Smart Parking Monitoring System
Internet of Things Project Report

Bálint Bujtor
University of Bologna
Bologna, BO 40121
`balint.bujtor@studio.unibo.it`

Abstract

In this project I implemented a full IoT pipeline for a smart parking slot monitoring system. In the end, I was able to collect, process, transmit, store and visualize the data collected by the sensors at the very beginning of the pipeline. Seeing the results, I was able to implement a project that can solve a problem that is present in our everyday life. With slight modifications this project can be adjusted to solve totally different problems, as well.

1 Introduction

As the project topic I chose to implement a smart parking monitoring system which has already been implemented in modern garages, all around the world. This project was a perfect way to learn and practice these bleeding-edge IoT technologies and to try out the methodologies, algorithms which have been presented throughout the semester.

The IoT pipeline starts with processing the signal data from the IoT device placed in its native environment. I use a PIR sensor to detect movement in the parking slot and an ultrasonic distance sensor to measure the distance and thus the occupancy in the parking slot. After processing the data which is done by and ESP32 Wroom microcontroller the data is being sent through an MQTT broker to a database which acts as another client in the MQTT protocol. From this client which is implemented in Python, the data is sent to a database service, called InfluxDB. The final visualization is done by another software, called Grafana Dashboard. In the following paragraphs I am going to introduce briefly the aforementioned services and technologies.

2 Project Architecture

The project that I have chosen to implement consists of multiple elements as most of the IoT applications. The application can be distributed into 5 major parts that I present below from a bottom to top style, starting from the elements closer to the physical, hardware included parts. The whole structure of the application can be seen below (1).

2.1 Hardware Architecture

The first component is the physical, or hardware layer. As I mentioned before the hardware layer consists of 3 major components which are the following. I am using a PIR HC-SR501 (2) sensor to detect the motion in the parking slot. This sensor uses infrared rays to detect movement around its

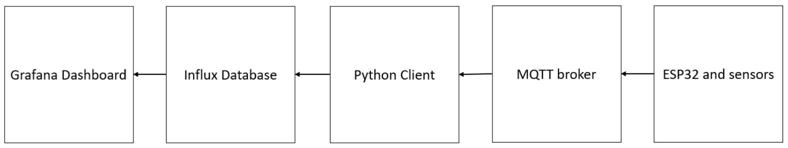


Figure 1: Application pipeline

Figure 2: HC-SR501 sensor

vicinity. The sensor that I am using is also equipped with a plastic half orb with hexagonal shaping to increase its field of view.

The other sensor that I am using is a HC-SR04 (3) ultrasonic sensor to measure the distance of the objects in front of it. It uses ultrasonic waves that reflect from the objects they hit. The distance measurement can be acquired indirectly, since the sensor itself measures the time elapsed between transmitting the outbound wave and receiving the reflected wave. This time is stored in microseconds.

Figure 3: HC-SR04 sensor

Both sensors are connected to an ESP32 WiFi (4) equipped microcontroller through jumpers. The sensors both need 5V power supply and since the microcontroller only has one, it was necessary to make this power output available for more than one device. This has been achieved with a breadboard and other jumpers.

2.2 Data processing

The ESP32 microcontroller is intended to handle reading the data from the sensors, processing and interpreting it locally and broadcasting it through a WiFi connection. I chose to transmit 3 data sources. First I transmit the measured distance by the ultrasonic sensor in cm-s. Based on a threshold number and the ultrasonic sensor's distance measurement I also transmit if the given slot is occupied by a vehicle. The third data stream is the motion sensor's output, which signals if there is motion in the slot.

Figure 4: ESP32 microcontroller

2.3 MQTT Protocol

For the communication part I chose to use the MQTT protocol [1]. This protocol is widely used in IoT applications since it does not require a high quality connection and is still able to transmit the data for longer distances, without significant data loss. The other benefit of this protocol is that it is lightweight which makes it suitable for devices with low computing capacity, like IoT applications. The protocol usually runs over TCP/IP protocol and works in a publish/subscribe methodology, meaning that the clients can subscribe to certain topics. The clients will get any incoming data from the topic they are subscribed to and they can also send messages and data with certain topic specification and different QoS levels.

2.4 Python Client

One of the requirements for the projects was to create another client that receives the incoming data streams and stores them in a database from which the data visualization can be done later. I have chosen to implement this client in Python language. The task of this client is to subscribe to all of the topics broadcast by the different microcontrollers in the IoT network and store the data until it is being sent to the database, by the client.

