Project Report:

BEE 484 Sensors and Sensor Systems

Pinecones GPS & Environmental Data System

December 7, 2018

Authors:

Ian Etheridge, Andrew Steeble, Sam Stevens & Michle Alem

Abstract

A wearable and portable GPS tracking system with local environmental data reporting is presented. The device allows two users to share GPS location data via Wifi. Relative distance and direction are reported between peer devices. Barometric pressure, air quality, temperature and altitude data are reported individually to each user. Applications include but are not limited to hiking, hunting, camping and child monitoring.

Survey

Benchmarking

Garmin sells GPS and sensor systems designed for rugged wilderness traveling and exploring including the Foretrex 601 (Figure A) which retails for $249.99 and InReach Explorer (Figure B) that retails for $449.99. The Foretrex 601 is a wrist-watch style system and the InReach Explorer is a handheld, cell-phone style device.

The Foretrex i sensor systems include a 3-axis accelerometer, 3-axis compass, and a barometric altimeter. The system also includes a GPS navigation system which uses multiple satellite systems for location accuracy. It is relatively inexpensive for what the system offers. The system also has a functional display which can be manipulated by the user.

The InReach Explorer is a higher end version of the Foretrex 601. This is a handheld system featuring wireless communication through a satellite network. The system also has a backlight, improving nighttime reading ability. The system is very similar to a smartphone, using an application based UI. This system also accesses accurate GIS data that can be downloaded before travel. These systems are the higher end of GPS device currently on the market.

These systems offer an intriguing look into sensor application for outdoors. Our own application of sensors is an attempt at creating a reliable, inexpensive sensor and location package to keep track of party members in the wilderness. These systems would inform the party individuals of important environmental information to maximize safety.

Figure A: Foretrex 601

Figure B: InReach Explorer

Principle Operation Physics

Our device incorporates an air quality sensor, a combination barometric pressure, altitude and temperature sensor and the GPS sensor. The following sections detail the working principles for each of these sensors.

Air Quality Sensor:

The air quality sensor in the Seed Studio Grove Air Quality sensor package is the Winsen MP503. It is a CMOS type gas sensor consisting of a heating element and a semiconductor substrate. When heated, the sensor is able to chemically react with oxygen near its surface forming an oxide layer by bonding free electrons in the doped semiconductor with oxygen. Thus the degree of oxidation that occurs is inversely proportional to the conductivity of the sensor's semiconductor and directly proportional to the sensor's resistance, R_S.

In the presence of fresh air, oxidation of free electrons from the semiconductor material is maximized, significantly decreasing the conductivity of the semiconductor in the sensor and yielding a high sensor resistance, R_S. In the presence of a reducing agent (e.g. a pollutant such as CO_2, formaldehyde, alcohol, acetone, methanol, etc.) the absorption of Oxygen (and the formation of the oxide layer) is significantly decreased keeping the conductivity of the sensor much higher, yielding a very low R_S.

The semiconductor is placed in series with a voltage source, V_C, and a load resistor, R_L, and the output voltage measurement is made across R_L as shown in Figure 1. A normalized resistance curve for the semiconductor's resistance, R_S, relative to its resistance in the presence of fresh air, R_o, is presented in Figure 2. *Note: V_H is the voltage applied to the heating element.*

The MP-503 sensor is embedded in the Grove Seed Studio peripheral and the peripheral's schematic is shown in Figure 3. The output voltage from Pin 4 on the MP-503 (the voltage across the load resistor R_L) is the non-inverting input to a comparator used to detect changes relative to the comparator's output voltage.

The library code accompanying this sensor uses an interrupt to periodically sample the output through an analog read of the output voltage. Based on the difference between the previous and current levels (or a sufficiently high present reading) the sensor's library functions

determine the air quality level as fresh air (low analog reading deviation and low present reading level), low pollution (moderate analog reading deviation or moderately high present reading level, or as high pollution (large analog reading deviation or very high present reading level).

Figure 1: Air Quality Measurement Circuit for Winsen MP503 Sensor (Reference [1]).

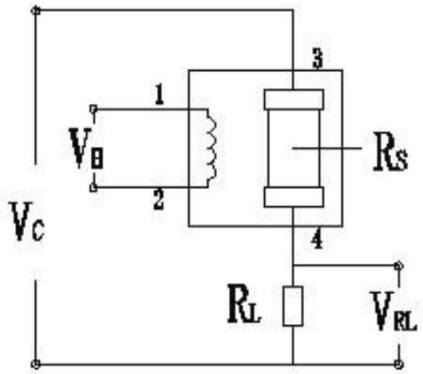


Figure 2: Sensitivity Curve for Winsen MP503 Sensor (Reference [1]).

