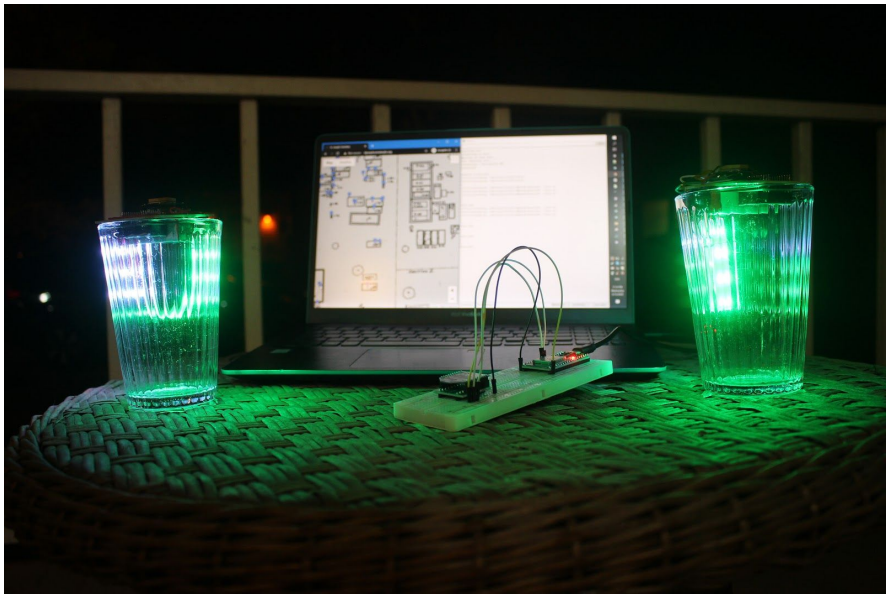
Gravesite Beacons



Claire Luikart -- luikart3
IoT Spring 2020 Capstone Project

The Project

Narrowing down a project topic was difficult. I had so many ideas ranging from interactive art project giving physical space for digital social media memorials to formalizing a series of small IoT centric projects to use as teaching platforms with my own students to building a device to fire bouncy balls at command to keep my cats entertained during the day. From my many ideas I tried to choose one that was practical and would expand areas and topics I have had less experience with. This brought me to the Gravesite Beacon project.

While teaching computer science at a Catholic high school, I have encountered many religious sisters with strange technology requests. Most recently, I have been volunteering with a group of sisters and volunteers who tend and support St. Joseph's Historic Cemetery by helping them digitize their records. One of the difficulties they have had with this is that while the map and records of the cemetery contain the exact names and dates related to the individuals buried there, they have no means to dispense this information to visitors of the cemetery and use the data in a meaningful manner.

Opened in 1824, the cemetery contains a wealth of history[1]. The earliest settler's remains were transferred there from the old Spanish founded cemetery. It is the Catholic diocese cathedral cemetery. Founders of Southern University, free-people of color, priests, plantation owners, artists, and so many more reflections of the cross section of the city are laid to rest there. Burial traditions and styles from many countries that have ruled over and lived in the area are captured within its gates. Foil wrapped bricks, small graveside offerings, house shaped tombs, oven mausoleums, wrought iron crosses and many more subtle curiosities can be found interspersed between more modern granite slabs.

[1] https://historicalbatonrouge.blogspot.com/2011/07/catholic-cemetery.html

A gravehouse.

The historical marker at the gate.

A damaged headstone.

It is a living history. While restricted to family lines, burials do still occasionally happen here. In addition to the few new interments, the press of entropy has taken its toll. Some graves are broken open. Some headstones have worn down to illegible smoothness. The ground is unlevel; there are many more buried here than have markers. The identities of some have been lost to time immaterial. In an era before social media gave a platform for any and every person to tell their own story and leave some mark, for many, their grave is the testament to their lives and one of the few records they existed[2]. Part of the importance of this project is providing an additional record and means for these individuals to continue on in the historical record.

While conducting my field test, I met two women in the cemetery. They were sisters looking for the graves of their relatives who had lived in Baton Rouge. I helped them find them using the csv records file and map I had with me in my testing kit. Ease of access to full cemetery records is not normally this straightforward. For more active and better kept cemeteries, there is normally a registry at the gate or some leger you can look up plot information in provided you can find the right groundskeeper willing and able to help. For older, less active cemeteries, there is less or no need for a constant on site caretaker. Making these situations more complex is that many of these older cemeteries have irregular plot allocation schemes or only partial records. This can leave any visitors in a lurch should they be unable to find a specific plot. While there are many web services that provide lookup for finding grave locations[3], the information they have on hand is limited, does not include a specific location and is information filtered through data collection. Part of my project aims to alleviate this need for a physical, on site presence to locate specific graves by digitizing the records.

For the women I met in passing, even though they had never met their relatives and they had been dead for some time, visiting the gravesite provided a sense of deeper

[2] https://www.houmatoday.com/article/DA/20160201/News/608082491/HC
[3] https://www.findagrave.com/

understanding to the chain of events that led to them. I know, personally, when I find myself in the neighborhood in New Orleans, I'll pop by the cemetery where my family's tombs and graves are and take a moment to reflect and remember. Cemeteries provide an important physical link to our collective historical past[4]. They provide a space to reflect on our lives and remember those who have departed. Death is part of life and in all places, especially south Louisiana, we have a wealth of traditions associated with death, mourning, burial and remembrance. Preservation of cemeteries is part of this with many families participating in annual grave cleanings for All Souls Day in November and groups organizing to "save our cemeteries" both at large[5] and small[6] scale. This project is an extension of this preservation from the physical to digital space.

The stated goals of this project are to create an interactive mapping of the known gravesites within St. Joseph Cemetery and to build a beaconing system that allows these individual sites to be found by visitors. This system also provides an additional historical record of the individuals interred at the cemetery. The core driving factors are to not only preserve an important slice of history and culture but to bring it into the modern era through an accessible platform.

[4] https://www.youtube.com/watch?v=W4-0iAzFIcI
[5] https://www.saveourcemeteries.org/
[6] https://archive.sjabr.org/news/newsdetail.cfm?id=838 and http://www.louisianafolklife.org/LT/Articles_Essays/cem_pres.html

Design Considerations and Execution

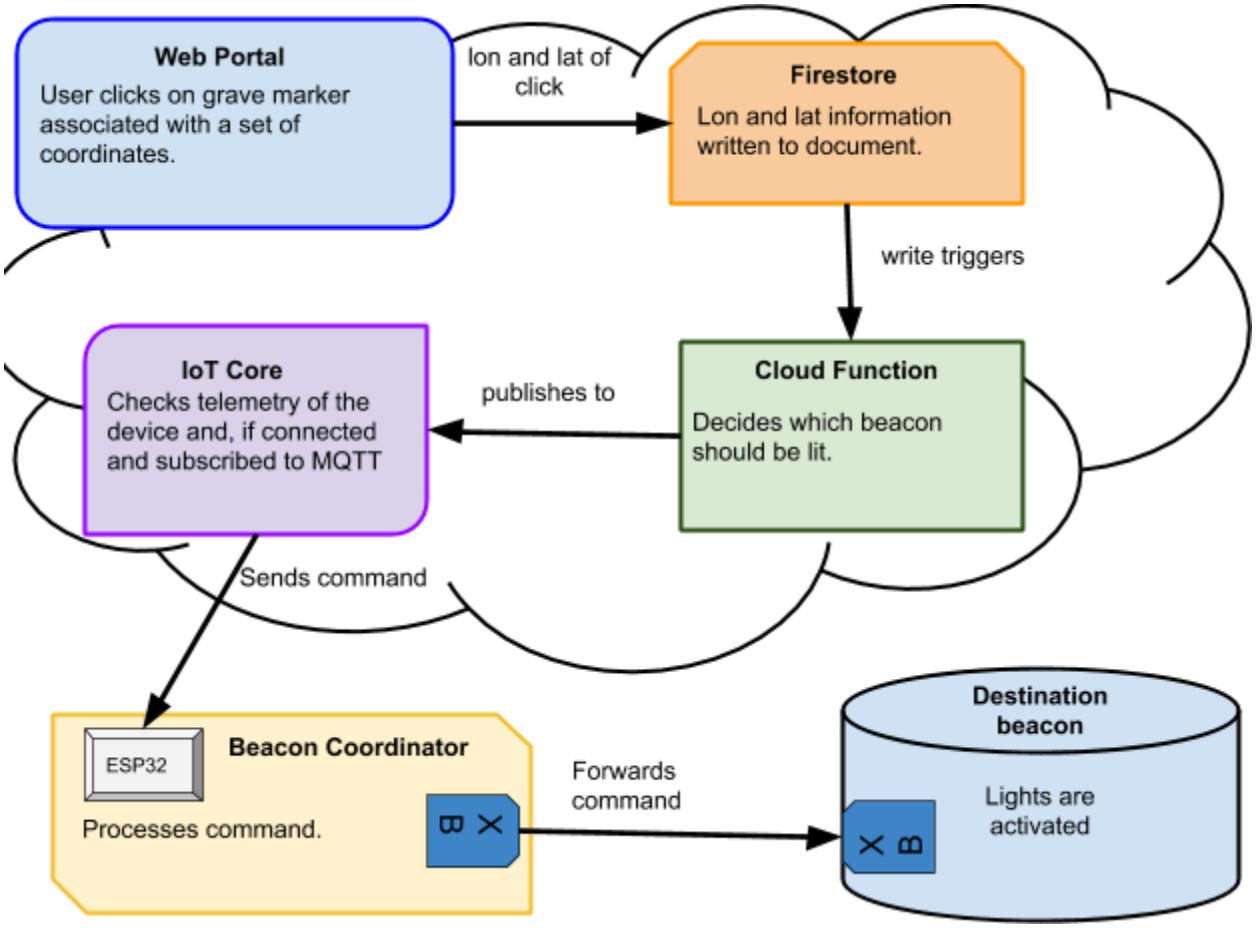
Nearly all physical parts for this project were sourced from previously created labs, leftovers from personal projects and leftovers on loan from my workspace. While this design choice was mostly fueled by availability of parts and finances, power consumption and practicality of design were also major considering factors.

