Building an occupancy sensor with a $5 ESP32 and a serverless DB

created on 2023-10-20

Have you ever wanted to design a full end-to-end software solution to collect occupancy data across a college campus?

I didn't think I would either, but here we are.

The inspiration

During my first year in college, we had Sodexo as our dining provider. They had a contract with Bluefox, who provides occupancy sensors to report the number of people within a dining hall. I'd like to think they used this data to improve dining hall operations. I couldn't tell you what they *actually* used it for. I can say that after some FOIA requests a friend made that returned PDF's with awful kerning, these devices work by counting smartphone MAC addresses from Bluetooth advertising packets. This was a pretty cool way for me to avoid crowds in the dining hall - toss the API call into Grafana, and you have a live chart of how busy the dining halls are.

The downfall

Unfortunately, the university switched dining hall providers to Aramark. Aramark does not contract Bluefox to provide the same occupancy counts, which meant no more occupancy data, which meant no more skipping busy hours.

The climb back

The idea of tracking occupancy metrics with a bluetooth beacon was stuck in my head. What design decisions and considerations would you need to make?

- How accurate is BLE beacon count, as a proxy for occupancy?
  - Some people carry around headphones, smartwatches, etc - but some don't carry any devices at all (or keep Bluetooth off on their phone).
- How accurate is BLE beacon availability time, as a proxy for dwell time?
  - Can we use unique MAC addresses' churn to detect this?
  - Is the built-in MAC address randomization that is common across many different manufacturers going to impact this? (Even though this privacy feature has flaws)
- How can we communicate the results back to a central server?
  - WiFi seems the obvious choice for me. Not every location will have easy-to-access WiFi, though.
  - LoRa could be an option, depending on the distribution of beacons. Here's some range testing numbers, though this is affected by the antenna's gain and locations (here's another test with a far higher range).
- How can we collect the data?
  - Should we use a time series database?
- How can we analyze the data?
  - Can we predict trends in the longer time spans, excluding special events like homecoming weekend, finals week, etc?

I found these questions tumbling around in my head, so I began with some preliminary testing - writing some simple code to count the number of devices detected on my laptop's Bluetooth adapter. Success - it was surprisingly easy to write code to scan for `x` seconds every `y` seconds and save it to a SQLite database at regular intervals. So, I carried my laptop to dining halls, Chick-Fil-A, Starbucks, etc and waited.

And waited.

And waited.

I spent a lot of time collecting data while sipping on coffees and milkshakes. You know, for data collection purposes. This is all very academic, of course.

No other reason.

Anyway, accuracy - In smaller areas (like a single-room Starbucks), I found the count to be pretty accurate. At the very least it reflected trends in occupancy very quickly. When more people arrived, the charts quickly climbed.

In larger areas like dining halls, the counts seemed accurate by my guesstimate. There's no way for me to count everyone in a dining hall, with complicated layouts and different seating areas, especially when I'm not certain of my bluetooth adapter's range (is it picking up people on the terrace through the walls? etc). But it most definitely matched the trends around class changes - when classes got out, people went to eat, which rapidly increased the number of people I saw, and the number of beacons my laptop detected.

Long term deployment

Okay, sounds great, it looks like we have some kind of method validation. But I don't plan on cloning myself in every dining hall and sitting 24/7, so what can we do to create a small device to collect the same data?

Raspberry Pi - Maybe?

My first thought was - Raspberry Pi Zero W. It's small and cheap, has Wi-Fi and Bluetooth, and definitely is in stock somewhere on Earth.

I rewrote my simple code (in Rust, no less!) to handle everything gracefully (reboots, no network, adapter loss, etc). Linux Bluetooth is incredibly painful to handle in a headless way. Binding to DBus requires cross-compiler magic and not even Cross was getting me out of it. After struggling enough through a million different compiler flags, a power outage that caused me to lose my progress on the Makefile (also, yes I use Make with Rust), and at last setting up a QEMU bridge, I was able to get my binary to run on my Pi. I even wrote all of the patching magic to make it connect to Wi-Fi (try doing THAT headlessly, when there's a portal you have to sign into!), install the necessary libraries on start-up, make a service to run my executable, and automagically update when I push a new update. Okay, that's a lot of moving parts. Let's boot it up and hope it works...

