SLEEPING DISORDER DETECTION VIA TREMOR SENSING

A Design Project Report
Presented to the School of Electrical and Computer Engineering of Cornell University in Partial Fulfillment of the Requirements for the Degree of Master of Engineering, Electrical and Computer Engineering

Submitted by
Rishi Sharan
MEng Field Advisor: Dr. Bruce Land
Degree Date: May 2016

Abstract

Master of Engineering Program

School of Electrical and Computer Engineering

Cornell University

Design Project Report

Project Title: Sleeping Disorder Detection via Tremor Sensing

Author: Rishi Sharan

Abstract: The aim of this project is to obtain information about a user's sleep cycle and any present problems therein by use of a non-invasive tremor sensor. In short, I wish to take a signal produced by a tremor, process it to a form that I can extract meaningful data from (frequency, excursion length, etc.), and assign a label to the tremor. The label assigned to the tremor would ideally be the disorder that caused it.

I am looking into characterizing tremors based on the cepstrum of the time domain signal obtained from two accelerometers attached to the hands. [4] outlines a variety of features to look for in the signal. I looked at papers that outline the properties of tremors and how different disorders produce different types of tremors.

The final product is a glove with an Arduino chip, accelerometer, and Bluetooth transceiver sewn onto it that will transmit the position at a rate of 30 Hz to a computer running Matlab that will parse and analyze the data. Challenges include preventing aliasing, identifying short impulses, and identifying tremors themselves from background noise.

Contents

Executive Summary

The aim of this project is to obtain information about a user's sleep cycle and any present problems therein by use of a non-invasive tremor sensor. In short, I wish to take a signal produced by a tremor, process it to obtain the information that I want and classify it based on a dataset that I have trained the model on. Thus we can classify the goals as: obtain the tremor data from the user at a high enough frequency, process the data to a form that presents meaningful information, and classify it based on features present.

I am looking into characterizing tremors based on the cepstrum of the time domain signal obtained from two accelerometers attached to the hands. [4] outlines a variety of values to look for in the signal. I looked at papers that outline the properties of tremors and how different disorders produce different types of tremors.

The final product is a glove with an Arduino chip, accelerometer, and Bluetooth transceiver sewn onto it that will transmit the position at a rate of 30 Hz to a computer running Matlab that will parse and analyze the data. Challenges include preventing aliasing, identifying short impulses, and identifying tremors themselves from background noise.

Overall, the first and second goals were definitely achieved at the end of the project. Obtaining the data was easily done by a commercially available accelerometer with a Bluetooth module built in. From there, I was able to synchronize three different outputs, one for each axis on the accelerometer, into one periodic function such that each axis was equally weighted. This latter detail is crucial to making the system as a whole invariant with respect to how the user was positioned.

The final goal, classification of tremors, was not as cut and dry as the others and was achieved in a lower capacity. The analysis was able to be performed on the data, but due to the difficulty of obtaining usable data and the lack of professional diagnoses to use as labels, classification was based on the physical aspects of the tremors themselves, rather than creating a link between tremor and disorder.

1 Introduction

Sleeping disorders affect around 40 million Americans, including my father, my cousin, and myself, which is a leading reason why I took on this project. The diagnosis and

preventative measures both are arduous and uncomfortable. For example, my father must, on occasion, wear a breathing mask when going to sleep. The goal of this project is to determine a noninvasive method to detect and diagnose a sleeping disorder in the patient. A noninvasive sensor in this case is defined as any sensor that does not pierce the skin or involve hooking up electrodes to the user. The features that I am looking into are tremors in the user's hand. A tremor is an involuntary, rhythmic muscle movement involving one or more parts of the body [1], most commonly in the hands, and can be caused by a variety of reasons: lack of sleep or nutrients, high stress, or neurological disorders. Any or all of these may present themselves in a patient's sleep.

In order to use tremor sensing to detect sleeping disorders, the logical first step is to first detect a tremor. Once detected, however, the tremors are not in a form where they are easily classifiable and I must process the data to obtain meaningful information. Following this, I can use machine learning (ML) techniques on a data set to characterize the disorder. Although they are low frequency, tremors might also be low amplitude. [3] lists moderate tremors as having an excursion of less than 2 cm. Slight tremors have even lower amplitude. For this reason, a high resolution detector might be needed. Given that tremors most commonly occur in the hands, some kind of bracelet that can monitor motions in the hand may be the best kind of sensor. It can stay close to the skin while also being non-invasive.

2 Standards and Definitions

In this section, I define formulae, acronyms, etc. that will be used throughout this report.

- The imaginary unit will be denoted $j = \sqrt{-1}$ as is convention in electrical engineering applications.
- Any vector will be denoted by a boldfaced letter, e.g. $\mathbf{v}$, and the k^{th} index of the vector $\mathbf{v}$ will be denoted as $\mathbf{v}_k$ or $v[k]$ interchangeably. Multiple indices will be denoted with a colon in the latter format, the m^{th} through the n^{th} indices of $\mathbf{v}$ will be $v[m:n]$. If no specification is given, all vectors will be assumed to be column vectors. As code is written in Matlab, all vectors will be 1-indexed, i.e. the first and last indices of an n-length vector will be $v[1]$ and $v[n]$ respectively.