During the prototyping process, I started with the areas most familiar to me. With each build addition or revision, I attempted new techniques and started pruning away less necessary components. The primary focus was creating functional devices, a web portal and providing communication from device to portal each other.

The Web Portal

The current web portal.

The web portal provides the means for the user to select which beacon needs to be activated. It displays a map of the graveyard and sites are marked with pins. The user can hover over these pins to see relevant information including name and plot number. Clicking on the marker pin initiates the chain of events to light a beacon.



High level overview of the chain of communication from click to beacon activation.

The Portal Itself

When creating the web portal, two possible avenues were considered. The first option was to create the web portal using Google's Maps API. This provides easy access to Google Maps style tools, a slick user interface, easy mobile integrations and allows geolocation of pins that could easily correspond to burial locations allowing user selection of their desired beacon. However, past the free trial of Google Cloud Services, there is a usage fee.

The second option was to use an older, "Web 1.0" approach in the form of the HTML Image Map tag. This requires the map of the graveyard to be loaded into the page as an image and a series of interactive areas to be mapped to the image. This requires considerable more initial setup and requires mapping from GPS coordinates to image coordinates and does have some limitations especially with shape options. However, it is essentially free.

Both options were explored[7] and the final users of this system were consulted. I was able to secure an individual to pay for any future costs incurred past the $300 of free services offered by the Google Maps API, I opted to go that route as it has built in support for the core features desired: location pins and map overlay.

[7] See "Final Scratch Work" directory for examples.

To add location pins for each grave, geoJSON was employed. The geoJSON specifications provide standardization and support for describing map features including multiple geometries, coordinates, feature type and a space for additional user defined properties[8]. For this specific instance, the format includes the plot and section information as well as the individual's name in the properties section. In this situation, all geometries are points. Below is a specific example from the dataset to demonstrate the format.

```
{
  "geometry": {
    "coordinates": [
      -91.174183,
      30.452212
    ],
    "type": "Point"
  },
  "properties": {
    "fname": "Phillip",
    "lname": "Theille",
    "plot": "29",
    "section": "1"
  },
  "type": "Feature"
 }
```

I wrote a short Python script to convert the csv file of the cemetery record[9] into this format. This is the aptly named csvtogeojson.py in the provided code. It requires the geojson python package[10] to run and it assumes the existence of a nameNplotNcoord.csv file with fname,lname,plot, section, Lon, and Lat columns. The script strips leading whitespaces from the names. It will output a graves.json file for future consumption.

```
import csv
from geojson import Feature, FeatureCollection, Point #import via
pip install geojson
import json
import numpy

csvFilePath = 'nameNplotNcoord.csv'
```

[8] More information on the specification here: https://geojson.org/ and https://tools.ietf.org/html/rfc7946
[9] This file was provided "as-is" and does have a few quirks like loose spaces and surprise characters
[10] https://pypi.org/project/geojson/

```
features = []
with open(csvFilePath) as csvFile:
    csvReader = csv.DictReader(csvFile)
    for row in csvReader:
        if row['Lat'] != '': #check that this is a located plot
            latitude = float(row['Lat'])
            longitude = float(row['Lon'])
            features.append(
                Feature(
                    geometry = Point((latitude, longitude)),
                    properties = {
                        'fname' : (row['fname']).strip(),
                        'lname' : (row['lname']).strip(),
                        'section' : (row['Section']).strip(),
                        'plot' : (row['Plot']).strip()
                    }
                )
            )
collection = FeatureCollection(features)
with open("graves.json", "w") as f:
    f.write('%s' % collection)
```

Since part of this project does involve cleaning data and matching existing plots to geolocations, only a subset of the graves were geolocated for the sake of expediency. Section 1 was added manually for testing purposes. I am exploring methods to do this proceducally in the future using OCR to lift plot numbers and mapping GPS coordinates to pixel coordinates but that is not the primary scope of this project and course.

With the geoJSON formatted file in hand, I was able to utilize the Google Maps API to easily and quickly import and place points for these individual gravesites.

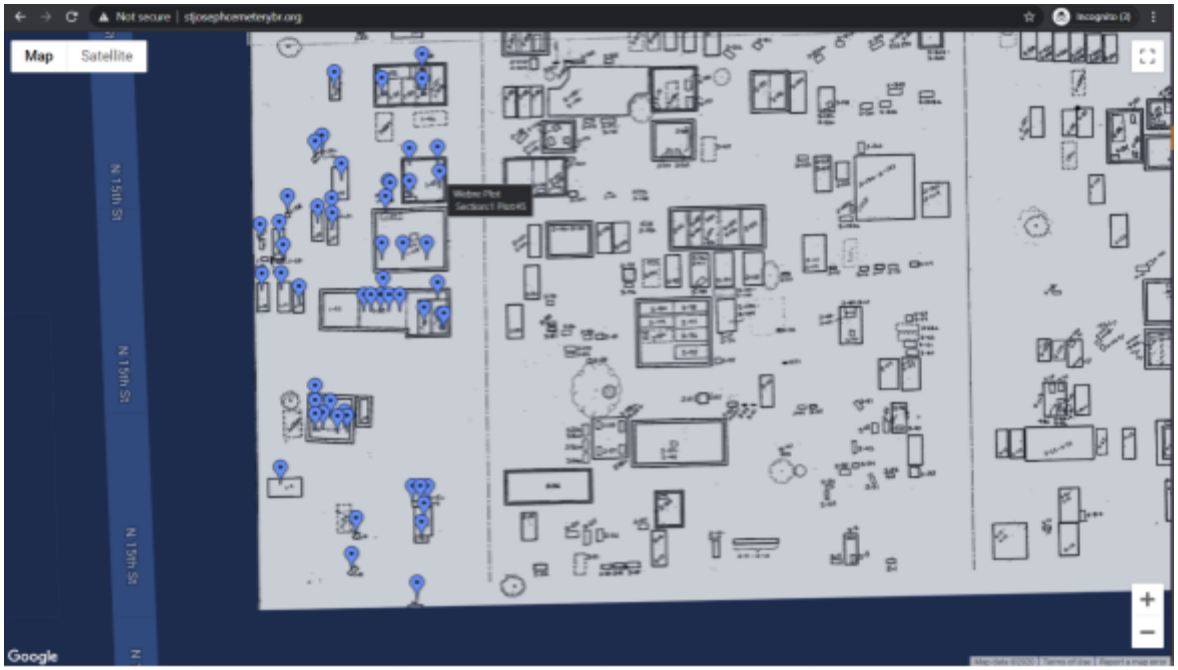
Figuring out Map Overlay proved somewhat difficult. The documentation included several samples of code for different image and custom map methods and little explanation for where the given coordinates for the image were rendered from. After several false starts, I found a simple method for overlaying map data requiring the image and a boundary for each of the four directions given in the form of a longitude or latitude[11]. The map used was lifted from a pdf provided by the organization[12]. Excess edging was removed and the map rotated to match the skew of the actual plot of land the map represents. Finding the boundary values was done by opening Google Maps, selecting the given corners of the cemetery then making minor adjustments in by sight.

Additional styling and formatting of the map includes switching the color scheme to a customized color scheme requested by the clients, changing the markers to blue, adding

[11] https://developers.google.com/maps/documentation/javascript/examples/groundoverlay-simple

[12] The original map is included in the provided repository in the “Final Scratch Work” directory as St. Joseph’s Cemetery Map

restrictions to zoom and pan functionality to keep the focus tight on the cemetery and turning off Street View. I found this helpful styling tool that took much of the tedium out of creating the styling document.



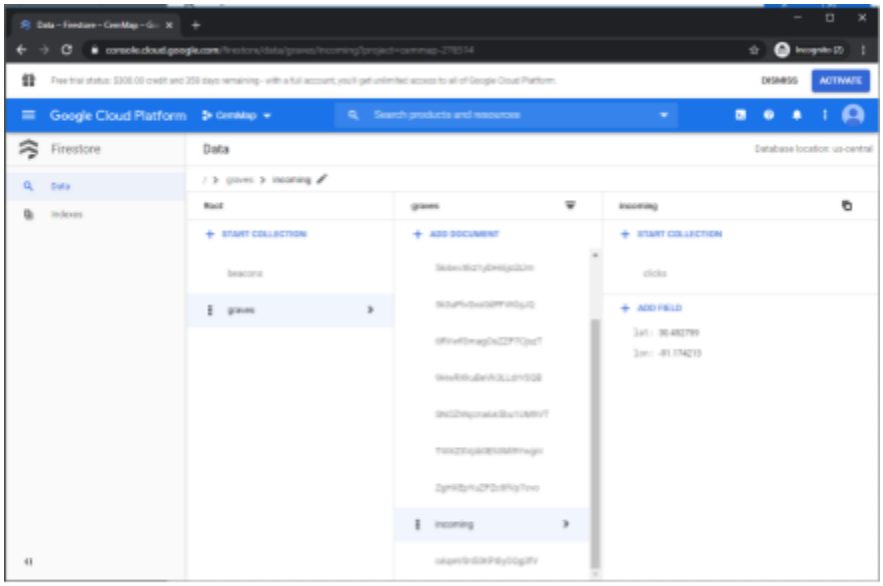
The current web portal.

