Reprogrammable wireless ad-hoc client-server solution for geofence applications

Autonomous and Cooperative Systems

January 4, 2019

Authors

Andreas Koj - koja@student.chalmers.se

Martins Eglitis - eglitis@student.chalmers.se

Advisors

Oliver Harms - harms@chalmers.se

Contents

1 Transceiver

1.1 Hardware

1.1.1 Setup

At the beginning of the project, we were provided with the following components:

- Arduino Uno - the microcontroller board. Key specifications:
  - Microcontroller - ATMega328P
  - Voltage - 5V
  - Flash - 32KB
  - SRAM - 2KB
  - EEPROM - 1KB
  - Frequency - 16MHz
- SparkFun GPS Logger Shield - the GPS module. Key specifications:
  - Standard - NMEA 0183
  - Voltage - 3V3 (regulated)
  - Communication - UART, serial asynchronous
- nRF24L01+ - the RF module. Key specifications:
  - Standard - 2.4GHz ISM
  - Voltage - 3V3 (regulated)
  - Communication - SPI, serial synchronous

At first, we soldered the GPS headers and then wired the RF module to the default Arduino SPI pins. Later, we installed the 3V3 battery in the GPS shield which allows to store the last fix resulting in much lower fix times. For example, when we tested the beacon at ground level (Örebrogatan 4, 418 71 Göteborg), the TTF was very inconsistent - from 20 minutes to not fixing at all. When we tested it at higher distance from the ground, 3rd floor (Rännvägen 6, 412 58 Göteborg) we noticed that the results were much better and the TTF was as low as 20 seconds. The battery was present in both test cases.

It is important to remember to use the serial switch on the GPS shield and set it to SW mode. It allowed us to have greater flexibility and map the pins more convenient. Although both the GPS shield and the RF

Figure 1.1: Arduino Uno

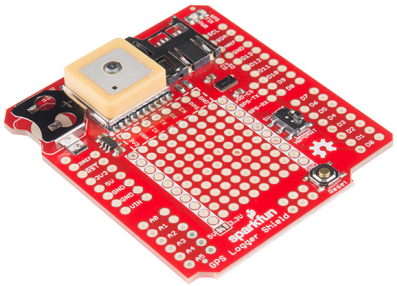

Figure 1.2: GPS shield

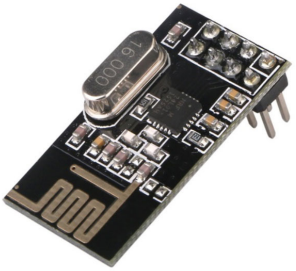

Figure 1.3: RF24 module

module have voltage regulators, supplying the required voltage from Arduino results in less power wasted as heat and increases the battery lifetime. On embedded projects both heat and battery lifetime are crucial parameters, which is why we recommend using the 3V3 voltage and reduce the number of unnecessary components, for example, the GPS logger shield LEDs (around 40mAh saved).

The available hardware resources for HMI were scarce, if any at all. The only form of having some form of hardware feedback was the builtin LED on the Arduino. However, because the RF module was using the pin D13 on Arduino for the software SPI, the functionality of the LED was disabled. We decided to mount a pushbutton and an LED on the prototype board of the GPS shield. The pushbutton provided us with a simple input and the LED allowed us to output the results in a physical manner.

1.1.2 Architecure

The communication architecture of a transceiver is as follows:

- The GPS shield communicates with the satellites using the NMEA 0183 standard.
- Arduino communicates with the GPS shield using the serial, asynchronous UART.
- The RF module communicates with the RF modules on beacons using the 2.4GHz ISM.
- Arduino communicates with the RF module using the serial, synchronous SPI.
- Arduino communicates with another computer using the serial, asynchronous UART.

