A Low Cost Wireless Sensor Interface for the Quantification of Tremor in Parkinson's Disease

A Thesis

Presented to

The Faculty of the Department of Biomedical Engineering

University of Houston

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

In Biomedical Engineering

By

Hasan Siddiqui

August 2016

A Low Cost Wireless Sensor Interface for the Quantification of Tremor in Parkinson's Disease

_____

Hasan Siddiqui

Approved:

_____

Chair of the Committee, Nuri F. Ince, Assistant Professor
Department of Biomedical Engineering
University of Houston

Committee Members:

_____

Ahmet Omurtag, Assistant Professor,
Department of Biomedical Engineering
University of Houston

_____

Yingchun Zhang, Assistant Professor
Department of Biomedical Engineering
University of Houston

_____

Joohi Jimenez-Shahed, M.D.
Department of Neurology
Baylor College of Medicine

_____

Suresh K. Khator, Associate Dean,
Cullen College of Engineering

_____

Metin Akay, Founding Chair and Professor,
Department of Biomedical Engineering
University of Houston

A Low Cost Wireless Sensor Interface for the Quantification of Tremor in Parkinson's Disease

An Abstract

of a

Thesis

Presented to

The Faculty of the Department of Biomedical Engineering

University of Houston

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

In Biomedical Engineering

By

Hasan Siddiqui

August 2016

Abstract

Deep brain stimulation surgery involves placing an electrode in the deep brain to suppress the motor symptoms of patients with Parkinson's disease (PD). Currently physicians use the standard Unified Parkinson's Disease Rating Scale to describe the tremor. This scale involves subjective anchor-based observations by the clinical expert. A wireless accelerometer system is presented that was built from off the shelf components to objectively quantify tremor scores.

The system consists of a Teensy microcontroller and two accelerometers. It wirelessly transmits the readings through a Bluetooth module. The data is received by a custom C/C++ that parses and transmits the data to the Simulink environment for real-time visualization and analysis. The system is used to record data from patients with PD during and after DBS surgery. In this thesis, we describe the wireless accelerometer system in detail and study the correlation of sensor readings with UPDRS scores in the different DBS states. In particular, we provide data showing that such a system can be used for the objective quantification of tremor symptoms in PD patients.

Table of Contents

List of Figures

List of Tables

1 Overview

1.1 Parkinson's Disease

Parkinson's disease (PD) is a neurodegenerative disorder that affects approximately one millions Americans [1]. Symptoms include bradykinesia, rigidity, postural instability, freezing of gait, and tremor.

Tremor is defined as an involuntary rhythmic oscillation of a body part [2]. The characteristic tremor of PD, is usually a 3-6 Hz distal resting tremor, but patients with PD may also exhibit postural and kinetic tremors [3]. Rest tremor occurs when a body part is at rest or relaxed while postural tremor occurs when a body part is maintained at a position against gravity, and kinetic tremor occurs during the movement of a body part. Rest tremor is the most common and most recognizable symptom of PD, and is, by itself, a positive diagnostic criterion for PD [4].

PD is caused by a loss of pigmented dopaminergic neurons in the substantia nigra [5]. PD is usually often treated with a dopamine (DA) precursor, levodopa. This drug effectively does a good job at improvising the motor symptoms of PD, but is associated with the long-term development of motor complications.

Deep brain stimulation (DBS) surgery is widely used in PD patients where medications do not sufficiently control motor symptoms, often with dramatic clinical benefit. DBS surgery involves implanting electrodes into specific targets in the brain and connecting the electrodes to an implanted pulse generator (IPG). There are two parts to this procedure. The first is where the electrodes are implanted in the deep brain, and the second is when the IPG is connected and implanted. The IPG is then programmed to deliver stimulation at a particular voltage, rate, and pulse width. DBS programming for

each patient includes the selection of optimal electrode contacts, voltage, and pulse width to maximize efficacy and minimize side effects of stimulation [5][6].

Placement of high frequency stimulating electrodes in the region of the ventral intermediate nucleus of the thalamus can markedly reduce tremor in these conditions, and stimulation of either the subthalamicnucleus or the internal segment of the globus pallidus may not only reduce tremor, but also decrease bradykinesia, rigidity, and gait impairment that plague people with PD. Furthermore, DSB is approved for Essential Tremor, PD, dystonia and OCD and is under investigation for the treatment of pain, depression, epilepsy and Tourette syndrome.[7].

1.2 Unified Parkinson's Disease Rating Scale

Currently, the main method of describing the tremor is using the Unified Parkinson's Disease Rating Scale (UPDRS). This scale involves subjective anchor based observations made by a clinical expert. It is a 10-20 minute test that consists of a several questions from 4 different categories. Each question is answered on a 5 point scale from 0-4 (0=normal, 1=slight, 2=mild, 3=moderate, 4=severe). The UPDRS scale has received an update in 2008 that addresses some problems with the original scale, but still contains inherent problems that arise from using human observations as the main measurement tool. In particular, the motor sub-score (part III) is subject to difficulties with inter-rater reliability, depends on the rater's experience, and only represents symptom severity at a single point in time. These problems include both variations in scores by different raters as well as only using a small temporal resolution of ~10 minutes to determine severity. Physicians can use the clinical history and examination observations from clinic appointments to determine how best to titrate medications for optimal results [8][9][10].

1.3 Tremor Analysis

Tremor is defined as “a rhythmic, involuntary movement of a body part” [2]. Tremor exists in all humans in small magnitude and can be physiological. However, pathologic tremor disorders exist, and are classified according to phenomenology into three categories: rest, postural and kinetic tremor [11]. Physiological tremor has different origin and cause compared to pathological tremor and manifests differently in terms of amplitude and frequency. Pathologic tremors such as essential tremor and Parkinson’s disease are associated with a characteristic frequency range, as noted in Table 1.1. The frequency of the pathological tremor tends to remain constant with slight variations [13] [14]. Tremor is the most recognizable symptom of PD, so it is also one of the most studied and used metrics to help differentiate PD from other tremor disorders.

Table 1.1 Involuntary Hand Tremor[3][4][14]

Tremor Type	**Frequency**
Normal Hand Tremor	9-25 Hz
Essential Tremor	5-10 Hz
Parkinson’s Disease	3-6 Hz

Tremor analysis refers to the recording of identifiable characteristics of tremor through the use of special hardware or techniques. The two most useful features from the tremor analysis are frequency and amplitude. Tremor frequency refers to the number of oscillations per second and is usually measured in cycles per second (Hz), while tremor amplitude refers to the degree of linear or angular displacement of the limb and is typically measured in degrees or millimeters. Although tremor frequency is usually used to classify tremor, tremor amplitude has an important role in describing the severity of

tremor. The degree of linear or angular displacement of the limb or body part, or the tremor amplitude, is generally measured in millimeters or degrees.

There are a few different methods used to analyze tremor. The most commonly used are electromyography, spirograms, and accelerometry. [15]. EMG provides additional useful information about the activity of muscles involved in the generation of tremor. EMG activity may be recorded using needle, wire electrodes, or more typically surface electrodes overlying active muscles. The EMG can provide information about motor unit recruitment and synchronization and can also clarify the relationship between involved muscles and tremulous movements, revealing whether antagonist muscles (such as flexors and extensors of the wrist) are working at the same time or alternately to produce tremor. To utilize the EMG most appropriately in tremor analysis, the signal has to be processed by rectification and integration or smoothing to place its frequency profile into the tremor range. The recordings require a relatively high sampling frequency of 500 Hz or more. EMG recordings require a lengthy setup for using needle or wire electrodes. EMGs that utilize surface electrodes may have issues in locating landmarks for proper electrode placement because of loose skin or excess adipose tissue [16][17].

Archimedean spirals drawn on a digitizing tablet can be analyzed in both the x–y plane of the tablet and the z plane of pen pressure perpendicular to the tablet. By mathematically ‘‘unraveling’’ drawn spirals and averaging multiple trials together, tremor characteristics such as frequency, direction, and amplitude can be detected and quantified, as well as an abundance of other variables, including drawing speed and acceleration, loop-to-loop width tightness and variation, and drawing pressure over time [15][18].

Accelerometry is accomplished through the use of accelerometers that measure static or dynamic acceleration forces. Miniature accelerometers can be attached to the limbs and occasionally the head, neck, or trunk, and usually do not interfere with voluntary or involuntary movements. These signals can be fed into a microcontroller or computer to filter out low frequency signals such as drift and high frequency electronic interference. Integration is needed to determine the displacement of the oscillating body part and to better perceive the sinusoidal motion of the tremor [19]. Microcontrollers are capable of recording and analyzing large amounts of accelerometric data quickly and efficiently. They can give almost instantaneous accelerometer signals. The signals can be either pre- or post-processed to remove low frequency noise such as drift or high frequency noise such as 60 Hz power signals from electricity. A single accelerometer's data can be difficult to appreciate clinically because sinusoidal motion is not easily perceived in accelerometric or rotational units. If using a gyroscope or multiple accelerometers, one can calculate the displacement using mathematical double integration. Tremor analysis is most useful when the clinical signs are subtle, such as distinguishing between Parkinsonian tremor and essential tremor.

1.4 Previous Work

Although accelerometers were proposed for the quantification of movement as early as the 1960s, the technology was not at a point where they could be efficiently utilized due to cost, reliability and size, which often disturbed natural movements [20]. Since its beginnings, accelerometer technology has evolved to the point of demonstrating high levels of quality and reliability. Their gradual miniaturization plus the decreased cost

from the economies of scale, has brought the technology to a point where they are more practical for recording in clinical settings.

Salarian et al. have studied movement characteristics with their wired accelerometer system shown in Figure 1.1. Their system used a wired accelerometer in a custom casing to record directly to an SD flash card. This storage solution didn't allow real time visualization/processing. Their device had a bulky size for the sensor casing that could alter natural movement [10][21][22].

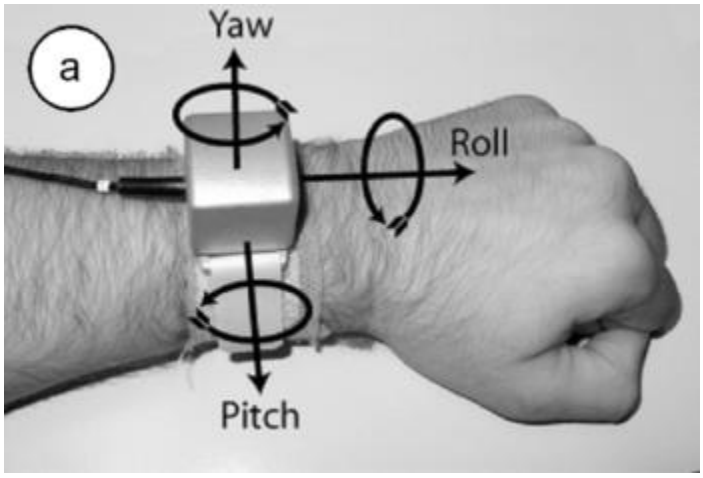


Figure 1.1 Salarian *et al.* device for capturing accelerometer data [10].

In the past half-decade, wireless accelerometer systems have been increasing in use. LeMoyne et al. have used wireless systems to record data with patients, without tethering them to wires[23]. One of the earliest wireless accelerometer systems used an iPhone for the sensor as shown in Figure 1.2. This was convenient because it did not require much construction or programming but remained expensive and bulky. Since then, the field has improved with more specialized devices and modules to record wirelessly [23][24].

Figure 1.2 LeMoyne *et al.* using an iPhone to capture accelerometer data wirelessly[23]

Based on the limitations of present technology, we present an accelerometry system that will accurately measure tremor characteristics through an inexpensive, lightweight, wireless system that is robust enough to capture data at a high level of fidelity, while still being modular and customizable enough to add to our existing research interface. An accelerometer based system does not require high sampling frequencies. 50 Hz to 100 Hz is high enough to capture the required data. The wireless interface removes sophisticated and crowded wires in the operating room space.

2 Hardware/Software

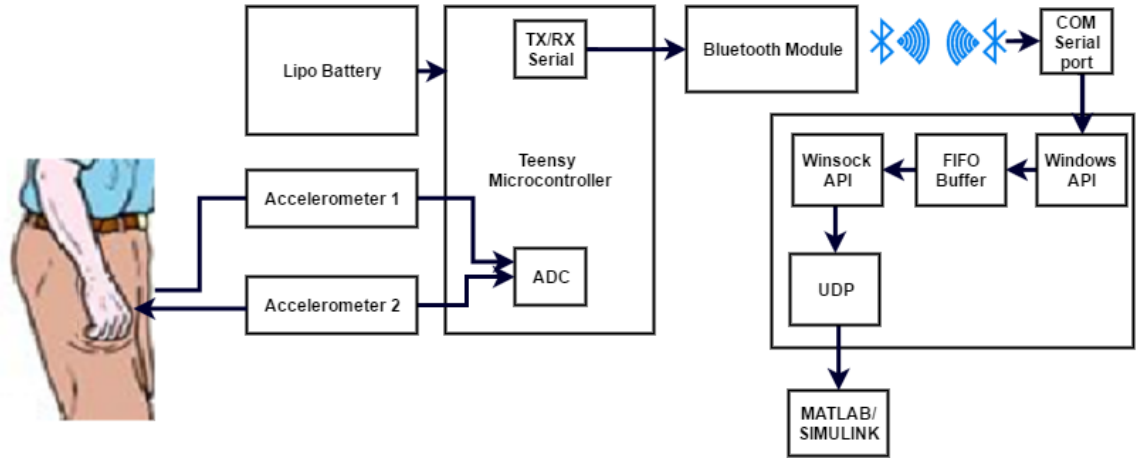


Figure 2.1 Schematic of the system from analog accelerometer data to digital recording and visualization in Simulink

2.1 System

A schematic diagram of the wireless accelerometer system is shown in figure 2.1. The system consists of a 72 MHz Teensy 3.1 microcontroller with a Cortex-M4 core that digitizes two GY-61 3-axis accelerometers with an ADXL335 chip with 10bit A/D resolution and 100Hz sampling frequency. The accelerometers were soldered to one end of a HyperThin Flexible HDMI cable (Sanho Corporation, Fremond, CA). The other end of the HDMI cable was connected to a 5pin JST connector. The matching plugs for the connectors were soldered to the microcontroller. The cable was chosen to allow the subjects full range of motion, while not weighing them down or getting tangled. It allowed for quick removal and storage of the system. The system wirelessly transmits the readings through an HC-05 Bluetooth module (BT) with a CC2541 chip at a baud rate of 115200.The data is received by a custom C++ program that parses and transmits the data through the local UDP port to be recorded by MATLAB/SIMULINK. The total price was $70.92, with the detailed list of components listed in table 2.1 below.

