Urinalysis

1 Abstract

This work is a proof of concept of a urinal availability system, which displays the current state of a set of urinals in real-time. To estimate the occupancy of the urinals only two motion sensors, which act like a light-barrier, and a set of probability calculations are used. An ESP32 is used as the edge operator, a Raspberry Pi Zero W as the gateway and a local machine with a Node.js client running as the backend. The state of the urinals is displayed on a website and updated in real-time. The communication between the devices is done with wiring, Bluetooth Low Energy, MQTT and Socket.IO.

2 Introduction

Men generally think in three states of availability for urinals: Fully occupied, bro code breaking and urinals available. The bro code is to leave one space between you and others and is ingrained in most men's brains. When the bro code cannot be respected, many will choose to wait until a person leaves. While this bro code may seem silly, there are many real-world applications for it. Many men are not comfortable peeing when the urinal directly next to them is occupied, which is a common case of Paruresis or "shy bladder".
Urinalysis (urinal + analysis) was designed to display the state of availability of the urinals outside of the restroom, to avoid the awkward waiting time after checking to see if there's any free urinal that you're comfortable with.
To determine if there's unoccupied urinals that still respect the bro code if occupied, the exact occupancy of all the urinals is required. This is because with five urinals, you can completely block them with only two people, by going to the second and fourth urinal, but on the other hand, you could have three people at the urinals and still have one bro code respecting urinal left, namely when all three people are right next to each other at first, second and third, or third, fourth and fifth. Since it wouldn't be sensible to install sensors at each urinal for privacy reasons and because of the limited amount of hardware and time for this project, a simpler solution had to be developed. Luckily, men are quite deterministic when it comes to choosing a urinal, which enables us to estimate the current occupancy with only information about people entering and leaving. Although not perfectly accurate, this reduces the components required to one light barrier sensor and an algorithm based on probability. Together with the other components in the system, this idea was implemented and tested in the span of two months.

3 Related Work

We couldn't find any similar projects or publications in our research. We assume it's because of the unscientific nature of this project.

While men naturally know which toilet to take, it is a complicated social structure to explain to others. This guide showcases the general thought-process although we heavily disagree with the condescending tone.
Which Urinal should you really pee in

4 System Design and Implementation

4.1 System Overview

The Urinalysis system consists of three parts as shown in figure 1. When a person enters or exits the urinals section of the restroom, two movement sensors trigger. The captured sensor data on the edge is sent to the gateway. This data is then used to determine if a person entered or exited, and an appropriate event is raised by the gateway. The backend reacts to this event and estimates the current occupancy state of the urinals. If this new occupancy state effects the bro code for the next person (for example if they can't honour it anymore), another event is raised, which makes the website update as well.

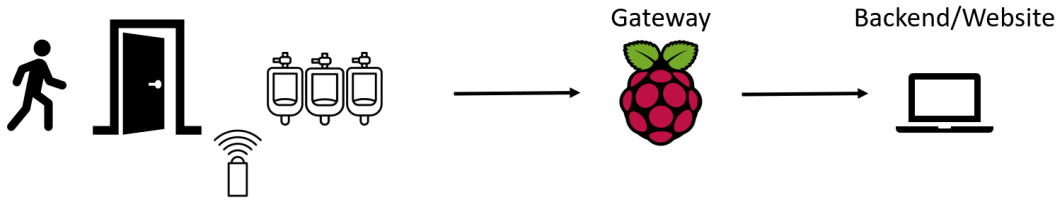

Figure 1: System Overview

4.2 System Architecture

When a person walks into the urinal area of a restroom, the ESP32 on the edge observes a sensor state change. Whenever this happens, the state of both sensors is sent to the gateway (Raspberry Pi Zero W) via Bluetooth Low Energy (BLE) as a notification.
After the gateway determined if a person entered or exited, it publishes an MQTT message on the appropriate topic through the MQTT.js library.
The MQTT broker relays the message to the backend, which is listening on those topics.
The backend keeps score of how many people are using the urinals at every given moment as well as the possible occupancy states and their probabilities. From these it can determine the most likely bro code state (available, bro code breaking or full). Whenever the bro code state changes, the backend publishes a message including the new state to the topic bro code changed topic. The website, which runs an express.js web server, receives this message via the MQTT broker because it subscribed to said topic. The website then emits a Socket.IO event, which is received on all the web-clients, that are connected to the website. The frontend code (JavaScript) updates the HTML on the website according to the new bro code state received via Socket.IO.

