Shelly Christmas Coding Challenge Documentation: The Indoor Positioning System

Jernej Cukjati

December 31, 2023

Abstract

For Shelly Christmas Coding Challenge, the Indoor Positioning System (*IPS*) was selected. The *IPS* focuses primarily on the usage of smart Shelly devices with Bluetooth Low Energy (*BLE*) capability. They are used as Bluetooth beacons, which are detected by Android smartphones (with Android, minimal sdk version 33). Smart phone application considers the known locations of the beacons and their Received Signal Strength Indicator (*RSSI*), to compute approximate location of the user. The *IPS* was tested with three Shelly devices, in a single floor of a family house. Nevertheless, this can be applied to a larger building, using a sufficient number of beacons. Furthermore, project can be extended to other Bluetooth devices as well as other technologies (e.g. WiFi). To comply with the project requirements, the primary focus was on Shelly.

1 User interface

Application, which provides the User Interface (*UI*), was written in Java, for Android smart phones, with minimal sdk version 33. In my case Xiaomi Redmi Note 11 pro 5G was used. To make the *UI* (figure 1) as simple and user friendly as possible, it consist only of:

- **custom made toolbar (1)**,
- **TouchImageView (2)**, which shows floor plan
- and the **button (3)**, which triggers the interpolation algorithm.

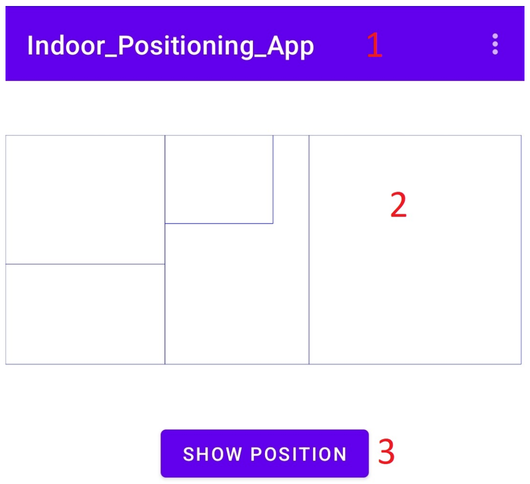


Figure 1: Initial UI

1.1 Toolbar

Toolbar provides basic drop-down menu with two options (figure 2):

- **Show devices**,
- **Show grid**
- and **MQTT Broker**

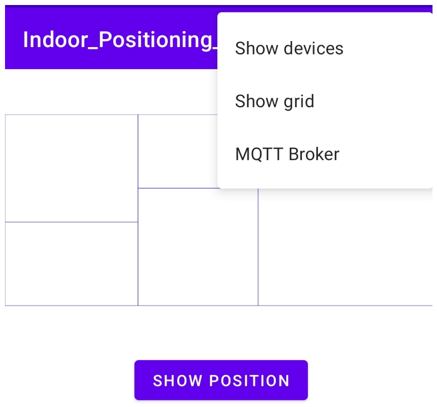


Figure 2: Initial UI

Option **Show devices** allows the user to see the location of the Shelly devices currently detected (figure 3)

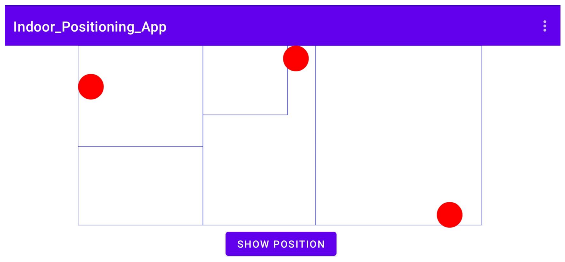


Figure 3: Location of the Shelly devices are marked with the red dots

Show grid shows the gridded area, where each rectangle contains interpolated value of *RSSI*. Those values are considered when calculating the user position (figure 4).

