saveFileDialog1.FileName, sb.ToString());
    }
}

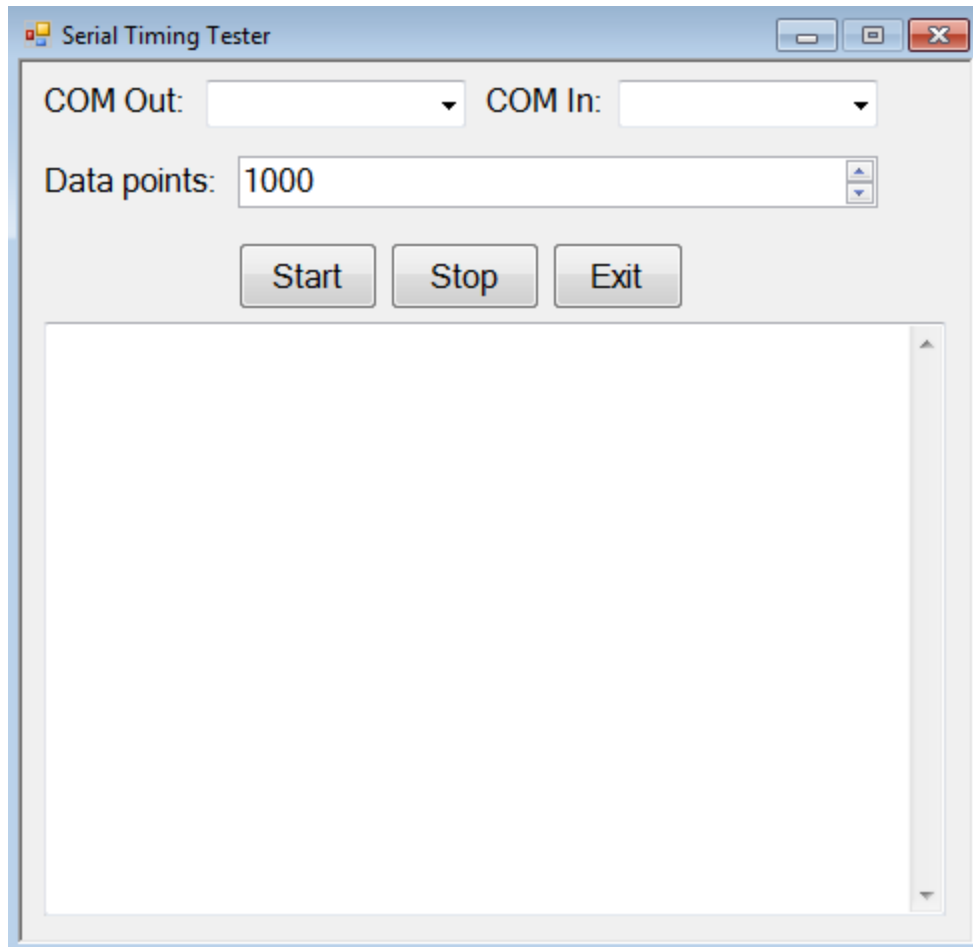
public void StopRecording()
{
    dataPoints = 0;
    // Enable all custom options
    comPortOut.Enabled = true;
    comPortIn.Enabled = true;
    startButton.Enabled = true;
    exitButton.Enabled = true;
    // Diable the stop button
    stopButton.Enabled = false;

    // Close Com ports
    if (serialPort1.IsOpen)
    {
        serialPort1.Close();
    }
    if (serialPort2.IsOpen)
    {
        serialPort2.Close();
    }
}
```



```
private void label1_Click(object sender, EventArgs e)
{
}
}
```

Time Testing GUI



TTL Time Testing Code

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.IO.Ports;
using System.IO;
using System.Diagnostics;
using System.Threading;

namespace fNIRautomation
{
    public partial class Form1 : Form
    {
        // Start stopwatch
        Stopwatch stopwatch = new Stopwatch();

        decimal dataPoints;
        bool correctByte;
        bool byteReceived; // Has a byte been sent? True or false
        int inByte;
        decimal waitTime;
        long totalTicks;
        long stopTicks;
        long startTicks;
        double TTLms;

        //Open a stringbuilder
        StringBuilder sb = new StringBuilder();

        Random random = new Random();

        // Declare byte to send
        byte[] receivedByte = new byte[1];
        byte[] lastByte = new byte[1]; // read byte allocation

        public Form1()
```

```
{
    InitializeComponent();
}

private void Form1_Load(object sender, EventArgs e)
{
    /// Load up COM port settings
    List<String> tList = new List<String>();
    comPortOut.Items.Clear();
    foreach (string s in SerialPort.GetPortNames())
    {
        tList.Add(s);
    }
    tList.Sort();
    comPortOut.Items.AddRange(tList.ToArray());
    comPortOut.SelectedIndex = 0;
}

private void button1_Click_1(object sender, EventArgs e)
{
    /// Disable all custom options
    comPortOut.Enabled = false;

    startButton.Enabled = false;

    /// Set the serial ports to selected values
    serialPort1.PortName = comPortOut.SelectedItem.ToString();
    serialPort1.BaudRate = 9600;

    serialPort1.Open();

    /// Set number of data points to take based on inputted value
    dataPoints = dataPointBox.Value;
    /// waitTime = wait.Value * 1000;

    Process.GetCurrentProcess().ProcessorAffinity = new IntPtr(2); // Uses the
second core
    Process.GetCurrentProcess().PriorityClass = ProcessPriorityClass.High; //
Prevents other processes from interrupting Threads
    Thread.CurrentThread.Priority = ThreadPriority.Highest; // Prevents other threads
from interrupting this thread
    /// Clear the stringbuilder (for multiple testing in one session)
    sb.Clear();
}
```

```
stopwatch.Start();
while (stopwatch.ElapsedMilliseconds < 1500) // A warmup of 1500ms to
stabilize the CPU cache and pipeline
{ }
stopwatch.Stop();

// Testing begins here
serialPort1.DiscardOutBuffer();
serialPort1.DiscardInBuffer();

// First byte received should be 1, so pretend last byte received was 0
lastByte[0] = 48;
stopwatch.Start();

for (int repeat = 0; repeat < dataPoints; ++repeat)
{

    // stopwatch.Reset();
    startTicks = stopwatch.ElapsedTicks; // mark start of test
    // Test Start
    while (serialPort1.BytesToRead == 0)
    { }
    stopTicks = stopwatch.ElapsedTicks; // mark stop of test

    serialPort1.Read(receivedByte, 0, 1);
    serialPort1.DiscardInBuffer();

    // Is the byte received the same as the one sent out?
    if (receivedByte[0] == 49) // received byte was 1
        correctByte = (lastByte[0] == 48);
    else if (receivedByte[0] == 48) // received byte was 0
        correctByte = (lastByte[0] == 49);

    // Store current byte to lastByte to be compared for correctByte received
    lastByte[0] = receivedByte[0];
    totalTicks = stopTicks - startTicks;
    TTLms = 25 - ((totalTicks * 1000.0) / Stopwatch.Frequency);

    // Add to string builder for storing in .txt file
    sb.AppendLine(totalTicks + "\t" + correctByte + "\t" + TTLms); // Add the new
data points to the StringBuilder
    textBox1.AppendText(totalTicks + "\t" + correctByte + "\t" + TTLms); // Add
the new data points to the StringBuilder
```

```
        textBox1.AppendText(Environment.NewLine);

        // clear serial port buffers
        // serialPort1.DiscardOutBuffer();

    }

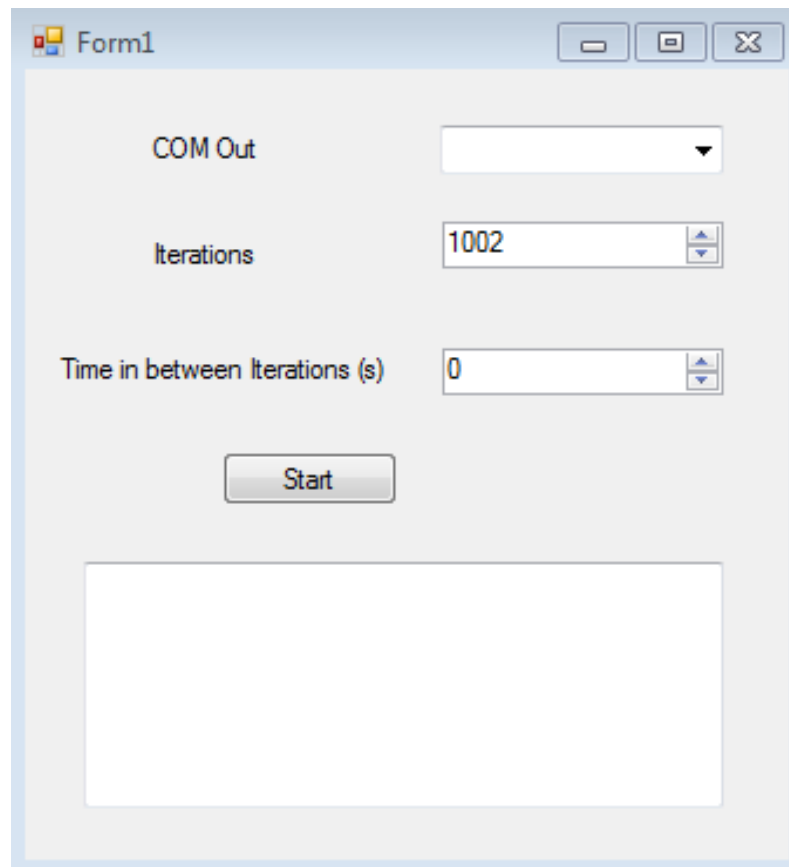
    StopRecording();
    this.Invoke(new EventHandler(SaveDialog));
}

public void SaveDialog(object sender, EventArgs e)
{
    /// When the timer runs out or STOP is pressed, a Save Dialog appears
    SaveFileDialog saveFileDialog1 = new SaveFileDialog();

    saveFileDialog1.Filter = "txt files (*.txt)|*.txt|All files (*.*)|*.*";
    saveFileDialog1.FilterIndex = 1;
    saveFileDialog1.RestoreDirectory = true;
    saveFileDialog1.FileName = "DeviceTest0";
    if (saveFileDialog1.ShowDialog(this) == DialogResult.OK)
    {
        File.WriteAllText(saveFileDialog1.FileName, sb.ToString());
    }
}

public void StopRecording()
{
    // Close Com port
    if (serialPort1.IsOpen)
    {
        serialPort1.Close();
    }
}
}
```

TTL Time Testing GUI



The image shows a Windows-style application window titled "Form1". The window contains three input fields and a button. The first field is a dropdown menu labeled "COM Out". The second field is a numeric spinner box labeled "Iterations" with the value "1002". The third field is another numeric spinner box labeled "Time in between Iterations (s)" with the value "0". Below these fields is a "Start" button. At the bottom of the window is a large, empty rectangular area, likely intended for displaying test results or logs.

MATLAB Analysis Code Example

```
%% Serial to Serial Data Analysis - Computer 1, Windows 7, USB, P1 to P2
% Nicholas Grzczkowski

%% Preparation

% Cleanup workspace
clc, clear, close all

% Identify Testing Configuration
testId = '1, Windows 7, USB, P1 to P2'; %declare computer test number here
height = 170; % height of individual histogram bars, raise if clipping, lower if they look
like lumps
errorSpace = .1; % space around error bar, raise if they look too close to borders

% Import all relevant data in this folder
DeviceTest01 = importdata('DeviceTest01.txt');
DeviceTest02 = importdata('DeviceTest02.txt');
DeviceTest03 = importdata('DeviceTest03.txt');
DeviceTest04 = importdata('DeviceTest04.txt');
DeviceTest05 = importdata('DeviceTest05.txt');
DeviceTest06 = importdata('DeviceTest06.txt');
DeviceTest07 = importdata('DeviceTest07.txt');
DeviceTest08 = importdata('DeviceTest08.txt');
DeviceTest09 = importdata('DeviceTest09.txt');
DeviceTest10 = importdata('DeviceTest10.txt');

% store all in data.mat
save('data');

%% Load data into usable files

% Not loading ticks, just calculated milliseconds
success = [DeviceTest01.textdata(:,2) DeviceTest02.textdata(:,2)
DeviceTest03.textdata(:,2) DeviceTest04.textdata(:,2) DeviceTest05.textdata(:,2)
DeviceTest06.textdata(:,2) DeviceTest07.textdata(:,2) DeviceTest08.textdata(:,2)
DeviceTest09.textdata(:,2) DeviceTest10.textdata(:,2)];
lagtime = [DeviceTest01.data DeviceTest02.data DeviceTest03.data DeviceTest04.data
DeviceTest05.data DeviceTest06.data DeviceTest07.data DeviceTest08.data
DeviceTest09.data DeviceTest10.data];

% compile all lag times into one array
```



```

completeLag = [lagtime(:,1); lagtime(:,2); lagtime(:,3); lagtime(:,4); lagtime(:,5);
lagtime(:,6); lagtime(:,7); lagtime(:,8); lagtime(:,9); lagtime(:,10)];

% Number of bins array
bin = min(lagtime(:)).05:max(lagtime(:));

%% Finding accuracy, averages, standard deviations

% Check if 100% accurate
% Create a cell array of all 'True', then compare to output file
trueCellArray = repmat({'True'},[1001 10]);
allCorrect = isempty(setxor(trueCellArray,success)) %if this is 1, all values are true

for i = 1:10
    meanTests(i) = mean(lagtime(:,i)); % Column 1 will contain averages
    stdTests(i) = std(lagtime(:,i)); % Column 2 will contain std dev
end

%% Plots

% All tests in individual plots
ind_plots = figure;
for i = 1:10
    y = 1:length(lagtime);
    s2(i) = subplot(2,5,i);
    plot(y,lagtime(:,i));
    % All tests in graph
    title(['Test ', num2str(i)])
    axis([0 1000 min(lagtime(:)) max(lagtime(:))])
    xlabel('Iteration')
    ylabel('Lag time (ms)')
end
suptitle(['Plots of All Tests, Computer ', num2str(testId)])
print(ind_plots, '-dpng',['Ind_Plot_comp ',testId]);

% plot of all tests
all_plots = figure;
plot(completeLag)
axis([0 10000 min(lagtime(:)) max(lagtime(:))])
suptitle(['Plots of All Tests, Computer ', num2str(testId)])
xlabel('Iteration')
ylabel('Lag Time (ms)')
print(all_plots, '-dpng',['All_Plot_comp ',testId]);

```

```

%% Histograms

% All tests in individual histograms
ind_hist = figure;
for i = 1:10
    s(i) = subplot(2,5,i);
    hist(s(i),lagtime(:,i),bin)
    axis([0 max(lagtime(:)) 0 height])
    xlabel('Lag time (ms)')
    ylabel('Number of instances')
    title(['Test ', num2str(i)])
end
suptitle(['Histograms of All Tests, Computer ' , testId])
print(ind_hist, '-dpng', ['Ind_Hist_comp ',testId]);

% Histogram of all tests
all_hist = figure;
hist(completeLag,bin);
xlabel('Lag time (ms)')
ylabel('Number of instances')
suptitle(['Histogram of All Tests, Computer ' , testId])
print(all_hist, '-dpng',['All_Hist_comp ',testId]);

%% Bargraph with error bar
bargraph = figure;

bar(meanTests,'r','Edgecolor','b')
axis([0 11 min(meanTests(:))-max(stdTests(:))-errorSpace
max(meanTests(:))+max(stdTests(:))+errorSpace]);
hold;
errorbar(meanTests,stdTests,'k')
suptitle(['Mean Lag Time Across All Tests, Computer ' , num2str(testId)])
xlabel('Test Number')
ylabel('Lag time (ms)')
print(bargraph, '-dpng',['Bargraph_comp ',testId]);

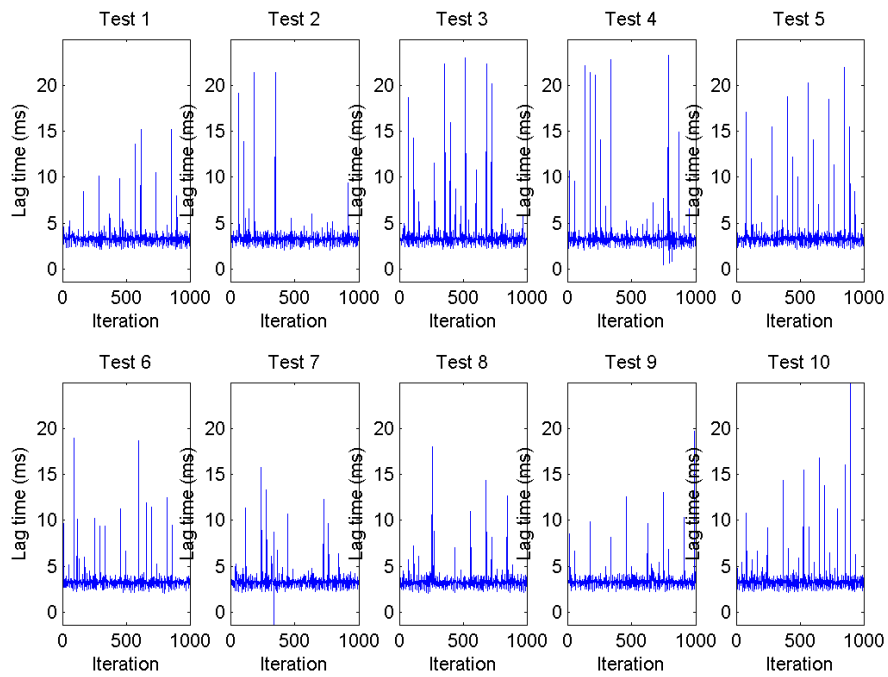
%% Overall calculations
meanAll = mean(lagtime(:)) % mean of all data points
stdAll = std(lagtime(:)) % standard deviation of all data points
stdBetweenTests = std(stdTests) % std deviation between entire test scores

```

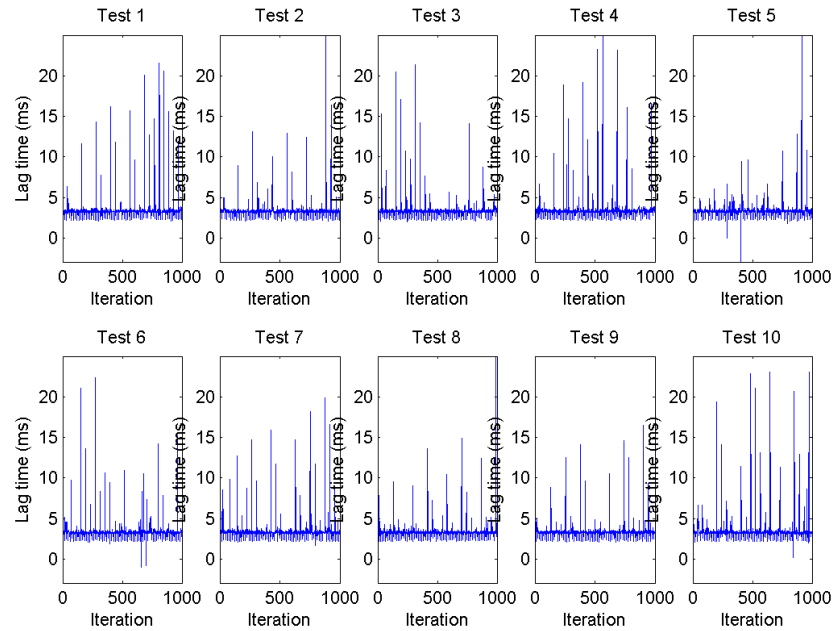
Appendix D: All Testing Results

Serial to Serial and TTL to Serial

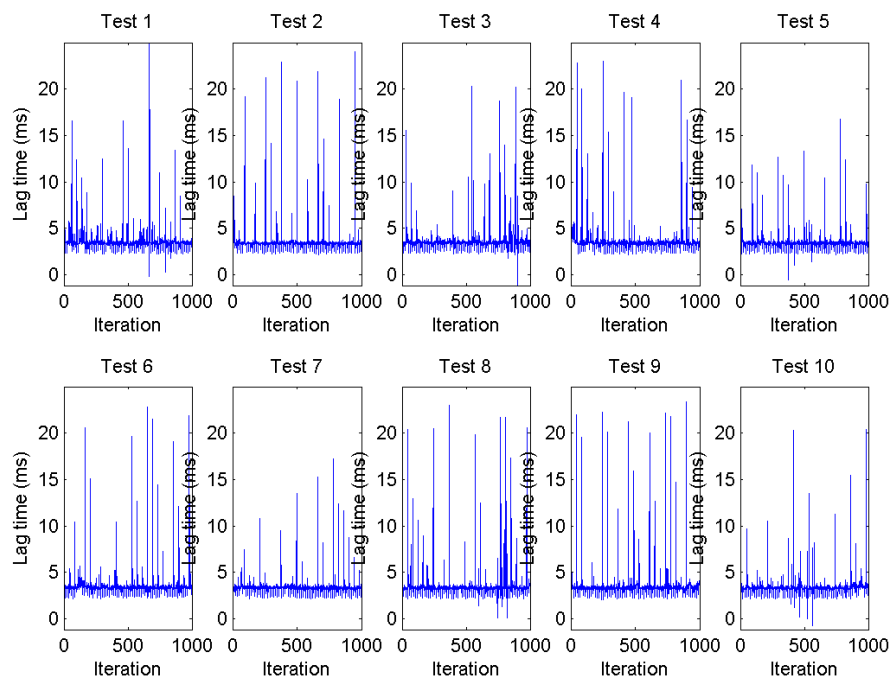
Plots of All Tests, Computer 1, Windows 7, USB, TTL to P3



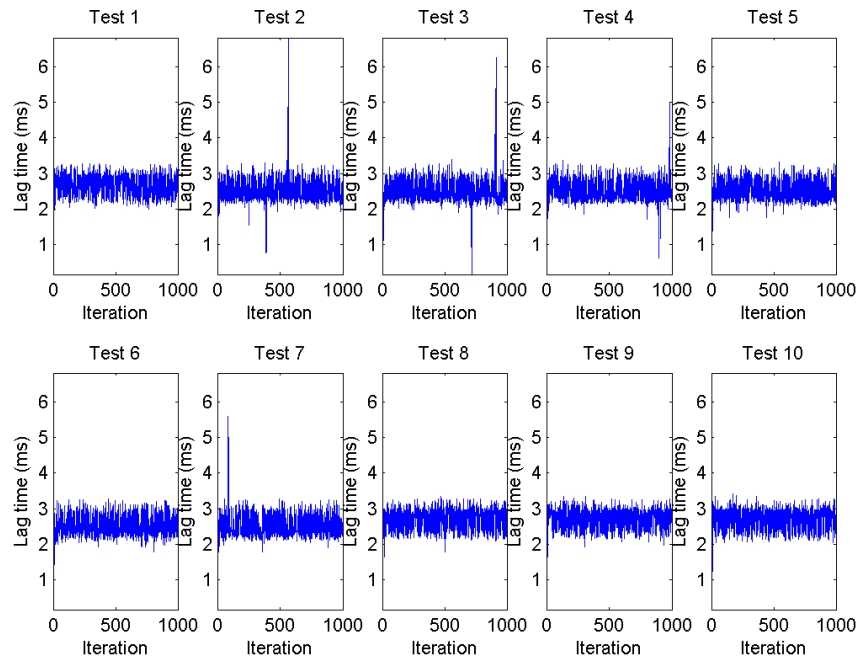
Plots of All Tests, Computer 1, Windows 7, USB, TTL to P2



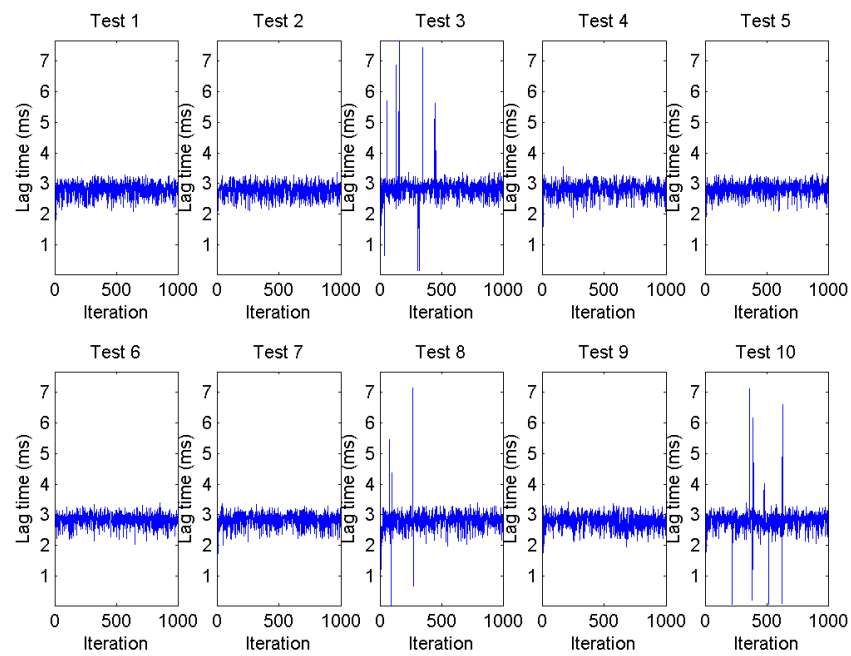
Plots of All Tests, Computer 1, Windows 7, USB, TTL to P1



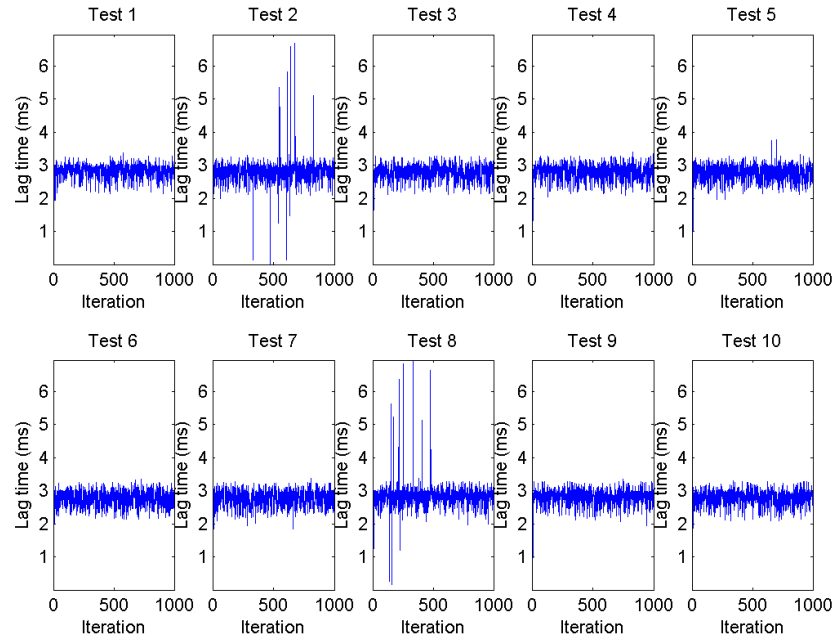
Plots of All Tests, Computer 1, Windows 7, USB, P4 to P3



