Tempo-Sensing Feedback System for Student Conductors

Stephen Hoeprich

ECE-499: Capstone Design Project

Advisor: Professor John Spinelli

March 14, 2016

SUMMARY

The purpose of this project is to design and build a system which will serve as a tool for student conductors. The system will give feedback including the tempo that they are conducting in real time and a plot of conducted tempo over time, thereby allowing them to learn quicker and more thoroughly. The system will consist of two components: a beat detector which calculates tempo and an app for display. The beat detector must be small enough that it does not impede conducting. It must be able to detect the tempo that the conductor is conducting within 5 beats per minute of the actual tempo and over a realistic range of tempos. It must also be able to acquire that tempo quickly and must differentiate indicated beats from other motions of the arm. The app must be simple and display the tempo in a large format in the center of the screen, updating the tempo every time a new beat is conducted. The app must also be able to produce a plot of tempo over time for the conductor to analyze later as well. Finally, the two components must communicate wirelessly. For the final version of this project, most of the above design requirements were met, with the exception of the creation of a graph. The final design ended up being slightly different from the preliminary, with calculations being made in the app. The project remained within the budget, but the timeline suffered some fluctuations due to unforeseen setbacks. Nevertheless, these were adjusted to and a working final product was delivered by week 9 of winter term 2016. Valuable lessons were learned through the entire process and there is great potential for expansion on the overall concept of the project. Through this project, student conductors will have access to a device which will provide them with valuable feedback and will enhance their learning curve. This device will train student conductors to become familiar with what certain tempos “feel like,” preparing them to conduct more consistently and at accurate tempos when not using the device later in their careers.

TABLE OF CONTENTS

SUMMARY **2**

TABLE OF CONTENTS **3**

TABLE OF FIGURES AND TABLES **5**

1. INTRODUCTION **6**

2. BACKGROUND **9**

2.1 PREVIOUS METHODS AND WORK **9**
2.2 POTENTIAL PROJECT IMPACTS **10**

3. DESIGN REQUIREMENTS **12**

3.1 DESIGN REQUIREMENTS FOR BEAT DETECTOR **12**
3.2 DESIGN REQUIREMENTS FOR APP **14**
3.3 DESIGN REQUIREMENTS FOR ENTIRE SYSTEM **15**

4. DESIGN ALTERNATIVES **16**

4.1 COMPONENT ALTERNATIVES AND JUSTIFICATIONS **16**
4.2 OTHER ALTERNATIVES AND JUSTIFICATIONS **17**

5. PRELIMINARY PROPOSED DESIGN **19**

5.1 BEAT DETECTOR DESIGN **21**
5.1.1 Beat Detector Construction ***21***
5.1.2 Beat Detector Function ***21***
5.1.2.1 Beat Detection ***22***
5.1.2.2 Tempo Calculation ***24***
5.2 APP DESIGN **25**
5.2.1 App Function ***25***
5.2.2 App Display and Layout ***26***
5.3 FUTURE WORK **28**

6. FINAL DESIGN AND IMPLEMENTATION **29**

6.1 MOVEMENT SENSOR DESIGN **31**
6.1.1 Movement Sensor Construction *31*
6.1.2 Movement Sensor Function *34*
6.2 APP DESIGN **35**
6.2.1 App Function *35*
6.2.1.1 Beat Detection *36*
6.2.1.2 Tempo Calculation *37*
6.2.2 App Display and Layout *39*

7. PERFORMANCE ESTIMATES AND RESULTS **43**

7.1 PRELIMINARY DESIGN PERFORMANCE ESTIMATES **43**
7.2 PERFORMANCE RESULTS **44**
7.2.1 Testing Process *46*
7.3 IMPROVEMENTS **47**

8. PROJECT SCHEDULE **49**

9. COST ANALYSIS **51**

10. USER MANUAL **52**

10.1 ASSEMBLY AND PROGRAM SETUP **52**
10.2 INITIAL OPERATION AND USAGE **55**
10.3 TROUBLESHOOTING AND ADJUSTMENTS **56**
10.4 STORAGE AND MAINTENANCE **58**

11. DISCUSSION, RECOMMENDATIONS, AND CONCLUSION **59**

11.1 DISCUSSION **59**
11.1.1 Problem *59*
11.1.2 Solution and Implementation *60*
11.1.3 Results *60*
11.2 RECOMMENDATIONS **61**
11.2.1 Future Work *61*
11.2.2 Lessons Learned *62*
11.2.3 Expansions *63*
11.3 CONCLUSION **63**

12. REFERENCES **65**

13. APPENDICES **67**

APPENDIX A **68**
APPENDIX B **73**
APPENDIX C **74**
APPENDIX D **76**
APPENDIX E **79**
APPENDIX F **80**
APPENDIX G **85**

TABLE OF FIGURES AND TABLES

FIGURE 1: CONDUCTING PATTERNS FOR VARIOUS TIME SIGNATURES……6

FIGURE 2: TOP-LEVEL DESIGN CONCEPT……12

FIGURE 3: BLOCK DIAGRAM OF BEAT DETECTOR……19

FIGURE 4: BLOCK DIAGRAM OF APP……20

FIGURE 5: PSEUDOCODE REPRESENTING THRESHOLD DETECTION……22

FIGURE 6: PLOT OF MAGNITUDE OF ACCELERATION VS. TIME WITH GENTLE CONDUCTING AT 40 BPM AS DETECTED BY MPU6050 AND IMUDUINO, PLOTTED WITH MATLAB……23

FIGURE 7: PSEUDOCODE REPRESENTING TEMPO CALCULATION PROCESS……25

FIGURE 8: PROCESS FOR CONNECTION TO NRF BTLE TRANSMITTERS IN CURRENT VERSION OF APP…...26

FIGURE 9: DISPLAY OF TEMPO DATA IN CENTER OF SCREEN IN CURRENT VERSION OF APP……...27

FIGURE 10: BLOCK DIAGRAM OF MOVEMENT SENSOR WORN ON USER'S WRIST.....29

FIGURE 11: BLOCK DIAGRAM OF FINAL VERSION OF BEAT DETECTION AND TEMPO CALCULATION ALGORITHM AS PERFORMED ON SMARTPHONE……30

FIGURE 12: 3D-PRINTED ARDUINO NANO CASE REDESIGNED TO FIT IMUDUINO……..31

FIGURE 13: PROTOTYPE VERSION OF MOVEMENT SENSOR……...33

FIGURE 14: FINAL VERSION OF MOVEMENT SENSOR……33

FIGURE 15: CIRCUIT SCHEMATIC FOR FINAL VERSION OF MOVEMENT SENSOR……34

FIGURE 16: PSEUDOCODE REPRESENTING FINAL VERSION OF BEAT DETECTION AND TEMPO CALCULATION ALGORITHM……..39

FIGURE 17: NEW LAYOUT OF APP……40

FIGURE 18: BLUETOOTH CONNECTION PROCESS WITHIN APP……..41

FIGURE 19: APP IN OPERATION, SHOWING A TEMPO OF 62 BPM……42

FIGURE 20: TESTING PROTOTYPE WITH PROFESSOR JOHN COX……...46

FIGURE 21: PROJECT TIMELINE AS PLANNED IN SPRING 2015……..49

FIGURE 22: ACTUAL PROJECT TIMELINE……50

FIGURE 23: POWER LED ON……....53

FIGURE 24: MALE-MALE MICRO-USB TO USB CABLE……53

FIGURE 25: BATTERY CASES AND SWITCH ARE LOCATED OUTSIDE THE MAIN CASE……54

FIGURE 26: LOCATION OF "ZEROADJUST" IN SKETCH "BLUETOOTHMAG2"……..57

TABLE 1: SRG-APPROVED BUDGET WITH ALL DESIGN OPTIONS……51

1. INTRODUCTION

Large musical ensembles—such as orchestras, wind bands and singing groups—consist of musicians and a single director or conductor. The primary responsibility of this conductor is to shape and mold the music being played into something more than just notes on a page. The conductor applies their own musical expertise by adjusting aspects of performance, facilitating rehearsal of the music, and controlling expressive elements such as changes in the volume or speed of the music. Ultimately they are responsible for synchronizing the ensemble; bringing all elements of the performance together—musicians, music, and expression—to create one perfectly coordinated exhibition of sound and emotion. Conductors use their hands and usually a baton to reflect these expressions and to indicate the tempo and time signature of the music being played. Tempo is the speed at which music is played, measured in beats per minute (bpm) and time signature is an indication of how many beats are in each measure of music and which duration of note receives the beat. These are both indicated by movements of the conductor's baton in various directions and with different patterns in the air specific to certain time signatures, as seen in Figure 1.

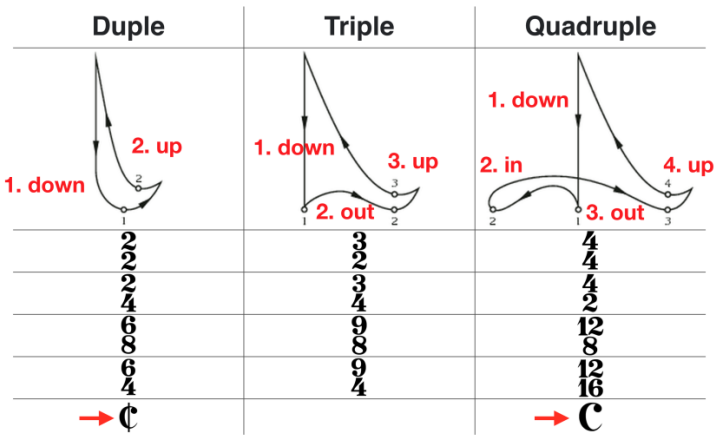


Fig. 1: Conducting Patterns for Various Time Signatures [3]

Conductors learn their art through years of education and practice. In fact, for most professional conductors, a master's degree is essentially a minimum requirement [8]. The Director of Performance at Union College, Professor John Cox, revealed that it is very rare for a student conductor to have the opportunity to lead an ensemble, even at a small school such as Union. Hence, student conductors must practice mostly on their own to master the intricacies of conducting. A common challenge for student conductors is accuracy and consistency when conducting at a certain tempo. While experienced conductors are able to draw upon an internal feel built up over years of practice, inexperienced conductors are more apt to conduct too fast or slow without a reference. They are also likely to stray away from the intended tempo.

Therefore the objective of this project is to provide a student conductor with a tool which will aid them in maintaining accuracy and consistency in conducting a specific tempo, without sacrificing the educational value of building an internal sense of tempo. This will enable the student conductor not only to learn quicker and more thoroughly, but also to receive as much feedback as possible from the rare opportunity to lead an ensemble. A two-component system will be built to accomplish this objective. The first component will be worn on the user's wrist, and will pick up their arm movements. The second component will be a phone app which will display the tempo at which they are conducting in real time. This app will also graph the tempo over time for the student conductor to analyze after conducting, showing where they sped up or slowed down and at what rate. The two components will communicate via Bluetooth. Since this system will tell a student conductor exactly what tempo they are conducting in real time, they will understand how that tempo feels, building their internal sense of tempo. It is expected that this system will greatly accelerate a student conductor's learning curve and will be extremely useful in their preparation for professional careers.

The rest of this report is organized as follows. Section two will cover some common methods student conductors use to improve their skill, similar projects done in the past, and the potential impacts of the system from social, sustainability, manufacturability and ethics perspectives. Section three will outline the design requirements of the project, specifying how it must perform within each component and how the different components must interact with each other. Section four will discuss design alternatives and the justifications for each method and component ultimately chosen. Section five will explain the preliminary design of the system in its entirety with as much detail as possible. Section six will go into fine detail describing the complete and final design of the system. Section seven will compare the performance expected from the preliminary design with the performance given by the final design, will discuss the testing of the prototype, and suggest improvements to the final design. Section eight will compare the initial project schedule with the actual timeline of events that took place. Section nine will analyze the cost of the project and discuss how funds were used. Section ten will be a comprehensive user manual providing instructions on assembly, operation, troubleshooting, and storage. Section eleven will provide a discussion of the project, recommendations and future work, and a conclusion. Section twelve is a list of references used throughout this report. Section thirteen contains seven appendices referred to throughout the report.

2. BACKGROUND

2.1 Previous Methods and Work

A technique for conductors to test the accuracy of the tempo they are conducting is by conducting along with a metronome. A metronome is a device which produces an audible clicking noise once per beat at a tempo preset by the user. It provides a fixed reference for only one tempo at a time, but never wavers. Student conductors can practice with a metronome to build up their senses of specific tempos but it is not the optimum training device. A metronome does not meet the requirements of this project as it is purely a reference. If used during rehearsals or performances, the clicking sound takes away from the quality of the music and can be a distraction. A metronome will not ever change tempo unless reset and cannot adjust to the conductor themselves. Student conductors will also often conduct along with recordings of music to practice such things as time signature and tempo changes [6]. However this is once again a passive method, matching a tempo that is already preset, as opposed to having real-time feedback to a tempo being conducted by the student at that very moment.

After an extensive search in the United States Patent and Trademark Office database, the only patented systems that are remotely similar to this project are unsatisfactory to meet the design requirements. Every system on the USPTO database having a goal of detecting the motion of the user in relation to music either uses motion capture or influences the playback of a music recording. Take, for example, a system entitled "*Apparatus and Method for Detecting Performer's Motion to Interactively Control Performance of Music or the Like*" [1]. This system uses a motion detector and a series of control systems to influence the rate at which a music recording is played based on the physical motion of the user. The user can effectively utilize this

system to speed up or slow down music that is played based on the motion of their hand or other body parts in the same manner that a conductor influences the tempo of an ensemble they are leading. However this system, like others using motion detection to influence the tempo of a pre-recorded piece of music, does not display a numerical tempo as a result of the user's input, hence making it impractical for solving the problem addressed in this report.

2.2 Potential Project Impacts

This project's impacts will be contained mostly within the relatively small music community. Arguably its biggest impact will be within the realm of music education as it will be attractive not only to student conductors but to music teachers and their students as well. Student conductors will use it for their own personal development, enhancing their conducting abilities and helping to build their internal senses of tempo. However, it can also be beneficial to non-conductors. For a music teacher, the system's tempo detection and display features can be used in a multitude of ways to help teach concepts to a student, such as tempo consistency and how to follow a conductor. For example, a perfect conducting pattern is not necessary for the system to detect tempo, so a music teacher will be able to simply move their hand up and down to indicate a certain tempo for a student to follow when playing their instrument. Many music teachers are also ensemble directors in schools and communities but sometimes lack a formal and comprehensive education in conducting [18]. Therefore this project will be beneficial to them as they will be able to use it to reinforce and improve their own conducting during rehearsals and performances.

The product is designed to be highly manufacturable, using a relatively low-cost microprocessor with embedded accelerometer and Bluetooth capabilities, housed in a generic plastic case and simple wristband, while the accompanying app will remain free. Ideally, costs will remain low, considering that most musicians, students, and teachers have relatively low salaries [13]. Despite how useful the system is, if it is perceived as too expensive by its own user base, then it is likely to not be used at all. An affordable product also means it is more readily available for use in low-income areas—meaning music teachers, students and aspiring conductors who seek to use the system in these areas will not be held back by its cost. In a study by Northwestern University, children from low-income families who were placed in music courses showed greater performance in reading and comprehension than their peers [10]. The system could have a tremendous impact in these areas if kept at a reasonably low price, giving opportunity to those who may be at a financial and social disadvantage.

Ethically, this project will have a positive impact on all who use it. Despite the fact that it will aid in a student conductor's learning process and will potentially accelerate their learning curve, it will not give one student an unfair advantage over another. The system is merely designed to give feedback to the user. If the user's conducting abilities drastically improve after using the system, it is a result of learning from the feedback given. This system will not cause the user to develop a dependency for this same reason. Similar to training wheels on a bicycle, the system is designed to let the user "feel" the tempo at which they are conducting. By merely using it multiple times, the user will quickly learn what specific tempos feel like and eventually will not need the system in order to conduct well.

3. DESIGN REQUIREMENTS

In order for this system to function in an effective manner, it must satisfy or surpass a series of design requirements. These requirements are identified based on both the problems the project aims to solve and its user base. The design requirements are divided into three sections: requirements for the beat detector, requirements for the app, and requirements for the system as a whole. See Figure 2 for a top-level breakdown of the system design.

