NITC
NATIONAL INSTITUTE for TRANSPORTATION and COMMUNITIES

FINAL REPORT

V2X: Bringing Bikes Into the Mix

NITC-RR-1027 ■ March 2019

NITC is a U.S. Department of Transportation national university transportation center.

TREC
TRANSPORTATION RESEARCH and EDUCATION CENTER

V2X: BRINGING BIKES INTO THE MIX

Final Report

NITC-ED-1027

by
Stephen Fickas
University of Oregon

for

National Institute for Transportation and Communities (NITC)
P.O. Box 751
Portland, OR 97207

March 2019

Technical Report Documentation Page

1. Report No. NITC-ED-1027	2. Government Accession No.	3. Recipient's Catalog No.
4. Title and Subtitle V2X: Bringing Bikes Into The Mix		5. Report Date March 2019
		6. Performing Organization Code
7. Author(s) Stephen Fickas		8. Performing Organization Report No.
9. Performing Organization Name and Address Computer and Information Science Department Deschutes Hall University of Oregon		10. Work Unit No. (TRAIS)
		11. Contract or Grant No.
12. Sponsoring Agency Name and Address National Institute for Transportation and Communities (NITC) P.O. Box 751 Portland, OR 97207		13. Type of Report and Period Covered
		14. Sponsoring Agency Code
15. Supplementary Notes		

16. Abstract

This project demonstrates how an inexpensive system (hardware and software) can add new functionality to existing signal controllers, giving bicyclists an efficient way to cross a controlled intersection. The system integrates three components: (1) a Bike Connect box that resides near the signal-controller and is connected to it, (2) an application that runs on a Bike Connect device (currently an iPhone) and requests a green light at the correct approach-distance, and (3) a cloud-based publish/subscribe (pub/sub) component that handles cellular-communication between phone app and box.

One stumbling block for the project was a means to obtain reliable GPS data to compute distance while walking, biking and standing still (being idle). We report on our evaluation of 4 methods: raw GPS, averaged GPS, line of best fit and speed. We found that a combination of methods was most effective and describe that combination and its results. The final system was put into place and tested with 120 separate rides. We report on our results and potential future paths to take the research.

17. Key Words Smart Transportation, Active Transportation, V2X, IoT		18. Distribution Statement No restrictions. Copies available from NITC: http://nitc.trec.pdx.edu	
19. Security Classification (of this report) Unclassified	20. Security Classification (of this page) Unclassified	21. No. of Pages 30	22. Price

ACKNOWLEDGMENTS

This project was supported with financial support from the National Institute of Transportation and Communities under grant number 1027.

We are grateful to the City of Eugene Transportation Office, and Matt Rodrigues, Mike Firchland and Andrew Kading in particular. Two graduate students took on projects related to the grant: a big thanks to Brian Williams and Ben Bennett. We also thank Professor Schlossberg for his valuable insight.

CITATION

Fickas, Stephen. *V2X: Bringing Bikes Into the Mix*. NITC-ED-1027. Portland, OR: Transportation Research and Education Center (TREC), 2019.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF ALGORITHMS

EXECUTIVE SUMMARY

This project focuses on a type of transportation that is currently left out of V2X conversations: bicycling. The project demonstrates how an inexpensive system can add new functionality to existing signal controllers, giving bicyclists an efficient way to cross a controlled intersection. The system proposed and demonstrated integrates three components: (1) a Bike Connect box that resides near the signal-controller and is connected to it, (2) an application that runs on a Bike Connect device (currently an iPhone) and requests a green light at the correct approach-distance, and (3) a cloud-based publish/subscribe (pub/sub) component that handles communication between phone app and box.

As a separate project we developed a set of video lessons that provide a clear and detailed roadmap for giving students a chance to explore V2X technology and, in the end, produce something that can be used in their own community, i.e., the Bike Connect box. This related project can be referenced at Project Phenom. It contains course topics on: (1) Doing electronic (solderless) breadboarding to connect a modern Internet of Things device, a cellular embedded computer, into the heart of their system. (2) Demonstrating how the embedded computer (a Particle Electron) can easily control relays, which provide the virtual push-button. (3) Packaging up their system into a container that can be placed on a signal pole. (4) Lessons on how to employ C++ and object-oriented programming to control the Electron from the cloud. When completed, a student will have built a key component of the proposed system.

In this project report we will highlight the technology that has allowed us to implement the other 2 components of the system: the Bike Connect device and its application; the pub/sub component that manages communication in the system. We will focus, in particular, on one challenge we faced with the phone application, that of determining an accurate distance measure for a bike rider using our application. With an accurate distance measure, the phone application can calculate when to place a green request as the bike rider approaches the intersection.

The full system was made operational in spring of 2018 and is now in beta testing. Please contact the author, Stephen Fickas (fickas@cs.uoregon.edu), for more information.

1.0 INTRODUCTION

In the rush to develop and install smart-transportation (V2X) systems so that they can support connected and autonomous vehicles, people on bike seem to be generally left out. At best, bicycles are viewed as obstacles to be ‘seen’ and avoided by sensored-up cars. This is an extremely limited view of the future, especially at a time when cities are significantly investing in bicycle infrastructure and bike share systems, and where the private sector has discovered a possible transportation market in bike and scooter share systems.

This project focused on integrating cyclists purposefully into the connected systems environment by developing direct and indirect means for people on bike to interact virtually with signal-controlled intersections in an eventual effort to improve safety and efficiency. Through the development of a low-cost traffic signal box add-on and a phone app, we were able to add virtual requests and useful information to cyclists’ experiences at the location tested in Eugene, Oregon.

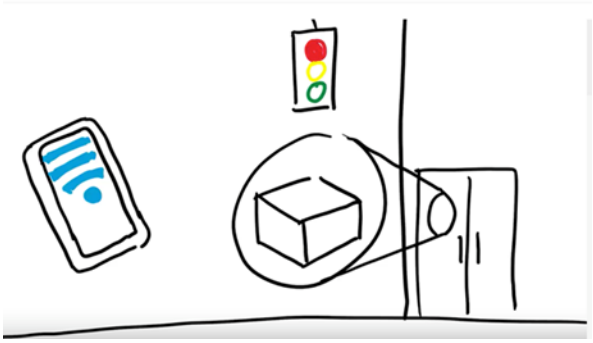

Figure 1: Illustration of Bike Connect System. The project developed V2X technology by constructing a Bike Connect device that can communicate with a Bike Connect box (circled) to trigger a green light from a distance.

In this project report, we will highlight the technology that has allowed us to implement our vision of giving bikes an effective means to integrate with a city's transportation grid, which we call the "Bike Connect" system. We have implemented both a Bike Connect box and a Bike Connect device (in the form of an iPhone app) for use at an intersection along a major bike-corridor in Eugene. We used a publish/subscribe (pub/sub) software architecture and explored a number of GPS algorithms for calculating distance, direction, and speed traveled by a cyclist. The entire system was field tested in Eugene, on its busiest bicycle corridor, the intersection of Alder and 18th. We describe our approach and results in more detail in the following sections.

2.0 BIKE CONNECT SYSTEM OVERVIEW

The Bike Connect System is designed to enhance the experience, comfort, efficiency, and safety of people on bike by making traffic signal processes and information more cyclist attentive. The core aspect of the Bike Connect system is to increase the likelihood that someone on bike will experience a green light when approaching a signal. There are two primary aspects of this effort: 1) active traffic signal engagement by approaching bicycle users; and 2) passive information sharing to cyclists of traffic signal status. This project tackles active engagement. We discuss our ideas for passive engagement in the Future Work section.

On the active engagement side, we have developed a low cost, add-on control box that can attach to any traffic signal control box in the nation and allow for cell connectivity to cyclists through a secure app, which we have also developed for testing purposes. The system essentially 'calls' the traffic signal in advance of an approaching cyclist based on the direction and speed of travel in order to maximize the likelihood of a green when arriving at an intersection with variable timed signals – like virtually pressing the crosswalk button in advance. There are four benefits from this new infrastructure:

1. The signal turns green in advance of the cyclist arriving based on the virtual call to the signal;

2. If the signal sequencing does not allow for transitioning to a green for the cyclists prior to arrival, the signal will turn green much faster because of the advance notification;
3. Loop detectors in the ground (or video or laser) are no longer needed nor is it needed to educate cyclists where to position a bike to put a call into the signal (this benefit should not be underestimated and is a particular problem at the intersection where the field test took place); and
4. Cyclists' adherence to the signal (i.e. waiting for the green instead of crossing against a red) increases because the user knows the signal has been alerted to her/his presence.

'Urban Bike Buddy' app icon

The following outlines how the system works:

- We attached the Bike Connect 'box' to a specific traffic control signal box (i.e., 18th and Alder, Eugene). The box listens for requests from bicyclists and when one is received, signals the controller through normal means. More details are given in section 4.

- A cyclist downloads our app from the app store and registers using their email account.

- At the start of a trip, the cyclist starts the phone app ('Urban Bike Buddy'), which establishes communication with the control box. The app shows a yellow bar to alert the user that communication is established.

- When the cyclist is within a predetermined time to reach the intersection (calculated from distance and speed), the app places a request to the box. When the box acknowledges the request, a green bar is shown on the app.

- The system resets after the bicyclist has crossed the intersection.