2.5 Influx Database

After the Python client receives the data it needs to be stored somewhere. In my application the time of the information is crucial, thus I needed a database in which the time information can be stored easily. I have chosen Influx Database for this problem, because it is a time series database, meaning that it is optimized for storing and serving time series through associated time and value(s) pairs. The InfluxDB [2] is not only perfect because of the prior point to this problem, but it is also optimized for fast (real time), high storage availability tasks where time information is important (like in most IoT applications).

As the points presented above show, InfluxDB is an appropriate choice for my task. In my database I store the broadcast sensor data with the location (garage and slot) information for each query.

2.6 Grafana Dashboard

Although, the InfluxDB has some built-in visualization tools, it does not offer an all-round and flexible solution for showcasing the received data. To be able to create a flexible, clean and simple dashboard I chose to use Grafana Dashboard [3] that is the go-to solution in visualizing real-time data in most of the cases in the field of IoT or any real-time applications.

3 Project Implementation

In this section I will discuss how I implemented my project. I started working on the project from a bottom to top style, meaning that I started with the problems closer to the data source.

3.1 Assembling the hardware

First of all, I assembled the the hardware with the two sensors using a breadboard and some jumpers (5) (6). Since the ESP32 only has one 5V output I needed to create the 5V and the GND voltage level that can be accessed from more than one place. After, I connected the data output of the PIR sensor to pinout number 13, and the power and ground pinouts accordingly. I connected the ultrasonic sensor's trigger pin to pinout number 25 and the echo pin to pinout number 26. The trigger pin is responsible for sending the sound waves from the sensor, meanwhile the echo pin returns the elapsed time in microseconds.

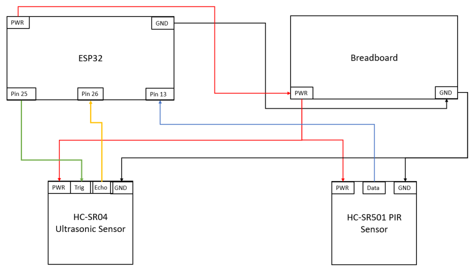


Figure 5: Hardware blueprint

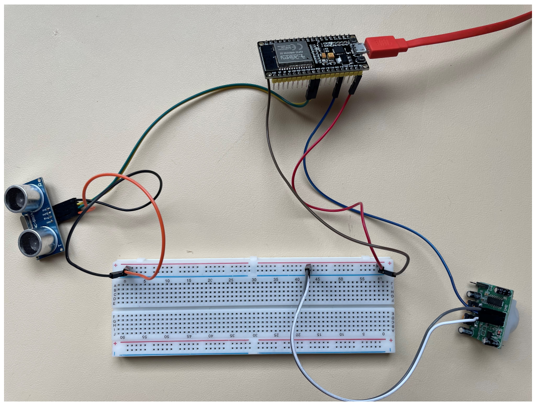


Figure 6: Picture of the assembled hw

3.2 Developing the software for the ESP32

After assembling the circuits for the project, I implemented the software that runs on the micro-controller. This sub-task consists of solving 4 problems. Connecting to the local WiFi channel, connecting and sending data through the MQTT broker and reading the sensor data of the two sensors with a given time interval. I implemented a function which connects to the specified WiFi network. It tries to connect every 5 seconds until it succeeds.

The other function in the source code implements publishing the sensor data to the MQTT broker. First, the function keeps trying to connect to the broker until it succeeds. Then, depending on the function parameters it chooses a a topic and publishes the sensor data with the chosen topic. In the source code I emulated two different parking slots by defining different topics for them and publishing a slightly modified sensor data from the same sensor. Finally the data publishing function returns with a bool value if the data was published successfully.

Usually, the algorithms running on embedded systems consist of two major parts. The first part is the setup where we initialize the instances and pins of the controller. The second part is a loop function which runs continuously after the setup has been executed. I followed the same methodology in my project.

First, I initialised the used input and output ports and created the instances for the WiFi connection and the MQTT communication.

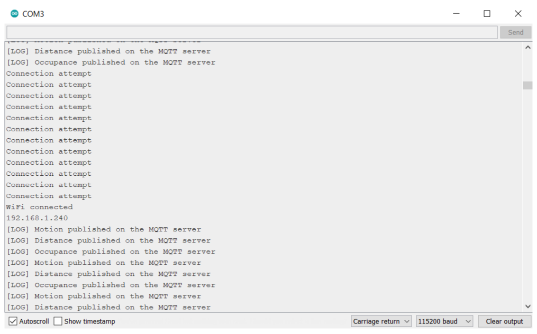
Then, in the loop part first I read the sensor value of PIR sensor. I use a state machine to decide if there has been a motion. If the current sensor value differs from the previous one, that means there has been a change in motion. (either the motion ended or started). If they are the same, there was no motion. Then I publish the sensor values for both topics to the MQTT broker. For the second topic I simply invert the value to be published for the first topic.

