GNG 2101
Introduction to Product Development and Management for Engineers and Computer Scientists

Accessible Directions
Group: A5

Faculty of Engineering
2020

Project Deliverable J: User Manual

Professor: Dr.Hanan Anis

Authors : Ngendahayo Dan Stéphane
Aya Sassi
Yasin Topcuoglu
Medhanit Mamo
Zak Provenzano

Abstract

Visual impairment affects millions of people worldwide. Despite the large number of people affected by this condition, there are few navigation systems on the market targeting this specific group of people. This report summarizes the work our team did in collaboration with a visually impaired client to explore the problems with the navigation solutions available currently. Additionally it explores the steps and processes we used to design a whole new navigation system that addresses these problems. The results and the final design is discussed in relation to the initial problems presented by the client.

Table of contents

List of figures

List of tables

1. Introduction

According to the World health organization visual impairment affects an estimated 285 million people globally. For the visually impaired retaining independence in day to day life is important. However, this is made difficult by the lack of reliable navigation available specifically to visually impaired individuals. Simple tasks like going to work or grocery shopping can be tedious and potentially dangerous. The navigational solutions available on the market currently do not meet the needs of individuals with this condition for few important reasons. These reasons are that many depend on sound as the main source of feedback to the user. They are often not hands free, they have no indoor functionality and the ones that meet all these requirements are unaffordable.

Many of these problems are as a result of failure in the design process. When designing a product for users, while observing and reading about the experience of individuals can help in creating a working system, talking to people that have firsthand experience is important to address their needs fully. Keeping this in mind, our group worked with Kim who is totally blind and coordinates a program called Get Together with Technology. In partnership with her our team was able to design a product that addresses many of the problems mentioned above from indoor functionality to affordability.

2. User manual

2.1 Capabilities and important features

The main objective of the final product was to provide a navigation system designed to facilitate indoor and outdoor navigation for the visually impaired. As such, the prototype has two main components: an obstacle detection system for indoor navigation and the GPS audio/hands free navigation system for outdoor navigation. Those two functions are not mutually exclusive and are intended to work simultaneously together. Those key functions are the foundation of the remaining features of the product.

Since the navigation system required Internet connectivity, implementing a convenient way to connect the device to a wifi network was imperative. There are two ways for this requirement to be met; the first approach is by bluetooth connectivity from the user's phone to device; and the second approach is to have the device automatically connect to the wifi on its own. The later option was chosen as a method to connect the device to a wifi network. This feature allows the device to automatically scan for open wifi networks, sort them based on their signal strength and connect to the strongest wifi signal detected. Although this method is more or less convenient than the bluetooth connectivity route, there is still a risk of not detecting any open wifi network nearby. As such, it is recommended for the user to activate a mobile hotspot to enable the device to always have an internet connection. More detailed information will be provided in the operational instructions and guidelines section of this document.

The core functionality of the navigation system relies on Google Maps directions API, this application programming interface allows the device to retrieve the best route from a predefined origin point. Although the Arduino library used to interact with the API is not very developer friendly, it had enough flexibility to be able to provide some form of customization to the software. The software is capable of specifying the mode of transportation, the starting and arrival time for the journey.

Finally, the system would not be complete without some form of GPS tracker to always know where the user is located. This is the reason why the prototype features geolocation and this technology will be explained in more details in the following section. In summary, geolocation is capable of collecting the user's GPS data based on their IP address. This allows the navigation system to always update the route depending on the user's location and adjust itself accordingly.

2.2 Prototype assembling

Following the iterative engineering design process, two focused prototypes were made in order to test and experiment with different components to be included in the final product. The following section discusses the results found in the first two prototypes and shows how that result was instrumental in assembling the final prototype.

Prototype 1:

The goal of this prototype was to test out specific design features such as the weight, aesthetics, battery life and the obstacle detection. These features were defined based on the client's needs and were used to compare the previously generated concepts in a decision matrix. The chosen concept from the decision matrix was a belt design with 4 vibration motors and an ultrasonic sensor under the buckle but after consulting with the client, two major concerns were raised. The first concern raised by the client was the placement of the vibration motors being on the hip, the client stated the vibrations are not felt at the hip over clothing and this was evident by the complaint of a similar design received in the past. In order to address this issue the design for the first prototype featured straps over the shoulders where the vibration motors would be placed and a bag containing all of the electronics that sits over the chest area (as seen in *figure 1*).

