Aalto University
School of Electrical
Engineering

ELEC-D0301 Protopaja
2021

Project report

Project #01
Method for automatized positioning for luminaires.

Date: 31.8.2021

Team members
Igor Oinonen
Alexander Popov

Information page

Students
Igor Oinonen
Alexander Popov

Project manager
Igor Oinonen

Sponsoring Company
Ensto Building Systems Oy

Starting date
31.5.2021

Submitted date
31.8.2021

Tiivistelmä

Tässä projektissa keskitytään prototyypin kehittämiseen langatonta lamppukartoitustyökalua varten, joka mittaa RSSI:n avulla muiden lamppujen signaalien voimakkuutta ja käyttää kerättyjä tietoja koordinaattijoukon muodostamiseen lamppujen ryhmälle. Tämän lähestymistavan testaamiseksi käytämme RuuviTag-älymoduulin BLE-antureita, koska niissä on avoimen lähdekoodin laiteohjelmisto, jota voidaan muokata tarpeidemme mukaan. Mukautettu laiteohjelmisto on kirjoitettu C:llä ja ladattu RuuviTagiin langattomasti tai nRF52 devkitin kautta. Mukautetun laiteohjelmiston avulla RuuviTagit voivat mitata signaalin voimakkuutta muiden ryhmän tagien välillä. Isäntäkone voi sitten muodostaa yhteyden tagien ja kerätä signaalin voimakkuustietoja. Tietoja käytetään sitten XY-tason tagien suhteellisten koordinaattien laskemiseen. Lisäämällä parin RuuviTagin tunnetut koordinaatit voimme saada koordinaatit koko ryhmälle. Lopputuotteessa ainoa manuaalinen tehtävä on verrata näitä koordinaatteja pohjapiirroksessa oleviin koordinaatteihin epätarkkuuksien tarkentamiseen, mikä vähentää työtaakkaa merkittävästi.

Abstract

This project focuses on creating a prototype for a wireless lamp mapping tool that would use RSSI to measure signal strength to other lamps and use collected data to produce a set of coordinates for a group of lamps. To test this approach, we are using RuuviTag smart module's BLE sensors, as they have open-source firmware that could be edited for our needs. The custom firmware is written on C and is uploaded to RuuviTag over the air or via nRF52 devkit. With the custom firmware RuuviTags can measure signal strength between themselves and other tags in a group. The host pc can then connect to the tags and collect the signal strength data. The data is then used to calculate the relative coordinates of tags in XY plane. By adding actual coordinates of a couple of tags we can then obtain the coordinates for the whole group. In a final product the only manual task is comparing those coordinates to those available in a floorplan to adjust for inaccuracies, reducing the workload by a large margin.

Table of Contents

1. Introduction

Smart lighting solutions solve many important problems of the modern world. Not only they could provide better working condition for employees, but also increase energy efficiency. One of the problems, however, that comes with smart lighting is increased setup complexity. Each individual light source needs to be configured to know its approximate location in the room and be able to communicate with other light sources. Such configuration may take up to several hours for a worker to manually test each individual lamp and register its corresponding location in a lamp mapping software. Currently there are no readily available off-the-shelf automated method to accomplish this task, so a novel approach is required to speed up the process. The goal of our project is to pick a possible solution, evaluate its viability for such a task and develop a working proof of concept prototype.

2. Objective

The first part of the project is choosing main approach. One of the important factors in approach selection was keeping everything as simple as possible without requiring additional components or external hardware, thus relying solely on the Bluetooth module provided with the smart lamps. We also considered trying out other options but decided that our main approach is sufficient.

Our main approach is based on Received Signal Strength (RSSI) localization. Light sources are substituted by programmable Bluetooth enabled smart modules, these modules broadcast signals with known power and received signal is measured on each module, from this data approximate distance can be derived. Then this network of distances will be analyzed to construct human friendly graphical 2D representation of module locations.

