Smart Experience Sampling

User Manual

Quinten van der Valk,

Kevin Giesberts,

Mick Thelosen,

Marijn Verschuren,

Martijn Koppejan

16-01-2025

Contents

Hardware

Components

Clicker

The clicker consists of:

- Battery
- PCB
- Pogo Pins
- Keyboard Switch

These parts are mounted to a tray which slides into the housing and is screwed in from the bottom of the housing.

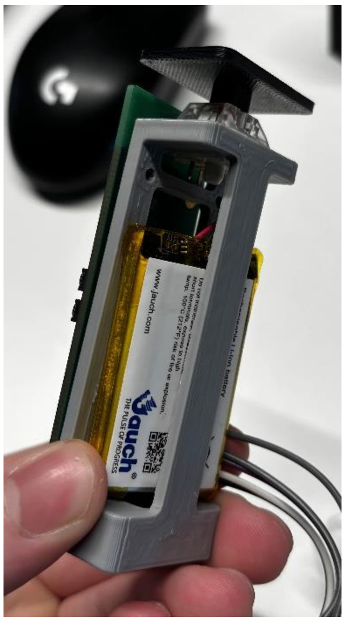


Figure 1 Clicker Tray Assembly

Beacon

The beacon consists of:

- Battery holder
- 2x 18650 batteries
- PCB

Server

The server consists of:

- Raspberry Pi 5
- Power Supply
- STM32WB07 & DWS1000

- Screen to display the QOTD.

The STM32WB07 & DWS1000 are development boards for the chips that handle the UWB and Bluetooth communication that are connected to the server (RPI) over USB and act as a receiver for the incoming data sent by the clickers.

PCB

Both the clickers and the beacons use the same PCB. The PCB handles the battery charging and the inputs of the switch. The PCB is designed using KiCAD. The files for this will be able to be found in the GitHub repository.

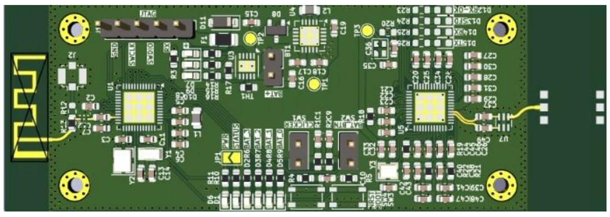


Figure 2 PCB.

Embedded software

The code that runs on the components can all be found on the GitHub repository. The code is developed using the STM32CubeIDE and is written in C.

Software

Frontend

The frontend code is a typescript NextJs application, using Shadcn ui as a component library. To run the code, clone dashboard repository, edit next.config.ts to be the correct backend url, and run `npm install` to install all dependencies.

To run the code in development mode run:

```
npm run dev
```

Backend

The backend is a python flask API. To get started, create a .env file, from the template provided by env.example in the main folder. The system requires the client url for configuring CORS and will not accept requests if this isn't provided.

The database options are optional, but recommended.

To install the required packages before running the code, execute:

```
pip install -r requirements.txt
```

Database

The database is setup to be a MySQL database, however the program accepts any SQL based database. To change the database provider, simply change the DB_TYPE option in .env in the backend code.

Raspberry Pi

For the tablet on which the question of the day is displayed we use a Raspberry Pi. The code that runs on this Raspberry Pi is written in Python. To install all the requirements needed run `pip install –r requirements.txt`

Web page

The webpage is a simple website written in Flask. It only shows the question of the day, but it also has the option to display the number of clickers inside of the building if the API supports it. The webpage automatically fetches the latest question of the day from the API every ten minutes. Change the `__API_URL__` variable at the top of the `main.py` file to the correct URL for the API. Then simply run `python main.py` and open the URL that Flask is running on.

Receiving click data

To receive click data we use one of the beacons which we connect to the Raspberry Pi via USB. The Raspberry Pi then reads the serial output from this beacon, decodes it and sends the data to the API. Change the `__API_URL__` variable at the top of the `ser.py` file to the correct URL for the API. You can also change the com port it listens on and the baud rate by changing the `__PORT__` and the `__BAUDRATE__` variables respectively. Then simply run `python ser.py` to start receiving click data.