Figure 1: Placement of the vibration motors

The second concern raised was the issue of the ultrasonic sensors being blocked either by clothing or by the position of the body in the case of blind people in wheelchairs. The group came up with the idea of encasing the ultrasonic sensor in a weatherproof case that can be clipped on any clothing. For the prototype seen in *figure 2,* no budget was given and components provided in lab package as well as items found in the house were used to build the prototype

Figure 2: First prototype built using household items

The picture in *figure 3* shows the circuit we built for the ultrasonic sensors. This part of the design was hidden inside the pouch to protect it from extreme conditions. The following picture, f*igure 4*, is the schematic showing how the sensor connects to the Arduino UNO microcontroller.

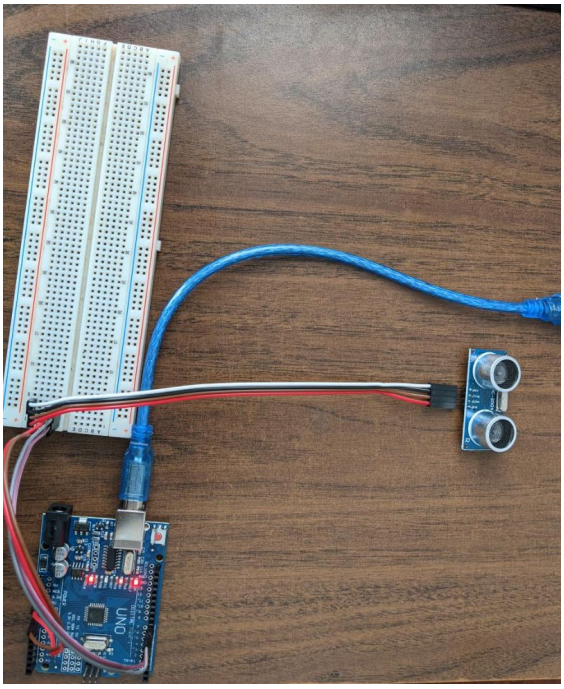

Figure 3: Circuit testing ultrasonic sensors built using real components.

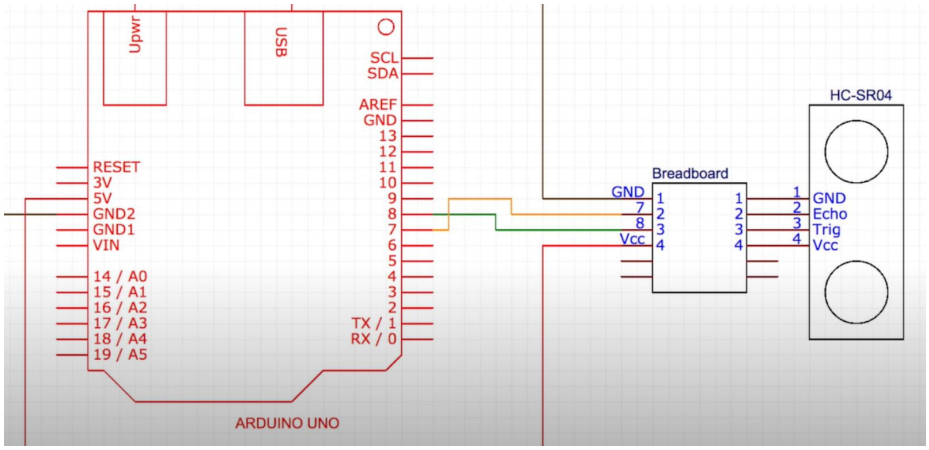

Figure 4: Schematic for prototype 1 testing ultrasonic sensors

In the end, the prototype performed as expected for most of the tests and further explanation of each result is shown in the table below.

Table 1: Prototype 1 performance testing

Specification	Expected Results	Actual Results	Specifications
Usable indoors	It was expected that	The device is usable	The device can detect

	the device is usable for indoor navigation through the use of the ultrasonic sensors	indoors. This was tested by building the circuit for the ultrasonic sensors and arduino and the results were as expected. The sensors can detect incoming obstacles and deliver information	obstacles ahead of the user using the sensor and warn them with the vibrational motors
Aesthetics	It was expected that the device is inconspicuous and a neutral colour so it blends with the outfit.	The strap is not the most distinct but is a neutral colour (black).	The strap may not be the most stylish however this design allows for maximum functionality. It provides durability, hands free use, indoor navigation as well as outdoor
Hands free	It was expected to have the device completely hands free	The device is completely hands free since all the components are in the pouch	The strap that goes around the chest allows for complete hands free use and does not affect any movement
Weight	300 g	700 g. This is above the expected value but falls within a reasonable weight the client agreed to in the first client meeting.	The Strap is about 200 g. Each AA battery is about 23g and 18 of them are needed for a total of about 414 grams. The breadboard weighs about 30g and the arduino is about 25g. The wires and battery holders are very light so the total weight of the entire package comes to 700g
Cost	$100	$97.81	The total cost of all the components came up to $97.81, which