3. Hardware used

- 10 RuuviTag wireless sensors
- RuuviTag Development Kit
- Bluetooth 4 enabled laptop (not as part of the project)

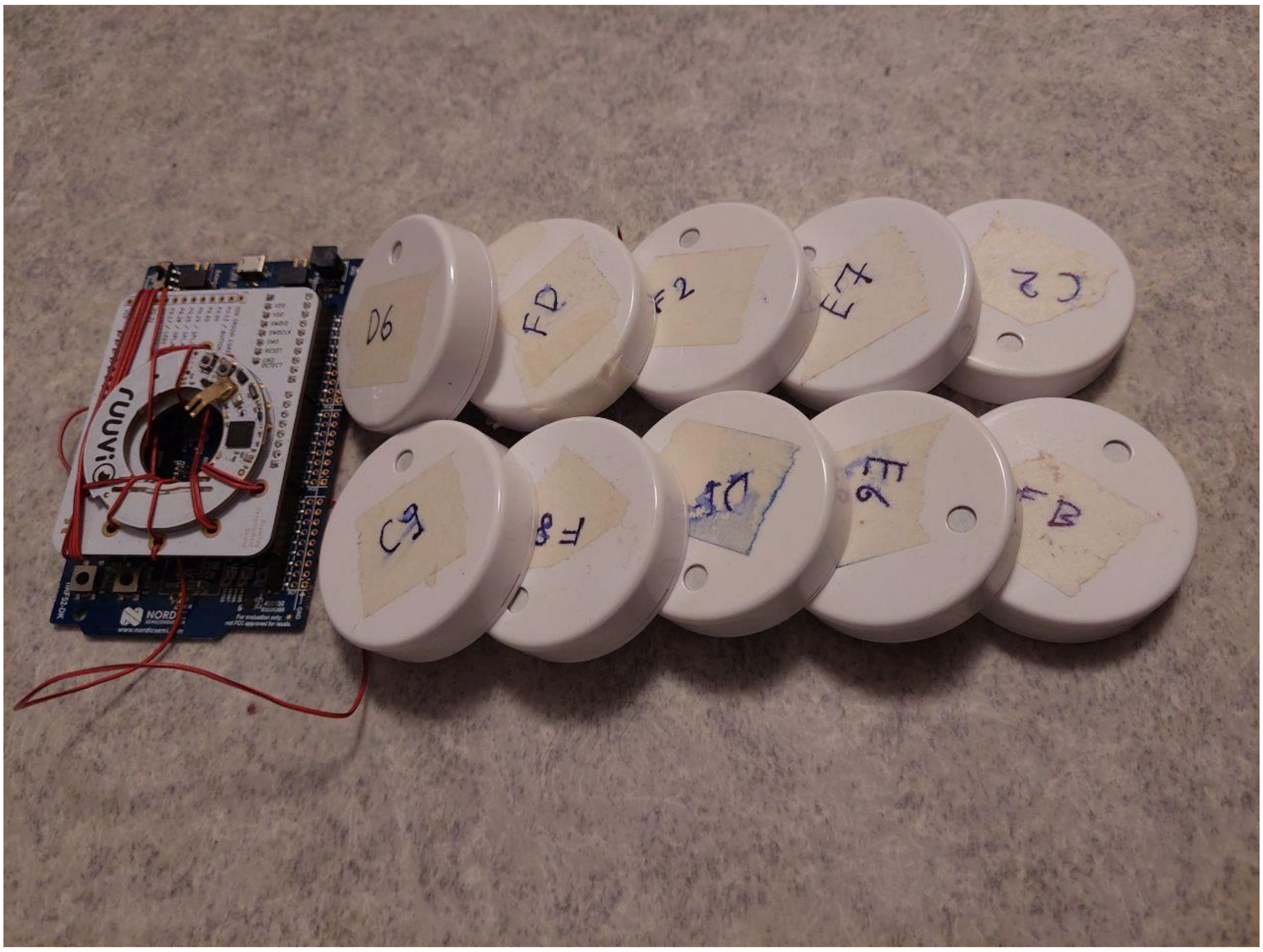

Figure 1 Hardware

4. Programming RuuviTags

The first step of the project is collection and aggregation of signal level data. By default, RuuviTags do not provide such data, so their firmware has to be customized to collect and store signal level data about every other device in its vicinity.

