Aalto-yliopisto, Sähkötekniikan korkeakoulu
ELEC-D0301 Protopaja
2018

End report

Project #6
ROTS - Room Occupancy Transmitting Sensor

Date: 30.8.2018

Oskari Luostarinen
Tommi Penttilä
Severi Tikka
Olli Riikonen

Info page

Jäsensivu
Oskari Luostarinen
Tommi Penttilä
Severi Tikka
Olli Riikonen

Project manager
Oskari Luostarinen

Sponsoring company
Granlund Oy

Starting date
4.6.2018

Return date
31.8.2018

Summary

In this report we will go through our process and conclusion of our project.
We did our project for a company called Granlund OY, which is a part of the European Eteacher project. The aim of the project is to lower the amount of electricity consumed in buildings by changing the habits of people. The aim of our project is to build a sensor array that is able to decide if there are people in a room and if the lights are on or off. We tried many different ways to achieve this, of which a few have worked in a desirable way. In this report we will showcase which have succeeded to produce the wanted results. We have divided different parts of the project into their own categories.

Sisällysluettelo

1. Johdanto

We were given vague instructions on how to approach the problem, which was to produce a proof-of-concept of an easily movable sensor module that will be able to teach people to act so that they save electricity. The module we produced is a device that detects people in a given room, deciphers if the lights are on or off. It decides this via machine learning on a server, although we still have not been able to train it enough in different situations for the module to detect this reliably. This has been due to time constraints.

The project went through many twists and turns considering the technology we would use for detection of human presence and communications. We used a plethora of microphones, different RGB-sensors and such, only to decide on not to use any of them. Microphones were deemed too unstable and deciphering speech from ambient sound proved to be too difficult. The RGB-sensors we tried were too small to accurately detect anything from a wide area.

A server was built for the module to communicate with. The server tracks data given by the module to determine room occupancy and luminosity. Software implementation was designed to be simple enough to be implemented in the proof-of-concept scale and timeframe but scalable for possible later expansion. Arduino code was done using C++ and Arduino libraries. Server backend was done using python3. Server frontend was done using Flask, html, JavaScript and CSS. Database was implemented using mongoDB. All the services are running in scalable cloud services.

2. Objective

The project main goal, put into one sentence, is

"To create a proof-of-concept device for measuring room occupancy".

To further define the problem, we have set up defining goals:

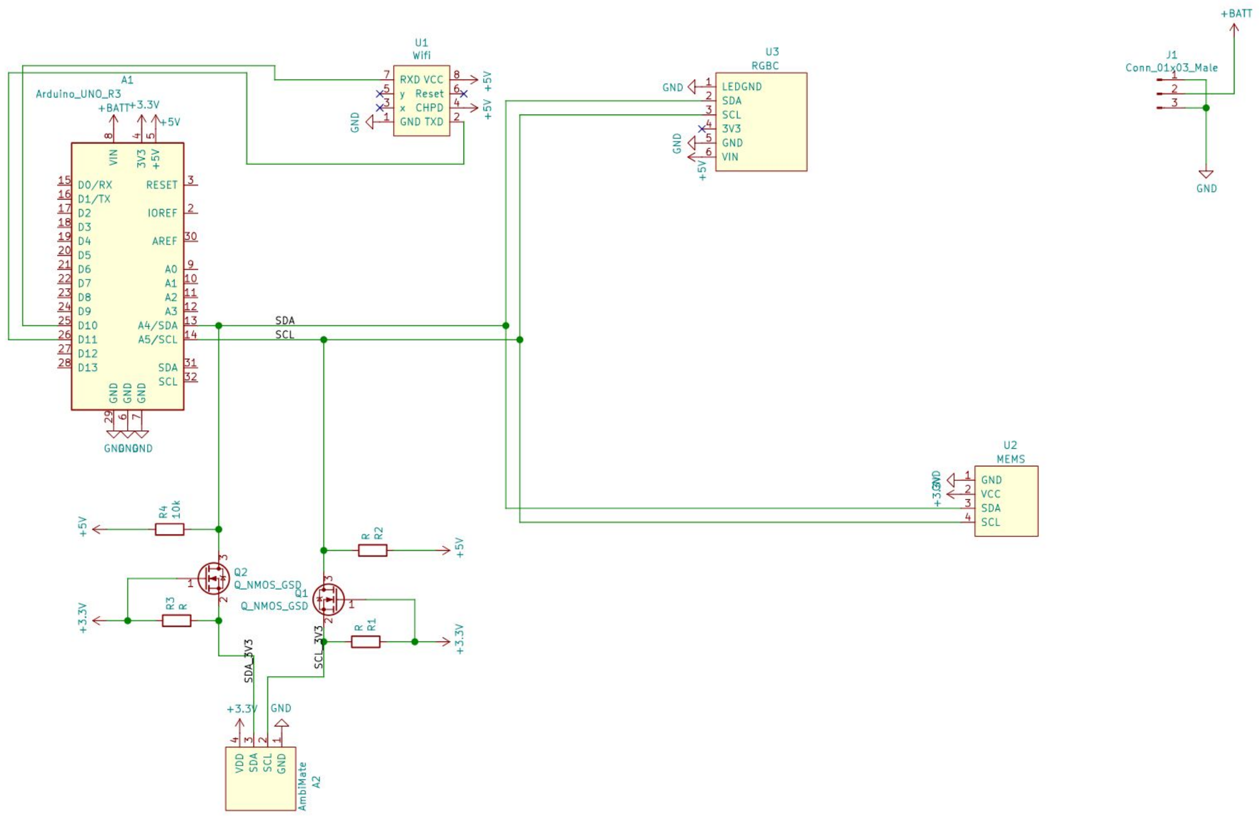
- Study of electricity consumption of a room while it's not occupied
- Measure and present information on how long room has been occupied
- Measure and present information on how many people have been in the room at different times
- Expected user will be companies and the public sector
- Easily movable from place to another
- Optimise device energy efficiency