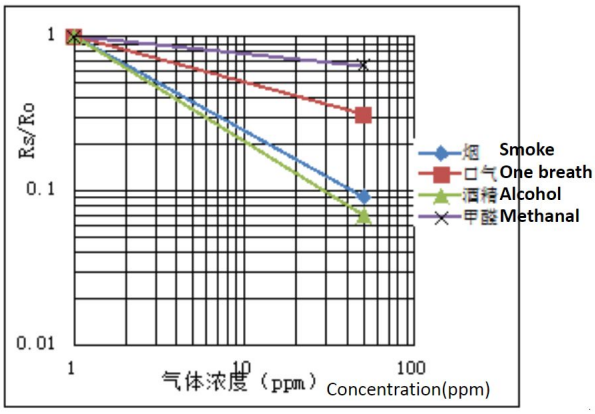
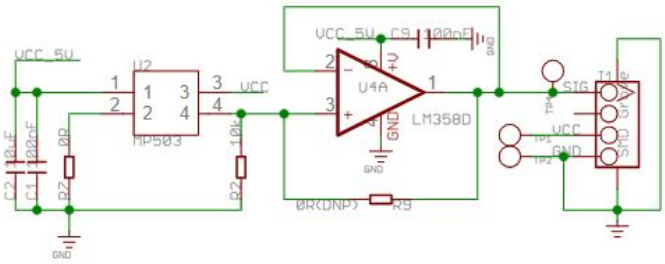


Figure 3. Grove Air Quality Sensor Schematic from Ref [3].



Adafruit MPL3115A2 - I2C Barometric Pressure, Altitude & Temperature Sensor

The MPL3115A2 sensor uses a MEMS absolute pressure sensor and a 24-bit analog to digital converter to provide pressure, temperature and altitude data via an I2C connection to the Arduino. Altitude is subsequently determined from calculations derived from the absolute pressure measurement. Absolute pressure is determined relative to a zero reference value measured for an inserted vacuum chamber during the manufacturing process.

Atmospheric pressure is determined from the equation:

$$p = p_o\{1 - (L- h)/T\}^{gM/RL} \qquad \text{(Eq. 1)}$$

where the constant value are defined in Table 1. Substituting the values from Table 1 into Eq 1. yields the simplified equation for atmospheric pressure:

$$p = p_o(1- h/44330.77)^{5.255876} \qquad \text{(Eq. 2).}$$

This equation can be solved for the altitude, h:

$$h = 44330.77(1 - (p/p_o)^{.190263} \qquad \text{(Eq 3).}$$

Table 1: Pressure Equation Values (Ref [8])

Variable	Value	Unit	Description
p_o	101.325	kPA	Sea Level atmospheric standard pressure
L	.0065	K/m	Temperature Lapse Rate from Sea Level
T	288.15	K	Sea Level Standard Temp
g	9.80665	m/s^2	Seal level acceleration due to gravity
M	.0289644	kg/mol	Mean molecular weight of air
R	8.31432	N*m/(K*mol)	Universal gas constant

The MEMS pressure sensor takes advantage of the piezoresistive effect to determine strain resistance from the atmospheric pressure (P1) at a given location as shown in Figure 4. As atmospheric pressure causes the region surrounding the vacuum chamber to deflect inward, four piezoresistive strain-resistors embedded within the flexible diaphragm are measured via a wheatstone bridge yielding a voltage output correlated with the mechanical stress due to the atmospheric pressure on the diaphragm.

The sensor has a measurement range from 20 kPA to 110 kPA and a corresponding altitude calculation range of 698 m below sea level to 5.574 km above sea level with a pressure resolution of 1.5 Pa corresponding to an altitude resolution of 0.3 m.

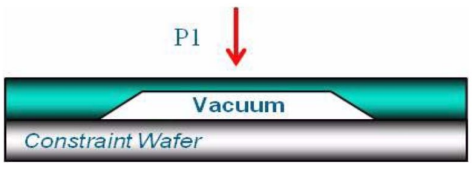
The data sheet and supporting documents for this chip refer to the temperature sensor as a "high resolution on chip temperature sensor" without specifying its design implementation. More generally, a solid state temperature sensor (such as the TMP36 used in our Embedded Systems course lab work) makes use of the relationship between the thermal voltage, V_T, and terminal voltage, v_{be}, between the base and emitter terminals of a bipolar junction NPN transistor where

$$v_{BE} = e^{vBE/VT} \quad \text{(Eq 4)}$$

$$V_T = kT/q \quad \text{(Eq 5)}$$

where k is Boltzmann constant, 8.62×10^{-5} eV/K, and q is the charge of an electron, 1.60×10^{-19} C. The chip amplifies this voltage change making it available for an analog reading relative to the reference level of the microprocessor.

