6-2023

Hiking Tracker to Give Directions Along a Trail

Jennifer Williamson
Union College - Schenectady, NY

Hiking Tracker to Give Directions Along a Trail

Jennifer Williamson

ECE-499: Capstone Design Project

Advisor: Cherrice Traver

March 15, 2023

Summary

The goal of this project is to create a device that will give directions to a user along a hike. The directions will appear in real-time and be accompanied by the current time, and the distance the user has traveled. The components used to accomplish this were a processor, GPS component, a compass and a display. For the final design, the most important requirement was met: giving the directions to the user. In addition to that, the compass direction the user was traveling was displayed. The time and distances were not able to be displayed on the screen, although they could be calculated. The size of the final design was also not in the form of a watch but rather still on a breadboard, due to time constraints and the choice to focus on the ability to add more trails, making it a more practical device. The device was tested along a chosen trail on Union College's Campus but it does have the ability to add more trails. Valuable lessons were learned throughout the process including overall design and software development practices and there are improvements that can be still made to make the device more practical.

Table of Contents

Table of Figures and Tables

1. Introduction and Project Definition

The objective of this project is to create a stand-alone wearable GPS device that has a single purpose: to give user's directions along a hike. Due to this focused functionality, the device will be of a lower cost than the devices on the market. The device was tested for one trail but has the ability to add more trails through mapping done by the user.

The device involves a processor and 3 other main components. The first component is the GPS module which will receive the user's GPS coordinates to be able to determine their location and compare it to the trail. The next component is a compass which will retrieve the current direction that the user is traveling which will be used to dictate the next direction to give the user. The last component is a display which will display information about the hike to the user. All of these components will be discussed further in section 4.

The rest of this report is organized as follows: section 1 will give background information discussing further the goals of the project and current devices on the market. Section 2 goes into details on the specifications of the project along with constraints and standards that are important to the design. Section 3 discusses design alternatives and how the final design came to be. Section 4 gives an overview of the design and then goes into detail discussing the specific components and the software. Section 5 gives an overview of the results of the testing and how well the final implementation of the device meets the design specifications. Section 6 provides a conclusion and recommendations. Section 7 is a user manual for the device. Sections and 9 are the references and appendices respectively.

1.1 Needs Statement

People who want to hike need an easy way to get updated information on the progress of their hike to help them not get lost. Around 2,000 hikers get lost each year due to common factors such as falling off the trail and darkness [3] [14]. Having real-time updates such as directions and distance could have helped prevent this, since if hikers know how far they have traveled and how far they have left to go it can give them a better understanding of the timing of their hike which can be especially important as it gets dark. While there are watches on the market such as the Apple Watch and Garmin Watches that can be used for hiking, these are expensive and have more functionality than just directions and progress updates based on distance/speed.

1.2 Objectives Statement

The goal of this project is to design a device for hikers that will provide them updates on their hike as it progresses. It will be cost effective since it will have a focused functionality of providing the hiker with real-time directions along with metrics such as distance. This is in opposition to more expensive options that have features such as receiving text messages, heart rate detection and calories burned.

1.3 Background

1.3.1 Background Research

The overall objective of this project is to provide a hiking device for users that is low cost and effective. The motive behind this project is that a lot of people hike a lot, or are new to

hiking and want some way to not get lost, that does not involve carrying a phone around that may have bad service. Important data that highlights how helpful this device can be for people is shown below in Figure 1 which highlights the causes of deaths among hikers:

Figure 1: The largest % of hiking-related deaths are attributed to three things: lack of experience, lack of knowledge; and poor physical condition from [3]

The overall message from this graph is that poor judgment when it comes to hikes can result in death. This includes poor judgment from not having enough knowledge of hiking or the trail or poor judgment based on your physical condition and how much you can handle. This device is going to focus on combating poor judgment that results from not knowing the trail. If someone has a device that will direct them along the hike, they will hopefully not get lost and are less likely to venture off the trail and end up injuring themselves. Another cause of death among hikers is darkness [14]. This can be prevented or helped with this device, since the user

will be aware of how much they have traveled and how much further they have left to travel. It is still up to their own judgment whether they should turn around, but at least they can consider how late it is and whether or not it will be too dark soon to continue hiking safely.

1.3.2 Current Devices on the Market

Considering the devices on the market today, there is the Apple Watch that has a cost of around $250 (only considering devices for sale directly through apple) [2]. The Apple Watch is able to provide directions (through the Maps App) , but it also does a lot more: heart beat detection, messaging, and payment just to name a few. Thus the device for this project will cost less because it has a singular focus on direction giving rather than providing a smart phone in the form of a watch.

Another company that sells smart watches is Garmin and one device they sell is the fēnix® 5S Plus [6]. The cost is $700, but it has maps built into the system (displaying a map on the screen for ease of use), music, weather and sleep tracking capabilities just to name a few. This is one of the higher end devices that Garmin sells but it highlights how much you can get out of a watch if you choose to spend more money.

Since the goal is to give directions to a user along a hike, there should be a way of communicating these directions with the user in a way that is accessible, which is why the form that was chosen for this project is a watch. It is easy to wear and accessible and does not add complication to the user's hiking experience rather just adds an option to look at their wrist and see directions to help them stay on track.

Aside from cost, which is a big consideration for this project, usability, portability and aesthetics are important to address. Since this is meant to be something that people carry with them, without the need of other technology, this will have to be a battery powered device,

which is where portability comes in. The components and circuitry must be a size to be able to fit on a wrist, and the more compact the device the easier it is to use and the more appealing it is to customers. Large Apple Watches are 45mm, and the other devices on the market are comparable so that is sizing to consider and the larger end was examined since that is realistic for what size this project would compare to [22].

When considering components, battery life has to be calculated. Components should be low-power to allow for longer battery life [21]. The newest versions of Apple Watch, Fitbit and Garmin are expected to last upwards of 15 hours (on average) but given that is calculated with the device not constantly running and this device will be constantly running to give directions it is not expected that the battery life will last as long.

2. Design Specifications, Constraints and Standards

Given that my device is something that customers will read about and then choose to buy for their own hikes, the specifications are written in regards to the customer and what they can expect the device to do. The requirements are summarized below:

- The user will be provided with real-time directions, the distance they have traveled and have left and current time
- The device will detect when the user gets lost
- The device should be able to store multiple trails

The rest of the section goes into these in further detail.

2.1 User

The user will see a screen, discussed further in the next section, but there will also be

two buttons for their use. The first button is the start/stop button which they will use to indicate that they wish to start the hike. It can also be used as a pause button (if they wish to stop or want to go off the trail to look at something but do not want the device to think that they are lost). Hitting this button once will start the hike, again will pause the hike, and once more will end the hike. Then pressing it again will start the hike, and the cycle continues. It will be indicated on the screen (where the direction would be displayed) whether the user has paused or ended the hike. The second button is the "mute button" for the buzzer. The buzzer beeps when the user does not follow the direction given to them. This means that they are expected to end up at a certain coordinate and they do not arrive there. Once the buzzer goes off, it will continue to make noise until the user turns it off , or they correctly reach the nearest coordinate to them that is along the trail.

2.2 Display and Size

The display is outlined in the diagram below:

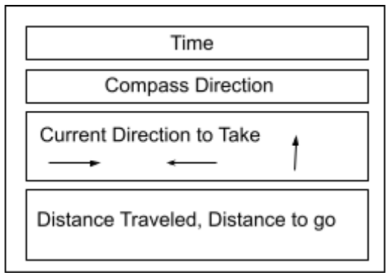


Figure 2: Outline of screen

On the screen is the time, followed by the current compass direction the user is traveling, the current direction to take (in the form of an arrow), the distance traveled and the distance left to go. All of this information will be displayed on a screen that will be able to be worn on the

wrist of the user as they travel along their hike. It will be the correct size to fit a wrist but also the correct weight so that it is comfortable to the user.

2.3 Constraints

When considering parts for this project, while size and form factor is important since this device is designed to be wearable, there is also a consideration of the power of the components and how they all fit together and connect to the microcontroller. The device should be small and be able to fit on a wrist, but it is also important that it is low power (so that it can last longer and be useful for the user throughout their entire hike). The other consideration is how all the parts will connect to the processor, making sure that there are enough pins for all of them to be interfaced. All of this was considered when picking parts and is described in more detail in the design section.

In conclusion, considering that the size should be wearable, it should be around 55 mm, which is 10mm larger than a large apple watch and 4 mm larger than a large garmin watch. It should also be around or less than 200g in weight. 200g is the weight of a heavier watch and given the time scope of the project the heavier end is probably more realistic. The battery should last a few hikes between having to be changed so should last around 6-8 hours. It was calculated that 4AA batteries (the standard 6V battery pack) is expected to last 5 hours which is a little lower than the ideal but will work for testing purposes.

2.4 Standards and Codes

Since the device is going to be using the user's GPS coordinates (received from the GPS module), it is important to follow the standard on representing and exchanging geographical coordinates. This is standard ISO 6709:2022, which specifies different types of

representation, including longitude and latitude along with horizontal position representation, specifically discussing how to represent coordinates both in terms of unit of measurement and order of coordinates [12]. GPS location representation is an important part of this project so it is important to follow this standard. The rules of this standard apply to how the coordinates are going to be displayed to the user, namely latitude comes before longitude, and north latitude and east longitude are both positive.

Another important standard is I2C [13]. Multiple components will be using I2C, which is utilizing one of the important features of I2C: it can have multiple devices sharing the two wires it uses. The two wires are serial data and serial clock, and only having two reduces the complexity of circuits needed, which is important for my project considering size is a constraint. It is worth noting there will be additional overhead needed for addressing, since the processor will be communicating with the different devices.

3. Design Alternatives

3.1 Trail Representation

After doing research of current solutions out there for hiking devices, two different options came about for displaying the trail to the user. Most hiking watches on the market use a trail based approach. This involves representing the trail as a map on a screen; it would look like google maps, where it would display a trail and the user's progress would be displayed as a line along the trail, updating as they moved along. Another representation is just giving statistical numerical information about the trail. This means just providing information such as distance you have traveled, and distance you have left to travel.

First, considering the “map-based” approach, there are positive aspects of this design; it is very user friendly and easy to use. Seeing your location on the map move with you makes it clear what part of the trail you are on and what part you are approaching. However, screen size becomes an issue when considering this approach, as well as considering if the screen should be touch screen. Also solving the problem this way becomes much less of an algorithmic approach and more of a graphical problem. My interest in the problem is more geared towards the algorithmic approach, which is why the following approach was considered.

Another approach, would be providing the user with numerical information about their hike, including the distance they have traveled and how much further they have left to go. The positive aspects of this approach include providing a clear summary to the user, and on the back-end of the project, trail representation is not needed on the screen and it can be dealt with behind the scenes, hopefully making it easier to eventually expand this project to include more trails. This is not seen as “typical hiking watch” in the sense that the directions are not map-based which is what people are used to with all the technology out there such as google maps. This difference however can make it more appealing to the market and it aligns more with my interest in the project of working with an algorithm rather than displaying a map to the user.