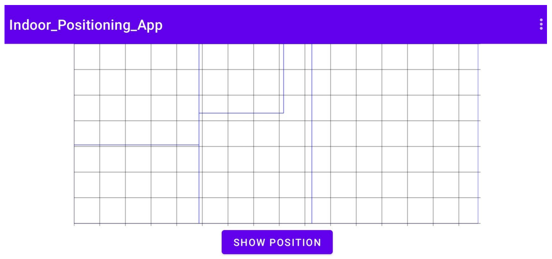
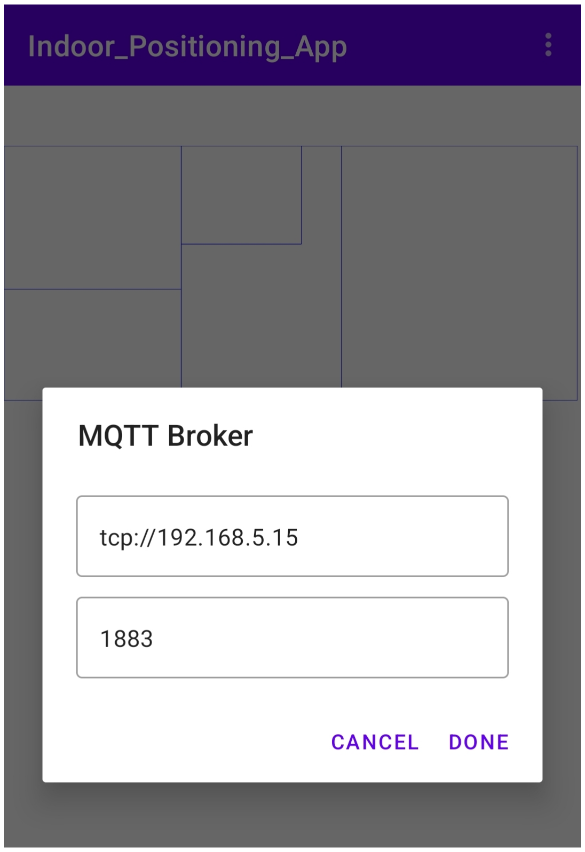


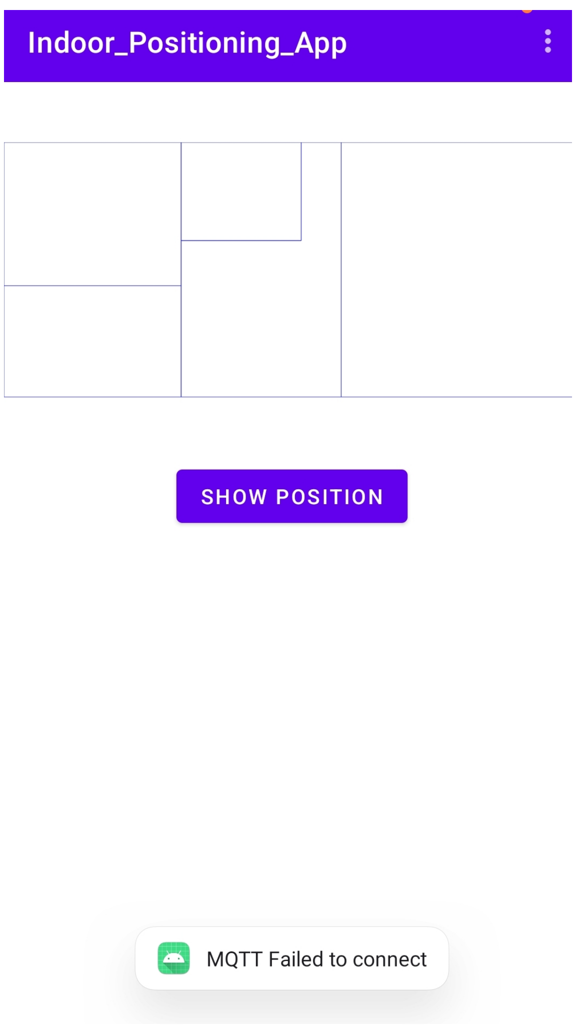
Figure 4: Gridded area

MQTT Broker shows the user pop-up window and allows him to enter IP and PORT number of the MQTT broker (figure 5). The Port textbox performs text validation and allows the user to enter up to 4 numbers. After the user confirms the input, the Toast message is displayed in case or successful or invalid connection(figure 6)



```
tcp://192.168.5.15
1883
```

Figure 5: MQTT broker popup



MQTT Failed to connect

Figure 6: Toast message if connection to MQTT Broker failed

1.2 TouchImageView images preparations

TouchImageView displays considered floor of the selected building. Floor plans were prepared as Multipolygon layers in GeoPackage (*GPKG*) files. This was done by using the QGIS software. It should be noted that even though these plans are based on the real building, some rooms were modified and details (such as doors, windows, electrical installation etc.) were omitted. This was done for privacy reasons.

To read and display *GPKG* files, GeoPackage Android Lib was used. Initially, it was though that application would considered multiple floors of the building. However, due to the lack of "Shelly beacons", only one floor was considered. Nevertheless, some leftovers for multiple floor usage are still present in our code. Additionally, if anyone would like to update *UI* to list multiple floors, **FloorAdapter** and **Floor** class can be used. They were removed in commit 7103330, on the main branch.

1.3 Additional notes

For the application to work properly, the smartphone must have enabled Bluetooth and location.

