Project documentation

DISTANCE BUTTON

Autumn Semester / 2018 / E5IoT

Vojtěch Rychnovský (au603522), Cuong Nguyen (au607527)

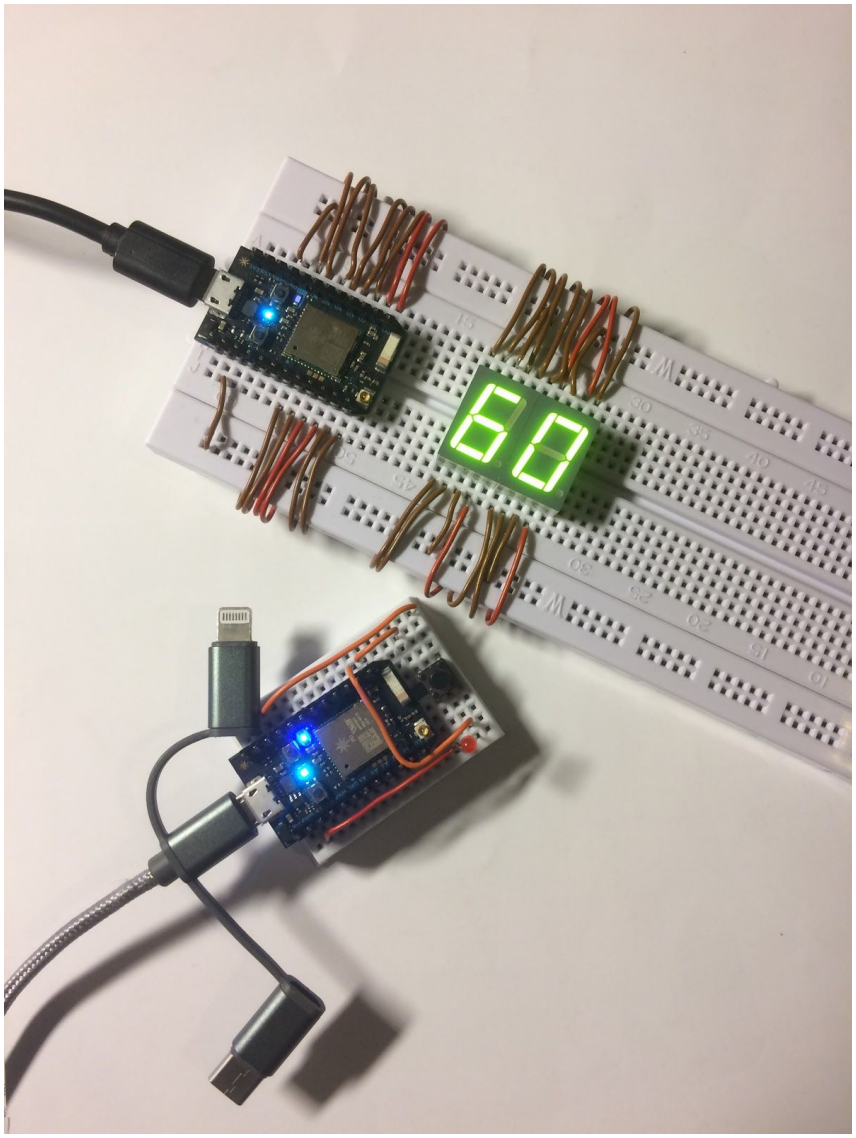


Table of contents:

Our Team & Work Report

Vojtěch Rychnovský

Tasks and responsibilities:

- Working on Project documentation
- Analysis and implementation of Google Maps API
- Distance calculation

Cuong Nguyen

Tasks and responsibilities:

- Working on Project documentation
- Hardware parts
- Digits display

Introduction

Project annotation

The goal of this project is to create an IoT device connecting two users by sharing their location and showing the walking time between them.

Project repository

Public repository with all source codes and documentation can be found on:
https://github.com/rychnovsky/E5IoT_project

Project Description

The core part of our project is a pair of devices (stations) connected to the Internet. Each of the stations will belong to one of the users.

These two stations are located on different places (for example one at home, second at school). One station is called client station and the other can be called master station. The client station contains one Photon, a LED and a simple button. The master station contains another Photon and a numeric display.

When the button is pressed, the Client Photon fetches its location from Google Maps Geo-Location API and publishes this event into Particle Cloud. The Master Photon is subscribed to this event and receives this location. When this action is triggered, it fetches its own location also from Google Geo-Location API.