The main interface for the Bike Connect system app, called 'Urban Bike Buddy', is shown in figure 2 The 'Start Trip' button starts communication between the app and the bike box. The gray box changes color: yellow – communication is established; green – a request has been sent to the box and acknowledged. The Amazon logo initiates voice-activated 2-way controls via the Alexa system, which was not an official component of this project, but one that received some initial testing to create a screen-free option for cyclists.The app can run in background mode, which frees the user from always remembering to start the app at the beginning of a trip. However, the user then loses the color-coded visual cues. We are working on a sound system that can be used in background mode to complement the visual cues. In particular, we would like to give a rider the option to carry their phone in a backpack and still hear a chirp when a request has been made.

Figure 2: Bike Connect Phone App Interface.

3.0 FIELD TESTING LOCATION

Once the Bike Connect control box and basic app functionality and communication systems with the control box were developed in the lab, the system was field-tested. Working with the City of Eugene traffic engineering team, the system was deployed at the corner of Alder and 18th. Eugene is a Gold level bike friendly community and Alder is Eugene's most traveled bicycle corridor, connecting southern neighborhoods to the University of Oregon, downtown, and an extensive river path system. In the 1970s, Alder became the first street in the nation to have a contraflow bicycle lane (13th - 18th Streets), directly adjacent to the University of Oregon. Alder is a bicycle boulevard in the southern residential portion (40th to 19th Streets) and in 2012, Alder was re-designed from 19th to the river (approximately 5th Street) largely as a two-way buffered bike lane with a two block portion a protected 2-way bike lane with car parking serving as the physical buffer.

The intersection of Alder and 18th is central to this corridor and is a variable timed signalized intersection with a bicycle-only signal for Alder travel. Loop detectors and advanced loop detectors currently exist in both directions on Alder to recognize the presence of bicycles. When initially installed, bike detection was done by signal-mounted video detection, but was switched to loop detectors when the video could not be reliably calibrated. The loop detector senses a bike at the stop-line and places a request on the bike phase. From on-site observation we noticed that many cyclists waited for the light at the wrong location (did not trigger the detector), became frustrated, and went against a red light.

Figure 3: Alder Street section adjacent to University of Oregon. Alder St., equal allocation to bicycle travel (on right) as motorized vehicle travel (on left). (Source: Google Street View, August 2017 (accessed 7/4/2018)).

Figure 3: Intersection of Alder and 18th (looking south). Intersection of Alder and 18th (looking south) with signal phases for bike only (left) and car only (right) and the terminal loop detector bottom left (many people on bike wait in the crosswalk not knowing what the bike loop detector symbol is for). (Source: Google Street View, August 2017 (accessed 7/4/2018)).

Figure 4: Intersection of Alder and 18th (looking north). Intersection of Alder and 18th (looking north) with bike-only signal (18th is one-way the other direction at this location). (Source: Google Street View, August 2017 (accessed 7/4/2018)).

4.0 BIKE CONNECT SYSTEM ARCHITECTURE

In this section we will describe the overall system architecture that supports our efforts. There are 3 main components: (1) a Bike Connect box that connects to a signal controller, (2) a Bike Connect device that is a phone running our application, and (3) a cloud service that handles communication between box and device. We will describe the role of each of these 3 components in this section.

THE BIKE CONNECT BOX

At the heart of the box is a Particle Electron ($65), Particle's cellular IoT platform. Briefly, the Electron provides limited cell-service to the box for a reasonable price - $3/month. The Electron supports C++ applications and has a GPIO pin breakout. We used the GPIO pins to connect to and control relays in the box. The relays, in turn, were connected to the signal controller (a McCain 2070 controller, though NEMA controllers would connect in similar fashion) to place a request to the bike-phase at the intersection. The set-up is as follows: [bike connect box] ➔ [DC Isolator] ➔ [C-1 pin] ➔ [Controller]. The DC Isolator allows the output signal from our box to interact with the controller hardware.

The cellular connection is to the Particle Cloud, which we will discuss shortly. Further details on the box, including how to build a box and program it, are found in the Project Phenom report.

THE BIKE CONNECT DEVICE

We chose to focus on the Apple iPhone as the first device we piloted. All iPhone models since the iPhone 4 come with an integrated GPS receiver. The model 7 used for testing has the following sensors: touch ID fingerprint sensor, barometer, three-axis gyroscope, accelerometer, proximity sensor, and ambient light sensor. 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. Figure 6 shows the phone on the handlebars. However, note that the phone does not have to be mounted on the handlebars to operate. It will also work effectively in a pocket or bag and will run in both foreground and background mode.

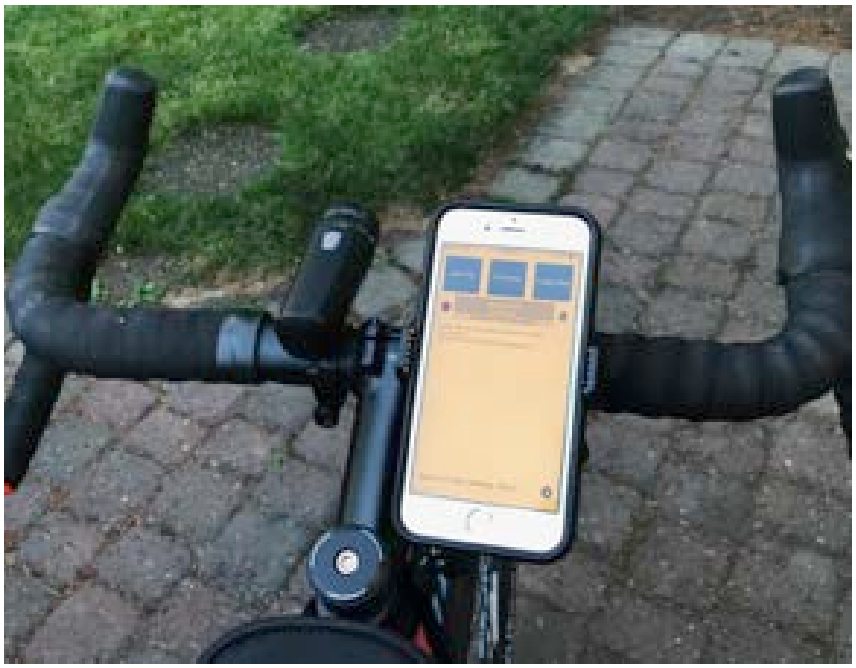

Figure 5: IPhone on handlebar

One of the sensors we rely on is GPS. We need to determine the distance of the user from the intersection to do effective timing of the request for a green. The criticality of accurate GPS readings motivated the project team to focus on improving the raw readings we obtained from the GPS chip.

THE BIKE CONNECT SOFTWARE

The iPhone app was written in Swift. Apple's Xcode 9 was used as the integrated development environment (IDE). Apple provides a full software suite for developing iOS applications, which includes the code editor, compiler, and a simulator to test the device. In addition, Xcode 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. Figure 7 gives an example of the CPU and Disk analysis gauges.

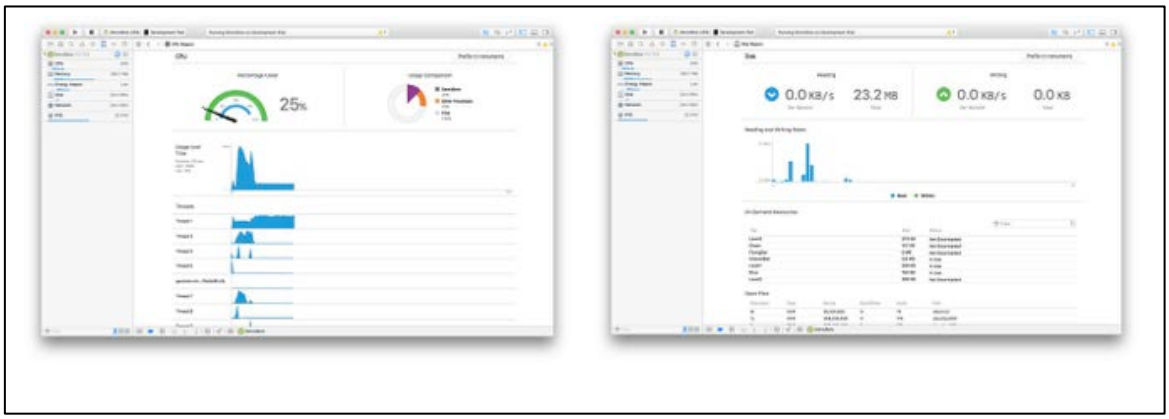

Figure 6: CPU gauge (left) and Disk gauge (right)

CLOUD SERVICES

We chose to separate the box and device communication through a middleman, i.e., a cloud service. This followed from our initial (and unsuccessful) attempts of using P2P communication (i.e., direct device to box communication) through Bluetooth and WiFi. The upside to the P2P approach is that neither box nor device requires Internet access. And in particular, the phone does not require a data plan. However, there are also downsides:

- We need Internet communication with the box to monitor its operation and make changes to its software. No Internet (neither wired nor WiFi) was available at the box location. We found cellular was needed.

- Bluetooth is advertised as having a 100-meter range. However our tests showed the range to vary widely from 10 meters to 50 meters.

- The Bluetooth “handshake” of establishing secure communication between box and bike also varied from milliseconds to seconds and sometimes failed entirely.

- We had similar issues with a WiFi handshake: establishing secure connections between the box (running a WiFi access point) and the device (a client) varied in time.

- Even with secure communications, we were reluctant to place the box in an open P2P environment.

