BICYCLE CRASH DETECTION: USING A VOICE-ASSISTANT FOR MORE ACCURATE REPORTING

by

BRIAN WILLIAMS

A THESIS

Presented to the Department of Computer and Information Science
and the Graduate School of the University of Oregon
in partial fulfillment of the requirements
for the degree of
Master of Science

June 2018

THESIS APPROVAL PAGE

Student: Brian Williams

Title: Bicycle Crash Detection: Using a Voice-Assistant for More Accurate Reporting

This thesis has been accepted and approved in partial fulfillment of the requirements for the Master of Science degree in the Department of Computer and Information Science by:

Dr. Stephen Fickas	Chair

and

Sara Hodges	Interim Vice Provost and Dean of the Graduate School

Original approval signatures are on file with the University of Oregon Graduate School.

Degree awarded June 2018

© 2018 Brian Williams
All rights reserved.

THESIS ABSTRACT

Brian Williams

Master of Science

Department of Computer and Information Science

June 2018

Title: Bicycle Crash Detection: Using a Voice-Assistant for More Accurate Reporting

It is estimated that over half of bicycle crashes are not reported. There are various reasons for this, such as no property damage or physical injuries sustained. In order to improve the likelihood that bicycle riders will report a crash, I have developed Urban Bike Buddy, a smartphone application which uses the internal sensors of the device to detect a crash. The application interacts with Alexa to help guide the user through the crash reporting process.

The innovative features of my work are the ability to initiate communication with Alexa without user interaction. In addition, there is an intersection controller that has been connected to extra hardware that allows bicycle riders to request a crossing signal during their approach based on the speed that they are riding. These features add value to bicycle riders, and will help contribute to a safer environment for bicycle riders, automobiles, and pedestrians as well.

CURRICULUM VITAE

NAME OF AUTHOR: Brian Williams

GRADUATE AND UNDERGRADUATE SCHOOLS ATTENDED:

University of Oregon, Eugene, OR, USA
Lane Community College, Eugene, OR, USA

DEGREES AWARDED:

Master of Science, Computer and Information Science, 2018, University of Oregon
Bachelor of Science, Computer and Information Science, 2010, University of Oregon
Associate of Science, Computer Science, 2007, Lane Community College

AREAS OF SPECIAL INTEREST:

Internet of Things (IoT)
Cloud Computing
Data Mining
Software Engineering
Robotics

PROFESSIONAL EXPERIENCE:

Software Engineer, BHS Pharmacy, 2010-present

ACKNOWLEDGEMENTS

I would like to acknowledge the following individuals, without whom this project would not have been possible. First, my wonderful wife Jackie. Her support and encouragement throughout my return to school has been indispensable. She ensured that I remembered to eat and always made sure that I took a break now and then so that I would not get burned out.

My research advisor, Dr. Stephen Fickas, was responsible for coming up with the original idea that became Urban Bike Buddy.

His desire to provide a better environment for bicyclists in Eugene helped me to realize the importance of this project, and pushed me to develop something that I would not have even considered otherwise.

Jason Prideaux was also a very great resource throughout this process. He allowed me to distribute the application to testers via the TestFlight platform. Although he was not required to provide this support, he ran the builds and updated TestFlight accordingly so that the users could receive new versions of the application as they became available. Without his help, testing would have been much more difficult.

Finally, I would like to acknowledge the City of Eugene. In particular, Matt Rodrigues and Andy Kading, the city's traffic engineers who worked with us on the project and allowed us to connect our experimental project to the intersection traffic controller. Again, without all of the above people, none of this would have even been possible. I am grateful for all of the support that I have received throughout this process.

I dedicate this work to the bicycle riding students and faculty of the University of Oregon.

TABLE OF CONTENTS

Chapter Page

I INTRODUCTION 1

Motivation 2

Vision Zero 4

Activities of Daily Bicycling (ADBs) 4

II METHODOLOGY 6

Development Environment 7

Xcode 7

Continuous Integration 8

Algorithms 10

Basic Threshold Monitoring 11

Fall Index 12

Two-Phase Detection (PerFallD) 12

iFall 12

Hardware 13

Particle Electron 13

Smartphone 14

GPS 14

Accelerometer & Gyroscope 16

Cloud Services 16

Amazon Alexa 16

Alexa Voice Services 17

Login With Amazon 17

Amazon Cognito 17

AWS Lambda 17

Amazon DynamoDB 18

Google Pub/Sub 18

Natural Language Processing 19

Google Cloud Speech 20

Chapter Page

Amazon Transcribe . . . 20

Push Notifications vs. Publish/Subscribe . . . 21

III RELATED WORK . . . 23

Snowboy . . . 25

iBikeEugene . . . 25

GIS Map . . . 25

IV RESULTS . . . 26

Evaluation . . . 26

User Feedback . . . 28

Open Issues . . . 32

Microphone Access . . . 32

Data Usage . . . 32

Battery Usage . . . 33

Sensor Accuracy . . . 33

Simulated Incidents . . . 34

Other Limitations . . . 35

V CONCLUSION . . . 36

Scope and Limitations . . . 36

Future Work . . . 36

APPENDICES

A SAMPLE DATA . . . 39

ADBs . . . 39

Notable Events . . . 43

Simulated Crash Events . . . 45

B ADDITIONAL CODE LISTINGS . . . 47

C URBAN BIKE BUDDY: INFORMATION FOR APP USERS . . . 48

Background . . . 48

Downloading the App . . . 48

Using the App . . . 48

What you see on the App . . . 49

Chapter Page

An important note . . . 49

Contact information . . . 49

REFERENCES CITED . . . 50

LIST OF TABLES

Table Page

2.1 CSV data sample 10

4.1 User 1 feedback 29

4.2 User 2 feedback 29

4.3 User 3 feedback 30

4.4 User 4 feedback 30

4.5 User 5 feedback 31

LIST OF LISTINGS

Listing Page

2.1 Configuring Location Services 15

2.2 Alexa skill configuration 18

B.1 Generating Bokeh Plots 47

CHAPTER I

INTRODUCTION

With the growing number of people who are choosing a bicycle as a mode of transportation, the safety and convenience of bicycling are becoming increasingly important. Providing added incentives to riding a bicycle may entice more individuals to choose a bicycle over an automobile. However, with more people riding bicycles, we also see an increase in incidents related to bicycling.

It is reported in *Demographics of Mobile Device Ownership and Adoption in the United States* (2018) that around 95% of American's currently own a cellular phone of some kind, with 77% of these being smartphones. I would like to take advantage of the capabilities of the smartphones that many people are already using. I use the combined sensor data of an embedded GPS, an accelerometer, and a gyroscope to detect when a bicycle crash occurs. This data is then shared with the city to help identify critical safety areas.

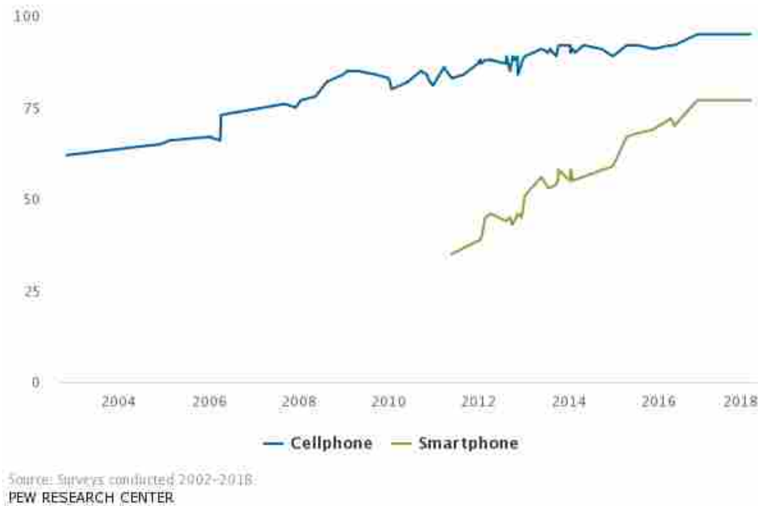


Figure 1.1 % of U.S. adults who own the following devices

Not only has the number of people who own smartphones continued to increase, the computing power available in these devices which we carry around with us every day has grown exponentially. For example, an iPhone 7 is powered by a quad-core 2.34 Ghz processor with 2 GB of RAM. This is more computing power than many personal computers had not long ago. In addition to this, many current smartphones are also equipped with multiple sensors such as a GPS, accelerometer, gyroscope, etc.

Motivation

The Urban Bike Buddy app is being developed as part of a research grant supported by the National Institute for Transportation and Communities (NITC). The goal is to improve the user experience of people on bikes by increasing the likelihood that traffic signals turn green as the bike approaches. Future iterations of the app may actively influence pre-determined, variable traffic general signals (they are responsive to the presence of people waiting) and may give cyclists information to speed up or slow down to catch an upcoming green light.

One motivation for this project is the growing use of voice assistants, such as Siri and Alexa, in our everyday lives. So far, device manufacturers have mostly locked-down the use of their hardware to their own voice assistant. Third-party voice assistants can be used, but there is added user interaction required. My goal is to use the sensors of a smartphone, an iPhone in this case, to detect certain movements and initiate communication with a third-party voice assistant. I have added this functionality to an application that I was developing that is used by bicycle riders to transmit their location to the city traffic control system, hopefully allowing for expedited crossing of certain intersections. The system is currently being developed at the corner of 18th and Alder. We are working closely with the city of Eugene, Oregon to integrate this capability to both allow a more pleasant experience for riders, and improve safety for bicycles, pedestrians, and motorized vehicles.

Another major motivation for Urban Bike Buddy is that while there is plenty of work being done in regards to the safety of motor vehicles, such as crash detection and crash prevention, there is not equivalent work being done regarding bicycle safety. During my research, I came across many studies focused on vehicle safety, but only a small number that focused on bicycle safety. For example, Bhavthankar and Sayyed (2015) have developed a stand-alone system powered by an ARM microcontroller which uses an accelerometer to detect a crash in a vehicle. If a crash is detected by their device, there is a GSM modem that allows an SMS message to be sent to a pre-determined emergency contact. The system will include the GPS location of the crash incident so that the emergency contact will know where it occurred. Their device also includes a gas sensor, temperature sensor, and an alcohol sensor. The data collected from these additional sensors allows the emergency contact to better determine the severity of the crash. The software powering the system can contact medical services as well. There is a function built in to

the software that allows the user to easily notify contacts that they are okay, thus preventing the unnecessary dispatch of services.

Others have taken crash detection a step farther and are working toward crash prediction. The hope is that predicting a crash before it happens can alert or assist the driver in preventing a crash altogether. Kim and Jeong (2014) are working on precisely this. In fact, they note that over 60% of automobile crashes are caused by driver inattention. Therefore, a crash prediction system can provide a first step toward subsequent actions such as attempting to evade an obstacle. Addressing the situation from this direction is a novel approach, and the algorithm discussed in their work is able to classify different scenarios such as being rear-ended, being cut off, and being collided with from the side. It is also able to attempt to predict near-miss scenarios of the aforementioned classifications.

Gurupriya, Kaviya, Mohana, Raj, and Bebington (2016) also developed a stand-alone system to be used by motorcycle riders inside of their helmet. Like Bhavthankar and Sayyed's device, the device proposed here includes an accelerometer, a distance sensor, a GPS, and a GSM modem. Their device is powered by an Arduino microcontroller, and also similarly to Bhavthankar and Sayyed, their device could use the embedded modem to send an alert to a nearby hospital or emergency service.

Other vehicle related research has been done to predict the severity of injury from an accident. Augenstein, Digges, Ogata, Perdeck, and Stratton (2001) developed the URGENCY algorithm, using data from on-board crash recorders to assist in identifying which crashes that are most likely to have time critical injuries. Although their work was done a decade ago, they noted at the time that approximately 250,000 people are involved in serious injury crashes in the United States annually. That doesn't seem like very many when compared to more recent statistics: an estimated 2.35 million people are injured annually in automobile crashes in the United States [1].

It is estimated that the actual number of bicycle accidents exceeds the officially reported number by nearly two times (Juhra et al., 2012). This is due to the nature of bicycle accidents, which often do not involve a third party or property damage. By allowing bicycle riders to easily report crashes from their smartphone, I hope to increase the number of reports and help provide the city with more accurate statistics than are currently available in order to address problem

[1] http://asirt.org/initiatives/informing-road-users/road-safety-facts/road-crash-statistics

areas. Eugene is a very bicycle-friendly city, but there is much room for improvement. One goal of this project is to begin to allow bicycle riders an all-around more inviting experience while riding in the city. At the time of this writing, only one intersection has currently implemented the functionality provided by Urban Bike Buddy, but it is my hope that if a positive outcome is observed, then it will just be the first of many intersections that will allow bicycle riders to request a signal change while approaching an intersection.

Vision Zero

Vision Zero is a strategy to eliminate all traffic fatalities and severe injuries, while increasing safe, healthy, equitable mobility for all. First implemented in Sweden in the 1990s, Vision Zero has proved successful across Europe and now its gaining momentum in major American cities [2]. As a participant in the Vision Zero network, Eugene is a great city in which to be researching bicycle safety. I have had the pleasure of working with people from the community who are involved in the planning, design, and management of programs related to traffic and bicycling. Their feedback has been a great resource in the development of this app. More details will be discussed in the User Feedback section.

Activities of Daily Bicycling (ADBs)

Activities of daily living (ADLs or ADL) is a term used in healthcare to refer to people's daily self care activities such as eating, bathing, getting dressed, working, and leisure activities. In order to obtain a meaningful measure of accuracy for my algorithm, I had to distinguish those activities that occur during a normal bike ride. I will refer to these activities as Activities of Daily Bicycling, or ADBs. Typical examples of ADBs are things like riding up a curb, running over a pothole, riding over rough terrain such as gravel, and many other things that we would not want to have trigger the incident detection algorithm. Many hours of "normal" riding were recorded by x number of volunteers in different terrain to establish a baseline for these ADBs.

A sampling of the ADB data measurements can be found in Appendix A. Most of the data shown is normal riding activity, but some, such as A.6 show a spike in the sensor data. This is precisely the type of thing that we want to avoid having trigger the crash detection. It could very easily have been a pothole or some other normal riding occurrence which was not a significant event. There are also some graphs showing the readings during known normal riding activities

[2] `https://visionzeronetwork.org/`

Figure 1.2 Map of current Vision Zero cities as of January 2018

that may cause a spike in the readings. For example, Figure A.16 shows the readings recorded while riding straight off of a curb. Other recordings show different scenarios such as potholes, sharp turns, and sudden stops. In addition to these ADBs, there are also a couple examples of what the sensor readings look like during a simulated crash incident.

One important thing to note about the ADB graphs is that the normal riding activity graphs vary in the amount of data they are displaying. For example, some may display a short ride of only two to three minutes, while others are seven minutes, eight minutes, or even longer. This is most easily noticeable when compared to the graphs of activities which are expected to cause a spike in the readings. These data were captured specifically to illustrate these scenarios, and were recorded with much shorter durations. The remainder of this paper is organized as follows: Chapter II will discuss the methodologies used throughout the different phases of the project, Chapter III will address work of interest that is related to the project, Chapter IV will evaluate the results of the work and any open issues. The paper is concluded with Chapter V, followed by the appendices and references.