			is right under budget
Battery life	5 hours	~ 30 hours for the arduino and ~2 hours of continuous vibration for the vibrators	The 9V battery on the arduino can provide about 30 hours of continuous use and the 4.5 V batteries attached to each vibrating motor can provide about 2 hours of continuous vibration
Length	50 - 100 cm	~ 30 cm	The straps are pretty seamless since they basically stick to the body of the user. The only added bulk is the length of the pouch which is about 30 cm
Width	5 cm	~ 5 cm	Again, the straps don't add noticeable width, the only added width is the pouch which is about 5cm to the front of the chest.
Durability	Drop proof from about <2 metres	The device is drop proof from around 2 metres	Since the electronics are covered in the pouch, a drop isn't as impactful and the pouch provides some protection. The pouch is also waterproof. This makes the device extra durable

Prototype 2:

The purpose of this specific prototype was to incorporate the vibration motors with the ultrasonic sensor capabilities of the system we were designing. It used most of the components we used in the final prototype. This included the ultrasonic sensor, the vibration motors and the arduino. It works by receiving data from the sensors, then this data is passed onto the arduino board which acts as the control center. This board instructs the vibration motor to notify the user. There were few changes made in the circuit of the final prototype compared to this one. We learned we

cannot connect the vibration motor directly to the arduino so a motor shield driver was added to the circuit to accommodate the 4 vibrators required. This meant connecting this arduino board to wifi was not going to be possible using a wifi module so we exchanged the arduino uno with an arduino uno wifi rev2 board. This allowed us to connect to wifi directly without needing the module. Overall this prototype was important, as it showed us the limitations of our initial circuit design.

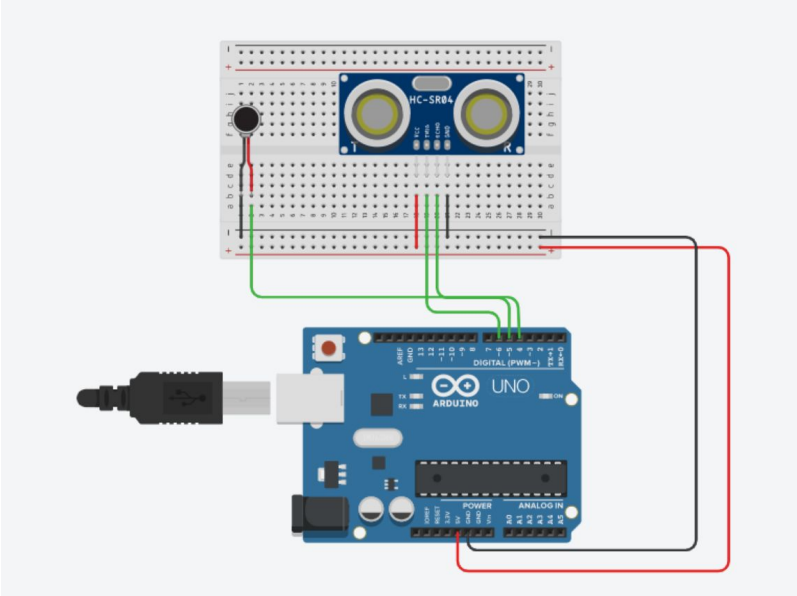

Figure 5: Schematics of first prototype testing the vibration motors

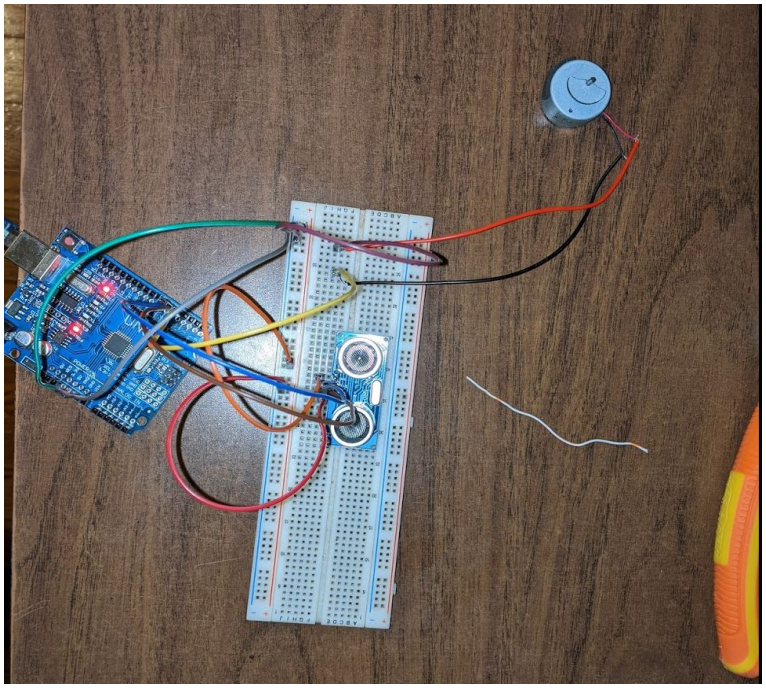

Figure 6: Prototype 1 built with real life components

Final product:

The final iteration of our product combined the circuit from prototype 2 with the design from prototype 1 with a few tweaks considering some unfortunate circumstances. The major incremental improvement came in the Google Maps navigation API integration as well as the input system.