For all of these reasons we chose to have the iPhone device use its own cell service to communicate with the box through a cloud service. However, we have not lost the goal of allowing bike riders without a cell plan to participate. During the grant period, we experimented with building a bike-computer from basic and inexpensive parts. In particular, we explored a bike computer with a Particle Electron at its core. This does not remove the need for a cell plan, but reduces it to $36/year. Figure 8 shows a prototype that we were pleased with.

Figure 7: Prototype Bike Computer in a box and stuffed into small bike bag.

Given that both box and device in our trials had persistent cloud connections, we next considered a means for them to communicate. The Particle Cloud implements what is called a publish/subscribe (or pub/sub) service. For our past IoT projects, we have found that a pub/sub architecture greatly simplifies the actual application code that needs to be written. And simpler code leads to fewer bugs and is easier for humans to understand.

The pub/sub service of the Particle Cloud works as follows. A set of functions are defined as cloud-accessible on the Bike Connect box, i.e., C++ functions are written on the Particle Electron and marked as cloud-accessible. In essence, these functions “subscribe” to (listen for) events. The Bike Connect device, i.e., the Swift code running on the iPhone, publishes events. An event in this case is a request for a green light. The Particle Cloud connects the publisher (Swift code requesting a green) with the subscriber (C++ function that closes the relay). Another subscriber that is built-in to the Electron operating system is an “update application software”. This allows the system developers to change (publish) the application software on the Electron over the air, also known as over-the-air programming. Given that the box is up on the signal pole, this is a huge advantage.

The pub/sub service also scales. Any number of boxes (signals) can be subscribers. Any number of devices (bikes) can be publishers. For example, a bicyclist traveling on a signalized corridor could publish to each signal box as it becomes next in line. The Particle Cloud handles the multiplexing of messages (see Figure 9).

Finally, the pub/sub service allows more than devices to subscribe. We were able to write a logging subscriber that hears all published messages and logs them in a separate database we maintain. These log files can be processed offline and do not require any special logging code to be built-in to either the box’s C++ software or the Swift device software.

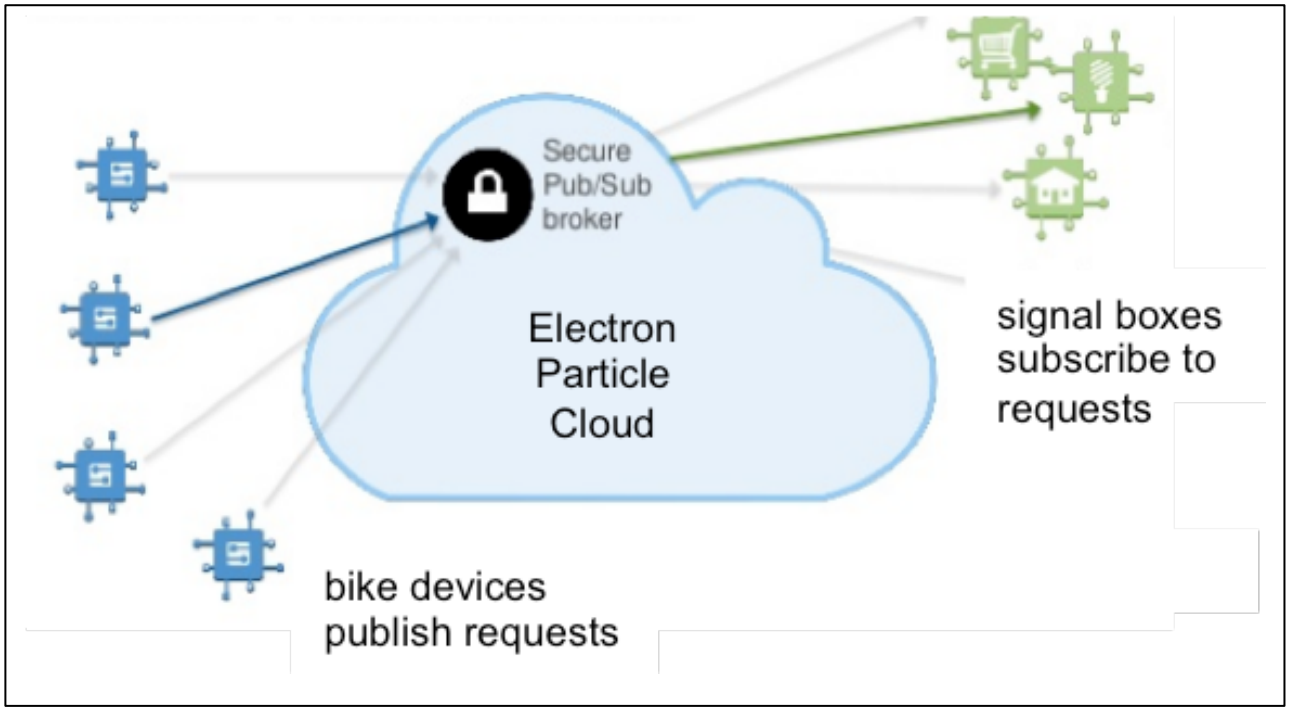

Figure 8: The Electron Particle Cloud

A critical function of our system is to determine, in real time, the distance traveled by a user. With this information we can calculate both speed and when the user enters and exits the triggering circle around an intersection. Our prime interest is in biking and walking, both slow-speed modes of travel when compared to motor-vehicles. Our initial testing results showed that calculating distance for slow-speed travel was a challenge. We decided to see if we could improve slow-speed accuracy as part of this project. The following sections layout the problem and our solution.

CALIBRATING GPS

The Global Positioning System (GPS) was developed and launched by the United States government in the 1970s for use by the military. In the 1980s it was made available for civilian use. The goal of GPS is to allow a device to accurately locate itself from anywhere on Earth at any time. To achieve this goal, GPS consists of a series of 24 satellites orbiting the Earth in a pattern such that at any time and from any location at least 4 satellites will be electronically visible (Hofmann-Wellenhof, Lichtenegger, & Collins, 2012, p. 4). Each GPS satellite continuously broadcast a signal with information about its location as well as its current clock time. GPS receivers listen to this broadcast and through calculating the current location of the satellites via propagation delay and triangulation, its location relative to the satellites can be calculated. This location can then be transformed into a latitude and longitude coordinate on Earth. There is inherent error in the location calculation, but under optimal conditions the location calculated will be within 5 meters of the actual location of the receiver.

SOURCES OF ERROR

There are many sources of error that can occur when transmitting a GPS signal from the satellites to the receivers. Much of this error can be accounted for, such as clock drift of the satellites and signal propagation delay in the upper atmosphere (Grewal, Weill, & Andrews, 2007, p. 103-130). To account for the clock drift, the satellites will periodically synchronize their clock with a common clock. The small drift between synchronizations will have a negligible effect on the overall location calculation. The signal propagation delay through the upper atmosphere can be accounted for by calculating the typical signal propagation delay based on the position of the sun relative to the rotation of the Earth. Given that conditions in the upper atmosphere rarely change unpredictably, this delay can be calculated with high accuracy.

There are some sources of error that cannot be accounted for in the location calculation. This error is the cause of the 5 meters of uncertainty in the location calculation, even under optimal conditions. The primary sources of this error are signal propagation in the lower atmosphere and signal multipath delay. Signal propagation delay in the lower atmosphere occurs when the signal is slowed by bumping into particles in the air such as water particles in clouds. As the weather changes so does the signal propagation delay making it impossible to account for. The largest source of error comes from multipath delay. Multipath delay occurs when the signal bounces off objects before arriving at the receiver. As it is impossible to determine how many times the signal has bounced and off what types of objects, there is no way to calculate the delay. Multipath delay is most noticeable when the receiver is near a building, leading to a large variation in the location calculation between readings. The error will typically be much larger than the 5 meters of error as is typical under optimal conditions.

PREVIOUS WORK

There has been much work in improving the accuracy of GPS location calculation while walking using both GPS and an Internal Navigation System (INS) (Cho, Mun, Lee, Kaiser, & Gerla, 2010; Eliasson, 2014; Godha, Lachapelle, & Cannon, 2006). This was accomplished by attaching an INS device to a person, often on their foot, and using a Kalman filter combining the data from the INS device with the GPS coordinate to better determine the actual position of the person. The purpose of the INS device was to determine when the person was taking a step. With previous knowledge of the test subjects stride length, the GPS coordinate received could be more accurately calculated.

Another study tested the use of a Kalman filter to increase the accuracy of GPS coordinate measurements (Eliasson, 2014). A Kalman filter is an algorithm used to better estimate the actual value of data that contains inaccuracies. It does this by looking at the measurements over time to make a prediction of the current state. In the case of GPS, the current state is the location of the receiver. The Kalman filter averages the predicted state and the measured state to estimate a more accurate location measurement. The average is weighted based on the known noise level of the reading. In the case of GPS, this noise level is calculated by the receiver for each location measurement. In this study, Eliasson compared the overall distance measured using the Kalman filter against using an averaging method, similar to the algorithm described in section 2.3. They found that there was little difference between the averaging method and the Kalman filter.

In the remaining sections we will describe our own research into dealing with GPS error to obtain an accurate value for distance traveled over time.

DISTANCE CALCULATION METHODS

Four methods of calculating distance from GPS data were evaluated in this study. A key property of each method is their ability to update the distance calculation in real time along with the ability to track a wide variety of activity. The intended use of the distance calculation is to calculate distance of low speed activity such as waking, biking, jogging, etc. The four methods chosen were: (1) calculating distance between raw GPS readings, (2) calculating distance based on the speed value provided by the GPS receiver along with the time between GPS updates, (3) averaging GPS readings together into a single point, and (4) taking the line of best fit of a collection of GPS readings and calculating the distance along that line. Each of these methods will be explained in detail in the following sections.