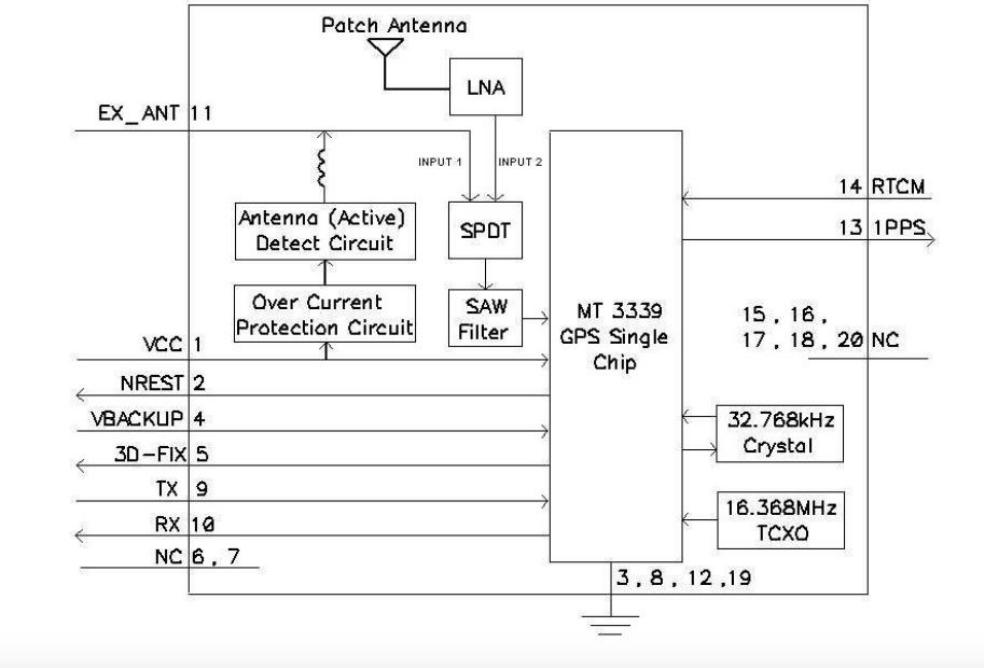
Figure 4. Absolute pressure sensing element for the MPL3115A2 (Ref [8]).



GPS Sensor

The PmodGPS from Digilent uses the GlobalTop FGPMMOPA6H GPS sensor chip. This chip includes a MT3339 GPS chip, the external components to build a circuit using the MT3339, and a ceramic patch antenna. Figure 5 shows a system block diagram for the GlobalTop FGPMMOPA6H GPS sensor chip.

Figure 5: System Block Diagram for the GlobalTop FGPMMOPA6H GPS [12]



Subsequent discussion of this GPS sensor focuses on the interaction between the GPS sensor and the orbiting satellite system. Additional circuitry detail can be found in references [11], [12] and [13]. There are 24 satellites orbiting the Earth at about 20,000 km for the purpose of GPS. These satellites are the transmitters and our PmodGPS is a receiver.

GPS satellites broadcasts two carrier waves: L1 (1575.42MHz) and L2 (1227.60MHz). These frequencies are chosen such that they eliminate what is called ionospheric dispersion, which causes systematic range errors due to properties of waves passing through the ionosphere. What is called the pseudorange is a calculation of the distance between the satellite and the receiver. The pseudorange is not the actual distance from satellite to receiver, there are many

biases and environmental conditions that affect this calculation. Figure 6 shows the pseudorange equation and a description of its variables. Multipath takes into account the transmitted signal bouncing off objects before getting to the receiver.

Figure 6: Pseudorange Equation and Description of Variables [15]

$$p = \rho + d_{\&\text{rho}} + c\,(dt - dT) + d_{ion} + d_{trop} + \varepsilon_{mp} + \varepsilon_p$$

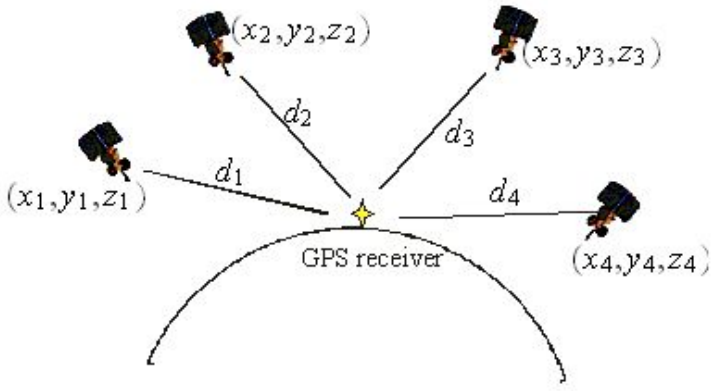
where:
p = the pseudorange measurement
ρ = the true range
d_ρ = satellite orbital errors
c = the speed of light
dt = satellite clock offset from GPS time
dT = receiver clock offset from GPS time
d_{ion} = ionospheric delay
d_{trop} = tropospheric delay
ε_{mp} = multipath
ε_p = receiver noise

There are two pseudorandom noise codes (PRN) that are modulated onto the carrier frequencies. The Course/Acquisition-code (C/A-code, wavelength of 300 meters) is modulated onto L1 and the Precision-code (P-code, wavelength of 30 meters) is modulated onto L1 and L2. The P-code is not authorized for civilian use so that the government can control the level of accuracy available to the public.

To transmit the signals from the satellites involves transmitting a carefully formulated code called pseudo-random sequences. The received signals and the transmitted sequences are compared, and the travel time for the signal is found by measuring when the two signals are most closely correlated.

