Let Music Ease Your Bike Ride!

A Design Project Report

Presented to the School of Electrical and Computer Engineering of Cornell University in Partial Fulfillment of the Requirements for the Degree of

Master of Engineering, Electrical and Computer Engineering

Submitted by

Siyu Chen

MEng Field Advisor: Bruce Land

Degree Date: January, 2015

Abstract

Master of Engineering Program

School of Electrical and Computer Engineering

Cornell University

Design Project Report

Project Title: Let Music Ease Your Bike Ride!

Author: Siyu Chen

Abstract:

The topic of my master of engineering design project is designing an android application which helps soothing the bikers' pain when they are tired from bike riding. The application can switch to different playlists based on bikers' speed and heart rate. The whole system mainly consists of two parts: the embedded hardware attached on the bike which probes bike speed and heart rate using a microcontroller, and the customized android music player application where the playlists switching AI (Artificial Intelligence) resides. These two parts communicates via bluetooth connection. This design project is an implementation-orientated project, and all the designs are deliberately considered in terms of good user experience and hardware mounting simplicity.

Executive Summary

In places like Ithaca where there are a lot of steer slopes, riding a bike can be real challenging. People always get tired after a long ride or when riding up a steer slope, and would want some inspiring and encouraging music to help them regenerate and get back on track. My design project is implementing a system which helps bikers to get rid of tiredness by playing inspiring music after diagnosing that the biker is not in good physical condition. The human interface of the whole system is a customized android music player which has AI functioning as discussed above. However, for the proper function of the music player AI, inputs from MCU (microcontroller unit) which works as training attributes is needed to implement.

In this design project, an android application which works both as a physical condition monitor and a music player with AI (Artificial Intelligence) mounted is implemented. The design project consists of five major steps. First, requirement analysis was performed before any real implementation design started. In this design phase, the functionality of the whole system was deliberately and concretely defined. Second, high level design of both embedded hardware and android software was performed. In this design phase, the collaboration and functional partitioning between embedded hardware and android application were defined, while the functionality, expected outcome, and the test strategy were designed. Third, the hardware was built and tested independently for both accuracy and robustness. Forth, the customized android music player was implemented incrementally, with each separate function being thoroughly tested before moving to the next function realization. Last, the whole system was tested in reality in terms of debugging and performance improvement.

This design project is an implementation-oriented project which consists of circuit design, microcontroller programming using C/C++, android application development, and UI design by writing xml layout file. In this project, I learned the basic developing cycle of software development, and learned how to write efficient android app which manages and release resources properly. Last but not least, I learned how to build a system from scratch up only by myself and how to master new things in a short time.

Table of Contents

List of Figures

1. Introduction

1.1 Background

In places like Ithaca where there are a lot of steer slopes, riding a bike can be real challenging. People always get tired after a long ride or when riding up a steer slope, and would want some inspiring and encouraging music to help them regenerate and get back on track. Basically, the main purpose of my design is to use music to sooth the pain from exhausting bike rides.

There are massive numbers of music app release in android market. Most of them don't aim at implementing AI (Artificial Intelligence) on app in the work out setting. In other words, they are not smart enough to play different music based on users' physical condition. When people are working out, they tend to listen to music to have distraction from physical tiredness. Hence, a music player which is smart enough to diagnose people's tastes for different music in different physical condition is in demand.

My design project is implementing a system which helps bikers to get rid of tiredness by playing inspiring music after diagnosing that the biker is not in good physical condition. The human interface of the whole system is a customized android music player which has AI functioning as discussed above. However, for the proper function of the music player AI, inputs from MCU (microcontroller unit) which works as training attributes is needed to implement.

1.2 Functional Requirement

To ensure basic functionality of the system, certain requirements are needed and they are listed below.

Priority	Name	Category	Additional Description
Must	Real-time Speed Measurement	Circuit	N/A
Must	Real-time Heart Rate Meter	Circuit	N/A
Must	Microcontroller Design	Circuit/Software	Using microcontroller to process electric signal while driving Bluetooth module
Must	Bluetooth connection	Circuit/Software	N/A
Must	PuTTY Serial Connection for	Test	N/A

	Measurement Accuracy& Robustness Test		
Must	Bare Bone Android Music App Development	Software	N/A
Must	UI Design	Software/Design	N/A
Optional	User Experience Improvement	Software/Design	N/A
Must	Software Bluetooth Connection and Socket Stream Management	Software	N/A
Must	Android App Software Monitor	Software	embedded a speed and heart rate monitor into the main UI of the app
Optional	Resources Release Optimization	Software	Releasing memory or CPU resources when threads are not actually in execution loop
Must	AI Design and Implementation	Software	N/A
Must	System Level Test	Test	N/A

1.3 Cost

The cost of my design mainly comes from embedded hardware. The components I used are listed below with their costs.

Name	Number	Unit Cost / $
Arduino UNO Board	1	20
Bluetooth Silver Mate	1	40
12 v Buttery Sockets	1	5
Magnet & Sensor Pair	1	2
IR emitter/Receiver	1	N/A
Capacitors	Several	N/A
Resisters	Several	N/A
Op-amp	2	N/A
Wire	Several	N/A
Dupont Line	8	0.1

The cost of the system is 68 dollars for hand-assembled. For mass-production, the price can be reduced to less than 30 dollars.

2. High Level Design

The major component of the system is an android smart phone application which can switch to different music playlists adapting to different bike speed patterns and bikers' real-time heart rate. The playlists' switching AI is a combination of KNN (K-nearest neighbor) algorithm, average speed hysteresis control, and on-line expert learning. To build this AI, a training phase which takes real-time speed and heart rate as training attributes is needed before any actual AI control.

The design system mainly consists of two parts. The first part is the hardware embedded on the bike which handles the speed and heart rate measurement. The second part is the customized android app where the playlists' switching AI is built. These two parts must be connected for real-time data transmission. In general, both USB connection and Bluetooth connection can be used as time-efficient communication methods. In practice, Bluetooth module was chosen because bluetooth is wireless, maximize the separation and independence of the bike hardware and smartphone. Meanwhile, using USB as connection may cause severe safety issue, since USB wire is very likely to get stuck into the spinning wheel. However, the disadvantage of bluetooth connection is battery consuming, so in the Android app I designed, battery efficiency management was deliberately taken care of.

2.1 High-level Design of Embedded Hardware

For the bike embedded hardware design, I chose Arduino UNO microcontroller. Arduino UNO board is a single-board microcontroller; the hardware consists of an open-source hardware board based on ATmega328, a 16 MHz ceramic resonator, a USB connection, a power jack, an ICSP header, a reset button and several chip LEDs. Arduino board is a sophisticated open-source hardware board which is designed to make the application of interactive objects or environment more accessible. Arduino has its own IDE (Integrated Design Environment) and a set of library which makes writing software code extremely easy for beginners. The Arduino UNO board does not need an additional programmer for downloading code to the microcontroller, instead, the USB and the on-board Atmega16U2 programmed as a USB-to-serial converter can be used along with its IDE to download code to the on-board microcontroller. In my design, the Arduino UNO board

is responsible for measuring the bike speed and heart rate, as well as driving the Bluetooth module which talks to Android smart phone.

For Bluetooth hardware, I chose Bluetooth Mate Silver (WRL-12576 RoHS) manufactured by Sparkfun Company (http://www.sparkfun.com). These radios work as a serial (RX/TX) pipe, and are a great wireless replacement for serial cables. Any serial stream from 2400 to 115200 bps can be passed seamlessly. The bluetooth chip being used by Bluetooth Mate Siler is RN-42, which is perfect for short range, battery powered applications. The RN-42 uses only 26 uA in sleep mode while still being discoverable and connectable. RN-42 has a communication radius of 10 meters, which is sufficient enough for the bike & android system.

In all method for measuring bike speed, the easiest way is to count time interval for a full revolution of the bike wheel. In order to minimize the damage to bike integrity as well as making the whole embedded hardware detachable from the bike, the magnet and magnetic sensor pair are used to signal a whole revolution of the wheel. By attaching a magnet to the spoke of the wheel and fixing the magnetic sensor on the hub, we can easily get the revolution signal. There are a lot of alternative hardware design choices such as IR transceiver pair, optical grating & obstacle pair, DC motor etc., but the magnet & magnetic sensor pair was chosen because they can be dismounted easily, and it minimizes the circuitry stretching to the wheel while having good stability and reliability.