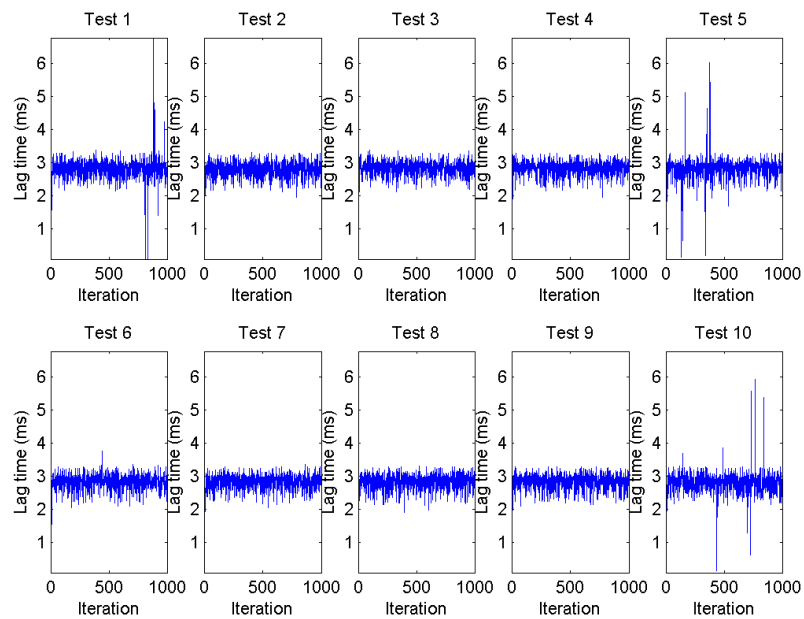
Plots of All Tests, Computer 1, Windows 7, USB, P4 to P2



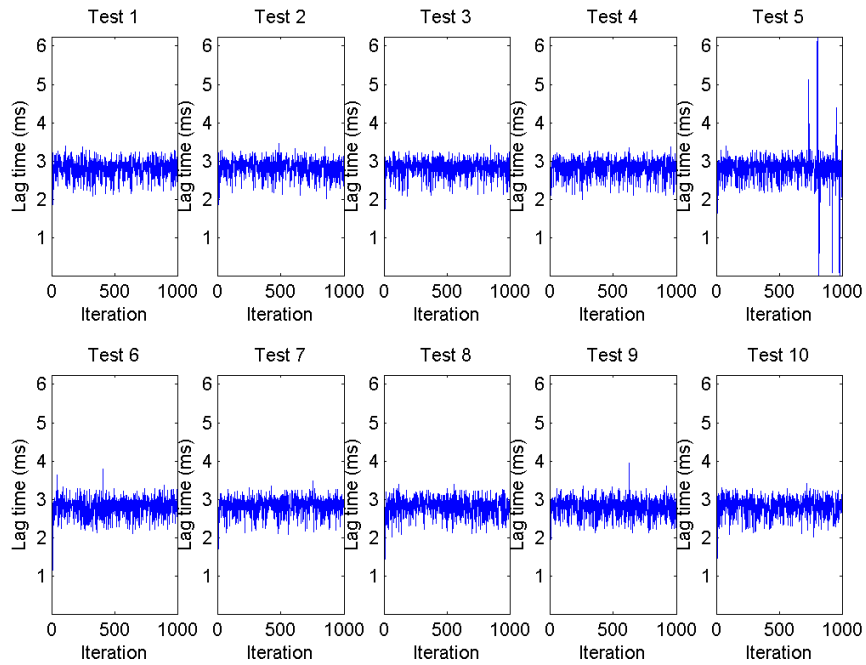
Plots of All Tests, Computer 1, Windows 7, USB, P4 to P1



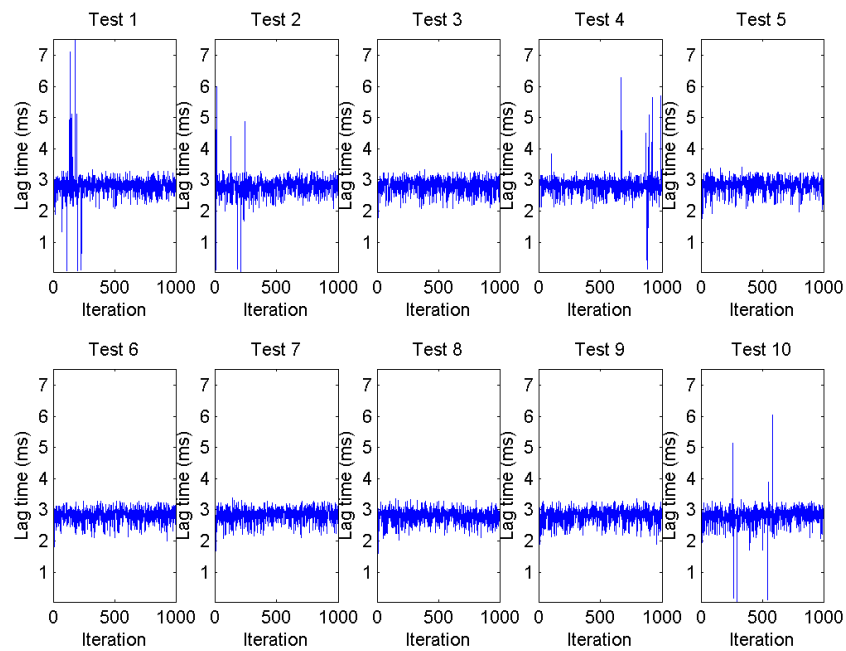
Plots of All Tests, Computer 1, Windows 7, USB, P3 to P4



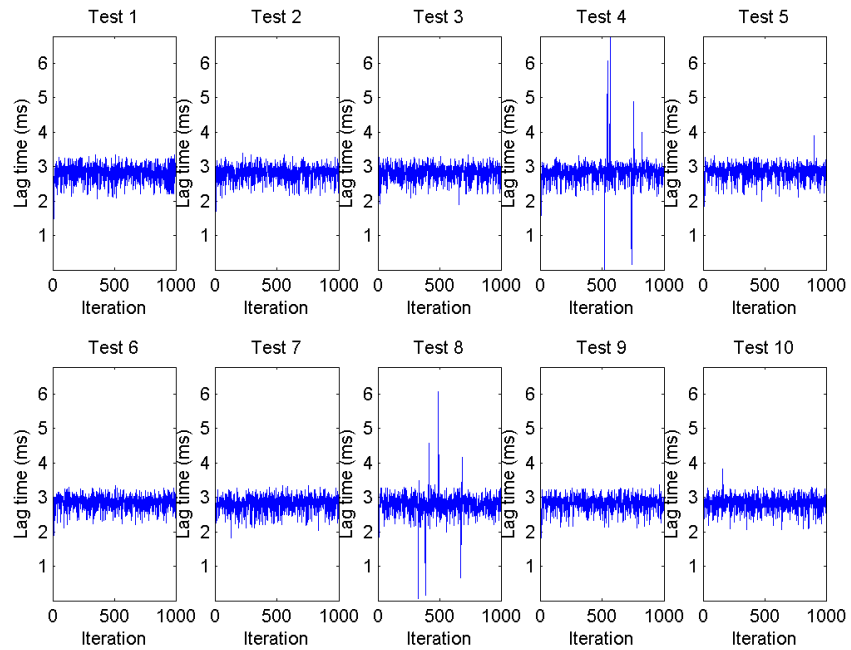
Plots of All Tests, Computer 1, Windows 7, USB, P3 to P2



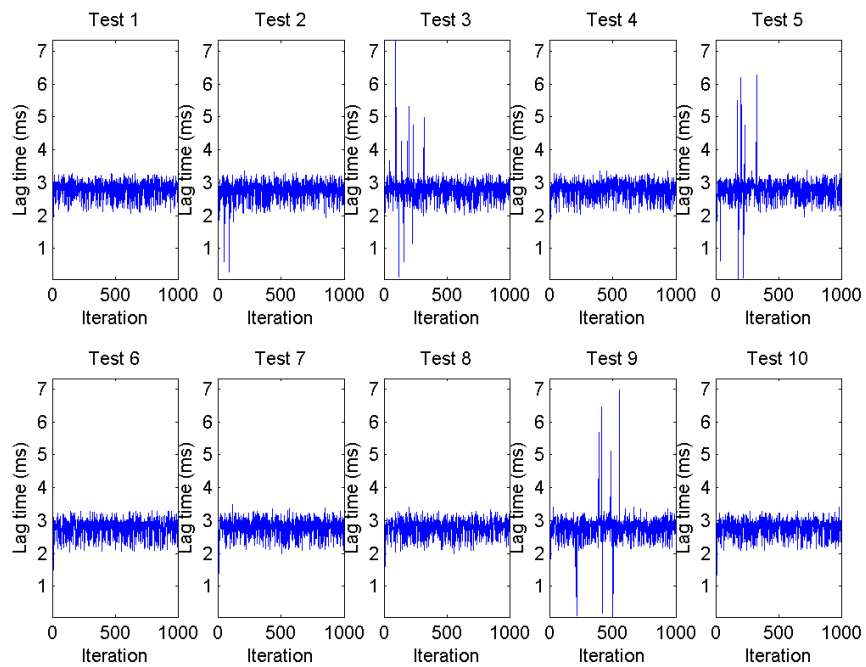
Plots of All Tests, Computer 1, Windows 7, USB, P3 to P1



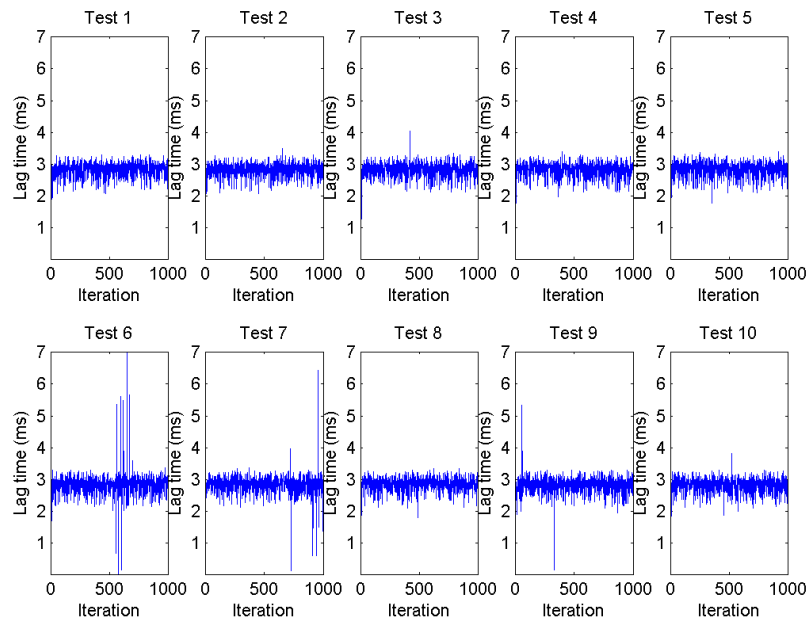
Plots of All Tests, Computer 1, Windows 7, USB, P2 to P4



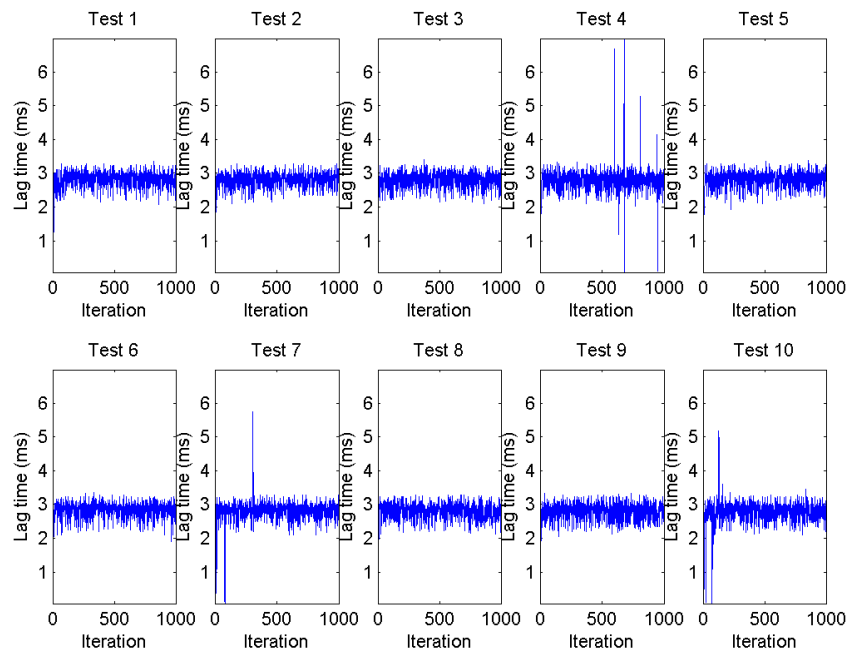
Plots of All Tests, Computer 1, Windows 7, USB, P2 to P3



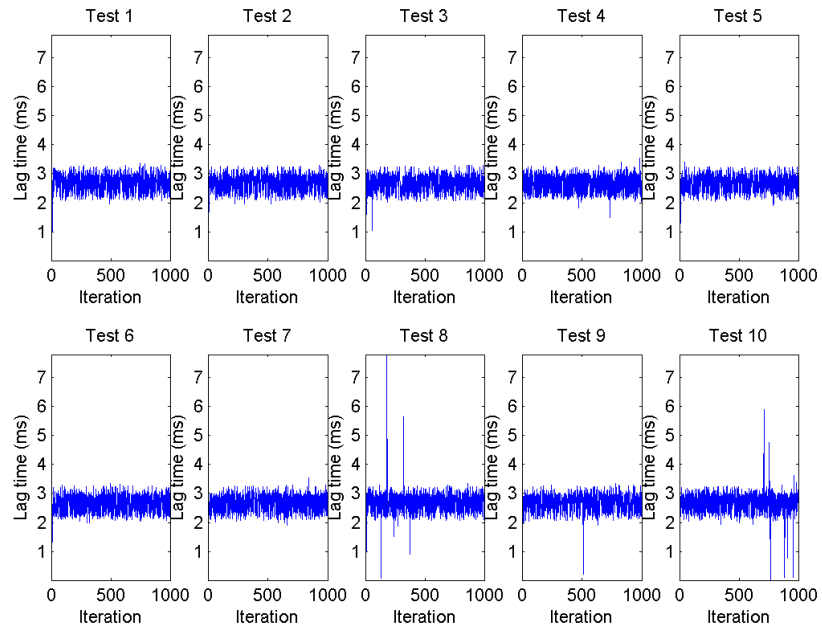
Plots of All Tests, Computer 1, Windows 7, USB, P2 to P1



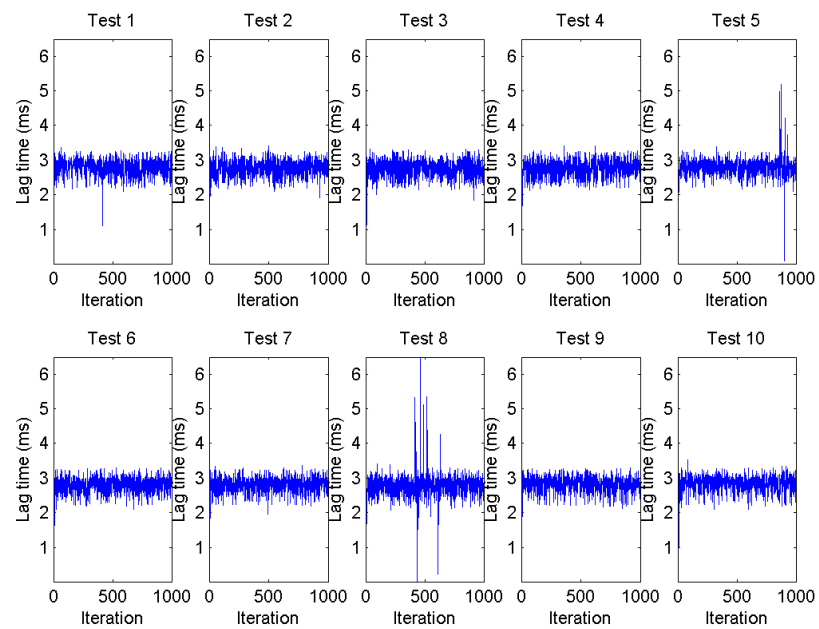
Plots of All Tests, Computer 1, Windows 7, USB, P1 to P4



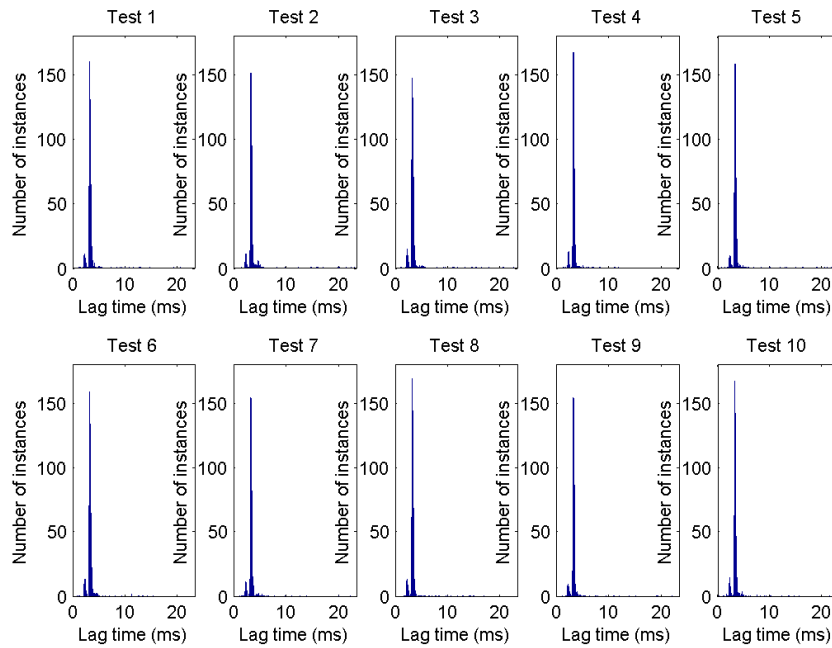
Plots of All Tests, Computer 1, Windows 7, USB, P1 to P3



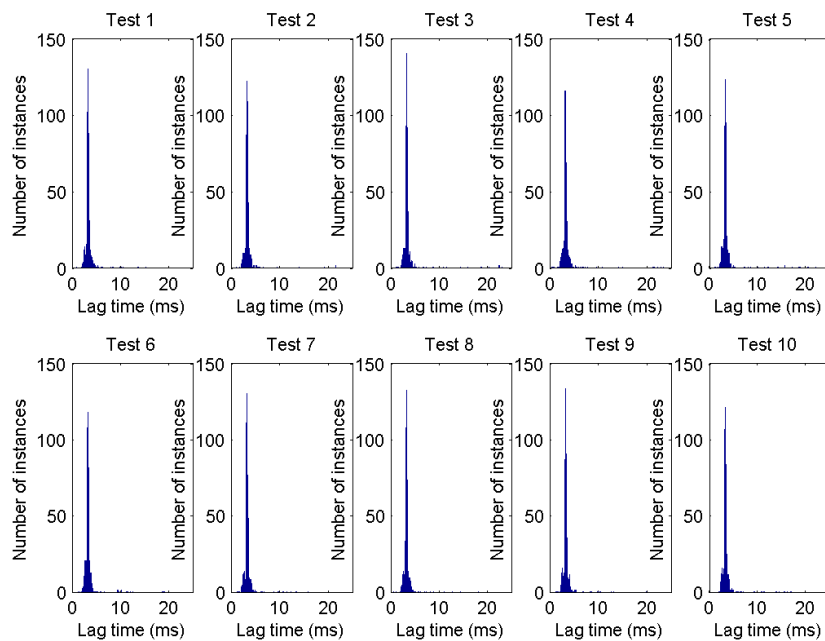
Plots of All Tests, Computer 1, Windows 7, USB, P1 to P2



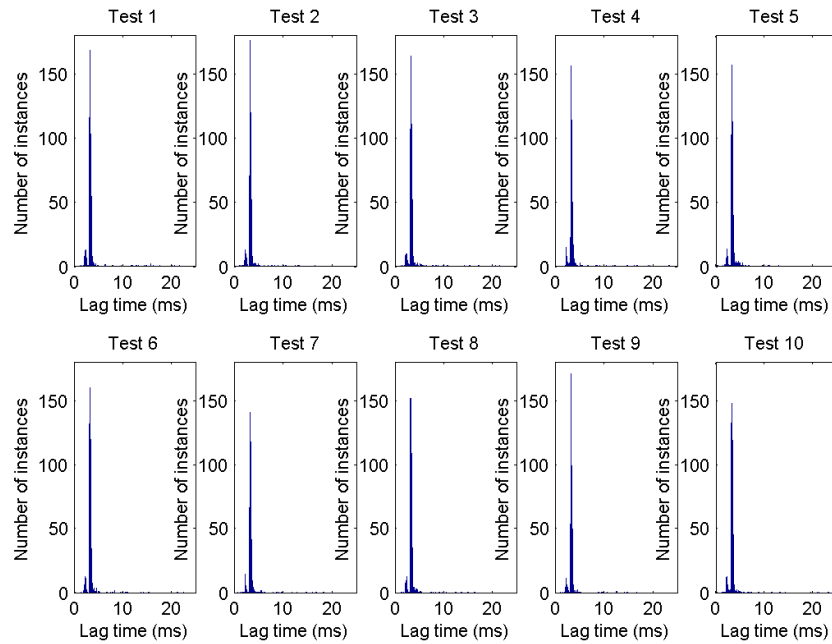
Histograms of All Tests, Computer 1, Windows 7, USB, TTL to P4



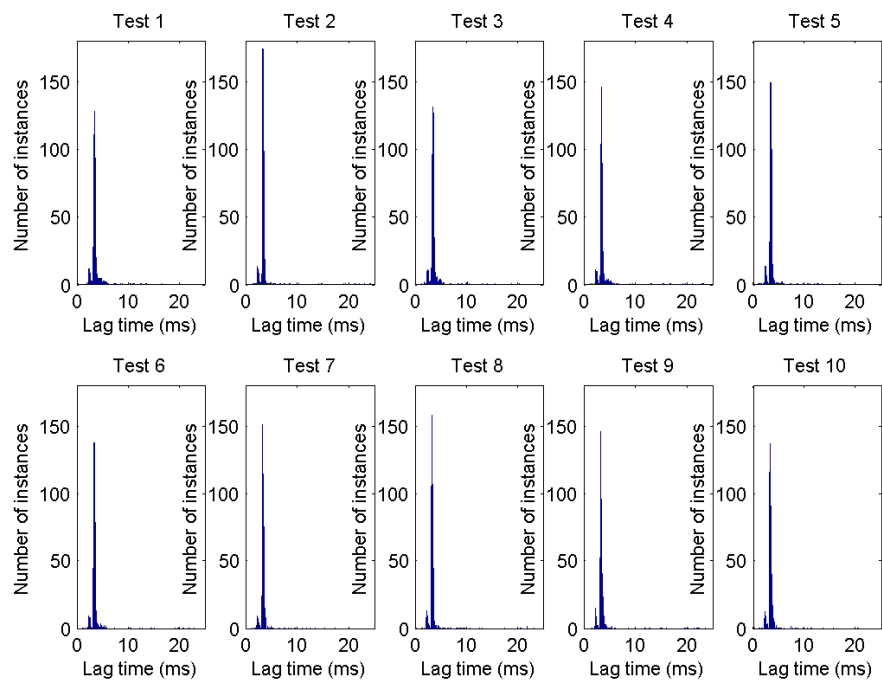
Histograms of All Tests, Computer 1, Windows 7, USB, TTL to P3



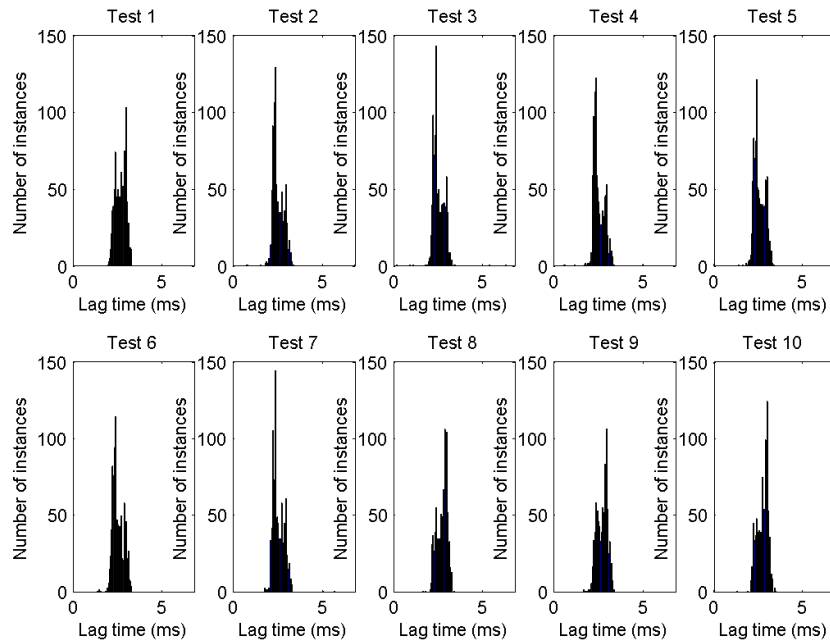
Histograms of All Tests, Computer 1, Windows 7, USB, TTL to P2



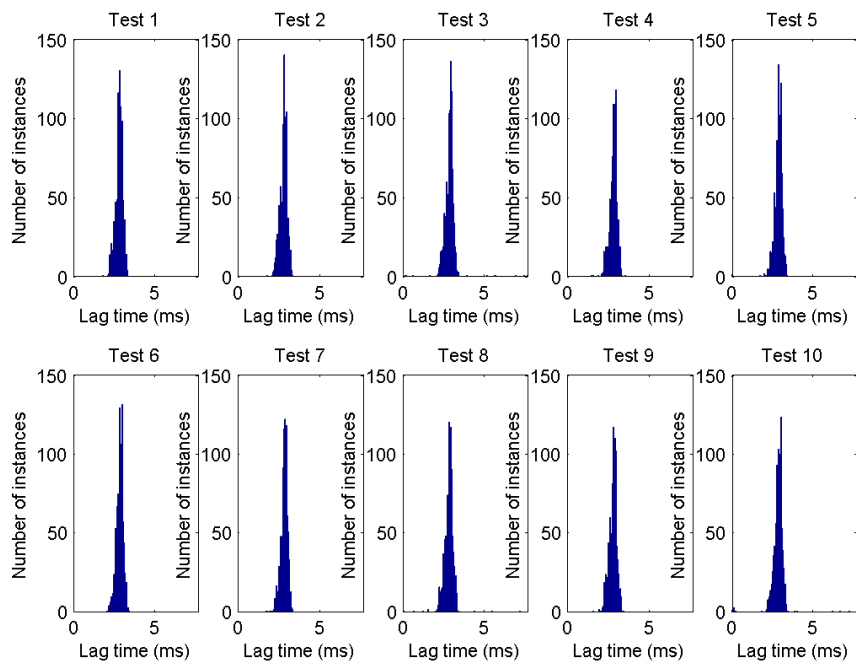
Histograms of All Tests, Computer 1, Windows 7, USB, TTL to P1



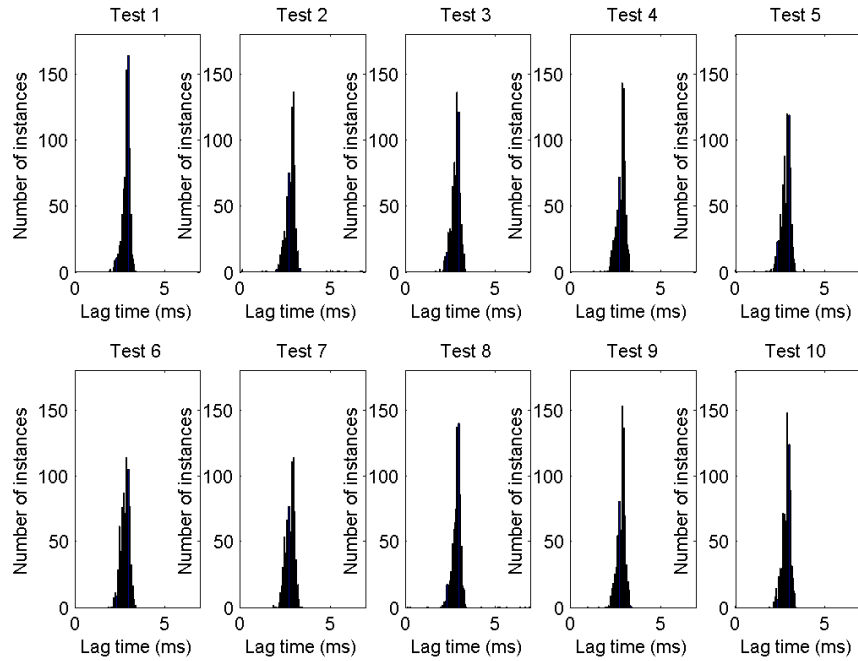
Histograms of All Tests, Computer 1, Windows 7, USB, P4 to P3



