



# Snare Drum Notator

## Design Report

05/01/2017

---

### Team MusIQ

Domn Werner - [wern0096@vandals.uidaho.edu](mailto:wern0096@vandals.uidaho.edu)

Phillip Kearns - [kear4811@vandals.uidaho.edu](mailto:kear4811@vandals.uidaho.edu)

Nathan Groggett - [grog6141@vandals.uidaho.edu](mailto:grog6141@vandals.uidaho.edu)

Hue Purkett - [purk2552@vandals.uidaho.edu](mailto:purk2552@vandals.uidaho.edu)

Scott Dennis - [denn2725@vandals.uidaho.edu](mailto:denn2725@vandals.uidaho.edu)

## Table of Contents

<b>Table of Contents</b>	<b>1</b>
<b>Executive Summary</b>	<b>4</b>
<b>Background</b>	<b>4</b>
<b>Problem Definition</b>	<b>5</b>
Project Goals	5
Notation	5
Software	5
Deliverables	5
Stakeholder Benefits	6
<b>Project Plan</b>	<b>6</b>
Team Roles and Responsibilities	6
Scott Dennis	6
Nathan Groggett	6
Phillip Kearns	6
Hue Purkett	6
Domn Werner	6
Schedule	6
Planned	6
Actual	7
Budget	7
<b>Concepts Considered</b>	<b>8</b>
Microphone Detection	8
Drum Pad Mounted Motion Sensor or Pressure Pad Sensor	8
Stick Mounted Motion Sensors	8
Wristband Mounted Motion Sensors	9
<b>Concepts Selected</b>	<b>9</b>
Design	9
Hardware	9

Intel Edison	9
Sparkfun Blocks	10
Software	10
Python	10
C#	10
MusicXML	11
Protocols	11
Bluetooth Low Energy (4.0)	11
<b>System Architecture</b>	<b>12</b>
GUI	12
Bluetooth Communication	12
Data Processing	12
Filter	13
Note Class	14
NotetoXmlTranslator Class	14
MusicXML Generation	14
MusicXmlGenerator Class	14
<b>Design Evaluation</b>	<b>15</b>
Testing Procedures	15
Test Results	15
<b>Future Work</b>	<b>15</b>
Rhythms and Rudiments	15
Edison Setup	15
GUI	16
Metronome	16
<b>Appendix</b>	<b>17</b>
Drawings	17
Tables and Figures	18
Computer Programs	20
Programming	20



Data Sheets	20
Intel Edison	20
Original Project Schedule	20
Actual Project Schedule	20
File Management Overview	20

## Executive Summary

The Snare Drum Notator project nurtures creativity and reduces the time between creating and sharing snare drum sheet music.

Traditionally, writing sheet music, even using dedicated software, is slow and cumbersome. By leveraging the parsing of data from two wireless inertial measurement units (IMUs) and custom written software, the Snare Drum Notator project provides snare drum composers with nearly instantaneous feedback in the form of accurate snare drum sheet music for what they play.

This rapid turnaround can easily speed up snare drum composing by an order of magnitude due to the highly integrated hardware and software relationship present in the final product. Such a benefit is further compounded by reducing fatigue, increasing productivity, and encouraging the composer to actually play the instrument. Sitting in front of a computer screen or scribbling on staff paper is the past. This system is the future.

## Background

Generally, percussion composing consists of three steps: creating, playing, and notating.

In the creating step, the composer lets their creative juices flow as they try to come up with a groovy “lick” or musical idea that fits well with the overall piece of music they are composing. This is a notably difficult step due to the sheer amount of possible permutations in music.

The playing step can be thought of as a validation for the creating step. The composer will try to play their idea (often on a practice pad) to see if it actually works for the desired instrument and difficulty level that they are composing for. It is also extremely rewarding and useful to be able to hear your idea ‘come to life’. Note that this step is often mixed with the first one, but the creativity step is still a clear prerequisite.

Lastly, if the composer is happy with their idea, they will often immortalize it by writing it down as sheet music. This is the most important step because it is also an avenue of clear communication between musicians. However, this is also the most time-consuming step, even with dedicated notating software -- doubly so if writing rough drafts on paper.

