Networks and Autonomous Systems Project

Universidade de Aveiro

Sebastian D. González

Networks and Autonomous Systems Project

Dept. de Eletrónica, Telecomunicações e Informática

Universidade de Aveiro

sebastian.duque@ua.pt(103690)

July 23, 2024

Contents

List of Figures

Glossário

MQTT	Message Queuing Telemetry Transport.
OBU	On-Board Units.
RSU	Road-Side Units.

ParkMyTroti

1.1 Project Overview

ParkMyTroti is a system addresses the specific needs of scooter users, providing real-time parking availability and scooter control geographic tracking. Users receive up-to-date information on parking availability at their destination, with plans to include features for covered parking options. Currently, the system operates using Vanetza and MQTT messaging for communication and data exchange, ensuring accurate and timely information on parking availability. This allows vehicles to exchange data about scooters location and report it to a station where the parking decisions will be handled.

Unlike existing systems that cater primarily to cars, this project focuses on scooter users, filling a gap in the market by offering tailored solutions for micro-mobility parking challenges. Real-time data and in-app guidance will help users locate available parking zones, optimizing space usage and convenience [1].

The project's Repository can be found in GitHub.

Figure 1.1: ParkMyTroti logo

1.2 Objectives

1. **Parking Map:** A key feature of my scooter parking system is its ability to provide users with precise location details, including specific addresses and exact parking availability. This level of detail is crucial for scooter riders, who often need to navigate in new and unknown spaces and plan their routes carefully.

2. **RSU Tracking** A critical aspect of the scooter parking system is its ability to track the real-time location of all On-Board Units (OBU) using Road-Side Units (RSU). This feature ensures that the system can accurately monitor the movement and positioning of scooters, enabling it to provide users with up-to-date information on parking availability and occupancy.

3. **OBUs and RSU communication:** To enable effective communication between OBU and RSU, a standardized and efficient approach is crucial. The system should adhere to established protocols and best practices to ensure interoperability, reliability, and performance.

1.3 System Architecture

The scooter parking system architecture is designed to facilitate efficient communication between scooters OBU and RSU using Message Queuing Telemetry Transport (MQTT) and **Vanetza** protocol. In the simulated environment, a simulation coordinator is crucial for managing and synchronizing the various components of the scooter parking system, ensuring a smooth and accurate simulation that realistically replicates real-world scenarios. The key components of the architecture are:

1. **OBU:** Represents the scooters of the simulation environment. They send its information is broadcast using MQTT to ensure other components of the system, like the server and RSU, to be aware of their current state.

2. **RSU:** Works has a manager and tracker of all the OBU. It communicates directly with the OBU and it works with the server to enable the real-time monitoring and control of the OBU and the parking lot statuses via a web interface. It hosts the MQTT broker that functions as the central communication hub for the OBU and the `server.py`. The MQTT broker:

   ⇒ Facilitates the exchange of messages between different parts of the system.

   ⇒ Allows OBU to publish their status and location updates.

   ⇒ Allows `server.py` to subscribe to these updates and also publish relevant commands or status messages back to the OBU.

3. **server.py:** Acts as the backend server that handles the MQTT communication with OBUs and RSUs. It processes parking requests and responses, and pushes updates to a web client (the index.html) ensuring seamless real-time communication and data display capabilities.

   ⇒ Facilitates bidirectional data flow between RSUs, OBUs, and the frontend.

   ⇒ Provides dynamic updates on parking availability to optimize resource utilization.

4. **index.html:** Is the front-end interface or dashboard for visualizing the status and movements of OBUs and displaying information related to parking availability and system status.

   ⇒ It has a map component using Leaflet.js that shows the real-time positions of OBUs as they move along their designated paths and parkings.

   ⇒ Shows the current availability of parking spots. This information is updated based on messages received from the MQTT broker that reflect the status of each parking lot.