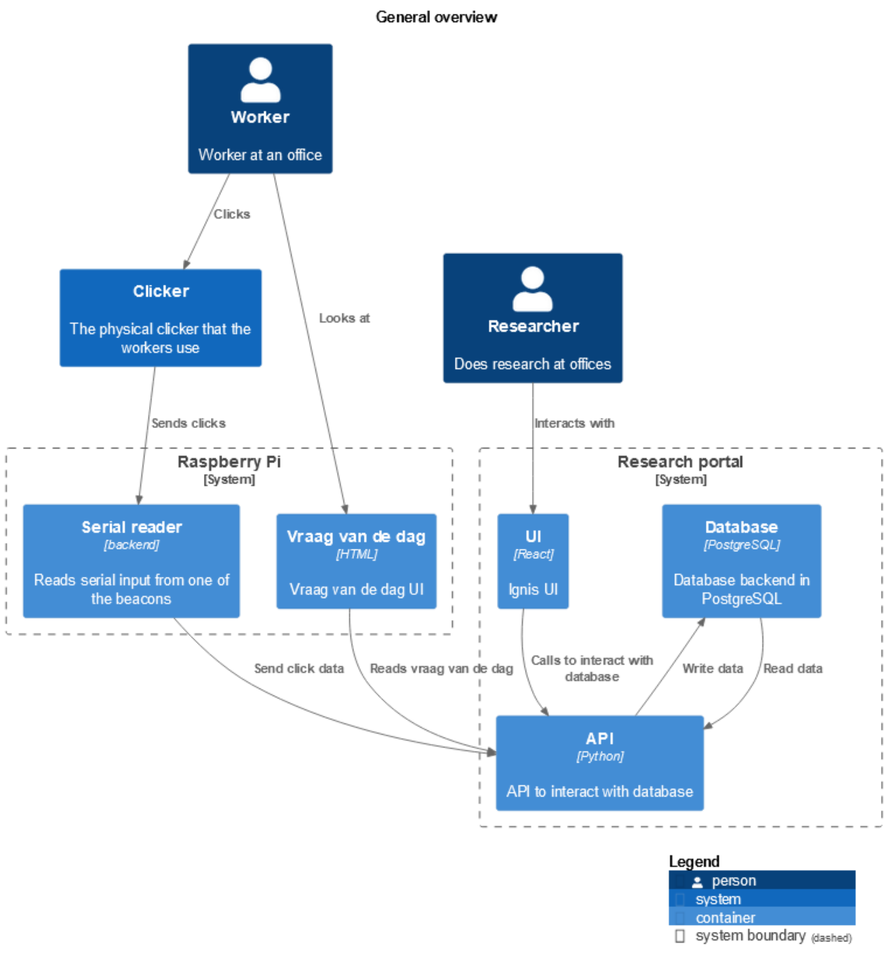
Decoding the raw click data

We need to send the following data from the clickers to the Raspberry Pi.

- Target ID
  - This is the a number indicating where the data is supposed to go. Think of it like an address on an envelope.
- Number of beacons
  - This is the number of beacons it successfully made contact with. This is necessary to know when the message will end
- The UUID of the Clicker
  - Each board has a unique UUID which is used to identify the clicker that was used to click.
- UNIX timestamp of the click
  - The timestamp in UNIX format of when the click was registered
- Beacon data
  - For every beacon we have data that is why the number of beacons is so important.

- We get the following information from the beacons
    - Beacon UUID
        - Same as the UUID of the clicker
    - Distance in meters
        - The distance from the clicker to this beacon in meters
    - Distance in centimeters
        - The distance from the clicker to this beacon in centimeters

C4 Model



Setup

Prerequisites:

- A map and the dimensions of the environment.
- STM32CubeIDE
- ST-Link

Assemble the beacons and clickers.

If a beacon is new and has never been used before a channel on which the beacon will communicate needs to be specified. Every beacon must have a specific channel between (1-39). Channel 0 is reserved for the server. To do this open Beacon_Software in CubeIDE and navigate to the main.c.

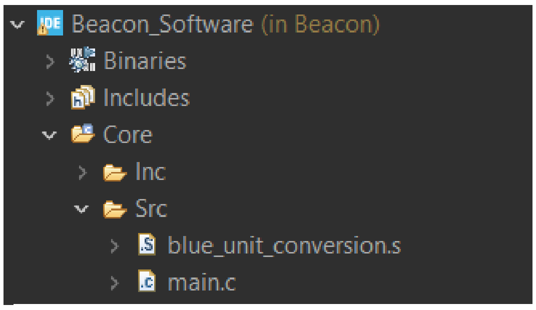
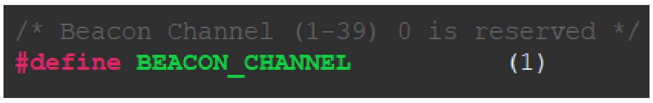


Figure 4 Beacon_Software main.c

This is where the BEACON_CHANNEL is defined at the top of the file change the beacon channel. Then upload the new code using the upload button.



```
/* Beacon Channel (1-39) 0 is reserved */
#define BEACON_CHANNEL          (1)
```