Determining the position of an object on Earth is called ranging. For a GPS receiver to accurately determine its position, a minimum of four satellites are needed. Three are needed for the three physical dimensions but a fourth satellite is needed to help compensate for the effect of spacetime relativity.

Figure 7a : Diagram of Minimum GPS Receiver and Satellite Relationship [17]



The satellite orbits are such that at least 4 are visible at anytime from any point on Earth and every satellite uses an atomic clock with a nominal period of 1 nanosecond. To harness such resolution, the actual period must be determined with an accuracy of about 20 nanoseconds and requires taking into account General and Special Relativity theories.

General Relativity predicts that being farther from the Earth's mass results in less spacetime curvature and thus clocks on the satellites would appear to move faster than clocks on Earth, leading by 45 microseconds per day due to the satellite's distance from the surface of the Earth. Special Relativity predicts that since the satellites are moving relative to an observer on the ground, the clocks on the satellites would appear to move slower and would lag clocks on Earth by 7 microseconds per day due to the satellite's distance from the surface of the Earth.

The combination of these two effects results in a net difference of satellite clocks leading Earth clocks by 38 microseconds per day. Engineers took this into account before the satellites were deployed by designing the satellite atomic clocks frequency to slow down appropriately as they approached their designated orbit to match the atomic clock frequency on Earth.

GPS receivers also contain circuitry to calculate the 3D trilateration and compensate for the effects of Relativity. Further, the signals that the satellites transmit are microwaves, which move at the speed of light; this means that the wave's velocity is known. The GPS receiver compares the time the signal left the satellite to the time it receives the signal and multiplies that offset by the known velocity of the signal to calculate the distance to the satellite.

Trilateration is the process the GPS receivers use to determine their position on the Earth by comparing the distances between at least 4 satellites to pinpoint a location that satisfies those distances. Exactly how the receiver calculates this information given the sensed GPS satellite signals was indeterminable from our research. Figure 7b shows the system of equations to determine position based on data from at least 4 satellites, where *c* is the speed of light and *tb* is the time difference between transmission and reception. GPS receivers are constructed such that they know the codes sent from each satellite and can distinguish between them.

Figure 7b: Distance Calculation System of Equations for GPS Receiver [17]

$$\sqrt{(x-x_1)^2+(y-y_1)^2+(z-z_1)^2}+ct_B=d_1$$
$$\sqrt{(x-x_2)^2+(y-y_2)^2+(z-z_2)^2}+ct_B=d_2$$
$$\sqrt{(x-x_3)^2+(y-y_3)^2+(z-z_3)^2}+ct_B=d_3$$
$$\sqrt{(x-x_4)^2+(y-y_4)^2+(z-z_4)^2}+ct_B=d_4$$

Pros & Cons

In order to overcome the limitations of the benchmark systems and improve upon the current systems implemented we developed a cost effective product incorporating the data from the described sensor systems using an Arduino Mega for processing. A Raspberry Pi system is used in the current implementation for Wifi communication between the devices.

A major advantages of our system is its cost ($135) compared to the benchmarking systems. The other major contribution of our system is the inclusion of air quality monitoring and warning for high pollution scenarios which is not included in the benchmarking systems.

The need for such a system has become more pronounced in recent years as wildfires have increased in number and scale throughout the world. Our system offers users a method through which to determine it is safe to continue outdoor activity in the current conditions. Additionally, when paired with a simple alarm function (a passive buzzer and LED) this system may help prevent camping accidents and warn individuals near a fire of the danger while it is still possible to control damage or escape the region.

One of the major cons of our system is the weight of this implementation (13.5 oz) compared to the much lower weights for the benchmark systems. However, it is relevant to note that our current implementation is a prototype using both an Arduino Mega and Raspberry Pi development kits. In a subsequent version of this product, a single processor, mounted on a PCB would be used eliminating both development boards. Additionally, a power source such a coin style battery directly mounted on a PCB would likely be used rather than the USB portable charging devices used for power in the prototype edition. Furthermore, it is acceptable in this edition to have a higher weight since the product is belt strapped around the waist in a fanny pack rather than used as a wrist-worn or handheld device.

System Overview

The center of our system is an Arduino Mega microcontroller. Connected to the Arduino Mega is a PmodGPS sensor from Digilent, a barometric pressure and temperature sensor from Adafruit, and an air quality sensor from Grove. The Arduino Mega microcontroller acquires and processes the data received by each of the sensors and displays the results on an LCD screen for the user.

The Arduino Mega is also interfaced with a Raspberry Pi microcontroller that reads latitude and longitude data from the Arduino Mega serial port and processes it via Python. The Raspberry Pi then connects to a Wifi network and sends the latitude and longitude data to the other user allowing the peer system to process that data. The peer device then processes the data through a custom algorithm to determine distance and direction of the peer device from the perspective of the current device performing the processing.