3. Elektroniikka

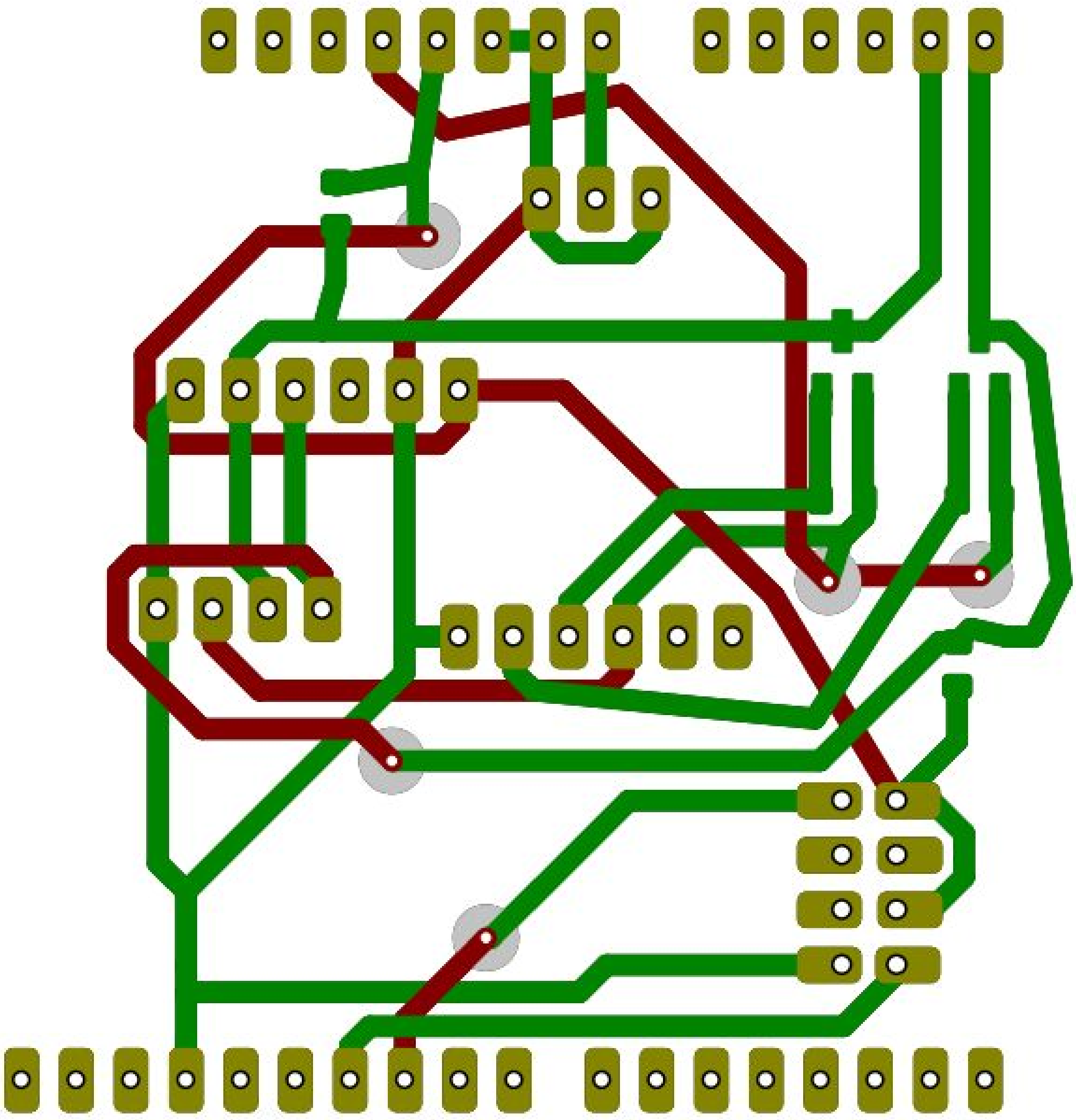
3.1 *The sensors*

The sensor modules consists of 3 different sensors the Ambimate MS4 Series sensor module, an Adafruit 1334 RGBC sensor module and an Omron D6T-44L-06 MEMS IR sensor. The Adafruit 1334 gives us data on lux values, RGBC values and colour temperature values on graph which we can use to determine are the lights on or off. The Omron D6T-44L-06 gives thermal data via a camera in an 4x4 grid which we use to determine whether there are people in the room or not. Lastly the Ambimate MS4 we use for additional data which can be used for a variety of uses.