RAW GPS DATA

The first method evaluated was calculating distance from raw GPS data. Raw GPS data is the latitude and longitude value determined by the GPS system on a device with no filtering or modification of this value. The total distance is calculated by calculating the distance between consecutive GPS coordinates, which is calculated using the Haversine formula for measuring the distance between two points on a sphere (Chopde & Nichat, 2013). See Equation 1.

(φ_1, γ_1) and (φ_2, γ_2). Where r is the radius of the Earth.

$$d(\varphi_1\, \gamma_1\, \varphi_2\, \gamma_2) = 2r\ arcsin\left(\sqrt{sin^2\left(\frac{\varphi_2 - \varphi_1}{2}\right) + \cos(\varphi_1)\, cos(\varphi_2) sin^2\left(\frac{\gamma_2 - \gamma_1}{2}\right)}\right)$$

Equation 1: Haversine formula. Distance d is a function of two latitude and longitude coordinates

This method is called every time the system receives a GPS update from the system. The distance and prev_time variables are persistent between calls to this method.

The raw GPS distance calculation method was selected because it is a good baseline of comparison for the other methods. The raw GPS data distance calculation does not factor in the natural error of GPS readings discussed later in this section. It would be expected that the raw GPS data method will over-estimate the total distance traveled. This can be illustrated by considering the distance traveled between two points A and B. One path is a straight line between A and B and the other path is a zig-zag line between A and B. If the total distance of each path is calculated, the zig-zag path will be longer than the straight-line path. To relate this back to the raw GPS data distance calculation method, the straight line would be the actual path taken and the zig-zag path would be the path calculated due to the error in the GPS readings. See Algorithm 1 for pseudocode of the raw GPS data distance calculation.

```
UpdateDistance (lat, lon):
   if prev_lat != null && prev_lon != null:
      distance += haversine(prev_lat, prev_lon, lat, lon)
   prev_lat = lat
   prev_lon = lon
```

Algorithm 1: Pseudocode for distance calculation using raw GPS data

This method is called every time the system receives a GPS update from the system. The distance, prev_lat, *and* prev_lon *variables are persistent between calls to this method. The* haversine *function is defined by Equation 1.*

The next method evaluated was calculating distance from speed. The Google Location API (Google, 2018) exposes a speed value along with the latitude and longitude coordinate for each location update. The speed value attempts to capture the velocity the device is moving at the time of the location update. The speed value is calculated by the GPS receiver by calculating the Doppler shift of the satellite signals received. The Doppler shift can be calculated by comparing

the frequency received from each satellite against the expected frequency. With the calculated Doppler shift value, a velocity can be determined. (Hofmann-Wellenhof, et al., 2012, p. 6). Given that only the total distance of an activity is calculated and not the path taken, the velocity value alone can be used to calculate distance between two points. The distance traveled since the last location updated is calculated by dividing the speed at the current update by the time that has passed since the previous update. See Algorithm 2 for pseudocode of the GPS speed distance calculation.

```
UpdateDistance (speed):
   if prev_time != null:
      distance += speed / (Time.Now - prev_time)
   prev_time = Time.Now
```

Algorithm 2: Pseudocode for distance calculation using GPS speed data

AVERAGING GPS DATA

The next method evaluated was calculating distance by averaging a set of raw GPS coordinates. This method works by collecting a set of *n* coordinates and averaging them to form a single point. Next, another set of *n* coordinates are collected and averaged. The distance between the two averaged locations is added to the overall distance. The distance between the two coordinates is calculated using Equation 1. See Algorithm 3 for pseudocode of the GPS averaging distance calculation. This algorithm is similar to the algorithm in Eliasson, 2014. This method was chosen because it is more resilient to the error of GPS coordinate readings. The assumption is that the error is distributed evenly among a set of GPS readings in relation to the actual location of the device. Given this assumption it can be concluded that the average of a set of points will be closer to the actual location than an individual reading. However, there is a tradeoff between having a large and small *n* value. Larger *n* values will reveal a coordinate closer to the actual location but will have a longer delay in calculating the distance value due to the overhead of collecting all *n* coordinates before calculating the distance. Due to this delay, the distance calculated may cut out large portions of actual distance traveled if the actual path taken is not in a straight line. Smaller *n* values will result in a less accurate location value in relation to the actual location of the receiver, but will calculate the distance value more frequently. The higher frequency calculation will result in less smoothing of curved paths.

```
UpdateDistance (lat, lon, set_size):
   coord_list.Add(vector(lat, lon))

   if coord_list.Size == set_size:
      avg_lat = 0
      avg_lon = 0

      for point in coord_list:
         avg_lat += point.x
         avg_lon += point.y
      avg_lat /= set_size
      avg_lon /= set_size

      if prev_lat != null && prev_lon != null:
         distance += haversine(prev_lat, prev_lon, avg_lat, avg_lon)

      prev_lat = avg_lat
      prev_lon = avg_lon
      coord_list.Clear()
```

Algorithm 3: Pseudocode for distance calculation using the average location of a set of GPS coordinates. This method is called every time the system receives a GPS update from the system. The distance, prev_lat, prev_lon, and coord_list variables are persistent between calls to this method. The haversine function is defined by Equation 1.

LINE OF BEST FIT

The final method evaluated was calculating distance by looking at the line of best fit for a set of points. This method works by collecting a set of *n* coordinates and calculating the line of best fit for those coordinates. After that the estimated length traveled along that line is calculated and added to the overall distance. The line of best fit is calculated using the least square method. The slope of this line is turned into a normalized directional vector. Using the normalized directional vector of the line of best fit, a scalar project is performed with the vector between two consecutive points to determine the length traveled along the line of best fit. The Haversine formula from Equation 1 is used to calculate the distance along the line segment. The sum of these distances calculates the total distance traveled for that set of n coordinates. See Figure 10 for a visualization of the calculation. See Algorithm 4 for pseudocode of the method.

```
UpdateDistance (lat, lon, set_size)
   coord_list.add(vector(lat, lon))

   if coord_list.size == set_size:
      avg = vector(0,0)

      for point in coord_list:
         avg += point
      avg /= set_size

      rise = 0
      run = 0

      for point in coord_list:
         rise += (point.x – avg.x) * (point.y – avg.y)
         run += (point.x – avg.x) ** 2

      best_fit = vector(rise, run).normalized
      last_point = coord_list[0]

      for i = 1 to set_size:
         // Vector projection
         p = dot(best_fit, coord_list[i]-last_point)
         // Distance along the project line segment
         distance += haversize(last_point, last_point + (best_fit*p))
         last_point = coord_list[i]

      coord_list.clear()
      // The last point is added to the list to not lose distance between sets
      // of points
     coord_list.add(last_point)
```

Algorithm 4: Pseudocode for distance calculation using the line of best fit of a set of GPS coordinates

This method is called every time the system receives a GPS update from the system. The distance and coord_list variables are persistent between calls to this method. The haversine function is defined by Equation 1. The dot function is a vector dot product.

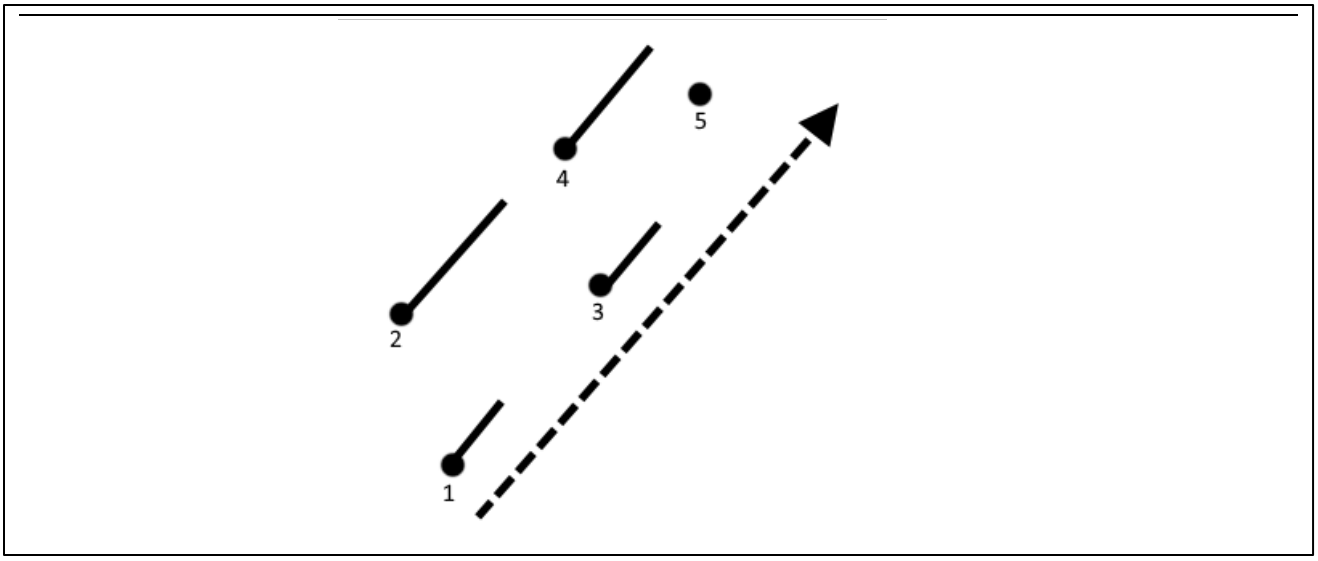

Figure 10: Visualization of the line of best fit distance calculation method. The numbers indicate order the points were received. Note that the points are GPS coordinate readings not the actual location of the receiver. The dashed line shows the directional vector of the line of best fit. The solid lines show the distance calculated.