4.1. Setting up development environment

Firmware building and development toolset consisted of SEGGER Embedded Studio for development on devkit and several GNU build tools for packaging release packages. Overall setup was done according to official RuuviTag documentation, but it is important to note that the instructions did not work entirely in our case. In the step “Setting up the project” one should use firmware from our repository and the branch “ensto_proto”. Additionally, for building release packages on Windows additional steps must be performed if needed.

4.2. Setting up development board

One of the tags was removed from its case and placed on the RuuviTag shield included with the dev board. Dev board is connected to the development computer via USB interface, allowing direct development and debugging on the board via SEGGER Embedded Studio. Please note that the tag must be ziptied or otherwise fixed to the devboard to allow for reliable connection.

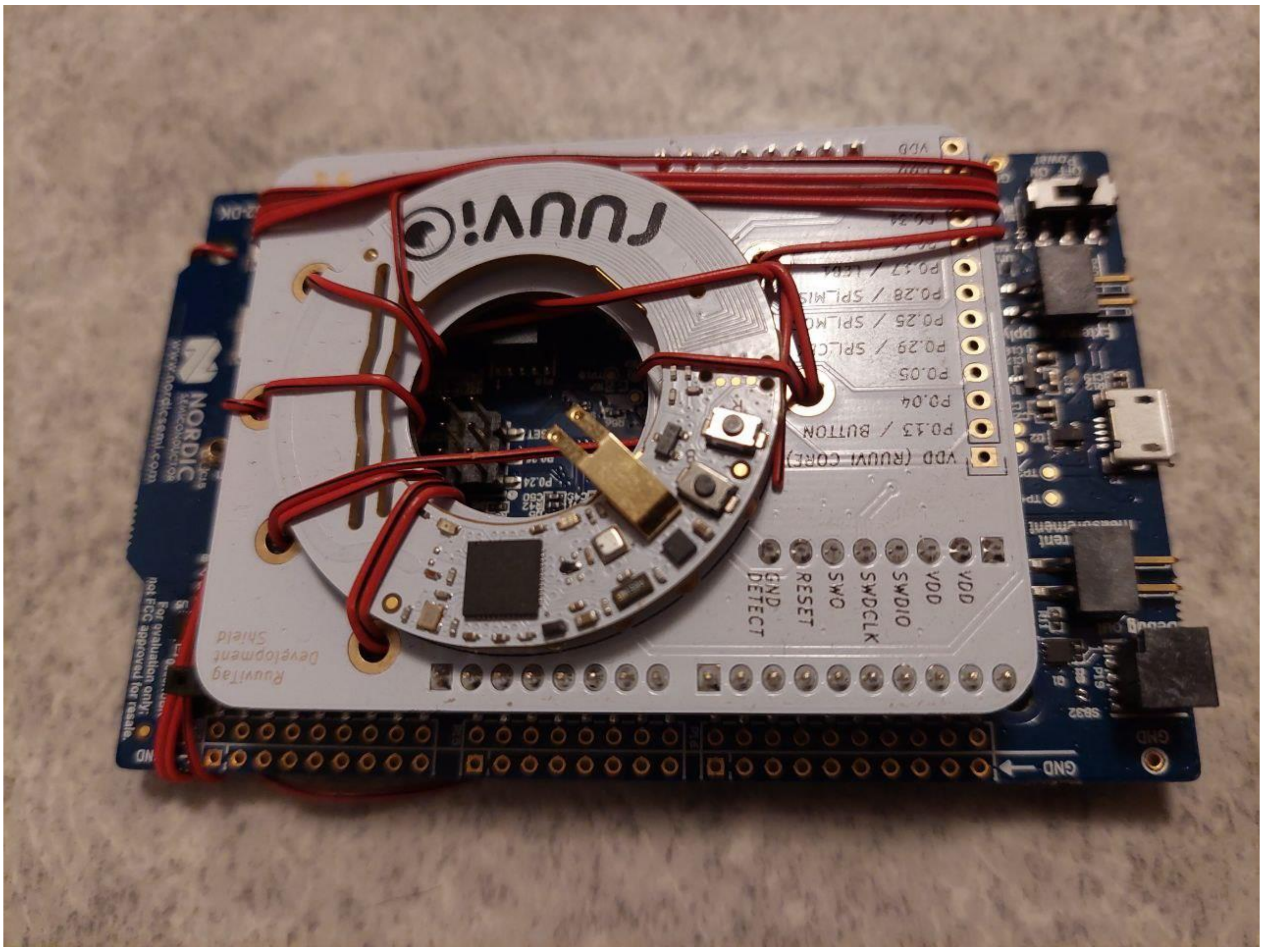

Figure 2 Improvised development board

4.3. Signal level measurements

By default, RuuviTags broadcast temperature, pressure and other information on the interval of couple of seconds. We scan for any advertisement packets in the “my_on_scan_isr” procedure which is defnined in main.c. Broadcasts that are not coming from other tags are filtered out and received signal strength level is stored in memory using “update_tag_info” procedure (also defined in main.c). If the signal level received was already measured before, new value is updated using a moving average (memory factor can be configured in app_ensto.h).

4.4. Possible improvements and known bugs

- Currently fixed amount of space is allocated in memory for storage of signal level information. If there are more tags than that, one should probably implement some priority algorithm to choose which tags are excluded from the storage.
- Broadcast listening code stops after random period of time, for this reason we must re-enable it every time host computer is connected via GATT service. This is implemented in handle_gatt_conncted method in app_comms.c.