These two locations are used to calculate a distance between these two places and time necessary to walk this distance. For the most accurate calculation of walking time the Google API is used again. This walking time will be displayed on the master station.

Requirement Analysis

Stakeholders analysis

As this project is developed as a school project, there are specific types of stakeholders. There is an unordered list of all of them.

- **School / course** - especially requirements listed in the Blackboard
- **Possible users of this system**
    - There are two **friends** living in the same town who visit each other very often. The one who is leaving the house would like to inform the other friend, that he is coming and how long it will take.
    - **Children** can easily inform their **parents** that they are leaving school and if they don't come home in expected time, parents know, that there is a problem.
    - A couple living far away from each other can let each other know, how long it would take to walk to the partner's place.

Requirements

In this section we will summarize all of the requirements coming from different stakeholders. Features of our system correspond to these requirements.

Non-functional requirements

Req 1: The system will be built on the Particle Photon platform
Req 2: The platform shall have Wifi connectivity
Req 3: The platform shall have digital or analog I/O connected sensors and actuators
Req 4: The software and hardware design must be shared in a public Github repository. All relevant project files will be uploaded there.
    a. Hardware documentation, schematics, datasheets and pcb layouts are to be uploaded in pdf format
    b. Software files are to be uploaded in raw source code format, e.g. .C, CPP, .h, .py, etc.

Functional requirements

Req 1: System will consist of two stations.

Req 2: These stations will be connected wirelessly.

Req 3: When a button on one of the stations is pressed, system will show walking time between Photons on the other one.

Req 4: The locations of the Photons will be fetched from Google Maps API.

Req 5: The walking time will be calculated by Google Maps API with path (streets, buildings..) in mind. In other words, this time will not be only straight line between these points.

Req 6: The walking time will be shown on a display, which will be a part of the master station.

Req 7: The display will be capable of showing at least 2 digits, that means that range of the values on display will be from 0 to 99.

Req 8: The system will indicate an error, when the distance is invalid to show (the time is longer than 99 minutes or any error).

Possible opportunities

Our system has many opportunities for future features and development. These feature are out the scope of this project, but there is a possibility of their future implementation.

1) **Smartphone notifications**
   Our system can send a push notification to user's mobile phone, when someone is sharing location with him.
2) **Connection to a mobile app**
   The information shown on the station's display can be shared to user's mobile phone for example using Blynk app
3) **Continuous location sharing**
   Each station requires internet connection to share its location with the other station. If we use another type of Photon which is able to use cellular data, we could implement continuous location sharing. That means, that the walking time shown on the display would be synchronized with the real position of the device / user.

System Design

System overview

We have split our system to two part:

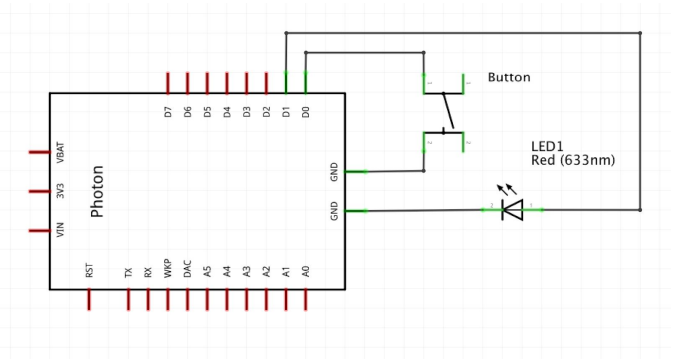
1. **Client station:** this Photon shares its location, it starts the communication
2. **Master station:** this Photon receives remote location, it invokes distance calculation and it shows the result on display

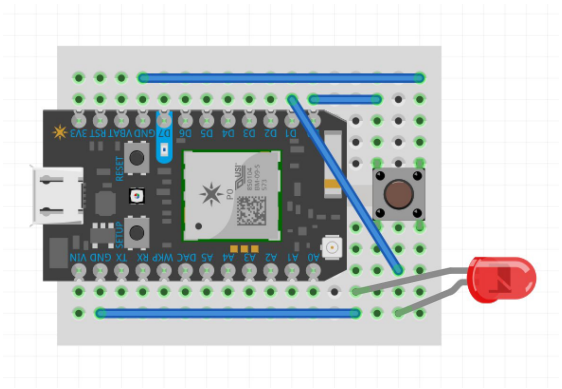
Client Station Design

This station consists of following components:

- 1 x Particle Photon
- 1 x Mini Pushbutton Switch
- 1 x LED, Red

(diagrams in better resolution are in the git repository)

