Software and Hardware Systems for Autonomous Smart Parking Accommodating both Traditional and Autonomous Vehicles

April 2021

A Research Report from the Pacific Southwest Region University Transportation Center

Mohammad Abdullah Al Faruque, UC Irvine

Mohanad Odema, UC Irvine

Luke Chen, UC Irvine

Pacific Southwest Region UTC

University Transportation Center

University of California Irvine

TECHNICAL REPORT DOCUMENTATION PAGE

1. Report No. PSR-19-30	**2. Government Accession No.** N/A	**3. Recipient's Catalog No.** N/A
4. Title and Subtitle Software and Hardware Systems for Autonomous Smart Parking Accommodating both Traditional and Autonomous Vehicles		**5. Report Date** April 2021 **6. Performing Organization Code** N/A
7. Author(s) Mohammad Abdullah Al Faruque, https://orcid.org/0000-0002-5390-0497 Mohanad Odema Luke Chen		**8. Performing Organization Report No.** PSR-19-30 TO-027
9. Performing Organization Name and Address METRANS Transportation Center University of Southern California University Park Campus, RGL 216 Los Angeles, CA 90089-0626		**10. Work Unit No.** N/A **11. Contract or Grant No.** USDOT Grant 69A3551747109 Caltrans 65A0674 TO-027
12. Sponsoring Agency Name and Address U.S. Department of Transportation Office of the Assistant Secretary for Research and Technology 1200 New Jersey Avenue, SE, Washington, DC 20590		**13. Type of Report and Period Covered** Final report 1/1/20 to 4/1/21 **14. Sponsoring Agency Code** USDOT OST-R

15. Supplementary Notes
Project webpage: https://www.metrans.org/research/software-and-hardware-systems-for-autonomous-smart-parking-accommodating-both-traditional-and-autonomous-vehicles

16. Abstract
Parking infrastructure is suffering from congestion as the number of vehicles circulating in urban areas is growing and expansion is not a cost-effective solution. In parallel, developments in autonomous vehicle technology mean that driverless vehicles are predicted to be in circulation by the 2020s and makeup 40% of vehicle travel by the 2040s. Expected benefits of autonomous vehicle travel include reduced congestion through vehicle sharing and reduced walking distance for passengers who can be dropped off chauffeur-style by autonomous vehicles. However, empty vehicle cruising, or the case in which autonomous vehicles cannot efficiently locate parking and circle instead, can potentially increase congestion. Given that this new technology has the potential to exacerbate existing congestion issues, it is necessary to develop a solution for parking congestion integrated with autonomous vehicles. Our project addresses this issue by providing a full-stack solution including sensors to monitor occupancy, Fog systems to perform local data pre- processing, and SDR radios to communicate with autonomous vehicles.

17. Key Words Smart Parking; Parking Congestion; Autonomous Vehicles	**18. Distribution Statement** No restrictions.

19. Security Classif. (of this report)	**20. Security Classif. (of this page)**	**21. No. of Pages**	**22. Price**
Unclassified	Unclassified	24	N/A

Form DOT F 1700.7 (8-72) Reproduction of completed page authorized

Contents

About the Pacific Southwest Region University Transportation Center

The Pacific Southwest Region University Transportation Center (UTC) is the Region 9 University Transportation Center funded under the US Department of Transportation's University Transportation Centers Program. Established in 2016, the Pacific Southwest Region UTC (PSR) is led by the University of Southern California and includes seven partners: Long Beach State University; University of California, Davis; University of California, Irvine; University of California, Los Angeles; University of Hawaii; Northern Arizona University; Pima Community College.

The Pacific Southwest Region UTC conducts an integrated, multidisciplinary program of research, education and technology transfer aimed at *improving the mobility of people and goods throughout the region.* Our program is organized around four themes: 1) technology to address transportation problems and improve mobility; 2) improving mobility for vulnerable populations; 3) Improving resilience and protecting the environment; and 4) managing mobility in high growth areas.

U.S. Department of Transportation (USDOT) Disclaimer

The contents of this report reflect the views of the authors, who are responsible for the facts and the accuracy of the information presented herein. This document is disseminated in the interest of information exchange. The report is funded, partially or entirely, by a grant from the U.S. Department of Transportation's University Transportation Centers Program. However, the U.S. Government assumes no liability for the contents or use thereof.

California Department of Transportation (CALTRANS) Disclaimer

The contents of this report reflect the views of the authors, who are responsible for the facts and the accuracy of the information presented herein. This document is disseminated under the sponsorship of the United States Department of Transportation's University Transportation Centers program, in the interest of information exchange. The U.S. Government and the State of California assumes no liability for the contents or use thereof. Nor does the content necessarily reflect the official views or policies of the U.S. Government and the State of California. This report does not constitute a standard, specification, or regulation. This report does not constitute an endorsement by the California Department of Transportation (Caltrans) of any product described herein.