Prior to implementing the navigation and input system, we had to have the device automatically connect to an open wifi network on its own. Subsequently, this feature was integrated using the Wifinina library for Arduino UNO Wifi boards. This library allows the device to scan for nearby wifi networks, determine if they're open or secured and also calculate their signal strength. Using those pieces of information, we were able to apply a bubble sorting algorithm to sort the scanned wifi network by their signal strength. The following code then code then attempts to connect each open wifi network starting by the strongest signal detected.

```
103
104 void connectToStrongestWiFi(){
105
106     if(WiFi.status()==WL_NO_MODULE){
107       Serial.println("Communication with Wifi module failed");
108
109     }else{
110        if(WiFi.firmwareVersion()<WIFI_FIRMWARE_LATEST_VERSION){
111       Serial.println("Please upgrade the firmware");
112     }
113     int numSsid =WiFi.scanNetworks();
114     if(numSsid==-1){
115       Serial.println("No available networks detected");
116     }else{
117     Serial.println("Number of available networks: "+String(numSsid));
118
119     int indices[numSsid];
120     for (int i = 0; i < numSsid; i++) {
121       indices[i] = i;
122     }
123     //sorting in decreasing signal strength
124     for (int i = 0; i < numSsid; i++) {
125       for (int j = i + 1; j < numSsid; j++) {
126         if (WiFi.RSSI(indices[j]) > WiFi.RSSI(indices[i])) {
127            int temp=indices[j];
128            indices[j]=indices[i];
129            indices[i]=temp;
130         }
131       }
132
133     }
134     int openNet=-1;
135     int k=0;
136     for(int i=0;i<numSsid;i++){
137       if(WiFi.encryptionType(indices[i])==ENC_TYPE_NONE){
138         openNet=indices[i];
139         break;
140       }
141     }
142     int status= WL_IDLE_STATUS;
143     if(openNet==-1){
144        Serial.println("No open networks available");
145     }else{
146       String str;
147          while(status!=WL_CONNECTED){
148       str="Attempting to connect to: ";
149           str.concat(WiFi.SSID(openNet));
150            Serial.println(str);
151            Serial.print(".");
152            status=WiFi.begin(WiFi.SSID(openNet));
153            delay(10000);
154          }
155          str="SuccessFully connected to: ";
156          str.concat(WiFi.SSID());
157          Serial.println(str);
158          str="Wifi local IP: ";
159          str.concat(WiFi.localIP());
160          Serial.println(str);
161
162
163     }
```

Figure 7: Arduino function for connecting to the strongest open Wifi detected

The incorporation of Google Maps navigation API with Arduino utilized the Google Maps direction API library. This library enables HTTP requests to Google Maps and returns a response object containing information about a given route based on an origin and destination

point. The specific monumental field embedded in the response object was the maneuvers. Maneuvers are turn signals ranging from: turn-slight left to turn-sharp left, turn-slight right to turn-sharp right etc. These signals are then processed by the code seen below and change the current direction variable to the corresponding signal which then activates the vibration motors after the following function is called.

```
374
375 void checkGoogleMaps() {
376
377
378   DirectionsResponse response = api.directionsApi(origin, destination, inputOptions);
379
380
381
382   Serial.println("Checking route from: "+response.start_address + " to "+response.end_address);
383   Serial.println("Distance in Kilometers: "+response.distance_text);
384
385
386  String instruction = response.maneuver;
387
388  Serial.println("turn signal: "+instruction);
389
390  if(instruction!="\0"){
391    if(instruction=="left-turn"||instruction=="turn-slight-left"||instruction=="turn-sharp-left"){
392      direction='l';
393    }else if(instruction=="right-turn"||instruction=="turn-slight-right"||instruction=="turn-sharp-right"){
394      direction='r';
395    }else if(instruction=="uturn-left"||instruction=="uturn-right"){
396      direction='b';
397
398    }else{
399      direction='f';
400    }
401  }
402
403
404 }
```