- The Discrete Fourier Transform (DFT) of a vector $\mathbf{v}$ will be denoted as $\mathbf{V} = \mathcal{F}\{\mathbf{v}\}$ and is computed by:

$$V[k] \stackrel{\text{def}}{=} \sum_{n=0}^{N-1} v[n] \cdot e^{-2\pi jkn/N}$$

  The input and output of this computation will be referred to as the time domain representation or frequency domain representation of the signal respectively.

- The Inverse DFT (iDFT) of a vector $\mathbf{V}$ converts the frequency domain representation back into the time domain representation and will be denoted as $\mathbf{v} = \mathcal{F}^{-1}\{\mathbf{V}\}$ and is computed by:

$$v[n] \stackrel{\text{def}}{=} \sum_{n=0}^{N-1} V[k] \cdot e^{2\pi jkn/N}$$

- The DFT cepstrum of a vector $\mathbf{v}$ will be denoted as $\mathbf{v}_c$ and is computed by:

$$\mathbf{v}_c = |\mathcal{F}^{-1}\{\log|\mathcal{F}\{\mathbf{v}\}|^2\}|^2$$

  As the cepstrum is the iDFT of a scaled DFT of a signal, the units are still seconds.

3 Design Alternatives

3.1 Feature Vector

Originally, the feature vector was to be the concatenation of the first 12 coefficients of the cepstrum of each of the three axes. The problem with this approach is that it is not position invariant. Normally, position variance can be remedied by subtracting out the DC component of the signal, a very quick computation. In this case, however, the problem was the fact that the signed 8-bit integer (used for speed) could overflow or underflow. For example, in a given hand position, let us say that the position data is outputting a value of 120. Movement in the direction of increasing position could cause the position data to wrap around to a negative value, causing a large spike in high frequency components when, in actuality, the data is still relatively low frequency. To remedy this, I use the approach outlined in Section 4.1.1.

I plan to implement a basic neural net machine learning algorithm in order to combine my current algorithms and create a better nonlinear classifier. I can also use a kernalized

Radial Basis Function (RBF) model because it is a universal classifier. It is not tenable for large n or d being the size of the training set and dimension but my feature vector is relatively short and speed is not a concern. From here, I will need to figure out a way of assigning labels to the training data in order to actually use the algorithms. I also plan to write more functions to generate more/different features if necessary.

3.2 Classification Algorithms

All machine learning algorithms have tradeoffs and assumptions about the underlying data.

3.2.1 Support Vector Machine

SVM is a binary, linear classifier. I have extended my implementation to classify the vector iteratively until all labels are exhausted but I do not believe this to be a valid approach. Still, it was very easy to code so implementing it was not a problem. SVM has the benefit of being more robust to noise than kNN and can be kernalized for speed. I use a function $\phi(x)$ to map a vector from the input vector space to a combination of features: $\mathbf{x} = [x_1, x_2, x_3, \ldots, x_d] \mapsto \begin{bmatrix} 1 \\ x_1 \\ x_2 \\ \vdots \\ x_d \\ x_1x_1 \\ \vdots \\ x_dx_d \\ \vdots \\ x_1x_2\ldots x_d \end{bmatrix}$. This allows me to use nonlinear decision boundaries such as distance from the mean in a Gaussian model when we have a dataset with the same mean but a different variance on the two classes. I can also introduce slack variables to account for datasets that aren't linearly separable. The problem with using it iteratively is that

3.2.2 Ridge Regression

Regression is more immune to noise than the other two models I used since it inherently assumes some noise in the classifier but it is also a binary classifier in my implementation. It also assumes that the inputs are deterministic instead of behaving as the realization of a random variable which may not be a good assumption to make given the random nature of tremors. I used a squared loss function to calculate error for a misclassified point and used that to generate the best decision boundary for the dataset. I don't plan on using this model because it doesn't offer me anything the other two do, ease for kNN and nonlinearity and speed for a kernalized SVM.

3.2.3 Neural Net

I considered using an artificial neural network to classify points but I thought it would be too complicated for the task at hand and would be more prone to overfitting than the method I ended up settling on.

3.3 Fourier Linear Combining

Fourier Linear Combining (FLC) is a method outlined in [7]. The purpose in this paper was to adaptively cancel noise in physiological tremors by computing a correlated version of the noise and subtracting it from the signal. Over time, the correlated noise becomes more and more accurate and the approximation becomes more and more accurate. At the time that most of the results had been obtained, I had not found this paper so any future work done would aim to implement this technique.

