

GNG 1103

Design Project User and Product Manual

The Emergency Beacon

Submitted by:

The 9ers, group D9

Elsa Lange, 300168393

Karen Hakko, 300196284

Jacob Troop, 300186278

Sandeep Sinha, 300166121

Tri Thai, 300201869

April 11, 2021

University of Ottawa

Table of Contents

List of Acronyms and Glossary	5
Introduction	7
Overview	8
Conventions	10
Cautions & Warnings	10
Getting started	10
Set-up Considerations	11
User Access Considerations	12
Accessing the System	12
System Organization & Navigation	12
Altitude Subsystem	12
Location Subsystem	12
Voice Subsystem	15
Light Subsystem	16
Exiting the System	18
Using the System	18
Altitude Subsystem	18
Location Subsystem.....	19
Voice Subsystem.....	22
Light Subsystem.....	23
Interconnections/ Data Transmission	24
Troubleshooting & Support	29
Error Messages or Behaviors	29
Special Considerations	29

Maintenance	30
Maintenance of the Case	30
Maintenance of the Components	30
Support	31
Product Documentation	31
Subsystems of the prototype	31
BOM (Bill of Materials)	31
Altitude Subsystem	31
Location Subsystem	32
Voice Subsystem	33
Light Subsystem	33
Equipment List	33
Altitude Subsystem	33
Location Subsystem	33
Voice Subsystem	34
Light Subsystem	34
Acrylic Case	34
Instructions	34
Altitude Subsystem	34
Location Subsystem	35
Voice Subsystem	36
Light Subsystem	37
Testing and Validations	38
Altitude Subsystem	38
Location Subsystem	39

Voice Subsystem	39
Light Subsystem	39
Conclusions and Recommendations for Future Work	40
Bibliography	40
APPENDICES	43
APPENDIX I : Design Files	43
Other Appendices	46
Appendix II	46
Appendix III	49
Appendix IV	52
Appendix V	52
Appendix VI	54

List of Figures

Figure 1 - Top View of the Emergency Beacon Without the Filter	8
Figure 2 - Front View of the Emergency Beacon	9
Figure 3 - Input and Output Systems of the Emergency Beacon	9
Figure 4 - Inkscape Drawing	11
Figure 5 - Altitude Subsystem	13
Figure 6 - Location Subsystem	14
Figure 7 - Voice Subsystem Connections Using Tinkercad	15
Figure 8 - Voice Subsystem	16
Figure 9 - Light Subsystem Connections Using Tinkercad	17
Figure 10 - Light Subsystem	17
Figure 11 - Altitude Subsystem Physical Prototype	18

Figure 12 - Altitude Subsystem Output Data	19
Figure 13 - Location Subsystem Physical Prototype	20
Figure 14 - Location Subsystem Output Data	21
Figure 15 - Light Subsystem Physical Prototype	22
Figure 16 - Light Subsystem Output Data	23
Figure 17 - Voice Subsystem Physical Prototype	24
Figure 18 - Emergency Beacon Physical Prototype	25
Figure 19 - Emergency Beacon Output Data	28
Figure 20 - The Wired Altitude Subsystem	34
Figure 21 - The Wired Location Subsystem	35
Figure 22 - The Wired Voice Subsystem	37
Figure 23 - The Wired Light Subsystem	38

List of Tables

Table 1 - List of Acronyms	5
Table 2 - Glossary	6
Table 3 - Bill of Materials of the Altitude Subsystem	30
Table 4 - Bill of Materials for Location Subsystem	31
Table 5 - Bill of Materials for the Voice Subsystem	32
Table 6 - Bill of Materials for the Light Subsystem	33
Table 7 - BMP180 and Arduino Uno Pin Set Up	35
Table 8 - Beitian BN-880 and Arduino Uno Pin Set Up	36
Table 9 - List of Deliverables	42

List of Acronyms and Glossary

Table 1- List of Acronyms

Acronym	Expansion
BOM	Bill of Materials
CAD	Canadian Dollar
FSR	Function Statistics Record
GND	Ground
GPS	Global Positioning System
ID	Identification
LCD	Liquid Crystal Display
LED	Light Emitting Diode
SCI	Serial Clock
SDA	Serial Data
UTC	Coordinated Universal Time
VIN	Voltage Input
2D	Two-Dimensional
3D	Three Dimensional

Table 2 - Glossary

Term	Definition
Acrylic	A plastic made from acrylic acid
Arduino IDE	An integrated development environment to create software for the Arduino microcontroller
Arduino Uno	The Arduino Uno is an open source microcontroller that consists of a programmable circuit board.
Barometric Pressure	The pressure caused by the weight of air, also referred to as atmospheric pressure
Beitian-BN-880	A microchip included in the emergency beacon that includes a GPS sensor and compass
BMP-180	A sensor included in the emergency beacon that is capable of measuring barometric pressure
Breadboard	A construction base for the prototyping of electronics
Fidelity	The degree of exactness with something that is copied or reproduced
Inkscape	Inkscape is a free and open-source vector graphics editor used to create vector images, primarily in Scalable Vector Graphics format
Jumper Wires	Wires that are used to connect the breadboard to the Arduino pins
Power Rail	A column on the breadboard that transmits power when connected to a power supply
Protoboard	A specific type of breadboard that allows the creation of a permanent connection through soldering.
Raspberry Pi	A small and low cost computer that is capable of being programmed to perform functions or interpret data.

Resistor	Limit the amount of current going to certain components in the circuit
Serial Monitor	A separate pop up window within Arduino IDE where data received by the Arduino-Uno will be printed
Solder	A fusible metal used to create a permanent bond between wired connections
Subsystem	A self containing system within a larger system
Talkie Library	A speech library for the Arduino code and microcontroller
Threshold	A magnitude or intensity that must be exceeded for a certain reaction, phenomenon or result to occur.
Tinkercad	An online 3D modelling program
Tiny GPS++	A library with code related to the Arduino microcontroller and GPS modules

1 Introduction

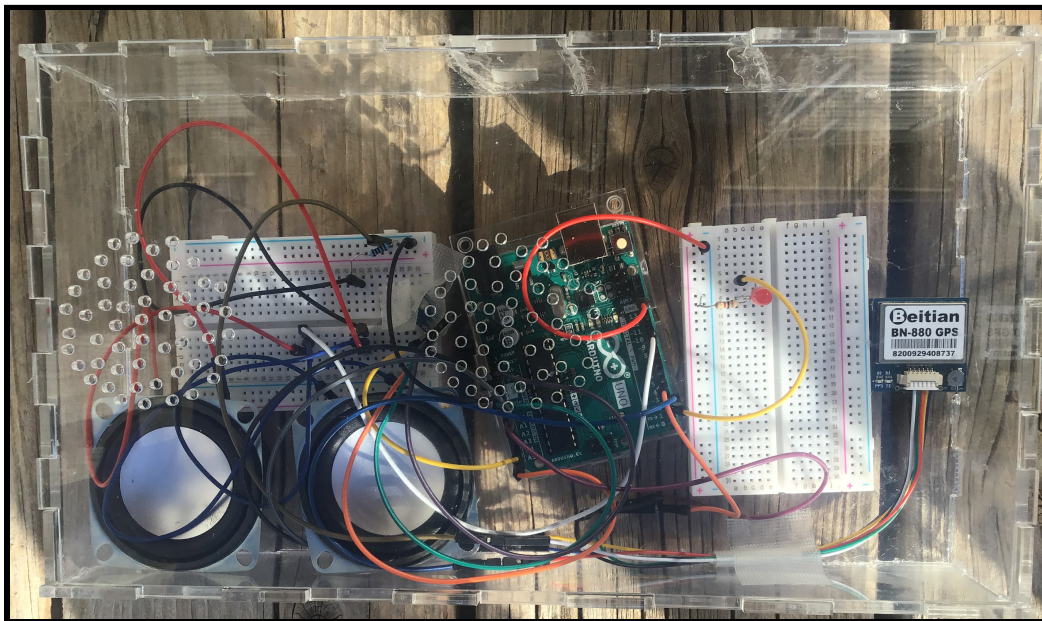
This user manual has been created with the purpose of guiding users and clients through the installation and proper usage techniques of the emergency beacon. With the rise of delivery services, the emergency beacon is a necessary add-on system created for drones which alerts the operator of a downed drone and alerts nearby individuals to not touch the device (a technician is on the way).

It is assumed that the dimensions and weight requirements, one kilogram (including the case and this is an estimate and not done with an actual scale) and 54.372 mm (height), 271.920 mm (width), 152.585 mm (length) will not hinder the drone in operation, in terms of its aerodynamic feasibility. In addition, there is a clear path from the drone's power supply to the appropriate location on the drone. Lastly, it is assumed that serial communication between two arduino microcontrollers accurately reflects the process with a RPi.

2 Overview

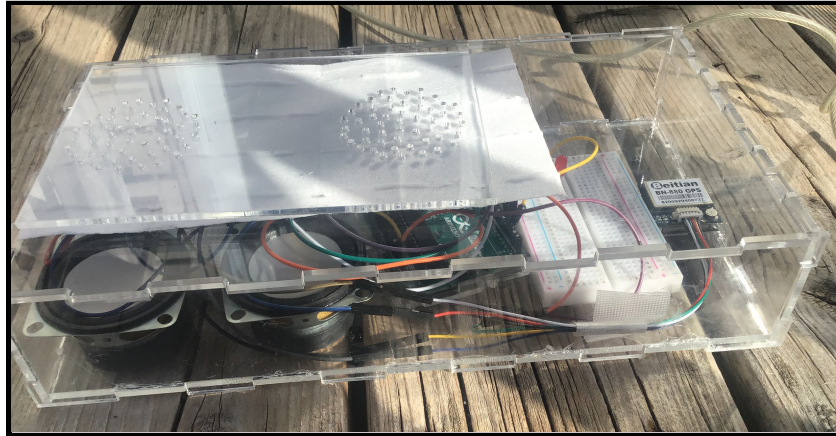
The emergency beacon was created to ensure the safety of pedestrians and the drone while adhering to Transport Canada altitude regulations. As requested by our clients, the emergency beacon needed to transmit accurate and quick location information about the drone to the remote operator in live time by interpreting the data received from the sensors. The product is a necessary addition to any autonomous delivery service as it ensures safety with its voice and light subsystems and reliability of data with its location and altitude subsystems. This emergency beacon includes all the necessary functions in one compact design and comes with its own acrylic case. A picture of the prototype is below:

Figure 1 - Top View of the Emergency Beacon Without the Filter



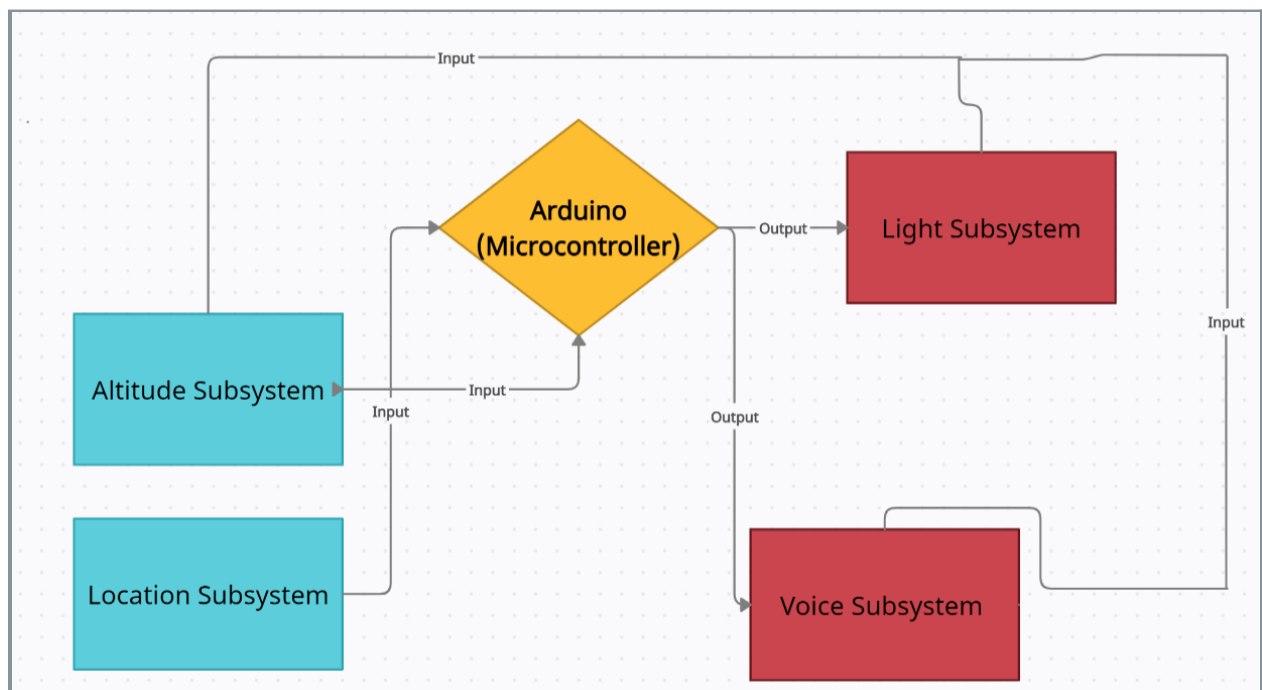
In the picture above, there are two speakers, one LED light, the Beitian BN-880 (the location device), the BMP180 (the barometric pressure sensor), two breadboards and an Arduino uno. The electronics are encased in an acrylic case.

Figure 2 - Front View of the Emergency Beacon



The figure above shows the prototype from the front view. On top of the top of the case is the filter paper and speaker holes.

Figure 3 - Input and Output Systems of the Emergency Beacon



In the figure above, the product is separated by four subsystems: the voice, light, altitude and location subsystems. All of the subsystems are connected to the Arduino (the microcontroller). The two input data is the relative altitude (provided by the altitude subsystem) and the coordinates (provided by the location subsystem) of the emergency beacon. The outputs are flashing lights (provided by the light subsystem) and an automated message (provided by the voice subsystem).

2.1 Conventions

N/A

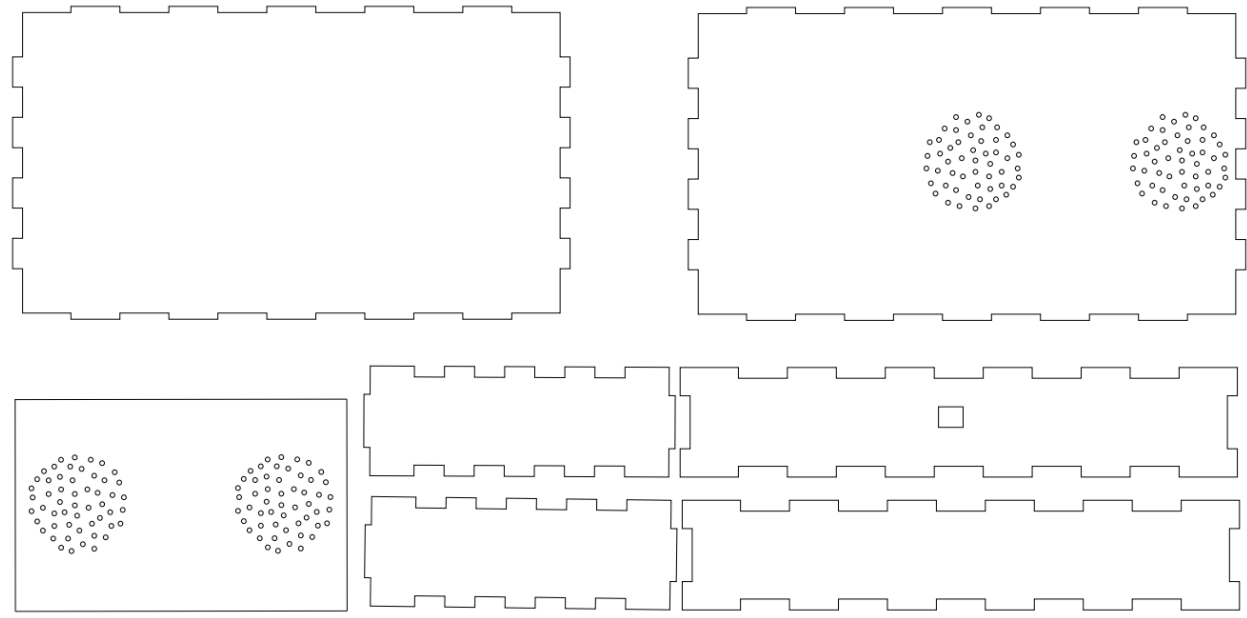
2.2 Cautions & Warnings

Before using the product outdoors, be wary of rain as it may damage the electronics. If using the product with a drone, you must adhere to Transport Canada's drone safety regulations which can be found [here](#).

3 Getting started

1. Purchase all the necessary equipment, as listed in section 3.1, to facilitate the process of building this system, the emergency beacon.
2. Assemble the altitude subsystem as shown in Figure . For this subsystem, a barometric pressure sensor, an Arduino and four subsystems are needed. To verify the set up, run the code provided in [10.1 Appendix II](#) and make adjustments based on the results.
3. Assemble the location subsystem next using Figure shown below. For this subsystem, a GPS module, an Arduino, a breadboard and six wires are required. Run the code provided in [10.2 Appendix III](#) to ensure the functionality of the subsystem.
4. Assemble the voice subsystem using Figure . The voice subsystem requires two speakers, an Arduino, a breadboard, six wires and a 100 Ω resistor. Remember to test the set up using the code provided in [10.3 Appendix IV](#).
5. Assemble the light subsystem with the help of Figure shown below. This subsystem needs eight red LED lights, 8 1 k Ω resistors, an Arduino, a breadboard and 17 wires. Run the code, shown in [10.4 Appendix V](#), to verify that the subsystem is functional.
6. Once all four subsystems are functioning separately, combine them using Figure and test it again using the code provided in [10.5 Appendix VI](#).
7. To get more detailed steps and images, visit the group's Makerepo page using [this link](#).
8. Use the inkscape drawing shown in Figure to laser cut the acrylic piece to make the encasing.
9. Once the box is ready, assemble the four subsystems into the case using Figure for help.

Figure 4 - Inkscape Drawing



The Figure above shows the Inkscape drawing used to laser cut our encasing. A link to this drawing is attached [here](#). Inkscape can be downloaded from [here](#). Once the drawing is opened, it will be ready to laser cut, but if the user wishes to see the outline of the parts, a change in the thickness needs to be made in stroke style.

3.1 Set-up Considerations

Equipment List:

1. Arduino Uno (Connector)
2. Breadboard (Connector)
3. Wires (Connector)
4. Location device (Input)
5. Altitude sensor (Input)
6. Speakers (Output)
7. LED lights (Output)
8. Resistors (Connector)
9. Acrylic $\frac{1}{8}$ inch thick sheet (24 by 12 inches)
10. Laser cutter
11. Soldering equipment
12. E600 industrial strength adhesive (glue)

3.2 User Access Considerations

The user for this product is the JAMZ operator who will be in charge of the whole drone and the emergency beacon as an add-on system. The system is only activated in case of an emergency and therefore only the operator will have access to the system itself to respond to the emergency.

3.3 Accessing the System

The overall system is the emergency beacon and it has four subsystems, two of which are sending constant data to the JAMZ operator and those are the altitude and the location subsystems. The other two, the voice and light subsystems, are dependent on the altitude subsystem and would only turn on when the altitude sensor reaches a threshold of less than the one specified in the code.

There is no need for a password or access ID to turn on the emergency beacon because it is preprogrammed. Any changes or modifications that need to be made can easily be made to the code itself and tested afterwards.