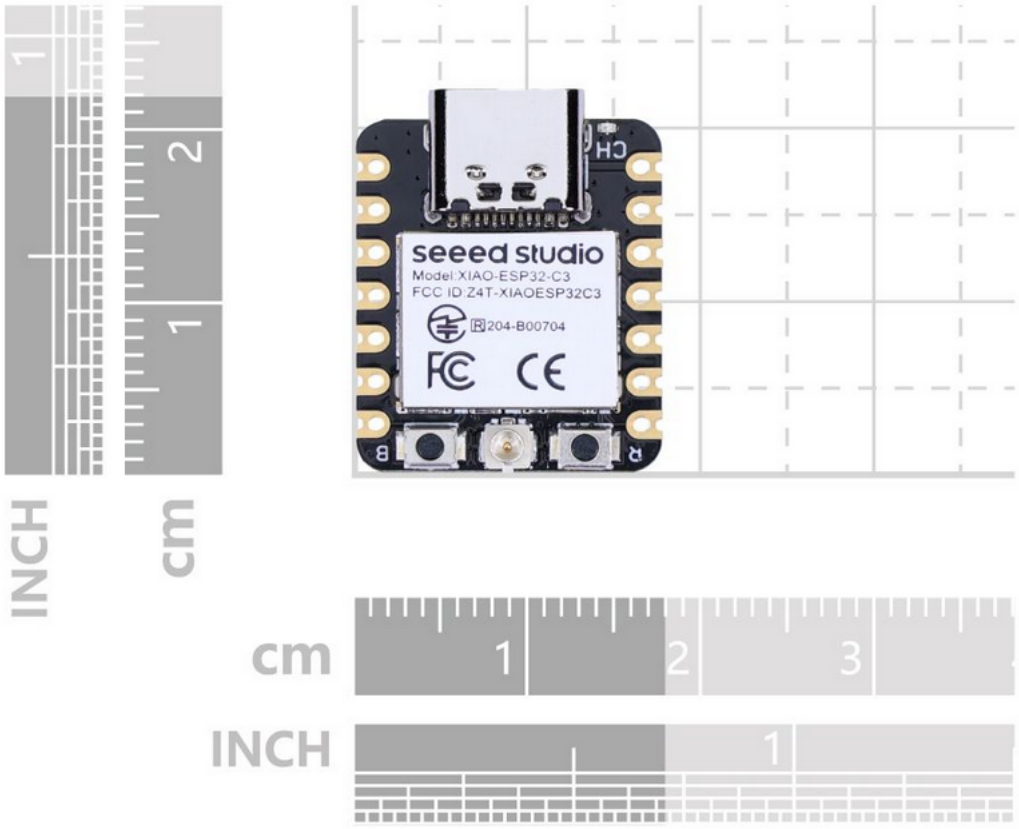
Nothing.

That's right, absolutely nothing worked. Not even the automagic wifi connection via Mac address fiddling hacks.

Moving on

If you're smarter than me, you may have realized that's ~~a bit~~ WAY too much complexity. We really do not need a whole Linux kernel at all. We need two things - reliable Wi-Fi, and reliable Bluetooth. Okay, so shelve the Pi Zero W and Orange Pi Zero W I bought. What's this nonsense about a device that can do these two things at an even cheaper price and smaller footprint?...

ESP32?



On paper, it looks great - Wi-Fi, Bluetooth, extremely low power usage (🌱🌍♻️🌿🌞💚), very cheap, and very tiny.

I purchased one off of Amazon since I didn't want to wait for overseas shipping (not losing momentum in a nerd snipe like this is **CRITICAL**). I bought a random ESP32-WROOM-32 with an OLED display, since I thought it would be cool to display the data on the screen live. I rewrote my data collection code in C++ form (away from Rust!) since the Rust ecosystem for ESP32 is not all the way there yet.

After fidgeting with the display code enough to get it working (`SSD1306Wire display(0x3c, 5, 4);` if you're wondering), it worked great. I asked campus IT to whitelist the MAC address, wrote up some Cloudflare functions into a D1 database as my data ingest, and set out to work.

Deployment

I ~~hid~~ placed my data collection device in my campus library on a crisp fall morning, sat myself down at my laptop, saw the data rolling in, and did a silent celebration. Off to my Principles class to learn about scope rules then...

Why did everyone leave Swem library and never come back???

Obstacles

One major problem that I ran into was the poor specs of the random ESP32 device I had.

The above issue shows the device crashing at about 250 devices. At first, I was worried this was a bug with the result count being stored in a 1-byte number, like a u8 (thus capping at ~255 devices). A quick `Serial.print` made me realize it would crash also at about 249, 265, etc - randomly around this area. So, not a bit-overflow issue (at least, not in the integer part!).

Our library fills up quickly with studious twamps - it wouldn't last a minute in finals season if it couldn't handle any more than that.

The problem - identified

By saving the results during a scan into a data structure until the end of the scan, it piled up data about the device's scan strength, advertised services, manufacturer ID, etc. While this seems like great data to collect, I had one thing in mind - the number of unique devices (for now).

By debugging the heap size constantly, I realized that the scan results was filling up the small amount of RAM it had. This was bad news - at first, I didn't see a way to still collect the data while not actually collecting the data.

