EngPhys 4US2

Project 2 Report

Abdullah Mir

400070827

Instructor: Dr. Fang

TA: Guha Ganesh

Introduction

An indoor positioning system, or IPS, is generally seen as a network of devices that track and locate objects in an enclose space, likely where other, larger scale tracking technologies would fail, such as satellite or GPS. They're often fairly high precision systems, a necessary feature considering the small areas in which they're implemented.

The IPS I've designed uses Bluetooth low energy, a form of wireless network connection technology similar to traditional Bluetooth, but retaining lower power consumption. I used the ESP32 microcontrollers to scan for a specific Bluetooth device, which acted as the tracking object.

IPS FLOW

On the following page, I've included a flowchart showing a simplistic explanation of how the system works. A small, Bluetooth speaker acts as the BLE tag being tracked by 3 microcontrollers placed in 3 separate bedrooms around my apartment. The microcontrollers scan for all available Bluetooth devices and isolate for the speaker; they then check and store the RSSI, or relative signal strength, value of the speaker. This RSSI value gives a representation of distance from a given microcontroller. The RSSI values are then sent by the ESP32s to NodeRed via MQTT protocol, where they are displayed in their raw forms and on a graph with respect to time. NodeRed also compares all 3 RSSI values to determine which scanner the tag is closest to, which then appoints the room in which it's located. The room location is also present on the NodeRed dashboard.

Bluetooth speaker acts as BLE tag that's tracked

3 separate ESP32 microcontrollers scan for available Bluetooth devices and isolate the tag [located in kitchen, bedroom, and living room]

Each ESP publishes the RSSI value as a string to NodeRed via MQTT

NodeRed converts RSSI strings to integer values and compares to find which ESP the tag is closest to, thus determining which room it's in

NodeRed dashboard displays RSSI values and which room the tag is in

Algorithm

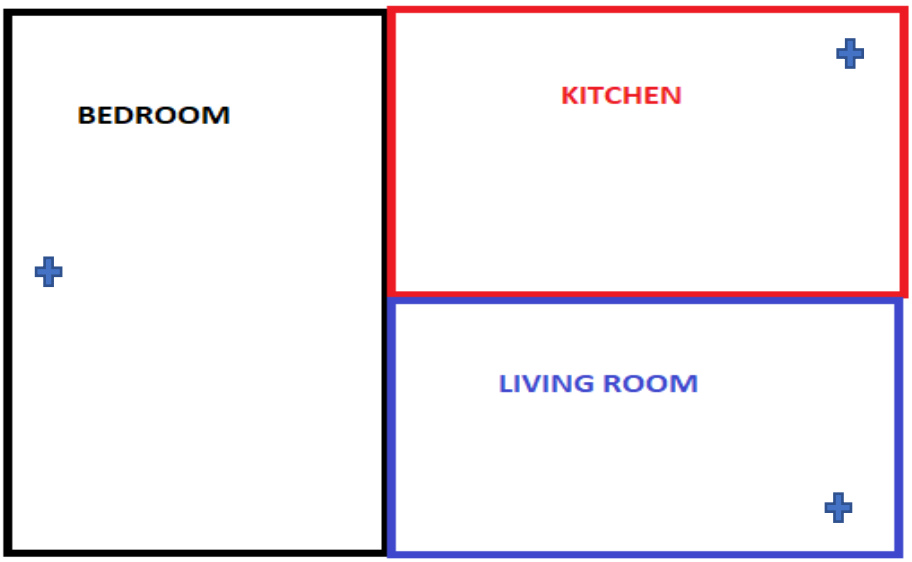


Figure 1: High-level floorplan. The crosses indicate the location of the microcontrollers acting as scanning beacons

My IPS uses a simplistic algorithm to determine location. NodeRed receives the RSSI values and compares them all to check which microcontroller the speaker is closest to, which then directly determines room the tag is in.

I initially started with planning to use boundary values to determine location. I tried to find the RSSI value from each microcontroller on the boundaries of the rooms, but I found it difficult to reproduce a given boundary value, so I decided to opt for a comparison of signal strengths instead as it requires less precision.

The algorithm I used is fairly dependent on the floorplan and the width and length of each room involved. I was fortunate enough to have a simplistic floor plan that allowed for a basic comparison between RSSI values to extrapolate tag location.

There is only one 'decision' involved in the algorithm, wherein the RSSI values are compared to see which of the strings that state room location to be output.