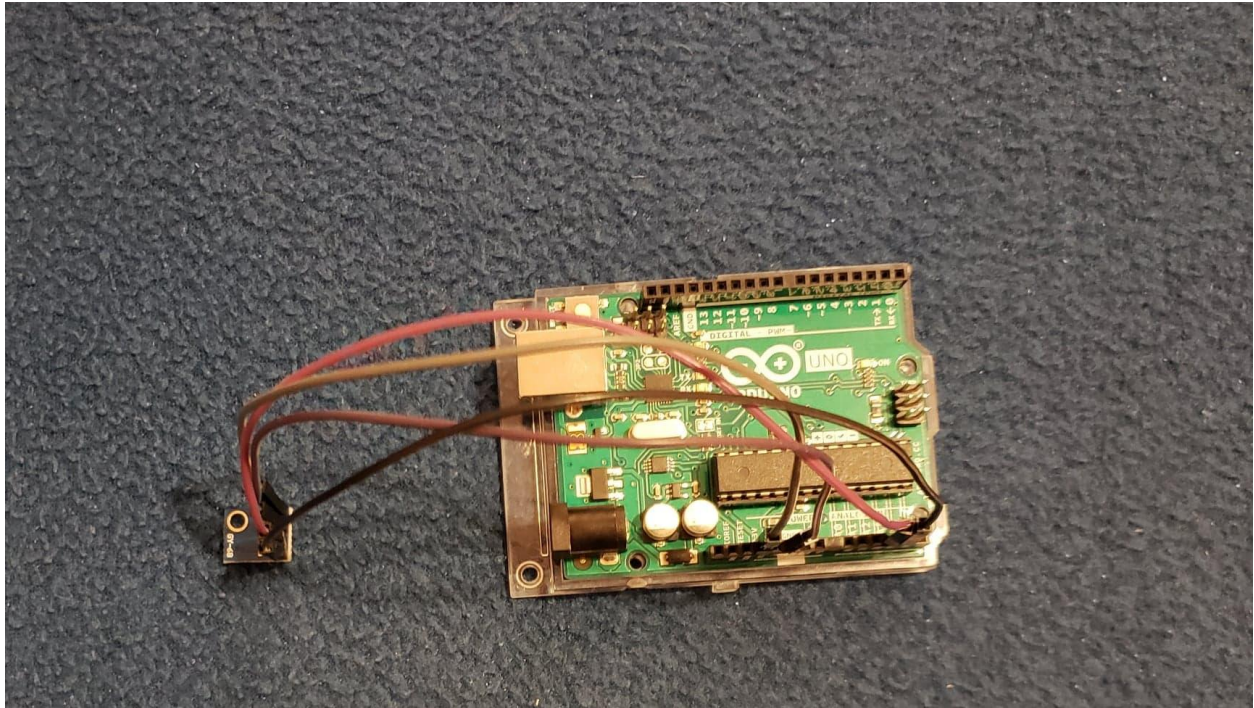
3.4 System Organization & Navigation

The following subsections will describe the hierarchical architecture of the emergency beacon described in this report.

3.4.1 Altitude Subsystem

The altitude subsystem is one of the input systems that senses the surrounding pressure and temperature of the drone, converts it to altitude data and sends it to the Arduino as well as comparing it threshold value in the code. From there, it transfers the data to the RPi microcontroller which saves the data onboard of the drone. If the threshold is met, a message will be sent to the JAMZ remote operator. If the threshold is reached, the output subsystems will be activated. Below is a figure that shows the proper connections that are needed to properly set up the altitude sensor and connect to the Arduino. Some of the equipment used for this subsystem are the barometric pressure sensor, an Arduino and some wires.

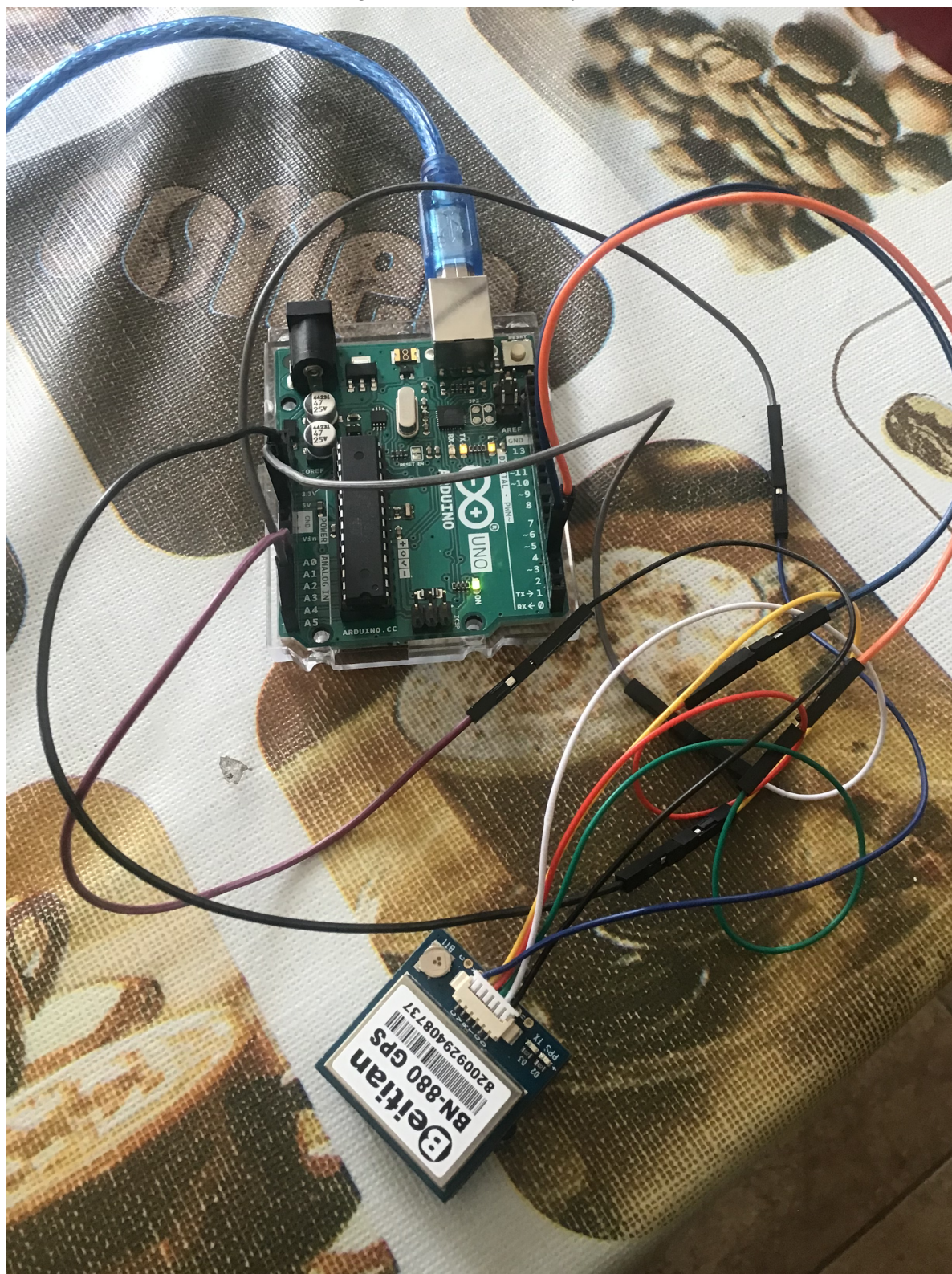
Figure 5 - Altitude Subsystem



3.4.2 Location Subsystem

The location subsystem is another one of our input subsystems and it transmits the longitude and latitude coordinates to the Arduino. Then, it outputs those coordinates to the JAMZ operator. This subsystem runs continuously and sends constant data to the operator so the drone can be easily tracked at all times. Below is an image showing all the connections needed for a functional location system. Some equipment used include the GPS module, an Arduino and some wires that came with the GPS module.

Figure 6 - Location Subsystem



3.4.3 Voice Subsystem

The voice subsystem is one of our output subsystems as it outputs an automated message once the threshold has been reached by the altitude system. The specific message is “Danger, danger, move, operator is on alert.” Below is a figure displaying the proper wiring used for this subsystem. Some equipment used in this subsystem can include two speakers, an Arduino, a breadboard, a resistor and some wires.

Figure 7 - Voice Subsystem Connections Using Tinkercad

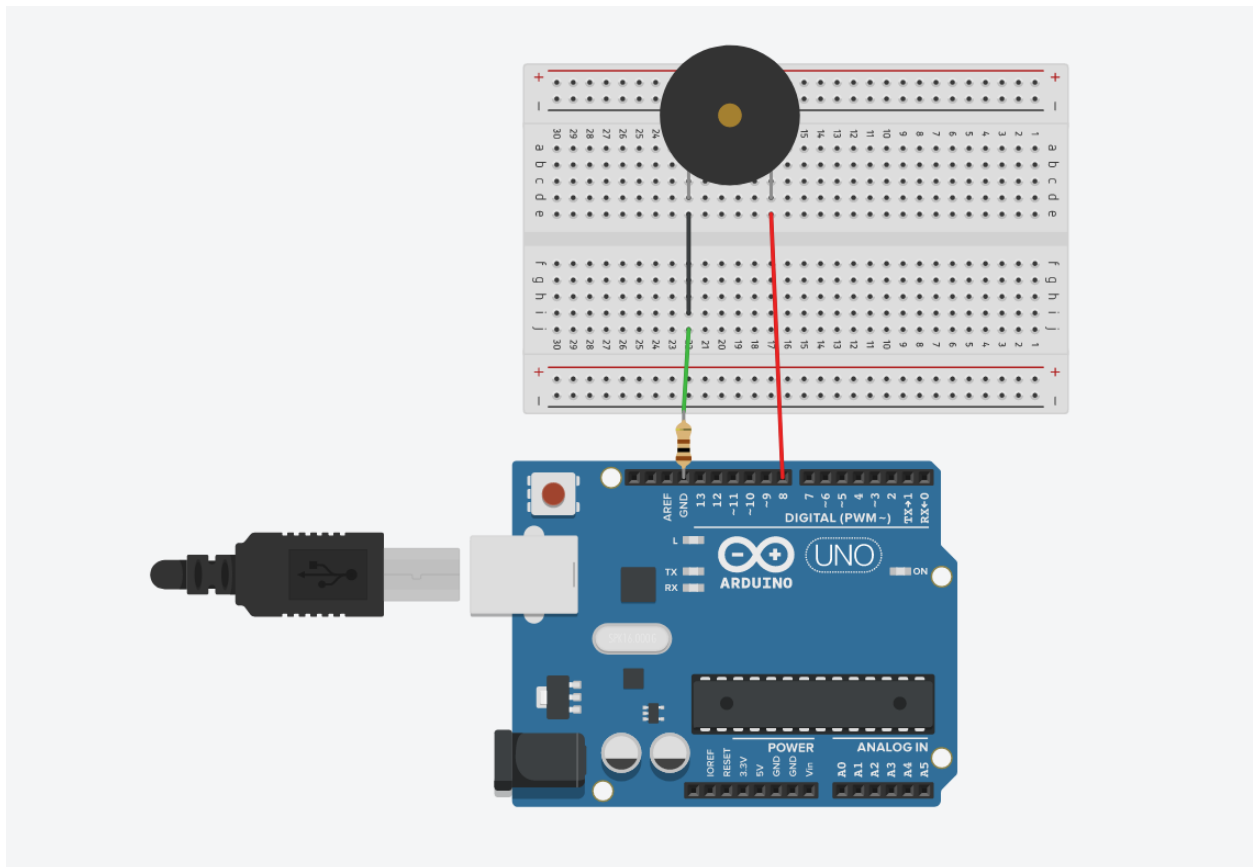
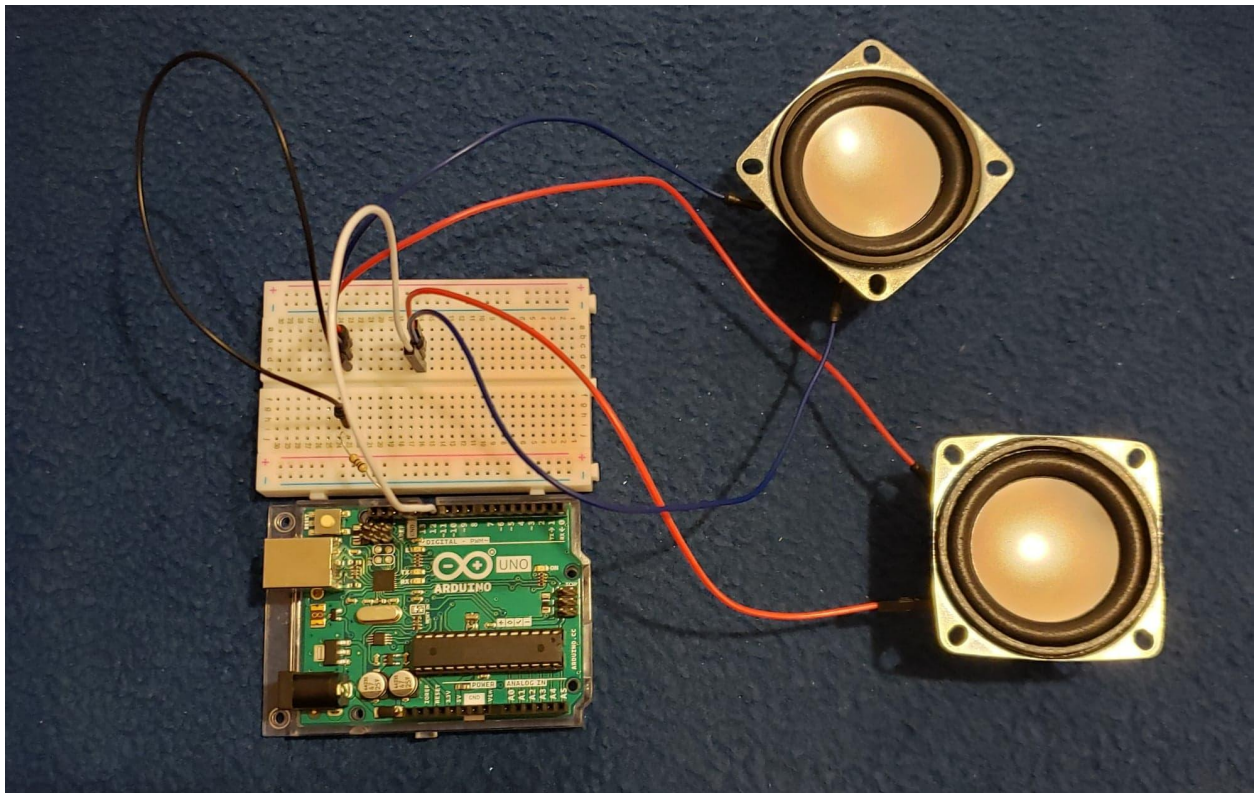


Figure 8 - Voice Subsystem



3.4.4 Light Subsystem

The light subsystem is the other output subsystem and if the threshold has been reached, the LED light turns on, one after the other, to serve as a visual warning to bystanders. Below is a picture of the proper wiring for the light subsystem. Some equipment used in this system are the LED lights, the breadboard, an Arduino, resistor and wires.

Figure 9 - Light Subsystem Connections Using Tinkercad

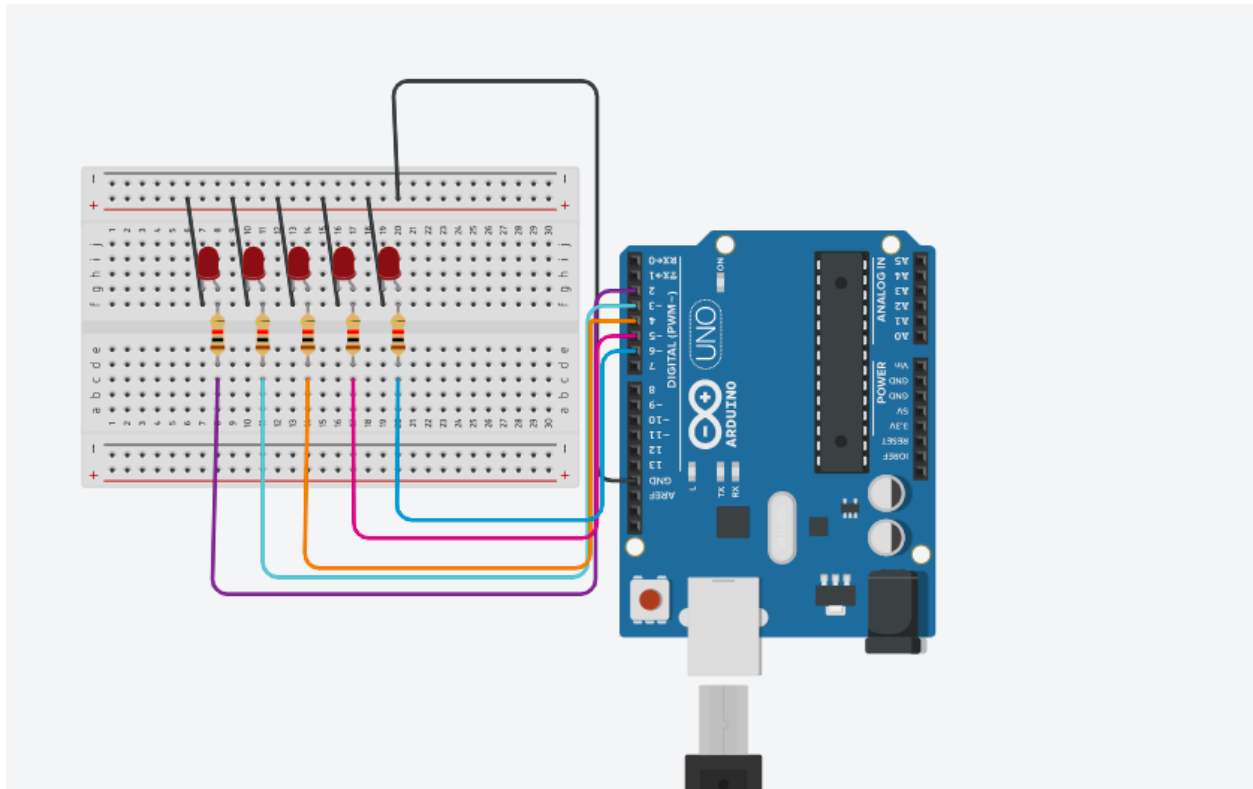
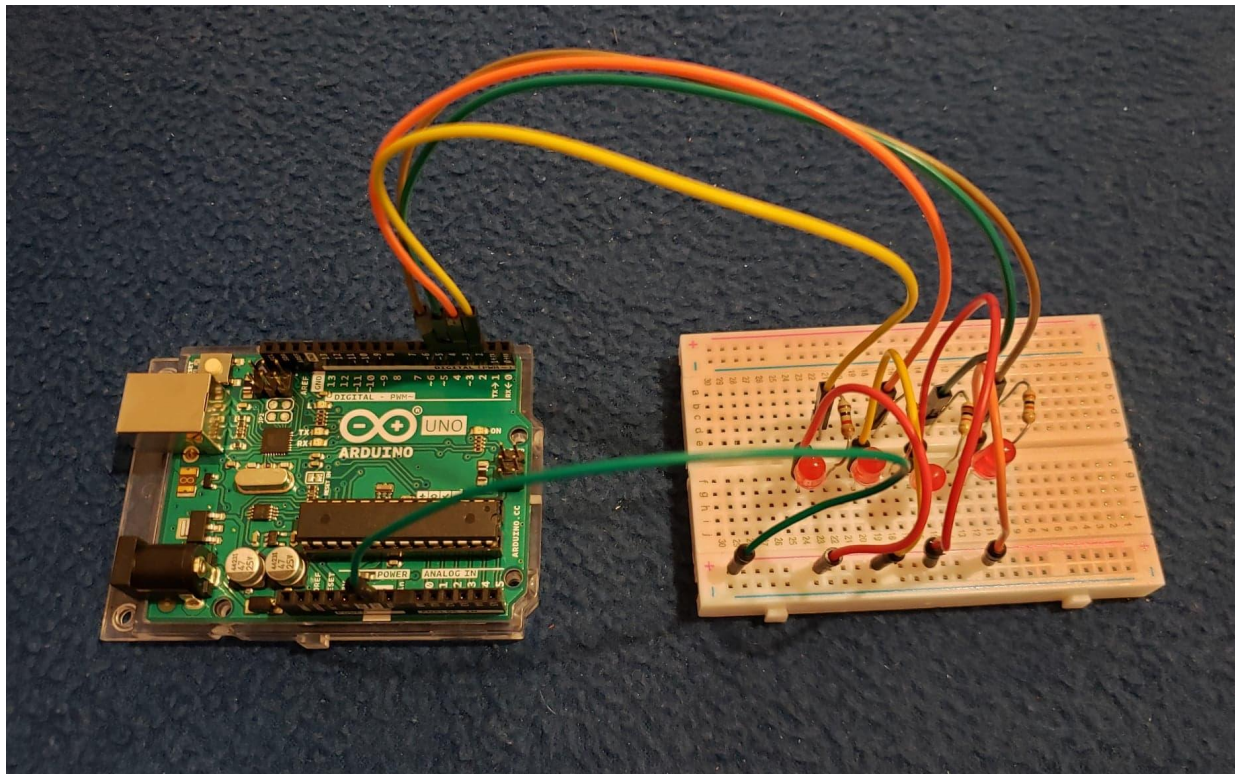


Figure 10 - Light Subsystem



3.5 Exiting the System

Once the threshold has been reached, the emergency beacon will loop infinitely until an operator turns it off manually and this feature was included to ensure no one approaches the downed drone. The operator will be the only one capable of turning the system off by cutting the power supply.

4 Using the System

In this section of the product manual, we will review the different subsystems that make up the emergency beacon and look into their required inputs, produced outputs, the systems various functions and features as well as how to use them.