Table 2.1 Total cost of system

Item	Price
Teensy	$19.80
ADXL335 x 2	$14.95
HC-06 Bluetooth Module	$5.73
850mAh LiPo battery	$9.95
Hyperthin HDMI cable	$18.99
5-pin JST SM Plug + Receptacle	$1.50
Total	**70.92**

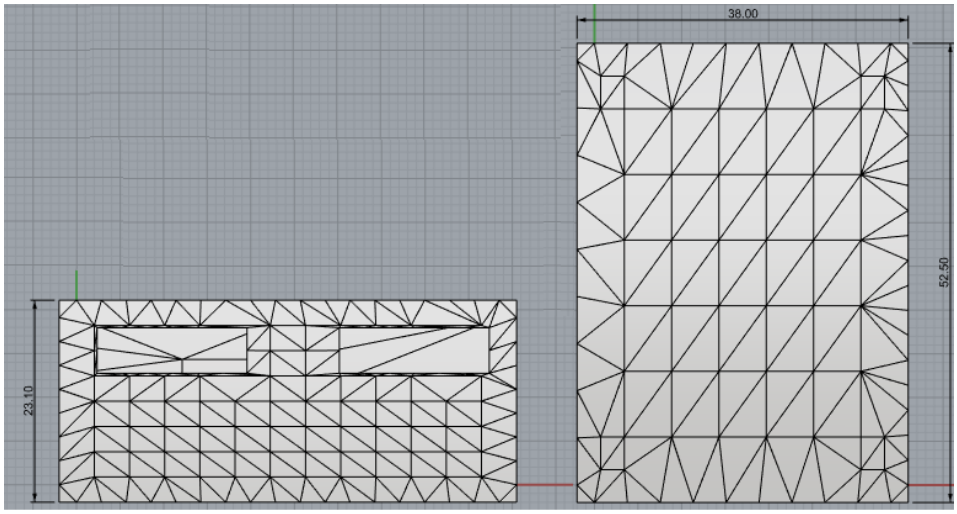


Figure 2.2 3D printed enclosure

The system was enclosed in a custom 3D printed box. The 3D file was created on Rhino3D (McNeel North America, Seattle, Washington). Figure 2.2 shows the 3D file with the dimensions of the box (52.0 mm x 38.0 mm x 23.1 mm). After assembly, the accelerometers could be unplugged for storage. Figure 2.3 shows the system before and after being assembled.

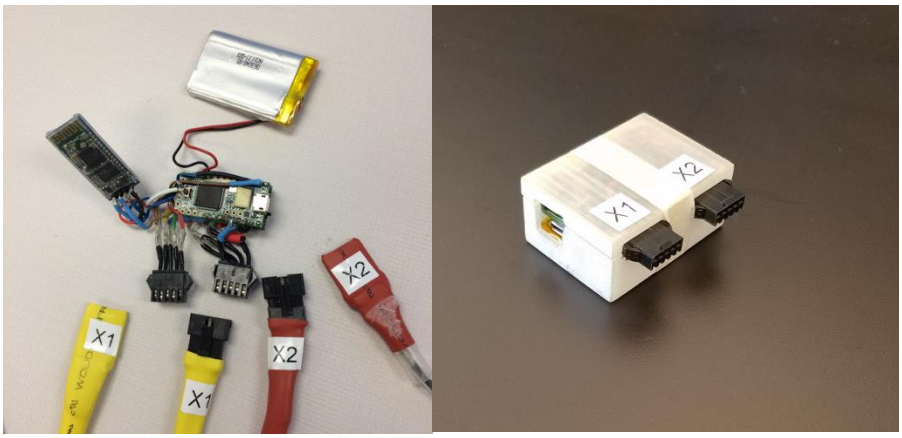


Figure 2.3 Accelerometer system before and after being assembled in 3D printed enclosure

2.2 Arduino

Arduino is the name of a family of microcontroller boards with low cost and diverse functionality which received noticeable interest in the past decade [25]. The boards are a combination of an ATMEL microprocessor including RAM, flash memory, and input/output channels. In this study, we use the Teensy 3.1 (PJRC, Portland, Oregon) as shown in figure 2.4. It runs on a 72 MHz Cortex-M4 core with 64 kb RAM and 256 kb flash memory. It has 2 ADCs with 13 bit possible resolution (10 bit used in our system).

Figure 2.4 Teensy 3.1 microcontroller.

Part of the Teensy/Arduino package is a programming environment, where code is written in a C-like language, and transferred to the Teensy using a USB cable. After

programming, a Teensy can work while being connected to a PC (and thereby transmit data), or operate standalone. The Arduino family can be used as a tool for a measurement platform. First of all, it connects easily by USB to a PC, and can transmit data using a serial port to these operating systems. It is also open source hardware, which means that everybody can access, modify, and use the board design. Likewise, the software is free and open source. This has led to both a low price of a board (~$20) and a large community that develops hardware and software compatible with the Arduino. The boards can be connected to actuators and sensors, either commercial or build from scratch. Many extensions of a board come as so-called shields, additional small boards that are plugged into an Arduino board. For instance, some shields provide wireless access (WiFi, Bluetooth, etc.), whereas others allow storing data on flash memory cards. In addition, this idea of extending a board by plugging in components is taken further with small components that can be connected by plugging them directly into the I/O pins. A pinout of the available I/O pins are shown in figure 2.5. Users can connect LEDs, small vibrating motors, buttons, accelerometers, gyroscopes, or hall sensors to the Arduino within seconds. With the active community and open source code, this guarantees an opportunity to use a wide range of sensors to create new kinds of measurement tools [26].

A USB cable connects Arduino and PC, and provides a serial connection. On the PC side, driver software creates a virtual serial (COM) port. This serial port can be accessed with any software that can communicate with a serial port. The driver offers several baud

rates; for instance, the driver for the Teensy offers 57,600 and 115,200 baud as the fastest speeds.

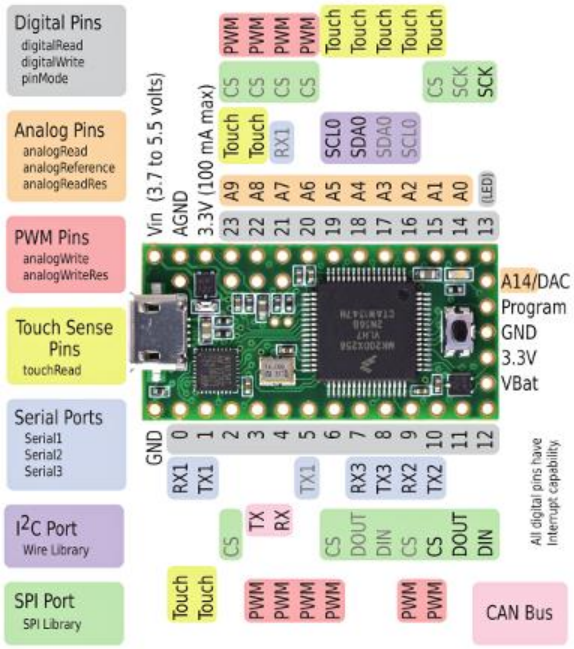


Figure 2.5 Teensy 3.1 pinout sheet

Despite its simplicity and low price, the Arduino can function as a reliable controller for experimental settings. When investigating how an Arduino can replace more complicated hardware as a standalone controller of experimental input and stimuli, D'Ausilio confirmed that signals generated by an Arduino were reliably constant in length and delay. He also found that typical combinations of input and output were performed with remarkable accuracy and precision, often with standard deviations of only microseconds [26], [27].

2.3 Accelerometer Sensor

Accelerometer sensors measure the acceleration experienced by the sensor and anything to which the sensor is directly attached. Accelerometer sensors have many applications. The most common commercial application is impact sensors for triggering

airbag deployment in automobiles: when the acceleration exceeds a certain amount, an accident is assumed and the airbags deploy. Such sensors are designed to be rugged and reliable, and are made in high volume and at low cost by several chip manufacturers. Airbag sensors don't need to be very accurate: with a threshold of ~50 g's, an accuracy of 1 to 2 g is acceptable [1]. When working with accelerometers in the earth's gravitational field, one should always consider the acceleration due to gravity. The signal from an accelerometer sensor can be separated into two signals: the acceleration from gravity, and external acceleration.

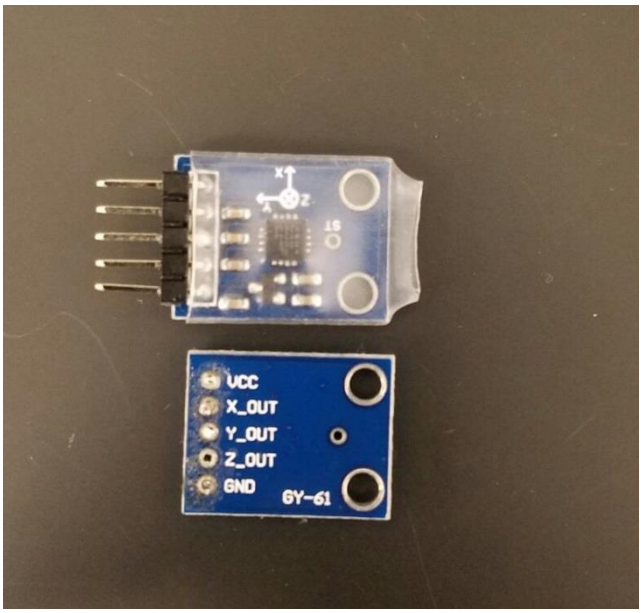


Figure 2.6 ADXL335 accelerometer modules. Top module has headers and wrapping already in place.

The goal of this thesis is to measure the 3 dimensional acceleration of the human hand with accuracy and precision, while not weighing down or preventing normal motion. With these necessities in mind, we required a small, lightweight accelerometer. The ADXL335 was adequate for our study. It is shown in figure 2.6 with and without the protective shrink wrap and headers. It is a complete 3-axis acceleration measurement system. The ADXL335 has a measurement range of ±3 g minimum. The output signals

are analog voltages that are proportional to acceleration. The accelerometer can measure the static acceleration of gravity in tilt-sensing applications as well as dynamic acceleration resulting from motion, shock, or vibration. The sensor is a polysilicon surface-micromachined structure built on top of a silicon wafer. Polysilicon springs suspend the structure over the surface of the wafer and provide a resistance against acceleration forces. Deflection of the structure is measured using a differential capacitor that consists of independent fixed plates and plates attached to the moving mass. The fixed plates are driven by 180° out-of-phase square waves. Acceleration deflects the moving mass and unbalances the differential capacitor resulting in a sensor output whose amplitude is proportional to acceleration. Phase-sensitive demodulation techniques are then used to determine the magnitude and direction of the acceleration [28][29].

2.4 Bluetooth

Bluetooth is a wireless communication technology for exchanging data over short distances. It uses a master-slave type of network gathering a maximum of eight active nodes. A Bluetooth unit includes a radio which operates in the 2.4GHz frequency in the free ISM-band (Industrial, Scientific, and Medical). Bluetooth devices communicate to each other through a piconet.

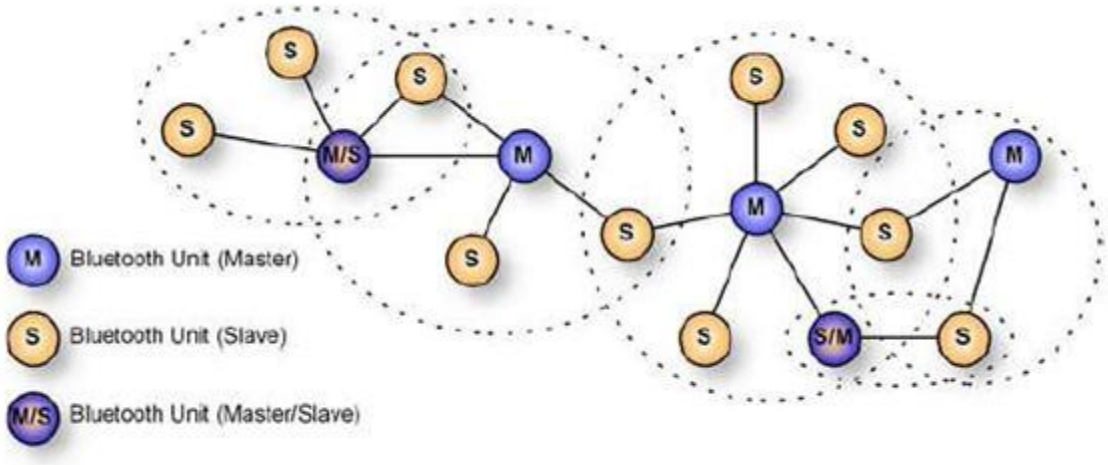


Figure 2.7 Bluetooth piconet showing different possible combinations of master/slave connections

A typical piconet is shown in figure 2.7. It is formed with up to 8 devices. One of the devices is the piconet master, while up to 7 others can connect as piconet slaves. Although there is only one master device in each piconet, a device can be a master as well as a slave if it is in multiple piconets. The master schedules packet transmission to every slave node in slots of 625us. After each transmission, the master device waits another 625 us for packet reception. Therefore, the maximum theoretical speed at which a Bluetooth device can send information is 1.25 ms [30]. The HC-05 Bluetooth module was chosen as the wireless protocol for our system because it provided an acceptable transfer speed as well as low power consumption.

Figure 2.8 HC-05 Bluetooth module (BT) with a CC2541 chip

2.5 Teensy Programming

The Teensy was programmed to read both the accelerometer's data and composes it in a form that will be read by our software. It will do this by scaling the values to a range of consistent length while adding a terminator character (!) to the end of each packet. It will then output the values over a Bluetooth serial port at 100 Hz. A flow chart of the logic is shown in figure 2.9.

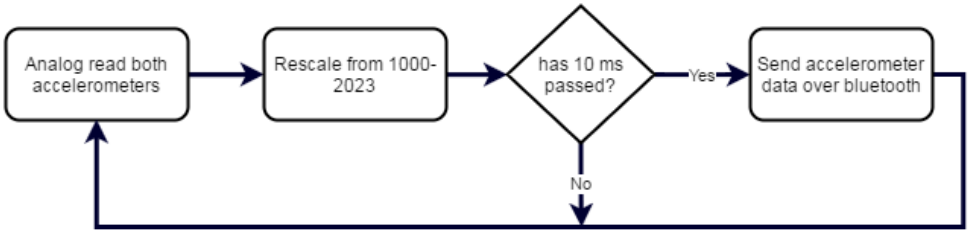


Figure 2.9 Flowchart of teensy programming

2.6 Custom software

SIMULINK has known issues that prevent it reading directly from a virtual serial port created from a Bluetooth connection. In order to circumvent this issue, we needed to create a workaround. First we used an open source TCP/IP based redirection program to forward all the serial data to a TCP/IP address that MATLAB/SIMULINK could then read. The program faced problems with the parsing in that it would not align the data

correctly. It also was very crash prone and would require a reboot of the whole model. This would be impossible for data collection, especially in the OR.

We then set to create a custom C++ solution from scratch that would solve the issues that the previous solutions left. The program was created in Visual Basic 14 (Microsoft, Inc.). We based it on the Windows Multimedia Timer for high resolution (<5 ms) timing. The program creates a timer that runs every 1 ms to check for new data in the serial port (real or virtual). It then reads the data until it comes to a custom terminator character (!). From there, it checks to see if the packet is the correct size; if it is, the UDP port is open and the transmission begins.

Because of the nature of Bluetooth transmission not being as stable as direct USB, many of the packets were being delayed inconsistently. This was causing errors in dropped and delayed packets. To compensate for this, a 10 packet buffer array was introduced to the program which fills up to 5 packets before beginning the transmission. This puts the inherent delay of the system at 50 ms, which is adequate for our purposes. The first in first out (FIFO) buffer will continuously output a packet every 10 ms. A flow chart of the logic of the buffer is shown in figure 2.10.