This method was chosen because, like the averaging method, it is more resilient to GPS error. The difference between this method and the averaging method is that this method attempts to estimate the actual path taken by the receiver rather than estimating the position of the receiver. Like the averaging method there is a tradeoff for using a larger and smaller *n* value. Larger *n* values will result in a more accurate line of best fit for activity moving in a relatively straight line. However, if the path of the activity is along a curved path, the line of best fit will smooth the calculated path taken, resulting in a smaller distance calculation than what occurred in reality. Smaller *n* values will result in a less accurate line of best fit but due to the more frequent updates will not smooth the calculated path taken as much as larger *n* values.

EXPERIMENTAL TESTING

Each method described was implemented and tested on a variety of activities. For both the averaging method and line of best-fit method, an *n* value of 5 and 10 were tested. The three activities chosen to test were biking, walking, and idle. Six devices were used in total. The GPS service on each device was configured to receive a location update approximately once every second. For each activity tested, four trials were conducted. All six devices were set to collect GPS data for each trial resulting in 24 data points per activity. The following pieces of data were collected on each device per trial: total distance calculated for each distance calculation method, average speed of the device, and maximum speed of the device. All GPS updates for each trial were saved to the device to enable playback of the GPS data at a later time. All trials were conducted on days with clear skies; optimal weather conditions for GPS data collection.

BIKING TEST

Biking was chosen because it is one of the highest speed non-motor assisted activities commonly performed. The biking trials were performed on a biking trail rather than a street to ensure consistent trials. To simulate biking on the street, where stopping at intersections is common, there was a 30 second stop for every mile of riding. See Figure 11 for an approximate mapping of the path taken. Two trials were riding East to West and two trials were riding West to East. The exact same path was followed for all four trials. The total actual distance of each trial was 4.12 miles.

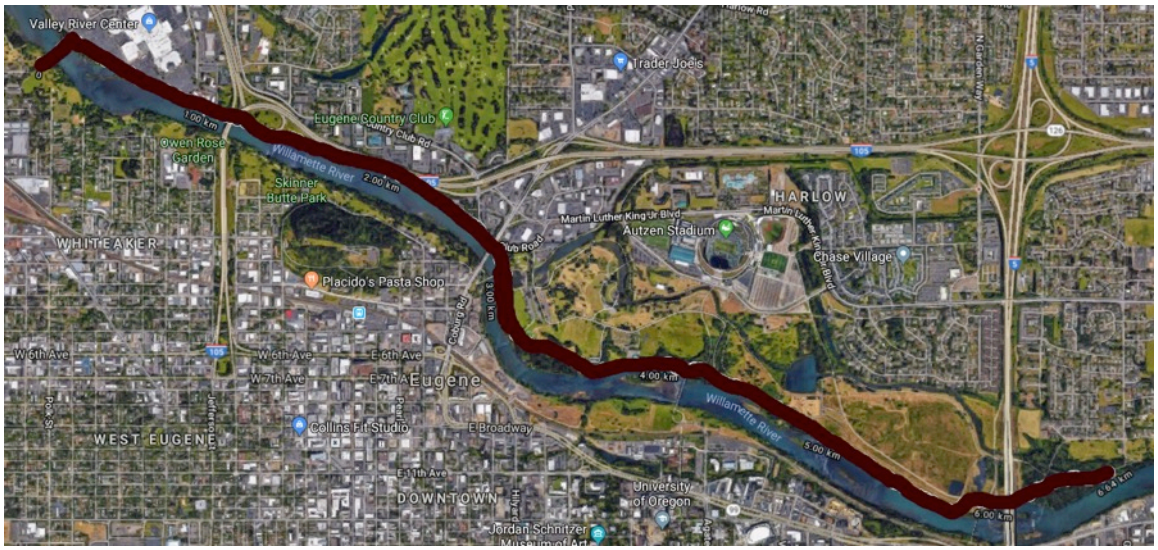

Figure 11: Path taken for the biking trials mapped via Google Maps

WALKING TEST

Walking was chosen because it is very low speed and often follows a path that has abrupt changes in direction. The walking trials were performed on a residential street. See Figure 12 for an approximate mapping of the path taken. All four trials followed an identical path. The total actual distance of each trial was 1.0 miles. Each trial was performed continuously without any artificial stops.

Note that the bike route is different than the walking route. We chose each to match the reality of our test intersection. Our observation was that bicyclists followed roughly straight paths that correspond to the bike lanes and street corridors that surround our test intersection. Pedestrians, on the other hand, were observed to follow a variety of paths, and in particular, a convoluted path through campus near the test intersection.

IDLE TEST

An idle test was performed to test the accuracy of the distance calculation with no movement. Many times, activity such as biking or walking will have brief moments of stopping, e.g., stop lights, cross walks, etc. It would harm the accuracy of the overall distance calculation to have these moments of stoppage increase the overall distance calculated. Given the natural error of GPS readings it would be expected that many of the methods tested would erroneously calculate non-zero distance while the device is stationary. The idle test was performed by placing the devices on a table outdoors and collecting GPS data for 10 minutes.

Figure 9: Path taken for the walking trials mapped via Google Maps

BIKING RESULTS

See Figure 13 for the distance calculation results for the biking trials. Bike riders were asked to pedal as they normally would on a 4-mile commute. The biking trials resulted in an overall average speed of 4.409 meters per second (roughly 9 MPH). The results show that the averaging method with an *n* value of both 5 and 10 are extremely accurate in the distance calculation with low variance. The line of best fit and raw GPS data methods both tend to overestimate the total distance but also have low variance. The speed method on average vastly underestimates the total distance and has a high variance and is an outlier compared to the other methods. Our first conjecture was that the speed value is fully dependent of the GPS location readings, and hence, is noisy in concordance with noisy location information. However, using raw GPS data was

accurate. It remains unclear to us why obtaining accurate speed data is so error-prone. Nevertheless, we did find a way to combine raw GPS and speed in an effective way, which we will discuss shortly. Given time, we would have liked to have built and explored our own speed function from just the raw accelerometer data (Seifert&Camacho, 2007).

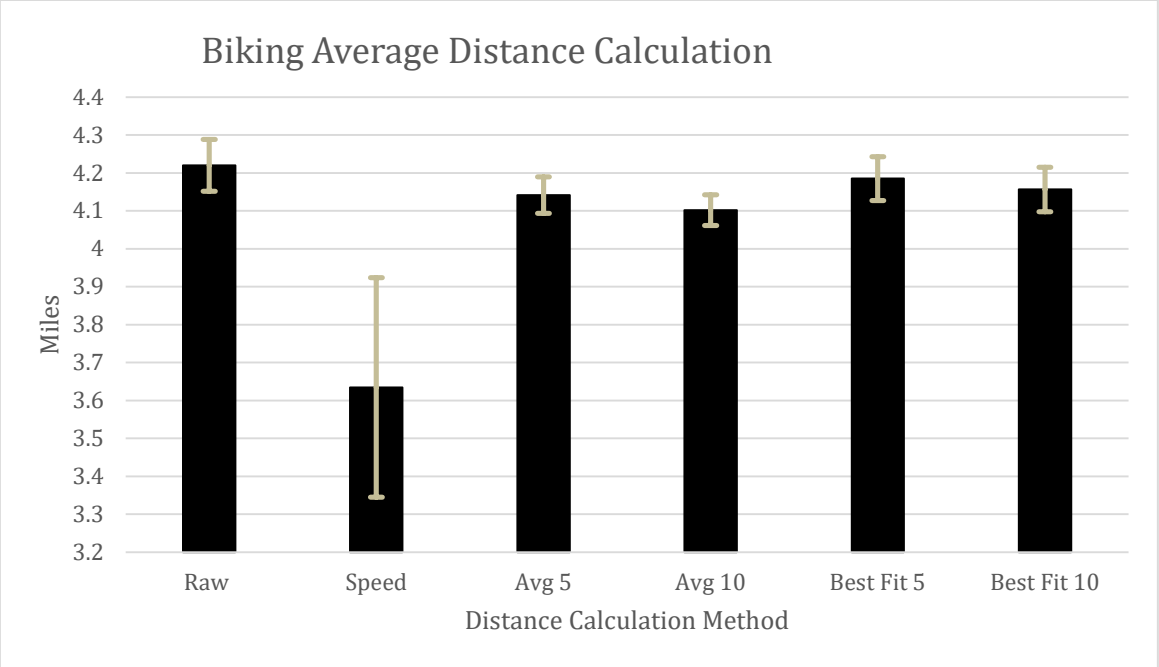

Figure 13: Results of the biking distance calculations. The bars in black represent the average distance calculated. The gray bars represent the standard deviation. Actual distance traveled was 4.12 miles.