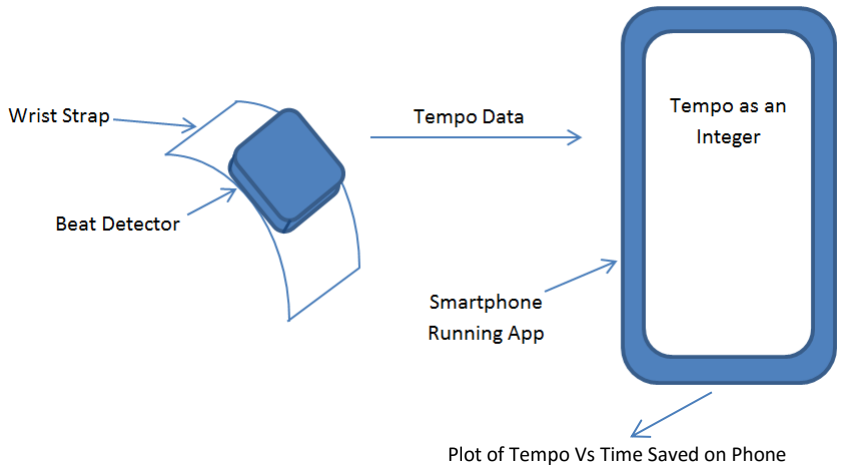


Fig. 2: Top-Level Design Concept

3.1 Design Requirements for Beat Detector

The beat detector is defined as the portion of the system containing a microprocessor within a wristband and plastic case which is worn on the conductor's wrist. It detects the motion of the conductor's arm and is able to pick up the beats of music as indicated through such motion. Within the microprocessor, the tempo will be calculated and transmitted via Bluetooth to the smartphone app.

First and foremost, the beat detector assembly—that is the microprocessor containing the accelerometer, Bluetooth transmitter, and battery—must be small enough to fit in a wristband with a plastic case no larger and weighing no more than a wristwatch, and must communicate with the app wirelessly. The reason for this is so that the conductor's hand is not weighted, dragging a wire, or otherwise impaired. If this were so, it would adversely affect their conducting style such that if they were to conduct without the system, their conducting form would be altered. The battery must also last for at least 96 hours of use. Due to the size requirements of the beat detector assembly, it will need to be powered by a very small battery. However, this battery should not need to be frequently replaced, as this will take away from the usefulness of the system.

Secondly, the beat detector must be able to differentiate between actual beats as indicated by the conductor and other motions unrelated to tempo. Sometimes conductors will add flourishes or other gestures when conducting that are expressive in nature, and if these are recognized as beats, the calculated tempo will be thrown off.

Third, the beat detector should calculate the first tempo within five seconds of when the conductor begins conducting and should update the calculated tempo once every beat. This is important because if the conductor happens to be leading an ensemble and using the system, that ensemble will begin playing as soon as the conductor begins conducting. If the conductor needs to adjust tempo, they should be able to do so as soon as possible to avoid confusing the ensemble or negatively affecting the rehearsal or performance. The tempo should be updated once every beat because continuously updating feedback will be necessary for novice conductors to maintain consistency.

Finally, the beat detector must contain a tempo calculation algorithm which takes detected beats as input and yields tempo as a single integer for an output. This algorithm must be able to accurately detect any tempo from 40-200 bpm, within +/- 5 bpm of the actual tempo. This calculated tempo will then be wirelessly transmitted to the app. This algorithm is designed to reflect what is practical for the user. 40-200 bpm is the range of tempos a human can realistically conduct without straying off tempo. Therefore the tempo calculation algorithm does not need to be able to detect tempos far outside of that range. A variation of +/- 5 bpm is about what a human can typically detect by feel. Even if there is a discrepancy in calculated tempo which is less than or equal to +/- 5 bpm, it will more than likely not be detected by the user and will not affect the way they learn with the system.

3.2 Design Requirements for App

The app will receive the tempo as wirelessly transmitted from the beat detector and will display it as a number on the phone screen. The app will also record the tempo over time and will display it as a graph once the conductor finishes conducting.

First, the app must display the tempo in large numbers which stay onscreen until updated, while being simple and aesthetically pleasing. The larger the numbers are on the app, the easier they will be to see. The conductor should be spending the majority of their time focusing on the ensemble and the music, not trying to read small numbers on a phone. The numbers should stay stationary and centered on the screen until updated. This way, they do not disappear before they need to. The less confusing the app is, the easier it will be to use.

Second, the app must log tempo data over time, plot it, and save the plot to the smartphone's internal memory. This plot will allow the conductor to go back later after a rehearsal or performance and analyze where they sped up or slowed down and at what rate. This gives them additional feedback after the rare opportunity to be in front of an ensemble. If the plot is saved to the smartphone's memory, the conductor can retrieve the saved plot from the phone and move it to a computer or other system to view it on other devices.

3.3 Design Requirements for Entire System

The overall system will need to be able to operate in a standard indoor environment—with room temperature and humidity. It must also comply with FCC wireless regulations. The system is designed for use in concert halls and rehearsal spaces, which are almost always indoors. Rarely does a musical ensemble with a conductor perform or rehearse outside, and never in inclement weather. Therefore, the system has no need to be weatherproofed or designed for outdoor environments. FCC wireless regulations [17] are placed upon every wireless system and apply to this project as well. The Bluetooth transmitter used in this project passed wireless regulations as it is integrated into a microprocessor which is sold as one unit [19].

4. DESIGN ALTERNATIVES

One of the first steps in bringing this project to fruition was brainstorming several options for components, software, and methods. The options are explained below, and separated into two categories: component alternatives and other alternatives. The justifications for the choices made within each category are included within the description of the alternatives.

4.1 Component Alternatives and Justifications

Arguably the most consequential decision made was which microcontroller to implement with this project as the beat detector. The microcontroller needed to be small enough to operate within the design requirements, and contain a reliable embedded accelerometer and Bluetooth transmitter. After extensive research, two chips were found that satisfied these requirements: the Femtoduino IMUduino board [9] and the MetaWear RG board [11]. Both have their merits and drawbacks, but ultimately, the IMUduino was chosen.

The IMUduino is 15 x 41mm microcontroller with an nRF8001 Bluetooth Low Energy (BTLE) transmitter [14] and an MPU6050 six-axis gyroscope and accelerometer [12]. It is an Arduino UNO/Arduino Leonardo clone and is easily programmed via the Arduino IDE. Unfortunately, the IMUduino has a few drawbacks: it is about $30 more expensive than the MetaWear RG board and has no integrated battery. The MetaWear RG board on the other hand is 26 x 17mm and uses an nRF51822 BTLE transmitter [15] and a BMI160 6-axis gyroscope and accelerometer [5]. Neither combination of BTLE transmitter and accelerometer offered any marked advantage over the other. While the MetaWear RG board offers an integrated USB-rechargeable battery, smaller size and better affordability, it does not offer the ease of

programming directly via the Arduino IDE. All of its control is done via a dedicated app from its manufacturer, MbientLab. If any specific task needs to be carried out that is not already covered in the dedicated app, then a brand new app needs to be written. Ultimately, the IMUduino was chosen for this reason, since a tremendous amount of troubleshooting and debugging would be necessary for this project and working through the Arduino IDE was far simpler and more familiar than crafting a complete Android app to do anything more than simply receive data.

The second decision to make was based on how to house the beat detector. The options were to house it within the base of a baton or to implement it on a wristband. If it were housed within a baton, it would not only be aesthetically pleasing but also would be marketable as a single system including the baton. While it did not have these perks, the wristband offered more flexibility as choral conducting is often done without a baton. It also allowed for a slightly larger microcontroller and would be easier to use out of the box. For these reasons, the wristband approach was chosen over the baton, and the Voguestrap TX51012BK Allstrap 16 mm Black Nylon Adjustable Sport Watch Band was selected. This wristband was adjustable to fit any wrist, and featured a nylon strip which would be easy to attach the plastic case to.

4.2 Other Alternatives and Justifications

Another decision to make was regarding the implementation of the tempo display. The tempo could be displayed and a graph could be created and saved either on a computer or on a smartphone. Using a computer would be far easier, bypassing writing complicated apps and offering more flexibility in data interpretation and representation. Using a smartphone offers small size and great portability but introduces the challenges of app writing and data

interpretation on a phone. The implementation of the smartphone was decided upon since its small size allows for conductors to bring it with them wherever they go. Reading tempo data off of a computer screen would be impractical especially if the conductor chose to bring the system with them to a rehearsal or performance. Since app writing is somewhat outside the field of Electrical Engineering, a pre-existing app would be slightly modified instead of writing a brand new one. The source code for an Android app that reads data from a Bluetooth transmitter was downloaded from the website of Nordic, the manufacturer of many Bluetooth products, and included permission to modify [16].

The last decision to be made was whether or not to implement the app in Android, iOS, or both. Android is easier to work with and customize, while iOS proves to be more challenging. In addition, the software required to write iOS apps is not free. One may think that iPhones are more popular, but in fact more Android phones seem to be sold than iPhones [20]. Lastly, only an Android phone was available for testing. Based on all these factors, it was decided to modify the Android version of the app alone.

5. PRELIMINARY PROPOSED DESIGN

The two separate components: the beat detector and the app, were crafted simultaneously and based on the design choices made above. The detailed design of each major component is discussed below, with each component covered in its own section. A reference for the complete design breakdown of the entire system is found below in Figures 3 and 4.

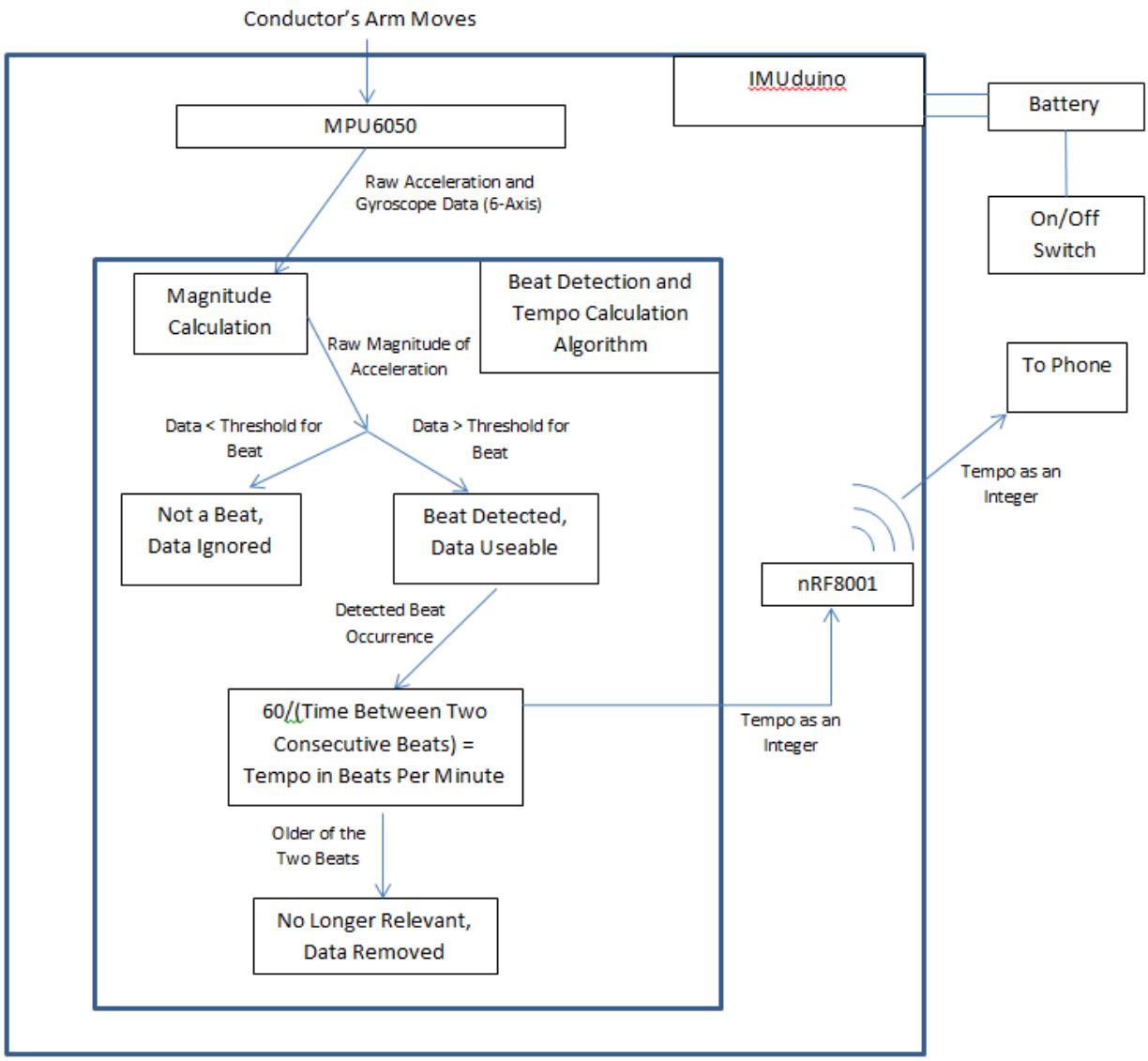


Fig 3: Block Diagram of Beat Detector

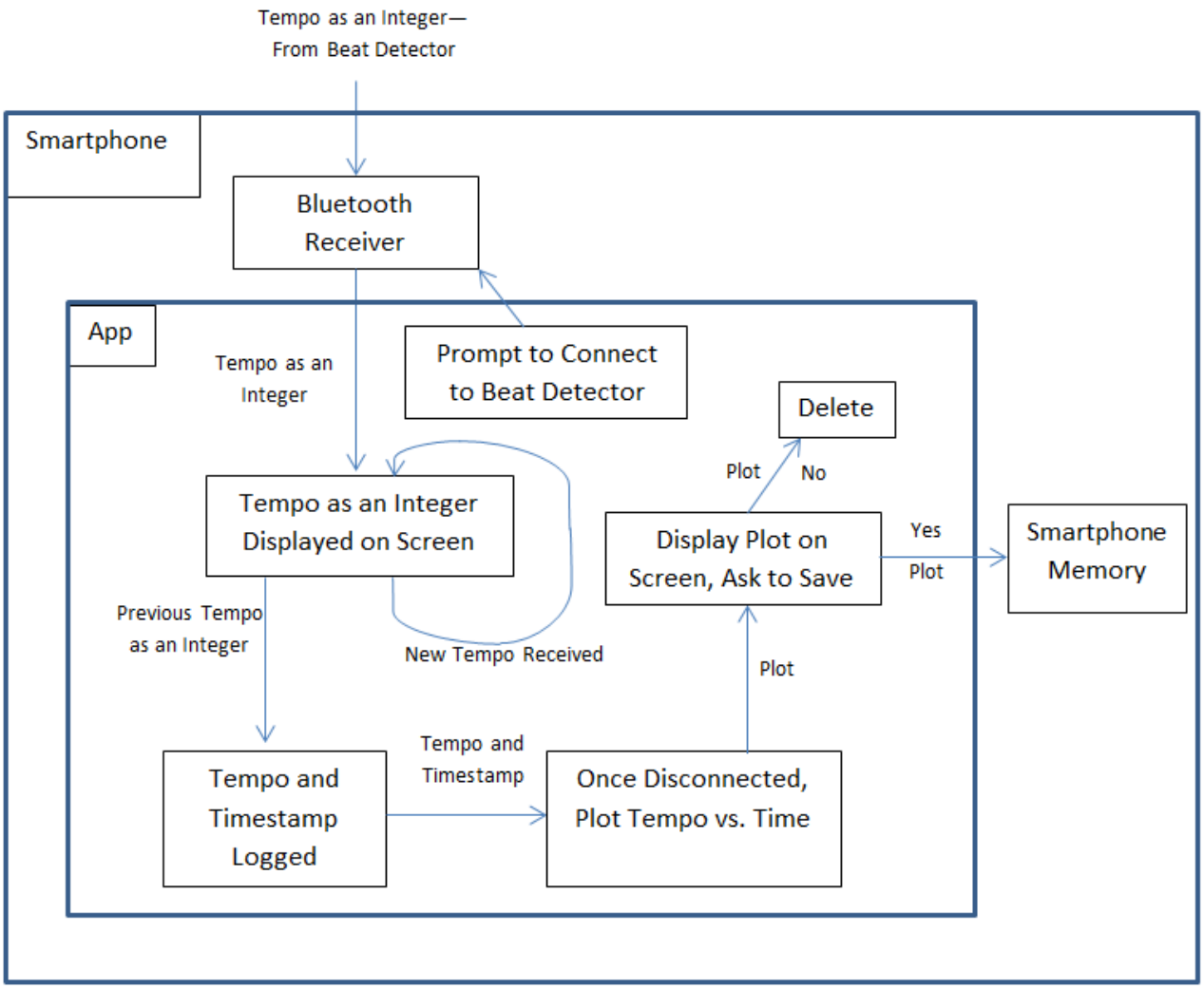


Fig.4: Block Diagram of App

5.1 Beat Detector Design

The beat detector is the component of the system that picks up whenever a conductor indicates a beat. The only input is the motion of the conductor's arm and the only output is the tempo which they are conducting represented as an integer. This output is then communicated to the smartphone via Bluetooth. A block diagram of the beat detector is found in Figure 3.