3.2 The electrical circuit



All the electrical components are connected via a self made electrical circuit combined with an Arduino UNO shield. This “shieldcircuit” makes sure that all the different sensors and the wifi-module gets the voltage needed. It also levelshifts the SCL and SDA signals for certain parts to an acceptable voltage. The inclusion of an Arduino UNO was purely because of ease of use. The Arduino is powered either by four 1,5V batteries, an USB cord or a 9V adapter from the wall. We did not have time to measure the power usage, but it should be at acceptible numbers. This power usage can be further knocked down by removing the Arduino UNO and replacing with a pure microcontroller.



A PCB file picture from KiCad which was used to develop the circuit.

4. Chassis

4.1. ***Requirements***

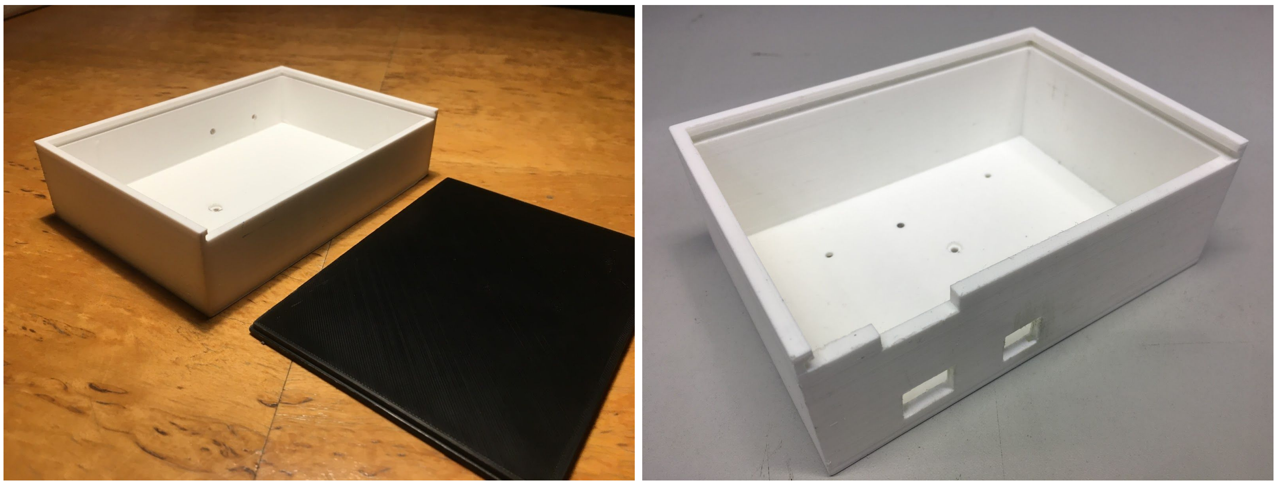
The starting point for the Chassis design process was to define what features it needed to have. It ofcourse needs to be able to protect the components inside and make the product a complete package. Since this is not a commercial product, the process focus wasn't on aesthetics but rather on usability. Granlund requested for the product to be easily moveable so that had to be taken into account. The chassis could also be used to direct cables. Because some of the sensors had limited visibility, the sensor compartment's direction had adjustable. So that gives us four major requirements: directable sensor compartment, enough spacious, built in cable management and easy moveability

4.2. Modeling

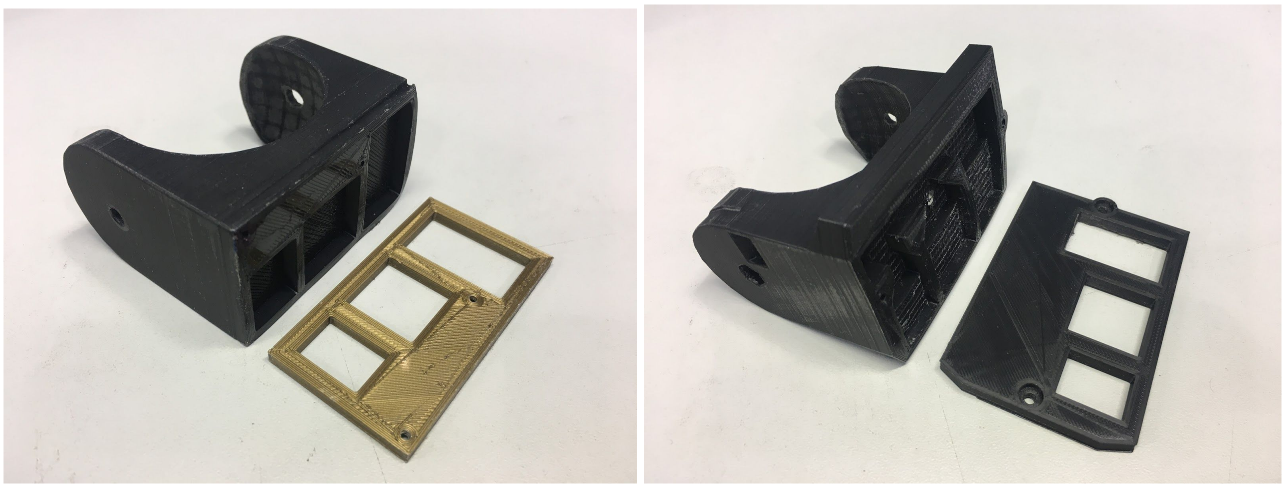
The modeling phase started with carefully measuring the dimensions of the different components, brainstorming and drawing a rough sketch. The mounting mechanism became the first problem. One of the ideas was to have a separate mounting bracket that would be screwed to the wall or ceiling. Since the product had to be easily moveable and screws weren't a popular idea. Because of its simplicity we ended up using strong double sided tape. The sensor compartment became the second problem. We found some small 3D-print models of camera tripods and decided to take inspiration from them and make our own version. Instead of a bracket for a camera, we had a sensor compartment and instead of tripod legs, we had a box which had everything else inside. While the project got along, we added different new features to the chassis. This included: holes for the Arduino ports, grooves and holes for cable management, an on/off switch to conserve battery charge and screw and hex nut holes.