Figure 8: Arduino code requesting the best route and parsing the response

As the Arduino Google Maps API library lacked the specific maneuver field needed for the above function, some changes needed to be made in the library source code to include the maneuver field. The following figures show the added code in the GoogleMapsDirections.cpp and GoogleMapsDirections.h files in order to retrieve the specific field needed from the JSON response.

```
void DirectionsListener::value(String value) {

//ADDED CODE
  if(currentKey=="maneuver"){

          responseObject.maneuver = value;

}

/****/
  if (depthFromLegsKey == 1) {
    if (currentKey == "start_address") {
      responseObject.start_address = value;
    } else if (currentKey == "end_address") {
      responseObject.end_address = value;
    }
  } else if (depthFromLegsKey == 2){
    if (currentParent == "distance") {
      if (currentKey == "text") {
          responseObject.distance_text = value;
      } else if (currentKey == "value") {
        responseObject.distance_value = value.toInt();
      }
    } else if (currentParent == "duration") {
      if (currentKey == "text") {
          responseObject.duration_text = value;
      } else if (currentKey == "value") {
        responseObject.duration_value = value.toInt();
      }
    } else if (currentParent == "duration_in_traffic") {
      if (currentKey == "text") {
          responseObject.durationTraffic_text = value;
      } else if (currentKey == "value") {
        responseObject.durationTraffic_value = value.toInt();
      }
    }
  }
}
```

Figure 9: Modified Google Maps directions API source code from GoogleMapsDirections.cpp file

```
struct DirectionsResponse{
  int distance_value;
  String distance_text;
  int duration_value;
  String duration_text;
  int durationTraffic_value;
  String durationTraffic_text;
  String start_address;
  String end_address;
  String maneuver;//ADDED RESPONSE FIELD
};
```

Figure 10: Modified response object from GoogleMapsDirections.h file

Since the user's location is needed for each Google Maps directions API request, some form of GPS tracking had to be implemented. The method chosen for this particular task was geolocation. Geolocation technology utilizes IP address tracking of any devices connected to the internet. From this concept, we used a location API from Unwired labs which geolocate registered devices using complex algorithms. The figure below shows geolocation code from Unwired labs.

```
}

void getUserLocation(){
  //see : https://www.instructables.com/Location-Tracker-With-NodeMCU-ESP8266/ as a starting point

   /*TO-DO
    * Update the orgin variable for the journey
    *
    * */
    DynamicJsonDocument doc(1024);

    // WiFi.scanNetworks will return the number of networks found
    int n = WiFi.scanNetworks();


    // now build the jsonString...
      String jsonString = "{\n";
      jsonString += "\"token\" : \"";
      jsonString += token;
      jsonString += "\",\n";
      jsonString += "\"id\" : \"saikirandevice01\",\n";
      jsonString += "\"wifi\": [\n";
      for (int j = 0; j < n; ++j) {
          jsonString += "{\n";
          jsonString += "\"bssid\" : \"";
          //jsonString += WiFi.BSSID(j);
          jsonString += "\",\n";
          jsonString += "\"signal\": ";
          jsonString += WiFi.RSSI(j);
          jsonString += "\n";
          if (j < n - 1) {
              jsonString += "},\n";
          } else {
              jsonString += "}\n";
          }
      }
      jsonString += ("]\n");
      jsonString += ("}\n");
      //Serial.println(jsonString);

      WiFiClient client;
      //[ client.setInsecure(); ]

      //Connect to the client and make the api call
      Serial.println("Requesting URL: https://" + (String)Host + endpoint);
      if (client.connect(Host, 443)) {
          Serial.println("Connected");
          client.println("POST " + endpoint + " HTTP/1.1");
          client.println("Host: " + (String)Host);
          client.println("Connection: close");
          client.println("Content-Type: application/json");
          client.println("User-Agent: Arduino/1.0");
          client.print("Content-Length: ");
          client.println(jsonString.length());
          client.println();
          client.print(jsonString);
          delay(500);
      }

      //Read and parse all the lines of the reply from server
      if (client.available()) {
        Serial.println("Client available");
          String line = client.readStringUntil('\r');
          auto error = deserializeJson(doc,line);
          if (!error) {
              latitude    = doc["lat"];
              longitude   = doc["lon"];
              accuracy    = doc["accuracy"];

              Serial.println();
              Serial.print("Latitude = ");
              Serial.println(latitude, 6);
              Serial.print("Longitude = ");
              Serial.println(longitude, 6);
              Serial.print("Accuracy = ");
              Serial.println(accuracy);
              origin.concat(latitude);
              origin.concat(",");
              origin.concat(longitude);
          }else{
            Serial.println("Deserialization error");
          }
      }


```

Figure 11: Unwired labs geolocation function for Arduino

Finally, the last addition to the software was the input system. The intended input system uses a messaging app called Telegram. Telegram is a free messaging app similar to WhatsApp and available from Apple's App Store and Android's Play Store. This messaging app allows users to create a computer program called a chat bot, designed to simulate conversations with human users. Chat bots are powered by rules and sometimes artificial intelligence to interact via a chat

interface. Thus, using a rule or command based public chat bot with Telegram where the user can send inputs such as their destination and origin grants the capability to add rules and commands for future software updates.