Resolution

I decided to brush up on my C++ data structures programming and write a small hashset. After all, the data structure for the scan results was very bloated - if I could override that, I would be able to control the heap size more closely, right?

On every callback, then, we'd insert the MAC address into a hashset, then clear the built-in result structure to allow for more memory.

```
// Callback for a new scan result
class MyAdvertisedDeviceCallbacks : public BLEAdvertisedDeviceCallbacks {
  Codeium: Refactor | Explain | Generate Function Comment
  void onResult(BLEAdvertisedDevice advertisedDevice) {
    // Add to set.
    addToSet(advertisedDevice.getAddress().getNative());

    // Display results
    display.clear();
    display.setTextAlignment(TEXT_ALIGN_CENTER);
    String numberString = String(scanResults);
    display.drawString(64, 22, numberString);
    display.drawString(64, 42, "devices found");
    display.display();

    // Clear results. The results will grow heap to extreme sizes.
    // I'll manaually manage these in a set, thank you!
    // This will create churn as the callback will be called
    // thousands of times, but the alternative is >250 devices cap,
    // which is not what we want.
    pBLEScan->clearResults();
  }
};
```

Unfortunately, this is not ideal - **if and only if** there's some results to check for duplicates, the callback is only called on new devices. When we clear this every time, we give it amnesia, thus every single BLE advertisement packet causes a callback. We *do* check for duplicates in `addToSet` (it is a hashset!), but this will definitely cause hundreds of duplicate callbacks to our hashset, *and* heap thrashing since we're allocating and deallocating the result structure every single callback. That's okay (for now), it's better to repeatedly check a hashmap with no more than 1000 entries (a very quick procedure) very often, than have our capacity limited to 250 people.

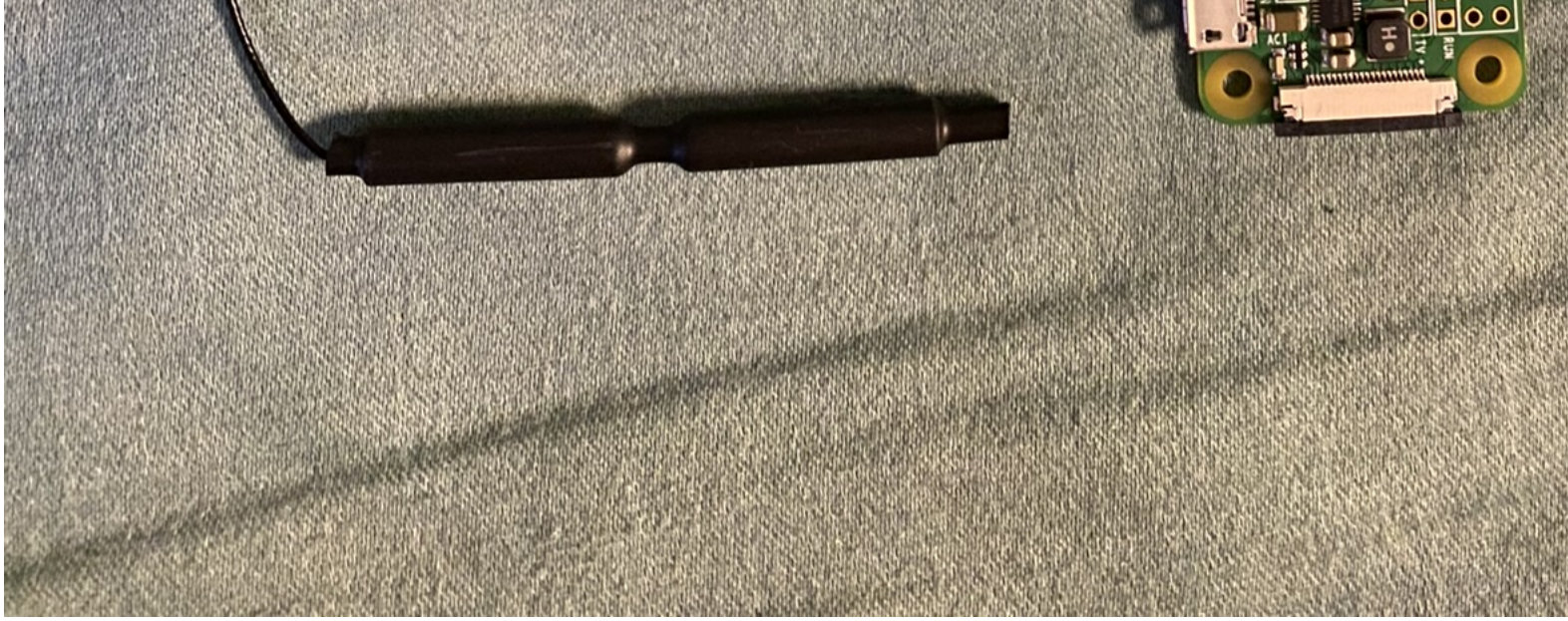
More obstacles

Okay, we now have a perfect way to scan for devices for long periods of time. Oh would you look at the calendar - it's fall break! I'll leave it in the library and have it report back so I can see how packed it is over break (yes, there will be studying done on campus over break - we are a nerdy college). Perfect, even some long-term testing over the 5 day weekend! Surely nothing bad will happ-

That's about 400 devices before the crash - on a fall break day in a college. As mentioned - nerdy.