CHAPTER II

METHODOLOGY

In this chapter, I will be discussing the different methodologies used in the planning, development, and evaluation phases of the project. The primary method of demonstrating the functionality and capabilities developed for this project will be the Urban Bike Buddy iPhone application. While this is a fully functioning iPhone application, it is not available on the App Store. It has only been made available to a small, controlled group of testers through Apple's TestFlight program. These testers were all volunteers, and their feedback was crucial in the development of Urban Bike Buddy. The chapter is laid out as follows: I begin by introducing the voice assistant used in the project. Next, I will discuss in detail the development environment and development process used. That section will be followed by describing the different hardware that was used for the project. Finally, I will conclude the chapter with a discussion of all the different cloud-based technologies that are used in the project.

The voice assistant that I have chosen to use for this project is Amazon's Alexa. This was a difficult decision, since the application is being developed for an iPhone, where the natural choice is Siri. However, Alexa is currently available on more devices, and the Amazon Developer Services have been very pleasant to work with. I hope to use the voice capabilities of Alexa to allow riders to report accidents, near-misses, and other roadside hazards. This data will then be analyzed and coalesced with other reporting methods to give the city a better picture of bicycle safety. At the time of this writing, the newest, publicly available, crash statistics for the state of Oregon are for the calendar year 2015 [1]. This data was released on May 20, 2017. According to statistics that were made available, there were 960 crashes reported that involved bicycles, and a total of 965 bicyclists involved in these crashes. Of those 965 bicyclists, 8 were fatally injured.

Bicyclist behaviors that were the most frequently reported contributors were 1) failure to yield right-of-way, 2) disregarding traffic signal, and 3) riding on pavement facing traffic. Some of these behaviors can stem from waiting too long at a traffic signal with no indication that you have been recognized. This is one area in particular that I hope to address with the signal triggering capabilities of Urban Bike Buddy. If more riders can be encouraged to practice lawful, safe riding habits, then it may help reduce the number of crashes caused by bicyclists themselves. Reducing

[1] https://www.oregon.gov/ODOT/Data/Documents/QuickFacts_2015.pdf

the number of crashes should generally improve bicycle safety, and if more people can perceive riding a bicycle as a safe alternative to driving, it may further increase bicycle ridership in Eugene and elsewhere.

Development Environment

The Urban Bike Buddy application discussed in this paper was written in Swift for iOS. The application targets a minimum iOS version of 10.3 and supports running on iPhone, iPad, or Mac. However, I will only discuss what is relevant to developing and running the application for the iPhone, as the intention is to have the phone mounted on the handle bars of a bicycle, and all algorithms discussed were developed with this in mind.

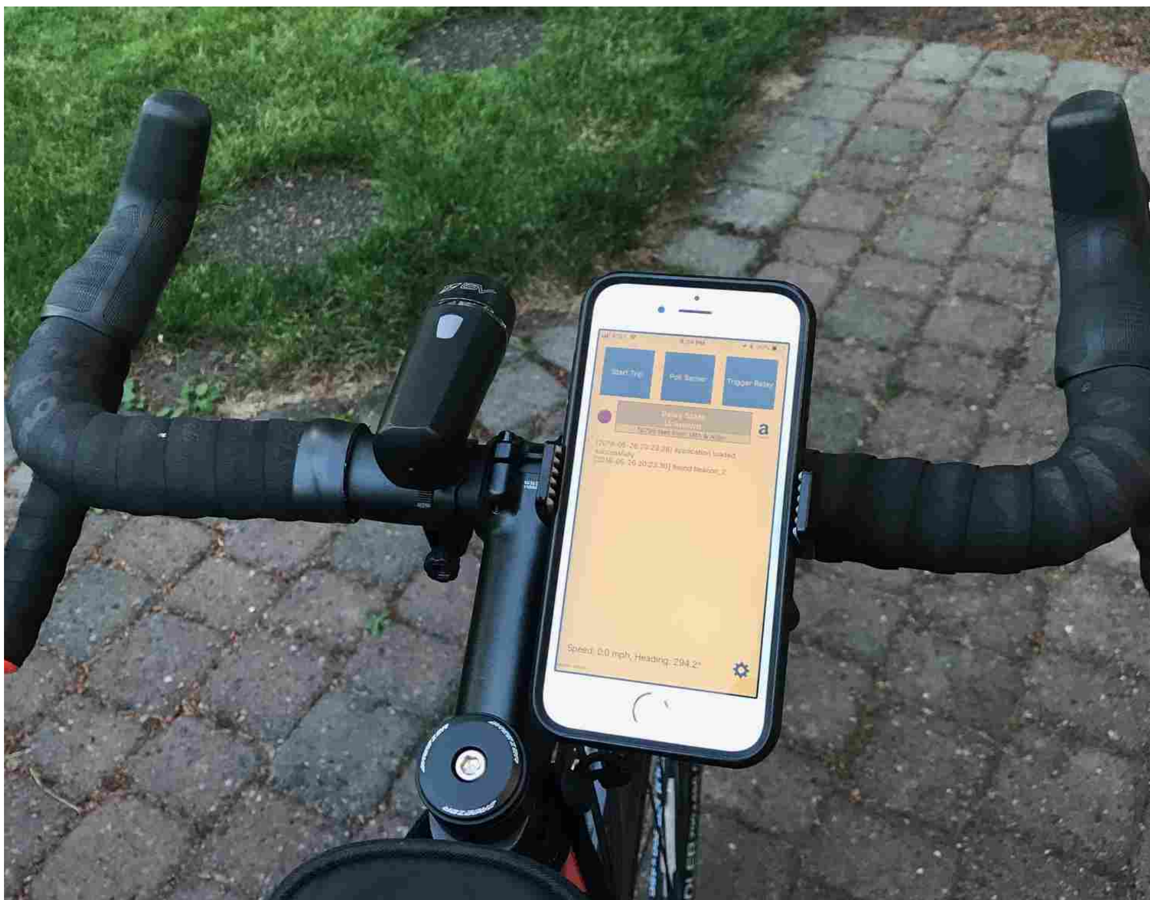
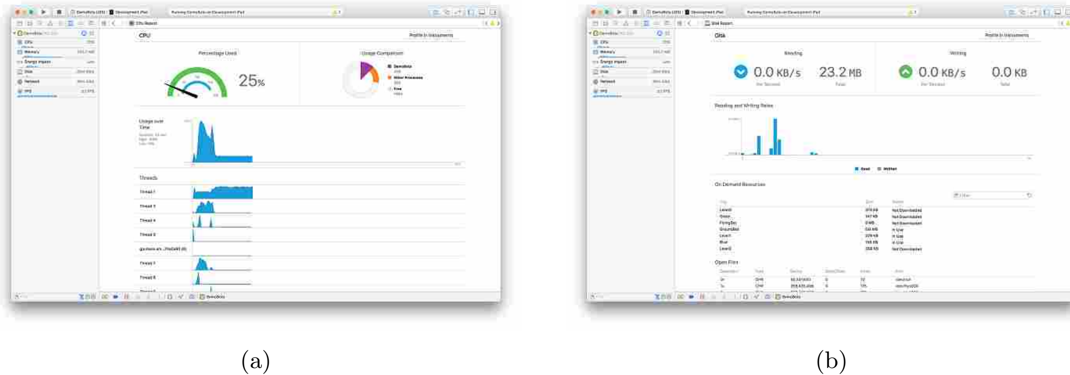


Figure 2.1 iPhone running Urban Bike Buddy mounted on handlebars

Xcode. Apple's Xcode 9 was used as the integrated development environment (IDE) for this project. Apple provides a full software suite for developing iOS applications, which includes the code editor, compiler, and a simulator to test the code locally before deploying it to a

device. In addition, Xcode also comes with a powerful debugger. When running the application on a device attached to the development machine, the debugger can display CPU usage, memory consumption, disk usage, and network usage to allow detailed performance analysis. If this information is insufficient, there is also a profiling tool provided with Xcode called Instruments (*Measure Energy Impact with Xcode*, 2016).

Rather than calculating distances using the law of cosines on the coordinates of two points as in Huang and Chang (2015), I chose to use the `CLLocation.distance(from:)` method that is provided by the Apple Core Location framework. This method calculates the orthodromic, or great-circle distance, by tracing a line between two points following the curvature of the Earth and measuring the length of the resulting arc.



(a) (b)

Figure 2.2 a shows a sample CPU gauge and b shows a sample disk gauge.

Continuous Integration. I used the Travis CI service for continuous integration. This allowed me to ensure that every new commit that I pushed to the remote repository would build successfully. Any issues that arose during a new build were able to be corrected before submitting the build to Apple for approval. In addition, this provided a second layer of testing outside of the primary development environment. One issue that I have encountered often as a developer is that something will work great on the development machine, which typically has many additional libraries installed, but when publishing it to the end-user, the application may be buggy, or not work at all. The use of continuous integration made it very easy for me to catch bugs, or other issues, before the build was submitted to Apple. It is very easy for a developer to accidentally forget to add a new file to the version control system when making a new commit, so tools such as this will catch the issue long before the code is submitted for a final build.

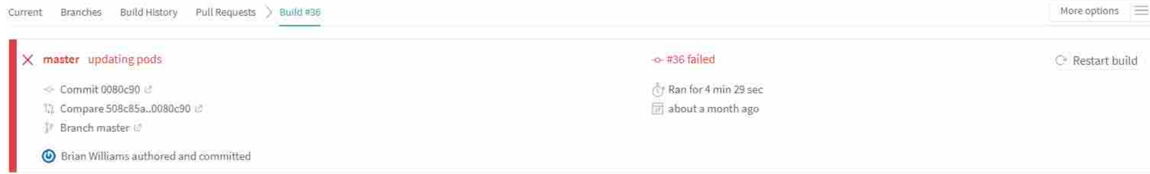


Figure 2.3 Travis CI console showing failed build

Figure 2.3 shows the Travis CI console displaying a failed build. Since the build process is initiated automatically when pushing a new commit to the GitHub repository, it is very quick to catch certain types of issues. Figure 2.4 shows the build output log that is also displayed in the console. Although it looks like a lot of information to digest, this does help developers track down build issues. In this case, a new file `finish.wav` was added to the project, but the project settings in Xcode were not set up correctly to include that file as a resource with the project.

```
5199 CpResource /Users/travis/build/Downloads/finish.wav /Users/travis/Library/Developer/Xcode/DerivedData/Traffic_Buddy-duhsvdqkrqgkkievuawfdejqtlpj/Build/Products/Debug-iphonesimulator/Traffic\
     Buddy.app/finish.wav
5200     cd /Users/travis/build/dubstylee/Traffic-Buddy
5201     export PATH="/Applications/Xcode-9.2.app/Contents/Developer/Platforms/iPhoneSimulator.platform/Developer/usr/bin:/Applications/Xcode-
     9.2.app/Contents/Developer/usr/bin:/Users/travis/.rvm/gems/ruby-2.4.2/bin:/Users/travis/.rvm/gems/ruby-2.4.2@global/bin:/Users/travis/.rvm/rubies/ruby-
     2.4.2/bin:/Users/travis/.rvm/bin:/Users/travis/bin:/Users/travis/.local/bin:/Users/travis/.nvm/versions/node/v6.11.5/bin:/usr/local/bin:/usr/bin:/bin:/usr/sbin:/sbin:/opt/X11/bin"
5202     builtin-copy -exclude .DS_Store -exclude CVS -exclude .svn -exclude .git -exclude .hg -resolve-src-symlinks /Users/travis/build/Downloads/finish.wav
     /Users/travis/Library/Developer/Xcode/DerivedData/Traffic_Buddy-duhsvdqkrqgkkievuawfdejqtlpj/Build/Products/Debug-iphonesimulator/Traffic\ Buddy.app
5203 error: /Users/travis/build/Downloads/finish.wav: No such file or directory
5204
5205 CompileStoryboard Traffic\ Buddy/Views/Base.lproj/LaunchScreen.storyboard
5206     cd /Users/travis/build/dubstylee/Traffic-Buddy
5207     export PATH="/Applications/Xcode-9.2.app/Contents/Developer/Platforms/iPhoneSimulator.platform/Developer/usr/bin:/Applications/Xcode-
     9.2.app/Contents/Developer/usr/bin:/Users/travis/.rvm/gems/ruby-2.4.2/bin:/Users/travis/.rvm/gems/ruby-2.4.2@global/bin:/Users/travis/.rvm/rubies/ruby-
     2.4.2/bin:/Users/travis/.rvm/bin:/Users/travis/bin:/Users/travis/.local/bin:/Users/travis/.nvm/versions/node/v6.11.5/bin:/usr/local/bin:/usr/bin:/bin:/usr/sbin:/sbin:/opt/X11/bin"
5208     export XCODE_DEVELOPER_USR_PATH=/Applications/Xcode-9.2.app/Contents/Developer/usr/bin/..
5209     /Applications/Xcode-9.2.app/Contents/Developer/usr/bin/ibtool --errors --warnings --notices --module Traffic_Buddy --output-partial-info-plist
     /Users/travis/Library/Developer/Xcode/DerivedData/Traffic_Buddy-duhsvdqkrqgkkievuawfdejqtlpj/Build/Intermediates.noindex/Traffic\ Buddy.build/Debug-iphonesimulator/Traffic\ Buddy.build/LaunchScreen-
     SBPartialInfo.plist --auto-activate-custom-fonts --target-device iphone --target-device ipad --minimum-deployment-target 10.3 --output-format human-readable-text --compilation-directory
     /Users/travis/Library/Developer/Xcode/DerivedData/Traffic_Buddy-duhsvdqkrqgkkievuawfdejqtlpj/Build/Intermediates.noindex/Traffic\ Buddy.build/Debug-iphonesimulator/Traffic\ Buddy.build/Base.lproj
     /Users/travis/build/dubstylee/Traffic-Buddy/Traffic\ Buddy/Views/Base.lproj/LaunchScreen.storyboard
5210
5211 ** BUILD FAILED **
5212
5213
5214 The following build commands failed:
5215         CpResource /Users/travis/build/Downloads/finish.wav /Users/travis/Library/Developer/Xcode/DerivedData/Traffic_Buddy-duhsvdqkrqgkkievuawfdejqtlpj/Build/Products/Debug-iphonesimulator/Traffic\
     Buddy.app/finish.wav
5216 (1 failure)
5217
5218
5219 The command "xcodebuild clean && xcodebuild build -workspace "Traffic Buddy.xcworkspace" -scheme "Traffic Buddy" -allowProvisioningUpdates -sdk iphonesimulator ONLY_ACTIVE_ARCH=NO" exited with 65.
5220
5221 Done. Your build exited with 1.
```

Figure 2.4 Travis CI console showing build output log

Once a build failure occurs, Travis CI e-mails the developer to notify them of the issue. Finding issues such as these early in the process is a great time saver, because otherwise the issue may not be detected until the app is submitted to Apple for review. After a new commit has been pushed to the repository, Travis CI will notify the developer again to indicate whether or not the issue was fixed, or if the build is still failing. If the previous build was passing, the developer will not be notified of successful builds by default. Figure 2.5 shows the Travis CI console after fixing the previous issue, indicating a passing build.