4.1 Altitude Subsystem

The altitude subsystem is made up of a BMP-180 microchip, male to male wires and the overall system's microcontroller, the Arduino Uno. From a user perspective, if the system is set up properly and attached to the drone in a location where the temperature is from -40 to 85°C , then the system will provide accurate altitude data to ± 1 m. The main purpose of the altitude subsystem is to determine when the output subsystems should be activated, but the actual altitude can be transmitted to the drone operator which will be explained further in [section 4.5](#).

Figure 11 - Altitude Subsystem Physical Prototype

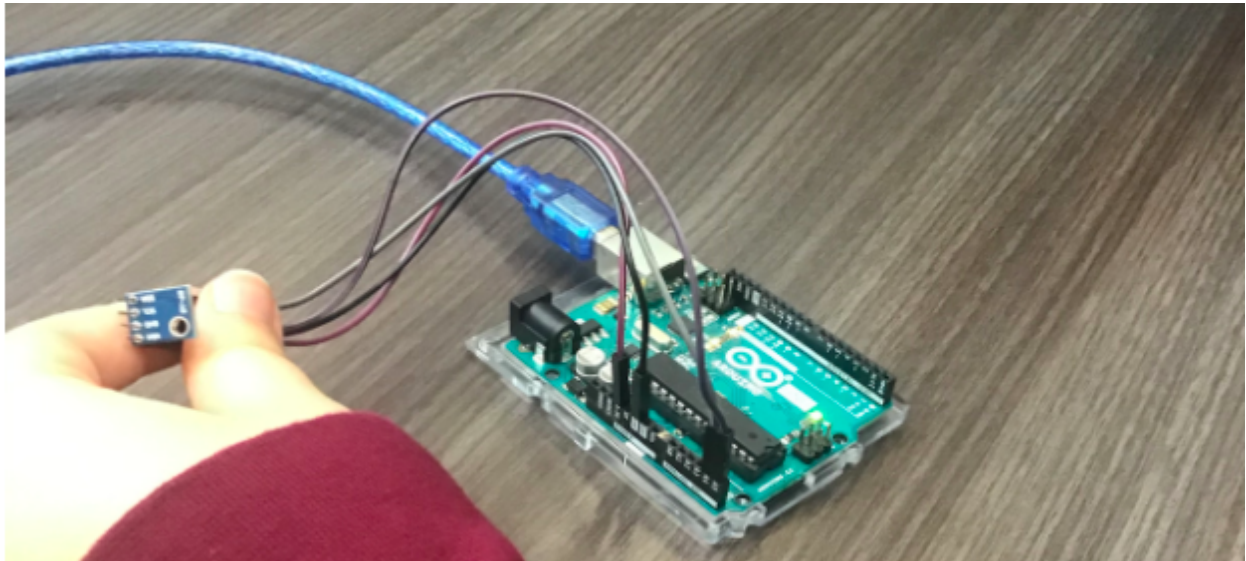


Figure 12 - Altitude Subsystem Output Data

```

COM3
REBOOT
BMP180 init success
baseline pressure: 1002.51 mb
relative altitude: 0.3 meters, 1 feet
relative altitude: 0.1 meters, 0 feet
relative altitude: -0.3 meters, -1 feet
relative altitude: 0.1 meters, 0 feet
relative altitude: -0.0 meters, -0 feet
relative altitude: 0.1 meters, 0 feet
relative altitude: 0.2 meters, 1 feet
relative altitude: 0.3 meters, 1 feet
relative altitude: 0.1 meters, 0 feet
relative altitude: 0.6 meters, 2 feet
relative altitude: 0.0 meters, 0 feet
relative altitude: 0.1 meters, 0 feet
error retrieving temperature measurement

relative altitude: 44330.0 meters, 145440 feet
relative altitude: 0.6 meters, 2 feet
relative altitude: 0.2 meters, 1 feet
relative altitude: 0.2 meters, 0 feet
relative altitude: 0.7 meters, 2 feet
relative altitude: -0.1 meters, -0 feet
relative altitude: 0.0 meters, 0 feet
relative altitude: -0.3 meters, -1 feet
relative altitude: -0.1 meters, -0 feet
relative altitude: -0.2 meters, -1 feet
relative altitude: 0.0 meters, 0 feet
relative altitude: 0.4 meters, 1 feet
relative altitude: 0.7 meters, 2 feet
relative altitude: 0.0 meters, 0 feet
relative altitude: 0.2 meters, 1 feet
relative altitude: -0.1 meters, -0 feet
relative altitude: 0.1 meters, 0 feet
relative altitude: 0.3 meters, 1 feet
relative altitude: 0.1 meters, 0 feet
  
```

4.2 Location Subsystem

The location subsystem is made up of a Beitian-BN-880 GPS module that uses wires to connect to the Arduino Uno microcontroller. It is important to understand that the Beitian-BN-880 GPS module uses satellites in order to determine the location and thus, this device will only function when the system is outdoors and able to communicate properly with the satellites. The uncertainty on the device is ± 2 m.

Figure 13 - Location Subsystem Physical Prototype

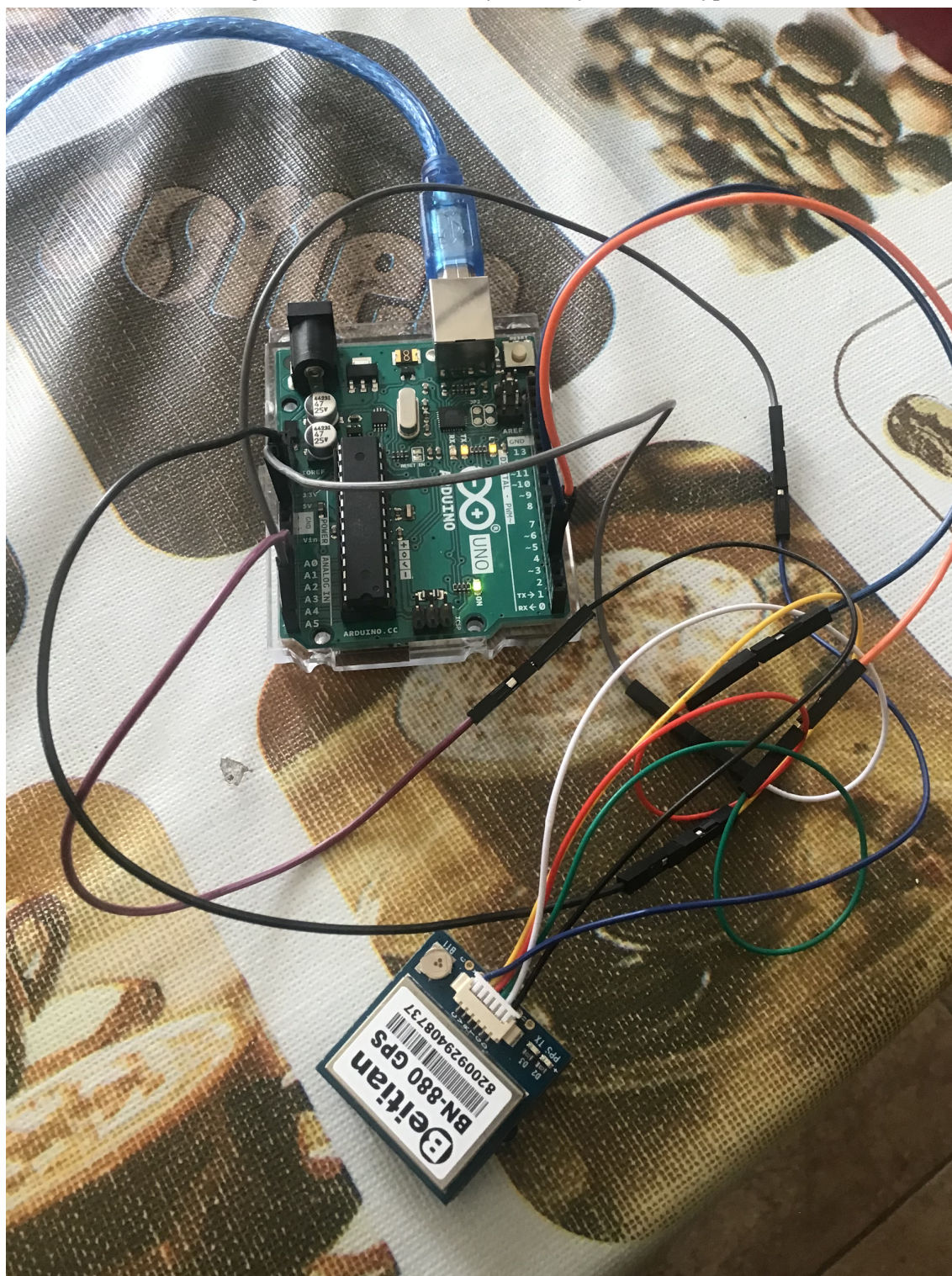


Figure 14 - Location Subsystem Output Data

14:04:17.822 -> FullExample.ino
 14:04:17.822 -> An extensive example of many interesting TinyGPS++ features
 14:04:17.822 -> TestingFullExample.ino
 14:04:19.408 -> An extensive example of many interesting TinyGPS++ features
 14:04:19.408 -> Testing TinyGPS++ library v. 1.0.2
 14:04:19.408 -> by Mikal Hart
 14:04:19.408 ->

Sats	HDOP	Latitude (deg)	Longitude (deg)	Fix	Date	Time	Date Alt Age (m)	Course --- from GPS ---	Speed	Card	Distance ---- to London ----	Course Card	Chars RX	Sentences RX	Checksum Fail
14:04:19.455 ->	100.0	*****	*****	****	03/25/2021	18:04:17	568	*****	*****	*****	*****	*****	0	0	0
14:04:20.476 -> 0	100.0	*****	*****	****	03/25/2021	18:04:18	571	*****	*****	*****	*****	*****	404	0	0
14:04:21.462 -> 0	100.0	*****	*****	****	03/25/2021	18:04:19	579	*****	*****	*****	*****	*****	739	0	0
14:04:22.477 -> 0	100.0	*****	*****	****	03/25/2021	18:04:20	584	*****	*****	*****	*****	*****	1143	0	0
14:04:23.467 -> 0	100.0	*****	*****	****	03/25/2021	18:04:21	590	*****	*****	*****	*****	*****	1478	0	0
14:04:24.471 -> 0	100.0	*****	*****	****	03/25/2021	18:04:22	597	*****	*****	*****	*****	*****	1802	0	0
14:04:25.491 -> 0	100.0	*****	*****	****	03/25/2021	18:04:23	603	*****	*****	*****	*****	*****	2217	0	0
14:04:26.516 -> 0	100.0	*****	*****	****	03/25/2021	18:04:24	606	*****	*****	*****	*****	*****	2621	0	0
14:04:27.504 -> 0	100.0	*****	*****	****	03/25/2021	18:04:25	613	*****	*****	*****	*****	*****	2956	0	0
14:04:28.539 -> 0	100.0	*****	*****	****	03/25/2021	18:04:26	621	*****	*****	*****	*****	*****	3360	0	0
14:04:29.526 -> 0	100.0	*****	*****	****	03/25/2021	18:04:27	626	*****	*****	*****	*****	*****	3695	0	0
14:04:30.513 -> 0	100.0	*****	*****	****	03/25/2021	18:04:28	630	*****	*****	*****	*****	*****	4099	0	0
14:04:31.546 -> 0	100.0	*****	*****	****	03/25/2021	18:04:29	635	*****	*****	*****	*****	*****	4434	0	0
14:04:32.531 -> 0	100.0	*****	*****	****	03/25/2021	18:04:30	640	*****	*****	*****	*****	*****	4838	0	0
14:04:33.524 -> 0	100.0	*****	*****	****	03/25/2021	18:04:31	648	*****	*****	*****	*****	*****	5173	0	0
14:04:34.557 -> 0	100.0	*****	*****	****	03/25/2021	18:04:32	653	*****	*****	*****	*****	*****	5577	0	0
14:04:35.543 -> 0	100.0	*****	*****	****	03/25/2021	18:04:33	661	*****	*****	*****	*****	*****	5912	0	0
14:04:36.562 -> 0	100.0	*****	*****	****	03/25/2021	18:04:34	667	*****	*****	*****	*****	*****	6316	0	0
14:04:37.582 -> 0	100.0	*****	*****	****	03/25/2021	18:04:35	733	*****	*****	*****	*****	*****	6701	0	0
14:04:38.619 -> 0	100.0	*****	*****	****	03/25/2021	18:04:36	740	*****	*****	*****	*****	*****	7055	0	0
14:04:39.652 -> 0	100.0	*****	*****	****	03/25/2021	18:04:37	745	*****	*****	*****	*****	*****	7459	0	0
14:04:40.640 -> 0	100.0	*****	*****	****	03/25/2021	18:04:38	752	*****	*****	*****	*****	*****	7794	0	0
14:04:41.672 -> 0	100.0	*****	*****	****	03/25/2021	18:04:39	758	*****	*****	*****	*****	*****	8198	0	0
14:04:42.658 -> 0	100.0	*****	*****	****	03/25/2021	18:04:40	765	*****	*****	*****	*****	*****	8533	0	0
14:04:43.676 -> 0	100.0	*****	*****	****	03/25/2021	18:04:41	770	*****	*****	*****	*****	*****	8937	0	0
14:04:44.651 -> 0	100.0	*****	*****	****	03/25/2021	18:04:41	770	*****	*****	*****	*****	*****	9272	0	0

☐ Autoscroll ☒ Show timestamp Newline 115200 baud Clear output

The figure above shows the results of this test on the location subsystem.

4.3 Light Subsystem

The light subsystem is attached to the breadboard and receives its power from the 3V pin on the Arduino which is attached to the power rail. Other connections are also made between the Arduino and the breadboard that allows the lights to turn on one at a time sequentially. The timing of the LED lights is determined by a function in the code and the flash pattern of each light can be modified depending on the user's preference. As it relies on the altitude threshold (which calculates a baseline altitude and outputs the change in altitude based on the baseline), it is important to ensure the optimal threshold is chosen to ensure safety of pedestrians and the drone.

Figure 15 - Light Subsystem Physical Prototype

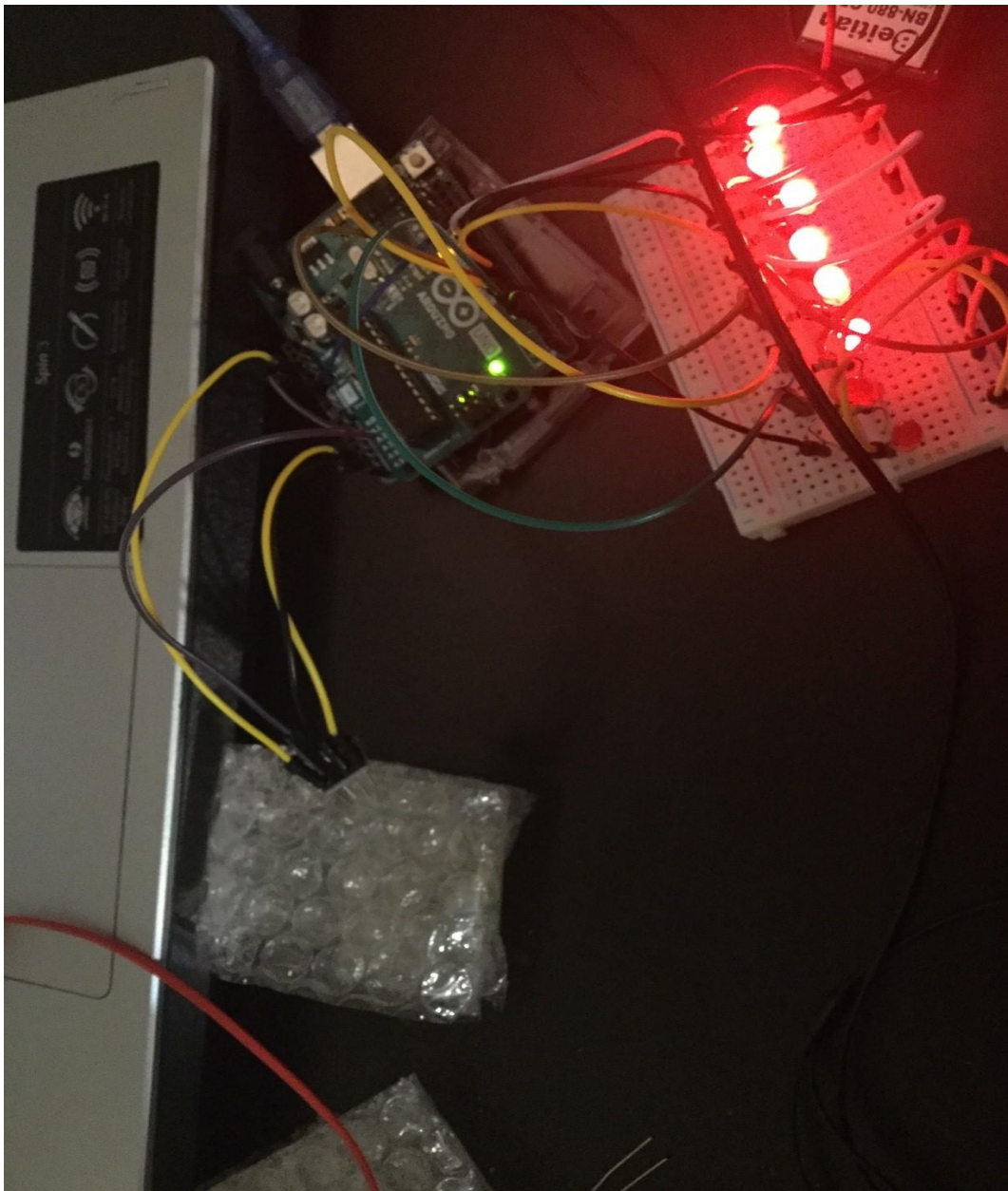
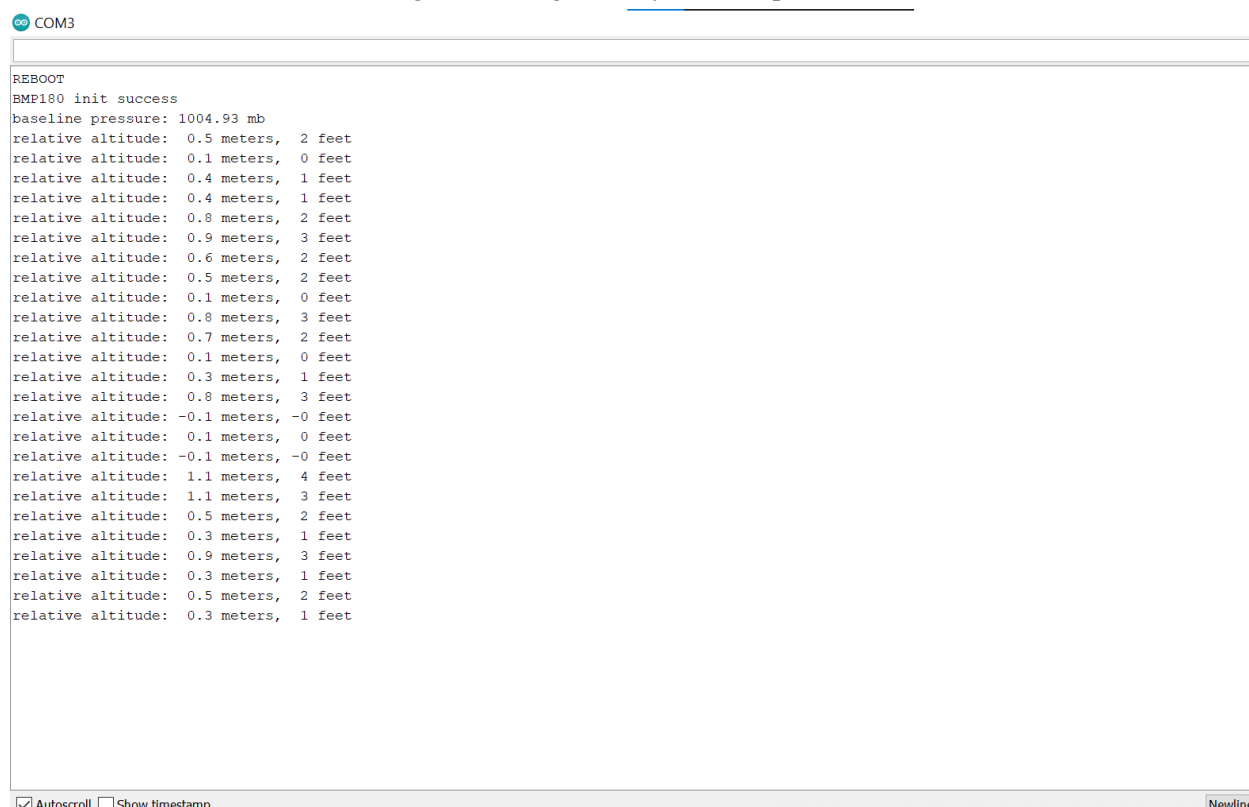