Disclosure

Dr. Mohammad Abdullah Al Faruque conducted this research titled, "Software and Hardware Systems for Autonomous Smart Parking Accommodating both Traditional and Autonomous Vehicles" at UC Irvine. The research took place from 1/1/20 to 4/1/21 and was funded by a grant from Caltrans in the amount of $65,000. The research was conducted as part of the Pacific Southwest Region University Transportation Center research program.

Agreement Number 65A0674 TO-027

Final Technical Report

Project Title: ***Software and Hardware Systems for Autonomous Smart Parking Accommodating both Traditional and Autonomous Vehicles***

PI: Mohammad Abdullah Al Faruque
Department of Electrical Engineering and Computer Science, University of California, Irvine, Irvine, CA
Email: alfaruqu@uci.edu
Team Members: Mohanad Odema, Luke Chen

1. Introduction

Parking infrastructure is suffering from congestion as the number of vehicles circulating in urban areas is growing and expansion is not a cost-effective solution. In parallel, developments in autonomous vehicle technology mean that driverless vehicles are predicted to be in circulation by the 2020s and makeup 40% of vehicle travel by the 2040s. Expected benefits of autonomous vehicle travel include reduced congestion through vehicle sharing and reduced walking distance for passengers who can be dropped off chauffeur-style by autonomous vehicles. However, empty vehicle cruising, or the case in which autonomous vehicles cannot efficiently locate parking and circle instead, can potentially increase congestion. Given that this new technology has the potential to exacerbate existing congestion issues, it is necessary to develop a solution for parking congestion integrated with autonomous vehicles. Our project addresses this issue by providing a full-stack solution including sensors to monitor occupancy, Fog systems to perform local data pre-processing, and SDR radios to communicate with autonomous vehicles.

Current infrastructure supports parking guidance information and a parking reservation system for traditional vehicles with smartphone-equipped users. DSRC has also been successfully employed in V2V and V2I communications. As such, the research challenge is integrating autonomous vehicles into existing smart parking platform options. This entails not only securing DSRC connections between smart parking systems and autonomous vehicles, but also ensuring that the system provides sufficient information for successful parking services in real time. The challenge of integration is addressed by developing a full stack system that will accomplish the following: monitor a parking garage's occupancy, classify vehicles within the parking garage, aggregate location data for available spaces and associated mapping data, and assign them to the respective vehicles to be routed to it.

2. System Architecture

2.1 The Wireless Sensor Network (WSN) and the fog

A. Overview

For the purpose of monitoring occupancy in the parking structure, data needs to be aggregated by a local IoT module from the deployed sensors installed around the structure. As such, the Wireless Sensor Network (WSN) responsible for collecting the occupancy data from different sensors distributed around the parking structure is a core subsystem of the proposed full stack solution. From here, different types of sensors were to be assessed based on the criteria of cost effectiveness, ease of setup, and type/quality of information provided. Mainly, these sensors are used for the primary purpose of tracking available parking spaces. Preliminary research revealed that known sensor options include LDR (Light Dependent Resistors) and PIR (Passive Infrared Resistors) seemed like the most suitable candidates for the occupancy monitoring functionality.

However, the sensors on their own cannot be deployed to send their raw data readings to the local IoT module (which in our case would be the Parking Tracker Fog System). As this would require the entire sensors to be connected directly to it; this is quite unfeasible as it would require wired connections to the module which, on its own regard, has a limited number of such connection ports. In this context, the wireless sensor network would be designed to have a multitude of sensing motes as an intermediary between the sensors and the central unit. They would be conceptually deployed around different areas of the structure with the capability to communicate wirelessly with the local module in a star-like topology. Furthermore, each sensing mote can have a multitude of sensors connected to it depending on the number of ports each designated mote has. This would allow extension of the network as the communication is based on a wireless protocol, and in theory allows planning the WSN in an XMesh configuration allowing some motes to be intermediate hops to further extend the network reach. From here, our main design objective was to establish the star communication allowing the local IoT module to aggregate the received sensor data from the sensing motes to maintain the occupancy functionality.