After analyzing and considering both ways to solve the trail representation problem as discussed above, the “numerical approach” was picked to solve the problem. The next step after the decision, was to decide how to give the user directions.

3.2 Directions

After deciding to display numerical information about the hike on the watch, it made sense to keep the concept of numbers and words on the screen, thus an option for directions was

displaying them in the form of commands: “Take a right in 15 feet”, “Stay to the left ahead”, etc. These directions would appear on the screen as necessary to make sure the user always knows what way to go. Considering that these commands would contain a number of characters, size of the screen became a consideration. This is discussed in more detail in the design section, but traditional LCD screens would not be able to fit all the information and comfortably be worn on a wrist. Thus a screen with more flexible text sizes was chosen: an OLED 128 X 64 graphic display. Since this can display more than just characters, arrows were chosen as the final direction giving mechanism used on the screen, since arrows are easy to understand and follow.

Giving audio directions was another design that was considered. While this would have led to a smaller screen size being possible, audio can be hard especially if you are walking or hiking with a group and having a conversation. Also given that the user is already looking at the device for the distance they have traveled, adding directions in the form of arrows as discussed above would be an easy addition that wouldn’t interfere with the user talking or the peace and quiet they may want on a hike.

After deciding to give directions in the form of arrows, algorithms had to be considered. A lot goes into this and it will be discussed further in the report but one big issue that was encountered was the fact that to correctly give an arrow that will get the user to go in the right direction, the user’s current cardinal direction needs to be known. Thus, the next design decision was deciding how to incorporate the compass direction of the user.

3.3 Compass Directions

Knowing the direction the user is traveling will help make the directions useful and accurate to the user. Different directions are calculated depending on the direction the user is

traveling. The design decision was whether to keep the work in the code and just use the user's current direction to figure out the next direction, or to provide the user with the compass direction they should travel next.

One of the advantages of providing the user with a compass direction, is that there is less chance for mistakes in the code regarding calculating the direction based on their current compass direction; it is a clear direction to the user. It is clear in the sense that if they know the direction they are traveling they could be able to figure out where to go next. This is where the big disadvantage comes in: it is not intuitive to everyone to be given a compass direction and figure out where to go next. So while you are leaving it up to the user to find their way, it is not always easy to translate the direction to which way to travel next. To make it easier, a compass could be added to the screen with the current user's direction, but this does not alleviate all the trouble and the user still has to think about where to go next.

Thus, after all this consideration, it was decided to deal with the compass direction in the code and give the user directions in the form of an arrow, but to still provide the user with awareness of what direction they are traveling. So it will say the current direction the user is traveling and the next direction to take. The user does not need to use their current direction to help them figure out where to go next, but it is nice to have that on watch if they want to know.

3.4 User Interaction

The last aspect of the design to consider is how the user will interact with the device. It was decided that the device will have a display namely an OLED display to provide all the information to the user. In terms of starting the hike, while there are other alternatives such as voice activation, a push button was chosen. A push button was chosen over a switch since it is easy to press and is how most devices on the market start tracking activities. A buzzer was

chosen to alert the user that they have gone off trail since it is a way to get the user's attention easily and will force the user to look at the device and see where they went wrong.

4. Final Design and Implementation

4.1 Design Overview

The final design has the following components working together: the compass directions and the arrow direction. Currently, the screen will display the following information to the user on their hike:

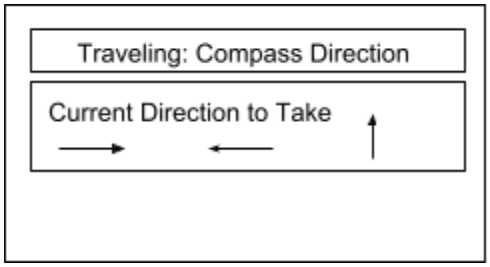


Figure 3: Display during Hike

The other aspects of the screen that do not currently meet the design specifications are the current time, distance the user has traveled and the distance they have left. There is also no buzzer to indicate that the user has gone off trail.

4.2 Trail Data

One trail, which is shown below, was used to test the device. It is a 0.79 mile loop around Union College Campus.

Figure 4: Test Trail

A few important aspects of the trail to note are that the trail does cross itself and all along the large outer loop there are multiple places the user could think they are supposed to turn but are not. The data for this trail included 45 points with longitude, latitude and compass bearing information. Details about how this is stored in Arduino is shown below in the software section.

A new trail can be added using an online resource [16]. The process for adding a trail is described in more the User Manual but it involves mapping the trail, downloading the data, running a python script and then updating the data in the Arduino code.

4.3 Processor Research

Considering the processor and what criteria is important, there is a balance needed between physical size, number of pins and voltage. Picking off of size and number of pins, initially the Arduino Nano seemed like a good fit for the project. However, given that my parts

can all be powered by 3.3V and small rechargeable batteries are around 3.7V, it did not make sense to have a processor like the arduino nano that needs 6-20V. Thus other processors had to be considered.

The processor needs to be able to be powered by 3.7V, have enough pins, and have enough memory. For pin consideration the top choices for the display and compass have I2C interfaces which can save pins, while the GPS is a serial connection. There are 2 buttons and a buzzer thus besides the I2C and serial pins there needs to be 5 digital pins available.

The memory for the project was roughly calculated to be 0.3KB for the storage of the trail and the program. The trail will have around 50 data points at 4 bytes each equaling 0.2KB and the program itself will be around 0.1KB, which is where the 0.3KB came from. Given that we want this project to scale, the memory anticipated to run 10 different trails is 3KB. This is going to be factored in when choosing a processor.

In addition to these specifications discussed above, programming the processor is also something to consider. The first aspect of this, is how it will connect to a computer (i.e USB or USB to TTL adapter), and what IDE is used. The more familiar the IDE, the better since that is one less thing that will need to be learned.

One option is using ATMega 328P, which is the microcontroller used on the Arduino Nano [1]. It can be powered by 1.8-5V, has enough pins and has separate program and data memory: 1KBytes of data EEPROM and 32KBytes of In-System Self-Programmable Flash program Memory. USB to TTL adapters can be purchased and the Arduino IDE or MPLAB can be used, both of which are familiar.

While other options were considered like the Adafruit Flora Wearable [9] and the Lilypad Arduino [15], the Arduino Nano was chosen as the processor due to the familiarity

with the Arduino IDE and the size.

4.4 Parts List and Justification

The parts for the project include a GPS module, compass, display and a battery. For the GPS module, some notable qualities to consider were its power and current consumption and how it will interface with the processor. The table below shows two GPS modules that were considered:

GPS Module	Operating Voltage	Current	Interfacing	Other Notes
Neo-6m u-blox GPS Module [25]	2.7 - 3.6 V (3.0 V)	45mA- 60 mA	UART, SPI	
L80-M39 GPS Module [19]	3 - 4.3 V (3.3V)	45 mA	UART	Moisture sensitive - no more than 7 days exposure before being soldered to a PCB board

Table 1: GPS Module Research

Looking at the table, there are not many differences between the two except for the Neo-6m is lower power and the L80 has moisture sensitivity making it harder to work with. Thus, for now the Neo-6m seems like the best choice for the GPS module but that may change depending on which processor is chosen.

For the compass, two options arose from research and they are shown in the table below:

Compass	Operating Voltage	Current	Other Notes
HMC5883L 3 axis digital compass [7]	2.16 - 3.6 V	100 uA	
MMA8451 - Accelerometer, 3 Axis Sensor Evaluation Board [17]	1.95V to 3.6V	6 µA to 165 µA	I2C, Has built in tilt/orientation detection

Table 2: Compass Research

The power has the same maximum (3.6V) and the same current, but having the built-in orientation detector makes the MMA8451 a better option since an additional tilt sensor would be needed otherwise. Having a tilt sensor will be important for this project since it is important to know when the user has lifted their wrist and the compass direction detected agrees with the direction they are traveling. If their arm is by their side and moving along with them, the compass direction detected will not match up with their correct direction. Thus, the MMA8451 sensor is the better choice for the compass module.

For the display, the main constraint was the size. First, the different parts of what is being displayed was considered in terms of character length: the directions could take up to 20 characters to display, if the specific distance to the next point is added in (example: take a left in 15 feet). This means if using an LCD display it would take up 1 line of a 20 character display or 2 lines of a 16 character display, which are the two most common sizes. It does not make sense to display the user's current GPS coordinates since it would take up too much space and be hard for the user to easily decipher given the small changes in coordinates over a 1-3 mile hike. The

last things to consider for the display are the compass direction the user is traveling, which is 1-2 characters and the two distances (traveled and to-go) which would take around 14 characters. So a 2X16 LCD display would not be enough since the directions would take two lines and the 2nd line wouldn't have enough for the other information. A 2 X 20 could work but that becomes longer and thus more difficult to put on a wrist. A 4 X 20 display is just too big for a wrist device.

Thus after ruling out common LCD displays the next option considered was an OLED display. This turned out to be the better option for this device, given the size and flexibility of what could go on the screen. With the correct design, the screen would appear more like a typical watch compared to the LCD Screen. The graphics allow for more options such as arrows, which I will be using and the size of the characters can be changed with the arduino library. After some research on OLED displays an I2C option arised which works well considering the need to minimize the number of pins used. Thus the display for this project will be the 128 X 64 OLED graphic display which takes 3 or 5 V and has a 35mA current draw [7].

For the battery, while rechargeable batteries would be ideal given the nature of the device, a 6V battery pack is what is currently being used. It needs to be changed every 2-3 hikes since it lasts around 5 hours.

4.5 Block Diagram

Shown in Figure 5 is the block diagram. This gives an overview of connections necessary for everything to work together.

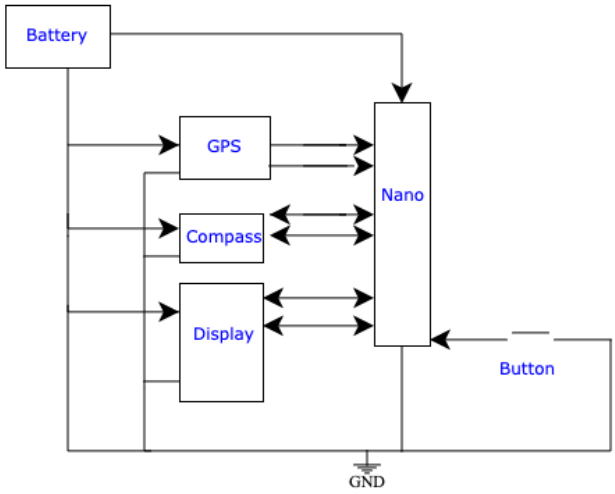


Figure 5: Block Diagram

As shown in the Block Diagram , the battery will power all the devices including the processor. Important connections to note that are shown in the block diagram are that the compass and display are both I2C devices, which saves pins. The GPS is a serial communication. Serial pins can be an issue for some microcontrollers, specifically arduino and thus a software serial library is used (more detail in software section).

