MUSIC AND WEARABLE TECHNOLOGY

Major Qualifying Project

Advisor:

William R Michalson

Written By:

Matthew Barreiro

Erin Ferguson

Liam Perry

A Major Qualifying Project

WORCESTER POLYTECHNIC INSTITUTE

Submitted to the Faculty of

Worcester Polytechnic Institute

In partial fulfillment of the requirements for the

Degree in Bachelor of Science

in

Electrical and Computer Engineering

August 2016- March 2017

Table of Contents

1.Introduction

Iso-principle is the concept of shifting someone’s mood through music. This is done through “entrainment,” which is the synchronization of the body’s rhythms to an outside stimulus. One such source of rhythm is music. Music has been shown to alter a subject’s mood and physiology. Certain types of music can induce a positive shift in the mood. These positive mood shifts can be used to improve exercise performance in casual exercisers. Music has also been shown to take mood down. For example, music’s interaction with the autonomous nervous system can cancel the physical effects of an anxiety response.

The goal of our product is to select and play music on Spotify based on processed biosignals to bring about a shift in mood or physiology. We plan to accomplish this by creating an Android app that interfaces with a smartwatch, like an Moto360, and Spotify.

The market of wearable devices has expanded in recent years, leading to a wealth of information about our physiology in real time. Given this development, there is plenty of opportunity to use that information to do something new. Music has been shown to alter a listener’s physiology and, under particular conditions, improve physical performance. By using a wearable to monitor physiological values, music can be used to shift those values to a more desirable state.

A system that can achieve this functionality would consist of a wearable to measure and send the physiological values, and a device that can play music and can use those values to choose the appropriate music to play. The system that we will design consists of a smartwatch, an android smartphone, and an Android app that connects the two, chooses the music, and plays the music.

2.Background

2.1Music and Psychology

Music influences physiology and psychology. “Iso-principle” is the concept of matching a patient’s mood to music and then using the music to alter the mood in some way. [1] This is done through entrainment, which is when the body’s rhythms, such as heart rate and respiration, get synchronized to an external rhythm, such as music. [2] The two largest factors in psychological effects of music are melody and rhythm. A melody is the most conscious way that a listener will be affected by music, because a melody is something that the listener can focus on. A melody also stimulates the imagination and can be easily remembered. Melodies that employ brass instruments and electronic sounds have been found to cause a positive shift in mood, while melodies that mostly rely on harps, chimes, and strings cause a downshift in mood. The music’s tempo has the largest effect on mood, even though the effect is subconscious. Music with tempos greater than resting heart rate (60-100 beats per minute (BPM)) [3] are stimulating and elevate mood. Those that are less than resting heart rate can soothe and lessen mood.

Music has also been shown to distract from negative stimuli. This is why music can enhance exercising in untrained athletes. Music can distract from anxiety attacks by influencing the autonomic nervous system. When someone experiences anxiety, their heart rate, respiratory rate, and blood pressure increase. When music is played, the vagus nerve, which is part of the autonomic nervous system, tells the parasympathetic nervous system, which controls heart and respiration rate, to slow down, ending the anxiety attack [4]. Music is also thought to release endorphins, which elevate mood and stop anxiety. [5]

2.2 Physiological Relationships

In addition to influencing the psyche, music also has measurable effects on physical systems of the body. The team developed a series of experiments to learn about music's effects on physiology and the relationship between different biological parameters.

We designed experiments to determine the effect of music on heart rate. At first, we utilized an Asiawill Pulsesensor for Arduino, seen in Figure 1, to take heart rate. With this set up, one team member listened to 2 minutes of a song with either high or low BPM and high or low valence, defined in Table 1, as well as 2 minutes of silence, while not moving. We found that there wasn’t that much of a difference in average heart rate under these conditions. The team was skeptical about this finding because of the variability in output. For example, a heart rate of 65 BPM, determined by the palpation method at the wrist, was read by the fingertip sensor as some value between 60-75 BPM. Another issue with this sensor is that the sensor must be perfectly aligned on the fingertip to read any data at all. This presented issues, particularly after our running trial, as it required a physically excited team member to come down from a run and perfectly align a small sensor on his fingertip in the short amount of time before his heart rate began to drop. To overcome this obstacle, we decided to abandon this sensor and instead get the data from a team member’s Apple Watch.

FIGURE 1: PULSESENSOR

The Apple Watch has a feature where a user can manually initiate a workout, labeled as "create new workout." During this workout, the Apple Watch constantly records heart rate measurements. Once the workout has ended, the user has the ability to export full exercise history, and send this filet via email to a computer for processing in Excel. We used this method for our second experiment. During this experiment, one team member sat and did not listen to music in for thirty minutes. Then that same member listened to music with low, but increasing, valence and a constant tempo of 140 BPM for 30 minutes. Heart rate was taken constantly during this trial. The results of this experiment showed that variance in the heart rate decreased when the subject was listening to music. This data supports the idea that music can help regulate cardiac function. A graph of the variance in heart rate can be seen in Figure 2 below. The variance was taking using a windowing technique. Only the most recent 25 data points are considered. Variance of heart rate without music is displayed to the left of the red divider and variance of heart rate while listening to music is displayed to the right of the red divider. It is important to note how much lower the variance is once the participant started listening to music.

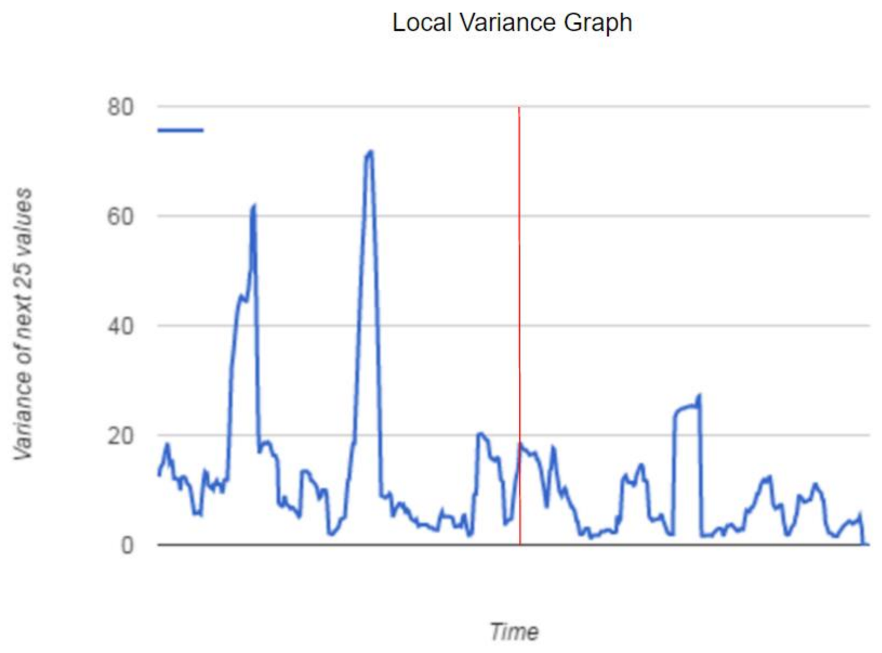


Figure 2: Local Variance of Heart Rate with and without Music

The team then decided to see if it was possible to infer other biological relationships from data collected by the smart watch. We decided to study respiration rate because this is something that changes with exercise. The hope was to be able to correlate the respiration rate to heart rate and display it in the app. The team decided to pursue three different types of sensors to capture respiration rate. The types of sensors tested were a lapel microphone, an infrared sensor, and a conductive rubber cord.

A lapel mic taped under the nostril was thought to be able to record breath sounds for analysis. This method captured the actual breath sounds and analyzed them to find the respiration rate. The breathing patterns were recorded using Audacity, an audio editing program, and imported into MATLAB for analysis. The script took a Fourier Analysis of the data to determine the frequency of respiration. Unfortunately, this method was extremely susceptible to motion artifacts because the microphone was just taped to the face during exercise. Recording the breath sounds with such a low-quality microphone also made the data very noisy. The data that was Fourier transformed was not clean, so the graph of the transform did not have a peak at any discernable location. This made this an inappropriate way to collect respiration rate data.

Figure 3: Lapel Microphone

A SHARP infrared proximity sensor was the next method tested. This method used the location of the front of the chest to identify when the person breathed. This was set up by placing reflective tape on the test subject and setting the infrared sensor on a flat surface facing the test subject. The infrared sensor was tuned to sense the small changes in the position of the chest during breathing. The reflective tape made it easier for the sensor to "see" the chest, since dark colored clothes absorb light. The changes were recorded using an oscilloscope. When the person breathed in, the chest expands, so the infrared sensor outputs high. When the person breathes out, the chest contracts, and the sensors outputs low. Though it was possible to get readings from this sensor, this idea was abandoned because it was not practical for use during exercise since an exerciser's body moves unpredictably during a workout. During the initial test, both the sensor and the subject remained stationary. This setup was not practical for a real world situation. This led the team to develop a motion resilient method of measuring breath rate.

Figure 4: SHARP Infrared Sensor

The method we ended up using is a conductive rubber cord. This measures the changes in the size of the chest during respiration. The test apparatus was made up of an adjustable elastic band and the conductive rubber cord. This cord acted like a potentiometer in a voltage divider because the change in the stretch of the cord changed the resistance. The voltage was measured across the variable resistor, so the voltage output correlated to the degree of stretch. The data was collected using an Arduino UNO. The rate of change of the voltage was correlated to respiration rate using a MATLAB script.

This respiration sensor is attached to the person using safety pins and an elastic waistband. The waistband size is adjusted to wrap around the test subject's chest. The cord is electrically connected to the rest of the circuit using alligator clips. This was the best solution explored because it is portable and is not easily disrupted during exercise.