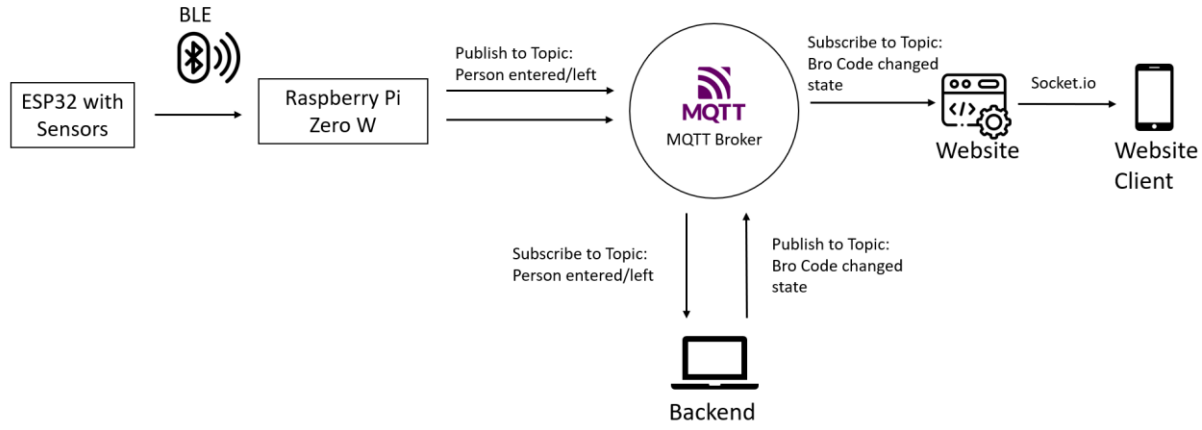

Figure 2: System Architecture

4.2.1 Noteworthy technologies used

4.2.1.1 ESP-IDF

The Espressif IoT Development Framework provides an SDK with libraries and tools for developing applications on ESP32. With a purpose-built VSCode extension the development experience for building, flashing and monitoring applications is exceptional. It also includes a mechanism to configure the project and set up things like the Bluetooth stack in an easy-to-use text-based menu.

4.2.1.2 Node.js

Node.js allows JavaScript code to run on a machine directly, outside of a web browser. It has an event-driven architecture capable of asynchronous I/O. Thanks to its many open-source libraries, there is a lot of support for other technologies like BLE and MQTT. Next to its simplicity, this is one of the main reasons we decided to build the logic for the gateway as well as the logic for the backend in Node.js.

4.2.1.3 TypeScript

TypeScript is a programming language built on top of JavaScript to add a type-system. Although still dynamically typed at runtime, this type-checking at compile time allows for a much better developer experience. In the end, the code transpiles directly to JavaScript.

4.2.1.4 Bluetooth Low Energy (BLE)

Bluetooth Low Energy is a wireless network technology aimed at novel applications for all technological fields. BLE consumes very little power compared to the original Bluetooth but still allows for similar range. That makes it a great fit for IoT where BLE is often used to send sensor data from one device to another. In this project a central and a peripheral were implemented. Other roles are broadcaster, which is like peripheral but does not allow connections or data transmissions, just advertisements, and observer, which is like central but doesn't initiate a connection and only listens for advertisements.

4.2.1.5 MQTT

MQTT is a lightweight message protocol designed for IoT. It's ideal for devices that have limited network bandwidth and other resource constraints. It's a publish/subscribe system in which clients can subscribe to topics and public messages on topics. For message transport a broker is used, which dispatches messages between publishers and subscribers.

4.2.1.6 Socket.IO

Socket.IO is a bi-directional low-latency communication protocol built for the web. As such it has amazing support for Node.js and JavaScript/TypeScript. It is built on top of WebSocket with additional features like automatic reconnect and namespaces.

4.3 System Implementation / Functional Software Architecture

4.3.1 Sensors

Since we don't care about general movement in the room, we removed the dome of the sensors and added cardboard tubes to only allow light from one point, like a beam.
Each one then acts like a light barrier that can detect a person walking through but not in which direction. However, if there are two of them, the order of activation can be used to calculate whether a person entered or exited resulting in a "directional" light barrier.
Only knowing when a person enters or exits, isn't enough to accurately deduce the exact occupancy of the urinals though. Luckily, we can get impressively accurate estimations with good algorithms on the backend, which allows us to use only two sensors, regardless of the numbers of urinals.
Each sensor features an open collector "alarm" pin. This pin is pulled low when the sensor detects motion. To read from this pin, it is connected to a GPIO pin on the ESP with an internal pull-up resistor. The power is connected to the 5V pin on the ESP, which is enough but there are claims that the sensor readings would be much more stable on higher voltages.