- Instead of moving average one should probably just store best signal level reached, but this has proven to cause random inaccurate results, reasons unknown, but could be related to varying battery voltage levels of the tags or other similar issues.

5. Data acquisition

The host computer is receiving and parsing packets via simple python script. The script is using BLE GATT protocol to collect the data from each individual tag to a single system. The script was only tested on Windows platform but should work on Linux or macOS also. Script repeatedly asks tags. Here is an example of the script output data in format “receiver => sender/count/RSSI”, where received is the receiver tag of the broadcast signal, sender is first byte of sender tags MAC, count is count of measurements done and RSSI being the signal level value in hex, negated in dBm.

```
c2:b6:ed:fd:75:88 => c9/007c/1d e6/006c/23 f2/0082/21 f8/006b/0f fb/0045/1a df/000a/10 e7/0046/0e
c9:0c:d4:28:6f:32 => f8/0039/1b fb/0056/0f c2/006e/1f df/00ad/21 e6/0061/12 e7/0036/22 f2/0022/20
df:8f:d1:70:f5:8b => c9/007e/20 e6/0025/1f e7/0068/0f f2/0089/1c f8/0082/10 fb/004f/1c c2/004f/11
e6:e0:fa:e2:0f:b9 => e7/0078/28 fb/0065/0f c2/0096/22 df/0058/1e f2/003c/0e f8/0057/20 c9/0020/11
e7:bc:aa:46:fd:19 => c9/0062/21 e6/0087/29 f2/0052/24 f8/0032/0f c2/001a/0e df/004d/0f fb/003f/28
f2:d8:b7:af:70:e1 => c2/0049/23 df/0080/1c e6/007f/0e e7/0017/25 f8/0003/18 fb/0036/13 c9/0043/1f
f8:7e:71:a8:43:4e => c2/0032/10 df/005f/10 e6/0086/20 f2/0019/17 fb/0035/1c c9/0057/1a e7/001f/0f
fb:ef:a9:53:4b:43 => e6/0022/0f f2/0037/10 e7/002c/26 f8/002e/1b c9/0032/0e df/0011/1c c2/0014/1a
```

5.1. Converting signal level data into meter distance data

Signal levels are converted using following formula:

$$r = 10^{(b+s)/(10 \times c)}$$

Where ***r*** is the distance in meters, ***b*** is base signal level at a distance of 1 meter, ***s*** is the received signal strength and ***c*** is calibration constant.