3.4 Hardware

3.4.1 Analog Hardware

I considered using two single-axis analog accelerometers and a PIC32 microcontroller to process data but this was unfeasible. Due to the weight of the microcontroller, the frequency characteristics of the tremor would be significantly altered. An option would be to transmit the accelerometer data wirelessly but that would require a Bluetooth or

other wireless protocol module and an ADC, again altering the weight of the glove. In addition, the signal would be fairly weak and would need an amplifier to get reasonable results which would need to be on the glove. If the amplifier was frequency selective, there would be channel effects that would need to be inverted which would require additional processing before being transmitting.

3.4.2 Higher Sampling Frequency

[8] is a product with a 9-axis accelerometer with a much higher sampling frequency. I think this would be the best way to improve the results. My sampling frequency was enough to capture a bulk of the tremor energy but sharp movements that were observed in some samples were completely unable to be caught.

4 Data Acquisition

Data acquisition was done on a LightBlue Bean. From [4], although used for a different purpose, key features to look for include:

Time Domain	Autoregressive (AR) Coefficients, features of the probability distribution of the time domain series, Decay of the autocorrelation function
Frequency Domain	Power spectral density (PSD) features of accelerometer signal (peak frequencies, higher order moments), PSD coefficients, Cross-Spectral analysis

The sample rate is set at 30 Hz per channel, with the three channels being the x, y and z axes. According to [2], the upper limit on the frequency of a human body tremor is around 12 Hz. By the Shannon-Nyquist sampling theorem, we should sample at a frequency greater than 24 Hz. Because this is a lower bound, any frequency above that will also meet the constraint.

Unfortunately, the process is not going to be stationary but first order statistics are enough to classify them. The data is sent over a wireless channel to a computer for storage. Real time processing was not a concern for this project and was not supported due to

synchronization problems.

The Arduino code calls for a reading of the accelerometer at a rate of 30Hz which is transmitted to the computer. The Bean returns a datatype that stores the 3 acceleration values as three 10 bit signed integers. Arduino's built in`Serial.println` (which would print the values -512 to 511) function is too slow to send the data as is so I used `Serial.write` which can only transmit and not print byte values (0b10000000 to 0b01111111). Because I set the acceleration to have a dynamic range of $\pm 2g$, using 8 bits per value gives me a resolution of about 1% of a g which is sufficient. To properly send the data without the need of a buffer, I shifted the 10 bit values right by 2 places and transmitted this new 8 bit value. Matlab could then read it as an `int8` for processing.

4.1 Tremor Extraction

Once the entire signal has been recorded, I must extract meaningful data from it. This signal is largely silence with data interspersed. To obtain the signal, I experimented with various techniques and what worked best was the following:

1. Wait for silence and block off the incoming signal into frames of some length n. For my purposes, I used $n = 100$.

2. Subtract the mean value of this frame from all incoming data points to prevent over/underflow (outlined in Section 4.1.1). In order to prevent any value from diverging, a regularization constant is added, set in my software as 0.01.

3. Continually find the 8 point cepstrum $\mathbf{x}_c$ of the signal. More data points can be used if needed, such as a high noise environment.

4. If there is a spike in $x_c[2]$ greater than 3 times the noise variance:

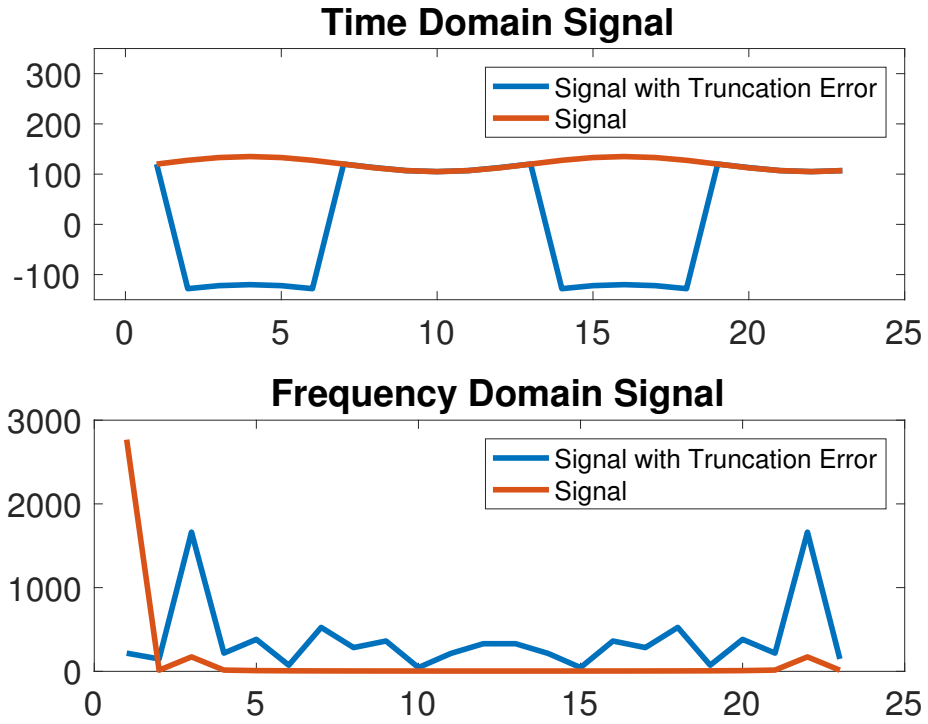
   (a) If there is a spike in the time domain, record the data as long as $x_c[2]$ is above the threshold and x_t is oscillating with an amplitude greater than 3 times the noise variance.