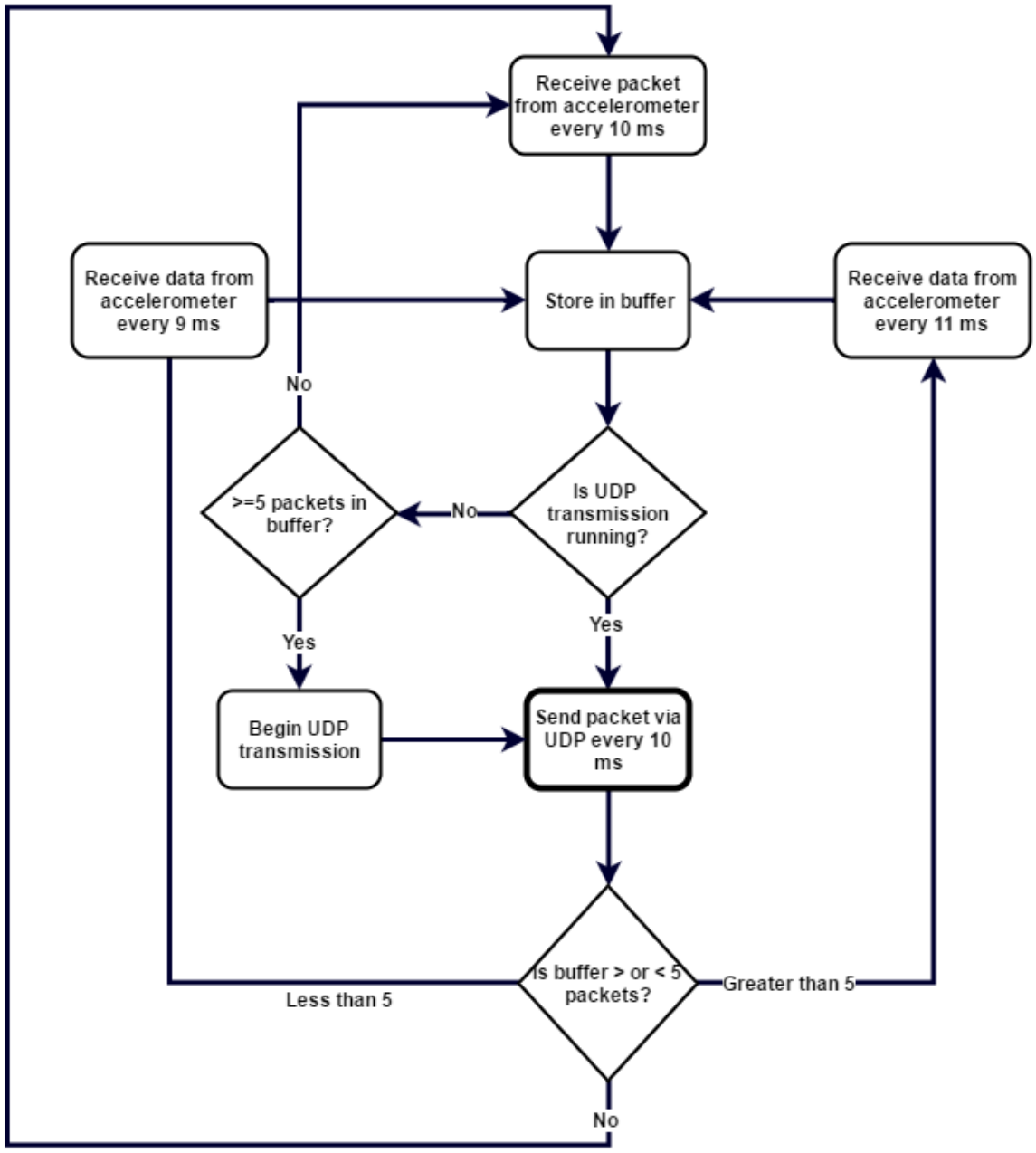


Figure 2.10 Flow chart of custom C++ program to parse and transmit data via USB.

In the figures below, the utility of the addition of the buffer is shown. In figure 2.11, the same signal is sent via BT-UDP. The signal is more discrete and not as smooth. This is due to the inconsistent jitter from the BT communication. Figure 2.12 shows

another sine wave sent through BT-UDP, except this time the FIFO buffer is implemented.

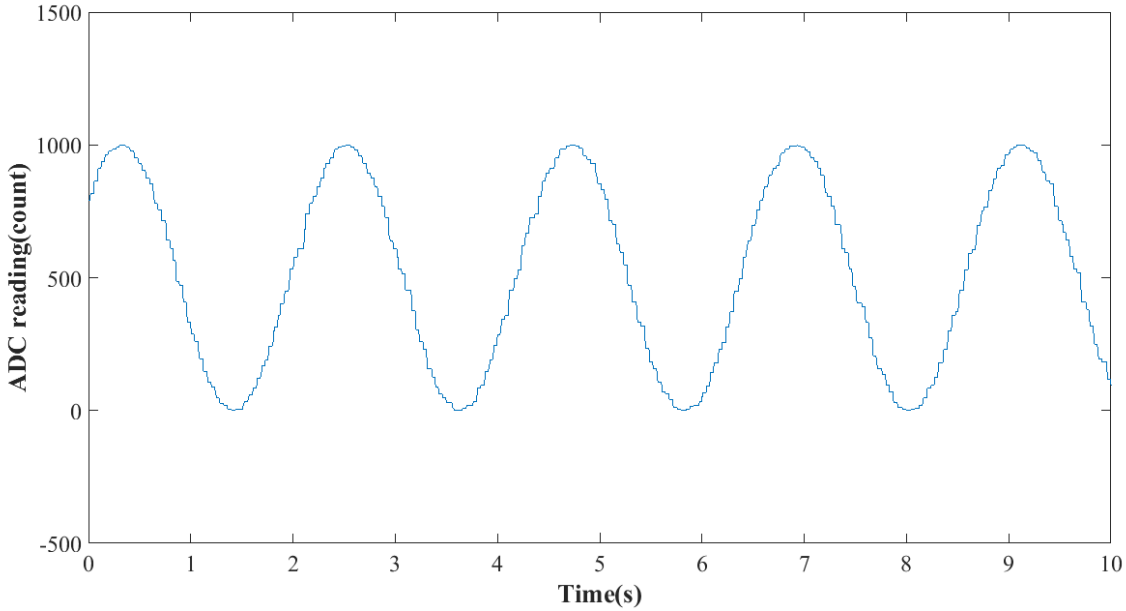


Figure 2.11 BT-UDP connection before buffer is added

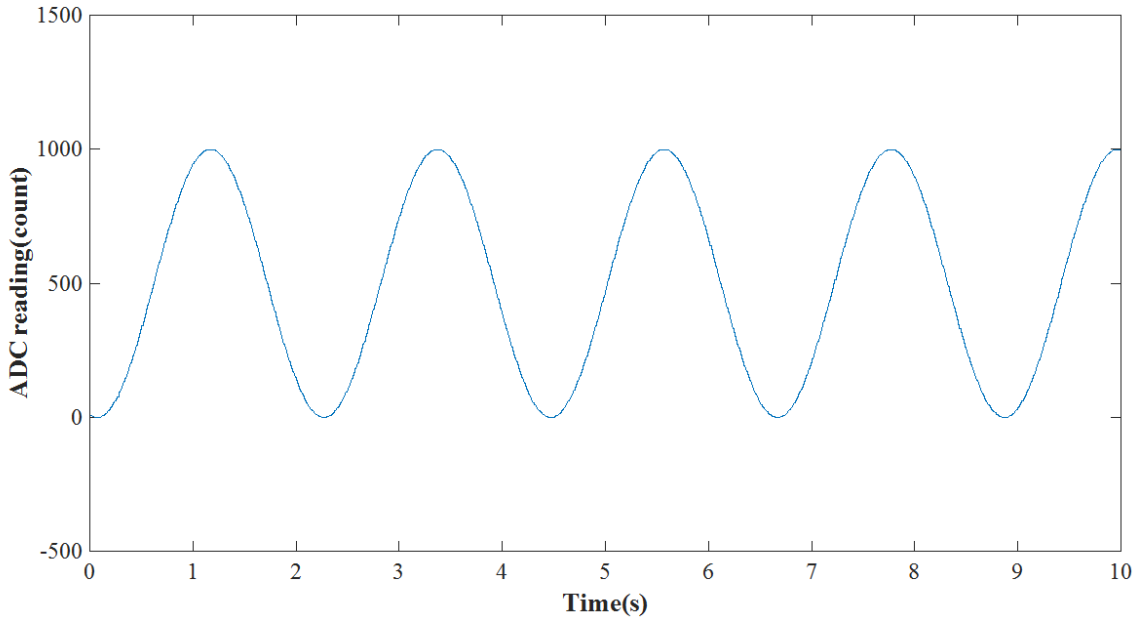


Figure 2.12 BT-UDP connection after buffer is added

With the buffer, the BT-UDP signal is comparable to the USB-UDP signal. The dynamic sized buffer allows for a consistent flow of data to Simulink, which improves

the signal fidelity while adding a small fixed latency to the data. The buffer will vary in size but allow for consistent flow of data.

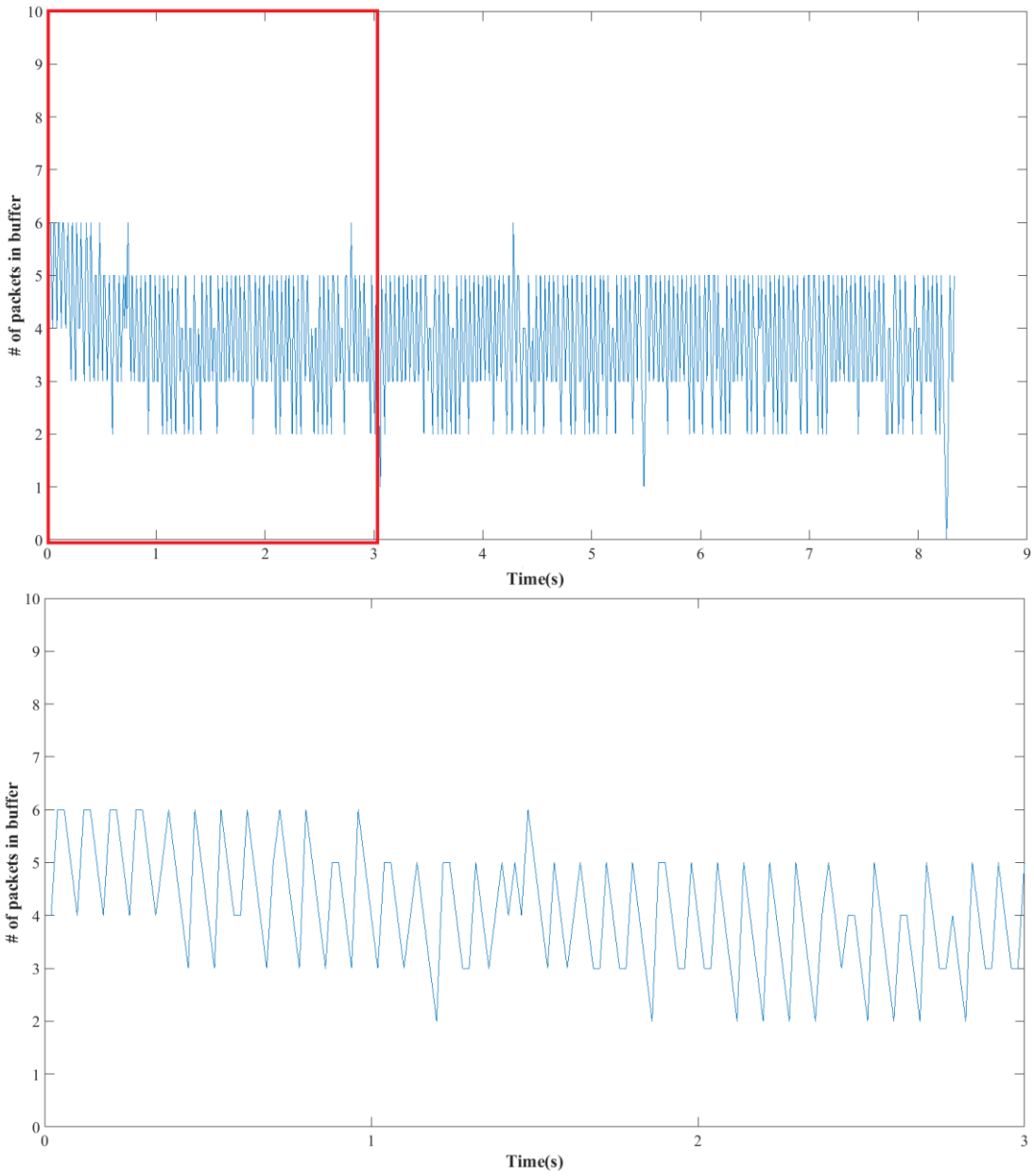


Figure 2.13 Buffer level during data transmission

The buffer level is usually at its highest point at the beginning of the UDP transmission. This is because it stores 5 packets before it begins. After this point, the buffer will empty slightly before stabling out. It remains steady at approximately 2-5

packets in the buffer, while rarely going up to 6 or down to 1. The buffer level's variance can be seen in figure 2.13. As long as it never gets to 0, the UDP port will always have data available, thus compensating for the jitter of the BT communication.

2.7 Latency

The wireless nature of the system could present issues with timing that we needed to evaluate. We needed to calculate the latency of the whole system to determine if the fidelity of our wireless solution was acceptable for the signals we needed. To calculate this we sent a 5 V 10 Hz pulse wave for 500 seconds from a function generator to both the microcontroller as well as the parallel port of a windows computer (optimized for real time computation) as shown in figure 2.14. The Teensy is connected via USB or BT while parallel port is connected directly to the bus. We then recorded and compared the timing of the two signals on MATLAB/SIMULINK as shown in figure 2.15. We used a parallel port for its extremely low latency. It has been measured at less than 1 ms in the literature[31].