Figure 16 - Light Subsystem Output Data



```

COM3
REBOOT
BMP180 init success
baseline pressure: 1004.93 mb
relative altitude: 0.5 meters, 2 feet
relative altitude: 0.1 meters, 0 feet
relative altitude: 0.4 meters, 1 feet
relative altitude: 0.4 meters, 1 feet
relative altitude: 0.8 meters, 2 feet
relative altitude: 0.9 meters, 3 feet
relative altitude: 0.6 meters, 2 feet
relative altitude: 0.5 meters, 2 feet
relative altitude: 0.1 meters, 0 feet
relative altitude: 0.8 meters, 3 feet
relative altitude: 0.7 meters, 2 feet
relative altitude: 0.1 meters, 0 feet
relative altitude: 0.3 meters, 1 feet
relative altitude: 0.8 meters, 3 feet
relative altitude: -0.1 meters, -0 feet
relative altitude: 0.1 meters, 0 feet
relative altitude: -0.1 meters, -0 feet
relative altitude: 1.1 meters, 4 feet
relative altitude: 1.1 meters, 3 feet
relative altitude: 0.5 meters, 2 feet
relative altitude: 0.3 meters, 1 feet
relative altitude: 0.9 meters, 3 feet
relative altitude: 0.3 meters, 1 feet
relative altitude: 0.5 meters, 2 feet
relative altitude: 0.3 meters, 1 feet

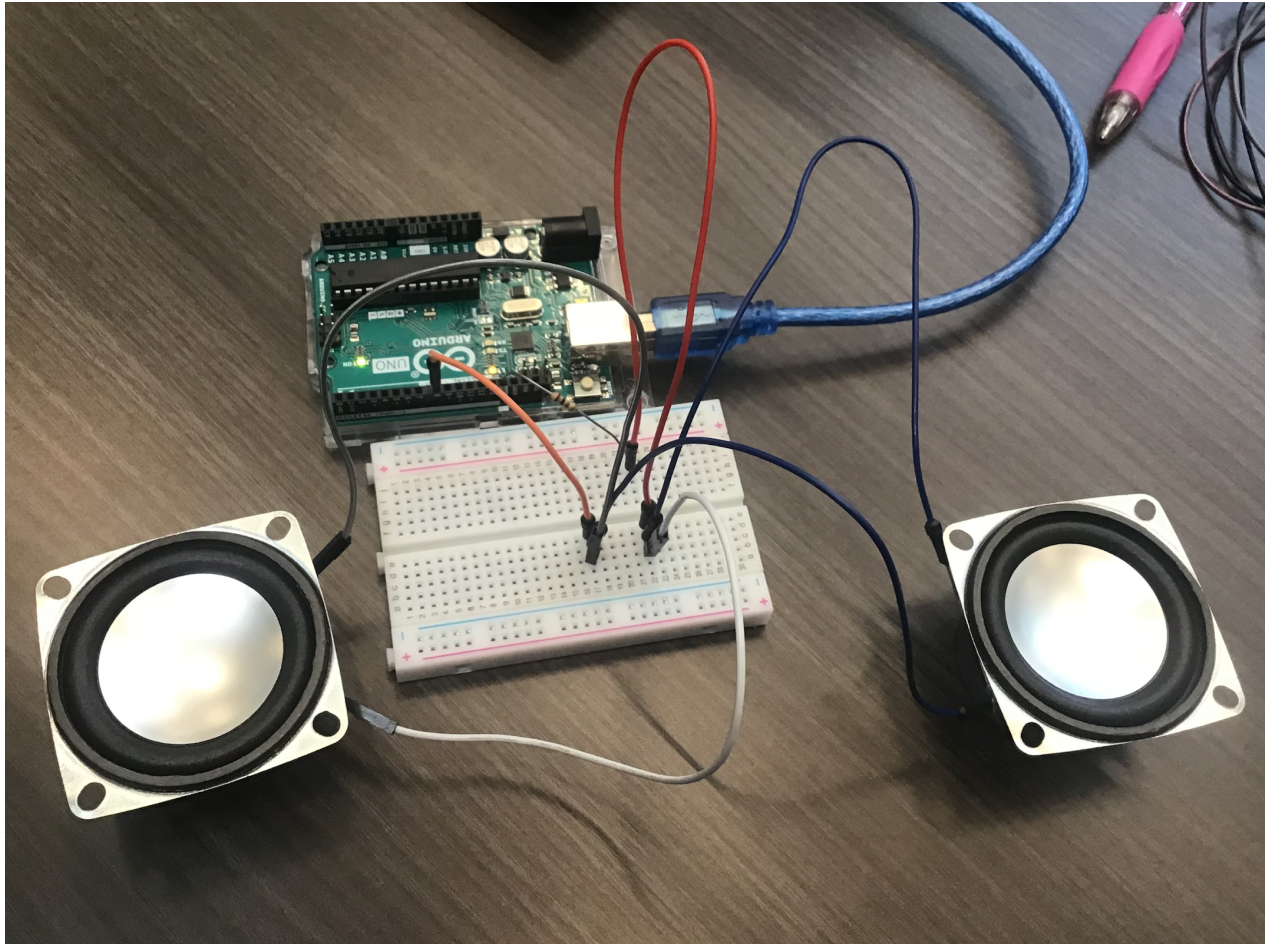
```

Autoscroll Show timestamp Newline

4.4 Voice Subsystem

The voice subsystem is attached to the breadboard and receives its power from the 5V pin on the Arduino which is attached to the power rail. The voice subsystem can be activated by the altitude subsystem passing the threshold value as previously mentioned. The exact period of time the drone must be stopped before the speakers are activated can be changed within the code depending on user preference. Also, the actual message relayed from the speakers was created using words from the “Talkie” library for Arduinos and can be modified to suit users' needs. This subsystems final feature is its ability to inform the operator when the system has been activated.

Figure 17 - Voice Subsystem Physical Prototype



4.5 Interconnections/Data Transmission

The overall beacon interconnections involve all of the subsystems attachments to the Arduino and therefore their connections to each other. The code for this feature is a compilation of all the systems code which rely heavily on each other to activate certain systems. Since all the interpreted data from any component on the beacon passes through the Arduino this data could be transmitted to an operator through a wired connection to another microcontroller or by attaching a transmission device to the Arduino itself. The data is reliable as there is a low uncertainty on the sensors and the updates may be adjusted. Also, this data is collected very frequently and can be provided to the beacons user continuously in real time or only once threshold values have been passed depending on user needs and preference.

Figure 18 - Emergency Beacon Physical Prototype

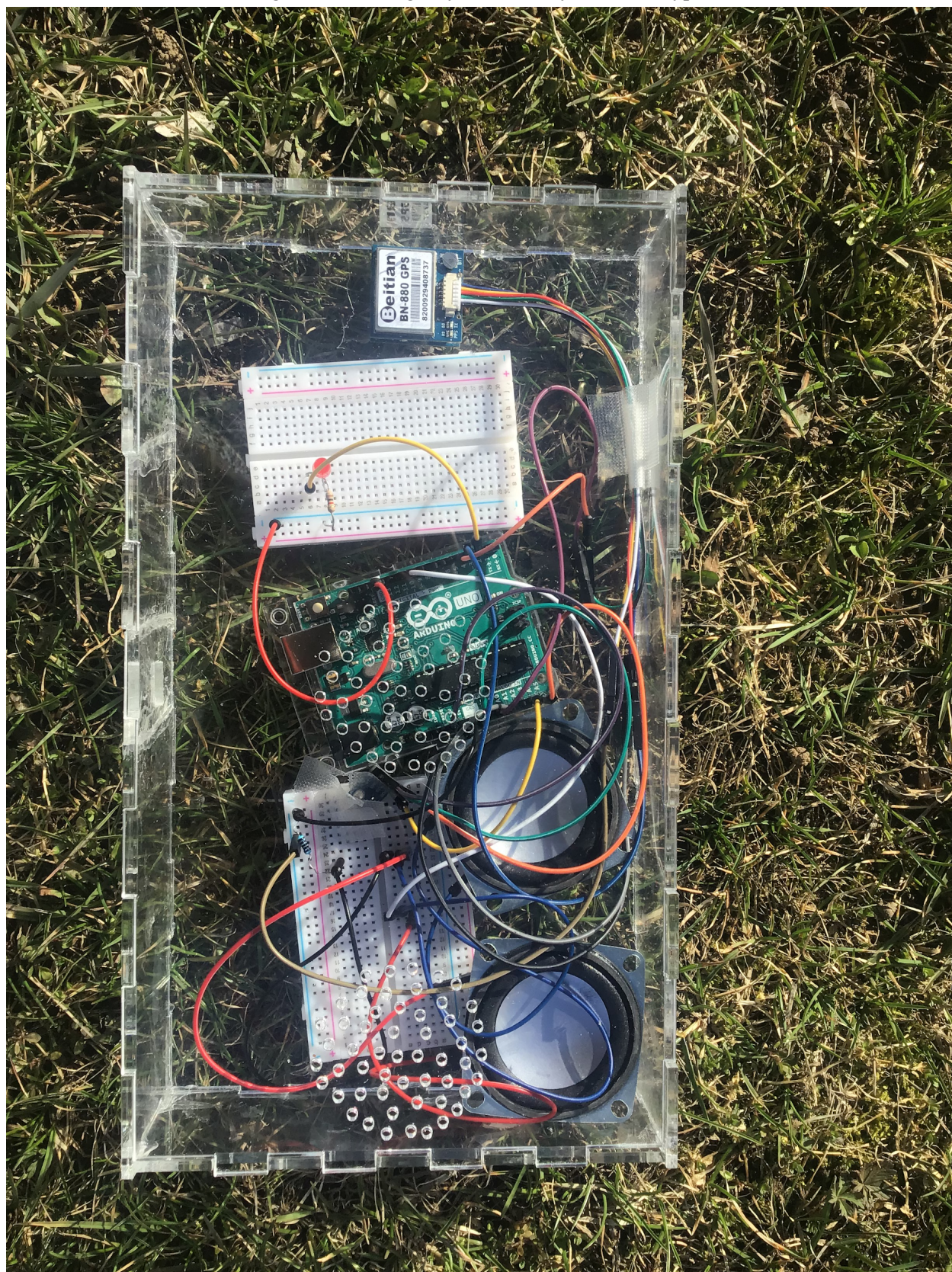
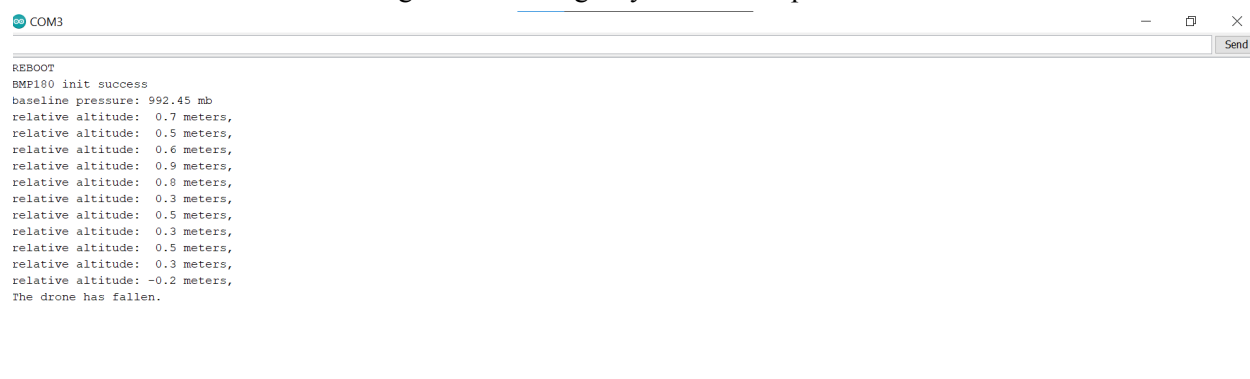






Figure 19 - Emergency Beacon Output Data



```

COM3
REBOOT
BMP180 init success
baseline pressure: 992.45 mb
relative altitude: 0.7 meters,
relative altitude: 0.5 meters,
relative altitude: 0.6 meters,
relative altitude: 0.9 meters,
relative altitude: 0.8 meters,
relative altitude: 0.3 meters,
relative altitude: 0.5 meters,
relative altitude: 0.3 meters,
relative altitude: 0.5 meters,
relative altitude: 0.3 meters,
relative altitude: -0.2 meters,
The drone has fallen.

```

5 Troubleshooting & Support

5.1 Error Messages or Behaviors

Since this emergency beacon contains multiple subsystems, there are a few error messages that can be encountered while the emergency beacon is running. These errors may be fixable using the methods recommended in this section of the user manual. In general, check the wiring to make sure all of the components are properly connected.

A common error that can be encountered is that the location output data can be seen as a series of garbled numbers and symbols on the serial monitor. This can be caused by the Arduino not being outside, and the Gps sensor not collecting actual location data. The fix to this is to simply take the beacon outside in an open area where it can function properly.

Another common error for the location subsystem is an improper baud rate. If the baud rate is too small then inverted question marks will print to the serial monitor. Simply change the baud rate in the serial monitor to match the line “serial.begin(#####)”.

An associated error for the voice and light subsystems is the resistors not connecting properly to the breadboard.

5.2 Special Considerations

Since the emergency beacon is made up of very fragile and water sensitive parts, some precautions may have to be taken in order to preserve the integrity of the beacon. These precautions will decrease the likelihood of the emergency beacon failing in any way.

The first precaution that should be taken is to never leave the beacon in an area with excess moisture in the air, such as near a fridge, outside on a rainy day or near a pool because it may seep through the filter and damage the electronics.

Another precaution that should be taken is to never open the casing and touch the components inside unless there is an issue with the components, so that no unnecessary damage is caused.

A major consideration to consider is that the time zone of the location subsystem's code is currently based off of UTC, meaning that the results from the code will also be in UTC. The user must change this function to suit their local time zone.

Two physical considerations are that the speakers are faint so put your ear close to the holes to hear, and that the LED lights are best seen in the dark .

The final consideration that should be taken is warming up the GPS module. The GPS module will need 15-20 minutes to warm up and establish connection with a satellite, during which time, it must be turned on, and have a direct link to the satellite (i.e. outside with nothing over it). After this warm up period, the module will function normally.

5.3 Maintenance

The maintenance of the emergency beacon system can be divided into two parts: maintenance of the case and maintenance of the electronic components. In this section, the main maintenance needs will be addressed.

5.3.1 Maintenance of the Case

To maintain the case, one should wipe the exterior down regularly to remove any particles/dust that may cling to the case. The speaker holes may also be cleaned with a cotton swab, and if necessary, a minimal amount of water can be used to help the swab pick up dirt particles. For the interior of the case, one must take out the emergency beacon and wipe it down in a similar fashion to the exterior, taking care to ensure that the interior is completely dry before re-placing the subsystem.

5.3.2 Maintenance of the Components

To maintain the electronic components, a fidelity check is recommended every month to ensure that each component is still functioning and serving its designed purpose. The electronics all need to be changed after a certain amount of usage which can be found on their product websites. Another point about electronic maintenance is to change the jumper wires every month, to ensure that there is no potential wear and tear on the wires, and to ensure the overall fidelity of the system.

5.4 Support

The first step to technical support for our end users is to try to examine a problem by yourself. The first thing that should be checked is the wiring in the subsystems to see if any connections have loosened or disconnected. The next step is to check the code to see if there are any errors or typos.

If you have further questions, or need assistance with your emergency beacon, please search our MakerRepo for a solution to your problem at the following [link](#).

6 Product Documentation

6.1 Subsystems of the prototype

In this section, every subsystem includes: a bill of materials, the equipment lists, set up instructions and tests and validations.

6.1.1 BOM (Bill of Materials)

6.1.1.1 Altitude Subsystem

Table 3 - Bill of Materials of the Altitude Subsystem

Item	Number	Purpose	Cost (in CAD with tax and shipping included)	Source
Arduino Uno	1	Has the code to convert the absolute pressure to altitude with a code.	\$19.21	https://makerstore.ca/shop/ols/products/arduino-uno-r3
Barometric Pressure Sensor	1	Measures the absolute pressure which can be converted to altitude with a code.	\$7.89	https://voltatek.ca/sensors/155-bmp180-digital-barometric-pressure-sensor.html
Wires	1 set of 10 wires	Connect the Arduino to the barometric pressure sensor.	\$1.13	https://makerstore.ca/shop/ols/products/jumper-cables-per-10
		Total Cost	\$28.23	

6.1.1.2 Location Subsystem

Table 4 - Bill of Materials for Location Subsystem

Item	Number	Purpose	Cost (in CAD with tax and shipping included)	Source
Arduino Uno	1	Has the code to convert the absolute pressure to altitude with a code.	\$19.21	https://makerstore.ca/shop/ols/products/arduino-unor3
Beitian BN-880	1	Transmits location data to the Arduino. This product comes with its own set of wires.	\$29.98	https://www.amazon.ca/DIYmalls-HMC5883-Compass-Raspberry-Betaflight/dp/B086ZKFFY4/ref=sr_1_10?crd=2FZ0BDUGG1BEC&dchild=1&keywords=arduino+gps+module&qid=1614045144&srefix=arduino+gps+%2Caps%2C177&sr=8-10
		Total cost	\$49.19	

6.1.1.3 Voice Subsystem

Table 5 - Bill of Materials for the Voice Subsystem

Item	Number	Purpose	Cost (in CAD with tax and shipping included)	Source
Arduino Uno	1	Has the code to convert the absolute pressure to altitude with a code.	\$19.21	https://makerstore.ca/shop/ols/products/arduino-uno-r3
Speakers	2 for specified price	Connects to an Arduino and can play an automated message.	\$14.55	https://www.amazon.ca/Gikfun-Speaker-Stereo-Loudspeaker-Arduino/dp/B01N74TGFM/ref=asc_df_B01N74TGFM/?tag=googleshopc0c-20&linkCode=df0&hvadid=292954337233&hvpos=&hvnetw=g&hvrnd=18098208748923028760&hvpone=&hvptwo=&hvqmt=&hvdvcmdl=&hvlocint=&hvlocphy=9000694&hvtargid=pla-491322515456&psc=1
Wires	1 set of 10 wires	Connect the Arduino to the speakers.	\$1.13	https://makerstore.ca/shop/ols/products/jumper-cables-per-10
Resistors	1 100 ohm resistor	Needed on the breadboard.		https://makerstore.ca/
Breadboard	1	Connect electric components	\$11.30	https://makerstore.ca/shop/ols/products/breadboard
Total cost			\$46.19	

6.1.1.4 Light Subsystem

Table 6 - Bill of Materials for the Light Subsystem

Item	Number	Purpose	Cost (in CAD with tax and shipping included)	Source
Arduino Uno	1	Has the code to convert the absolute pressure to altitude with a code.	\$19.21	https://makerstore.ca/shop/ols/products/arduino-unor3
Red LED Lights	8	Lights that will turn on.	\$4.80	https://makerstore.ca/shop/ols/products/
Wires	3 set of 10 wires	Connect the Arduino to the LED lights.	\$3.39	https://makerstore.ca/shop/ols/products/jumper-cables-per-10
Breadboard	1	Connect electric components	\$11.30	https://makerstore.ca/shop/ols/products/breadboard
Resistors	8 x 1 kohm	Needed on the breadboard.		https://makerstore.ca/
		Total Cost	\$38.7	

6.1.2 Equipment list

6.1.2.1 Altitude Subsystem

- TinkerCAD to show the visualized circuits of the subsystems.
- Arduino IDE
- Sparkfun BMP180 Arduino library for the physical altitude subsystem.

6.1.2.2 Location Subsystem

- TinkerCAD to show the visualized circuits of the subsystems.
- Arduino IDE
- TinyGPS++ Arduino library for the physical location subsystem.

6.1.2.3 Voice Subsystem

- TinkerCAD to show the visualized circuits of the subsystems.
- Arduino IDE
- Talkie Arduino library for the physical voice subsystem.

