Team Instru-MEN-tation

Final Report and User Manual

04/29/22

Team members:

Joshua Garot - jpg262@nau.edu

Ibrahim Aljarbou - ia342@nau.edu

Yaqoub Abdulkarim - yma36@nau.edu

Table of Contents

3. **Introduction**

Client Background:

We will discuss our clients, the challenge we are trying to solve, and a high-level overview of our design. Dr. Chun-Hsing Jun Ho, an Associate Professor in the Department of Civil Engineering, Construction Management, and Environmental Engineering, is the first customer. He was a lead faculty member in the Civil Engineering Graphics program at the Washington Institute of Technology. In 2010, he received his Ph.D. from the University of Utah. Dr. Kyle Nathan Winfree, an Associate Director of Undergraduate Programs at the School of Engineering, Information, and Applied Sciences, is the second client. The University of Delaware awarded him a Ph.D. in Biomechanics and Movement Science. He has several journals and conference proceedings articles. The issue that our clients have presented to us is that we need to modernize all of the previously utilized technology to identify road obstructions. Dr. Ho was helping us with the overall project, and he gave us a bike to test our system. In addition, he gave us some feedback on the progress that we have made. In the end, we will provide him with all the data that have been collected throughout the testing. Dr. Winfree was helping with the technical issue that we faced and any problems that we faced either in the software or the hardware. At the same time, he provided us with the lab so that the team could meet there at any time and use the equipment necessary.

Problem Statement:

Most of our job begins from scratch when we should look for a new component and create it instead of using outdated technology. It is all because technology advances every day, and we need to replace each piece of equipment to better our job and make it more user-friendly for our clients. The overall look of our design is that we incorporated various components. The Arduino Nano, which serves as the project's microcontroller, is the most critical component. In addition, GPS, accelerometer, OpenLog (SD card), and a battery are used. Our entire system will be in a compact box to fit a phone holder. The team will utilize a QWICC cable to link all of the components. The GPS will be used to pinpoint the location of each obstacle, while the accelerometer will help in detection. Everything that the GPS and Accelerometer detect will be saved in the OpenLog. Simultaneously, it will display the date and time on the SD card, allowing us to determine when and where the incident occurred. We will be able to get the date and time using the GPS since it will be connected to the satellite, and the SD card will have a memory space of 32GB.

4. **Design Process**

Design Process Description:

Research is needed to understand how parts of the overall design will work. Before prototyping individual components for our system, we split the research into three main parts so each group member would be responsible for a more significant part of the overall design. Josh is responsible for the GPS chip, Yaqoub is responsible for the accelerometer, and Ibrahim is responsible for the OpenLog. This continues throughout the entire project. Our team started in the Fall semester by researching each of our respective components. Each of us found two sources to write about that would help our team start designing prototypes to help us learn more about our chips with experimentation. After prototyping was completed, we came together to create a plan to develop our project. Starting over winter break, this plan included polishing the prototypes we had made with our new chips that used Sparkfun's QWIIC connection system in order to create a cohesive project.

Functional Decomposition:

GPS

There are many sources on GPS chips, so it was hard to find some that would help our project specifically. The best article was found in [1]. The article starts with a short summary of why the designer chose each specific product for their project and includes four other sections. They decided on a GY-GPS6MV2 GPS chip and the most powerful Arduino chip at this time, the ATmega 2560 [1]. Although the GPS they chose is different from ours, their overall design is similar to what is needed for our project. The system uses a Ublox Neo6M GPS chip, an antenna, a backup battery, an EEPROM memory, a power supply, and a signaling LED that indicates positioning [1]. The second part of the article goes through the Arduino Program to retrieve data from the GPS chip and put it into a readable format.

The first function takes the GPS chips data in an ASCII form and updates the system in a set number of seconds specified by the user [1]. The second function stores the data from the chip on the SD card. This data includes the date, time, Latitude, Longitude, and other data that were not mentioned [1]. This works by writing a .csv file with whatever parameters the user wants, putting this file onto the SD card used in the system, then putting it into the system. The author chose a CSV file because it is one of the easier ways to get Google Maps to recognize data. Our design will be using a .txt file for data storage which will be discussed later in this paper. The third and final function is used to determine if the GPS is physically moving. The way it does this is with an algorithm that accounts for the error in the chip's measurement of distance. The author states that the accuracy of their model is 2.5 meters and explains that they have tested it to be accurate within 2 meters [1]. Because he has tested it to be valid up to 2 meters, that number is also the

parameter used to determine if the GPS is moving or not by checking if the difference between two distances is more significant than 2 meters and adding the new distance to the total. To test the prototypes, the author put the device in the passenger seat of their car and drove around their neighborhood. This helped them test different aspects such as writing coordinates, date, time, and distance [1]. Our team's original GPS chip was made for cars, so this final part of the article was important in understanding how to test our GPS chip prototype. The chip has since been changed to one that is not made for a car.

A second article examines how the GPS frequency reference affects the functions needed to do GPS in systems like the Tomahawk cruise missile and SLAM. The topics covered in this article include temp stability, temp gradient, Allan variance (AVAR), SSB Phs Noise, aging, warmup time, hysteresis, frequency gradient, acceleration sensitivity, frequency perturbations, and voltage sensitivity [2]. These will all be covered through three GPS system parameters: time-to-first-fix performance (TTFF), carrier tracking loop performance, and code tracking loop performance [2]. The main help from this article was learning about all of the different variables that can affect the GPS chip itself or its data. The essential factor understood is that the chip will need some warmup time in order to connect to satellites. TTFF's standards include temperature stability, warmup, aging, hysteresis, and temperature gradient [2]. TTFF is most sensitive to temp stability and hysteresis because extending battery life requires a precise, lightweight GPS receiver to operate on standby mode, causing an issue of reacquisition [2]. The temp stability is a frequency uncertainty because it must be searched to find the correct frequency to track the GPS signal. Carrier tracking loop performance is sensitive to frequency perturbations which can cause the tracking loop to lose lock, but this also depends on the loop type and order [2]. The two-loop types discussed are the Phase Lock Loop (PLL) and Frequency Lock Loop (FLL). At the time of the conference, this article from the author states that the physics of frequency perturbation is not fully understood yet. Hence, their analysis of how it affects the two different loop types is mostly theoretical. Code tracking loop performance is when the tracking loop functions are the only tracking mechanism available due to high jamming conditions [2]. The main parameter affected here is acceleration sensitivity, which has different parameters known to affect performance, are blank size, electrode type, and mounting techniques.

Accelerometer

It was not easy to find relevant articles related to the accelerometer. However, we found the closest articles about the accelerometer. The third article talks more about developing a real-time motion for wearable photoplethysmography to measure heartbeats. They used a 3-axis accelerometer similar to the one we will use for our project because there are many types of 3-axis accelerometers [3]. The old team used the same accelerometer but without directly connecting to the x, y, and z axes. However, the one that my team will use has a direct connection to the x, y, and z axes. Continuing with the article, they used a two-dimensional active noise cancellation algorithm to compensate for the distorted signals by motions using the directional accelerometer data [3]. They used many sensors to measure body motion, such as an

infra-red LED, a photodiode for a PPG sensor, and a 3-axis accelerometer. Their product is being used on the finger, and they put the accelerometer on the circuit. The optical sensor is located on the inner layer of the band, as shown in figure 1.