2.2 High-level Design of Bluetooth Connection

As mentioned in previous part, the Bluetooth Mate Silver module I used is of good stability for duplex TX/RX serial communication. In my design, Arduino board is the master and the android smart phone is the slave. I made this design choice because writing to bluetooth module buffer is not blocking while listening to incoming data is blocking. So, if Arduino board wants to listen to incoming data from android smart phone, it has to have a separate thread doing so. For simplicity, I decided not to introduce multi-thread to Arduino microcontroller. In other words, to keep things simple, Arduino writing to bluetooth buffer is allowed and Arduino listening to incoming data will not allowed.

Because of the aforementioned implementation decision, Arduino only knows the time interval between each revolution instead of the real speed because the wheel size cannot be set by user to Arduino microcontroller. So Arduino microcontroller only transmits the time interval as raw data to the smart phone, and the android app takes over the task of calculating as well as displaying the real-time speed.

2.3 High-level Design of Android App

The Android app is the only user interface (UI) for the whole design system, so it should have both AI and speed & heart rate monitor implemented. The development of the Android app mainly consists of two parts: functionality realization and UI design. These two parts of design are dependent and nested with each other, so a systematic scheme is indispensable before any actual software development being done.

To develop a heavily customized android app, the best design methodology is incremental design, which is also known as divide & conquer. In consideration of functionality, a music player with different playlists is the base of the app. So, my first step of the software development is building a well functioned music player (bare-bone version is the available resources on-line [1]) with 4 playlists: one (we will call it overall playlist below) containing all the music, the other three (we will call it functional playlists below) containing music to be played in different mode (identifying as energetic, tired, and normal mode). User can add different music to different functional playlists when browsing the overall playlist, or users can hit previous & next button in main interface to switch to different music and hit corresponding heart button to add the current playing music to different functional playlists. After this design step, the main UI is shown in Figure 1.

The second step of my app development is handling the bluetooth connection and wireless data transmission. In this step, two UI activities are added – bluetooth pre-connection UI and setting UI (shown in Figure 2 and Figure 3). Bluetooth pre-connection UI is responsible for handling connection to the Silver Mate Bluetooth module before the main UI. In setting UI, Users can enter the wheel size and enable speed measurement using a toggle button. When finishing the second step, the android app can display the current bike speed as well as the wheel size on main UI while playing music.

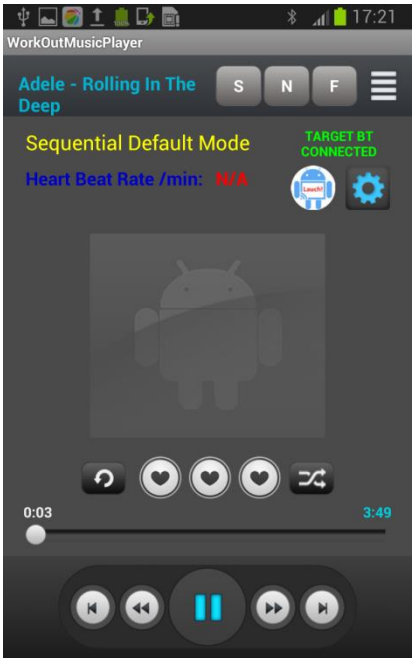

Figure 1 – Main UI Design

Figure 2 – Bluetooth Pre-connection UI

The third step of my app development is learning speed & heart rate pattern. In this step a training phase UI is added. Users can enter training phase by clicking on speed display field on screen, and the training phase takes real-time speed and real-time heart rate as training attributes as well as taking corresponding binary tags which indicate whether the biker is tired or not as training classification results. In this training phase, the real-time data will be displayed at UI as shown in Figure 4, and user can click the image button to toggle the binary training classification tags with 1 indicating “tired” and 0 indicating “energetic”. The training data will be written to a file in external flash memory which will be used for part of switching playlists AI when the on-line learning starts.

The forth step is building the playlists switching AI. Generally, the combination of KNN algorithm, average speed hysteresis control (details discussed in section 4.3) and on-line expert learning method is chosen as the control AI. In this step, the prediction result of whether the biker is tired or not will be displayed on screen, and music playlists is switched from one to anther also by this prediction result. Users can click the prediction result field on screen to indicate a wrong prediction, and the corresponding real-time attributes and classification tag will be added to training data as new samples. This is

exactly what on-line learning should do. Meanwhile, two seek bars are added in setting UI to let users change the desirable controlling sensitivity for speed hysteresis control. Control information including hysteresis threshold and average speed update frequency can be displayed on screen by clicking the setting button after on-line learning starts.

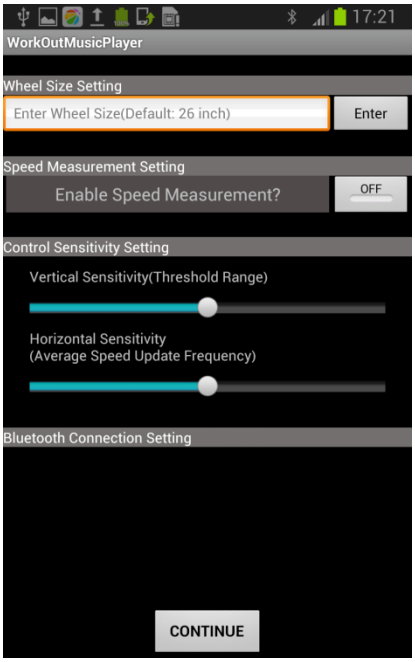

Figure 3 – Setting UI

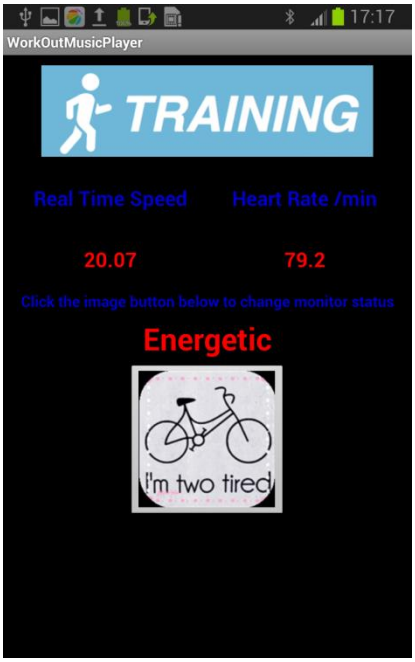

Figure 4 – Training UI

The last step is testing the app in a real bike system and debug & optimizing the system in functionality level. In this step, the playlists switching AI will be thoroughly tested on prediction accuracy, running efficiency as well as robustness.

The design cycle of android application is shown in Figure 5.

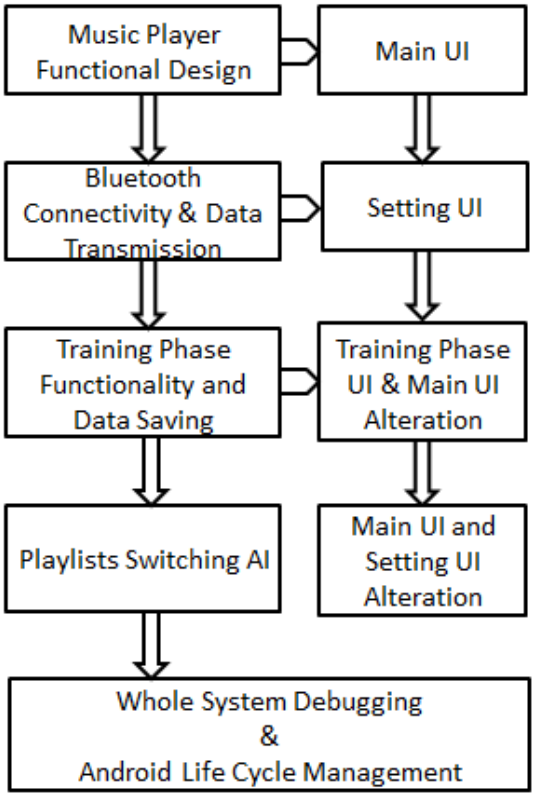

Figure 5 – Android Application Design Cycle

3. Bike Embedded Hardware Based on Arduino Board

In this section, I will talk in details about one of the two sub-systems in the whole design system: the bike embedded hardware. The bike embedded hardware is responsible for counting the time interval between two consecutive revolutions, and calculating the time interval between two consecutive heart beats. It should also be responsible for driving the bluetooth module and transmit the raw data to the android smart phone. Test of the functionality is important in terms of stability and reliability. The overall bike embedded system hardware is shown in Figure 6.