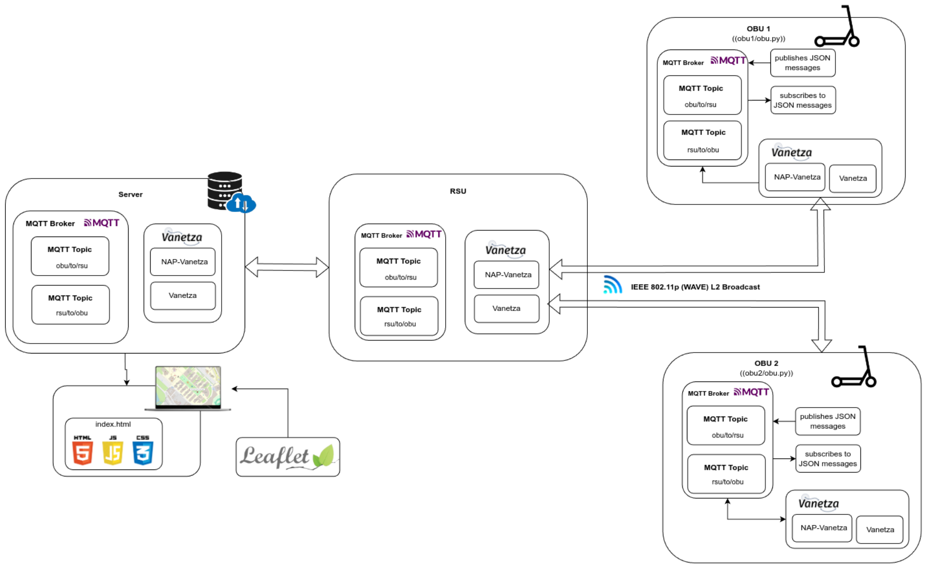

Figure 1.2: System Architecture

1.4 RSU

As previously mentioned, the RSU is the intermediary that facilitates communication and coordination between OBU and the overall system. It has the knowledge of the parking lots available:

```
AVAILABLE_PARKING = [
    {"name": "ParkingLot1", "latitude": 40.630321, "longitude":
    -8.657457},
    {"name": "ParkingLot2", "latitude": 40.629555, "longitude":
    -8.656427}
]
```

Listing 1.1: AVAILABLE_PARKING dictionary

It also tracks in real time the OBU location in real time saving it in `obu_locations`.

```
obu_locations = {
    1: {"lat": 40.6312, "lon": -8.6564},
    2: {"lat": 40.6296, "lon": -8.6565},
}
```

Listing 1.2: Example of obu/to/rsu message

In the code example 1.2 1, 2 are the IDs of the OBUs and each ID maps to a dictionary containing `lat` (latitude) and `lon` (longitude) keys, which represent the coordinates of the corresponding OBU.

1.4.1 Types of RSU Messages

1.4.1.1 RSU Acknowledge

Is a message created by the RSU to acknowledge the receipt of a message from an OBU and is published into the topic `"rsu/to/obu"`

```
response = {
    "response": "RSU Acknowledges the message from OBU 2",
    "timestamp": "2024-07-07T12:34:56.789Z"
}
```

Listing 1.3: Example of rsu/to/obu message

1.4.1.2 parking_status

Is a message that provides OBUs with real-time updates on `available parking locations` and their `ccupancy status`[1], along with a `timestamp` to ensure the information is current.

```
message = {
    "available_parking": [
        {"name": "ParkingLot1", "latitude": 40.630321, "longitude":
    -8.657457},
```

[1]If True ParkingLot is available if False Indicates ParkingLot is occupied

```
        {"name": "ParkingLot2", "latitude": 40.629555, "longitude":
     -8.656427}
    ],
    "parking_status": {
        "ParkingLot1": True,
        "ParkingLot2": False
    },
    "timestamp": "2024-07-07T12:34:56.789Z"
}
```

Listing 1.4: Example of parking_status message

1.5 OBU

Has said before they represent the scooters in the simulation environment. They are in constant communication with RSU and have a default route that goes around the campus and other one to the parking lots.

```
[40.6312, -8.6564], [40.6312, -8.6563],
[40.6305, -8.6554], [40.6292, -8.6550],
[40.6292, -8.6550], [40.6285, -8.6561],
[40.6285, -8.6561], [40.6289, -8.6576],
[40.6289, -8.6576], [40.6294, -8.6582],
[40.6294, -8.6582], [40.6298, -8.6588],
[40.6298, -8.6588], [40.6301, -8.6592],
[40.6301, -8.6592], [40.6305, -8.6587],
[40.6305, -8.6587], [40.6308, -8.6585],
[40.6308, -8.6585], [40.6312, -8.6581],
[40.6312, -8.6581], [40.6314, -8.6578],
[40.6314, -8.6578], [40.6312, -8.6564]
```

Listing 1.5: Defaul OBU route arround the campus

OBUs send periodic event notifications to the server regarding their status, such as whether they are currently **moving** or **parked**.