6.1.2.4 Light Subsystem

- TinkerCAD to show the visualized circuits of the subsystems.
- Arduino IDE

6.1.2.5 Acrylic Case

- Inkscape
- Laser cutter

6.1.3 Instructions

6.1.3.1 Altitude Subsystem

Figure 20 - The Wired Altitude Subsystem

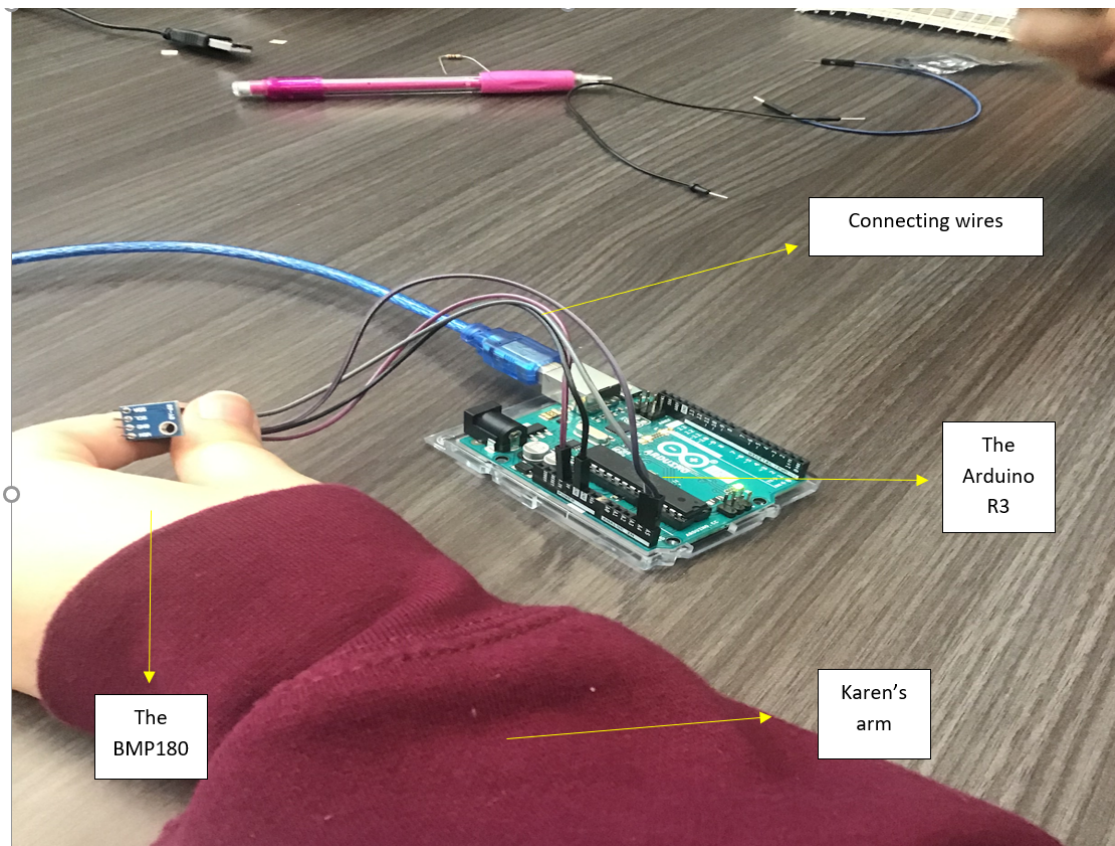


Table 7 - BMP180 and Arduino Uno Pin Set Up

BMP180 Pin	Arduino Uno Pin
VIN	3.3 V
GND	GND
SDA	A4
SCL	A5

The male to male wires need to be wired to the proper ports. On the BMP180, the VIN pin is wired to 3.3V pin on the Arduino Uno. Moreover, the GND pin on BMP180 is connected to the GND pin on Arduino Uno. The SDA pin connects to the A4 port on Arduino Uno and the SCL is connected to the A5 pin. Note that the BMP180 must be snug with its wires so soldering is recommended to ensure proper connections. Tape was used in the final prototype but it will impact the accuracy of the sensor.

6.1.3.2 Location Subsystem

Figure 21 - The Wired Location Subsystem

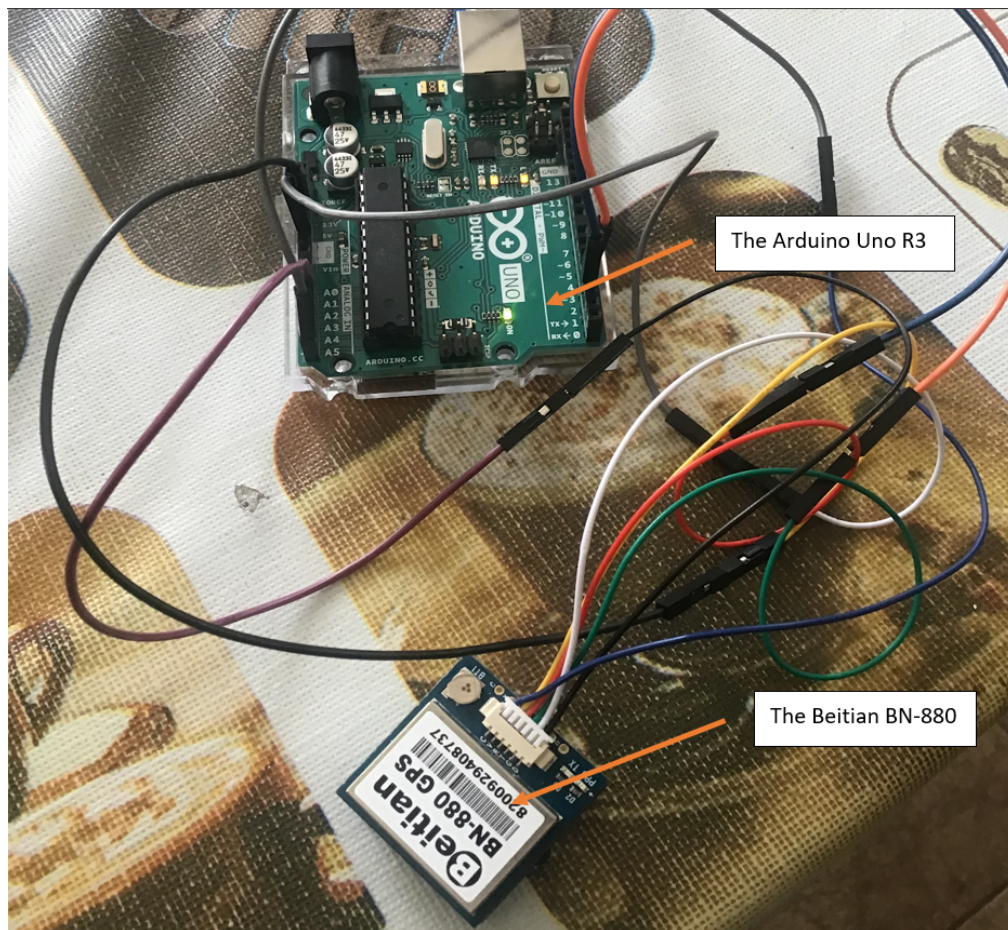


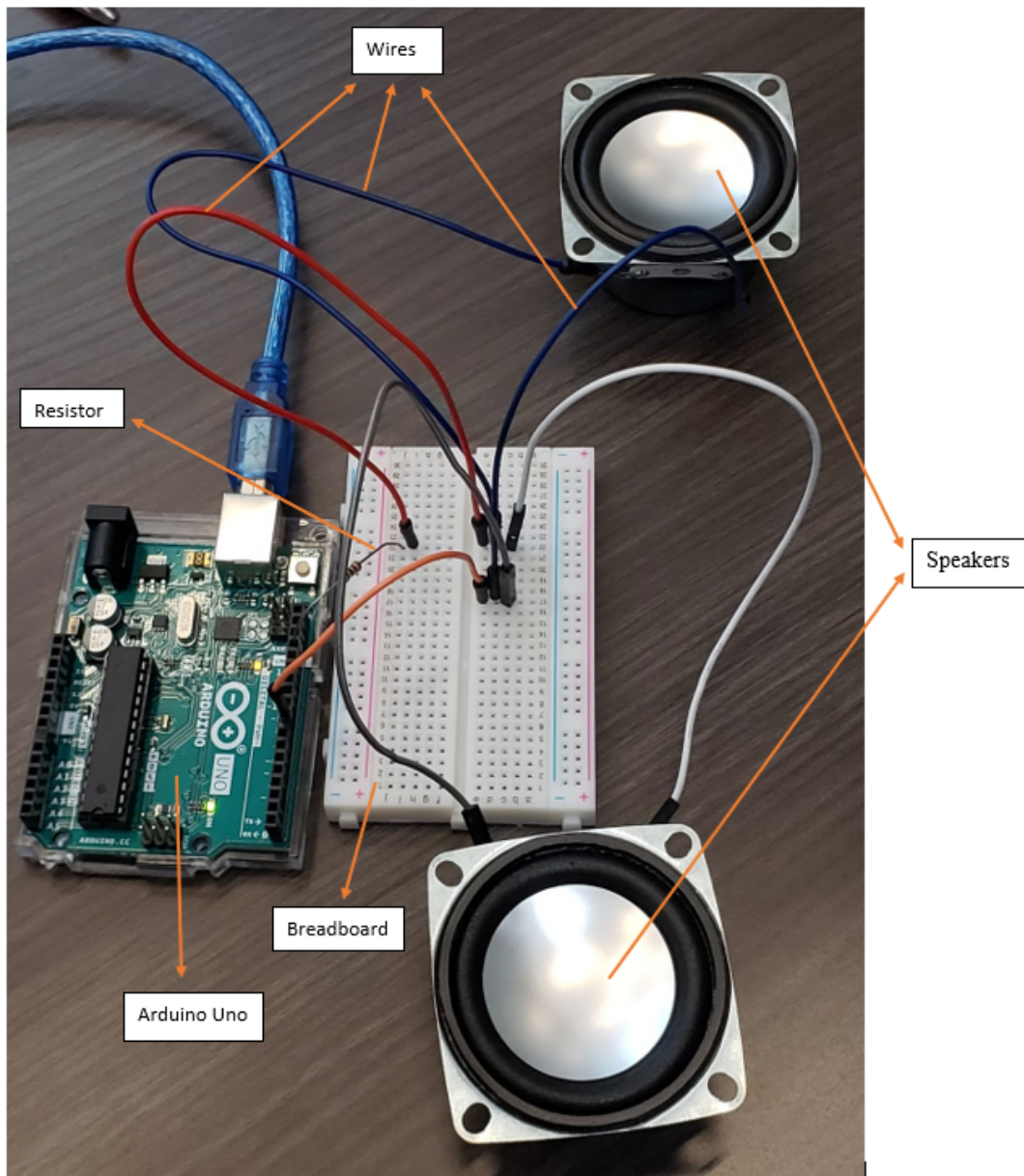
Table 8 - Beitian BN-880 and Arduino Uno Pin Set Up

Beitian BN-880 Pin	Arduino Uno Pin
V	5 V
GND	GND
R	3
T	4
D	A4
C	A5

Each port on the Beitian BN-880 pin has to be wired properly to the corresponding port on the Arduino Uno pin. The “V” port on the Beitian BN-880 corresponds to the 5V battery supply on the Arduino Uno. Furthermore, the The R and T ports on the Beitian BN-880 connect to pins 3 and 4 ports, respectively on the microcontroller. The “D” and “C” pins correspond to the A4 and A5 pins on the Arduino (note that these pins can connect to any of the analog input pins on the Arduino).

6.1.3.2 Voice Subsystem

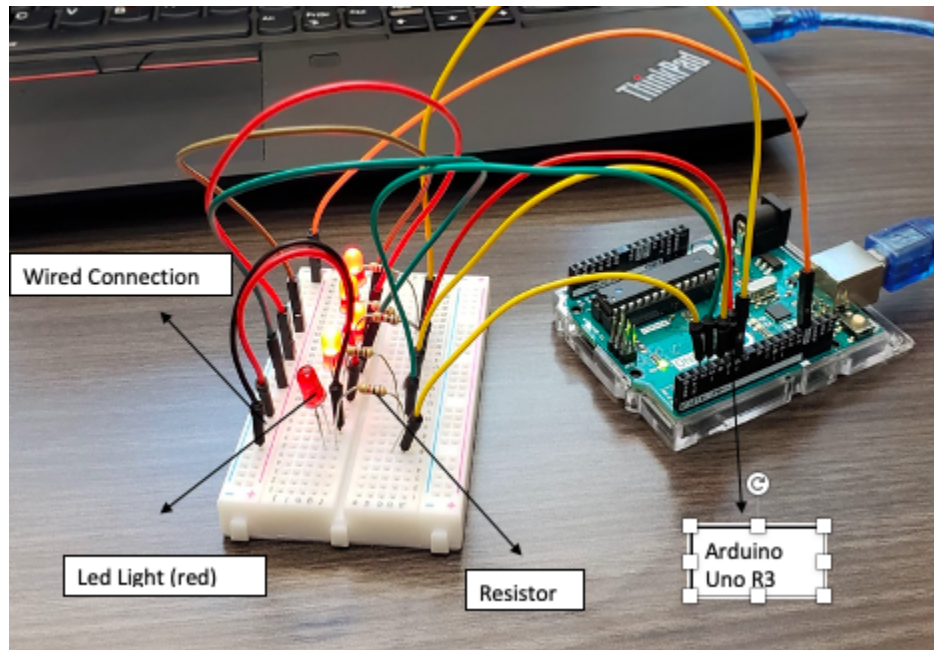
Figure 22 - The Wired Voice Subsystem



Each speaker has two pins to wire, the right pin is the negative meanwhile the left one is positive. The negative of one speaker goes to the 17 c on the 17th horizontal line, meanwhile the negative of the other speaker also goes to the 17 d. On the hand, the positive of the two speakers are wired to the c and d on 22nd horizontal line. One wire from 22e to 22i is attached. The resistor from 22j is connected to pin 11 on Arduino while the negative set has a wire connecting from 17e to GND (ground).

6.1.3.1 Light Subsystem

Figure 23 - The Wired Light Subsystem



The long side and short side of the LED lights is connected to the two corresponding g6 and g8 on the breadboard. The gold side of the resistor is attached from g6 to the breadboard's c6 then a wire from b6 is wired to pin 2 on Arduino. Next, one wire is placed from f8 to the +8 on the Breadboard and one is wired from +1 on the Breadboard to GND on the Arduino Uno. Based on the number of the LED lights used, we can repeat the steps and move up the breadboard and up the pins on Arduino.

6.2 Testing & Validations

In this section, the numerous testing will be explained for all the subsystems. Each subsystem was first tested through TinkerCAD then both isolated and physical testing was done.

6.2.1 Altitude Subsystem

The altitude subsystem was tested through TinkerCAD using a similar sensor as the BMP180 could not be simulated on the software and can be found in “Deliverable F” on MakerRepo. Subsequently, the barometric pressure sensor was tested using an Arduino and two codes: one that tested if the device was working and another that tested if it satisfied design criteria. The latter can be found [here](#). Additionally, the altitude and light subsystem were tested together which can be found in “Deliverable H” on MakerRepo.

6.2.2 Location Subsystem

The location subsystem was tested through TinkerCAD using a GPS sensor and printed to LCD screens as well as another code that printed to the serial monitor using the same code for each test. Then the location subsystem would be physically tested alongside the altitude subsystem. It was then tested using the Beitian-BN-880 and was tested isolated (using two different codes which can be found on “Deliverable H”) and integrated with the voice subsystem and in the final prototype (also on “Deliverable H”).

6.2.3 Voice Subsystem

The voice subsystem was tested through TinkerCAD using only one speaker and an initial code and a consistent loud noise was displayed which satisfied the design criteria for prototype I in “Deliverable F”. Next, a physical test was performed when all the components were delivered and that was using the same initial code and the two speakers to verify the wiring. Once the final code was updated using the “talkie” library, the same physical test was performed again using the physical components to verify that the speakers can display the automated message as can be shown in “Deliverable H” on MakerRepo.

6.2.4 Light Subsystem

The first light subsystem prototype was connected to the Arduino on TinkerCAD first. The code was also made to test and the 5 LED lights successfully turned on when the code was activated. After finishing creating the visual circuit on TinkerCAD, the LED lights were physically connected to the Arduino tested using the same initial code used in the previous test. All five red LED lights turned on one after the other. In the next step, the code was modified so the lights systems would be dependent on the altitude subsystem. Then the light and voice subsystems were combined together and connected to one Arduino; an integrated code created from the two isolated codes were used to test. These subsystems were functioning well as expected as the speakers relayed the message and the Beitian BN-880 printed to the serial monitor.

7. Conclusions and Recommendations for Future Work

In conclusion, this deliverable will supply the end user of the emergency beacon with the necessary knowledge of how to start up the emergency beacon and how to access and use the system and subsystems within the emergency beacon. The user manual will also teach the end user the correct way to utilize the system and interpret the data. Finally, this manual will teach the user how to interpret and correct error messages or abnormal behaviour, as well as provide recommendations on how to maximize the efficiency and lifespan of the emergency beacon.

Some lessons learned throughout this project include time management as extra time is often needed to complete any part. Also, making a list of tasks was always helpful as to not forget any important work needed. Lastly, doing a lot of research is very useful as it helps with the understanding of complicated systems and allows for additional learning opportunities that can be later applied in the making of the system.

For future work, soldering the components would be on the top of the list. This helps keep all the components and wires in place where there is no risk of anything disconnecting. In addition to soldering, a 3D printed base is needed for the protoboard (the device that would replace the breadboard when soldering) to safely sit on. Next on the list would have been to make the encasing for the emergency beacon smaller and more aerodynamic as to allow the drone to fly smoothly without any setbacks from the case. Also, as mentioned in the manual, the voice system is not loud enough to be heard from afar which is a problem and would be included in any future work.

8 Bibliography

<https://tc.canada.ca/en/aviation/drone-safety/flying-your-drone-safely-legally>

<https://makerepo.com/elang099/843.an-emergency-beacon-by-the-9ers-for-gng-1103>

[https://makerepo.com/rails/active_storage/blobs/eyJfcmFpbHMiOnsibWVzc2FnZSI6IkJBaHBBbGd0IiwiaXNhwIjpudWxsLCJwdXkiOiJibG9iX2lkIn19--c2d2aa432d822b75952427e4541339f5c7d8a0e8/Encasing%20Model%20%20\(Acrylic\).svg](https://makerepo.com/rails/active_storage/blobs/eyJfcmFpbHMiOnsibWVzc2FnZSI6IkJBaHBBbGd0IiwiaXNhwIjpudWxsLCJwdXkiOiJibG9iX2lkIn19--c2d2aa432d822b75952427e4541339f5c7d8a0e8/Encasing%20Model%20%20(Acrylic).svg)

<https://inkscape.org/>

<https://makerstore.ca/shop/ols/products/arduino-uno-r3>

<https://voltagek.ca/sensors/155-bmp180-digital-barometric-pressure-sensor.html>

<https://makerstore.ca/shop/ols/products/jumper-cables-per-10>

https://www.amazon.ca/DIYmalls-HMC5883-Compass-Raspberry-Betaflight/dp/B086ZKFFY4/ref=sr_1_10?crid=2FZ0BDUGG1BEC&dchild=1&keywords=arduino+gps+module&qid=1614045144&srefix=ar

[duino+gps+%2Caps%2C177&sr=8-10](#)

https://www.amazon.ca/Gikfun-Speaker-Stereo-Loudspeaker-Arduino/dp/B01N74TGFM/ref=asc_df_B01N74TGFM/?tag=googleshopc0c-20&linkCode=df0&hvadid=292954337233&hvpos=&hvnetw=g&hvrاند=18098208748923028760&hvpone=&hvptwo=&hvqmt=&hvdev=c&hvdvcmdl=&hvlocint=&hvlocphy=9000694&hvtargid=pla-491322515456&pssc=1

<https://makerstore.ca/shop/ols/products/jumper-cables-per-10>

<https://makerstore.ca/shop/ols/products/breadboard>

<https://makerstore.ca/shop/ols/products/>

<https://makerstore.ca/shop/ols/products/jumper-cables-per-10>

<https://makerstore.ca/>

https://docs.google.com/document/d/1rAdpHB7Nflo50R5ew4saTYi4TC_BRwigtCAi7fyclIE/edit#heading=h.x7smm61ffty5

https://docs.google.com/document/d/e/2PACX-1vRbEMLjiKBfBfQuN9wOEKuCdJIJmWnY2fNUZ-_XfNtOAPZnLwkBmrAjPBgrxI_X7UmdQgFAMYYUwJ9n/pub

https://docs.google.com/document/d/e/2PACX-1vT7_qSzuUl6u0mst_uJe-nDiSd1zNCu6VJ3M345qIwwW3eSJb2Su2quVw8-M4mDGhBZ5pqTRdTj3npv/pub

https://docs.google.com/document/d/e/2PACX-1vSp8M0eRZrOT51Qu1SYqEol5MN-v4clRF6kzvIM6ZEXvMJd_I2ai_474PDO53aDXbUiHdRAFOf8yGEC/pub

https://docs.google.com/document/d/e/2PACX-1vTfzKodImx9WpUGqy_K-N7akrsWrbX42PHMmZKq2jW-jBjrprkk43JR4ysLN3pxbaMx2PdMcEdxOACk/pub

https://docs.google.com/spreadsheets/d/e/2PACX-1vQlWKnEOJDR8rWDGJnl6WPF171UOkRN4w0v8_0I1PWJMiRuuW_CjZOM50-CcWvQZ0Ao1logRJYRnA3/pubhtml