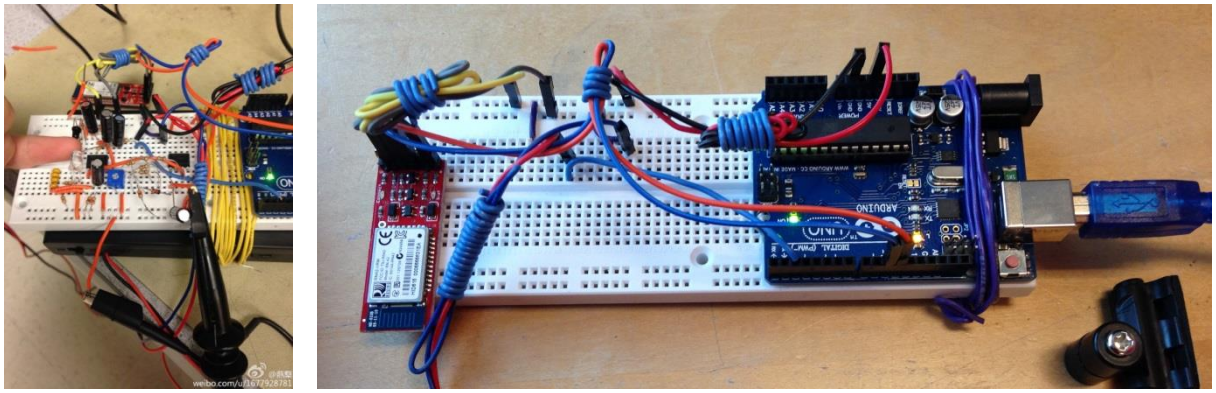

Figure 6 – The Bike Embedded Hardware Overall

3.1 Speed Measurement Principle

In my design, a pair of magnet & magnetic sensor is used for signaling each individual revolution. The design choice has been discussed in the first section. In this part, I will discuss the design & principle details about the speed measurement hardware.

The operating principle of the magnetic sensor is shown in Figure 7. Basically, it follows the Hall Effect. The behavior of the magnetic sensor is that when being near a strong magnetic field, it behave like a short circuit, when removing the magnetic field, it behave like an open circuit. Based on this behavior, I mounted a magnet on the spoke of the wheel, and then I fixed the sensor on the hub. So, for each revolution, the sensor meets the magnet for one time and the electric property between the two points of the sensor gets changed for one time. This electric property change can be used for signaling a whole revolution. Figure 8 shows how the magnetic & magnetic sensor pair is fixed on the bike.

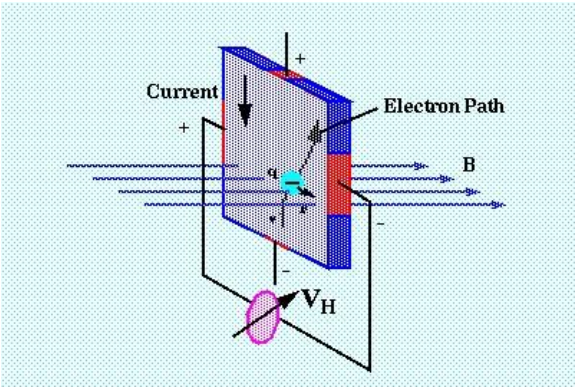

Figure 7 – The Magnetic Sensor Operating Principle

Figure 8 – The Detailed Position of Magnet & Magnetic Sensor

3.2 Heart Rate Meter Implementation

The heart rate meter can be built using noninvasive infrared light which probes blood pressure and pulse rate in a fingertip. The principle of using infrared light to probe heart rate is that finger absorbs more IR if there is more blood and since each heart beat sends more blood into the finger, the IR absorption is changed. However, this change is very small, so a serious of amplifiers is needed. Meanwhile, the band-pass filters are also needed because it helps to get rid of unwanted steady signals and noise at higher frequencies, avoiding them from being amplified too much and blocking the desired small changing signal. The circuit schematic of the heart rate meter is shown in Figure 9. [2]

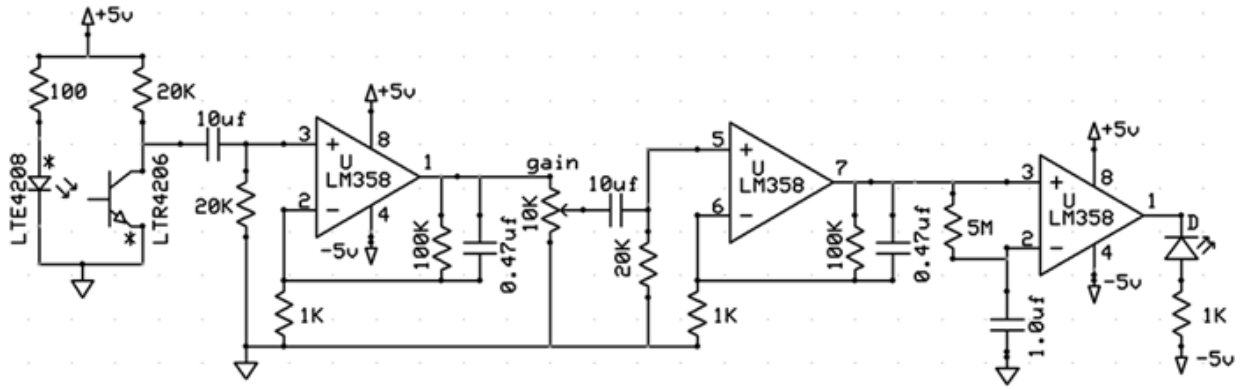

Figure 9 – The Schematic of Heart Rate Meter

3.3 Hardware Schematic

The output of the magnetic sensor and the heart rate monitor are fed to two external interrupt ports in Arduino board, and counting for time intervals are handled in the interrupt service routine (ISR). The block diagram for the whole system is shown in Figure 10.

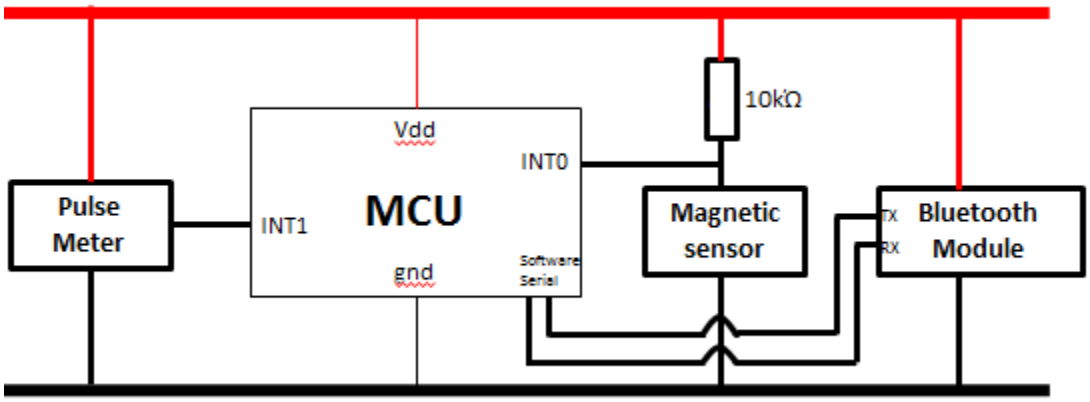

Figure 10 – Schematic for the Whole Bike Embedded Hardware System

3.4 Arduino Software Code

One consideration of choosing Arduino board rather than building my own custom MCU board is that Arduino IDE & relevant usable library makes software development on MCU very easy and neat. Arduino has a SoftwareSerial library which can extend any output ports of the MCU to TX/RX serial port. Since Arduino UNO board use ATmega32 MCU, which has only 1 TX/RX port and is used to download code from Arduino IDE,

SoftwareSerial library is extremely useful for handling other TX/RX serial connectivity. As for counting time interval between two consecutive revolutions, an Arduino built-in function called millis() which returns the current time since last time reset in the unit of millisecond can be used. For a typical bike speed which is 10 m/s and a typical wheel size which is 26 inches, the time interval for a whole wheel revolution is 207.24 ms. So using Arduino system function millis() to calculate time interval of a whole wheel revolution only introduce up to 0.5% error. For the typical heart rate which is 60 beat/min, using millis() function only introduces 0.1% error. So, we can use millis() function for both attributes calculation since it is accurate enough for this design system. Meanwhile, the counter which millis() function uses overflows in 49 days, which is obviously enough for maintaining the correctness of counting time interval for the bike system.

4. Android App Development

The real challenge for this design project is developing this highly customized android app. The android app developing cycle is shown in Figure 5. In this section, I will talk in details about the functionality implementation and the AI design choice.

4.1 A Simple Android Music Player

The functionality of the simple music player which is the base of the app has been discussed in section 2.3. In this section, the implementation details are mainly focused on.