5.1.1 Beat Detector Construction

At the highest level, the beat detector is the IMUduino microcontroller housed inside of a plastic case, which is attached to an adjustable wristband. The IMUduino will be powered by coin cell batteries which will be connected to the unregulated voltage pins on the board. The specific voltage of these batteries has not yet been determined. The plastic case will house the IMUduino, will have a compartment for the battery, and will contain a switch connecting the battery and the board and functioning as an On/Off switch. At the next level, the active sensors on the IMUduino will be the MPU6050 accelerometer and gyroscope as well as the nRF8001 BTLE transmitter.

5.1.2 Beat Detector Function

The beat detector will be first powered on by setting the switch to the "ON" position. This will connect the battery to the IMUduino and will power the board. With the beat detector attached to the conductor's wrist, the conductor will begin conducting. The MPU6050 will pick up the movements of the conductor's arm and will feed these raw data into the beat detection and tempo calculation algorithm, stored on the IMUduino, and discussed below in sections 5.1.2.1

and 5.1.2.2. The algorithm will identify when a beat occurs and will calculate the tempo as an integer in beats per minute. This number will be sent to the nRF8001 BTLE transmitter and will be communicated to a paired smartphone running the app discussed in section 5.2.

5.1.2.1 Beat Detection

First, raw data arriving from the MPU6050 will be read into the algorithm. These will be acceleration data from the X, Y, and Z axes and hence will be sets of three numbers for changes of velocity in space as detected by the MPU6050. The first step of the algorithm is to take the magnitude of these data. That is to perform the following calculation: $\sqrt{X^2 + Y^2 + Z^2}$ for each set of data. It is far more reliable to detect a beat from analyzing the magnitude of these data as opposed to data from a single axis. Next, the magnitude data will be run through a threshold detection code. Pseudocode for this threshold detector is shown below in Figure 5.

```
%R=Magnitude of Acceleration

%Threshold Detection
if R(i)>Threshold & R(i-1)<Threshold
    Status=Threshold_Crossed
elseif R(i)<Threshold
    Status=Below_Threshold
else
    Status=Above_Threshold
end
```

Fig. 5: Pseudocode Representing Threshold Detection

The code essentially determines whether the current magnitude of acceleration is above a certain threshold and if the previous magnitude of acceleration was below it. This is indicative of the crossing of a threshold and only occurs when the velocity changes in a positive direction, that is, when the conductor's arm is speeding up. This way, the deceleration that occurs after indicating a beat is not also mistakenly identified as a beat, since any time the magnitude is below that threshold, the data is ignored. The value for this threshold will be a point in the acceleration data where a beat can be easily distinguished from background movements based on graphical data. Figure 6 shows a plot of the magnitude of acceleration over time as detected by the MPU6050 when conducting in a gentle manner at a tempo of 40 bpm. The sharp spikes at the beginning and end are anomalies in the data.

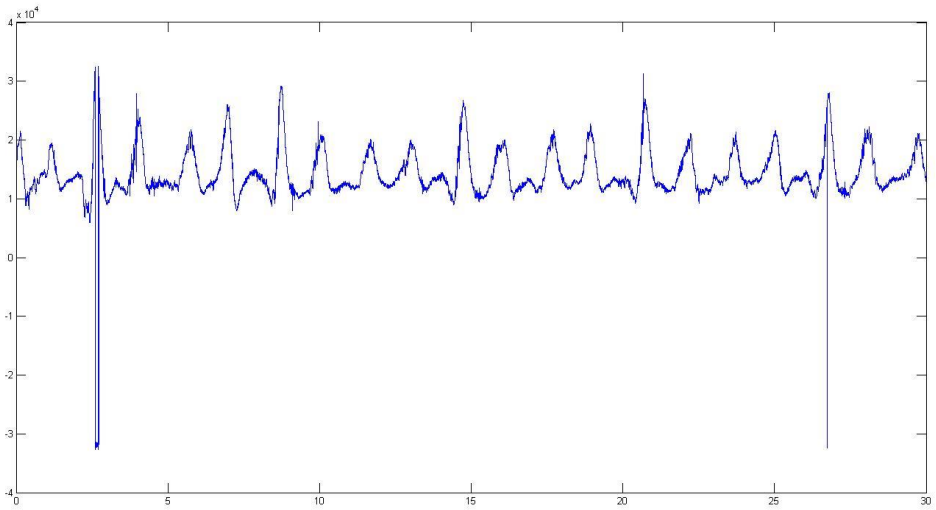


Fig. 6: Plot of Magnitude of Acceleration vs. Time with Gentle Conducting at 40 bpm as Detected by MPU6050 and IMUduino, Plotted with MATLAB

Note how it is extremely easy to distinguish beats from the background, especially the first beat of every measure, where the acceleration spike is a little bit higher. The threshold can hence be set at a relatively high level and is therefore far less susceptible to movements which are not indicative of beats. Each time the magnitude of acceleration crosses the threshold in a positive direction, an occurrence of a beat will be logged and sent into the next step of the algorithm: tempo calculation. Otherwise, the data will be ignored.

5.1.2.2 Tempo Calculation

The tempo calculation is rather simple at this point. First, when a beat is detected, the time at which it was detected will be logged. Then the tempo will be determined through the following simple calculation: $Tempo = \frac{60}{TimeNew - TimeOld}$, where *TimeOld* is initially set to 0. *TimeOld* is representative of the time the previous beat occurred and *TimeNew* is representative of the time the current beat occurs. Every time there is a new beat, *TimeNew* will be reset to the time that beat occurs, and *TimeOld* will be reset to the time the previous beat occurred, i.e. the last value of *TimeNew*. Once the tempo is determined based on the above calculation, it will be checked to see if it falls between the minimum and maximum tempos, 40 and 200 bpm respectively, as outlined in the design requirements. If it does, the tempo will be rounded to the nearest integer and sent to the nRF8001 to be transmitted to the smartphone app. Figure 7, below, shows pseudocode that outlines the tempo calculation process.

```
minTempo=40      %Minimum tempo
maxTempo=200     %Maximum tempo
timeOld=0        %Previous reference time initially set to 0
while beatDetector=ON;  %While the beat detector is on, run this loop
    if threshold=crossed
        beat=detected;          %Beat Detected
        timeNew=currentTime;    %Record time beat was detected
        TempoInit=60/(timeNew-timeOld); %Tempo Calculation
        if TempoInit > minTempo & < maxTempo %Is tempo between min and max?
            Tempo=round(TempoInit); %If it is, round it and store it
        end
        timeOld=timeNew;    %Now the time that the last beat that was
    end                     %detected will be used for reference in tempo
end                         %calculation instead of zero.
```

Fig. 7: Pseudocode Representing Tempo Calculation Process

5.2 App Design

Every time a valid tempo is determined through the algorithm above, it will be sent to a modified smartphone app via Bluetooth. This app receives the tempo as an integer and has two outputs: first it displays the current tempo on the phone screen and second, it logs the tempo vs time until conducting ceases and plots it. A block diagram of the app is found in Figure 4.

5.2.1 App Function

The app will be a modified version of a pre-existing app, nRF UART as produced by Nordic. Nordic released the source code for this app on their website with permission to modify. The app currently has a connect button, and once connected, it automatically receives any text-based data from any nRF BTLE transmitter and scrolls it on the screen as the information is

updated. This app will be modified so that the tempo as a single integer is displayed on the screen. When a new tempo is received, the old tempo will be replaced instead of simply scrolled up. It will also internally log each tempo with a timestamp and display a plot of tempo vs. time once the Smartphone is disconnected from the beat detector. The phone will display a prompt of whether or not to save the plot, and if "YES" is chosen, the plot will be saved to the phone's internal memory and can be transferred to a computer or other device for further viewing.

5.2.2 App Display and Layout

The app will follow similar display and layout to that nRF UART app. However, the UART logos and text will be removed. The connection procedure will not be changed. A large "Connect" button is displayed at the top of the screen. Once pressed, the user will be able to connect to nearby Bluetooth transmitters. This process is shown in Figure 8.

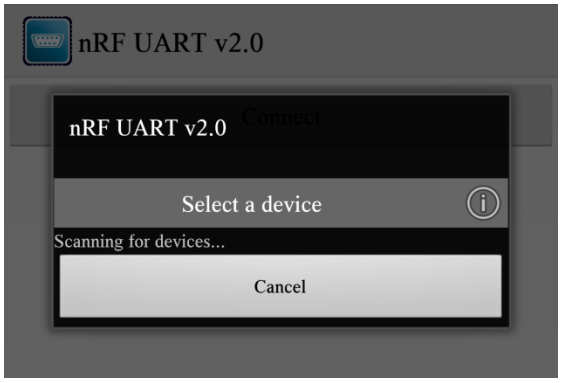


Fig 8: Process for Connection to nRF BTLE Transmitters in Current Version of App

Next, the app will begin to display incoming data. This incoming data will be tempo as an integer and will be displayed in the center of the screen in large font. Whenever a new tempo arrives, it will simply replace the previous tempo in the center of the screen. The format of this tempo display is shown in Figure 9.

Fig 9: Display of Tempo Data in Center of Screen in Current Version of App

Once disconnected from the beat detector, the app will plot the logged tempos vs. time. This plot will automatically appear and fill the screen, with a prompt asking whether or not to save the app also appearing in the center of the screen. If the user selects “YES”, then the plot will be saved to the smartphone’s memory as a Portable Network Graphic (PNG) file. If the user selects “NO,” then it will be deleted. While the modifications for the tempo display are complete, the code for the plot has not yet been implemented and is one of the primary focuses for upcoming work.

5.3 Future Work

This project is far from complete, but upcoming work entails further testing, creating the case for the beat detector, finding the proper battery and switch and connecting them to the IMUduino, implementing the algorithm in the Arduino IDE, and completing the app modifications. There is much to be done, but a prototype is almost ready for testing. Once testing begins, there will be a great deal of debugging and adjusting components and algorithms. However, by March 2016, the final version of the system will be complete and student conductors everywhere will have a new tool at their disposal to help them learn.

6. FINAL DESIGN AND IMPLEMENTATION

The final design of the system differed mainly from the preliminary in that the IMUduino did not have sufficient memory onboard to store both Bluetooth protocols and the algorithm for detecting beats and calculating tempo. Hence the component worn on the user's wrist is no longer called the "beat detector" and will now be dubbed the "movement sensor," and the algorithm were implemented on the app. A complete breakdown of the final version of the system is shown in the block diagrams below in Figures 10 and 11.

