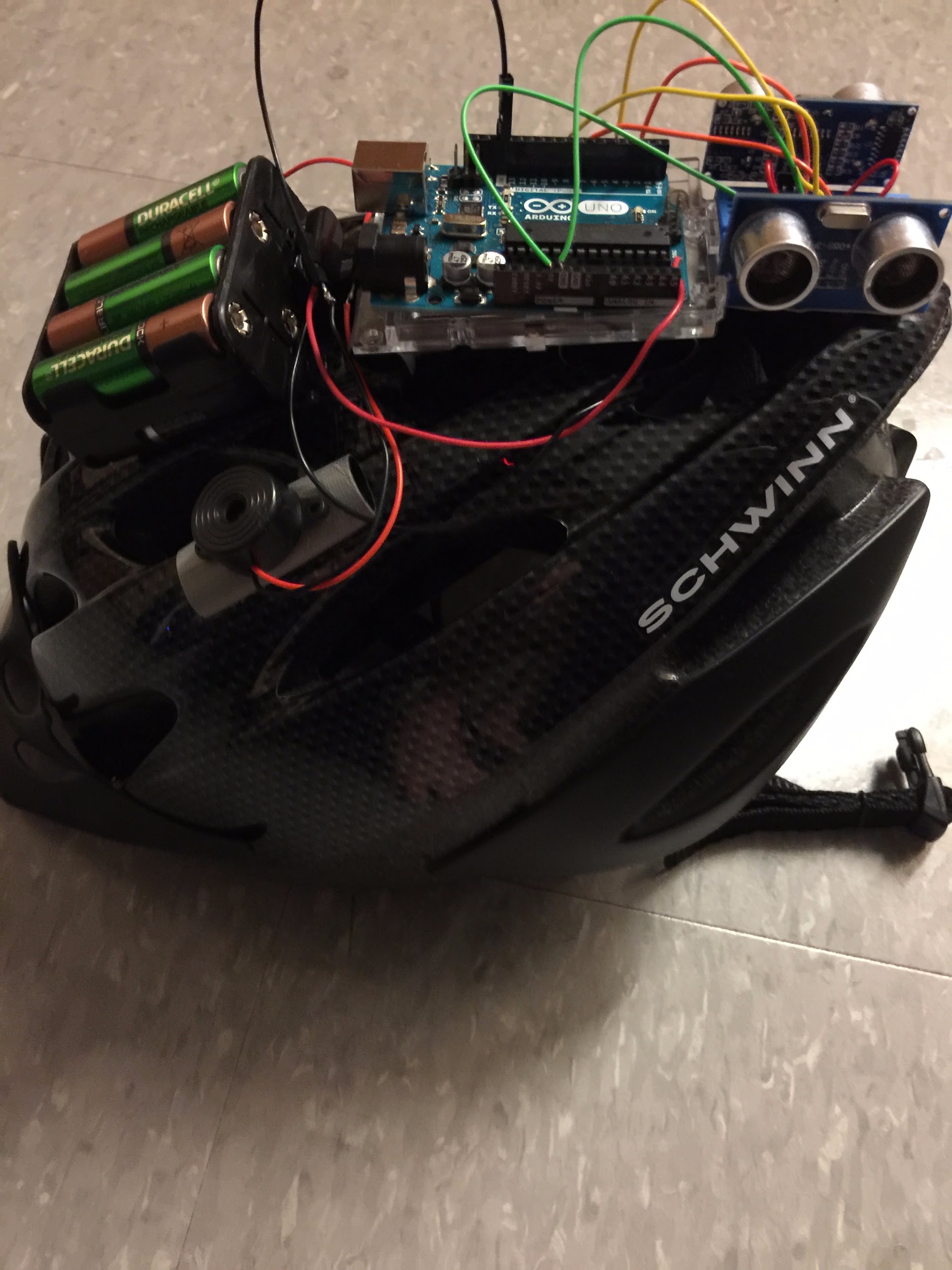
Elevate

System Manual

Version 1.0

Matthew Horger, Kevin Tayah, Jake Rauch

Table of Contents

Introduction…………………………………………………………………………… 3

The Helmet

Hardware ……………………………………………………………………… 5

Software ……………………………………………………………………… 6

Operations ……………………………………………………………………… 7

Data Summary ………………………………………………………………… 8

Future Improvements ……………………………………………………… 9

Mobile Application

Software ……………………………………………………………………… 11

User Interface ………………………………………………………………… 12

API Services …………………………………………………………………… 13

Database

Python Services ……………………………………………………………… 15

mySQL ………………………………………………………………………… 16

Plone CMS …………………………………………………………………… 17

Integration of Services …………………………………………………… 18

Google Maps ……………………………………………………………… 19

Accounts …………………………………………………………………… 20

Contact Information ………………………………………………………………… 21

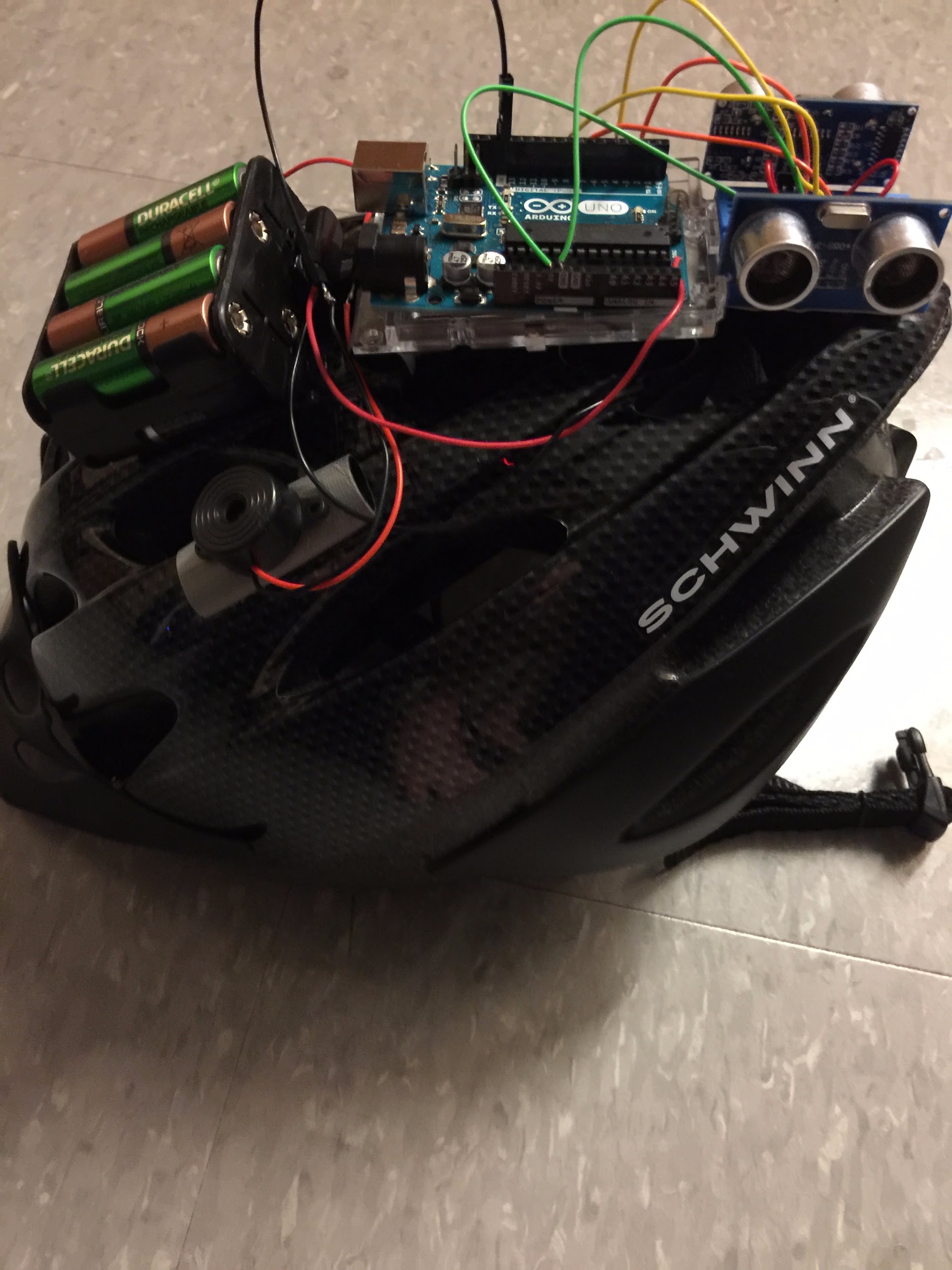