4.3. Polish and Assembly

After the all the pieces of the final version were printed, assembly could start. This phase went more or less well after some filing.

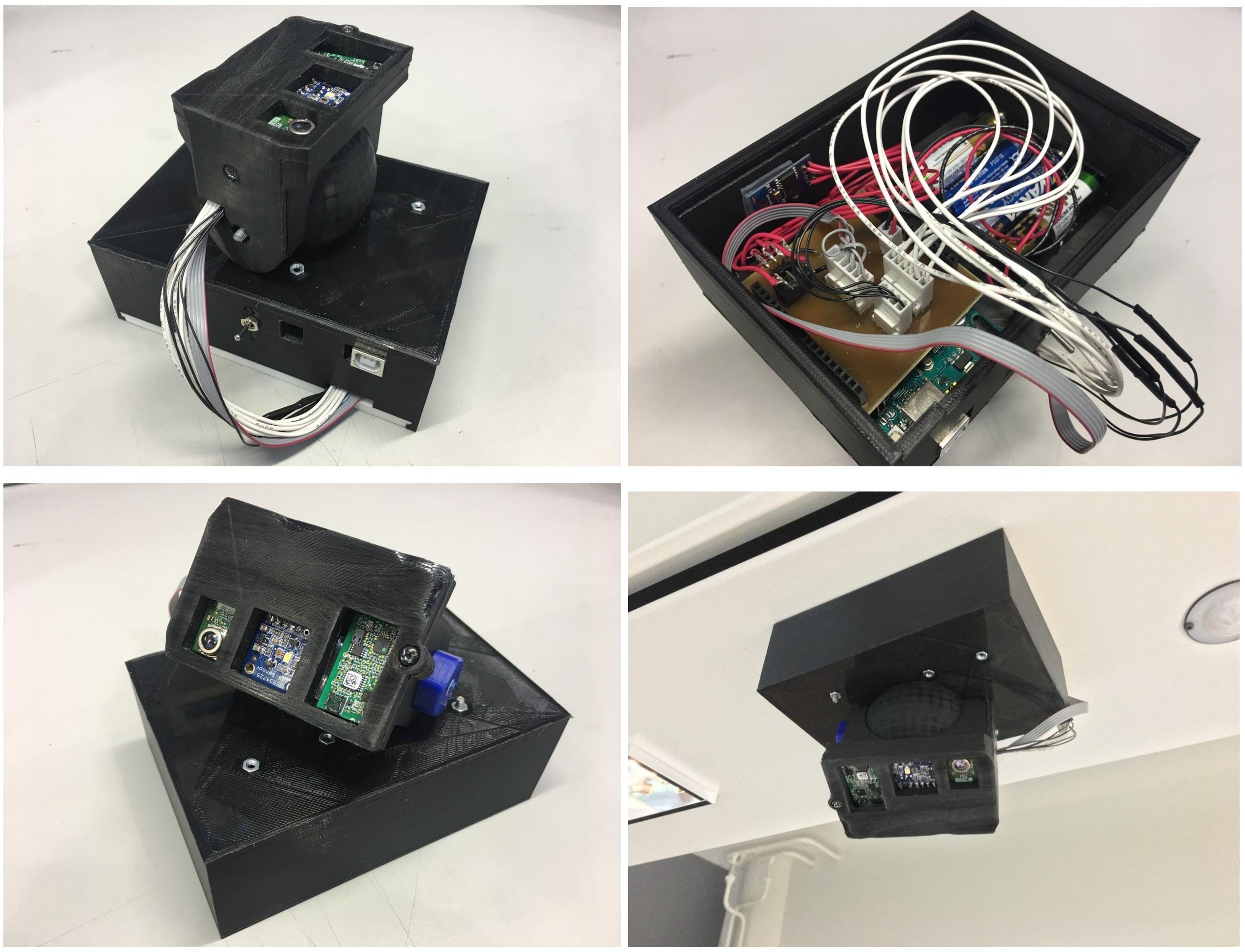


First and second version of the chassis box



First and second version of the sensor compartment

Some pictures of the tool-free chassis box cover


Finished product

5. Software

The software was implemented in four parts: measurements, uploading the measurements to a database, processing the measurements and showing the results in the UI. The implementation can be divided into three logical partitions: the measurements, the server and the estimation, which are explained in detail in the following sections. The source code for the project is available at https://version.aalto.fi/gitlab/riikono2/grandproto

5.1 Measurements

The measurements are executed and slightly processed in Arduino Uno and developed with C++ after which they are uploaded to the server. Measurements are collected from the three sensors used using I2C protocol. The loop which is executed continuously on the board periodically checks the measurements and sends the data into the server. The selected interval for the measurements is 10 minutes. In the case of collecting training data, the measurement cycle can be shorter, as thus we gain more data points for evaluation. The general architecture and interaction of the software is presented in Figure XXX.