The microcontrollers and sensors are all stored in a fanny pack that is comfortably worn around the user's waist with the LCD screen displayed externally. The Arduino Mega is powered by the Raspberry Pi and the Raspberry Pi is powered by a portable USB battery. Figure 8 shows a system schematic of the hardware implementation using Fritzing software sans external USB battery connected to the Raspberry Pi. All contents were placed in a waist-worn "fanny pack" with the LCD screen displayed on top for the user and the barometric and temperature sensors poking through the fanny pack so they were exposed to the atmosphere.

Figure 8-a: System Hardware Configuration

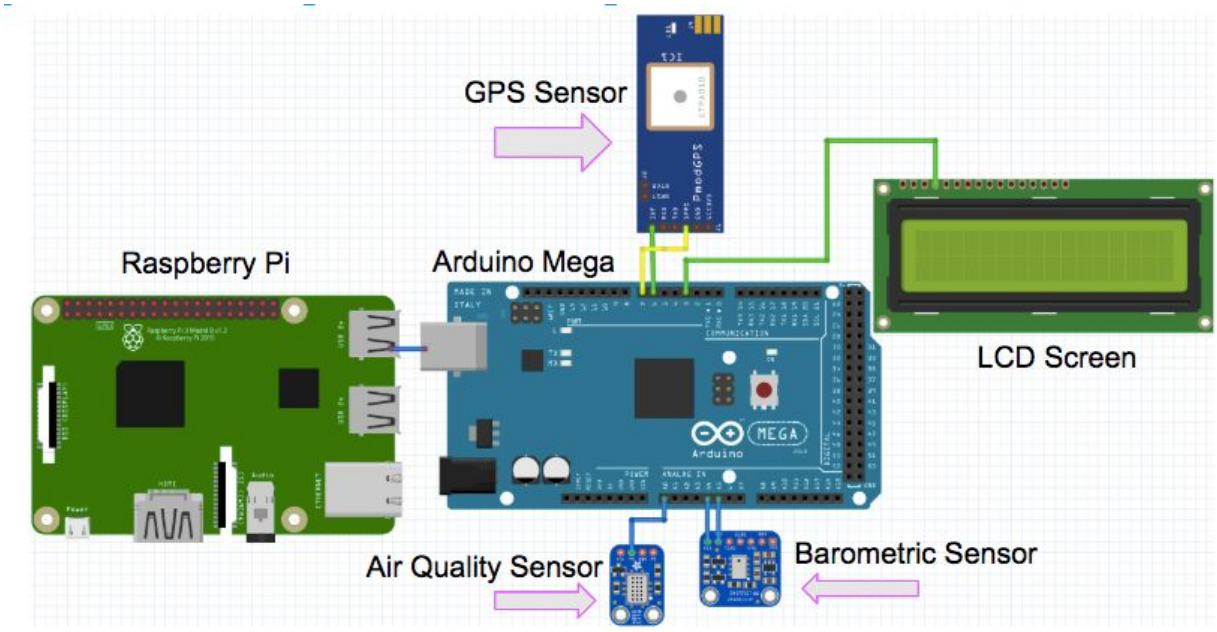


Figure 8-b: System Realization

To reduce and simplify code, the calculations for the distance in meters, direction in degrees, and cardinal direction were normalized for the greater Seattle area. The calculations do

not take the curvature of the Earth into effect since any distances we have been testing can be linearized without substantial error. Also, the physical distance in longitude degrees changes from the equator to the poles so the value we used was the distance between longitude degrees at the equator multiplied by the cosine of Seattle's latitude to better reflect the difference in longitude degrees local to the greater Seattle area. Even with these modifications, we were able to get highly accurate results. Errors would become more significant if trying to implement this code cross-country, for example, and generally unreliable if globally implemented.

Results

Our final system was able to complete the required task of our original project statement. The system had experienced a few unexpected system modifications due to the limitations of our Arduino Uno, but overall the project was a success. To begin with, the GPS sensor was not as reliable as we would have hoped. During some of our primary testing, we found that it could show significant error its location estimation. We tested it in an apartment in Seattle with multiple windows. While the sensor had 5 to 6 satellites visible, it still returned an inaccurate location. This can be seen in Table 2, comparing the true and measured latitude and longitude measurements. We did find that the sensor worked much better outdoors, returning GPS coordinates that were closer to the actual location. Testing was performed in the Renton Park area (explained below). Otherwise the GPS sensor performed well, with it's only limitations being the ability to acquire satellite information indoors.

During a test of the system, we drove a car through a back road near Phillip Arnold Park in Renton, Washington. While driving on this road, we acquired GPS latitude and Longitude coordinates. These coordinates appeared to be off by a small factor when compared to the map, which can be attributed to the level of accuracy displayed to the user. The GPS may have also been compensating for the speed at which the car was driving, and the timing of when the GPS data was pulled from the sensor. Otherwise, the sensor performed remarkably. The speed data acquired from the sensor was also accurate. We kept the car in motion at a constant speed of 12 mph, which translates to 19.31 km/hr. The speed that the GPS sensor returned to the user was 19.710 km/hr, which is accurate to ± 0.4 km/hr, this can be seen in Figure 9.