**Introduction**

Hello! Thank you for showing interest in Elevate and our operations. This document contains all the pertinent information you need to know about maintaining our system and what our services provide for our customers in terms of technical qualities and attributes. If you have any further questions, or have inquiries about improving our system, please contact our business team at contact@evelate.org.

**The Helmet**

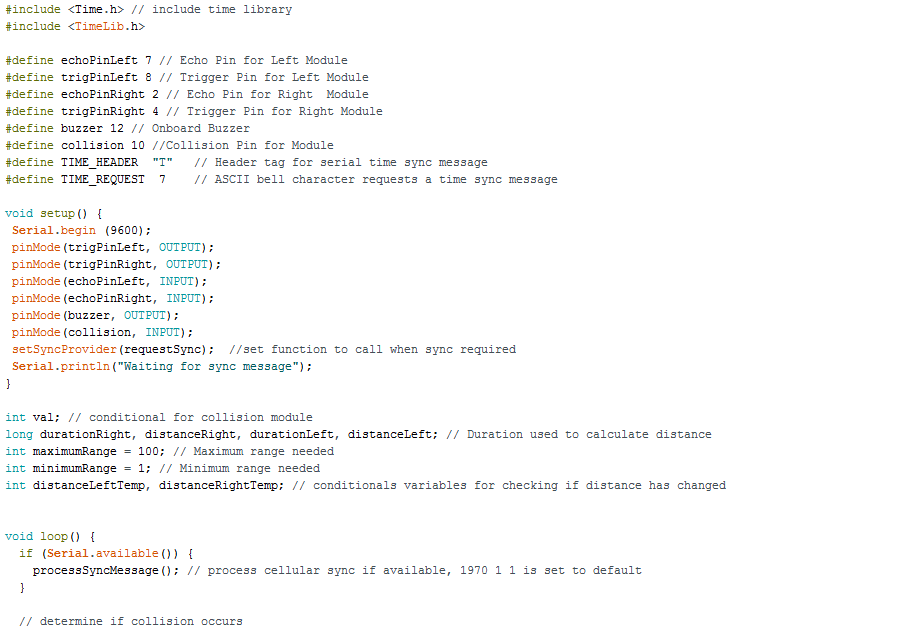
**Hardware**

Our helmet uses an Arduino Uno Rev V3 as the main controller for our functions. Connected to the Arduino is two HC-SR04 ultrasonic modules connected to a parallel circuit so that we can read in the trigger pins simultaneously for readings. Connected in the same circuit but with 5v connected to a I/O pin on the arduino is a 9v piezo buzzer (pin 12, since pin 13 is being using for TX/RX reset by default). This buzzer will get supplied 5v when an object is detected in the ultrasonic modules (more on this in software and operations). A collision sensor is also wired from the front of the helmet to the same parallel circuit except the trigger pin, like the buzzer, is hooked to a separate I/O pin on the Arduino (pin 10). The arduino is powered via 4 AA rechargeable batteries that can last up to 6 hours for a full charge or 4 AA non-rechargeable batteries that can last up to 9 hours for a full charge.



**Software**

The Arduino is powered via Arduino Markup Language. You can download the free editor from the arduino website or you can try out the new web editor that they released. We recommend you download the software. The helmet works by checking the trigger pins and collision pin during the void loop function and sending voltage to the buzzer if any electricity was read in by the digitalRead() function on each respective pin. If the distance hasn’t changed from the previous reading stored in the variable, don’t send voltage to the buzzer. If collision has occurred, timestamp the time into EPROM memory. You must include the time library for the timestamp to work.