Okay, awesome. Another issue, one that manifests itself after 3 hours of use with zero indication of issues. After fighting with the debugger for long enough, and even tossing in periodic reboots (it boots very quick so this adds almost no time at all), I chalked this up to a bad board/BT adapter - it seemed to run, but it instantly returned zero devices on every scan. That's okay - while I was messing with this Amazon one, I bought quite a few others, along with the rest I've bought over the course of the whole project.

XIAO ESP32S3
113991114
SEEED STUDIO MOB2303150012
Made in China 05/24/23
elfatmaster Consulting GmbH
EU REP Bethmannstr. 30,60325 Frankfurt am Main,Germany
contact@eufmaster.com
ATTENTION
Best to keep away from fire.
Innovate with China
seeed studio
seeed studio
Model:XIAO-ESP32-S3
FCC ID:Z4T-XIAOESP32S3
MICRO SD CARD
H618
Cdtech
20U5622



Seeed Studio XIAO ESP32S3/C3, WaveShare ESP32S3 Zero, Unbranded ESP32-WROOM with OLED, Orange Pi Zero W (untouched), Raspberry Pi Zero W (L->R, T->D)

After testing all of these, the only one reliable to work for long periods of time (one month currently) was the XIAO ESP32C3/S3. Both work acceptably, but I decided to go with the C3, since it's RISC-V, which is awesome (for my ideals), and since it's cheaper, which is also awesome (for my wallet).

Another benefit of switching to a better manufacturer is more SRAM - I was able to switch away from my hand-written hashmap implementation, since the RAM was able to hold the results data structure much better without crashing. I've seen as high as 1000 devices detected with no sign of slowing down. This probably reduces CPU usage as well - no more heap and callback churn!

After I found a device that worked far better, I moved my deployment location to my dorm room window so I would have an easier deployment cycle - here it is with numerous academic buildings in the background.


seeed studio
Model:XIAO-ESP32-C3
FCC ID:Z4T-XIAOESP32C3
R204-B00704
CH

My RISC-V based ESP32C3, scanning from my dorm room window in front of Washington Hall, during homecoming weekend.

Final data collection

Now that I've gotten my data collection working successfully, let's look at the data for one day.

Data collection for one day. Notice the peaks? That's right about the time that classes switch.

There's something to note about this. The device might be in my dorm, but it largely is not limited to the dorm's own inhabitants. Otherwise, it would be at its max in the early morning and drop as the day went on. If I wanted it to measure exclusively dorm inhabitants, I would probably place it more centrally (instead of out a window), but it still would not be great at this, since dorms are the *worst* at permeating bluetooth signals through many walls.

That's okay, I'll just keep that in mind as I analyze the data - it's mostly picking up students as they go into the two nearest academic buildings, not the dorm inhabitants.

The peaks start up around 7:50, right before the 8 am classes start in Ewell and Washington halls. I suspect this is detecting the students initially leaving dorms, shuffling along to their classes, and the drop is as students enter the buildings. Then, the peak at 8:50 is probably as students leave their 8 AMs and go to their 9 AMs, entering the range of the device only to immediately leave it as the numbers drop at 9 AM. Same for 9:50/10 AM, and 10:50/11 AM. These are all class switch times.

These all point to method validation - it seems like this device really is good at tracking trends in the movement of students around it. The antenna is also seemingly very high range - I didn't expect it to reach the ~160ft into Washington Hall, and the ~100ft into Ewell. The altitude of being on the 3rd floor probably helps with it, though.