4.6 Software Design

4.6.1 Trail Information

The trail is stored in the data in 3 different variables. The first is the trail name, which is stored in a character pointer named trail_name. It looks like the following for the “Union College Loop” that was used for testing: `char * trail_name = "Union College Loop";`

The next aspect of the trail is the coordinates that define the trail. This is important because the user’s location from the GPS is compared to the coordinates of the trail to see where

they are along it. The coordinates are stored in a 2D float array. The number of rows in the array is the number of points that were plotted when defining the trail and the number of columns is 2 because each point has a latitude and longitude value. The number of rows is stored in a variable "size" to keep track of the number of points that define the trail. The definition of the "list_of_coordinates" variable for the "Union College Loop" test trail is shown below (not all 45 points are included) : const float list_of_coordinates [][2] PROGMEM =

{{42.81899433349342,-73.93036384706402},{42.81877791772899,-73.93049259309673}, . . .}

The name includes the keyword "PROGMEM" meaning that the array is stored in flash memory instead of SRAM. While this is slower, it was necessary to be able to store all the necessary information on the trail.

The last aspect of the trail to store is the compass bearings or headings. This is the direction that the user is facing at each point as they travel along the trail. This is how the directions are calculated. Knowing the current direction of the user and the next direction along the trail, the correct arrow to be calculated and displayed. The headings are stored in a 1D float array that is the same size as the number of rows in the coordinate array, since every point has a compass bearing attached to it. The definition of this variable for the "Union College Loop" test trail is shown below (not all 45 points are included) : const int list_of_headings[] PROGMEM =

{208,208,208,118,118,117,117,169,165,169,245,243,242,117, . . . }

It also has the keyword PROGMEM for the same reason addressed above. These three variables (and the variable "size") define a trail and can be changed/updated to add new trails.

4.6.2 Setup and Global Variables

The setup for the code involves the initialization of the display, GPS and compass along with some global variables. These include the starting heading of the trail along with the first direction the user is to make (will be straight or ↑) , which is stored in the global current_dir. Current_dir and next_dir are used to calculate the direction to display and are both valued between 1-9 indicating the number of the arrow they represent. The conversion can be seen below in Figure 6:

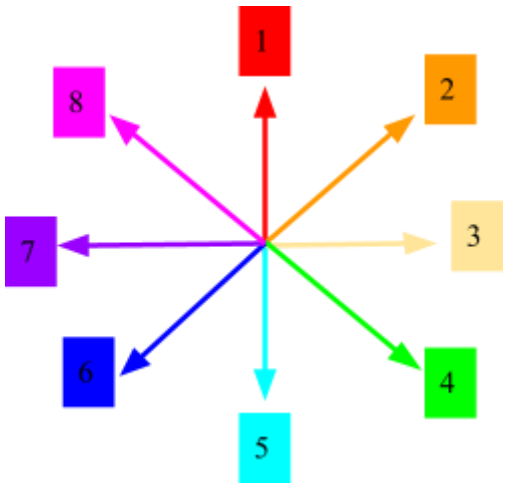


Figure 6: Conversion between Number and Arrow

Three global boolean values, GPSconnected, start and flag are set to false in the setup. GPSconnected indicates whether the GPS has connected to satellites and the start variable is used to indicate when the user has pressed the button to start the hike. The flag variable is used to know when to update to the next direction. It is initially set to false but when the user is close to the next point along the trail, it becomes true and stays true until they have passed that point and the next direction can be calculated. Its purpose is to keep the direction on the screen for the appropriate amount of time.

Other global variables include current_llat and current_lon which are the current latitude and longitude received from the GPS. Also num_coordiantes_checked and index which are variables that iterate through the list of coordinates as the user moves along the trail. Lastly, the global variable “arrow” is the arrow to display on the screen next and it is also a number between 1-9 following the pattern shown above in Figure 6.

4.6.3 Algorithm

The main algorithm is one large loop that compares the user’s current location on the trail with the list of coordinates that define the trail to determine the correct direction to display on the screen.

Going into further detail of the algorithm, before any calculation of directions is made, the code waits for the user to press the button to indicate that they want to start the hike. Once the button is pressed, the code to connect the GPS to satellites runs. Once the user has pressed the button, the GPS has connected, and there are still points the user hasn't hit (indicated by num_coordiantes_checked being less than the “size” variable) the code enters into the general algorithm for the direction calcatuing.

The first step in the algorithm is setting the iterative variable (num_coordinates_checked) to 0 if the user has just started the hike. Then the code to check the GPS connection runs again. This continually runs so that the GPS losing connection does not require the user to start the hike over. The GPS connecting code is shown below in Figure 7::

```
void connectGPS(){
 GPSconnected = false;
 for (unsigned long start = millis(); millis() - start < 1000;)
 {
   while (ss.available())
   {
```

```
        char c = ss.read();
        if (gps.encode(c)){
          GPSconnected = true;
      }
    }
    }
  }
. . .
```

Figure 7: Code for connecting the GPS

The next step is to get the user's current latitude and longitude (from the GPS) , which is done using a call to a library function. In addition to getting these values, the next latitude and longitude values along the trail are found. These two sets of points need to be compared against each other, which is shown below in Figure 8:

```
if(GPSconnected && (num_coordinates_checked < size)){
     get_GPS_location();
     float next_lat;
     float next_lon;
     next_lat = pgm_read_float(&list_of_coordinates[num_coordinates_checked][0]);
     next_lon = pgm_read_float(&list_of_coordinates[num_coordinates_checked][1]);

   // testing that the user has reached the next point
   if ( !(flag) && (abs(current_lat - next_lat) < 0.00015) && (abs(current_lon -
next_lon) < 0.00015)) {
     flag = true;
   }
. . .
```

Figure 8: Code for Comparison of Latitude and Longitude

These points are compared against each other as shown and 0.00015 was a number that was chosen after some testing to figure out the appropriate "closeness" to the next point while factoring in the fact that the accuracy of the GPS could cause problems such as not marking that

a user has reached a point when they have. A distance calculation would have been more accurate such as when a person is within 20 feet of the next point the direction would update but that was not able to be accomplished.

The points are compared as discussed above to indicate when the user is close enough to the next point so that the arrow to display can be calculated. Thus when the flag is true, the direction calculation occurs and that is seen below in Figure 9:

```
if (flag){
     next_heading = pgm_read_byte(&list_of_headings[num_coordinates_checked + 1]);
     next_direction = convert_heading_to_cardinal_direction(next_heading);
     next_dir = get_num_arrow(next_direction[0], next_direction[1]);
     if (next_dir + (9-current_dir) > 8){
       arrow = next_dir - (8 -(9-current_dir));
     }
     else{
       arrow = next_dir + (9-current_dir);
     }
. . .
```

Figure 9: Code for Calculating Direction to Display

Helper functions are called and the next arrow to display on the screen is stored in the variable arrow. This arrow should be displayed until the user has reached the point they are supposed to. That is where the last check comes in, shown below in Figure 10.

```
// keep on the current direction on the screen until you have reached the next point
     if((abs(current_lat - next_lat) > 0 ) && (abs(current_lon - next_lon) > 0 )){
       num_coordinates_checked = index;
       index = index + 1;
       current_dir = next_dir;
       flag = false;
     }
   }
   else{
   }
   display_main_screen(arrow);
```

```
 }
   if(num_coordinates_checked > (size - 1)){
     display_screen(2);
   }
. . .
```

Figure 10: Code for Updating Variables for Next Direction Calculation

Once the user has passed the point they are supposed to reach, all of the variables are updated as seen above in Figure 10. Throughout the hike the current arrow is displayed on the screen through the function call testScreen(arrow) and once the user has hit all the points the ending screen is displayed indicating they have completed the hike. This process continues until num_coordinates_checked is greater than variable "size" meaning that the user has hit all the points on the trail and thus completed the hike.

4.6.4 Functions

There are multiple functions that are used in the code. Starting with the screen functions, there is one function that is used to change between the two starting screens and the ending screen. The other screen function displays all the information to the user throughout the hike. This includes the compass direction which is received through a call to another function that calculates the direction based on the magnetic readings from the compass. The arrow is also displayed and is passed in as a parameter to this function. It is a number between 1 and 9, thus displaying the arrow is done through a series of if statements that draw the correct line and triangle depending on the value of "arrow". A piece of this code is shown below in Figure 11:

```
int xstart = 60;
 int  xend = 60;
 int  ystart = 30;
 int  yend = 37;
 int direction = arrow;
```

```
 if (direction == 1){
   display.fillTriangle(xstart, ystart - 4, xstart + 5, ystart, xstart-5, ystart,
WHITE);
 }
 if (direction == 2){
   xstart = 55;
   xend = 65;
   ystart = 35;
   yend = 28;
   display.fillTriangle(xend + 6, yend -3 , xend -3, yend - 3, xend + 3, yend+3,
WHITE);
 }
 if (direction == 3){
   xstart = 50;
   xend = 65;
   ystart = 31;
   yend = 31;
   display.fillTriangle(xend + 10, yend, xend, yend-4, xend, yend + 4, WHITE);
 }
. . .
```

Figure 11: Code for Drawing Arrow

The last two functions are helper functions to calculate the direction. The first one named "get_num_arrow" takes in individual characters of a direction for example "N" and "W" for "NW" or "N" and "" for N and converts it to the corresponding number according to Figure 6.

The last function "convert_heading_to_cardinal_direction" takes in an integer value for the heading and converts it to the correct cardinal direction according to Figure 12:

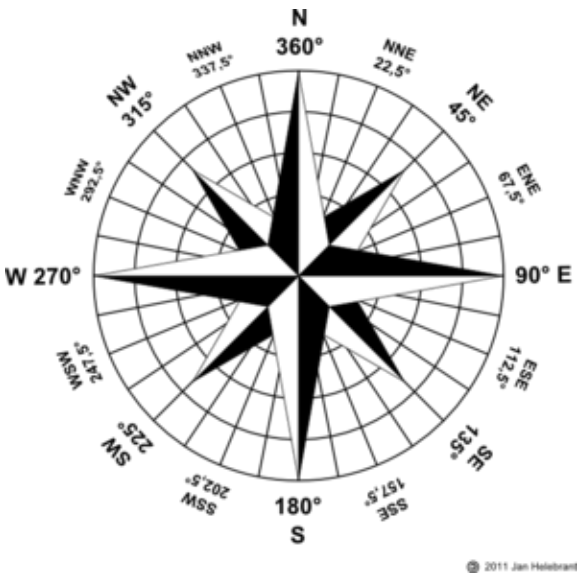


Figure 12: Chart for Converting Between Compass Bearing and Cardinal direction [5]

For example, "NE" is any value between 22.5 and 67.5, "E" is any value between 67.5 and 112.5, etc. Together all these functions work together to calculate the direction and display the necessary information to the user.