**Operations**

In theory, the only operations are taking place in the void loop() once the setup initializes the serial console for debugging purposes and sets each pin type for respective input, output. The Arduino simply sends and checks voltages and operates code once a conditional has been fulfilled. The ultrasonic sensors use an echo pin to check for objects in the sensor range and translate that to voltage to be sent over the trigger pin. The collision sensor is simply a heavy-duty light switch to detect pressure and send voltage. The piezo buzzer simply receives voltage and turns the current into sound.

**Data Summary**

The only data associated with the Arduino is the serial console that is useful for debugging purposes. The time is set to the default value of January 1, 1970 unless it updates using synchronous calls to the pctime() or using GPS. We ideally would recommend GPS especially considering our future improvements. The distance readings are in centimeters using a conversion factor for the speed of light (distance / 58.2). If GPS was enabled, more data would include JSON coordinates and locational server data.

**Future Improvements**

We at Elevate would love to fit the helmet with GPRS and GPS capabilities in order to send direct JSON coordinates to our Python server associated with our database. We would also love to fit the helmet with temperature sensors to make sure the modules are not getting too hot. In order to fit the helmet with GPS and GPRS coordinate capacity, you must have a cell carrier and an available GPS satellite to connect to in order to receive data.

**Mobile Application**

**Software**

Elevate uses Javascript and AJAX calls to obtain our information for users and manipulate it. To view and maintain our website, simply download a text editor or use notepad provided by Windows. Everything is stored locally in a folder and for users to access it, it must be hosted by a provider or port forward on your machine. Our main code works using scripts calling a function callback upon completion of the call to obtain route mapping.

**User Interface**

Elevate’s user website and route mapping uses HTML 5 to display information and CSS for stylized for our organizational needs. To change any information, simply edit the style for your desired look. Please use w3schools if you need any help syntax wise. Our website uses a search bar that passes in a location into a function callback to obtain a map location. Our website also asks for current location from the browser to try and avoid having users enter their location. This uses scripting techniques found in the documentation of our code.

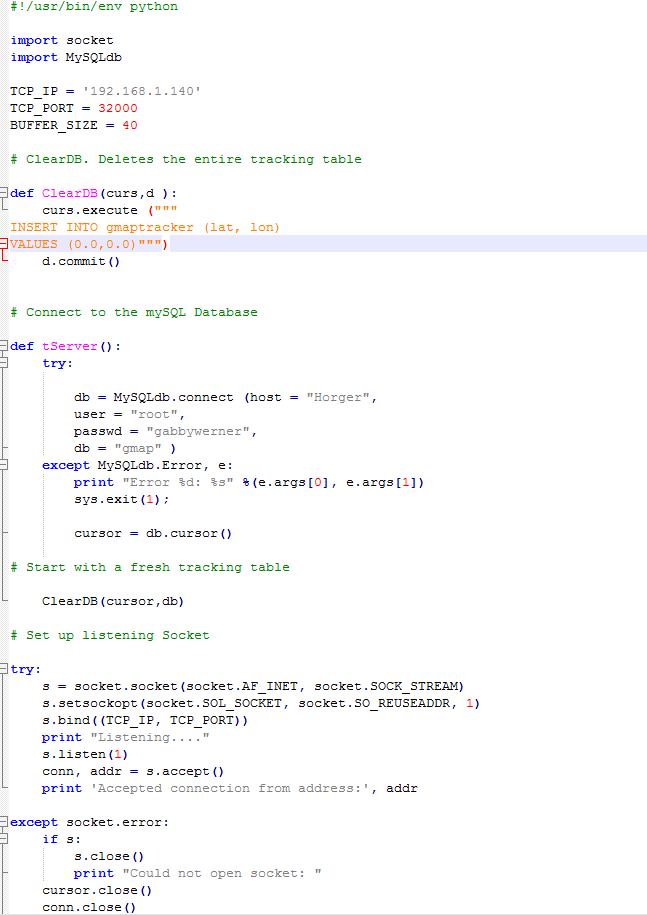
**API Services**

Elevate uses Google Maps API in order to draw and populate a map with specific routes given a function that maximizes elevation change over an array of JSON coordinates. This map uses a specialized API key and a synchronous AJAX call to obtain the data from the service Google provides. We are only allowed a certain amount of calls per week under the free plan. If you are interested in obtaining more calls, please purchase the business plan under Google Maps API hosting.