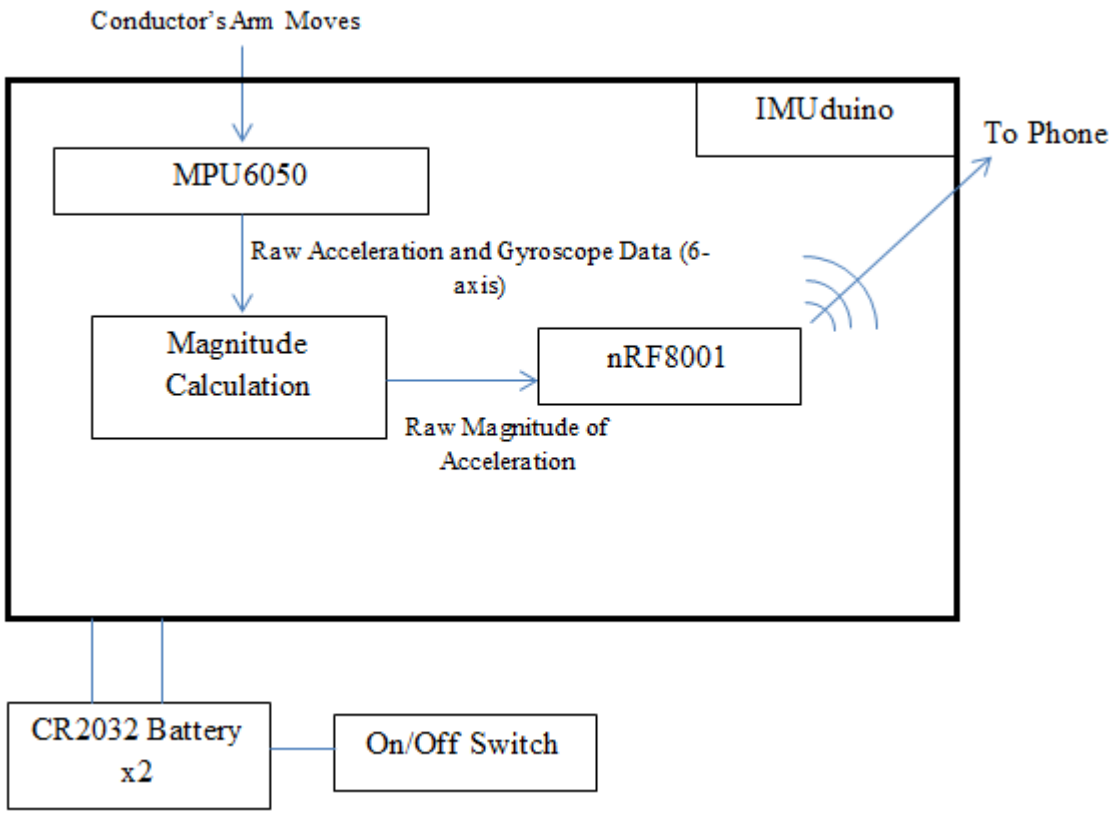


Figure 10: Block Diagram of Movement Sensor Worn on User's Wrist

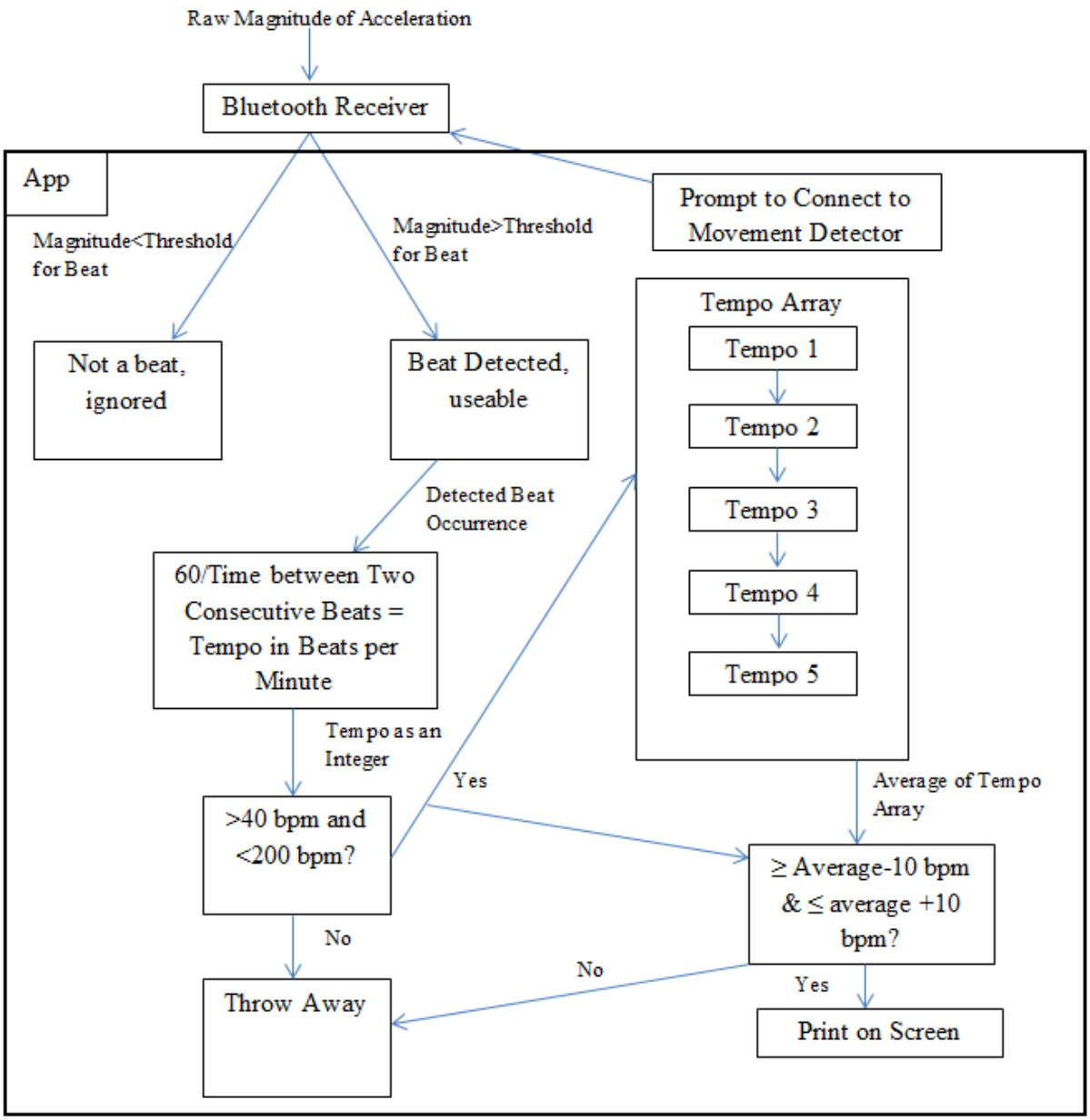


Figure 11: Block Diagram of Final Version of Beat Detection and Tempo Calculation Algorithm as Performed on Smartphone

6.1 Movement Sensor Design

The movement sensor is housed in a case mounted on a wristband which is worn on the user's wrist. This movement sensor picks up the movement of the conductor's arm, calculates the magnitude of acceleration, and streams this data to the smartphone app. A block diagram detailing the function of the movement sensor can be found in Figure 10.

6.1.1 Movement Sensor Construction

The movement sensor consists of the IMUduino microcontroller, with its integrated MPU6050 accelerometer/gyroscope and nRF8001 Bluetooth Low Energy transmitter, a DPST switch, and two CR2032 coin cell batteries. These components, with the exception of the batteries, are housed within a 3D-printed plastic case. This case was originally designed for an Arduino Nano, and its dimensions were modified in Google SketchUp to fit those of the IMUduino [2]. This case is displayed below in Figure 12.

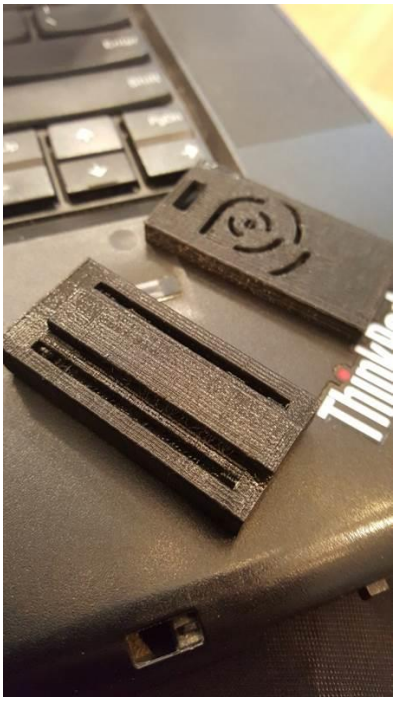


Figure 12: 3D-Printed Arduino Nano Case Redesigned to Fit IMUduino

The batteries were housed in standard 20mm coin cell battery cases. A pair of CR2032 coin cell batteries was picked because these batteries offer the largest capacity for readily-available coin cells. Each battery provides 3V and 240 mAh [7] for a total of 6V and 480 mAh. This gives a battery life of approximately 9 hours. When connected to the regulated voltage pins of the IMUduino, 43 mA of current was drawn due to the Bluetooth transmitter constantly transmitting, so high-capacity batteries were necessary. The entire assembly, including the case and batteries, was mounted on a wristband, specifically the Voguestrap TX51012BK Allstrap 16 mm Black Nylon Adjustable Sport Watch Band. During testing, it was found that the clip on the back of the 3D-printed case was not sufficient to secure the case to the wristband during use, so a bit of tape was added to ensure it remained attached to the wristband. Two working versions were constructed. The first was a prototype, without a case or battery cases, contained in a static-proof plastic bag and is displayed in Figure 13. This version was used for testing. The second

version was the final version, including the battery cases, minimal loose wiring, and a containing case. It is displayed in Figure 14. A circuit schematic is shown in Figure 15. The IMUduino is receiving power at the + and – Vin pins.

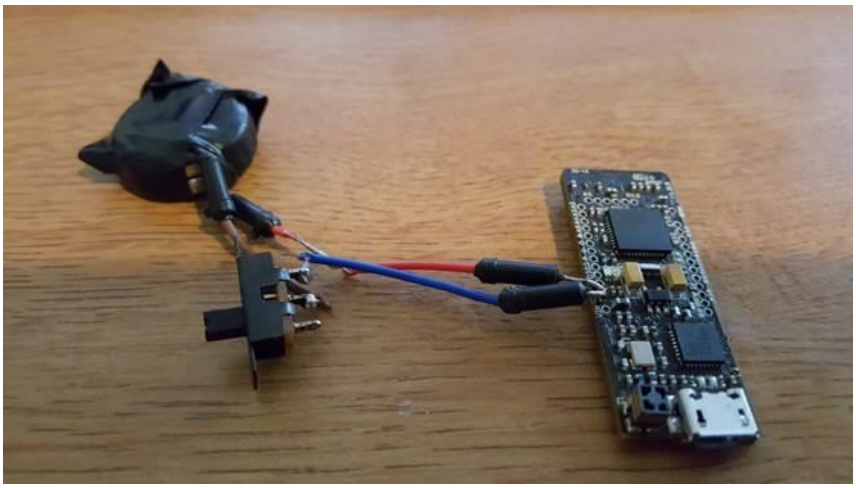


Figure 13: Prototype Version of Movement Sensor

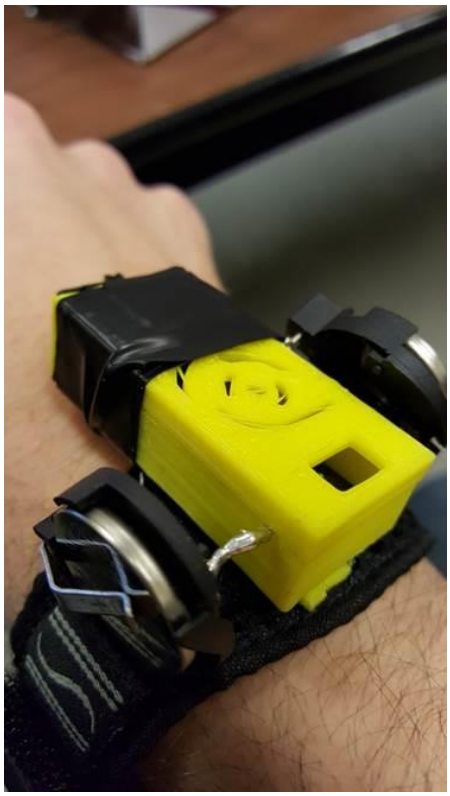


Figure 14: Final Version of Movement Sensor

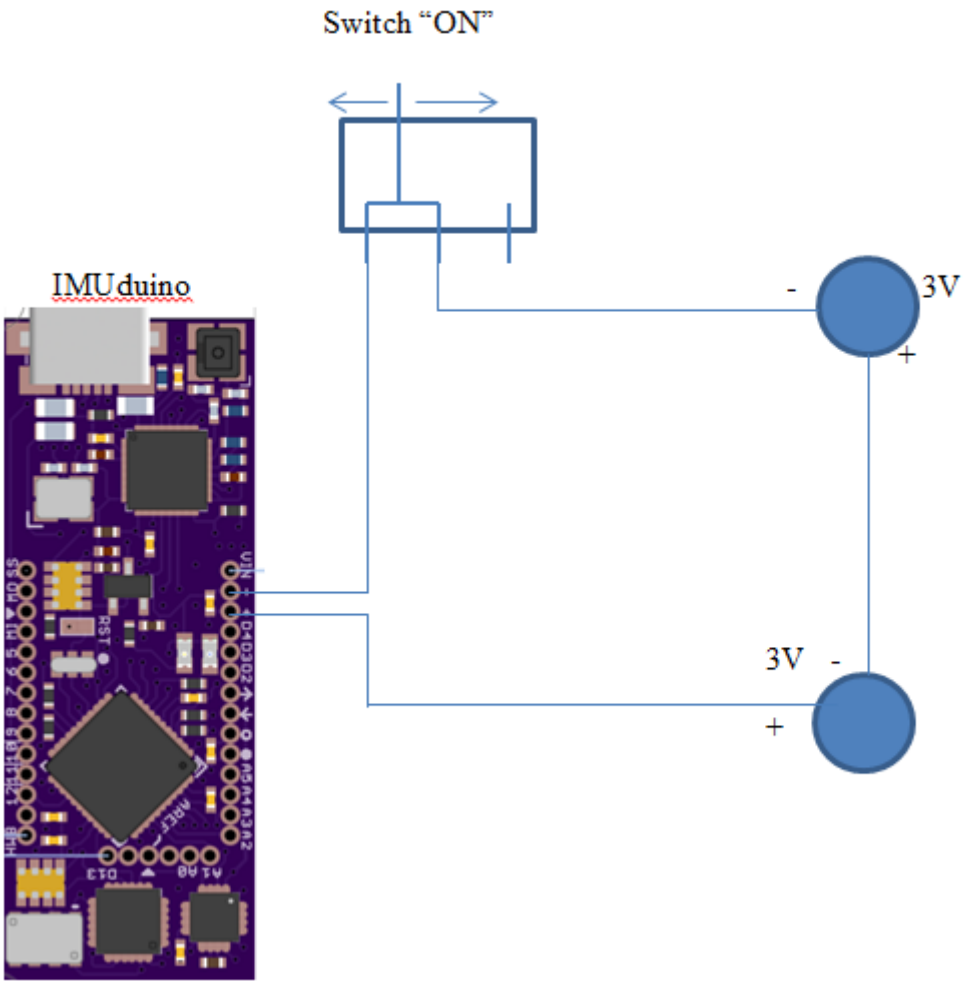


Figure 15: Circuit Schematic for Final Version of Movement Sensor

6.1.2 Movement Sensor Function

The movement sensor takes in the movement of the conductor's arm as an input. All the conductor needs to do is ensure that the switch is on and begin conducting. An orange LED on the IMUduino will light up, indicating that the board is receiving power. The conductor's movement is picked up by the MPU6050. The MPU6050 feeds raw acceleration data in all three axes into a simple calculation for the magnitude of acceleration: $\sqrt{X^2 + Y^2 + Z^2}$. Performing this

calculation produces a single number instead of three values and is sufficient information for detecting beats. Beats prominently stand out in a plot of magnitude of acceleration vs. time, and can be seen in Figure 6. This magnitude of acceleration is then streamed via the nRF8001 BTLE transmitter to the smartphone. The code for this operation, including the Bluetooth protocols and magnitude calculation, can be found in Appendix A.

6.2 App Design

The accompanying app receives the magnitude of acceleration as an input and outputs tempo as an integer. Within the app is code which detects beats, calculates tempo, and displays a single tempo in the center of the screen and updates roughly once per beat. A block diagram detailing the function of the algorithm on the app can be found in Figure 11.

6.2.1 App Function

App function can be broken down into two primary functions of the algorithm: beat detection and tempo calculation. Each of these functions originally was intended to be implemented on the IMUduino. Unfortunately, the IMUduino did not have enough memory onboard to store this code in addition to the Bluetooth protocols required to make the system wireless. Thus, the algorithm that had been successfully implemented in the Arduino IDE, when the system used a serial cable for testing, needed to be transferred to the Android app using Android studio. In both of these functions, the way data from the Bluetooth transmitter is interpreted by the app needs to be understood. Data arrives in packets of several bytes at a time,

and is stored in a byte array. This byte array was first converted into a string and then parsed for integers. At this point, instead of a byte array being fed into the functions, the data now was a stream of integers—specifically the magnitude of acceleration. At this point, the threshold check was performed followed by the tempo calculation if the threshold was crossed. After successfully calculating a tempo as an integer, this integer was converted back into a string so it could be displayed on the screen.

6.2.1.1 Beat Detection

The beat detection function proved itself strong and effective throughout testing. It already was fairly simple to begin with, so there was no need to change it as the project evolved. This was the same algorithm developed in the preliminary design as represented in pseudocode in Figure 5. The actual code can be found embedded in both versions of the algorithm, in Appendix C and Appendix D. It is a fairly simple threshold detector. If the magnitude of acceleration is greater than the adjustable threshold, it is most likely a beat. Once the threshold has been crossed in this direction, a flag is set to 1 to let the program know that a beat has been detected. At this point, the tempo calculation function goes to work, explained below. Once the magnitude of acceleration is determined to be less than the threshold, the flag is reset to 0 and the program knows that any data that comes in is not to be used in calculation.

6.2.1.2 Tempo Calculation

The tempo calculation function occurs only as a valid beat is detected. Two versions of this function were developed. The variables used in both versions are outlined in Appendix B. The code for the first version can be found in Appendix C and the code for the second version can be found in Appendix D. The block diagram in Figure 11 details the function of the second version.

The first version uses a simple finite impulse response filter. Pseudocode for this algorithm can be found in Figure 7. Once it is determined that the magnitude of acceleration is greater than the threshold and therefore the motion of the conductor's hand is indicative of a valid beat, the algorithm logs the time the threshold was crossed and calculates the tempo in beats per minute using the following formula: $Tempo\ (bpm) = \frac{One\ Minute}{timeA - timeB}$, where *timeA* is the time when the threshold was crossed and *timeB* is initially set to zero. At this point, the program assigns *timeB* the value of *timeA* and checks if the tempo is within the design requirements. If it is, it averages the tempos of the last two beats and prints this tempo on the screen. Next, it stores the tempo to be used in the next averaging calculation. Lastly, the script sleeps for 300ms, as a tempo of 200 bpm (the max in the design requirements) is a beat once every 300ms. This is a method of avoiding noise in the calculations. Data is still incoming from the movement sensor, but the phone will not read it for 300ms. This algorithm essentially used a sliding window average of two beats to calculate tempo. It adjusted to changes in tempo very quickly, but suffered from poor precision and an occasional doubling of tempo as the threshold was sometimes crossed twice by sweeping movements of the conductor's arm. To solve these problems, a second version of the algorithm was developed.