We measured ***b*** to be around -69 dBm and guessed ***c*** to be around 3.77.

6. Constructing node space

At this stage we have a network of interconnected nodes with known distances to each other. The distances might not completely represent reality, but they have an important property that real distances, due to the fact that signal might be obstructed, but not amplified, are never greater than the observed ones.

We begin by randomly distributing nodes on a 2D plane. Then we apply gradient descent-based algorithm to minimize the difference of distances on the network and the 2D plane. As a result, with high probability most of the nodes will organize themselves into a shape, that closely resembles the spatial arrangement of the actual real-world nodes. The only problem is that this shape forms its own basis, and so the user interaction is needed to correctly apply linear transformations to the system.

Implementation in Golang can be found here. Example input datafile here.

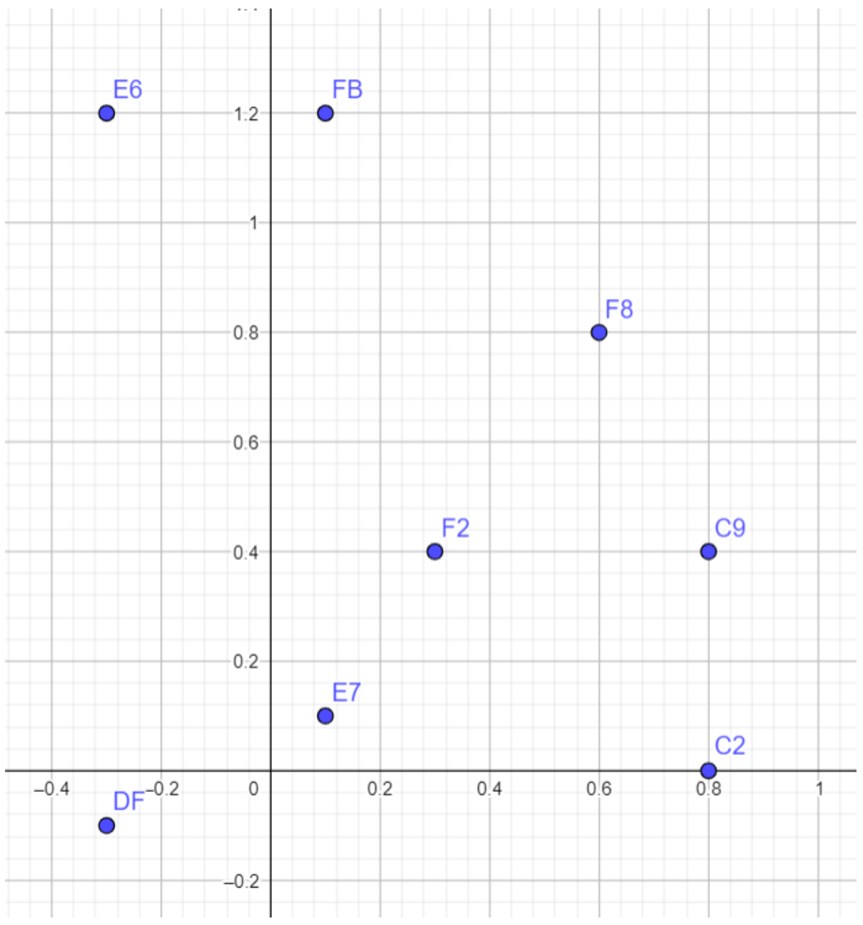

Figure 3 Example coordinate set output from algorithm

7. User interface for fine tuning (not implemented)

Unfortunately, we were unable to complete this part of the project, but here the description of the final project as we see it.