This project greatly facilitates the last two composing steps. Once the composer comes up with an excerpt which they like, they can simply play it and export what they played as sheet music with an automated and user-friendly process.

## Problem Definition

### Project Goals

#### Notation

The system should notate the following:

- Volume (velocity)
- Sticking (left, right, or both hands)
- Position on the pad
- Back of the stick vs front of the stick
- Rudiments (rolls, flams, rimshots, ...)
- Accents (louder hits) vs taps (softer hits) relative to current volume
- Tempo
- Whether the user is hitting the rim or the pad
- The note timing (quarter, sixteenth, triplet notes)
- Dynamics (increasing/decreasing volume)
- Time signature
- Rests

Refer to the Rhythms and Rudiments tables for an enumeration of the rhythms and rudiments this project aimed to support.

#### Software

The software side of the project should perform all the logic to convert the IMU data into a format that can be used to render sheet music. It should also expose a simple graphical user interface (GUI) to facilitate the exporting of the sheet music or file to be used for rendering the sheet music. This format will ideally be universally supported by common music notation software.

### Deliverables

The hardware deliverables for this project are two hand-mounted Intel Edisons complete with Sparkfun sensor blocks. Each Edison is running data capture code and hand-specific (right/left) data analysis code, and capable of connecting via Bluetooth to a PC for signal analysis. The Edisons should be able to be worn in an adjustable and non-invasive manner that allows a comfortable and secure fit without impeding the playing motion.

The software deliverables include the aforementioned Windows software program for analyzing the Inertial and Magnetic Unit (IMU) data and GUI for displaying the sheet music played. Documentation on how to set up and use the software and hardware will also be provided.

## Stakeholder Benefits

Professional composers will gain more time to create and play their music when they no longer have to spend anywhere near as much time committing it to paper.

People learning to drum can use this system as a learning aid by playing some sheet music and seeing how close they got.

## Project Plan

### Team Roles and Responsibilities

Scott Dennis

- Data Processing Programmer
- Note/Rhythm Calculations Programmer

Nathan Groggett

- Bluetooth Engineer/Programmer
- Information Researcher

Phillip Kearns

- Hardware Prototyping Engineer
- Motion Signal Analyst

Hue Purkett

- MusicXML API Programmer
- GUI Designer and Programmer

Domn Werner

- Server side Edison API Programmer
- Client side Edison Programmer
- Team Leader

## Schedule

Planned

Initial Testing/Pattern Classification: Oct-Nov

Communication Protocols:	Nov-Dec
Field Testing - School of Music:	Jan-Feb
Revisions/User Interface:	Feb-March
Final Testing/Revisions:	March-May

### Actual

Concept Development:	Sep-Dec
Hardware Assembly:	Oct
Fiddling with Bluetooth:	Nov-March
Note Identification and notation:	Feb-Apr
Testing:	Feb-Apr

### Budget

The hardware the project used turned out to be inexpensive.

Product	Quantity	Price
Arduino 101	2	\$60
Intel Edison	2	\$100
Battery, Base, 9DOF	2 (6)	\$185
Watch Band/Base	2	\$20
Hardware	2	\$6
TOTAL		\$371



## Concepts Considered

### Microphone Detection

This method of note detection would utilize a microphone to determine when the user played notes. The limitation with this method lies in the fact that certain rudiments in snare drum music *sound* the same, but are different. An example would be doubles, in which two short notes are played quickly with one hand. This would sound identical to one note played with each hand, but the notation required is different for each scenario. A microphone would not be able to distinguish between the two.

Additionally, a microphone could be limited by ambient sound in the user's environment. Extra work would have to be done to distinguish the difference between snare drum notes and other, unrelated sounds. This also may limit the user's options with regard to playing surface; playing on a practice pad sounds much different than an actual drum. Finally, if there were other drummers nearby playing rhythms of their own, it may be nearly impossible to distinguish the user's rhythm from the other drummers' rhythms.

