Raspberry Location Tracking

Darius Birescu, Dan Blaga June, 2018

1. **Repository**

The project codebase, history, diagrams can be found at the following git repository:

<https://github.com/dariusbirescu/RaspberryLocationTracking>

1. **User requirements**
2. The system should compute the remaining distance to the destination
3. The system should be able to track a persons’ location
4. The system should periodically save the data of the previous and current user locations
5. The system should operate in any vehicle
6. The system should notify the user if the current location has not changed in a period of 5 minutes
7. The system should be able to accept input parameters regarding the next location to be visited
8. **System overview**

The system overview is depicted in Figure 1:

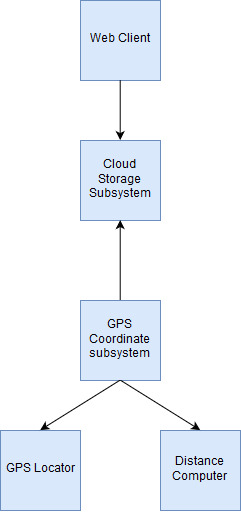


Figure 1: System overview diagram

GPS coordinate subsystem: contains the GPS-related modules (getting the coordinates, computing the remaining travel distance), as well as the coordinates that the user will input into our device.

GPS Locator: Gets the current coordinates from the ipinfo API. Can be extended with a physical module.

Distance computer: Will compute the remaining distance to destination, using the haversine distance formula. It is also responsible with notifying the user by displaying a message on the LCD screen should the distance not decrease after an interval of 5 minutes.

Cloud storage: The location data is stored on Firebase.

Web Client: Allows basic interpretation of the stored coordinates, as the user will be able to see the paths that they have taken, and the points where the system notified the user that they were standing still.

1. **Circuit design**

Figure 2 illustrates the circuit design:

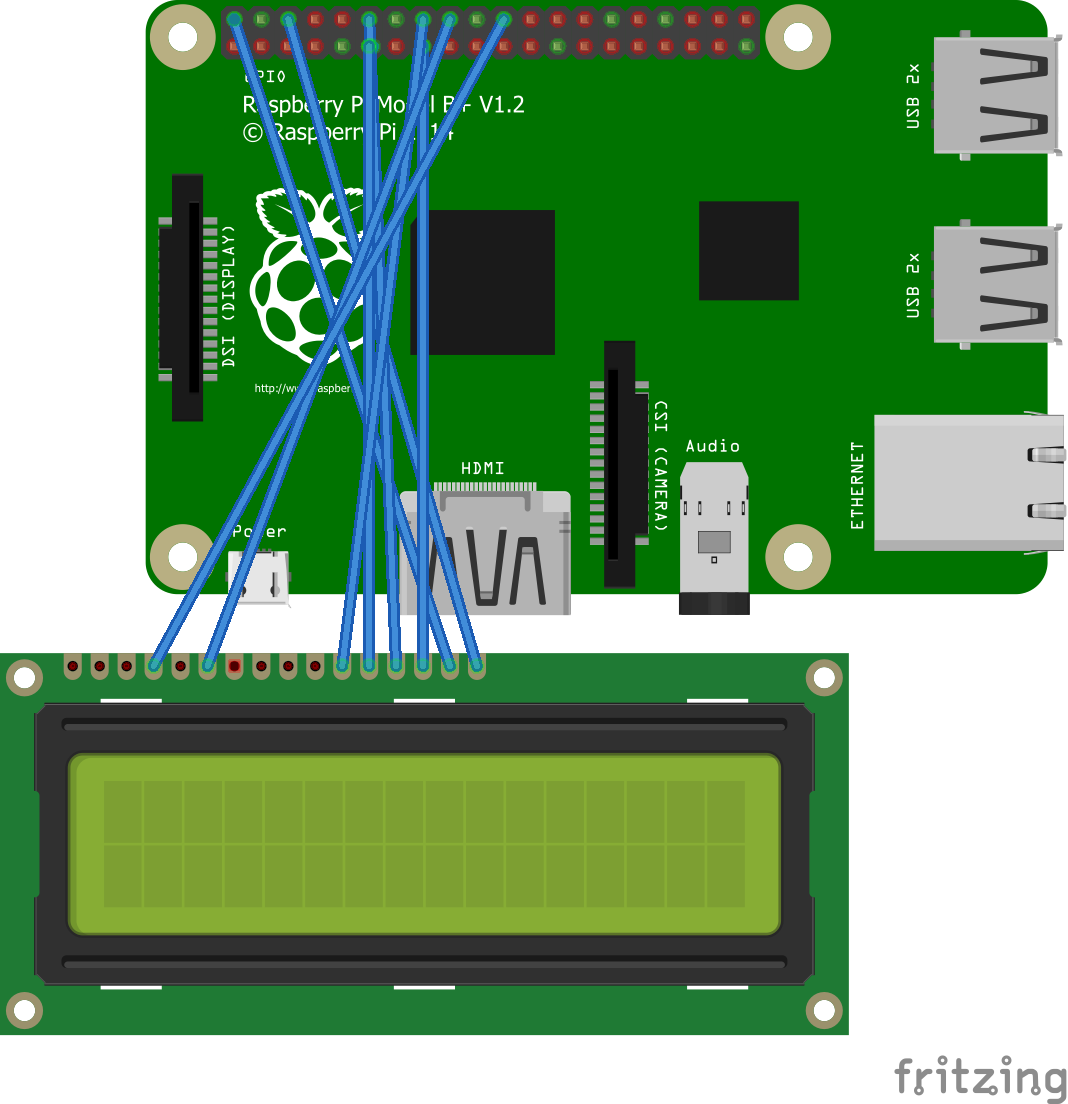


Figure 2: Fritzing diagram

Raspberry B+ was the board we had access to, and for our purposes, it is sufficient. We have connected a single 16x2 LCD shield, so that the user might see the distance remaining to the destination, and if they are not moving. The LCD shield is wired directly to the GPIO pins of the Raspberry.

1. **Software design**

The software and data flow can be seen in Figure 3:

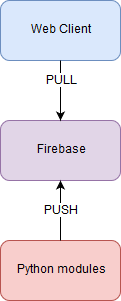


Figure 3: Software modules involved

* 1. **Python modules**

wifitest.py: reads the coordinates from the ipinfo API, and provider the latitude, longitude, region, location, organisation that owns the server to which the raspberry is connected and the IP.

coordextractor.py: gets the information from wifitest, and returns just the latitude and longitude. This module was implemented as a „middleman”, so that we could change the coordinate provider and not have that impact the rest of the system.

destinationService.py: this module is responsible for reading the destination coordinates, initialising the LCD screen and printing the requierd messages, sending the command to publish the coordinates to Firebase, computing the remaining distance to the destination and checking if the user is getting closer to the destination.

publisher.py: uploads the latitude and longitude of the current location and the destination and the remaining distance to be traveled to Firebase. It also gets from Firebase the remaining distance so that it can compare it with the current computed distance in destinationService.

* 1. **Firebase**

Firebase is a PaaS (Platform as a Service), offering developers a quick list of functionalities supported by a traditional backend. In this case, it is used as a repository for the data that we upload to it.

* 1. **Web client**

Our User-Interaction is realised by the means of an Angular web application. The user can see through this UI application what is the distance the tracking device still must cover in order to reach its destination. The Angular application also uses Angular Material Components for a pleasant design. The UI simply makes the necessary calls to the Firebase cloud storage to fetch the data needed to be displayed inside the UI.

This is the method through which the UI gets the data from the cloud:

getMessages() {

return this.http.get(this.myURL).map(res=> res.\_body);

}

1. **Results and further work**

The current version of the project supports the following functionalities:

* Getting the current GPS coordinates based on the network IP
* Storing data to Firebase
* Client implementation for retrieving the data from Firebase (web client)

The following list of extensions and improvements was identified for the future:

* Extend client implementation to support Android app
* Extend client interaction with the device to allow multiple ways to introduce the destination
* Integrate a hardware GPS module for more precise GPS tracking
* Enhance the UI design of the Web Client

1. **References**

* Draw IO [last seen: May 2018]: <https://www.draw.io/>
* Fritzing [last seen: May 2018]: fritzing.org/
* Firebase Database [last seen: May 2018]: https://firebase.google.com/