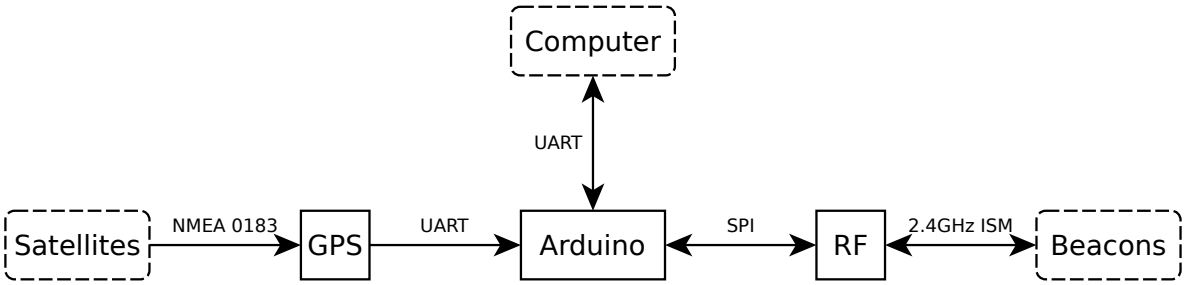

Figure 1.4: Communication architecture

The solid rectangular structures in figure 1.4 are the main building blocks of the transceiver. The fine-dashed rounded rectangular structures are the endpoints. Endpoints encapsulate and hide unnecessary details, for example, how the NMEA strings are constructed within the satellite, so we can concentrate on the base structures. The same goes for beacons - we are only interested in the structure of the messages sent. This modular approach is perfectly suited for this project where parts of the system (transceiver and beacons) can be built independently of each other, relying only on the endpoints.

1.2 Software

1.2.1 Setup

The first step to acquire the source code is to clone the repository from one of the available two repositories https://git.sitilge.id.lv/sitilge/chalmers-autonomous-and-cooperative-systems or https://github.com/sitilge/chalmers-autonomous-and-cooperative-systems. The code responsible for the transceiver is located under src/transceiver.

The second step is to install the Arduino IDE. In our opinion, development using the Arduino IDE is cumbersome, yet is the easiest way to search and install Arduino libraries. The libraries then, for most Linux systems, can be located under ~/Arduino/libraries directory. There are a few 3rd party libraries required for the transceiver, some of which will also be used in beacons:

- SPI.h - a library for communication with SPI devices.
- RF24.h - the driver for the RF24 transceiver.
- nRF24L01.h - an optimized fork of the RF24.h library.
- TinyGPS++.h - the driver for the GPS receiver.

- SoftwareSerial.h - a library for software based serial communication.

The project relies on three types of libraries - the system libraries, the 3rd party libraries, and the local libraries. We structured our code by keeping similar functions and variables seperated. It allows the codebase to be clear and intuitive. This modularity also allows to reuse the libraries in other parts of the project. For example, the local PIP libraries used in the transceiver can also be used in beacons.

Once the source code and the libraries have been acquired, the code can be uploaded to the Arduino. One way of doing so is using the Arduino IDE, however, there are some drawbacks. It may happen that the selected port as well as the board has changed due unknown reasons. Also, when burning another board, for example, a beacon which is based on Arduino Micro, the information about the board is stored on the device. It is then retrieved by the Arduino IDE and automatically selected. Uploading without changing the board yields errors. A better alternative is to compile, upload and watch the serial output using the standard GNU/Linux tools. The Arduino IDE basically is a frontend of the Arduino prototyping platform SDK. Although, the documentation is not up-to-date and some of the parts are missing, it is possible to provide command line arguments for the arduino program. Some of the arguments are the upload target, the port, and the board. The serial output can be read by using the screen command. Moreover, the whole procedure can be automated by implementing a simple shell script, which is included in the project.

1.2.2 Communication protocol

In our opinion, the communcation protocol between the transceiver and beacons was the most difficult part. Because the message size of RF24 is only 32B, it only makes it possible to send very limited amount of data. The structures of incoming and outgoing vertices are as follows:

```
//vertice incoming struct
struct verticeIncoming {
        uint8_t id;
        uint8_t order;
        uint8_t count;
        float x;
        float y;
};

//vertice outgoing struct
```

```
struct verticeOutgoing {
        float x;
        float y;
};
```

The size of uint8_t on the particular ATMega328P is 1 byte and the the size of float is 4 bytes. It means that the payload size of an incoming vertice is 11 bytes and the payload size of an outgoing message is 8 bytes. The approach of sending structures is far more superior than sending strings:

- The payload size is greatly reduced. Sending the same logical information means sending each digit as a character, the values have to be seperated by seperator, for example, a comma, and, the string has to end with a null in order to be a fully qualified C string.
- The complexity and time is greatly reduced. In order to send the payload as a string, the structure has to be converted to a string in the sender and then parsed in the receiver. It adds code on both sides, resulting in extra complexity, time, and makes the system more error prone.
- The security is improved. When sending the payload as a string is that it read by humans easily. On the other hand, when sending the payload as a structure, it has to be casted to the proper struct thus hiding the obvious information.