The Arduino code then retrieves those messages using the Telegram BOT API library from Arduino libraries, processes the inputs and calculates the best route while updating the user's location. The figures below show the message handling and the main loop function that defines the structural logic of the final product.

```
211 void handleNewMessages(int message) {
212
213
214     String chat_id = String(bot.messages[message].chat_id);
215
216     String text = bot.messages[message].text;
217
218       if (text.startsWith("/destination")) {
219         Serial.println(text);
220         text.remove(0, 13);
221         destination = text;
222         bot.sendMessage(chat_id, "Set Destination: " + String(text), "");
223       }
224       if (text.startsWith("/origin")) {
225         Serial.println(text);
226         text.remove(0, 8);
227         origin = text;
228         bot.sendMessage(chat_id, "Set Origin: " + String(text), "");
229       }
230
231
232     if (text.startsWith("/indoorMode")) {
233         Serial.println(text);
234         mode = 'i';
235         bot.sendMessage(chat_id, "Set mode: indoor", "");
236       }
237      if (text.startsWith("/OutdoorMode")) {
238         Serial.println(text);
239         mode = 'o';
240         bot.sendMessage(chat_id, "Set mode: outdoor", "");
241       }
242
243       if (text == "/start") {
244         String welcome = "Welcome from Google Maps Bot\n";
245         welcome = welcome + "/destination : set destination\n";
246         welcome = welcome + "/origin : set origin\n";
247
248         bot.sendMessage(chat_id, welcome, "Markdown");
249       }
250
251
252 }
```

Figure 12: Telegram Chat bot message handling function

```
426
427
428 //main function
429 void loop ()  {
430
431   //works
432 //bot.sendMessage("1417975382","test Message");
433   //getUserLocation();
434   checkGoogleMaps();
435
436
437   int numNewMessages = bot.getUpdates(bot.last_message_received + 1); // fetch the number of new messages received
438   if(numNewMessages)
439        Serial.println("New messages: "+String(numNewMessages));
440  else
441      Serial.println("No messages");
442
443   if(checkObstacleDistance()<=MIN_DISTANCE){
444      Serial.println("Obstacle!!!");
445      backwardVibration();
446     }
447   backwardVibration();
448     if(millis() - bot_lasttime > BOT_MTBS){
449
450         int numNewMessages = bot.getUpdates(bot.last_message_received + 1); // fetch the number of new messages received
451    while(numNewMessages) {
452
453       Serial.println("Handling bot messages");
454       handleNewMessages(numNewMessages);
455       getUserLocation();
456
457
458       if(origin!="" && destination!=""){
459
460         checkGoogleMaps();
461
462         if(direction!='\0'){
463
464           switch(direction){
465
466             case 'l':
467                leftVibration();
468             break;
469
470             case 'r':
471                rightVibration();
472              break;
473
474             case 'f':
475                forwardVibration();
476              break;
477
478             case 'b':
479               backwardVibration();
480              break;
481           }
482
483         }else{
484           Serial.println("No directions found");
485           break;
486         }
```

Figure 13: Main loop function for the final product

2.3 Operational instructions and guidelines

1. Loosen chest pack straps
2. Place chest pack over shoulders
3. Tighten strap to fit
4. Unzip pouch and plug in battery

Caution: If board is very hot to touch immediately unplug battery and let the device cool down

2.4 Troubleshooting

- Device not powering on
    - Check connection to battery or take out and re-insert battery. If device is still not powering on replace battery
- Vibration not working
    - Click the reset button on the Arduino board. If vibration is still not working check the connection of the vibration motor wires with the board and the vibration motor itself
- Ultrasonic sensor not picking up obstacles
    - Check that the wires are inserted into the ultrasonic sensor firmly and check that the connection to the board is firm as well
- Device is overheating
    - Check connections for short circuits and make sure exposed wires are not touching each other.

3. Project files

- https://makerepo.com/rails/active_storage/blobs/eyJfcmFpbHMiOnsibWVzc2FnZSI6IkJBaHBBc2dvIiwiZXhwIjpudWxsLCJwdXIiOiJibG9iX2lkIn19--1367080baa25b8c21eec4208c4a7a47534c44a4b/ArduiNav.ino
    - This link is the full arduino code that needs to be uploaded to the arduino board in order for the device to work. This file can be downloaded and run on Arduino IDE. Once compiled, it can be uploaded onto the Arduino board by USB connection. Once the uploading is complete the device will start to send navigation feedback and the ultrasonic sensor will constantly search for obstacles.