4.6.5 Memory

The final sketch used 25404 bytes (82%) of program storage space and the global variables used 876 bytes (42%) of dynamic memory. Some memory issues were encountered early on but were solved by using PROGMEM. Also most of the variables in the code are global variables since they are used in the main loop although some could probably become local variables if more time was able to be spent refactoring the code.

4.7 Design Conclusion

To conclude the discussion of the design, a comparison of this design implementation versus the design requirements is necessary. This design implementation, while it does not meet all the design requirements since it does not display the current time of the distances to the user, it does meet the major design requirement of this device which is that it gives directions to a user along a trail. This is accomplished by the arrows on the screen.

The device does make sure that the user does not get lost (by giving them directions) but it does not let them know when they have gone off trail. It will not update to the next direction if they have gone off trail but it will just continually show the previous direction until they get back on the trail. There is no buzzer so that aspect of the design requirements is not met.

One final aspect to address is the size of the device. For this final implementation the device is still on a breadboard and is not currently in the form of a watch but can still be carried on a trail (although not in the most convenient way).

5. Final Results and Design Evaluation

5.1 Component Testing

The individual components of the device needed to be tested. This included the GPS, compass and the display.

5.1.1 Compass Testing

Beginning with the compass, testing was done to make sure that it could accurately receive the compass bearing of the user. The compass in general is not accurate 100% of the time and that is due to the calibration that had to be done and the fact that there are other

electronic components interfering with the magnetic values that the compass is gathering. This could not be avoided since the compass component has to be close to the other components the device involves. To deal with the inconsistencies, sampling of the data was done so that the result was not skewed by outlier readings. Even with this precaution, it is not always accurate. The value would usually be within + or - 5/10 of the value obtained from an IPhone but the device just cared about the conversion of the raw number to a direction so most of the time it would give the correct compass direction such as N, S, etc.

5.1.2 GPS Testing

The GPS was tested to make sure that it was able to get the time and the current longitude and latitude of the user. The time could be received and was correct matching the current time in hours, minutes and seconds. The testing for the longitude and latitude showed that there was variation in readings in a stationery location but after a few seconds the values were steady and had a 0.0005% error.

5.1.3 Display Testing

Testing the display just tested that everything could be displayed on the screen. The following is a picture of everything displayed on the OLED display:

Figure 13: Screen from Display Testing

This was done right when the OLED display arrived, with hard-coded values to just make sure that everything could be displayed in a way that was visually appealing. This is not representative of what was actually able to be displayed but rather what could be displayed. It is a measure of the performance of the display independent of the other components and what the code looks like.

5.2 Behavior Testing

Here is an overview of the behavior of the device. When you first turn the device on you see this screen:

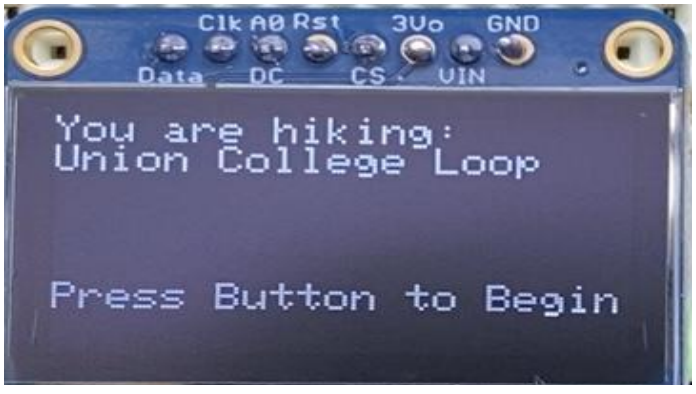


Figure 14: Starting Screen

The test trail was named “Union College Loop” but the name of whatever trail is currently downloaded to the device will appear on the second line. This screen will appear until you press the button indicating that you wish to start the hike. Then this screen will appear:

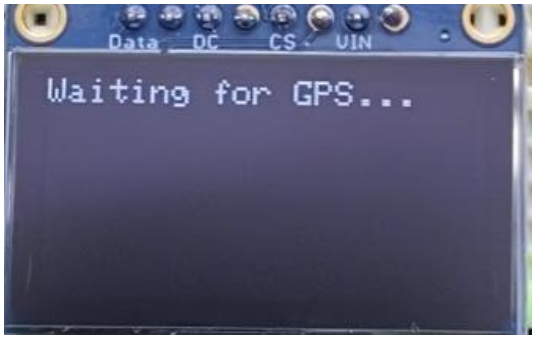


Figure 15: GPS connecting Screen

This tells the user that the GPS is connecting. The hike can’t start until it has been connected to satellites so that the user’s location along the trail can be determined. Once the GPS has connected the user sees the following screen showing information about their hike:

Figure 16: Current Screen Displayed during Hike

Once the hike is completed, the following screen is displayed:

Figure 17: Final Screen

While the time and distance could be received/calculated in the code, they were not able to be displayed on the screen. The reason for this was never resolved but was most likely due to library conflicts. Thus the display just displayed the direction the user was traveling and the arrow direction as shown above.

The compass direction is as accurate as the compass is at receiving the direction and that was discussed earlier in the component testing.

Quantifying how accurate the arrow directions are was challenging. The arrows update as the user goes along the trail and are just dependent on the user's location so the user can go fast or slow and the directions will appear at the correct time so the user can make the correct turn or stay straight. Finding the exact time to update the next direction was challenging and ended with a 0.0002 difference between the user's current longitude and latitude and the point they are supposed to hit next along the trail. This caused the arrow to update early enough so that the user had time to see the direction and know what to do. Although it was hard to quantify, through testing the arrows did make sure the user got from start to finish along the trail without getting lost.

5.3 Performance Evaluation

Comparing the current design to the original design specifications, the two components that were not implemented are the buzzer and the “mute” button for the buzzer. Due to time constraints for the project the buzzer was not able to be implemented and is referenced in the design changes section below. All of the other components were implemented and work as expected.

Looking at the behavioral aspects of the device, while there are aspects that do not work such as the time and the distances, overall the device works well as it is able to get the user from start to finish along the trail, providing real-time directions along the way. It also has the ability to add more trails as discussed above. However, it is not in the form of a watch which makes it hard to carry and somewhat difficult to use at the moment. The current battery being used is a 6V battery pack and needs to be changed every 2-3 hikes. These limitations are addressed in the design changes section.

5.4 Design Changes

Given that the design specifications stated that the distance and time would be displayed, getting those to work with the other information being displayed would improve the performance of the project. Also implementing the buzzer would meet the requirement of notifying the user when they went off trail.

In addition to that, having a pause button would improve performance, since it would be a useful feature to the user. Implementing this in such a way that the battery could be turned off during a pause would help save battery life too which would be a great feature. Right now if you step off trail and come back where you left off the device will continue to work since it is based

off where you are on the trail, however if you enter the trail a little ahead or behind where you left off it will not work because of the design of the code, thus a pause button would be useful.

Building off of what was stated above about the user, having to enter the trail where they left off the code right now is written in such a way that the user must follow the trail in a certain order or else the directions will not work. They must follow the trail in the correct order and not skip any portion of it. The code iterates through the trail and thus it must be followed in a specific way. Fixing the code and allowing the user to traverse the opposite direction or skip a part of the trail (or not have to enter in the exact same spot they entered) would improve the performance.

Lastly, making the device smaller would improve usability. Carrying it on a breadboard right now is not the most efficient and convenient way to track a hike so making it into a wearable device would be ideal. Making it wearable would also mean addressing the battery issue and the ideal solution would be using a rechargeable battery.

6. Discussions, Conclusions and Recommendations

6.1 Discussion

Starting all the way back in ECE-497 the goal in mind for this project was to create a device to help people try new hiking trails without having to spend a lot of money on a smart device. The device was supposed to be able to provide the user with real-time updates throughout their hike so they could try new trails without the fear of getting lost or having to worry about navigating the trail and can instead enjoy their time spent outside.

The design of this system included selecting components necessary to meet the design requirements. and putting it all together in hardware and software. The necessary components included a processor, a GPS to get the user's current location, a compass to get the direction the user is traveling and a display to present information about the hike to the user. All of these components were put together in hardware and software to get the final product.

The final product, while it did not meet all the requirements, it met the main requirement which was to provide correct directions to a user from start to finish along a hike. It lacked extra information about the hike such as the distance traveled and the current time.

6.2 Recommendations

Future work of this project will include working to get everything displayed together. This will come on my-own time but hopefully progress will be made before Steinmetz Day. There is an opportunity for future work in getting the size reduced for this project but may or not be explored.

Some of the important lessons learned from this project include overall design and software development. In terms of overall design, it is good to be ambitious but do not underestimate how much time it will take to implement everything together. That is where most of the time was spent in ECE-499: putting all the parts together. The components working individually was one battle but getting it all to work together was difficult and required a lot of thought and problem-solving skills.

In terms of software development, it is important to fully understand and outline an algorithm before implementing it. It always seems easier to start implementing and work through it as it progresses but a lot of effort went into changing and updating the final algorithm when

some of the problems could have been prevented with more thought before putting it into the IDE.

Another issue with software was the Arduino libraries. While they were very useful and allowed the project to be completed, they are not always written well or well-documented and can cause a lot of problems that may not arise right away.

6.3 Conclusion

In conclusion, this project implemented a device that can give directions along a trail of the user's choosing. The system may not have met every design requirement but the overall goal was accomplished. While problems did arise, mostly in software, valuable skills were learned in the process and the overall final product is something to be proud of.

7. User Manual

7.1 Assembly

Below is a picture of the device.

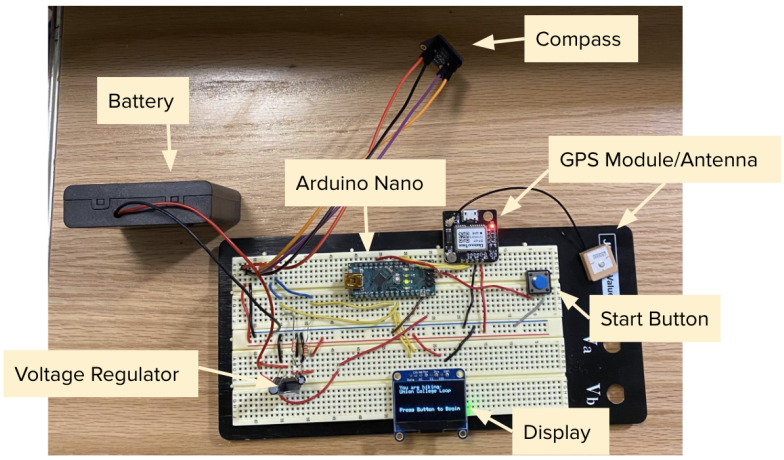


Figure 18: Device Prototype (with labels)

A cable is needed to connect the Arduino Nano to the computer to download new trails. The power of the battery is connected in row 10 near the voltage regulator and the ground is connected in the blue strip in the middle. Once the battery is turned on the reset button on the Arduino Nano can be pressed to get the starting screen.