### Drum Pad Mounted Motion Sensor or Pressure Pad Sensor

A sensor on the drum pad itself was also considered. This would take the form of either a 'mounted' sensor attached or built into the drum pad, or a pressure pad sensor. The drawback with these designs, similar to microphone detection, lies in the difficulty of determining the sticking (i.e. which hand played which note) of the piece.

In addition to this, a drum pad mounted sensor would require either a method with which the user could install the sensor on their desired playing surface (e.g. their practice pad) or that the sensor be built into a custom practice pad. If the installation design was used, variation in surface (such as differences in practice pad size, material, etc.) could have an impact on the data being sent. A custom practice pad with a built in sensor would eliminate that risk, but would prevent the user from using their own practice pad or using the device on a different surface, such as an actual snare drum.

### Stick Mounted Motion Sensors

This method would either involve two sensors that could be attached to drumsticks or custom drum sticks with sensors built into them. The concern in both cases was usability. Regarding the former of the two cases, we did not want to have an external fixture on the drum stick that might inhibit the user's ability to play. Regarding the latter, we wanted the user to be able to use their own sticks with the product, since there are variations in size, weight, etc. Additionally, if a stick with a built in sensor were to break, it would require a repair or replacement stick.

## Wristband Mounted Motion Sensors

Very similar to the final design that was used, wrist mounted motion sensors fulfilled the requirements that the previous considerations did not. During the design phase of the project, this was the decided upon method. The limitation with wristband mounted motion sensors only became apparent after this design had been tested with hardware. It was found that wrist mounted sensors yielded less useful motion information than sensors mounted on the back of the hand, which was the motivation to switch from this design to the final design: glove mounted sensors.

## Concepts Selected

### Design

#### Glove Mounted Motion Sensors

Two glove-mounted sensors allow accurate detection of rhythms, rudiments, and sticking. By mounting the sensors on the back of the gloves, the user's playing ability is not inhibited at all. The user has the benefit of being able to play on any surface with any pair of drumsticks they choose. Naturally, ambient noise does not have any effect on the application. This design fulfilled all of our requirements in the most effective way.

### Hardware

#### Intel Edison

The Intel Edison was chosen for three key reasons: versatility, functionality, and compact form factor, in addition to the extremely low cost.

##### Versatility

- Built-in wireless connectivity over Wi-Fi and Bluetooth LE.
- Ease of development across a number of environments and languages, including Arduino, Python, and C.
- Wide array of Sparkfun breakout boards for additional functionality,

##### Functionality

- For such a small device, the Edison is quite powerful.
- Intel Atom SoC, dual-core CPU @ 500 MHz
- 1 GB RAM, 4 GB Flash

##### Form Factor

- Barely larger than a couple postage stamps, the Edison can easily be worn in the

- wrist or back of the hand without being too much of a burden.
- Allows for accurate data collection without impeding the playing motion.
  - Strike detection/data logging, motion classification.

## Sparkfun Blocks

The Edison is a powerful and versatile platform, but in order to achieve the maximum performance out of the system, there were a few key additions that needed to be made. Sparkfun makes a number of blocks that can be assembled into a stack beneath the Edison for expanded functionality, all communicating over an I2C bus. We chose the following three:

### Base Block

- Includes Micro AB USB ports to allow for attachment of various peripherals.
- Console port for programming access and OTG port for storage.

### 9DOF IMU Block (9 Degrees of Freedom inertial and Magnetic Unit)

- Houses a 3-axis accelerometer, gyroscope, and magnetometer, as well as a thermometer.
- Sends data through I2C bus in standard (100kHz) and fast (400kHz) modes.
- Adjustable ranges for each sensor ( $\pm 2 - 16$  G, etc.).

### Battery Block

- Single cell, rechargeable LiPo battery rated at 400mAh provides 4-8 hours of continual use.

## Software

### Python

Python was the language of choice on the client side (the Edisons). We chose Python because it was the language used in Intel's example of communicating via Bluetooth Serial Port Profile and reading data from the IMU. Furthermore, the scripting nature of the language encouraged our rapid prototyping and test driven development methodologies.