4. Conclusion and recommendations for future work

Overall, this project has taught us many valuable lessons. However, the one lesson to really take away is to always have the necessary hardware to test the software. Software often has many bugs and can only be fixed through testing. If the hardware that is required for testing is not available, then testing code and fixing it is very hard. Therefore, before it is too late it is necessary to have working hardware at hand. In our case, the hardware was received late due to shipment delays. Therefore, another lesson to be learned is to always order as soon as possible and have a plan B since shippers can not be relied upon.

5. Bibliography

Matt Schlicht, The Complete Beginner's guide to Chatbots, Chatbotsmagazine, (2016, April) https://chatbotsmagazine.com/the-complete-beginner-s-guide-to-chatbots-8280b7b906ca

Daniel Lonescu, Geolocation101, How it works, the apps and your privacy, PCWorld, (2010, March 29th), https://www.pcworld.com/article/192803/geolo.html

Location Tracker with Node MCU ESP 8266, Instructable, https://www.instructables.com/Location-Tracker-With-NodeMCU-ESP8266/

Directions API, Google Maps Platform, https://developers.google.com/maps/documentation/directions/overview

Telegram Bot API, Telegram, (2020, November 4) https://core.telegram.org/bots/api#making-requests

6. Appendix

Bill of Materials

- Vibrating Mini Motor disc
    - Link: https://x2robotics.ca/robot-kits/motors-and-servos/dc-motors/vibrating-mini-motor-disc?sort=pd.name&order=ASC

    - The vibrating mini motor disc is a buzzing motor that is ideal for haptic feedback. It is completely sealed and tiny so it can be embedded or placed easily on a project. The disc has two wires to control and power it. It can be powered directly from a battery or a microcontroller. It works using a minimum of 2V up to 5V. The higher the voltage, the stronger the vibration.
    - Quantity: 4 of the motor disc is ideal for placement in the front, back, right and left side of the user
    - Price per unit: 2.99$
    - Justification: This product is perfect for the design because it doesn't have any physical moving parts making it safe and also easier to hide with straps. It is also pretty small in size which will help to minimize bulk on the product

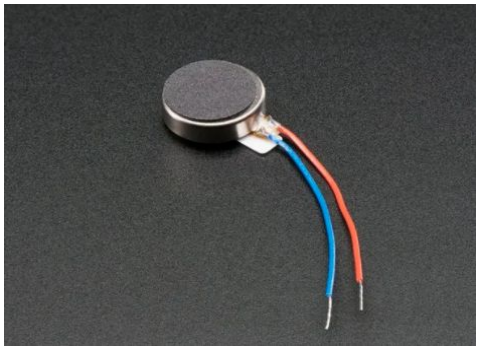

Figure 14: Picture of Vibrating Mini Motor Disc

- HC-SR04 Ultrasonic sensor
    - Link: https://www.robotshop.com/ca/en/hc-sr04-ultrasonic-range-finder-tys.html
    - The ultrasonic sensor is a range or distance detector that can be used to translate sound wave travel time to a distance measurement.
    - This sensor offers a detecting range of 3cm to 4m.
    - Quantity: 1
    - Price per unit: 2$ CAD
    - Justification: This Ultrasonic sensor is ideal since it is very cheap and will detect obstacles. It is also pretty small making easy to place on the product or around the body.

Figure 15: Picture of the HC-SR04 Ultrasonic sensor

- NOENNULL Chest Pack
    - Link: https://www.amazon.ca/NOENNULL-Outdoor-Streetwear-WomenSport-Pocket/dp/B086GCCHDK/ref=sr_1_32?dchild=1&keywords=chest+strap&qid=1602105957&sr=8-32
    - This chest pack is waterproof, dirty proof and stylish. It has a front bag with the capacity to hold many items. The bag has clips to hold on to the straps so that it is secured in place. The straps are adjustable to meet everyone's needs.
    - Quantity: 1
    - Price per unit: $35.59CAD
    - Justification: This chest strap provides the best quality for its price. It is also fully adjustable which is ideal for comfort and allows for compatibility with all body types. It also has a pouch at the front which is useful for hiding cables and electronics. The pouch will also provide extra protection since it is dirty proof ans waterproof

Figure 16: Picture of Noennull chest pack

- Jumper Cables
    - Link: https://www.amazon.ca/RGBZONE-120pcs-Multicolored-Dupont-Breadboard/dp/B01M1IEUAF/ref=sr_1_4?dchild=1&keywords=male+to+male+wires&qid=1607635535&s=electronics&sr=1-4
    - The jumper cables can be used to create extensions. It contains 40 of male-male, female-female, and male-femal
    - Quantity: 1
    - Price per unit: $10.99

Figure 17: Picture of jumper cables

- Arduino UNO Wifi Rev2
    - Link: https://www.amazon.ca/Arduino-Uno-WiFi-Microcontroller-rev2/dp/B07MK598QV

- This board is where the whole system will run from and where the phone will connect to.
- Quantity: 1
- Price per unit: For our project it was free since we had it at hand, however the cost if you were to purchase it is $69.99
- Justification: This board is where all the hardware will connect to.