We need user interface that allows for fine tuning of the data. First of all basic linear transformations have to be applied, such as mirroring, skewing, scaling and shifting of the acquired shape to match the floor plan. Such operations are merely required to construct a transformation matrix from node-space to floorplan-space.

When nodes are aligned properly on the floor plan user can begin adjusting nodes that are placed incorrectly or anchoring correctly placed nodes to a fixed position. Every change or update of data on the nodes causes another run of the algorithm as described above, but instead of starting with random node locations, we start with last output and add new user-added constraints to the data. This will adjust position of non-anchored nodes in real time. The accuracy of the resulting data is improved significantly with every iteration of this process.

8. Reflection on the project

8.1. Overall project success

The goals for the project were to create a system, which can be used to determine locations of lamps in a large set of luminaires. Since using floorplan to make small adjustments in a set of received coordinates was a viable option, we have decided to invest more time in obtaining more accurate shape and relative distances for the set RuuviTags, rather than calculating exact coordinates of them. This reduced our workload by a large margin by removing the need for accurate calibration of the Bluetooth modules. With our chosen method we were able to obtain more or less accurate representation of real-world layout of RuuviTags. Gathered data is easily recognizable by human and when used alongside a floorplan drastically decreases the workload of setting up a lighting system.

Initially we planned to develop complete single-application solution, but failed due to time constraints and lack of expertise with graphical applications. Instead, we have two pieces of software in Python and Golang for data gathering and processing respectively with graphical representation produced in any available plotting software.

8.2. Improvements

Due to time constraints, we decided not to do develop a prototype with better RSSI signal propagation. Both us and sponsoring company agreed, that using a lower frequency carrier signal e.g., 900 MHz, is more suitable for this task, since it is affected by distance and obstacles to a lesser extent compared to 2.4GHz. More accurate calibration as well as measuring luminaire casing signal dampening capacity can also yield more accurate results in a final product.

8.3. Timetables

At the beginning of the project, we were having an optimistic picture of project complexity and requirements, but as we started working hands-on on the project we realized, that most of the tasks were way more difficult than we expected and were prepared for. This resulted in many delays in the timetable, as we had to explore many new fields that were new to us. For instance, we had basically zero prior knowledge of embedded programming. Nevertheless, our passion for learning new stuff allowed us to complete all of the essential parts of the project, although we had to focus only on one RSSI based solution without evaluating any other ways to solve the task.

8.4. Risk analysis

At the beginning of the project, we determined that the only possible risk would be that no meaningful results will be delivered at the end. We succeeded in achieving a working prototype that is able to provide insights into applicability of our solution.

The most severe risk for the final product that we have encountered during the project was the fact, that most readily available systems are not suited for this task and would introduce a lot of bugs and inaccuracies into the system as well as lots of signal noise to the setup site. Developing an in-house solution that focuses on this task is a must in our opinion.

9. Discussion and Conclusions

The project went well, and we reached our personal goals in learning and adapting new skills. Most importantly we have greatly increased our teamwork skills and cooperation performance.

List of Appendixes

- Source code for currently used fork of RuuviTag firmware
    - https://github.com/dexter3k/ruuvi.firmware.c/tree/ensto_proto
- Source code for data collection from tags
    - https://gist.github.com/dexter3k/1b3b1ec250d810c9535cdca410dea101
- Source code for data processing algorithm
    - https://gist.github.com/dexter3k/4f1134c572fdc3bc4e532767a8b6121d

References

- RuuviTag firmware repository
    - https://github.com/ruuvi/ruuvi.firmware.c
- SEGGER Build instructions
    - https://lab.ruuvi.com/ses/
- Gradient Descent
    - https://en.wikipedia.org/wiki/Gradient_descent