**Database**

**Python Services**

A big part of our database is accepting GPS coordinates. Due to a lack of budget and time constraints, we were unable to fit the helmet with GPS and GPRS coordinate functionality. However, the Python server accepts a network connection (which would be our GPRS SMS connection) and parses GPS JSON coordinates into a mySQL database that is hosted locally on a machine. This operates using a network adapter specifically designed for Python 2.7 (mySQLDBA). Once coordinates have been received, it closes the socket connection in order for another connection to be obtained and coordinate transfer to occur again.

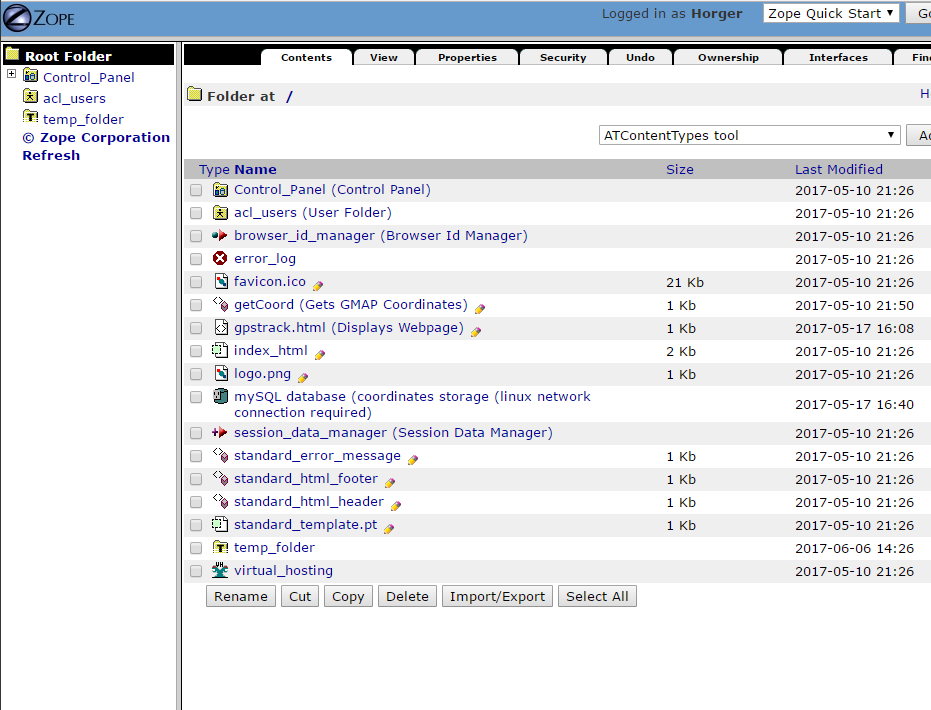


**mySQL**

Our mySQL server is hosted locally on a machine. It consists of a single table with three options. One option is for longitude coordinates that restricted to a long data type and only 10 digit accuracy. One option is for latitude coordinates that are restricted to a long data type like the longitude option. The other option is an id field just to make sure that overflow does not populate and fill the entire database. It uses a restricted function that only allows 50 entries before it fills the first entry to be overwritten.