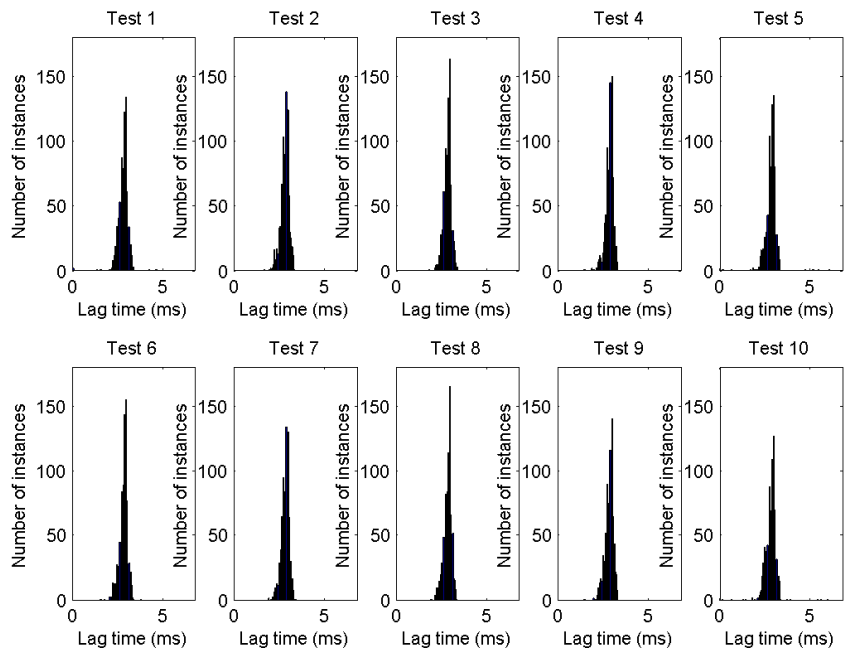
Histograms of All Tests, Computer 1, Windows 7, USB, P4 to P2



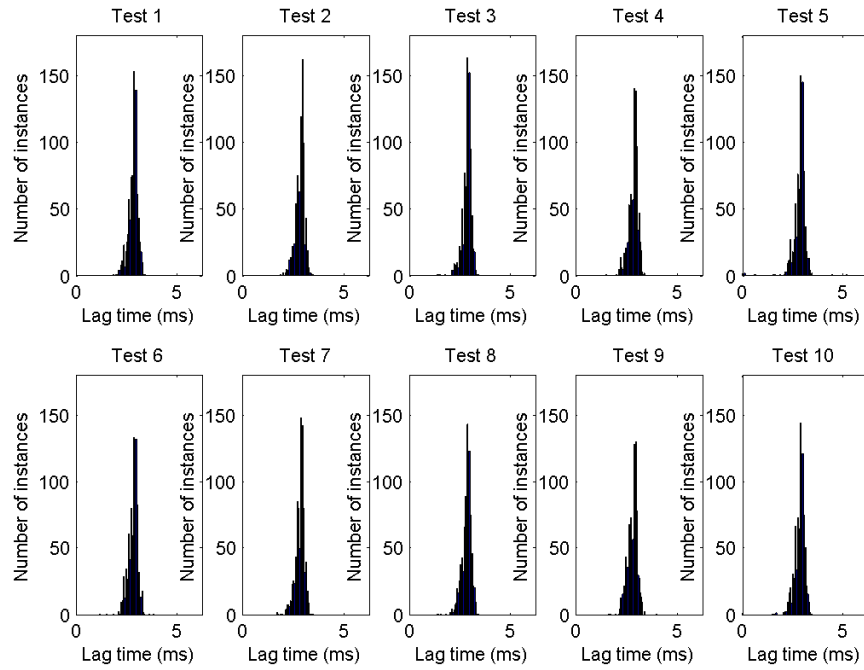
Histograms of All Tests, Computer 1, Windows 7, USB, P4 to P1



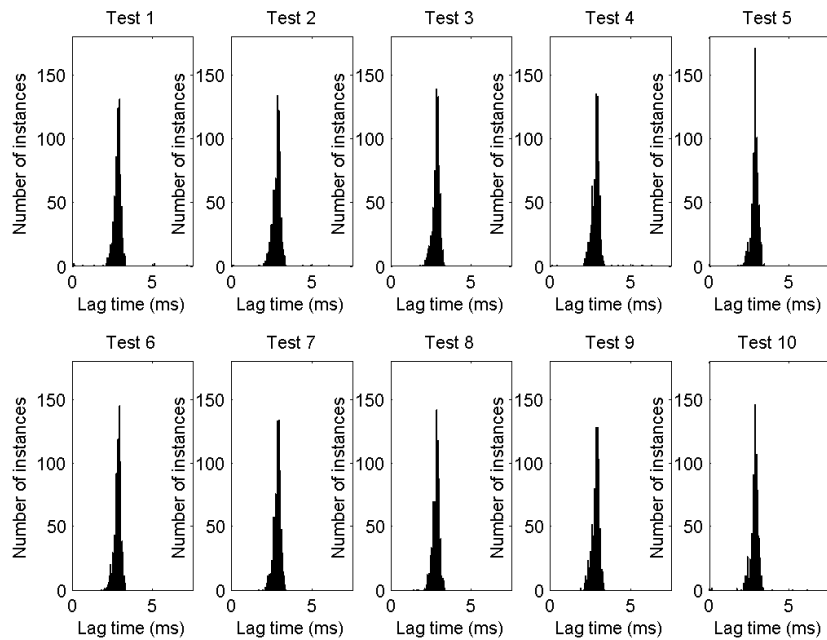
Histograms of All Tests, Computer 1, Windows 7, USB, P3 to P4



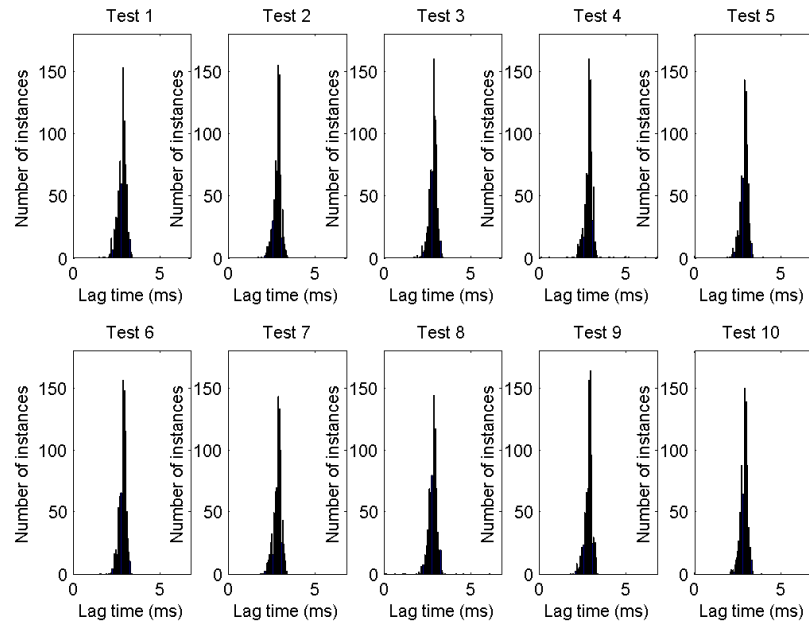
Histograms of All Tests, Computer 1, Windows 7, USB, P3 to P2



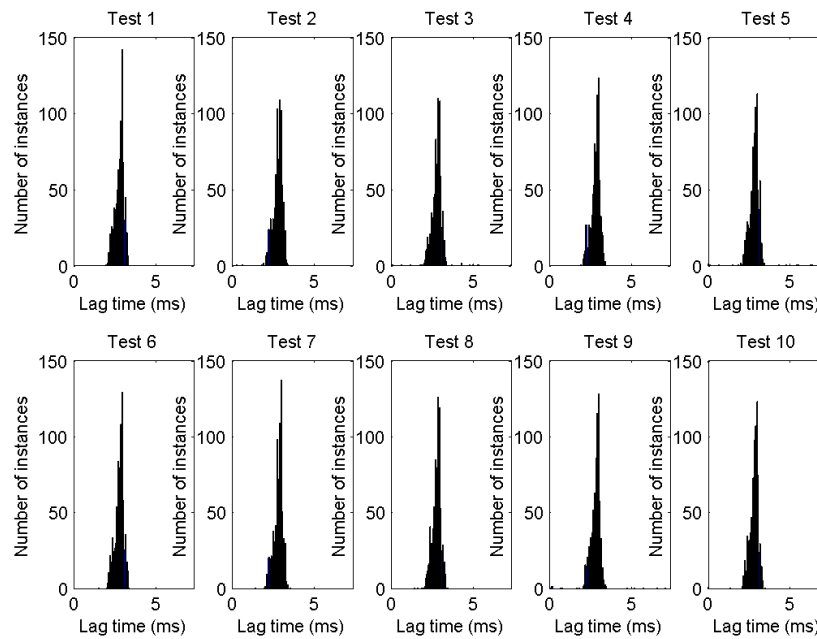
Histograms of All Tests, Computer 1, Windows 7, USB, P3 to P1



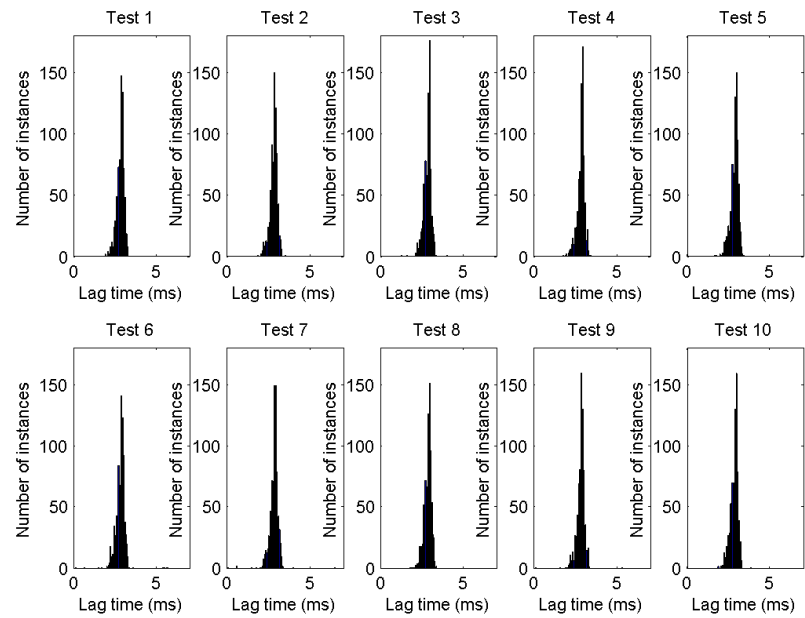
Histograms of All Tests, Computer 1, Windows 7, USB, P2 to P4



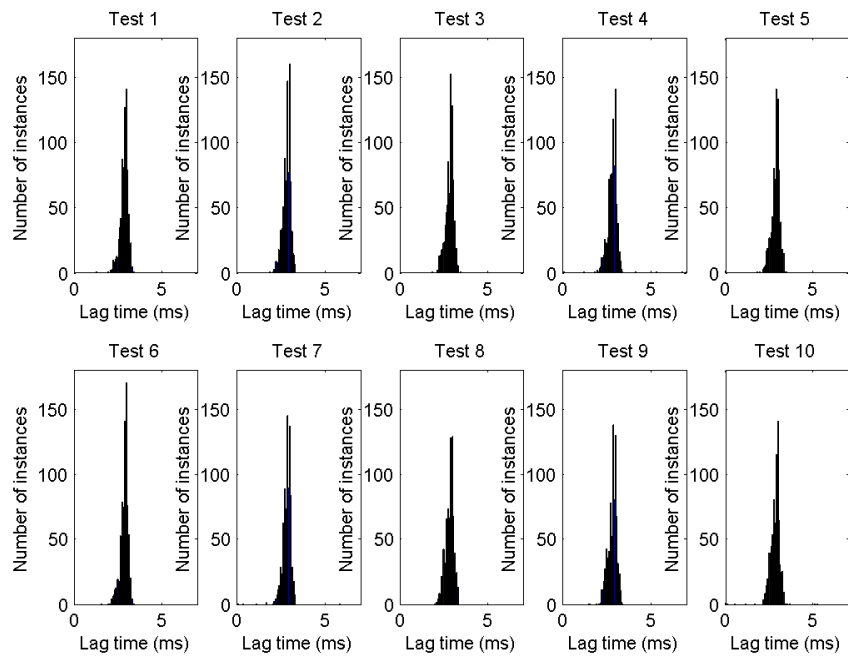
Histograms of All Tests, Computer 1, Windows 7, USB, P2 to P3



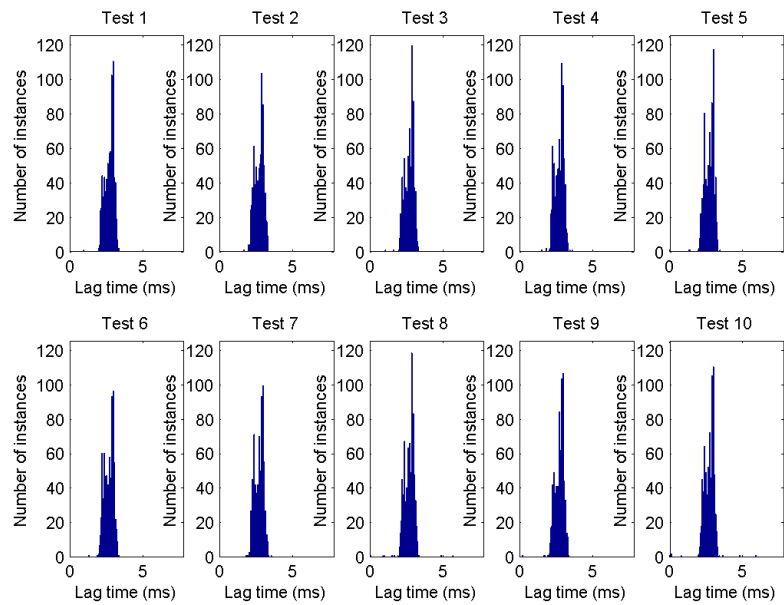
Histograms of All Tests, Computer 1, Windows 7, USB, P2 to P1



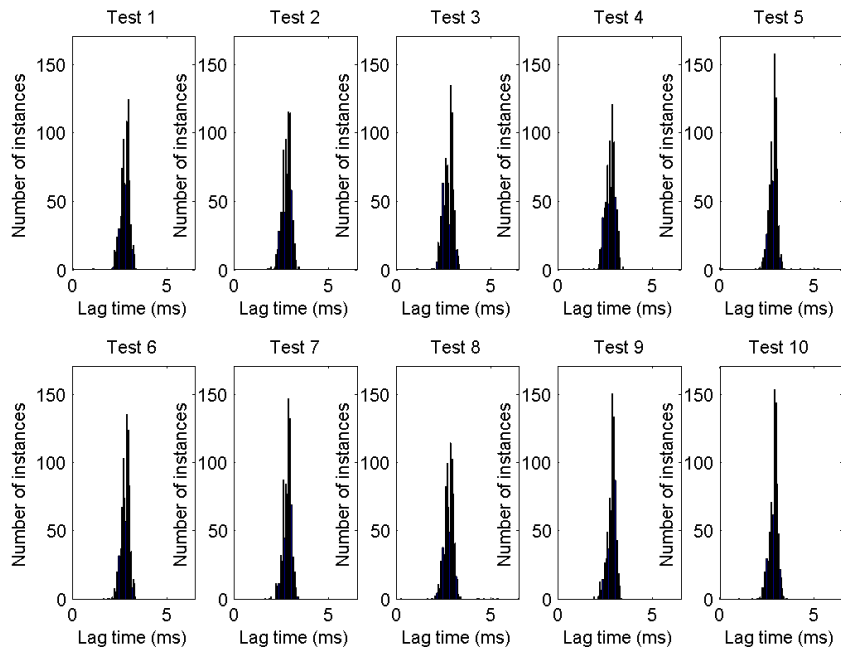
Histograms of All Tests, Computer 1, Windows 7, USB, P1 to P4



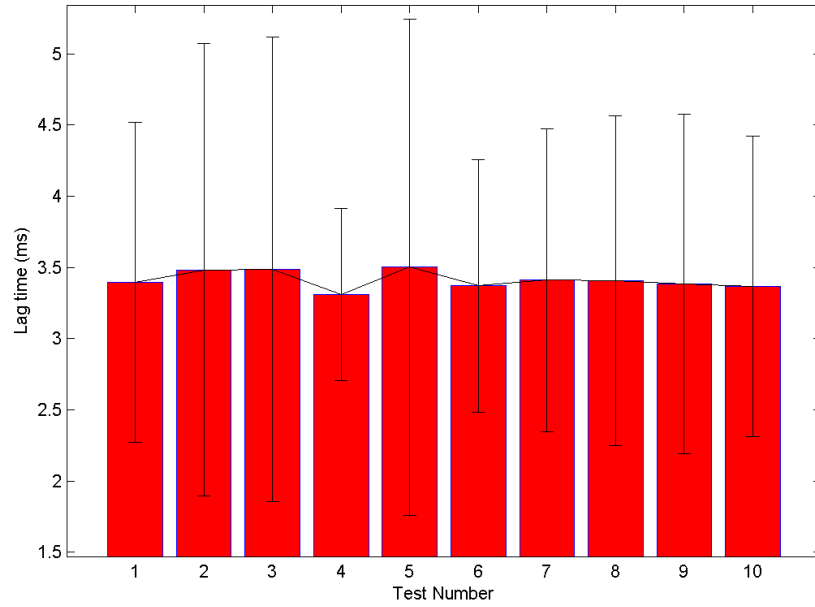
Histograms of All Tests, Computer 1, Windows 7, USB, P1 to P3



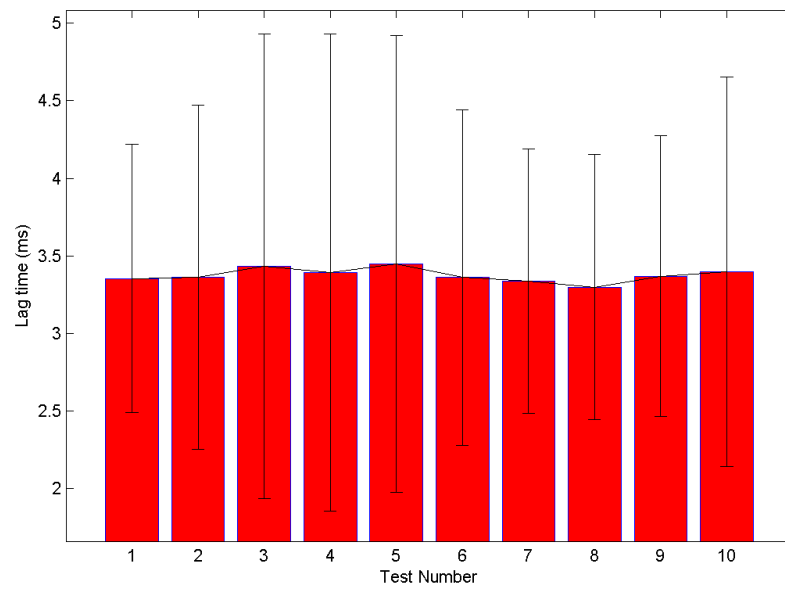
Histograms of All Tests, Computer 1, Windows 7, USB, P1 to P2



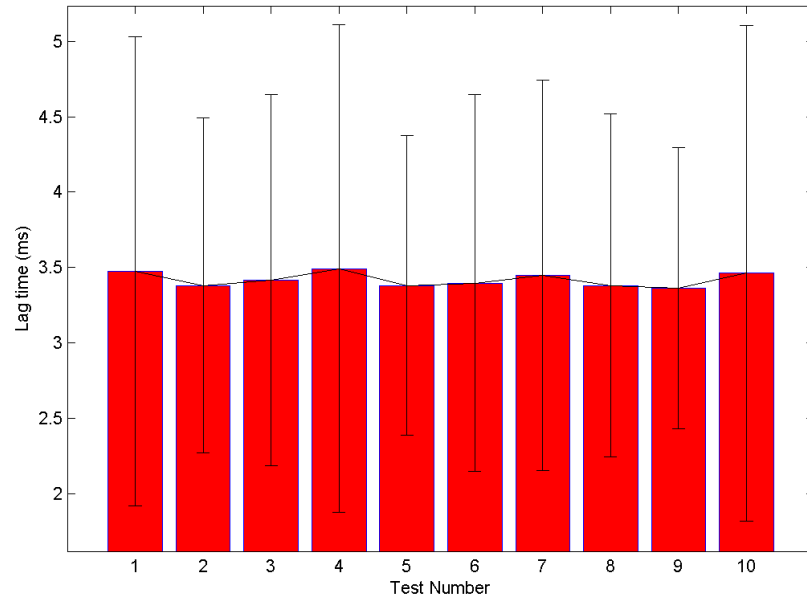
Mean Lag Time Across All Tests, Computer 1, Windows 7, USB, TTL to P4



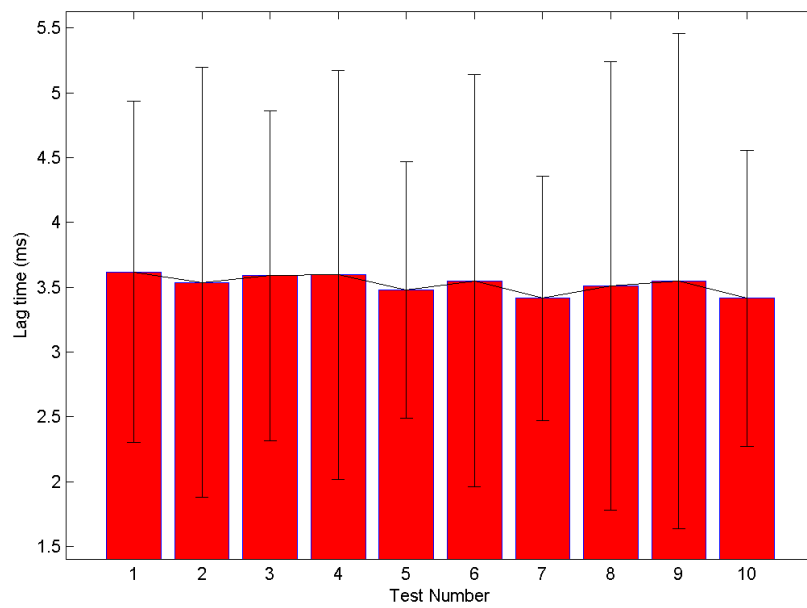
Mean Lag Time Across All Tests, Computer 1, Windows 7, USB, TTL to P3



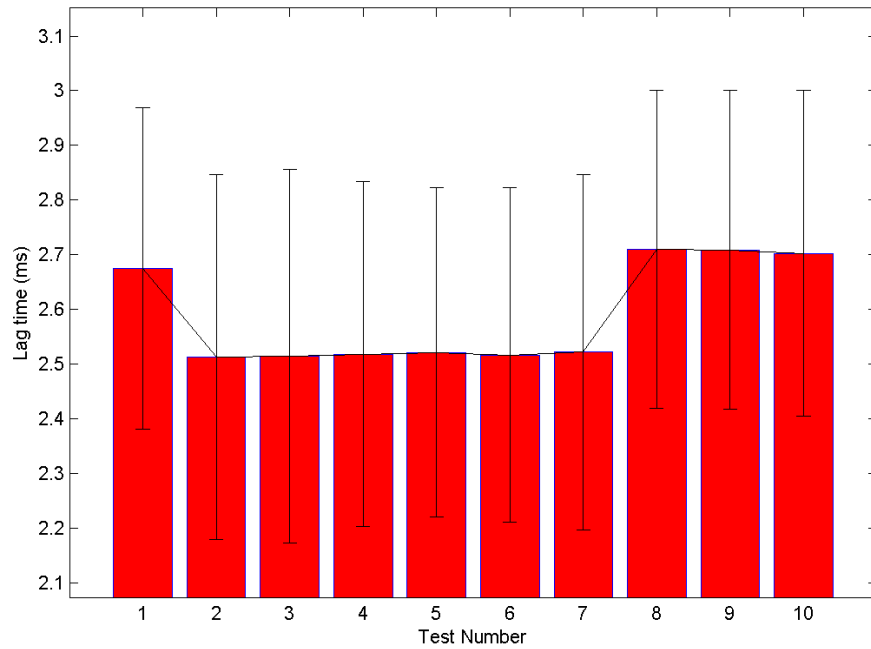
Mean Lag Time Across All Tests, Computer 1, Windows 7, USB, TTL to P2



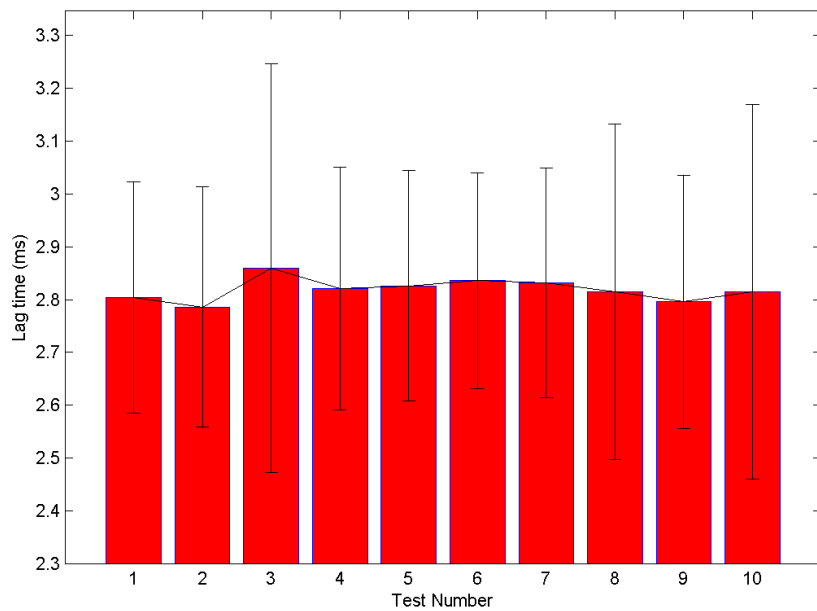
Mean Lag Time Across All Tests, Computer 1, Windows 7, USB, TTL to P1



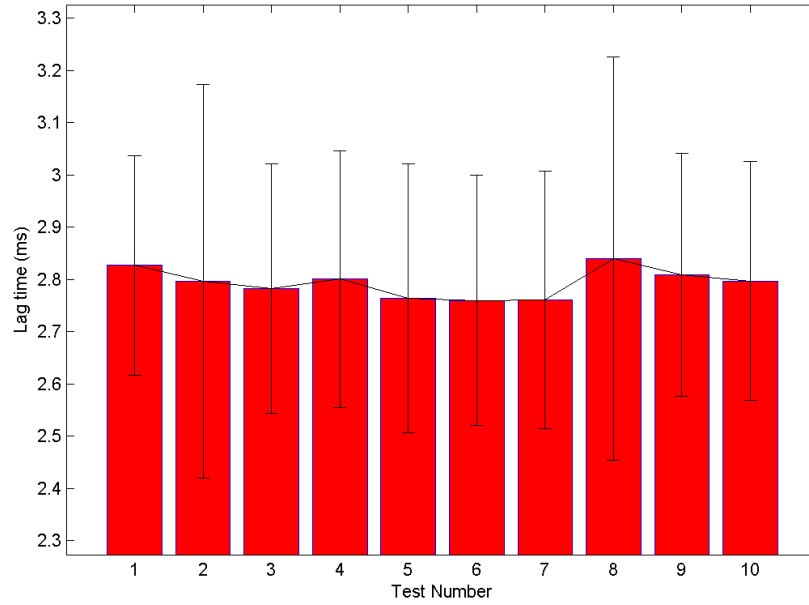
Mean Lag Time Across All Tests, Computer 1, Windows 7, USB, P4 to P3