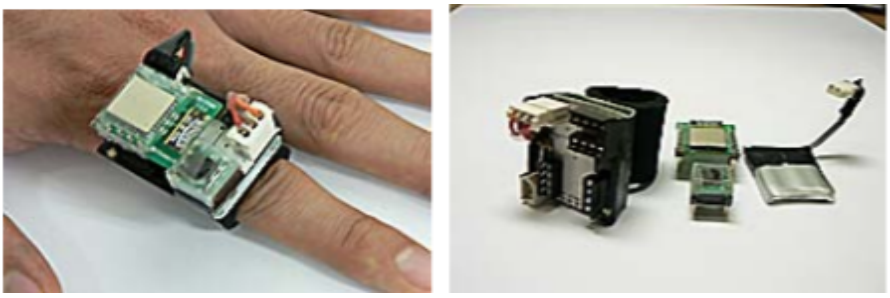

Figure 1: Wearable PPG sensor

The fourth article talks about the application of digital accelerometers for wired and non-wired Mechanomyography (MMG). Mechanomyography is a signal device that records reliable and valid signals of muscle activity. They also mentioned the work that they have done to extend the digital platform to design a wireless MMG sensor. According to the article, "The accelerometer was first compared to a piezoelectric contact sensor by Watakabe et al. who found that they were both adequate devices for measuring an MMG signal, but that the accelerometer had the additional advantage of measuring in physical units of acceleration, m/s^2," [4]. In addition, an accelerometer provided greater bandwidth which they found out in further studies. Also, an accelerometer is the most robust MMG sensor available and one of the cheapest devices to use. As shown in figure 2, the experimental setup and the position of the accelerometer that is used for the MMG device to detect the muscle activity for each movement. They applied a 3 Hz with increasing the current levels over time.

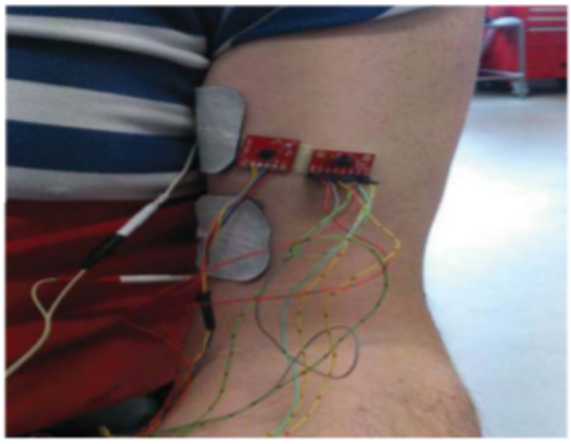

Figure 2: Experimental Setup & accelerometer position (ADXL335)

In 1923, one of the accelerometer industrial applications was found. It could measure the vibration and the acceleration of industrial components because it weighed about 1 lb and was large. The fifth article mostly talked about the MEMS accelerometer from engineering to medicine. MEMS stands for Micro Electro Mechanical Systems, a manufacturing technique that combines mechanical and electrical components to build tiny integrated devices or systems. Interestingly, the application for the accelerometer changed over time, which went from an engineering application to a clinical application which is day-to-day functioning of the hospital, interacting with the patients, and solving problems. So, the MEMS accelerometer that the article reached with their studies is a tiny electromechanical device that checks the acceleration in multiple axes of an object. As you can see in figure 3, picture (a) shows an accelerometer embedded in the earpiece of an IndyCar safety system. For picture (b), it is a wireless-aided accelerometer to monitor the Geumdang Bridge in Korea.

Figure 3: (a) An Accelerometer embedded. (b) A Wireless-Aided Accelerometer.

OpenLog

The data logging process is routine and standard in many projects. Hence, finding articles that use OpenLog was not a difficult task. Though, we looked for similar works in order to grasp the idea of synchronizing multiple data inputs - the GPS and Accelerometer in this case - into the SD card. In article [5], we found that the original project back in 2018 used the OpenLog for similar purposes; however, they did not connect the OpenLog in both RX and TX pins; they only connected the RX pin on the OpenLog to the TX pin on the Arduino (as shown in Figure 4), which indicates that the OpenLog did not send data directly to the Arduino. Furthermore, the project from the article [6] built a system that identifies road irregularities; however, their system did not use an OpenLog or a similar chip; in fact, the collected data was immediately sent to the cloud. This indicates that there are different approaches to storing data for such projects. The research we did was not meant to copy other projects but to understand the many ways data can be stored.

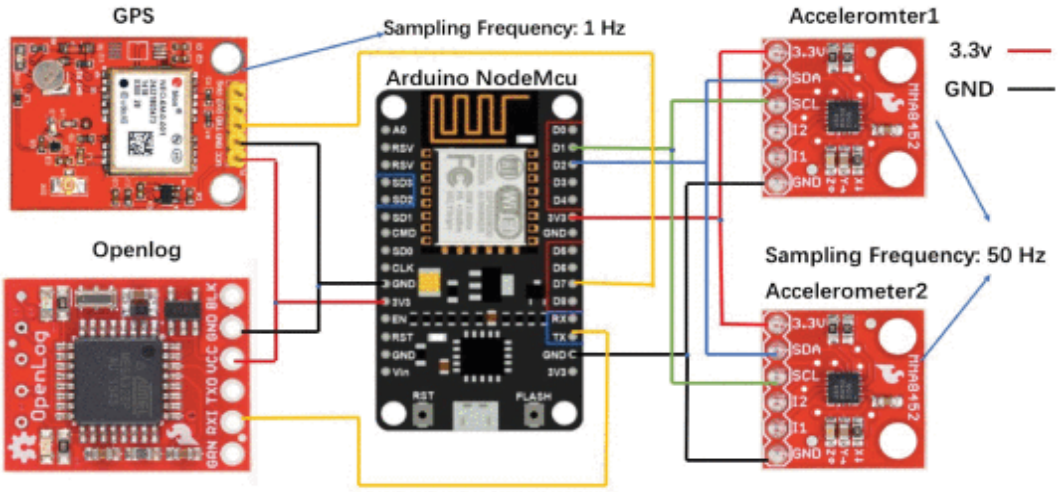

Figure 4: Previous Project System Architecture

Prototype Findings:

GPS

The GPS chip is crucial to the project's success. Josh built two different prototypes: an Arduino MKR1000, breadboard, wires, and GPS chip, while the second uses Sparkfun's Qwiic cable system with a shield that connects to an Arduino nano 33 IOT and the GPS chip. Neither of the prototypes constructed was able to output data accurately because the antenna needed for the chip had not been delivered. The demonstration of the prototype went as planned but not well. The program Josh edited from a Sparkfun example compiles and uploads to the Arduino perfectly fine but only outputs zero for the latitude and longitude data. The board is also supposed to have a flashing blue light when connected to satellites that blink at a default rate of 1Hz. The absence of this light signifies that an antenna is needed for further prototype testing. The longest part of the project was manipulating the example program multiple times over to try and get some accurate data when we did not think an antenna was necessary for data output. The results from this prototype will significantly influence the project because the latitude and longitudinal data are key features of the project's design. The project aims to find obstacles in a bicycle's path, mark where those obstacles are on a map, then upload that data wirelessly to a cloud to be dispersed to other bicyclists using a mobile app. Our team is tasked with making everything except the mobile app and wireless uploading of data which makes up at least half of the total project, resulting in the GPS data being a quarter of the project. The code for Josh's prototype is illustrated below in figure 5. The program results are not included because the only output is that Latitude and Longitude equal zero. This issue was later solved when our team purchased the antenna.

```
// data lines printed to serial monitor, no actual data rn, all zeroes still (need antenna)
#include <Wire.h> //Needed for I2C to GPS

#include "SparkFun_Ublox_Arduino_Library.h" //http://librarymanager/All#SparkFun_u-blox_GNSS
SFE_UBLOX_GPS myGPS;

long lastTime = 0; //Simple local timer. Limits amount if I2C traffic to Ublox module.

void setup()
{
  Serial.begin(9600);
  while (!Serial); //Wait for user to open terminal

  Wire.begin();

  if (myGPS.begin() == false) //Connect to the Ublox module using Wire port
  {
    Serial.println(F("Ublox GPS not detected at default I2C address."));
    while (1);
  }

  myGPS.setI2COutput(COM_TYPE_UBX); //Set the I2C port to output UBX only (turn off NMEA noise)
  myGPS.saveConfiguration(); //Save the current settings to flash and BBR
}

void loop()
{
  //Query module only every second. Doing it more often will just cause I2C traffic.
  //The module only responds when a new position is available
  if (millis() - lastTime > 1000)
  {
    lastTime = millis(); //Update the timer

    long latitude = myGPS.getLatitude();
    Serial.print(F("Lat: "));
    Serial.print(latitude);

    long longitude = myGPS.getLongitude();
    Serial.print(F(" Long: "));
    Serial.print(longitude);
    Serial.print(F(" (degrees * 10^-7)"));

    Serial.println();
  }
}
```

Figure 5: GPS Prototype Program

Accelerometer

The accelerometer is one of the most critical parts of the project. Without it, we cannot collect any information regarding the data. Yaqoub's part is to get the accelerometer to get the values of the x, y, and z axes that are needed. We can detect the obstacles the bike passes over from the z-axis value. At first, the accelerometer made the proper values of the axes except for the z-axis. Yaqoub expected the z-axis to give a 0g value instead of a 1g value. After long-lasting research and talking to multiple people who could potentially help, There was not that much help online since no one had worked on this type of accelerometer before. Yaqoub reached the latest output value, which must be because of gravity. The value for the x and y was correct except when moving it even a little bit, indicating that the high and low bits are in the opposite location of the

reading register. So, Yaqoub tried to fix it, but it still keeps giving him random values when he moves it after working on this issue for three weeks. Finally, he got the output values close enough to the needed values for the project, which is shown in Figure 6. The biggest challenge was rearranging the addresses for the x, y, and z axes. It was so confusing, while also tough to try to find an answer/solution. Hopefully, he can get the values that are needed for the project.

```
X= 16   Y= 0   Z= -1
X= 0   Y= 0   Z= -1
X= 0   Y= 0   Z= -1
X= 0   Y= 0   Z= -1
X= 0   Y= 0   Z= -1
X= 0   Y= 0   Z= -1
X= 0   Y= 0   Z= -1
X= 0   Y= 0   Z= -1
X= -3   Y= 0   Z= 0
X= -3   Y= 0   Z= 0
X= -3   Y= 0   Z= 0
X= -3   Y= 0   Z= 0
X= -3   Y= 0   Z= 0
X= -3   Y= 0   Z= 0
X= -3   Y= 0   Z= 0
X= 10   Y= 0   Z= 0
X= 10   Y= 0   Z= 0
X= 10   Y= 0   Z= 0
X= 10   Y= 0   Z= 0
```

Figure 6: Accelerometer output data

OpenLog

Since the OpenLog will be essential for logging the data from the two subsystems, it is essential to build a prototype beforehand in order to ensure that the system runs smoothly and efficiently. Because we did not have any data to test the functionality of the OpenLog connection yet, Ibrahim used "fake data" for this prototype. He connected it to the Arduino through a UART connection system instead of I2C since he used the Arduino Uno for this prototype. Though, since this particular microcontroller does not allow the use of the UART pins (0 and 1) and USB power at the same time, he used the SoftwareSerial library (Figure 7) to create digital pins elsewhere (in this case, pin 10 as RX and pin 11 as TX). Ibrahim found that the data he used was transmitting to the SD card as he expected. Thus, a similar implementation of the OpenLog can be used for our main project despite the fact that we would likely use an I2C communication system instead of UART, as this prototype prepared us for what is ahead and gave us a clear picture of the data flow.

```
#include <SoftwareSerial.h>

SoftwareSerial Serial1(10, 11); // RX, TX
```

Figure 7: SoftwareSerial Library

5. **Final Design**

System Architecture:

Figure 8 shows the final system architecture that the team created. Each system has its components where the front one is the primary and the second system in the back is the secondary. The front design has a GPS chip, SD card, Accelerometer, and Battery, and all of them are connected by a QWICC cable. The back system has the same components except for the GPS. All collected data will be saved on the SD card as a text file and given to the client.

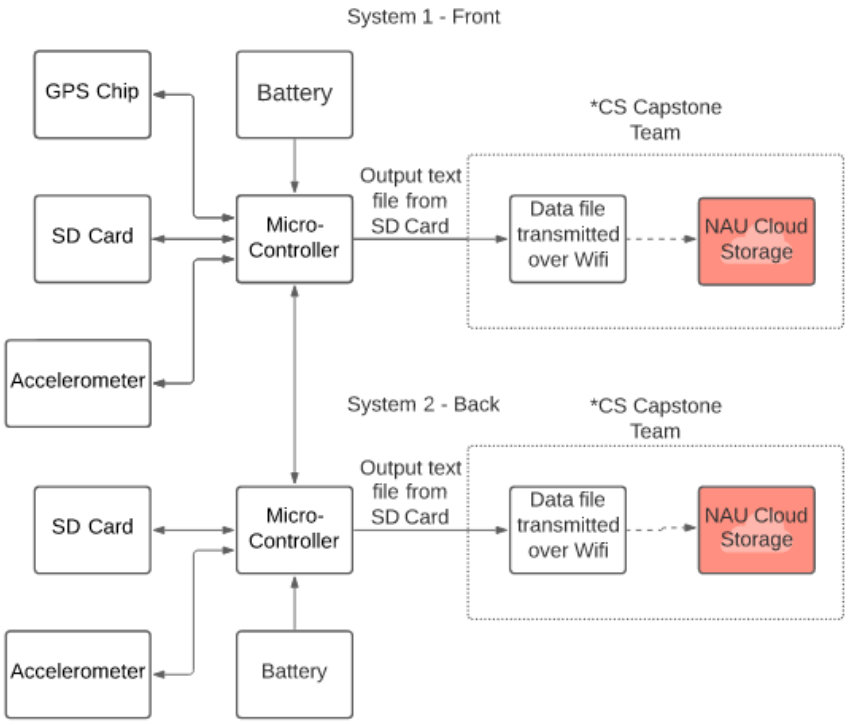

Figure 8: System architecture

Figure 9 shows the level 2 design for the GPS and the steps of how the GPS works. GPS needs to connect to the satellite when it first starts to get an accurate location and get the exact date and time. The antenna will be used with GPS to get the Latitude and Longitude. GPS's job is to identify the location of each obstacle and the time the test was taken. The GPS will upload its location every second to ensure that we are getting an accurate location. Once the GPS collects data, it will be saved on the SD card.