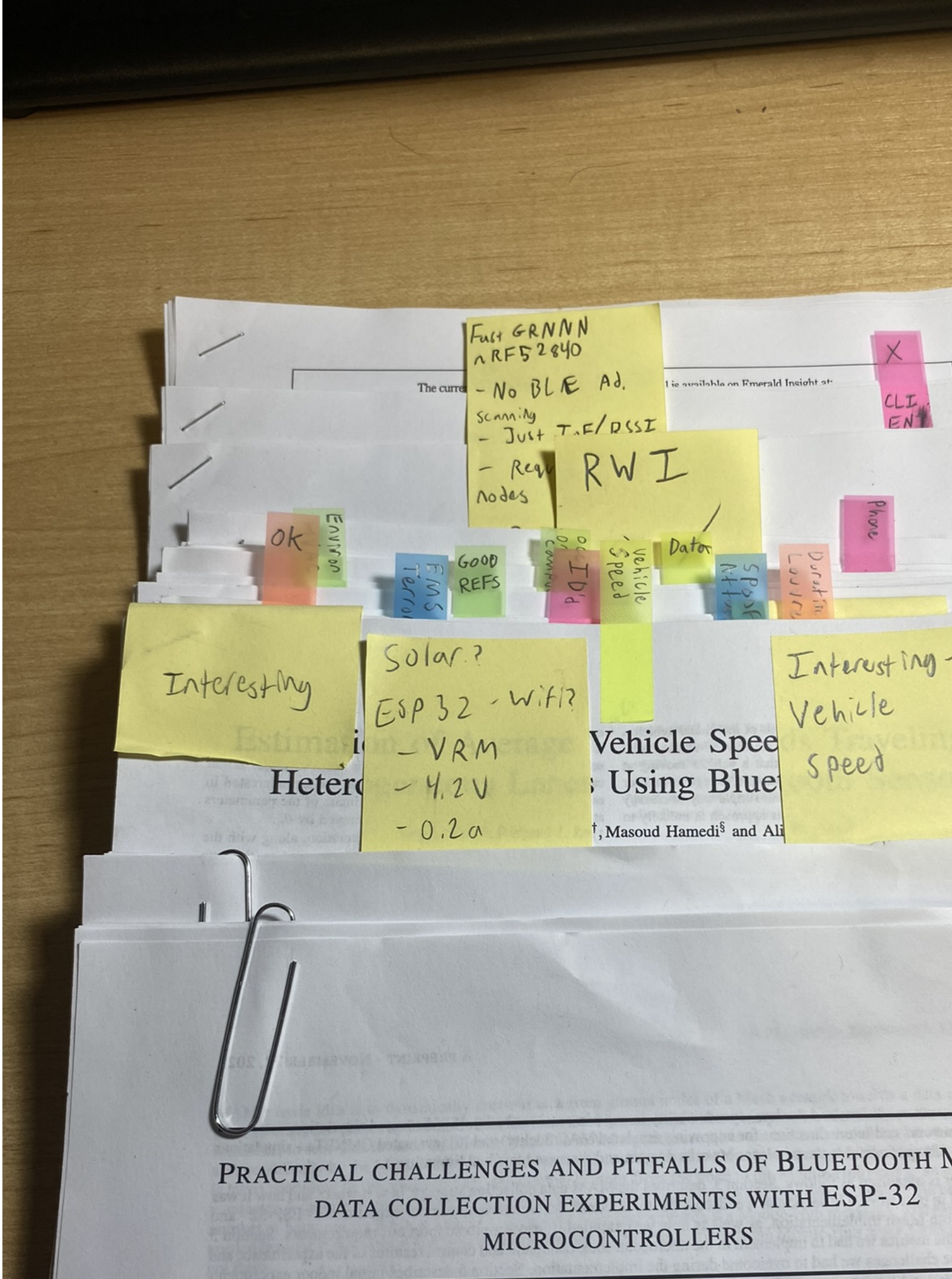
Time series forecasting?