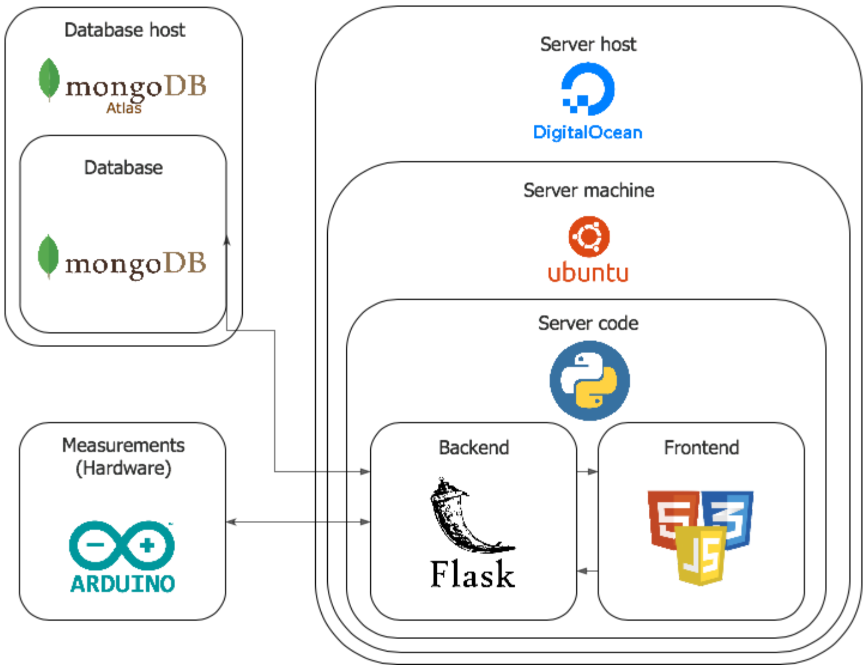


Figure XXX. General architecture of the project software implementation

5.2 Server

The server is running in DigitalOcean virtual Ubuntu 16.04 machine. The server consists of both backend and frontend, both of which are developed using python3. The server backend implements a simple processing method for the estimations, a handle for the database, as well as a simple API for uploading and showcasing the results. These are implemented in most modular and individual way possible, so that scaling the solution with, for example, Kubernetes cluster is easy.

5.2.1 API

The api implemented mainly focuses on the handling of the incoming measurements. The API checks for incoming http requests of type POST and GET for URL /api/upload/<api_key>/<room_id>/<data> and parses the data for the corresponding user and room given in the api_key and room_id arguments respectively. The system checks if a user with provided api_key exists for integrity. The data given is of json format and only measurement keys hardcoded into the api are accepted.

The API also provides a method to get data forgiven api key with url /api/get/<api_key>/q where q is a mongoDB query formatted as json. This will return the found data as json.

Other API calls are possible to implement if necessary. Note that the security issues with using the api_key are immense and should be considered should the device be scaled for larger audiences.

5.2.2 Database handle

The data is saved into a mongoDB, hosted in MongoDB Atlas. The current plan used is the free-tier one with obvious limitations. Should the product need scaling, the database can be scaled with ease. The server backend implements a DB_handle() class which handles the connection and queries to the database when the server is running. The connection is done with pymongo. Even though the usage and scaling of a mongoDB is easy, the integrity and schema checks can be disastrous. The implementation on this project is extremely naive and should be considered accordingly.

5.2.3 GUI

Data gathered and processed can be examined in the simple GUI which the server provides as well. The GUI is done using Flask and the html templates it launches with retrieved information. Information displayed relies heavily on graphs implemented with chart.js javaScript library. The navigation and outlook of the site is done using Bootstrap 4, enabling the site to be at least somewhat responsive and mobile device friendly. In figure XXX a you can see the GUI in desktop mode and in figure XXX b in mobile mode.