WALKING RESULTS

For the walking trials, walkers were asked to choose their walking speed as they would use on a 1-mile trip. See Figure 14 for the distance calculation results for the walking trials. The walking trials resulted in an average speed of 1.483 meters per second. The results show that the raw GPS data method is the most accurate with low variance. The averaging and line of best fit methods both underestimate the total distance traveled with low variance. The line of best fit method with both *n* value of 5 and 10 is slightly more accurate than the averaging method. Like the biking trials, the speed method underestimates the total distance with a high variance.

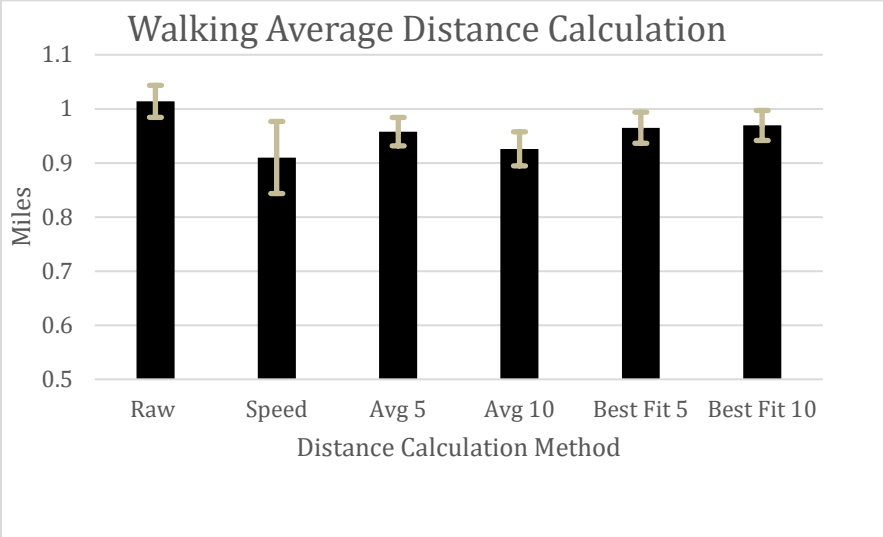

Figure 14: Results of the walking distance calculation. The bars in black represent the average distance calculated. The gray bars represent the standard deviation.

COMPARISON OF BIKING AND WALKING

To easily have a visual comparison of the methods, the average percent error of each method was calculated, see Figure 15. Based on this comparison it is clear that the speed method does a very poor job at estimating the distance traveled for each activity. There is a large difference in error with both the averaging and line of best fit methods between biking and walking. The most likely explanation of this is with the type of paths taken for each of the trials. The biking trials followed a relatively straight path while the walking trials followed a path that had many curves and corners. These corners were most likely cut off by the smoothing of the averaging and line of best fit methods resulting is an underestimate of the total distance. For the raw GPS method there was very little difference in the percentage of error between walking and biking. This further supports the idea that the smoothing of the averaging and line of best fit methods result in an underestimate of distance for winding paths.

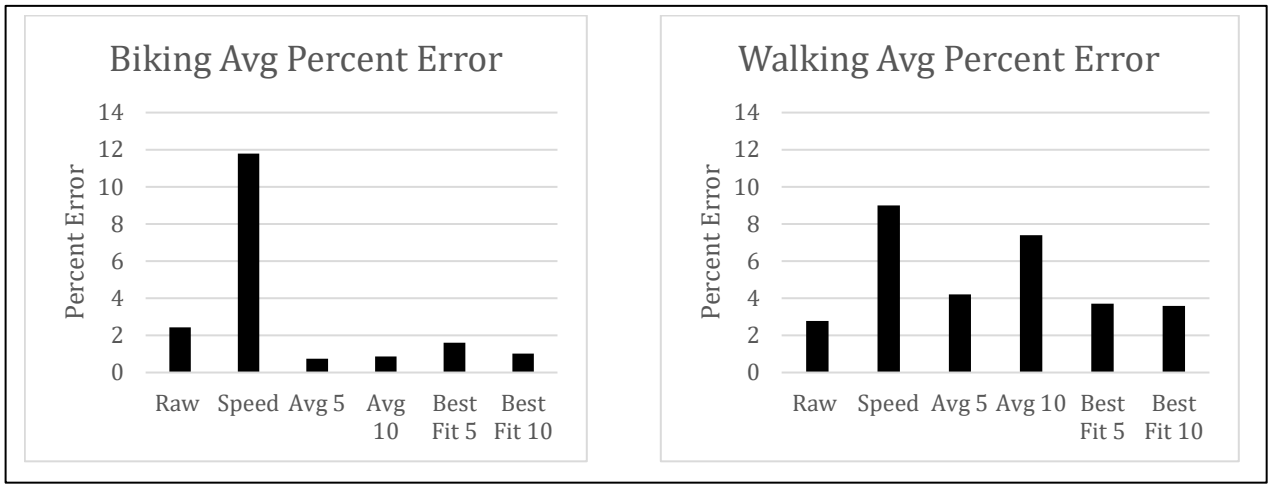

Figure 15: Percent error of the total distance traveled of biking and walking

IDLE RESULTS

See Figure 16 for the distance calculation results for the idle trials. As expected, the methods that rely on GPS coordinates generated a significant amount of distance when it actually should be 0. This is likely due to the natural error in the GPS readings from the device. More interestingly, the speed method is the only accurate method for calculating distance while not moving. While the speed may be highly inaccurate at low speed, it is highly accurate at no speed.

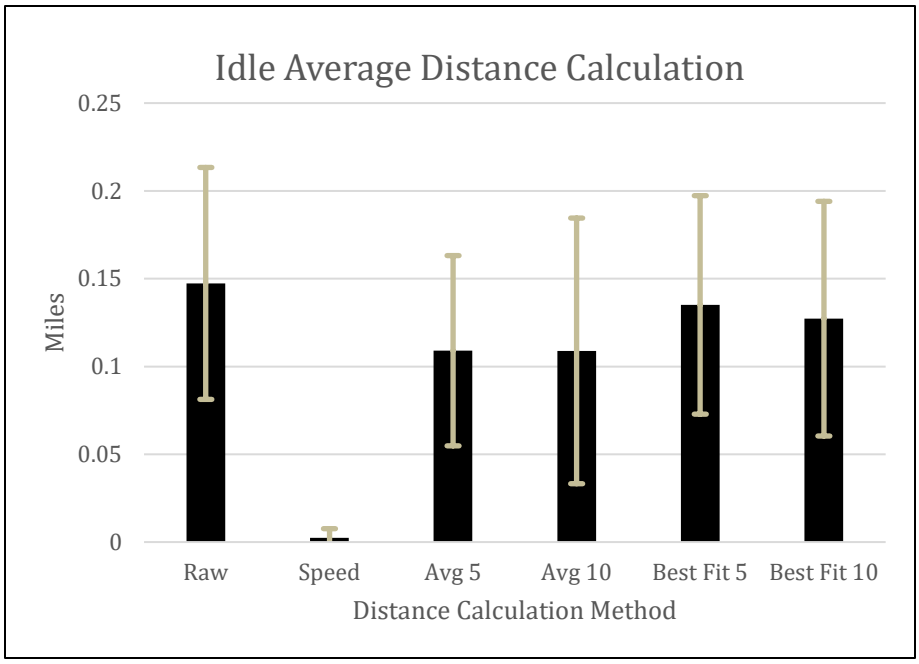

Figure 16: Results of the idle distance calculation. The bars in black represent the average distance calculated. The gray bars represent the standard deviation.

A NEW APPROACH: GPS DATA FILTER

Looking at the results from the previous sections, there is no single best method that accurately calculates distance for all three activities: biking, walking, and idle. The raw GPS data, averaging, and line of best fit methods do a reasonably good job calculating the distance while in motion but do a poor job while idle. On the other hand, the speed method does a poor job calculating distance while in motion but does a great job calculating distance while idle. Given this dichotomy, we propose a filter that retains the accurate distance calculation of the non-speed methods when in motion while filtering out data when idle. The calculation methods will remain the same, but if the speed reading is below a certain threshold, the GPS update will be filtered out. The threshold value was determined by looking at the average speed value calculated from the idle trials. From these trials the threshold value chosen for the GPS data filter was 0.6 meters per second (1.3 MPH). If any speed is below 0.6 meters per second the update will not be considered in the overall distance calculation.

DATA FILTER RESULTS

The distance calculation with the GPS data filter was evaluated on the same data that was collected from the original experimentation using the GPS data playback explained in previous sections. See Figure 18 for the distance calculation results with the GPS data filter. The most important result is that all distance calculation methods had nearly zero distance calculated for the idle test. To better view how the distance calculation changed with and without the GPS data filter the percent improvement was calculated for each distance calculation method with the filter and without the filter: See Figure 17.