Figure 2.5 Travis CI console showing fixed build

Algorithms. Although the proposed algorithm is derived from fall detection and crash detection algorithms, these solutions do not apply specifically to a bicycle rider and need to be adapted to fit the context. These algorithms are either designed specifically to detect falls in elderly people (Chen, Kwong, Chang, Luk, & Bajcsy, 2006), or to detect a vehicle to vehicle crash. Also, some of these algorithms depend on additional information, such as the speed and direction of an oncoming vehicle (Kim & Jeong, 2014). To keep my solution simple, only known sensor data from the user is considered. A typical fall detection system has two components: the detection component, and the communication component. I will focus primarily on the detection component in this section.

In order to determine an optimal threshold for triggering the crash detection algorithm, data has been collected from normal everyday bicycle trips by the volunteer testers. Because we only have a small number of people who have access to the app, the data set is limited, but over twenty hours of sensor data was provided. These data were submitted in the form of a CSV file, e-mailed directly from within the app at the end of a bicycle trip. A code listing has been provided in Appendix B to illustrate how the data was translated from the CSV files into line graphs for visualization. Table 2.1 shows a small sample from one of the submitted CSV files.

Date	Type	val
2018-05-23 08:49:38.4690	A_t	0.0302967310975201
2018-05-23 08:49:38.4690	A_v	0.00101790216787182
2018-05-23 08:49:38.6500	A_t	0.038208967233578
2018-05-23 08:49:38.6500	A_v	0.0221486111309621
2018-05-23 08:49:38.8430	A_t	0.0797636889019096
2018-05-23 08:49:38.8430	A_v	0.062172904797057
2018-05-23 08:49:39.0240	A_t	0.280868485956767
2018-05-23 08:49:39.0240	A_v	0.122206689062159

Table 2.1 CSV data sample

Accelerometer-based algorithms such as this have been used often in fall detection, and their effectiveness has been evaluated in real-world fall scenarios. Bourke, Obrien, and Lyons

(2007) point out that many algorithms are developed using young, healthy subjects, which may not be very effective in detecting a fall in an elderly subject. They go on to describe their own algorithm that captures ADL data from elderly subjects, in addition to having falls simulated by young, healthy subjects.

Another study by Bagalà et al. (2012) evaluated thirteen different accelerometer-based algorithms on available real-world fall data. They also point out that the data that is primarily used to develop the algorithms is often not the same type of data that is being evaluated. For example, it is difficult to use simulated fall or crash data to predict a real-world fall or crash. This is an ongoing difficulty in many areas, especially where a real-world crash or fall could be harmful, so they are not easy to measure in a testing environment. However, since some of the real crash data is being made available, it may be possible to develop an algorithm using crash data that has already been collected.

The paper by Habib et al. (2014) surveys the current state of smartphone-based solutions for fall detection and prevention, many of which use some form of threshold monitoring to detect a fall or imminent fall. Many different algorithms are discussed, but most smartphone-based solutions seem to used a Threshold-Based Algorithm (TBA). There are a few algorithms of special interest which will be discussed below.

At the time of this writing, the only bicycle-specific research that I was able to discover was that done by Candefjord et al. (2014) at Chalmers University of Technology in Sweden. Their work is being used in a commercial product, so I was unable to learn any additional information about their proprietary algorithm. However, similar work has been pursued in relation to all-terrain vehicles (ATVs) by others at Chalmers University of Technology. Matuszczyk and Åberg (2016) worked on this as a Master's thesis.

Basic Threshold Monitoring. Threshold-Based Algorithms typically use readings from a tri-axial accelerometer to calculate the signal magnitude vector, or SMV.

$$SMV_i = \sqrt{|A_{x_i}|^2 + |A_{y_i}|^2 + |A_{z_i}|^2} \tag{2.1}$$

The values A_x, A_y, and A_z represent the accelerometer reading of each axis of the accelerometer. SMV_i represents the value calculated at time i. Another formula that is used in some fall detection algorithms is the amplitude vector in relation to the absolute vertical direction, which can be

calculated as:

$$A_v = |A_x \sin\theta_z + A_y \sin\theta_y - A_z \cos\theta_y \cos\theta_z| \tag{2.2}$$

The combination of these two vectors is primarily what is used in the crash detection algorithm of Urban Bike Buddy. If both calculated values exceed a certain threshold, then the crash detection process is initiated, and the user is prompted if they would like to report a crash.

Fall Index. Because of the issues that arise with simply detecting a sudden change in the vector calculations to signify a fall, Yoshida et al. (2005) propose an additional value known as the Fall Index.

$$FI_i = \sqrt{\sum_{k=x,y,z} \sum_{i-19}^{i} ((A_k)_i - (A_k)_{i-1})^2} \tag{2.3}$$

This value can help in some situations, but due to the high sampling frequency and fast acceleration changes, it will miss falls that happen slowly. In fact, the performance can actually decrease in some specific situations, such as a forward fall when using only a smartphone to detect the fall (Casilari & Oviedo-Jiménez, 2015).

Two-Phase Detection (PerFallD). Dai, Bai, Yang, Shen, and Xuan (2010) propose PerFallD, a pervasive fall detection system implemented on mobile smartphones. The algorithm they introduce is an expansion of the basic threshold monitoring. Rather than rely solely on the signal magnitude vector, they use Formula 2.2 described above. If the difference in one of the magnitude calculations exceeds a certain threshold $Th_t t$ within a triggering time window $win_t t$, the difference between the minimum and maximum magnitude vector values during a checking time window $win_c t$. If the difference is less than yet another threshold value $Th_c t$, then a fall is considered detected. Urban Bike Buddy implements the two-phase detection portion of this algorithm, but the trigger time and check time windows are not implemented currently. Adding this functionality in the future could increase the accuracy of the Urban Bike Buddy crash detection algorithm.

iFall. Another popular threshold-based algorithm is iFall(Sposaro & Tyson, 2009). The detection aspect is similar to other basic threshold monitoring algorithms. What makes their solution different is that they have added a short timer to allow the user to return to their previous

position after a fall is detected in order to reduce the number of false positives. They reduce this number even further by giving the user a short time window to cancel the fall detection notification step. If the user does not cancel the notification within the allotted time, the contacts that were setup to receive notification are sent a message. Emergency medical services are only notified if the emergency contact is unable to determine whether a fall occurred or not. The combination of all of these aspects makes their solution attractive because it results in fewer false alarms reported to emergency services. Even if the algorithm does falsely detect a fall, there are multiple opportunities to interrupt the process to avoid a false alarm from reaching the emergency services. While Urban Bike Buddy does not automatically notify emergency contacts, I did still implement the timer aspect to allow the user to reduce the number of false positives that are detected by the algorithm.

In addition to the fall or crash detection, there is also the question of how to respond to a detected incident. One way that some have chosen to respond to an incident is to share the GPS location with a pre-defined contact. For example, when a vehicle crash is detected, Sulochana and Manohar Babu (2014) will send the GPS location of the crash over the cellular GSM network to allow a more prompt response by either the emergency contact or emergency medical services.

Hardware

I will discuss the different types of hardware used in the project in this section. Urban Bike Buddy is really just a user interface to interact with the different hardware being used. The app allows the user to interact with the smartphone, which then communicates via the Global System for Mobile Communication (GSM) with the Particle Electron and other cloud services.

Particle Electron. Particle's cellular IoT platform includes prototyping hardware, mass production modules, SIM cards, a global cellular network for connectivity in 100+ countries, and cloud services required to scale your product from first prototype to mass production. All Electron family products can be setup in minutes using Particles mobile app or browser-based setup tools. Guided tutorials, example projects, and community-contributed firmware libraries make it easy to build your IoT application at any skill level. Easily troubleshoot bugs and issues with the help of free email support, professional development services, and our friendly online community of customers (*Particle — Electron (2G/3G/LTE) cellular hardware*, 2018).

Smartphone. The smartphone that will be discussed in this section will be the iPhone 7 (Model A1778) that was the primary device used during the testing phase. All iPhone models since the iPhone 4 come with an integrated GPS receiver. The model A1778 used for testing phone has the following sensors: touch ID fingerprint sensor, barometer, three-axis gyroscope, accelerometer, proximity sensor, and ambient light sensor. Although it could be interesting to incorporate other sensors in the future, I only consider the accelerometer and three-axis gyroscope, as well as the assisted GPS receiver. The iPhone 7 is powered by the Apple A10 Fusion system on a chip (SoC). The A10 has four CPU cores with a maximum CPU clock of 2.34 Ghz. The A10 is the first Apple-designed quad-core SoC, with two high-performance cores and two energy-efficient cores. Also embedded into the A10 is the M10 motion coprocessor, which services the accelerometer, gyroscope, compass, and barometer. In addition, the iPhone 7 has 2GB of LPDDR4 RAM, a 12 megapixel camera, a 7-megapixel front-facing camera, and a 4.7-inch Retina HD LED-backlit widescreen that has a resolution of 1,334 x 750 pixels at 326 ppi. Without any accessories, the phone measures in at 138.3 x 67.1 x 7.1mm and weighs 138 grams.

GPS. GPS is used for various functions within Urban Bike Buddy: to include the user's location when reporting incidents, as well as to determine the distance of the user from the nearest "important" intersection. At the time of this writing, only one intersection has been connected to the Particle cloud that we are able to interact with through the application.

The iPhone 7 is equipped with a Broadcom BCM47734 chip. In addition to GPS, this chip also supports Galileo, GLONASS, QZSS, and other global positioning systems. The iPhone uses Assisted GPS (AGPS), so that when the phone is unable to get a strong satellite signal, it will attempt to determine the user's position by communicating with cellular towers and nearby wireless networks. Because of this, the phone is able to provide a location when a standalone GPS may be unable to, but this can also lead to inaccurate readings. In an effort to preserve battery life, iOS provides multiple accuracy options [2]: kCLLocationAccuracyThreeKilometers, kCLLocationAccuracyKilometer, kCLLocationAccuracyHundredMeters, kCLLocationAccuracyNearestTenMeters, kCLLocationAccuracyBest, kCLLocationAccuracyBestForNavigation. Urban Bike Buddy uses kCLLocationAccuracyBest, as this setting combines the GPS readings with other sensor data to improve the accuracy. This does decrease the battery life, however, as will be discussed in the

[2] `https://developer.apple.com/documentation/corelocation/cllocationaccuracy`

Open Issues section. In order to help developers address this issue, Apple has provided a few extra options to their Core Location framework to give the ability to turn off location updates when the application is in the background, or when the user is not moving. Listing 2.1 shows the location services settings used by Urban Bike Buddy.

```
1 if CLLocationManager.locationServicesEnabled() {
2     locationManager.activityType = CLActivityType.fitness
3     locationManager.allowsBackgroundLocationUpdates =
          true
4     locationManager.desiredAccuracy =
          kCLLocationAccuracyBest
5     locationManager.distanceFilter = 5
6     locationManager.pausesLocationUpdatesAutomatically =
          true
7
8     locationManager.startUpdatingLocation()
9 }
```

Listing 2.1 Configuring Location Services

As shown in Listing 2.1, `pausesLocationUpdatesAutomatically` is set to `true` to allow the location manager to automatically pause when it determines the user is not moving according to the specified `activityType`. In this case, bicycling is included in the fitness activity type.

One important thing to note, however, is that when the location manager pauses location updates, they are not resumed automatically, and your application must implement the functionality to resume location updates when appropriate. In the case of Urban Bike Buddy, I ultimately disabled automatic pausing in order to easily allow triggering the relay when approaching the intersection with the app running in the background. If the user is not on a ride, however, automatic pausing of location services is allowed, as the location is only needed to determine the distance from the nearest intersection that has been integrated into our system. If the location services are paused while the user is not on a trip and the app is running in the background, location services will resume once the user reopens the app. This allows the distance displayed in the app to reflect the user's current location while they have the app running in the foreground.

Accelerometer & Gyroscope. Both the accelerometer and gyroscope are on the same 6-axis MEMS chip by InvenSense. [3] At the time of this writing, I am not able to discover with certainty which chip that is used in the iPhone 7, but my best educated guess would be the ICM-20690 chip. [4] This is InvenSense's chip that is designed to be used in smartphones and tablets, so it seems to be a reasonable guess. Rather than using the raw sensor data provided by the accelerometer and gyroscope readings, I chose to use the device motion data from Apple's `CMMotionManager`. This data takes into account the readings from the accelerometer, gyroscope, and magnetometer and applies Core Motion's sensor fusion algorithms. These algorithms provide more useful data by filtering outliers. It is possible to get even more accurate data by applying additional filtering, but the readings from the device motion data were sufficient for my purposes.

Cloud Services

Choosing a provider for cloud computing services can be a daunting task. There are many different providers, including some of the big names in the industry such as Amazon, Microsoft, and Google. In order to help determine which provider is best for a certain situation, Garg, Versteeg, and Buyya (2013) have developed a framework for ranking cloud computing services. They propose the Service Measurement Index Cloud framework, or SMICloud, which helps customers find the most suitable provider for their needs. SMICloud provides service selection based on quality of service requirements and ranking of services based on previous user experiences and service performance. Their framework can assist customers when searching for a new provider of cloud services. Some of the factors discussed in their work were considered while choosing providers for Urban Bike Buddy, but one of the most important determining factors in my specific case was convenience. Because I already had an existing Amazon Web Services account, Microsoft Azure account, and Google Cloud Services account, I limited my selections to those providers. Ultimately, I did not end up using any Microsoft services in this project, but they were considered in the planning phase of my research.

Amazon Alexa. The voice assistant of choice for this project was Amazon Alexa. By now, most people have at least heard of Alexa. Over 20 million Echo devices have been sold, in addition to the many third-party devices which are also compatible with Amazon's cloud-based

[3] `https://www.electronicproducts.com/iPhone_7-whatsinside-198.aspx`

[4] `https://www.invensense.com/products/motion-tracking/6-axis/icm-20690`

Alexa service. The reason I chose Alexa was not just the popularity, but also the maturity of the platform. Amazon has been leading the way in the area of consumer home voice products with the first Amazon Echo being released in November 2014. The suite of products available from Amazon Web Services makes it a very easy entry into development for Alexa skills.

Alexa Voice Services. Alexa Voice Services allows a developer to add Alexa functionality to their application or device. This means that Alexa is not limited to Amazon devices such as the Echo. We are starting to see many manufacturers release smart home speakers and other smart devices that come with Alexa capabilities built-in. AVS can be used to add these capabilities. The Alexa Voice Services SDK is simply a means of communicating with the Alexa service. It is completely separate from the actual Alexa skill that controls the user interaction. The skill developed for Urban Bike Buddy went through a few different iterations as different challenges were encountered and goals were changed along the way. For the most part, the interaction with Alexa in the app is limited, and is primarily used to illustrate a proof of concept.

Login With Amazon. In order to allow integration with Amazon's Alexa voice services, it was necessary to implement some sort of authentication mechanism. For this purpose, I chose Amazon's own Login With Amazon SDK. This simplified the process by allowing users to login with their existing Amazon account, and also made it easy to differentiate between users. However, in the future I would like to perhaps implement an integrated system into the application that will automatically authenticate with Amazon, and simply differentiate between users by device ID, removing the need for users to provide their own login credentials to the application.