2 Backend

2.1 MQTT

As seen in figure 7, Message Queuing Telemetry Transport (*MQTT*) protocol was used to establish connection between our devices.

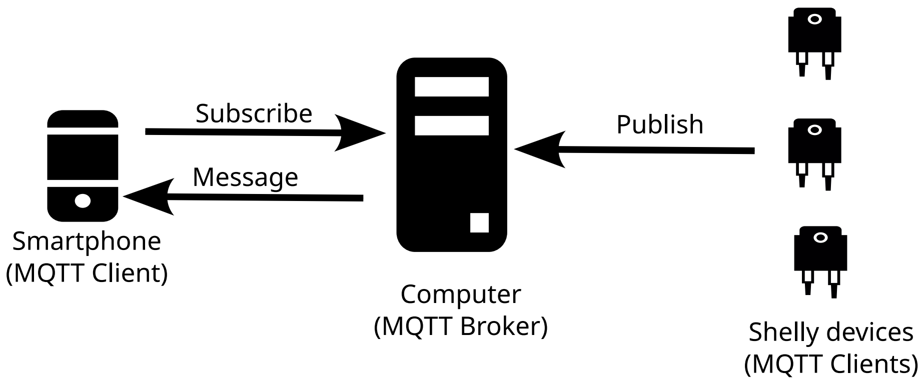


Figure 7: MQTT Device communication

2.1.1 MQTT Broker

Firstly, MQTT Broker was established. I used open source MQTT broker Eclipse Mosquitto. Setup was done according to YouTube tutorial, on a laptop ASUS GL553VD.

2.1.2 MQTT Clients

Secondly, Shelly devices were connected to MQTT Broker. This was done with the help of this video. I used three Shelly devices which support scripting and have Bluetooth capability. These were: two Shelly Plus Plug S, and a Shelly Plus i4. Each device run a script that would publish a message to our MQTT Broker. In detail, scripts had hard coded constants, which would represent the position of the host device in the building. Secondly, each device utilized its Bluetooth capability to scan for other Shelly devices and obtain their *RSSI* value (figure 8 and figure 11). At the end, the script published MQTT message, which consisted of: source device location, its name, name of the detected device and its *RSSI* (table 1).

The message was separated with the hash sign, for example:
80:64:6f:d0:33:22##1435##0##0##ShellyPlusPlugS-80646FD03320##-71##shellyplusi4-083af2013470.

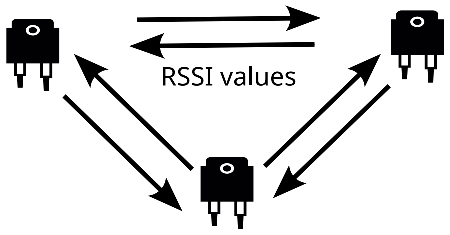
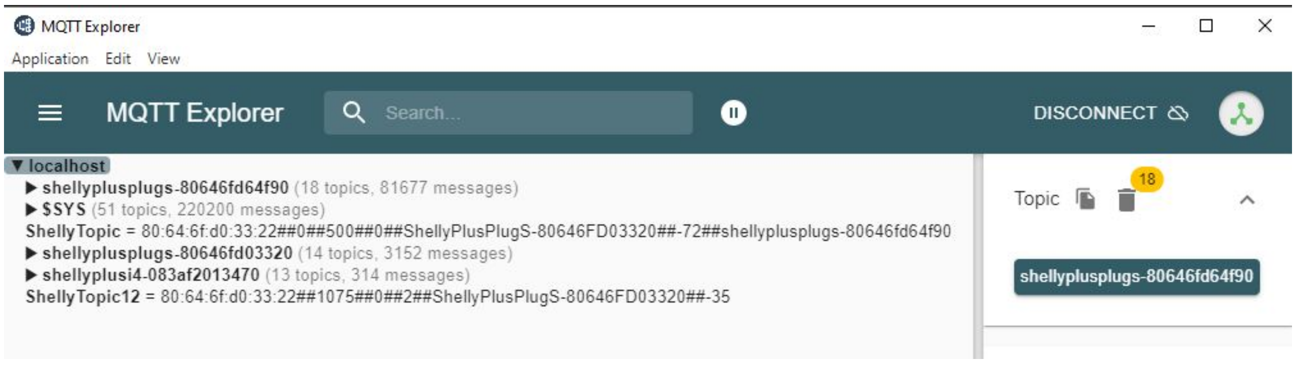