<https://docs.google.com/document/d/e/2PACX-1vQFnEFIXYRHJccINZ-O6FCBWuvRW42a9FjM4onCHzMaCb30FLwqZ52s9Dxp5L2LF4tWk0Ho8D9QfZTg/pub>

<https://docs.google.com/document/d/e/2PACX-1vQFnEFIXYRHJccINZ-O6FCBWuvRW42a9FjM4onCHzMaCb30FLwqZ52s9Dxp5L2LF4tWk0Ho8D9QfZTg/pub>

https://docs.google.com/document/d/e/2PACX-1vTjPmsvXzO8je-lSqYKrKuOUL9dQuGLEu1EW7JRirMqP6Lao7zux3gNu4_9dHS62yBYwq_a4auu4lod/pub

<https://docs.google.com/presentation/d/19sxfu4h9NrIkOTl1abxEmGEIOiuN93Ar74unvNw23bU/edit?usp=sharing>

https://docs.google.com/presentation/d/e/2PACX-1vSSeBUgqp4IwiRk1V5-w48OX3sYLxvqophGFsJ7UAnTxRfy_FFZrxqH7qw1N_-LNHgvWh5o5zg5cp7w/pub?start=false&loop=false&delayms=3000

<https://www.sparkfun.com/products/retired/11824>

<https://github.com/mikalhart/TinyGPS>

<https://github.com/going-digital/Talkie>

9 APPENDICES

9.1 APPENDIX I: Design Files

Table 9 - List of Deliverables

Document Name	Document Location and/or URL	Issuance Date
Deliverable B-Needs Identification and Problem Statement	https://docs.google.com/document/d/e/2PA-CX-1vRbEMLjiKBfBfQuN9wOEKuCdJIJmWnY2fNUZ-_XfNtOAPZnLwkBmrAjPBgrxI_X7UmdQgFAmYYUwJ9n/pub	January 31, 2021
Deliverable C-Design Criteria and Design Specifications	https://docs.google.com/document/d/e/2PA-CX-1vT7_qSzuUl6u0mst_uJe-nDiSd1zNCu6VJ3M345qIwwW3eSJb2Su2quVw8-M4mDGhBZ5pqTRdTj3npv/pub	February 7, 2021
Deliverable D-Conceptual Design	https://docs.google.com/document/d/e/2PA-CX-1vSp8M0eRZrOT51Qu1SYqEol5MN-v4clRF6kzvIM6ZEXvMJd_I2ai_474PDO53aDXbUiHdRAFOf8yGEC/pub	February 21, 2021
Deliverable E-Project Schedule and Cost	https://docs.google.com/document/d/e/2PA-CX-1vTfzKodImx9WpUGqy_K-N7akrsWrbX42PHMmZKq2jW-jBjrprkk43JR4ysLN3pxbaMx2PdMcEdxOACk/pub	February 28, 2021

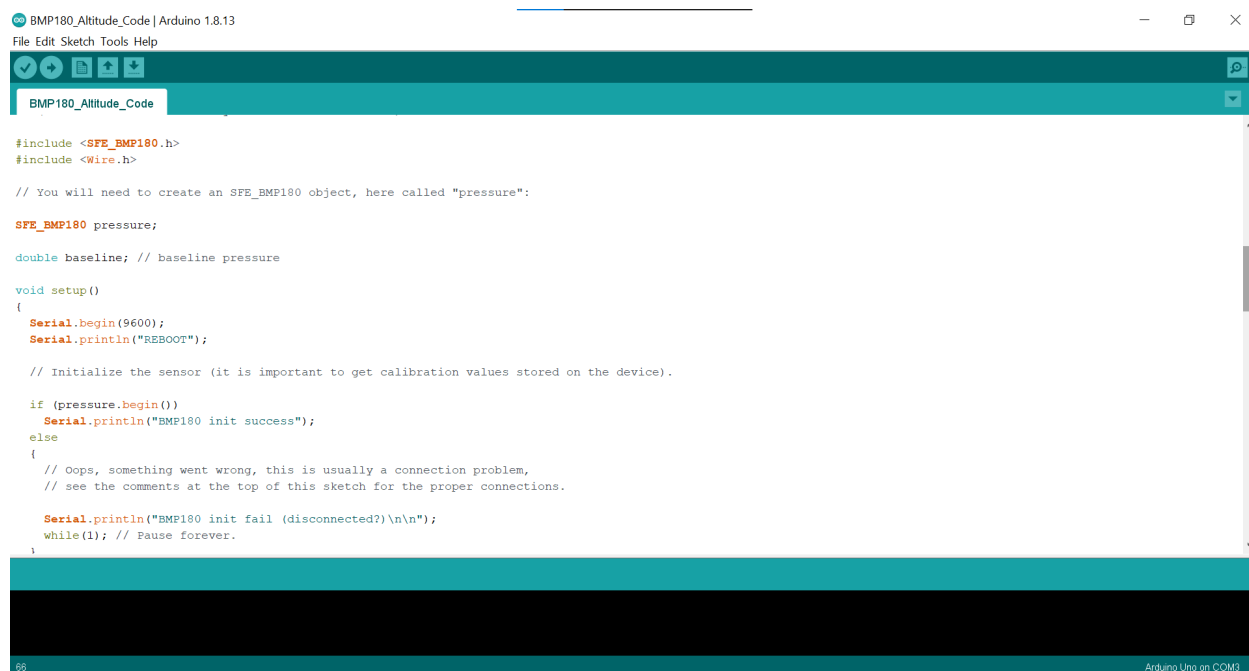
Bill of Materials	https://docs.google.com/spreadsheets/d/e/2PACX-1vQlWKNEOJDR8rWDGJnl6WPF171UQkRN4w0v8_0I1PWJMiRuuW_CjZOM50-CcWvQZ0Ao1logRJYRnA3/pubhtml	February 28, 2021 (Updated Throughout)
Deliverable F-Prototype I and Customer Feedback	https://docs.google.com/document/d/e/2PACX-1vQFnEFIXYRHJccINZ-O6FCBWuvRW42a9FjM4onCHzMaCb30FLwqZ52s9Dxp5L2LF4tWk0Ho8D9QfZTg/pub	March 9, 2021
Deliverable G-Prototype II and Customer Feedback	https://docs.google.com/document/d/e/2PACX-1vQFnEFIXYRHJccINZ-O6FCBWuvRW42a9FjM4onCHzMaCb30FLwqZ52s9Dxp5L2LF4tWk0Ho8D9QfZTg/pub	March 14, 2021
Deliverable H-Prototype III and Customer Feedback	https://docs.google.com/document/d/e/2PACX-1vTjPmsvXzO8je-lSqYKrKuOUL9dQuGLEu1EW7JRirMqP6Lao7zux3gNu4_9dHS62yBYwq_a4auu4lod/pub	March 28, 2021
Deliverable I-Design Day Presentation	https://docs.google.com/presentation/d/19xfu4h9NrIkOT11abxEmGEIQiuN93Ar74unvNw23bU/edit?usp=sharing	April 8th, 2021

Deliverable J-Final Presentation	https://docs.google.com/presentation/d/e/2PACX-1vSSeBUgqp4IwiRk1V5-w48OX3sYLxvqophGFsJ7UAnTxRfy_FFZrxqH7qw1N_-LNHgvWh5o5zg5cp7w/pub?start=false&loop=false&delayms=3000	March 31, 2021
----------------------------------	---	----------------

10 Other Appendices

10.1 Appendix II

Code for the Altitude Subsystem



The screenshot shows the Arduino IDE interface with the file "BMP180_Altitude_Code" open. The code is for an Arduino Uno on COM3, using Arduino 1.8.13. The code includes the SFE_BMP180.h library and the Wire.h library. It defines a pressure object of type SFE_BMP180 and a baseline pressure variable of type double. The setup function initializes the serial port at 9600 baud, prints "REBOOT", and initializes the BMP180 sensor. It then checks if the sensor initialization was successful. If successful, it prints "BMP180 init success". If not, it prints "BMP180 init fail (disconnected?)\n\n" and enters an infinite loop to pause forever.

```
BMP180_Altitude_Code | Arduino 1.8.13
File Edit Sketch Tools Help

#include <SFE_BMP180.h>
#include <Wire.h>

// You will need to create an SFE_BMP180 object, here called "pressure":

SFE_BMP180 pressure;

double baseline; // baseline pressure

void setup()
{
  Serial.begin(9600);
  Serial.println("REBOOT");

  // Initialize the sensor (it is important to get calibration values stored on the device).

  if (pressure.begin())
  {
    Serial.println("BMP180 init success");
  }
  else
  {
    // Oops, something went wrong, this is usually a connection problem,
    // see the comments at the top of this sketch for the proper connections.

    Serial.println("BMP180 init fail (disconnected?)\n\n");
    while(1); // Pause forever.
  }
}
```

86 Arduino Uno on COM3

```

BMP180_Altitude_Code | Arduino 1.8.13
File Edit Sketch Tools Help

BMP180_Altitude_Code
// see the comments at the top of this sketch for the proper connections.

Serial.println("BMP180 init fail (disconnected?)\n\n");
while(1); // Pause forever.
}

// Get the baseline pressure:

baseline = getPressure();

Serial.print("baseline pressure: ");
Serial.print(baseline);
Serial.println(" mb");
}

void loop()
{
  double a,P;

  // Get a new pressure reading:

  P = getPressure();

  // Show the relative altitude difference between
  // the new reading and the baseline reading:

  a = pressure.altitude(P,baseline);

```

```

BMP180_Altitude_Code | Arduino 1.8.13
File Edit Sketch Tools Help

BMP180_Altitude_Code
// Get a new pressure reading.

P = getPressure();

// Show the relative altitude difference between
// the new reading and the baseline reading:

a = pressure.altitude(P,baseline);

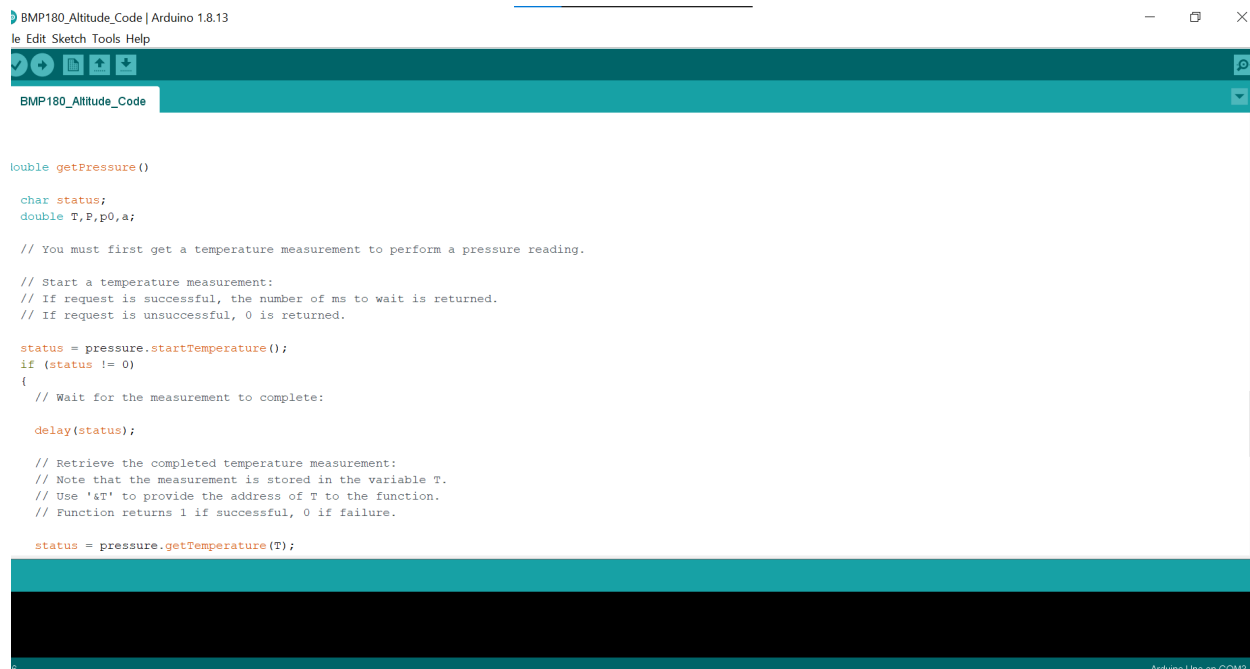
Serial.print("relative altitude: ");
if (a >= 0.0) Serial.print(" "); // add a space for positive numbers
Serial.print(a,1);
Serial.print(" meters, ");
if (a >= 0.0) Serial.print(" "); // add a space for positive numbers
Serial.print(a*3.28084,0);
Serial.println(" feet");

delay(500);
}

double getPressure()
{
  char status;
  double T,P,p0,a;

  // You must first get a temperature measurement to perform a pressure reading.

```



BMP180_Altitude_Code | Arduino 1.8.13

File Edit Sketch Tools Help

BMP180_Altitude_Code

```
double getPressure()

char status;
double T,P,p0,a;

// You must first get a temperature measurement to perform a pressure reading.

// Start a temperature measurement:
// If request is successful, the number of ms to wait is returned.
// If request is unsuccessful, 0 is returned.

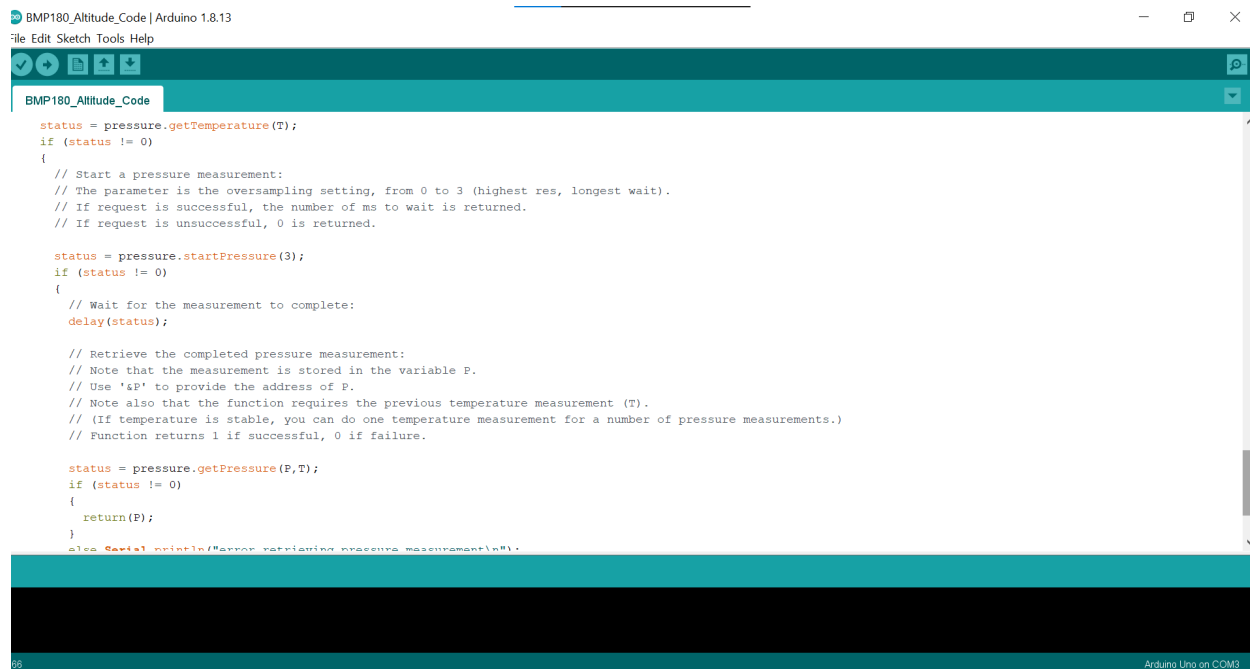
status = pressure.startTemperature();
if (status != 0)
{
    // Wait for the measurement to complete:

    delay(status);

    // Retrieve the completed temperature measurement:
    // Note that the measurement is stored in the variable T.
    // Use '&T' to provide the address of T to the function.
    // Function returns 1 if successful, 0 if failure.

    status = pressure.getTemperature(T);
```

6 Arduino Uno on COM3



BMP180_Altitude_Code | Arduino 1.8.13

File Edit Sketch Tools Help

BMP180_Altitude_Code

```
    status = pressure.getTemperature(T);
    if (status != 0)
    {
        // Start a pressure measurement:
        // The parameter is the oversampling setting, from 0 to 3 (highest res, longest wait).
        // If request is successful, the number of ms to wait is returned.
        // If request is unsuccessful, 0 is returned.

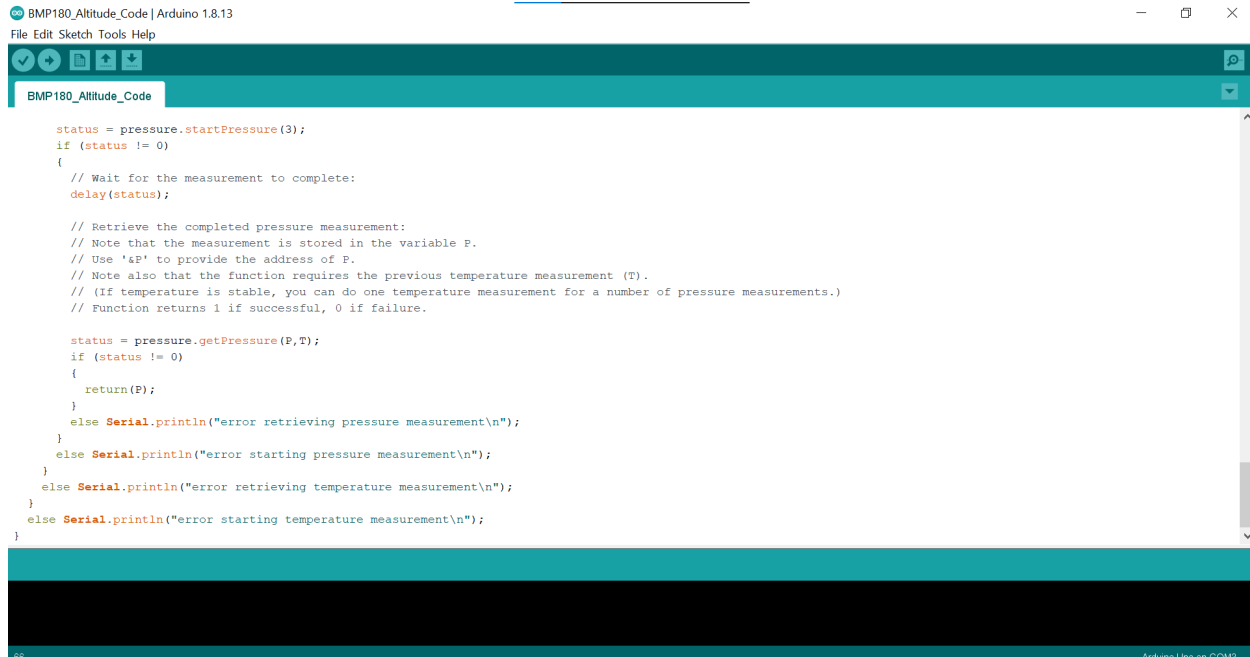
        status = pressure.startPressure(3);
        if (status != 0)
        {
            // Wait for the measurement to complete:

            delay(status);

            // Retrieve the completed pressure measurement:
            // Note that the measurement is stored in the variable P.
            // Use '&P' to provide the address of P.
            // Note also that the function requires the previous temperature measurement (T).
            // (If temperature is stable, you can do one temperature measurement for a number of pressure measurements.)
            // Function returns 1 if successful, 0 if failure.

            status = pressure.getPressure(P,T);
            if (status != 0)
            {
                return(P);
            }
            else Serial.println("Error retrieving pressure measurement");
```

66 Arduino Uno on COM3



```

BMP180_Altitude_Code | Arduino 1.8.13
File Edit Sketch Tools Help

status = pressure.startPressure(3);
if (status != 0)
{
  // Wait for the measurement to complete:
  delay(status);

  // Retrieve the completed pressure measurement:
  // Note that the measurement is stored in the variable P.
  // Use 'sP' to provide the address of P.
  // Note also that the function requires the previous temperature measurement (T).
  // (If temperature is stable, you can do one temperature measurement for a number of pressure measurements.)
  // Function returns 1 if successful, 0 if failure.

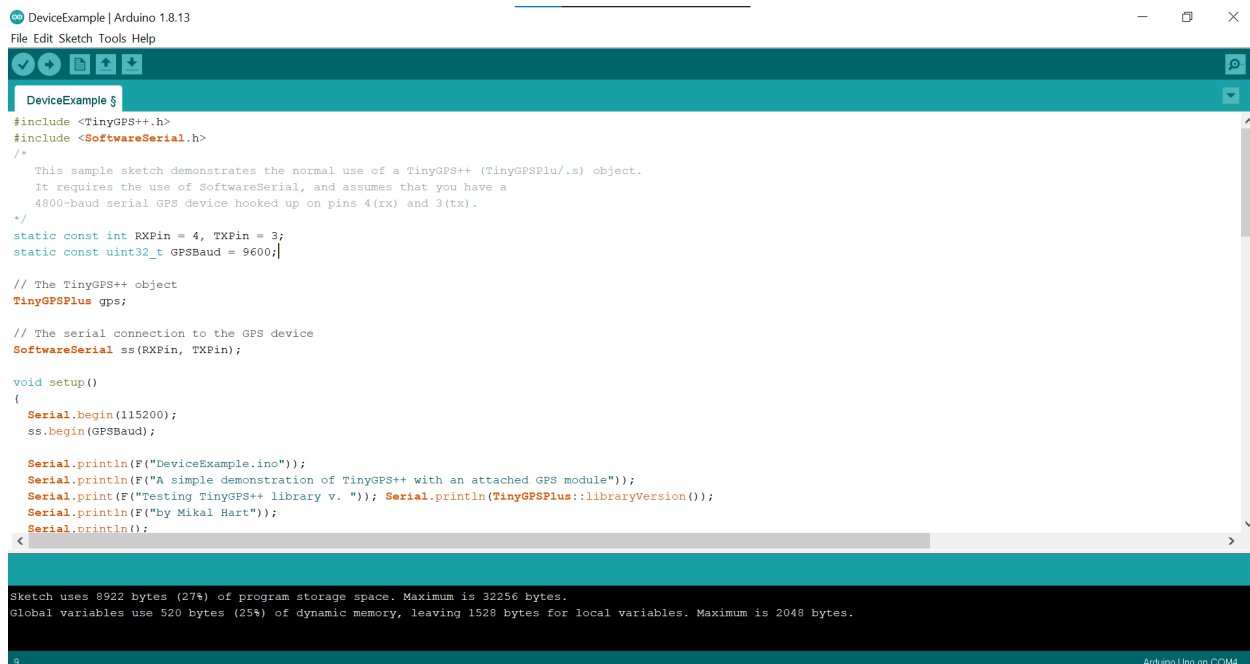
  status = pressure.getPressure(P,T);
  if (status != 0)
  {
    return(P);
  }
  else Serial.println("error retrieving pressure measurement\n");
}
else Serial.println("error starting pressure measurement\n");
}
else Serial.println("error retrieving temperature measurement\n");
}
else Serial.println("error starting temperature measurement\n");
}
}

```

The figures above show the code used to print the altitude data to the serial monitor. The code was retrieved from [this website](#).

