CEN3907C

Parking Availability System

Pre-Alpha Build Report

Erik Meurrens, Benjamin Simonson, Ryan Jalloul, Evan Tobon, Samer Khatib

Repository Link: https://github.com/emeurrens/parking-availability-system

Video Explanation Link: https://youtu.be/zqVpl0zXWvA

Architectural Elements

External Interface

The external interface relies on a sensor module and camera for detecting and recording vehicles entering and exiting the parking facility. When a vehicle disrupts the sensor beam, the sensor triggers the camera module to capture a photograph of the entity. This image is then immediately transferred by a Python script to a PostgreSQL database for preprocessing, decoding, and interpretation. If the object detection model confirms that the entity is a motorized vehicle, a global counter in the database is adjusted to reflect the updated occupancy count.

To communicate parking availability to users, a mobile application interface has been developed using the Flutter and Dart framework. The app reads directly from the PostgreSQL database, providing a real-time display of parking space availability for users on their devices.

Network connectivity is established via the Raspberry Pi's onboard wireless module (model 3 B+). In cases where signal strength or reliability may be limited, additional options, such as soldering an external antenna or adding a secondary Wi-Fi module, can be implemented. This enhanced connectivity supports data transfer to AWS for image processing and the reception of image interpretation results, extending the device's "edge" capabilities.

Persistent State (e.g. databases)

The AWS EC2 module that hosts the PostgreSQL database serves as the persistent state, reliably storing essential data related to vehicle activity within the parking facility. It records vehicle entry and exit events, maintaining an up-to-date count of vehicles present in real-time. This database also logs other relevant details, such as timestamps for each entry and exit, which may support historical analysis and usage trends over time.

Additionally, the database is designed to interface with a mobile application, which allows users to view live parking facility availability. As vehicles enter or leave, the database updates the occupancy status and pushes this information to the application, providing users with accurate, current information on parking space availability. This seamless communication between the database and the mobile app ensures a dependable system for managing and displaying parking data for users. It will also serve as a permanently available resource for users all over the world assuming they choose to access the parking facility's availability .

Internal Systems

The Raspberry Pi (RPi) manages a seamless internal communication loop with both a camera module and a supersonic sensor, enabling automated image capture and data processing.

Through the I2C communication protocol, the RPi listens for a trigger signal from the supersonic sensor, which activates when a vehicle approaches or departs from the parking facility. Upon receiving this signal, the RPi initiates the camera module to capture an image of the vehicle. For testing purposes, the RPi manually triggered to ensure the camera module functioned appropriately (see Figure 1 for an example).

Once captured, the image undergoes an initial processing stage where the RPi converts it into binary data. This binary data is then prepared for transfer to a remotely hosted PostgreSQL database that is hosted remotely on an AWS EC2 instance. The RPi securely transmits the image data over an encrypted connection, enabling storage and subsequent analysis within the database. Based on whether a vehicle is detected entering or exiting the parking facility, a designated counter in the database is either incremented or decremented accordingly, effectively keeping a real-time record of vehicle occupancy within the facility.

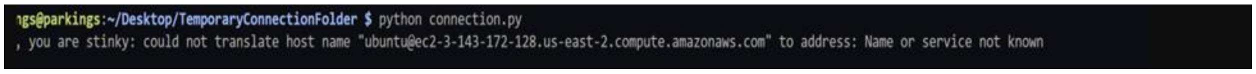
This internal communication and data-handling system ensures that the RPi not only captures critical information through its connected modules but also processes and transmits this data in a secure, efficient manner, forming the foundation for subsequent analytical steps within the broader system.

Figure 1. Mini Group selfie.

Information Handling

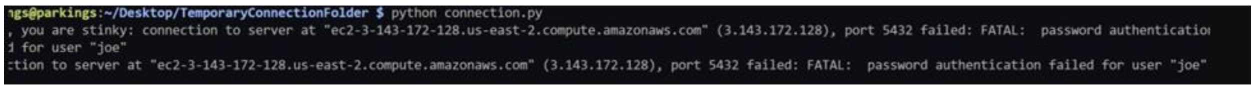
Communication

The Raspberry Pi (RPi) has been successfully configured to connect with a PostgreSQL database hosted on an AWS EC2 instance, a major milestone in establishing seamless communication with external systems via a Python script. In the future, the RPi will transmit data as image bitmaps of vehicles, which will undergo preprocessing, decoding, and analysis to detect vehicle entry into the parking facility. Setting up this connection involved several key challenges. Initially, the lack of a traffic rule to accept requests from the RPi prevented communication. Later, issues arose when configuring a PostgreSQL cursor via the psycopg2 library, as it was initially assumed that the RPi needed to connect directly to the EC2 host before accessing the database (see Figure 1). The final challenge involved newline characters inadvertently added to the secrets file when copying credentials from Amazon, which caused authorization errors when the RPi attempted to connect. Using try-catch blocks, the newline issue was swiftly identified and resolved, enabling a successful and stable connection (see Figure 2).



```
ıgs@parkings:~/Desktop/TemporaryConnectionFolder $ python connection.py
, you are stinky: could not translate host name "ubuntu@ec2-3-143-172-128.us-east-2.compute.amazonaws.com" to address: Name or service not known
```

Figure 2. EC2 Connection failure due to psycopg2 misunderstanding.



```
ıgs@parkings:~/Desktop/TemporaryConnectionFolder $ python connection.py
, you are stinky: connection to server at "ec2-3-143-172-128.us-east-2.compute.amazonaws.com" (3.143.172.128), port 5432 failed: FATAL:  password authenticatio
d for user "joe"
ction to server at "ec2-3-143-172-128.us-east-2.compute.amazonaws.com" (3.143.172.128), port 5432 failed: FATAL:  password authentication failed for user "joe"
```

Figure 3. Connection authorization failure due to incorrect interpretation of credentials.

Integrity and Resilience

Some measures to ensure integrity and resilience have been taken to secure the connection between the RPi and the PostgreSQL database. Through a dedicated secrets directory, the RPi securely connects to the database which is only viewable to the RPi. Additionally, the database contains a security rule in which it is set to accept traffic specifically from the RPi's IP address, enabling secure read and write operations. However, considering the RPi will be set up in a public parking facility, further ensuring the security of the values in the secret's directory will add an additional layer of security.