Much of the initial operation of the algorithm is the same as in the first version, but the calculation and averaging of the tempo is performed much differently. In this version, instead of an FIR filter, the noise removal is done through the averaging of an array of the previous five tempos. Pseudocode for this algorithm is found in Figure 16. As in the previous algorithm, once the threshold is crossed, the time is logged, and the initial tempo calculation takes place where, as above, it is calculated using $Tempo\ (bpm) = \frac{One\ Minute}{timeA - timeB}$. After this number is calculated, the time is reset in preparation for the next beat. Next the tempo is checked to see if it is within the design requirements and less than or equal to double the previous tempo. This check filters out any errors that result from crossing the threshold twice due to a sweeping arm movement. If these checks are passed, the tempo as calculated is added to an array called *tempoArray*, and assigns *tempoB* the value of *tempoA*, as in the last version. *TempoArray* holds the values of the last five tempos. The current tempo is compared to the average of this array, and if it is within 10 bpm of the average, it will be printed on the screen. After printing, the script sleeps for 300ms. The length of *tempoArray* is kept at 5 with an index variable which resets to 0 every five beats. This algorithm yielded far more precise results than the last and eliminated the doubling errors seen before. However, it takes five full beats to build up *tempoArray* with valid tempos, so when the tempo changes, it can take up to five seconds to display as opposed to less than two for the old version. However, precision and accuracy in the displayed tempo outweighs quick response time.

```
timeB=0
tempoArray=[60,60,60,60,60]
if magnitude>threshold && flag==0 %if beat detected
    flag=1 %indicator of above or below threshold
    timeA=now %Log time threshold crossed
    tempo=OneMinute/timeA-TimeB %calculate tempo using difference in times
    timeB=timeA %reset time for next beat
    if tempo1 meets design requirements && <= 2*previous tempo
        tempoArray[i]=tempo %add tempo to tempoArray
        increment
    end
    average=(sum of tempoArray)/5 %calculate average of tempoArray
    if tempo is within +/- 10 bpm of average
        print it onscreen
    end
    sleep(300) %Fastest tempo is 300ms between beats
end
elseif magnitude<threshold %No beat detected
    flag=0
end
if number of elements in tempoArray is > 5 %keep tempoArray at 5 elements
    reset index to 0
end
```

Figure 16: Pseudocode Representing Final Version of Beat Detection and Tempo Calculation Algorithm

6.2.2 App Display and Layout

The app for this project is a modified version of the open-source nRF UART app, which was designed for interfacing with nRF Bluetooth transmitters. The app was modified as seen in Figures 8 and 9 to display a number in the center of the screen and have it replace itself instead of scrolling up, as was the default with nRF UART. The app's design was overhauled. The app was renamed to "Capstone Tempo App", the nRF UART logos were removed, the color scheme and icon were replaced, and a background image was added. The screen will now stay on for as

long as the app is open as well. A screenshot of the new layout of the app can be seen below in Figure 17.

Figure 17: New Layout of App

The connection process is identical to the original app, where the “connect” button is pressed and a list of available Bluetooth devices is displayed. The user picks “UART” to connect to the IMUduino. This process is shown in Figure 18.

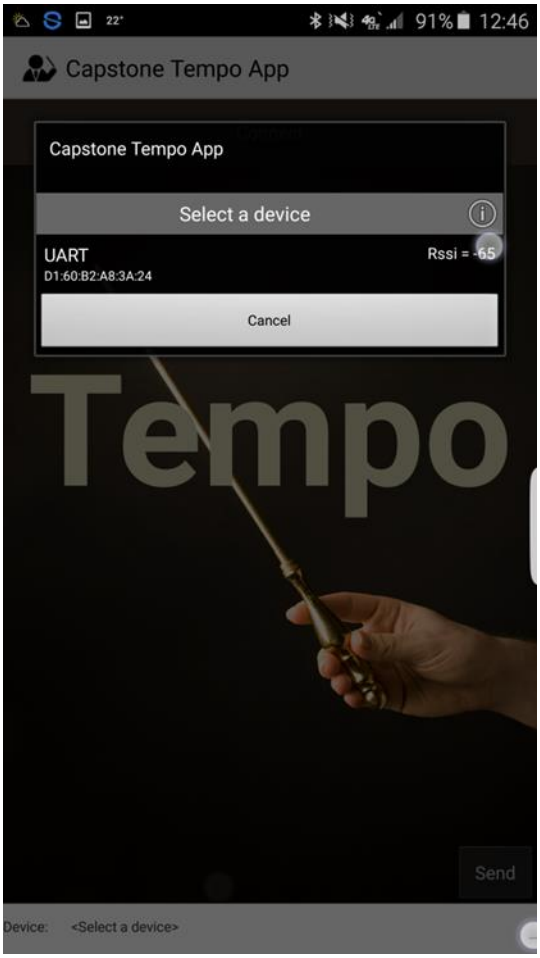


Figure 18: Bluetooth Connection Process within App

Once the user has connected and the movement sensor is turned on, the app will begin receiving data from the sensor and will display tempo once the user begins conducting and the tempo array has been filled. Once the user wishes to stop, all they need to do is press the "disconnect" button and the phone will stop receiving data. The app in operation can be seen below in Figure 19.

Figure 19: App in Operation, Showing a Tempo of 62 BPM

7. PERFORMANCE ESTIMATES AND RESULTS

Within this section, the expected performance of the preliminary design will be discussed and compared with the performance of the final design. The level of satisfaction of the design requirements will be discussed. Finally the testing process will be described in detail and recommendations to improve performance will be addressed.

7.1 Preliminary Design Performance Estimates

The preliminary design was expected to produce a tempo but not with a particularly excellent degree of precision. With the algorithm being as simple as it was, it was not expected to deliver data within the design requirements of the displayed tempo—being +/- 5 bpm of the actual tempo—however erroneous measurements were not expected either. This algorithm was expected to fit on the IMUduino with enough space left over for Bluetooth protocols, with the app receiving only the tempo as it came in. The IMUduino was not expected to draw a particularly high amount of current and thus would be able to be powered for a long time with a simple coin cell. It was expected that the Bluetooth transmitter would transmit consistently and without connection issues for as long as was necessary. The app was expected to display the tempo easily and deliver a plot of tempo vs time at the end of the conducting session as well.

7.2 Performance Results

In reality, not every aspect of any project works perfectly as designed, and this project was no exception. The first version of the algorithm was indeed not extremely precise and yielded fluctuations in tempo of around 20 bpm. It also was subject to occasional doubling of tempo when the threshold was crossed twice during a sweeping movement of the conductor's arm. This was fixed through a second version of the algorithm which, when implemented reduced the fluctuations in tempo to around 10 bpm and completely removed erroneous measurements where the tempo suddenly doubled. Unfortunately, through experience, it was determined that there was not sufficient memory onboard the IMUduino to store both the algorithm and the Bluetooth protocols, even after taking several measures to slim down the size of the code. This meant the best course of action was to implement the algorithm on the Android app, where storage space would not be an issue. The IMUduino drew a surprising amount of current at 43 mA since the Bluetooth transmitter was always broadcasting data. This significantly impacted the battery life of a pair of small coin cell batteries such that instead of the 96 hours of battery life as preferred in the design requirements, the two CR2032s would only last for about 9 hours. A major problem which could not be solved in time was the fact that the Bluetooth transmitter on the IMUduino would simply stop transmitting after several minutes of operation. The app would stop receiving data from the IMUduino, and even after successfully reconnecting, it would still not display any new data. The power LED remained on the whole time. Only if the board was turned completely off, given some time to rest, and turned back on would transmission resume. The longer the IMUduino was given to "rest," the longer it would work properly before dropping the connection again. This issue was reproducible on multiple boards. Lastly, there was not enough time to implement a graph of tempo vs. time as detailed in the

preliminary design section. Between redesigning the algorithm, adjusting to the setback of insufficient memory on the IMUduino and ensuring prototypes were tested, the plot became less of a priority over the course of the term.

Despite these issues, most of the design requirements outlined in section 3 were partially if not completely met. First, the movement sensor weighs less than a wristwatch and measures 5.5 x 2 x 1.5 cm. It meets the design requirement of being small, lightweight, and wireless. Next, the algorithm is able to prevent gestures that are not beats from interfering with the tempo calculation. By implementing a series of checks and averages within the algorithm, it is able to effectively filter out noise, meeting this design requirement. Next, the algorithm is able to display a tempo within five seconds of the start of conducting. With some slower tempos, it may take the full five seconds, but regardless, this design requirement was met. Sometimes not every beat will be read into the algorithm due to an error being filtered out, but on average the tempo will consistently update on every beat. Therefore this requirement was partially met. The tempo algorithm is able to calculate tempos from 40-200 bpm without any issues. In fact, when the algorithm is allowed to calculate tempos beyond 40 or 200 bpm, it successfully does so from 20-220 bpm, exceeding this design requirement. However, the accuracy of tempo measurements was not within +/-5 bpm. The tempo fluctuates up to +/- 10 bpm but the system is still highly functional, partially meeting this design requirement. The tempo is displayed in very large numbers on the screen of the phone, making it easy to see from a distance. They remain in place, not scrolling, and are replaced each time a new tempo is calculated. The app is in general very simple and easy to use, therefore this design requirement was also met. The app does not display a graph of tempo over time after the conclusion of conducting, however this is the only design requirement that was not met.

7.2.1 Testing Process

The testing process was fairly straightforward. First, once the first working algorithm was implemented on the first wireless prototype, it was taken to Professor John Cox and Professor Tim Olsen of the Union College Music Department. They were asked to conduct in various styles and at various tempos with the prototype and then after the testing session was over, they were given a questionnaire to fill out regarding their experience with the system. This questionnaire is attached as Appendix E. The questionnaire asked for them to rate various aspects of the system's performance on a scale of 1-10. They were also asked to suggest any additional features they as conductors might appreciate on the system and if they had any further ideas. Their feedback was critical for developing the second version of the algorithm and sparking ideas for future work. Images of the testing with Professor Cox are shown below in Figure 20.

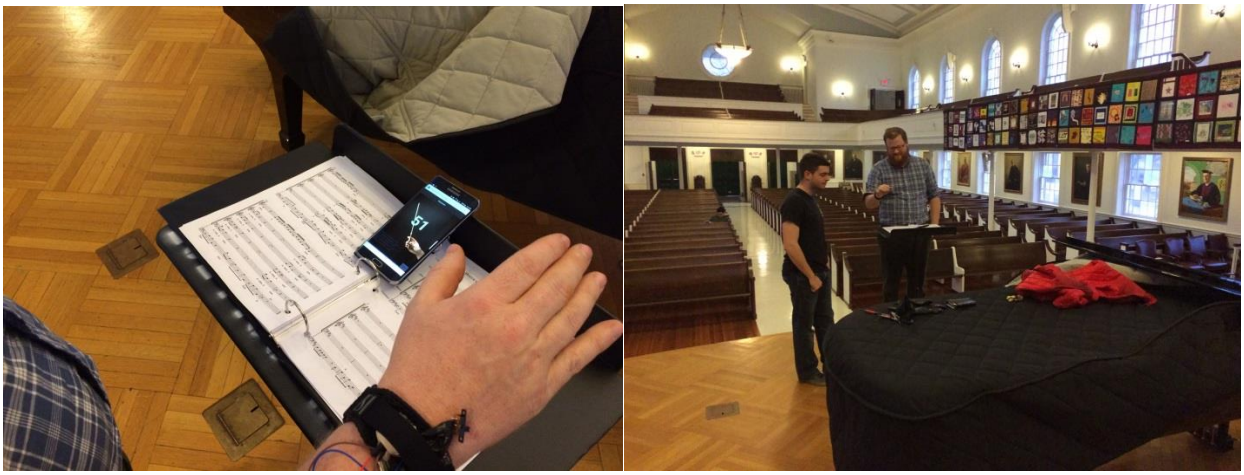


Fig. 20: Testing Prototype with Professor John Cox

No way to test the exact accuracy of the system was found or implemented. Ideally, this would have consisted of a mechanical device that moved up and down with sufficient acceleration to cross the threshold and with programmable precision, yet at very low frequencies. The device that acts the most like this is a mechanical metronome. However, mechanical metronomes operate based on a precisely placed weight on an arm that moves back and forth. If the device were attached to the arm, lightweight as it is, it would throw off the accuracy of the metronome. A mechanical testing device could have been built, but there simply was not sufficient time in the term. Thus, the precision of a professional conductor was regarded to be the most reliable testing method.

7.3 Improvements

There are four main improvements that could be implemented on the system. First, a higher capacity battery should be used instead of the two coin cells. In order to achieve 96 hours of battery life, a 6000 mAh battery would be needed [4]. The size and weight of such a battery is completely impractical, therefore the best solution is to compromise between battery life and battery size. A smaller 1000 mAh battery will give approximately 16 hours of battery life which will be sufficient for several long sessions with the system, but will not achieve the 96 hour goal as previously mentioned. Another option is to use a rechargeable battery, which eliminates the hassle of changing and buying new batteries. The second primary improvement is to redesign the algorithm such that it accurately detects tempos within +/- 5 bpm of the actual tempo. Focusing on minimizing the tempo fluctuations and perhaps investigating the way the MPU6050 takes in data will help improve the algorithm. The third improvement to be made would be to fulfill the

design requirement of adding a plot of tempo vs. time. This plot would be shown on the screen after the “disconnect” button is pressed and could be saved as an image file, with the ability to be moved from the phone later. Lastly, the system should be tested with a precise mechanical device, even if the device needs to be built from scratch. Testing the absolute accuracy of the system is critical to understanding whether or not it can reflect tempos with the precision of +/- 5 bpm in the first place.

8. PROJECT SCHEDULE

Initially a schedule was developed in the spring of 2015. This schedule emphasized having a working prototype ready by week 5 of fall term 2015. This was so that the majority of time for the project was spent testing and debugging. Tasks were grouped into major categories and given deadlines. The resulting timeline can be seen in the Gantt chart below in Figure 21.

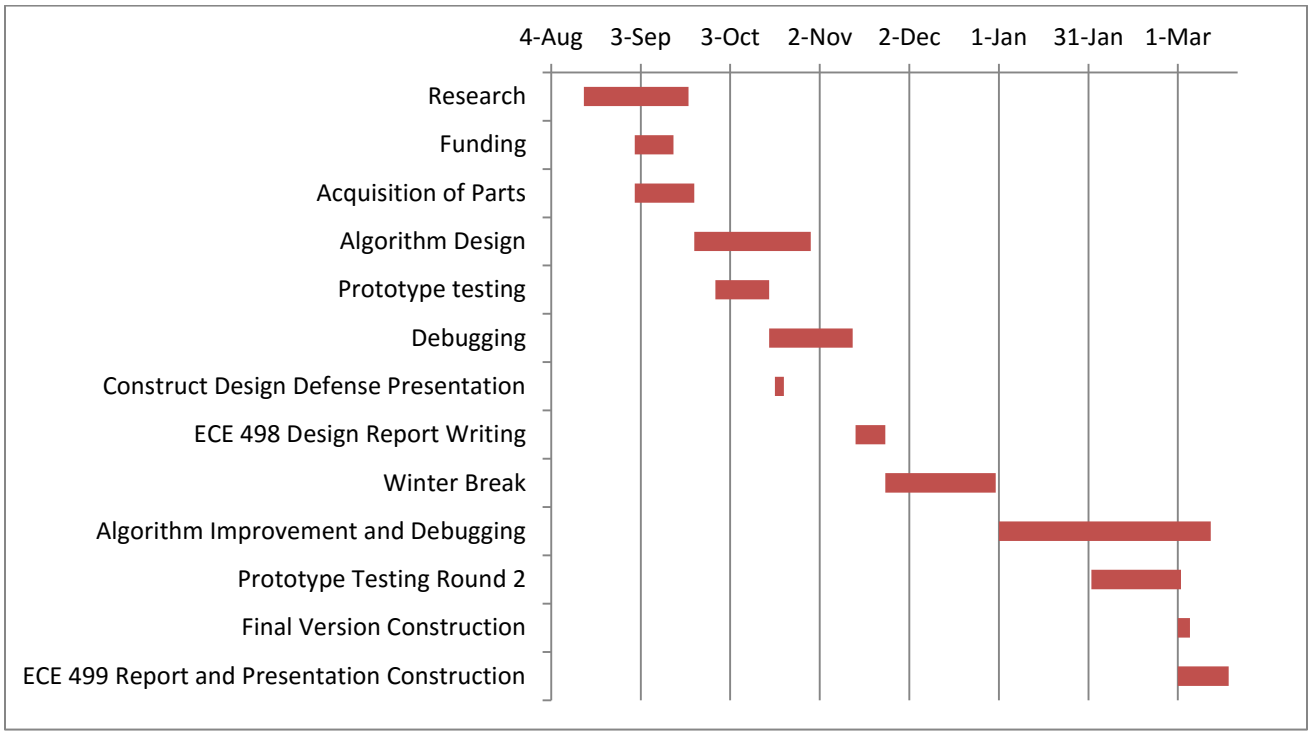


Figure 21: Project Timeline as Planned in Spring 2015

While it was a good idea to set tight deadlines to encourage a fast project tempo, the timeline did prove itself fairly ambitious. Supplies were not received until well into October, and much more time was needed to understand how to manipulate the data from the IMUduino, learn how to use Java and Android Studio and modify the app. Hence an algorithm was not prepared until the end of winter break. Once returning to school, it was determined that the algorithm

would not fit on the IMUduino, so more time was used in attempting to slim it down and then eventually translate it to Java and implement it on the app. After the first round of testing, the algorithm needed to be improved. This meant that while a better algorithm would be ready for the final version, the testing and subsequent improvement of a second prototype would not be feasible with the time left. Fortunately, the code did not require nearly as much debugging time as expected, which allowed for some time to be made up. In addition, several weeks' worth of "catch-up time" was allotted in the schedule and allowed for exactly that to happen. If it was known that the app would be the final platform for the algorithm, a few weeks' time could have been saved. Regardless a final working version was ready by week 9 of winter term 2016. The Gantt chart in Figure 22 displays the actual project timeline.