Currently, the web portal is only thoroughly tested in Chrome. Because it is lacking certificates, Chrome will prompt you that there is danger. Please select “Advanced” and then “Proceed” when initially accessing the site.

From the Portal to the Cloud

Communication from the Map to the Google Cloud Services account I had created was done by means of creating a Firebase and Firestore. Firebase[13] is a cloud based app platform from Google and Firestore[14] is the associated serverless document database. There is web supported write capacity for both and I was able to configure the click action on the individual markers to write a record to the Firestore.

[13] https://firebase.google.com/docs/web/setup
[14] https://firebase.google.com/docs/firestore



The current Firestore configuration.

The Firestore document structure currently includes a document that is overwritten every time a user clicks. This document is located at /graves/incoming. This provides the function trigger. Long term storage of clicks is underconsideration. The Firestore does currently contain the collection /beacons/ as well for future implementation.

From the Cloud to the Coordinator

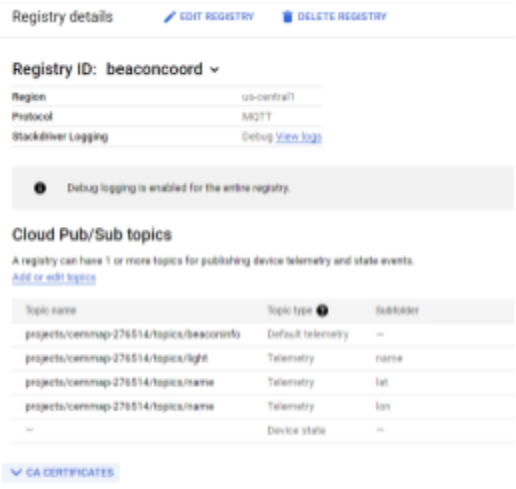
Once the record is written to the Firestore, it triggers a cloud function[15]. This cloud function will issue a message to the MQTT connection of the coordinator device. Currently, it randomly generates which beacon to publish to and sends the message “light on” to that command subfolder. However, this is the point in the communications chain where the incoming click data should be processed to find the nearest known beacon. This task is not possible at the moment as I have not been able to successfully pull GPS coordinates from the coordinator to the cloud so the nearest known beacon is a mystery. However, this is a solid proof of concept that a cloud issued command can function as a switch for the beacon.

The starting basis for my function can be found here and here. A copy of the cloud function is included in the code repository in the “google cloud deployed assets” as cloud function contents.txt. Here, the index.js and package.json are both included.

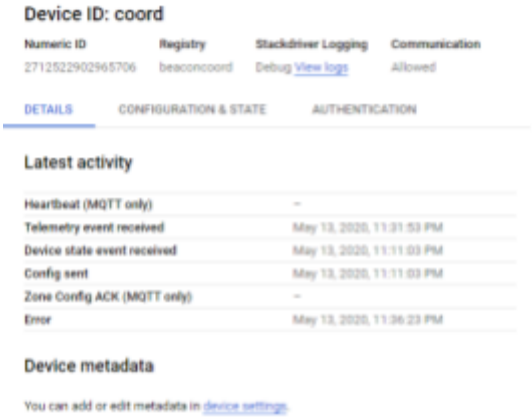
In order for this command to be sent, we have to configure devices for the command to be sent to using the Google IoT Core. First we must create a device registry, then add a device to it and provide a certificate for the device to authenticate its connection. As only one device needed to be created, this was done through the console interface instead of the Google provided command line interface. I followed much of the documentation found here. The created registry is the beaconcoord registry. It is configured to use MQTT as its communications protocol. Once the registry was created, the coord device was created. Generation of

[15] https://cloud.google.com/functions/docs

certificates is not an automated process within the Google IoT Core[16]. The keys and certificates were created using openssl in a later described process during the physical device configuration process. (See Beacon Coordinator)



The beaconcoord device registry.



The coor device.

Additional Notes for the Web Portal

From my incredibly limited experience with using cloud services[17], comparing the Google Cloud and Amazon Web Services there are many similarities but also some stark differences. The most noticeable of these to me is that Google Cloud automates a great deal of the permission creation and interconnectivity of assets. Since it sections the assets on the cloud by project, the other assets within a project are given some access rights by default. For example, when I set up the cloud function to process incoming clicks, it generated the correct permissions for access to the associated Firestore. Establishing more fine grain controls is possible but the default setting seems to more often than not be sufficient. This subdivision of access by project is part of why having visible API keys when using Google Cloud seems less frowned upon in my readings and forum dredges. Since the key is only linked into one project and you can configure and impose limitations on incoming connections for that key, the key itself can provide some access control besides just to the API resource.For the most part, I like this approach of providing interlocked assets within individualized project containers.

Hosting for the site was configured using Google Domains and Google Storage[18].The site is not yet registered through a certificate authority pending revision and approval by the cemetery's governing body and the strong possibility of folding in of the content to another site.

Had I been aware of the utility of Google Firebase at the beginning of this project, that route would have been taken from the start. In the future, I may move from Google Domains hosting to Google Firebase hosting to better incorporate the Firestore usage and establish tighter access control.

---

[16] https://cloud.google.com/iot/docs
[17] Outside of broad concepts, I know what I learned in lab 4 and what I have learned for this final.
[18] https://cloud.google.com/storage/docs/hosting-static-website

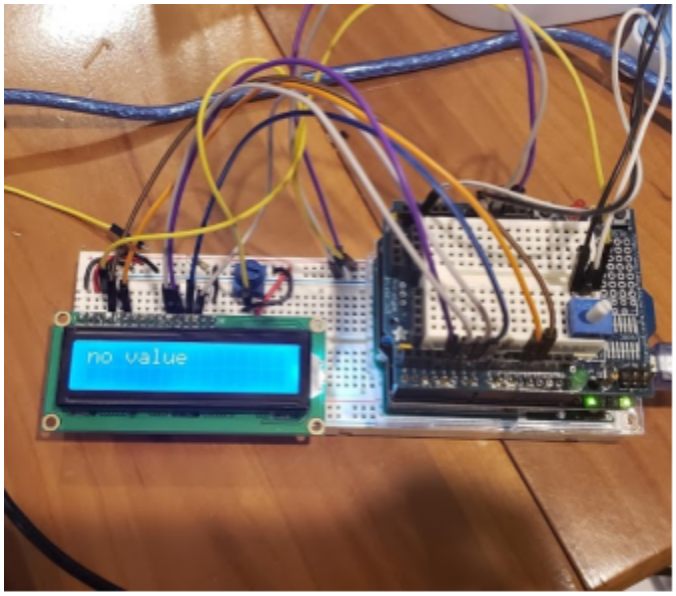
The web portal code is located in the Web Portal directory of the attached code repository. The API access keys for the associated Firebase and Maps are included as it is linked to the specific project on Google Cloud and not the general account.

The Beacon Coordinator

This device receives communication from the web portal and sends that beacon the power up signal. In all incarnations, this device includes the coordinator XBee.

Revision 1: Testing Device

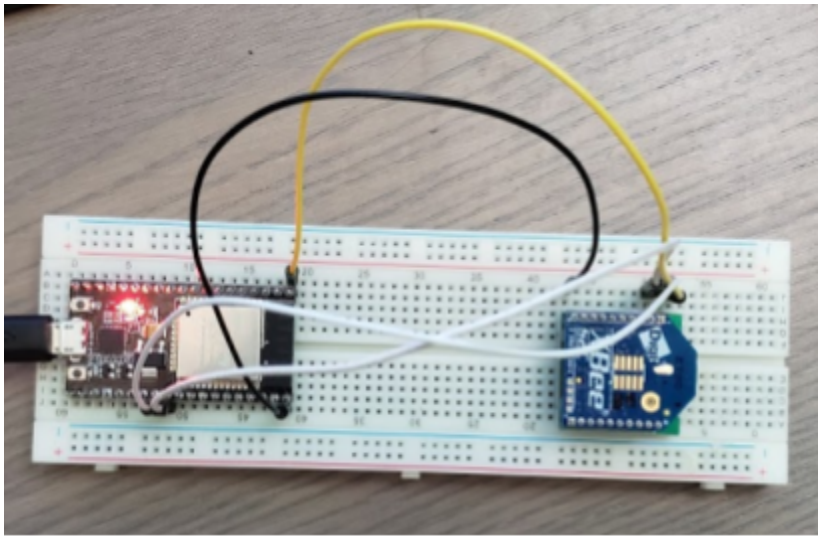
The first iteration of the beacon device consisted of an Arduino Uno with XBee shield holding the coordinator. As an input, it has a potentiometer. Depending on the reading from this potentiometer, it will send no message or a message for beacon A, B or C. As an output, it has an Arduino LCD. This LCD will display any incoming information from the XBee communication if present, or the current potentiometer level.



Beacon coordination revision 1.

This device was built as a quick means to test viability of communication between coordinator and beacon without using XCTU. This device was used during the field test process to verify beacon functionality.

Revision 2: Cloud Connected Device



The beacon coordinator.