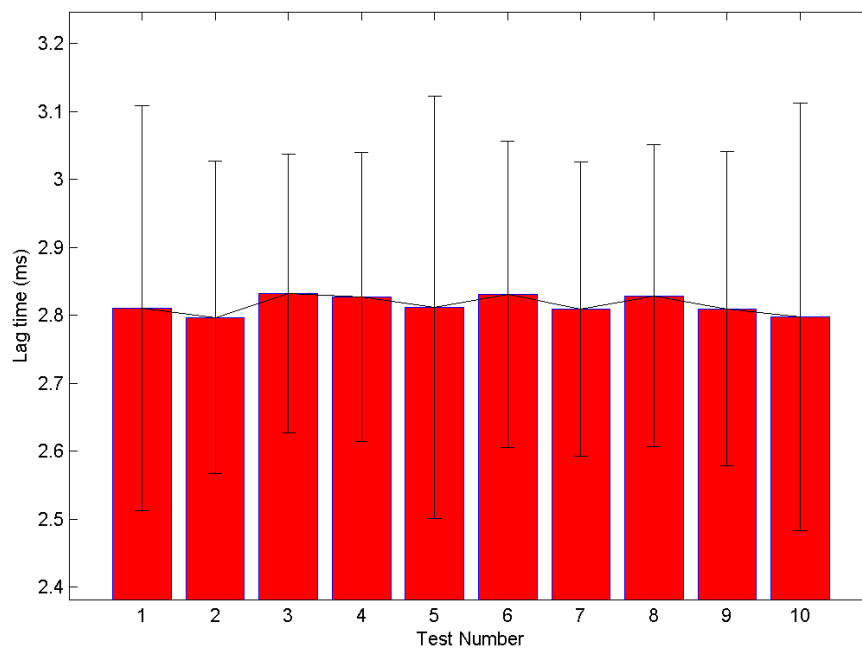
Mean Lag Time Across All Tests, Computer 1, Windows 7, USB, P4 to P2



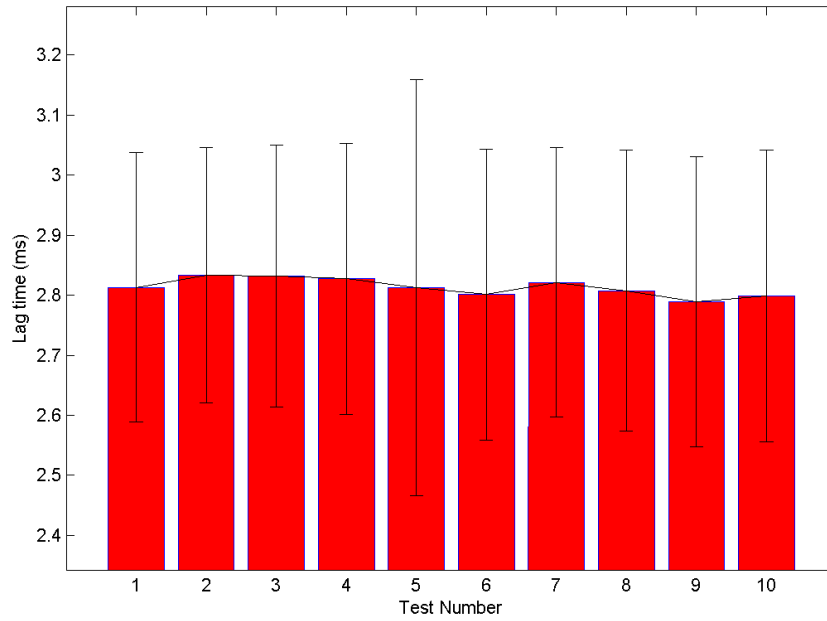
Mean Lag Time Across All Tests, Computer 1, Windows 7, USB, P4 to P1



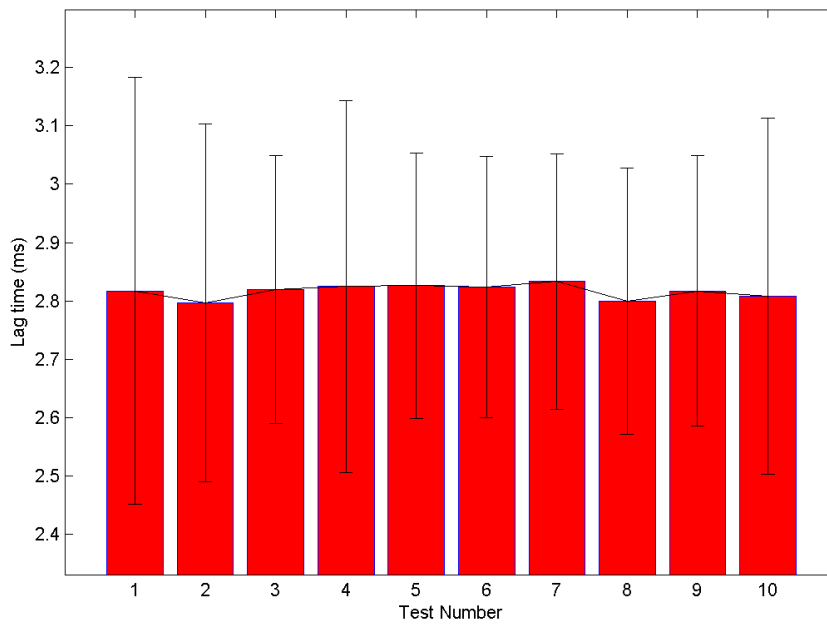
Mean Lag Time Across All Tests, Computer 1, Windows 7, USB, P3 to P4



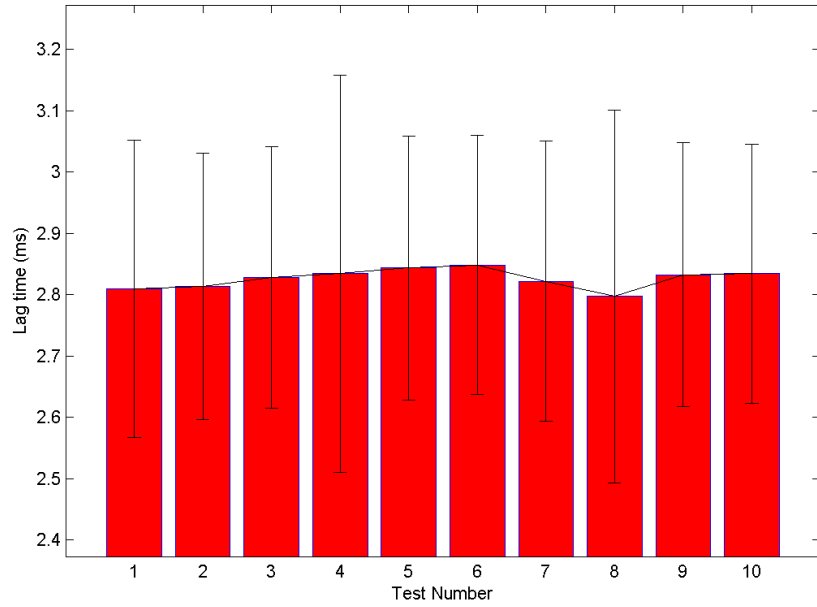
Mean Lag Time Across All Tests, Computer 1, Windows 7, USB, P3 to P2



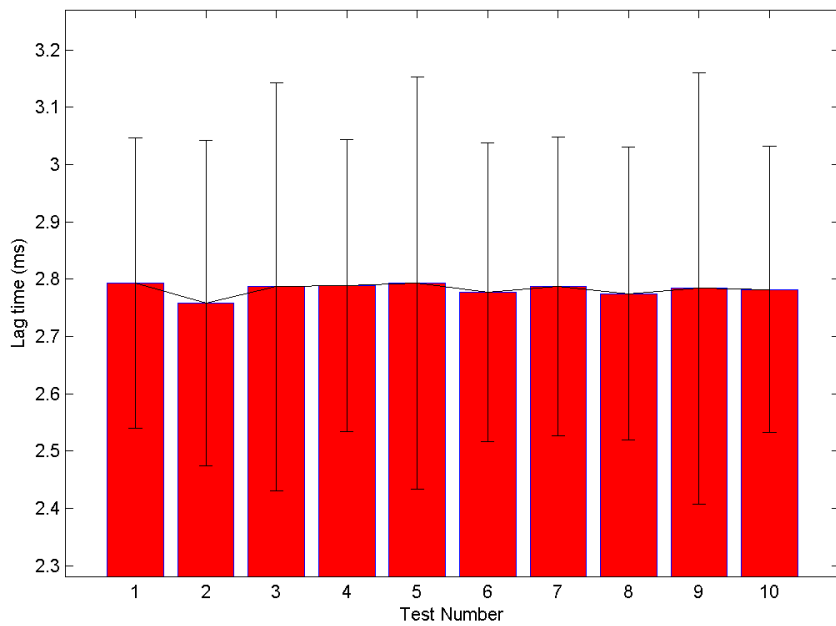
Mean Lag Time Across All Tests, Computer 1, Windows 7, USB, P3 to P1



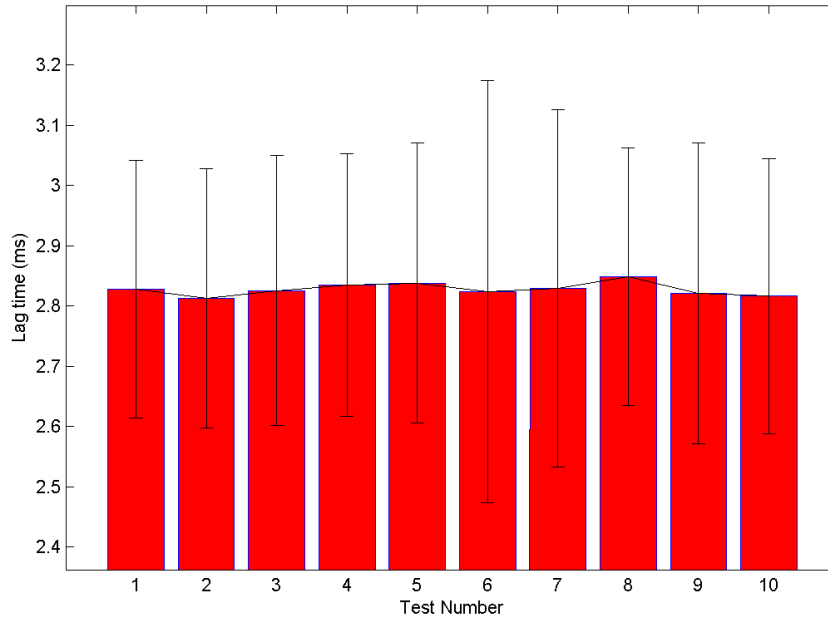
Mean Lag Time Across All Tests, Computer 1, Windows 7, USB, P2 to P4



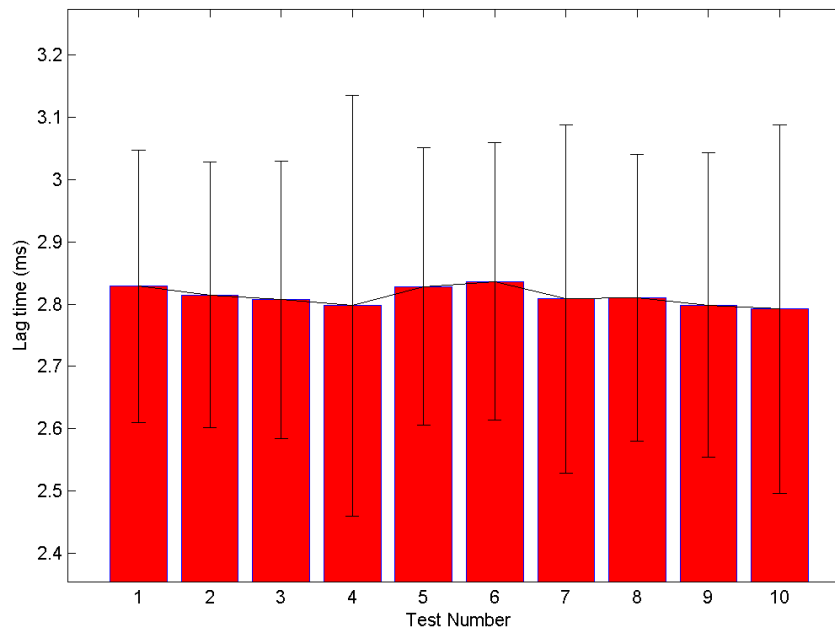
Mean Lag Time Across All Tests, Computer 1, Windows 7, USB, P2 to P3



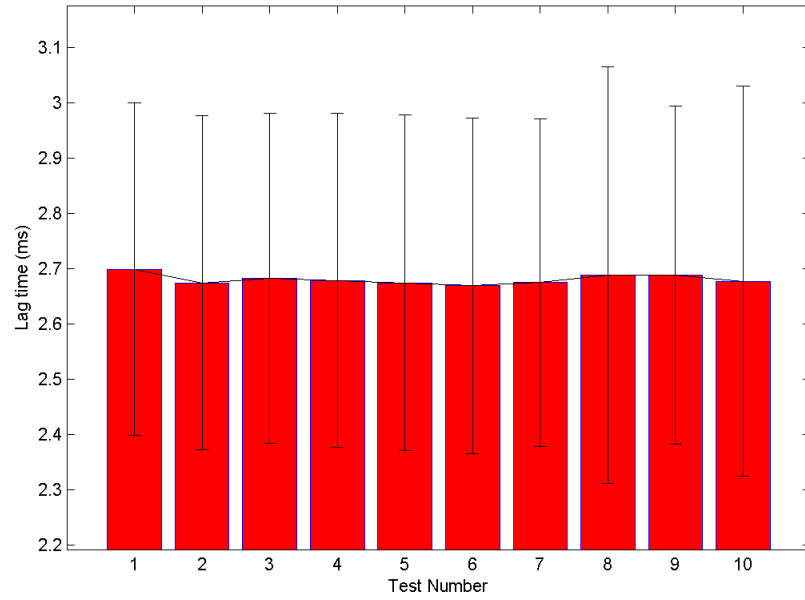
Mean Lag Time Across All Tests, Computer 1, Windows 7, USB, P2 to P1



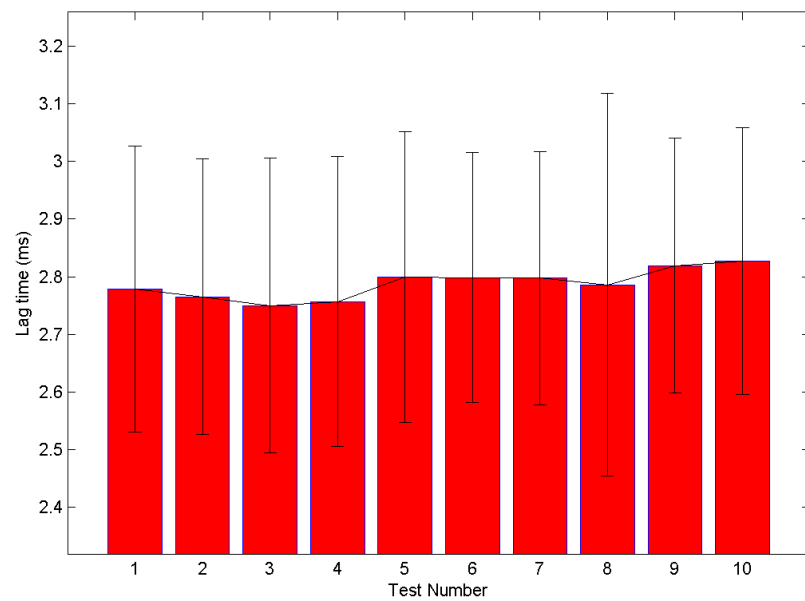
Mean Lag Time Across All Tests, Computer 1, Windows 7, USB, P1 to P4



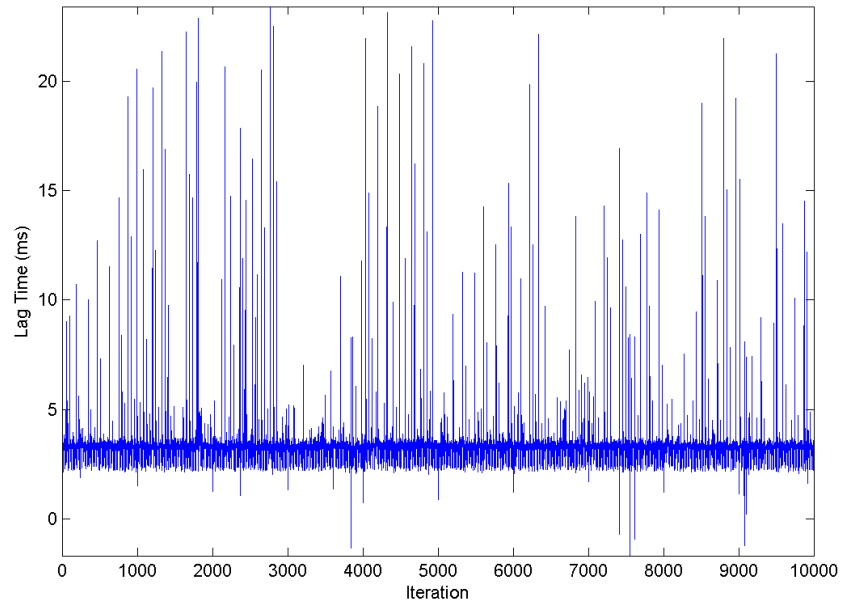
Mean Lag Time Across All Tests, Computer 1, Windows 7, USB, P1 to P3



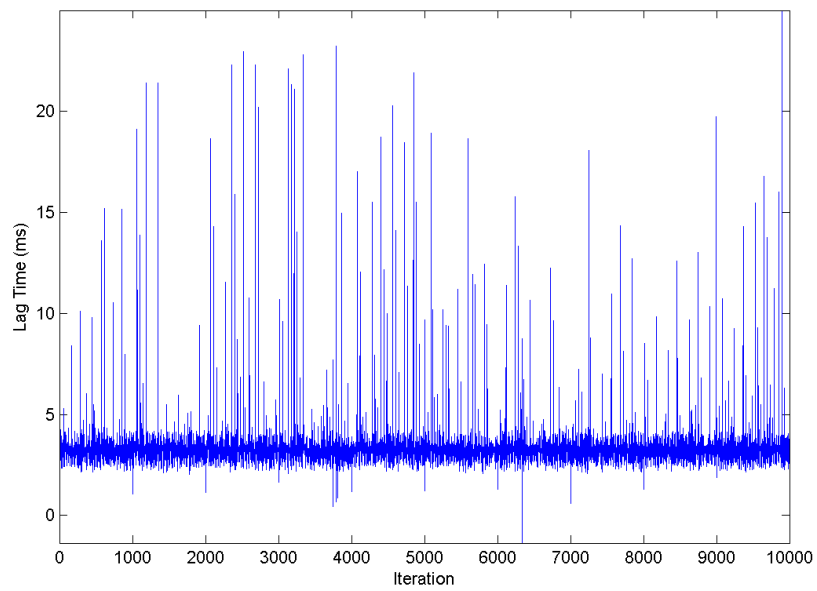
Mean Lag Time Across All Tests, Computer 1, Windows 7, USB, P1 to P2



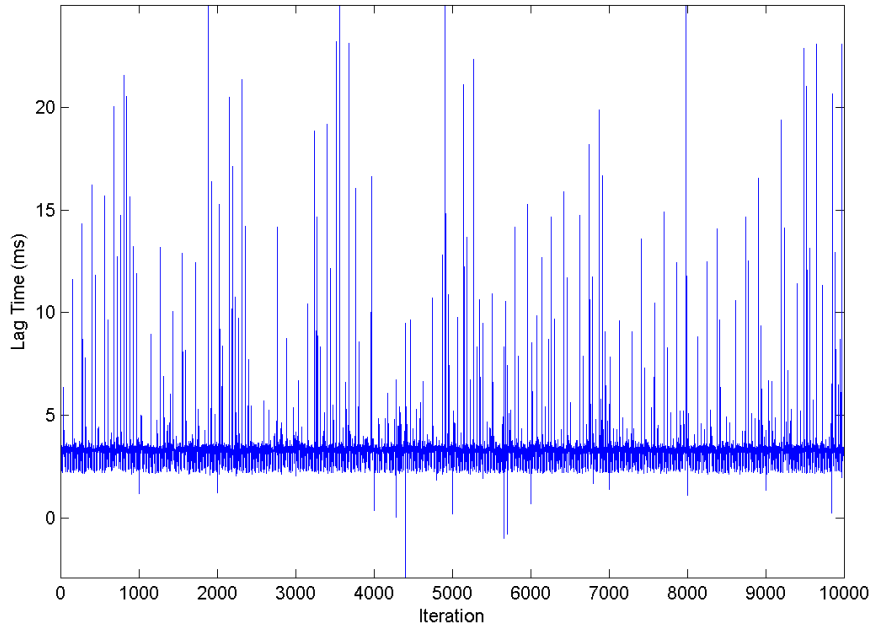
Plots of All Tests, Computer 1, Windows 7, USB, TTL to P4



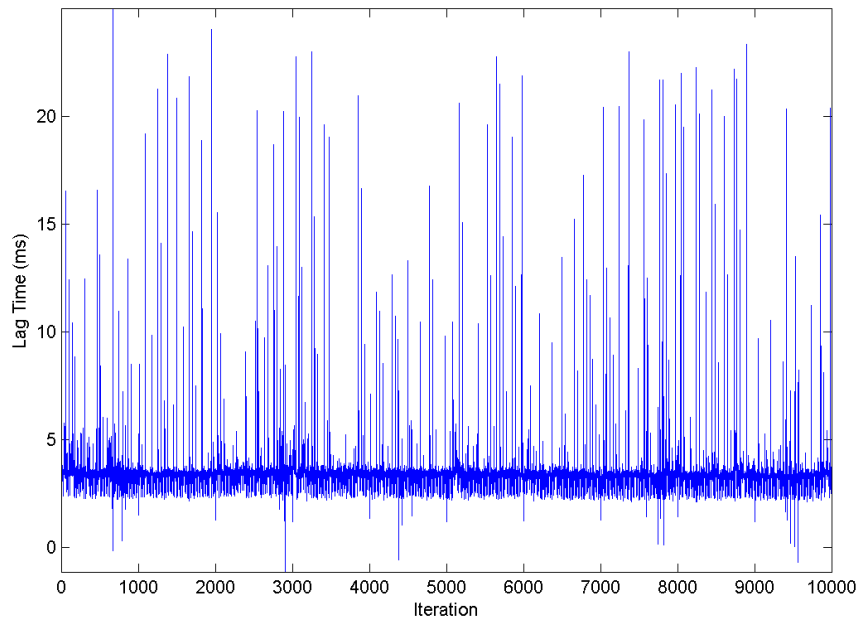
Plots of All Tests, Computer 1, Windows 7, USB, TTL to P3



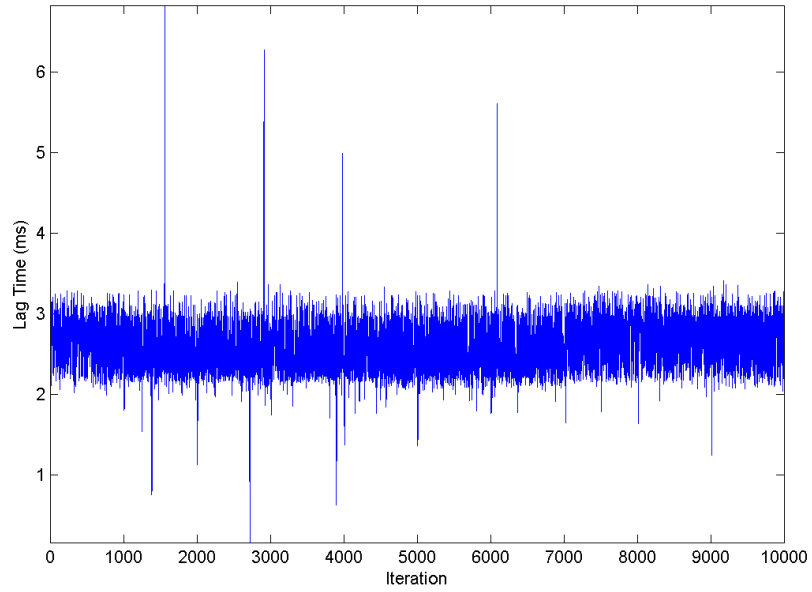
Plots of All Tests, Computer 1, Windows 7, USB, TTL to P2



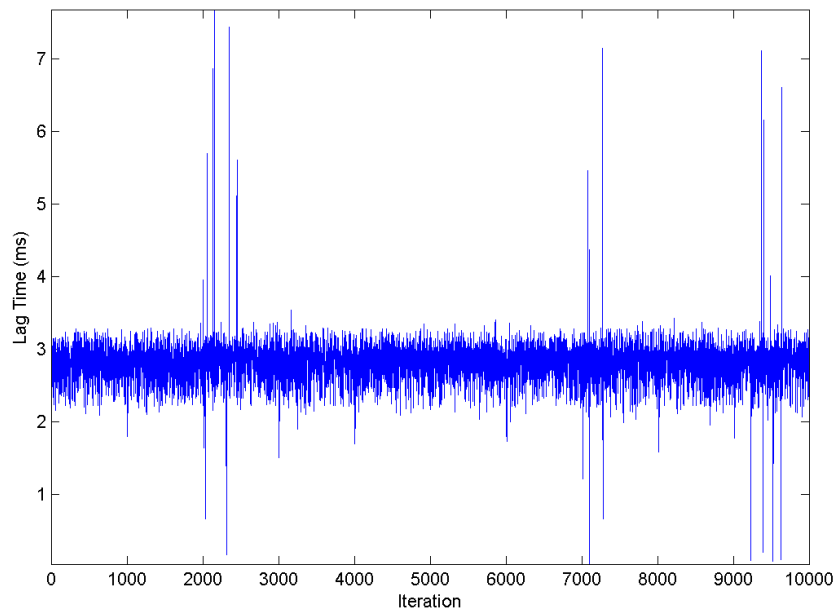
Plots of All Tests, Computer 1, Windows 7, USB, TTL to P1



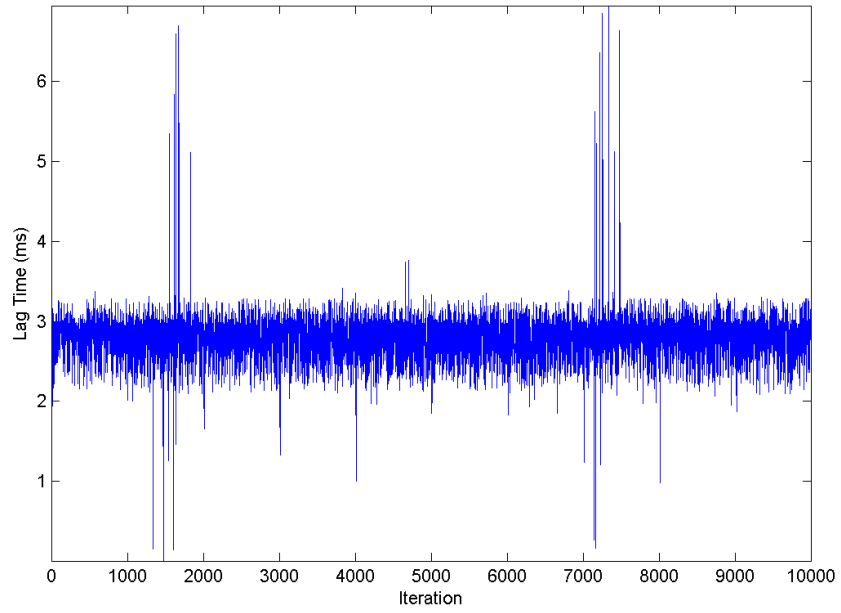
Plots of All Tests, Computer 1, Windows 7, USB, P4 to P3