   (b) This will result in some k length signal which will be our data signal.

This process will be done over all three axes synchronously to find a signal.

4.1.1 Signal Correction

Once the signal has been recorded and extracted, I will need to extract the features from it. Originally, I had planned to use the concatenation of the first 12 coefficients of the cepstrum from all three axes but there are a few problems with this approach, primarily the problem with overflow and underflow (other flaws are outlined in Section 3.1). For example, if the average value of the silence was around 120, a slight perturbation could cause the value to wrap around to a negative number. Let us say that the amplitude of an incoming tremor is 15 units: at the peak of this signal, the value should read -121. If the signal was silent for an extended period, I subtracted out the average value, making sure to keep all position values within the dynamic range of a signed 8-bit integer (-128 to 127). This correction was computed in a silent portion of the data vector and was not changed until a tremor was detected. Shown below is an example of this phenomenon.



In addition to over/underflow, a tremor can be oscillating in multiple dimensions. Using the scheme outlined above would produce different feature vectors depending on the orientation. To this end, I took the three separate axis signals and, before further processing, computed $v_{total}[n] = \sqrt{v_x^2[n] + v_y^2[n] + v_z^2[n]}$ where $\mathbf{v_x}$, $\mathbf{v_y}$, and $\mathbf{v_z}$ are the

signals on each axis. Care must be taken to synchronize the signals properly, motion of the whole arm will undoubtedly erroneously trigger the data to save the data as a tremor when it is in fact noise. This method gives the feature vector both position and rotation invariance and was adapted from the process outlined in [6]. In my case, periodicity was a strict constraint due to the unpredictable motion of the human body and what motion was meant to be captured and analyzed. Once periodicity was established, the signals were synchronized with the zero crossings as outlined and combined with simple spherical projection.

4.1.2 Feature Vector Generation

From here, I have a 1-dimensional signal constructed from the original 3-dimensional data but no feature vector to use for classification. Data is still in the time domain which is not suitable for classification due to not being time invariant or low dimensional. If the time domain signal was my feature and the sample was started 1 time step later, it would register as vastly different. To this end, I needed a low dimensional and time and phase invariant feature. The DFT cepstrum was the feature I ultimately used, the formula for which is shown in Section 2. The first 12 values of the result will be the feature vector used.

5 Classifying Tremors

The subject of machine learning (ML) involves learning the structure of data through underlying features, in this case frequency and amplitude. There are many algorithms and methods used in the field, some of which are outlined in Section 3.2. The algorithm I used to classify the feature vector is k Nearest Neighbors (kNN) which I thought was best for several reasons. The algorithm simply assigns the mode of the labels associated with the k nearest points to the input to be classified where k is a user chosen parameter. kNN assumes that similar inputs have similar outputs and does not scale well as the dimensions of the feature vector increase due to the fact that volumes in hyper-dimensional objects go to 0. The resulting "Curse of Dimensionality" means that the pairwise distances of independent and identically distributed points in high dimensional spaces are normally distributed with

variance that is inversely proportional to the number of dimensions. The proof is as follows:

> It can be shown that the volume of a d-dimensional hypersphere with radius r is $\frac{2r^d\pi^{d/2}}{d\Gamma(d/2)}$ where Γ is the generalized factorial function, $\Gamma(k) = (k-1)!$ for integer k [9]. Additionally, it can be shown that a d-dimensional hypercube with side length r is r^d [10]. If we inscribe a unit hypersphere inside a hypercube (necessarily with side length 2), we get all points within Euclidean distance 1 of the origin and get the corresponding ratio of volumes:
>
> $$\begin{aligned}\frac{V(d\text{-cube})}{V(d\text{-sphere})} &= \frac{2^d}{\frac{2\pi^{d/2}}{d\Gamma(d/2)}} \\ &= \frac{2^d d\Gamma(d/2)}{2\pi^{d/2}} \\ &= \mathcal{O}(2^{d/2}d\frac{d}{2}!)\end{aligned}$$
>
> Which diverges as d increases. This means that as $d \to \infty$ the proportion of points within distance 1 of the origin goes to 0. Consequently, the probability of a randomly selected point being within distance 1 of the origin goes to 0.

kNN does, however, have 3 key features that led me to choose it:

1. It is a supervised algorithm. A supervised machine learning algorithm is one that assigns a label to the inputs that it classifies. In this case, the label is the sleeping disorder. I believe that a supervised algorithm is best due to the fact that diagnoses in general must have a label to them, e.g. partitioning patients in a clinic based on health is much less useful than telling them what their ailment is. A supervised algorithm uses a training set to "train" the model to output a certain label when presented with a feature vector that fits certain criteria. Once trained, the model will accept members of a testing set and output a label that best matches the input given the model.