After building out the web application, research and exploration turned to how to close the communications gap between the portal and the beacon coordinator. This led me to the conclusion that adding a device to Google's IoT Cloud and using Google's provided cloud APIs was a viable solution and possible with the equipment I already had on hand. It requires the coordinator device to include an ESP32. My initial thought was to configure a series of topics for inbound and outbound messages from the coordinator to the core and vice versa. Inbound would contain beacon names and GPS coordinates for storage and outbound would contain information about which beacon to trigger. However, the Google Cloud MQTT setup is slightly different from the AWS MQTT setup. Since this is an actual physical device hooked up to the Google Cloud and registered as a device, a handful of premade topics exist. Additionally, there is a preconfigured topic to issue commands[19] through.

Initial testing was done following the instructions and using the framework and resources found at kevinlutzer's GitHub.This base strictly adds in Arduino MQTT connectivity specifically to the Google IoT Core. While it is not intended for production, it provides authentication to the project and works for our current purposes. It builds on another GitHub from 256dpi which provides Arduino MQTT support and is tremendously helpful reading.

In order for this device to function using this framework, the Google IoT Core must have been previously configured as described in the "From Cloud to Coordinator" section. The keys

19 https://cloud.google.com/blog/products/iot-devices/introducing-cloud-iot-core-commands-increased-flexibility-to-control-your-fleet-of-embedded-devices

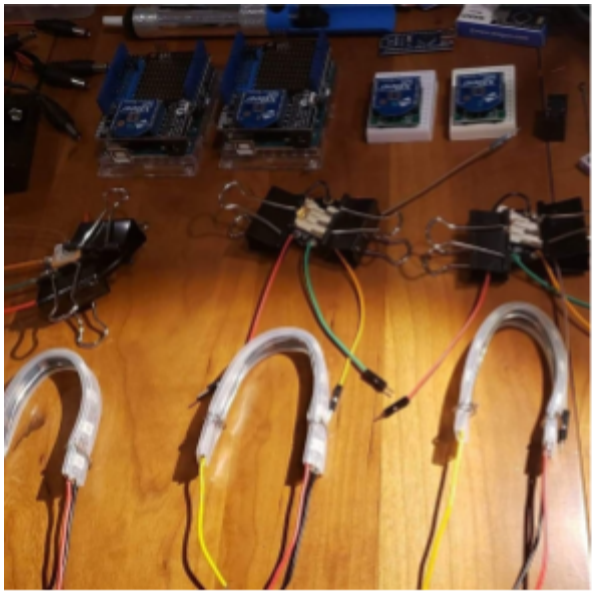
and certificates created here are the ones used in the device creation process and connection configuration code[20].

The current incarnation of the beacon coordinator is composed of an ESP32 and the coordinator configured XBee.

Additional Notes

The code for the initial testing device can be found in the code repository in the “beacon coordinator code” directory as beaconcoordinatorv1. The code for the second, cloud connected revision of the device can be found in the code repository in the “beacon coordinator code” directory as beaconcoordinatorv2. It uses the base found in kevinlutzer’s GitHub as linked above but includes additional code for Xbee communication out. My wireless SSID, password and ec key are currently included in the code’s configuration file and require modification before use.

The Beacon Devices



Build day.

These devices are intended to be planted in individual areas. For the scope of this initial prototype, 3 beacons were constructed. Beacon B is an older revision of the internals using a breadboard as a base. Beacons A and C both have small protoboards with their connections soldered in place to minimize their footprint. Additionally, these protoboards fit neatly into the prefabricated beacon housing that is currently in use. Each beacon is composed of an Arduino Nano, an Adafruit Flora GPS, a strip of 9 Neopixels, an XBee and a 9V battery.

[20] Found in this example at line 43 https://github.com/GoogleCloudPlatform/google-cloud-iot-arduino/blob/master/examples/Esp32-lwmqtt/ciotc_config.h

The Housing

The housing for the device was a critical component in design. As these devices are intended to function outdoors for extended periods of times uninterrupted, a sturdy, waterproof case. The initial housing is a deconstruction of the existing housing of the solar lamp. It is a trapezoidal prism with a hollow inside. The lid snaps in place providing a tight seal.The housing also has a reflective material on the interior bottom and a bevel to encourage reflection of light to the outside. The perma-proto board that the beacon internals are attached to fits neatly into the top lip of the casing it is held in place with glue dots[21]. The casing is then closed.

The prefabricated housing.

Additionally, several drafts of potential 3D printed and laser cut housing have been included. These were not able to be physically prototyped at this time but are included in the “Reflections, Next Steps and Future Expansion” section.

The Controller

The initial prototyping for these beacons took place on a pair of Arduino Uno’s in part due to their ease of access and XBee shield support. While they are able to fit into the prefabricated enclosure, they are bulky. Attempting to fix this issue, the project was moved onto the Arduino Nano[22]. So far this has been sufficient for the prototyping needs. Currently, the GPS and XBee transmit to the Nano are connected to interrupt pins for future experimentation and implementation.

The Communication

Communication from the base station to the individual beacons was done by means of configuring an XBee network. The XBee devices onboard the beacons posed several challenges. In the interest of consuming little power, they should be able to sleep, meaning they should be edge nodes. However, assuming the communications are all originating from one fixed point potentially at the periphery of the cemetery, it is possible that individual beacons will

[21] https://www.amazon.com/Glue-Dots-Permanent-Dispenser-Adhesive/dp/B004OKTTWM
[22] http://www.circuitstoday.com/arduino-nano-tutorial-pinout-schematics

need to forward on their messages to the appropriate beacon meaning they should be configured as router nodes. In their current configuration, the beacon XBees are set as routers.

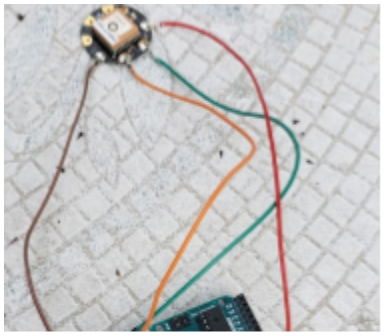
The XBee communication configuration is included in the root directory of the code repository.

The GPS

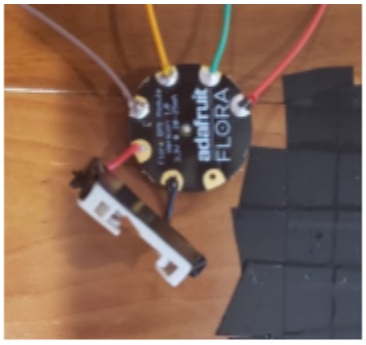
The GPS module selected for this project was the Adafruit Flora GPS[23]. Unfortunately, the number on hand did limit the number of devices that could be constructed.

In order for the GPS module to properly get its coordinates, it needs to get a “fix” on a satellite. The satellite has a precise atomic clock on board. These satellites are synced to each other and known ground stations. As the satellite is constantly broadcasting its time and we know that radio waves have a fixed speed, position can be computed by the GPS receiving device using the variance in known time from the acquired broadcast time information. For the device to function properly, it needs to have its ceramic antenna facing up, as parallel to the ground as possible and be minimally obstructed. There was concern that the current enclosure would block this vital process but testing shows that this is a nonissue.

An additional 3V coin cell battery was attached to the GPS module at the recommendation of the documentation. The battery allows the GPS module to act as a real time clock and more quickly attain satellite fixes[24]. The coin battery holder was glued to the bottom of the Flora to allow a better fit into the housing. An external antenna was tested but no appreciable difference was found in quickness of fix or accuracy so it was excluded from the build for the sake of space.



Initial testing and playing with the Flora GPS module.



The Flora GPS and battery holder.

Affixing the coin cell holder to the bottom of the GPS module.

The GPS module delivers data to the Nano in a NMEA formatted string. NMEA stands for National Marine Electronics Association who standardized the data format. The message includes longitude, latitude, speed in knots, altitude, direction of longitude, direction of latitude, a checksum, a time stamp and a ton of additional data[25]. The beacon devices currently only retain

[23] https://www.adafruit.com/product/1059
[24] https://learn.adafruit.com/flora-sensors/wearable-ultimate-gps-module?embeds=allow
[25] https://www.gpsworld.com/what-exactly-is-gps-nmea-data/

the longitude and latitude data and directions. The NMEA data can be converted to degrees[26]. This was considered for the storage and transmission of the data but precision was lost in the resulting coordinate. Since these devices will be in close quarters, more precision is prefered.

The GPS and wireless communications devices have not interfered with each other. Though they operate in the same frequency band (ultra high frequency), the code staggered access and execution of these radio frequency activities such that they should not be occurring simultaneously. It has not been observed that inbound communication from the coordinator device is impacted. This area is still open for investigation.

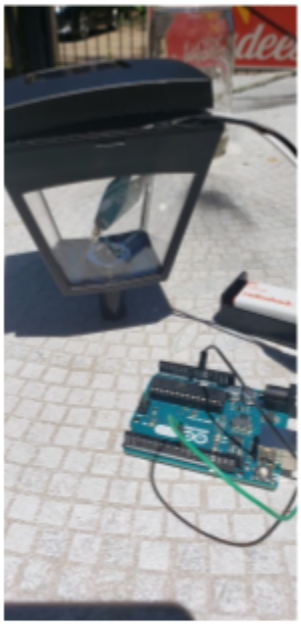
On beacon startup, the GPS connects and begins searching for a fix. Once a fix is found, the longitude and latitude are stored onboard the device and sent on to the coordinator via the XBee. As the device runs, it will periodically check for new GPS communication. If the device is moved and a new GPS coordinate is detected, it will save this new coordinate and forward on the new information to the coordinator. This is the beginning of allowing the beacons to self report their location to the cloud and the starting point for adding in theft detection.

The Lighting Element

The lighting element of the beacon is how it signals to the user it's location. This does require that these lights be visible during the day as the primary use case involves visitors to the cemetery during the day accessing the beacons.