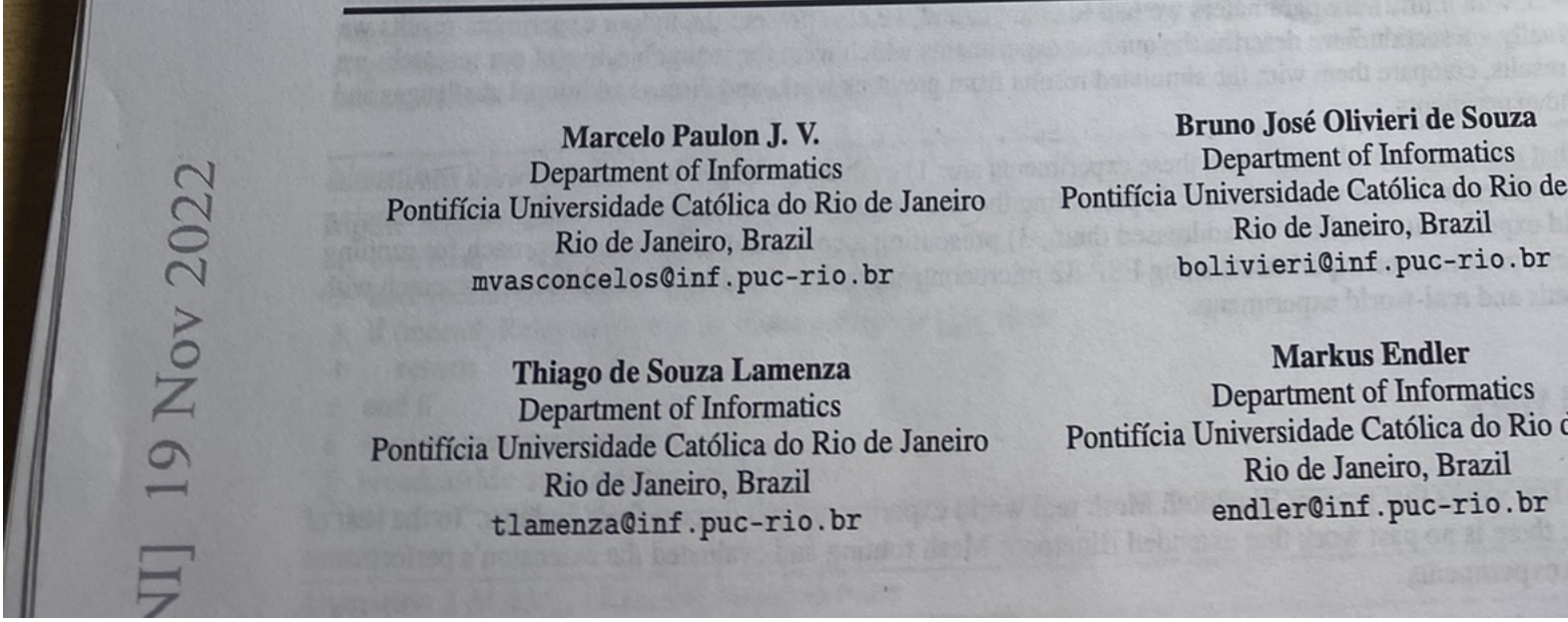
This seems like the perfect target for time series forecasting, like with NeuralProphet. The data is chock-full of hourly, daily, and weekly trends. I added the functionality to predict these trends and so far, it's very good at predicting daily trends; the longer (week, month, and season-long) trends will likely converge after enough data is collected.

Further thoughts

Of course, this is not a solved project. I've written the code to parse this into a Cloudflare DB, into Grafana, and some forecasting, but there's more to be done.

I performed an enormous amount of literature review over the course of this project - reading dozens of research papers on the topic, finding out what works and what didn't, what I would try and what I would avoid. Many questions are raised in these papers, that I still want to answer.


Fast GRNN
on RF52840
- No BLE Ad.
scanning
- Just T.F/RSSI
- Req
nodes
RWI
X
CLI
EN
Phone
Data
vehicle
speed
GOOD
REFS
EMS
Terro
ID'd
Environ
ok
Interesting
Solar?
ESP32 - WiFi?
- VRM
- 4.2V
- 0.2a
Interesting -
Vehicle
Speed
Vehicle Spee
Using Blue
Masoud Hamedi§ and Ali
PRACTICAL CHALLENGES AND PITFALLS OF BLUETOOTH M
DATA COLLECTION EXPERIMENTS WITH ESP-32
MICROCONTROLLERS



Hours of doc review

For example, how well of a proxy is a BLE beacon count for actual population count?

- Does this depend on the demographics? How do we find a correction factor (like for `x` beacons, there's roughly `0.7x` people, given the multiple devices people carry around)?
- For example, in the computer science building, is the linear rate to population higher, since we carry so many gadgets around? Or is it lower, because we know to turn off Bluetooth when not using it (BT firmware zero days, you see)?
- Or in the staff building, is it lower compared to students, since they are less likely to carry around a bunch of tech?
- Maybe the rate is lower in a dining hall, since not many people are using multiple devices like they would in a lecture hall (laptop, iPad, etc for notes)?

Other questions came to mind, like:

- Can we improve the accuracy by setting an RSSI minimum, for which devices weaker than it do not count, to ensure only those who are really nearby get counted?
- Can we improve the accuracy by filtering by manufacturer ID, so it's only Apple + common Android manufacturers? Would this help, since Apple Watches, AirPods, MacBooks, etc would all still add to the count?
- What kinds of privacy accomodations do I need to keep in mind? I already only track pure beacon numbers. I don't track actual MAC addresses like Bluefox did, but do I need to add noise to the data? Is the existing data noisy enough to avoid deanonymization? Is it realistic to identify a single person in the data without anything but the number of devices? What's the best scan duration? Too quick, and it won't find all of the hundreds of devices that exist. Too long, and we risk inaccuracy (counting devices that have since left, data showing large drifts within short periods of time, etc).
    - Is a dynamic scan length best? (Think about cooking a bag of popcorn - when a period of time passes without any change, you're done)

Lots of questions to answer. I plan on validating my data with real-world population data collected in a place where it's easy to get the "ground truth". Maybe I will reach out to a place on campus who either already tracks occupancy (the gym with swipe in/out), or will investigate more thoroughly in a place where it is trivial to do so myself (limited entry/exit, like a dining hall, or a Starbucks).