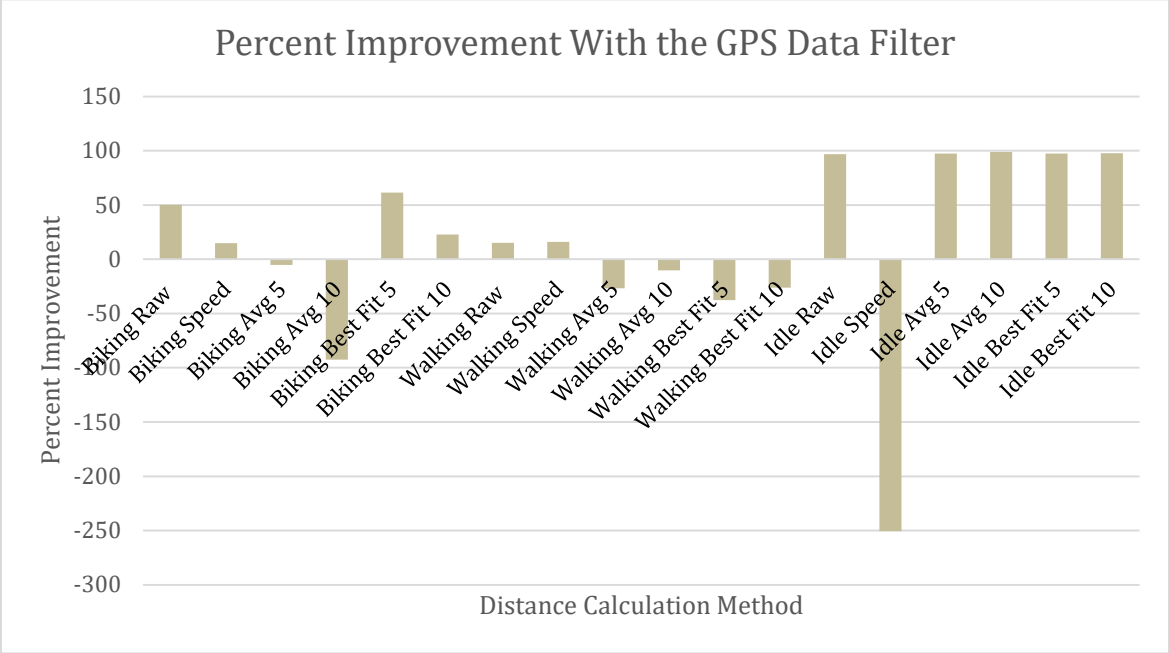

Figure 10: Percent improvement of the accuracy of the distance calculation with the filter over the distance calculation without the filter

The filter showed improvement for the idle trials for every method except the speed method. However, since we are only using the speed value as a threshold, this is not an issue.

The averaging method saw a decrease in accuracy for both biking and walking. This implies that the averaging method underestimates the overall distance. For the biking trial, the reason the averaging method without the filter was so accurate was because the distance while stopped was being added to the overall distance. Had those stops been longer the averaging method would begin to overestimate the distance. With the filter this error was filtered out leaving the total distance calculated shorter than without the filter, resulting in a decrease in accuracy.

The line of best fit method saw in increase in accuracy for biking but a decrease in accuracy for walking. The line of best fit method with both an n value of 5 and 10 overestimated the distance without the filter. Filtering out the error generated by the stops resulted in a distance extremely close to the actual distance traveled. The line of best fit method with an n value of 5 had an average error of 0.0256 miles for a 4.12 mile route. However, there was a decrease in accuracy in the walking trials. Our hypothesis is that we may not be able to use just a single speed value threshold for all activities. If a person is biking then .6 meters per second looks good. If they are walking, another value may be needed. We discuss this further in section 6 on Future Work.

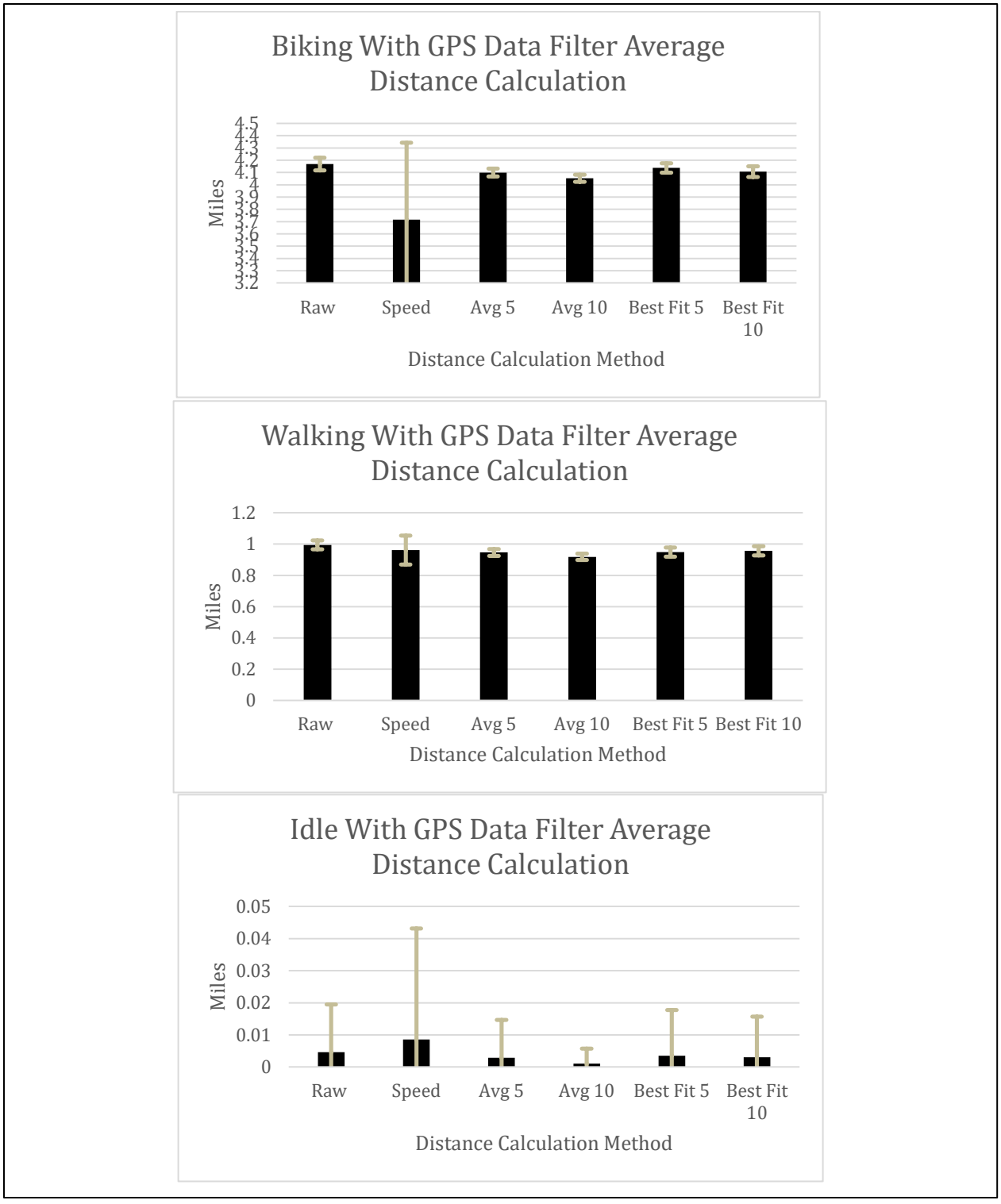

Figure 11: Results of the distance calculation with the GPS data filter for (a) biking, (b) walking, and (c) idle

The raw GPS data method saw an increase in accuracy for both the biking and walking trials. The raw GPS data method does not suffer from the smoothing effects that the averaging and line of best fit methods do. The filtering for the biking trials filtered out the updates while stopped. This decreased the total distance traveled. Without the filter this method overestimated the overall distance traveled. With the filter the overall distance calculation still overestimated, but not by as much. Because the raw GPS data method does not suffer from the smoothing effects like the averaging and line of best fit methods do, the walking distance calculation is highly accurate, with an average error of 0.024 miles for a 1 mile route.

We chose to use the raw method filtered at .6 meters per second as the best compromise between biking and walking and used that method in the trials we discuss next.

5.0 PROJECT RESULTS

Our first Bike Connect box has been operational since May 15th, 2018 at the intersection of 18th and Alder in Eugene, Oregon. Figure 19 shows the major grouping of phases into 4 classes: (a) phases for cars and peds traveling E/W, (b) phase for car traveling South, (c) phase for bikes traveling N/S, and (d) phase for pedestrians traveling N/S. The current form of our box is capable of requesting greens for 4 separate phases. For this project, the box is connected solely to the bike-phase that has its own bike-light (panel c).

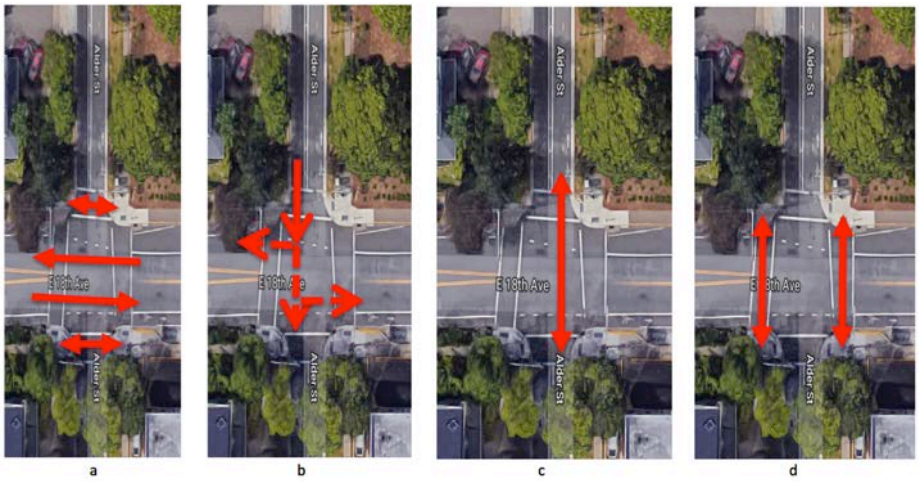

Figure 19: The phases at 18th and Alder

Because this project involved a 3rd party (us) installing a control system onto an existing municipal traffic signal, there was a concern with having our system create unintentional issues with non-bike phases, e.g., erroneously place calls on the bike phase when no bike is present and hence delay other phases unintentionally. To mitigate this concern, we gave the City a virtual 'kill' switch, which they could activate from their own computers. When activated, our system will go into a sleep state. This was tested to the City's satisfaction. To date, the City has had no need to use the switch.