On hand candidates for the lighting element included: several form factor of Adafruit Neopixels[27], Adafruit sequin LEDs[28], automotive accent LEDs[29], car headlight accent rings[30]. The selection process included connecting each lighting option one at a time to an Arduino Uno. A multimeter was connected to the LED leads as well to monitor power usage. Information was also compared to the datasheet. In addition, simple visual observation of the LED's performance and visibility in sunlight and at distance was monitored.

Testing the Neopixel U.



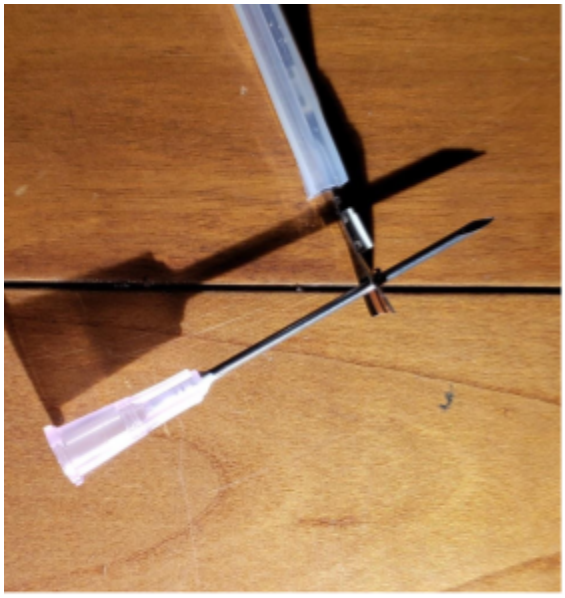
Failed individual Neopixel test.

[26] https://learn.adafruit.com/flora-gps-jacket
[27] https://learn.adafruit.com/adafruit-neopixel-uberguide/form-factors
[28] https://www.adafruit.com/product/1757
[29] https://www.superbrightleds.com/moreinfo/led-light-modules/round-led-accent-light/17/113/?redirect_disc=0
[30] https://www.superbrightleds.com/moreinfo/led-halo-rings/led-angel-eye-headlight-accent-lights-cob/1135/2693/?redirect_disc=0

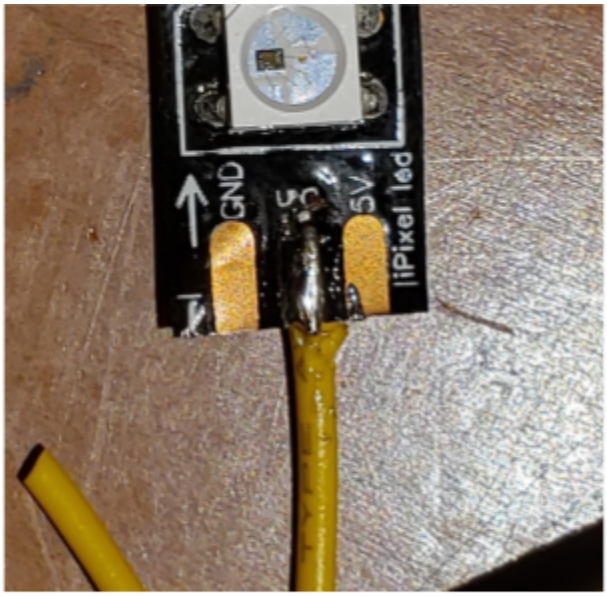
Results are displayed in the table below.

	Advantage	Disadvantage
Adafruit Neopixel individual	Programmable color/behavior. Easy to solder a chain of. Easy to affix. Visible in daylight.	Have to load in a library. One alone is not enough.
Adafruit Neopixel ring	Programmable color/behavior. Easy to affix. Visible in daylight.	Have to load in a library. Position and angle of the lights is rigid.
Adafruit Neopixel strip	Programmable color/behavior. Flexible. In a waterproof casing. Easy to affix. Visible in daylight.	Have to load in a library. Flexible. Soldering these is difficult.
Sequin leds	Low power draw. Many onhand. Easy to attach leads to. No additional libraries needed.	Single light insufficient to be seen in daylight. Single color. Only on hand colors white and red which sets a menacing tone.
Headlight accent rings	Really pretty. Very bright. Attractive blue color. On hand. Visible in daylight.	Requires at least 7V to run. This is not native output of an Arduino GPIO pin. While not impossible to find power boosters for an Arduino, adds to the footprint of the device.
Automotive accent LEDs	Come with handy bits of prebuilt stick'em included on the back of the light.	Dim.Power requirements over Arduino pinout.Single color.

Neopixel strips were chosen for the final product as they are the most flexible, allowing tinkering with their behavior to find a good signal and are visible in daylight. My past experience with these strips has shown that the solder pads can and will separate from the strip. To minimize this, holes were poked in the strip, the wires hooked through then soldered in place.

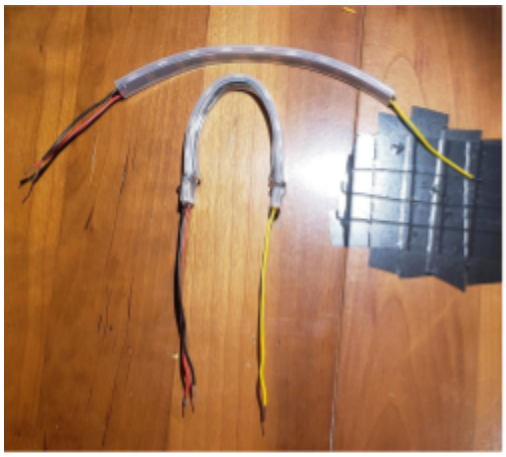


Creating the hole with a needle.

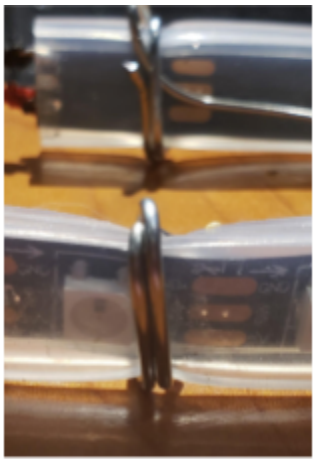


The Din wire hooked and soldered.

This method has been effective in the past for securing the solder point and ensuring connectivity of the lights. This is a technique devised by my colleague, Linda Weimer, one of the sewing teachers, during a joint project between the technology and sewing classes.[31] To support the flexible Neopixel strip and keep its form, a length of steel wire was bent into a U shape and the ends twisted around the protective silicone coating of the strip. The ends of the strip were waterproofed with silicone sealant.



Before and after addition of the frame

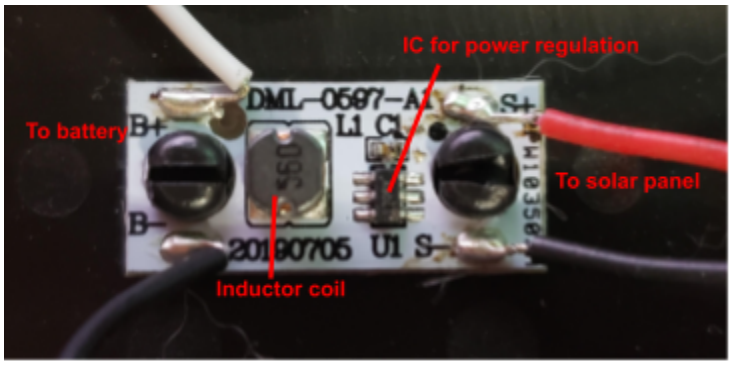


Detail of the frame fastenings.

[31] https://www.youtube.com/watch?v=XdCef0wDUFc

The Power

Reverse engineering the built in solar power collection system was difficult. The original garden lights were designed to run on 1.5V batteries. The existing framework solar lantern has several parts: the solar panel, the rechargeable 1.5V nickel metal hydride battery, an LED and a voltage regulation circuit to prevent battery overcharging. Since this regulation circuit was designed to power an 1.5V battery, which cannot power the beacon device in its current incarnation.

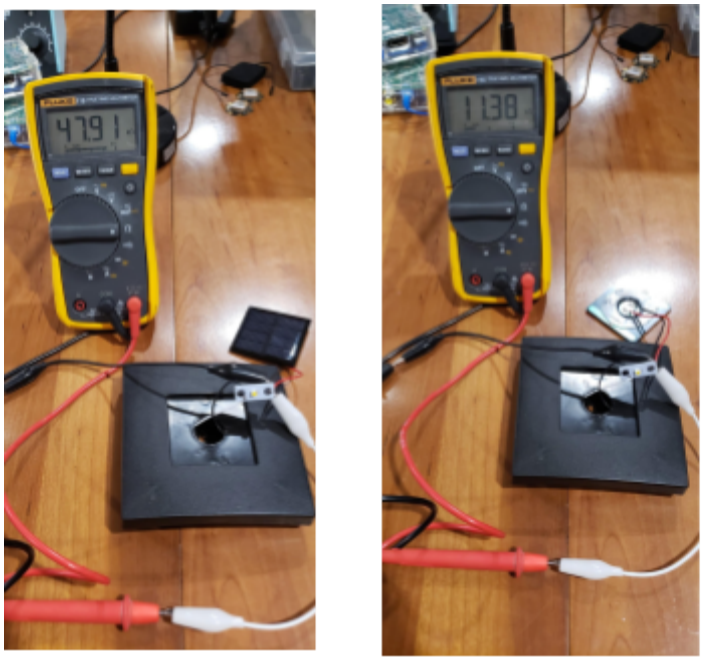


The built in control system.