Eight classes are associated with the functionality of the simple music player: class WorkOutMusicPlayerActivity, class PlayListActivity, class PlayListFastActivity, class PlayListNormalActivity, class PlayListSlowActivity, class SongsManager, class PlayListManage, and class Utilities. Basically, each of them is responsible for one specific functionality or UI performance. By calling getPlayList() in SongManager, an ArrayList containing information(name and path) of all songs in external SD card music folder(Absolute path: /storage/extSdCard/music/) in the form of HashMap<String, String> would be returned. Collaborating with a method public void playSongs(int songIndex) in class WorkOutMusicPlayerActivity, which extracts music path from the aforementioned HashMap and add it to the data source of a MediaPlayer instance, a single song can be played.

Class PlayListActivity is responsible for generating a ListView of all music. Each music in the ListView can be clicked to play or be added to different functional playlists. Class PlayListFastActivity, PlayListNormalActivity, and PlayListSlowActivity are responsible for generating ListView of corresponding functional playlists. Class Utilities contains all the method needed to calculate and update timer SeekBar in music player main UI.

In class WorkOutMusicPlayerActivity, other than main UI thread, there is another background runnable thread, mUpdateTimeTask which updates the timer bar of the music player main UI.

4.2 Bluetooth Connection and Data Transmission

Android OS has rich resources of higher level API and relevant classes which can handle Bluetooth discovering, connection, and duplex data transmission including writing to Bluetooth TX write buffer and listening for incoming raw bytes.

Class BTConnection handles bluetooth connection and data transmission by creating a bluetooth socket and getting input and output stream from this socket. First, this activity (this class extends class Activity) searches for all available bluetooth devices using an instance of bluetooth adapter and put them in a ListView. Then for each item of the ListView, onClick listeners are set to enable the corresponding bluetooth connection. Basically, bluetooth connection is handled in a separate thread, and the connection socket is obtained by calling createRfcommSocketToServiceRecord(UUID) method of the bluetooth device. UUID (Universal Unique Identifier) represents some common service protocol that bluetooth device supports. Specifying UUID when creating rfcomm service makes the client which connects to the host being able to identify the host. For a serial port, the UUID can be defined as follows.

```
public static final UUID MY_UUID = UUID.fromString("00001101-0000-1000-8000-00805f9b34fb");
```

After generating the socket, the connect method in BluetoothSocket instance can be called to launch the real connection. The simplest way to pass the BluetoothSocket to another class which extends Activity is to create a VariableHolder class and initiate BluetoothSocket field in it. The VariableHolder class can be accessed by any classes in the same package; by adopting this small trick, the problem of intent.putExtra not being able to pass BluetoothSocket to another activity can be easily solved.
After Bluetooth connection, WorkOutMusicPlayerActivity is invoked. An inner class BTtranceiver which extends class Thread is used to maintain duplex data transmission (read and write) and displays the incoming data on the main UI. The Bluetooth read which listens to incoming data is blocking, so Bluetooth read cannot be handled in main UI thread, otherwise it would block the UI and make everything visible very slow. The bluetooth write, on the other hand, is not blocking, and can be handled in main UI

thread. Constructing an inner class which extends Thread to maintain duplex data transmission is a good and simple design choice since it allows read and write to be constructed in the same class but in different thread by overriding run method, letting bluetooth listening method running in background thread, and constructing regular method bluetooth write in UI thread. In order to make sure the data displaying having good instantaneity, this new thread should be granted highest executing priority. Note that the raw bytes from bluetooth read buffer should be processed to corresponding real-time speed or real-time heart rate before Handler post them on UI.

4.3 Implementation of Playlists Switching AI

After clicking the launch button, the on-line learning starts and the prediction result is used for playlists switching. The major idea behind this AI choice decision is finding the best trade-off between learning efficiency and prediction accuracy. The playlists switching AI should have high learning and predicting efficiency because there are limited hardware resources in a smart phone and there are already a bunch of applications sharing the limited resources. The learning AI should have good calculation efficiency in order to keep the UI responds quickly. Meanwhile, the learning AI should also have good prediction accuracy, but the more sophisticated the learning model is, the more intense the calculation would be. So, although kernel SVM (Support Vector Machine) usually has good practical prediction accuracy and has good overfitting control mechanism, it is not a good learning AI in this design because solving convex optimization problem would consume a lot of CPU resources and thus makes the application extremely slow.

In this design, the playlists switching AI is a combination of KNN (K Nearest Neighbor), average speed hysteresis control, and on-line expert learning. It will be discussed in details in this section. Figure 11 shows what the main UI looks like after AI learning algorithm starts to run.

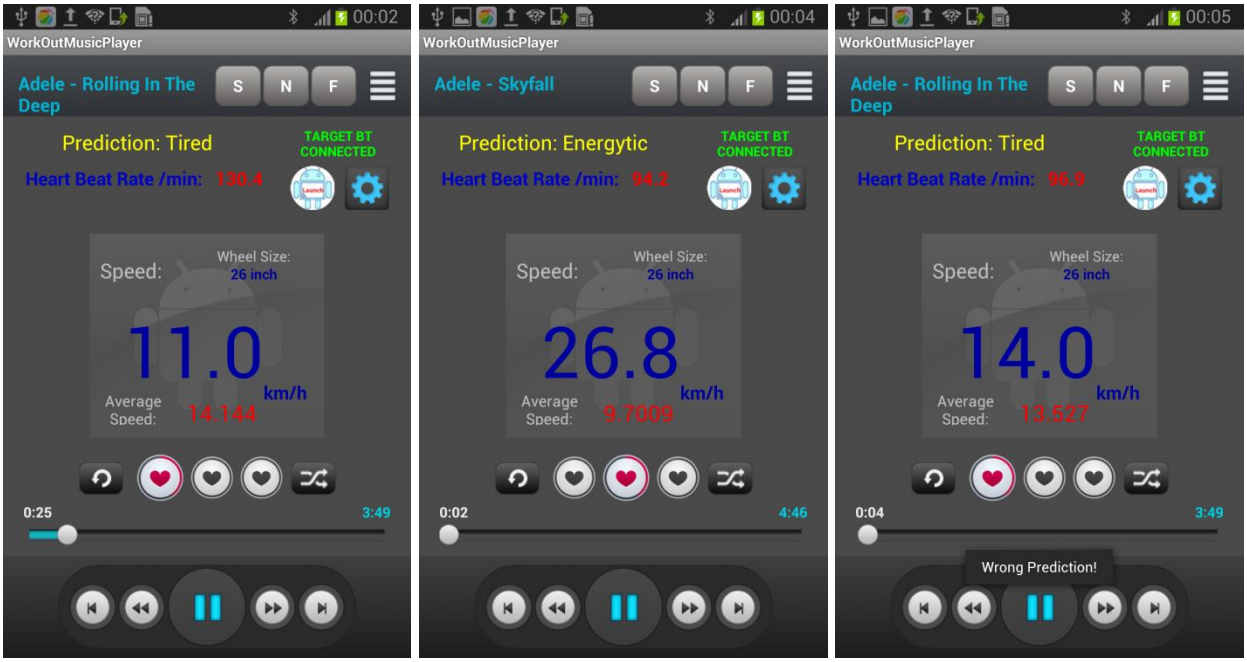

Figure 11 – Main UI after AI Learning Algorithm Starts

(The third picture indicates wrong prediction after user clicking the mode display field)

4.3.1 K Nearest Neighbor (KNN) Methodology

The KNN algorithm is among the simplest of all machine learning algorithms. It is a type of instance-based learning, and the basic idea of this principle is to weight the contributions of the neighbors so that the nearer neighbors contribute more to the average than the more distant ones. A newly observed sample can be classified as in the same class as the majority votes of its k-nearest neighbors'.

The training examples are vectors in a multidimensional feature space, each with a class label. And the training phase of the algorithm consists only of storing the feature vectors and class labels of the training samples. In KNN, a similarity function is needed to find the nearest neighbors. A commonly used similarity function for continuous variables is Euclidean distance, and it is also a good choice in this design. Intuitively, the more similar the newly observed sample is to a training sample, the more likely that the class labels are the same.

4.3.2 Average Speed Hysteresis Control Method

Dynamic average speed hysteresis control is a very simple but efficient method for tiredness prediction. The principle behind is very straightforward and intuitive. Basically, the average speed in a certain period of time is used as the separating point of whether the biker is tired or not. In order to introduce some robustness and increase the prediction accuracy, the hysteresis is adopted. To be specific, when real-time speed exceeds some certain value plus the dynamic average speed, the prediction decision is changed to “energetic”; when real-time speed is less than dynamic average speed deduct some certain value, the prediction decision is changed to “tired”. The certain value mentioned above is the sensitive threshold which can be adjusted in setting UI. The hysteresis dynamic average speed control is shown in Figure 12.