10.2 Appendix III

Code for the Location Subsystem



```

DeviceExample | Arduino 1.8.13
File Edit Sketch Tools Help

#include <TinyGPS++.h>
#include <SoftwareSerial.h>
/*
  This sample sketch demonstrates the normal use of a TinyGPS++ (TinyGPSPlus) object.
  It requires the use of SoftwareSerial, and assumes that you have a
  4800-baud serial GPS device hooked up on pins 4 (rx) and 3 (tx).
*/
static const int RXPin = 4, TXPin = 3;
static const uint32_t GPSBaud = 9600;

// The TinyGPS++ object
TinyGPSPlus gps;

// The serial connection to the GPS device
SoftwareSerial ss(RXPin, TXPin);

void setup()
{
  Serial.begin(115200);
  ss.begin(GPSBaud);

  Serial.println(F("DeviceExample.ino"));
  Serial.println(F("A simple demonstration of TinyGPS++ with an attached GPS module"));
  Serial.print(F("Testing TinyGPS++ library v. ")); Serial.println(TinyGPSPlus::libraryVersion());
  Serial.println(F("by Mikal Hart"));
  Serial.println();
}

Sketch uses 8922 bytes (27%) of program storage space. Maximum is 32256 bytes.
Global variables use 520 bytes (25%) of dynamic memory, leaving 1528 bytes for local variables. Maximum is 2048 bytes.

```


DeviceExample | Arduino 1.8.13

File Edit Sketch Tools Help

DeviceExample \$

```

void setup()
{
  Serial.begin(115200);
  ss.begin(GPSBaud);

  Serial.println(F("DeviceExample.ino"));
  Serial.println(F("A simple demonstration of TinyGPS++ with an attached GPS module"));
  Serial.print(F("Testing TinyGPS++ library v. ")); Serial.println(TinyGPSPlus::libraryVersion());
  Serial.println(F("by Mikal Hart"));
  Serial.println();
}

void loop()
{
  // This sketch displays information every time a new sentence is correctly encoded.
  while (ss.available() > 0)
    if (gps.encode(ss.read()))
      displayInfo();

  if (millis() > 5000 && gps.charsProcessed() < 10)
  {
    Serial.println(F("No GPS detected: check wiring."));
    while(true);
  }
}

```

Sketch uses 8922 bytes (27%) of program storage space. Maximum is 32256 bytes.
Global variables use 520 bytes (25%) of dynamic memory, leaving 1528 bytes for local variables. Maximum is 2048 bytes.

Arduino Uno on COM4

DeviceExample | Arduino 1.8.13

File Edit Sketch Tools Help

DeviceExample \$

```

}

void displayInfo()
{
  Serial.print(F("Location: "));
  if (gps.location.isValid())
  {
    Serial.print(gps.location.lat(), 6);
    Serial.print(F(", "));
    Serial.print(gps.location.lng(), 6);
  }
  else
  {
    Serial.print(F("INVALID"));
  }

  Serial.print(F("  Date/Time: "));
  if (gps.date.isValid())
  {
    Serial.print(gps.date.month());
    Serial.print(F("/"));
    Serial.print(gps.date.day());
    Serial.print(F("/"));
    Serial.print(gps.date.year());
  }
  else

```

Sketch uses 8922 bytes (27%) of program storage space. Maximum is 32256 bytes.
Global variables use 520 bytes (25%) of dynamic memory, leaving 1528 bytes for local variables. Maximum is 2048 bytes.

Arduino Uno on COM4

```

DeviceExample | Arduino 1.8.13
File Edit Sketch Tools Help

DeviceExample $
{
  Serial.print(F("INVALID"));
}

Serial.print(F(" "));
if (gps.time.isValid())
{
  if (gps.time.hour() < 10) Serial.print(F("0"));
  Serial.print(gps.time.hour());
  Serial.print(F(":"));
  if (gps.time.minute() < 10) Serial.print(F("0"));
  Serial.print(gps.time.minute());
  Serial.print(F(":"));
  if (gps.time.second() < 10) Serial.print(F("0"));
  Serial.print(gps.time.second());
  Serial.print(F("."));
  if (gps.time.centisecond() < 10) Serial.print(F("0"));
  Serial.print(gps.time.centisecond());
}
else
{
  Serial.print(F("INVALID"));
}

Serial.println();
}

Sketch uses 8922 bytes (27%) of program storage space. Maximum is 32256 bytes.
Global variables use 520 bytes (25%) of dynamic memory, leaving 1528 bytes for local variables. Maximum is 2048 bytes.
Arduino Uno on COM4

```

```

DeviceExample | Arduino 1.8.13
File Edit Sketch Tools Help

DeviceExample $
{
  Serial.print(F("INVALID"));
}

Serial.print(F(" "));
if (gps.time.isValid())
{
  if (gps.time.hour() < 10) Serial.print(F("0"));
  Serial.print(gps.time.hour());
  Serial.print(F(":"));
  if (gps.time.minute() < 10) Serial.print(F("0"));
  Serial.print(gps.time.minute());
  Serial.print(F(":"));
  if (gps.time.second() < 10) Serial.print(F("0"));
  Serial.print(gps.time.second());
  Serial.print(F("."));
  if (gps.time.centisecond() < 10) Serial.print(F("0"));
  Serial.print(gps.time.centisecond());
}
else
{
  Serial.print(F("INVALID"));
}

Serial.println();
}

Sketch uses 8922 bytes (27%) of program storage space. Maximum is 32256 bytes.
Global variables use 520 bytes (25%) of dynamic memory, leaving 1528 bytes for local variables. Maximum is 2048 bytes.
Arduino Uno on COM4

```

The code was found in the TinyGPS++ library. This library can be downloaded from [here](#).

10.3 Appendix IV

Code for the Voice Subsystem

```
#include "Talkie.h"
#include "Vocab_US_Large.h"
#include "Vocab_Special.h"

Talkie voice;

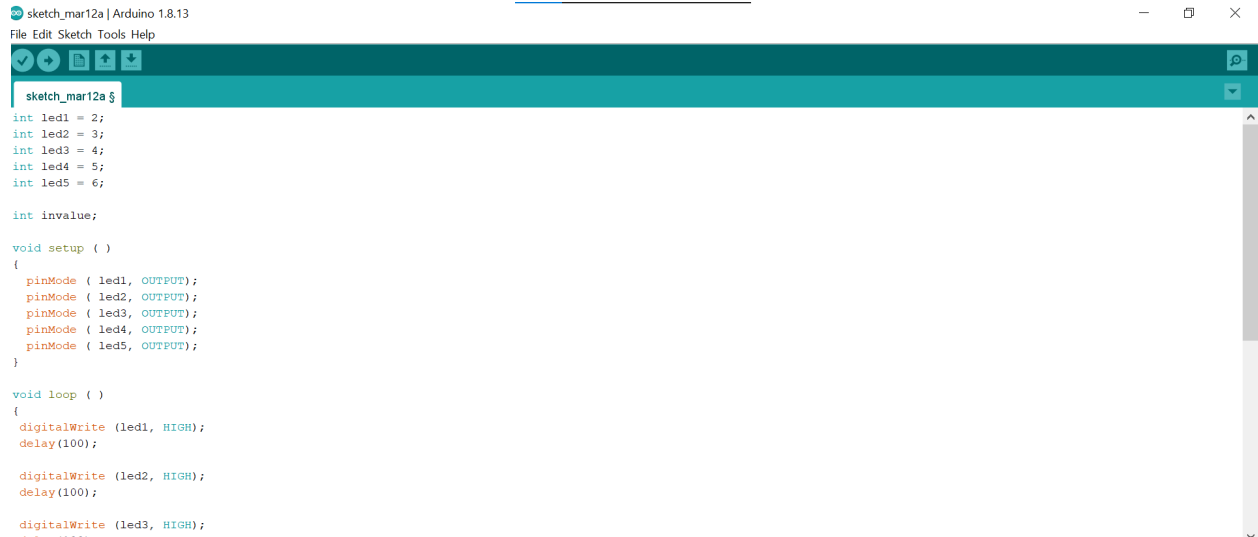
void setup() {
}

void loop() {
    voice.say(spPAUSE2);
    voice.say(sp2_DANGER);
    voice.say(sp2_DANGER);
    voice.say(sp2_MOVE);
    voice.say(sp2_OPERATOR);
    voice.say(sp2_IS);
    voice.say(sp2_ON);
    voice.say(sp2_ALERT);
}
```

The code was found in the Talkie library. This library can be downloaded from [here](#).

10.4 Appendix V

Code for the Light Subsystem



```
sketch_mar12a | Arduino 1.8.13
File Edit Sketch Tools Help

sketch_mar12a $

int led1 = 2;
int led2 = 3;
int led3 = 4;
int led4 = 5;
int led5 = 6;

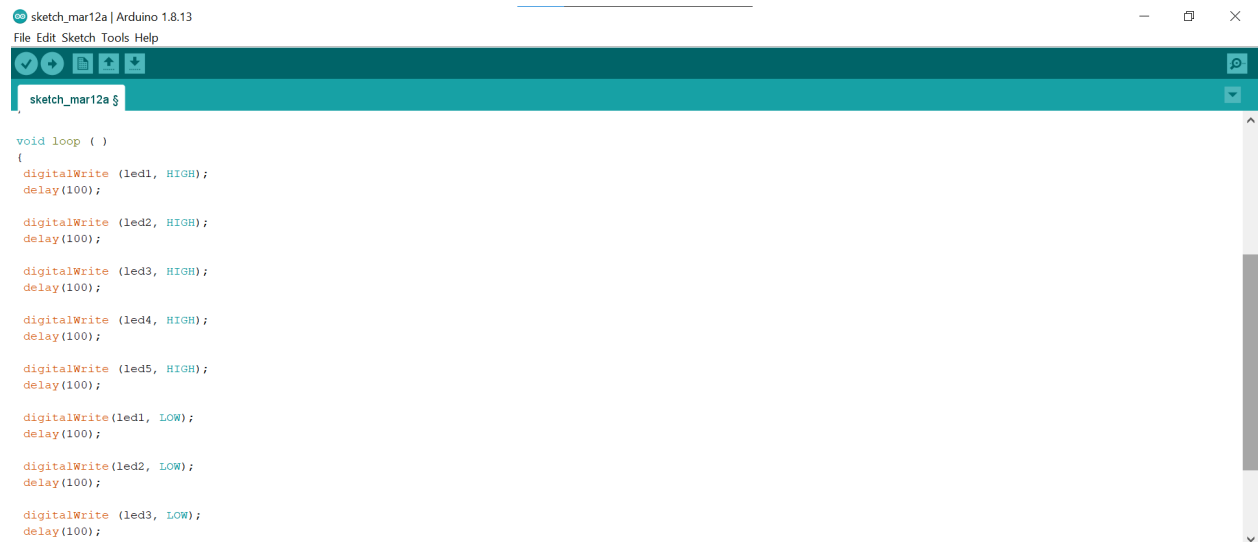
int invalue;

void setup ( )
{
  pinMode ( led1, OUTPUT);
  pinMode ( led2, OUTPUT);
  pinMode ( led3, OUTPUT);
  pinMode ( led4, OUTPUT);
  pinMode ( led5, OUTPUT);
}

void loop ( )
{
  digitalWrite (led1, HIGH);
  delay(100);

  digitalWrite (led2, HIGH);
  delay(100);

  digitalWrite (led3, HIGH);
  ...
```



```
sketch_mar12a | Arduino 1.8.13
File Edit Sketch Tools Help

sketch_mar12a $

void loop ( )
{
  digitalWrite (led1, HIGH);
  delay(100);

  digitalWrite (led2, HIGH);
  delay(100);

  digitalWrite (led3, HIGH);
  delay(100);

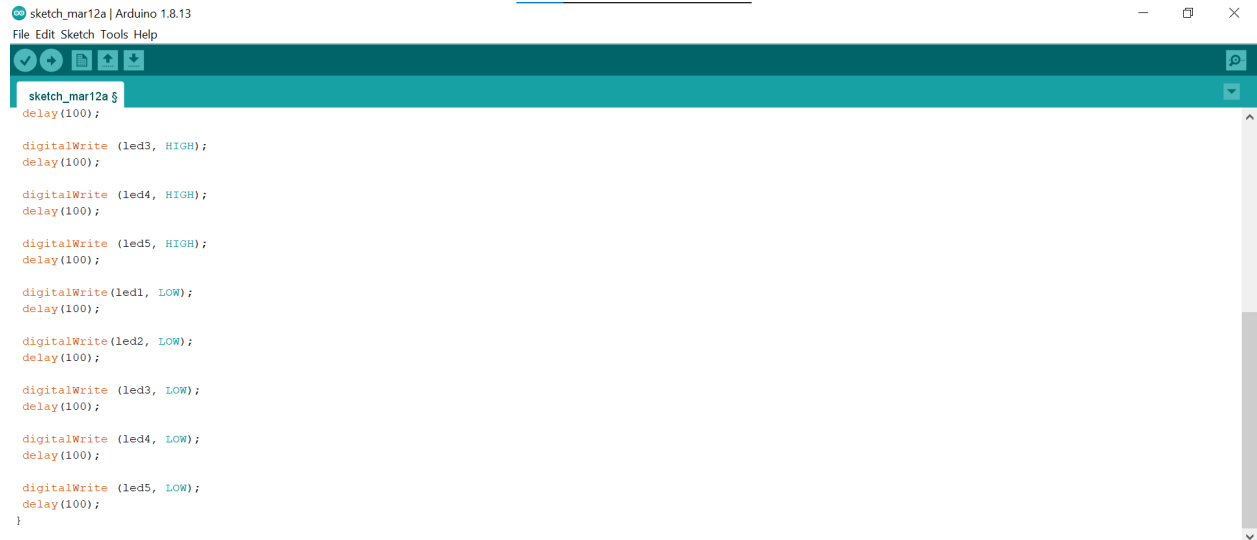
  digitalWrite (led4, HIGH);
  delay(100);

  digitalWrite (led5, HIGH);
  delay(100);

  digitalWrite (led1, LOW);
  delay(100);

  digitalWrite (led2, LOW);
  delay(100);

  digitalWrite (led3, LOW);
  delay(100);
}
```

```

sketch_mar12a | Arduino 1.8.13
File Edit Sketch Tools Help

sketch_mar12a $
delay(100);

digitalWrite (led3, HIGH);
delay(100);

digitalWrite (led4, HIGH);
delay(100);

digitalWrite (led5, HIGH);
delay(100);

digitalWrite (led1, LOW);
delay(100);

digitalWrite (led2, LOW);
delay(100);

digitalWrite (led3, LOW);
delay(100);

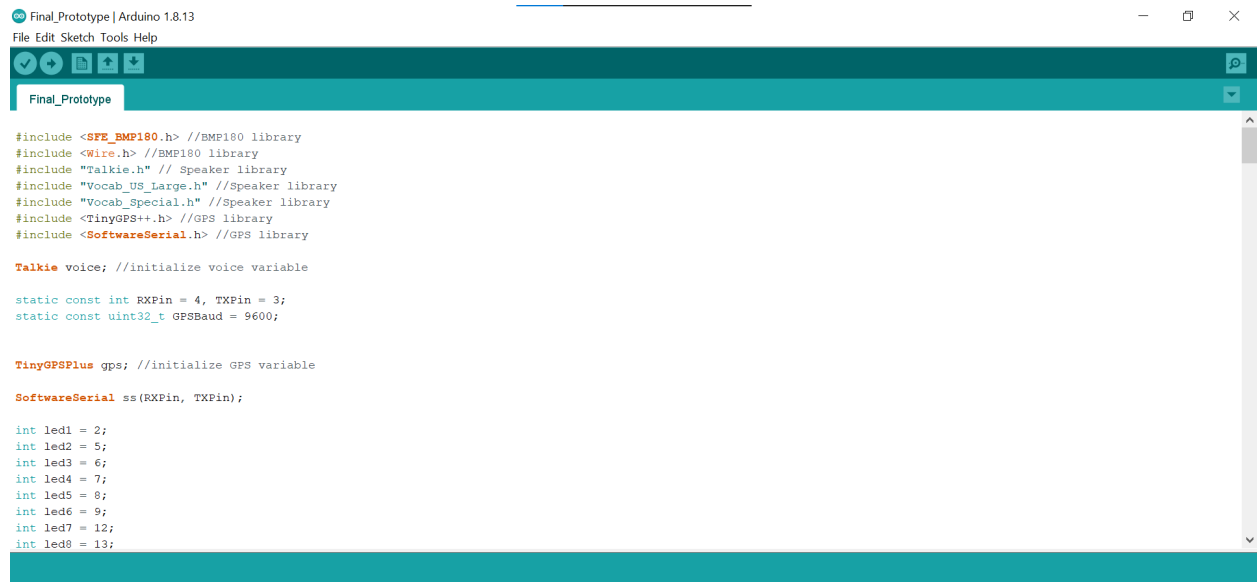
digitalWrite (led4, LOW);
delay(100);

digitalWrite (led5, LOW);
delay(100);
}

```

10.5 Appendix VI

Code for Final Prototype



```

Final_Prototype | Arduino 1.8.13
File Edit Sketch Tools Help

Final_Prototype

#include <SFE_BMP180.h> //BMP180 library
#include <Wire.h> //BMP180 library
#include "Talkie.h" // Speaker library
#include "Vocab_US_Large.h" //Speaker library
#include "Vocab_Special.h" //Speaker library
#include <TinyGPS++.h> //GPS library
#include <SoftwareSerial.h> //GPS library

Talkie voice; //initialize voice variable

static const int RXPin = 4, TXPin = 3;
static const uint32_t GPSBaud = 9600;

TinyGPSPlus gps; //initialize GPS variable

SoftwareSerial ss(RXPin, TXPin);

int led1 = 2;
int led2 = 5;
int led3 = 6;
int led4 = 7;
int led5 = 8;
int led6 = 9;
int led7 = 12;
int led8 = 13;

```

Final_Prototype | Arduino 1.8.13
File Edit Sketch Tools Help

```

Final_Prototype
int led7 = 12;
int led8 = 13;

int threshold_failed = 1;

int intValue;

SFE_BMP180 pressure;

double baseline;

void setup() {
  Serial.begin(9600);
  Serial.println("REBOOT");

  if (pressure.begin())
    Serial.println("BMP180 init success");
  else
  {
    Serial.println("BMP180 init fail (disconnected?)\n\n");
    while(1);
  }

  baseline = getPressure();

  Serial.print("baseline pressure: ");

```

Final_Prototype | Arduino 1.8.13
File Edit Sketch Tools Help

```

Final_Prototype
Serial.print("baseline pressure: ");
Serial.print(baseline);
Serial.println(" mb");

{
  ss.begin(GPSBaud);

  Serial.println(F("DeviceExample.ino"));
  Serial.println(F("A simple demonstration of TinyGPS++ with an attached GPS module"));
  Serial.print(F("Testing TinyGPS++ library v. ")); Serial.println(TinyGPSPlus::libraryVersion());
  Serial.println(F("by Mikal Hart"));
  Serial.println();
}

{
  pinMode (led1, OUTPUT);
  pinMode (led2, OUTPUT);
  pinMode (led3, OUTPUT);
  pinMode (led4, OUTPUT);
  pinMode (led5, OUTPUT);
  pinMode (led6, OUTPUT);
  pinMode (led7, OUTPUT);
  pinMode (led8, OUTPUT);
}

```

```

Final_Prototype | Arduino 1.8.13
File Edit Sketch Tools Help

Final_Prototype

}
void displayInfo() //location function
{
  Serial.print(F("Location: "));
  if (gps.location.isValid())
  {
    Serial.print(gps.location.lat(), 6);
    Serial.print(F(", "));
    Serial.print(gps.location.lng(), 6);
  }
  else
  {
    Serial.print(F("INVALID"));
  }

  Serial.print(F(" Date/Time: "));
  if (gps.date.isValid())
  {
    Serial.print(gps.date.month());
    Serial.print(F("/"));
    Serial.print(gps.date.day());
    Serial.print(F("/"));
    Serial.print(gps.date.year());
  }
  else
  {

```

```

Final_Prototype | Arduino 1.8.13
File Edit Sketch Tools Help

Final_Prototype
else
{
  Serial.print(F("INVALID\n"));
}

Serial.print(F(" "));
if (gps.time.isValid())
{
  if (gps.time.hour() < 10) Serial.print(F("0"));
  Serial.print(gps.time.hour());
  Serial.print(F(":"));
  if (gps.time.minute() < 10) Serial.print(F("0"));
  Serial.print(gps.time.minute());
  Serial.print(F(":"));
  if (gps.time.second() < 10) Serial.print(F("0"));
  Serial.print(gps.time.second());
  Serial.print(F("."));
  if (gps.time.centisecond() < 10) Serial.print(F("0"));
  Serial.print(gps.time.centisecond());
}
else
{
  Serial.print(F("INVALID"));
}

Serial.println();
...