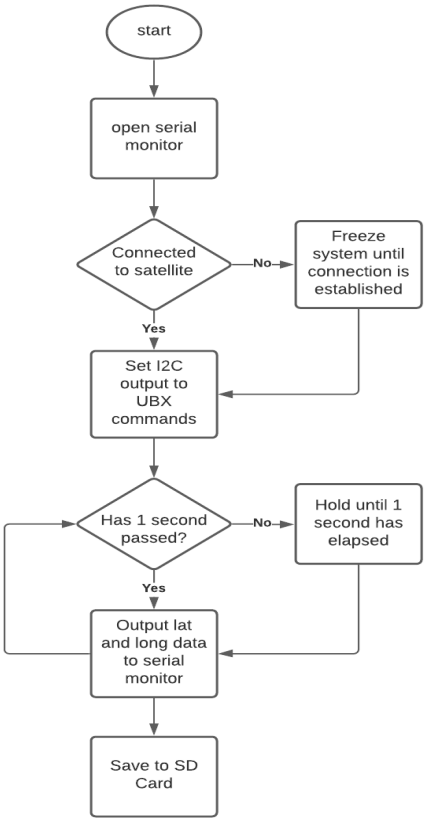

Figure 9: GPS level 2 design

The accelerometer's operation is represented in Figure 10. It begins by examining the axes' values. The X-axis and Y-axis identify whether the bike turns or moves straight, uphill, or downhill. The Z-axis change will decide whether or not there are any obstacles on the paths. It then gathers all the data and saves it to the SD card. At least 100 Hz is the sampling frequency of the accelerometer. The team will be able to pinpoint the position of each obstacle using the GPS. Both data sets will be stored simultaneously.

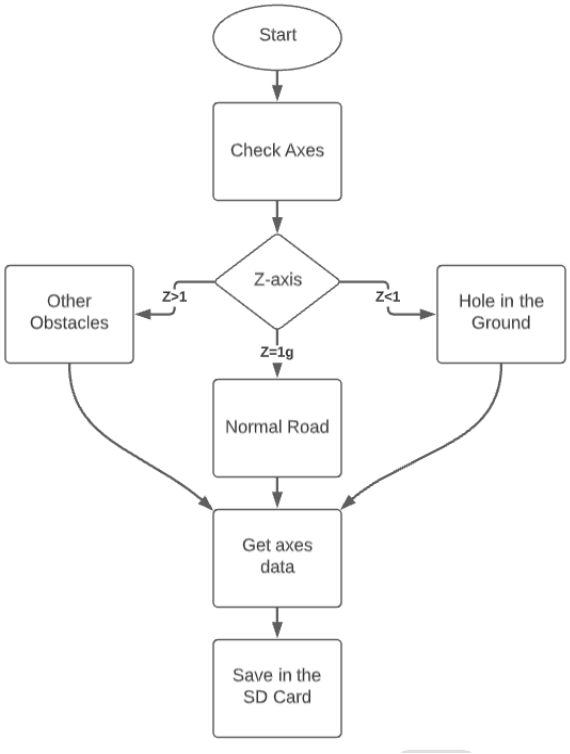

Figure 10: Accelerometer level 2 design

The GPS and Accelerometer input data must be combined and synced into one string before sending any data to the SD card. Each GPS/Accelerometer input is aligned with its matching GPS/Accelerometer input. The combined data from the Arduino will be transmitted to the SD card through the I2C connection between the Arduino Nano and the OpenLog, which is the saving procedure. When the bike is moving, the storing process begins. As a result, when there is movement, the team must store the essential data on the SD Card and erase the unneeded data after uploading it to preserve space. Figure 11 represents the OpenLog level 2 design.

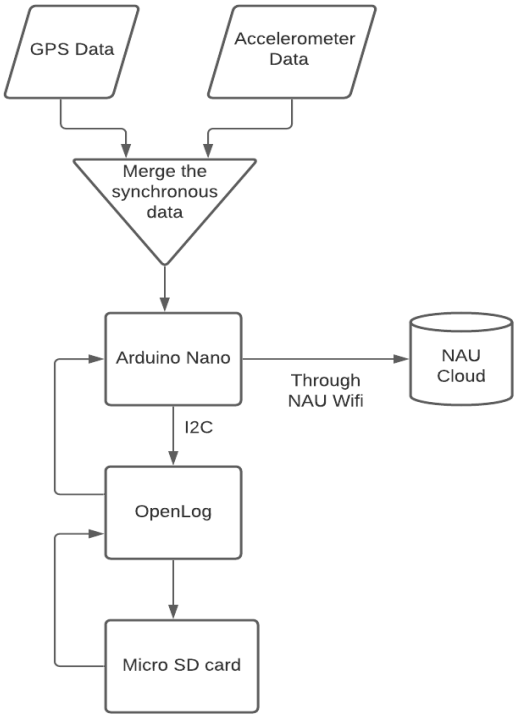

Figure 11: OpenLog level 2 design

Major components of the system:

The main component of the system is the *Arduino NANO 33 IoT* microcontroller. This microcontroller is perfect for this project because it is specifically designed for IoT systems and powerful enough to handle multiple subsystems. The first significant subcomponent is the *SparkFun Triple Axis Accelerometer Breakout - KX132 (Qwiic)*. We chose this accelerometer chip for its impressive output data rate, making it ideal for the system to fit the clients' requirements. Another major subcomponent is the *SparkFun GPS Breakout - ZOE-M8Q (Qwiic)*. This chip can efficiently operate at 1Hz and has a horizontal accuracy of 2.5m which is more than enough for this project. Furthermore, the output data is saved on the *SparkFun Qwiic OpenLog* chip with a *SanDisk Ultra 32GB* MicroSD card inserted into it. The memory card is large enough to record over 1000 round trips around the NAU campus. Finally, the system is powered with a *PKCell LP503562* Rechargeable Battery with 3.7V/1200mAh and can power the entire system for more than 4 hours. All subcomponents are designed for Sparkfun's Qwiic Connect System and are connected via Sparkfun's Qwiic cables. The system is secured in a 3D printed waterproof container that should protect the system from bike accidents and harsh weather conditions. Additionally, a reset button cable that can be pressed from outside the container is added in case of an unexpected system error.