Dynamic average speed is the average speed in a period of time which is updated frequently (the update frequency can be set by users in setting UI). It is calculated in the following way. In the bluetooth listening thread, real time speed is pushed into an ArrayBlockingQueue which has capacity of 100 elements. In the AI prediction thread, When ArrayBlockingQueue gets full, it calculated the average speed based on all the elements in that blocking queue, and then pop out some number of old real time speed data. The number of elements being popped out is another indicator of controlling sensitivity, and it can also be set in setting UI by user. One of the most important things in maintaining the ArrayBlockingQueue is that it should never be full for a long time, otherwise it will block the bluetooth listening thread and make the main UI act extremely slow.

To better visualize the control parameter (hysteresis threshold and dynamic average speed update frequency), a new feature is introduced to the setting button. After launching the control mode, clicking on setting button would return a dialog with basic information of control parameter as shown in Figure 13, that is, two hysteresis boundaries, and an average speed update distance.

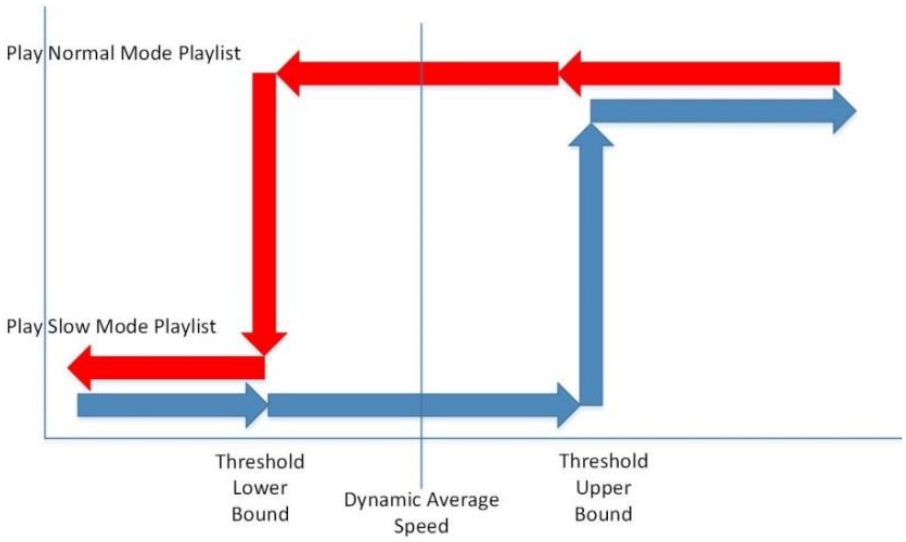

Figure 12 – Hysteresis Dynamic Average Speed Control

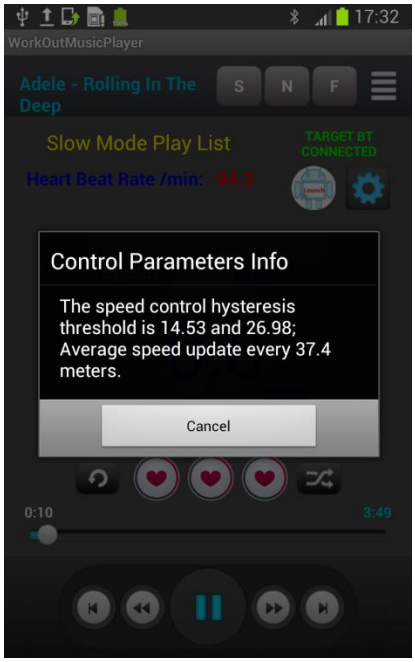

Figure 13 – Dialog Showing Control Parameter Information

4.3.3 On-line Expert Learning Methodology

In this design, KNN (K Nearest Neighbor) algorithm and average speed hysteresis control both work as an individual prediction expert. An on-line expert learning mechanism is used to combine these experts' prediction together and make the final prediction decision. To be specific, the prediction result (tired or energetic) is based on the majority votes of both experts' prediction multiplied by their own confident weight. Every time there is a misprediction reported by user, the corresponding mispredicted

experts will have to reduce its confident weight by half. This on-line expert learning mechanism guarantees that the “bad” expert will have little influence on the prediction decision after making certain amounts of mistakes. Furthermore, it is called on-line learning because the experts learn from mistakes. Every time a mistake is made, the real-time speed and heart rate as well as classification tag would be written to training data file as a new training sample. The next time KNN algorithm runs, the AI can learn from more samples, which helps increasing the future prediction accuracy.

4.4 Maintaining Multi-thread in Android App

As was stated in the previous part, four threads are running in the android app design: main UI thread, Bluetooth data listening thread, controlling thread, and the thread which updates timer bar in main UI. It is important to maintain the concurrency of these running threads since there are sharing resources between threads. Basically, if a method in a class is defined as synchronized, the instance of this object can only be grabbed and updated by one thread at a time. So, two classes called SpeedLearnFlagHolder and BTReadThFlagHolder are created and the methods in them are defined as synchronized to maintain concurrency for sharing resources when different thread trying to access them at the same time.

4.5 Life Cycle of the Android App Design

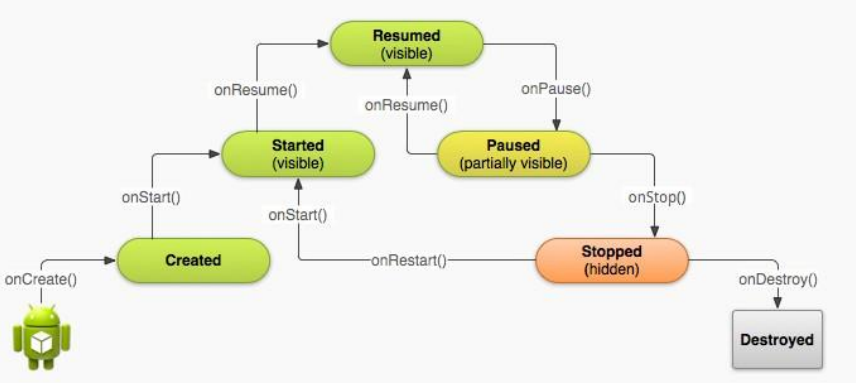

Figure 14 – Life Cycle of an Android Activity

Figure 14 shows a simplified illustration of the Activity lifecycle, expressed as a step pyramid. This shows how, for every callback used to take the activity a step toward the

Resumed state at the top, there's a callback method that takes the activity a step down. The activity can also return to the resumed state from the Paused and Stopped state.

In my app design, all the setup work is done in the overrode onCreate() method. For class WorkOutMusicPlayerActivity, that includes setting up content view of layout file (a .xml file in res/layout folder), defining all the functionality of Button, ListView, TextView, Handler, etc., setting up three different running thread, setting up all the object activity listener, handling music play, etc. The onPause() method should not be overrode, and in the onDestroy() method, the Bluetooth connection should be canceled, the BluetoothSocket static field in VariableHolder class should be set back to null, the ArrayBlockingQueue should be emptied, all the playlist should be set to null, all the running thread should be interrupted and set to null, and all the MediaPlayer resources should be released.

5. Experimental Result

In this section, various types of experiments are done to verify the functionality of this design project. There are four major parts that need to be tested. The first one is the speed measurement, which verifies both the circuit and ISR processing feasibility. The second one is the heart rate measurement, which verifies the circuit reliability. The third one is the bluetooth transmission which should be proved quick and error-free. And the last one is the prediction results (tired or energetic) verification.

5.1 Speed Measurement Unit Test

Speed is measured using magnet & magnetic sensor pair and the time interval of a whole wheel revolution is calculated in an external ISR. For unit test of this speed monitor, the output signal of the magnetic sensor should have periodically steady time intervals when user peddles in a constant rate. When user peddles in increasing rate, the output signal should also become denser.

The output signal of the magnetic sensor is shown in Figure 15 and Figure 16, from which we can see that the time interval differentiates little when user peddles at a constant rate, and time interval becomes shorter when peddling speeds up. PuTTy is used to print out the time intervals calculated in ISR in the setting of constant peddling rate, and the result is shown in Figure 17. The standard deviation of the measurement data shown in Figure 17 is 6.11. As a result, compared with the mean which is 403.17, the standard deviation can be considered small enough, which means the real time speed measurement is steady and reliable.