The compass is on jumper wires so it can be positioned accordingly to represent the direction that the user is traveling. Adjusting it may be necessary before you start the hike, making sure that the head of the compass (opposite the pins - the side that says MMA8451) is facing to the left of the user as shown below in Figure 19. The antenna of the GPS can be moved around and should be positioned in the best way possible to be able to connect to satellites. It is shown facing up (circled in red) in Figure 19.

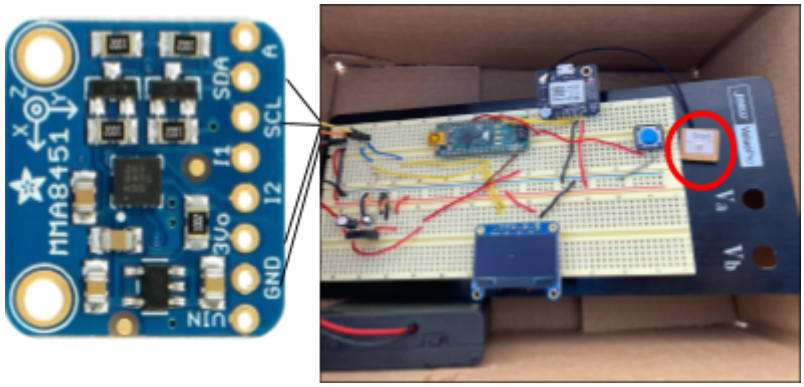


Figure 19: Compass Orientation and Antenna Positioning

7.2 Using the Device

To start using the device make sure the battery is turned on and you should see this screen (with your specific trail name on the line below “You are hiking:”)

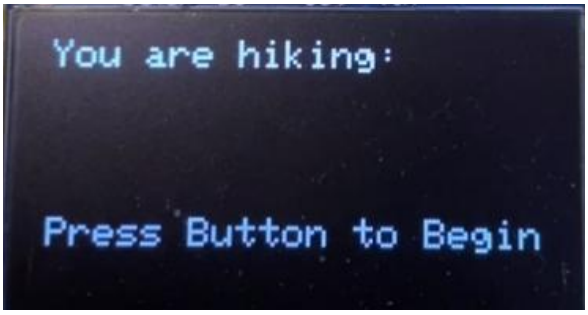


Figure 20: Starting Screen

If you do not see this screen, check your batteries and make sure everything is still plugged in. Once you see this screen you should also notice that the light of the GPS has turned on. Note the trail name displayed on the screen, more information on how to change this is given below. Once you are ready to hike this trail, press the button. Then you should see this screen:

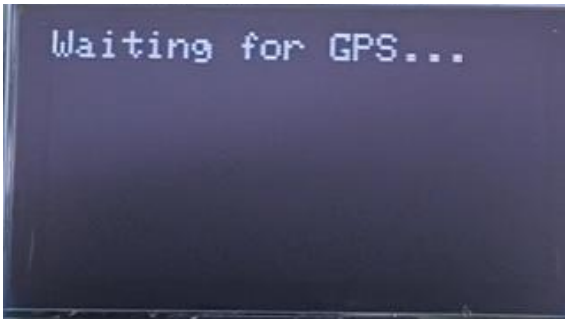


Figure 21: GPS Connecting Screen

This screen will appear until the GPS has connected. Make sure the antenna of the GPS is not being blocked. Once the light starts flashing on the GPS it means it is connecting but the screen will change automatically when it connects.

Once it has connected, you will get the screen below where the direction you are current traveling with be displayed next to “Traveling”:

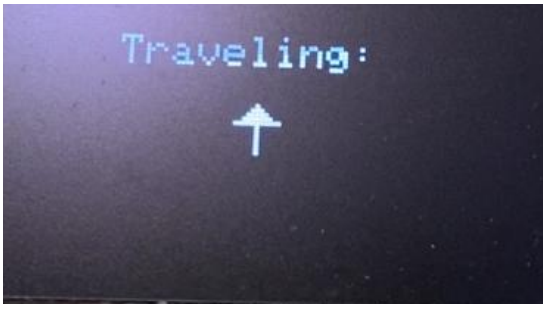


Figure 22: Current Screen Displayed during Hike

This is the screen you should see the entire hike. It will update in real-time to provide you the correct direction to take and direction you are traveling. Once you are done with the hike you will see this screen:

Figure 23: Final Screen

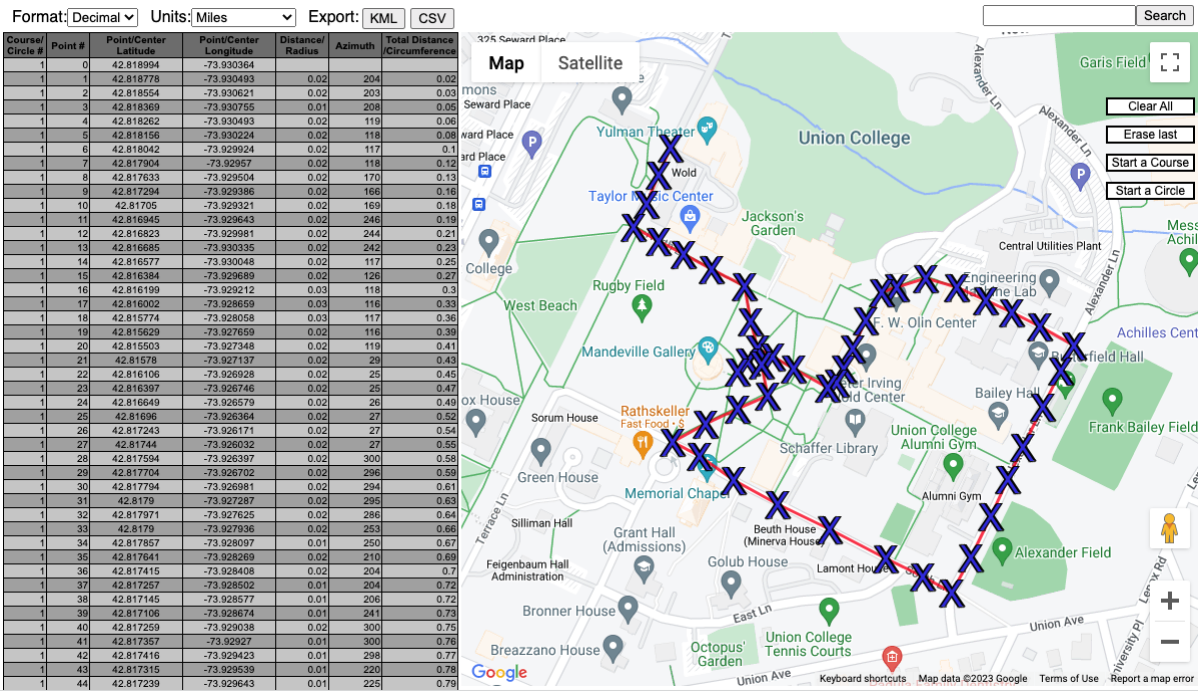
To store your device make sure to turn the battery off. Before your next hike update and download the code if you wish to try a different trail.

7.3 Adding a New Trail

The process of adding a new trail involves the following tools:

1. https://www.acscdg.com/
2. The Python Script

To start, go to the website and zoom in on the area where you want to map the trail. For the table on the left make sure the settings on the side say Format: “Decimal” and Unit: “Miles”. Now you can start mapping the trail by pressing “Start a Course” and then clicking along the trail so that the blue X’s appear. Make sure to map it in the order that you want to complete the trail and that there is an X before and after every turn. Your screen should look like this (for your test trail):



Course/ Circle #	Point #	Point/Center Latitude	Point/Center Longitude	Distance/ Radius	Azimuth	Total Distance /Circumference
1	0	42.818994	-73.930364			
1	1	42.818778	-73.930493	0.02	204	0.02
1	2	42.818554	-73.930621	0.02	203	0.03
1	3	42.818369	-73.930755	0.01	208	0.05
1	4	42.818262	-73.930493	0.02	119	0.06
1	5	42.818156	-73.930224	0.02	118	0.08
1	6	42.818042	-73.929924	0.02	117	0.1
1	7	42.817904	-73.92957	0.02	118	0.12
1	8	42.817633	-73.929504	0.02	170	0.13
1	9	42.817294	-73.929386	0.02	166	0.16
1	10	42.81705	-73.929321	0.02	169	0.18
1	11	42.816945	-73.929643	0.02	246	0.19
1	12	42.816823	-73.929981	0.02	244	0.21
1	13	42.816685	-73.930335	0.02	242	0.23
1	14	42.816577	-73.930048	0.02	117	0.25
1	15	42.816384	-73.929689	0.02	126	0.27
1	16	42.816199	-73.929212	0.03	118	0.3
1	17	42.816002	-73.928659	0.03	116	0.33
1	18	42.815774	-73.928058	0.03	117	0.36
1	19	42.815629	-73.927659	0.02	116	0.39
1	20	42.815503	-73.927348	0.02	119	0.41
1	21	42.81578	-73.927137	0.02	29	0.43
1	22	42.816106	-73.926928	0.02	25	0.45
1	23	42.816397	-73.926746	0.02	25	0.47
1	24	42.816649	-73.926579	0.02	26	0.49
1	25	42.81696	-73.926364	0.02	27	0.52
1	26	42.817243	-73.926171	0.02	27	0.54
1	27	42.81744	-73.926032	0.02	27	0.55
1	28	42.817594	-73.926397	0.02	300	0.58
1	29	42.817704	-73.926702	0.02	296	0.59
1	30	42.817794	-73.926981	0.02	294	0.61
1	31	42.8179	-73.927287	0.02	295	0.63
1	32	42.817971	-73.927625	0.02	286	0.64
1	33	42.8179	-73.927936	0.02	253	0.66
1	34	42.817857	-73.928097	0.01	250	0.67
1	35	42.817641	-73.928269	0.02	210	0.69
1	36	42.817415	-73.928408	0.02	204	0.7
1	37	42.817257	-73.928502	0.01	204	0.72
1	38	42.817145	-73.928577	0.01	206	0.72
1	39	42.817106	-73.928674	0.01	241	0.73
1	40	42.817259	-73.929038	0.02	300	0.75
1	41	42.817357	-73.92927	0.01	300	0.76
1	42	42.817416	-73.929423	0.01	298	0.77
1	43	42.817315	-73.929539	0.01	220	0.78
1	44	42.817239	-73.929643	0.01	225	0.79

Figure 24: Example for Adding a Trail

When this is done, you can download the table by pressing the CSV file. Then make sure that the python code is somewhere on your computer that you can run it and put the CSV in the same file as the code and name it ‘Trail_Data.csv’. Run the python script and the output is the information the arduino code needs. You can copy and paste it directly into the main file of the arduino code (since the names of the variables are already there), under the trail information.