Table 2: Quantitative Data Gathered During Preliminary Experiments

Measurement	Measured Value	Actual Value	Difference (Actual - Measured)	Source For Actual Value
Current Longitude [°]	47.6264991	47.622775	0.0037241	Link
Current Latitude [°]	-122.358357	-122.357328	-0.001022	Link
Current Altitude [m]	101	111	10	Link
Distance Between Systems [m] *	5.66 k	5.21 k	450	Link
Direction from one system to the other [Degrees / Compass]*	51.99	52.009	0.019	Link
Speed of System [km / hr]	19.31	19.710	-0.4	Road Test
Temperature [°C]	2.19	1	1.19	Weather App
Barometric Pressure [inHg]	30.08	30.11	0.03	Weather App

*Calculated using University of washington Latitude and Longitude: Lat: 47.65533509999999, Lon: -122.30351989999997

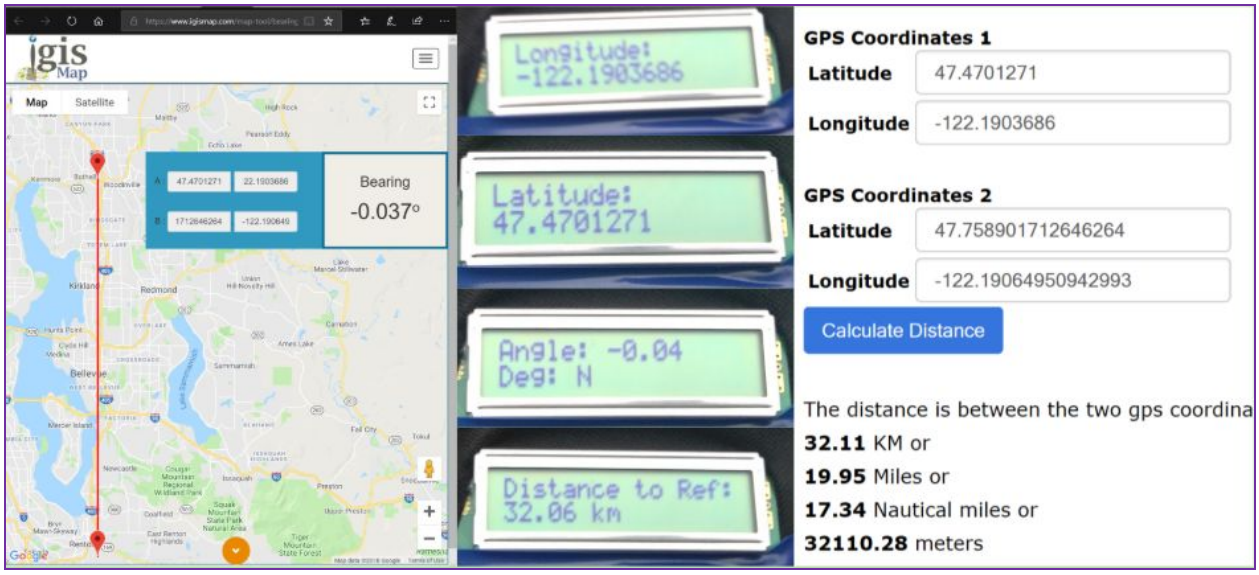
Figure 9: Speed of Car from GPS Sensor

The second requirement of the GPS algorithm involved coordinate transfer by uploading the coordinates to the cloud, and calculating the distance between this system and the peer system. The calculation algorithm is handled by the Arduino Mega and its accuracy is limited by the accuracy between the float calculations in the processor. We found that the bearing calculation was highly reliable, while the distance could have significant levels of error.

During our test, we compared the localized GPS coordinates near Phillip Arnold Park to the coordinates at University of Washington Bothell. The system displayed the output that can be seen in Figure 10. As can be seen, the returned bearing to get from Phillip Arnold park to UW Bothell is -0.04 degrees North. The distance to UW Bothell returned was 32.06 km. As can be seen from the true calculated distance and bearing, these are acceptable results. The level of accuracy which is returned to the user is relevant to the calculation performed in the code programmed on the Arduino.