2. It does not use binary labels. In addition to having a label, kNN assigns nonbinary

labels. There are various sleeping disorders and one label is not enough to show that information. It does nobody any good to know if they are ill or not, the particular disorder should be known. Although there are ways of making algorithms that produce binary labels to output more than two, they may be overfit to the training set and may have higher testing error than desired.

3. It creates nonlinear decision boundaries. Real data is generally nonlinear and, as such, an algorithm that produces linear boundaries would suffer. The decision boundaries produced by this algorithm are not dependent on the algorithm itself, up to the choice of k, only the data.

k Nearest neighbors is a relatively slow algorithm due to the need to check pairwise distances to all training vectors in the dataset but using KD trees or Ball trees greatly speeds up this process. Both of these trees are fairly easy to make and traverse and take each split lowers the time complexity to $\mathcal{O}(n \log n)$ on average. In addition, even without the use of trees, the process is completely vectorizable and uses no loops which Matlab is extremely quick in processing. Other algorithms considered are outlined in Section 3.2. Figure 1 shows the decision boundary produced by the 3NN algorithm. Appendix **??** lists the code used to generate this figure.

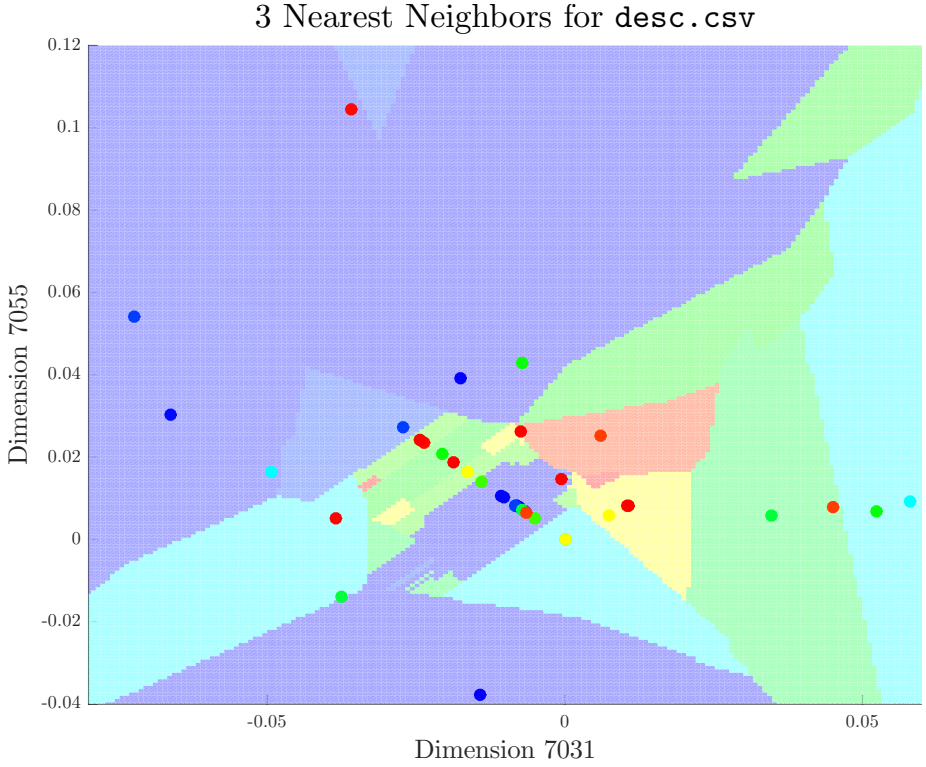


Figure 1: Decision boundaries produced by kNN

6 Hardware

Hardware for this project is very simple, all processing was done in software. The only hardware part used was a LightBlue Bean that came prepackaged with the accelerometer and Bluetooth module that I needed. Figure 2 is a high level design flowchart for the project. The Arduino code calls for a reading of the accelerometer at a rate of 30Hz which is transmitted to the computer. Arduino's built in`Serial.println` function is too slow to send the data as an integer