You can store multiple trails just change the name of the variables to account for that and make sure the trail you want is stored in the following variables:

- char * trailName
- const float list_of_coordinates [][2] PROGMEM
- const int list_of_headings[] PROGMEM

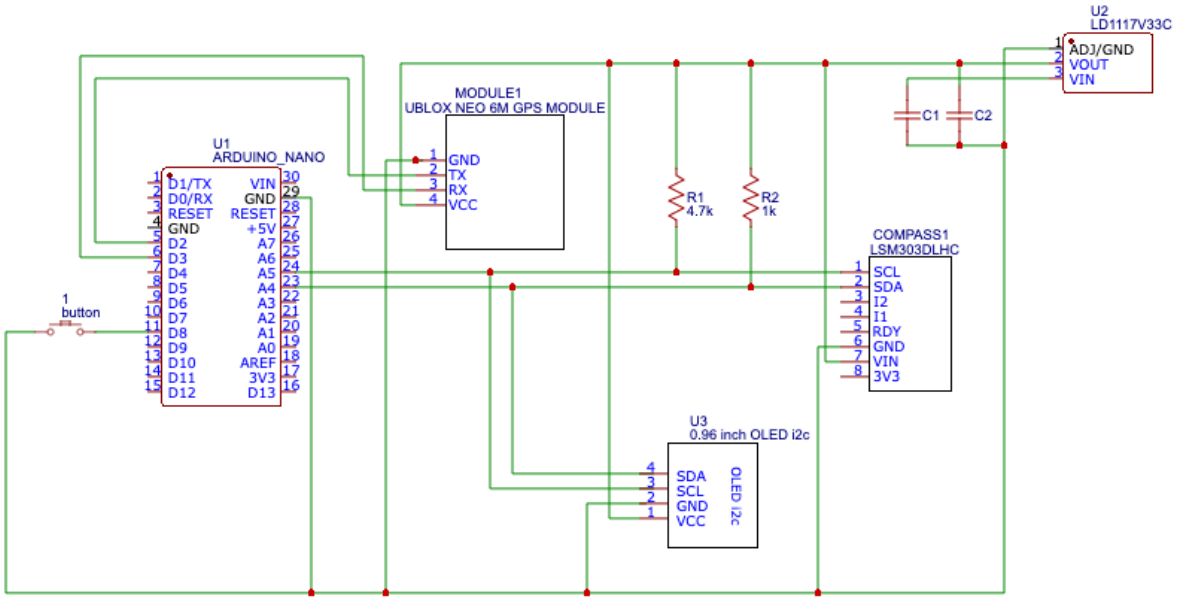
8. References

[1] Atmel “8-bit AVR Microcontrollers” ATmega328/P datasheet, Nov 2016
[2] “Apple Watch - Compare Models.” *Apple*, www.apple.com/watch/compare/
[3] Carroll, Vicki. “What's Killing America's Hikers?” SkyAboveUs, SkyAboveUs, 10 Oct. 2012, https://skyaboveus.com/climbing-hiking/Whats-Killing-Americas-Hikers.
[4] Chris Veness, www.movable-type.co.uk. “Movable Type Scripts.” *Calculate Distance and Bearing between Two Latitude/Longitude Points Using Haversine Formula in JavaScript*, https://www.movable-type.co.uk/scripts/latlong.html.
[5] “Direction and Bearing.” *Learning Journeys*, http://ajltct.weebly.com/direction-and-bearing.html.
[6] Garmin, and Garmin Ltd. or its subsidiaries. “Garmin Fēnix® 5S plus: Multisport GPS Watch.” Garmin, 2018, www.garmin.com/en-US/p/603201/pn/010-01987-00.
[7] Honeywell. *3-Axis Digital Compass IC HMC5883L - Adafruit Industries*. cdn-shop.adafruit.com/datasheets/HMC5883L_3-Axis_Digital_Compass_IC.pdf.
[8]Gilmour, Mike. “Hiking Accidents Statistics: 18 Facts Trends to Consider (Explained).” *RV and Playa*, 31 Oct. 2021, https://www.rvandplaya.com/hiking-accidents-statistics-facts-trends-to-know/#:~:text=About%2 02000%20people%20get%20lost,which%20can%20lead%20to%20death.
[9] Industries, Adafruit. “Flora - Wearable Electronic Platform: Arduino-Compatible.” *Adafruit Industries Blog RSS*, www.adafruit.com/product/659.
[10] Industries, Adafruit. “Monochrome 1.3’ 128x64 OLED Graphic Display - Stemma QT / Qwiic.” *Adafruit Industries Blog RSS*, www.adafruit.com/product/938.
[11] Industries, Adafruit. “3.3V 800mA Linear Voltage Regulator - LD1117-3.3 to-220.” *Adafruit Industries Blog RSS*, www.adafruit.com/product/2165?gclid=CjwKCAjw79iaBhAJEiwAPYwoCObQtkkWW5vQ3eV cOlPNJfWbIDa21Cr1basf1p8ltTCSWQxLgj1HeRoC7YcQAvD_BwE.
[12] “ISO 6709:2022: Standard Representation of Geographic Point Location by Coordinates.” ISO, 27 Sep. 2022, www.iso.org/standard/75147.html.
[13] *I2C-Bus Specification and User Manual - Pololu*. www.pololu.com/file/0J435/UM10204.pdf
[14] Kidd, Jason. “How Many Hikers Get Lost a Year? What to Do If You Get Lost.” Outside Pulse, 1 Apr. 2022.
[15]“Lilypad Arduino 328 Main Board.” *DEV-13342 - SparkFun Electronics*, https://www.sparkfun.com/products/13342.
[16] “Mapping and Distance Tools.” *Mapping and Distance Tools*, https://www.acscdg.com/
[17] “MMA8451 3AXIS ACCEL BREAKOUT BRD.” *Digikey Electronics*, 2019, www.digikey.com/en/products/detail/adafruit-industries-llc/2019/4990790?utm_adgroup=Evaluation+Boards+-+Sensors&utm_source=google&utm_medium=cpc&utm_campaign=Shopping_Product_Development+Boards%2C+Kits%2C+Programmers_NEW&utm_term=&utm_content=Evaluation+Boards+-+Sensors&gclid=Cj0KCQjw--2aBhD5ARIsALiRlwCiOJzNemABAiglQ8nr H8q3KBHPkoSASQ6tr48alP9e-NBuu_N28soaAqZwEALw_wcB.
[18] “PROGMEM - Arduino Reference.” *PROGMEM*,

https://www.arduino.cc/reference/en/language/variables/utilities/progmem/.
[19] Quectel. *L80 Hardware Design - Mouser Electronics*.
www.mouser.com/datasheet/2/1052/QWSC_S_A0004134860_1-2576267.pdf .
[20] Roupioz, Laure. “Improvement of Hiking Network for GPS Users.” *Wageningen University & Research EDepot*, 2008, edepot.wur.nl/.
[21] S. Seneviratne et al., "A Survey of Wearable Devices and Challenges," in IEEE Communications Surveys & Tutorials, vol. 19, no. 4, pp. 2573-2620, Fourthquarter 2017, doi: 10.1109/COMST.2017.2731979.
[22] “Smart Watches Dimensions & Drawings.” *Dimensions & Drawings | Dimensions.com*, https://www.dimensions.com/collection/smart-watches.
[23] Stern, Becky. “Getting Started with Flora.” *Adafruit Learning System*, 2012, learn.adafruit.com/getting-started-with-flora/overview.
[24] “3.7V 100mAh Rechargeable Lithium Battery LIPO 401220.” *EElectronicParts*, www.eelectronicparts.com/products/3-7v-100mah-rechargeable-lithium-battery-lipo-401220-100 -mah-replace-65mah-70mah-80mah.
[24] “Ublox NEO-6m GPS Datasheet.” ElectronicWings, www.electronicwings.com/components/ublox-neo-6m-gps/1/datasheet
[25] “VXO7803-500.” *Digikey Electronics*, www.digikey.com/en/products/detail/cui-inc./VXO7803-500/7350287?utm_adgroup=DC+DC+Converters&utm_source=google&utm_medium=cpc&utm_campaign=Shopping_Product_Power+Supplies+-+Board+Mount_NEW&utm_term=&utm_content=DC+DC+Converters&gclid=CjwKCAjw79iaBhAJEiwAPYwoCDKMQADeV1tFvabiE5YCo-GR4yFt01i6d6YkRTaAzEgqe_VKV KPSkhoCO-8QAvD_BwE.
[26] Zandbergen, P., & Barbeau, S. (2011). Positional Accuracy of Assisted GPS Data from High-Sensitivity GPS-enabled Mobile Phones. Journal of Navigation, 64(3), 381-399. doi:10.1017/S0373463311000051

9. Appendix

Appendix A


U2
LD1117V33C
ADJ/GND
VOUT
VIN
C1
C2
MODULE1
UBLOX NEO 6M GPS MODULE
GND
TX
RX
VCC
U1
ARDUINO_NANO
D1/TX
D0/RX
RESET
GND
D2
D3
D4
D5
D6
D7
D8
D9
D10
D11
D12
VIN
GND
RESET
+5V
A7
A6
A5
A4
A3
A2
A1
A0
AREF
3V3
D13
R1
4.7k
R2
1k
COMPASS1
LSM303DLHC
SCL
SDA
I2
I1
RDY
GND
VIN
3V3
1
button
U3
0.96 inch OLED i2c
SDA
SCL
GND
VCC
OLED i2c

Appendix B

```
#include <Wire.h>
#include <math.h>
#define button 8

// Global Variables
int index;
int num_coordinates_checked;
int arrow;
int current_dir;
int next_dir;
int starting_heading;
char * starting_direction;
int next_heading;
char * next_direction;
float current_lat, current_lon;
boolean buttonState;
boolean prevButtonState;
boolean start;
boolean flag;
boolean GPSconnected;

// Trail Information
char * trailName = "Union College Loop";
const float list_of_coordinates [][2] PROGMEM =
{{42.81899433349342,-73.93036384706402},{42.81877791772899,-73.93049259309673},{42.818
553631501125,-73.93062133912944},{42.81836869312232,-73.93075544958019},{42.8182624516
7584,-73.93049259309673},{42.8181562100468,-73.93022437219524},{42.818042098464176,-73
.92992396478557},{42.8179043773081,-73.9295699131956},{42.817632868987744,-73.92950364
850638},{42.81729446564417,-73.92938563130973},{42.81705049929419,-73.92932125829337},
{42.81694509608333,-73.92964312337516},{42.81682311233947,-73.92998108171103},{42.8166
8538846847,-73.930335133301},{42.81657688131458,-73.93004835317667},{42.81638406704155
4,-73.92968893716868},{42.81619912217393,-73.92921150396403},{42.81600237170753,-73.92
865896890696},{42.8157741403823,-73.92805815408762},{42.815628544096796,-73.9276588234
8775},{42.81550262270859,-73.92734768724202},{42.81577951729378,-73.92713749211846},{4
2.81610612390208,-73.9269282798153},{42.816397313881396,-73.92674588960229},{42.816649
15275778,-73.92657892777395},{42.81696001495644,-73.92636435105275},{42.81724333104215
,-73.92617123200368},{42.817440077560185,-73.92603175713491},{42.81759353940969,-73.92
639653756093},{42.817703716912916,-73.92670230938863},{42.81779421971515,-73.926981259
12618},{42.817900461966126,-73.92728703095388},{42.81797129003204,-73.92762498928975},
```

```
{42.817900461966126,-73.92793612553548},{42.81785717810812,-73.92809705807637},{42.817
64075836369,-73.92826871945333},{42.81741450054825,-73.92840832005065},{42.81725696494
968,-73.92850226027342},{42.81714510768223,-73.92857709454458},{42.81710575824099,-73.
92867365406912},{42.81725922092019,-73.92903843449514},{42.81735659716701,-73.92927042
739855},{42.81741562108006,-73.9294233133124},{42.81731528039437,-73.92953864830004},{
42.817238549171954,-73.92964325445162}};
const int list_of_headings[] PROGMEM =
{208,208,208,118,118,117,117,169,165,169,250,250,250,117,126,117,115,117,116,118,29,25
,24,25,26,26,27,299,296,293,295,285,252,249,201,201,200,201,250,250,250,250,220,225};


void setup()
{
 compass_init();
 GPS_init();
 display_init();
  // button
 pinMode(button, INPUT_PULLUP);

 // set variables
 num_coordinates_checked = 0;
 starting_heading = pgm_read_byte(&list_of_headings[0]);
 starting_direction =  convert_heading_to_cardinal_direction(starting_heading);
 current_dir = get_num_arrow(starting_direction[0], starting_direction[1]);
 GPSconnected = false;
 flag = false;
 start = false;
}
```