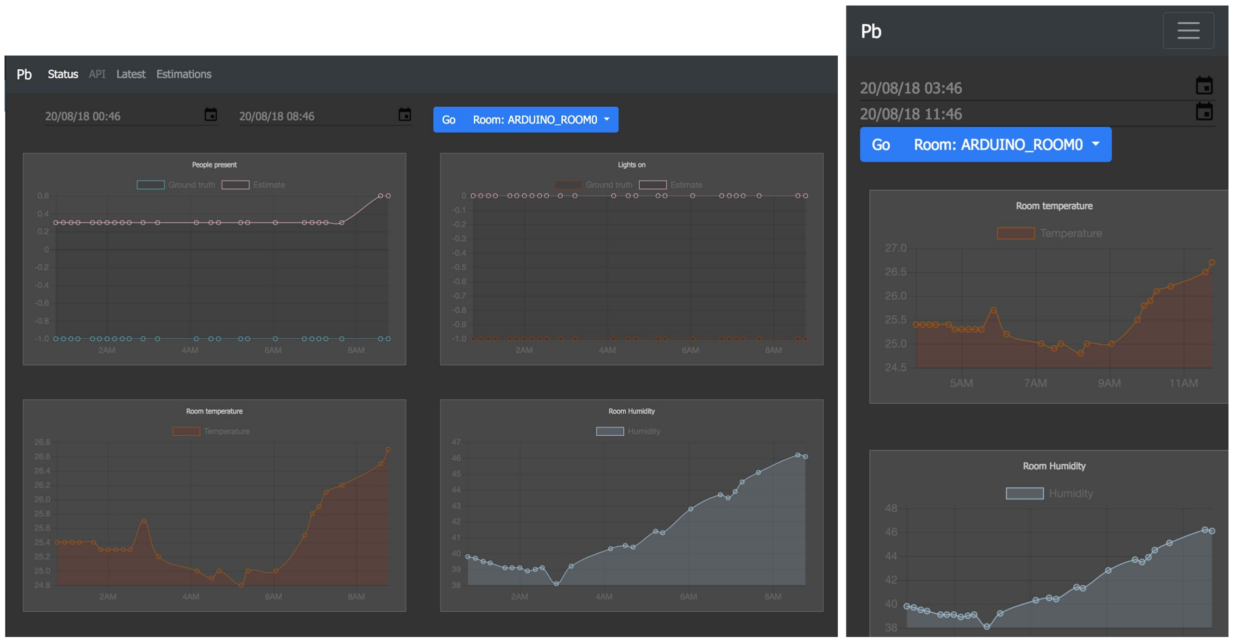
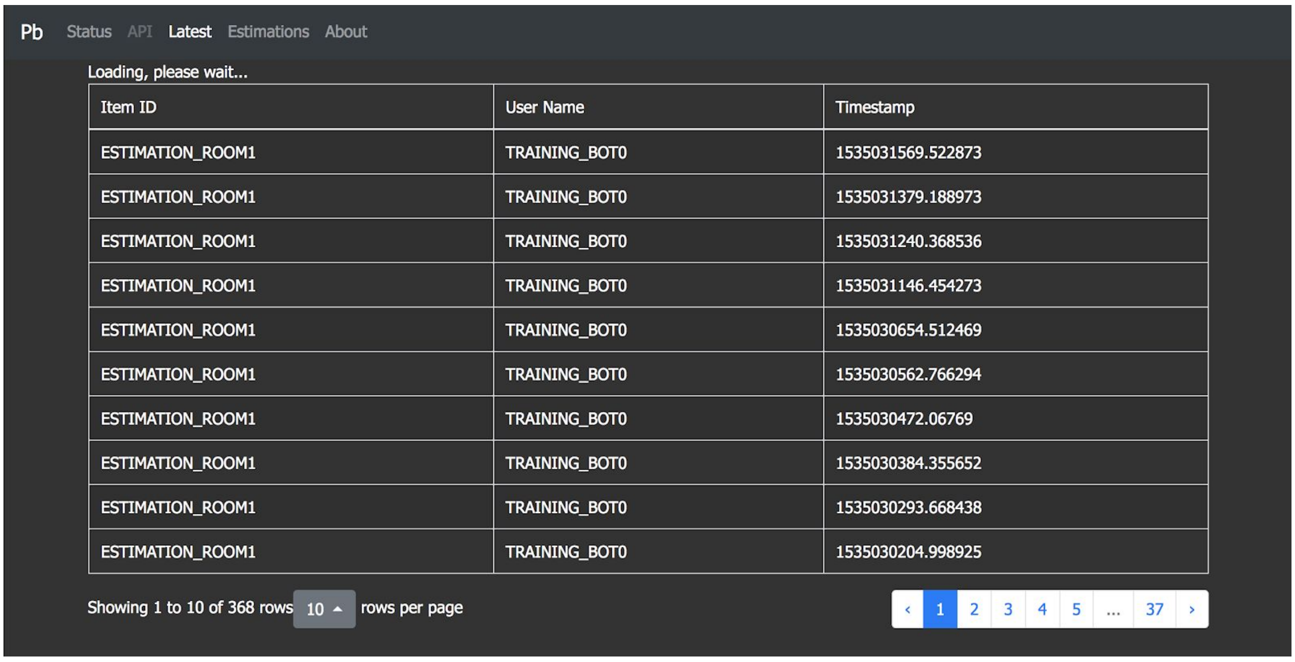


Figure X (a) Project GUI in desktop environment (b) Project GUI in mobile environment

In addition to the status of a room, which portrays the measurements and estimations of a room in selected timespan, the GUI has pages for displaying the latest uploads for a user, as well as more detailed view on the estimations. Figures XXX and XXY show the pages of a list of the latest data uploads, and estimations respectively. These pages were added to give an easier access for observing data gathering and estimation performance.



Loading, please wait...