Several approaches were taken to attempt solar harvesting using nickel metal hydride and lithium polymer. I tried to incorporate several different prefabricated voltage regulation circuits[32] but found they required more input amperage than is available from the single solar panel.

During exploration as to the properties of the solar panel itself, led me to several conclusions. In full sun, when connecting a multimeter directly to the solar panel and measuring the voltage, it was able to pull 2.1V in full sun and 1.7V in heavy clouds. There is a significant difference in the resistance of the panel when it is well lit versus when it is not well lit. I think the basic device uses this resistance to determine when to fire the built in LED. Using these values and Ohm's law, I was able to calculate the possible current of the panel to be 43.83 *microamps* for 2.1V and panel facing up resistance values from my outdoor measurements (47.91kOhm). Without additional panels to pull down a higher voltage or some other means to alter the resistance, this panel cannot work with the battery charging and voltage regulation circuit boards I had on hand as their expected input is higher than what is currently provided by the panel.

[32] https://www.adafruit.com/product/14 and https://www.amazon.com/gp/product/B07T816M2X/ref=ppx_yo_dt_b_asin_title_o06_s01?ie=UTF8&psc=1



Testing the resistance of just the solar panel facing up and down.
Both readings are in kilo-ohms.

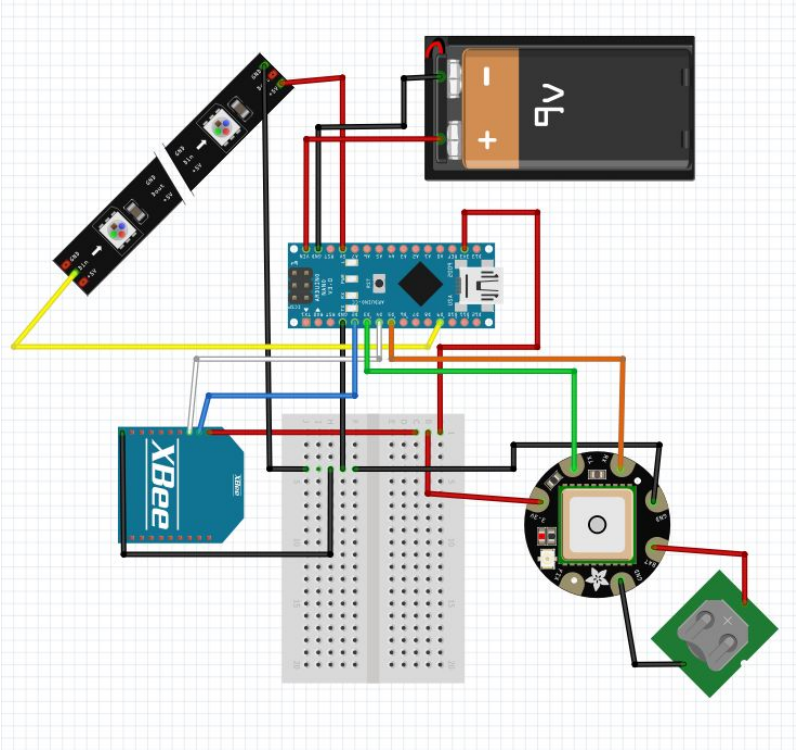
I did have mild success in getting one of my rechargeable 9V to pull a charge by adding a diode in line to the solar panel and connecting it directly to a battery[33]. Deploying this method into the revision 2 beacon made me nervous as there was no regulation IC device to stop the battery from overcharging. Sure, it would prevent backdraw but I did not want to risk frying what of the device I had built or ruining a battery. Power was put on the side burner in favor of completing cloud to device communication. Currently, the devices in use are operating with a 9v battery without solar. Future correction of this includes better sourcing the solar panels or finding a way to include multiple ones on the lid of the beacon.

[33] https://www.instructables.com/id/9V-Solar-Battery-Charger/

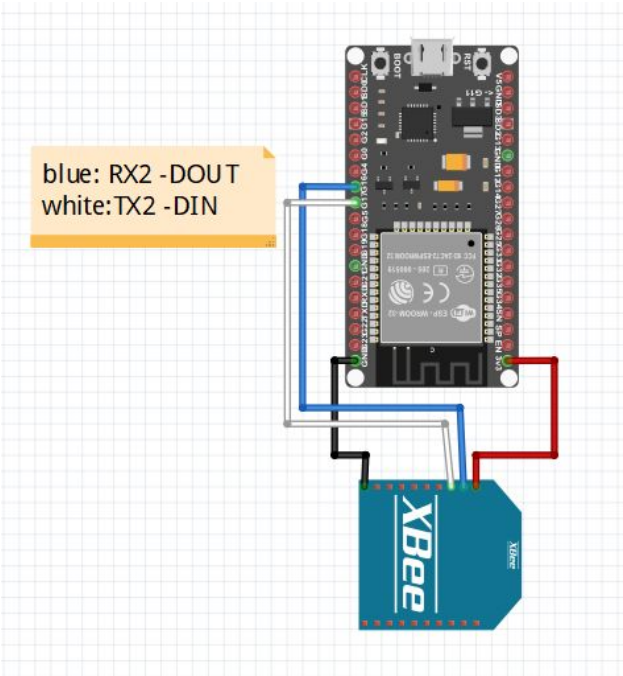
Current Products

Wiring Diagram

Higher resolution copies of both diagrams can be found in the repository included with this document in the “diagrams” directory.
Beacon rev2:

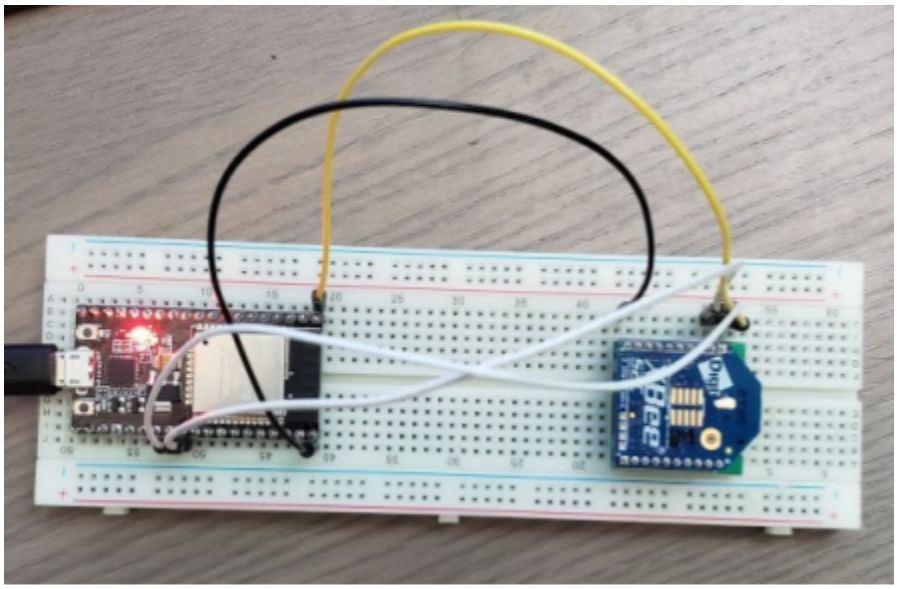


Beacon Coordinator rev2:



Note: there are several different pinouts for the varying ESP32 boards depending on manufacturer. The one above is assumed to be the nodemcu ESP32S variety.

The Beacon Coordinator



The beacon coordinator.

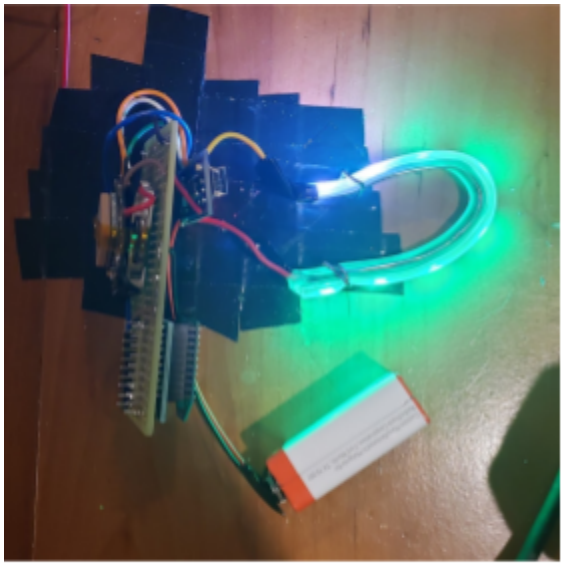
The beacon coordinator is a surprisingly simple device consisting of only an ESP32S and an XBee. The current code for this is Esp32-lwmqtt.ino located in the “beacon coordinator code/beaconcoordinatorv2” directory.

The next steps for this device include sending the GPS data up the line to the cloud for processing and use.

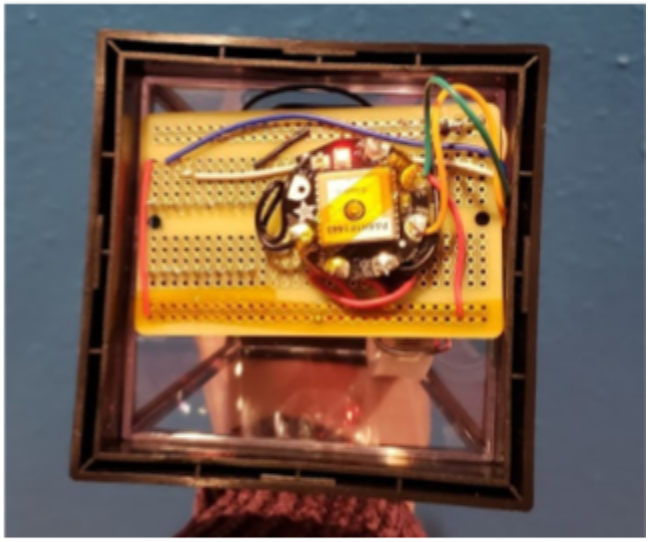
The Beacons

Beacon internals revision 1.