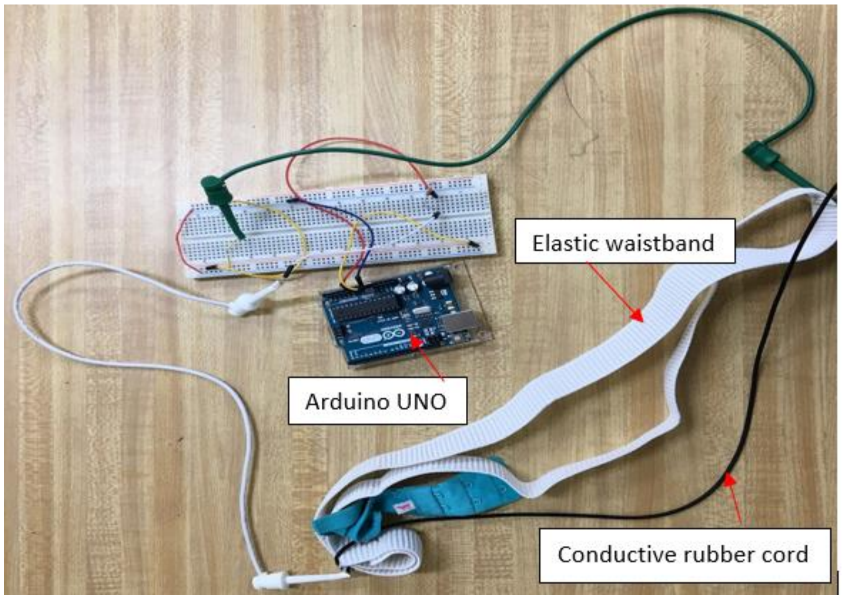


FIGURE 5: RESPIRATION SENSOR

Some problems with this method were the fragility of the cord. The cord could only be used for a few experiments before it wore out and snapped. As the experiments went on, the cord became more stretched out, so the output voltage was not consistent. The general proportions were still similar so it was possible to use this method to get usable respiration rate data.

Once the team decided on using the conductive rubber cord method to get data, the team designed a variety of experiments to determine is respiration rate was correlated to heart rate. It was determined that while there is a relationship between the heart rate and respiration rate in some participants, it is not strong enough to generalize for all cases. The team decided not to try to display this information in the app. More information about these experiments can be found in Appendix A.

After the team decided to use heart rate as the parameter to track, the team pursued measuring running pace while listening to different types of music. The runners' paces were used as the metric for better or worse performance. We hoped to find what qualities of music changed someone's running pace from their normal pace, without music. To complete this experiment, we compiled 3 different playlists with different beats per minute (BPM) and different valence levels. These terms can be found in Table 1 below. Participants listened to our playlists for 10 minutes while running around an indoor track. The runners' lap times were recorded. All participants were volunteers. Because there was no incentive and because all participants were students, not all participants completed the full trial. The sample size was also extremely small.

Table 1: Selected Musical Qualities

Echo Nest Song Attribute	Definition
Beats per Minute (BPM)	Tempo of the song
Energy	The higher the value, the more energetic the song
Dancebility	The higher the value, the easier it is to dance to
Valence	The higher the value, the more positive the song’s mood

The first playlist, Playlist A, had high BPM (160-168) and low valence (3-7). Generally, participants were slower than their control run with no music. We can accept this result because lower valence means lower energy. This makes the participants less “pumped up.” A lot of the music in this playlist had a lot of rests and no distinct melody. One participant described this music as “empty space.” Another said it was “boring...and made [running] more annoying.” The average pace for the 2 participants for the control experiment was 1:09/lap. The average pace while listening to the first playlist was 1:11.3/lap.

The second playlist, Playlist B, had high BPM (160-170) and high valence (71-97). We expected this playlist to cause the runners to go fastest because it has a high BPM for the step rate to sync to and the highest energy level for the runner to latch on to. The runners’ response to this music was positive. One participant described it as “catchy.” The average pace for the 2 participants in the control experiment was 01:09/lap. The average pace while listening to this playlist was 01:07.6.

The next playlist, Playlist C, has low BPM (47-65) and high valence (83-90). We expected this section to have an average lap time between the previous two. We were surprised to find that this produced the fastest laptime of all of the trials. The average lap time for the two runners in this trial was 01:06.65. The participants described this music as “groovy.” Most of the music in this playlist is R&B music, which research has shown to be an effective motivator during workout.

The final test used a compilation of all of the different types of music previously tested. While the previous experiments hoped to give us a general sense of the performance with different music, this test was the most powerful because it showed how the performance can change during one workout. Only one participant was available for this test, so all results are suggestive, but not conclusive.

The graph below shows the lap times of the runner when listening to different types of music. The green and blue segments have the quickest lap times are both have music with High Valence. This is in line with the data from the previous set of tests, which shows that lap time decreases with high valence music. The orange section, which has low valence and low BPM, has the longest lap time, which suggests that this may be the best music to use for a future cool down feature in the app.

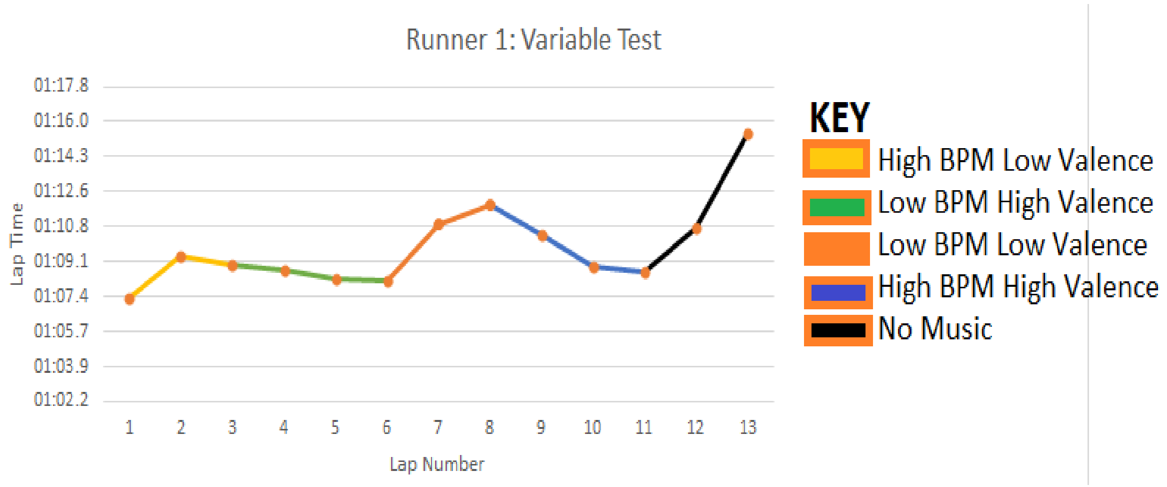


FIGURE 6: LAP TIME WITH DIFFERENT MUSIC TYPES

The data we collected has led us to draw some reasonable suggestions. We believe that valence has a stronger effect on performance than BPM. This may be because valence is a measure of the mood of the song. Music with a melody that someone can latch onto, which uses electronic instruments and brass instruments, is shown to increase performance because it increases the mood of the situation. Exercise performance is increased when the brain is distracted by "positive" music. The increased valence increased the positivity of the situation, and distracted the runners to help them reach their best times.

Some limitation of the data include limited sample size. All of the participants were students. Because of this, not all participants finished the trial. In addition, all participants have similar health styles. We did not have a wide range of participants. If this project was continued, it would be advisable to get this project IRB certified. This way, it would be simpler to recruit a wider panel of participants, instead of just asking peers for a favor.

2.3 Music and Physiology

In a recent study by Tohoku University, participants performed exercises on a stationary bicycle. Participants who listened to music during the exercise had a faster heart rate recovery than participants who did not listen to music. Usually during exercise, the parasympathetic nervous system's activity decreases and heart rate increases. This implies that music increases the activity of the parasympathetic nervous system. The implications of this are that listening to music during exercise could be beneficial to cardiac health because the parasympathetic nervous system can slow the heart down therefore reducing cardiac stress. [6] In a 2012 study at Sheffield Hallam University, participants using music required 7% less oxygen to complete a workout on a stationary bike than participants who did not listen to music. This also supports the claim that music activates the parasympathetic nervous system. [7]

A study from 1991 indicated that high valence, happy, music induced lower skin temperature, measured at the fingertip, than low valence, sad, music. This is the opposite of the results of previous studies, however. The 1991 study makes a point to distinguish between valence and arousing music, however, and implies that previous studies did not make this distinction. It then argues that it is possible that increasing finger temperature observed in earlier studies was due to the arousing level of the music, not the valence of the music. [8]

2.4 Prior Art

By looking at prior art we understand the scope of what's happening inside wearable devices.

To understand what kind of mechanics go into mobile devices like our product, we watched and evaluated a teardown of the Fitbit ForceTM [9]. All of the components were understandably small. The battery, in particular, was listed as having a capacity of 50mAh. Its other components were also tiny, including an accelerometer, a microcontroller, barometer, and battery charger; however, the Bluetooth antenna is much larger compared to the rest of the components on the PCB.

In addition to fitness trackers like the Fitbit, smartwatches were also researched. The primary focus was on watches compatible with Google's Android phone operating system. There are two major watch operating systems in the Android wearable market. The first is Google's Android Wear, which was initially released by Google in 2014. The main competitor to Android Wear, in terms of operating systems used on full-featured smart watches, is Tizen. This Linux-based OS has existed for longer than Android Wear, but has only been used in wearables since 2014 [10]. Unlike Android Wear which was developed specifically for wearables, Tizen is a general purpose, Linux-based OS that runs on a wide variety of devices [11].

Android Wear devices were selected for further evaluation due to the compatibility between these devices and Google Fit, a service that syncs health data to other devices and software for easy access. We narrowed our search down to three devices: the Asus Zenwatch [12], the second generation Motorola Moto360 [13], and the Huawei Watch [14]. The significant common feature between all three of these devices, excluding the Android Wear OS, was the optical heart rate sensor. Compared to the previously mentioned device from Fitbit, the batteries on these devices have a much higher capacity (ranging between 300 and 400 mAh). All three devices have accelerometers or similar multi-axis sensors, and the Huawei watch has the added benefit of containing a barometer as well. It is fairly safe to conclude, then, that these devices are about equivalent to the device from Fitbit, at least in regards to available sensor information.