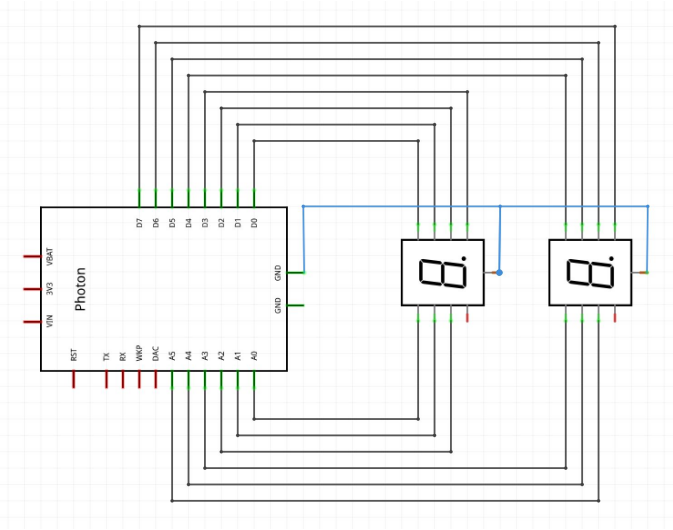


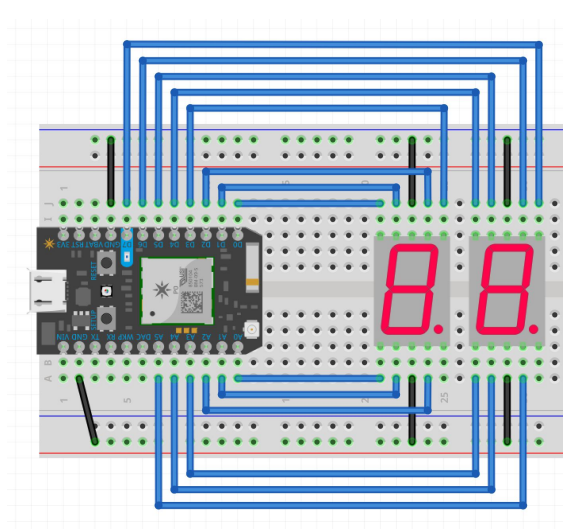


Master Station Design

This station consist of following components:

- 1 x Particle Photon
- 2 x Seven Segment Display





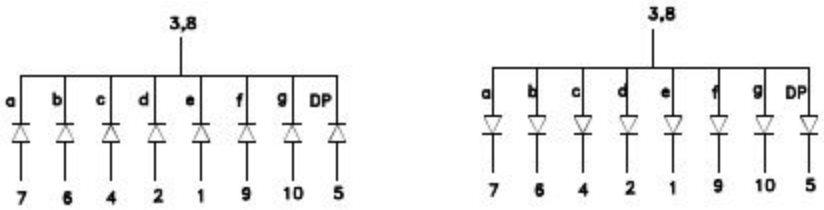
Hardware Design

Seven-segment Display

In this project, we are going to display the time taken to move from this station to the other on the seven-segment displays. Because the requirements are displaying at least two digits (from 00 to 99), we will use two seven-segment displays to be able to fulfill the requirements.

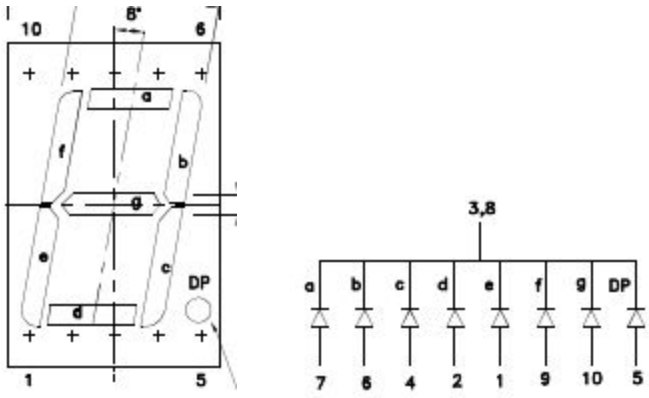
Seven-segment displays are often used in digital clocks, basic calculators or any devices which display numeric information. The reason we chose it to work on, not any other type of display, is that it is simple but powerful enough for us to achieve our goal. Also, it is in our lab in the campus in Aarhus so we can pick it up and work with it immediately and conveniently.