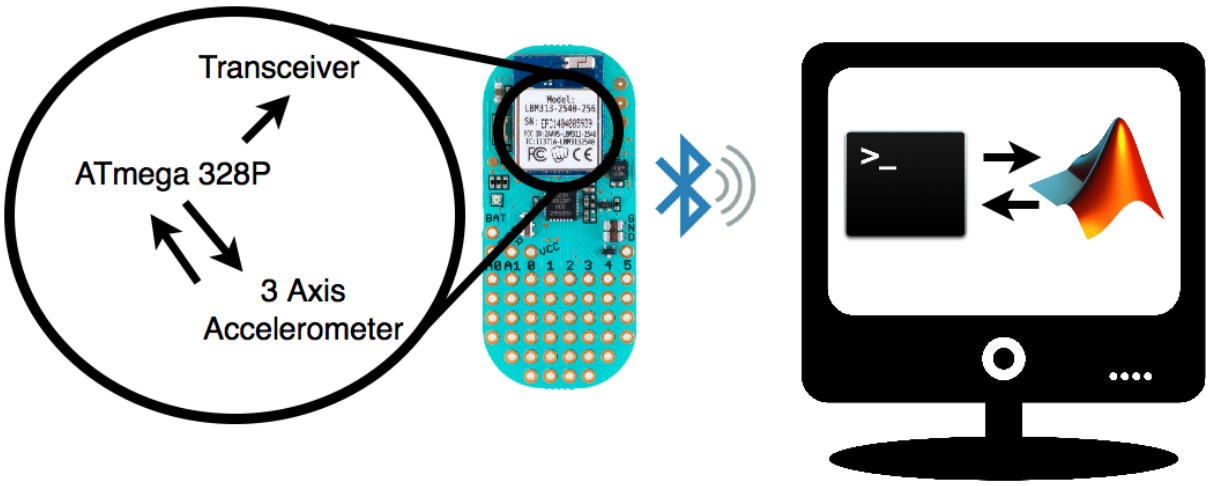


Figure 2: Schematic for the Design

6.1 Limitations

The chip used was limited to 9600 Baud which equates to about 30 Hz per axis. During research done before purchasing the chip, I found out that tremors occur at or under 12 Hz so the chip was able to sample at just a high enough rate to satisfy the Shannon-Nyquist limit.

7 Results and Discussion

Tremor Characteristics	Classification Accuracy
Still arm tremor	6/7 classified accurately
Total arm tremor	3/6 classified accurately
Short tremor	1/3 classified accurately

Table 1: Results of Algorithm measured by Classification Accuracy

Table 1 shows classification successes for 3 types of tremors. A success was determined by whether or not the tremor detected was classified to other tremors with the same physical characteristics which were observed as well as recorded. Datapoints that did not exhibit any tremor were also added into the training data to avoid silence being misclassified. All disorders in this experiment were self reported so it may be prone to inherent human error or biases. To mitigate this, the labels used were based on the physical characteristics

of the tremors rather than any diagnosis. I believe this was best when no professional diagnosis was available to me personally but can be easily changed.

Datasets were collected from 3 individuals who exhibited tremor while sleeping. Each dataset potentially contained multiple tremors that could be classified and multiple datasets were collected from each person.

Thus, the experiment design was as follows: An individual would have a dataset recorded which was used as an input to a function that processed and classified the tremor. Accuracy was based on whether or not the output of kNN assign the same label to the input tremor as I had assigned myself by observing the subject (the ground truth). Due to the lack of professional diagnoses, the labels were based on the physical characteristics of the tremor (still arm, etc.)

7.1 Still Arm Tremor

Still arm tremor, like the name suggests, is a tremor where the arm itself is still and only the hand exhibits the tremor. According to [3], this type of tremor occurs at frequencies around 3-6 Hz. They typically have a low to moderate excursion length, less than 2 cm. Still arm tremors can be considered the high signal-to-noise ratio (SNR) regime of the experiment, external motions that could affect the measurements are at a minimum. In this regime, classification accuracy was highest. This is a rather cut and dry case, there are minimal lurking variables that could cause error. The only error was misclassified as silence meaning the tremor in question was of extremely low amplitude. This would lead to low power in the frequency band between 3-6 Hz which, when lowered further by the logarithm function, would lead to such a misclassification. The figure below shows a typical waveform of a tremor in this regime.

7.2 Total Arm Tremor

Total arm tremor is a tremor where the entire arm moves and not just the hand. Contrary to the previous case being the high SNR regime, this is the low SNR regime of the experiment. This is due to the motion that affects the hand, and thus my measurements, while still originating from the hand itself. Movement of the arm increases the excursion

length and changes the frequencies present. Total arm tremors were only classified with about 50% accuracy on my testing set. I believe this relatively low accuracy is largely due to an indirect feature of the noise. Depending on how the arm and the whole body are positioned, the arm will have very different movement options. For example, if a user were to be sleeping on their side with the arm being measured underneath their body, the upper arm would likely not be able to move and the forearm would have a much more limited range of motion than it would if the user were sleeping on their back. This is not a simple case of positioning on a sphere as was mentioned in Section 4.1.1, the motions themselves are fundamentally different. In many engineering practices, the noise, while random, has predictable statistics such as expected value and variance. In this case, the noise is highly dependent on how the user tends to sleep and move in their sleep. As such, I cannot think of a reliable and reproducible way of controlling for this case other than having training data for all the various sleeping positions.