6. **Results**

Requirements Testing Results:

Type of Test	Status	Req #	Requirement		
Inspect		1	GPS Chip for location		
UTS	*	1.1	GPS sampling frequency at least 1 Hz		
		1.1.1	Allows for the most accurate spatial resolution		
Inspect		2	Accelerometer to recognize obstacles		
UTS	*	2.1	Accelerometer sampling frequency at least 100 Hz		
		3	Battery life		
UTS	*	3.1	It needs at least 3 hours battery life to collect data		
		3.1.1	Estimated trip takes about 25 minutes from North to South Campus and back to start		
Inspect		4	SD card for storage		
UTS	*	4.1	At least 100MB of memory to store multiple trips		
		4.1.1	Currently using MicroSD Card with Adapter (32G)		
		5	Being able to locate different obstacles location in different trail		
Integration		5.1	Being able to locate holes that are at least 3 inches deep		
Integration	*	5.2	Must be able to locate other obstacles in the trail		
		5.2.1	Obstacles may include fallen branches, rocks, and other small objects that a bike can pass over		
Integration		6	Searches for a WiFi connection when the system is stationary		
Integration		6.1	Save battery life by searching for WiFi when not collecting data.		
		6.2	System is considered stationary when sitting still for at least 10 minutes		
Inspect		7	The size of the whole system will be no larger than a water bottle		
		7.1	The system will fit inside a water bottle that fits in most water bottle holders on a bike		
		7.1.1	Typical holder size: 5 in. tall and 2-⅞ in. diameter		
Integration		8	Must be robust and durable		
UTS		8.1	It needs to be able to handle the same wear and tear as the bike itself		
UTS		8.1.1	Should be waterproof enough to withstand rain conditions		
		8.1.2	Should be able to withstand hot and cold weather temperatures		
UTS		8.1.2.1	Maximum temperature: 185° F		
UTS		8.1.2.2	Minimum temperature: -40° F	GPS (-40 to 85 °C)	Accc (-40 to 105 °C)

Figure 12: Requirements spreadsheet with a color indication of requirements testing results

Test results:

The most important test was combining all the components for the final testing. The final testing took different routes, got the data needed, and applied it to the map. We used three different places to test our product. The first route was taken from W. A. Franke College of Business building is in the southwest of campus to the Center for International Education building in the northwest of campus. The second route was taken from Skyview Apartments, located in the middle of campus, to the University Union, located in the middle north of campus. The final route was taken from Skydome in the southeast of campus to the Health and Learning Center located in the northeast of campus (figure 13).

As you can see, figure 13 shows a sample example of a map with all the obstacles detected from a mobile app. The mobile app was developed by another team hired by Dr. Ho to run in parallel

with our project during testing. Our product had issues with data output missing random characters, but our data would look very similar to figure 13 if it was mapped. We did three preliminary testing in each location to compare the data we got from each testing time. For the three tests done in the exact location, each team member did one test to see if the person's weight on the ride's speed would affect the output results. It only affects it by a small number that is hard to detect most of the time, and it will not be an issue. As you can see, figure 14 shows different types of obstacles that the system could detect in different locations.

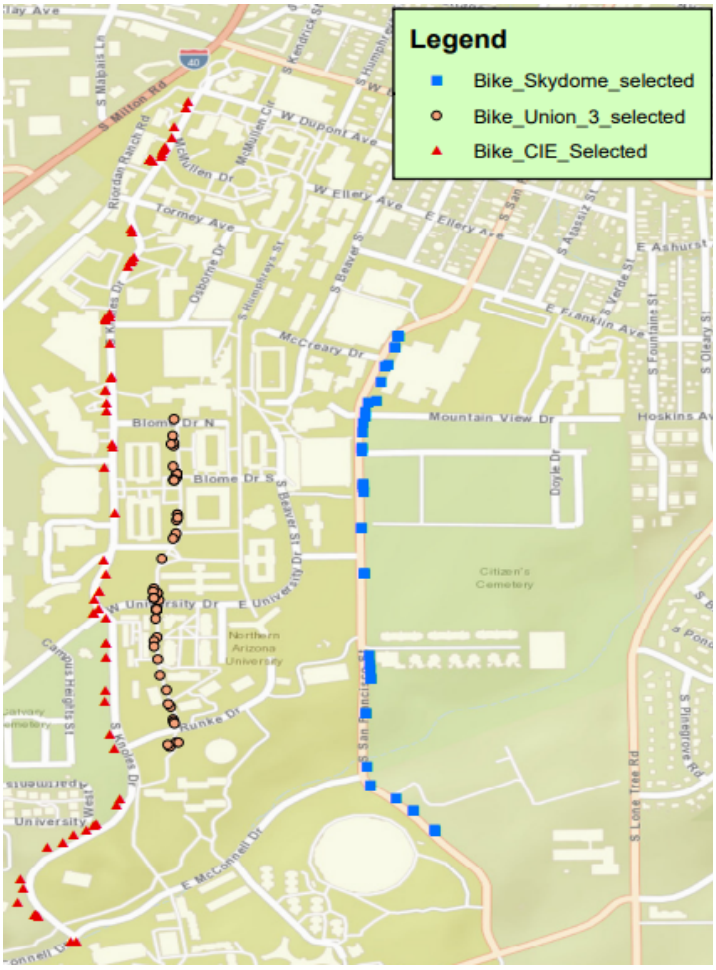

Figure 13: Sample example for obstacles location

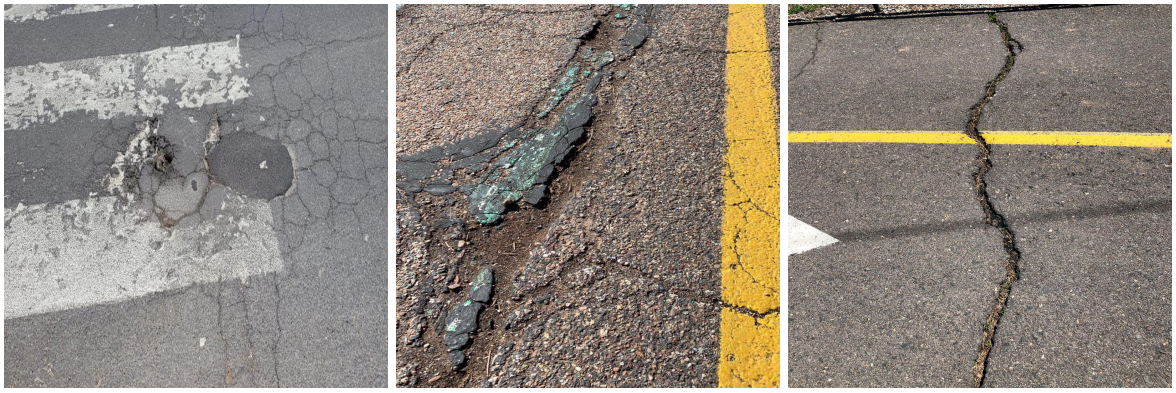

Figure 14: Different types of obstacles in a different location

Analysis of Results:

The team conducted three different weather condition tests. One of the tests was when it was raining outside, and it passed the test. The other requirement was that it was snowing, so they ran the test in that environment, and it passed with everything operating as planned. Even though the weather was meant to disrupt the communication between the satellite and the GPS, the GPS continued to function normally. Because there was a slight breach in the box, the team assumed that water would enter it; nevertheless, nothing did. Before doing the test in those conditions, the team conducted a test in which the temperature was too high, but the equipment passed the test without issue since the components can withstand high temperatures. So far, all of the tests have gone as planned. At the same time as the team was testing, they double-checked that they were meeting the essential requirements, which were all satisfied satisfactorily and without issues. The figure shows data collected during the testing, such as date, time, hazard (Accelerometer), and hazard location (GPS). The raw data output shows time (in ms), latitude, longitude, date, and time every 1 second, and time (in ms), x, y, and z values every ~0.1 seconds.