Further work

I am not sure whether I will be taking this further - I'm currently talking to some professors about the use cases for some university committees, or perhaps further academic (and hopefully publish-worthy) research. I am also considering selling it to brick-and-mortar businesses that want to measure occupancy trends. It's a pretty packaged up solution - everything from the front-end to the back-end is built already. This is vaguely what the setup looks like for any given deployment:

- Set up Wi-Fi in config (either have network whitelist the MAC if it's a portal, or connect to open/password protected Wi-Fi on boot)
- Change machine and site IDs in config
- Plug device/devices into outlet in central/convenient locations
- Set up Grafana dashboard to read each device from backend
- Set up Grafana dashboard to read predicted trends as well, in separate charts

If you're interested in any of this, please let me know! I'd love to hear from you.

Conclusion

I hope you enjoyed reading my blog post about analyzing occupancy trends and embedded development. If you have any suggestions, comments, questions, or angry fists, you can email me at maesposito@wm.edu.

> Think this was cool? Hiring software engineering interns for Summer 2024? Check out my resume here. I'm very passionate about software development (I did all of this without the promise of *any results* - all in my spare time because I loved doing it!), and I'm always ready to embark on new coding adventures.

Bibliography

Ahmad, J., Larijani, H., Emmanuel, R., Mannion, M., & Javed, A. (2020). Occupancy detection in non-residential buildings – A survey and novel privacy preserved occupancy monitoring solution. *Applied Computing and Informatics*, *17*(2), 279–295. https://doi.org/10.1016/j.aci.2018.12.001

Apolónia, F., Ferreira, P. M., & Cecílio, J. (2021). Buildings Occupancy Estimation: Preliminary Results Using Bluetooth Signals and Artificial Neural Networks. In M. Kamp, I. Koprinska, A. Bibal, T. Bouadi, B. Frénay, L. Galárraga, J. Oramas, L. Adilova, Y. Krishnamurthy, B. Kang, C. Largeron, J. Lijffijt, T. Viard, P. Welke, M. Ruocco, E. Aune, C.

Gallicchio, G. Schiele, F. Pernkopf, … G. Graça (Eds.), *Machine Learning and Principles and Practice of Knowledge Discovery in Databases* (pp. 567–579). Springer International Publishing. https://doi.org/10.1007/978-3-030-93733-1_42

Baronti, P., Barsocchi, P., Chessa, S., Mavilia, F., & Palumbo, F. (2018). Indoor Bluetooth Low Energy Dataset for Localization, Tracking, Occupancy, and Social Interaction. *Sensors*, *18*(12), Article 12. https://doi.org/10.3390/s18124462

Barsocchi, P., Crivello, A., Girolami, M., Mavilia, F., & Palumbo, F. (2017). Occupancy detection by multi-power bluetooth low energy beaconing. *2017 International Conference on Indoor Positioning and Indoor Navigation (IPIN)*, 1–6. https://doi.org/10.1109/IPIN.2017.8115946

Billah, M. F. R. M., & Campbell, B. (2019). Unobtrusive Occupancy Detection with FastGRNN on Resource-Constrained BLE Devices. *Proceedings of the 1st ACM International Workshop on Device-Free Human Sensing*, 1–5. https://doi.org/10.1145/3360773.3360874

Chen, Z., Jiang, C., & Xie, L. (2018). Building occupancy estimation and detection: A review. *Energy and Buildings*, *169*, 260–270. https://doi.org/10.1016/j.enbuild.2018.03.084

Demrozi, F., Turetta, C., Chiarani, F., Kindt, P. H., & Pravadelli, G. (2021). Estimating Indoor Occupancy Through Low-Cost BLE Devices. *IEEE Sensors Journal*, *21*(15), 17053–17063. https://doi.org/10.1109/JSEN.2021.3080632

Ding, Y., Han, S., Tian, Z., Yao, J., Chen, W., & Zhang, Q. (2022). Review on occupancy detection and prediction in building simulation. *Building Simulation*, *15*(3), 333–356. https://doi.org/10.1007/s12273-021-0813-8

Dodier, R. H., Henze, G. P., Tiller, D. K., & Guo, X. (2006). Building occupancy detection through sensor belief networks. *Energy and Buildings*, *38*(9), 1033–1043. https://doi.org/10.1016/j.enbuild.2005.12.001

Feng, C., Mehmani, A., & Zhang, J. (2020). Deep Learning-Based Real-Time Building Occupancy Detection Using AMI Data. *IEEE Transactions on Smart Grid*, *11*(5), 4490–4501. https://doi.org/10.1109/TSG.2020.2982351