Next, I collect the data from the ultrasonic sensor. First I set the trigger pin output to Low and after, I set it high for 10 microseconds so that the sensor transmits a sound wave. Afterwards I read the received duration from the echo pin input. Finally, I calculate the distance based on the elapsed duration and the speed of sound. The distance is stored in cm-s. Then, I publish the data for both distance topics in the different parking slots. For the second topic I simply divide the read distance by two.

The third data stream that I publish to the broker is the occupancy of the given slot which is calculated by a comparison with a predefined constant variable, that can be changed based on the height of the garage.

The loop ends with a 30 seconds wait, then the algorithm starts to read the sensor values again.

Before continuing to the next step of the project I tested if processing the sensor values is working. I simply printed the sensor values to the Serial Monitor. After controlling that they work properly, I also checked if the connecting to the WiFi is working. After these test steps I proceeded to the next step (7).



```
[LOG] Distance published on the MQTT server
[LOG] Occupance published on the MQTT server
Connection attempt
Connection attempt
Connection attempt
Connection attempt
Connection attempt
Connection attempt
Connection attempt
Connection attempt
Connection attempt
Connection attempt
Connection attempt
Connection attempt
WiFi connected
192.168.1.240
[LOG] Motion published on the MQTT server
[LOG] Distance published on the MQTT server
[LOG] Occupance published on the MQTT server
[LOG] Motion published on the MQTT server
[LOG] Distance published on the MQTT server
[LOG] Occupance published on the MQTT server
[LOG] Motion published on the MQTT server
[LOG] Distance published on the MQTT server
```

Figure 7: Testing the embedded algorithm

3.3 MQTT broker

In order to access the published data from another entity, an MQTT broker is needed that will receive and forward the incoming data. For this reason, I installed an MQTT broker on my computer from the official MQTT website. [1] After the installation, I tested if publishing the messages from the microcontroller works properly.

3.4 Python client

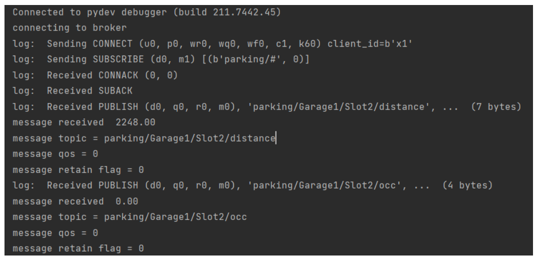
The next step in the IoT pipeline was to receive the published sensor data from the MQTT broker and forward it to the Influx Database. I created a Python algorithm that implements this functionality.

This algorithm consists of two main parts. The first one is an MQTT client that is able to subscribe to topics on the MQTT broker and receive the data. I used the paho-mqtt library to implement this [4]. The library has built in function that can be overwritten for different scenarios (on connection, receiving new messages etc.). For easier debugging I overwrote these functions. By default, the MQTT broker is listening to messages on port 1883, I also use this port for communication. The client is subscribed to every message with the topic *parking*. In this client algorithm I needed to assure that all the incoming data is going to be forwarded to the database. For this I created a Queue object. When a new payload is received the program places the payload to this Queue. Then, it picks and pops the oldest element from the Queue, creates a measurement point object with the data and the corresponding tags and forwards it to the database. It forwards the payloads until the Queue object is not empty. With this approach it can be assured that every message is being forwarded to the database. When the user stops the algorithm it disconnects from the broker before exiting.

The other part is an influx database client that is able to write data to the database. When the MQTT broker receives a new payload and creates a Point object from it, it is the influx database client's task to forward the new measurement to the server. InfluxData offers a library to do this, we only need to specify the folder and the database where the data point should be forwarded, as well as a security token. I am forwarding the incoming results as float numbers (8).