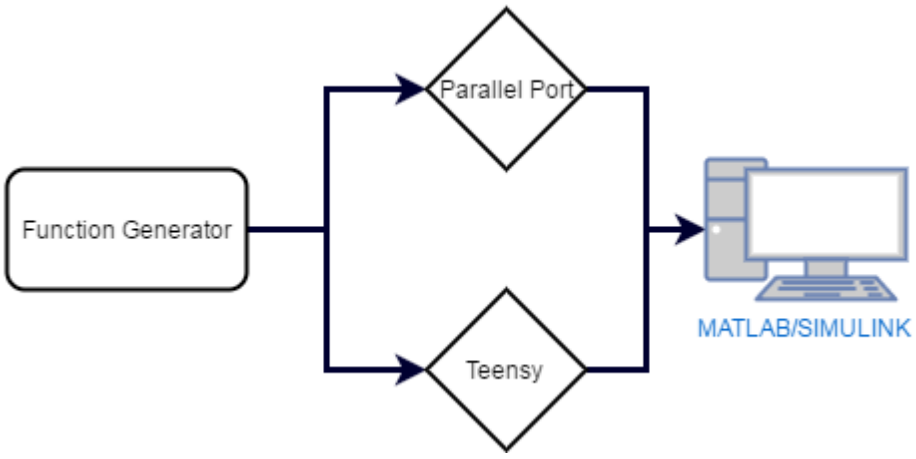
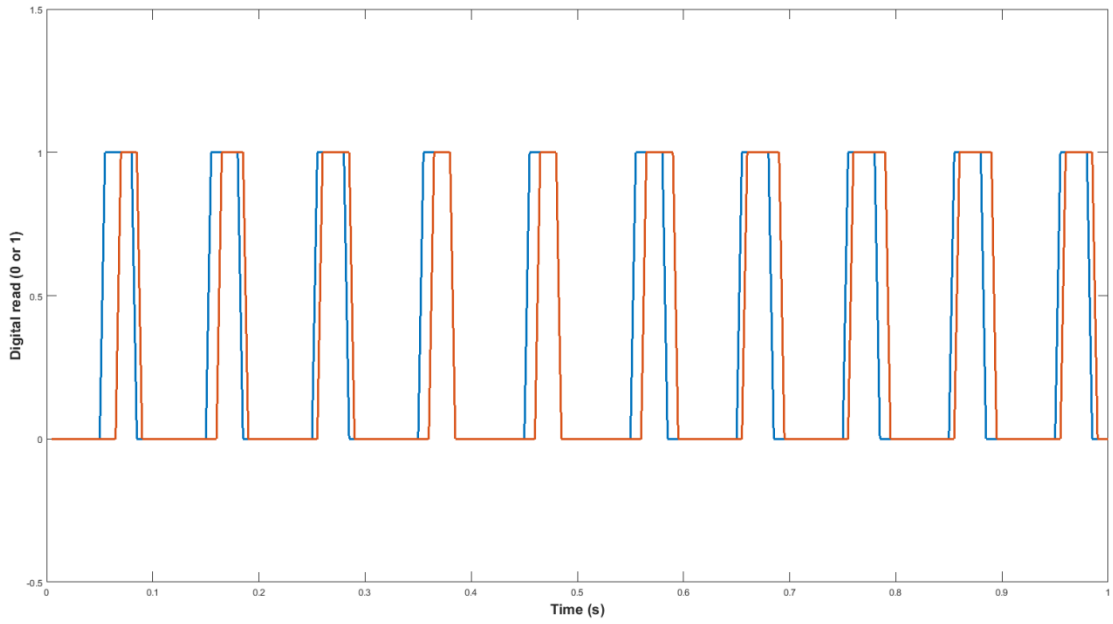
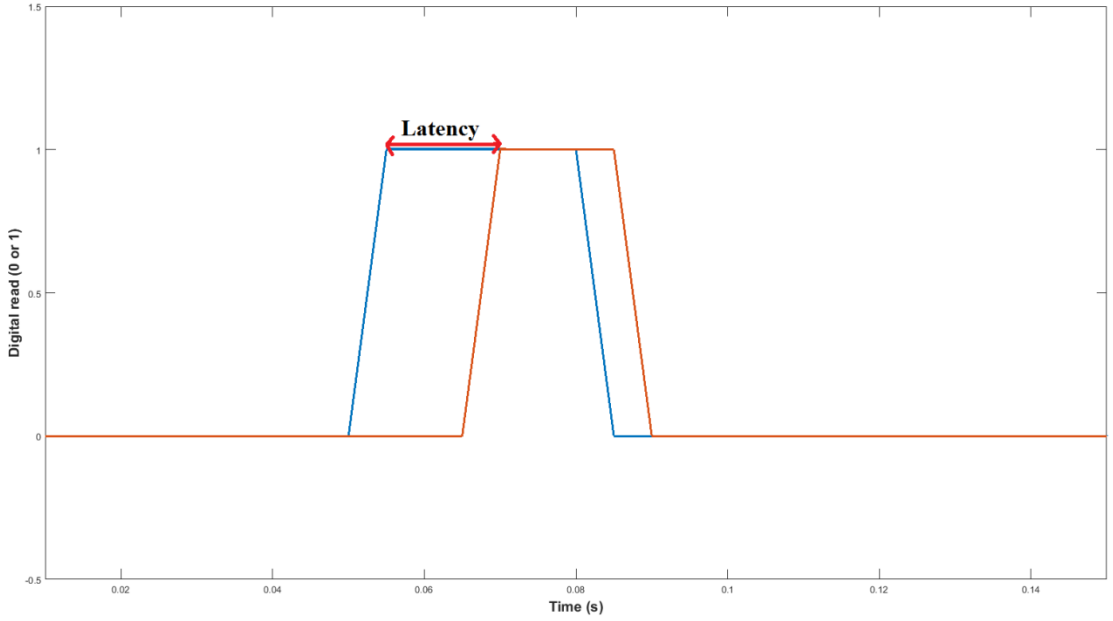


Figure 2.14 Flowchart of latency test. The Function generator emitted a pulse signal simultaneously to both systems.


Digital read (0 or 1)
Time (s)


Latency
Digital read (0 or 1)
Time (s)

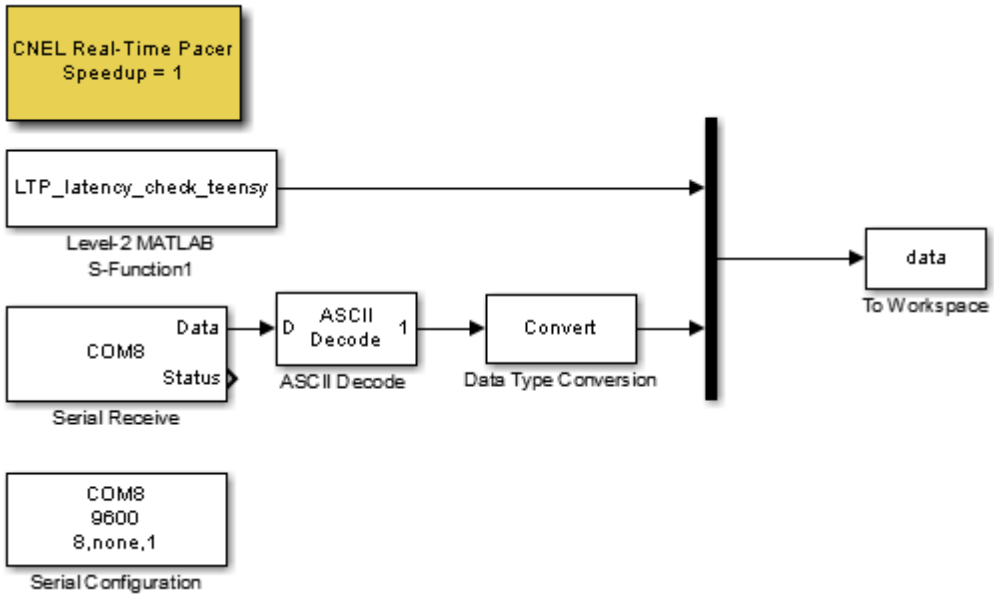


Figure 2.15 Simulink model for testing latency of the system. When testing USB-UDP and BT-UDP, the 'Serial Receive' and 'Serial Configuration' blocks were replaced by a 'UDP Receive' block.

After running each test, the peaks of the signals were subtracted from each other to give us the latency of the system. Three different protocol setups were used: direct USB, USB to UDP, and BT to UDP.

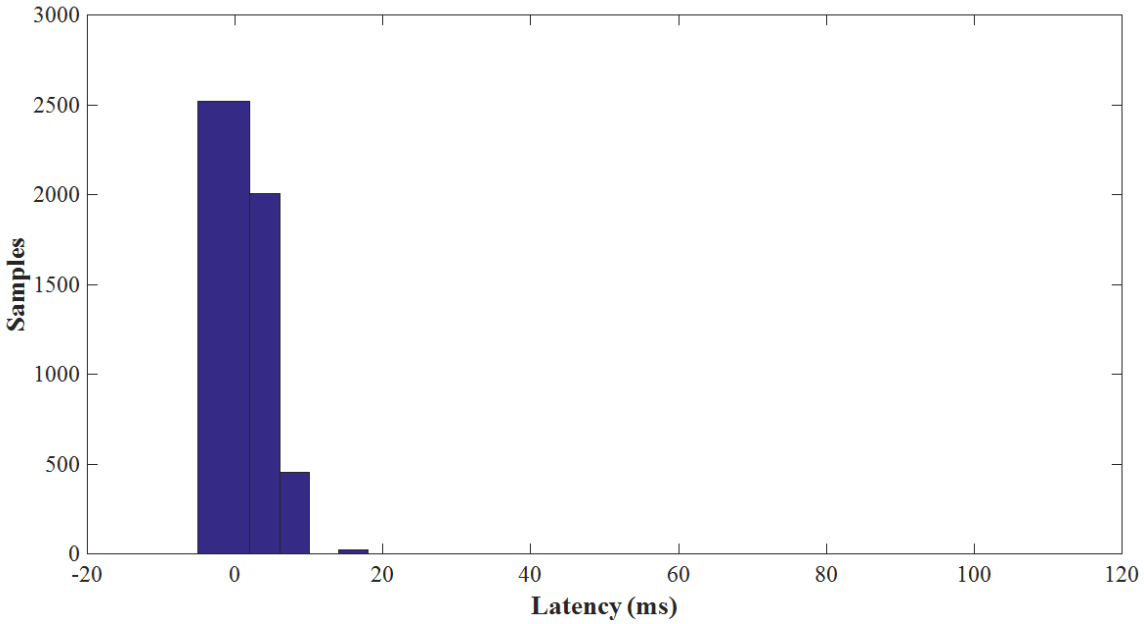


Figure 2.16 Histogram of latency between parallel port and USB. This hardwired solution was expected to be the most reliable and lowest latency.

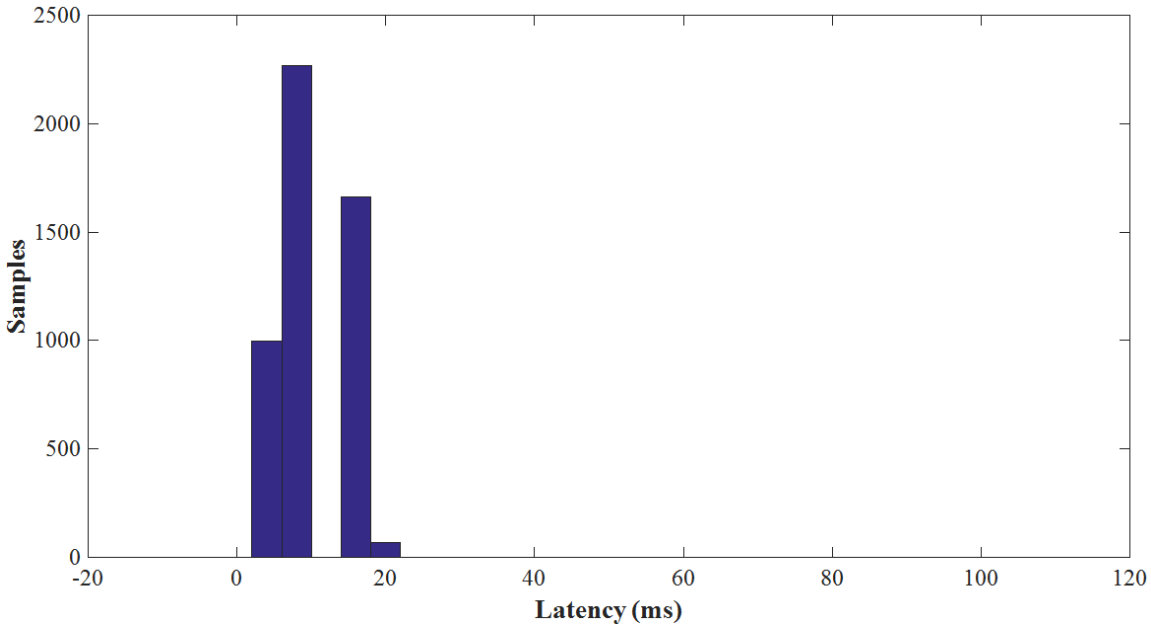


Figure 2.17 Histogram of latency between parallel port and USB-UDP.

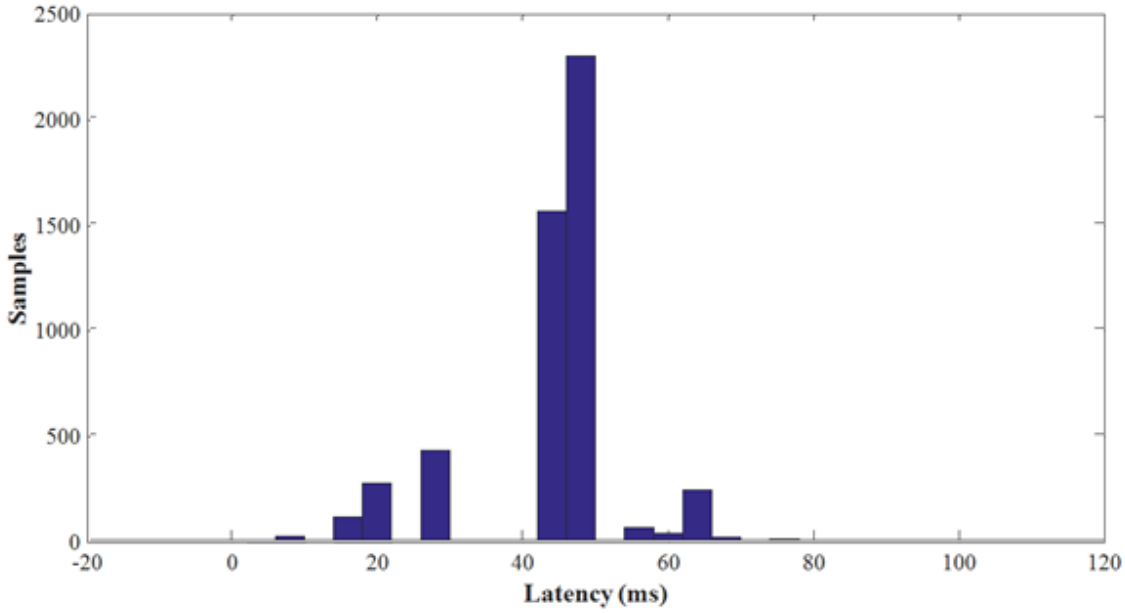


Figure 2.18 Histogram of latency between parallel port and BT-UDP. This setup was used in the actual system.

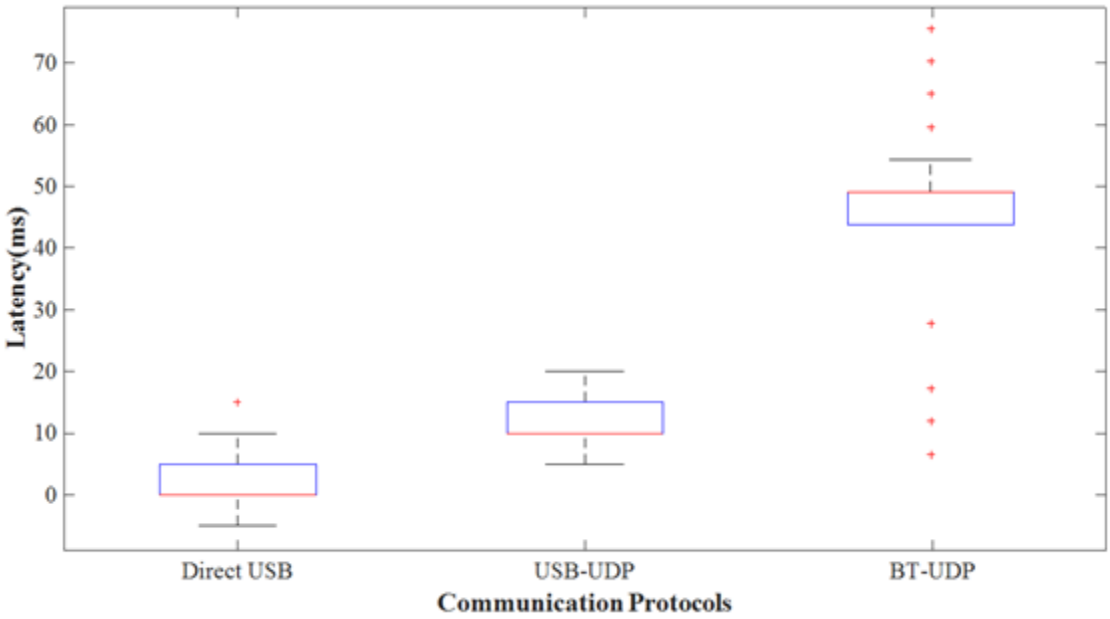


Figure 2.19 Box plot of latency between all protocols.