Figure 8: Shelly devices obtaining *RSSI* values

Table 1: MQTT message

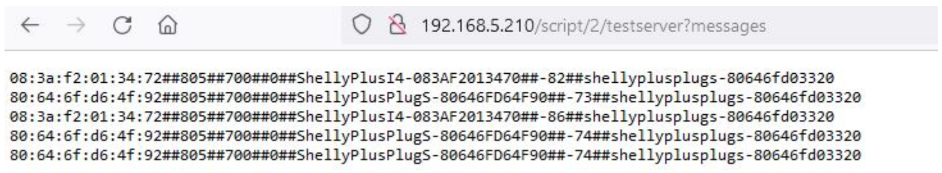
	Source location					
Detected Bluetooth MAC	X	Y	Floor	Detected name	Detected RSSI	Source name

The proper functioning of the script, was tested with MQTT Explorer (figure 9). Additionally, HTTPServer endpoint for GET request was registered, which allows the user to view messages through web interface (figure 10). User can view messages by performing *GET* request on: [ShellyIP]/script/[script_number]/testserver?messages



```
▼ localhost
  ▶ shellyplusplugs-80646fd64f90 (18 topics, 81677 messages)
  ▶ $SYS (51 topics, 220200 messages)
  ShellyTopic = 80:64:6f:d0:33:22##0##500##0##ShellyPlusPlugS-80646FD03320##-72##shellyplusplugs-80646fd64f90
  ▶ shellyplusplugs-80646fd03320 (14 topics, 3152 messages)
  ▶ shellyplusi4-083af2013470 (13 topics, 314 messages)
  ShellyTopic12 = 80:64:6f:d0:33:22##1075##0##2##ShellyPlusPlugS-80646FD03320##-35
```

Figure 9: Messages in MQTT explorer



```
08:3a:f2:01:34:72##805##700##0##ShellyPlusI4-083AF2013470##-82##shellyplusplugs-80646fd03320
80:64:6f:d6:4f:92##805##700##0##ShellyPlusPlugS-80646FD64F90##-73##shellyplusplugs-80646fd03320
08:3a:f2:01:34:72##805##700##0##ShellyPlusI4-083AF2013470##-86##shellyplusplugs-80646fd03320
80:64:6f:d6:4f:92##805##700##0##ShellyPlusPlugS-80646FD64F90##-74##shellyplusplugs-80646fd03320
80:64:6f:d6:4f:92##805##700##0##ShellyPlusPlugS-80646FD64F90##-74##shellyplusplugs-80646fd03320
```

Figure 10: Messages viewed through GET request

To subscribe to a MQTT topic, published by Shelly devices, MQTT Android Service library was used.

2.2 Android Bluetooth scan

To locate the user within the building, *RSSI* values of Shelly devices from users smartphone were also needed. To obtain this values, functions of BeaconScanner class were taken from existing Indoor Positioning System repository, Beacon-scanner project.

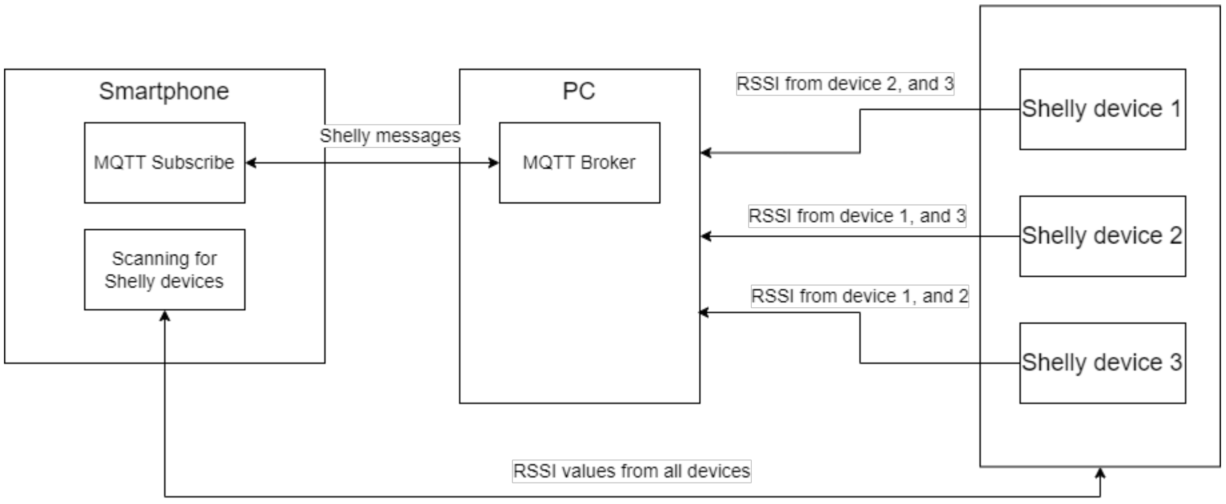

Smartphone
MQTT Subscribe
Scanning for Shelly devices
Shelly messages
PC
MQTT Broker
RSSI from device 2, and 3
RSSI from device 1, and 3
RSSI from device 1, and 2
Shelly device 1
Shelly device 2
Shelly device 3
RSSI values from all devices