### C#

We chose C# as our server-side language because it is a mature, managed language that provides extensive interaction with Windows, our target platform. It has a healthy ecosystem of APIs to support our needed functionality and is familiar to and easy to use for programmers that have used C++ and/or Java. Our chosen GUI framework, WPF/Windows Forms also leverages C# in the code behind, so there would be no need to learn another language in our limited programming time.

## MusicXML

There are several file formats for storing sheet music (or musical data that can be rendered as sheet music), MIDI being most famous. Few of these formats have the ability to store information related to snare drum specific musical elements, such as sticking and rudiments (MIDI is one that cannot). Of these few formats that can store this required information, only one format remains actively developed, supported, and used: MusicXML.

MusicXML is extremely versatile and relatively easy to use. Additionally, popular notation software (such as Finale and Sibelius) can open and edit MusicXML files. Consequently, if the user has access to one such program (as many musicians who write music do) then they can edit any output from our software.

## Protocols

### Bluetooth Low Energy (4.0)

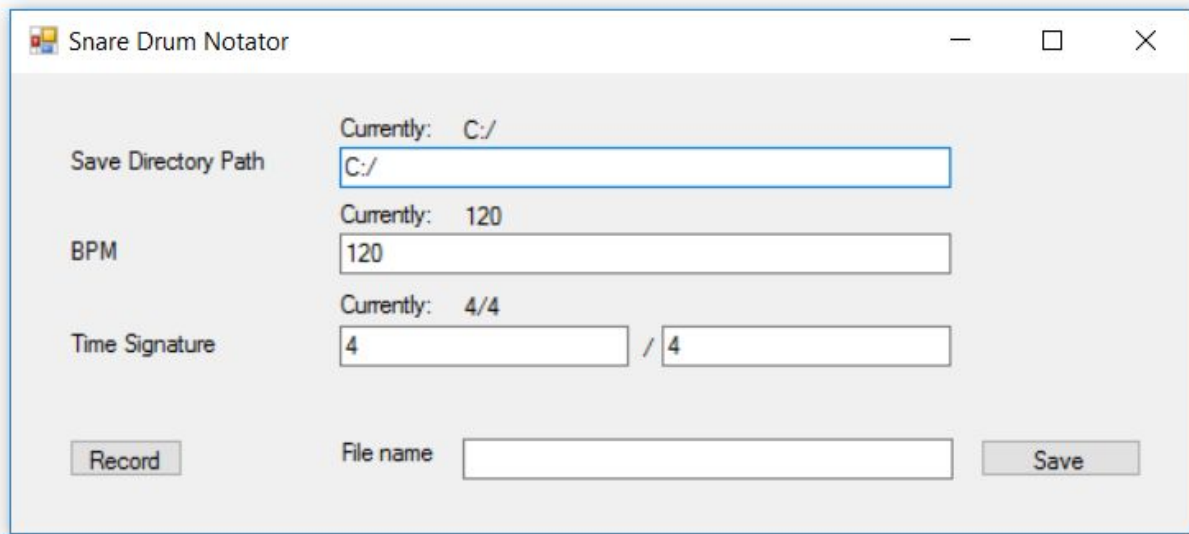
Bluetooth 4.0 (BLE) is a wireless personal area network technology that was utilized in this project to allow communication between the Edisons and the PC. This was done by leveraging the Serial Port Profile (SPP), which makes use of Radio Frequency Communication (RFCOMM) protocol to emulate RS-232 serial ports. Though it took some troubleshooting, we were able to achieve two-way Bluetooth SPP communication between the Edisons and the PC.

On the Intel Edison the library used to implement Bluetooth was the pyBlueZ interface build in python on top of the linux BlueZ library. On the server side the C# 32 feet library made by In The Hand was utilized for initiating and continuing communication.

## System Architecture

### GUI

Our GUI uses a simple and clear design to receive and express information.