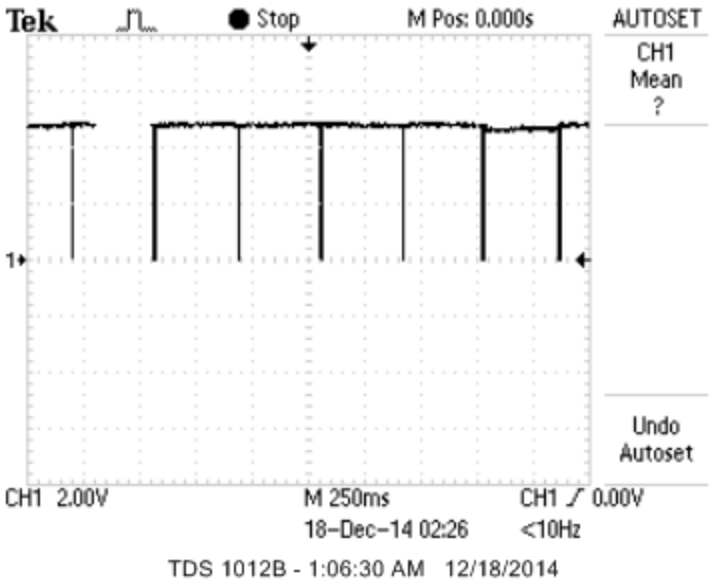

Figure 15 – The Output of the Magnetic Sensor When User Peddles in Constant Speed

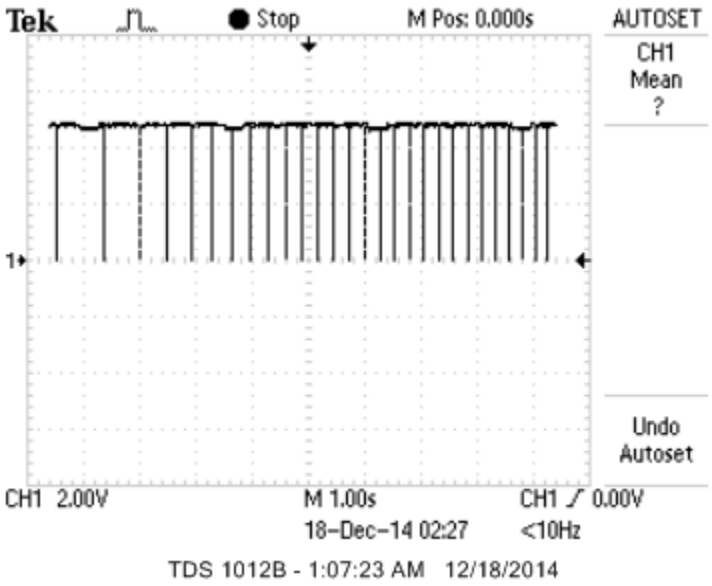

Figure 16 – The Output of the Magnetic Sensor When User Accelerative Peddles

Figure 17 - PuTTy Window for Speed Measurement Test with the Time Interval in Milliseconds Transmitted

5.2 Heart Rate Measurement Unit Test

The heart rate measurement circuit is shown in section 3.2. The output of the second amplifier is shown in Figure 18. After auto-threshold adjustment, the signal is sharpen based on average voltage of the transient period, and the corresponding output signal is shown in Figure 19. PuTTy is used to record the measurement data of heart rate and the result is shown in Figure 20. To test the reliability of the heart rate meter, 42 measurement data which indicates time intervals between two consecutive heart beats in milliseconds is recorded when the user's physical condition is steady. The distribution of these heart rate measurement data is drawn in Figure 21. By writing a python script, we can get that the mean of these data is 928.34 and the standard deviation is 28.22. We can see from the histogram (Figure 21) that the measurement data approximates normal distribution, in which the mean is a good representation of the real heart rate and the standard deviation is small enough compared with the mean so that the error range of the measurement is allowable.

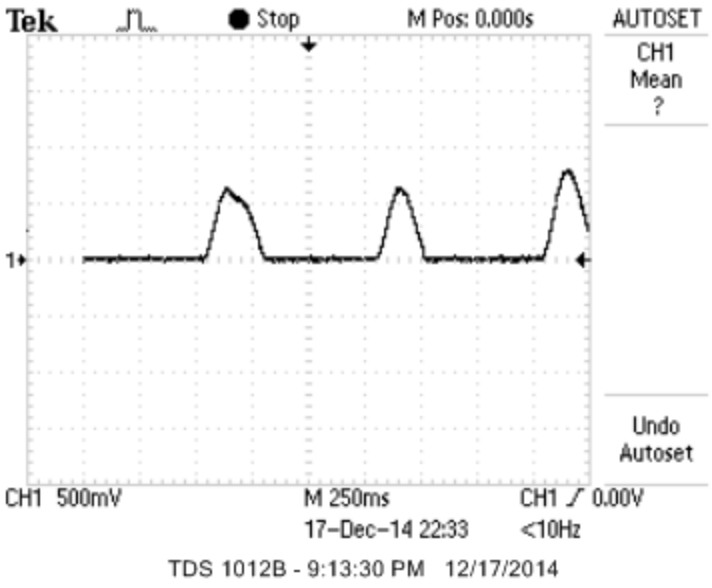

Figure 18 – Output Signal of the Second Amplifier in Heart Rate Meter Circuit

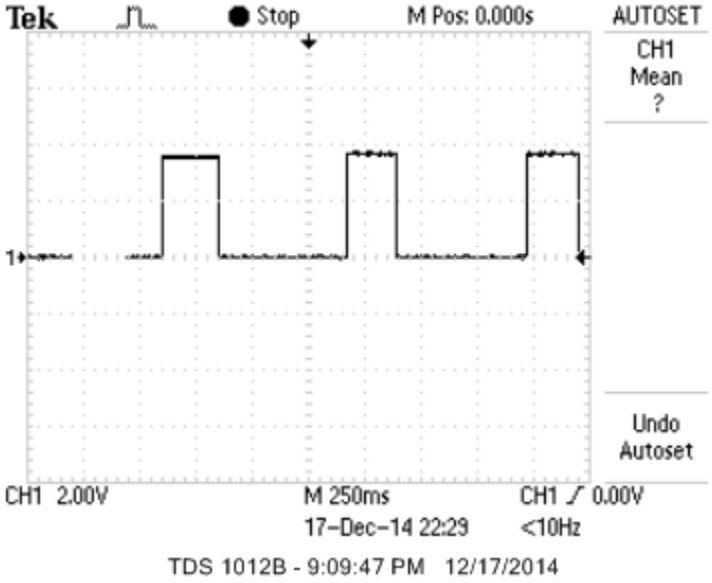

Figure 19 – Output Signal of the Heart Rate Meter Circuit

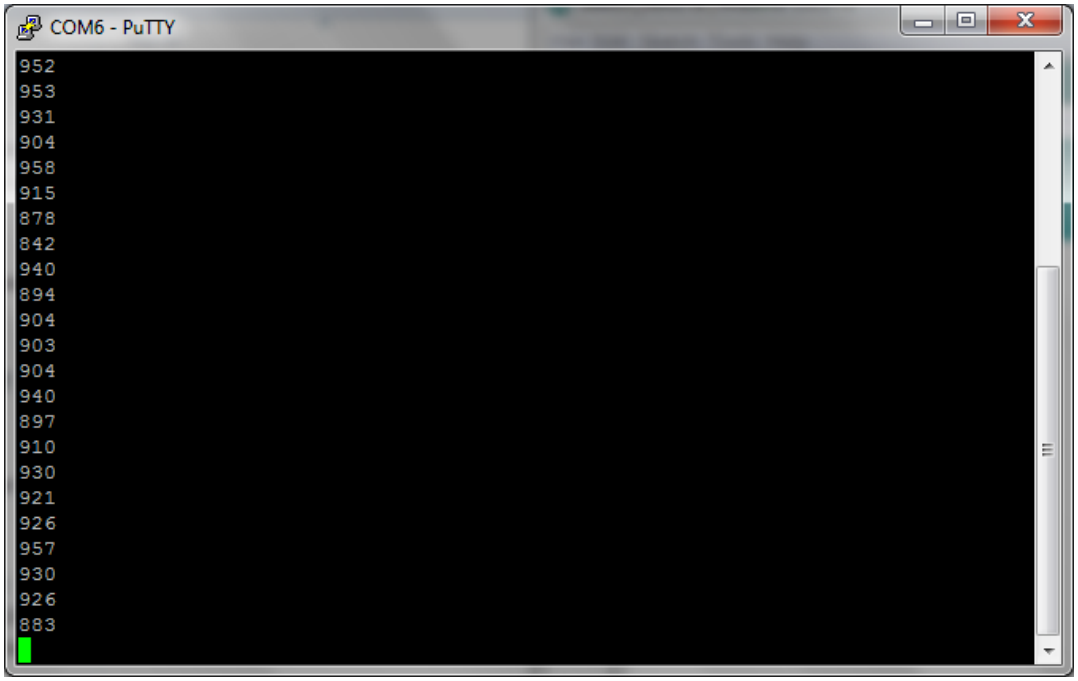

Figure 20 – PuTTy Window for Heart Rate Measurement Test with the Heart Beat Interval in Millisecond Transmitted