Item ID	User Name	Timestamp
ESTIMATION_ROOM1	TRAINING_BOT0	1535031569.522873
ESTIMATION_ROOM1	TRAINING_BOT0	1535031379.188973
ESTIMATION_ROOM1	TRAINING_BOT0	1535031240.368536
ESTIMATION_ROOM1	TRAINING_BOT0	1535031146.454273
ESTIMATION_ROOM1	TRAINING_BOT0	1535030654.512469
ESTIMATION_ROOM1	TRAINING_BOT0	1535030562.766294
ESTIMATION_ROOM1	TRAINING_BOT0	1535030472.06769
ESTIMATION_ROOM1	TRAINING_BOT0	1535030384.355652
ESTIMATION_ROOM1	TRAINING_BOT0	1535030293.668438
ESTIMATION_ROOM1	TRAINING_BOT0	1535030204.998925

Showing 1 to 10 of 368 rows 10 rows per page

Figure XXX. GUI page for the latest data additions.

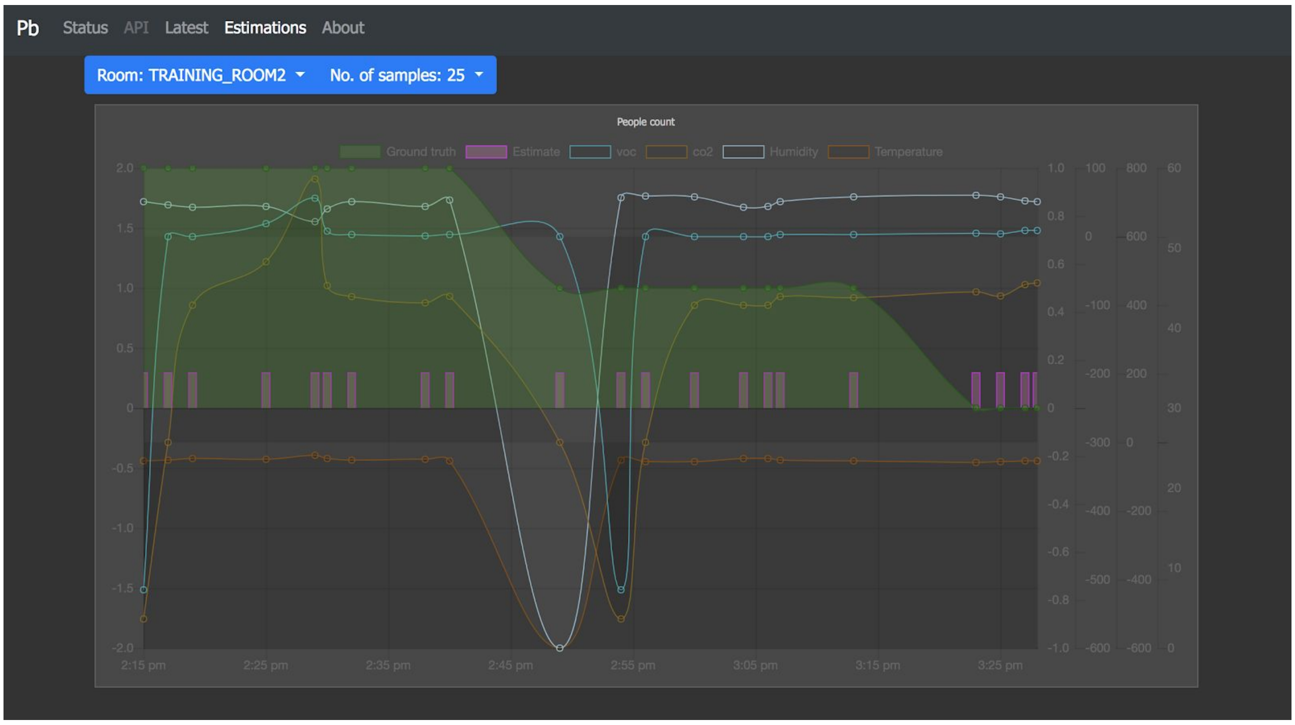


Figure XXX. GUI page for inspecting the latest estimations in a room.

Pages explaining the usage of the API URLs, as well as the technologies used in, and motivation behind the project were planned but not implemented due time restrictions.

5.3 Estimation

As one of the goals of this project was to estimate the occupancy of the room, as well as are the lights on in the room, some estimation from the measurements was needed. We used six different out-of-the-shelf classifiers provided by python package sklearn: naive-bayes, support vector machine, k nearest neighbours, logistic regression, decision tree and linear discriminant analysis. The implemented class ClassifierSelector() trains each of these models with provided training data and selects the best classifier for estimating the selected value: lights_on or people_presence.

The count of the people proved to be too difficult of a challenge given our gathered training data. Instead, we decided to estimate if any person was present in the room. The estimator trained is used for every measurement that is uploaded to the server using the API for estimating people presence and whether lights are on in the room or not. The estimator returns a decimal value in range 0-1, describing the probability from 0 to 100% respectively, of lights being on or person being present in the room.

The best classification rates we achieved with the gathered training data was 86% for estimating lights on situation. The estimation of people being present did not work at all: the classificators trained performed worse than random guess. This lead us to implement another, more crude estimation method: thresholding estimation. This method simply looks of values of selected features

in measurements are within predefined thresholds and gives a probability similar to classificator on the estimates. The features and their respective thresholds were selected by examining the gathered data. In figure XXX is portrayed the estimate, and the observed ground truth of lights being on, as well as people presence in examined time period in one room. The estimation accuracy on the lights on value is considerable, but the people presence is desolate.