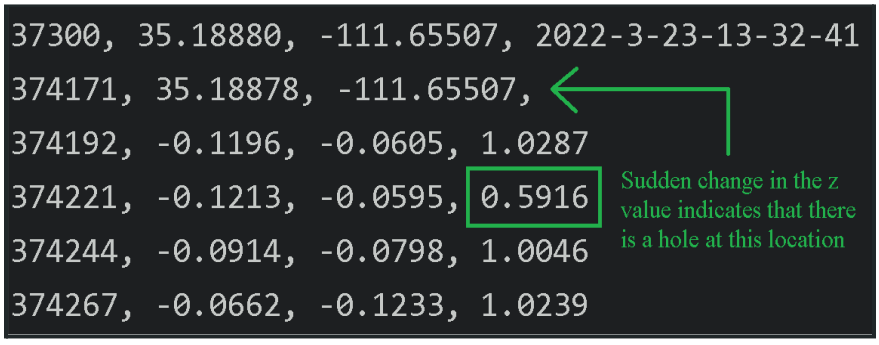

Figure 15: Section of the output file

7. Conclusion of Capstone Report

Major Requirements and Results:

As a team, we picked five essential requirements to ensure that we would be able to finish the project and be on the right path. The first one is that the GPS sampling frequency needs to be at least 1 Hz which means that every 1 sec, the GPS requires to provide the system's location. Therefore, we need to be given the spot as fast as possible to get an accurate location for each obstacle. The second requirement is that the accelerometer sampling frequency needs to be at least 100 Hz to detect as many obstacles as possible. At the same time, we need the system to catch as many obstacles as possible, even if they are in a row. The third requirement is that the battery needs at least 3 hours of battery life to collect as much data as possible. Additionally, the battery we have can work at least 3 hours to test 6 round trips around the campus. The fourth requirement is the memory card in at least 100MB of memory to store multiple trips, and the one we have has 32GB, which is more than enough to store around 1000 trips. The final requirement is to locate other obstacles in the bike trail, which our system did by detecting most of the obstacles on the bike path, and most of them were around 3 inches in depth. On top of that, the system was able to detect 3 inches above the ground, and the team used a utility hole as an example of the trial bump, as you can see in figure 16.

Figure 16: Left image is a crack, and the right image is a utility hole

Lessons Learned:

In summary of the team's work, we learned many things throughout work on this project. The most important one that every team needs to have is that communication is the key to the success of every project. That is what we have done from the beginning until the end. We faced many challenges throughout building the system, and we were able to fix most of them to continue to succeed. One of the things that we realized at the end of the project is that the missing characters that we had in the .txt file can be fixed by changing the baud rate of the Arduino. Unfortunately,

we could not change the baud rate due to using I²C communication, which prevented us from changing it as the OpenLog was running slower than the accelerometer. The team wanted to use the second system, but we found that it would take more time to connect them wirelessly. Before that, we ordered a wire to connect both systems, but the wire was not long enough, and that is why we switched to wirelessly. We wanted to connect the system to NAU WiFi, but it was impossible to connect the Arduino due to the NAU two-step verification. Thanks to Dr. Winfree, Dr. Ho, Mahsa, Dr. S, and Yifei for their assistance. Every person that was just listed is a vital part of the project, and each helped guide us in different ways for the duration of the project.

8. **User Manual**

Introduction:

Thank you for choosing Team Instru-MEN-tation to work on your project. Updating the components for the original Instrumented Bike Project is a huge necessity in order to make Dr. Ho's ideas feasible. It has been approximately three years since the system was updated, and we all know how long three years is in terms of technological advancement. The purpose of the original instrumented bike is the same as it is now. Dr. Ho wants to create a system that makes cyclists' time on roads and trails safer by having them gather data with our system and making that data available to everyone through a mobile application. The collected data is a combination of obstacles in the bike trail like rocks or branches and cracks or potholes inroads that the cyclists may run over. We made a few minor changes to the original system while also upgrading most of the original components. Our team provides you with a custom 3-D printed case that has been spray-painted with Krylon's outdoor spray paint to make it mostly weatherproof. That also fits inside the typical cell phone holder a cyclist may have on their bike. Not only does the system look better, but it now functions better with an upgraded accelerometer, GPS, and OpenLog MicroSD card datalogger for data storage. This user manual aims to help you, the client, successfully uses and maintain the upgraded Instrumented Bike system. Our goal is to provide you with enough information that you can easily use in order to benefit from this product for years to come! This project's overall problem is that cyclists do not currently have the safest way of knowing whether a new trail or road is safe. Another problem this product will help face is the road conditions that drivers do not necessarily see: large potholes and cracks in the road. Most modern cars have enough suspension and clearance that potholes or cracks do not immediately impact their driving experience. This only delays the amount of time between roads being fixed which creates a relatively unknown problem to most people that smaller cracks and potholes can lead to accidents just as large ones do. Creating a way to make roads safer for cars will also make the same road safer for cyclists, which is why this problem is at the forefront of our project's objectives. The subsystems of this project are relatively simple. There are two main subsystems and one minor subsystem that relies on the first two subsystems. The main ones are the GPS and accelerometer. The accelerometer is slightly more important than the GPS because this is the chip that senses when the cyclist is moving over an object in their path. The accelerometer uses an x,y, and z coordinate plane as calibration. The x-axis is parallel to the bike's frame, and the y-axis is perpendicular to it, while the z-axis goes through both of these planes to measure the vertical displacement of the system, representing the bike's vertical displacement. The x-axis represents the bike's tilt and shows whether the cyclist is going uphill or downhill, while the y-axis shows whether the bike is tilting left or right. The GPS is the second most important subsystem because without it, there would be no way to track where an obstacle is on a map and when exactly the cyclist encountered that obstacle. The GPS uses an antenna to connect to a satellite that provides the latitude and longitude of the system while also