4.3.2 Edge

The application on the edge is built with ESP-IDF and running on an ESP32. It sets up a peripheral Bluetooth Low Energy (BLE) stack using NimBLE with a single vendor-specific service that features a single characteristic. The characteristic is notify-only, so you cannot read from or write to it on demand but must wait for a notification. After initialization, the ESP continuously reads the pins for both sensors and combines them into a single 8-bit value. In it, the left sensor is represented by the high nibble and the right one by the low nibble. If a sensor is sensing motion, the respective nibble is true (1), otherwise false (0). 00010000 would mean the left sensor encountered movement, the right didn't.
When one of the sensors' state changes, and there is an established BLE connection, the combined current state of both sensors is sent over BLE as a notification.

4.3.3 Gateway

On the gateway, a Node.js application first sets up a central BLE stack with Noble and tries to connect to the edge. When the peripheral is connected, the application searches for a specific service on the peripheral that includes the light barrier characteristic. When found, it subscribes to the characteristic's notifications. Because of inaccuracies and fluctuations in the sensor readings, only the first notification within a specified interval is considered. The first sensor reading received will normally be a change from no motion to motion on one sensor so depending on which sensor that is, it can be deduced whether a person entered or exited. Ignoring subsequent notifications for a while avoids double counting people.
When a person enters or exits, an empty MQTT message is published on the topic "person_entered" or "person_exited" respectively using MQTT.js.

4.3.4 MQTT Broker

The MQTT Broker used in this Project is Eclipse Mosquitto. It is running inside Docker and set up with a Docker Compose file for easy management and maintenance. It runs on the default port without any authentication or authorization.

4.3.5 Backend

The backend is a Node.js application, which first subscribes to the two enter/exit MQTT topics. Upon receiving a message, it calls the appropriate function of a Urinals object. The Urinals class implements a set of probability calculations and holds all the possible

occupancy states of the urinals and their probabilities as well as the estimated current bro code state. At the core is a function that returns a probability for each urinal in a specified urinal occupancy, which represents the chance that the next person would choose this urinal. This probability is calculated using the distance to the next person, the symmetry (odd is preferred) and the distance from the entrance as tiebreaker. Whenever a person enters, this function is used over all currently possible states to derive all the next possible states and their probabilities. Whenever a person exits, the person that first entered is removed across all states to derive the new ones.
After the new states are calculated, the bro code state is determined and if it changed, an update is sent over MQTT on the "bro_code_changed" topic. For archiving purposes, the update is also written to a CSV file with the current timestamp.

4.3.6 Website

The website has the purpose of displaying the current bro code state and update in real-time. It consists of a Node.js backend and an HTML frontend. The website-backend uses Express.js to serve the static frontend and sets up a Socket.IO server to communicate with the frontend. Additionally, it uses MQTT.js to setup a subscription for the "bro_code_changed" topic and emits an event over Socket.IO whenever the bro code state changes.
The frontend is a pure HTML, CSS and JavaScript site (single HTML file) that displays the current state and updates it whenever an update is received from the website-backend over Socket.IO.

5 Evaluation

The goal was to develop a system to observe the occupancy of the urinals and display the according bro code state outside of the restroom. It was planned to use an OLED display for this visual output but unfortunately, there was no Node.js library for programming the OLED display, and efforts to write our own on top of SPI were started only to realize there was not enough time. The display also supports I2C but that would've required soldering, so it wasn't an option. Another option was adding an Arduino to the system which communicated with the display using existing libraries and took commands from the Raspberry Pi, but this approach was also abandoned because of the little time we had left.
Instead, we decided to implement a website, where the current state of the urinal occupancy can be showed. This was a great compromise for the prototype because it didn't remove complexity but shifted it from hardware-interaction to real-time web communication.
The project fulfilled all the other goals as planned, and the system is working as intended. Implementing the edge-gateway communication was the most challenging part but it was interesting to learn about BLE and ESP-IDF. The use of the motion sensors to detect if a person entered or exited the urinals required some tinkering too. The sensors are very sensitive and produce unreliable signals at times, an issue that we learned may be due to the low voltage of 5V. We also realized late that the sensors need to calibrate for about ten seconds and can't handle any movement during that time whatsoever.
Learning about all the technologies used like BLE, MQTT, GPIO, ESP-IDF and Socket.IO was both interesting and fun, and together with a working prototype, we call this project a great success.