Appendix C

```
#include <Adafruit_MMA8451.h>
Adafruit_MMA8451 mma = Adafruit_MMA8451(0x1D);
#define COMPASS_ADDRESS 0x1D ///< See datasheet for Address; 0x3D for 128x64, 0x3C for
128x32

void compass_init(){
 mma.begin(0x1D);
 mma.setRange(MMA8451_RANGE_2_G);
}

char * get_compass_direction(void){
 Wire.beginTransmission(COMPASS_ADDRESS);
 float sum = 0;
 // average samples
 for(int j = 0; j < 100; j++){
   mma.read();
   float heading = atan2(-mma.x, -mma.y);
   float declinationAngle = -0.24; // 14 degrees to radians
   heading = (heading + declinationAngle) * 180/PI;
   if (heading < 0){
     heading = heading + 360;
   }
   if(heading > 360){
     heading -= 360;
   }
    sum += heading;
  }
  float heading = sum/100;
  char * direction = convert_heading_to_cardinal_direction(heading);
  Wire.endTransmission();
 return direction;
}
```

Appendix D

```
#include <TinyGPS.h>
#include <SoftwareSerial.h>

TinyGPS gps;
SoftwareSerial ss(4, 3);

unsigned long age;
unsigned long date, _time;

void GPS_init(){
 ss.begin(9600);
}
void connectGPS(){
 GPSconnected = false;
 for (unsigned long start = millis(); millis() - start < 1000;)
 {
   while (ss.available())
   {
     char c = ss.read();
     if (gps.encode(c)){
       GPSconnected = true;
   }
 }
 }
}

void get_GPS_location(){
 gps.f_get_position(&current_lat, &current_lon, &age);
}
```

Appendix E

```
#include <Adafruit_SSD1306.h>
#define SCREEN_WIDTH 128
#define SCREEN_HEIGHT 64
#define OLED_RESET     -1
#define SCREEN_ADDRESS 0x3D // 0x3D for 128x64
Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, OLED_RESET);

void display_check(){
 Wire.beginTransmission(SCREEN_ADDRESS);
 display.display();
 Wire.endTransmission();
}

void display_init(){
 Serial.begin(9600);
 if (!display.begin(SSD1306_SWITCHCAPVCC, SCREEN_ADDRESS))
 {
   Serial.println(F("SSD1306 allocation failed"));
   for (;;);
 }
 display.clearDisplay();
 display.display();
}

void display_screen(int v){
 Wire.beginTransmission(SCREEN_ADDRESS);
 display.clearDisplay();
 display.setTextSize(1);
 display.setTextColor(WHITE);
 display.setCursor(0,0);
 if(v == 0){
   display.println(F("You are hiking: "));
   display.println(trailName);
   display.setCursor(0,45);
   display.println(F("Press Button to Begin"));
 }
 else if (v == 1){
   display.println(F("Waiting for GPS..."));
 }
```

```
 else{
   display.println(F("Hike Complete"));
 }
 display.display();
 Wire.endTransmission();
}

void welcome_screen(){
 Wire.beginTransmission(SCREEN_ADDRESS);
 display.clearDisplay();
 display.setTextSize(1);
 display.setTextColor(WHITE);
 display.setCursor(0,0);
 display.println(F("You are hiking: "));
 display.println(trailName);
 display.setCursor(0,45);
 display.println(F("Press Button to Begin"));
}

void display_main_screen(int arrow){
 Wire.beginTransmission(SCREEN_ADDRESS);
 display.clearDisplay();
 display.setTextSize(1);
 display.setTextColor(WHITE);
 display.setCursor(28,10);
 display.print(F("Traveling: "));
 display.println(get_compass_direction());

 int xstart = 60;
 int  xend = 60;
 int  ystart = 30;
 int  yend = 37;
 int direction = arrow;
 if (direction == 1){
   display.fillTriangle(xstart, ystart - 4, xstart + 5, ystart, xstart-5, ystart,
WHITE);
 }
 if (direction == 2){
   xstart = 55;
   xend = 65;
   ystart = 35;
   yend = 28;
```

```
   display.fillTriangle(xend + 6, yend -3 , xend -3, yend - 3, xend + 3, yend+3,
WHITE);
 }
 if (direction == 3){
   xstart = 50;
   xend = 65;
   ystart = 31;
   yend = 31;
   display.fillTriangle(xend + 10, yend, xend, yend-4, xend, yend + 4, WHITE);
 }
 if (direction == 4){
   xstart = 70;
   xend = 50;
   ystart = 32;
   yend = 25;
  display.fillTriangle(xstart - 2*sqrt(2), ystart + 2*sqrt(2), xstart + 2*sqrt(2),
ystart -2*sqrt(2), xstart + 5*sqrt(2), ystart + 2*sqrt(2),  WHITE);
 }
 if (direction == 5){
   xstart = 62;
   xend = 62;
   ystart = 24;
   yend = 32;
   display.fillTriangle(xend, yend + 5, xend +4, yend, xend-5, yend, WHITE);
 }
 if (direction == 6){
   xstart = 75;
   xend = 55;
   ystart = 25;
   yend = 32;
   display.fillTriangle(xend + 4*sqrt(2), yend + 2*sqrt(2), xend + 1.5*sqrt(2), yend -
1.5*sqrt(2), xend - 1.5*sqrt(2), yend + 1.5*sqrt(2),  WHITE);
 }
 if (direction == 7){
   xstart = 55;
   xend = 70;
   ystart = 31;
   yend = 31;
   display.fillTriangle(xstart - 8, ystart, xstart, ystart-3, xstart, ystart + 3,
WHITE);
 }
 if (direction == 8){
```

```
   xstart = 70;
   xend = 50;
   ystart = 39;
   yend = 28;
   display.fillTriangle(xend - 3*sqrt(2), yend -2*sqrt(2), xend + 2*sqrt(2), yend
+5*sqrt(2), xend +4*sqrt(2), yend -1.5*sqrt(2),  WHITE);
 }
 display.drawLine(xstart,ystart,xend, yend, WHITE);
 display.display();
 Wire.endTransmission();
}
```

Appendix F

```
void loop()
{
 if(!start){
   display_screen(0);
 }
 if(start && !(GPSconnected)){
   display_screen(1);
   connectGPS();
 }
 if(start && GPSconnected){
   int size = sizeof(list_of_coordinates) / sizeof(list_of_coordinates[0]);
   connectGPS();

   if(GPSconnected && (num_coordinates_checked < size)){
     get_GPS_location();
     float next_lat;
     float next_lon;

     next_lat = pgm_read_float(&list_of_coordinates[num_coordinates_checked][0]);
     next_lon = pgm_read_float(&list_of_coordinates[num_coordinates_checked][1]);

   if(index == 0){
     num_coordinates_checked = 0;
   }
   else{
     num_coordinates_checked = num_coordinates_checked;
   }
   display_check();

   // testing that the user has reached the next point
   if ( !(flag) && (abs(current_lat - next_lat) < 0.00015) && (abs(current_lon -
next_lon) < 0.00015)) {
     flag = true;
   }
   else{
   }
   if (flag){
     next_heading = pgm_read_byte(&list_of_headings[num_coordinates_checked + 1]);
     next_direction = convert_heading_to_cardinal_direction(next_heading);
```

```
      next_dir = get_num_arrow(next_direction[0], next_direction[1]);
      if (next_dir + (9-current_dir) > 8){
        arrow = next_dir - (8 -(9-current_dir));
      }
      else{
        arrow = next_dir + (9-current_dir);
      }
      // keep on the current direction on the screen until you have reached the next
point
      if((abs(current_lat - next_lat) > 0 ) && (abs(current_lon - next_lon) > 0 )){
        num_coordinates_checked = index;
        index = index + 1;
        current_dir = next_dir;
        flag = false;
      }
    }
    else{
    }
    display_main_screen(arrow);
  }
    if(num_coordinates_checked > (size - 1)){
      display_screen(2);
    }
  }

  buttonState = digitalRead(button);
  if((buttonState==LOW) && (prevButtonState==HIGH)) {
    start = !start;
  }
  prevButtonState=buttonState;
  }
```

Appendix G

```
int get_num_arrow(char direction1, char direction2){
 if ((direction1 == 'S') && (direction2 == 'W')){
   return 6;
 }
 else if ((direction1 == 'N') && (direction2 == 'E')){
   return 2;
 }
 else if ((direction1 == 'S') && (direction2 == 'E')){
   return 4;
 }
 else if ((direction1 == 'N') && (direction2 == 'W')){
   return 8;
 }
 else if (direction1 == 'N'){
   return 1;
 }
 else if (direction1 == 'E'){
   return 3;
 }
 else if (direction1 == 'S'){
   return 5;
 }
 else if (direction1 == 'W'){
   return 7;
 }
 else{
   return -1;
 }
}

char * convert_heading_to_cardinal_direction(int heading){
 if (((heading >= 0) &&   (heading < 22.5)) || ((heading >= 337.5) &&  (heading <
360))) {
   return "N";
 }
 else if ( (heading >= 22.5) &&  (heading < 67.5)) {
   return "NE";
 }
 else if ( (heading >= 67.5) &&  (heading < 112.5)){
```

```
    return "E";
  }
  else if ( (heading >= 112.5) &&  (heading < 157.5)){
    return "SE";
  }
  else if ( (heading >= 157.5) &&  (heading < 202.5)){
    return "S";
  }
  else if ( (heading >= 202.5) &&  (heading < 247.5)){
    return "SW";
  }
  else if ( (heading >= 247.5) &&  (heading < 292.5)){
    return "W";
  }
  else if ( (heading >= 292.5) &&  (heading < 337.5)){
    return "NW";
  }
  else{
    return " ";
  }
}
```