There are two types of single digit seven-segment display: Cathode and Anode. The main difference between these two is how the LEDs are positioned:

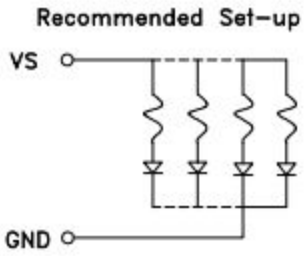


As you can see from the above pictures, the left one depicts the Cathode type and the right one depicts the Anode type. In the Cathode display, the pins 3 and 8 should be grounded meanwhile these pins are connected to the voltage supply in Anode display. For the purpose of working on this particular project, we choose the Cathode display.

The cathode seven-segment display has ten pins, five on top and five on the bottom. The display has three horizontal segments (top, middle and bottom) and two vertical segments on both sides. To make it work, we need to ground two middle pins on each side (three and eight). And then depending on the number which we want to display, we can supply voltage to that particular pin. Each segment in the display corresponds to a letter from a to g. Moreover, we have a pin number 5 refers to DP which is the dot, it is used to display decimal numbers. In the master station, we expect to receive the time taken to travel in only integer, so there is no need to use the decimal point in our project.



For instance, in order to display number four, firstly we need to indicate which segments to turn on. According to the image above, we want f, g, b, c segments to be lighted up and these four segments are assigned to pins 9, 10, 6, 4 respectively. So we connect these four pins to four output pins on the photon, then set those outputs to be high. Also, we ground two middle pins (three and eight) on top and bottom side, photon needs to be grounded. After that, number four is appeared on our display if everything is working properly.



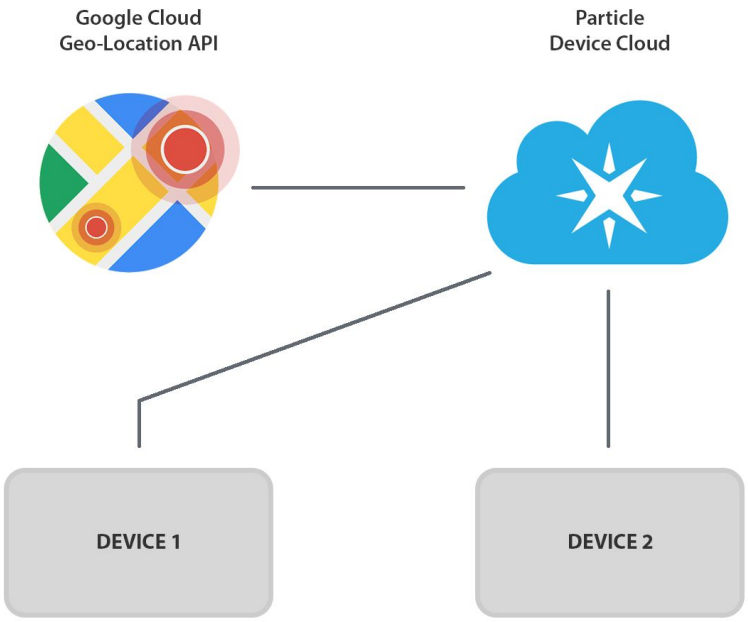
In the datasheet of our seven-segment display, the image above depicts the recommended set-up for the display, in which each LED should be put in series with its own resistor. In practical, when we work with the display, everything is functioning properly as expected without resistors so we do not place those in our breadboard.