In 2015, Spotify debuted a new feature called Spotify Running. This service picks upbeat songs at the same tempo as the runner's step rate using sensors on the smartphone. It works with step rates which range from 140 -190 steps per minute, which is a light jog to a sprint. By syncing with the runner's step rate, the service aims to help runners keep their pace. The running playlists are chosen based on the user's listening history and their stride rate. It does not use use biological parameters to select playlists. [15] The service is available to both free and premium Spotify users. Many users found the product interesting, but didn't find it useful because it would turn off during slower periods of exercise, such as warm up and cool down, and was not usable for any other form of exercise. [16]

Rockmyrun is an app that uses both heart rate and step rate, collected from a smartwatch, to select playlist with music from a variety of different genres. This can be used for a variety of exercises because it does not rely only on step rate, like the Spotify Running. This app uses pre selected playlists and does not pull from songs in the users' library. [17] Some users were disappointed at their lack of control of the music and others were not satisfied with the subscription service. People found the monthly fee unreasonably high and were disappointed with how the ads on the free service disrupted the workout. [18]

2.5 Market Research

The wearables market is growing very quickly. It is predicted that there will be 500 million wearable devices in the next two years. Approximately 300 million of these devices are Wearable Fitness Trackers (WFT). [19] Those who are willing to buy a device expect to spend an average of $300. Those who were not interested in purchasing a device said that their largest worry was short battery life. [20]

As of 2016, smartphones are the most used technology among adults. This has led to an increase in smartphone compatible technologies, including wearable technology. Those who are interested in buying a smartphone compatible WFT are interested in a bracelet with continuous monitoring. [21] The largest reason that consumers are interested in one of these WFT devices, is the perceived health benefits. These stem from the societal pressure the be fit and healthy. Consumers' perceived outcomes for using a WFT included "lose weight,..., and live a healthy lifestyle." Based on a study conducted in mid 2016, WFT user were more likely to live more active lifestyles than before they started to use the devices. [22] One of the reasons why these devices are so effective is because it allows each user to learn about his or her own data and set individualized goals. This increases the user's motivation for use and participation in fitness activities. [23]

In the last 5 years, paying Spotify users have grown from 2 million to 40 million. This number has increased largely because of the use of the Facebook app, which has 10.4 million daily users. Music is streamed in a variety of applications including exercise. According to one study done in 2012, over 90% of college students listen to music during exercise. Of these participants who used music, most listened to fast music with heavy rhythms, like hip hop or rock, during exercise.

2.6 Technical Research

From a technical standpoint, there are several areas that require research. We needed to investigate APIs, or Application Program Interfaces, which could be used to source information needed to recommend music based on some physiological data. We also had to research how we would collect this physiological data. In addition, we also looked into the Bluetooth wireless communication protocol. Finally, we looked into how we would combine all of this data into a working application.

2.6.1 Musical API Research

It is necessary to access audible properties of different songs in order to best recommend them. One provider of this information is The Echo Nest. This Massachusetts company, founded by two MIT Ph.D.s, provides information to many companies and services that help users discover new music [24]. One such company is Spotify, who actually acquired The Echo Nest in the first quarter of 2014 [25]. Historically, the primary way The Echo Nest has provided this information is through their Application Program Interface (API). In the time since Spotify's acquisition of The Echo Nest, most (if not all) of the features from The Echo Nest's API have been integrated into Spotify's Web API. In fact, as of May 31, 2016, The Echo Nest API has been discontinued completely in favor of the Spotify Web API [26].

There are several major components to the Spotify Web API; however the most relevant section of the Web API for this project is the section that allows us to "Get [Music] Recommendations Based on Seeds". This function allows for a "playlist-style listening experience" to be generated based on several variables [27]. The query has a few required arguments, such as seed_genres and seed_artists. The most relevant optional arguments all relate to the section listed as Tunable Track attributes. These attributes allow for tuning of the recommended music based on specific attributes of a track. The full documentation is listed in Appendix A.

It is important to note that other services exist that provide similar information regarding music. One such example, the Gracenote Web API, provides "a rich set of music metadata over HTTP to help power interactive experiences for any connected application" [28]. The API provides information such as genre and mood. Gracenote also provides a Rhythm API, which allows for "adaptive playlist, recommendation using seed track/artist/genre/era" [29]. While this API would potentially free us from the Spotify ecosystem, and potentially allow users to provide their own music files, it is also important to note that these APIs actually have the potential to over-complicate the system by requiring the use of several different APIs. This is because, in the case of allowing a user to provide music files, a content recognition system would have to be implemented. This system would be used to determine the content of the user provided file, and add it to some sort of database. This is necessary, as our system cannot provide recommendations based on physiological input conditions if the system is not familiar with the musical characteristics of the files the system is provided to play.

2.6.2 Fitness API/SDK Research

To track physiological values on a pre existing wearable device, we needed a fitness tracking platform. The one we looked into was the Google Fit platform. The Google Fit SDK, or Software Development Kit, consists of several APIs that are designed to "make building fitness apps and devices easier" [30]. It is designed to aggregate fitness data from multiple sources, allowing for the creation of a "fitness ecosystem". The APIs are provided in two forms: the platform-independent REST API, and the Android APIs. The focus of this project is the set of Android APIs.

On Android, the Google Fit APIs are built into Google Play Services, meaning they are natively supported by a large percentage of Android devices by default. In the case of Google Fit, the minimum required Android version to access the API is Android 2.3 (Gingerbread), however the Google Fit App requires Android 4.0 (Ice Cream Sandwich) or higher. Note that the latest version of Android is Android 7.1.1 (Nougat).

There are several APIs provided within the Google Fit Android API. The first of these APIs is the Sensors API, which provides access to raw sensor data collected by both the Android phone running Google Fit, as well as any "companion devices," such as wearables. The second API is the Recording API, which uses "subscriptions" to automate storage of sensor data "in a battery- efficient manner". The History API allows applications to access and modify previously recorded fitness data in bulk operations. The Sessions API enables storing data with metadata correlation to a specific session of exercise. There is also a Bluetooth Low Energy API, which allows access to Bluetooth Low Energy (BLE) sensors within Google Fit. Finally, there is a Config API, used to configure custom data types and modify settings within Google Fit.

Google Fit would be the primary point of interface between any application we develop and any wearable device in this project. While the use of a third-party API to implement this interface may seem to be overcomplicating the issue, it actually theoretically simplifies the process. The reasoning behind this claim is that, in theory, our app could be developed in such a way that most, if not all, Google Fit compatible wearables that contain the proper hardware could be implemented to work with our app with little to no extra work on our part. This is because any and all devices that have full Google Fit support should be passing information in the same way. This means that our application should be able to be built in a way that by supporting Google Fit, it in practice supports a wide variety of wearable devices. It is very important to note, however, that some wearable devices may require a user install an application on his or her phone in order for said device to communicate with Google Fit. Therefore, a device with native Google Fit support, such as an Android Wear device, will generally be preferable to a device that requires a separate app, such as several popular brands of dedicated fitness trackers.

2.6.3 Music App Research

In order to play the music for the end user, we needed to find an application that could fit our needs of being able to play Spotify playlists, but being modifiable to our needs. In our search, we needed to answer three questions: would the application be desktop or mobile based, would we write our own app or use an existing one, and if we were to use an existing app, what would we use?

The first question of deciding between desktop or mobile was focused on whether we would prioritize ease of development or the end product. The team had little to no mobile development experience, so a desktop developing environment would be easier for the team. However, as the end product would be on mobile, starting there would eliminate any transitioning hassle from desktop to mobile. We concluded that ease of transition would be the deciding factor here, in order to minimize our work to the final product, even if it may be harder to jump straight to mobile. If we found any application that would make the transition easy, we would develop on desktop first, otherwise we would start with mobile.

Next, we needed to compare making our own app to using an existing application. This required two steps: looking into how we could use Spotify API to make our own app, and what pre-existing applications already interface with Spotify. When we looked into writing our own app, we found that despite Spotify's API giving us the ability to use its playlists, there would still be an incredible amount of work involved to just create a barebones streaming application, implying building off of an existing application would be much simpler. Luckily, we found an application that interfaces well with Spotify called Tomahawk. It is a music player that interfaces with most used music streaming and collection services, including Spotify. As this program satisfied our requirements for the product, we didn't need to look any further.

Development for Tomahawk will be straightforward given that all of Tomahawk's source code is on GitHub. The mobile version is written in Java and Javascript, and the Tomahawk page on GitHub has directions for compiling the program. With the code being easily editable, the Tomahawk codebase is a great place to start for constructing our player application.

2.6.4 Bluetooth Hardware Research

To communicate from the wearable device to the music player, we needed to choose a wireless communication protocol. . Using a standardized protocol is reasonable in order to keep development costs manageable, and Bluetooth is one of the most prolific standards for short-range, low-bandwidth wireless communication. Bluetooth low energy, or BLE, was introduced with the Bluetooth 4.0 standard in 2010, and is particularly suited for mobile devices. This is due to the protocol aiming to reduce power consumption while conserving the same range as the full Bluetooth standard. With battery life being a main concern of wearable devices, BLE is the best option.

3 System Architecture

Based on our research, we determined the best course of action would be to purchase an existing fitness wearable, use it to transmit physiological data over Bluetooth, and use that data to play a suggested song using a custom music player. We decided to purchase a device instead of building a device, as it was determined that end users would be much more willing to purchase an app as opposed to a brand-new wearable. This would be especially true if a user already owned a wearable that contains the body sensors required to collect the data and the wireless interface required to be compatible with our app. This also allows us to focus more on the user experience and the recommendation engine, as opposed to having to devote time and energy to developing a new device. Fortunately, most wearable devices on the market are within our budget. An overview of the proposed system is provided below in Figure 6.