6 Application

Urinalysis can be used in any restroom with several urinals if they are separated from the stalls, for example at HSLU. We expect the system to bring the most value for three to seven urinals, but it should work regardless of count. Although it started as just a fun project, it could help greatly reduce anxiety of people suffering from pee shyness (Paruresis).

7 Conclusion

During this IoT course we managed to build and document an entire IoT system with edge, gateway and backend while learning about many interesting technologies on the way. Aside from meeting almost all the goals we had for the prototype, we also managed to use the technologies we were interested in, like MQTT and BLE, even though they weren't the easiest choices. A highlight of the project was the use of ESP-IDF, which was something new for all team members.
While we couldn't connect the OLED display to the Raspberry Pi Zero W, we quickly managed to find another solution without losing complexity or functionality. With more time or personnel, we wouldn't have been forced to abandon the small display and could have explored its capabilities, which would've been a great learning experience. If we continued this project, we might try to incorporate the estimated urinal occupancies more, for example as additional information on the website.

8 Contributions & Prerequisite

We are thankful for the help and advice from Ms. Nicoara on interfacing the sensors and the OLED display as well as the introduction to the ESP32 and IoT in general.

Joel already had some experience with Raspberry Pi and building real-time web applications. O had some programming experience beforehand as well as some knowledge of Node.js.

9 Major Milestones & Deliverables

At the beginning, we used the MoSCoW method to define goals for each component (edge, gateway, backend) with all the required features, a few optional ones, and some non-goals. Communication between the devices was the most important part for us and we started individually with BLE on the edge, BLE on the gateway and MQTT for gateway and backend. The sensor readings and the logic were built on top, and the website was done last.

9.1 Team and Roles

PROJECT WORK PACKAGES	OWNER
BLE peripheral on edge (ESP32)	Joel
Sensor reading on edge	Joel
BLE central on gateway (Raspberry Pi Zero W)	O. G.
Person enter/exit detection on gateway	O. G.
Algorithm for urinal occupancy and bro code state	Joel
MQTT broker setup with Docker	L. G.
MQTT communication between gateway and backend	L. G.
Website (backend + frontend + communication)	Joel

9.2 Project Planning / Timelines / Milestones & Deliverables

Figure 4 shows an overview of the milestones that were planned and reached.
On the 7th of December a connection between the edge and the gateway was established over BLE. On the same day, the gateway managed to exchange messages with the backend over MQTT. On the 15th of December, it was possible for the gateway to recognize if a person entered or exited the urinals using the sensor data from the edge. Thanks to the established connection from gateway to backend, these events could immediately be forwarded to the backend for processing. On the 22nd of December, the bro code state was first shown on the website in real-time, shortly before the submission of the report on 25th of December.

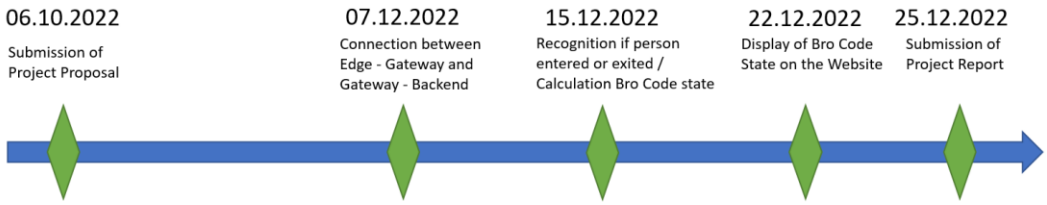

Figure 3: Milestones

10 Used references

- ESP-IDF
- Introduction to BLE (adafruit)
- Apache Mynewt NimBLE (BLE stack)
- NimBLE Peripheral Example
- Node.js
- Noble (BLE for Node.js)
- MQTT.js (MQTT client for Node.js)
- MQTT
- Eclipse Mosquitto (MQTT broker)
- Socket.IO
- Express.js (web framework for Node.js)