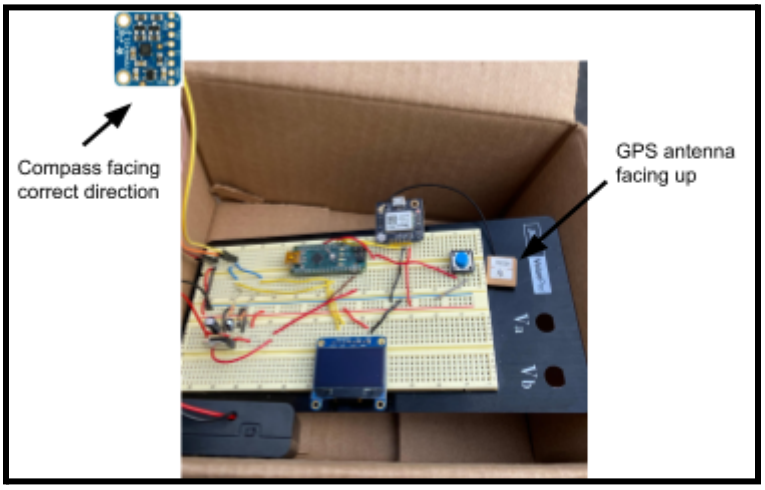
Appendix H

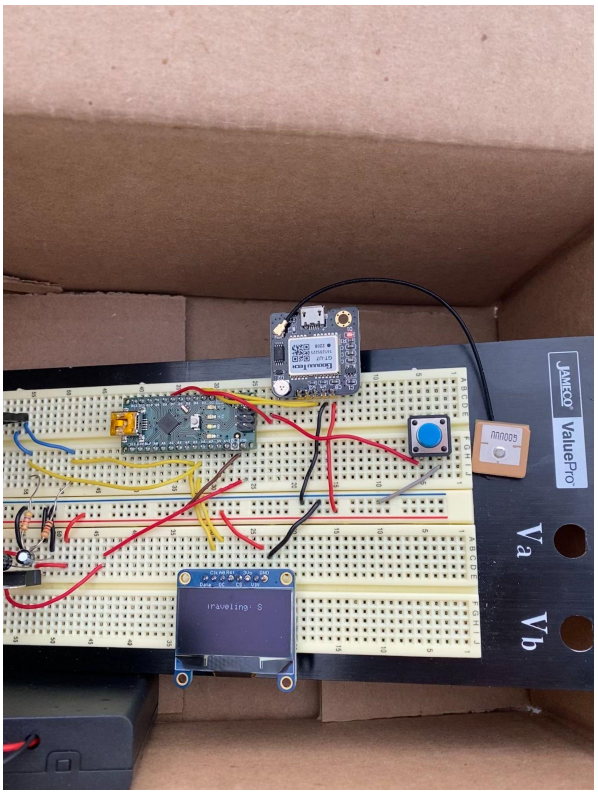
```
gps.get_datetime(&date, &_time, &age);
 // parse time
int hour = (_time / 1000000) -5;
int min = (_time / 10000) % 100;
display.print(hour);
display.print(F(" :"));
display.print(min);
```

Appendix I

```
 float R = 6373.0;
 current_lat = current_lat * (3.14/180);
 current_lon = current_lon * (3.14/180);
 next_lat = next_lat * (3.14/180);
 next_lon = next_lon * (3.14/180);

 float dlon = next_lon - flon;
 float dlat = next_lat - flat;

 float a = pow(sin(dlat / 2),2) + cos(current_lat) * cos(next_lat) *
pow(sin(dlon / 2),2);
 float c = 2 * atan2(sqrt(a), sqrt(1 - a));

// add distance between current and next to the previous distance traveled
// display once you have reached the next point
 distance = distance + (R *c);
 display.setCursor(45,43);
 display.print(distance/1.609, 2);
 display.println(F(" MI"));
 //display.print(F("Distance left: "));
 //display.println(trail_distance - distance);
 display.display();
```

Appendix J[1]





[2]

[1] During testing, due to the Frame rate of the IPhone Camera and the sunlight both the direction and arrow were not able to be captured at the same time which is why the pictures only have one or neither showing.

[2] This picture was taken near Wicker Wellness Center and shows that a left is to be taken ahead (as expected).

Appendix K

Hiking Tracker

ECE-499
Jennifer Williamson
Advisor: Professor Traver

Goals + Context

Goal: Create a stand-alone low cost device to give a hiker directions

Why: Devices on the market are:

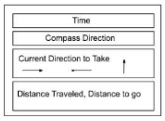
- Expensive
- Need a phone paired with it
- Have a lot more functionality

Who:

- New hikers
- People who want to try new hiking trails

Specifications + Standards

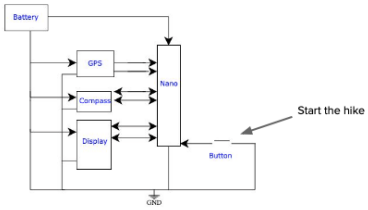
- **Behavior**:
  - Directions along a predetermined trail (0.8 miles on campus)
  - Ability to add more trails
    - Around 1-2 miles long
- **Display:**
  - In real-time



Test Trail

0.79 Miles

Design - Top level Block Diagram



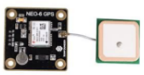
Design - Components

Processor - Arduino Nano

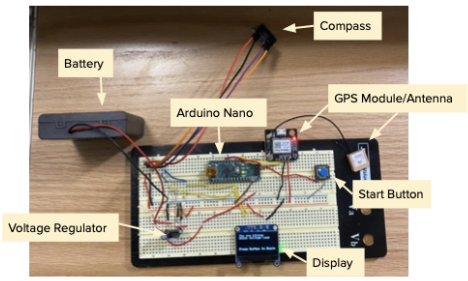
Display - 128x64 OLED graphic

Compass - MMA8451

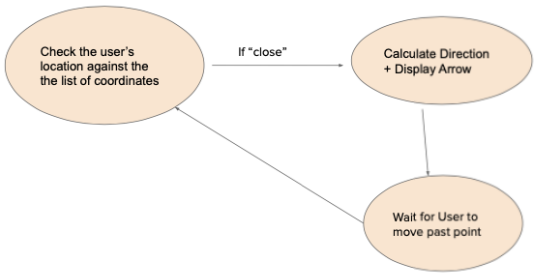
GPS - NEO-6M



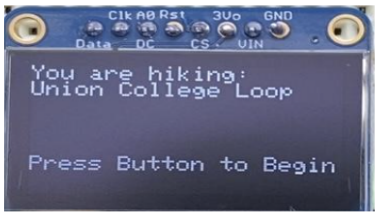
Design - Prototype



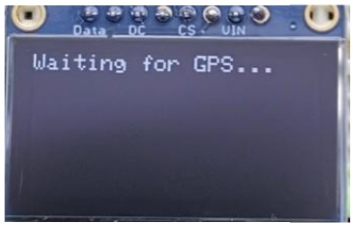
Software Design



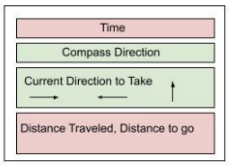
Design - Screen



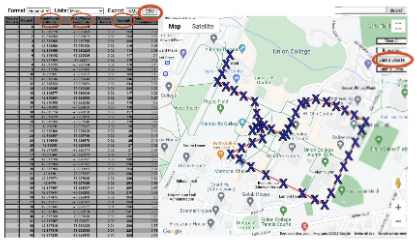
Design - Screen



Testing



Adding a Trail



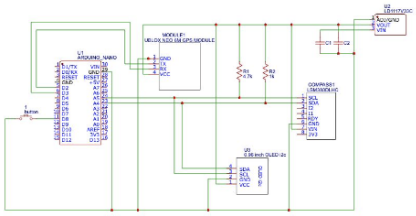
Results + Conclusions

Meets Specifications	Still Needs Work
Display ❖ Directions in real-time ❖ Compass directions Trail Addition	Display ❖ Distance ❖ Time

Lessons:
- ❖ Start merging code early
- ❖ Arduino libraries

Questions?

Appendix - Hardware Schematic



Appendix - Power and Current

Component	Voltage	Current Consumption
GPS	2.7V - 3.6V	60mA
Compass	1.95V - 3.6V	100uA
Display	3.3V (3 or 5 since it is has a built in regulator)	25mA
Nano	5V	19mA

Total Power consumption: 100mA

Appendix - Code

```
if(GPSconnected){
   gps.f_get_position(&flat, &flon, &age);
float next_lat;
float next_lon;
if(index == 0){ num_coordinates_checked = 0;}
else{ num_coordinates_checked = num_coordinates_checked;}
 next_lat =
pgm_read_float(&list_of_coordinates[num_coordinates_checked][
0]);
 next_lon =
pgm_read_float(&list_of_coordinates[num_coordinates_checked][
1]);
 if ( !(test) && (abs(flat - next_lat) < 0.0001) && (abs(flon
- next_lon) < 0.0001)) { test = true;}
```

```
if (test ){
      next_heading =
pgm_read_byte(&list_of_headings[num_coordinates_checked
+ 1]);
        next_direction =
convert_heading_to_cardinal_direction(next_heading);
        next_dir = get_num_arrow(next_direction[0],
next_direction[1]);
            if (next_dir + (9-current_dir) > 8){
              arrow = next_dir - (8 -(9-current_dir));}
           else{ arrow = next_dir + (9-current_dir);}
        if((abs(flat - next_lat) > 0 ) && (abs(flon -
next_lon) > 0 )){num_coordinates_checked = index;
          index = index + 1;
          current_dir = next_dir;
          count_coordinate = index;
          test = false;
          }}
 testScreen(arrow);}
```

Appendix - Code

Functions

```
convert_heading_to_cardinal_direction (heading):
   if 0 <= heading < 22.5 or 337.5 <= heading < 360:
       return "N"
   if 22.5 <= heading < 67.5:
       return "NE"
   if 67.5 <= heading < 112.5:
       return "E"
   if 112.5 <= heading < 157.5:
       return "SE"
   if 157.5 <= heading < 202.5:
       return "S"
   if 202.5 <= heading < 247.5:
       return "SW"
   if 247.5 <= heading < 292.5:
       return "W"
   if 292.5 <= heading < 337.5:
       return "NW"
```

```
get_next_arrow(current_direction, next_direction):
   if current_direction == "N":
       return get_num_arrow(next_direction)
   if current_direction == 'SW':
       if get_num_arrow(current_direction) <6:
           return get_num_arrow(next_direction) +3
       else:
           if current_direction =="W":
               return get_num_arrow("NW")
           else:
               return get_num_arrow("E")
   . . .
```

```
get_num_arrow (direction):
   if direction == "N":
       return 1
   if direction == "NE":
       return 2
   if direction == "E":
       return 3

   . . .
```