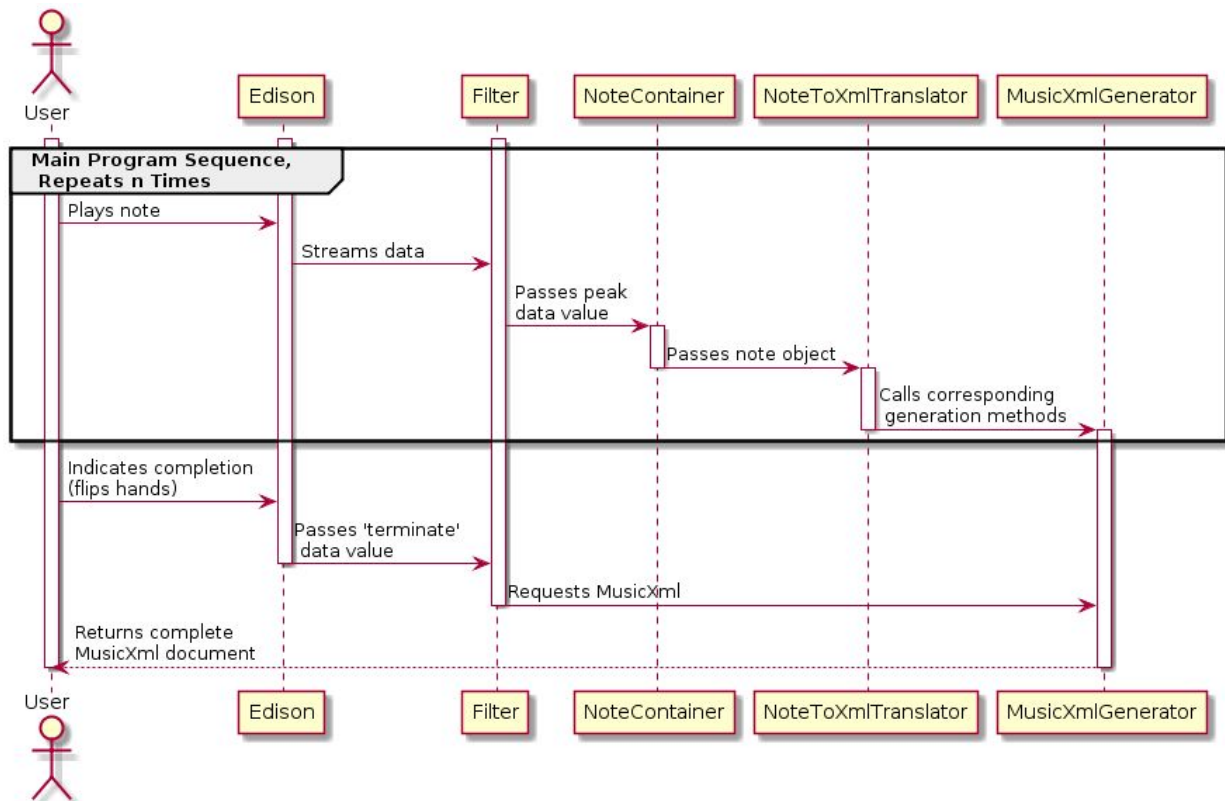
The text entry fields take what their labels suggest. The "Currently:" area shows you what value the associated variable currently has, just in case the user has entered nonsense into that field. To the right of each field, the word "invalid" is shown if the entered value is invalid.

### Bluetooth Communication

Bluetooth communication added a wireless connection between the user and the computer. This prevents the musician from being hindered while playing and decreases the hassle of setting up the device. In addition it allows for a hand gesture to be added so that the musician can stop recording by simply flicking one's wrists quickly. This prevents extraneous notes from appearing because the user no longer has to click stop to stop recording.

### Data Processing

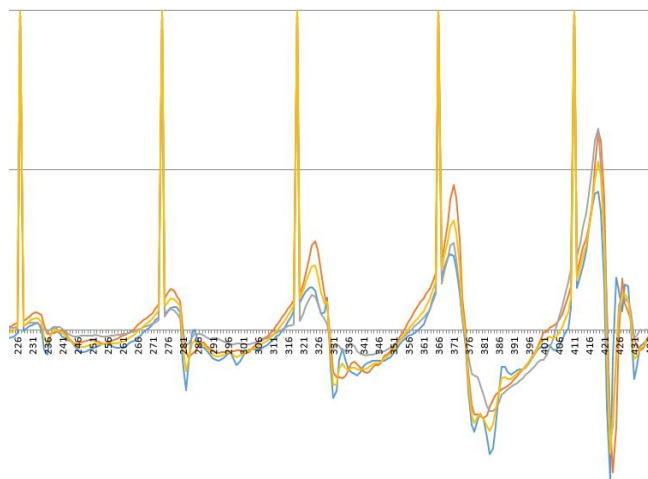
The notes the user plays are deciphered via processing the gyroscopic data streamed from the Edison modules to the PC. This is accomplished using a multi-step procedure involving several classes and components.