Table 2.2 Statistical features of latency (ms) test for all protocols. Note that the Direct USB had several latencies that were small enough to go undetected and reported as 0 ms.

Protocol	Mean	Standard Deviation	Median
Direct USB	2.0750	4.5203	0
USB-UDP	10.8050	3.7496	10
BT-UDP	45.1230	10.5878	40

3 Operating Room Data

3.1 Overview

The system was placed on the tremor dominant side of 3 PD patients during DBS surgery. The two accelerometers were placed on the subjects hand and foot. The subjects' activity was measured from when the microelectrode implantation began to when it reached 5 mm past the target site.

Figure 3.1Intraoperative recording from tremor dominant side of patient during DBS surgery. The arrow is pointing to the location of an accelerometer

The patients' tremor was visualized and recorded along with neural data. In figure 3.2, the accelerometer data wave form is shown.

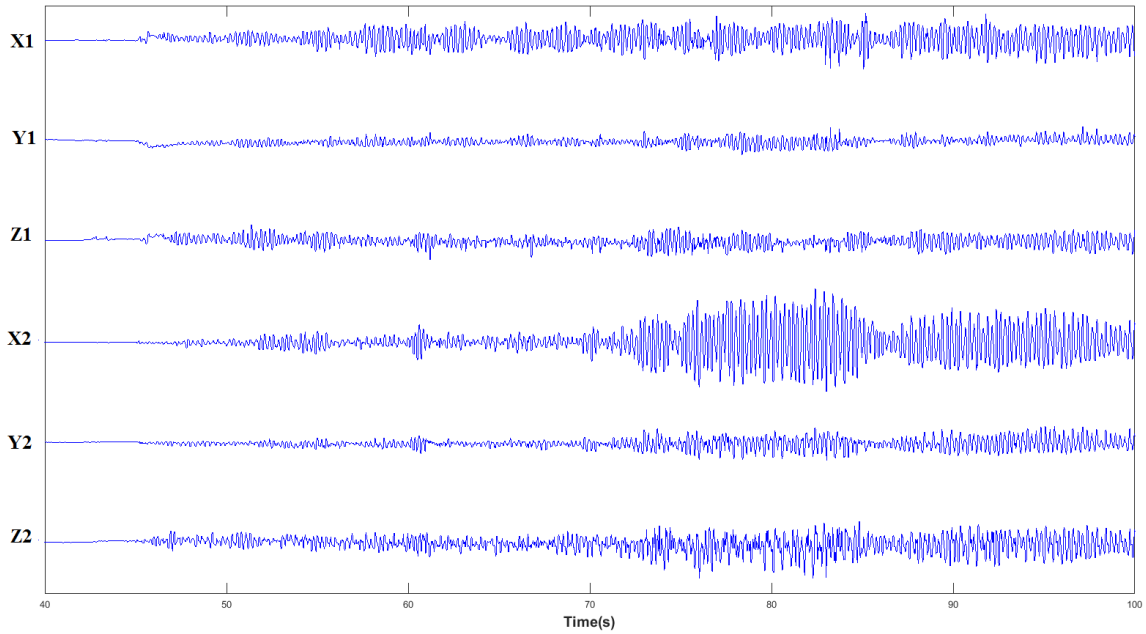


Figure 3.2 Accelerometer wave forms captured during DBS surgery

4 Ambulatory Data

4.1 Experimental Design

We collected data from 4 patients with PD treated with deep brain stimulation (DBS) in the Parkinson's Disease Center and Movement Disorders Clinic at Baylor College of Medicine. The two accelerometers were attached to either of the patient's hands. Patients were then asked to perform 3 tasks to examine tremor: resting quietly to assess rest tremor (rest), forward posture holding to assess forward postural tremor (fwd) and lateral posture holding to assess lateral postural tremor (lat). The tasks are shown in figure 4.1.

Figure 4.1 Patient during 3 different tasks of the procedure.

The tasks were performed under 3 DBS conditions: with the DBS on at their usual setting, with the DBS at a lower (intermediate) setting, and with the DBS off. Each patient took approximately 10 minutes to go through the whole exam.

We recorded all accelerometer data along with video using a custom model we developed in MATLAB/Simulink (The MathWorks Inc, Natick, Massachusetts). This model runs in soft real-time and receives the accelerometer data from the UDP port and displays the signal waveform and the time-frequency plot. The patients were recorded during these tests using a Logitech c270 webcam that was synchronized with the rest of

the data by capturing the frame number along with the accelerometer data [32] The recording set up we used is shown in figure 4.2.

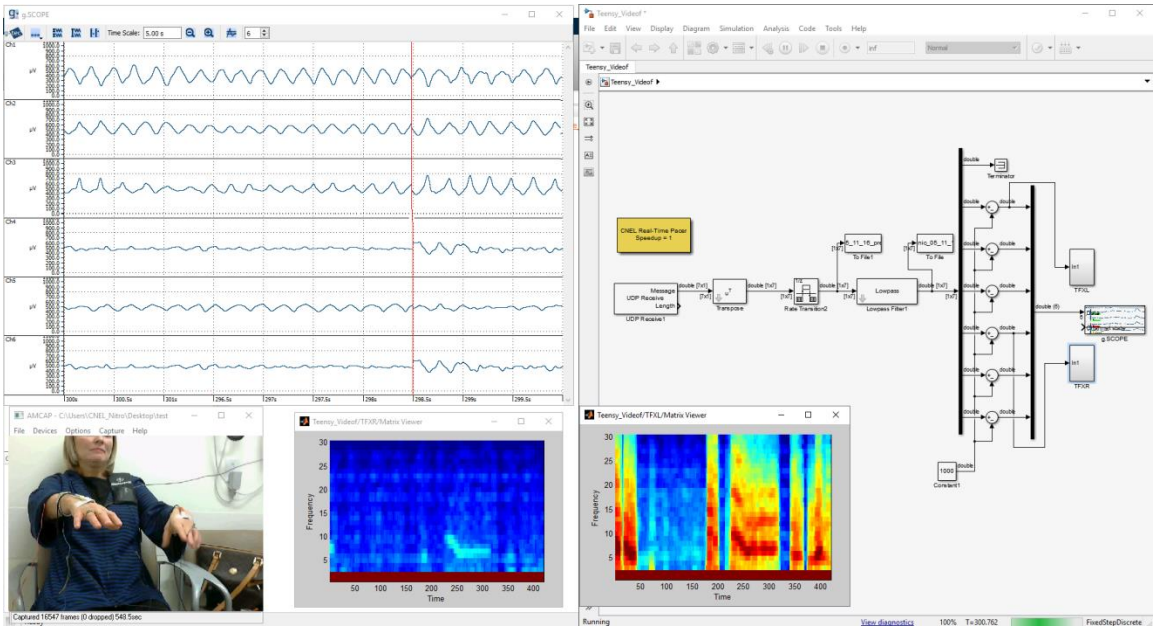


Figure 4.2 Screenshot of raw data and time-frequency map from patient trial.

4.2 Results and Discussion

The system recorded and visualized the changes in acceleration with respect to time. As is the case with accelerometers, they are prone to drift after time. To compensate for that, the data was detrended by subtracting the mean value of the data set from each value in the data set. After the drift was compensated for, the scale of the dataset was reshaped from 0 to 1023 (10 bit analog value) to -3 to 3(range of accelerometer). The data from before and after this scaling and detrending can be seen in figure 4.3.

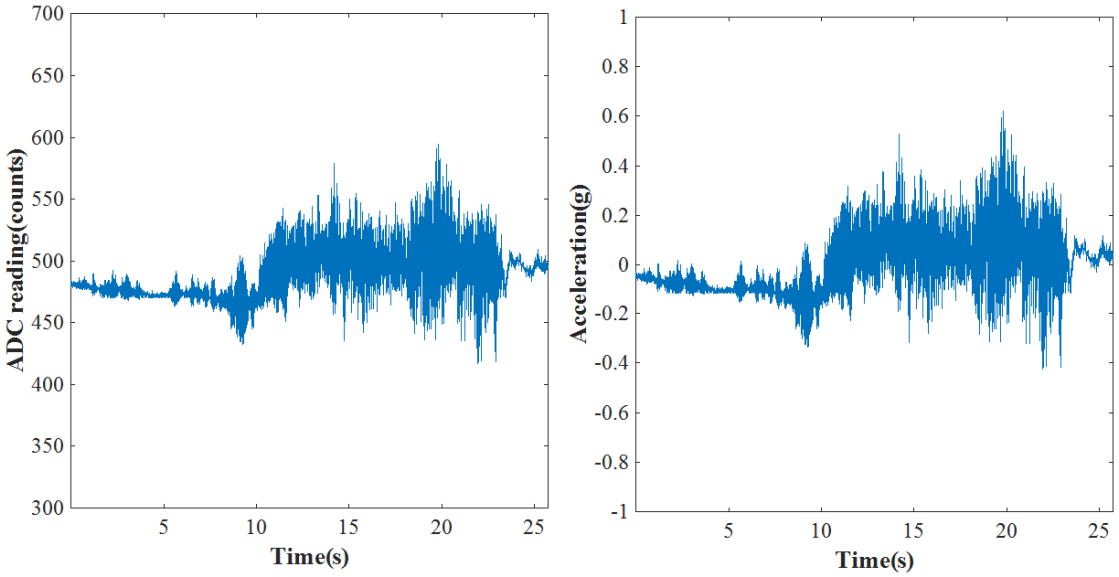


Figure 4.3 A tremor is shown before and after scaling to g's and compensating for drift by offsetting the data.

Time-frequency maps were used during the test for visualization as well as analysis. In figure 4.4, patient 3's fwd tremor recordings are shown. We can see the frequency of the tremor falls into the range we would expect in Parkinsonian tremor (3-6 Hz).

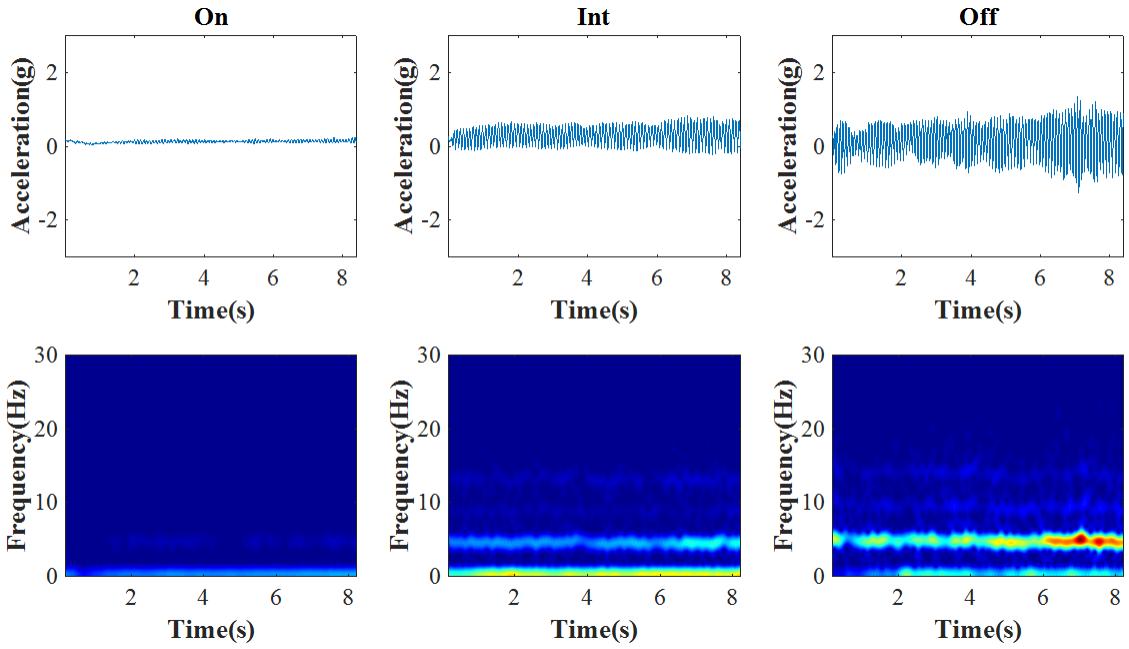


Figure 4.4 Comparison of patient 3 forward postural tremor between ON, INT, and OFF states. The time-frequency map illustrates the increasingly stronger presence of a tremor around the 5 Hz frequency band.

4.3 Analysis

The patients' data was segmented into 3 different parts (one for each task). The power spectrum density (PSD) estimates of each of these segments under the 3 DBS conditions (ON, INTERMEDIATE, OFF) were then calculated using a 1 second Hanning window with 50% overlap. An example of the PSD difference between DBS OFF and INT are

shown below in figures 4.5 and 4.6.

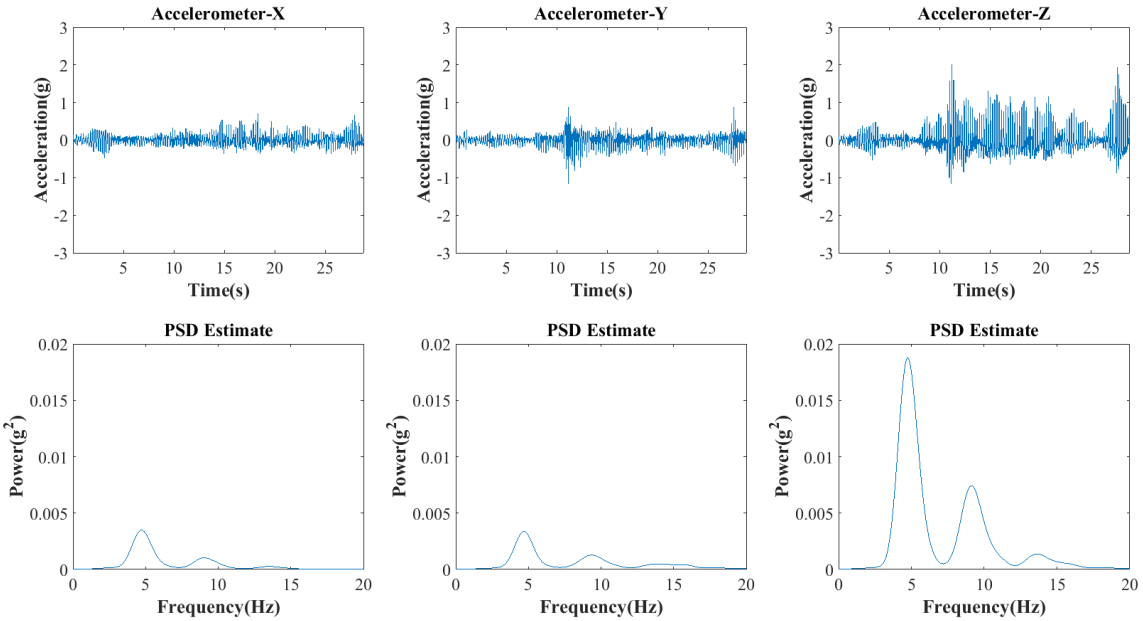


Figure 4.5 Patient 2 rest tremor while DBS is OFF. The peaks in the 3-6 Hz range are very prominent.