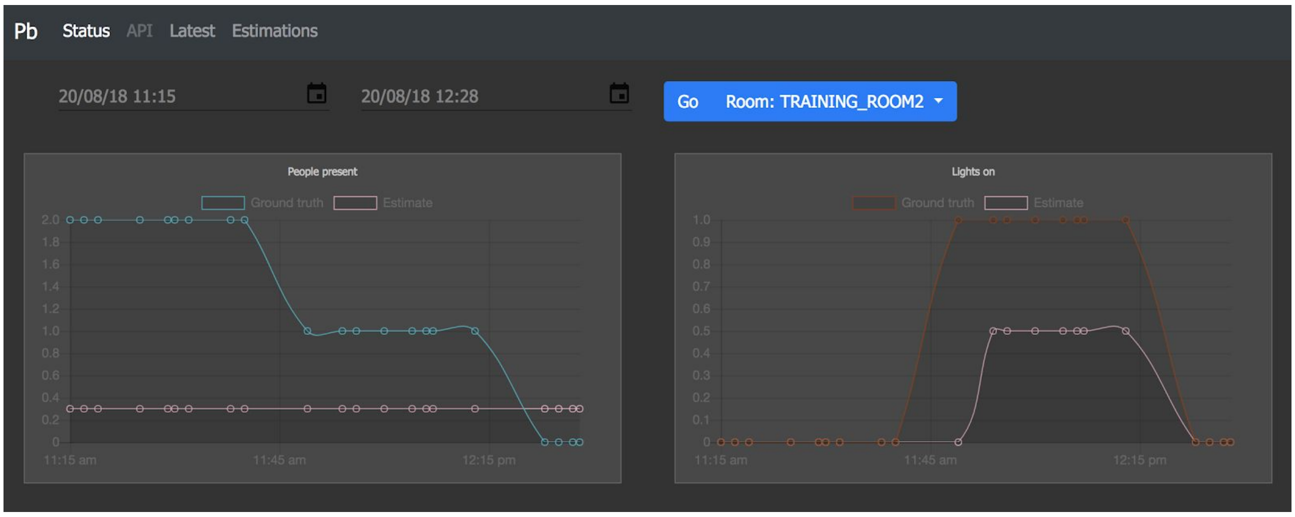


Figure XXX. Estimation and ground truth for a single room on specific time period.

5.4 Acknowledgements

5.4.1 Security

Even though the project software was developed with security and scalability in mind, the limited human resources, knowledge, testing and time made it impossible to make a completely safe product. The following points explain the biggest issues with the current implementation.

1. The server originally used nginx to handle incoming requests and hence limit the executions of scripts. however, the communication to the mongoDB Atlas was blocked by it and was

not resolved in time of the project deadline. Therefore the current implementation is prone to attacks.
2. The server has only a simple firewall set up and allows connection to the GUI and API only through selected port. The developer connection is secured with SSH but the traffic is not encrypted when using the API or GUI, hence making the data passed through the system an easy target, for example, man-in-the-middle -attacks.
3. In the developer phase, the password of one account to the mongoDB Atlas was hardcoded into the code. This has later been removed and password changed. The password needs to be entered each time the server is booted.

5.4.2 Estimation

The estimation of the presence and lights are not accurate. We had trouble gathering good training data due the difficulties with the circuit and especially the wifi-module, leaving us with only around 20 sufficient labelled training data points. Generally a good training of a classifier for problem of this complexity requires somewhere between 1,000-100,000 labelled data points.

5.4.3 Code integrity

As the project was developed in tight schedule, the code is not at all passable in terms of best practises. Many of the methods are lacking proper documentation and comments, and the structure of the code could be definitely improved. Moreover, the code contains methods for experimental features which are not explicitly stated to be ones.

5. Projektitoiminta

The project was done using our skillsets. This was the first project Oskari had done circuit design for, and the first project that Severi had done chassis design for. The software part is by far the most advanced as is Olli's skill in such subject matters. We all learned a lot during the project ranging from timetable adjusting to technical knowledge.

5.1. Reaching objective

We think that we have proven our design concept in the preliminary level, to further decide if we actually accomplished that further training is needed.

5.2. Timetable

We followed our original project plan as well as we could, even though some time constraints became apparent at the early stages. As an example finding the correct sensors took two weeks more than originally planned. Also since some of us had other things going on while the project was on, some additional delays had to be made. These delays lost us valuable time for training the server to analyse information.

5.3. Risk analysis / Riskianalyysi

Two of the risks were realized, scheduling fails most prominently. We had some buffer time for each task to be completed but those failed in the early stages as in the finding and testing of sensors took way longer than originally planned. The biggest problem in the late stages of the project was maintaining a stable wifi connection. This was due to the WiFi module we used, the ESP8266. The module was faulty, as it needed 3V3 to work on paper but actually the amount of voltage to get it working was 5V. This faulty module came up in the last 2-3 weeks of the project and there was no time to start looking for a new one. The ESP8266 had some trouble to maintain a stable connection, and this made data collection harder as the data would come through to the server at seemingly random times.

6. Yhteenveto ja johtopäätökset

All in all the project was a great learning experience for the whole team. Oskari as a project manager had to learn scheduling and motivational skills to keep the project on track, also he learned a lot of circuit design. Severi learned 3D modelling from scratch.

7. Liitteet

As attachments to this file you will find the original project plan and Datasheets to all the sensors we used.