Amazon Cognito. The Login With Amazon SDK integrates with Amazon Cognito. Amazon Cognito lets you add user sign-up, sign-in, and access control to your web and mobile apps quickly and easily. Amazon Cognito scales to millions of users and supports sign-in with social identity providers, such as Facebook, Google, and Amazon, and enterprise identity providers via SAML 2.0 (*Amazon Cognito - Simple and Secure User Sign Up & Sign In*, 2018).

AWS Lambda. Many of Amazon Alexa's skills are powered by AWS Lambda. While this is not strictly a requirement, Amazon does make it very easy to host the backend for an Alexa skill on their Lambda service. Although it is a very enticing platform for Alexa skills, AWS Lambda is not limited to powering Amazon services. On the contrary, with Lambda, you can run code for virtually any type of application or backend service - all with zero administration. Just

upload your code and Lambda takes care of everything required to run and scale your code with high availability. You can set up your code to automatically trigger from other AWS services or call it directly from any web or mobile app (*AWS Lambda - Serverless Compute*, 2018).

Amazon DynamoDB. DynamoDB is Amazon's NoSQL cloud database service. It can be used as a standalone database, but in the case of Urban Bike Buddy, it is only used by the Alexa skill which handles the crash reporting procedure. The skill is configured to maintain the current state in a DynamoDB table. Configuration is as simple as adding the `dynamoDBTableName` property to the Alexa handler in Lambda. Listing 2.2 shows the Lambda code used to configure the Alexa skill.

```
1 exports.handler = function (event, context) {
2     const alexa = Alexa.handler(event, context);
3     alexa.appId = APP_ID;
4     alexa.dynamoDBTableName = 'cycle-buddy';
5     alexa.registerHandlers(stateHandlers);
6     alexa.execute();
7 };
```

Listing 2.2 Alexa skill configuration

Google Pub/Sub. Throughout the development process, I used Google's Pub/Sub messaging service in order to maintain logging data related to the communication between the Urban Bike Buddy application and the Particle Electron device. This was achieved by adding integrations from within the Particle Console. Each published Particle event can be configured to also publish the event to Google's Pub/Sub service. Once this was complete, I developed a Python script that would periodically fetch any messages from the topic for future analysis. By handling the logging in this manner, it allowed me to separate that functionality from the application itself, reducing the cellular data consumption. In addition, publishing the messages to an external service also reduces the amount of data transmitted by the Particle Electron, which is running on a cellular data plan as well.

Figure 2.6 shows the flow of data between the different components of a publish/subscribe model. In Urban Bike Buddy, the Particle Electron device is the publisher of data. When a trigger request is received from the app, the Particle Electron will toggle a relay that is connected

to the GPIO of the device. When this request is received, the Particle Electron is also configured to publish a message to the Google Pub/Sub topic that was created for this project. As shown in the diagram, the topic is named "particle-logs." Messages that are published to this topic will be received by anyone who is subscribed to the topic. In this case, the subscription name is also "particle-logs." I have created a simple Python script that is responsible for the retrieval of the log messages from the subscription. If the script were running constantly, it would receive the messages as they are sent by the publisher; however, because the amount of data being sent is minimal, I have just been running the script every few days to fetch the new logs since the last time it was run. The source code for the Python script can be found in Listing B.1.

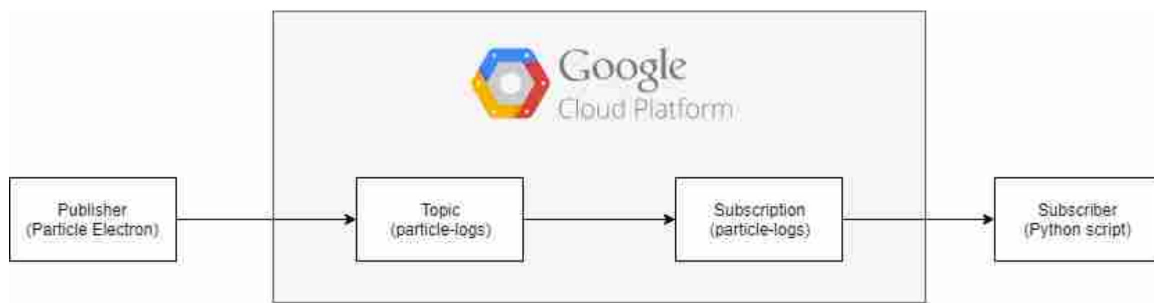


Figure 2.6 Diagram of publish/subscribe model

Although I chose Google's Pub/Sub service for this project due to the integration with the Particle Cloud Console, Amazon has their own pub/sub messaging service as well. Their service, Amazon SNS (Simple Notification Service), also includes mobile push notifications, easily allowing you to send push notifications to iOS, Android, Fire OS, Windows and Baidu-based devices (*Amazon Simple Notification Service (SNS)*, 2018). I may switch to SNS in the future for ease of use, to consolidate all of the services onto one platform.

Natural Language Processing

There are many Voice Assistant services available today. Some of the most well-known examples are Apple's Siri, Microsoft's Cortana, and of course Amazon's Alexa. These services are powered by powerful machine learning and artificial intelligence algorithms. In addition to the standard user interactions with Alexa, I chose to use a cloud-based transcription service to convert the user-submitted crash reports to text, allowing for easier analysis. Speech-to-text software has existed for many years; Nuance Communications' Dragon NaturallySpeaking was initially released in June of 2017, over twenty years ago. However, Dr. James Baker laid out the description of a

speech understanding system called DRAGON in 1975 (Baker, 1975). Throughout the development process of Urban Bike Buddy, I used cloud-based speech-to-text services from both Google and Amazon, as I will discuss in more detail below.

Google Cloud Speech. When I originally began development on Urban Bike Buddy, the main widely-available transcription service at the time was Google Cloud Speech. I added real-time transcription functionality to my app based on one of the sample projects available from Google. This would have been fine for the incident reporting, but during development, Amazon released a beta version of their own transcription service, Amazon Transcribe. Although Amazon Transcribe does not allow for easy real-time transcription, that was not a necessity of the project. For anyone looking into real-time, cloud-based streaming transcription, Google Cloud Speech is certainly a great option for this. However, I ultimately ended up changing services once I was accepted to be an early beta tester of Amazon Transcribe.

Amazon Transcribe. When I learned about the beta release of Amazon Transcribe, I quickly applied to become a tester. After a few weeks of not hearing anything, I thought that my application was going to be denied, or simply ignored altogether. However, I was eventually accepted to join the early beta program. Because the development of my application to this point was based on using a real-time transcription service to provide a textual representation of recorded user feedback, I had to make some changes to the flow of the incident reporting process.

Amazon Transcribe allows you to submit an audio recording to their API, and after processing the file, you will receive a text transcription back. One very appealing aspect of this service is the easy of which it is implemented by using another Amazon service, Amazon S3. In the case of Urban Bike Buddy, the incident report is recorded by a user on their device, and then submitted for upload to an S3 bucket containing the reports waiting to be transcribed. There is a limitation on the free usage of Amazon Transcribe, so to this point I have chosen not to automate the transcription process. I currently start a transcription job manually through the Developer console. At the time of this writing, the Amazon Transcribe service can transcribe audio files in FLAC, MP3, MP4, or WAV format, and is limited to a maximum of two hours in length. For best results, Amazon recommends using a lossless format, such as FLAC or WAV, with PCM 16-bit encoding. As shown in Figure 2.7, you must specify the audio format, as well as the language of the input file.

Figure 2.7 Creating a new Amazon Transcribe job

It is possible to have Amazon Transcribe identify the different speakers in an audio clip, a process known as diarization or speaker identification. When you enable speaker identification, Amazon Transcribe labels each fragment with the speaker that it identified. You can specify that Amazon Transcribe identify between 2 and 10 speakers in the audio clip. You get the best performance when the number of speakers that you ask to identify matches the number of speakers in the input audio.

Push Notifications vs. Publish/Subscribe

Two of the main messaging techniques in common use today are push notifications and publish/subscribe messaging. Many people are familiar with push notifications, as this is a common technique used by a smartphone to send an important piece of information to the end user. For example, if you have push notifications turned on for your News app, you may get a notice when an important story is breaking. The way that publish/subscribe works is a bit different. An app, or even an individual, can publish a message to a topic, and anyone who is subscribed to that topic will then receive the message, or can fetch the messages for a subscribed topic at a later time. As mentioned previously, I chose to use the publish/subscribe method in this project. One of the main factors in making this decision was that the Particle Cloud has Google Cloud Plat-

form services built into their developer console. This made it very easy to add Google Pub/Sub integration into my project.

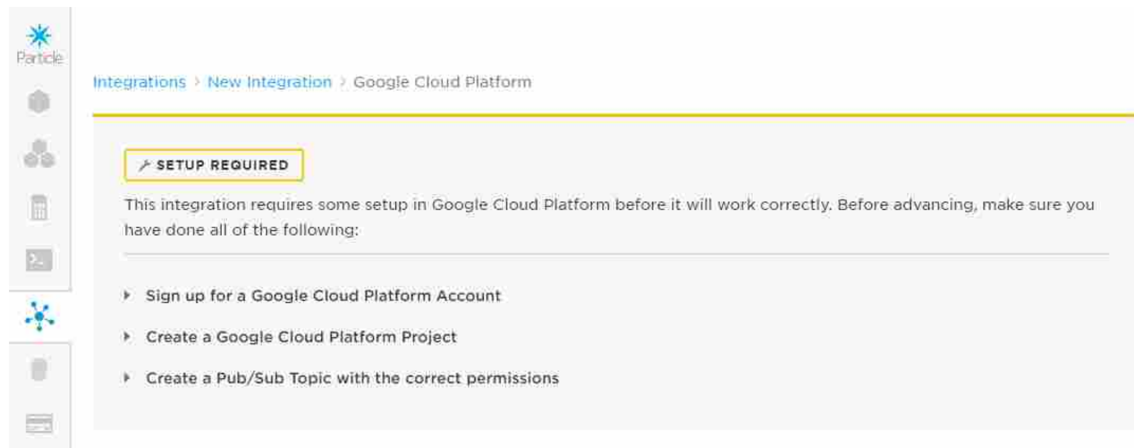


Figure 2.8 The Particle console Google Cloud Platform integration.

In the future, it would be nice to implement push notifications in the app, which would allow the user to be notified when a trigger is sent to the Particle Electron while the app is running in the background. Currently, the user can only see the trigger occur if they have the app open in the foreground; in addition to displaying when the user is within the range threshold to send a trigger, it will also show a text display that indicates when a trigger is sent.

CHAPTER III

RELATED WORK

Much work has been done in regards to accident detection and fall detection. The primary goal of this project is not to re-create the wheel, but to use the existing research and related algorithms for a new purpose. In order to develop an algorithm that would be useful in detecting near-misses and accidents on a bicycle, I started with some algorithms that are currently used in detecting falls as well as automobile accidents. I first evaluated how well these methods would fit with what I was trying to achieve in my project. Because a near-miss is less serious than an automobile accident, I assume that it would tend to be more similar to the fall detection algorithm, and a bicycle accident would tend to be more similar to an automobile accident than a fall, depending on the seriousness of the accident. One important consideration was the frequency at which to analyze the sensor data. A higher frequency provides increased accuracy, but can also result in a drain on the battery. Different frequencies were evaluated, ranging from 10 times per second (10 Hz) to every 5 seconds (0.2 Hz).

Most of the work related to crash detection algorithms is focused on improving response times to help reduce the response time by first responders. WreckWatch (White, Thompson, Turner, Dougherty, & Schmidt, 2011) is an automatic traffic crash detection and notification system for smartphones. Their system can automatically detect automobile crashes using a combination of accelerometer and acoustic data. Once a crash is detected, emergency services are immediately notified with situational information including the GPS coordinates, photographs, and the sensor data. Because their system is automatically notifying emergency services, it is crucial to minimize the number of false positive detections to avoid wasting valuable resources and the time of the first responders. While it is important to provide prompt medical care to vehicle crash victims, the work of my research focuses on accidents and near-misses involving a bicycle. The statistics related to bicycle accidents may not be very accurate, as it is estimated that only x% of incidents are reported. My goal is to improve the process of reporting incidents for bicycle riders, as well as to help identify problematic areas so that the city can address locations that have a high rate of incidents.

Once an incident has been detected, the goal is to initiate communication with a smart assistant; in our case, Amazon Alexa. Similar to other crash reporting methods, I display a short

timer to allow the user an opportunity to cancel the report. The cancellation is logged, and will be used in the future to further train the algorithm. If the user does not cancel within the specified time frame, then Alexa will be pinged using a pre-defined utterance. This allows us to programmatically initiate the communication, similar to pushing a button, but without user interaction.

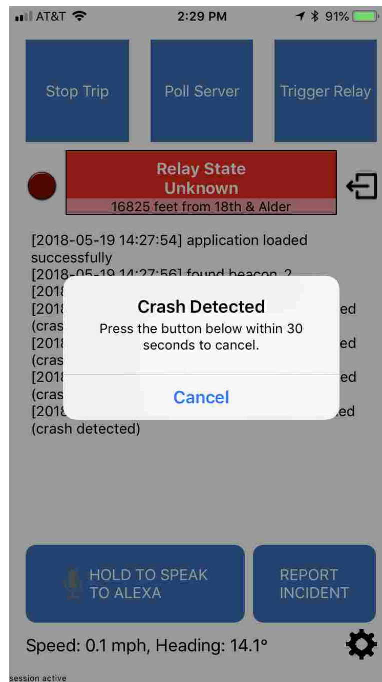


Figure 3.1 Display when Urban Bike Buddy detects a crash

Alexa queries the user to determine if an incident has indeed occurred. There is another opportunity here to cancel the incident report, or choose between a near-miss or crash report. The verbal report is transcribed using the Amazon Transcribe SDK and logged to an online incident reporting system. I hope to also integrate the reports into the existing City of Eugene's reporting system [1]. There is a smartphone application that was developed for their site, but the most recent version of the application is not compatible with newer version of iOS, so my incident reporting solution would provide this capability to users with current model phones. The Android version of their application was not evaluated, as my work was only focused on iPhone.

[1] http://www.webikeeugene.org

Snowboy

There is a project called Snowboy that is used to perform offline wake word detection. Snowboy can be run on many different types of devices, including a Raspberry Pi, Intel Edison, Ubuntu Linux, Mac OS X, iOS, and more. It can also be integrated with Alexa Voice Services, so one could have an offline engine that will be listening for a wake word, and once triggered, can then forward your commands to Alexa.

iBikeEugene

At the time I began working on this project, I came across an app that was developed for the city of Eugene called iBikeEugene. The app was available for both iOS and Android, but the iOS version was not compatible with the OS available on newer iPhones. While meeting with the city, I was told that they were in the process of redeveloping the iBikeEugene app and hoping to release it again in the near future. At the time of this writing, the app is not yet available, but the original version appears to have allowed users to report hazards directly from within the app, including the ability to capture a photo of the hazard and include it in the report. The app did not detect crashes, however, so there is still a need for something like Urban Bike Buddy to enable bicycle riders the ability to report crashes or near-misses, that could also potentially detect when a crash occurs and potentially use the data collected from within the app to assist in future investigation or city improvements to areas that pose safety concerns.