Figure 11: Obtaining RSSI values

2.3 Indoor Positioning algorithm

To estimate the users location, interpolation algorithm was utilized. As seen in Section 1, the observed area was gridded firstly. For each Shelly device name, obtained through MQTT, *RSSI* values were assigned and interpolated. In my case Inverse Distance Weighting (*IDW*) was used. Lastly, values in each cell were compared against *RSSI* values obtained from smartphone device. The cell, where the sum of distances were the smallest, denoted users position. The overview of the algorithm is seen in figure 12

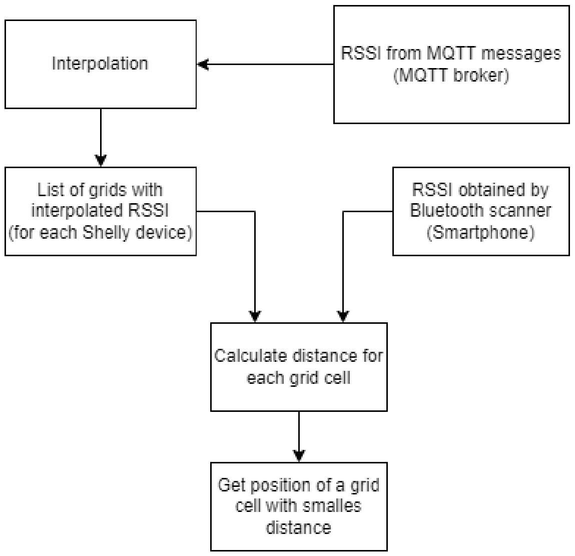

Interpolation
RSSI from MQTT messages (MQTT broker)
List of grids with interpolated RSSI (for each Shelly device)
RSSI obtained by Bluetooth scanner (Smartphone)
Calculate distance for each grid cell
Get position of a grid cell with smalles distance

Figure 12: Algorithm

3 Conclusion

For this coding challenge, the *IPS*, based on Shelly devices and Android smartphones, was implemented. Despite the time limit being less than two months, I manage to complete the project. Within this limit, most of the general and also some additional requirements were covered.

Documentation with Pictures

The whole project is well documented, within this paper. Additionally, visual aid in form of pictures and flowcharts is provided. To make sure the project is as reproducible as possible, links to youtube tutorials and used software are provided.

Error handling

In android project, try and catch statements are used. This mechanism allows to catch and handle possible errors that may occur during the code execution. Additionally, user friendly *UI* and messages are provided, allowing the user to understand the whats happening inside application.

Input validation

In Shelly script scanned data was validated for the appropriate device name, before publishing MQTT message to the broker. Additionally, the validation is implemented in the Android application. Potentially null or invalid values of the variables are validated before their usage.

Integration with External Services

Shelly script register HTTP Endpoint, which allows the user to access messages through *GET* request. Furthermore, the application can be integrated with external services using MQTT protocol.

Well-Written Code

I try to make code as clean and readable as possible. This was done by using consistent code formatting (indentations, position of the brackets) and naming style. Additionally, I tried to make function and class names descriptive. Complex sections of the code were explained within the comments.

GitHub Publication

Project was published on my GitHub repository: *https://github.com/JCUKI/Shelly-Christmas-Challenge*. For every new feature, new branch was made. This was done by using Gitflow. I tried to keep commit messages short and descriptive. Code additions were marked with "+", while deletions with "-". Minor code modifications were marked with "*".

Reproducible Results Instructions

This document (and README file) provide instructions on how to reproduce the project results. It also gives the necessary info on the hardware and software to make the installation and configurations possible.

Event Handling

Event handling is used multiple times through Android application. For example, event handlers are used to handle button clicks, scanning for Bluetooth devices and updating the *UI* from another thread.

Multiple Devices Usage

This project demonstrates the ability to work with multiple Shelly devices. It is also scalable and adaptable to other IoT devices. Moreover, this project can be extended to other protocols such as WiFi. *IPS* can also be used in a real-world environment by using sufficient number of Shelly devices.

User Interface and MQTT

This project includes the Android application. This allows the Android smartphones to communicate with Shelly devices through MQTT protocol. Additionally, Shelly Bluetooth functionality was included by scanning for other devices.