Filippoupolitis, A., Oliff, W., & Loukas, G. (2016). Occupancy Detection for Building Emergency Management Using BLE Beacons. In T. Czachórski, E. Gelenbe, K. Grochla, & R. Lent (Eds.), *Computer and Information Sciences* (pp. 233–240). Springer International Publishing. https://doi.org/10.1007/978-3-319-47217-1_25

Mashuk, M. S., Pinchin, J., Siebers, P.-O., & Moore, T. (2018). A smart phone based multi-floor indoor positioning system for occupancy detection. *2018 IEEE/ION Position, Location and Navigation Symposium (PLANS)*, 216–227. https://doi.org/10.1109/PLANS.2018.8373384

Meyn, S., Surana, A., Lin, Y., Oggianu, S. M., Narayanan, S., & Frewen, T. A. (2009). A sensor-utility-network method for estimation of occupancy in buildings. *Proceedings of the 48h IEEE Conference on Decision and Control (CDC) Held Jointly with 2009 28th Chinese Control Conference*, 1494–1500. https://doi.org/10.1109/CDC.2009.5400442

Oliff, W., Filippoupolitis, A., & Loukas, G. (2017). Evaluating the impact of malicious spoofing attacks on Bluetooth low energy based occupancy detection systems. *2017 IEEE 15th International Conference on Software Engineering Research, Management and Applications (SERA)*, 379–385. https://doi.org/10.1109/SERA.2017.7965755

Pratama, A. R., Widyawan, W., Lazovik, A., & Aiello, M. (2018). Multi-User Low Intrusive Occupancy Detection. *Sensors*, *18*(3), Article 3. https://doi.org/10.3390/s18030796

Rahaman, M. S., Pare, H., Liono, J., Salim, F. D., Ren, Y., Chan, J., Kudo, S., Rawling, T., & Sinickas, A. (2019). OccuSpace: Towards a Robust Occupancy Prediction System for Activity Based Workplace. *2019 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*, 415–418. https://doi.org/10.1109/PERCOMW.2019.8730762

Rueda, L., Agbossou, K., Cardenas, A., Henao, N., & Kelouwani, S. (2020). A comprehensive review of approaches to building occupancy detection. *Building and Environment*, *180*, 106966. https://doi.org/10.1016/j.buildenv.2020.106966

Sayed, A. N., Himeur, Y., & Bensaali, F. (2022). Deep and transfer learning for building occupancy detection: A review and comparative analysis. *Engineering Applications of Artificial Intelligence*, *115*, 105254. https://doi.org/10.1016/j.engappai.2022.105254

Shen, W., & Newsham, G. (2016). Smart phone based occupancy detection in office buildings. *2016 IEEE 20th International Conference on Computer Supported Cooperative Work in Design (CSCWD)*, 632–636. https://doi.org/10.1109/CSCWD.2016.7566063

Sikeridis, D., Papapanagiotou, I., & Devetsikiotis, M. (2019). *BLEBeacon: A Real-Subject Trial Dataset from Mobile Bluetooth Low Energy Beacons* (arXiv:1802.08782). arXiv. https://doi.org/10.48550/arXiv.1802.08782

Tekler, Z. D., Low, R., & Blessing, L. (2019). An alternative approach to monitor occupancy using bluetooth low energy technology in an office environment. *Journal of Physics: Conference Series*, *1343*(1), 012116. https://doi.org/10.1088/1742-6596/1343/1/012116

V., M. P. J., de Souza, B. J. O., Lamenza, T. de S., & Endler, M. (2022). *Practical Challenges And Pitfalls Of Bluetooth Mesh Data Collection Experiments With Esp-32 Microcontrollers* (arXiv:2211.10696). arXiv. https://doi.org/10.48550/arXiv.2211.10696

Valks, B., Arkesteijn, M. H., Koutamanis, A., & den Heijer, A. C. (2021). Towards a smart campus: Supporting campus decisions with Internet of Things applications. *Building Research & Information*, *49*(1), 1–20. https://doi.org/10.1080/09613218.2020.1784702

Yoshimura, Y., Krebs, A., & Ratti, C. (2017). Noninvasive Bluetooth Monitoring of Visitors' Length of Stay at the Louvre. *IEEE Pervasive Computing*, *16*(2), 26–34. https://doi.org/10.1109/MPRV.2017.33

Zim, M. Z. H. (2021). *TinyML: Analysis of Xtensa LX6 microprocessor for Neural Network Applications by ESP32 SoC*. https://doi.org/10.13140/RG.2.2.28602.11204

Zoto, J., La, R. J., Hamedi, M., & Haghani, A. (2012). Estimation of Average Vehicle Speeds Traveling on Heterogeneous Lanes Using Bluetooth Sensors. *2012 IEEE Vehicular Technology Conference (VTC Fall)*, 1–5. https://doi.org/10.1109/VTCFall.2012.6399146