GIS Map

The city of Eugene is currently working on adding their collected crash report data into a map indicating the location where incidents occur. The map is not currently publicly available, so I have not been able to view it, but when I discussed my project with the traffic planners from the city, they told me about the project. It is my hope that either once the project becomes publicly available, or once Urban Bike Buddy has been more widely tested, that I will be able to contribute the GPS locations of reporting crashes and near-misses to the city to include in their map. At this time, however, I do not know if the data that I am collecting as part of the crash report is sufficient to be included in their data.

CHAPTER IV
RESULTS

In this chapter, I will discuss the results of the work, including evaluation of the algorithm used, user feedback, and open issues. Generally, the thresholds used throughout testing resulted in false positives more frequently than I would like. In this case, a false positive result is triggering the crash detection algorithm when a crash did not occur. This mostly happened when something cause a spike in the sensor readings, such as hitting a speed bump or pothole. More work can be done to improve the accuracy of the algorithm, such as adding a delay during which the thresholds must again be reached in order to trigger the crash detection algorithm.

Evaluation

There are multiple different ways to evaluate the effectiveness of an algorithm. In this section, I will discuss some of these measures and how they are calculated. Work on finding the optimal threshold values for the algorithm used is ongoing, so I will not go into too much detail about the calculated values.

Accuracy: The accuracy of a test is its ability to differentiate between crashes and normal behavior correctly. To estimate the accuracy of a test, we should calculate the proportion of true positive and true negative in all evaluated cases. Mathematically, this can be stated as:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{4.1}$$

Sensitivity: The sensitivity of a test is its ability to determine the positive cases correctly. To estimate it, we should calculate the proportion of true positive in positive cases. This can also be viewed as trying to minimize the number of false positives. Mathematically, this can be stated as:

$$Sensitivity = \frac{TP}{TP + FN} \tag{4.2}$$

Specificity: The specificity of a test is its ability to determine the negative cases correctly. To estimate it, we should calculate the proportion of true negative in negative cases. This can also be viewed as trying to minimize the number of false negatives. Mathematically, this can be stated as:

$$Specificity = \frac{TN}{TN + FP} \tag{4.3}$$

		prediction outcome		
		p	n	total
actual value	p'	True Positive	False Negative	P'
	n'	False Positive	True Negative	N'
	total	P	N	

Figure 4.1 An example of a confusion matrix

Ideally, one could find values such that every positive estimate is a true positive (i.e., there are no false positives), and every negative estimate is a true negative (i.e., there are no false negatives). In practice, this is more difficult, so the goal is to find an acceptable value which maximizes the accuracy of the algorithm while minimizing the number of false positives and false negatives. At the time of this writing, I have the sensor change threshold for the crash detection algorithm set to 5.0. This has correctly classified 100% of simulated crashes, but the value is still low enough to incorrectly classify some false positives without additional filtering. With the value set to 5.0, I have so far had a false positive 8 times. With an estimated fifty true negative classifications, the accuracy of the algorithm is approximately 87%. I reached this value by plugging in the following values to the accuracy formula: `TP = 3, TN = 50, FP = 8, FN = 0`. These values also yield a specificity value of approximately 86%. Due to time constraints, there is not any additional input to the classification algorithm, so any time the sensor change threshold is reached, the crash detection is triggered. In the future, I would either continue to gradually increase the threshold to reduce the number of false positives while still trying to maintain 100% sensitivity (correctly classify all true positives), or add some additional logic to differentiate between different types of events.

User Feedback

In this section, I will discuss the general feedback that was received from both the volunteer app users, as well as the feedback that was solicited from city employees and other individuals who are working in the area of bicycle safety. Because the user base of the app was strictly regulated during the testing process, some of the feedback received was based solely on a first impression, or otherwise very limited exposure to the app in action.

The intersection signal triggering functionality of the app has been in use by testers for approximately one month so far, and there have only been a few hiccups along the way. One issue that has been experienced by multiple users is that while riding northbound on Alder Street, no trigger request was sent, even when the user was within the specified distance to sent the trigger request. Another issue that I have experienced personally, but have not had reports of others experiencing, is that a request was sent and an error occurred. In this instance, the trigger was actually successful, but there may have been some other sort of error such as network timeout or something like that while waiting for the acknowledgment from the Particle Electron.

User 1
1. Do you think that this app could improve safety for bicycle riders in Eugene? It has the potential to do that. Crashes for cyclists are traumatic events, and its easy to forget to do small, critical, things that will help cyclists in the future investigation and litigation. An app could be a great vehicle for that.
2. What additional information would be helpful to be included in the report? The information gathered should help fill in the state/local police information in a way that caters to bicycle/peds. Speaking with a lawyer that specializes in bike/ped would be a good start (https://www.tcnf.legal/attorney/raythomas/). Perhaps that app could be in two stages: one for the immediate post-crash needs (photos, license plates, etc), while the second stage would help to fill the rather lengthy full crash report form.
3. Is the interaction with Alexa intuitive and easy to follow? The Alexa was nice, but its limitations and quirks do not seem to render it as useful as it would need to be in a crash situation.
4. When do you think Alexa should prompt the user to report a crash/incident? If the app is running and a crash is detected, it should ask repeatedly until the user interacts to tell it "yes or no" to a crash. If "yes" it should guide the victim through the process. Alexa may be a better "calming" guide voice, rather than a person you talk with, due to her limitations (ambient noise of a crash scene, etc...).
5. Any additional comments/feedback? Crashes with bicycles have different thresholds for triggering investigations and police follow up, an example of this is that in PDX an investigation will only be opened by police if the victim is transported to the hospital. I see the potential of this app to be an on-the-scene guide to help victims in the event they can still use the device. There are many other intricate pitfalls in the reporting process that should be addressed to ensure victims don't get neglected by the system. The ACLU publishes an app called "OR Justice" that performs a similar function but for police encounters. It may be a good model to think about the app going forward. All in all a great first draft, with lots of potential going forward.

Table 4.1 User 1 feedback

User 2
1. Do you think that this app could improve safety for bicycle riders in Eugene? Yes, the ability to decrease wait time at signals will decrease the number of cyclists that run lights or take routes that may be less safe.
2. What additional information would be helpful to be included in the report? Larger buttons. The "help" button wasn't working. Maybe a sketch pad?
3. Is the interaction with Alexa intuitive and easy to follow? It would be easier if the app showed when it was recording (a visual mic). There seemed to be some lag.
4. When do you think Alexa should prompt the user to report a crash/incident? Sudden stop or other data indicating rapid change.
5. Any additional comments/feedback? A great start! Maybe a combo of Alexa and some quick touch choices would provide options that could speed up reporting!

Table 4.2 User 2 feedback

User 3
1. Do you think that this app could improve safety for bicycle riders in Eugene? Yes, love that it prompts riders to submit reports and is easy to complete.
2. What additional information would be helpful to be included in the report? It would be helpful if the app was also tailored to more preventative reporting, before a crash actually happens. Riding in Eugene, you often see difficult intersections and near misses that would be helpful to document. When you ride regularly, you often know these spots and would be great if I could easily report so the city could fix for others.
3. Is the interaction with Alexa intuitive and easy to follow? Yes, although it seems like I would just hit the submit a report button instead of waiting for Alexa.
4. When do you think Alexa should prompt the user to report a crash/incident? If you were really in an accident, 30 seconds seems quick to be in the state of mind to report immediately. Maybe there could still be a reminder after the 30 seconds is up. Or this could also be a great pop-hole reporter for the city to fix roads.
5. Any additional comments/feedback? Think this would be a very cool tool that would help riders and the city. Great job!

Table 4.3 User 3 feedback

User 4
1. Do you think that this app could improve safety for bicycle riders in Eugene? Potentially, yes. As reports are coded and mapped, city staff can analyze conditions that may be leading to crashes and/or near-misses and be able to install an intervention. Also, it may reduce response time for emergency responders who are called to the site of a bicycle crash. The app appears to let people make an emergency call automatically which could be important if the rider is otherwise unable to access their phone.
2. What additional information would be helpful to be included in the report? A picture of the site or condition that is presenting a hazard.
3. Is the interaction with Alexa intuitive and easy to follow? I haven't set it up.
4. When do you think Alexa should prompt the user to report a crash/incident? There should be user confirmation that something happened. Maybe, Alexa can send a message to an emergency contact who can then attempt to call the app user to see if everything is okay?
5. Any additional comments/feedback? I haven't really tested it in depth...

Table 4.4 User 4 feedback

User 5
1. Do you think that this app could improve safety for bicycle riders in Eugene? I think the app could give some comfort to users that if involved in a crash of some kind they would have a tool available to them to report or document what happened. I do not think that an app focusing on crashes is the type of thing that would motivate or support those currently not cycling to give it a try; in fact it may have the counter effect, reinforcing other societal messages that riding a bike is inherently dangerous and unsafe (wear a helmet, wear bright colors, etc.).
2. What additional information would be helpful to be included in the report? The app should include a list of basic rights that cyclists have and basic laws that clarify what those rights are. There is a great resource called "Pedal Power" created by Oregon attorneys and available online. The app's report should automatically record time of day, day of week, weather conditions, and GPS location. There should be a photo function for multiple images so images can be taken of an offending vehicle, potential offending adult, license plate, injury or property damage.
3. Is the interaction with Alexa intuitive and easy to follow? It was easy to hear and the voice prompt could be a good technique, especially if voice responses by the user were automatically recorded and ended with a key word like "end".
4. When do you think Alexa should prompt the user to report a crash/incident? I'm not sure if it should come on automatically or only by request by the phone user. I suppose if this was not phone-based and was instead embedded into the handlebars, then it could either come on by pushing a handlebar button or automatically, but if automatically it seems like it would be dependent on the type of crash. Low level incidents could prompt it within 30-60 seconds, but something more severe would need to be a bigger delay. I think maybe the incident report could be prompted by voice command when the user is ready - maybe if the sensor senses the bike has crashed, then a blinking light appears - on the phone or the handlebars - and that prompts the user to fill out the report when the user is ready.
5. Any additional comments/feedback? It seems critical that any information from this report be able to be automatically connected to city/police incident databases, as well as an internal database of the app for future spatial visualization or analysis. My understanding of the app as is, is that the report is collated as a single audio file with no real way to be automatically parsed into an incident database. Unless there is a coordinated script that automatically parses an audio file into database field data, I'm not sure how useful the audio file will be for anything other than an audio diary of the event. It may be useful for the app to have an audio diary function to collect a first person narrative of the incident and attach that to the permanent record, but that audio file really needs to be a single form of data in its own field.

Table 4.5 User 5 feedback

It has been suggested that the crash reporting aspect of the app has the potential to help improve bicycle rider safety. One person stated that since crashes are a traumatic event, it can sometimes be easy to forget to do small, critical things that will help cyclists in the future, such as details that could be used later in an investigation and/or litigation. However, it was also noted that the current implementation is lacking in some features that would make it even more useful in a crash situation.

Another suggestion was that instead of the current 30-second timeout allotted for the user to cancel Alexa's crash detection, it would be better to repeatedly inquire if the user experienced a crash or near-miss incident until the user responds accordingly. Perhaps after a certain number of times, the app would then notify an emergency contact that someone has experienced an incident and include the GPS location information. This idea was proposed by multiple users providing feedback, so it is certainly worth looking into in the future.

Because investigations and police involvement typically only occur when a certain amount of property damage has occurred, or a person is transported to the hospital. This means that many incidents go undetected and unreported, leaving riders on their own to deal with the fallout of a potentially dangerous or harmful situation. Urban Bike Buddy may be a useful tool in the future to help guide people through the information gathering process so that they will have everything they might need. In addition, sharing the collected data with the local city government will allow them to include the locations of incidents along with the data they currently gather from other reporting sources to provide a better overall picture of areas of concern.

Another user suggestion for the crash reporting process is to allow the user to capture a photograph of the crash incident to include with the report. This would allow them to have the image available to provide to their insurance in a situation where they needed to file a claim due to injury or property damage. The photograph could also be useful if there were a future investigation, either by an insurance company or police.

Open Issues

There are some issues that remain open in the development of Urban Bike Buddy. Some of the main issues will be highlighted in this section.

Microphone Access. As discussed throughout this paper, there are limitations on the use of the smartphone's microphone by third-party applications. One of the downsides of this is that the application cannot be "always listening" and therefore removed the microphone as a useful addition to the accident detection. This is not an issue on Android smartphones, as it is possible to develop an application that runs as a service. However, having an application constantly running in the background can reduce the battery life of the smartphone.

Data Usage. Another area of concern is the amount of data required to interact with Alexa. A sample interaction with Alexa would typically include a wake word, "Hey, Alexa," for

example. This is substituted in Urban Bike Buddy with a detected incident causing Alexa to wake up and respond. With an Echo device, the response from Alexa is contained in a JSON message as plain text and the device will speak the text aloud. However, on a smartphone, the process is a little different. The response is sent back as an audio recording, which uses much more data than plain text.

To illustrate the difference in data size, consider the text "Alexa, open Urban Bike Buddy." That string is 28-bytes long. Of course, the entire JSON message would be longer than that, but consider a two second WAV file encoded to the specifications required by Amazon. The resulting WAV file is approximately 55-kilobytes. That is over 2,000 times the amount of data compared to the plain-text string. You can imagine scaling this up to multiple users speaking longer messages and receiving audio responses back from the Alexa Voice Service. Building on the 55-kilobyte two second clip, a one minute clip would be 1.5 megabytes. This may not seem like a lot in the scheme of things, but large amounts of audio can quickly use up the limited storage space of a smartphone, as well as use a large amount of data on a cellular plan. I have attempted to reduce this issue as much as possible in Urban Bike Buddy by overwriting the spoken audio each time, rather than storing any history; likewise with the audio response from Alexa Voice Services. However, this does not help the amount of cellular data used by the communication between the application and Alexa. Users with concerns for their data usage should take special note of this fact. In the future, perhaps smartphone applications will be able to receive responses in a text-based format similar to the behavior of the Echo devices.

Battery Usage. The addition of the M-series motion coprocessor allows the phone to collect, process, and store sensor data even while the device is asleep. Applications can then later retrieve the data once the device is powered back on. This practice can help reduce power draw, thus improving battery life. Some methods use machine learning to detect crashes or falls. Delahoz and Labrador (2014) surveys some fall detection and fall prevention methods. However, due to the computational complexity of performing these algorithms on a smartphone, I chose to avoid this in order to increase battery life thus improving the usefulness of the application for longer duration bicycle trips.

Sensor Accuracy. It is possible to adjust the accuracy of the device sensors. The accuracy of the location provided by the smartphone GPS is set by using the `desiredAccuracy`

property of the `CLLocationManager`. The different possible setting options for this were discussed earlier, in the GPS section. More important to the crash detection algorithm is the accuracy of the accelerometer and gyroscope sensors. The sensors themselves are very accurate, but the determinant of the algorithm's effectiveness is the frequency at which the sensor readings are measured. If the sensors are read very frequently, then it will provide the most effective measure for the algorithm to detect a sudden change in position or speed, usually indicating a crash or other notable incident. However, reading the sensors too frequently can result in rapid battery consumption, and the application loses its usefulness and appeal. Providing a balance between battery consumption and algorithm effectiveness is one goal of Urban Bike Buddy. I made every effort to choose values for each setting that would contribute to this balance. There are some techniques that other projects have used to improve the accuracy of their algorithms, which will now be discussed.