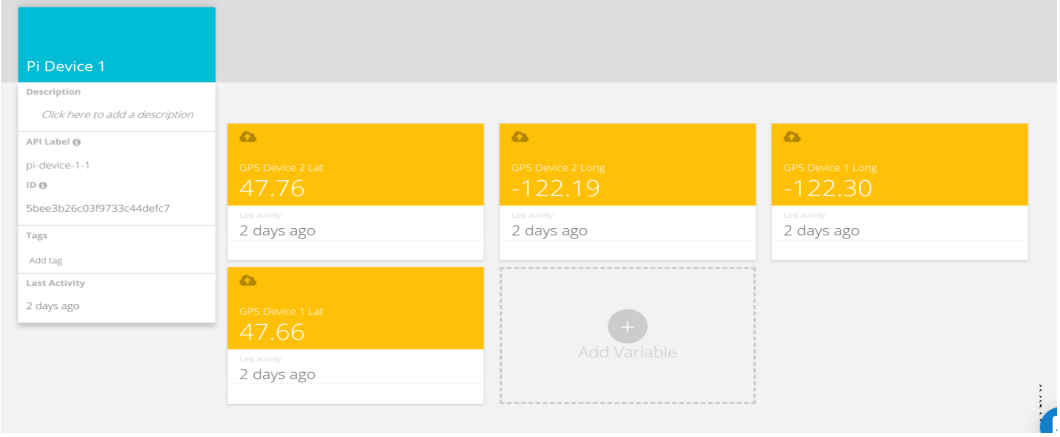
We did find a strange error when calculating the distance between University of Washington Bothell to UW Bothell. The system displays a distance of 15 km, when the true distance is 14.28 km. This is an interesting result because the true distance is smaller than what is displayed to the user. We could not explain this result in the time allotted to complete this project, so in a future project design we would spend some considerable time improving this calculation to improve accuracy.

Figure 10: GPS Distance and Bearing Comparison



The final testing of the GPS portion of the system was for uploading data to the cloud. Uploading was done through a serial input from a Raspberry pi to the Arduino Mega. The Raspberry pi would be constantly waiting for the Arduino to send information over serial input. When this happens, the Raspberry Pi will upload this data through an API to a website called Ubidots. The Raspberry pi would then take information from other variables in the cloud where a second device would upload their own GPS coordinates. This process can be seen in Figure 11. The data only displayed up to 2 decimal points, however the system stores the accuracy to what was uploaded to the system. This is good because there is no loss in accuracy from this process.

Figure 11: Display of Cloud Based GPS Storage



The next portion of our sensor system tested was the barometer. This system reported three details to the user: altitude, temperature and air pressure. The testing results are included in Table 2. Reliability of this sensor was related to the refresh rate of the sensor itself. Usually this sensor had a difficult time interpreting the indoor air pressure and calculating the altitude. This is due to the fact that altitude is determined as a function of air pressure as described in the device physics section of this report. While testing this sensor outdoors in cold weather, it performed very well. The outdoor temperature was found to be 1 degree celsius. The temperature that the sensor reported was 2 degrees celsius. This is within the level of accuracy reported by the data sheet, of an accuracy of ±1 degree celsius.

The altitude that the barometer reported was found to not be accurate. We found readings of -11 to -20 meters in some case while we were well above sea level. This had to do with the calibration of the sensor. Unfortunately, we could not calibrate the sensor well enough to report the data as useful to the user. Fortunately, the GPS sensor also reported an altitude reading, which was relatively accurate depending on the number of satellites in view. One particular sample is reported in Table 2. The final sensor data that the barometer reported was air pressure. This was found to be highly accurate, with a reading within 0.03 in HG.

Our air quality sensor only provides qualitative data and was tested by exposing the sensor to pollutants (such as a butane lighter) and smoke and it showed qualitative responses that were accurate, recognizing high pollution levels after this exposure. Efforts to generate qualitative data from this sensor were abandoned when it became clear that the GPS code required all execution from within the state machine function. This requirement made sensor sampling nondeterministic--the original intent was to use timers to generate consistently timed periodic sampling so that counter variables could be used to determine a recent percentage of exposure (by counting the number of high pollution reports over recent time period). In isolated tests, this aspect showed good response, cycling counters in prolonged states of smoke exposure, but unfortunately it could not be implemented in the final code.

Conclusion

Our system underwent quite a few changes as the project progressed due to technology implementation learning curves and sub-system compatibility. Our original implementation plan involved the use of the PmodWifi from Digilent for system communication that we had to abandon due to limitations with the Arduino microcontroller and lack of documentation of how to implement the device. Our best workaround for proof of concept was to interface the Raspberry Pi modules (already in the possession of a group member) with the Arduino for inter device communication. However, this also required us to upgrade the Arduino from the Uno model to the Mega model because the Raspberry Pi required a serial port for communication as did the PmodGPS and the Uno only had one serial port available.

Another significant source of complication was the state machine code logic used to operate the GPS. Attempts to encapsulate this state machine as a function that could be called from the main loop proved unsuccessful and caused system crashes. The goal here was to implement a rudimentary real time operating system via a function queue scheduling algorithm. In this scheme, a queue is used to insert scheduled tasks at timed intervals to the back of the queue while operations requiring immediate actions (interrupts) are inserted at the front of the queue and functions are called from the queue in FIFO order.

The function queue code was itself already developed as part of an embedded systems course assignment and tested. However, the inability of the GPS system to merge with this method required us to revise our project and incorporate all of our peripheral sensors into the GPS state machine. As discussed in the testing section, this made the system viable but introduced non-deterministic latency. Attempts to develop quantitative data based on the timing of samples from these sensors had to be abandoned. In a subsequent revision, use of a full real time operating system, such as μC, that allows time slicing and task context switches may make the state machine compatible with such an approach.