B. Implementation Details

In the proposed implementation, we decided to prototype the wireless sensor network using the multipurpose ESP32 development board [1]. The ESP32 would represent the functionality of a sensing mote having the sensors connected to it on one hand and sending wireless messages to the local IoT module on the other hand. Currently, the sensors used in our implementation are Light Dependent Resistors (LDRs) mounted on the ESP32 boards and interfaced through one of the General-Purpose Input Output Pins (GPIOs) with Analog to Digital Conversion (ADC) functionality. The connection configuration between the sensor and the ESP32 is shown in Fig. 1 and an actual screenshot of the circuit is in Fig. 2. In this circuit, the LDR sensor cathode end is connected to the ADC(1-0) GPIO36 pin which can take in the analog readings by the sensors and convert them to digital values to be processed by the board. It is also connected to a resistor which interfaced

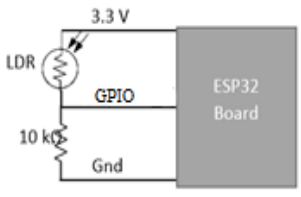

Figure 1: LDR and ESP32 connection configuration

from the other side with the ground terminal of the board. The alternative terminal of the sensor is connected to the 3.3V from the board. These sensors indicate the occupancy status based on the level of sensed light intensity. Furthermore, in order to assess the effectiveness of the message transmission from the sensing motes, we implemented the local IoT module functionality on a Raspberry Pi 3 [6] board to process the occupancy data. Figure 3 shows the connection between the ESP32 board and the local IoT module.

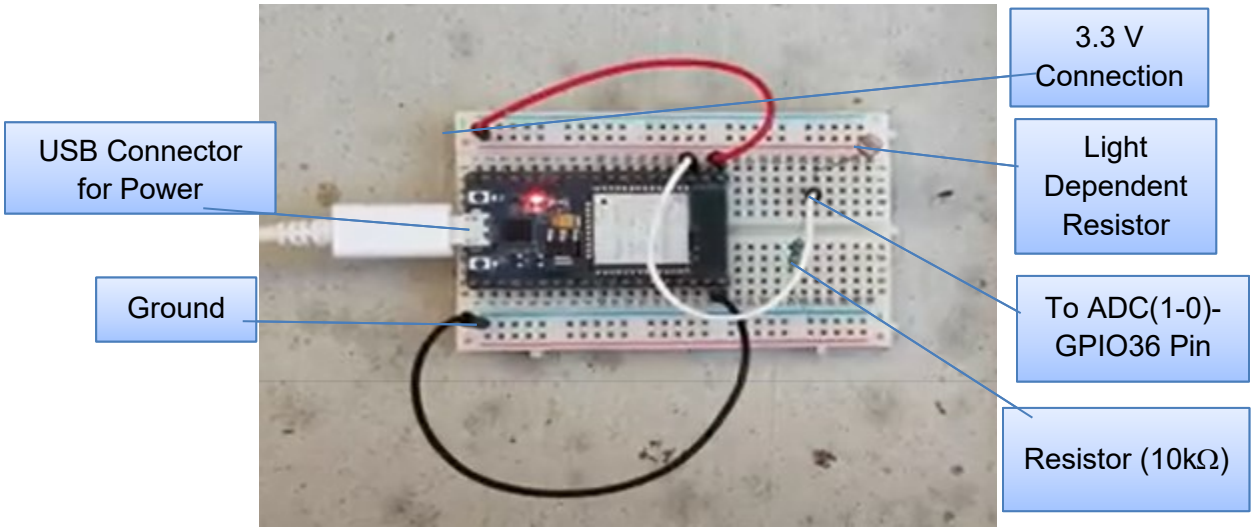

Figure 2: ESP32 Circuit Connection

An additional merit that can be exploited when using the ESP32 board is that ESP32 supports multiple power modes. So, the WSN nodes can be optimized to preserve their energy by switching to one of the lower power modes when they are not transmitting or receiving packets. In this context, the active mode requires the chip to operate around 240 mA operational current. Lower power modes include MODEM sleep, Light Sleep, and Deep sleep; such modes support waking up the main system based on the measurements from the acquired sensor data. The operating current can reach as low as 20mA, 0.8 mA, and 0.01 mA for each of the power modes respectively. This can aid us in our design to have the development board switch between the deep sleeping mode and active mode depending on the changes in the sensor readings associated in this application with the parking spots.