Software Design

Sequence diagram & Lifecycle

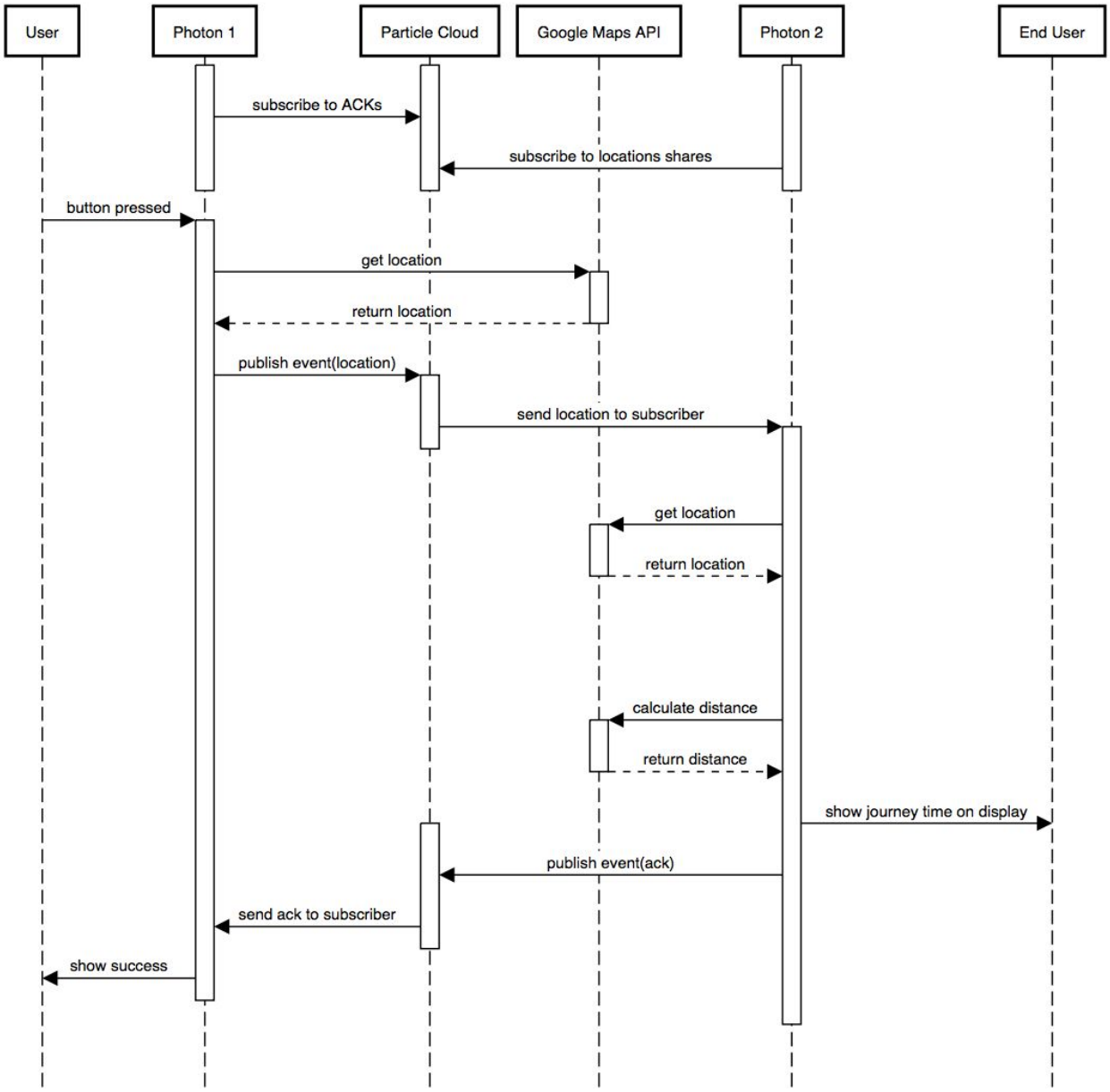
The two stations in our system communicate through Particle Cloud, they fetch location from Google Maps API and they calculate distance using another Google Maps API. Each of this part of the program are described in upcoming sections.

High level view on the system:



Usage of this external APIs means, that in the process between button pressing and presenting the result on the second side, there is numerous of asynchronous requests.

The app flow and communication between two stations in our system, Google APIs and Particle Cloud is shown by following sequence diagram:



To provide users better control over this relatively complicated sequence, both of the stations communicate their states with user by blinking their LED diodes.

- **Client Photon**
    - after button is pressed, red LED is on for 1,5 sec, location is requested
    - after location is received, red LED is on for 1 sec
    - after master acknowledges and finishes the process, blue integrated LED is on for 2 sec
- **Master Photon**
    - on startup, minus-minus blinks three times to make sure display is connected properly

- on remote location received, minus-minus is shown, location is requested
- after location is received, display is cleared, calculation is requested
- after calculation is received, the result is shown

Location fetching

There are many ways to get location of IoT device. Probably the most accurate method is using GPS module, but our Photon doesn't have one and we rejected using an external one. Our stations will be probably placed inside building which also talks against using GPS. The location can be also obtained by cellular towers, but this location is not very accurate and also we use WiFi as internet connection instead of cellular. Here comes the last and in our case the most suitable option - to obtain the location of our stations by WiFi access points. It means we use Photon's WiFi module as a location sensor. There is an API provided by Google called ***Geolocation API***[1]. Google has enormous database of WiFi access points and their locations, because they collect information about movement of Android phones, and they provide this locations to developers.

We have used ***Google maps device locator library***[2] provided by Particle IoT team. This library uses this API by sending all visible WiFi networks to API endpoint and receiving location in form of object with latitude, longitude and accuracy. Usage of this method is also easen by Photon itself, because there is a support for this API in Particle Console.