using the satellite's Real-Time Clock to provide the user with the exact date and time that the file was created on. This is essentially the start of the cyclist's trip. The data from the previous chips is then output as a CSV file to the system OpenLog, which puts that data into the MicroSD card. To measure exactly where an obstacle is, the data file also contains the time since the system's startup in milliseconds leading to each line of data provided by the Arduino function millis(). This provides the ability to do some postprocessing magic to determine where an obstacle is between two different GPS data points. The OpenLog uses a function similar to serial.print() in Arduino but with a few extra steps to transfer the data from the GPS and accelerometer to the MicroSD card. The data from the MicroSD card was supposed to be output to a cloud storage system like Google Drive or OneDrive after it fills up with rides. However, we were not able to accomplish this because of issues with NAU's Wifi protection that requires the user to provide login credentials to access the full network. Arduino's wifi connection does not have this option because you cannot forcibly input someone's NAU User ID and Password into a browser window through Arduino. Besides this issue and many others, our team was able to provide a project that satisfies our client's needs with a few extra commodities built-in, like a reset button that allows the cyclist to restart the system manually and indicate that a new trip is beginning. The issue with our data collected during testing was one of the hardest challenges we faced. This issue was that random characters were missing from every few output data lines, which made the data unethical to use. While testing our product, we tested another version of our project in parallel for data comparison for our client. This product was a mobile app that uses the user's smartphone to track the obstacles and then output the data in the phone's storage system as a mappable data file. Because of this, we were able to get an example of our data when it is mapped out, which can be seen in figure 13 in the project report. Our team is confident that if we were able to map our data, it would be almost identical to the other product's results in showing exactly where the obstacles are along the cyclist's path. Our product has several advantages over a smartphone, with the main one being that its power is all focused on gathering the data, and no power is diverted to any sub-processes as the phone may have. Our system also has the flexibility of quickly getting the data from the system by manually removing the MicroSD card and mapping the data using a MATLAB algorithm. The final advantage is the option of quickly replacing the battery with a new one if the first one runs out with its 3-4 hour powering capability.

Installation:

To use our system, the user must ensure that the 3.7V battery is fully charged and connected to the Arduino Qwiic shield via Sparkfun's Qwiic cable, making sure the cable colors red and black are connected to the red and black parts of the battery's connector. Furthermore, the GPS, accelerometer, and OpenLog chips should also be connected to the Arduino via the Qwiic cables. For better cable management, the user may connect the chips to each other using the Qwiic cables and then connect only one chip to the Arduino (rather than connecting each chip directly to the Arduino, we also recommend connecting the OpenLog directly to the

Arduino's Qwiic Shield by itself). Next, confirm that the Arduino's LED is blinking and that the red power LEDs are lit on each chip. Next, insert the microSD card into the OpenLog. To ensure data is being recorded to the microSD card, ensure the green light on the OpenLog is blinking. There is no specific rate at which it needs to be flashing. Finally, ensure that the chips, Arduino, battery, and reset button are secured in their respective places. Close the container lid and attach the system to a bike using a phone holder. The user may now push the reset button whenever ready to begin testing the bike trail.

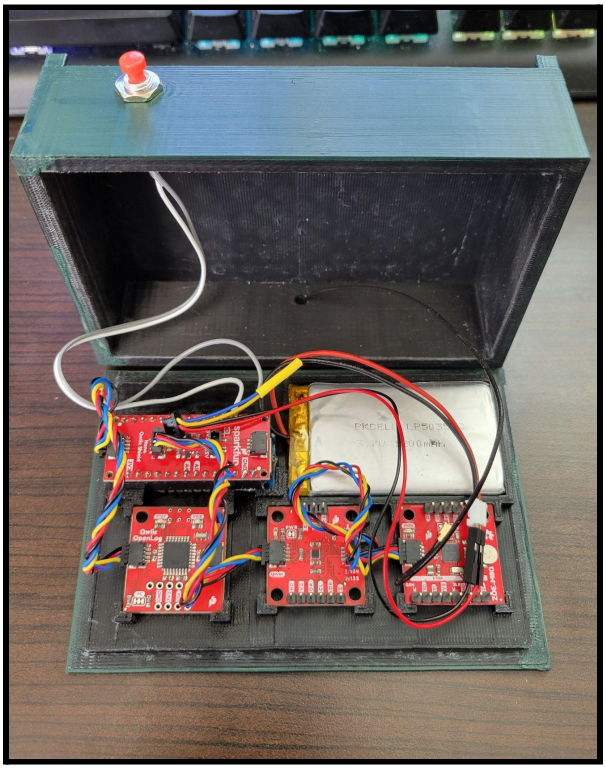

Figure 17: System container half opened with chips secured in place.

Configuration and Use:

When ready to test the bike trail, push the reset button and begin riding the bike. After you are done, remove the system from the phone holder and open the container lid. Detach the OpenLog from the container and remove the microSD card. Insert the microSD card into a computer to review the results. Typically, the files are named "LOGXXXXX.TXT" with X being the number. Your last trial will have the most significant number. You may rename the files if needed.

To record multiple trips without storing the data from the microSD card, open the lid and then press the reset button. While pressing the button, make sure the lights on each chip turn off, then turn back on again. Next, re-check that the lights are all on for each system and the OpenLog is

recording; as stated in the Installation section above, close the lid, put the system back in the cell phone holder, and begin your ride.

Maintenance:

To ensure that the system operates as expected, it needs to be regularly maintained. If the system is used for thousands of trips, the microSD attached to the OpenLog card will eventually fill up with old data. The microSD card can be removed from the OpenLog and inserted into a computer to delete some or all data that's in there (Do not delete the "config.txt" file). Additionally, the antenna attached to GPS and glued to the container lid may need additional gluing. Make sure to glue the container to the black sticker attached to the antenna and not the antenna itself. Finally, with regular opening and closing of the lid, the solder connection that connects the reset button to the Arduino may deteriorate over time. This connection may need to be re-soldered after heavy use.

Troubleshooting Operation:

1. **Hardware:**
    a. **Arduino**
    If the Arduino has not been used for some time, upload the code needed for the system to work.
    b. **GPS**
    You need to be outside for the GPS to work well since it has to connect to the satellite to grab the exact Latitude, Longitude, date, and time. If used inside, the value for Latitude and Longitude would be zero, and the date and time will be old.
    c. **OpenLog**
    When starting the system, hit the reset button to make sure to get a new .txt file. If the OpenLog green light is not flashing, the reset button must be pressed. The other option to fix the OpenLog flashing green light is disconnecting the battery and then connecting it again.
    d. **Battery**
    If the system does not work when the battery is connected, that means the battery needs to be recharged between 30 minutes to one hour to be fully recharged.