3.5 Influx DB

Without having an Influx Database server on the computer, forwarding the incoming data would not make sense. For that reason, I needed to install an InfluxDB server on my computer what I

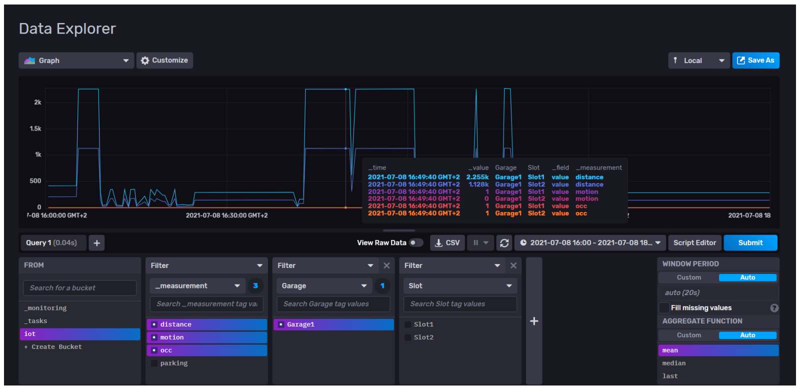


```
Connected to pydev debugger (build 211.7442.45)
connecting to broker
log:  Sending CONNECT (u0, p0, wr0, wq0, wf0, c1, k60) client_id=b'x1'
log:  Sending SUBSCRIBE (d0, m1) [(b'parking/#', 0)]
log:  Received CONNACK (0, 0)
log:  Received SUBACK
log:  Received PUBLISH (d0, q0, r0, m0), 'parking/Garage1/Slot2/distance', ...  (7 bytes)
message received  2248.00
message topic = parking/Garage1/Slot2/distance
message qos = 0
message retain flag = 0
log:  Received PUBLISH (d0, q0, r0, m0), 'parking/Garage1/Slot2/occ', ...  (4 bytes)
message received  0.00
message topic = parking/Garage1/Slot2/occ
message qos = 0
message retain flag = 0
```

Figure 8: Testing the python client

downloaded from the official site. The server receives data through port 8086, thus the Python client needs to send the data to this port on either local host or on the computer's IP address.

After installing the server, I tested if the server really receives the data from the Python client. The user is able to make some queries from the server and to visualize them with some basic graphs. I made these queries and showcased the incoming data, which can be seen below (9) (10).



_time	_value	Garage	Slot	_field	_measurement
2021-07-08 16:49:40 GMT+2	2.255k	Garage1	Slot1	value	distance
2021-07-08 16:49:40 GMT+2	1.128k	Garage1	Slot2	value	distance
2021-07-08 16:49:40 GMT+2	1	Garage1	Slot1	value	motion
2021-07-08 16:49:40 GMT+2	0	Garage1	Slot2	value	motion
2021-07-08 16:49:40 GMT+2	1	Garage1	Slot1	value	occ
2021-07-08 16:49:40 GMT+2	1	Garage1	Slot2	value	occ

Figure 9: Influx Database Query

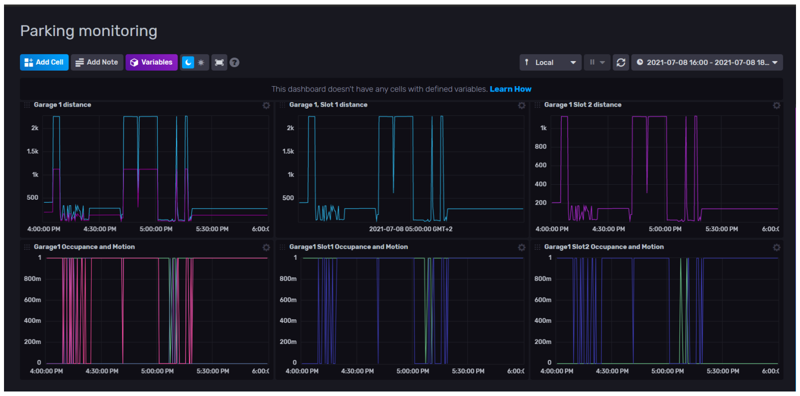

Parking monitoring
Add Cell
Add Note
Variables
Local
2021-07-08 16:00 - 2021-07-08 18...
This dashboard doesn't have any cells with defined variables. Learn How
Garage 1 distance
Garage 1. Slot 1 distance
Garage 1 Slot 2 distance
Garage1 Occupance and Motion
Garage1 Slot1 Occupance and Motion
Garage1 Slot2 Occupance and Motion

Figure 10: Influx Database visualization

3.6 Grafana Dashboard

However, it is possible to do the visualization with InfluxDB's built in tools, it does not offer so diverse options. To have bigger flexibility, I chose Grafana dashboard to visualize the data. First of all, I needed to connect the Dashboard to the my Influx Database. The Grafana Dashboard offers an easy to use configuration page in which the developer can choose the data source and specify every parameter in it. Since my Influx DB listens on port 8086 I had to give this as the source. The dashboard itself can be accessed through port 3000. I made the following graphs and panels in the Dashboard. On the dashboard, I show the incoming distances, motions and occupances in all of the slots in Garage1, and separately in each slots in Garage1. Also I am showing the last received distance, occupancy and motion values in each slot of Garage 1.