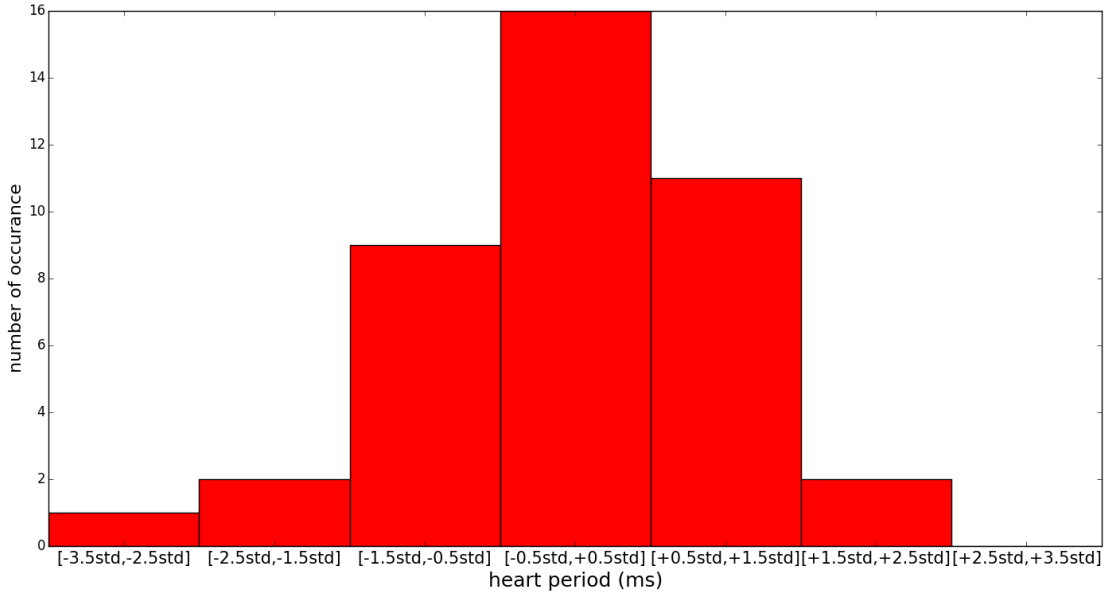

Figure 21 – Distribution of the Same Measurement

(Mean: 928.34, Standard Deviation: 28.22)

5.3 Bluetooth Transmission Test

As was discussed in former sections, bluetooth data transmission should be both quick and reliable. The bluetooth module is driven by Arduino microcontroller as software UART and the transmission is done periodically in Arduino's loop() function. Bluetooth serial debugger is used to test the quick and reliable bluetooth transmission. Hardware UART of the Arduino board which transmits the same data via USB to PuTTy is used to work as a control group so that the bluetooth transmission reliability can be tested. The testing result is shown in Figure 22, it is the same as what PuTTy shows.

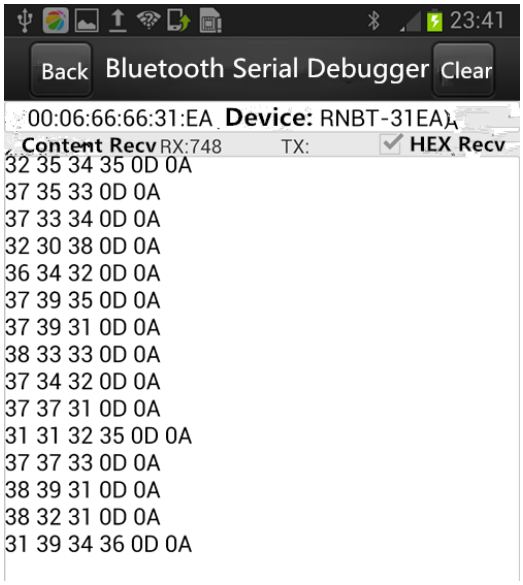

Figure 22 – Real Time Data in Bluetooth Serial Debugger

5.4 AI Performance

In this section, the prediction performance of AI is tested. As mentioned in section 4.3, there are two experts making predictions individually. They are KNN and dynamic average speed hysteresis control. The final prediction result is the weighted average of both experts, and each expert's confidential weight is the same at the beginning and reduced by half every time user reports a misprediction which is caused by corresponding expert's wrong decision. The AI performance is tested in three steps. First, after long times of on-line training, in which the reported misprediction is written into KNN training data, KNN training samples stored in the phone external storage space should be good and representative. These training samples should be sufficient

enough for correctly classifying new samples in low error rate. So the first step of the AI performance verification is testing the KNN misprediction frequency after long time training. Second, the other expert – dynamic average speed hysteresis control should be tested separately for prediction accuracy. Third, the on-line expert scheme as a whole should be tested for prediction accuracy. All the test should be road test.

5.4.1 KNN Training Data after Road Test

As is described in section 4.3, there are two ways for KNN to accumulate training data. In training phase, all the user feedback and the attributes (real-time speed and heart rate) are written to an external storage file as basic KNN training data. After the training phase, KNN accumulates new training samples every time it makes a misprediction. So we can expect that after a long time of training and reporting misprediction, KNN model can gather sufficient training data which gives good prediction with low error rate. To verify this, I did a simple road test.

First of all, I train the KNN in the training phase as discussed in section 2.3. After the training phase, I read out the training data and plot them in Figure 23. We can easily see from the figure that there are two clusters far apart from each other, and the KNN decision boundary is nearly a horizontal line, which means heart rate works as the major contribution for the prediction result. This result is reasonable since in the training phase, we only train KNN with very limited number of scenario, so we cannot expect the training data will provides sufficient accuracy for classifying all combination in 2-D plane as shown in Figure 23.

Second, after the basic training phase, KNN will predict based on the decision boundary shown in Figure 23. After 6 misclassification reported by user, new KNN training samples are written in external storage file, and the new KNN decision boundary is as shown in Figure 24. We can see from this figure that there are a big change in decision boundary since new points added to Figure 24 are all misclassified samples, which is mean to improve the KNN prediction accuracy. Meanwhile, we can see that the decision boundary becomes not smooth and at some place not reasonable since only 6 new samples are added to the plot and they are only local representative. With the number of misclassification report increasing, we can expect the decision boundary becomes

more reasonable. Figure 25 shows that when 5 more new points are added to the plot, the decision boundary becomes smoother and more reasonable. Based on Figure 23 – 25, we can expect that after a long time of training and reporting misprediction, KNN model can gather sufficient training data which gives good prediction with low error rate.

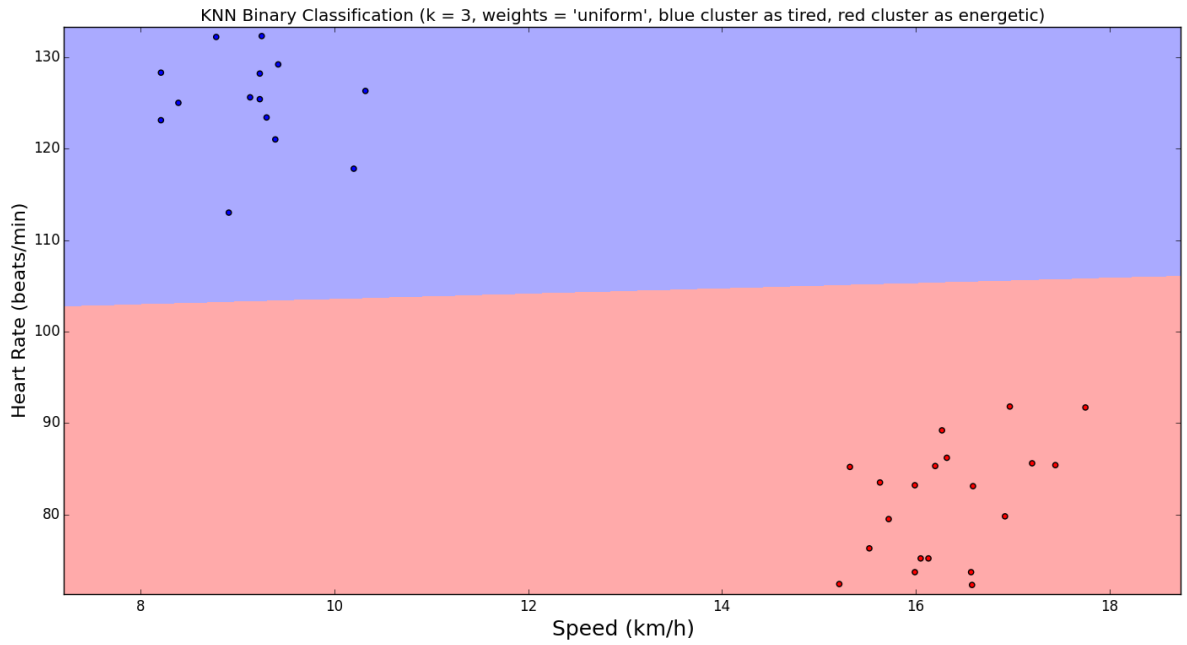

Figure 23 – KNN Training Samples and Decision Boundary After a Simple Training Phase