Because of the 32B limitation, a custom protocol had to be designed. During the preperation phase, we came up with some of the main problems for the protocol. The protocol had to take into account all the specifics of our adhoc system. The main problems and their solutions are:

- The protocol has to support numerous geofences and vertices. The user might want to change the number of geofences or vertices. We solved it by introducing variable values instead of static ones, for example, in loops and conditions. We also gave each beacon a unique identifier in order to distinguish it from others.
- Messages from beacons may not arrive in order. It affects both messages from one particular beacon or from different ones. It is important for the PIP algorithms that the vertices are provided in the correct order. Before passing the messages to PIP, we order them using a comparison function.
- Messages may not arrive at all. If, for example, a beacon goes offline and the geofence information is incomplete in the transceiver. The transceiver is unaware of what it should do next, for example, how long it should wait for the next message or how many false messages it must receive before claiming the beacon unreachable. We introduced TTL (time to live) variable which checks if a geofence has reached a threshold of false messages and then resets it.

- Geofences and vertices in their respective parent structures are not ordered. Beacons may be assigned different IDs, for example, 1, 7, or 122. Because we have very limited hardware resources, beacon with ID of 122 might not be possible to have if we decided to have a structure assigned to that particular ID. We solved it by introducing fixed size structures and conditional statements to skip empty structures.

1.2.3 Point in polygon algorithms

Point in polygon problem is a problem that tries to determine whether the given point lies inside or outside the given polygomn. Although there are more than two algorhitms available, we decided to use the two most popular ones - the ray casting algorithm and the winding number algorithm.

The ray casting algorithm pseudo code is:

```
count = 0

foreach (edge in polygon)
        if (ray_intersects(p, edge))
                count++

if count % 2 == 0
        return 0

return 1
```

ray_intersects is the function that checks if a ray that starts at point p crosses the edge. If the count of such intersects is even then the point is outside the polygon, otherwise the point is inside the polygon.

The winding number algorithm pseudo code is:

```
count = 0

foreach (edge in polygon)
        if (ray_intersects(p, edge, upward))
                if (point_lies(p, edge, left))
                        count++
```

```
        else if (ray_intersects(p, edge, downward))
                if (point_lies(p, edge, right))
                        count--

if count == 0
        return 0

return 1
```

ray_intersects and point_lies functions determine wether there is a valid upward right or downward left intersect of the x coordinate of the point p. The count is then incremented or decremented respectively. If the count of such intersects is 0 then the point is outside the polygon, otherwise the point is inside the polygon.

2 Related work

2.1 Beacons

2.1.1 Hardware

Our case study is based on two companies - Estimote and Kontakt. Both of them share the very similar features, with some exceptions, for example, Estimote has beacons with ambient light sensors while Kontakt offers encrypted communication and ID shuffling. We will look at the key specifications, which are:

1. Estimote offers Bluetooth beacons for various use cases such as asset tracking, proximity, presence verification, robotics, etc.
2. Depending on the beacon, they can be programmed using different programming languages - Objective-C, Swift, Java, Kotlin, and Javascript.
3. Additional built-in sensors - accelometer, temperature, magnetometer, ambient light, pressure, and tech - NFC, LED, GPIO, EEPROM, HDMI, push button.
4. Supports iBeacon and Eddystone protocols, which are the industry standards.

When compared to our beacon:

1. Although our beacon does not rely on Bluetooth, we rely on 2.4GHz radio which is the underlying technology of Bluetooth. Our solution suffers from not being recognized by modern mobile devices which have Bluetooth module built-in, but it has advantages in terms of application - it can be extended to use any protocol, such as, Wi-Fi, ZigBee, etc. Also, our adhoc solution can be adjusted to fit specific needs.

2. Our code is mainly written in C/C++. The use of low level languages is intentional since we have to deal with memory and these languages (especially C) are designed to handle such cases. In our opinion, it takes somewhat more effor to write low level code but we have more control over the hardware. Usage of higher level languages in this case might mean that they are interacting with an API, which is then responsible for taking care of lower level tasks.

3. Because of the nature of our adhoc system, there are no extra built-in sensors. However, because Arduino allows great customization and expansion by having numerous GPIO, power inputs, LED and EEPROM by default, most of the given can be implemented easily. Most of the tech (LED, GPIO, EEPROM, push button) are provided by the Arduino or mounted on the GPS shield prototyping space.