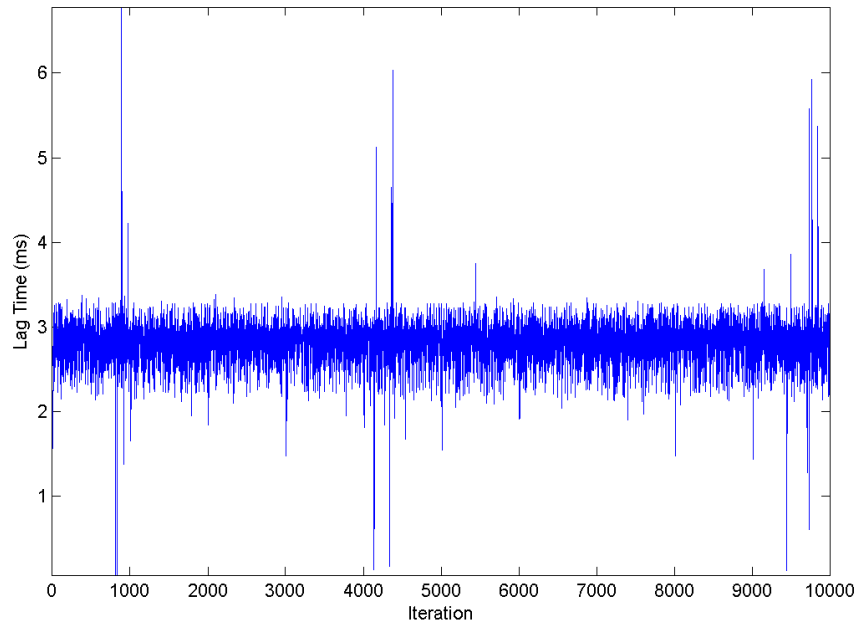
Plots of All Tests, Computer 1, Windows 7, USB, P4 to P2



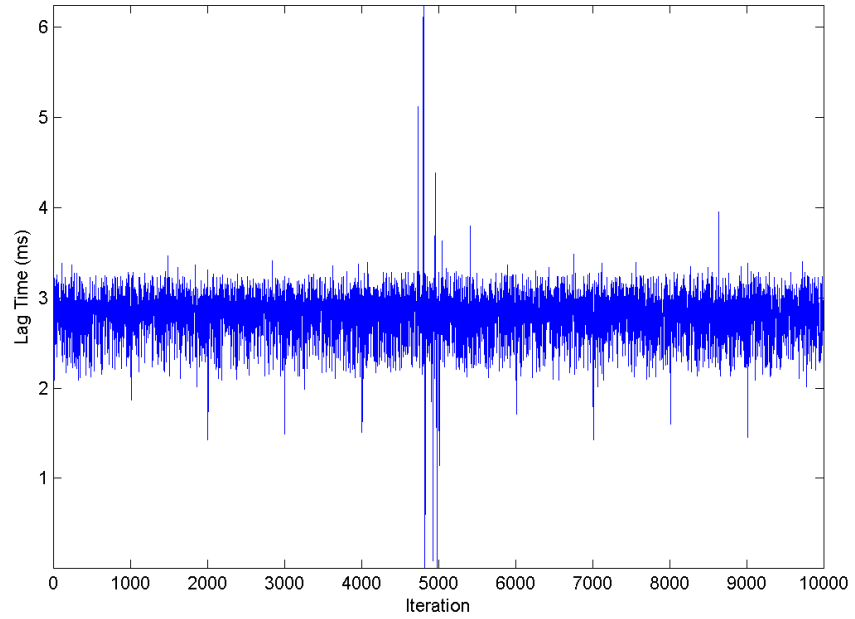
Plots of All Tests, Computer 1, Windows 7, USB, P4 to P1



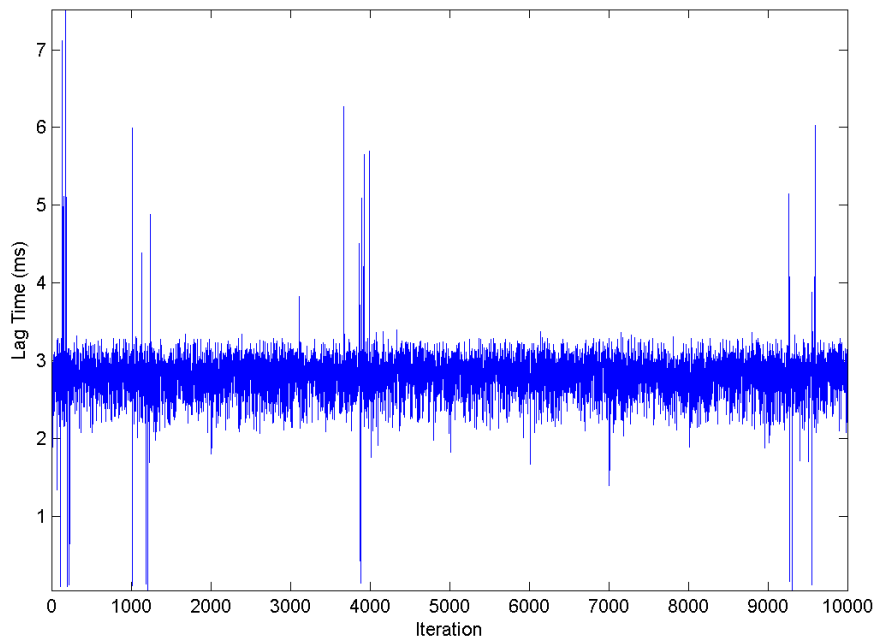
Plots of All Tests, Computer 1, Windows 7, USB, P3 to P4



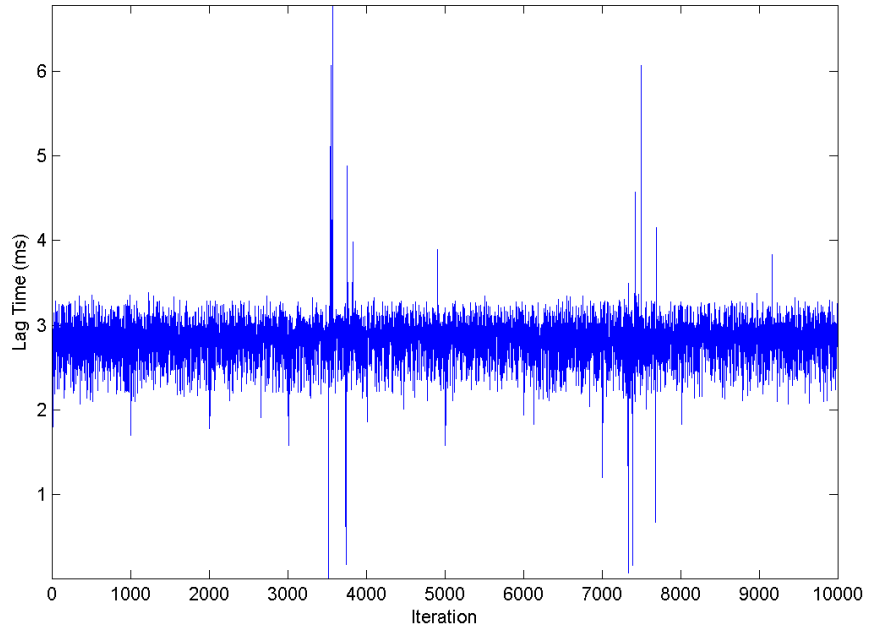
Plots of All Tests, Computer 1, Windows 7, USB, P3 to P2



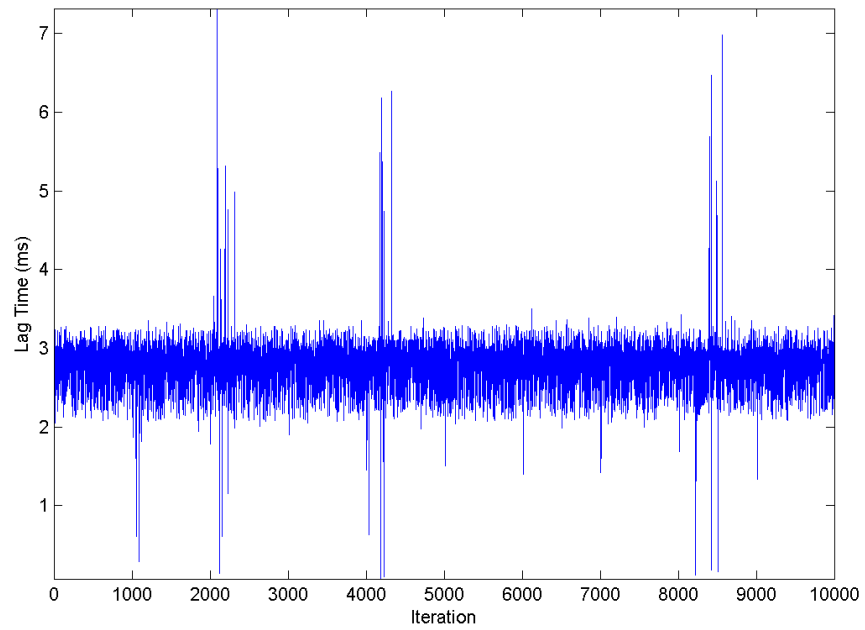
Plots of All Tests, Computer 1, Windows 7, USB, P3 to P1



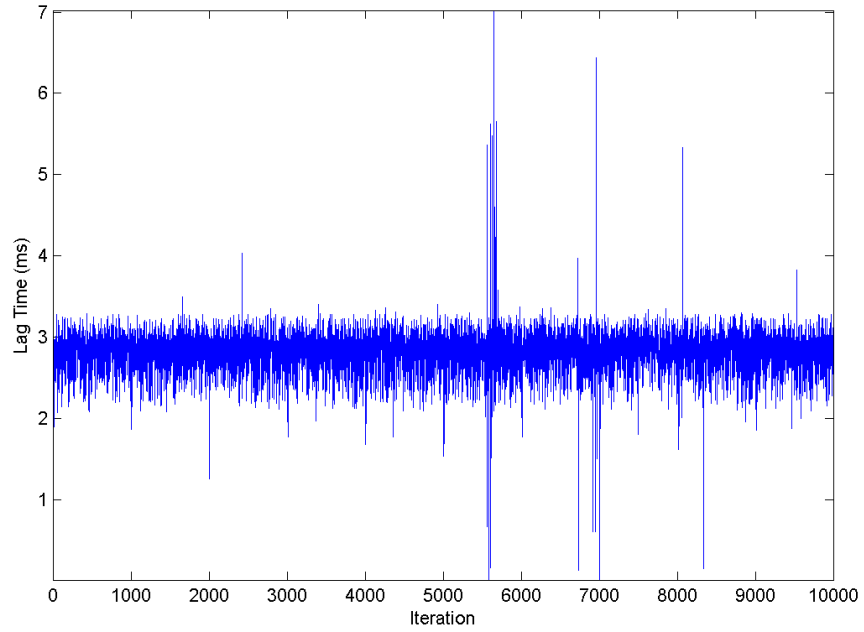
Plots of All Tests, Computer 1, Windows 7, USB, P2 to P4



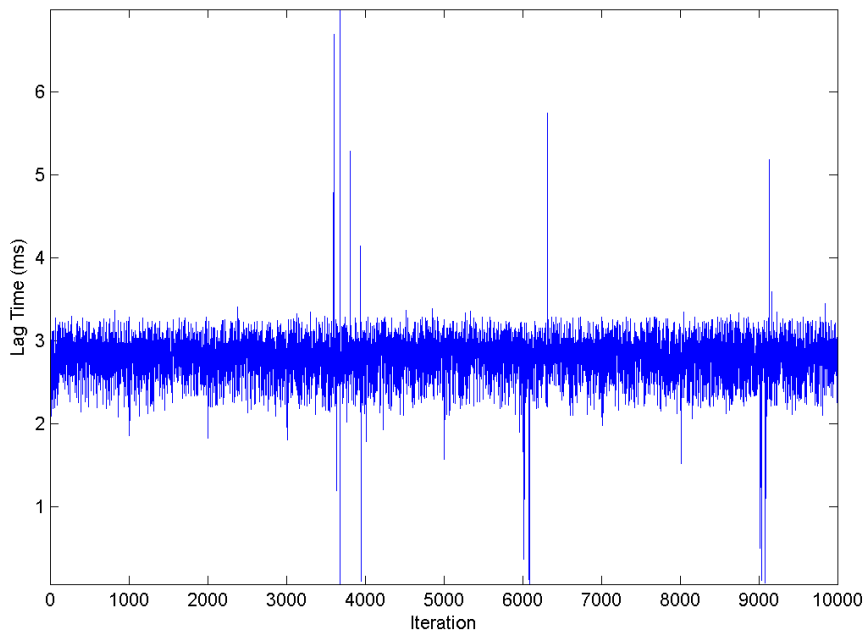
Plots of All Tests, Computer 1, Windows 7, USB, P2 to P3



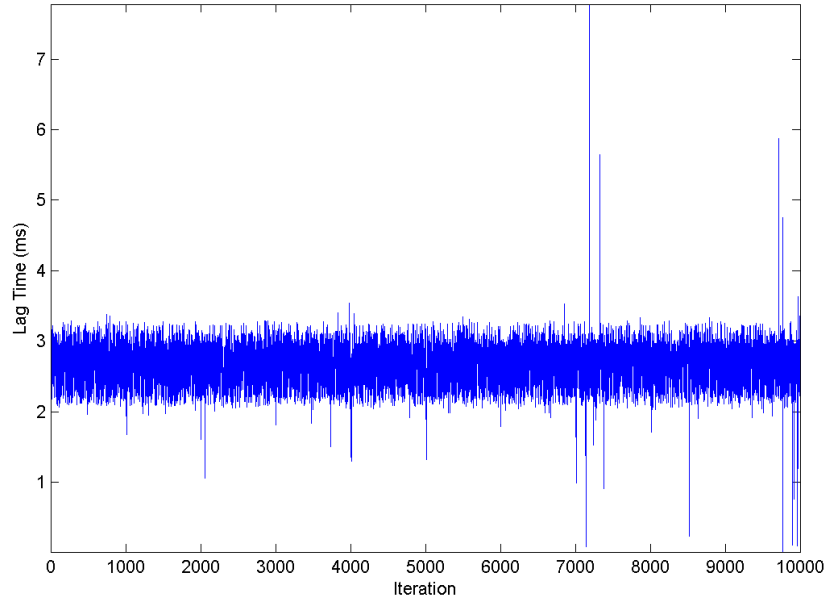
Plots of All Tests, Computer 1, Windows 7, USB, P2 to P1



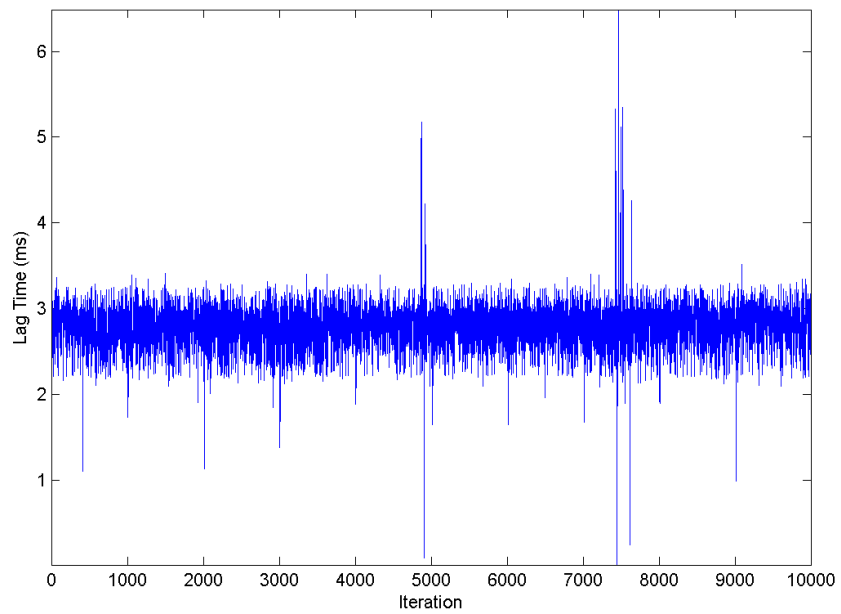
Plots of All Tests, Computer 1, Windows 7, USB, P1 to P4



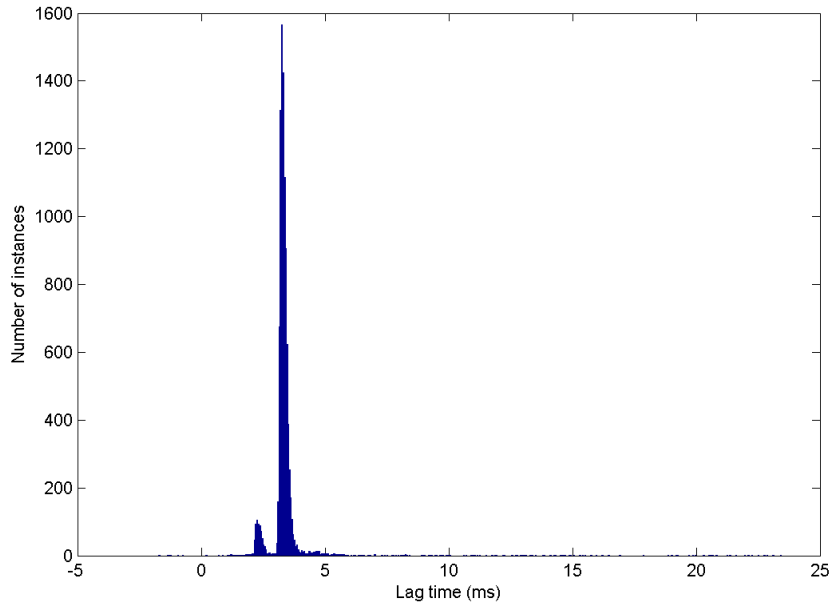
Plots of All Tests, Computer 1, Windows 7, USB, P1 to P3



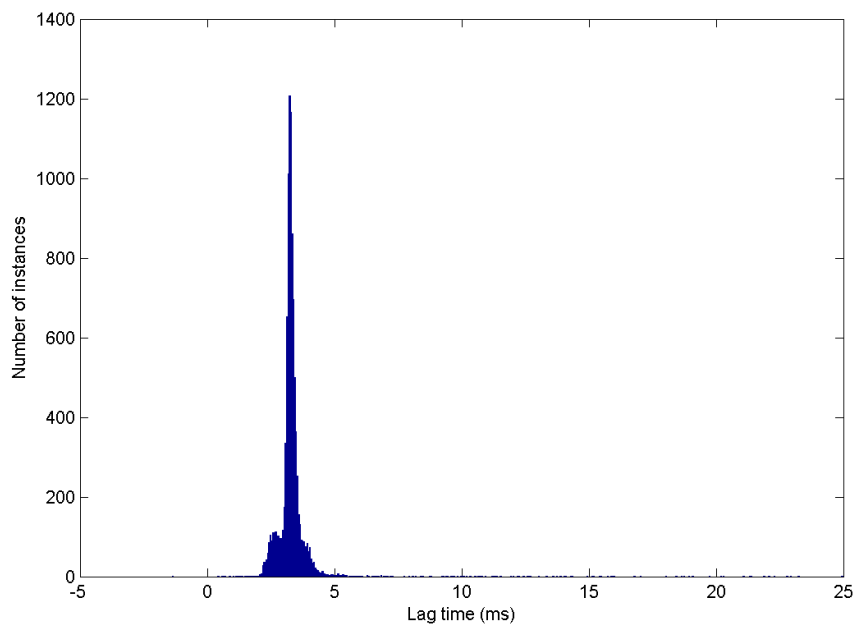
Plots of All Tests, Computer 1, Windows 7, USB, P1 to P2



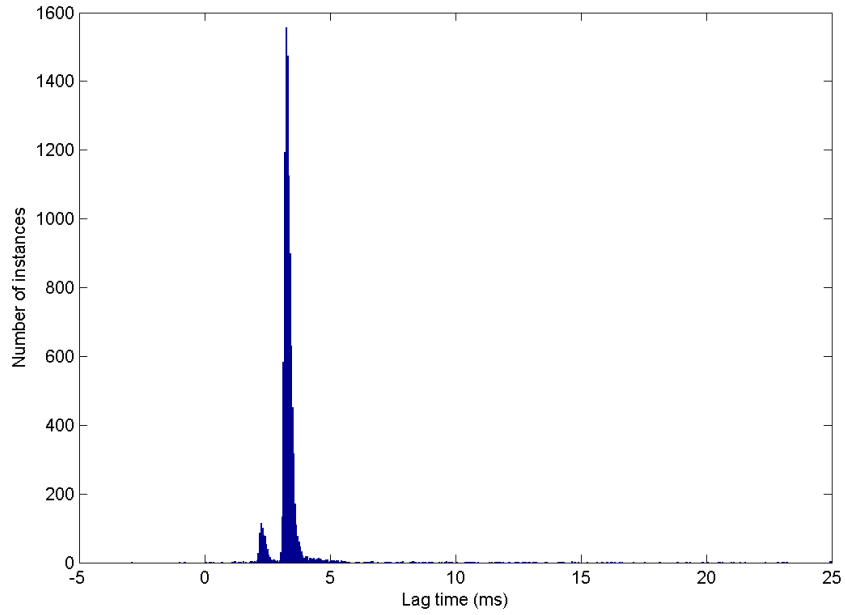
Histogram of All Tests, Computer 1, Windows 7, USB, TTL to P4



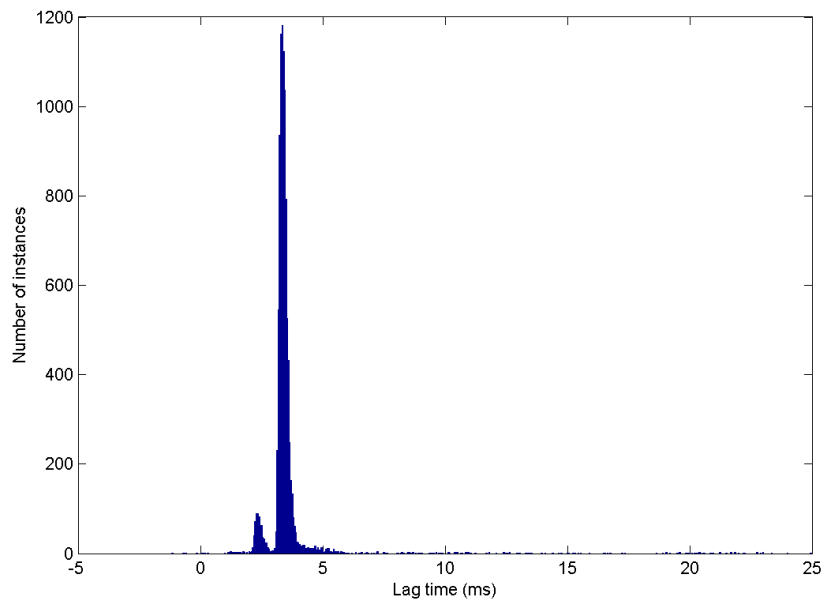
Histogram of All Tests, Computer 1, Windows 7, USB, TTL to P3



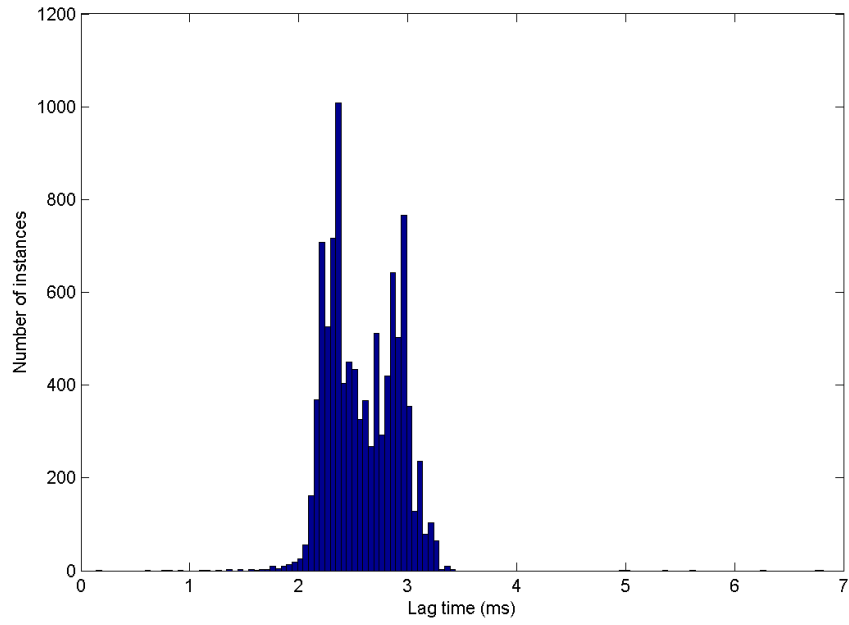
Histogram of All Tests, Computer 1, Windows 7, USB, TTL to P2



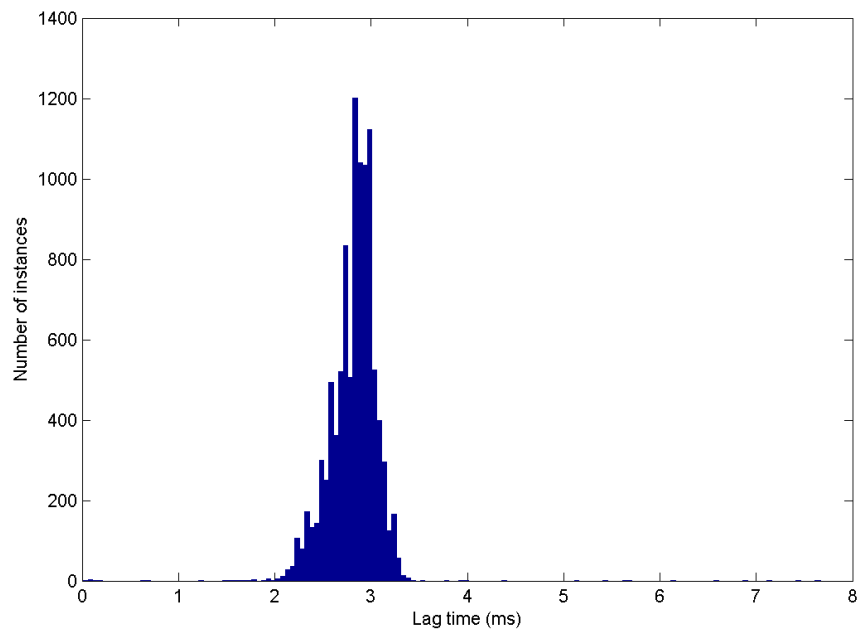
Histogram of All Tests, Computer 1, Windows 7, USB, TTL to P1



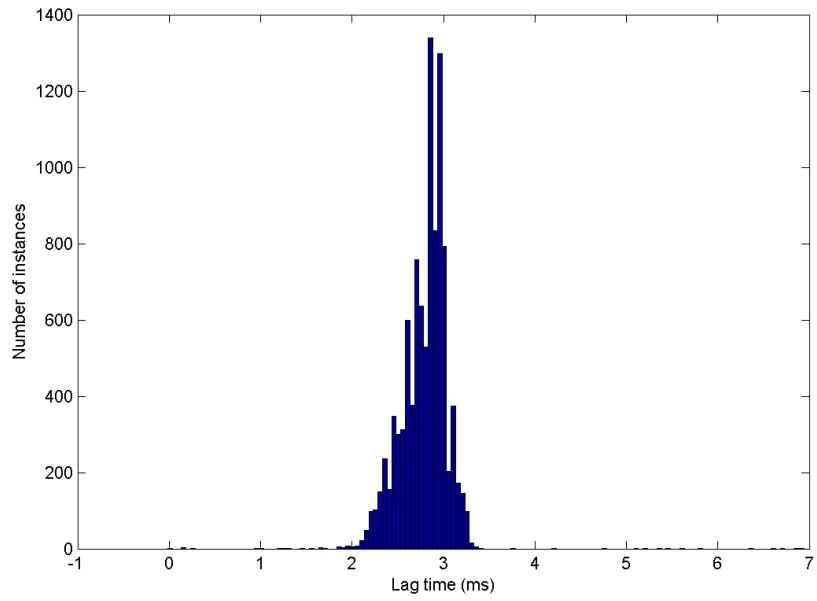
Histogram of All Tests, Computer 1, Windows 7, USB, P4 to P3