*Sequence diagram showing high level view of data processing*

## Filter

Through graphing of the data, it was determined that, with the exception of a few, specific cases, a spike or peak in the mean of x and y above a certain threshold corresponded with a note that the user played. Consequently notes are detected by passing the gyroscopic data through a filter, which looks for peaks in the mean of x and y above a minimum value.



*Example snippet of gyroscopic data sent by one Edison*



The above image is an example of gyroscopic data one Edison would generate. The different colored lines correspond to the different axes (x, y, z). The tall, steep spikes are artificial values inserted into the graph to indicate where the beat is. The 5 peaks that occur slightly after each beat were the result of the user playing quarter notes with a crescendo. The 6th and final smaller peak was one of the aforementioned “few, specific cases” in which a peak was created when a note was not played.

## Note Class

When a note is discovered, the datum at the top of the peak is saved and a Note object is instantiated from it. The note’s measure, beat, value, dynamic, sticking, rudiments, etc. are then determined. Many of these attributes are calculated via contextual information involving its neighboring Notes. This means there must be some delay before these calculations can be made, to ensure all nearby notes have been successfully been detected and gathered.

## NotetoXmlTranslator Class

Once the note attributes have been determined, the Note objects are handed over to the NotetoXmlTranslator class, which calls the appropriate MusicXmlGenerator methods to create the correct MusicXML for each note.

## MusicXML Generation

### MusicXmlGenerator Class

Generates MusicXML through use of the Decorator design pattern wherein each basic note is encapsulated in any number of rudiments and other modifiers and only the outermost layer need be interacted with to get the desired notation.

Supports whole notes through 32ndth notes, accents, backsticks, buzz stroke rolls, crescendos, decrescendos, dots, double stroke rolls, dynamics, flams, flat flams, rests, rim clicks, rimshots, sticking, strong accents, and tuplets.

Targets the Soundslice renderer and some pieces may not work with other MusicXML renderers.

## Design Evaluation

### Testing Procedures

Unit tests were written for each functional component of the data processing and MusicXML generation procedures. Integration tests were written for testing all components together, in which artificial data was fed to the software and it took all steps necessary to generate the MusicXML for that data.

In addition to unit tests, an application was written to output a Windows Excel file containing all of data sent by the Edisons and a graph of that data. This application was used during live testing, making it easier to diagnose errors in the code. The graph of the data could be examined if the program output incorrect MusicXML (e.g. the program produced dotted sixteenth notes when eighth notes were played) to help understand what features of the data caused the error.

### Test Results

By the final build of the project, all unit and integration tests succeeded using artificial data. When conducting live testing (using “organic” data generated by a user actually playing rhythms) the correct MusicXML was generally written, with some exceptions when faster or more complex rhythms were played.

## Future Work


### Rhythms and Rudiments

Our future work includes but is not limited to supporting additional rhythms and rudiments that we had to eliminate from the project due to lack of time, resources, and expertise. Such rhythms include support for arbitrary tuplets, and such rudiments include rimshots, rim clicks, stick clicks, backsticks, and buzz rolls, and alternating flams. This will require a significantly higher poll rate and somewhat increased sensitivity. An idea worth pursuing is using a neural network to learn what the user is playing as they play and improve itself continuously.

### Edison Setup

Currently, setup to start recording and writing sheet music involves a terminal session to each Edison, executing some system and bluetooth commands, and launching the python





script that streams the IMU data via bluetooth. Future work would include automating that process so that the user does not have to “get their hands dirty”.