Figure 18: Arduino UNO Rev2

- Arduino Motor Shield
  - Link: https://www.amazon.ca/Lysignal-Expansion-Arduinos-Diecimila-Duemilanove/dp/B073GRYT5R/ref=pd_lpo_147_img_1/135-4628925-9958513?_encoding=UTF8&pd_rd_i=B073GRYT5R&pd_rd_r=56a49624-5228-45eb-8dfc-5a7a80fdd537&pd_rd_w=vBPqK&pd_rd_wg=NpwzS&pf_rd_p=256a14b6-93bc-4bcd-9f68-aea60d2878b9&pf_rd_r=ZXW65J9F2HKYG6Z5KYJV&psc=1&refRID=ZXW65J9F2HKYG6Z5KYJV
  - This motor shield allows the power supply to be connected with the power supply and the motors
  - Quantity: 1
  - Price per unit: We had it at hand therefore it was free however if you were to purchase it, it costs $10.99

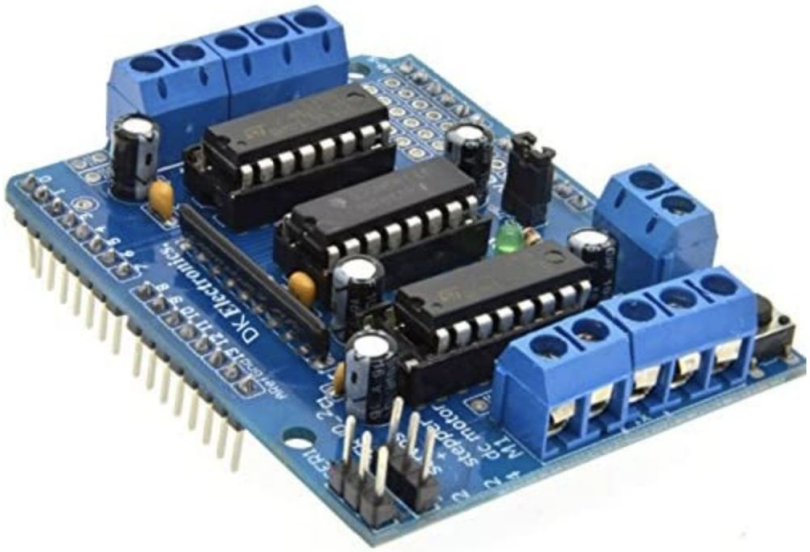

Figure 19: Arduino Motor Shield

- 9V batteries
  - Link: https://www.amazon.ca/AmazonBasics-Everyday-Alkaline-Batteries-8-Pack/dp/B00MH4QM1S/ref=sr_1_1_sspa?crid=KM2DGPG4VFS5&dchild=1&keywords=9v+battery&qid=1607635771&s=electronics&sprefix=9v+b%2Celectronics%2C158&sr=1-1-spons&psc=1&spLa=ZW5jcnlwdGVkUXVhbGlmaWVyPUExVENHSUxQSTFSWUFSJmVuY3J5cHRlZElkPUEwNTM3MzQ0M0YwSEE2WEVTQkxHNCZlbmNyeXB0ZWRBZElkPUEwMDMzODEzM0NJVjcyNkdFVkU1JndpZGdldE5hbWU9c3BfYXRmJmFjdGlvbj1jbGlja1JlZGlyZWN0JmRvTm90TG9nQ2xpY2s9dHJ1ZQ==
  - One 9V battery is enough to power the vibration motors and board
  - Quantity: One pack
  - Price per unit: We had them at hand therefore they were free however if you do not it is $10.63

Figure 20: 9V Batteries

Table 2: Price breakdown for all required parts

Part Name	Quantity	Unit Cost ($CAD)	Cost ($CAD)
Vibrating Mini Motor Disc	4	2.99	11.96
HC-SR04 Ultrasonic sensor	1	2	2
NOENNULL Chest Pack	1	35.59	35.59
Jumper Cables	1	10.99	10.99
Arduino UNO Wifi Rev2	1	Free	Free
Arduino Motor Shield	1	Free	Free
9V batteries	1	Free	Free
Total Price			$60.54