Beacon internals revision 2.



The beacon internal components.



Top view of beacon internals in housing.

Side view of closed beacon.

Rainy day waterproof test.

The beacons themselves are included in the images above. While not all initial goals for their final design have been met, they do successfully find and report their GPS information, do take commands from the beacon coordinator and work.

The current code for them is located in the “beacon code” directory as beaconv4.ino.

Next steps for the beacons include building better housing, downsizing and consolidating internals and cracking the solar issue.

Bill of Materials

Included are the parts used for this project, where they were sourced and approximate prices per device excluding shipping and tax.

item with link	**price in USD**	notes
Arduino Nano	6.66	
XBee	20.05	
XBee breadboard adapter	5.99	
Flora GPS	39.95	
Neopixel Strip	2.16	price calculated for 9 neopixels/8 cm of strip with 1 m as 26.95
Coin Battery Holder	1.95	

Coin Battery	0.60	
9V Battery Holder	1.20	
9V Battery	1.37	
perma-proto board 1/2 size	4.50	the ones I used were freebies on a big order but...
lawn light housing	6.00	
Cost per beacon rev 2:	90.43	

item with link	price in USD
ESP32S	8.00
XBee	20.05
XBee breadboard adapter	5.99
full sized breadboard	3.00
Cost of beacon coordinator rev 2:	37.03

Important Links

The current product and code repository can be found in the GitHub repository found here.
A video demonstration of the process, individual components and the current beacons in action can be found here.
A video containing a demonstration of the current beacons in action can be found here.
The web portal can be found here. As previously mentioned, I have not configured certificates for the hosting through Google Storage and Google Domains and am holding off on doing such pending movement from the powers that be.

Limitations and Issues Encountered

Several difficulties were encountered over the course of this project but good progress was made.

Programming challenges included using multiple Software Serial connections within the beacon software. This was somewhat tricky to make sure the correct serial connection was

being listened to as both the GPS module and XBee require serial communication. It took longer to debug than anticipated.

Pulling power from the solar panels has so far been unfruitful and somewhat frustrating. This was discussed earlier in the power section of this document but is worth mentioning here. After many tries, solar harvesting was relegated to a secondary goal in favor of getting devices up, functional and running.

Currently, because of Covid-19, the supply chain is a hot mess. Additional GPS components from Adafruit were unable to be ordered at this time as they are only filling essential orders and have converted their shop to be medical device manufacturers[34]. While they have successfully rolled over much of their ordering capacity to DigiKey, finding more of some of the parts I needed to expand and explore some of my stretch tasks, like fitting the Nano-XBee chips into speciality sockets and the clock crystal for programming the ATMega directly, were postponed due to delayed shipping.

The shutdown of the high school I work for, sudden transfer of all of the classes I teach to an online format and cancellation of all of the summer programs I run did also create a tremendous disruption in my original ambitious timetable and an overarching general malaise.

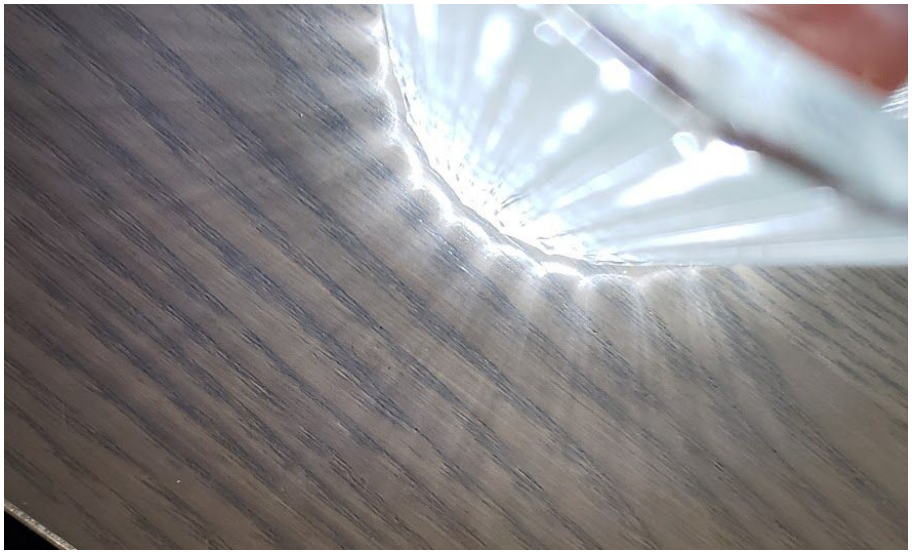
Reflection, Next Steps and Future Expansion

I will be continuing this project. I am honestly having fun with it and it has been an excellent avenue and structure for me to flex some areas I have historically had less experience in and needed a reason to grow, namely web development, use of cloud services and solar harvesting.

There are still some general issues to iron out with these prototype designs with regards to power consumption and harvesting. While I am disappointed that I was unable to get this component going, I do think that, knowing what I know now, I have a better handle on what options are available and how to best approach them. I have had issues getting my Nanos to wake from the sleep cycle on GPS or XBee communications but it is part of the next step in refining the beacon internals.

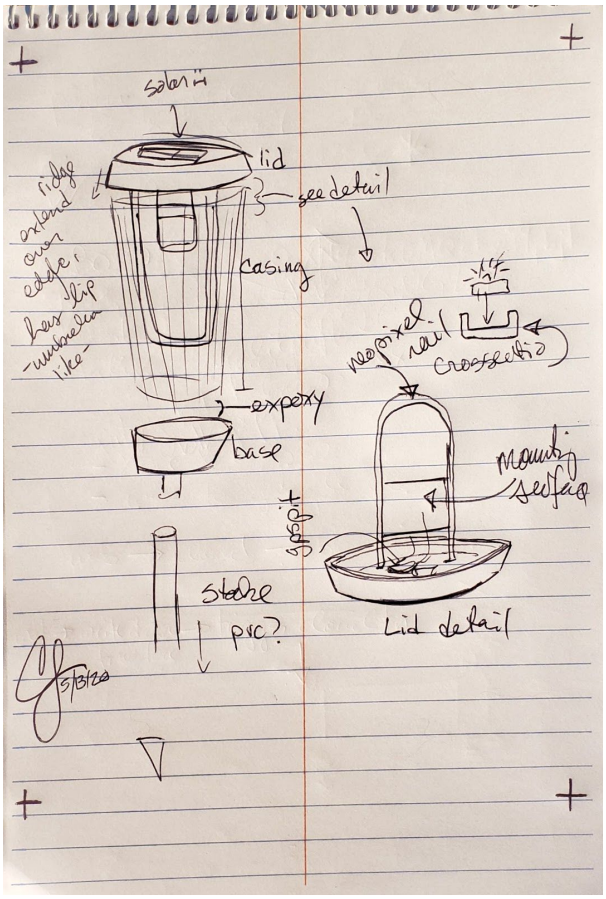
Future expansion of this project includes the possibility of manufacturing the beacon housing in-house using the provided 3D models as a starting point as well as creating additional beacons so that more specific locations and gravesites can be pinpointed. My initial draft of these 3D models looked remarkably similar to the prefabricated lawn lights. During the testing process, the beacon internals were held in place by a regular kitchenware glass to prevent the Neopixel strip from bending on the table. During this time, I noticed the appealing refraction effect caused by the texture of the glass.

[34] (Edit: they seem to have started shipping again today: May 14.)



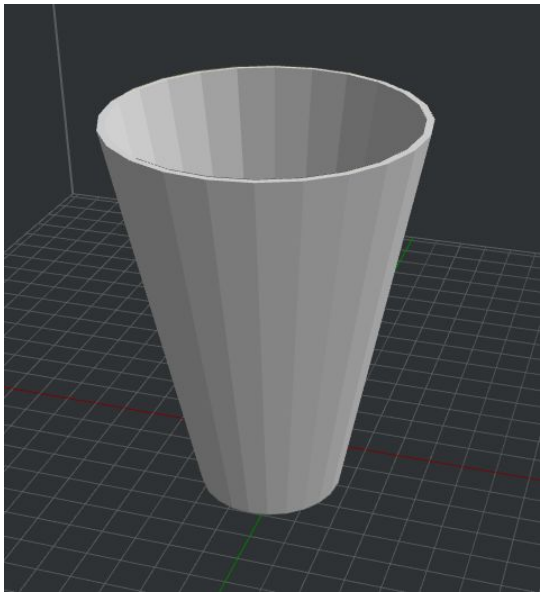
This led me down a different design path for the beacon housing. Instead of a cube base, I would make a more cylindrical base and try to flute the outside edges. This also works better for fitting in the Neopixel U. The lid of the beacon will have a lip around the edge that provides an overhang for the central part. This allows for a tighter seal and less chance of water incursion. On the bottom of the lid is a cavity to allow mounting of the GPS module. Part of the lid[35] is a frame to mount the Neopixel strip on and mounting space for the device internals.As the internals are not fully finalized yet, this current model assumes the Nano and XBee are in this socket connection device that has been ordered but not yet arrived. The central housing component needs to be clear or frosted to allow light to escape. I am considering making use of a vacuum former and polycarbonate sheeting to accomplish this[36]. The base component is fairly straightforward provinging coupling from the beacon casing to the supporting stake. A sketch of this can be seen below.