## GUI

A big quality of life improvement would be to roll our own renderer for the MusicXML. Currently, we rely on an online tool. If we had our own custom renderer we would be able to also support concurrent and partial rendering of the sheet music, giving the user instant feedback on what they play.

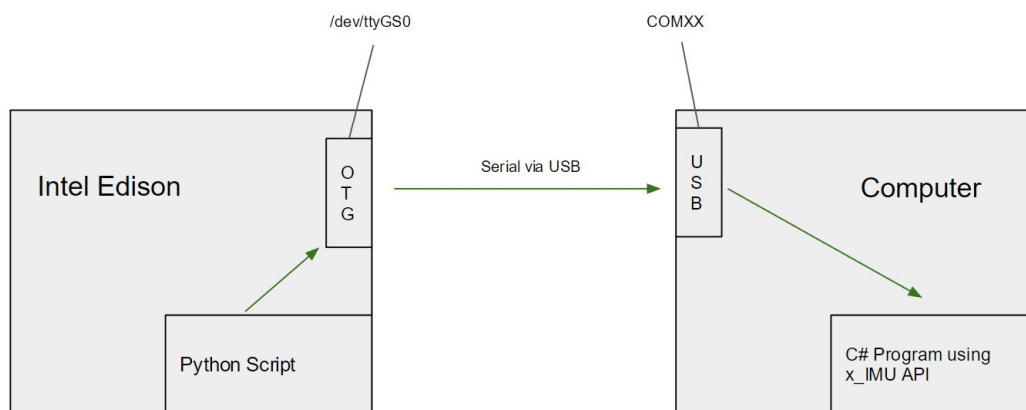
## Metronome

Naturally, a metronome plays a key role in the function of the system. There has to be a reference point for the beat, otherwise it would be impossible to accurately know what the user is playing. We currently rely on the computer for the metronome, but that creates a struggle between the computer’s perfection and the human user’s imperfection. Consequently, we believe an alternate route worth pursuing is putting another sensor on the user’s foot and use their foot tapping as the beat. This would make the user their own reference point and likely greatly reduce the guesswork of the detection and classification algorithm.

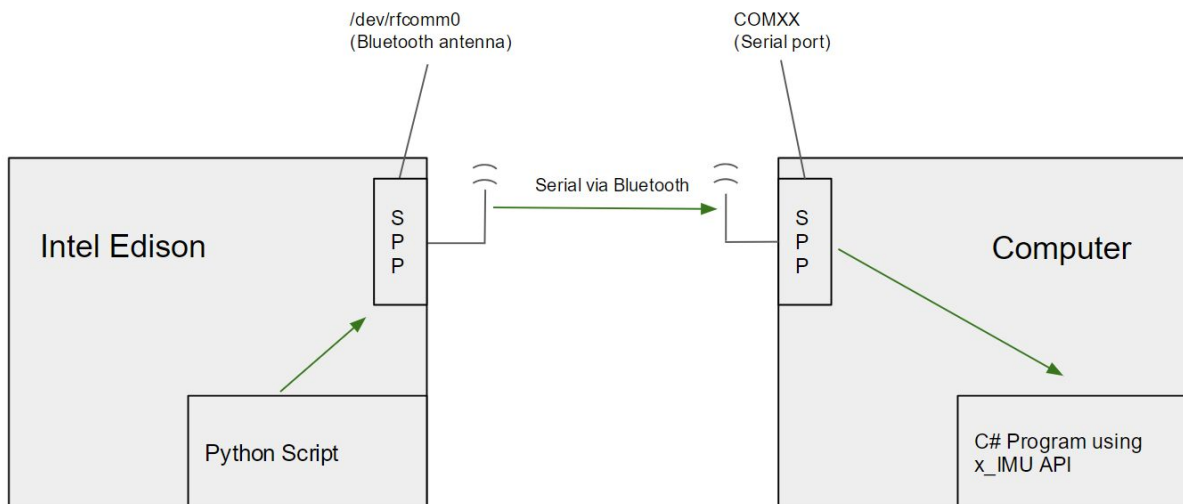
## Appendix

### Drawings





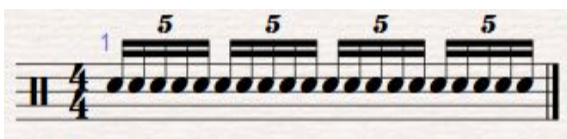
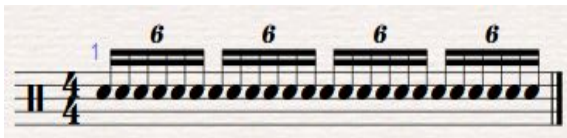