Beyond our many complications and sacrifices, the system performed with generally high accuracy and minor bugs arising only occasionally. The linearization of GPS coordinates around the Seattle area did not seem to significantly impact local measurements, the systems were able to boot up and operate from a portable USB battery, the fanny pack housing was easily wearable, and the data reporting/display was easily observed and interpreted by the user. For future projects, we learned that datasheets for potential parts of a system need to be cross referenced to ensure compatibility requirements before commiting to a product.

Response to Q&A

The main questions we received from the Q&A were the weight, final cost of our system, "is that a bomb?" and "what's inside the fanny pack?" These questions have been addressed throughout this report, but again we report a weight of 13.5 oz for our product and a cost of $135 for all of the components in the final design. A schematic demonstrating what's inside the fanny

pack was included in the system overview section of this report. Also, beyond the figurative sense of the word, this project is definitively not a bomb.

Collaboration Approach

Our group met in person, generally twice a week, in the library to discuss implementation plans and merge individual coding tasks. Initially, each member of the group took responsibility for learning how to obtain data from each of our sensors, control the wifi, LCD, etc. As the project progressed, these individually coded functions were merged and edited for consistent pin usage and to ensure that no contradictory aspects remained (e.g. Serial monitor prints from the library functions for the air quality sensor were omitted because the GPS required use of the UART pins). In the conclusion of this project, responsibilities were distributed for final coding and testing, poster creation, slide preparation and writing the sections of this report with final collaboration to review each of these deliverables.

References

[1] "Winsen MP503 Air Quality Gas Sensor Manual. Version 1.3, " Zhengzhou Winsen Electronics Company LTD." May 1, 2014. Retrieved Dec 1, 2018, from https://raw.githubusercontent.com/SeeedDocument/Grove_Air_Quality_Sensor_v1.3/master/res/Mp503%20English.pdf

[2] "Seed Studio Grove Air Quality Sensor V1.3 Schematic." Retrieved Dec 1, 2018, from https://raw.githubusercontent.com/SeeedDocument/Grove_Air_Quality_Sensor_v1.3/master/res/Grove_-_Air_quality_sensor_v1.3_sch.pdf

[3] " Seed Studio Grove Air Quality Sensor V1.3 Arduino Library." Retrieved Dec 1, 2018, from https://github.com/SeeedDocument/Grove_Air_Quality_Sensor_v1.3/raw/master/res/AirQuality_Sensor.zip

[4] “How Gas Detectors Work.” Thomas Industry Update Retrieved Dec 1, 2018, from https://www.thomasnet.com/articles/instruments-controls/How-Gas-Detectors-Work

[5] “Operating Principle -MOS Type Gas Sensors.” Figaro Engineering. Retrieved Dec 1, 2018, from https://www.figaro.co.jp/en/technicalinfo/principle/mos-type.html

[6] “MPL3115A2 - I2C Barometric Pressure/Altitude/Temperature Sensor” Retrieved Dec 1, 2018, from https://www.adafruit.com/product/1893

[7] “Xtrinsic MPL3115A2 I2C Precision Altimeter Datasheet.” Freescale Semiconductor Inc. Retrieved on Dec 1, 2018, from https://cdn-shop.adafruit.com/datasheets/1893_datasheet.pdf

[8] Guo, George. “Pressure Altimetry using the MPL3115A2.” Freescale Semiconductor Application Note. Document Number AN4528 REv 0, 07/2012. Retrieved on Dec 2, 2018, from www.nxp.com/docs/en/application-note/AN4528.pdf

[9] Sedra, A & Smith, K. Microelectronic Circuits, Seventh Edition. Oxford University Press. New York, 2015. PDF edition.

[10] “TMP Temperature Sensor.” Retrieved Dec 2, 2018, from https://learn.adafruit.com/tmp36-temperature-sensor/overview

[11] “PmodGPS Reference Manual”, Digilent Inc., Retrieved Dec 4, 2018, from https://reference.digilentinc.com/reference/pmod/pmodgps/reference-manual

[12] “FGPMMOPA6H GPS Standalone Module Data Sheet Revision: V0A”. GlobalTop Technology Inc. 2011. Retrieved Dec 4, 2018, from https://cdn-shop.adafruit.com/datasheets/GlobalTop-FGPMMOPA6H-Datasheet-V0A.pdf

[13] “MT3339 All-in-One GPS Datasheet”. Mediatek. 13 January 2017. Retrieved Dec 4, 2018, from https://labs.mediatek.com/en/chipset/MT3339

[14] “GPS Physics. Real World Physics Problems.” 2018. Retrieved Dec 5, 2018, from https://www.real-world-physics-problems.com/gps-physics.html

[15] “The Pseudorange Equation.” Penn State College of Earth and Mineral Sciences. 2018. Retrieved Dec 5, 2018, from https://www.e-education.psu.edu/geog862/node/1759

[16] “How Does the GPS Signal Work?”. Retrieved Dec 5, 2018, from http://www.physics.hmc.edu/research/geo/gps.html

[17] “The GPS.” Retrieved Dec 5, 2018, from http://www.math.tamu.edu/~dallen/physics/gps/gps.htm