When it comes to accuracy, this API provides surprisingly good results. Our testing shows that we can detect movement of Photon devices even inside buildings. This means that accuracy of this method is more than suitable for our use case.

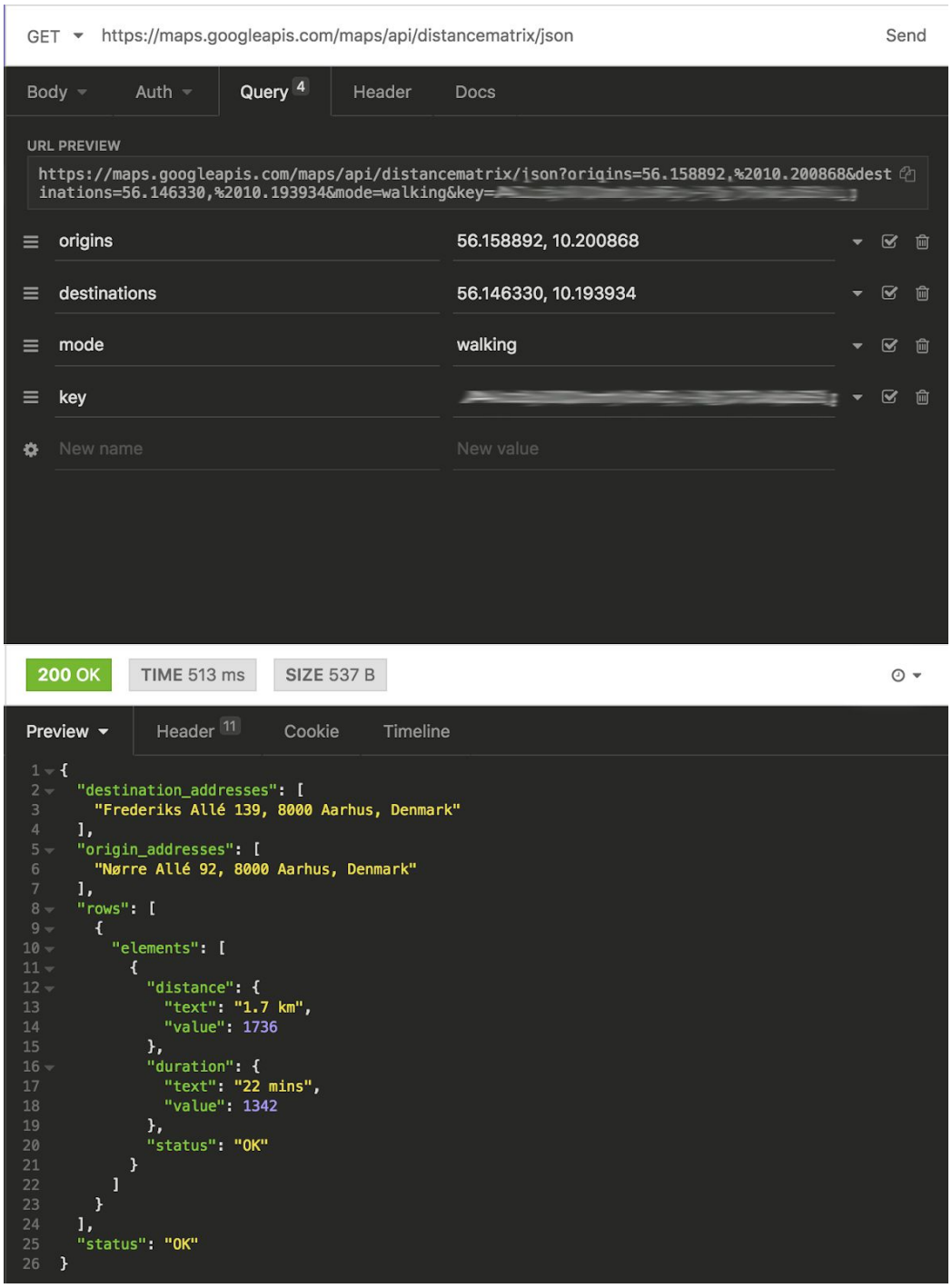
Distance calculation

For distance calculation we could use very simple functions: calculate distance between two coordinates and divide this value by average walking speed. But this implementation doesn't take in mind that we can't walk straight between two places, we have to go around buildings and other obstacles in our way. We also wanted to show more precise number, based on ascending and descending during the way. Therefore for calculating distance between two points, our app uses ***Google Maps Distance Matrix API***[3]. In this API we can't easily setup origin and destination by GPS coordinates and also choose mode of the transportation - in current implementation we use “walking” mode, but simply by this query parameter we can for example use also public transport data.