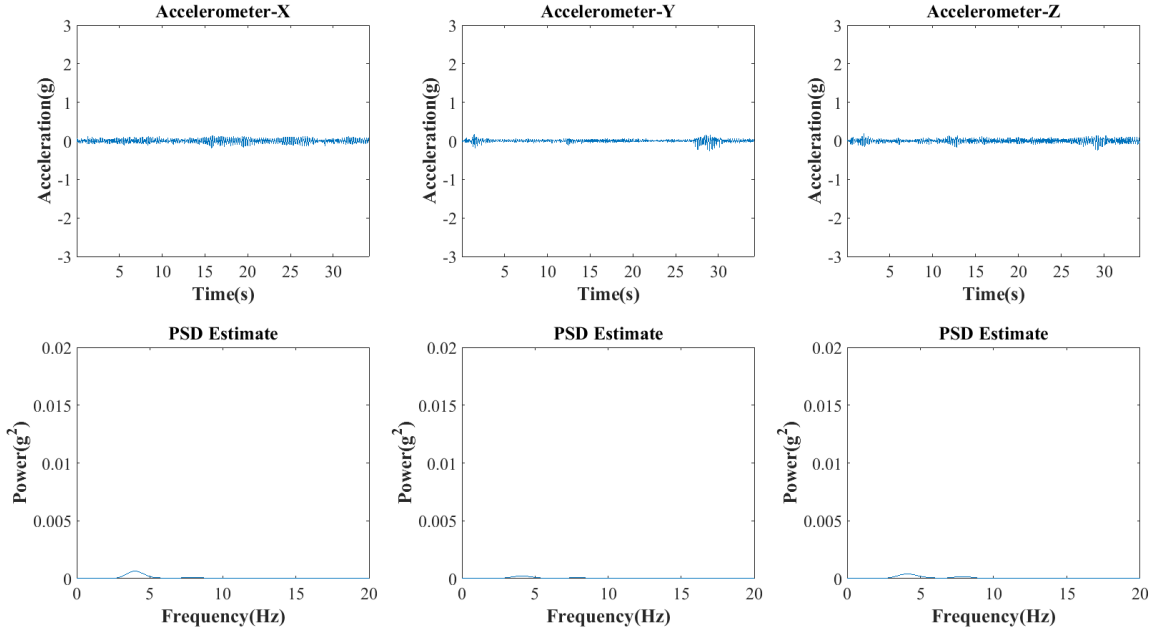


Figure 4.6 Patient 2 rest tremor while DBS is turned to an INT setting. The peaks in the 3-6 Hz range are much smaller and that is reflected in the less severe tremor in the time domain.

To capture the power of the tremor, the root mean square was calculated from between 3-6 Hz frequency. In typical parkinsonian tremor, it is expected that there is a

significant peak in this range, thus causing a large proportion of the power of the spectrum to also exist in this range. An example is shown in figure 4.5 and 4.6 which show a peak for all 3 axes in between 3-6 Hz while the DBS is OFF. This is drastically reduced when the DBS is set to an INT level. While there is still a peak in the same frequency range, the reduced tremor severity is reflected in the smaller peak.

Figures 4.7-4.10 show the total energy for each DBS state. For all but patient 4, there is a downward trend in power when going from DBS OFF to ON.

Table 4.1 Total energy (g) in 3-6 Hz band

Pat	Off			Intermediate			On		
	Rest	**Fwd**	**Lat**	**Rest**	**Fwd**	**Lat**	**Rest**	**Fwd**	**Lat**
1	0.27700	0.39260	0.22260	0.00131	0.06296	0.00002	0.00024	0.00037	0.00001
2	0.40814	1.96352	6.92297	0.02080	0.34970	2.59953	0.00798	0.00840	0.01928
3	7.60251	28.71398	5.10005	0.00358	0.10039	0.24431	0.00506	0.06531	0.09567
4	0.00038	0.11853	1.73476	0.00051	0.01230	0.21634	0.00049	0.00718	0.88312

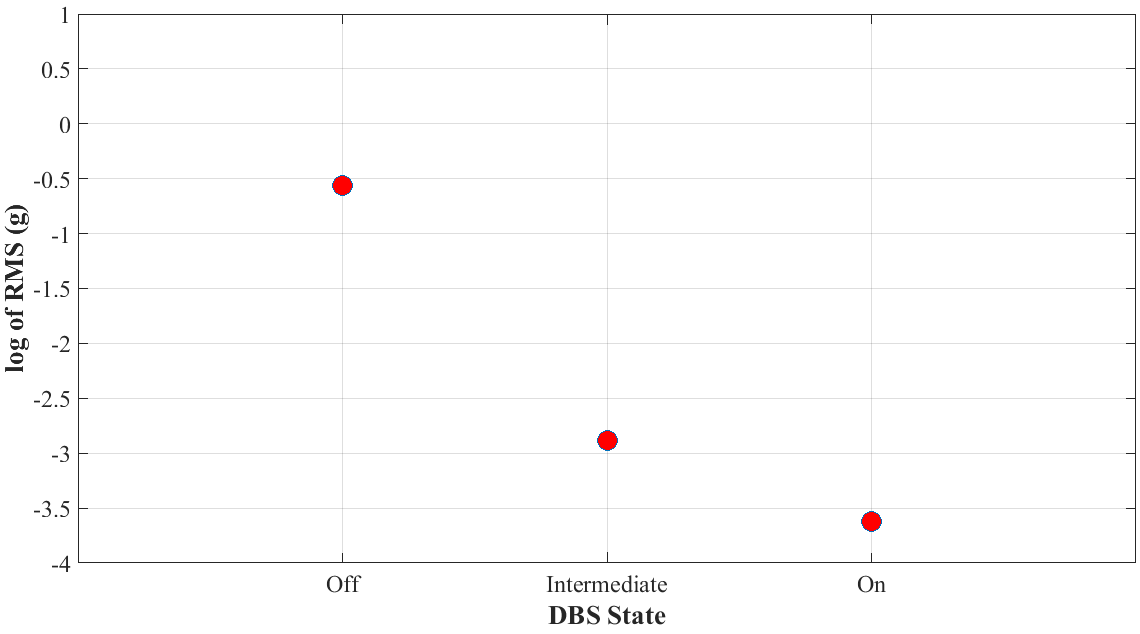


Figure 4.7 Log of total energy in 3-6 Hz band for subject 1 during rest task

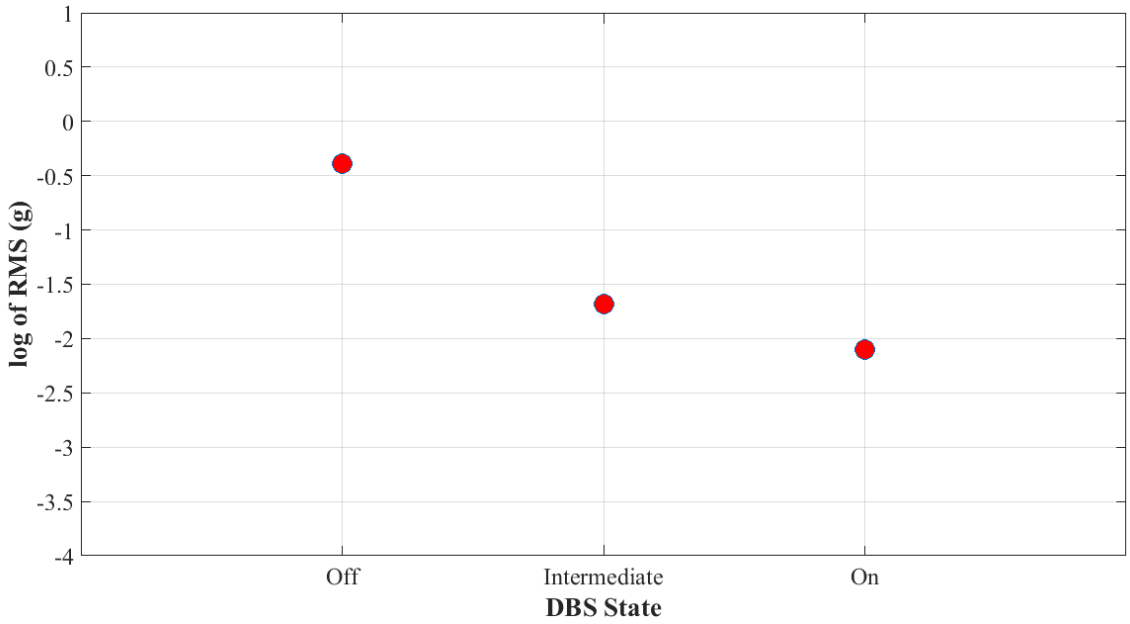


Figure 4.8 Log of total energy in 3-6 Hz band for subject 2 during rest task

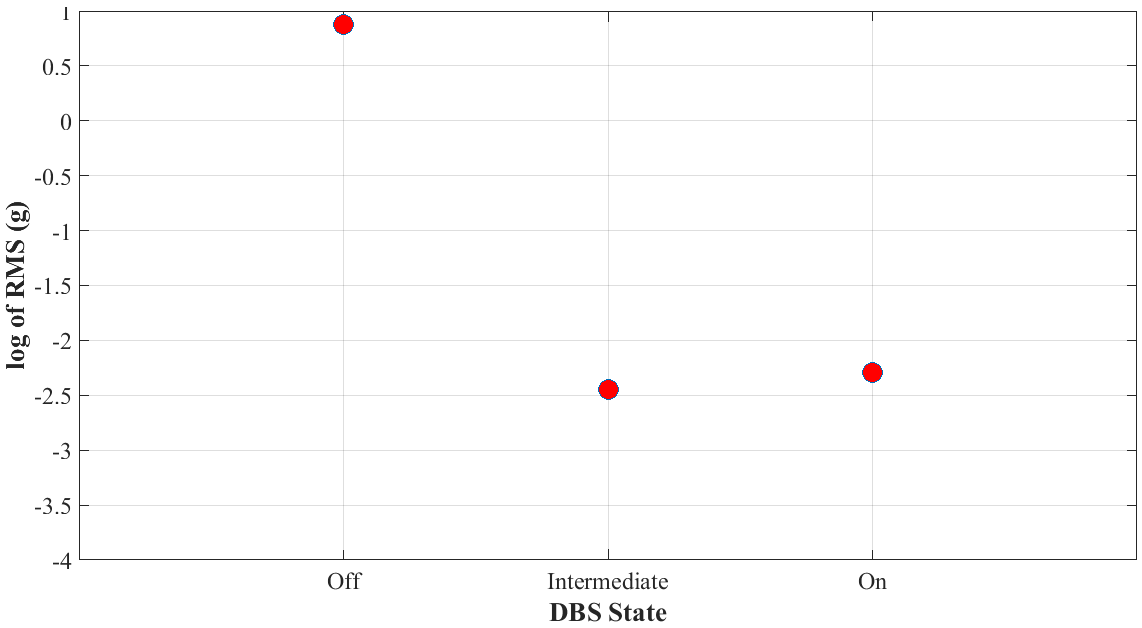


Figure 4.9 Log of total energy in 3-6 Hz band for subject 3 during rest task

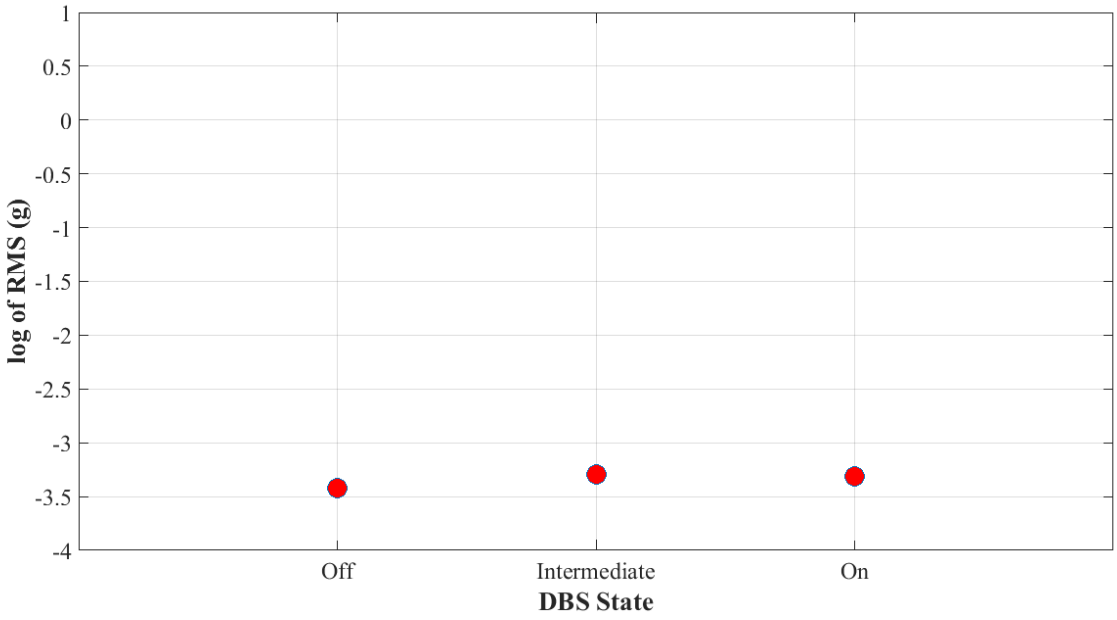


Figure 4.10 Log of total energy in 3-6 Hz band for subject 4 during rest task

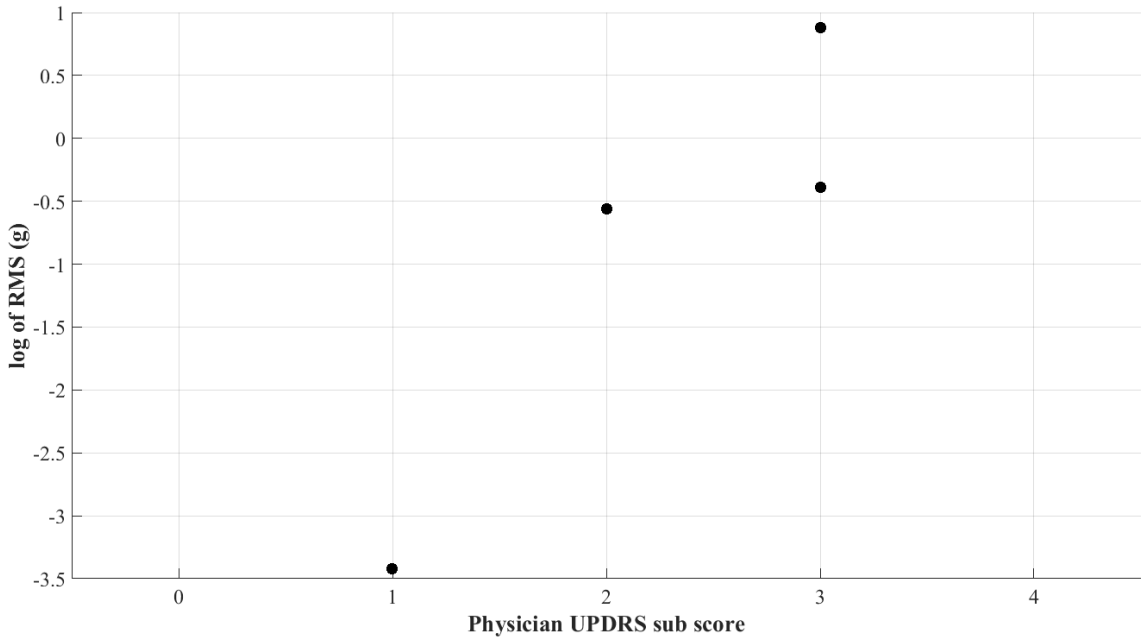


Figure 4.11UPDRS sub score vs log of total energy for all patients with DBS OFF and during rest task

In figure 4.11, the log of total energy during the rest task is compared to physicians UPDRS score during the rest task. The correlation was calculated between the total energy and the UPDRS score for the rest task.