Casilari and Oviedo-Jiménez (2015) use a second device, a smart watch, in order to collect more accurate data to detect falls. They use a variation of Two-Phase Detection (PerFallD), which combines both the signal magnitude vector and the amplitude vector with respect to the absolute vertical direction to increase the accuracy of the fall detection. They were able to get the most accurate results when wearing the phone in a fixed position, such as the chest or waist. Storing the smartphone in the users pocket produced less accurate results due to the fact that the phone can move around freely in the pocket. However, by adding the watch, they are able to combine the readings from both devices to detect a fall more accurately than algorithms which only use the smartphone.

Simulated Incidents. It should be noted that the data used for designing the algorithm was gathered from simulated incidents. Evaluating the performance of an algorithm based on simulated data is not sufficient, so further evaluation should be done once data from real incidents has been collected. However, all ADB data that was used is from real-life daily bicycle riding from the users who volunteered to test the app, and also chose to record their sensor data and submit it for inclusion in the training data. Some have addressed this limitation by using a crash test dummy (Candefjord et al., 2014). This allows the simulation of incidents that are more likely to be similar to those which occur in normal riding. Others have even used a trained stunt person to simulate more realistic crash incidents. A comparison between these two simulation methods

can be learned about in Madsen et al. (2017).

Other Limitations. Another limitation is that there is a fixed 8-second timeout for responses to Alexa. This is not a limitation of Alexa Voice Services, but of the Alexa service in general. Typically, an Alexa skill will re-prompt the user after the timeout. In the case of Urban Bike Buddy, this would be a difficult way to handle the problem, as there could be some delay between the original Alexa message and the user beginning to record their response. Since I am using Alexa Voice Services, the full user message is recorded before being transmitted to Alexa. It is easy to imagine a scenario where the user may have a response that is longer than 8 seconds. I could work around this limitation by prepending every user recording with a fixed audio recording that would inform Alexa the appropriate skill for the recording to be routed to. Rather than going that route, I chose to let the user determine when they were recording each section of the crash report, and then only submit the audio recording once they have finished recording everything.

CHAPTER V

CONCLUSION

Urban Bike Buddy was developed as a proof-of-concept to show that the sensors in a smartphone are sufficient for crash detection. It has also been shown that the capabilities provided by Alexa Voice Services can be leveraged to submit a crash report in a relatively interactive manner. The primary focus of this work has been to demonstrate these findings, as well as to help provide a more pleasant bicycle riding experience to riders along the Alder Street corridor on campus. Since the Particle Electron was connected at the intersection of 18th and Alder, it is much more likely that a rider using the app will receive a green light faster, and riders are able to wait at the signal without wondering if their presence has been detected. I will discuss some of the limitations of the project and opportunities for future work.

Scope and Limitations

As mentioned previously, one major limitation of the project is that there is currently only one intersection that supports the new functionality. However, this intersection is located on campus in a high-traffic bicycle corridor, so it is a very good location to begin testing. Also, the main functionality is complete, so it will be easy to expand to more intersections in the future with relatively little overhead.

In order for the city to connect our project to their infrastructure, there were some additional requirements that needed to be fulfilled before they would allow us to proceed. The most important requirement that they set forth was to implement a "kill-switch," that they would be able to use to remotely stop the system if any issues arose. In the process of adding this functionality for them, I also added the ability for them to adjust some of the settings related to the Particle Electron box such as the device timeout, as well as the thresholds at which the application will send a trigger to the device. After these requirements were met and some on-site testing took place, the city traffic engineers connected our box to their infrastructure, ultimately bringing the system live.

Future Work

My solution has been designed as a context-aware algorithm, with the assumption that the smartphone will be mounted on the handle bars of a bicycle while riding. Ideally, the algorithm would be adapted to allow multiple device locations such as in the user's pocket, or in a

backpack, etc. However, generalizing the algorithm is beyond the scope of this paper and is left as an opportunity for future work. In addition, there are other possibilities for implementing the functionality provided by the voice integration, such as using Alexa to guide the user through each step of the crash reporting process. Alexa could also have different prompts based on the type of incident detected. If a small spike is recorded, perhaps making a note of this and prompting the user once they have finished their ride to see if they had encountered a pothole or something else that may have caused the sudden jerking motion measured by the sensors that would cause a spike.

Urban Bike Buddy currently only detects one type of incident, which is considered as a crash. However, by adjusting the different parameters of the algorithm, it should be possible to add the ability to detect additional incidents. In the future, it would be nice to be able to distinguish different type of incidents such as a sudden stop, a sharp turn/swerve, or a fall. At the time of this writing, any incident that is detected by the algorithm is simply considered a crash, and Alexa will then prompt the user for the report.

Adding microphone-based accident detection might be interesting in the future, but phones typically do not allow applications to have full-time access to the microphone data. This is for both performance and security reasons. Allowing the microphone to be always listening could be abused by malicious developers to eavesdrop or collect sensitive information, unbeknownst to the user. Also, if apps were constantly using the microphone, the phone battery would drain faster than otherwise.

There is a project of interest called Snowboy mentioned in Chapter III. Snowboy is basically an always-on app that listens to the microphone; however, the audio processing is handled on the local device, and only once the wake word is detected does the device begin to send the captured audio to external services. It is also possible to run Snowboy on your own server, implement a custom vocabulary, and other interesting functionality without the need of a third- party service such as Amazon's Alexa. In the future, it could be interesting to add the functionality of Snowboy into the Urban Bike Buddy application.

All of the functionality that has been discussed related to Urban Bike Buddy is implemented in a way that is device agnostic. All of the third-party services are available for multiple platforms, so it would not be a difficult task to develop a comparable app for Android, or some

other mobile operating system. It is also possible that all of the same functionality could eventually be added to an embedded system that is actually built into a bicycle. With the advancements in technology and the cost of hardware decreasing, it is certainly not hard to imagine something like this being implemented on a wide scale in the near future.

It has been shown that a smartphone can be used as both a crash detection device, as well as a means of reporting a crash. The phone has the ability to submit an audio recording of a crash report, along with the GPS coordinated of the crash location, and also the ability to initiate a phone call or other alert message to emergency contacts in the event of a crash. In addition to being used as a safety device, the phone can also be used as a beacon to transmit the user's location to a smart, connected intersection, which can then request a signal change for the bicycle phase while the rider is approaching, possibly granting them a crossing signal without needing to stop and wait. Reducing the amount of time a bicycle rider waits at the intersection is also beneficial for bicycle safety, because some riders will simply ignore the signal and cross if they see that there are no cars coming. It it my hope that the combination of these benefits will entice more people to choose to ride their bicycle, if only for their commute to campus.

APPENDIX A

SAMPLE DATA

Below are graphs of some of the accelerometer data collected from normal bicycle riding activity during the testing of Urban Bike Buddy. The graphs are labeled with the activity being shown; if there is anything besides normal riding activity, the notable event is stated. The graphs are shown to display the challenge of differentiating between activities of daily bicycling and a crash event.