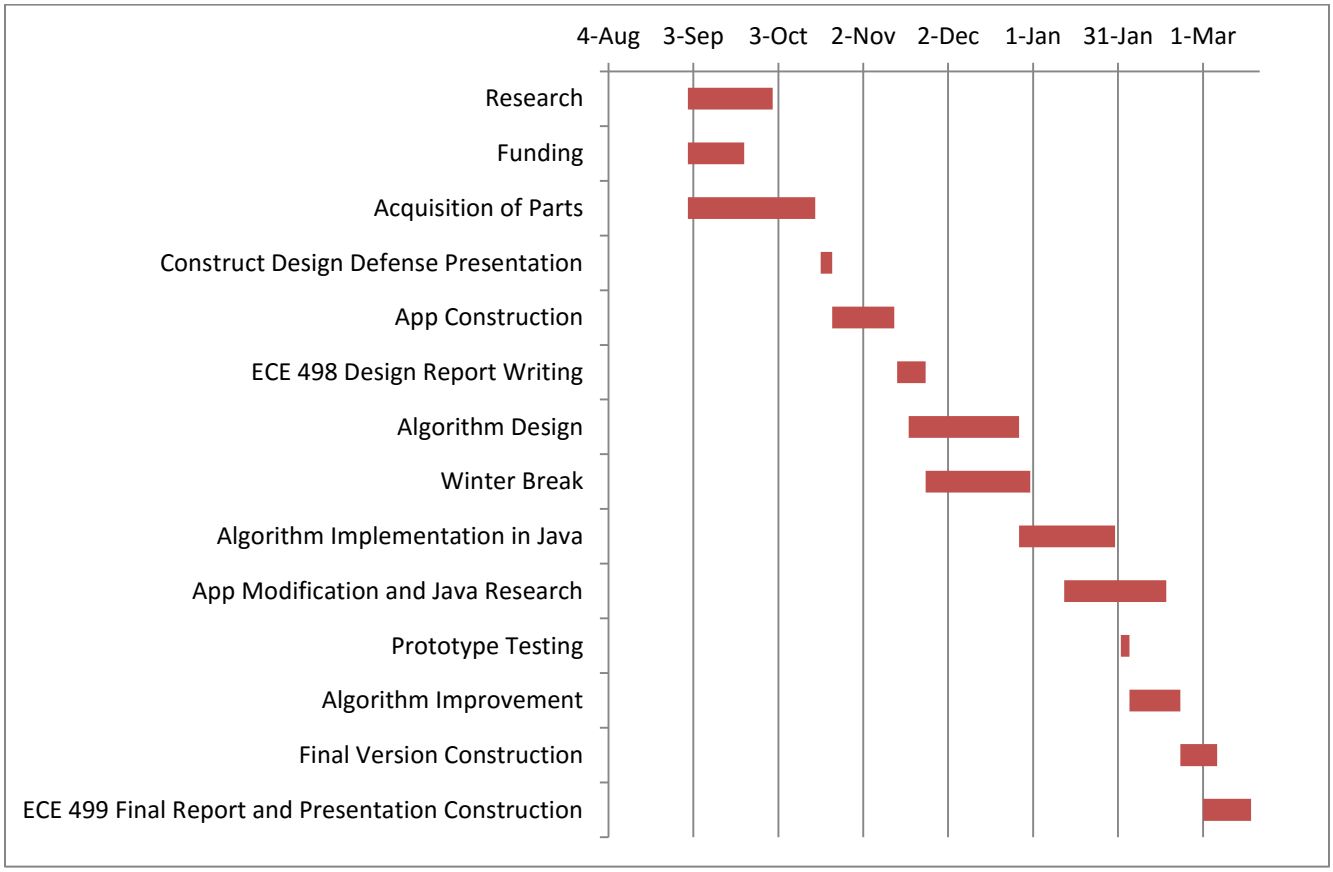


Figure 22: Actual Project Timeline

9. COST ANALYSIS

After submitting two applications for a student research grant (SRG), a budget of $394 was approved for the project. This budget was for all design options, including the other option for a microcontroller: the MetaWear RG board.

Table 1: SRG-Approved Budget with All Design Options

Component	**Quantity**	**Cost per Unit**	**Total Cost**
IMUduino board	2	$80	$160
MetaWear RG Board	3	$55	$165
Wristband	3	$10	$30
Funding for 3D printed case	2	$10	$20
Batteries	5	$2	$10
Bluetooth adaptor for PC	1	$9	$9
		Project Total	$394

The total cost of the project based on this budget was $394, however some components that were not factored into this budget were purchased after it was approved, such as the DPST switch, more than 5 coin cell batteries, two coin cell cases, and extra 3D printed cases for the IMUduino. If these are added to the budget, the total cost is increased to $416. The cost of a single movement sensor unit is approximately $100, with batteries included.

10. USER MANUAL

This section provides detailed instructions on how to assemble, operate, and maintain the system. It is assumed that the Arduino sketch and all necessary libraries are on the user's computer and that the app is available for download on the Google Play Store.

10.1 Assembly and Program Setup

1. Inspect the IMUduino board to ensure that there are no loose or broken solder connections and no bare wires. Compare to the circuit schematic in Figure 15. Always handle the board by the edges, and ensure you have discharged all static electricity built up on your body by touching a grounded conducting object such as a door handle.
2. Insert one CR2032 3V coin cell battery into each battery case. The "+" side of the battery should be visible when inserted properly into the case. Ensure the contacts are maintaining significant pressure on the battery and that it is securely seated in the battery case by shaking the case back and forth a few times.
3. Test that the IMUduino receives power by moving the switch to the ON position. If the orange LED on the board lights up, as seen in Figure 23, the board is receiving power. If the LED does not turn on, check the batteries to ensure they are seated properly, ensure the contacts are maintaining pressure with them, and that the "+" side of each battery is up. Check the connections to the board and switch as well.

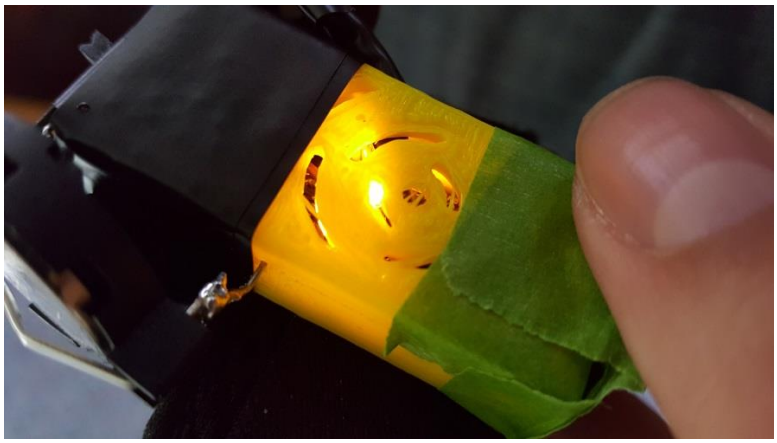


Figure 23: Power LED ON

4. Once it is clear that the IMUduino is receiving power, turn the switch back to the OFF position.
5. Connect the IMUduino to your computer with a male-male micro-USB to USB cable as seen in Figure 24. Once connected, the power LED will light up again since the board is receiving power from the USB cable. DO NOT turn the switch ON.

Figure 24: Male-Male Micro-USB to USB Cable

6. Open the Arduino IDE, select “board” from the Tools menu, then select “Arduino Leonardo.” The IMUduino is an Arduino Leonardo clone.
7. Select “Port” from the Tools menu, then select whichever COM port is listed with the Arduino Leonardo.
8. Load the sketch “bluetoothMag2,” compile it, then upload it to the IMUduino. Ensure the message “Finished uploading” appears in the IDE.
9. Disconnect the IMUduino from the computer by simply unplugging it.
10. Place the IMUduino and all wires inside the bottom half of the case, with the battery cases on the outside for accessibility and the switch in the large opening in the front, as seen in Figure 25.

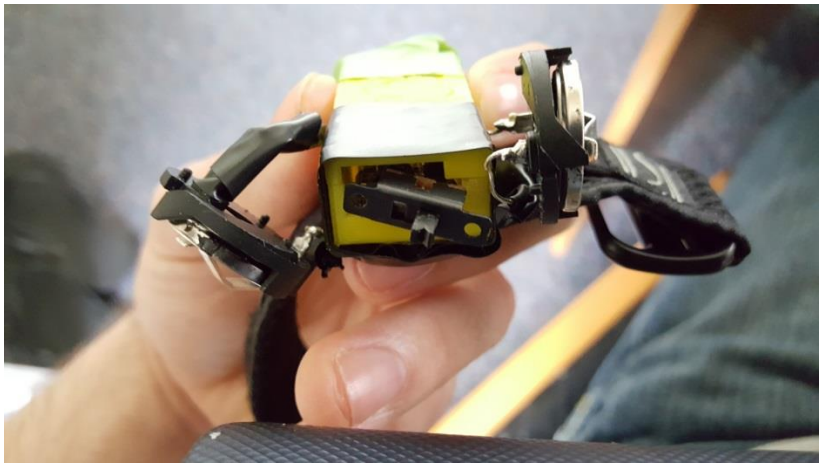


Figure 25: Battery Cases and Switch are Located outside the Main Case

11. Snap the top half of the case onto the bottom half and secure with a strip of tape.
12. Slide the clip on the bottom of the case onto the wristband so the case sits in an orientation where the batteries are closer to you and the case is on top of the wristband, as seen in Figure 14. Secure the case to the wristband by placing a strip of tape on each side of the clip.

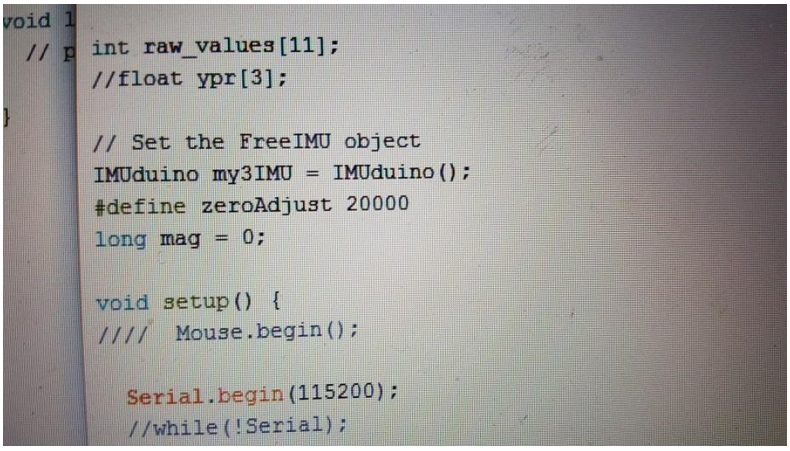
13. Secure the wristband to your wrist, adjust with the Velcro strap, and ensure the entire movement sensor assembly is comfortable.
14. On your Android smartphone, open the Google Play Store, search for “Capstone Tempo App” and download it.

10.2 Initial Operation and Usage

1. Turn the switch on the movement sensor to the ON position. Ensure the LED lights up.
2. Open the app, and press the “Connect” button, seen in Figure 17.
3. A list of nearby Bluetooth devices will appear. Look for “UART” and when it appears, select it. If it does not appear, check to see that the IMUduino is ON and press the “Scan” button to restart the scan. See Figure 18 for reference.
4. Once connected, begin conducting. The app will take approximately five seconds to begin displaying tempo, but will update once per beat after. The app will automatically keep your phone screen on.
5. Once you are finished conducting, press the “disconnect” button, or turn the switch on the movement sensor to the OFF position.

10.3 Troubleshooting and Adjustments

1. If the Bluetooth connection suddenly drops and the numbers stop updating, disconnect and turn the movement sensor OFF for at least one minute. Turn it back ON and reconnect to resume conducting. Also consider replacing the batteries.
2. If you are having trouble getting the system to pick up your movements, consider these things:
    a. In order for the system to pick up beats as consistently as possible, conducting must be done with relatively deliberate movements. A smooth, *legato*-style of conducting will be difficult for the system to pick up, unless long sweeping movements are used.
    b. The system is designed to detect tempos from 40-200 bpm. If you are conducting slower or faster than these limits, the system will not display the tempo.
3. If you wish to adjust the sensitivity of the accelerometer, follow these steps:
    a. Ensure the movement sensor is off and open the case so you will be able to connect it to a computer.
    b. Follow steps 5-7 in section 10.1 and load the "bluetoothMag2" sketch on the Arduino IDE but do not compile or upload it.
    c. Within the sketch, locate the constant "zeroAdjust." See Figure 26.



```
void l
  // p int raw_values[11];
      //float ypr[3];
}
      // Set the FreeIMU object
      IMUduino my3IMU = IMUduino();
      #define zeroAdjust 20000
      long mag = 0;

      void setup() {
      ////  Mouse.begin();

        Serial.begin(115200);
        //while(!Serial);
```

Figure 26: Location of "zeroAdjust" in Sketch "bluetoothMag2"

d. This is the adjustment for the acceleration due to gravity and inherently affects how beats are detected on the app, despite being on the IMUduino. It is set to 20000 by default. Changing this to a higher number will decrease the sensitivity while a lower one will increase it. Best performance is within the range of 16000-22000. Beware that setting this too low or too high will result in very poor performance.

e. If you choose to adjust this number, do so, compile the sketch and upload it to the IMUduino.

f. Follow steps 9-14 of section 10.1 and test the new sensitivity.

10.4 Storage and Maintenance

1. The battery life of the movement sensor is approximately 9 hours of use. To ensure the batteries are not drained when not in use, remove them from the battery cases when not using the device for extended periods of time. Replacing the batteries simply involves repeating steps 2-3 of section 10.1.
2. 20mm coin cells other than CR2032s may be used as substitutes, however the battery life will differ. Keep in mind that if the total supply voltage exceeds 9V, the board will be damaged.
3. Take special care to ensure the movement sensor does not get wet, excessively heated, subjected to heavy impacts, or exposed to static electricity as these events will damage or destroy it.

11. DISCUSSION, RECOMMENDATIONS, AND CONCLUSION

11.1 Discussion

After two terms of hard, consistent work, a system which greatly aids a student of conducting, or even a professional conductor, has been designed and successfully developed. This system, which meets or exceeds most design requirements, solves a common problem faced by students of conducting and provides a versatile and valuable tool, giving worthwhile feedback to the user.