Table 4.2 Pearson's correlation coefficient and Spearman correlation for total energy in 3-6 Hz band and UPDRS sub score during rest task

Characteristics	**Pearson Correlation**		**Spearman Correlation**	
	R	**p**	**ρ**	**p**
Total Energy 3-6 Hz Rest	0.4345	0.1581	0.7379	0.3333

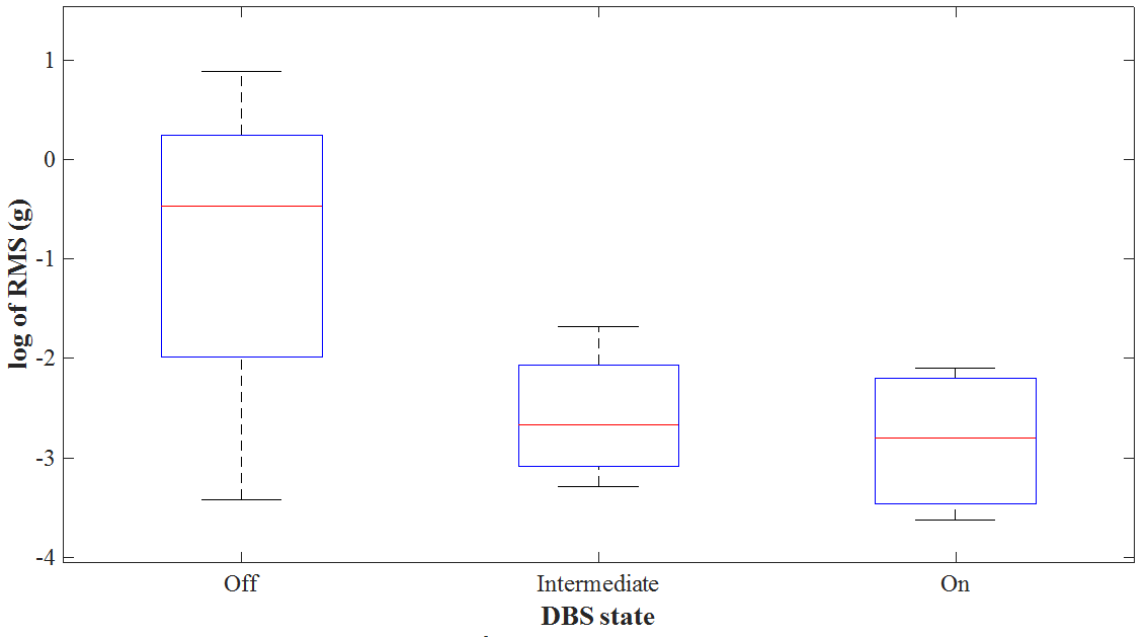


Figure 4.12 Log of total energy for all patients and all DBS states during rest

Table 4.3 Results of Pearson correlation and Spearman Rank Correlation between total energy between 3-6 Hz band of the accelerometer signal and the UPDRS sub scores for all tasks.

Characteristics	**Pearson Correlation**		**Spearman Correlation**	
	R	**p**	**ρ**	**p**
Total energy 3-6 Hz	0.569	0.0003	0.6998	4.61E-06

5 Discussion

5.1 Conclusion

This study shows a system that will allow objective, accurate information on the existence and severity of tremor as a tool to supplement clinical observations. It can objectively measure both frequency and severity (amplitude) of tremor, based on PSD analysis. The system was able to record accelerometer data that confirmed that the total energy of all 3 axes in the 3-6 Hz band can be used as a characteristic for UPDRS correlation. This wireless sensor interface will allow physicians to identify meaningful differences in tremor severity that might be difficult to stratify on an ordinal scale such as the UPDRS. Furthermore, the system can easily be added to the clinical assessment procedure.

5.2 Future Work

There are several possible optimizations that can be made to the system. Some of the changes are simple while others would need significant work or a complete redesign of the system.

Future work should include further miniaturization and improvement of battery life of the hardware. Although the Teensy was a good microcontroller for the scope of this study, a bare AVR chip would have a much smaller footprint. Not only would the size of the microcontroller be smaller, the Bluetooth module can be built into the AVR chip. Along with the small size, the less power consumption would allow for us to use a significantly smaller battery. This would help in the comfort level for the patient as the battery was the largest part of the system. This miniaturization and battery life improvement can allow for long term continuous recording. The dynamic temporal tremor data from long term recording can be analyzed and processed by computers and physicians to offer highly personal treatment plans as well as offer insight on how motor symptoms change over time in regards to DBS stimulation ON and OFF states.

In order to more accurately measure displacement from accelerometer readings, a second accelerometer can be affixed to the same limb. Although this would increase the number of sensors to 4, the added benefit of measuring displacement would be beneficial as it is a direct parameter to look for in UPDRS test. With a single accelerometer on each limb, the double integration leaves too much room for error for an accurate interpretation of displacement. Further research into more complex algorithms can be done to expand the scope of this device in order to detect more difficult to detect conditions such as bradykinesia, dystonia, and freezing of gait.

Although it was not done in this study, it would be interesting to use this system to quantify the time from turning the stimulator off to when the tremor emerges. While most patients' tremor begins shortly after the stimulation is turned off, a look into the statistical significance of between stimulation voltage, latency of tremor starting, and severity of tremor. Other studies can use this system to look at quantifying or describing the effect of frequency of stimulation on tremor reduction, the latency to onset or offset of tremor according to DBS condition, or intraoperative clinical testing to determine optimal electrode localization.

References

[1] E. Kandel, T. James, S. Sielgelbaum, and A. Hudspeth, *Principles of Neural Science*. McGraw-Hill Education, 2012.

[2] G. Deuschl, J. Raethjen, M. Lindemann, and P. Krack, "The pathophysiology of tremor.," *Muscle Nerve*, vol. 24, no. 6, pp. 716–35, Jun. 2001.

[3] O. J. A. Watts Ray L. , Standaert David G., *Movement Disorders*, 3rd ed. McGraw-Hill Education, 2011.

[4] J. Jankovic, "Parkinson's disease: clinical features and diagnosis," *J. Neurol. Neurosurg. Psychiatry*, vol. 79, no. 4, pp. 368–376, Apr. 2008.

[5] N. F. Ince, A. Gupte, T. Wichmann, J. Ashe, T. Henry, M. Bebler, L. Eberly, and A. Abosch, "Selection of Optimal Programming Contacts Based on Local Field Potential Recordings From Subthalamic Nucleus in Patients With Parkinson's Disease," *Neurosurgery*, vol. 67, no. 2, pp. 390–397, Aug. 2010.

[6] T. Hershey, J. Wu, P. M. Weaver, D. C. Perantie, M. Karimi, S. D. Tabbal, and J. S. Perlmutter, "Unilateral vs. bilateral STN DBS effects on working memory and motor function in Parkinson disease," *Exp. Neurol.*, vol. 210, no. 2, pp. 402–408, 2008.

[7] J. S. Perlmutter and J. W. Mink, "Deep Brain Stimulation," *Annu. Rev. Neurosci.*, vol. 29, no. 229, pp. 229–257, 2015.

[8] T. Mds, C. G. Goetz, W. Poewe, O. Rascol, and S. Christina, "State of the Art Review The Unified Parkinson ' s Disease Rating Scale (UPDRS): Status and Recommendations," *Society*, vol. 18, no. 7, pp. 738 –750, 2003.

[9] C. G. Goetz, B. C. Tilley, S. R. Shaftman, G. T. Stebbins, S. Fahn, P. Martinez-Martin, W. Poewe, C. Sampaio, M. B. Stern, R. Dodel, B. Dubois, R. Holloway, J. Jankovic, J. Kulisevsky, A. E. Lang, A. Lees, S. Leurgans, P. A. LeWitt, D. Nyenhuis, C. W. Olanow, O. Rascol, A. Schrag, J. A. Teresi, J. J. van Hilten, N. LaPelle, P. Agarwal, S. Athar, Y. Bordelan, H. M. Bronte-Stewart, R. Camicioli, K. Chou, W. Cole, A. Dalvi, H. Delgado, A. Diamond, J. P. Dick, J. Duda, R. J. Elble, C. Evans, V. G. Evidente, H. H. Fernandez, S. Fox, J. H. Friedman, R. D. Fross, D. Gallagher, C. G. Goetz, D. Hall, N. Hermanowicz, V. Hinson, S. Horn, H. Hurtig, U. J. Kang, G. Kleiner-Fisman, O. Klepitskaya, K. Kompoliti, E. C. Lai, M. L. Leehey, I. Leroi, K. E. Lyons, T. McClain, S. W. Metzer, J. Miyasaki, J. C. Morgan, M. Nance, J. Nemeth, R. Pahwa, S. A. Parashos, J. S. J. S. Schneider, A. Schrag, K. Sethi, L. M. Shulman, A. Siderowf, M. Silverdale, T. Simuni, M. Stacy, M. B. Stern, R. M. Stewart, K. Sullivan, D. M. Swope, P. M. Wadia, R. W. Walker, R. Walker, W. J. Weiner, J. Wiener, J. Wilkinson, J. M. Wojcieszek, S. Wolfrath, F. Wooten, A. Wu, T. A. Zesiewicz, and R. M. Zweig, "Movement Disorder Society-Sponsored Revision of the Unified Parkinson's Disease Rating Scale (MDS-UPDRS): Scale presentation and clinimetric testing results," *Mov. Disord.*, vol. 23, no. 15, pp. 2129–2170, 2008.

[10] A. Salarian, H. Russmann, C. Wider, P. R. Burkhard, F. J. G. Vingerhoets, and K. Aminian, "Quantification of tremor and bradykinesia in Parkinson's disease using a novel ambulatory monitoring system," *IEEE Trans. Biomed. Eng.*, vol. 54, no. 2, pp. 313–322, 2007.

[11] G. Deuschl, P. Bain, M. Brin, Y. Agid, L. Benabid, R. Benecke, A. Berardelli, D.

J. Brooks, R. Elble, S. Fahn, L. J. Findley, M. Hallett, J. Jankovic, W. C. Koller, P. Krack, A. E. Lang, A. Lees, C. H. Lucking, C. D. Marsden, J. A. Obeso, W. H. Oertel, W. Poewe, P. Pollak, N. Quinn, J. C. Rothwell, H. Shibasaki, P. Thompson, and E. Toloso, "Consensus Statement of the Movement Disorder Society on Tremor," *Mov. Disord.*, vol. 13, no. S3, pp. 2–23, 1998.

[12] E. Rocon, J. M. Belda-Lois, A. F. Ruiz, M. Manto, J. C. Moreno, and J. L. Pons, "Design and validation of a rehabilitation robotic exoskeleton for tremor assessment and suppression," *IEEE Trans. Neural Syst. Rehabil. Eng.*, vol. 15, no. 1, pp. 367–378, 2007.

[13] P. E. O'Suilleabhain and J. Y. Matsumoto, "Time-frequency analysis of tremors," *Brain*, vol. 121, no. 11, pp. 2127–2134, 1998.

[14] R. N. Stiles and J. E. Randall, "Mechanical factors in human tremor frequency.," *J. Appl. Physiol.*, vol. 23, no. 3, pp. 324–330, Sep. 1967.

[15] C. W. Hess and S. L. Pullman, "Tremor: clinical phenomenology and assessment techniques.," *Tremor Other Hyperkinet. Mov. (N. Y).*, vol. 2, pp. 1–15, 2012.

[16] E. Y. Hanada, C. L. Hubley-Kozey, M. D. McKeon, and S. A. Gordon, "The feasibility of measuring the activation of the trunk muscles in healthy older adults during trunk stability exercises.," *BMC Geriatr.*, vol. 8, p. 33, 2008.

[17] S. L. Pullman, D. S. Goodin, A. I. Marquinez, S. Tabbal, and M. Rubin, "Clinical utility of surface EMG: report of the therapeutics and technology assessment subcommittee of the American Academy of Neurology.," *Neurology*, vol. 55, no. 2, pp. 171–7, Jul. 2000.

[18] S. L. Pullman, “Spiral analysis: a new technique for measuring tremor with a digitizing tablet.,” *Mov. Disord.*, vol. 13 Suppl 3, pp. 85–89, 1998.

[19] R. LeMoyne, “Wearable and wireless accelerometer systems for monitoring Parkinson’s disease patients—A perspective review,” *Adv. Park. Dis.*, vol. 02, no. 04, pp. 113–115, 2013.

[20] J. B. SAUNDERS, V. T. INMAN, and H. D. EBERHART, “The major determinants in normal and pathological gait.,” *J. Bone Joint Surg. Am.*, vol. 35-A, no. 3, pp. 543–58, Jul. 1953.

[21] R. LeMoyne, C. Coroian, T. Mastroianni, P. Opalinski, M. Cozza, and W. Grundfest, “The Merits of Artificial Proprioception, with Applications in Biofeedback Gait Rehabilitation Concepts and Movement Disorder Characterization,” in *Biomedical Engineering*, InTech, 2009.

[22] K. M. Culhane, M. O’Connor, D. Lyons, and G. M. Lyons, “Accelerometers in rehabilitation medicine for older adults,” *Age Ageing*, vol. 34, no. 6, pp. 556–560, 2005.

[23] R. LeMoyne, T. Mastroianni, M. Cozza, C. Coroian, and W. Grundfest, “Implementation of an iPhone for characterizing Parkinson’s disease tremor through a wireless accelerometer application,” *2010 Annu. Int. Conf. IEEE Eng. Med. Biol. Soc. EMBC’10*, pp. 4954–4958, 2010.

[24] J. P. Giuffrida, D. E. Riley, B. N. Maddux, and D. a. Heldmann, “Clinically deployable kinesia technology for automated tremor assessment,” *Mov. Disord.*, vol. 24, no. 5, pp. 723–730, 2009.

[25] the free encyclopedia Wikipedia, “List of Arduino boards and compatible systems,” *Wikimedia Foundation*. [Online]. Available: https://en.wikipedia.org/wiki/List_of_Arduino_boards_and_compatible_systems.

[26] T. W. Schubert, A. D’Ausilio, and R. Canto, “Using Arduino microcontroller boards to measure response latencies.,” *Behav. Res. Methods*, vol. 45, no. 4, pp. 1332–46, 2013.

[27] A. D’Ausilio, “Arduino: A low-cost multipurpose lab equipment,” *Behav. Res. Methods*, vol. 44, no. 2, pp. 305–313, 2012.

[28] Analog Device, “Small, Low Power, 3-Axis Accelerometer (ADXL335),” *Analog Device, Inc*, pp. 1 – 14, 2009.

[29] T. K. Sethuramalingam and A. Vimalajuliet, “Design of MEMS based capacitive accelerometer,” *ICMET 2010 - 2010 Int. Conf. Mech. Electr. Technol. Proc.*, no. Icmet, pp. 565–568, 2010.

[30] M. C. Valenti, M. Robert, and J. H. Reed, “On the throughput of Bluetooth data transmissions,” *2002 IEEE Wirel. Commun. Netw. Conf. Rec. WCNC 2002 (Cat. No.02TH8609)*, vol. 1, no. C, pp. 119–123, 2002.

[31] A. De Clercq, G. Crombez, A. Buysse, and H. Roeyers, “A simple and sensitive method to measure timing accuracy.,” *Behav. Res. Methods, Instruments, Comput.*, vol. 35, no. 1, pp. 109–115, 2003.