7.3 Short Tremors

Short tremors are less than 2 seconds in length, usually less than 1 second. These tremors had the worst classification accuracy with my model, likely due to the high frequency content therein. Another factor leading to the high error could be due to movement restriction like in in Section 7.2. In the literature I used to research the types of tremors associated with sleeping disorders, tremors of this high frequency were not mentioned.

7.4 Disucssion

Overall, one can see that for tremors that had a relatively long duration, classification was reasonably accurate. Full arm motion was classified less accurately than still arm motion due to the accelerometer capturing the motion of the arm as well as the motion of the hand. Due to the periodicity constraint I had imposed to get the feature vector, I could exclude simple tosses and turns that were not tremors but if the whole arm moved with a higher excursion and frequency than the tremor, it would not register as a tremor. Fixing this would require higher sampling frequency or better resolution to capture the relatively small tremors. With a higher frequency or better resolution, I could weaken the periodicity

constraint to be a local extrema constraint. The worst regime to be classified was by far short tremors, observed as a full arm or full body motion that was extremely short and very high frequency. The sampling rate used was not enough to glean meaningful information from the signal and the success was mostly complete luck. The classification was also overfit due to the high latency in obtaining a dataset, I had to train and test on less data than I had desired. In any potential future work, sampling at a higher rate and obtaining much more data would be of utmost importance.

8 Conclusions

I believe my model and reasoning are correct but there are several aspects to the problem that I was not able to preordain that very much affected the results. I believe the most noteworthy aspect to my approach was discerning tremor from generic motion and from silence. This former is not a trivial problem and I think that imposing a periodicity constraint on the movement is key. In addition, the choice of feature vector and data processing methods are also well founded. Projecting multidimensional data down can result in huge loss of information if handled poorly and I think I handled the problem well given the high accuracy in the high SNR regime. Furthermore, this leads me to believe my choice in algorithm was good. Obviously, more testing would need to be done but this seems to be a good framework.

The biggest problem with how I tackled the problem was, in my opinion, the movement issue. That was a huge bump in the road that wasn't noticed until very late into the year and my accuracy suffered for it. In addition, datasets were very difficult to obtain and due to the lack of professionally diagnosed labels, I was unable to classify my data on anything more than their physical characteristics. If further work is to be done in the area, focus should definitely be in finding reliable data that forms a strong basis for a full training set.

Hardware limitations presented themselves that I was not able to overcome but I believe these to be of minimal effect. Although the true bandwidth of a tremor is further than what I had initially assumed, the bulk of the energy was still capturable. Improving the hardware would not change the fundamental aspects to the process, but it may allow me

to relax some constraints or make less assumptions about the data. Furthermore, using more sensors would greatly aid in classification. A sensor on the user's arm would allow separation of the signals from the hand and the arm, lowering noise in both.

There are several areas to look into moving forward, namely classification. I used a single, basic algorithm that I thought was best suited for the problem at hand but there are many ways to strengthen it. For example, one can combine the prediction of several classifiers through a weighted average to achieve much better success than any of the individual algorithms. Furthermore, using supervised learning techniques is not all the rich field of machine learning has to offer. All in all, I think my efforts were a good start for the task at hand and any further work in the area would have a base to work off of.

9 References

[1] National Institute of Neurological Disorders and Stroke *"Tremor Fact Sheet."* Tremor Fact Sheet. N.p., n.d. Web. 05 Nov. 2015.

[2] *"Tremor Symptoms - Diseases and Conditions - PDR Health."* Pdrhealth. N.p., n.d. Web. 05 Nov. 2015.

[3] A. Ahmed and P. Sweeney *"Tremors."* Cleveland Clinic. N.p., n.d. Published July 2014

[4] Liberty, Matthee G., Christopher D. Roller, Daniel S. Simpkins, and Charles W. K. Gritton. *Methods and Devices for Identifying User Based on Tremor.* Hillcrest Laboratories, Inc., assignee. Patent US 7,236,156 B2. 26 June 2007. Print.

[5] Bruce Land *"DSP for GCC"* Cornell University, March 11, 2013. Web 5 November.

[6] Curtis, Susan and Alan Oppenheim. Reconstruction Of Multidimensional Signals From Zero Crossings. The Research Laboratory of Electronics at Massachusetts Institute of Technology. 1986. Web. 18 May 2016.

[7] Riviere, C.N., R.S. Rader, and N.V. Thakor. "Adaptive Cancelling of Physiological Tremor for Improved Precision in Microsurgery." IEEE Transactions on Biomedical Engineering IEEE Trans. Biomed. Eng. 45.7 (1998): 839-46. Web.

[8] "FSM-9". Hillcrestlabs.com. N.p., 2016. Web. 18 May 2016.