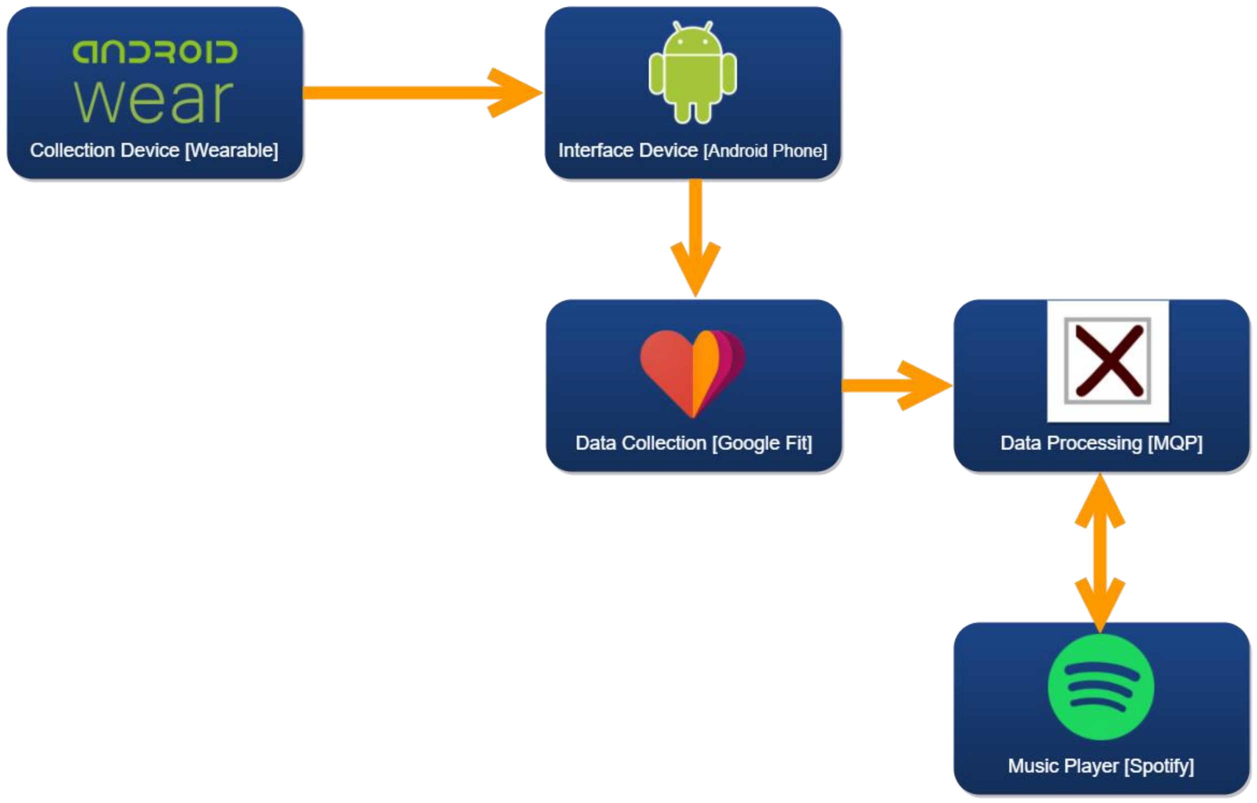

android wear
Collection Device [Wearable]
Interface Device [Android Phone]
Data Collection [Google Fit]
Data Processing [MQP]
Music Player [Spotify]

Figure 6: Proposed System Architecture

In this project, we will be using an Android phone to develop and test. This decision was based primarily upon the trend for Android Wear devices to cost much less than an Apple Watch for an iOS device. Additionally, Android Wear does work with Apple's iOS to some extent. Finally, the existence of Google Fit and its deep integration with Android was another major part of our decision. The music player that we will develop will decide on a song selection based on heart rate and step rate using an algorithm based on our physiology research.

3.1 Hardware Requirements

While interoperability with a large range of devices is desired, a minimum baseline must be set for compatible hardware. The user must supply this hardware, with the two primary components consisting of a monitoring device and an interface device. In our case, this means a smartwatch and a smart phone.

3.1.1 Monitoring Device

The first requirement for this system to work as intended is a method to collect relevant data about the physiological state of the user while they are exercising. A heart rate sensor and an accelerometer are a minimum requirement for the sensors for the project, but the more sensors that come with the device, the more data we have to work with to make suggestions. This decision was based on the results of our testing, presented in Chapter 2, which that found these two sensors seem to give the most meaningful data about a user's exercise activity. Heart rate data should be collected regularly in order to supply the song selection algorithm with up-to-date fitness data, so that recommendations can be more accurate. A heart rate sensor that requires little to no user interaction is preferable, as it allows the user to continue their exercise without having to stop to take their heart rate. While an electrocardiogram (ECG) sensor will provide more precise and accurate data than an optical pulse oximeter, either is acceptable. Step counts can either be collected from an onboard pedometer, or estimated from accelerometer data.

Data from these sensors will be collected through the Google Fit APIs. This project will focus on devices running Android Wear because Android Wear natively supports Google Fit. Therefore, any Android Wear device that contains the required sensors and properly conforms to the Google Fit API should work with little to no modifications to the developed software.

3.1.2 Interface Device

A device is required to interface with any sensors, and process the collected data. The chosen platform for this project is the Android mobile operating system by Google. This decision was based primarily on the trend for both Android mobile devices, as well as Android Wear devices, to be cheaper than the competitive iOS-based devices from Apple, such as the iPhone and Apple Watch. Finally, the native existence of Google Fit and its deep integration with Android was another major part of our decision. While not directly relevant for our project, iOS has some support for Android Wear devices, as well as the simpler Google Fit REST API.

A user must supply an Android device running a relatively recent update. The Android Developer Dashboard shows over 98% of Android devices that accessed the Google Play Store between November 28 and December 5, 2016 were running Android 2.3 or higher [31]. The device does not have to explicitly be a phone; however, the device must be able to maintain a constant connection to both the internet and any Bluetooth-connected sensors. The device also must be running an up-to-date version of Google Play Services. Generally speaking, this is the case with any Android device. The major exception would be if the device was running a custom build of Android, such as cyanogenmod [32], however it is often still possible for a user to install the required software dependencies on a custom build of Android.

3.2 Software Requirements

The application section of the project needs to be able to grab the data from the wearable device, calculate and play the right song depending on that data and other user input, and needs to be visually laid out in an intuitive way to the user. These requirements lay a baseline for coding the application.

3.2.1 Fitness Tracking

The Google Fit SDK contains several APIs that are designed to facilitate the creation of a fitness application. These APIs interact with both the mobile device the APIs are running on, as well as any compatible devices connected to the mobile device. In the case of this project, the most relevant API is the Sensors API. This API allows applications using Google Fit to directly access raw sensor data for devices connected through Google Fit. This is, to our knowledge, the only way to access instantaneous data, such as heart rate, through the Google Fit platform. Since this is a major component of our system, the Sensors API should be a focus in development.

3.2.2 Music Player

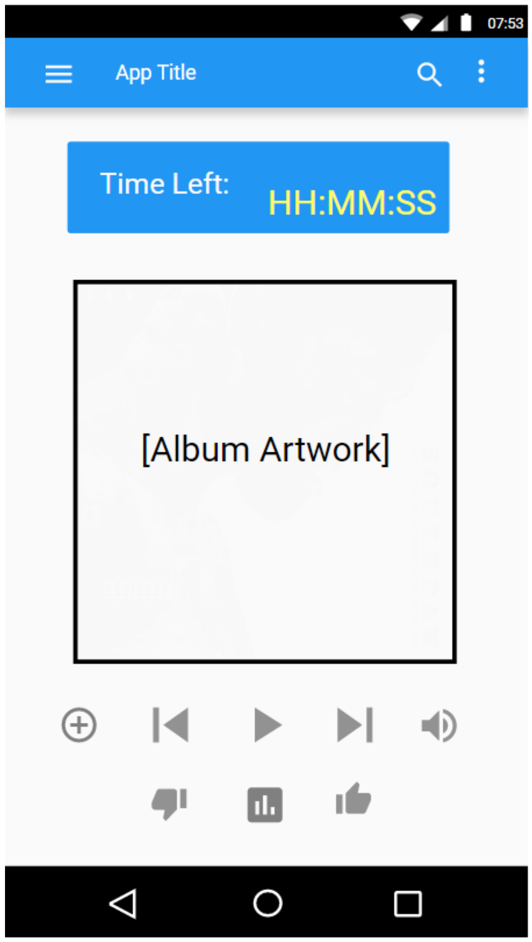
Once the application has the data from the wearable, the correct song needs to be played. This occurs in 3 parts: using an algorithm to evaluate the seed values, using those values to grab a song from Spotify, and then playing that song.

The algorithm will take the following inputs: heart rate and step rate, the current state of the exercise of the user, the user's previously determined resting and active heart rates, and any goals predefined by the user. This information will allow the algorithm to choose seed values designed to produce a list of songs optimized to achieve the fitness and health goals of the user. These seed values are the same attributes described in section 2.5.1. Once these seed values are determined, the Spotify API can be used to grab songs with those values, which can then be played by a Spotify-compatible streaming application.

Additionally, it is important to provide manual controls for the music player. There are many reasons why a user may need to pause the music and workout, so a pause feature should be implemented. Additionally, the algorithm may very easily grab music the user does not enjoy, so a system should be implemented to allow the user to dislike and/or skip songs. Finally, a system to allow a user to manually shift the tone of the workout should be implemented, particularly for users who may be dramatically shifting their exercise patterns throughout a workout.

3.2.3 UI Requirements

The user interface (UI) for the application will consist of two distinct interfaces, both based on the Material Design standards by Google [33]. There will be a basic interface on the Android Wear device screen, as well as a full-featured UI on the mobile device screen. Mockup examples of the Full UI and Wear UI are shown below in Figure 7 below



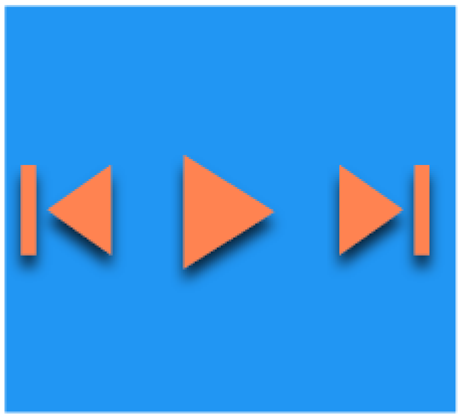


FIGURE 7:ANDROID APP (LEFT) AND WATCH APP MOCKUP (RIGHT)

Both screens will show heart rate data and time remaining in the workout. Additionally, both screens will allow for manually skipping songs, and for pausing and resuming the work if necessary. The Android Wear component must be much more compact, and will lack more advanced features such as configuring workout lengths and album art display. The Wear component should contain a quick workout start feature, which is preset by the user. For example, the user would be able to start a 30-minute workout, with hip-hop as the primary genre, and with cardio as the activity, by simply opening the app on the Wear device and selecting "quick start."

3.3 Other Requirements

To make money from the app, the app should be a free on the Play Store but supported with ad revenue. Only 33% of smartphone users said they would purchase an app, where as 93% of users would download a new app in the next year. [34] Approximately $1 per 1000 users is generated from free apps with ads per day. [35] In a study of 1600 smartphone users, 58% of smartphone users say they have downloaded a health app or a fitness app. 41% of these users would never pay for a smartphone app, while 84% would never pay over $6. Additionally, in the same study, 65% of health app users said that they use it at least once per day. [36] Based on this information, we plan to make the app free with an in app purchase to remove ads. This way, we can get ad revenue from people who would be more likely to download a free app and capitalize on those who would want to spend money on a fitness app.