---

[1] *Google Geolocation API:* https://developers.google.com/maps/documentation/geolocation/
[2] *Google maps device locator library:* https://github.com/particle-iot/google-maps-device-locator
[3] *Google Maps Distance Matrix API:* https://developers.google.com/maps/documentation/distance-matrix/

Distance Matrix API is available through HTTP request including *origin*, *destination*, API *key* and in our case *mode* as an optional parameter. Here is an example of calling this API:



Communication between Photons

For passing variables between two stations we use Particle Cloud, concretely Particle.Publish() and Particle.Subscribe() methods. By Publish() function, we can send only one variable, but we need to share

two variables - latitude and longitude. To accomplish this, we have introduced our very simple protocol to send two values - we send them in a string separated by semicolon, example: “latitude;longitude”.

Here is an example of published event containing location values fetched from Google API:

location-shared	56.139271;10.187274	vojtech-photon-2	11/20/18 at 11:28:05 pm

Testing and Verification

Our system has two parts, that needs to be tested. In particular, it is the display and the location fetching. Our testing consists of two stages. During the first one, we have tested these two parts separately, then we put them together and we tested them as one complex system during the stage two.

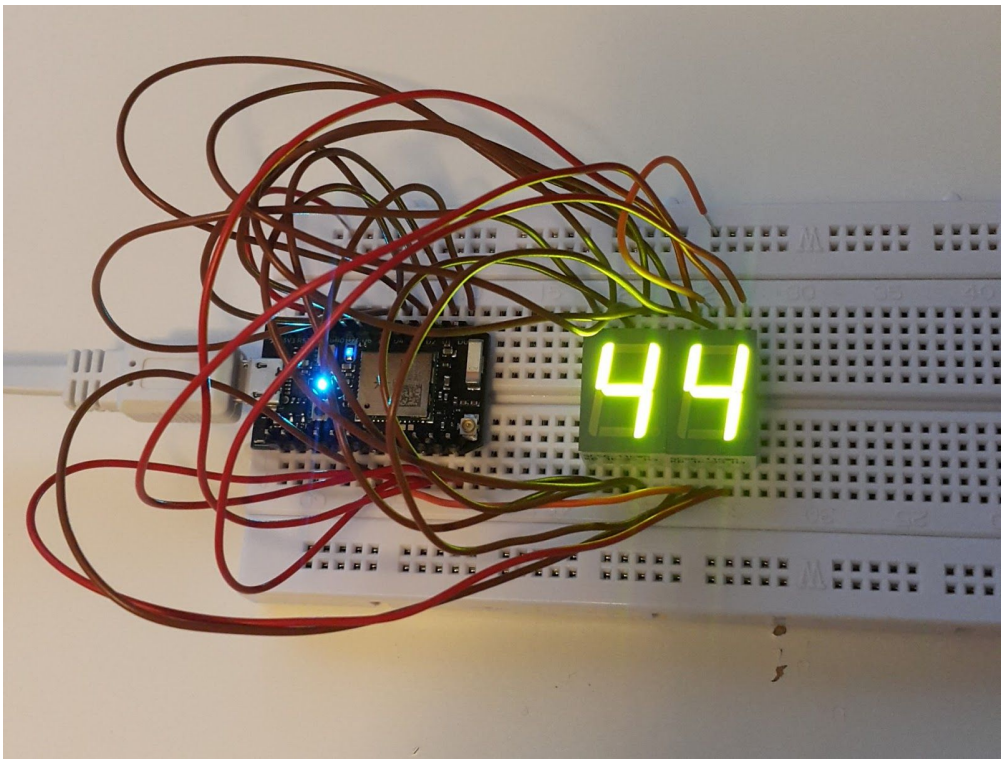
Test 1: Display testing

We have built our own display from seven-segment digits and we need to make sure, that we can show right numbers. The display consists from two digits, therefore it can display numbers between 0 to 99.

HOW: by make a test program that will count from 0 to 99

RESULT:

For example, we want to display number 44:



One problem I realized when doing the test is that there are two types of seven-segment display: Anode and Cathode (described in section “Hardware design” above). At first, I did not know that there are two types so I picked randomly one in the lab to work and it turned out to be the Anode type. I need to work with the Cathode type and this is the main problem I faced in the first attempt in this test.