**Plone CMS**

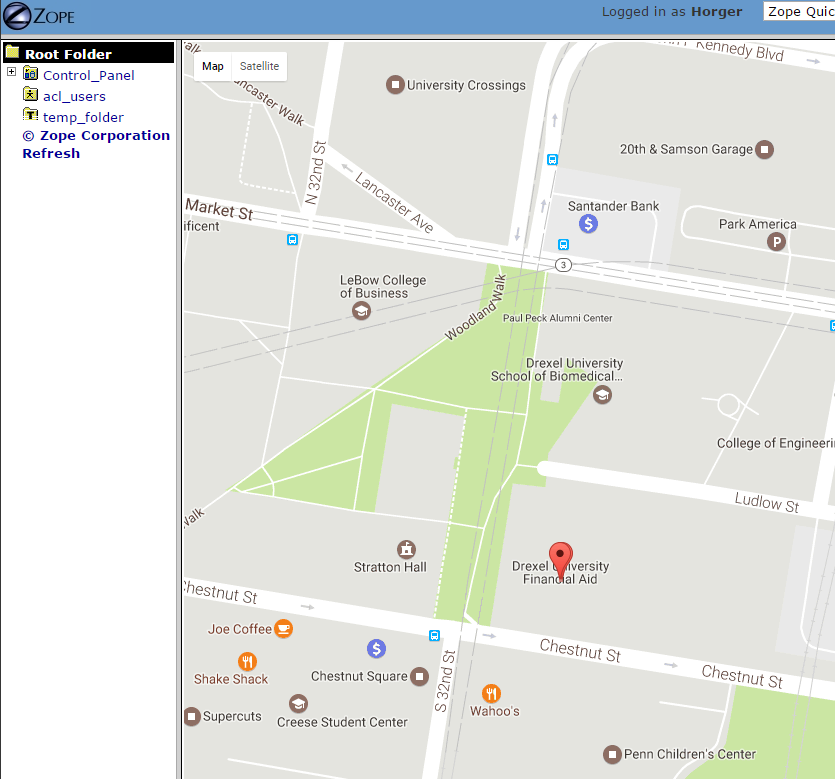
We use Plone Content Management System to host and integrate all of our services. It is hosted on a local machine the same as the Python Server and the mySQL database. However, for other machines to access it, the port 8080 must be forwarded. Plone uses Zope, an adapter that allows for a graphical user interface so database managers can add different modules for our services. It also allows for the creation of user accounts and more security options. Plone can be accessed via http://localhost:8080/manage\_main once the instance has been installed. Our recommendation is to install Plone for Windows and then try it using Ubuntu for better integration of Python and mySQL. There are better network adapters for mySQL for Ubuntu due to a non-binary distribution.



**Integration of Services**

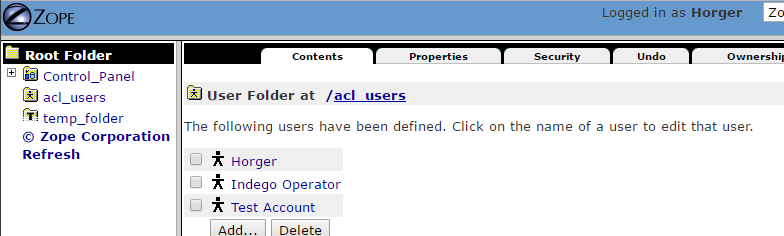
We use various methods and adapters to integrate mySQL and Python into our Plone instance in order to bring working functionality for our Google Maps webpage. We use a ZmySQL query using the Z Gadfly instance addon to select the recent coordinate in the latitude and longitude tables in our database. We then use a DTML method which also using XML to access this query and insert the coordinates into usable HTML or scripting capabilities. The nice thing about Plone is that we don’t have to port forward or even have the files in the same folder for it to work.

**Google Maps**

Once we have the DTML method working and obtaining coordinates, we have a HTML webpage that creates a new Google Maps using their API and our generated account key. We create markers using the same coordinates from the DTML method and place them on the map every 30 seconds. However, due to our network restrictions, we have our map stylised for elevation data so that markers can be placed locally for demonstration purposes. 

**Accounts**

Plone allows for the creation of user accounts via Zope. To create an account, go to the acl\_users section of your instance. There you can create new accounts and give them specific security options, such as access to certain elements of the instance. End users should only have access to the gpstrack.html. For users to login, remember, your instance must be port forwarded on 8080.



**Contact Information**

**Activation Hotline - 1800-ELEVATE (1-800-353-8283)**

**Support Team - support@elevate.org**

**Warranty Information - warranty@elevate.org**

**Business Inquiries - contact@elevate.org**