3.3.1 User Supplied Resources

In addition to the hardware requirements mentioned in Section 3.1, the user will also be expected to own or purchase a few products in order for the application to function. It is expected that the user's device be configured with a Google account, as is typically the case for any Android device. This is required for proper authentication with the Google Fit service, and to access the Google Play store. Additionally, as Spotify requires a Premium account for API access, the user is expected to be a Premium subscriber. As of December 20016, Spotify offers Premium for $10 per month. Family plans and student discounts are also available, at $15 and $5 a month respectively. Finally, it is expected that the user has personal goals regarding the usage of the application, and be able to set time goals for their workouts.

3.3.2 Developer Supplied Resources

Google Fit requires a user authorize different applications on an individual basis. In order to implement this feature in a production environment, a release certificate is required. More precisely, the developers must generate a release certificate, and use the SHA1 fingerprint of this certificate register the application on the Google API console in order to request an OAuth 2.0 Client ID. This then allows users to authenticate with the developed application through their Google account, and permit access to body sensors.

Once a release build of the application has been created, it would be possible to list the application on the Google Play Store. There is a one-time registration fee of $25 for access to the Google Play Developer Console. Then it is possible to upload an Android Application Package (APK) and publish the application.

4 Hardware Design

We evaluated several wearable devices, and initially narrowed our focus to three specific wearables: the Asus Zenwatch (first generation), the Motorola Moto360 (second generation 2), and the Huawei Watch (first generation). These devices were selected as they meet our minimum requirements and are spread across a large price range (approximately $100 to $300 at the time of research), which gives us a valid representation of devices in the market. Each device contains the minimum sensors required, namely heart rate and step counter, and runs the Android Wear operating system, which natively supports Google Fit. In addition to the required sensors, all three of these devices contain numerous other sensors that could possibly aid in the accuracy of the song decision algorithm..

Unfortunately, it was discovered that the cheapest option, the Zenwatch, was infeasible due to the fact that the user is required to physically touch the watch frame with his or her fingers in order to record heart rate data. This was determined to be impractical for continuous heart rate monitoring during an exercise routine, and as such we cannot recommend this device for this purpose. The Moto360 and Huawei watch both contain optical pulse oximeter sensors that use a Photoplethysmogram to determine heart rate in near-real time. After evaluation these two devices, the decision was made to purchase the Moto360 because, despite the higher price tag, the Huawei watch did not satisfy any more of our requirements than the Moto360.

Software development required an Android phone running a relatively recent version of the Android OS. We used the Android emulator that is included with Android Studio (Google's official SDK for Android), as well as the phones of our team members. The Android emulator works by running a full instance of Android on your computer, and allows for loading different hardware profiles and Android version. It is also possible to emulate Android Wear devices, however it is simpler to use a real device when working with sensor data. As such, most of our testing was done performed on a Google Pixel with the Moto360 connected. The Pixel was running Android 7.1.1 with the February 5, 2017 security patch. The Moto360 was running Android Wear 1.5.0.3336103 on top of Android 6.0.1.

5 Software Design

The design of the application came in three parts: the android wear side that collects data, the music player side, and the algorithm that ties the two together. To save on development time, the first two of these parts were modified versions of existing code, using SensorDashboard [37] and a Tuts+ tutorial [38] for a music player. The two were then connected by the algorithm that chooses the appropriate music by giving the algorithm the data, and then letting it pick through the music. Select code snippets can be seen in Appendix C.

5.1 Music Player

For this initial version of the software, we opted to not use Spotify in order to simplify the code and testing. This simplification allowed us create a working version in a reasonable amount of time. This alternative method was to use the music files on the mobile device with the varying attributes for different input conditions. As a result, all that was necessary for this section was a player that pulled files from storage and played them. The methods to write such an app were found on the Tuts+ website. This was followed to create a basic music player that could be modified to work with the algorithm.

The tutorial code contains the baseline necessities for the project. The player plays music and allows a user to pick which music to play from a list if they wish. It also has controls that allow the user to skip or pause songs. The player does not include a like system or any additional quality of life features, but this bare bones player gave us the functionality to make a working iteration of the app.

Some modifications were necessary to the given code for our purposes. In order to make the music player refresh the screen for an updated list of songs, part of the setup code of the original needed to be moved to its own function. This function was then called after the end of every song in order to keep the updated list displayed. Similarly, other edits were necessary to make sure the music player got the input data, and worked with the algorithm.

5.2 Data Collection

Similarly to the music player, existing, open source code was also found to help us in our development of the data collection module. In this case, an app called SensorDashboard had the functionality we needed. This app is designed to detect and plot all sensor data available on an Android Wear-based wearable. Because the app provides more functionality than the project required, it was stripped of many features, including the graphing functions. The original source code is available on GitHub under the Apache License, Version 2.0.

In order to utilize this code, we had to uncover how the code functions. The project is split into three components: mobile, wear, and shared. Mobile contains the majority of the code, and is the part of the project that runs on the Android phone or tablet. This component then processes and graphs this data collected by the wear portion. In our implementation, the data is used by the algorithm to update the song list. The wear portion runs on the android wearable, and includes all the data collection functions. After receiving a “wake-up” signal from the mobile component over Bluetooth, the wear component begins collecting sensor data and passing this data back over Bluetooth to the mobile component. The shared component contains a small amount of shared code that is used by the entire project.

Determining exactly how the data is passed to the mobile component from the wear component proved to be challenging. Eventually it was discovered that the developers of SensorDashboard used a third party data bus called Otto Bus to pass the data. From the website of the source, “Otto is an event bus designed to decouple different parts of your application while still allowing them to communicate efficiently. Forked from Guava, Otto adds unique functionality to an already refined event bus as well as specializing it to the Android platform.” [39] Utilizing the Otto, we wrote a function that grabbed the sensor data off of the bus so that the music player could work with those values.

5.3 Algorithm

The algorithm that picks the appropriate music brings both of the other two parts together. The function that grabs the sensor data was placed in the class in the music player. This class grabs the list of songs from the device to display and play. Because we did not use Spotify in this version of the app, the algorithm is not very developed. We were limited by data available in the files located in the local storage of our device, so we chose to look at the first letter of the file name. As a proof of concept, we made all songs starting with ASCII characters up to but not including the letter “O” sorted in one list, and all other songs placed in the other list. These two lists correspond to heart rates below 80 BPM and greater than or equal to 80 BPM, respectively. The code uses the algorithm to refresh the list after any music file finishes playing or is skipped by the user in order to keep an updated list of appropriate music.

6 Results

Despite not utilizing Spotify or Google Fit, the application behaves as expected. Given different heart rates and step rates, the application will display the respective appropriate music. However, because of the length of this project, we were not able to sufficiently test the app on people to give an affirmative answer on if it has the expected impact on the physiological values. But, given the testing we did perform, it does appear that for some valence and BPM values, there was a noticeable change.

7 Conclusions

Based on our research and tests, we determined that we will design an Android app that interfaces with a smartwatch that is Google Fit compatible and with Tomahawk for song selection. Further tests will be done to determine the best way to actually select the songs based on the signals collected. In the next term, the team will design and test our app to make sure that we can actually provide a positive exercise experience for users.

If this project was to be continued, several modifications could be made. Ideally, the project should be made to integrate with Google Fit. This was our original plan, however it turned out to be overcomplicated for this initial testing. Fit is much more universal, and has the potential to simplify issues related to hardware differences. It is also possible to integrate various other Bluetooth sensors, such as a Polar H7 heart rate sensor strap, via Fit. These sensors may work directly with Google Fit, or may require a separate integration through BLE standards. This has the potential to lead to a much more diverse list of supported hardware, providing additional selling points.

Another possible point of work would be to properly integrate this project with Spotify. This allows the user to setup and use the app in a reasonable manner. Spotify, and specifically the API that allows for playing songs based on seeds (Appendix D), allows for users to have an enormous library of music at the touch of a button. It also removes the requirement for a user to supply music files on his or her own.

To collect data, it would be very helpful to have the project IRB certified. This would allow the trial to have proper incentives and would increase the pool of applicants. Increasing the pool of applicants would prove if our assumptions are true for all or just moderately active college students.

Appendix A: Respiration Rate Sensing Details

To actually determine the relationship between the different parameters, the group designed two types of tests.

The first was to determine how the heart rate is affected by respiration rate. This test was performed by having the user sit and do different types of breathing while their heart rate was taken. As with previous experiments, the heart rate was taken by a team member's Apple Watch. The test subject sat at rest, breathing normally for 5 minutes, breathed slowly for 2 minutes, breathed regularly for 2 minutes, hyperventilated for 2 minutes, and back to rest for the final 2 minutes. The results of the experiment can be seen in the graph below. It was shown that heart rate does follow respiration rate. In the end, the team determined that the more important relationship would be to see how respiration rate followed heart rate because breathing is controllable and would not be what changes automatically during exercise.

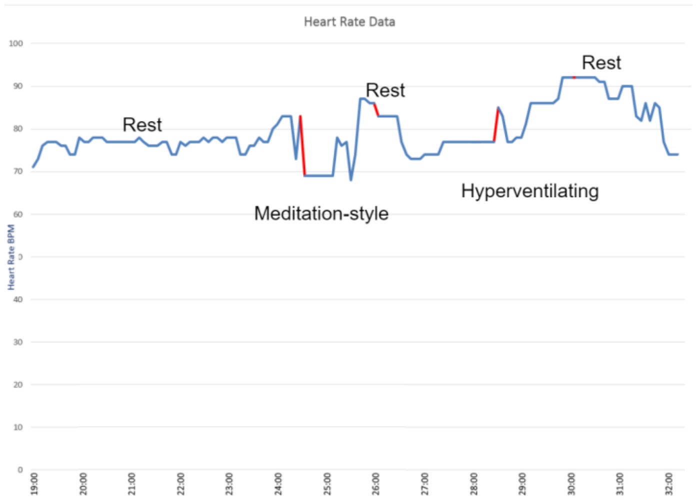


Figure 1: Heart Rate During Controlled Respiration

The second test was used to determine how well respiration rate followed the heart rate. This test involved volunteers from the WPI community. Three participants, each with a different level of personal fitness, took part in the testing. Participant 1 does not exercise at all, while participant 3 regularly exercises. Participant 2 does not exercise as frequently as participant 3, but they are a trained musician, meaning they have more control over their own breathing. The testing took place on a treadmill, and each participant was asked to run at varying speeds. Each participant's trial consisted of 7 parts, each 2 minutes long. The first part has the participant standing still to get baseline breath and heart rates. Then, for each successive part, the treadmill's speed was increased by 2 mph up to 6 mph. After 2 minutes at 6 mph, the speed was decreased every 2 minutes. The last part had the participant standing still to get their cooldown. This procedure was designed to get the participants' heart rates up to be able to see its effect on breath rate.