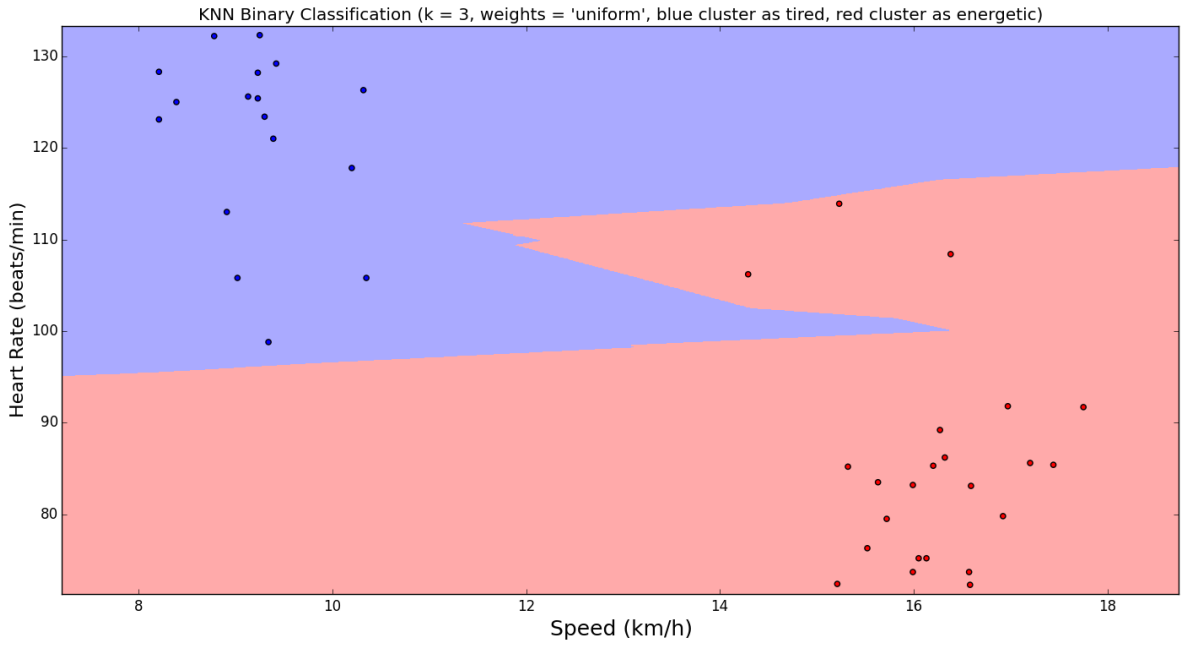

Figure 24 – KNN Training Samples and Decision Boundary After 6 On-line Training Misclassification Report

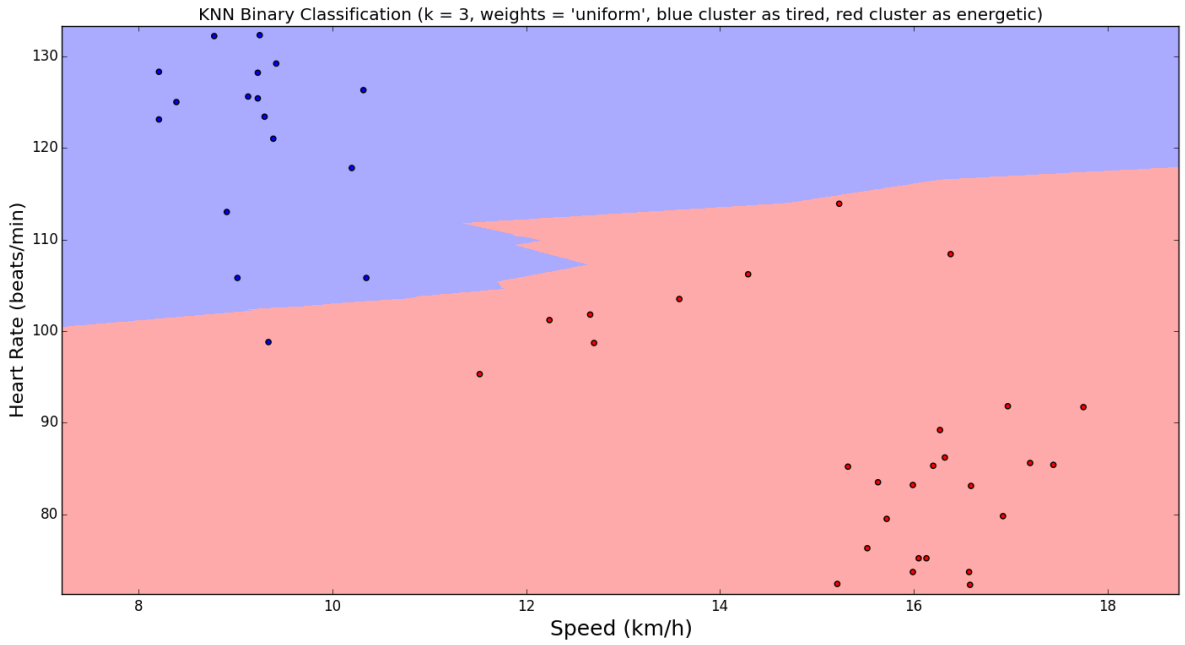

Figure 25 – KNN Training Samples and Decision Boundary After 11 On-line Training Misclassification Report

5.4.2 Road Test for Dynamic Average Speed Hysteresis Control

Dynamic average speed hysteresis control is a simple but efficient way to make prediction of whether the biker is tired or not. The major advantage of this method is that it does not require any training so that it is simple enough to maintain. Meanwhile, it has sensitive responds to external scenario change. To verify the real performance, this prediction expert was tested individually in a road test. The experience shows that this experts works well for long time non-stopping workout scenarios. In dense traffic light scenarios, a deceleration caused by road control would be mispredicted as biker getting tired.

6. Conclusion and Future Work

In this design, I build a smart android music player for bikers. KNN, average speed hysteresis control, and on-line expert learning are built as decision AI for music playlists switch prediction. The whole system contains not only the android software, but also contains Arduino microcontroller and peripheral circuit design. These two parts are designed individually and test individually. At last, a system level test is performed for debugging and optimization.

There are still some possible future works that can be done to refine the whole design.

1. More thorough road test should be done to verify the AI performance. The road tests should contain comprehensive traffic and road scenarios, and the accuracy criteria should be quantified in terms of number of mispredictions per mile.
2. The prediction is made by learning from heart rate and speed pattern. In the future, more attributes can be added such as gravity, slope, weather and so on.
3. Currently, the app is highly customized for single user because all the training data is store in one file. In the future, multiple user support can be made in the app.
4. Resources optimization can be further made in order to reduce the chance of thrashing. This app is intensively resources consuming because it has a lot of background running thread and high level of I/O traffic. Optimization should be made in order for this app to be a good residence and not killed by android operating system.
5. For safety consideration, buttons which are used for training feedback and misprediction report can be added to the handlebar. This avoids user clicking on phone screen when riding the bike, which makes the whole design system safer to use in reality.

Reference

[1] Avi Tamadaon. (2012). Android Building Audio Player Tutorial.

http://www.androidhive.info/2012/03/android-building-audio-player-tutorial/

[2] Bruce Land. (2011). Pulse Meter Project.

http://people.ece.cornell.edu/land/curie/curie_2011/Projects_2011/Pulse_meter_project_brl4.pdf