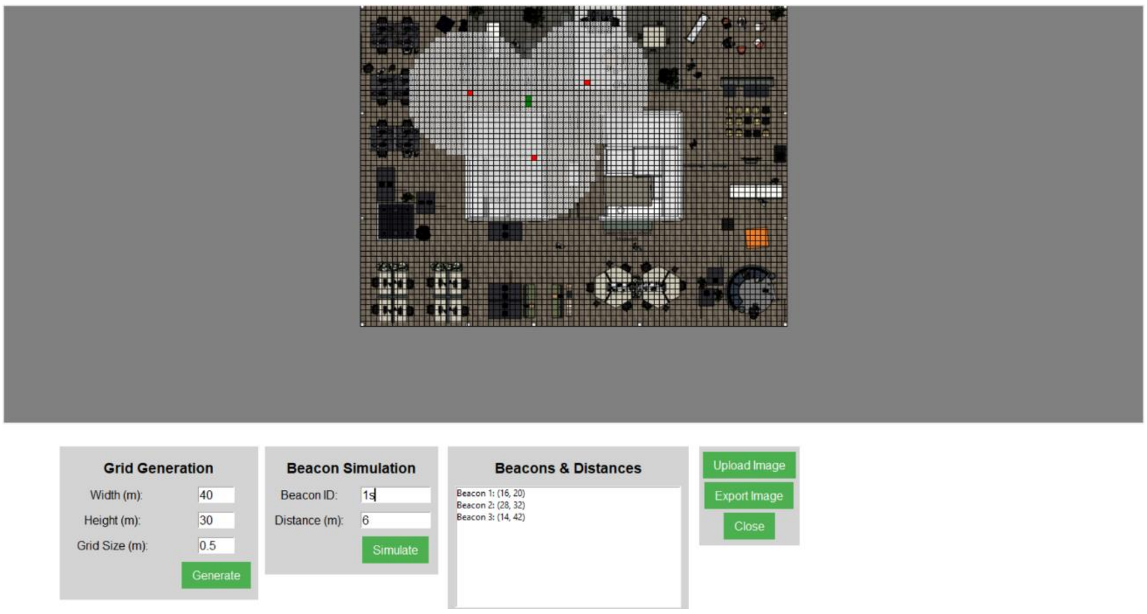
Figure 3 Beacon_Channel Definition

To set up a Clicker no manual software changes must be made.

The Development boards that act as the receiver for the incoming Clicks need to be connected to the RPI over USB and run the Server_Software. No changes must be made to this code. The messages are transferred to the RPI using Serial at a baud rate of 115200.

Placing Beacons

Beacons should be placed withing 25 meters from each other to get the best measurements. When placing beacons keep track of where you are placing them and put this into the beacon map application. Using the beacon visualizer application, you can check if at least 3 beacons overlap everywhere in the environment.



```
Beacon 1: (16, 20)
Beacon 2: (28, 32)
Beacon 3: (14, 42)
```

Figure 5 Beacon Visualizer

What would we improve in the future

Software

Viewing Beacon Battery Percentages
A feature that would be useful in the future is the ability to view battery percentages in the software. This would allow a maintenance worker or manager to see which beacon needs to be recharged. To achieve this, the software for both the beacon and the clickers would need to handle this information. This solution optimizes the process of replacing batteries for the beacons. Additionally, it could enable the possibility of notifications when the batteries are nearing the end of their life.

Enable Beacons with a Self-Positioning System
By "self-positioning," we mean a feature that allows all beacons to automatically create a map of their locations in a room. Currently, we have to manually input the positions of the beacons. In a future version, automatic mapping would be incredibly useful. Not only would this reduce maintenance time, but it would also provide a simple way to recreate the map of all beacons when needed. To make this work, an ideal solution would involve integrating a compass or some method to determine the direction from which signals are coming.

Hardware

Additional Feedback When Clicking
When a user clicks, they currently receive feedback from the button itself. However, the user has no way of knowing if the distance data has been successfully transmitted and calculated. This could be addressed by, for instance, adding a small vibration motor to the clicker. The motor could vibrate once when the distance data is retrieved and then again when the data is successfully sent. If the data cannot be sent immediately because the user is out of range of a screen, the clicker could vibrate once to indicate that the data has been retrieved and later vibrate again when the data is sent. In cases where the clicker sends the data later, it would vibrate once to confirm that the data has been sent and saved. This way, the user always knows whether the data has been successfully transmitted or not. Alternatively, this issue could also be addressed using an LED light as a form of feedback.

Figures