The output of the Respiration Sensor gave a waveform of how the participants were breathing, but to get a breaths per minute value to compare against the heart rate data, the data would need to be further analyzed. To do this, a MATLAB script was coded to accurately count the number of peaks in the waveform per minute, and can be found in Appendix B. The script would count the number of times the waveform passed above the average value, and decrement each from the count after a minute had passed. This average value is calculated while processing with a sliding window average. The resulting breath rate values are then plotted against the heart rate values. The output graphs for the three participants can be seen below.

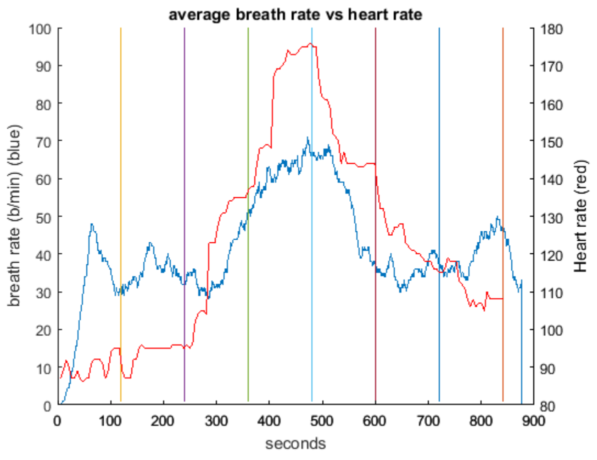


Figure 2: Participant 1(Not Active)

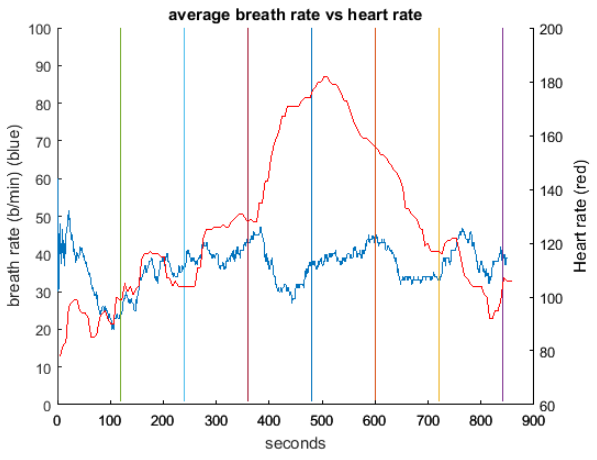


Figure 8: Participant 3 (Mildly Active, Trained Musician)

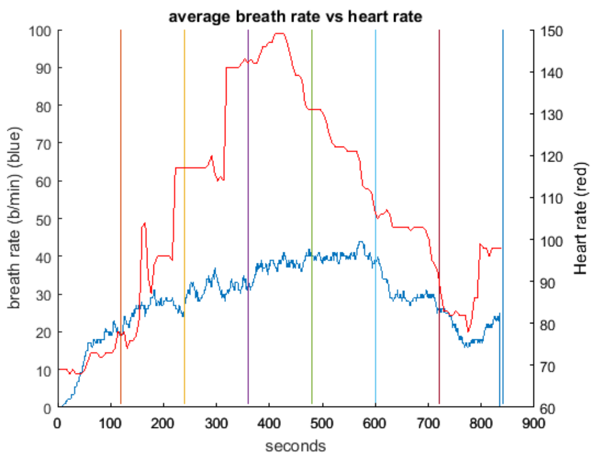


Figure 4: Participant 3 (Extremely Active)

The vertical bars on each of the graphs show the time when each of the parts of the trial start. It is very noticeable on all three graphs that the heart rate rises and falls as expected as the speed of the treadmill rises and falls. The breath rate data is less straight forward to discern. In all three graphs, there does not seem to be much to get out of the resting breath rates. There doesn't appear to be any correlation between those values and either the heart rate values or the current speed of the treadmill. Looking at the graphs for participants 1 and 3 for higher speeds, it does appear that breath rate follows heart rate, as they increase and decrease at similar times. However, for participant 2, there is a dramatic drop in breath rate in the fastest part of the trial. This could be due to participant 2's musical training that allowed them to control their breathing.

The data from the three participants confirms that there is no information that can be easily seen in breath rate data that can't already be observed in the heart rate. For users exercising who don't control their breathing, their breathing will simply follow their heart rate, eliminating any need to track breath rate. Additionally, if the user can control their breathing, the breath rate is pointless to track. Thus, breath rate is not necessary to be sensed to gain the data needed. The code used to process this data is seen in the MATLAB script below.

```
close all;
clc;
clear;

output = [
      %breath rate data goes here
];

Heart = [
%heart rate data goes here
];

TIMESTEP = .1;           %time between breath rate output values
AVERAGE_V = 0;

len = length(output);
h_len = length(Heart);
t = 1:len;
t_h = 1:h_len;
t=t*TIMESTEP;            %time of the breathrate output
t_h=t_h*TIMESTEP*53;     %converts x axis to seconds based off of 5.3 sample
frequency

%plots the waveform
plot(t,output);
xlabel('seconds');
ylabel('voltage');
title('breath rate waveform');


n=1;
cur=0;
detected=0;
rate=0;
num_breaths=0;
peak_t_record=zeros(100)+len;
rates=zeros(fix(len/10));
last_b=1;
next_b=1;

AVG_LEN=100;
recent_voltage=zeros(AVG_LEN,1);
```

```
%loop to analize all of the breath rate waveform
while n<len
    cur=output(n);

    %calculate AVERAGE_V
    recent_voltage(mod(n-1,AVG_LEN)+1)=cur;
    if(n<AVG_LEN)
        AVERAGE_V = sum(recent_voltage)/n;
    else
        AVERAGE_V = sum(recent_voltage)/AVG_LEN;
    end


    %if the last breath is over a minute ago
    if((n-(60/TIMESTEP))>peak_t_record(last_b))
        num_breaths=num_breaths-1;  %subtract total
        last_b=last_b+1;            %increment last_b index
    end

    %if the output is over the average and a peak hasn't been recently
    %detected
    if((cur>AVERAGE_V)&&(detected<1))
        detected=1;
        num_breaths=num_breaths+1;
        peak_t_record(next_b)=n;
        next_b=next_b+1;
    end

    %if the time is under a minute, approximate breath rate
    if(n>(60/TIMESTEP))
        rate=num_breaths;
    else
        rate=num_breaths/(n/(60/TIMESTEP));
    end

    %reset detected flag if under the average
    if((cur<AVERAGE_V)&&(detected>0))
        detected=0;
    end

    %only give rate every second
    if(mod(n,10)==0)
        rates(n/10)=rate;
    end
    n=n+1;
end

%modified time array to account for the change to every second
t_2= 1:(fix(len/10));
t_2=t_2*TIMESTEP*10;
```

```
%creates the verticle lines
l_1=zeros(100,1)+120;
l_2=zeros(100,1)+120*2;
l_3=zeros(100,1)+120*3;
l_4=zeros(100,1)+120*4;
l_5=zeros(100,1)+120*5;
l_6=zeros(100,1)+120*6;
l_7=zeros(100,1)+120*7;
figure();

%breath rate axis
ax1 = gca;
hold on;

%plot breath rate
plot(t_2,rates);
hold on;

%plot lines
plot(l_1,t_2(1:100));
hold on;
plot(l_2,t_2(1:100));
hold on;
plot(l_3,t_2(1:100));
hold on;
plot(l_4,t_2(1:100));
hold on;
plot(l_5,t_2(1:100));
hold on;
plot(l_6,t_2(1:100));
hold on;
plot(l_7,t_2(1:100));
title('average breath rate vs heart rate');
xlabel('seconds');
ylabel('breath rate (b/min) (blue)');

%heart rate axis
ax2 = axes('Position',get(ax1,'Position'),...
       'YAxisLocation','right',...
       'Color','none',...
       'XColor','k','YColor','k');
linkaxes([ax1 ax2],'x');
hold on;

%plot heart rate
plot(t_h,Heart,'Color','r');
ylabel(ax2,'Heart rate (red)');
```

Appendix B: Readme File for Android Studio Project

Music Wearable MQP 2017

This app is designed to select music based on fitness data collected on an Android Wear device. This implementation primarily serves as a proof-of-concept, and performs as such.

Notes/Helpful Info:

- This App was written for Android Wear 1.5 (1.5.0.3336103 if you really care) on Android 6.0.1. NO idea what will/will not work on the recently released Wear 2.0.
- Unpairing an Android Wear device from a phone causes the Wear device to factory reset. AKA you can't just share the device between phones.
- You will need to enable developer mode on both your phone and watch. This is generally accomplished by rapidly tapping "Build Number" in "About Phone/Device/Etc" within device settings, but YMMV.
- Make sure to authorize both the phone and watch for ADB

Using Android Studio and Wear with no Main Activity:

To fix the error in Android Studio that doesn't allow directly launching the Wear component ("Main Activity Not Found"), do the following (via StackOverflow):

Run -> Edit Configurations -> Android App -> wear -> General -> Launch Options -> Launch: Nothing

Alternatively, you can build the apks (Build -> Build APK from within Android Studio) and then install it manually via ADB (see below for how to connect to wear device over Bluetooth).

Debugging Over Bluetooth:

After configuring and enabling ADB (see Source: Android Developers and ADB help files), run the following to link the debugger to the watch:

```
adb forward tcp:4444 localabstract:/adb-hub     //Any port you have full access to should work.
adb connect 127.0.0.1:4444
```

You can then run any ADB command as follows:

```
adb -s 127.0.0.1:4444 [some command]
```

Source: Android Developers

Credits

Sensor Dashboard was originally written at the Android Wear Hackathon 2014 in London by Juhani Lehtimäki, Benjamin Stürmer and Sebastian Kaspari. It is avalible under the Apache 2.0 License (below).