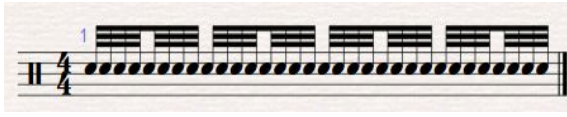

Serial Communication Diagram  
between Intel Edison and Computer



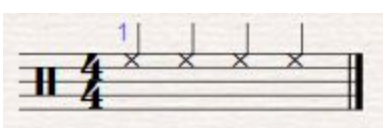



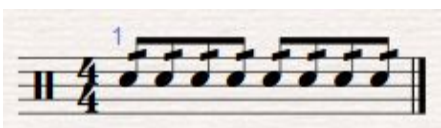
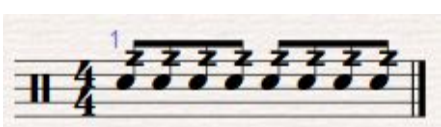
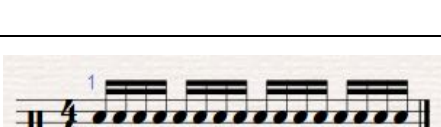


Bluetooth Communication Diagram  
between Intel Edison and Computer



## Tables and Figures

Rhythm	Description	Notation
Quarter Notes	One note per beat, evenly spaced.	
Eighth Notes	Two notes per beat, evenly spaced.	
Triplets	Three notes per beat, evenly spaced.	
Sixteenth notes	Four notes per beat, evenly spaced.	
Quintuplets	Five notes per beat, evenly spaced.	
Sextuplets	Six notes per beat, evenly spaced.	
Septuplets	Seven notes per beat, evenly spaced.	
Thirty Secondth Notes	Eight notes per beat, evenly spaced.	
Quarter Note Triplets	Three notes per two beats, evenly spaced.	

Rudiment	Description	Notation
Accents	Accented notes are marked with '>'. They are played louder than the rest of the notes.	
Rimshot	A simultaneous hit of the drum and the rim. Rimshots are usually marked with a thick x and a capo (^).	
Rim Click	A hit on the rim of the drum. Marked with a thin x on the top space of the staff lines.	
Flam	A "flammed" note is preceded by a very soft hit with the other hand played a split-second before the main note.	
Flat Flam	One note played simultaneously with both hands.	
Backstick	A hit on the drum played with the back of the stick. Marked with an inverted triangle on the second line from the top of the staff.	
Double Stroke Roll	Notes marked with a slash through their stem are doubled in value and played with the same hand of the original note.	
Buzz Stroke Roll	Notes marked with a 'z' through their stem are played with a pressing motion, causing heavy vibration and a large amount of softer hits.	
Crescendo/Decrescendo	A gradual increase/decrease in the volume of the notes being played.	

## Computer Programs

### Programming

- Visual Studio
- ReSharper
- Git

## Data Sheets

### Intel Edison

[http://download.intel.com/support/edison/sb/edison\\_pb\\_331179002.pdf](http://download.intel.com/support/edison/sb/edison_pb_331179002.pdf)

## Original Project Schedule

Initial Testing/Pattern Classification:	Oct-Nov
Communication Protocols:	Nov-Dec
Field Testing - School of Music:	Jan-Feb
Revisions/User Interface:	Feb-March
Final Testing/Revisions:	March-May

## Actual Project Schedule

Concept Development:	Sep-Dec
Hardware Assembly:	Oct
Fiddling with Bluetooth:	Nov-March
Note Identification and notation:	Feb-Apr
Testing:	Feb-Apr

## File Management Overview

Our supplemental files can be found in our Google Drive.

All our code can be found in our private Visual Studio Online Git repository.