The app was distributed to 10 testers who regularly ride the Alder Street corridor. We asked them to record (in a notebook) on each of 6 round trips (12 samples): (a) whether they received a green bar, (b) where the green bar appeared in relation to the existing in-pavement advanced loop (before reaching the loop, at the loop, after reaching the loop) and (c) whether they were forced to stop to wait for a green. We instructed them to ride at their normal pace. The system worked as intended with the following basic results taken from the 120 samples:

- The app's simple, color-coded indicators worked as expected: they allowed the cyclist to see (via a handle-bar mounted phone holder) the status of communication between app and traffic signal and whether a virtual call had been sent and acknowledged.

- When pedaling at their preferred speed, riders obtained a green bar before reaching the in-pavement advanced loop. We suspect, and have reported to the City Traffic staff, that the advanced loop placement is based on an average speed that is below what is actually seen along this corridor.

- In terms of the cyclist receiving a green light upon arrival, 74% of the time (89 samples) the rider saw a green bar and was able to proceed through the intersection uninterrupted. In 26% of cases (31 samples), the rider saw a green bar but was forced to stop. The feedback from riders was that they had missed a green light and were forced to wait for other phases to cycle back to the bike phase. We hypothesize that if we had more data about the signal, we could guide the rider in adjusting his or her pedaling speed to avoid a stop. For instance, if we knew the current active phase and its start time, and what follow-on phases were queued up, we could predict the time in the future when the bike phase will become active again. Using the rider's speed and distance, we could then suggest a change in speed that will get the rider to the intersection seeing a green. We are exploring this idea further in a separate NITC project. At the time of this project, we had no access to this type of controller data.

Our results demonstrate that purposefully designing systems to enhance the efficiency and comfort of people on bike via connected infrastructure communication is possible, can be done by easily retrofitting existing traffic system control systems, and can be done at relatively low cost. This is a much different approach toward most smart cities and connected and autonomous vehicle discussions where bikes are seen as objects to 'see' and avoid rather than modes of transport that should be planned deliberately for.

6.0 FUTURE WORK

To further refine the distance calculation, a fruitful possibility is changing the distance calculation methods based on what activity is being performed. Our results show that different calculation methods produce more accurate results depending on the activity being done. The accelerometer could be used to predict what activity is currently being performed and adjust the distance calculation method to choose the most accurate version for that activity as explained in (Ravi, Dandekar, Mysore, & Littman, 2005).

We chose a GPS data filter threshold by observation. This gave us satisfactory results for our trials, but also questions about whether the value will hold on different trial sites. More generally, there is work to be done on scaling our approach to (a) non-university sites, and (b) different hardware/software mixes than the iOS/Google mix that we chose. Most recently we have begun to study intersections close to Elementary and Middle Schools in Springfield Oregon. Our goal is to have our users be kids riding to and from school. The Springfield sites have a much broader range traffic flow than our 18th and Alder site. And the hardware/software mix is now Android/Google. We have just begun to establish testing routes in this new environment.

We noted that at the time of this project we had no access to controller information in real-time, e.g., what is the currently active phase, how long has it been active. However, during fall of 2018 the Eugene Transportation Office purchased a license from McCain Systems to obtain a limited real-time view of many of the signals in Eugene, including ours at 18th and Alder. The information we can obtain, in real-time, is the current active phase and when it became active. What we are missing from McCain is a view of what phases are queued. Nevertheless, working with the active-phase information, we believe we can improve the request-timing of our app. We are taking on this challenge as part of a separate but related NITC project, *Fast Track: Allowing bikes and pedestrians to participate in a smart-transportation system* (#1160).

This project is part of a larger goal to create and test a comprehensive, low-cost, ubiquitous system that brings cycling into the smart and connected communities' framework, focusing in four key areas: (1) Develop transportation scenarios for the challenges that children face when biking to school and evaluate those scenarios in a bicycle simulation lab. (2) Develop technology that will address those challenges by tapping into the larger smart-transportation infrastructure through an inexpensive active transportation device (Bike Connect) and, at the same time, extend that infrastructure to open up a new world of bike-friendly features (through the Bike Connect Device). (3) Tie scenarios, simulation and technology development together in a new agile framework especially suited for domains where field tests are problematic. (4) Demonstrate the suitability of our approach by focusing on four schools that are diverse both in the student body and the geographical challenges they present (as discussed above). This project addresses task 2. Current and future work is targeted toward the remaining 3 tasks.

7.0 CONCLUSION

We have demonstrated that it is possible to link active transportation modes into a V2X environment. We developed a Bike Connect box that is inexpensive. We then developed an app for a Bike Connect device (iPhone) that uses GPS information to place a request for a green bike-light. We linked box and device up through the Particle Cloud.

We explored the possibility of providing a bike computer that could act as the Bike Connect device, eliminating the need to have a phone with a monthly cell plan. We prototyped a computer that centered on the Particle Electron, the same computer used in our box. While the computer components were relatively inexpensive (less than $150), we were unable to find an inexpensive power supply to meet our needs. However, we remain interested in the idea and will continue to track the availability of an inexpensive power supply. We are also in discussion with JUMP Bikes, which recently opened bike-sharing in Eugene. Our goal is to make their bike computer, powered by a solar panel, a Bike Connect device.

A large portion of our effort focused on the use of GPS to effectively calculate distance traveled with low speed activities. All the methods tested, except for the speed method, yielded consistent results that were acceptably close to the actual distance traveled with some methods being slightly more accurate than others. While the speed was not effective at calculating distance, it was effective at providing a means to filter out data while not in motion. However, it was not found that a single method is clearly the best at calculating distance for all types of activity. Biking saw the line of best-fit method with a *n* value of 5 being the most accurate method while walking saw the raw GPS method as being the most accurate. There are many factors that play into which method will be best for a given situation. These factors are things such as the average speed of the activity, the typical type of path taken (straight lines or winding paths), and the type of surrounding in the area the activity is typically performed in, e.g., if the activity will occur near buildings this can have an impact on the overall distance calculation. Ultimately it comes down to choosing the method that will best suited for the expected use cases of the application.

One additional innovation of the project is that it recognizes that V2X technology does not have to be hidden behind company walls or reserved only for researchers at universities. Instead, developing V2X technology can be made an open project available to anyone, and in particular, students wishing to learn more about the Internet of Things and transportation. To demonstrate, as a separate project we developed a set of video lessons that provide a clear and detailed roadmap on giving students a chance to explore V2X technology and, in the end, produce something that can be used in their own community, i.e., the circled Bike Box in figure 1. This related project can be referenced at Project Phenom (nitc.trec.pdx.edu/research/project/1072). It contains course topics on:

1. Doing electronic (solderless) breadboarding to connect a modern Internet of Things device, a cellular embedded computer, into the heart of their system.
2. Demonstrating how the embedded computer (a particle electron) can easily control relays, which provide the virtual push-button.
3. Packaging up their system into a container that can be placed on a signal pole.
4. Lessons on how to employ C++ and object-oriented programming to control the electron from the cloud.

When completed, a student will have built a powerful V2X system that is ready to be employed at a signal in their community.

8.0 REFERENCES

Cho, D. K., Mun, M., Lee, U., Kaiser, W. J., & Gerla, M. (2010, March). Autogait: A mobile platform that accurately estimates the distance walked. In *Pervasive computing and communications (PerCom), 2010 IEEE international conference on* (pp. 116-124). IEEE.

Chopde, N. R., & Nichat, M. (2013). Landmark based shortest path detection by using A* and Haversine formula. *International Journal of Innovative Research in Computer and Communication Engineering*, *1*(2), 298-302.

Eliasson, M. (2014). A Kalman filter approach to reduce position error for pedestrian applications in areas of bad GPS reception.

Godha, S., Lachapelle, G., & Cannon, M. E. (2006, September). Integrated GPS/INS system for pedestrian navigation in a signal degraded environment. In *ION GNSS*(Vol. 2006).

Google. (2018, June). Android Documentation – Location. Retrieved from https://developer.android.com/reference/android/location/Location.

Grewal, M. S., Weill, L. R., & Andrews, A. P. (2007). Global positioning systems, inertial navigation, and integration.

Hofmann-Wellenhof, B., Lichtenegger, H., & Collins, J. (2012). Global positioning system: theory and practice.

Ravi, N., Dandekar, N., Mysore, P., & Littman, M. L. (2005, July). Activity recognition from accelerometer data. In *Aaai*(Vol. 5, No. 2005, pp. 1541-1546).

Seifert, K. and Camacho, O. (2007) Implementing Positioning Algorithms Using Accelerometers, *Freescale Semiconductor Application Note*, https://www.nxp.com/files-static/sensors/doc/app_note/AN3397.pdf

Yang, Y., Harbaugh, W., & McDonald, N. (2015). *Encouraging Active School Travel by Making it" Cool": A Quasi-Experimental Study Using Boltage* (No. NITC-RR-550).

Transportation Research and Education Center
Portland State University
1900 S.W. Fourth Ave., Suite 175
Portland, OR 97201