[32] H. A. Siddiqui, J. Jimenez-Shahed, A. Viswanathan, and N. F. Ince, “A Wireless Sensor Interface for the Quantification of Tremor Using Off the Shelf Components,” *2016 32nd South. Biomed. Eng. Conf.*, pp. 177–178, 2016.

Appendix A

Teensy Code

```
const int xpin1 = A9;
const int ypin1 = A8;
const int zpin1 = A7;

const int xpin2 = A6;
const int ypin2 = A5;
const int zpin2 = A4;

int x_1 = 0;
int y_1 = 0;
int z_1 = 0;

int x_2 = 0;
int y_2 = 0;
int z_2 = 0;

long interval=10;
long previousMs=0;

void setup()  {
 // analogReference(EXTERNAL);
 // analogReadResolution(12);
  //analogReadAveraging(32);
  Serial1.begin(115200);
}

void loop() {
  unsigned long currentMs=millis();

  x_1 = analogRead(xpin1);
  y_1 = analogRead(ypin1);
  z_1 = analogRead(zpin1);

  x_2 = analogRead(xpin2);
  y_2 = analogRead(ypin2);
  z_2 = analogRead(zpin2);

  x_1=map(x_1,0,1023,1000,2023);
  y_1=map(y_1,0,1023,1000,2023);
  z_1=map(z_1,0,1023,1000,2023);

  x_2=map(x_2,0,1023,1000,2023);
  y_2=map(y_2,0,1023,1000,2023);
  z_2=map(z_2,0,1023,1000,2023);

  if(currentMs-previousMs>interval) {
    previousMs=currentMs;

    Serial1.print(x_1);
    Serial1.print(',');
    Serial1.print(y_1);
```

```
    Serial1.print(',');
    Serial1.print(z_1);
    Serial1.print(',');
    Serial1.print(x_2);
    Serial1.print(',');
    Serial1.print(y_2);
    Serial1.print(',');
    Serial1.print(z_2);
    Serial1.print("!");
  }
}
```

Appendix B

Serial-UDP software code

```
#define _CRT_SECURE_NO_WARNINGS
#define _WINSOCK_DEPRECATED_NO_WARNINGS
#define PORTTYPE HANDLE
#define BAUD 115200

#include <winsock2.h>
#include <Ws2tcpip.h>
#include <stdio.h>
#include <windows.h>
#include <Windows.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdarg.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <errno.h>
#include <iostream>
#include <conio.h>
#include <vector>

#pragma comment(lib, "winmm.lib")
#pragma comment(lib, "Ws2_32.lib")

PORTTYPE open_port_and_set_baud_or_die(const char *name, long baud);
void close_port(PORTTYPE port, const char* cport);
void die(const char *format, ...);
bool setClipBoardString(std::string source);

UINT period;
UINT dur;
const char *COMport;
unsigned short port;
char COM[80];

UINT idx;
UINT dmp = 0;
```

```
double *data = NULL;
UINT counter = 0;
LARGE_INTEGER start, current, fs;

COMMTIMEOUTS timeout;
DWORD n = 0;
BOOL r;
char Recieve_Buffer[1024];
int iResult;
WSADATA wsaData;
SOCKET SendSocket = INVALID_SOCKET;
sockaddr_in RecvAddr;

DWORD dwCommEvent;
char  chRead;
PORTTYPE fd;
char ch[1];
COMSTAT comStat;
DWORD   dwErrors;
BOOL    fOOP, fOVERRUN, fPTO, fRXOVER, fRXPARITY, fTXFULL;
BOOL    fBREAK, fDNS, fFRAME, fIOE, fMODE;
char ParseBuf[1024];
int k = 0;

int ix = 0, tm = 0;
bool UDP_ready = FALSE;
double time_diff = 0;
int trigger_value;
int packet_size;
int packet = 0;
std::vector<char> sendbuffer(n, 0);
int packets_to_skip = 5;
int j=0;
float timer_buf;
float timer_buf_original;
float offset;
std::vector<char> buffer(n, 0);

void CALLBACK SynchSender(UINT nIDEvent, UINT msg, DWORD_PTR dwUser,
DWORD_PTR dw1, DWORD_PTR dw2)
{
    FILE *fp = *(FILE **)dwUser;

    //-----------------------------------------------------------------
-
    // Calculates time difference between sent packets
    QueryPerformanceCounter(&current);
    DOUBLE time_difference = ((double)(current.QuadPart -
start.QuadPart)) / fs.QuadPart;
    time_difference *= 1000;

    //-----------------------------------------------------------------
-
    // Get and clear current errors on the port.
    ClearCommError(fd, &dwErrors, &comStat);
```

```
    //-------------------------------------------------------------------
-
    // Reads and parses serial data. Then puts it into a buffer. Once
the buffer
    // fills up with the trigger value amount of packets, it enables
UDP tranmission
    // mode.
    while (comStat.cbInQue >= 1) {  // comStat.cbInQue bytes have been
received, but not read
        r = ReadFile(fd,
            ch,
            1,
            &n,
            NULL);

        ClearCommError(fd, &dwErrors, &comStat); // Get and clear
current errors on the port

        Recieve_Buffer[k] = ch[0];
        k++;

        if (ch[0] == '!' && k == packet_size) {
            j++;
            if (j > packets_to_skip){
                k = 0;
                if (ix <= buffer.size()){
                    for (int i = 0; i < packet_size; ++i){
                        buffer[ix] = Recieve_Buffer[i];
                        ix++;
                    }
                    packet++;
                }
                else{
                    printf("buffer overrun\n");
                }

                if (UDP_ready == FALSE && packet == trigger_value) {
                    printf("entering UDP mode...\n\n");
                    UDP_ready = TRUE;
                }
            }
            else {
                k = 0;
                printf("packet skipped\n");
            }
        }
        else if (ch[0] == '!' && k != packet_size) {
            printf("\nmisaligned packet\n");
            k = 0;
        }
    }

    //-------------------------------------------------------------------
-
    // Sends a packet of data to UDP and realigns buffer index
    if (UDP_ready && packet > 0 && time_difference >= timer_buf) {
        for (int i = 0; i < packet_size; ++i){
```

```
            sendbuffer[i] = buffer[i];
        }
        --packet;

        //-------------------------------------------------------------
-----
        // Adjusts the timer to send data to UDP by a given offset to
keep
        // the buffer level steady.
        if (packet > trigger_value + 5){
            timer_buf = timer_buf_original - offset;
        }
        else if (packet < trigger_value - 5){
            timer_buf = timer_buf_original + offset;
        }
        else{
            timer_buf = timer_buf_original;
        }

        start = current;//resetting time

        const char* SEND_BUFFER = &sendbuffer[0];

        iResult = sendto(SendSocket,
                         SEND_BUFFER,
                         packet_size,
                         0,
                         (SOCKADDR *)& RecvAddr,
                         sizeof(RecvAddr));

        if (iResult == SOCKET_ERROR) {
            wprintf(L"sendto failed with error: %d\n",
WSAGetLastError());
            closesocket(SendSocket);
            WSACleanup();
            exit(0);
        }

        ix = ix - packet_size;

        printf("Time Difference: %.4lg\tBuffer Level: %4d\r",
time_difference, ix);
        fprintf(fp, "%d\t%lg\t%lg\n",ix,time_difference,timer_buf);

        for (int i = 0; i < buffer.size() - packet_size; i++){
            buffer[i] = buffer[i + packet_size];
        }
    }

    //-------------------------------------------------------------------
-
    // Closes application after specified duration
    if (dur < idx) {
        idx++;
        close_port(fd, COMport);
        exit(0);
    }
```

```
}


int main(int argc, char **argv)
{
    system("cls");
    printf("\n\n\t\t--------------------------------------\n");
    printf("\t\tSerial to UDP Command Line Application\n");
    printf("\t\t--------------------------------------\n\n");

    if (argc == 9)
    {
        COMport = argv[1];
        packet_size = atoi(argv[2]);
        port = atoi(argv[3]);
        period = atoi(argv[4]);
        dur = atoi(argv[5]) * 1000;
        timer_buf_original = std::atof(argv[6]);
        offset = std::atof(argv[7]);
        trigger_value = atoi(argv[8]);

    }   else if (argc != 9){
        std::cout << "COM port(\"COM#\"):";
        std::cin.getline(COM, sizeof COM);
        COMport = COM;
        std::cout << "Bytes to read:";
        std::cin >> packet_size;
        packet_size = (int)packet_size;
        std::cout << "Port:";
        std::cin >> port;
        std::cout << "Period(ms):";
        std::cin >> period;
        std::cout << "Duration(s):";
        std::cin >> dur;
        dur = dur * 1000;
        std::cout << "Timer(ms):";
        std::cin >> timer_buf_original;
        std::cout << "Timer Offset(ms):";
        std::cin >> offset;
        std::cout << "Trigger Value:";
        std::cin >> trigger_value;

        char usage[100];
        sprintf(usage, "%s %d %d %d %d %g %.g %d", COMport,
packet_size, port, period, dur / 1000, timer_buf_original, offset,
trigger_value);
        setClipBoardString(usage);

        printf("\n\t\tParameters copied to clipboard\n\n");
            system("pause");
    }

    system("cls");
    printf("\n\n\t\t--------------------------------------\n");
    printf("\t\tSerial to UDP Command Line Application\n");
    printf("\t\t--------------------------------------\n\n");
```

```
    printf("COM Port:%s\nPacket
Size:%d\nPort:%d\nPeriod:%d\nDuration:%d\nTimer:%g\nTimer
Offset:%g\nTrigger Value:%d\n", COMport, packet_size, port, period,
dur/1000, timer_buf_original, offset, trigger_value);

    timer_buf = timer_buf_original;
    dur = ((double)dur) / period;
    data = new double[dur];

    TIMECAPS    tc;
    MMRESULT    m_nTimerId;
    UINT        wTimerRes;
    FILE        *fp;

    fp = fopen("Teensy_Buffer.txt", "w+t");

    //--------------------------------------------------------------------
-
    //resize buffers to double the size of the value at trigger point
    int n = packet_size * trigger_value * 2;
    buffer.resize(n);
    sendbuffer.resize(packet_size);

    //--------------------------------------------------------------------
-
    // Open COM port
    fd = open_port_and_set_baud_or_die(COMport, 115200);
    printf("\nport %s opened\n\n", COMport);
    printf("\t\tSending data from %s to 127.0.0.1:%d\n\n", COMport,
port);

    GetCommTimeouts(fd, &timeout);
    timeout.ReadIntervalTimeout = MAXDWORD; // non-blocking
    timeout.ReadTotalTimeoutMultiplier = 0;
    timeout.ReadTotalTimeoutConstant = 0;
    SetCommTimeouts(fd, &timeout);

    //--------------------------------------------------------------------
-
    // Initialize Winsock
    iResult = WSAStartup(MAKEWORD(2, 2), &wsaData);
    if (iResult != NO_ERROR) {
        wprintf(L"WSAStartup failed with error: %d\n", iResult);
        return 1;
    }

    //--------------------------------------------------------------------
-
    // Create a socket for sending data
    SendSocket = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);
    if (SendSocket == INVALID_SOCKET) {
        wprintf(L"socket failed with error: %ld\n", WSAGetLastError());
        WSACleanup();
        return 1;
    }
```

```
    //-------------------------------------------------------------------
-
    // Set up the RecvAddr structure with the IP address of
    // the receive and the specified port number.
    RecvAddr.sin_family = AF_INET;
    RecvAddr.sin_port = htons(port);
    RecvAddr.sin_addr.s_addr = inet_addr("127.0.0.1");

    //-------------------------------------------------------------------
-
    // Queries the timer device to determine its resolution.
    if (timeGetDevCaps(&tc, sizeof(TIMECAPS)) != TIMERR_NOERROR)
        exit(0);

    wTimerRes = min(max(tc.wPeriodMin, 1), tc.wPeriodMax);
    QueryPerformanceFrequency(&fs);
    timeBeginPeriod(wTimerRes);                                    //
requests a minimum resolution for periodic timers


    //-------------------------------------------------------------------
-
    // Starts timer that runs callback function for specified duration
    // with a specified period
    m_nTimerId = timeSetEvent((UINT)(period * 1),
//delay
                              wTimerRes,
//resolution
                              SynchSender,
//pointer to callback function
                              (DWORD_PTR)&fp,
//user supplied callback data
                              TIME_PERIODIC);
//periodic or oneshot

    QueryPerformanceCounter(&start);
    _getch();
    printf("\n\n");

    if (m_nTimerId){
        timeKillEvent(m_nTimerId);
        timeEndPeriod(wTimerRes);
        m_nTimerId = 0;
    }

    if (dmp == 0)
    {
        fclose(fp);
        fp = NULL;
    }
    return 0;
}

PORTTYPE open_port_and_set_baud_or_die(const char *name, long  baud)
{
    PORTTYPE fd;
    COMMCONFIG cfg;
```

```
    COMMTIMEOUTS timeout;
    DWORD n = 0;
    char portname[256];
    int num;

    if (sscanf_s(name, "COM%d", &num) == 1) {
        sprintf_s(portname, "\\\\.\\COM%d", num); // Microsoft KB115831
    }
    else {
        strncpy_s(portname, name, sizeof(portname) - 1);
        portname[n - 1] = 0;
    }

    fd = CreateFile(portname,
        GENERIC_READ | GENERIC_WRITE,
        0,
        0,
        OPEN_EXISTING,
        NULL,
        0);

    if (fd == INVALID_HANDLE_VALUE) die("unable to open port %s\n",
name);

    GetCommConfig(fd, &cfg, &n);
    cfg.dcb.BaudRate = 115200;
    cfg.dcb.fBinary = TRUE;
    cfg.dcb.fParity = FALSE;
    cfg.dcb.fOutxCtsFlow = FALSE;
    cfg.dcb.fOutxDsrFlow = FALSE;
    cfg.dcb.fOutX = FALSE;
    cfg.dcb.fInX = FALSE;
    cfg.dcb.fErrorChar = FALSE;
    cfg.dcb.fNull = FALSE;
    cfg.dcb.fRtsControl = RTS_CONTROL_ENABLE;
    cfg.dcb.fAbortOnError = FALSE;
    cfg.dcb.ByteSize = 8;
    cfg.dcb.Parity = NOPARITY;
    cfg.dcb.StopBits = ONESTOPBIT;
    cfg.dcb.fDtrControl = DTR_CONTROL_ENABLE;
    SetCommConfig(fd, &cfg, n);
    GetCommTimeouts(fd, &timeout);
    timeout.ReadIntervalTimeout = 0;
    timeout.ReadTotalTimeoutMultiplier = 0;
    timeout.ReadTotalTimeoutConstant = 1;
    timeout.WriteTotalTimeoutConstant = 0;
    timeout.WriteTotalTimeoutMultiplier = 0;
    SetCommTimeouts(fd, &timeout);
    return fd;
}

void close_port(PORTTYPE port, const char *cport)
{
    CloseHandle(port);
    printf("port %s closed\n", cport);
}
```

```
void die(const char *format, ...)
{
    va_list args;
    va_start(args, format);
    vfprintf(stderr, format, args);
    exit(1);
}

bool setClipBoardString(std::string source){
    if (OpenClipboard(NULL))
    {
        HGLOBAL clipbuffer;
        char * Clip_Buffer;
        EmptyClipboard();
        clipbuffer = GlobalAlloc(GMEM_DDESHARE, source.size() + 1);
        Clip_Buffer = (char*)GlobalLock(clipbuffer);
        strcpy(Clip_Buffer, LPCSTR(source.c_str()));
        GlobalUnlock(clipbuffer);
        SetClipboardData(CF_TEXT, clipbuffer);
        CloseClipboard();
        return true;
    }
    return false;
}
```