2. **Software:**
    The only thing that needs to be done in the software is to ensure that the computer has the library needed for each component used in the system. The libraries needed for each subsystem are listed below:

    GPS Library:
    #include "SparkFun_Ublox_Arduino_Library.h"

Accelerometer Library:
#include "SparkFun_Qwiic_KX13X.h"

OpenLog Library:
#include "SparkFun_Qwiic_OpenLog_Arduino_Library.h"

Conclusion:

We would like to thank you again for choosing Team Instru-MEN-tation to work on your Instrumented Bike Project. We hope this system will help you get more accurate and easy access to the data collected for many years of productive use. We are grateful and honored to work with you two: Dr. Chun-Hsing Ho and Dr. Kyle Winfree. While we are all moving on to professional careers, we would be happy to answer short questions in the coming months to help you get the product deployed and operating optimally in your organization. Please reach out to us at any time via the emails provided below:

- Joshua Garot: jpg262@nau.edu
- Ibrahim Aljarbou: ia342@nau.edu
- Yaqoub Abdulkarim: yma36@nau.edu

9. **Appendices**

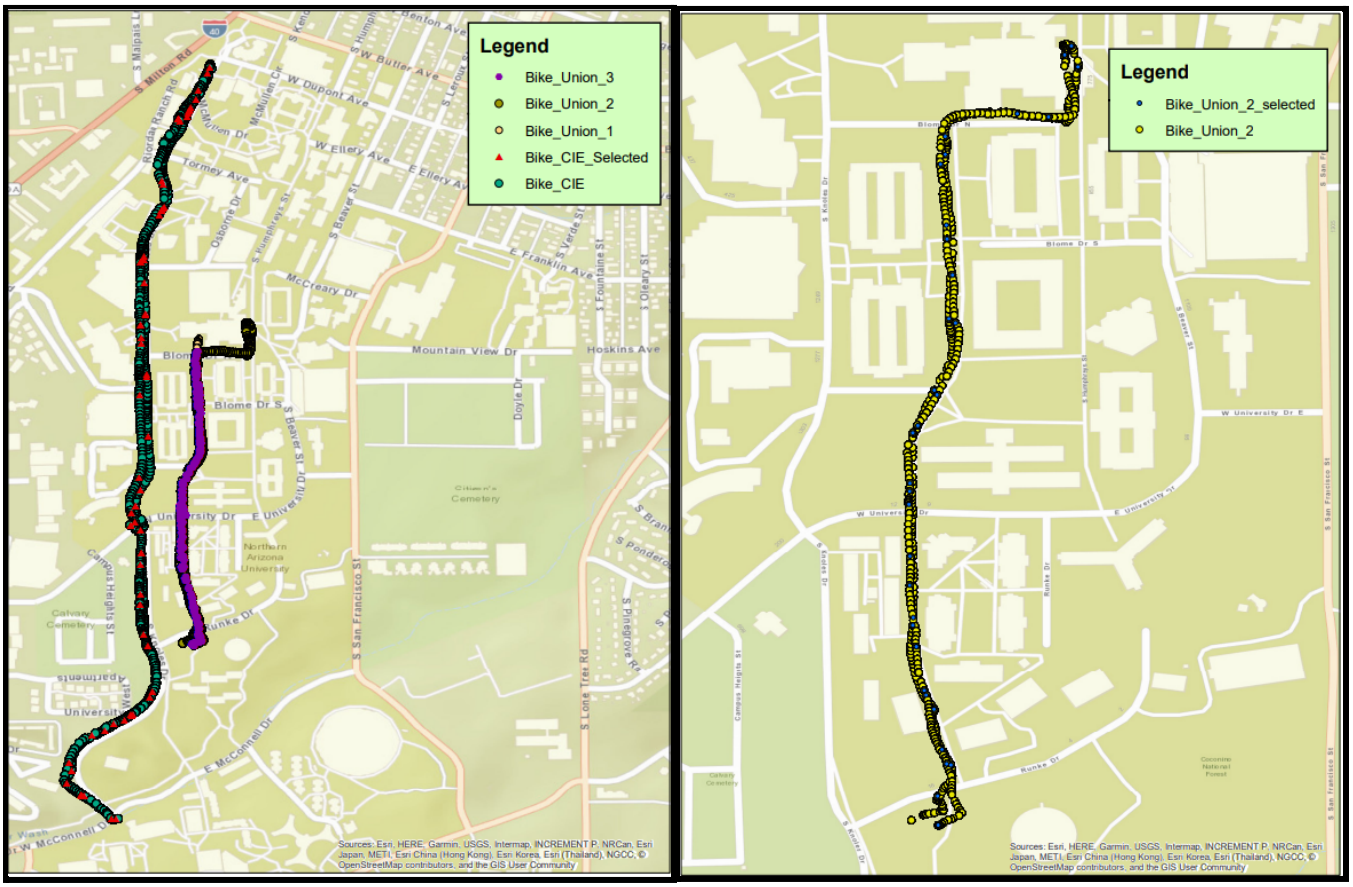

Figure 18: Maps from the mobile app showing all data points

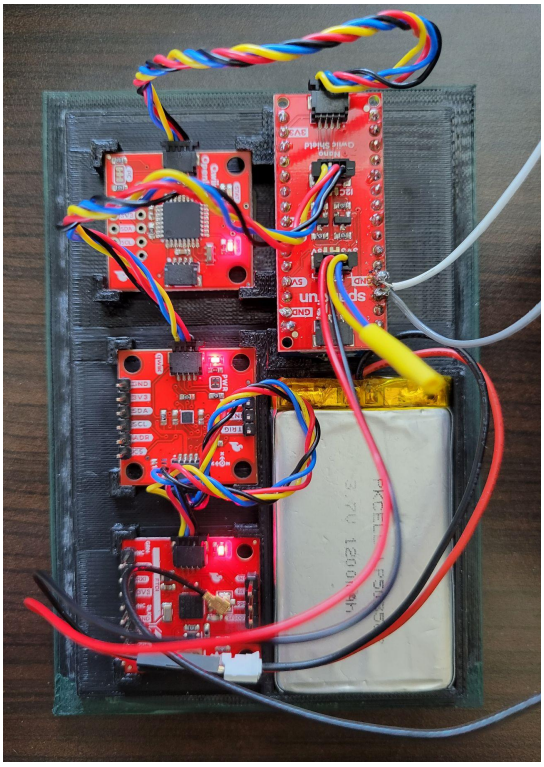

Figure 19: Close-up of system chips secured in their respective places.

Figure 20: System size compared to a cell phone.

References

[1] B. Radoslav, K. Miroslav, M. Guzan, I. Tomčíková and V. Melnykov, "GPS cycle computer," 2017 International Conference on Modern Electrical and Energy Systems (MEES), 2017, pp. 256-259, doi: 10.1109/MEES.2017.8248904.

[2] L. D. Vittorini and B. Robinson, "Application of low cost frequency standards for commercial and military GPS," 1993 IEEE International Frequency Control Symposium, 1993, pp. 821-834, doi: 10.1109/FREQ.1993.367482.

[3] H. Han, M. Kim and J. Kim, "Development of real-time motion artifact reduction algorithm for a wearable photoplethysmography", *Ieeexplore.ieee.org*, 2007.

[4] N. Campbell, T. Egan and C. Deegan, "The application of digital accelerometers for wired and non-wired Mechanomyography", *Ieeexplore.ieee.org*, 2017.

[5] P. Qiu, X. Liu, S. Wen, Y. Zhang, K. N. Winfree and C. Ho, "The Development of an IoT Instrumented Bike: for Assessment of Road and Bike Trail Conditions," *2018 International Symposium in Sensing and Instrumentation in IoT Era (ISSI)*, 2018, pp. 1-6