ADBs

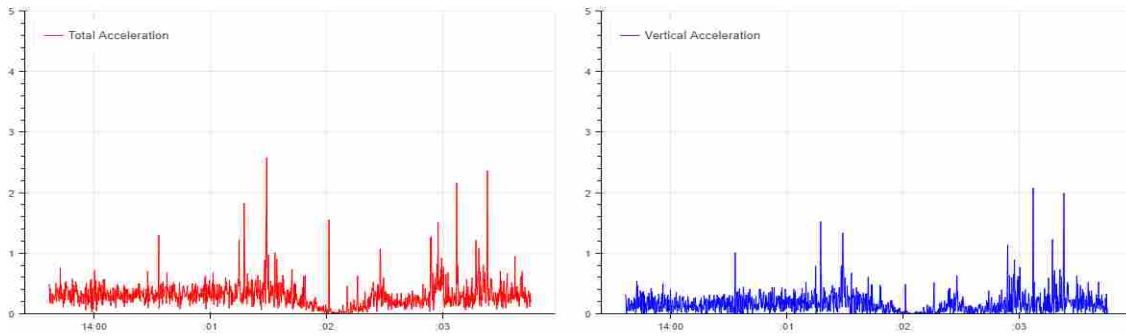


Figure A.1 Normal riding activity

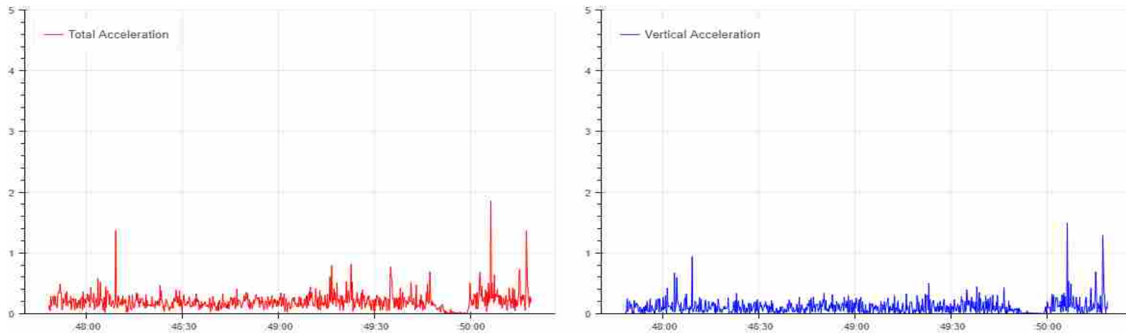


Figure A.2 Normal riding activity

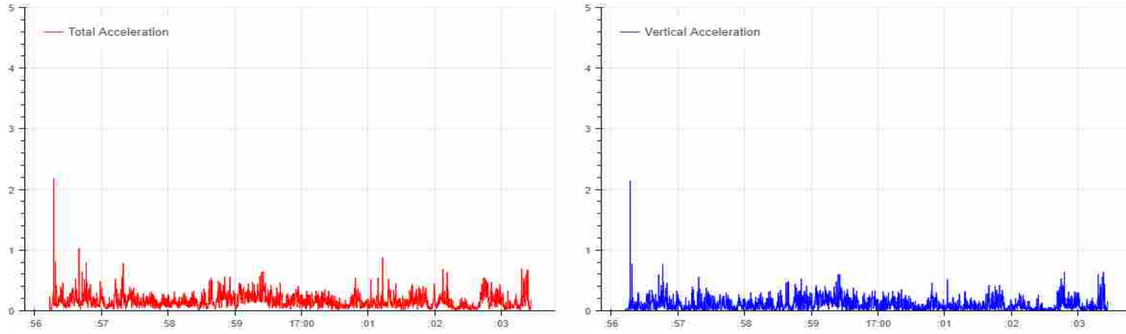


Figure A.3 Normal riding activity

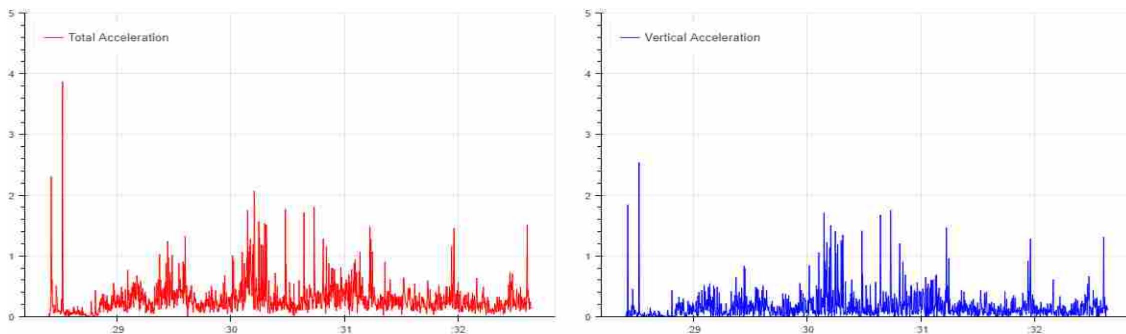


Figure A.4 Normal riding activity

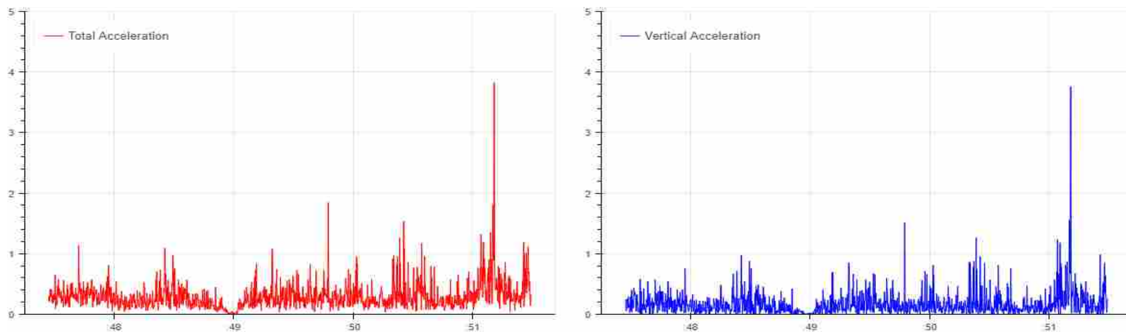


Figure A.5 Normal riding activity

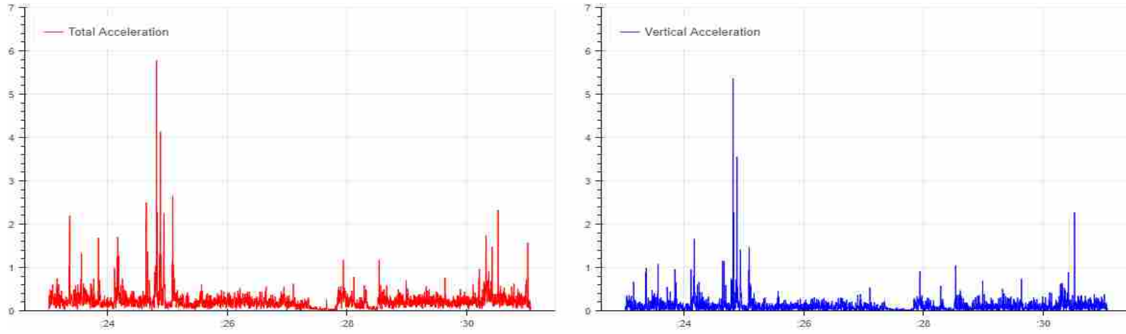


Figure A.6 Normal riding activity

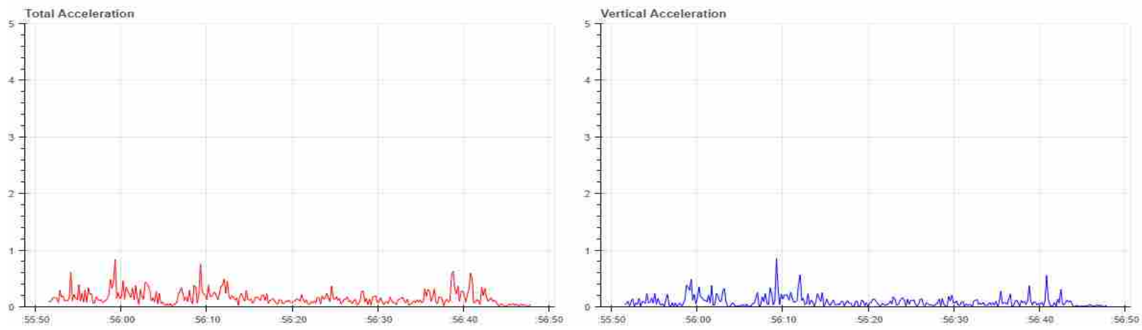


Figure A.7 Normal riding activity

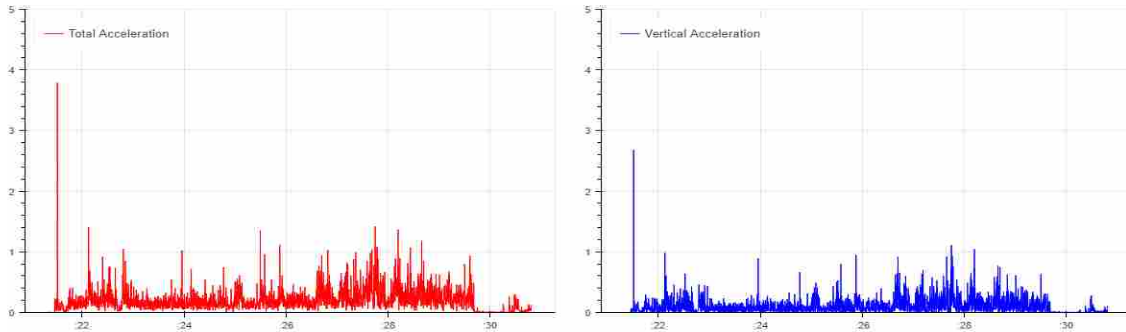


Figure A.8 Normal riding activity

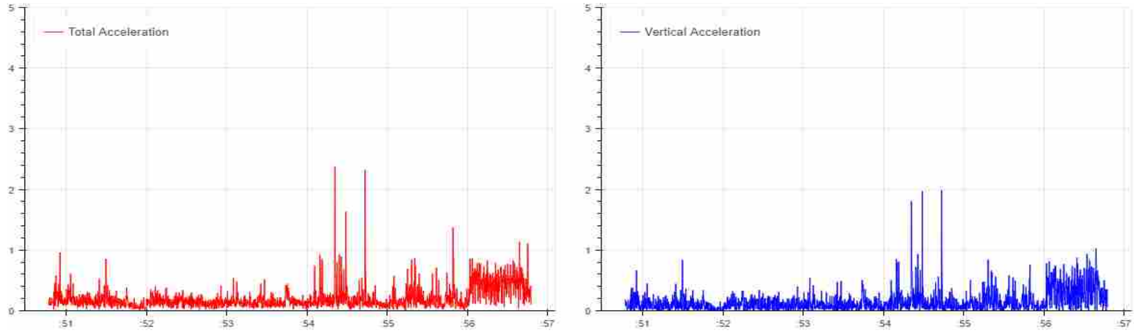


Figure A.9 Normal riding activity

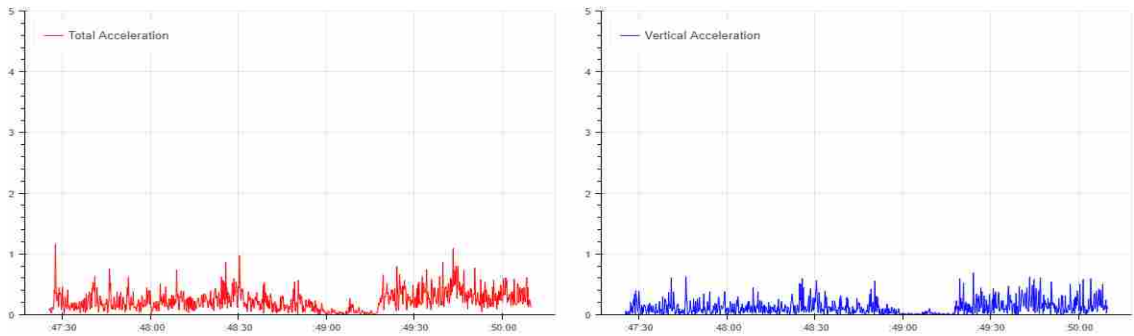


Figure A.10 Normal riding activity

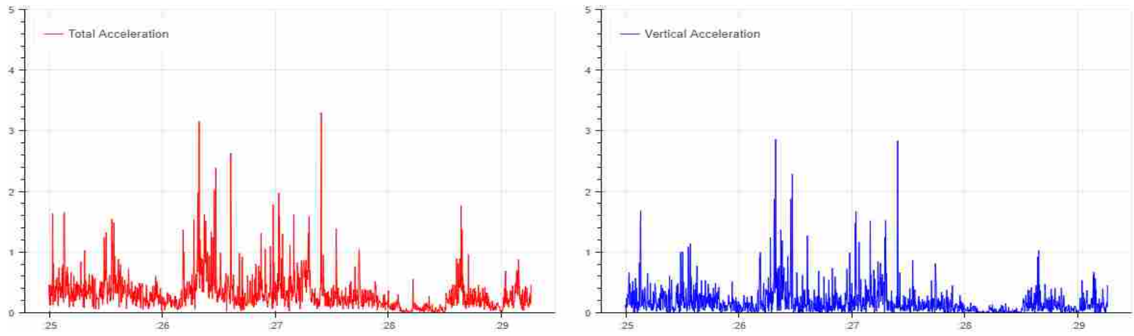


Figure A.11 Normal riding activity

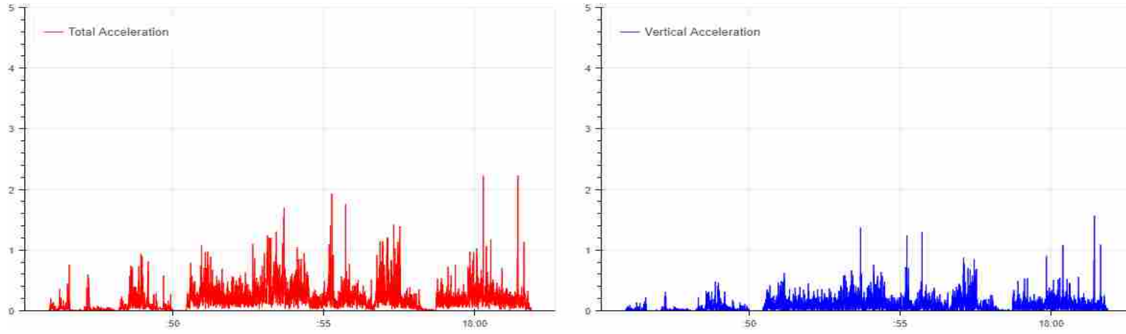


Figure A.12 Normal riding activity

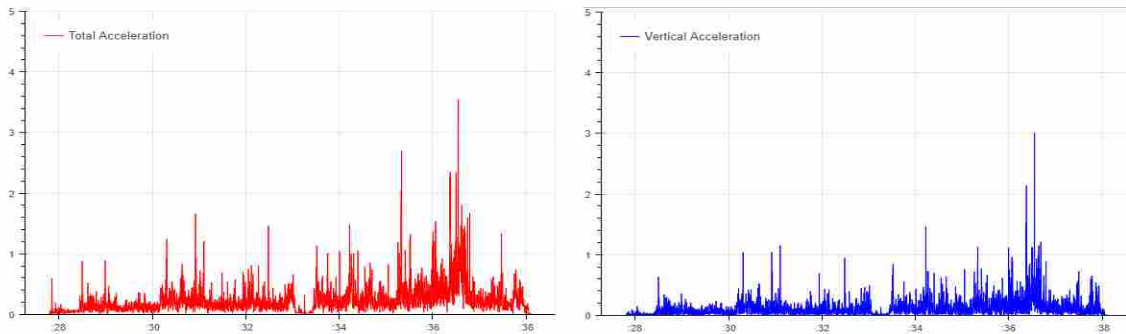


Figure A.13 Normal riding activity

Notable Events

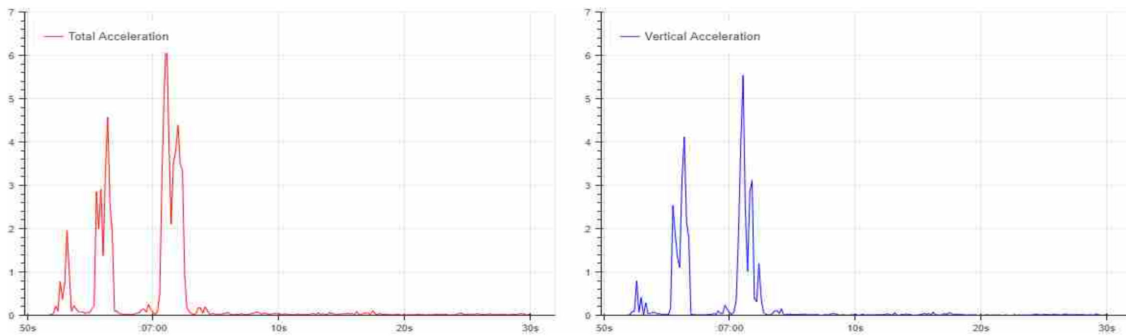


Figure A.14 Shake gesture

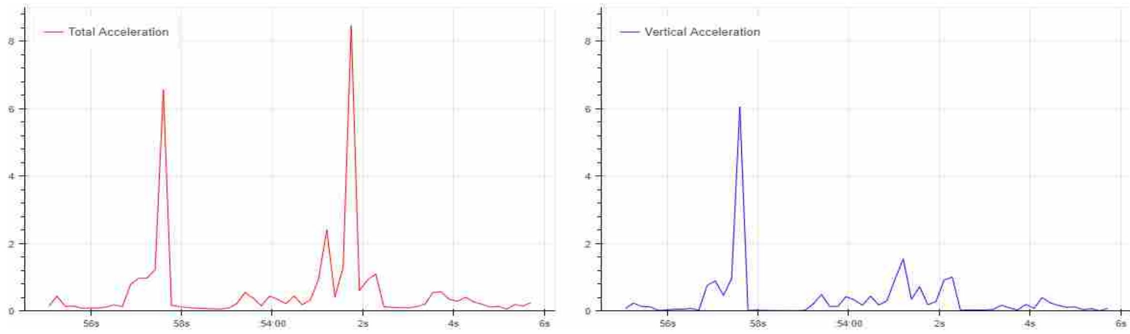


Figure A.15 Phone being dropped

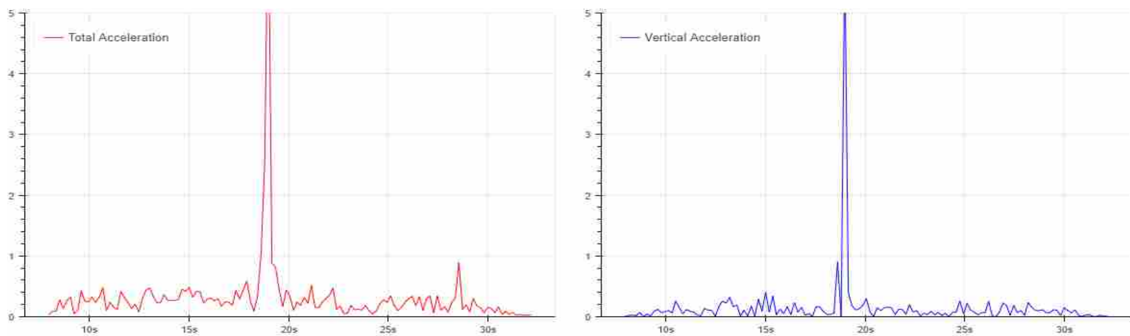


Figure A.16 Rider rides straight off curb

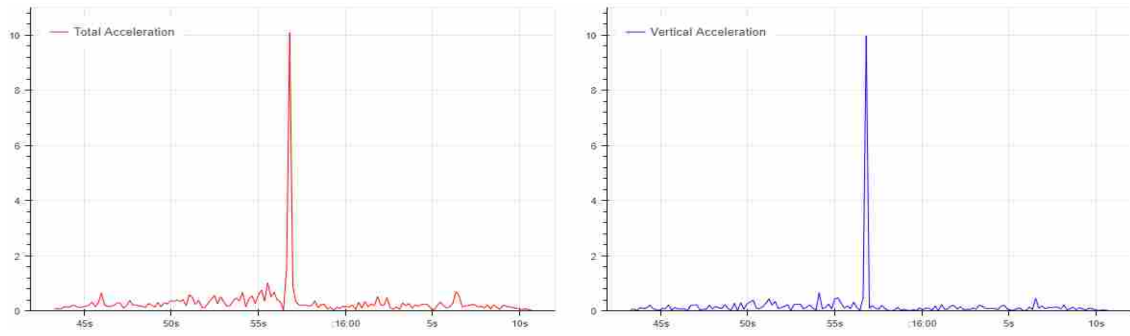


Figure A.17 Rider rides off curb to the left

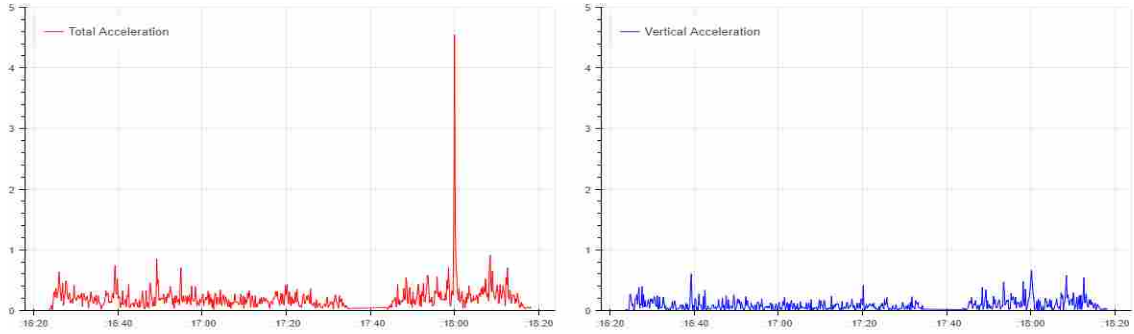


Figure A.18 Rider rides off curb to the right

Simulated Crash Events

One important thing to note about the simulated crash events is that the speed at which the event occurred was much slower than a typical bicycle crash would be likely to occur. The data is used here only for the purpose of illustrating the sensor data relative to other activities of daily bicycling sensor data.