[35] Or possibly a secondary part of the lid that clicks into the bottom of the lid
[36] I've needed an excuse to set one up at work.



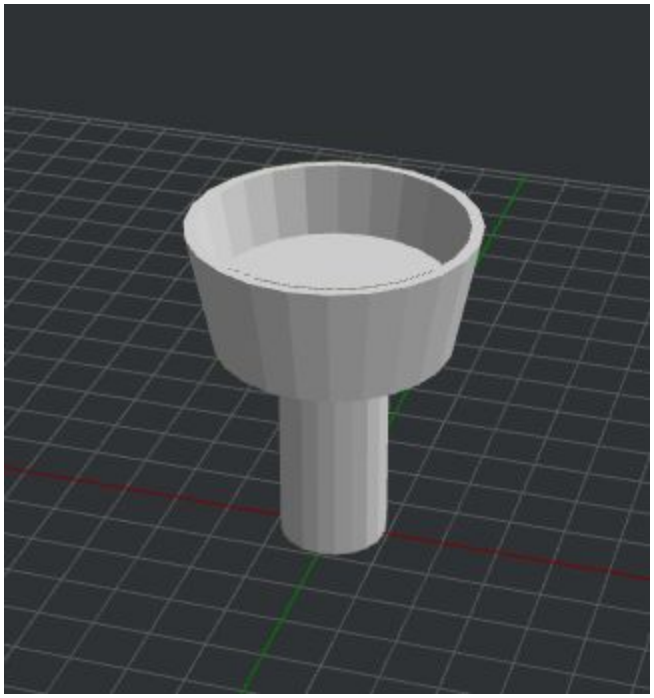
The initial 3D models for this are included in the “3D models” directory of the provided code repository included with this submission. Image captures of each model and their intended assembly are included below.

The case is the casing for the center of the beacon housing the lights.



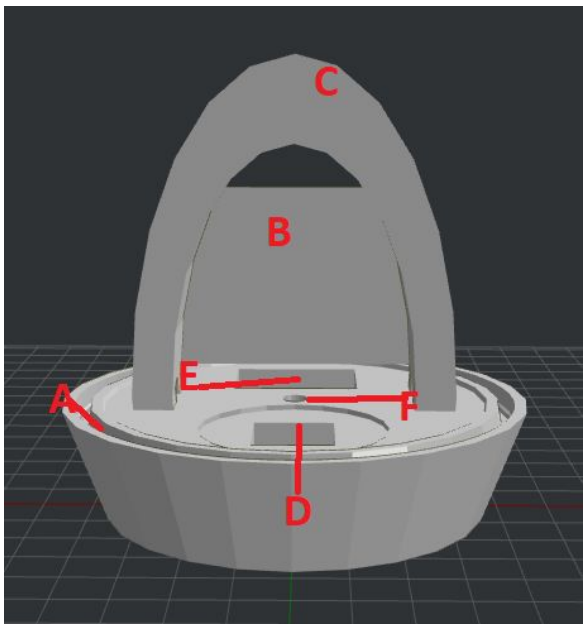
case.stl

The base is the bottom component of the beacon that fits into the stake. The stake is assumed to be a round piece of pipe with an internal diameter of ¾ inches. The case fits into the top of it and is held in place by epoxy.



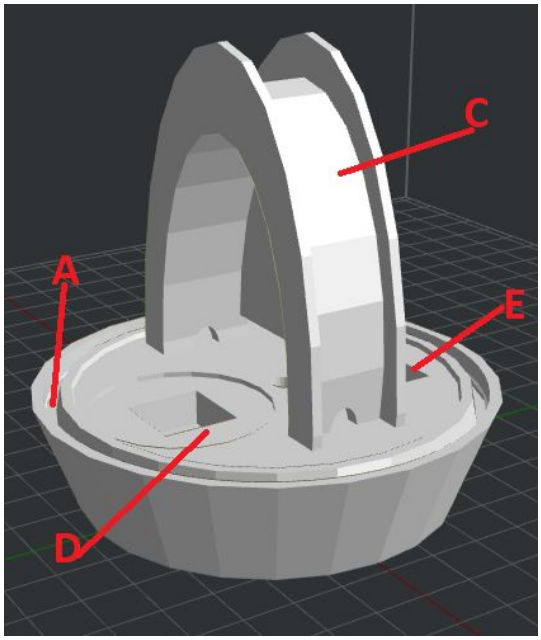
base.stl

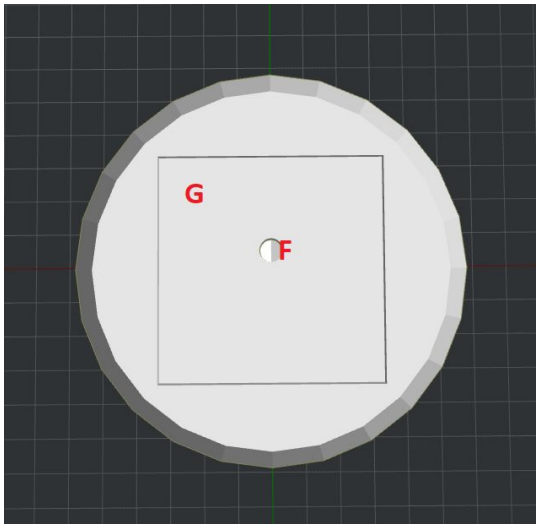
The lid has two included revisions. Observable in the front view is the internal lip to seal the lid to the top of the beacon (A), the mounting space for the Nano and XBee (B), the rail for the Neopixel strip(C), the slot for the Flora GPS (D), a slot for a battery (currently assumed 9V)(E) and a hole down the middle for wiring (F) to the eventual top of the lid where the solar panel is mounted(G). The two revisions differ in that the second breaks apart the model into two smaller models to aid in assembly.



(left) lidA.stl, front view

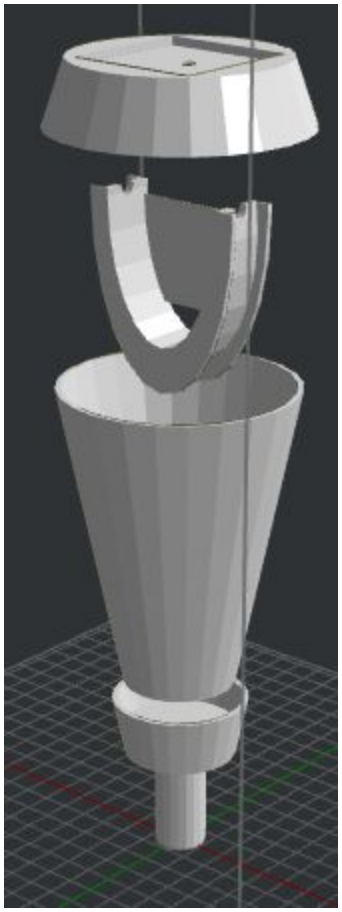
(right) lidA.stl, angle view





lidA.stl, bottom view with solar panel inset(G) and wiring channel(F).

These models are, as of this moment, untested and will undoubtedly need some revision. Both lid possibilities included with lidA being the single piece opton and lidB and rail&mount being the piecemeal option.


Break apart assembly view of the model

This prototype design was done using TinkerCAD.

Currently, the cloud to device communication is one way: from map click to the beacon coordinator. The individual beacons do send their GPS coordinates and name to the beacon coordinator on first GPS fix and subsequent changes in GPS location. This data is not yet processed or forwarded to the cloud. There is a Firestore collection in place for future addition of beacon information including name and geoposition. This is a high priority next task as this functionality will provide the information base that will allow the triggered cloud function to find the *best* beacon to send a command to light.

Additionally, I would like to look into options for removing the central coordinator device and phasing away from XBees. This could possibly include the individual beacons having onboard SMS capabilities and linking directly back to the Google Cloud hub. However, this would incur additional per use cost and could be a limiting factor as the organization managing this cemetery is a non-profit and has limited resources.

If this is not possible, negotiation for placement of the coordinator device within the neighboring community radio station[37] or at CARC[38] will commence. Based on previous professional interactions with CARC, I am optimistic that they will be open to the prospect of hosting a web connected coordinator.

Safety and security of the beacons themselves is also on the todo list. The beacons do currently send an update to the coordinator in the event of a GPS movement. This means, in the event of a beacon being moved arbitrarily, it can recover its position. Should the beacon be removed from the premises, its last known GPS transmission could narrow down the direction of its exit. Addition of a tilt sensor or motion sensor that then reports in when abnormal events occur are also possible security additions.

Aspirational offshoots of this project include an art piece reflecting on the curious modern condition we find ourselves in where passed individual's social media and web presence are active past their demise and function as a sort of memorial. There is also the potential of parts of this project to go to general use. Once I have more polish to it, I am considering approaching the New Orleans cemetery preservation group Save Our Cemeteries to see if the web portal could be of any use to their work in preserving and documenting.

Acknowledgements

A thanks to some of my friends and colleagues who helped me with this or pointed me in a good direction: Danielle Richmond for spitballing and workshopping initial project ideas, Colby Christopher for pointing the Google Maps API out to me and acting as a soldering "helping hands", John Richardson for helping find someone to pay for Google Cloud services in the future so that this project might continue, Sam Levy for a crash primer on solar circuit needs via e-mail and Taylor Chustz for hanging out at a distance with me in the cemetery and running the camera during the field test.

[37] http://whyr.org/
[38] http://www.cacrc.com/ (I love them. They are great folks.)