Licenses

Parts of this code are licensed as follows:

Appendix C: Select Code Snippets

```java
public class DeviceClient {

/* Incomplete code snippets follow */

public void sendSensorData(final int sensorType, final int accuracy, final
long
timestamp, final float[] values) {
long t = System.currentTimeMillis();

long lastTimestamp = lastSensorData.get(sensorType);
long timeAgo = t - lastTimestamp;

if (lastTimestamp != 0) {
if (filterId == sensorType && timeAgo < 100) {
return;
}

if (filterId != sensorType && timeAgo < 3000) {
return;
}
}

lastSensorData.put(sensorType, t);

executorService.submit(new Runnable() {
@Override
public void run() {
sendSensorDataInBackground(sensorType, accuracy, timestamp, values);
}
});
}

private void sendSensorDataInBackground(int sensorType, int accuracy, long
timestamp, float[] values) {
if (sensorType == filterId) {
Log.i(TAG, "Sensor " + sensorType + " = " + Arrays.toString(values));
} else {
Log.d(TAG, "Sensor " + sensorType + " = " + Arrays.toString(values));
}

PutDataMapRequest dataMap = PutDataMapRequest.create("/sensors/" +
sensorType);

dataMap.getDataMap().putInt(DataMapKeys.ACCURACY, accuracy);
dataMap.getDataMap().putLong(DataMapKeys.TIMESTAMP, timestamp);
dataMap.getDataMap().putFloatArray(DataMapKeys.VALUES, values);

PutDataRequest putDataRequest = dataMap.asPutDataRequest();
send(putDataRequest);
}

private void send(PutDataRequest putDataRequest) {
```

```
if (validateConnection()) {
Wearable.DataApi.putDataItem(googleApiClient,
putDataRequest).setResultCallback(new
ResultCallback<DataApi.DataItemResult>() {
@Override
public void onResult(DataApi.DataItemResult dataItemResult) {
Log.v(TAG, "Sending sensor data: " +
dataItemResult.getStatus().isSuccess());
}
});
}
}
}




public class MessageReceiverService extends WearableListenerService {

/* Incomplete code snippets follow */

@Override
public void onDataChanged(DataEventBuffer dataEvents) {
super.onDataChanged(dataEvents);

for (DataEvent dataEvent : dataEvents) {
if (dataEvent.getType() == DataEvent.TYPE_CHANGED) {
DataItem dataItem = dataEvent.getDataItem();
Uri uri = dataItem.getUri();
String path = uri.getPath();

if (path.startsWith("/filter")) {
DataMap dataMap = DataMapItem.fromDataItem(dataItem).getDataMap();
int filterById = dataMap.getInt(DataMapKeys.FILTER);
deviceClient.setSensorFilter(filterById);
}
}
}
}

@Override
public void onMessageReceived(MessageEvent messageEvent) {
Log.d(TAG, "Received message: " + messageEvent.getPath());

if (messageEvent.getPath().equals(ClientPaths.START_MEASUREMENT)) {
startService(new Intent(this, SensorService.class));
}

if (messageEvent.getPath().equals(ClientPaths.STOP_MEASUREMENT)) {
stopService(new Intent(this, SensorService.class));
}
}
}

```

```java


public class MusicService extends Service implements
MediaPlayer.OnPreparedListener, MediaPlayer.OnErrorListener,
MediaPlayer.OnCompletionListener
{

public void playSong()
{
//reset mediaplayer
player.reset();

//get song
if(debug==1)
Log.d(TAG,"Position = "+songPosn);

if(songPosn<0) {
songPosn = 0;
}

if(songPosn>=(5)) {
songPosn = 0;
}



Song playSong = filteredSongs.get(songPosn);

songTitle=playSong.getTitle();

//get id
long currSong = playSong.getID();
//set uri
Uri trackUri = ContentUris.withAppendedId(
android.provider.MediaStore.Audio.Media.EXTERNAL_CONTENT_URI,
currSong);

try
{
player.setDataSource(getApplicationContext(), trackUri);
}
catch(Exception e)
{
Log.e("MUSIC SERVICE", "Error setting data source", e);
}

player.prepareAsync();
}

public void onCreate()
{
//create the service
super.onCreate();
//initialize position
```

```
songPosn=0;
//create player
player = new MediaPlayer();
initMusicPlayer();
rand=new Random();
BusProvider.getInstance().register(this);
}

public void initMusicPlayer()
{
player.setWakeMode(getApplicationContext(),
PowerManager.PARTIAL_WAKE_LOCK);
player.setAudioStreamType(AudioManager.STREAM_MUSIC);
player.setOnPreparedListener(this);
player.setOnCompletionListener(this);
player.setOnErrorListener(this);
}


public void filterSongs()
{
if(debug==1)
Log.d(TAG,"Songs filtered with value: "+curHeartRate);
if(filteredSongs!=null)
filteredSongs.clear();
for(Song song:songs) {
if (curHeartRate >= 80)
{
if (song.getTitle().charAt(0) < 'O')
filteredSongs.add(song);
} else
{
if (song.getTitle().charAt(0) >= 'O')
filteredSongs.add(song);
}
}
}

@Override
public void onCompletion(MediaPlayer mediaPlayer)
{
filterSongs();
refreshView.refresh(filteredSongs);
if(player.getCurrentPosition()>0)
{
mediaPlayer.reset();
playNext();
}
}

@Subscribe
public void onSensorUpdatedEvent(final SensorUpdatedEvent event)
{
if(debug==1)
```

```
Log.d(TAG,"sensor event");
if(event.getSensor().getId()==13)
{
curStepRate=event.getDataPoint().getValues()[0];
}
else
{
curHeartRate=event.getDataPoint().getValues()[0];
if(debug==1)
Log.d(TAG,"heart rate set to "+curHeartRate);
}
//TextView textView = (TextView) findViewById(R.id.empty_state);
//textView.append(curHeartRate+", "+curStepRate+", "+"\n");
}

public void playNext()
{
if(shuffle)
{
int newSong = songPosn;
while(newSong==songPosn)
{
newSong=rand.nextInt(songs.size());
}
songPosn=newSong;
}
else
{
if(songPosn>=songs.size()) {
songPosn=0;
Log.d(TAG, "songPosn=0");
}
songPosn++;
Log.d(TAG, "songPosn++");

}
playSong();
}


}




public class SensorReceiverService extends WearableListenerService {


@Override
public void onDataChanged(DataEventBuffer dataEvents) {
Log.d(TAG, "onDataChanged()");

for (DataEvent dataEvent : dataEvents) {
if (dataEvent.getType() == DataEvent.TYPE_CHANGED) {
```

```
DataItem dataItem = dataEvent.getDataItem();
Uri uri = dataItem.getUri();
String path = uri.getPath();

if (path.startsWith("/sensors/")) {
unpackSensorData(
Integer.parseInt(uri.getLastPathSegment()),
DataMapItem.fromDataItem(dataItem).getDataMap()
);
}
}
}
}

private void unpackSensorData(int sensorType, DataMap dataMap) {
int accuracy = dataMap.getInt(DataMapKeys.ACCURACY);
long timestamp = dataMap.getLong(DataMapKeys.TIMESTAMP);
float[] values = dataMap.getFloatArray(DataMapKeys.VALUES);


if((sensorType==13)||(sensorType==21)) //only add new data if it's step
or
heart rate
{
Log.d(TAG, "Received sensor data " + sensorType + " = " +
Arrays.toString(values));
sensorManager.addSensorData(sensorType, accuracy, timestamp, values);
}

}


}




public class RemoteSensorManager {

private Sensor createSensor(int id) {
Sensor sensor = new Sensor(id, sensorNames.getName(id));

sensors.add(sensor);
sensorMapping.append(id, sensor);

BusProvider.postOnMainThread(new NewSensorEvent(sensor));

return sensor;
}

private void filterBySensorIdInBackground(final int sensorId) {
Log.d(TAG, "filterBySensorId(" + sensorId + ")");

if (validateConnection()) {
```

```
PutDataMapRequest dataMap = PutDataMapRequest.create("/filter");

dataMap.getDataMap().putInt(DataMapKeys.FILTER, sensorId);
dataMap.getDataMap().putLong(DataMapKeys.TIMESTAMP,
System.currentTimeMillis());

PutDataRequest putDataRequest = dataMap.asPutDataRequest();
Wearable.DataApi.putDataItem(googleApiClient,
putDataRequest).setResultCallback(new
ResultCallback<DataApi.DataItemResult>() {
@Override
public void onResult(DataApi.DataItemResult dataItemResult) {
Log.d(TAG, "Filter by sensor " + sensorId + ": " +
dataItemResult.getStatus().isSuccess());
}
});
}
}
}
```

For the complete code, see the team GitHub at:
https://github.com/matthewbarreiro/MusicWearableMQP2017

Appendix D: Query Arguments for Get Recommendations Based on Seeds Function in Spotify Web API

Request Parameters

HEADER FIELD	VALUE
Authorization	*Required*. A valid access token from the Spotify Accounts service: see the Web API Authorization Guide for details.

QUERY ARGUMENT	VALUE
limit	*Optional*. The target size of the list of recommended tracks. For seeds with unusually small pools or when highly restrictive filtering is applied, it may be impossible to generate the requested number of recommended tracks. Debugging information for such cases is available in the response. Default: 20. Minimum: 1. Maximum: 100.
market	Optional. An ISO 3166-1 alpha-2 country code. Provide this parameter if you want to apply Track Relinking. Because min_*, max_* and target_* are applied to pools before relinking, the generated results may not precisely match the filters applied. Original, non-relinked tracks are available via the linked_from attribute of the relinked track response.
max_*	*Optional*. *Multiple values*. For each tunable track attribute, a hard ceiling on the selected track attribute's value can be provided. See tunable track attributes below for the list of available options. For example, max_instrumentalness=0.35 would filter out most tracks that are likely to be instrumental.
min_*	*Optional*. *Multiple values*. For each tunable track attribute, a hard floor on the selected track attribute's value can be provided. See tunable track attributes below for the list of available options. For example, min_tempo=140 would restrict results to only those tracks with a tempo of greater than 140 beats per minute.
seed_artists	A comma separated list of Spotify IDs for seed artists. Up to 5 seed values may be provided in any combination of seed_artists, seed_tracks andseed_genres.

seed_genres	A comma separated list of any genres in the set of available genre seeds. Up to 5 seed values may be provided in any combination of seed_artists, seed_tracks andseed_genres.
seed_tracks	A comma separated list of Spotify IDs for a seed track. Up to 5 seed values may be provided in any combination of seed_artists, seed_tracks andseed_genres.
target_*	*Optional*. *Multiple values*. For each of the tunable track attributes (below) a target value may be provided. Tracks with the attribute values nearest to the target values will be preferred. For example, you might request target_energy=0.6 and target_danceability=0.8. All target values will be weighed equally in ranking results.