4. The main drawback of not using one of the industry standards is that our beacons are not able to communicate with popular transceiver devices, for example, smartphones. Therefore we are limited to use our transceiver only and adding more our transceivers requires more effort then finding a device that is iBeacon or Eddystone compatible.

2.1.2 Communication protocols

We will focus on studying the two most widely used protocols - iBeacon and Eddystone and compare them with the one we are using in our transceiver. Both of these protocols have the following features:

1. Based on Bluetooth Low Energy (BLE).

2. Only broadcasts.

3. Transmits a universally unique identifier (UUID).

4. Some of the settings can be changed - transmit rate, power, UUID variables.

However, there are some differences between them. iBeacon protocol:

1. Developed by Apple, released in June 2013.

2. Proprietary standard.

3. One frame.
4. Limited to 20 geofences.

Eddystone protocol:

1. Developed by Google, released in July 2015.
2. Open source standard.
3. Four different frames (UID, EID, TLM, URL).
4. Unlimited number of geofences.
5. Provides telemetry data, for example, battery level, sensor data.
6. Google launched the Google beacon platform, very useful to application developers.

When comparing our protocol with the two used in industry, we see that we share only 3. point of the shared features - we transmit the UUID. Our protocol differs in the other points. Instead of using BLE, we are using the underlying 2.4GHz radio. The beacon is able to send and receive data. No beacon settings can be changed from the transceiver because of the lack of advanced HMI.

Our protocol, similarly to iBeacon, sends out only one type of frame containing its UUID, the count of all vertices, the current vertice index, and lng and lat coordinates of the vertice. It is also capable of accepting one type of frame containing the lng and lat coordinates of the new vertice. Similarly to Eddystone, the number of geofences is basically unlimited if the memory is infinitely large.

2.2 Transceivers

As regards transceivers, we were not able to find a specific device designed for this objective. However, most modern cellphones that have Bluetooth modules installed can be considered as transceivers. For example, Android and iOS support creating and monitoring geofences.

3 Links

- https://store.arduino.cc/arduino-uno-rev3
- https://www.sparkfun.com/products/13750

- http://www.adh-tech.com.tw/?47,gp3906-tlp
- https://en.wikipedia.org/wiki/Point_in_polygon#Comparison
- https://en.wikipedia.org/wiki/Point_in_polygon#Comparison
- http://geomalgorithms.com/a03-_inclusion.html
- https://www.doogal.co.uk/polylines.php
- https://estimote.com/products/
- https://kontakt.io/ble-beacons-tags/
- https://developer.android.com/training/location/geofencing
- https://en.wikipedia.org/wiki/IBeacon
- https://en.wikipedia.org/wiki/Eddystone_(Google)

4 Evaluation

4.1 Performance

4 (Deltaparken)

Latitude,Longitude

57.687486,11.978673

57.687549,11.978993

57.687197,11.979371

57.687148,11.979044

8 (Restaurang)

Latitude,Longitude

57.688441,11.974555

57.68866,11.975211

57.688464,11.975413

57.688311,11.974954

57.688286,11.97498

57.688242,11.97485

57.688257,11.974832

57.688241,11.974785

16 (Library)

Latitude,Longitude

57.687969,11.975826

57.688173,11.976695

57.687709,11.977071

57.687723,11.977167

57.687448,11.977414

57.687425,11.977307

57.687273,11.977414

57.687146,11.976829

57.687112,11.976861

57.687075,11.976668

57.687241,11.976529

57.687218,11.976432

57.687316,11.976341

57.687301,11.976271

57.687405,11.976186

57.687425,11.976282

32 (Maskinhuset)

Latitude,Longitude

57.689316,11.976879

57.689495,11.977568

57.689272,11.977772

57.689206,11.977531

57.689137,11.9776

57.689263,11.978097

57.689075,11.978274

57.689097,11.978365

57.689211,11.97826

57.689283,11.978534

57.688624,11.979136

57.688549,11.978863

57.688647,11.978771

57.688621,11.978686

57.68843,11.978857

57.688237,11.978145

57.688188,11.97819

57.688144,11.978005

57.688182,11.977972

57.688157,11.977879

57.688376,11.97768

57.688448,11.977956

57.688575,11.977843

57.688512,11.977599

57.688613,11.977504

57.688606,11.977479

57.689009,11.977106

57.689034,11.97721

57.689117,11.977138

57.689074,11.976977

57.689165,11.976891

57.689191,11.97698