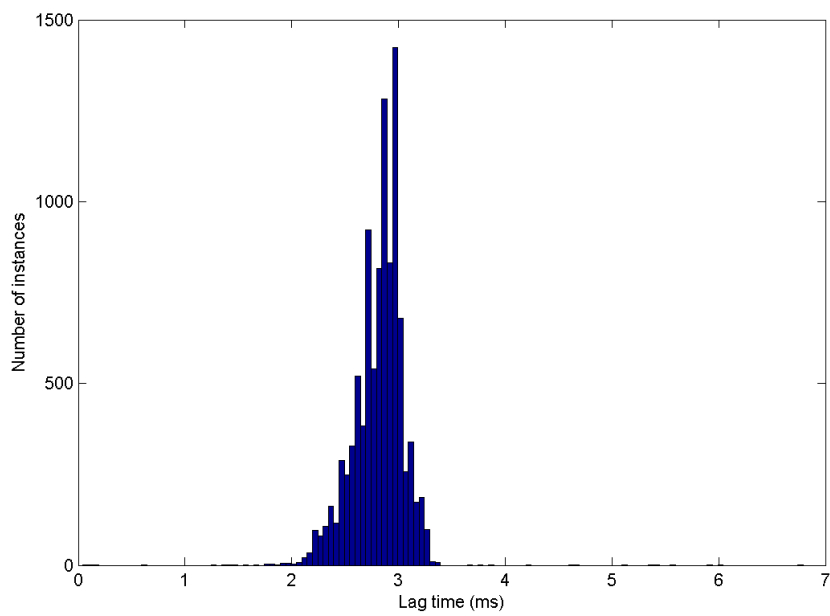
Histogram of All Tests, Computer 1, Windows 7, USB, P4 to P2



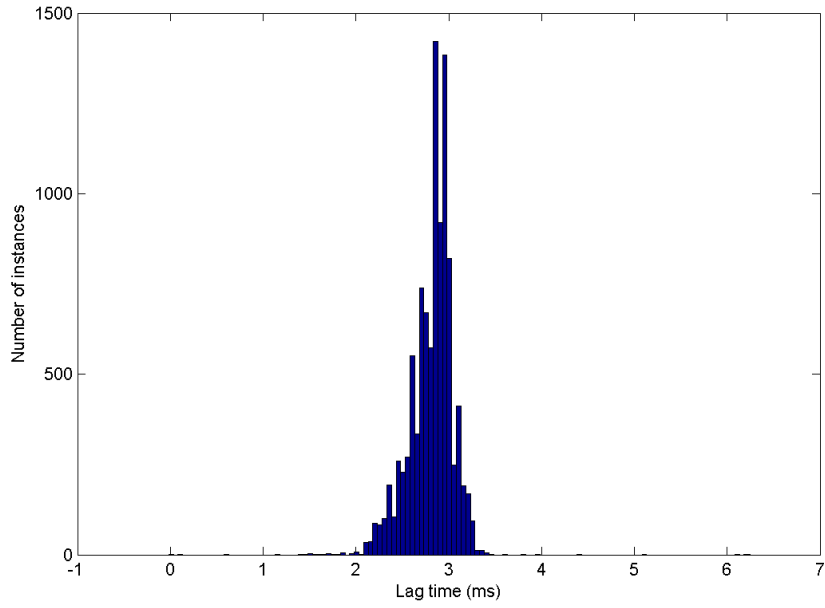
Histogram of All Tests, Computer 1, Windows 7, USB, P4 to P1



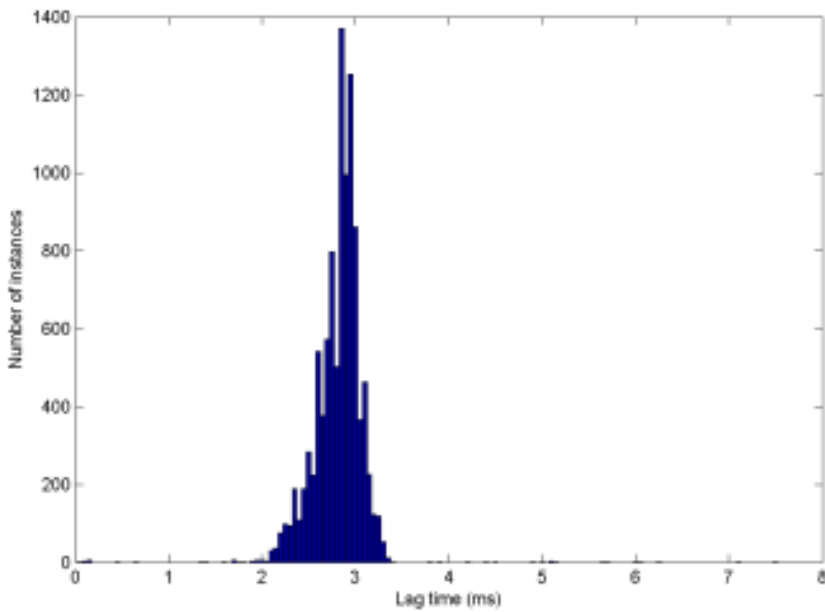
Histogram of All Tests, Computer 1, Windows 7, USB, P3 to P4



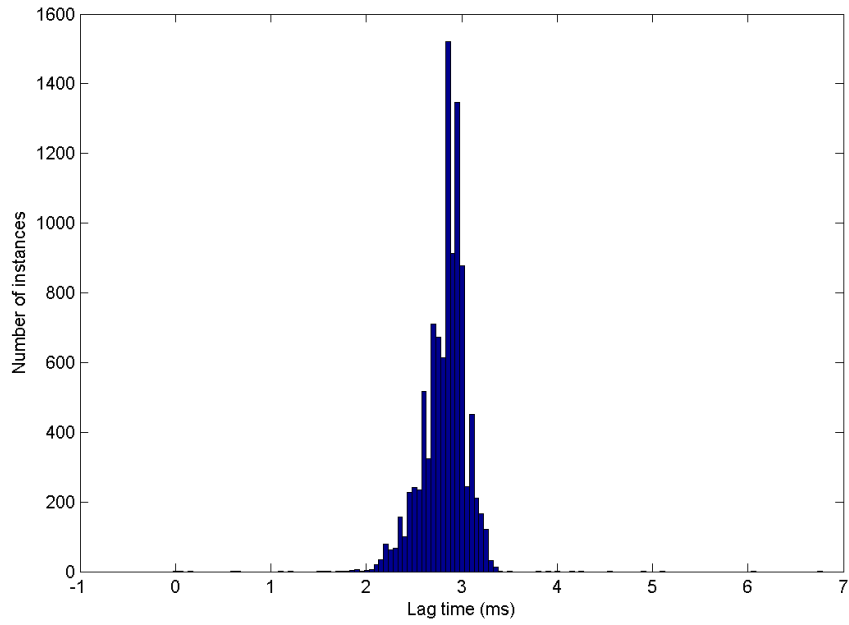
Histogram of All Tests, Computer 1, Windows 7, USB, P3 to P2



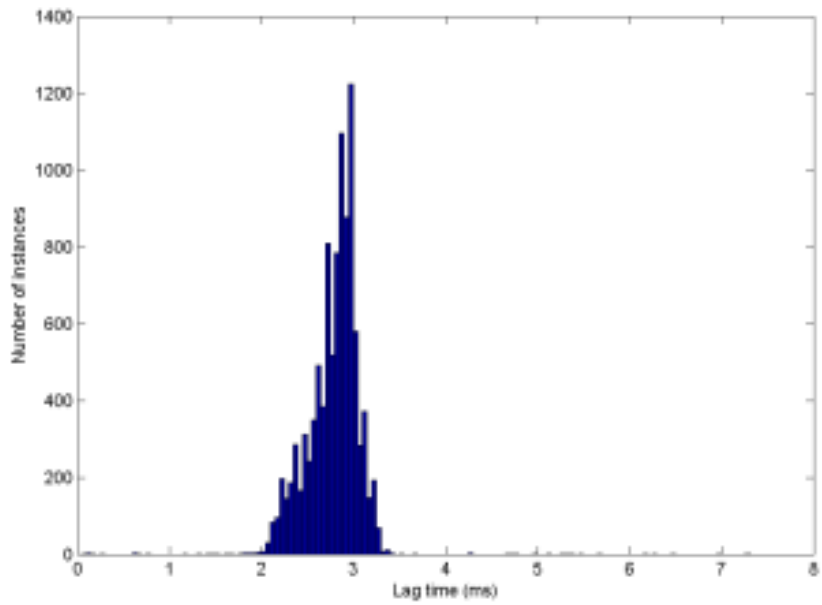
Histogram of All Tests, Computer 1, Windows 7, USB, P3 to P1



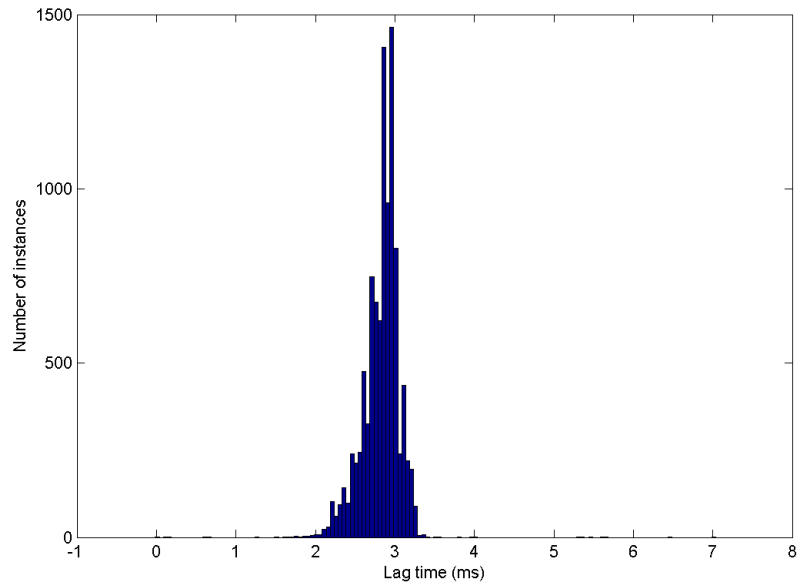
Histogram of All Tests, Computer 1, Windows 7, USB, P2 to P4



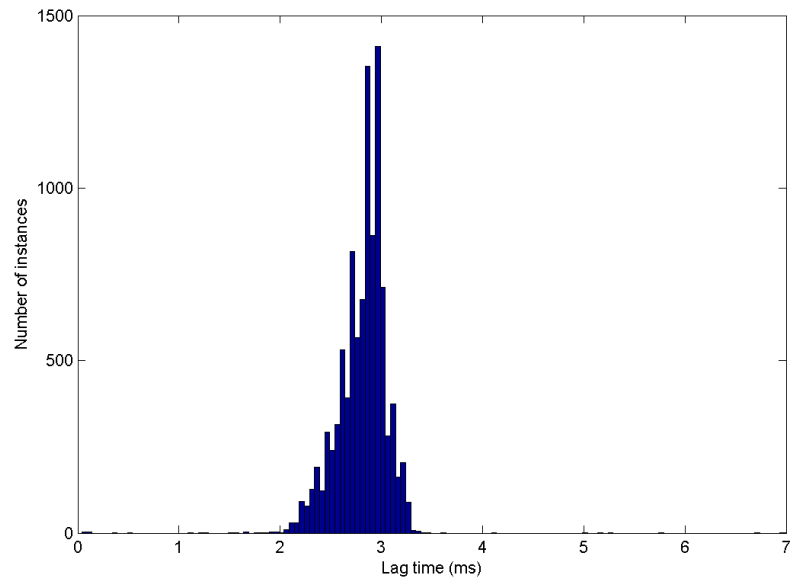
Histogram of All Tests, Computer 1, Windows 7, USB, P2 to P3



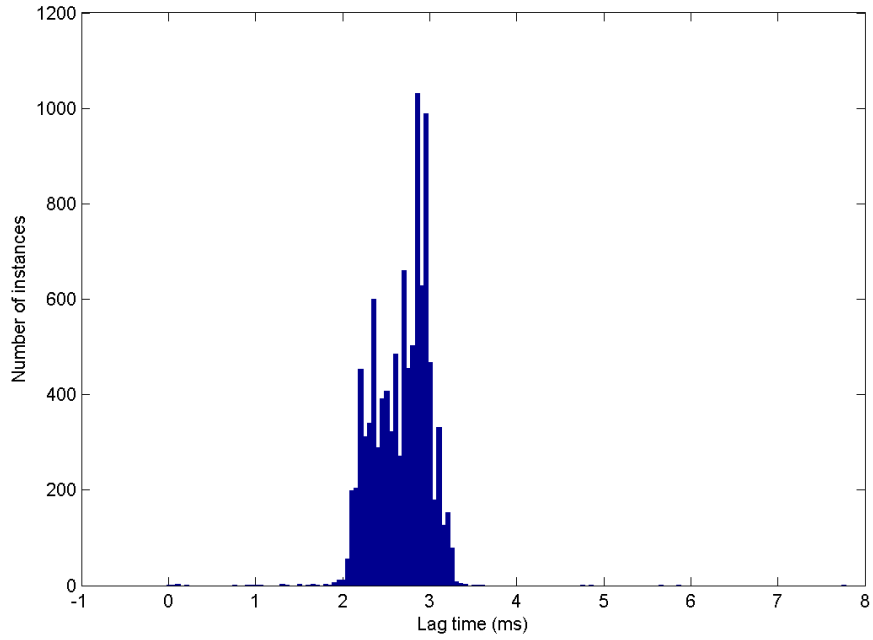
Histogram of All Tests, Computer 1, Windows 7, USB, P2 to P1



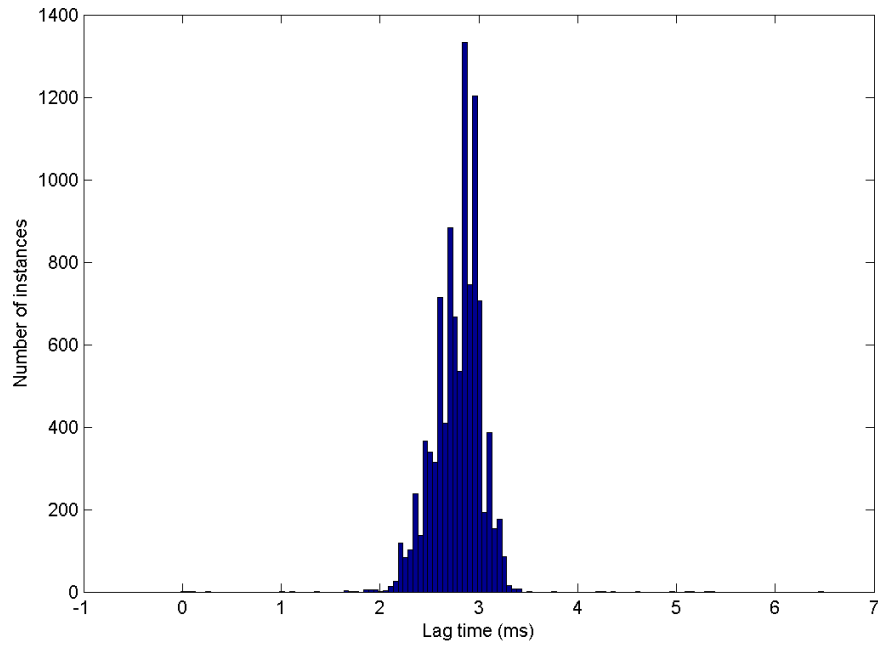
Histogram of All Tests, Computer 1, Windows 7, USB, P1 to P4



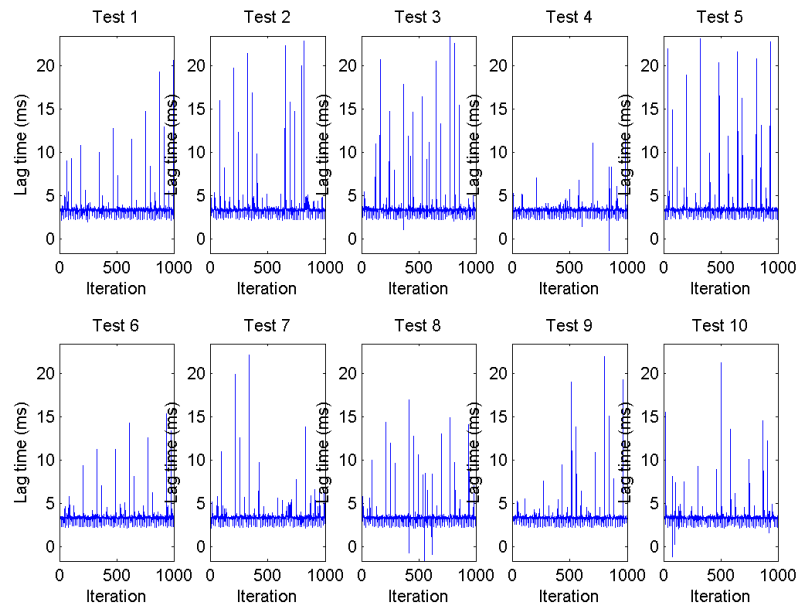
Histogram of All Tests, Computer 1, Windows 7, USB, P1 to P3



Histogram of All Tests, Computer 1, Windows 7, USB, P1 to P2

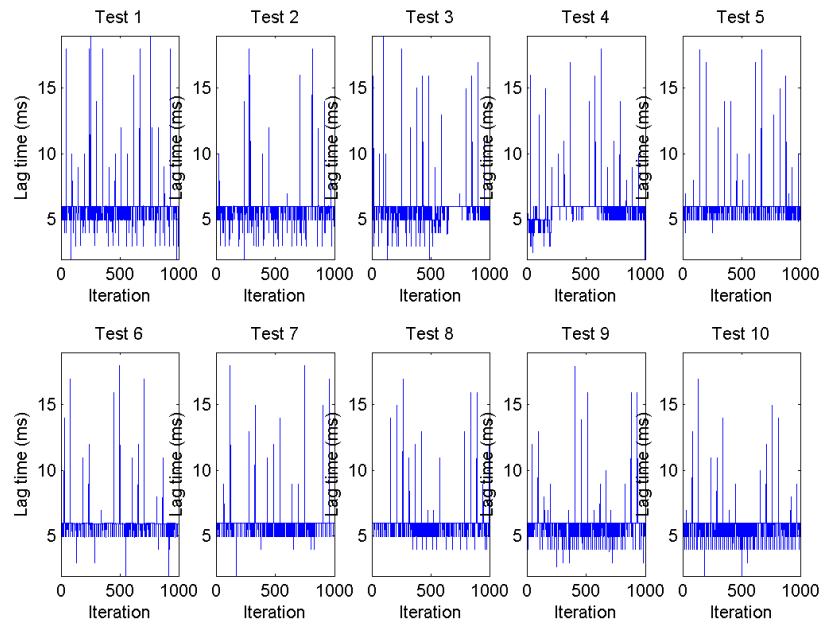


Plots of All Tests, Computer 1, Windows 7, USB, TTL to P4

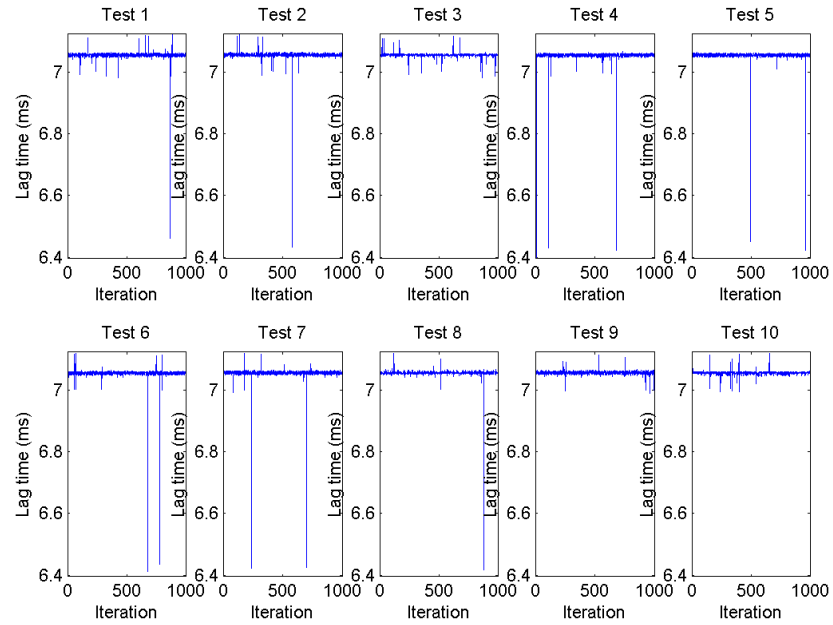


USB vs. Native Serial

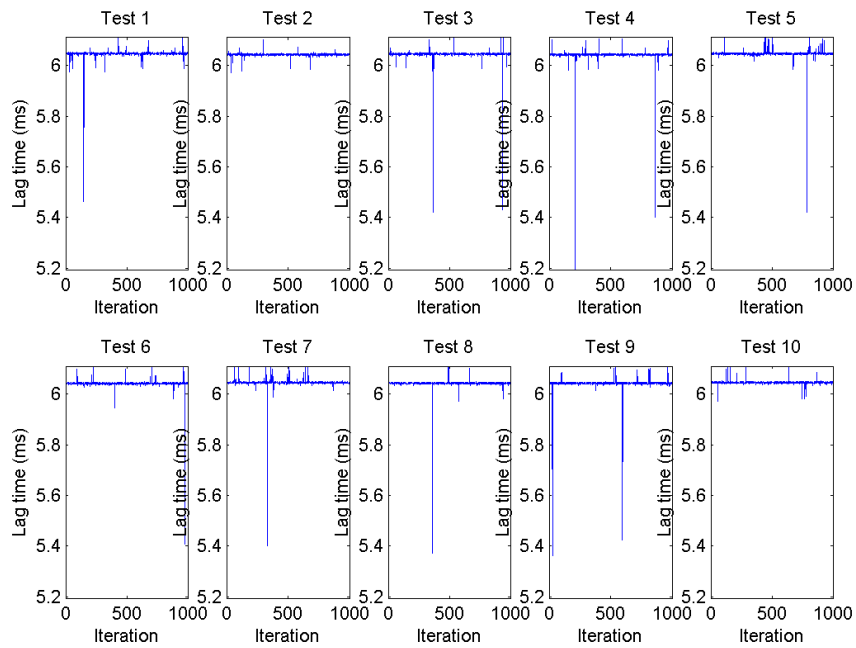
Plots of All Tests, Computer 2, Windows 7, USB, Mouse Still



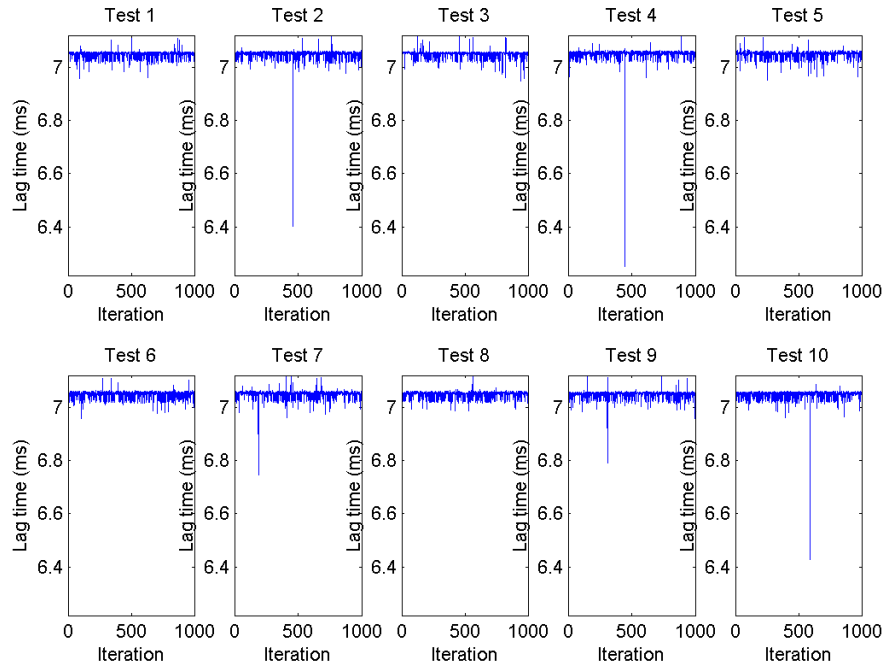
Plots of All Tests, Computer 2, Windows 7, Serial, Not Moving Mouse



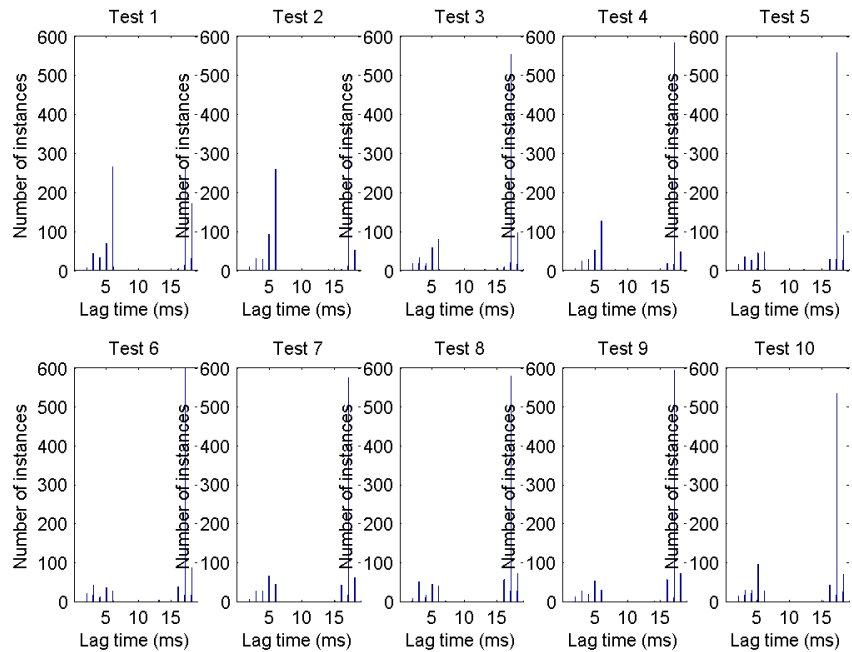
Plots of All Tests, Computer 2, Windows 7, Serial, No Device