Unlike many of my peers' systems, I choose not to include any form of filtering in my RSSI value data collection. I looked at a few types, and found Kalman filtering to be the consensus optimal method for systems of this sort. However, I didn't want any form of averaging and correction of the present value based on previous values. This is also due to the floorplan in my system. Considering the rooms are directly adjacent, the room location is subject to instant changes, and I wanted to preserve any drastic changes in the RSSI values; averaging/filtering the values can greatly reduce efficiency.

There are 2 main drawbacks with my algorithm.

Firstly, the lack of filtering obviously makes the data collection system prone to noise. When initially making the decision, I planned on sacrificing some degree to accuracy (in the form of added noise) for increased efficiency. However, in testing my complete system, I've found noise to have a rare and negligible effect on the operation of the system, so for the given application it appears there wasn't much accuracy sacrificed.

The second drawback is that decision algorithm itself is not optimized for a cuboid-style floorplan, like the one in my apartment. The RSSI values act as a function of the radius around each microcontroller, and thus this algorithm would be far more effective for circular rooms. However, there is a thick wall located between the bedroom and kitchen/living room are in my apartment. The wall aids in reducing signals strength through the wall greatly, enough such that even if the speaker is closer to the beacon on the other side of the wall, it'll still show a lower RSSI value just due to the interference of the wall. This allowed for the algorithm to work effectively in my apartment and increased accuracy greatly compared to what you'd expect for a cuboid floorplan.

Calibration and Optimization

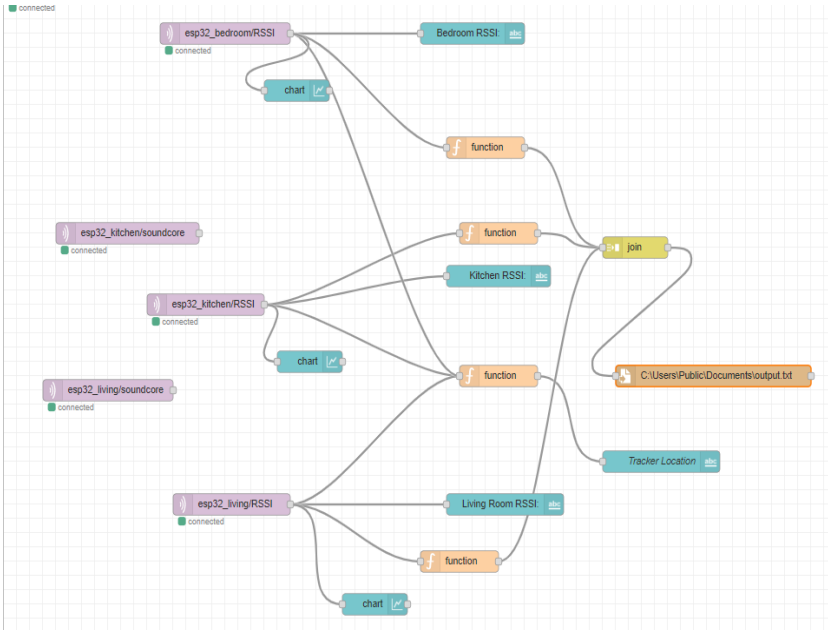
There wasn't much calibration and optimization required for my system. Considering the comparison algorithm, I didn't have to calibrate exact boundary values, and the decision is dependent on all 3 RSSI values rather than a specific RSSI value and a preset, calibrated threshold.

I tried optimizing how quickly the system could see a change in tag location by reducing the amount of time the beacons scanned for Bluetooth devices. However, in doing this, I found that the beacons wouldn't always find the speaker in time when it was further away. As such, I had to keep the scanning time fairly long.

Design Process/Troubleshooting

The design process occurred over the entirety of project 2. Every week/tutorial, I was able to learn about another necessary component required for the IPS. This eventually all culminated in me having a basic program already being written for each component.

Each microcontroller had generally the same code. They each had the WiFi and MQTT details set up the same, and used libraries I found online to connect to both of these networks. They then scanned for all Bluetooth devices for a preset amount of time; the speaker was isolated by checking the MAC address of each Bluetooth device and see if it matched the one I'd preset in the code. Once the speaker was isolated, the RSSI value was stored and published to the NodeRed server, with each ESP publishing to a different topic.