```

Final_Prototype | Arduino 1.8.13

File Edit Sketch Tools Help

Final_Prototype

```

void loop() {

{
  Serial.print(F("Location: "));
  if (gps.location.isValid())
  {
    Serial.print(gps.location.lat(), 6);
    Serial.print(F(",\n"));
    Serial.print(gps.location.lng(), 6);
  }
  else
  {
    Serial.print(F("INVALID"));
  }

  Serial.print(F("  Date/Time: "));
  if (gps.date.isValid())
  {
    Serial.print(gps.date.month());
    Serial.print(F("/"));
    Serial.print(gps.date.day());
    Serial.print(F("/"));
    Serial.print(gps.date.year());
  }
  else
  {
    Serial.print(F("INVALID"));
  }
}
}

```

Final_Prototype | Arduino 1.8.13

File Edit Sketch Tools Help

Final_Prototype

```

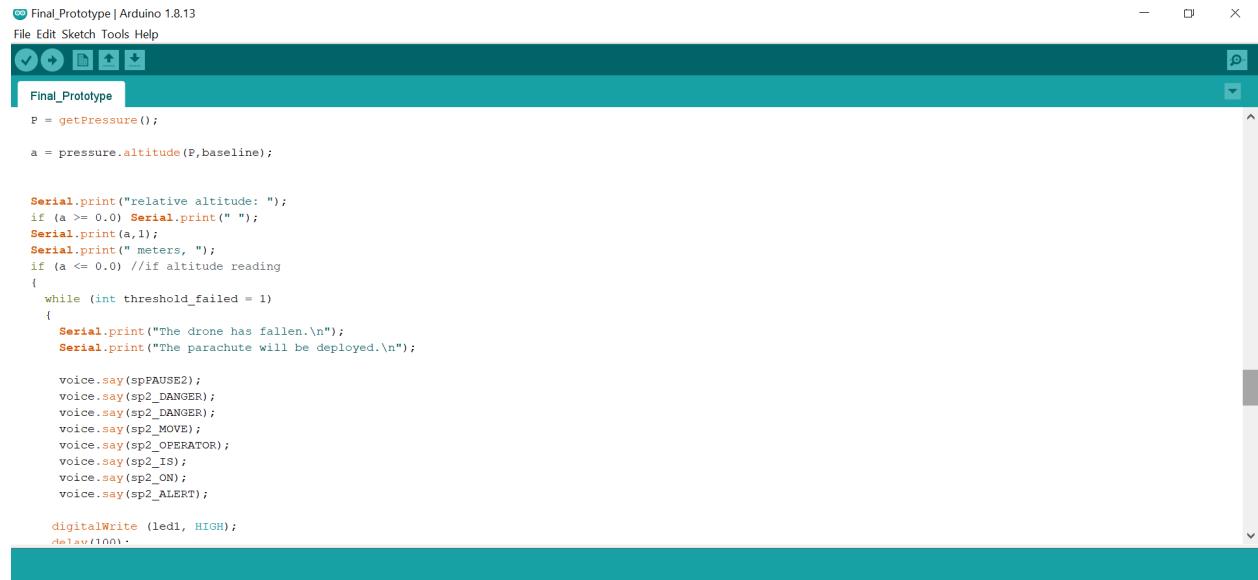
  Serial.print(F("INVALID"));
}

Serial.print(F(" "));
if (gps.time.isValid())
{
  if (gps.time.hour() < 10) Serial.print(F("0"));
  Serial.print(gps.time.hour());
  Serial.print(F(":"));
  if (gps.time.minute() < 10) Serial.print(F("0"));
  Serial.print(gps.time.minute());
  Serial.print(F(":"));
  if (gps.time.second() < 10) Serial.print(F("0"));
  Serial.print(gps.time.second());
  Serial.print(F("."));
  if (gps.time.centisecond() < 10) Serial.print(F("0"));
  Serial.print(gps.time.centisecond());
}
else
{
  Serial.print(F("INVALID"));
}

Serial.println();
}

double a,P;

```

The screenshot shows the Arduino IDE interface with a sketch titled "Final_Prototype". The code is written in C++ and implements a drone altitude monitoring system. It uses the `getPressure()` function to read pressure and `pressure.altitude()` to calculate altitude. The code includes a `Serial.print()` statement to output the relative altitude. It also features a `while` loop that triggers a series of voice commands (using `voice.say()`) and a `digitalWrite()` command (using `led1`) when the altitude falls below a threshold. The code is as follows:

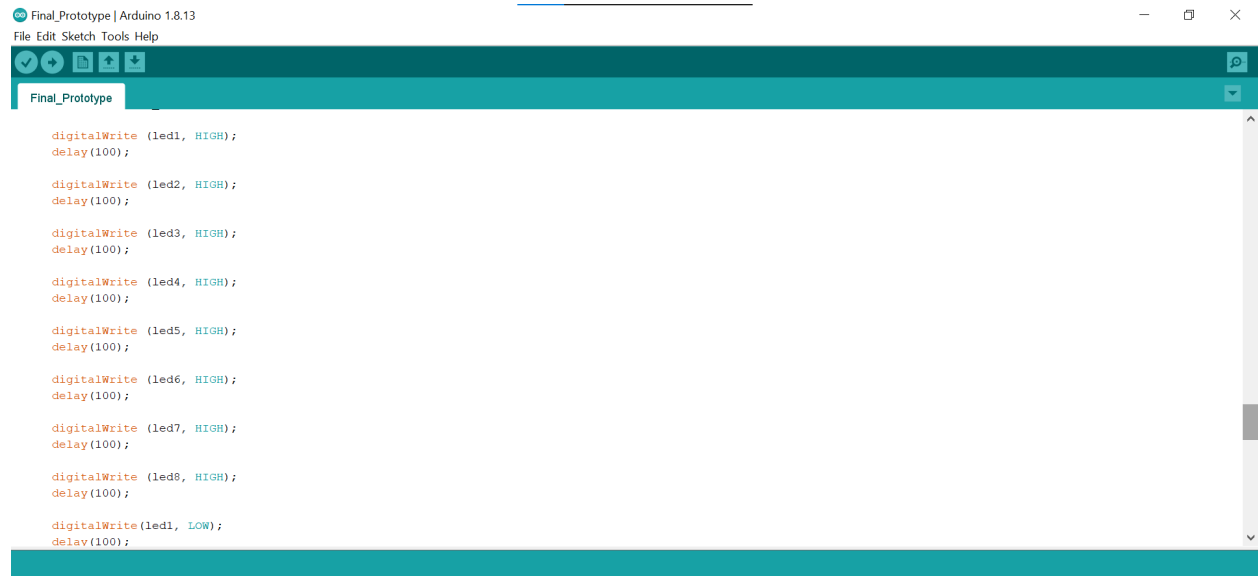
```
P = getPressure();

a = pressure.altitude(P,baseline);

Serial.print("relative altitude: ");
if (a >= 0.0) Serial.print(" ");
Serial.print(a,1);
Serial.print(" meters, ");
if (a <= 0.0) //if altitude reading
{
  while (int threshold_failed = 1)
  {
    Serial.print("The drone has fallen.\n");
    Serial.print("The parachute will be deployed.\n");

    voice.say(spPAUSE2);
    voice.say(sp2_DANGER);
    voice.say(sp2_DANGER);
    voice.say(sp2_MOVE);
    voice.say(sp2_OPERATOR);
    voice.say(sp2_IS);
    voice.say(sp2_ON);
    voice.say(sp2_ALERT);

    digitalWrite (led1, HIGH);
    delay(100);
```



The screenshot shows the Arduino IDE interface with a sketch titled "Final_Prototype". The code is written in C++ and implements a sequence of LED activations. It uses the `digitalWrite()` function to set the state of LEDs (led1 through led8) to HIGH or LOW, followed by a `delay(100);` statement. The code is as follows:

```
digitalWrite (led1, HIGH);
delay(100);

digitalWrite (led2, HIGH);
delay(100);

digitalWrite (led3, HIGH);
delay(100);

digitalWrite (led4, HIGH);
delay(100);

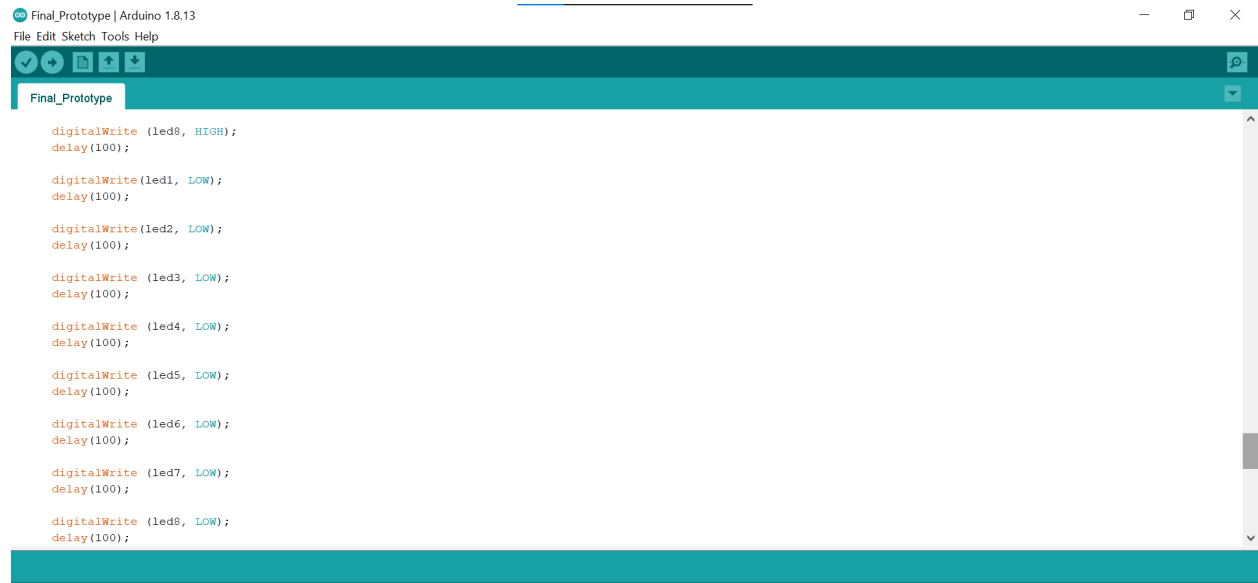
digitalWrite (led5, HIGH);
delay(100);

digitalWrite (led6, HIGH);
delay(100);

digitalWrite (led7, HIGH);
delay(100);

digitalWrite (led8, HIGH);
delay(100);

digitalWrite (led1, LOW);
delay(100);
```



Final_Prototype | Arduino 1.8.13

File Edit Sketch Tools Help

```
digitalWrite (led8, HIGH);
delay(100);

digitalWrite(led1, LOW);
delay(100);

digitalWrite(led2, LOW);
delay(100);

digitalWrite (led3, LOW);
delay(100);

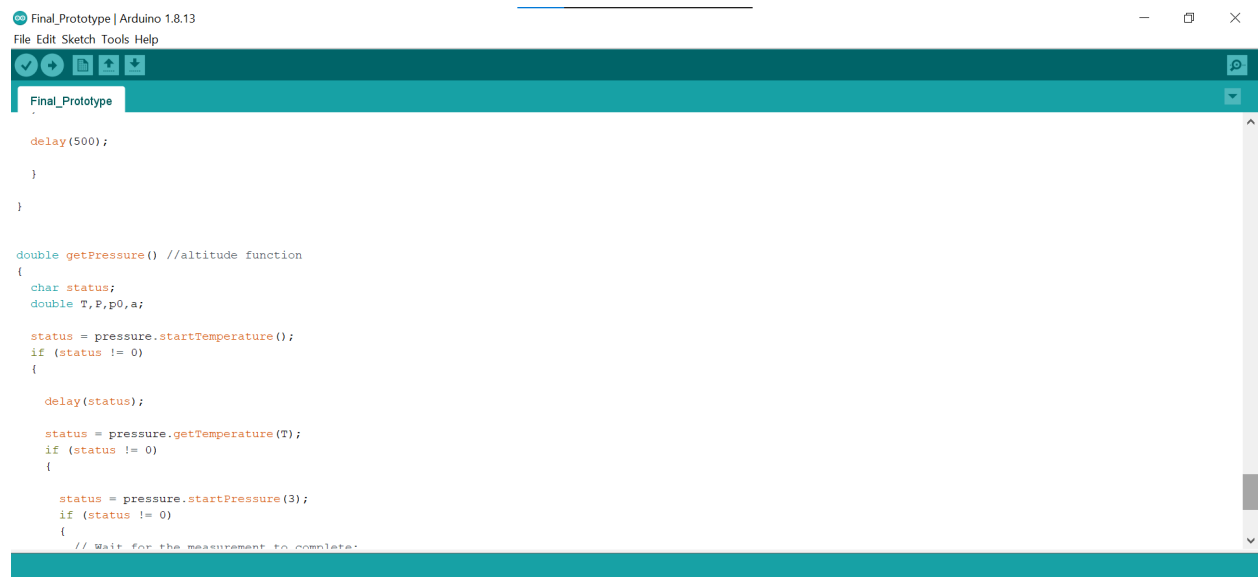
digitalWrite (led4, LOW);
delay(100);

digitalWrite (led5, LOW);
delay(100);

digitalWrite (led6, LOW);
delay(100);

digitalWrite (led7, LOW);
delay(100);

digitalWrite (led8, LOW);
delay(100);
```



Final_Prototype | Arduino 1.8.13

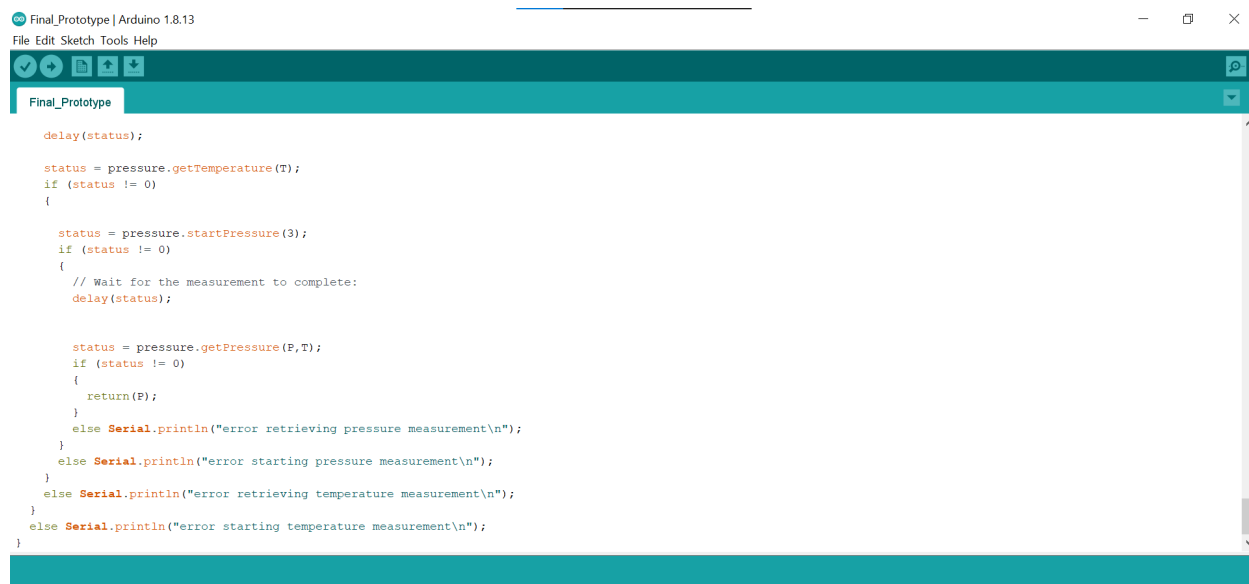
File Edit Sketch Tools Help

```
delay(500);
}
}

double getPressure() //altitude function
{
  char status;
  double T,P,p0,a;

  status = pressure.startTemperature();
  if (status != 0)
  {
    delay(status);

    status = pressure.getTemperature(T);
    if (status != 0)
    {
      status = pressure.startPressure(3);
      if (status != 0)
      {
        // Wait for the measurement to complete.
```



The screenshot shows the Arduino IDE interface with a sketch named "Final_Prototype". The code is written in C++ and uses the `pressure` library. It includes a `delay(status);` at the top. The main logic is enclosed in a large `if (status != 0)` block. Inside this block, there is a `status = pressure.startPressure(3);` followed by another `if (status != 0)` block. This inner block contains a comment `// Wait for the measurement to complete:` followed by `delay(status);`. After the inner block, there is a `status = pressure.getPressure(P,T);` followed by an `if (status != 0)` block that returns `P`. If the status is not 0, it prints an error message: `Serial.println("error retrieving pressure measurement\n");`. If the status is 0, it prints another error message: `Serial.println("error starting pressure measurement\n");`. The main `if` block also has an `else` clause that prints `Serial.println("error retrieving temperature measurement\n");`. Finally, there is an `else` clause for the main `if` block that prints `Serial.println("error starting temperature measurement\n");`. The code ends with a closing brace for the main `if` block.

```
delay(status);

status = pressure.getTemperature(T);
if (status != 0)
{

    status = pressure.startPressure(3);
    if (status != 0)
    {
        // Wait for the measurement to complete:
        delay(status);

        status = pressure.getPressure(P,T);
        if (status != 0)
        {
            return(P);
        }
        else Serial.println("error retrieving pressure measurement\n");
    }
    else Serial.println("error starting pressure measurement\n");
}
else Serial.println("error retrieving temperature measurement\n");
}
else Serial.println("error starting temperature measurement\n");
}
```

There is no source for this code because it was adjusted based on the codes for the previous independent subsystems.