[9] Weisstein, Eric W. "Hypersphere." From MathWorld–A Wolfram Web Resource. http://mathworld.wolfram.com/Hypersphere.html

[10] Weisstein, Eric W. "Hypercube." From MathWorld–A Wolfram Web Resource. http://mathworld.wolfram.com/Hypercube.html

Appendices

A Code Listings

A.1 LightBlue Bean Transmission

```
void setup() {
  Serial.begin(9600);
}

void loop() {
   AccelerationReading accel = Bean.getAcceleration();

   int8_t x = accel.xAxis>>2;
   int8_t y = accel.yAxis>>2;
   int8_t z = accel.zAxis>>2;

   Serial.write(x);
   Serial.write(y);
   Serial.write(z);

}
```

A.2 Tremor Extraction

```
function feature = tremorExtract(data)

fname = 'rrs72_tremor';

% get the three axes of the data
x = data(1:3:end);
y = data(2:3:end);
z = data(3:3:end);

% force column vectors
x = x(:);
y = y(:);
z = z(:);

% initialize everything
f = @(x) abs(ifft(log(abs(fft(x)))));
dftLen = 32;
reg = 0.01;
idx = 1;
len = 100;
maxEx = 5;
sFlag = 1;
feature = [];

% find first instance of silence in all channels
while sFlag;
    xs = x(idx:idx+len) - mean(x(idx:idx+len));
    ys = y(idx:idx+len) - mean(y(idx:idx+len));
    zs = z(idx:idx+len) - mean(z(idx:idx+len));
    m = [xs(:) ys(:) zs(:)];
    if sum(max(abs(m)) <= maxEx) == 3 ...
        idx == min([length(x) length(y) length(z)]);
        sFlag = 0;
    end
    idx = idx + 1;
end

m = mean(m);

% subtract the mean
x = x - m(1);
y = y - m(2);
z = z - m(3);

```

```
vr = [var(x(idx:idx+len)) var(y(idx:idx+len)) var(z(idx:idx+
   len))];


for i = idx:min([length(x) length(y) length(z)])-dftLen
    x_t = f(x(i:i+dftLen) + reg);
    y_t = f(y(i:i+dftLen) + reg);
    z_t = f(z(i:i+dftLen) + reg);
    % there is power in the second cepstrum coefficient and
       there is
    % movement
    if sum([x_t(2) y_t(2) z_t(2)] > 3*vr) == 3 ...
            && max([x(idx) y(idx) z(idx)]) > 3;

        % find the end of the tremor
        x_idx = findEnd(x(i:end),vr(1));
        y_idx = findEnd(y(i:end),vr(2));
        z_idx = findEnd(z(i:end),vr(3));
        end_idx = min([x_idx y_idx z_idx]);

        % find the second point where the absolute value
        % of the "derivative" is maximized
        % this will be used for synchronization
        dx = x(2:end) - x(1:length(x)-1);
        dy = y(2:end) - y(1:length(x)-1);
        dz = z(2:end) - z(1:length(x)-1);

        % find points where the absolute value of the
        % "derivative" is maximized, inflection point
        x_in = find(abs(x(1:length(x)-1)) > abs(x(2:end)));
        x_ip = find(abs(x(2:end)) > abs(x(1:length(x)-1))) +
           1;

        y_in = fin(abs(y(1:length(y)-1)) > abs(y(2:end)));
        y_ip = find(abs(y(2:end)) > abs(y(1:length(y)-1))) +
           1;

        z_in = find(abs(z(1:length(z)-1)) > abs(z(2:end)));
        z_ip = find(abs(z(2:end)) > abs(z(1:length(z)-1))) +
           1;

        % find inflection points
        x_inf = (x_in == x_ip);
        y_inf = (y_in == y_ip);
        z_inf = (z_in == z_ip);

```

```
        x_inf = x_inf(2);
        y_inf = y_inf(2);
        z_inf = z_inf(2);

        x = [zeros(max([x_inf y_inf z_inf]) - x_inf,1); x];
        y = [zeros(max([x_inf y_inf z_inf]) - y_inf,1); y];
        z = [zeros(max([x_inf y_inf z_inf]) - z_inf,1); z];

        break

    end
end

for i = 1:min([length(x) length(y) length(z)]) % use for loop
    in case
                                                % vectors are
                                                   different
                                                % length
    v(i) = sqrt(x(i)^2 + y(i)^2 + z(i)^2);
end

v_c = f(v);

% save for finding more tremors
feature = v_c(2:13);
data = data(3*end_idx+1:end);

save([fname '.mat'],'feature','data');

end

function idx = findEnd(v,thresh)
    for j = 1:length(v)
        if var(v(j:j+8)) < thresh
            idx = j;
            break
        end
    end
end
```

A.3 kNN Classifier

```
function l = KNNClassify(features,training,k,labels)
% rows of tr must be features of the training data
% f must be column vector of the same length as a row of tr
% k is the number of nearest neighbors
% l will be the label
%
% labels must be of type double. if labels are not, they must
    be indexed
% outside this function

d = sqrt((training*features).^2);
[d_s,d_i] = sort(d);
l = mode(labels(d_i(1:k)));

end
```