11.1.1 Problem

Conductors have a responsibility not only to leverage their musical expertise as the directors of musical ensembles but also to control performance. They do this through the action of conducting—moving their arms, with or without a baton, in specific patterns through the air to indicate the tempo and time signature of music. Students of conducting often struggle with maintaining consistency in conducting a specific tempo; they speed up or slow down over a period of time. They also tend to struggle with being able to start conducting at a specific tempo without reference. These skills are built over years of practice and hard work, which give the conductor an internal feel for these tempos.

11.1.2 Solution and Implementation

The objective of this project was therefore to give a student conductor a tool with which they can practice and gain valuable feedback from, whether on or off the podium. The resulting system detects and displays the tempo which the user is conducting at that very moment, with sufficient accuracy to keep them on tempo. The system consists of two parts: a wristband-mounted movement sensor and a smartphone app. The movement sensor interprets the motion of the conductor's arm and the app calculates and displays the tempo at which they are conducting. Through extended use of this system, the user will accelerate their learning curve and will build up their internal sense of tempo quickly.

11.1.3 Results

The system created met or exceeded all but one of the design requirements as outlined in section 3. It is able to quickly detect and calculate tempos conducted from 40-200 bpm with an accuracy of +/- 10 bpm. It is small, lightweight, easy to use, and does not hinder the motion of the conductor's arm in any way. The app is easy to use, has a simple connection process and displays tempo in the center of the phone screen in large format. Due to the success in meeting the design requirements, this system has very high potential to aid a student conductor in developing their skill and preparing them for professional careers.

11.2 Recommendations

This system, while functional, is certainly far from perfect. Several lessons were learned along the way, from the importance of backing up data to planning multiple design alternatives to fall back on. Additionally, there are several optimizations and improvements that can be made, along with possible expansions on the concept behind the project.

11.2.1 Future Work

The primary focus of future work is to meet the design requirements that were either partially met or not at all met. Specifically, this means increasing the accuracy of the algorithm such that it is able to calculate tempos with an accuracy of +/- 5 bpm instead of 10. In order to do this, a method of mechanically testing the exact accuracy of the system needs to be implemented. The next improvement to make is to add either a rechargeable battery or a higher capacity battery so the system works for a longer period of time or removes the hassle of changing batteries. Thirdly, the usefulness of the system could be massively increased by adding a plot of tempo vs. time which automatically appears after conducting concludes and can be saved as an image file. In order to accomplish this goal, the problem of the Bluetooth connection randomly dropping needs to be solved, otherwise the plot will only cover a smaller period of time and its usefulness will be diminished. The current version of the project, while less exposed than the prototype, is still not particularly aesthetically pleasing. Improving the 3D-printed case design and integrating the battery cases into it would help increase its aesthetic appeal. Lastly, placing a patent on the system is the final recommendation. While similar systems more than likely exist, none have yet been patented.

11.2.2 Lessons Learned

Several engineering lessons were learned over the course of this project. First, logging every detail of every little event is important, as while unanticipated at the time, they may be needed in the future. Additionally the importance of backing up data cannot be emphasized enough. At several points during the project, the operating system of the computer which all the coding was done on updated and restarted overnight. The Arduino software also performed automatic updates. Without backups, these updates could have presented bigger problems than they did. Keeping multiple copies of code allows for a return to the original version if something goes wrong. Another incredibly important lesson learned is to only implement one change at a time before testing or trying something new. This approach avoids having to search through long lists of changes to identify the cause of a problem if one should occur. From a design perspective, yet another lesson learned is that it is crucial to plan multiple ways to accomplish a particular goal. Contingency plans are necessary because if a fundamental flaw or insurmountable problem is encountered in the primary design approach, another option can be used as a fallback. For example when the IMUduino ran out of memory, the algorithm could simply be implemented on the app instead. This project was a solid reminder of the importance of patience. Multiple long and frustrating nights spent coding were rewarded with a working product at the end. This product will potentially make a difference in someone’s life and that is a huge reward on its own.

11.2.3 Expansions

There are a great deal of expansions that can be applied to this project based on its initial concept. First and foremost, it would be interesting to see if the system could be implemented on a smartwatch instead of on a dedicated microprocessor. Using a smartwatch means that there would be virtually no cost associated with the system itself and the entire project more than likely could be consolidated into a single app which interfaces with the accelerometer in the watch. Additionally, it would be interesting to see how the system would have developed if implemented with the MetaWear RG board. The system would have a rechargeable battery and its own dedicated app, which would have to be written entirely from scratch. However it has the potential to be less complicated, smaller and cheaper. Overall, any expansion that lowers the cost or improves the simplicity or aesthetics should be explored.

11.3 Conclusion

In conclusion, this project tackled the problem of tempo consistency and accuracy—faced nearly unanimously by students of conducting—and implemented a practical, reliable, and effective solution which will make a difference with these individuals. The system meets or exceeds nearly every design requirement and despite the few shortcomings performs very well. Several design alternatives were available but only the best of them were picked for implementation in the final product. While the final design differed from what was initially proposed in the preliminary design, it ultimately resulted in a better product. The project's development resulted in fluctuations within the design schedule, yet these setbacks were adjusted to and a quality product was still delivered on time—and within the allotted budget. This project

has the potential for several expansions and improvements, which will be the focus of future work. Above all, the lessons learned over the course of these last few terms were invaluable and will not be forgotten. Carrying an original idea through the phases of design completely to fruition is an experience like no other and is truly rewarding.

12. REFERENCES

[1] Yoshiki Nishitani, Eiko Kobayashi, *Apparatus and method for detecting performer's motion to interactively control performance of music or the like*, United States Patent, November 14, 2006

[2] Brian Spradlin, Arduino Nano V3 Case http://www.thingiverse.com/thing:608121/#files, December 24, 2014

[3] Derek Remeš, Basic Conducting Patterns, http://derekremes.com/teaching/tonaltheory/rhythm-meter/

[4] Battery Life Calculator, http://www.digikey.com/en/resources/conversion-calculators/conversion-calculator-battery-life

[5] *BMI160 Small, Low-Power Inertial Measurement Unit*, Bosch Sensortec, 2015

[6] Michael Miller, *The Complete Idiot's Guide to Music Theory*: Penguin, 2005, pp. 261-262

[7] *CR2032 Product Datasheet*, Energizer Inc

[8] How to Become a Music Conductor: Education and Career Roadmap, http://study.com/articles/How_to_Become_a_Music_Conductor_Education_and_Career_Roadmap.html

[9] IMUduino, http://femto.io/products/imuduino

[10] Stephanie Castillo, Low-Income Kids Benefit from Music Class, Show Greater Reading Skills, http://www.medicaldaily.com/low-income-kids-benefit-music-class-show-greater-reading-skills-329700, 2015

[11] *MetaWear RG/RPro*, MbientLab, 2015

[12] *MPU-6000 and MPU-6050 Product Specification Revision 3.4*, InvenSense, INC, 2013

[13] Music Teacher Salary, http://www.indeed.com/salary/Music-Teacher.html

[14] *nRF8001*, Nordic Semiconductor, 2015

[15] *nRF51822*, Nordic Semiconductor, 2014

[16] nRF UART App, https://www.nordicsemi.com/eng/Products/nRFready-Demo-Apps/nRF-UART-App2

[17] Office of Engineering and Technology Federal Communications Commission, *Understanding the FCC Regulations for Low-Power, Non-Licensed Transmitters*, Bulletin, 1993

[18] What Can You Do with a Music Degree? http://majoringinmusic.com/what-can-you-do-with-a- music-degree/

[19] Which regulations are passed by nrf8001 e.g. FCC, https://devzone.nordicsemi.com/question/462/which-regulations-are-passed-by-nrf8001-eg-fcc/, 2013

[20] Harry McCracken, Who's Winning, iOS or Android? All the Numbers, All in Once Place, http://techland.time.com/2013/04/16/ios-vs-android/, 2013

13. APPENDICES

The following pages contain appendices with the following materials:

- **Appendix A:** Code on IMUduino for calculating magnitude of acceleration and transmitting to smartphone via Bluetooth
- **Appendix B:** Variables used in Android Studio code
- **Appendix C:** First version of beat detection and tempo calculation algorithm as implemented in Android Studio
- **Appendix D:** Final version of beat detection and tempo calculation algorithm as implemented in Android Studio
- **Appendix E:** Testing questionnaire
- **Appendix F:** Final presentation slides
- **Appendix G:** Acknowledgements

APPENDIX A

```
#include <HMC58X3.h>

#include <I2Cdev.h>

#include <MPU60X0.h>

#include <EEPROM.h>

//#define DEBUG

#include "DebugUtils.h"

#include "IMUduino.h"

#include <Wire.h>

#include <SPI.h>

// Adafruit nRF8001 Library

#include "Adafruit_BLE_UART.h"

// Connect CLK/MISO/MOSI to hardware SPI

#define ADAFRUITBLE_REQ 10

#define ADAFRUITBLE_RDY 7

#define ADAFRUITBLE_RST 9

Adafruit_BLE_UART BTLEserial =
Adafruit_BLE_UART(ADAFRUITBLE_REQ, ADAFRUITBLE_RDY,
ADAFRUITBLE_RST);

aci_evt_opcode_t laststatus = ACI_EVT_DISCONNECTED;

aci_evt_opcode_t status = laststatus;

int raw_values[11];

//float ypr[3];

// Set the FreeIMU object

IMUduino my3IMU = IMUduino();

#define zeroAdjust 20000 //This number is representative of the
acceleration due to gravity that gets subtracted from the
```

```
magnitude calculation later. This number should be between 16000
and 22000 in order to get reliable results

long mag = 0;

void setup() {

////  Mouse.begin();

  Serial.begin(115200);

  Wire.begin();



  Serial.println(F("Adafruit Bluefruit Low Energy nRF8001 +
FreeIMU Print echo demo"));

  delay(500);

  my3IMU.init(true);

  BTLEserial.begin();

}

void loop() {

  btleLoop();

  if (status == ACI_EVT_CONNECTED) {

    my3IMU.getRawValues(raw_values);

    mag = ((sqrt((pow(raw_values[0],2))+

     (pow(raw_values[1],2))+

     (pow(raw_values[2],2))))-zeroAdjust); //zeroAdjust changes
sensitivity of accelerometer

   //Above is the magnitude of acceleration calculation

    btleWrite(

      String(mag));

  }

}
```

```
/*****************************************************************
***********/

/*!

    Constantly checks for new events on the nRF8001

*/

/*****************************************************************
***********/


void btleLoop() {

  // Tell the nRF8001 to do whatever it should be working on.

  BTLEserial.pollACI();

  // Ask what is our current status

  status = BTLEserial.getState();

  // If the status changed....

  if (status != laststatus) {

    // print it out!

    if (status == ACI_EVT_DEVICE_STARTED) {

        Serial.println(F("* Advertising started"));

    }

    if (status == ACI_EVT_CONNECTED) {

        Serial.println(F("* Connected!"));

    }

    if (status == ACI_EVT_DISCONNECTED) {

        Serial.println(F("* Disconnected or advertising timed
out"));

    }

    // OK set the last status change to this one
```

```
    laststatus = status;

  }

  if (status == ACI_EVT_CONNECTED) {

    // Let’s see if there's any data for us!

    if (BTLEserial.available()) {

      Serial.print("* "); Serial.print(BTLEserial.available());
Serial.println(F(" bytes available from BTLE"));

    }

    // OK while we still have something to read, get a character
and print it out

    while (BTLEserial.available()) {

      char c = BTLEserial.read();

      Serial.print(c);

    }

    // Next up, see if we have any data to get from the Serial
console

    if (Serial.available()) {

      // Read a line from Serial

      Serial.setTimeout(100); // 100 millisecond timeout

      String s = Serial.readString();

      btleWrite(s);

    }

  }

}

void btleWrite(String s) {

  // We need to convert the line to bytes, no more than 20 at
this time
```

```
  uint8_t sendbuffer[20];

  s.getBytes(sendbuffer, 20);

  char sendbuffersize = min(20, s.length());

  Serial.print(F("\n* Sending -> \"")); Serial.print((char
*)sendbuffer); Serial.println("\"");

  // write the data

  BTLEserial.write(sendbuffer, sendbuffersize);

}

byte * float2str(float arg) {

  // get access to the float as a byte-array:

  byte * data = (byte *) &arg;

  return data;

}
```

APPENDIX B

```
int threshold=0; //Magnitude of acceleration crossing this
threshold indicates a beat

int minTempo=40; //Lowest practical tempo

int maxTempo=200; //Highest practical tempo

double oneMin=6e10; //One minute in nanoseconds (used for
System.nanoTime)

int flag=0; //Prepared for first beat

double timeA=0;

double timeB=0;

double tempoA=0;

double tempoB=60; //Initially set to 60 so as not to throw off
tempo calculation in first few beats

double[] TempoArray = {60,60,60,60,60}; //A common tempo so as
not to throw off the averaging within the first few beats

int i=0; //index variable

double sum=0;

double average = 0;
```

APPENDIX C

```
final byte[] txValue =
intent.getByteArrayExtra(UartService.EXTRA_DATA); //Get data
from Bluetooth as byte array

          runOnUiThread(new Runnable() {

               public void run() {

                    try {

                         String text = new String(txValue);
                         //turn byte array to string

                             int mag = Integer.parseInt(text);
                             //String to integer, this is the
                             magnitude of acceleration

                             if ((mag>threshold)&& flag==0) {
                             //If a valid beat is detected when
                             magnitude crosses threshold

                                 flag = 1;

                                 timeA = System.nanoTime();
                                 //log time threshold crossed

                                 tempoA = oneMin/(timeA -
                                 timeB);

                                 //calculate tempo in beats per
                                 minute

                                 timeB = timeA; //reset time in
                                 preparation for next beat

                                 if ((tempoA >= minTempo) &&
                                 (tempoA <= maxTempo)) { //If
                                 the tempo is within design
                                 requirements...

                                     tempoA = ((tempoA + tempoB)
                                     / 2); //average the last
                                     two tempos

                                     String tempo =
                                     Integer.toString((int)
```

```
                    tempoA); //convert to
                    string...

                    String
                    currentDateTimeString =
                    DateFormat.getTimeInstance(
                    ).format(new Date());

                    tempoBox.setText(tempo);
                    //and print the tempo

                }

                tempoB=tempoA; //reset tempo to
                in preparation for new beat

                Thread.sleep(300); //sleep for
                300ms since with the max
                calculable tempo, beats are
                300ms apart

            }

             else if ((mag<threshold)&&flag==1)
             { //case where there is no beat
             (below threshold)

                  flag=0; //reset flag so next
                  beat can be detected

            }

        } catch (Exception e) {

           Log.e(TAG, e.toString());

        }

    }

});
```

APPENDIX D

```
final byte[] txValue =
intent.getByteArrayExtra(UartService.EXTRA_DATA); //Gets data
from Bluetooth as a byte array

     runOnUiThread(new Runnable() {

          public void run() {

               try {

                    String text = new String(txValue); //Turns
                    byte array data into string

                    int mag= Integer.parseInt(text); //String to
                    integer, this is the magnitude of
                    acceleration

                         if (TempoArray != null) {

                              if ((mag > threshold) && flag ==
                              0) {  //If a valid beat detected
                              when magnitude crosses threshold

                                   flag = 1;

                                   timeA = System.nanoTime();
                                   //Log time threshold was
                                   crossed

                                   tempoA = oneMin / (timeA -
                                   timeB); //Tempo calculation
                                   in beats per minute

                                   timeB = timeA; //Reset time
                                   to prepare to receive next
                                   beat

                                   if ((tempoA >= minTempo) &&
                                   (tempoA <= maxTempo) &&
                                   tempoA <= (2 * tempoB)) {
                                   //if the tempo calculated
                                   above was within the design
                                   requirements and not
                                   ridiculously different from
                                   the previous tempo...
```

```
        TempoArray[i] = tempoA;
        //Add it to an array

        i++;

        tempoB = tempoA;
        //Reset the tempo to
        prepare to receive next
        beat

    }

sum = TempoArray[0] +
TempoArray[1] + TempoArray[2]
+ TempoArray[3] +
TempoArray[4];

average = sum / 5; //take
average of tempo array

if (tempoA >= (average - 10)
&& tempoA <= (average + 10))
{ //if the calculated tempo
is within 10bpm of the
average of the last five...

        String tempo =
        Integer.toString((int)
        tempoA); //turn it back
        into a string...

        String
        currentDateTimeString =
        DateFormat.getTimeInsta
        nce().format(new
        Date());

        tempoBox.setText(tempo)
        ; //and print it on the
        screen

    }

Thread.sleep(300); //sleep
for 300ms since with the max
calculable tempo, beats are
300ms apart
```

```
                } else if ((mag < threshold) &&
                flag == 1) { //this is the case
                when there is no beat (below
                threshold)

                     flag = 0; //reset the flag to
                     prepare to receive next beat

                   }

                if (i > 4) { //keep the length of
                the tempo array at 5

                       i = 0;

                   }

               }


            } catch (Exception e) {

                Log.e(TAG, e.toString());

             }

        }

    });
```

APPENDIX E

Please answer the following questions on a scale of 0-10

- How comfortable is it on your wrist?:_____

0=Uncomfortable 10=Comfortable

- Usefulness of tempo feedback:____

0=Completely Useless 10=Extremely Useful

- Do you feel the numbers are accurate?:____

0=Not at all 10=Absolutely

- Easy to use?:____

0=Very Difficult 10=Very easy

- Did using the device distract from rehearsal?:____

0=It was very distracting 10=Didn't know it was there

- Was the display easy to read?:____

0=Difficult to read 10=Very easy to read

- Did the tempo feedback add to the rehearsal or practice experience?:____

0=Not at all 10=Very much

- Was the device able to pick up varying styles of beat indication and time signatures?:____

0=Completely unable 10=Completely able

- What was your overall experience like?:____

0=Terrible 10=Wonderful

Please answer the following questions in a few sentences.