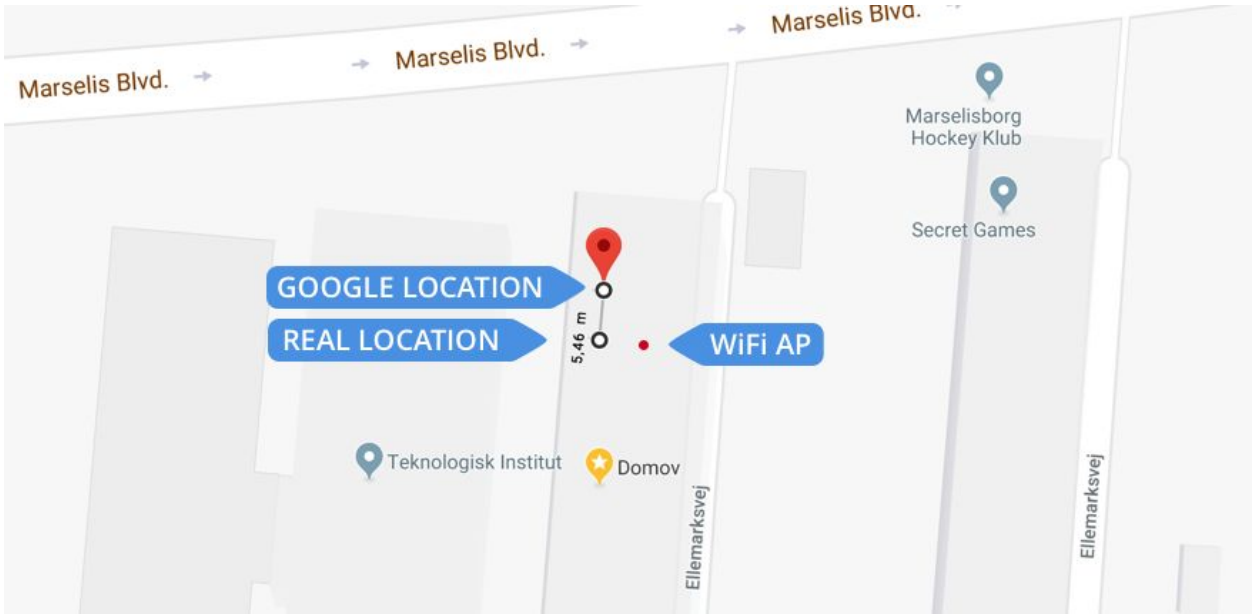
Test 2: Location fetching testing

Integration to Google map API is one of the most important services in our system. We need to know, how precise is the location we can get from the API.

HOW: by fetching the photon's location on different places and comparing them to a real position

TEST: This measurement shows difference between real location and location obtained by Google. Photon was placed in a student dorm inside a room with WiFi access point located in the corridor approximately 4 meters from photon location, this means there was a wall between Photon device and WiFi AP. This environment is very close to real use case situation.

RESULT: Difference between Photon and real location was around 5 meters. This precision is more than enough for calculating distance between two places.



Test 3: Interoperability testing

After we have tested the display and the location fetching, we need to make sure, that our program works together.

HOW: we will set two stations in different places in the campus and try them as a whole complex system

RESULT: By testing the system, we can see, that if we have the stations next to each other, the number shown on the display is equal to 00. If we put stations to different places, we can also see the right number. To ease the testing we have put a constant into our code to move one Photon from its actual location:

```
lon += 0.03;
lat += 0.02;
```

By this, we can have Photons on the same place and still get more interesting results.

Testing conclusion

By all these tests we have verified that our system works well.

We faced some challenges with the connection between Photon and the WiFi. This is something we can hardly fixed. Also, there can be some issues with Google API when we try to fetch the location. The API knows the location, returns the right information, but after a while, it returns error “Google doesn't know your location”. It means, that between two API calls, Google just forgot location of our IP address. We suggest further testing of this location API before considering this product as ready for production.

Conclusion

All in all, throughout the duration of the course, a project has been completed and it fulfills all the requirements to an acceptable degree. We actualized the idea of making a distance button as described in project annotation section in the very beginning. In another way, our project vision is achieved by the fact that we can show the walking time between client station and master station.

Because of the lack of experience working with hardware (in this course it is the Particle Photon in particular), we focused more on software part of the IoT world. But in the same time we gained some essential knowledge about hardware and how to implement IoT techniques in the real-world use cases, hence improve our personal vision.

To be able to make the project really usable in real life, we need to add more advanced features for example listed in Possible Opportunities section. We can also make it a little bit neater and looking more compact. But for the scope of this course, we think the project meets the expectation.