The NodeRed flow above is the one used for my system. The RSSI values are displayed on the dashboard directly, and are also sent to a function node that determines which is highest and outputs a string stating which room the tag is in; the string is also displayed on the dashboard. The RSSI values are also written to a text file in a clear manner. The NodeRed design process was fairly simplistic for the most part, thanks to the easy UI and intuitive nodes of the platform. The only difficult part was learning how function nodes work and how to use the 'context' mechanism in NodeRed to be able to access previous values of RSSI. I read through the tutorials for each of these in NodeRed's website, and found example codes in various forums online to help me in implementing them into my system.

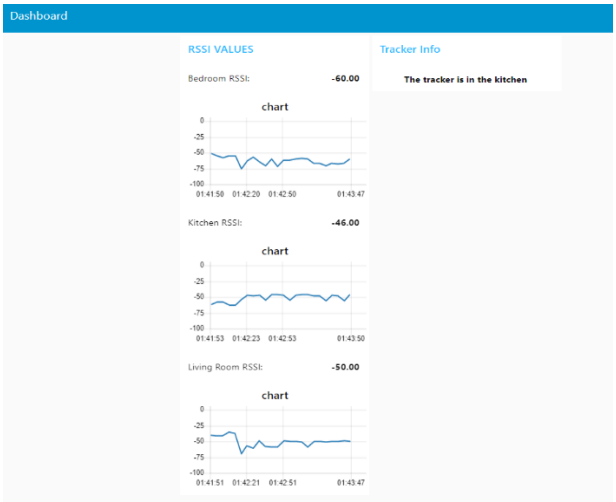
Troubleshooting took up a large portion of the time dedicated to this project. I initially had trouble in finding the speaker consistently. I was first comparing each Bluetooth device's name to a name I had preset, and I found that oftentimes the name wouldn't always be transmitted by the Bluetooth speaker. I then changed to matching the MAC address ,which worked flawlessly. I also had to ensure the Bluetooth speaker stayed on, as it had an integrated kill switch if nothing was connected to it for more than a few minutes. I connected my phone to it and played a 10 hour long video of just silence to ensure it'd stay on without being a disturbance.

I then had trouble with the MQTT server. For some reason, NodeRed would connect to the server but the microcontrollers could not. I found that starting MQTT in 'verbose' mode helped overcome this, thanks to advice from the TA.

Lastly, I had to troubleshoot the decision-making function node in NodeRed. I found that writing a simple code that compared RSSI values didn't work, as the node would then wait for all 3 microcontrollers to send an input before making the decision. I wanted to make the comparison everytime a new RSSI value was sent and compare it to the last RSSI value sent from the other 2 beacons. This was overcome by using the aforementioned 'context' mechanism that allowed me to store previous values.

User Interface/Date Collection

The data is collected by each microcontroller. The only relevant data is the RSSI value, and the process of collecting this was outlined in the *Algorithm* section.



The picture above shows the UI displayed by NodeRed. The string on the right indicates which room the speaker is in. It takes around 3.5 seconds for the room location to update after the tag enters the room. The delay is almost entirely due to the scanning time of the scanning beacons. This could be lowered by implementing some sort of interrupt/break once the desired device is found, so the scanners don't have to continue scanning for the entirety of the scanning time every iteration.

I chose to include each RSSI value, a graph of RSSI over time, and the string stating which room the tag is in on the dashboard. Given more time, I would include a distance measurement rather than RSSI so the graphs and raw values are more intuitive for any user. I would also include push notifications when the location changes.

Conclusion

This project showed me how achievable, interesting, and useful the technologies used in this system are. Prior to this, I would've expected making an indoor positioning system to require 8+ months of learning and working to produce, but I was able to produce this in merely a few weeks.

I was able to learn about Bluetooth technology, NodeRed, MQTT, and get more familiar in using microcontrollers due to this project. This is all knowledge I can apply in a variety of ways, and I feel that I have a better grasp in how to use them in any way I want, regardless of not having too much experience with it. This is because in learning how to do this, we started from the ground up, and learnt the basics and fundamentals and built upon them. This will allow us to easily translate the knowledge into other projects.

Indoor positioning systems are an important technology, already used in a wide variety of applications such as security, monitoring, and even retail. Despite the project I made being fairly simplistic, it is clear that implementing an IPS is fairly easy, even in prebuilt spaces. They are easy, cheap, and intuitive to use and I'm sure I will see more of them in the world around me.