- If it were sold, would you buy a refined version of this device to help you?

- What other features would you want to see on the app?

- Any additional ideas?

APPENDIX F

Tempo-Sensing Feedback System for Student Conductors

Stephen Hoeprich

Advisor: Professor John Spinelli

Goals

- Aid conductor in maintaining accuracy and consistency
- Still allows the conductor to learn on their own
- Easy to use and does not distract from leading the ensemble

Outline

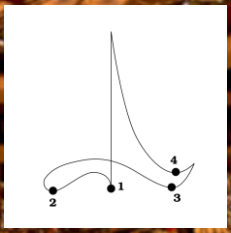
- Introduction
- Goals
- Performance Criteria
- Design and Implementation
- Results and Further Work
- Appendices

System (1 of 2)

- Two components: Movement sensor and smartphone app
  - Movement sensor detects movement and transmits that information to app
  - App detects beats, calculates tempo, and displays tempo

Introduction

- A conductor leads and directs a musical ensemble
  - Uses their hands or a baton to indicate the tempo of music being played
- Aimed at student conductors
  - Struggle with tempo consistency and accuracy without reference
  - Rare opportunities to lead ensembles
- System measures tempo based on the movement of the user's arm
  - Displays the live tempo, giving a reference and helping train consistency and accuracy



Conducting pattern for 4/4 time

System (2 of 2)

Movement Sensor

App

Performance Criteria

- Movement sensor
  - Must be small, fit on wristband, and weigh no more than a wristwatch
  - Must communicate with app wirelessly and reliably
- App
  - Must differentiate between beats and other movements
  - Display tempo within five seconds of starting
  - Calculates tempo +/- 5 beats per minute (bpm) of actual tempo, for tempos 40-200 bpm
  - Must be easy to read and use

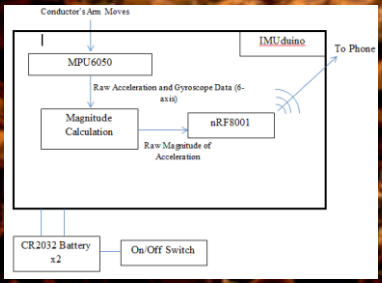
Movement Sensor (2 of 5)

- Implemented on IMUduino
  - Accelerometer and gyroscope: MPU6050
  - Bluetooth transmitter: Nordic nRF8001
- Detects magnitude of acceleration for the conductor's arm
  - Adjusted for acceleration due to gravity
  - $\sqrt{x^2 + y^2 + z^2} - Gravity$
- Transmits this to smartphone app
- Initially stored beat detection algorithm, not enough memory for Bluetooth
- Powered with 2 x CR2032 coin cell batteries, 6V @ 550 mAh

IMUduino

Design and Implementation

Movement Sensor (3 of 5)



Movement Sensor Block Diagram

Movement Sensor (1 of 5)

- Attached to user's wrist and picks up movement of the arm
- Microprocessor with accelerometer, gyroscope, and Bluetooth transmitter
- Microprocessor options:
  - Femtoduino IMUduino
  - Metawear RG board

Movement sensor (4 of 5)

Case

Movement Sensor (5 of 5)

Prototype version

Final version

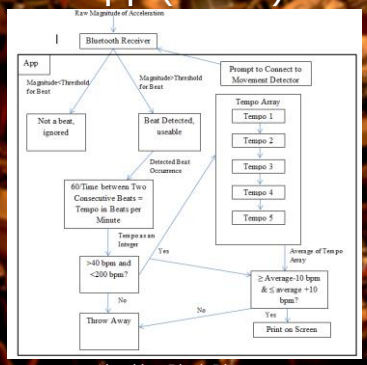
App (3 of 6)

- Initially app was “dumb” and just displayed data
- App eventually did all the heavy lifting
  - Beat detection
  - Tempo calculation
  - Display

App (1 of 6)

- Picks up magnitude of acceleration from movement sensor
- Determines what is an indicated beat
- Calculates tempo as each new beat comes in
- Displays tempo on phone screen

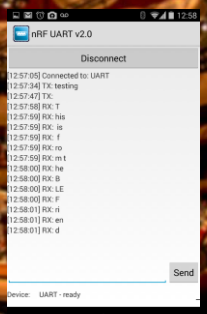
App (4 of 6)



Algorithm Block Diagram

App (2 of 6)

- Android only
- nRF UART
  - Works with nRF transmitters
  - Displayed data from Bluetooth as a scrolling list
  - Open-source software
- Modified the app
  - Displayed data centered without scrolling



nRF UART app unmodified

App (5 of 6)

```
minTempo=40       %Minimum tempo
maxTempo=200      %Maximum tempo
timeOld=0         %Previous reference time initially set to 0
while beatDetector=ON;  %While the beat detector is on, run this loop
    if threshold=crossed
        beat=detected;              %Beat Detected
        timeNew=currentTime;        %Record time beat was detected
        TempoInit=60/(timeNew-timeOld); %Tempo Calculation
        if TempoInit > minTempo & ≤ maxTempo %Is tempo between min and max?
            Tempo=round(TempoInit); %If it is, round it and store it
        end
        timeOld=timeNew;        %Now the time that the last beat that was
    end                         %detected will be used for reference in tempo
end                             %calculation instead of zero.
```

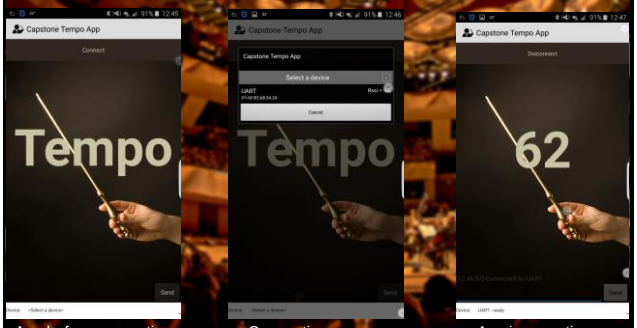
Pseudocode representing first version of beat detection and tempo calculation algorithm

App (6 of 6)

```
timeB=0
tempoArray=[60,60,60,60,60]
if magnitude>threshold && flag==0 %if beat detected
    flag=1 %indicator of above or below threshold
    timeA=now %Log time threshold crossed
    tempo=OneMinute/timeA-TimeB %calculate tempo using difference in times
    timeB=timeA %reset time for next beat
    if tempo1 meets design requirements && <= 2*previous tempo
        tempoArray[i]=tempo %add tempo to tempoArray
        increment
    end
    average=(sum of tempoArray)/5 %calculate average of tempoArray
    if tempo is within +/- 10 bpm of average
        print it onscreen
    end
    sleep(300) %Fastest tempo is 300ms between beats
end
elseif magnitude<threshold %No beat detected
    flag=0
end
if number of elements in tempoArray is > 5 %keep tempoArray at 5 elements
    reset index to 0
end
```

Pseudocode of final version of beat detection and tempo calculation algorithm

Appendix A



App before connection | Connection process | App in operation

Conclusions and Further Work (1 of 2)

- No patented system like this exists
- No way to test exact accuracy, but accurate to within 10 bpm with experienced conductors
- Battery life too short at 8.95 hours
  - 43 mA current draw

Criteria	Met?
Small, lightweight	YES
Wireless	YES
Filters out movements that are not beats	YES
Display within five seconds	YES
Easy to read and use	YES
Accurate within +/- 5 bpm	NO
4-200 bpm	YES

Appendix B (1 of 3)

```
int threshold=5;
int minTempo=20;
int maxTempo=220;
long zeroAdjust=25000;
double oneMin=6e10;
int flag=0;
int min_val=0;
double timeA=0;
double timeB=0;
double tempoA=0;
double tempoB=60;
double[] TempoArray = {60,60,60, 60, 60};
int i=0;
double sum=0;
double average = 0;
```

Variables

Conclusions and Further Work (2 of 2)

- Live demo on Tuesday at 12:50 Wold
- By Steinmetz presentation date
  - Nicer case
  - Solve smaller issues
    - Battery life
    - Stronger Algorithm
    - Fluctuations
  - Add graph of tempo vs time
  - Add ability to set a goal tempo and change color of tempo if above or below goal
  - Patent

Appendix B (2 of 3)

```
final byte[] txValue = intent.getByteArrayExtra(UartService.EXTRA_DATA); //Get data from Bluetooth as byte array
    runOnUIThread(new Runnable() {
        public void run() {
            try {
                String text = new String(txValue); //turn byte array to string
                int mag = Integer.parseInt(text); //String to integer, this is the magnitude of acceleration
                if ((mag>threshold)&& flag==0) { //If a valid beat is detected when magnitude crosses threshold
                    flag = 1;
                    timeA = System.nanoTime(); //log time threshold crossed
                    tempoA = oneMin/(timeA - timeB); //calculate tempo in beats per minute
                    timeB = timeA; //reset time in preparation for next beat
                    if ((tempoA >= minTempo) && (tempoA <= maxTempo)) { //If the tempo is within design requirements...
                        tempoA = ((tempoA + tempoB) / 2); //average the last two tempos
                        String tempo = Integer.toString((int) tempoA); //convert to string
                        String currentDateTimeString = DateFormat.getTimeInstance().format(new Date());
                        tempoBox.setText(tempo); //and print the tempo
                        //listAdapter.add("[" + currentDateTimeString + "] RX: " + text);
                        //messageListView.smoothScrollToPosition(listAdapter.getCount() - 1);
                    }
                    tempoB=tempoA; //reset tempo to in preparation for new beat
                    Thread.sleep(300); //sleep for 300ms since with the max calculable tempo, beats are 300ms apart
                }
                else if ((mag>threshold)&&flag==1) { //case where a beat has been detected but flag is still 1, do nothing
                    }
                if ((mag<threshold)&&flag==1) { //case where there is no beat (below threshold)
                    flag=0; //reset flag so next beat can be detected
                }
            } catch (Exception e) {
                Log.e(TAG, e.toString());
            }
        }
    }});
```

First version of beat detection and tempo calculation algorithm

Appendix B (3 of 3)

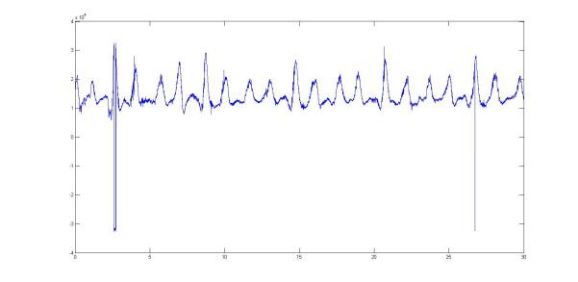
```
final byte[] txValue = intent.getByteArrayExtra(GattService.EXTRA_DATA); //Gets data from Bluetooth as a byte array
    runOnUiThread(new Runnable() {
        public void run() {
            try {
                String text = new String(txValue); //Turns byte array data into string
                int mag= Integer.parseInt(text); //String to integer, this is the magnitude of acceleration
                if (TempoArray != null) {
                    if ((mag > threshold) && flag == 0) {  //If a valid beat detected when magnitude crosses threshold
                        flag = 1;
                        time4 = System.nanoTime(); //Log time threshold was crossed
                        tempoA = oneMin / (time4 - time3); //Tempo calculation in beats per minute
                        time3 = time4; //Reset time to prepare to receive next beat
                        if ((tempoA >= minTempo) && (tempoA <= maxTempo) && tempoA <= (2 * tempoB)) {
                            //if the tempo calculated above was within the design requirements and not ridiculously different from the previous tempo...
                            TempoArray[i] = tempoA; //Add it to an array
                            i++;
                            tempoB = tempoA; //Reset the tempo to prepare to receive next beat
                        }
                        sum = TempoArray[0] + TempoArray[1] + TempoArray[2] + TempoArray[3] + TempoArray[4];
                        average = sum / 5; //Take average of tempo array
                        if (tempoA >= (average - 10) && tempoA <= (average + 10)) {
                        //if the calculated tempo is within 10bpm of the average of the last five...
                            String tempo = Integer.toString((int) tempoA); //Turn it back into a string
                            String currentDateTimeString = DateFormat.getTimeInstance().format(new Date());
                            tempoBox.setText(tempo); //and print it on the screen
                        }
                        Thread.sleep(300); //sleep for 300ms since with the max calculable tempo, beats are 300ms apart
                    } else if (mag < threshold) && flag == 1) { //this is the case when there is no beat (below threshold)
                        flag = 0; //reset the flag to prepare to receive next beat
                    }
                    if (i > 4) { //keep the length of the tempo array at 5
                        i = 0;
                    }
                }

            } catch (Exception e) {
                Log.e(TAG, e.toString());
                }
```

Final version of beat detection and tempo calculation algorithm

Appendix E

Component	Quantity	Cost per Unit	Total Cost
Imuduino board	2	$80	$160
MetaWear RG Board	3	$55	$165
Wristband	3	$10	$30
Funding for 3D printed case	2	$10	$20
Batteries (varying voltage)	5	$2	$10
Bluetooth adaptor for PC	1	$9	$9
			Project total
			$394

Appendix C



Magnitude of acceleration vs. time when conducting

QUESTIONS?

Appendix D

Testing with Professor John Cox of the Music Department

APPENDIX G

Professor John Spinelli: Thank you for your patience with me and your shared enthusiasm over the topic of this project. Your guidance and direction carried me through the entire experience and I sincerely appreciate everything you did as my advisor. Thank you for not giving me the answers and allowing me to discover them on my own.

Jessica Sanford: Thank you for teaching me how to work with Android Studio and for helping me reverse-engineer the nRF UART app. Thank you for helping me debug my second algorithm and thank you for printing out several iterations of the case for me. Lastly but certainly not least, thank you for putting up with my clunky amateur coding.

Gene Davison: Thank you for taking care of the very precise soldering required on the IMUduino board and for printing the poster.

Professors John Cox and Tim Olsen: Thank you for helping by testing out my prototype and giving me honest feedback. Your suggestions were responsible for the new and improved version of the algorithm and for most of the ideas in the future work and expansion sections. Thank you as well for your excellent leadership of the Union College musical ensembles of which I have been a part. My participation in music at Union was a primary motivator for the development of this project.

All Union College ECE Faculty: Thank you for giving me the tools and educational foundation to successfully pull this project off. Thank you Professor Hedrick for purchasing the DPST switch and for letting me use that roll of electrical tape during crunch time.

Family and friends: Thank you for putting up with my long sessions of isolation and seemingly unexplainable frustration. Your support and multiple coffees purchased did not go unappreciated.