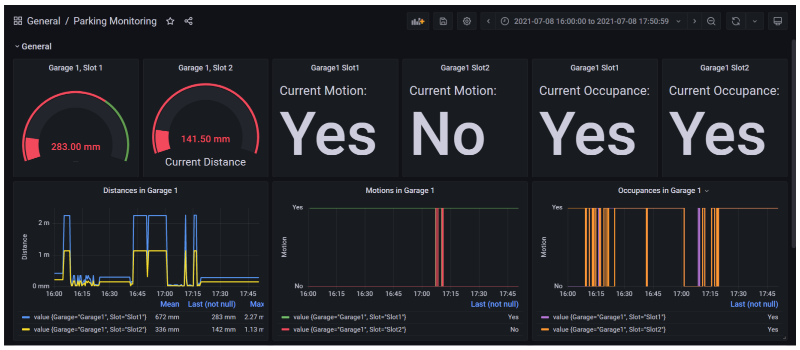


Figure 11: Grafana Dashboard

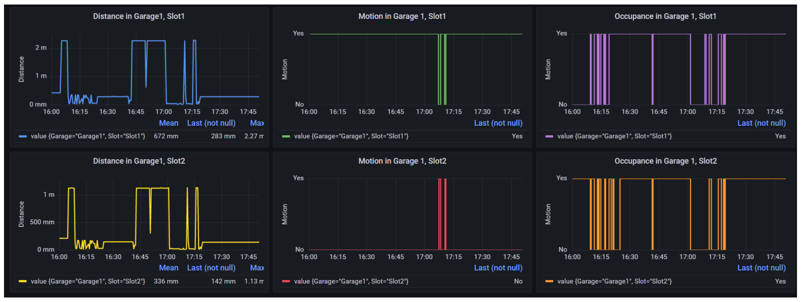

Distance in Garage1, Slot1
Motion in Garage 1, Slot1
Occupance in Garage 1, Slot1
Distance in Garage1, Slot2
Motion in Garage 1, Slot2
Occupance in Garage 1, Slot2

Figure 12: Grafana Dashboard

4 Results and Further Improvements

As the previously attached pictures show I was able to implement the full IoT pipeline and visualize the received data real-time on the Grafana Dashboards. The pipeline can be deployed in real-life environments and work after the sensor's have been calibrated.

However, there is always room for improvement and there can always be some bugs that have not been tested before. For example, the project has not been tested how it can handle the incoming data amount if we have a large amount of sensors (in a garage there is usually at least 500 slots). The efficiency of the algorithms can also be improved. The transmitted payload size can be further reduced by choosing appropriate data type for the sensor data (using integers or binary for the occupancy and movement data streams) or decreasing the topic string lengths (instead of using Garage1/Slot1/dist, 1/2/d can be used for example). I did not implement these changes because with one sensor it is not a bottleneck it helps debugging more and in a real-life scenario these can be changed easily in the source code.

5 References

[1] MQTT Protocol - `https://mqtt.org/`

[2] Influx Database - `https://www.influxdata.com/`

[3] Grafana Dashboard - `https://grafana.com/`

[4] Paho MQTT - `https://pypi.org/project/paho-mqtt/`

[5] ESP32 Arduino Extension installation troubleshooting - `https://randomnerdtutorials.com/esp32-troubleshooting-guide/`

[6] ESP32 Pin Layout - `https://randomnerdtutorials.com/esp32-pinout-reference-gpios/`

[7] Ultrasonic sensor usage - `https://create.arduino.cc/projecthub/abdularbi17/ultrasonic-sensor-hc-sr04-with-arduino-tutorial-327ff6`

[8] PIR sensor usage - `https://create.arduino.cc/projecthub/electropeak/pir-motion-sensor-how-to-use-pirs-w-arduino-raspberry-pi-18d7fa`

[9] Pubsubclient for Arduino - `https://pubsubclient.knolleary.net/api#subscribe`

[10] MQTT Tutorial - `http://www.steves-internet-guide.com/`

[11] Python InfluxDB Client - `https://docs.influxdata.com/influxdb/cloud/tools/client-libraries/python/`