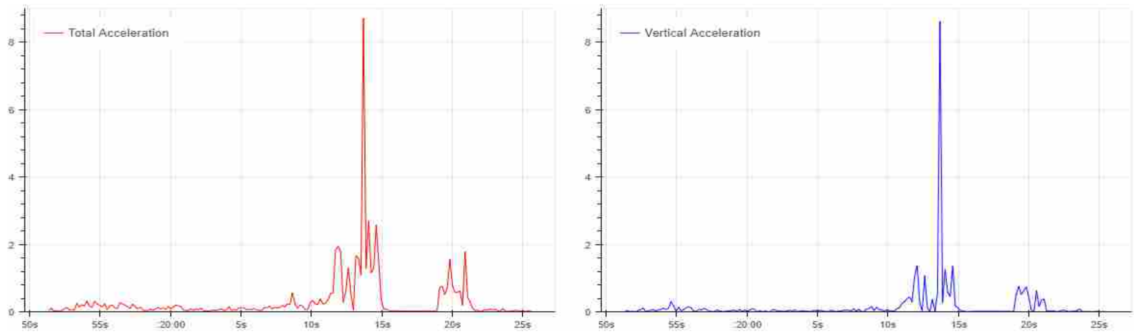


Figure A.19 Simulated crash 1

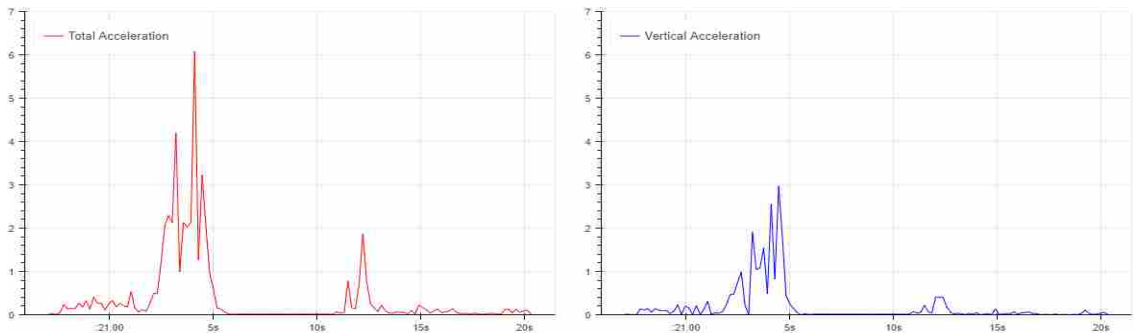
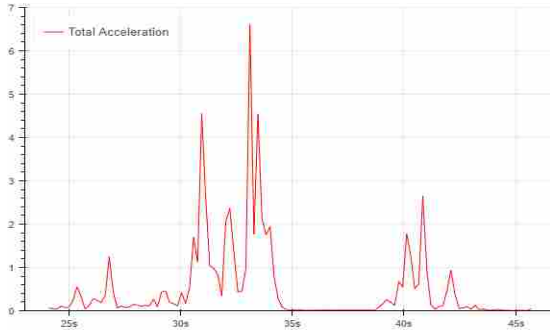


Figure A.20 Simulated crash 2



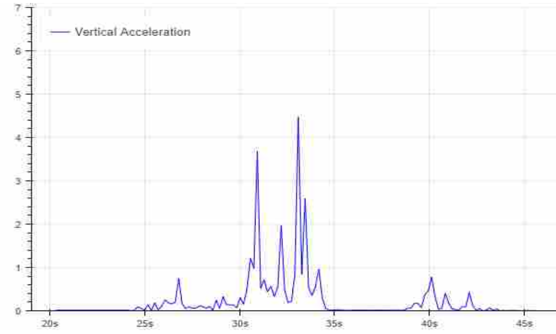
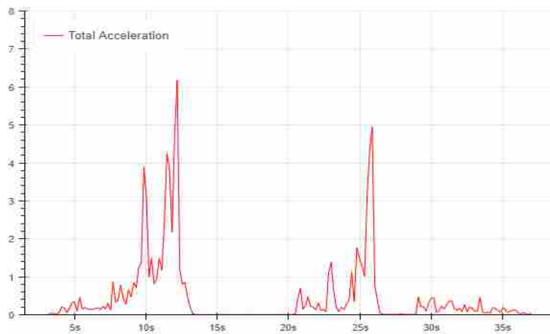


Figure A.21 Simulated crash 3



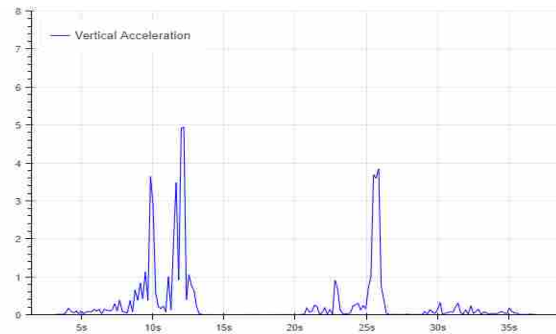
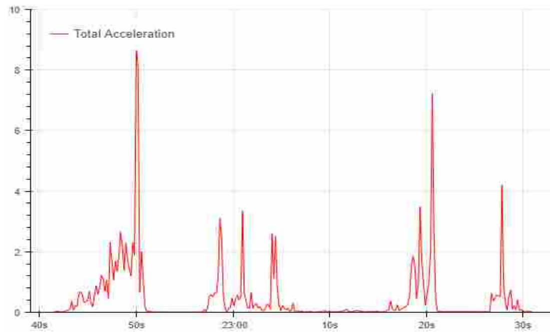


Figure A.22 Simulated crash 4



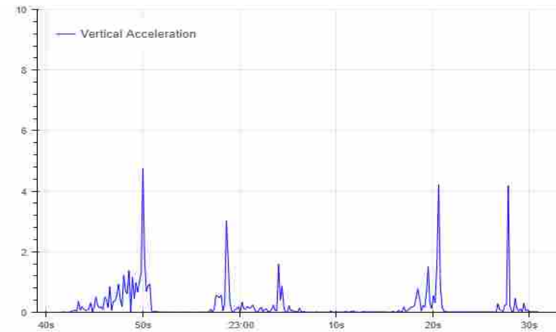


Figure A.23 Simulated crash 5

APPENDIX B

ADDITIONAL CODE LISTINGS

```
1  import pandas
2  import numpy as np
3  import sys
4  from bokeh.layouts import row
5  from bokeh.plotting import figure, show
6
7  def datetime(x):
8      return np.array(x, dtype=np.datetime64)
9
10 csv_file = sys.argv[1];
11 df = pandas.read_csv(csv_file);
12
13 df['Date'] = pandas.to_datetime(df['Date'])
14
15 total_acc = df.query("Type == 'A_t'")
16 vert_acc = df.query("Type == 'A_v'")
17
18 plot = figure(x_axis_type='datetime', y_range=(0, 5),
      plot_height=400, plot_width=600)
19 plot.line(datetime(total_acc.Date), total_acc['val'],
      color='red', legend='Total Acceleration')
20 plot.legend.location = 'top_left'
21
22 plot2 = figure(x_axis_type='datetime', y_range=(0, 5),
      plot_height=400, plot_width=600)
23 plot2.line(datetime(vert_acc.Date), vert_acc['val'],
      color='blue', legend='Vertical Acceleration')
24 plot2.legend.location = 'top_left'
25
26 p = row(plot, plot2)
27
28 # display the results
29 show(p)
```

Listing B.1 Generating Bokeh Plots

APPENDIX C

URBAN BIKE BUDDY: INFORMATION FOR APP USERS

Background

The Urban Bike Buddy app is being developed as part of a research grant supported by the National Institute for Transportation and Communities (NITC). The goal is to improve the user experience of people on bikes by increasing the likelihood that traffic signals turn green as the bike approaches.

The first test of this new system is with the bicycle-specific traffic light at Alder and 18th and will work in either the north or the south direction only. The app is not connected to either the car or the pedestrian-oriented traffic signals. We have added a special computer to the signal controller (which we call the bike-controller) that will talk to your app. Your app talks to it and it talks to the actual bike signal.

Future iterations of the app may actively influence pre-determined, variable traffic general signals (they are responsive to the presence of people waiting) and may give cyclists information to speed up or slow down to catch an upcoming green light.

Downloading the App

You will be sent an email to download Urban Bike Buddy app by redeeming a special code via the iPhone TestFlight app (yes, this is an app within an app, so two easy downloads). Once you have Urban Bike Buddy on your phone, there is no need to login or create a password.

Using the App

This version of the Urban Bike Buddy is very simple as we test its basic abilities. To use:

- Press the start trip button and thats basically all you need to do.
- It is helpful if you can also press the round red button under the Start Trip to record the data your phone is sending and receiving from the traffic signal so that we can evaluate everything later.
- When your trip is done, press the red square where the red circle used to be, and then press the mail envelope to send the data to us (you can type any comments into the email if you want we will read them and enjoy what you have to say).
- Press the Stop Trip button where Start Trip was.

What you see on the App

The app can run in the foreground or the background. If in the background, you can use other apps simultaneously if you wish. If in the foreground, you can watch it in action. Here is what you can expect to see.

- When you push the Start Trip button, the app starts paying attention to your distance from the signal at 18th and Alder. It has 3 distance zones that changes the status box from gray:
  - Red: Too far from 18th and Alder to start communication.
  - Yellow: Within range of the bike-controller (currently 200-500 feet away).
  - Green: Automatically tells the bike-controller you are coming (within 200 feet).
- When the bike-controller queues up the bike light, the phone will vibrate.
- The status-box reverses colors as you ride away from the intersection: green to yellow and yellow to red.
- If a trip has not been started, the status box will be gray or white.

An important note

The app puts in a request to the bike-controller just like pressing a pedestrian crossing signal. That does not mean the bike light will immediately turn green, but does tell the signal you are coming, which can help the signal accelerate through its other users to better accommodate you on a bike.

Contact information

This project is being led by the University of Oregons Dr. Stephen Fickas (Computer Science) and Dr. Marc Schlossberg (Planning, Public Policy and Management) with graduate student assistance from Brian Williams (who receives the app emails). If you have questions, comments, or suggestions, please send them to fickas@cs.uoregon.edu.

REFERENCES CITED

Amazon cognito - simple and secure user sign up & sign in. (2018). Retrieved 13 May 2018, from https://aws.amazon.com/cognito

Amazon simple notification service (sns). (2018). Retrieved 13 May 2018, from https://aws.amazon.com/sns

Augenstein, J., Digges, K., Ogata, S., Perdeck, E., & Stratton, J. (2001). Development and validation of the urgency algorithm to predict compelling injuries. In *Proceedings of the 17th international technical conference on the enhanced safety of vehicles (esv)* (pp. 4–7).

Aws lambda - serverless compute. (2018). Retrieved 12 May 2018, from https://aws.amazon.com/lambda/

Bagalà, F., Becker, C., Cappello, A., Chiari, L., Aminian, K., Hausdorff, J. M., . . . Klenk, J. (2012). Evaluation of accelerometer-based fall detection algorithms on real-world falls. *PloS one*, *7*(5), e37062.

Baker, J. (1975, Feb). The dragon system–an overview. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, *23*(1), 24-29. doi: 10.1109/TASSP.1975.1162650

Bhavthankar, S., & Sayyed, H. (2015). Wireless system for vehicle accident detection and reporting using accelerometer and gps. *Electronics and Communication Engineering (ECE). IJ*.

Bourke, A., Obrien, J., & Lyons, G. (2007). Evaluation of a threshold-based tri-axial accelerometer fall detection algorithm. *Gait & posture*, *26*(2), 194–199.

Candefjord, S., Sandsjö, L., Andersson, R., Carlborg, N., Szakal, A., Westlund, J., & Sjöqvist, B. A. (2014). Using smartphones to monitor cycling and automatically detect accidents: Towards ecall functionality for cyclists. In *International cycling safety conference 2014, 18-19 november 2014, göteborg, sweden.*

Casilari, E., & Oviedo-Jiménez, M. A. (2015). Automatic fall detection system based on the combined use of a smartphone and a smartwatch. *PloS one*, *10*(11), e0140929.

Chen, J., Kwong, K., Chang, D., Luk, J., & Bajcsy, R. (2006). Wearable sensors for reliable fall detection. In *Engineering in medicine and biology society, 2005. ieee-embs 2005. 27th annual international conference of the* (pp. 3551–3554).

Dai, J., Bai, X., Yang, Z., Shen, Z., & Xuan, D. (2010). Perfalld: A pervasive fall detection system using mobile phones. In *Pervasive computing and communications workshops (percom workshops), 2010 8th ieee international conference on* (pp. 292–297).

Delahoz, Y. S., & Labrador, M. A. (2014). Survey on fall detection and fall prevention using wearable and external sensors. *Sensors*, *14*(10), 19806–19842.

Demographics of mobile device ownership and adoption in the united states. (2018). Pew Research Center. Retrieved 19 May 2018, from http://www.pewinternet.org/fact-sheet/mobile/

Garg, S. K., Versteeg, S., & Buyya, R. (2013). A framework for ranking of cloud computing services. *Future Generation Computer Systems*, *29*(4), 1012–1023.

Gurupriya, C., Kaviya, K. K., Mohana, A., Raj, S. M., & Bebington, L. P. S. (2016). Gsm/gps based remote accident alert system using arduino. In *3rd national conference on recent trends in communication & information technology* (Vol. 2).

Habib, M. A., Mohktar, M. S., Kamaruzzaman, S. B., Lim, K. S., Pin, T. M., & Ibrahim, F. (2014). Smartphone-based solutions for fall detection and prevention: challenges and open issues. *Sensors*, *14*(4), 7181–7208.

Huang, C.-M., & Chang, Y.-J. (2015). A mobile-based novice accident detection and tracking system for bicycle. *National Yunlin University of Science and Technology*.

Juhra, C., Wieskoetter, B., Chu, K., Trost, L., Weiss, U., Messerschmidt, M., . . . Raschke, M. (2012). Bicycle accidents–do we only see the tip of the iceberg?: A prospective multi-centre study in a large german city combining medical and police data. *Injury*, *43*(12), 2026–2034.

Kim, T., & Jeong, H.-Y. (2014). A novel algorithm for crash detection under general road scenes using crash probabilities and an interactive multiple model particle filter. *IEEE Transactions on Intelligent Transportation Systems*, *15*(6), 2480–2490.

Madsen, T. K., Christensen, M. B., Andersen, C. S., Várhelyi, A., Laureshyn, A., Moeslund, T. B., & Lahrmann, H. (2017). Comparison of two simulation methods for testing of algorithms to detect cyclist and pedestrian accidents in naturalistic data. In *6th international naturalistic driving research symposium.*

Matuszczyk, G., & Åberg, R. (2016). *Smartphone based automatic incident detection algorithm and crash notification system for all-terrain vehicle drivers* (Master's thesis, Chalmers University of Technology). PDF.

Measure energy impact with xcode. (2016). Retrieved 24 Feb 2018, from https://developer.apple.com/library/content/documentation/Performance/Conceptual/EnergyGuide-iOS/MonitorEnergyWithXcode.html

Particle — electron (2g/3g/lte) cellular hardware. (2018). Retrieved 19 May 2018, from https://www.particle.io/products/hardware/electron-cellular-2g-3g-lte

Sposaro, F., & Tyson, G. (2009). ifall: an android application for fall monitoring and response. In *Engineering in medicine and biology society, 2009. embc 2009. annual international conference of the ieee* (pp. 6119–6122).

Sulochana, B., & Manohar Babu, B. S. (2014). Monitoring and detecting vehicle based on accelerometer and mems using gsm and gps technologies. *International Journal of Computer Science Trends and Technology (IJCST)–Volume*, *2*.

White, J., Thompson, C., Turner, H., Dougherty, B., & Schmidt, D. C. (2011). Wreckwatch: Automatic traffic accident detection and notification with smartphones. *Mobile Networks and Applications*, *16*(3), 285.

Yoshida, T., Mizuno, F., Hayasaka, T., Tsubota, K., Wada, S., & Yamaguchi, T. (2005). A wearable computer system for a detection and prevention of elderly users from falling. *Proc ICBM*, *12*, 179–82.