Tuneable Track attributes

ATTRIBUTE NAME	VALUE TYPE	VALUE DESCRIPTION
acousticness	float	A confidence measure from 0.0 to 1.0 of whether the track is acoustic. 1.0 represents high confidence the track is acoustic.
danceability	float	Danceability describes how suitable a track is for dancing based on a combination of musical elements including tempo, rhythm stability, beat strength, and overall regularity. A value of 0.0 is least danceable and 1.0 is most danceable.
duration_ms	int	The duration of the track in milliseconds.
energy	float	Energy is a measure from 0.0 to 1.0 and represents a perceptual measure of intensity and activity. Typically, energetic tracks feel fast, loud, and noisy. For example, death metal has high energy, while a Bach prelude scores low on the scale. Perceptual features contributing to this attribute include dynamic range, perceived loudness, timbre, onset rate, and general entropy.
instrumentalness	float	Predicts whether a track contains no vocals. "Ooh" and "aah" sounds are treated as instrumental in this context. Rap or spoken word tracks are clearly "vocal". The closer the instrumentalness value is to 1.0, the greater likelihood the track contains no vocal content. Values above 0.5 are intended to represent instrumental tracks, but confidence is higher as the value approaches 1.0.
key	int	The key the track is in. Integers map to pitches using standard Pitch Class notation. E.g. 0 = C, 1 = C♯/D♭, 2 = D, and so on.
liveness	float	Detects the presence of an audience in the recording. Higher liveness values represent an increased probability that the track was performed live. A value above 0.8 provides strong likelihood that the track is live.
loudness	float	The overall loudness of a track in decibels (dB). Loudness values are averaged across the entire track and are useful for comparing relative loudness of tracks. Loudness is the quality of a sound that is the primary psychological correlate of physical strength (amplitude). Values typical range between -60 and 0 db.

mode	int	Mode indicates the modality (major or minor) of a track, the type of scale from which its melodic content is derived. Major is represented by 1 and minor is 0.
popularity	int	The popularity of the track. The value will be between 0 and 100, with 100 being the most popular. The popularity is calculated by algorithm and is based, in the most part, on the total number of plays the track has had and how recent those plays are. Note: When applying track relinking via the market parameter, it is expected to find relinked tracks with popularities that do not match min_*, max_*and target_*popularities. These relinked tracks are accurate replacements for unplayable tracks with the expected popularity scores. Original, non-relinked tracks are available via thelinked_from attribute of the relinked track response.
speechiness	float	Speechiness detects the presence of spoken words in a track. The more exclusively speech-like the recording (e.g. talk show, audio book, poetry), the closer to 1.0 the attribute value. Values above 0.66 describe tracks that are probably made entirely of spoken words. Values between 0.33 and 0.66 describe tracks that may contain both music and speech, either in sections or layered, including such cases as rap music. Values below 0.33 most likely represent music and other non-speech-like tracks.
tempo	float	The overall estimated tempo of a track in beats per minute (BPM). In musical terminology, tempo is the speed or pace of a given piece and derives directly from the average beat duration.
time_signature	int	An estimated overall time signature of a track. The time signature (meter) is a notational convention to specify how many beats are in each bar (or measure).
valence	float	A measure from 0.0 to 1.0 describing the musical positiveness conveyed by a track. Tracks with high valence sound more positive (e.g. happy, cheerful, euphoric), while tracks with low valence sound more negative (e.g. sad, depressed, angry).

References

[1] C. J. Murrock, Psychology of Mood, 2005, pp. 141-155.

[2] M. Clayton, R. Sager and U. Will, "In Time with the Music: The Concept of Entrainment and its Significance for Enthomusicology," *ESEM Counterpoint,* vol. 1, 2004.

[3] Mayo Clinic, "Heart Rate: What's Normal?," [Online]. Available: http://www.mayoclinic.org/healthy-lifestyle/fitness/expert-answers/heart-rate/faq-20057979. [Accessed September 2016].

[4] S. J. Cox, "Vagus Nerve," Rice University, 2012.

[5] C. H. Hawkes, "Endorphins: the basis or pleasure?," *Journal of Neurology, Neurosurgery, and Psychology,* no. 55, pp. 245-250, 1992.

[6] T. Jia, Y. Ogawa, M. Miura, O. Ito, and M. Kohzuki, "Music Attenuated a Decrease in Parasympathetic Nervous System Activity after Exercise," *PLoS One*, vol. 11, Feb. 2016.

[7] C. J. Bacon, T. R. Myers, and C. I. Karageorghis, "Effect of music-movement synchrony on exercise oxygen consumption.," *The Journal of Sports Medicine and Physical Fitness*, vol. 52, pp. 359–365, Aug. 2012.

[8] J. M. Standley, "The Effect of Vibrotactile and Auditory Stimuli on Perception of Comfort, Heart Rate, and Peripheral Finger Temperature," *Journal of Music Therapy*, vol. 28, no. 3, pp. 120–134, Jan. 1991.

[9] adafruit, "Fitbit Force Teardown," *YouTube*, Nov-2013. [Online]. Available: https://www.youtube.com/watch?v=_yuwuokee4i. [Accessed: 15-Sep-2016].

[10] R. Whitman, "Samsung Announces Gear 2 And Gear 2 Neo Smartwatches Running Tizen, Available Worldwide In April," *Android Police Android News Apps Games Phones Tablets*, 2014. [Online]. Available: http://www.androidpolice.com/2014/02/22/samsung-announces-gear-2-and-gear-2-neo-smart-watches-running-tizen-available-worldwide-in-april/. [Accessed: 13-Oct-2016].

[11] "Tizen," *About*. [Online]. Available: https://www.tizen.org/about?langswitch=en. [Accessed: 13-Oct-2016].

[12] "ASUS ZenWatch (WI500Q) - Specifications," *ASUS ZenWatch (WI500Q)*. [Online]. Available: https://www.asus.com/us/zenwatch/asus_zenwatch_wi500q/specifications/. [Accessed: 13-Oct-2016].

[13] "Moto 360 by Motorola - Smartwatch Powered by Android Wear," *Motorola*. [Online]. Available: https://www.motorola.com/us/products/moto-360. [Accessed: 13-Oct-2016].

[14] "Huawei Watch," *GetHuawei.com*. [Online]. Available: http://www.gethuawei.com/buy-huawei-watch#watchstainlessleather. [Accessed: 13-Oct-2016].

[15] "Wearable Computing Devices, Like Apple's iWatch, Will Exceed 485 Million Annual Shipments by 2018," *ABI Research*. [Online]. Available: https://www.abiresearch.com/press/wearable-computing-devices-like-apples-iwatch-will/. [Accessed: 13-Oct-2016].

[16] "Wearable Tech Device Awareness Surpasses 50 Percent Among US Consumers, According to NPD," *NPD Group*, Jul-2014. [Online]. Available: https://www.npd.com/wps/portal/npd/us/news/press-releases/wearable-tech-device-awareness-surpasses-50-percent-among-us-consumers-according-to-npd/. [Accessed: 13-Oct-2016].

[17] "Daily Dose: Smartphones Have Become a Staple of the U.S. Media Diet,"*Nielsen*, 21-Apr-2016. [Online]. Available: http://www.nielsen.com/us/en/insights/news/2016/daily-dose-smartphones-have-become-a-staple-of-the-us-media-diet.html. [Accessed: 13-Oct-2016].

[18] A. Lunney, N. R. Cunningham, and M. S. Eastin, "Wearable fitness technology: A structural investigation into acceptance and perceived fitness outcomes," *Computers in Human Behavior*, vol. 65, pp. 114–120, Dec. 2016.

[19] J. Cumming and C. Hall, "The Relationship Between Goal Orientation and Self-Efficacy for Exercise," *Journal of Applied Social Psychology*, vol. 34, no. 4, pp. 747–763, 2004.

[20] "Our Company," *The Echo Nest*. [Online]. Available: http://the.echonest.com/company/. [Accessed: 13-Oct-2016].

[21] S. Murphy, "Spotify Acquires Music Data Company The Echo Nest," *Mashable*, Jun-2014. [Online]. Available: http://mashable.com/2014/03/06/spotify-acquires-echo-nest/#opsl08dbakq9. [Accessed: 13-Oct-2016].

[22] "Spotify Echo Nest API," *Spotify Developer*. [Online]. Available: https://developer.spotify.com/spotify-echo-nest-api/. [Accessed: 13-Oct-2016].

[23] "Get Recommendations Based on Seeds," *Spotify Developer*. [Online]. Available: https://developer.spotify.com/web-api/get-recommendations/. [Accessed: 13-Oct-2016].

[24] "Web API," *Gracenote*. [Online]. Available: https://developer.gracenote.com/web-api. [Accessed: 13-Oct-2016].

[25] "Rhythm API," *Gracenote*. [Online]. Available: https://developer.gracenote.com/rhythm-api. [Accessed: 13-Oct-2016].

[26] "Google Fit " *Google Developers*. [Online]. Available: https://developers.google.com/fit/. [Accessed: 13-Oct-2016].

[27] Gopal, Kaushik.”045: Bluetooth (LE) with Dave (devunwired) Smith.” Audio blog post. Fragmented. Fragmentedpodcast.com, 13 June, 2016. 28 September, 2016

[28] "Dashboards,". [Online]. Available: https://developer.android.com/about/dashboards/index.html. [Accessed: Dec. 14, 2016.]

[29] "About the project," CyanogenMod. [Online]. Available: https://www.cyanogenmod.org/about. [Accessed: Dec. 15, 2016.]

[30] "Material design for Android,". [Online]. Available: https://developer.android.com/design/material/index.html. [Accessed: Dec. 15, 2016.]

[31] em. Inc and A. R. Reserved, "Only 33% of US mobile users will pay for Apps this year," 2015. [Online]. Available: https://www.emarketer.com/Article/Only-33-of-US-Mobile-Users-Will-Pay-Apps-This-Year/1011965. [Accessed: Dec. 12, 2016.]

[32] S. Butner, "How much in advertising revenue can a mobile App generate?," *Small Business Chron*, 2016. [Online]. Available: http://smallbusiness.chron.com/much-advertising-revenue-can-mobile-app-generate-76855.html.[Accessed: Dec. 12, 2016.]

[33] "Survey: 58 percent of smartphone users have downloaded a fitness or health app," MobiHealthNews, 2015. [Online]. Available: http://www.mobihealthnews.com/48273/survey-58-percent-of-smartphone-users-have-downloaded-a-fitness-or-health-app. [Accessed: Dec. 12, 2016.]

[34] K. Gopal, “Otto,” *Square*. [Online]. Available: http://square.github.io/otto/. [Accessed: 04-Mar-2017].