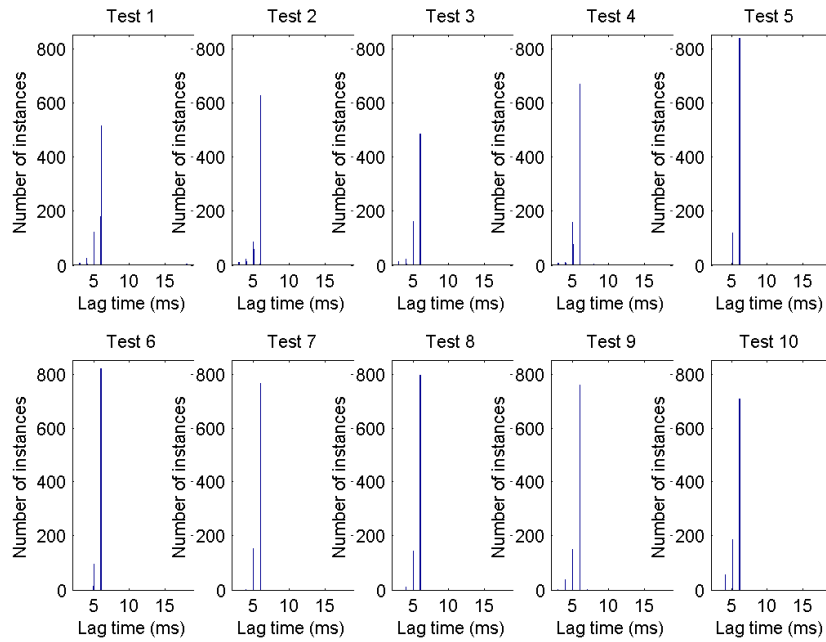
Plots of All Tests, Computer 2, Windows 7, Serial, Moving Mouse



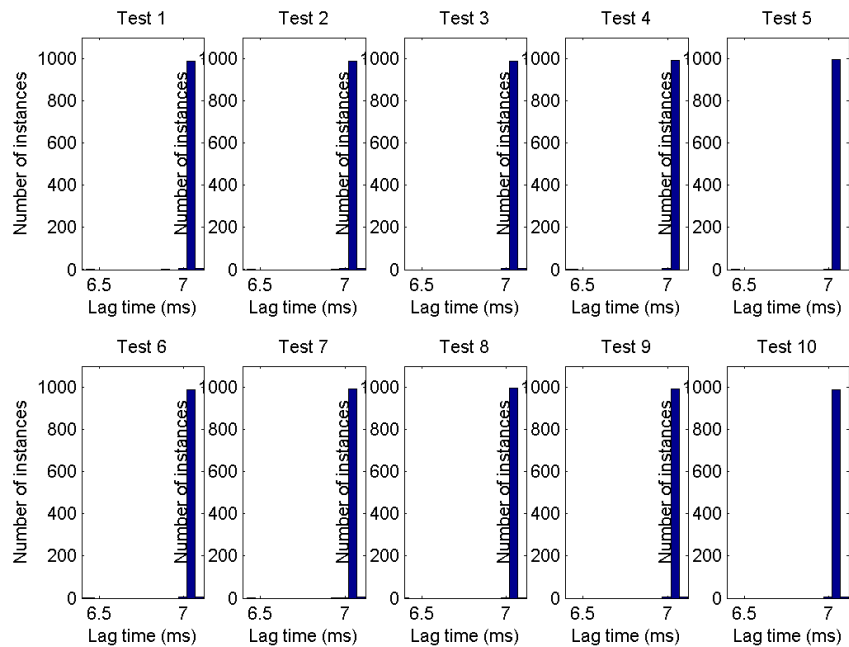
Histograms of All Tests, Computer 2, Windows 7, USB, Moving Mouse



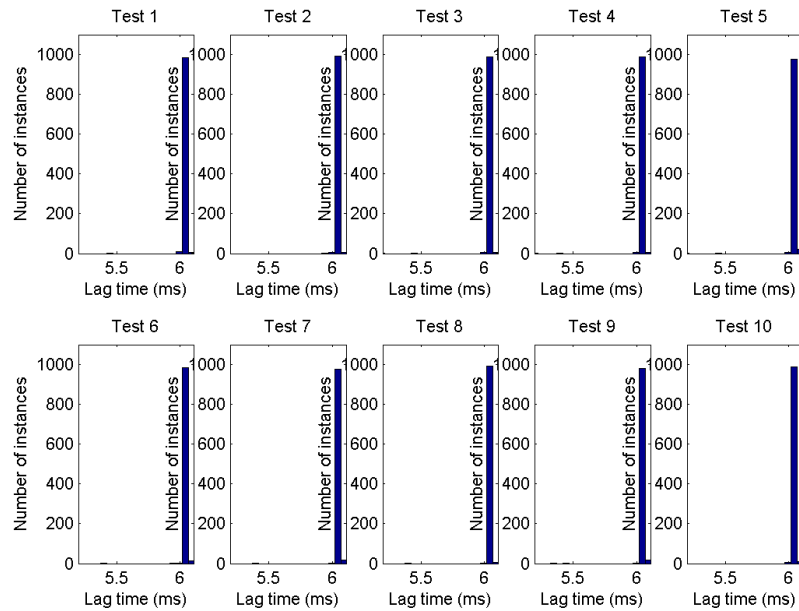
Histograms of All Tests, Computer 2, Windows 7, USB, Mouse Still



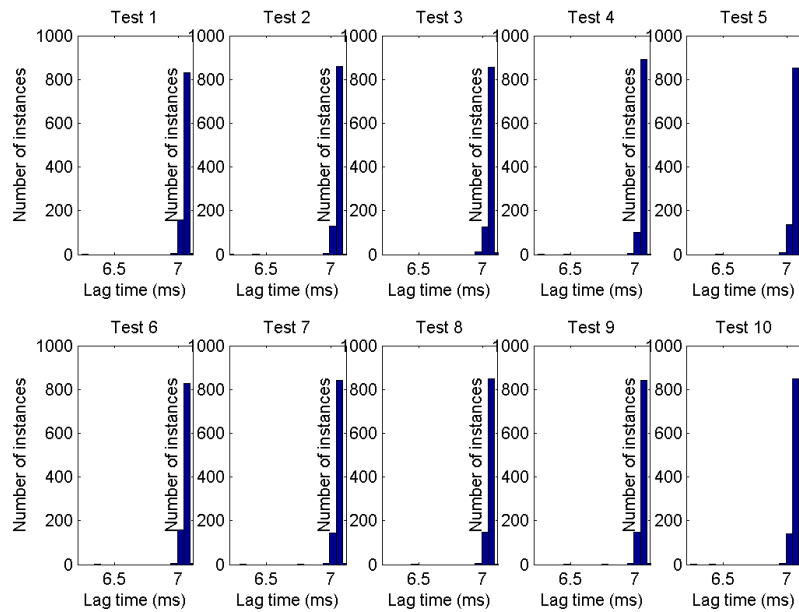
Histograms of All Tests, Computer 2, Windows 7, Serial, Not Moving Mouse



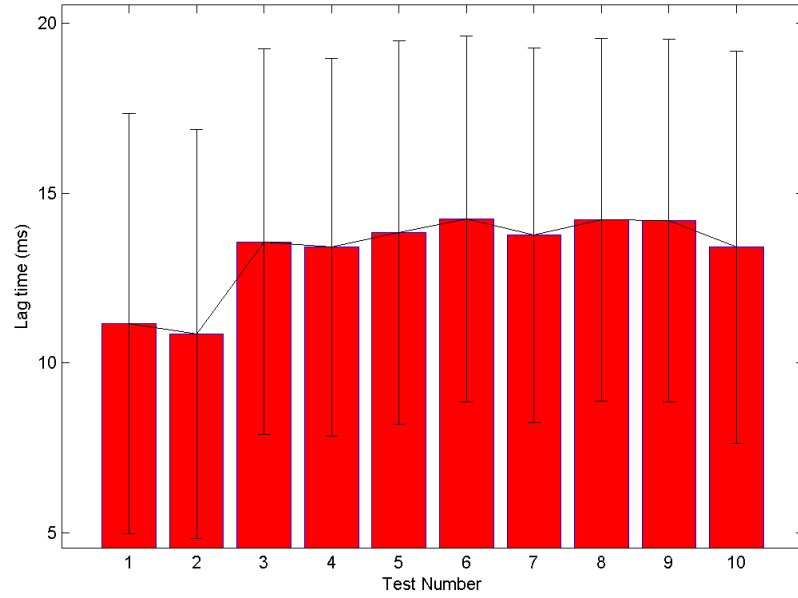
Histograms of All Tests, Computer 2, Windows 7, Serial, No Device



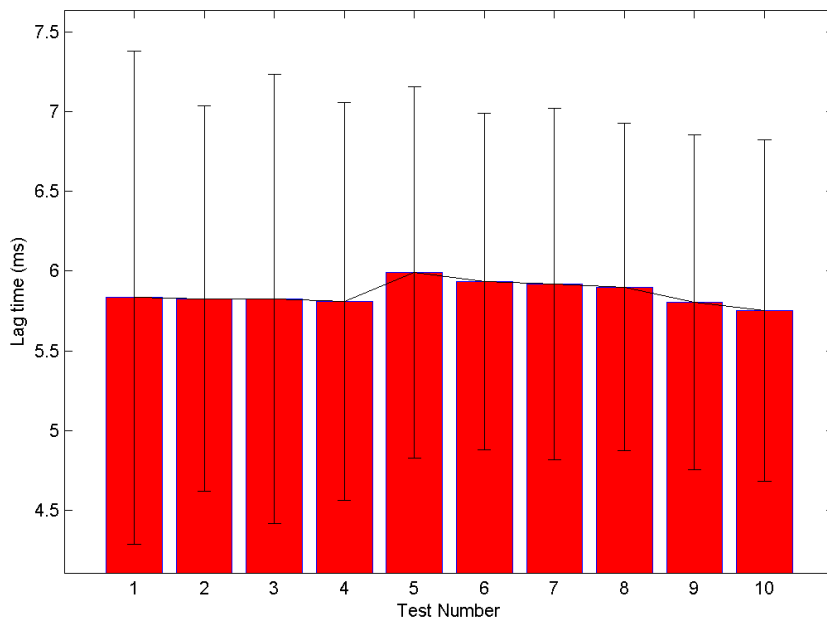
Histograms of All Tests, Computer 2, Windows 7, Serial, Moving Mouse



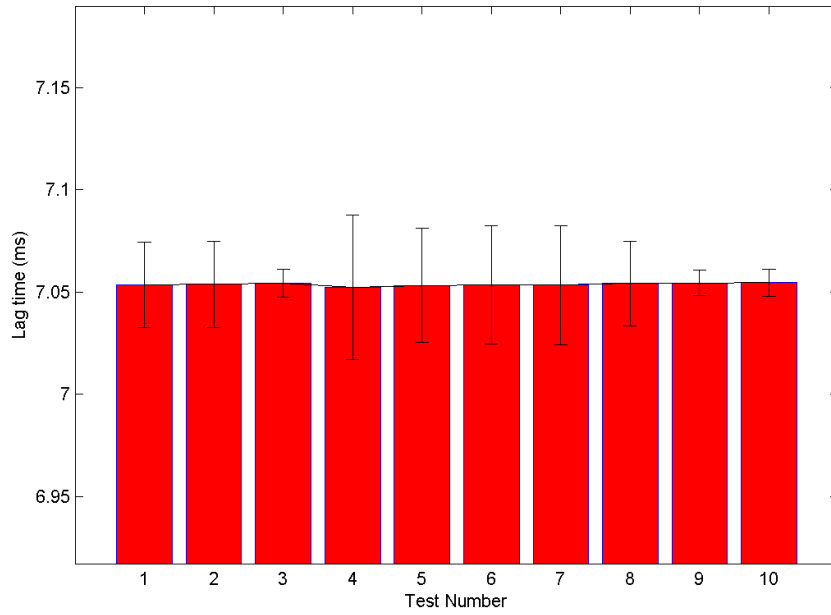
Mean Lag Time Across All Tests, Computer 2, Windows 7, USB, Moving Mouse



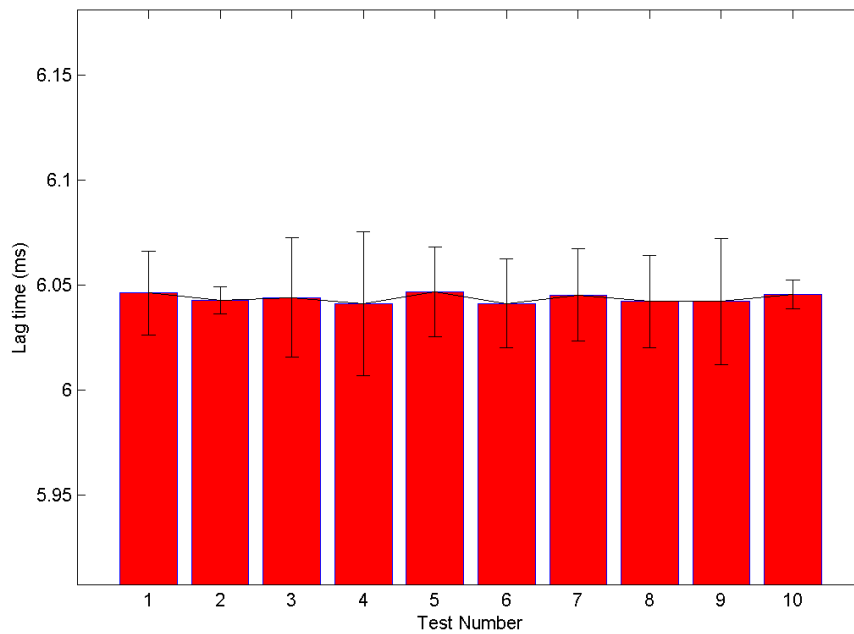
Mean Lag Time Across All Tests, Computer 2, Windows 7, USB, Mouse Still



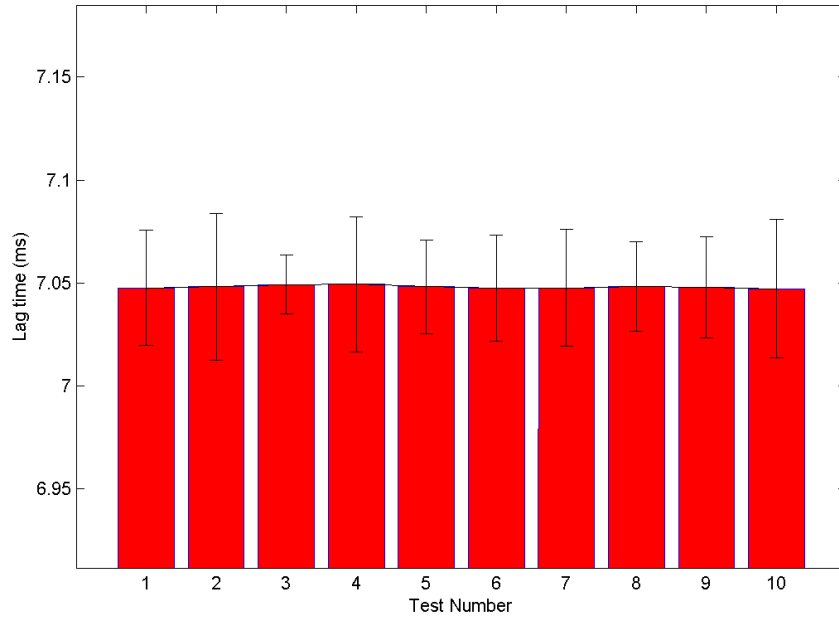
Mean Lag Time Across All Tests, Computer 2, Windows 7, Serial, Not Moving Mouse



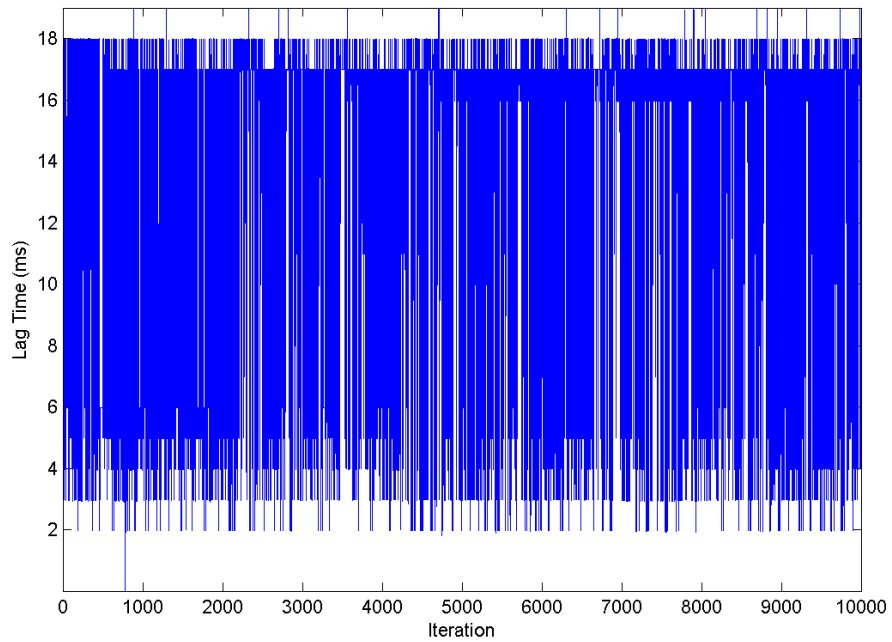
Mean Lag Time Across All Tests, Computer 2, Windows 7, Serial, No Device



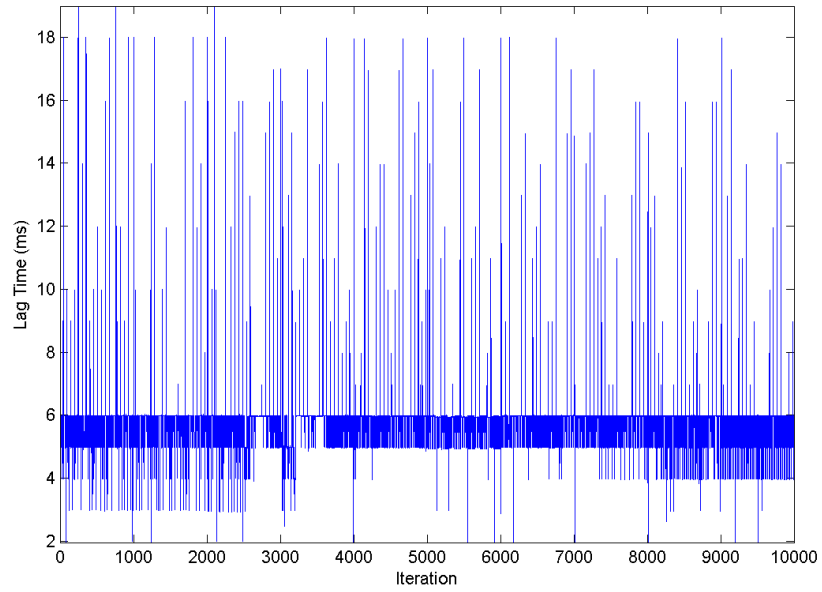
Mean Lag Time Across All Tests, Computer 2, Windows 7, Serial, Moving Mouse



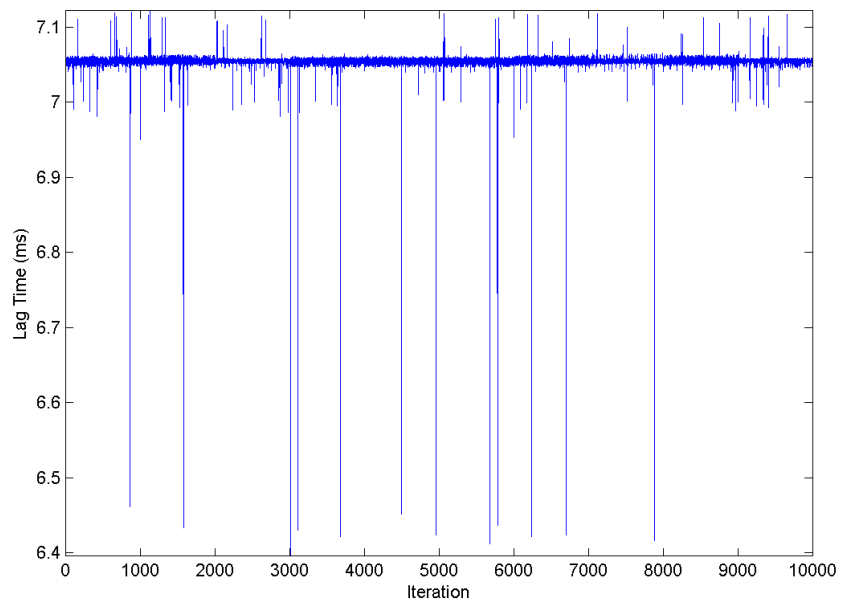
Plots of All Tests, Computer 2, Windows 7, USB, Moving Mouse



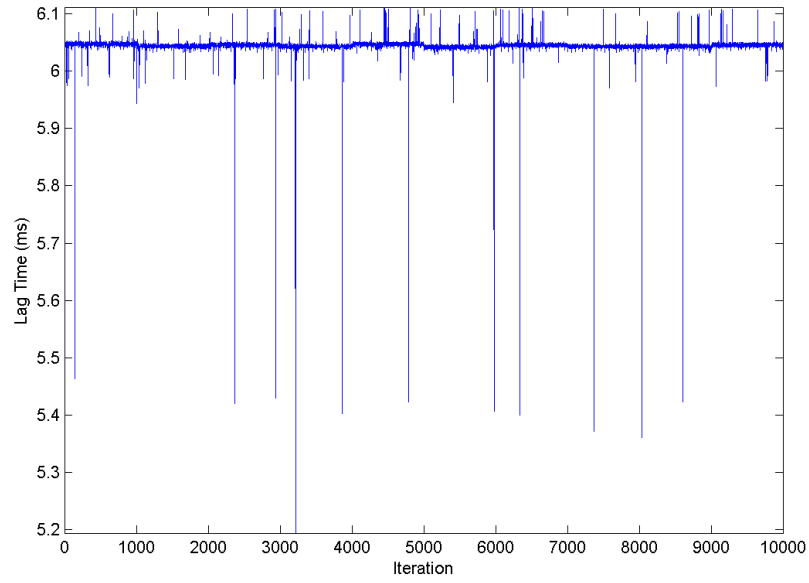
Plots of All Tests, Computer 2, Windows 7, USB, Mouse Still



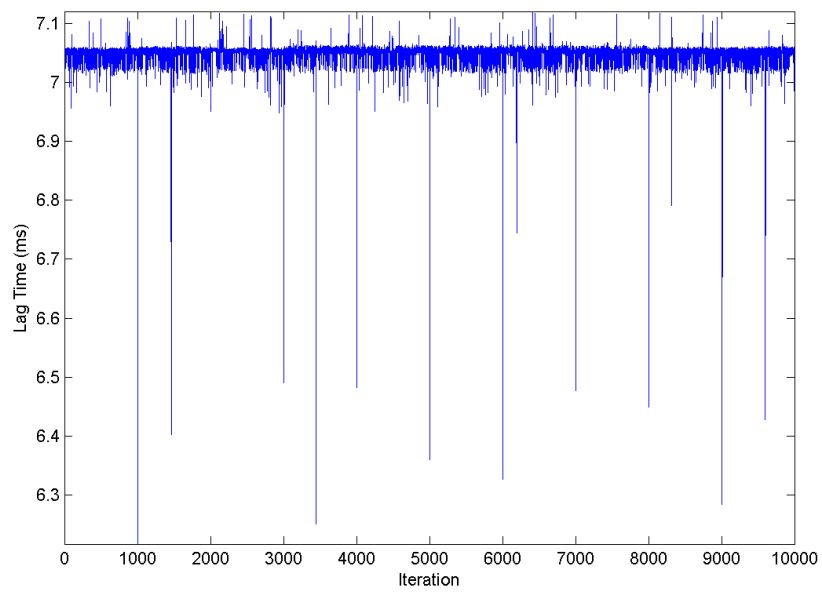
Plots of All Tests, Computer 2, Windows 7, Serial, Not Moving Mouse



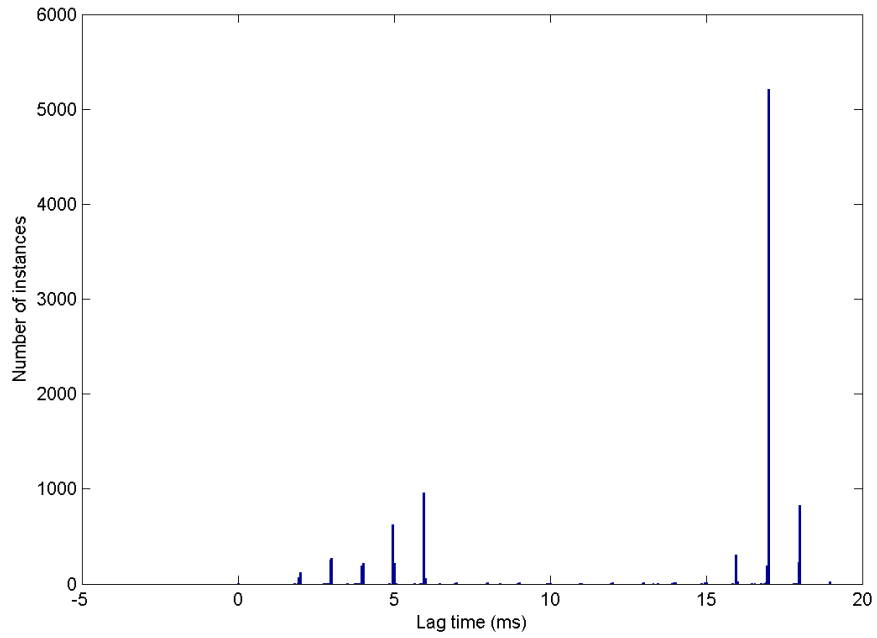
Plots of All Tests, Computer 2, Windows 7, Serial, No Device



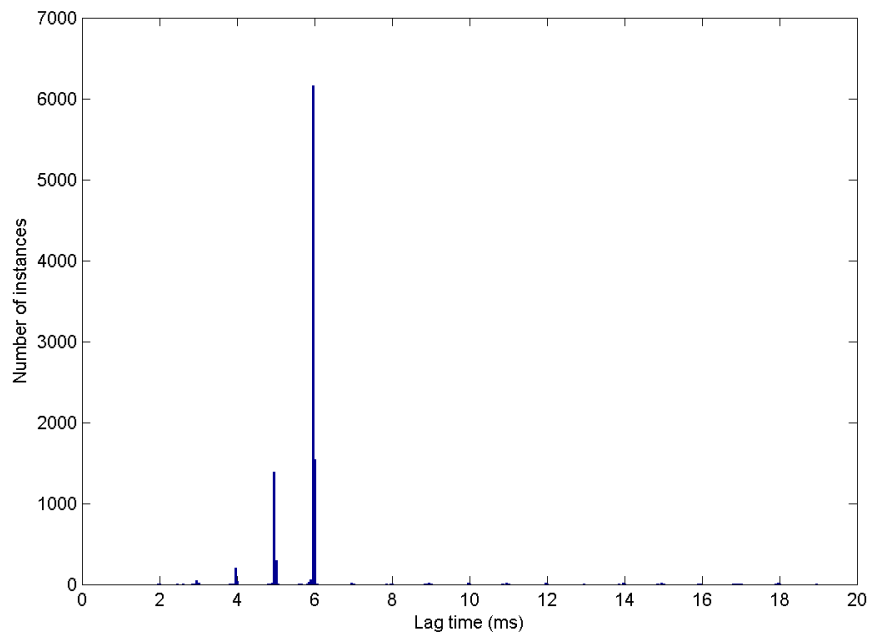
Plots of All Tests, Computer 2, Windows 7, Serial, Moving Mouse