1.5.1 Types of OBU Messages

1.5.1.1 Location Update Messages:

These messages are GPS updates to the server, which consist of a structured message containing the following fields: `station_id` (a unique identifier for the OBU), `latitude`, `longitude`, `status` (indicating whether the OBU is currently in status **moving** or **parked**), and timestamp (the time at which the update was sent)."

```
{
    "station_id": 1,
    "status": "moving",
    "latitude": 40.6314,
    "longitude": -8.6578,
    "timestamp": "2024-07-06T14:30:00Z"
}
```

Listing 1.6: Example of Location Update Messages in status "moving"

```
{
    "station_id": 2,
    "status": "parked",
    "latitude": 40.629555,
    "longitude": -8.656427,
    "timestamp": "2024-07-06T14:35:00Z"
}
```

Listing 1.7: Example of Location Update Messages in status "parked"

1.5.1.2 obu/to/rsu Messages

Represents a message that an OBU sends to a RSU through the "`obu/to/rsu`" topic. This message contains information about the OBU's current `location`, `status`, and a `timestamp`.

```
{
    "obu_id": 1,
    "message": "OBU 1 at new location with status moving",
    "location": {"lat": 40.6304, "lon": -8.6573},
    "timestamp": "2024-07-06T14:30:00Z"
}
```

Listing 1.8: Example of obu/to/rsu message

1.5.2 OBU movement

After receiving an `acknowledgment` from the RSU, the OBU will confirm its reception and see, in the packets received, the e `parking_status` (current occupancy status of parking lots) and `available_parking` (list of available parking locations) .

```python
def handle_acknowledgment(message):
    print(f"OBU received acknowledgment from RSU: {message}")
    parking_status = message.get("parking_status", {})
    available_parking = message.get("available_parking", [])
```

Listing 1.9: OBU received messages handling

Every time the OBU finish its default route it checks the `available_parking` and the `parking_status`. If a parking lot is found to be free, it sets `chosen_parking` to that location and then updates the route of the OBU to direct it to the selected free parking lot and change its `status`. If no parking spot is available, the OBU will default to its original route.

```python
# Select an available parking lot if any
chosen_parking = None
for parking in available_parking:
    if parking_status.get(parking["name"], True):  # Look for the first not occupied parking
        chosen_parking = parking
        break

# Update OBU routes based on parking availability
for obu in OBUs:
    idx = obu.get('current_idx', 0)
    if idx == 25:  # Update index if necessary
        if chosen_parking:
            # Extend the route with the parking lot coordinates
            extended_parking_route = PARKING_ROUTE + [[chosen_parking["latitude"], chosen_parking["longitude"]]]
            obu['animation_path'] = extended_parking_route
            print(f"OBU {obu['id']} is directing to {chosen_parking['name']} at location ({chosen_parking['latitude']}, {chosen_parking['longitude']})")
        else:
            print(f"OBU {obu['id']} continues on its default route.")
```

Listing 1.10: OBU movement handling

1.6 Front-end interface

The dashboard, implemented using HTML and JavaScript, utilizes the Leaflet library to visualize the real-time positions of OBUs and parking locations on an interactive map. When the application starts the user is redirected to this interface:

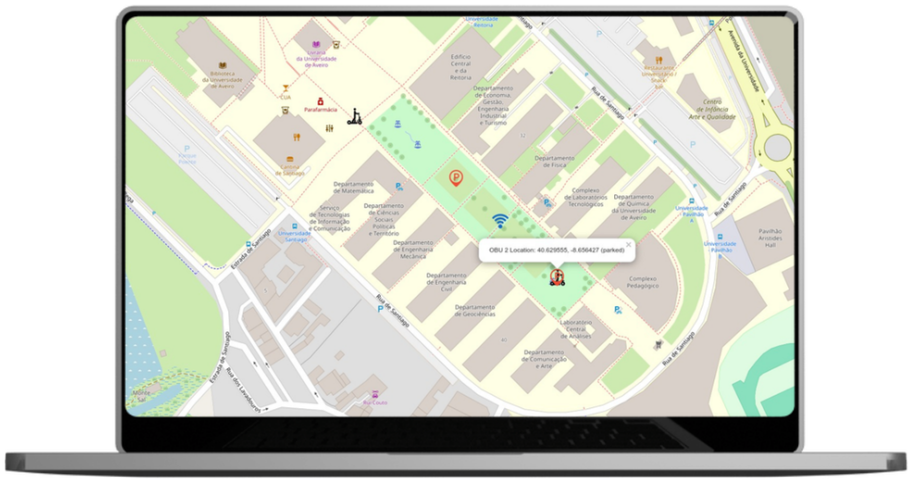

Figure 1.3: Dashboard interface

1.7 Demonstration

The simulation timeline is shown in Figure 1.4 and a video demonstration of my system can be found on YouTube.

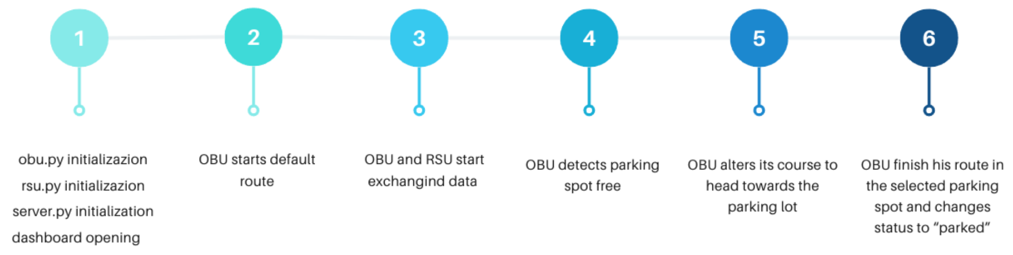
1
2
3
4
5
6
obu.py initializazion
rsu.py initializazion
server.py initialization
dashboard opening
OBU starts default route
OBU and RSU start exchangind data
OBU detects parking spot free
OBU alters its course to head towards the parking lot
OBU finish his route in the selected parking spot and changes status to "parked"

Figure 1.4: Demonstration Timeline

1.8 Conclusion and results

In summary, the project achieved the main objective to create an appealable interface where is possible to see all the OBUs in real-time movement and all the parking lots available. The scooters can efficiently find the free parking lots thanks to the messages exchanged between RSU and OBU. By leveraging MQTT for communication and Leaflet for visualization, the system provides a comprehensive solution for optimizing urban scooter parking management.

1.9 Future Work

Currently, my system is stuck in a inflexible routine, relying on pre-programmed paths and configurations that make it inflexible and slow to adapt to changing circumstances. To take it to the next level, we need to focus on giving it more autonomy and flexibility. One key way to do this is by allowing OBUs to plan their own routes in real-time, taking into account real-time traffic conditions and parking availability. This would allow them to adjust their routes dynamically, rather than following set paths, making them more efficient and better equipped to handle unexpected situations.

For future work, we plan to scale up the system by adding more OBUS dynamically to test the performance of the system under increased scooter traffic. This will allow us to evaluate the system's ability to handle a larger volume of scooters and identify potential bottlenecks.

Finally, we will integrate real-time weather information into the simulation to influence parking decisions. For example, in case that is raining, the scooters will be forwarded to the nearest parking spot under a roof. This will enable us to simulate the impact of different weather conditions on parking demand

and allocation, allowing for more accurate predictions and optimized parking strategies. To improve the negotiation process among OBUS, we will implement priority-based negotiations for parking allocation. This will ensure that each OBUS is prioritized based on its needs and requirements, reducing congestion and improving overall efficiency.

Bibliography

[1] D. Foley, *Micromobility for europe advocates for smarter parking solutions for e-scooters and e-bikes*, https://unagiscooters.com/news/2024/01/11/micromobility-for-europe-smarter-parking-solutions-for-escooters-and-bikes/, 2024.