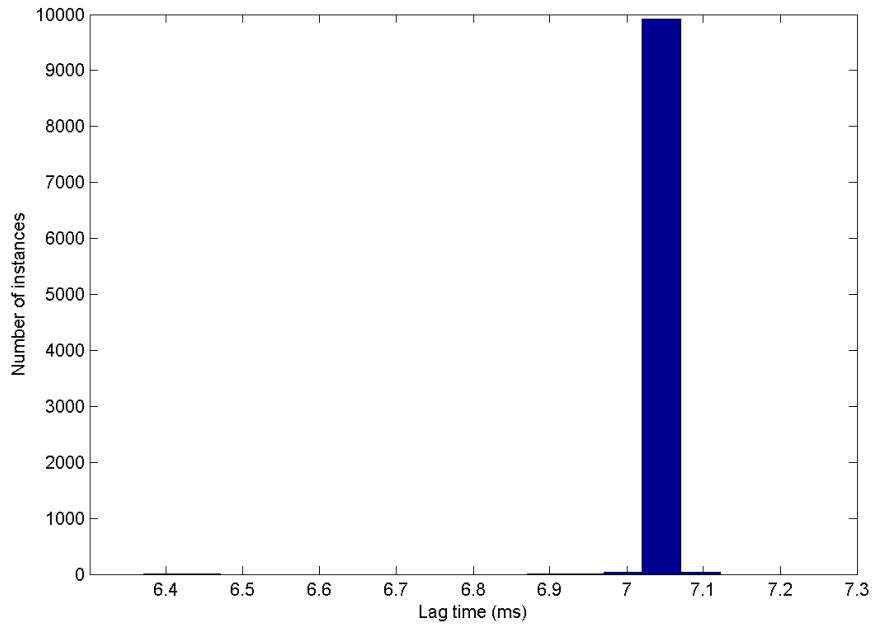
Histogram of All Tests, Computer 2, Windows 7, USB, Moving Mouse



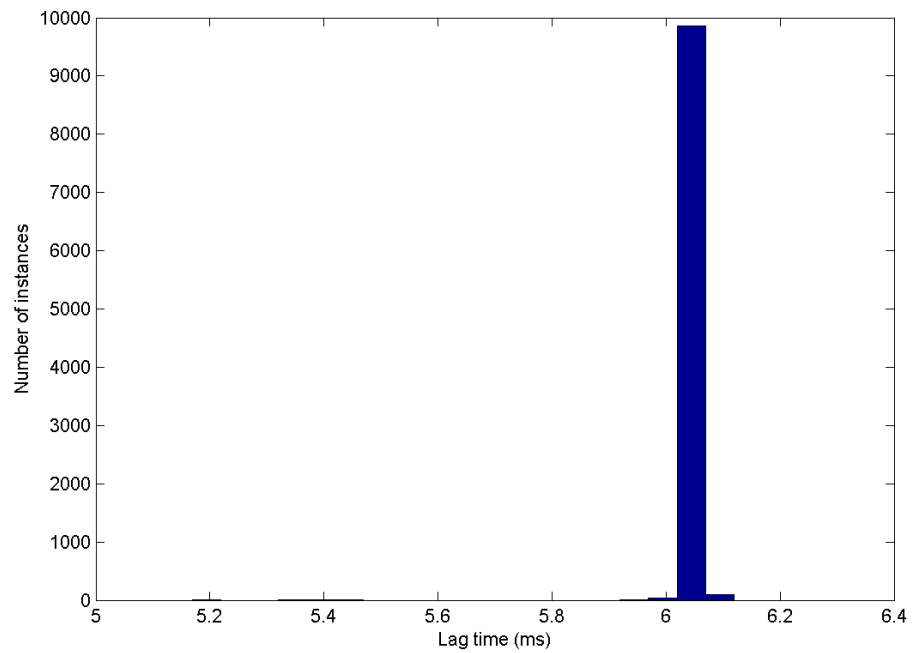
Histogram of All Tests, Computer 2, Windows 7, USB, Mouse Still



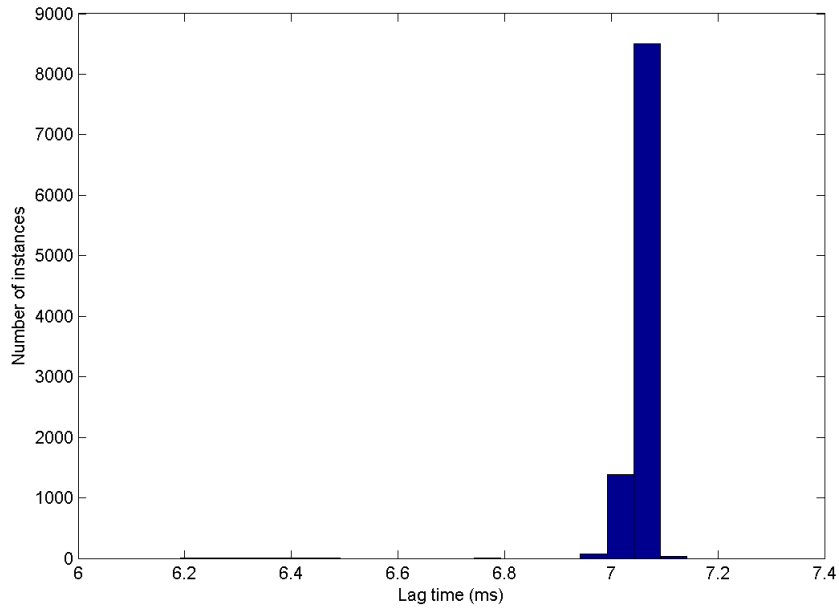
Histogram of All Tests, Computer 2, Windows 7, Serial, Not Moving Mouse



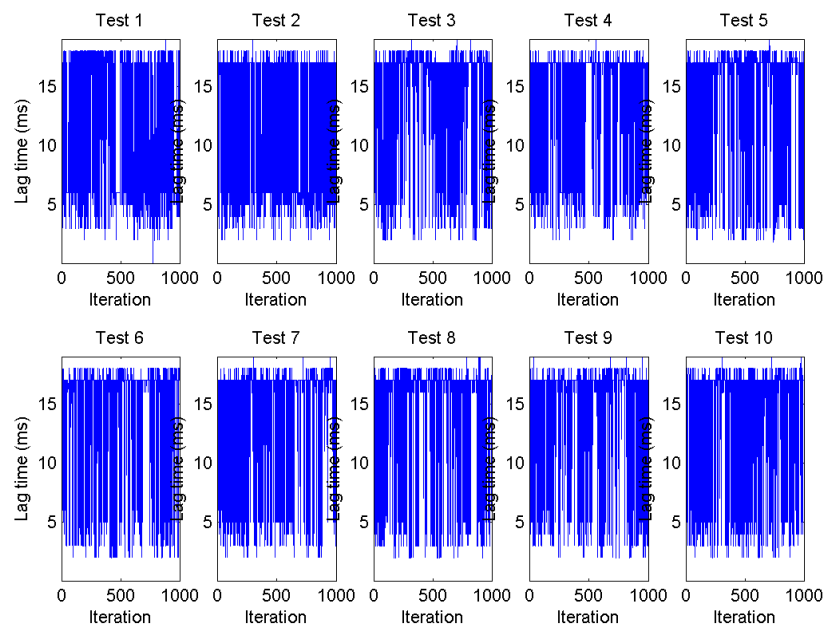
Histogram of All Tests, Computer 2, Windows 7, Serial, No Device



Histogram of All Tests, Computer 2, Windows 7, Serial, Moving Mouse

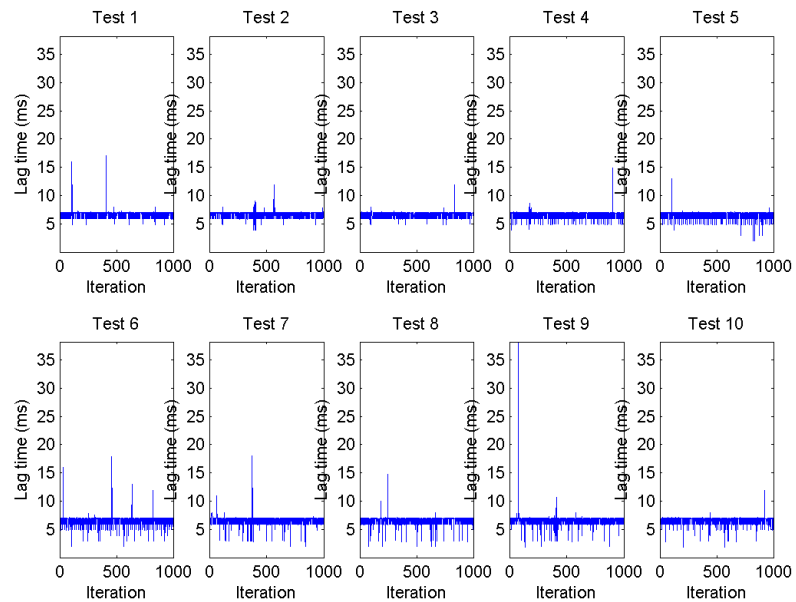


Plots of All Tests, Computer 2, Windows 7, USB, Moving Mouse

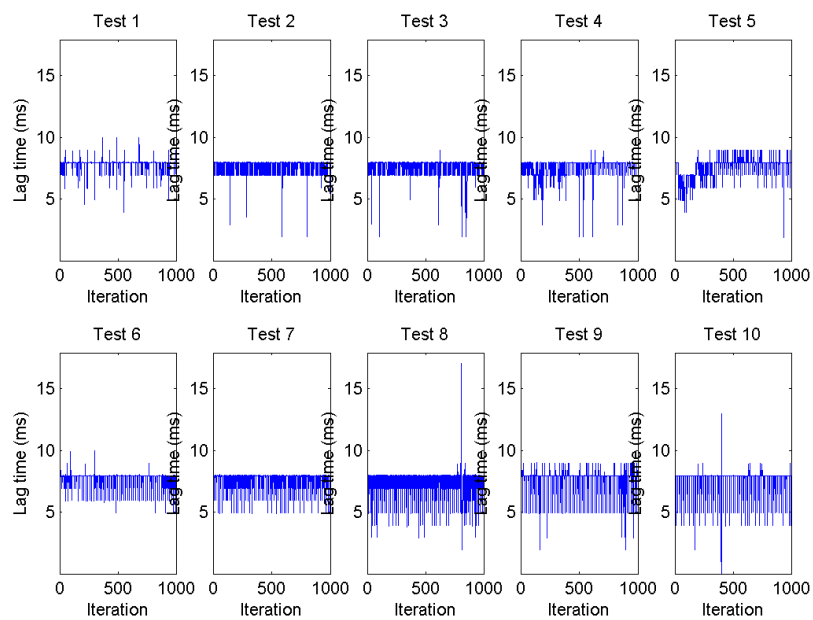


Operating Systems

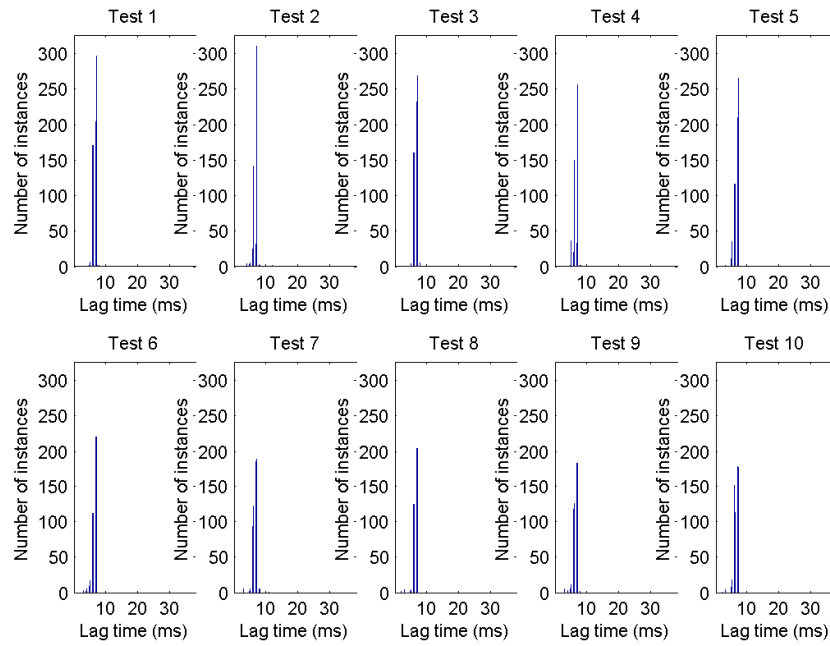
Plots of All Tests, Computer 3, Windows XP, USB



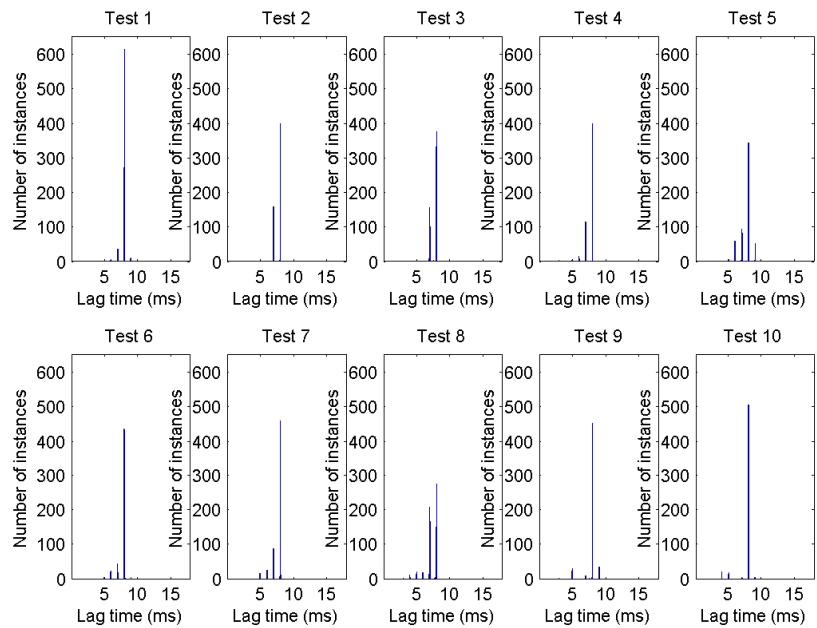
Plots of All Tests, Computer 3, Windows 7, USB



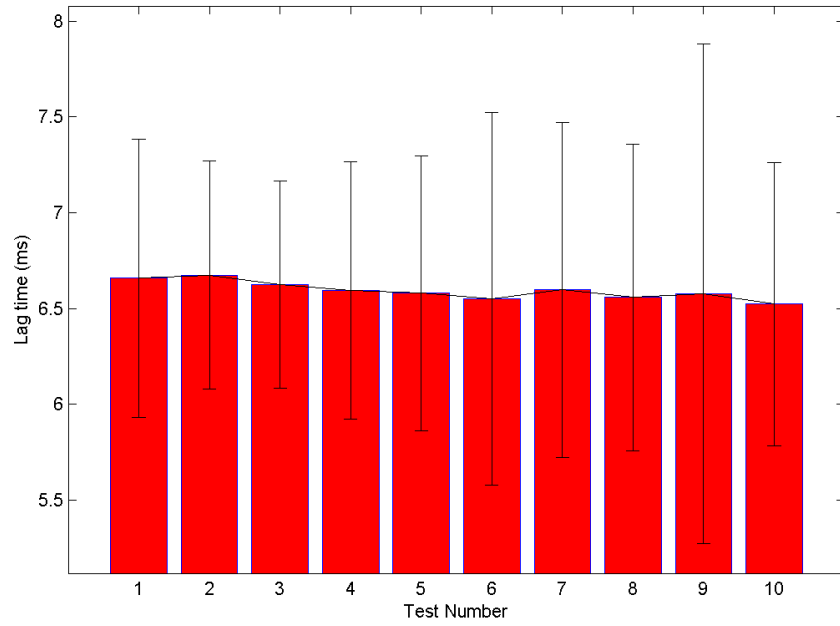
Histograms of All Tests, Computer 3, Windows XP, USB



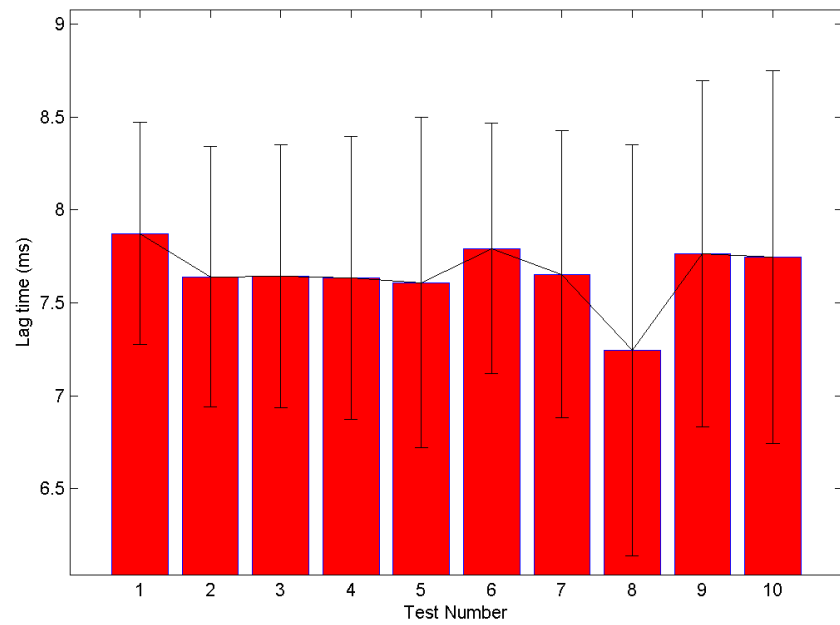
Histograms of All Tests, Computer 3, Windows 7, USB



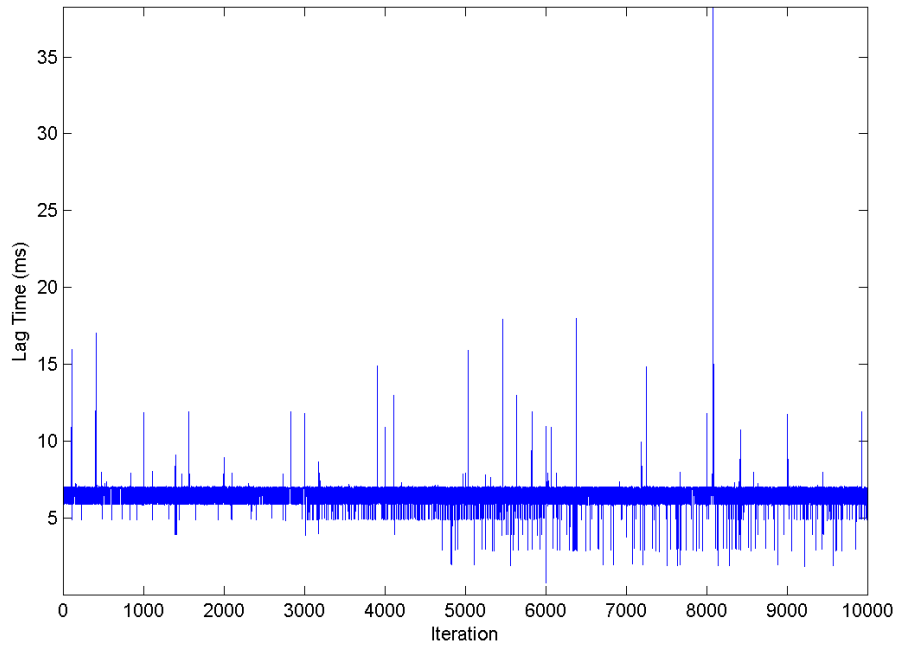
Mean Lag Time Across All Tests, Computer 3, Windows XP, USB



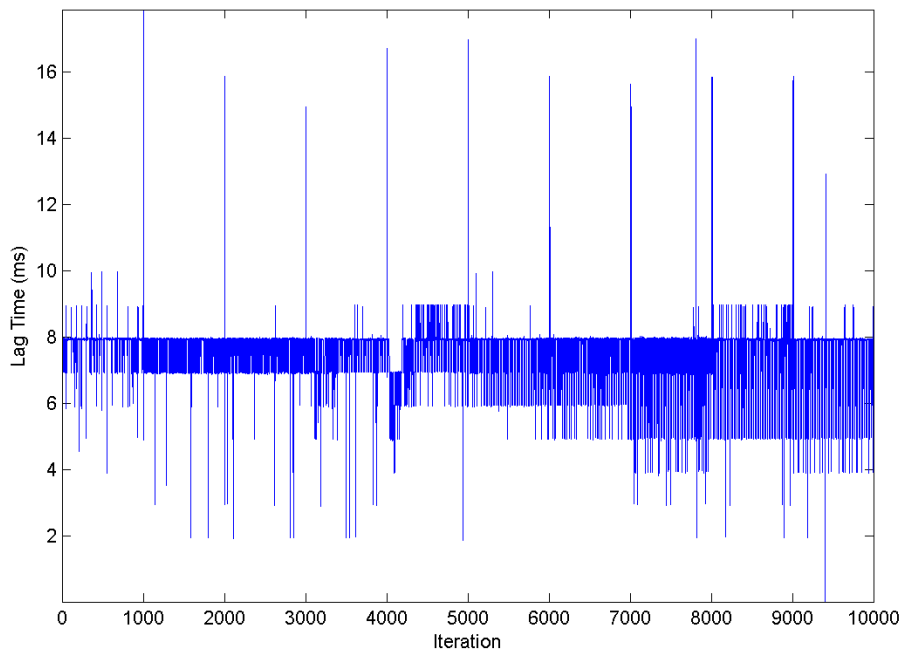
Mean Lag Time Across All Tests, Computer 3, Windows 7, USB



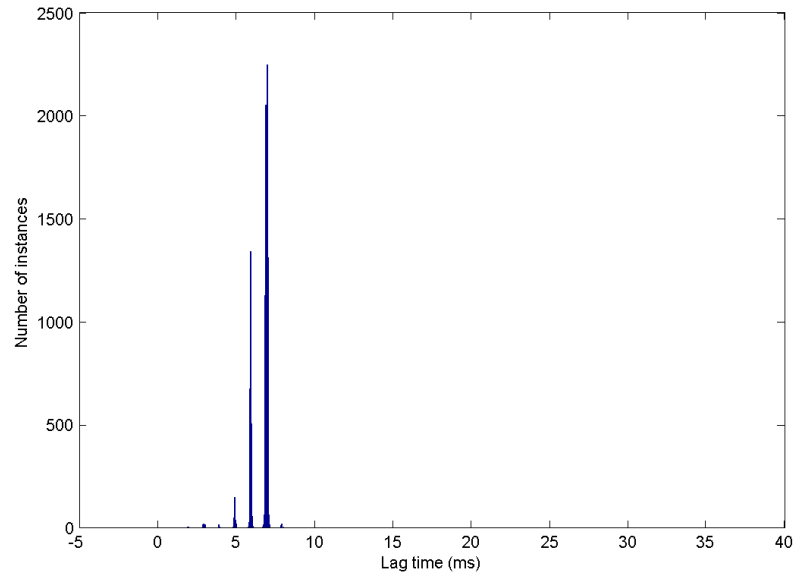
Plots of All Tests, Computer 3, Windows XP, USB



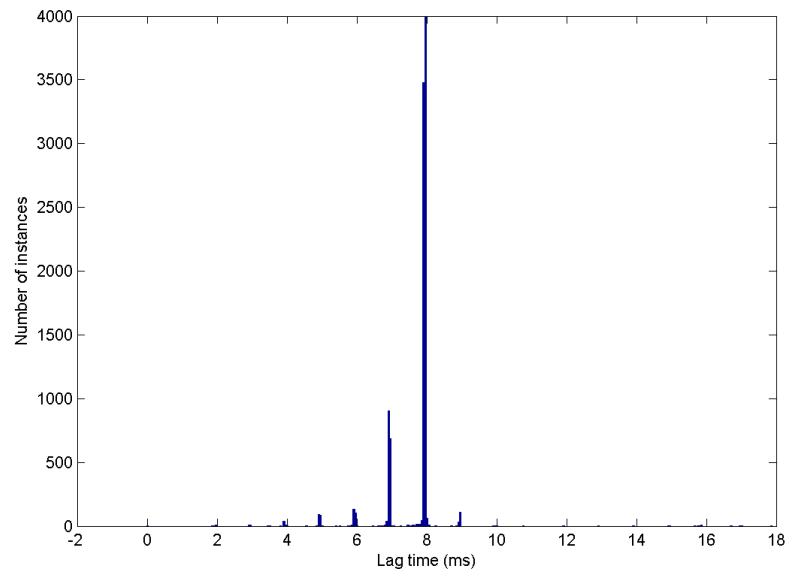
Plots of All Tests, Computer 3, Windows 7, USB



Histogram of All Tests, Computer 3, Windows XP, USB



Histogram of All Tests, Computer 3, Windows 7, USB



Appendix E: Development Platform Selection Spreadsheet

Manufacturer	Development Boards	Price	Processor	Speed	Memory	I/O po UARTs	Hardware Debugging Breadboard
	XMEGA-E5 Explained eval		23 ATmega32E5	32 MHz	32 KB Flash 4 KB SRAM 1024 Bytes EEPROM	26	2 N
	XMEGA-C3 Explained eval		23 ATmega384C3	32 MHz	384 KB Flash 32 KB SRAM	50	3 N
	XMEGA-A3BU Explained		23 ATmega328A3BU	32 MHz	256 KB Flash 16 KB SRAM 4096 Bytes EEPROM	47	6 N
	AVR Dragon Programmer		49				Necessary for all kits who debug capabilities
Arduno	Mega 2560		8-bit ATmega2560	16 MHz	256KB flash, 8 KB SRAM 1KB EEPROM	54	4 N
Cypress	PSoC 1		40 Cypress's proprietary 8-bit M8C core	24 MHz	4 KB to 32 KB flash, 256 bytes to 2 KB SRAM	64 depends	?
	PSoC 3 dev kit		99 Single-Cycle 8051 core	67 MHz	8 KB to 64 KB flash, 3 KB to 8 KB SRAM	72 depends	Y
	PSoC 4		99 ARMv7e Cortex-M0	48 MHz	16 KB to 32 KB flash, 4 KB SRAM	96 depends	Y
	PSoC 3LP dev kit		99 32-bit ARM Cortex-M0	67 MHz	256 KB Flash 64 KB SRAM	72 depends	Y
	PSoC 4 Pioneer Kit		25 ARMv7e Cortex-M0			depends	Y
Texas Instruments	Ti-UC Series Launchpad		13 32-bit ARM Cortex-M4	80 MHz	256 KB Flash, 32 KB SRAM 2 KB EEPROM	43	8 Y
	MSP430 Launchpad		13 26 MHz MSP430	25 MHz	128 KB Flash 8 KB SRAM	62	2 Y
	C2000 Launchpad		17 TMS320F380	80 MHz	64K Flash, 12K SRAM	22	1 Y
	Hercules Launchpad		20 32-bit ARM Cortex-FM42-132	100 MHz	384k flash, 32k SRAM	45	1 Y
Microchip	PIC18 Development Kit		165 8 bit PIC18F8722 and PIC18F87J11	40 MHz	128 Kb flash 3936 bytes SRAM 1024 bytes EEPROM	54	2 Y
	Explorer 16 starter kit		300 for any of their 16 bit or 32 bit chips				Y
	PIC 32 starter kit		50 PIC 32	80 MHz	512 Kb flash 32K SRAM		
Maxim Integrated	MAXQ2610-KIT Evaluation Kit		101 16-bit MAXQ610	12 MHz	64 Kb flash 2Kb SRAM 4 Kb ROM	4 (8)	2 Y
	MAXQ2010-KIT Evaluation kit		127.5 16-bit MAXQ2010	32?	32K x 16 Flash 2 Kb SRAM	7 (8)	2 Y
Silicon Labs	SIM3L3xx-B-DK		99 32-bit ARM Cortex-M3	80 MHz	32-256KB flash 8-32 KB SRAM	5 (16)	4 Y
	SIM3L3xx-B-EDK		299 32-bit ARM Cortex-M4	80 MHz		5 (16)	4 Y
	SIM3U3xx-B-DK		99 32-bit ARM Cortex-M5	80 MHz		5 (16)	4 Y
	SIM3U3xx-B-EDK		349 32-bit ARM Cortex-M6	80 MHz		5 (16)	4 Y
Intel	Galileo		80 32-bit Quark SoC X1000	400MHz	512 Kbytes SRAM 256 DRAM	14	2 N
Rasperry Pi	Model A		25 ARM11	700MHz	256MB RAM	17	1 N
	Model B		35 ARM11	700MHz	512MB RAM	17	1 N