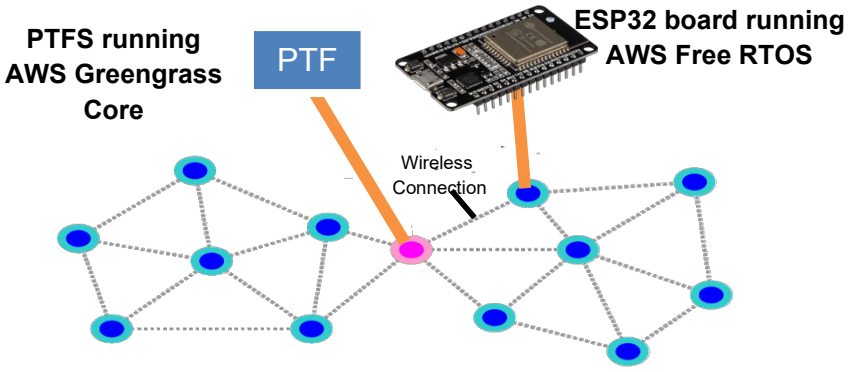

Figure 3: Wireless Sensor Network Design. Currently a star topology between the PTFS and one ESP32 board which can be scaled to include further boards in an XMesh configuration

C. AWS IoT Framework

From here, the AWS IoT framework has been chosen to achieve the connectivity functionalities between different components of the hierarchical design. Mainly because the framework is established to facilitate communication between the different levels of design hierarchy, which in our case is resembled as cloud-fog-edge hierarchy. In this context, the occupancy data would propagate from the sensing mote across the fog device and to the cloud as shown in Fig. 4. To achieve this, AWS FreeRTOS [2] was installed on ESP32 as real-time operating system to facilitate connection of each one of them to the fog node. For the IoT module (Fog device), AWS IoT Greengrass core [3] was deployed on the Raspberry Pi to extend its functionality to act locally on the collected data using the AWS Lambda functions. In addition, devices running AWS IoT Greengrass can act as an intermediary between the sensing nodes and a central cloud; which conceptually is capable of managing multiple fog devices representing multiple parking structures. Detailed steps of how to configure ESP32 to run AWS FreeRTOS and how to configure the Raspberry Pi to run AWS IoT GreenGrass are provided in [4] and [5] respectively.

Figure 4: Occupancy Data Propagation from the ESP32 device through the PTFS and to the central AWS cloud

2.2 PTFS Proof of Concept

A. Overview

For the proof of concept of the PTFS functionality, a hardware-in-the-loop simulation is set up consisting of 3 main components (Figure 5): The Fog Node, the Edge Device, and the Vehicle. The Fog Node is the core of our Parking Tracker Fog System (PTFS). It is implemented using the AWS IoT Greengrass framework on the Raspberry Pi 3 B+ model [6]. The job of the Fog Node is to process occupancy data from the Edge Devices and to handle parking requests from vehicles entering the smart parking structure. The Edge Device is implemented using the multipurpose ESP32 development board which runs the Amazon FreeRTOS kernel and communicates to the Fog Node through MQTT publisher/subscriber protocols. The Edge Device is an intermediary unit between the multitude of sensors and the Fog Node. It is a sensor mote that aggregates and sends sensor data to the Fog Node for processing. The Vehicle we used is the TI Robotics System Learning Kit (TI-RSLK) robotics vehicle from Texas Instrument [7]. It acts as the autonomous/traditional vehicle agent that interacts with a smart parking structure. The vehicle currently communicates wirelessly to the Fog Node through a server using the TCP/IP protocol. Incorporating support for DSRC shall be detailed in the following subsections 2.3 and 2.4.

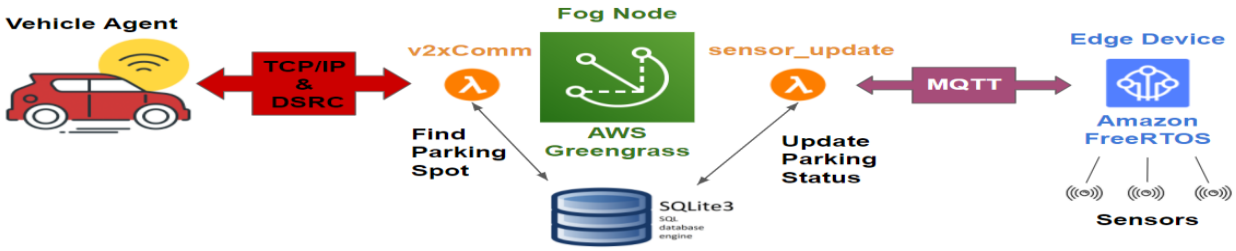

Figure 5: Hardware-in-the-loop structural Diagram

B. Implementation Details

Figure 6: v2xComm lambda function log

The AWS IoT framework uses lambda functions to run application logic on local Fog Nodes. In our application, the Fog Node has two main tasks. First to manage the parking requests of Vehicles and second to process occupancy sensor data from the Edge Devices. Therefore, we deployed two lambda functions to the Fog Node called sensor_update and v2xComm. Sensor_update manages the communication between the Fog Node and the Edge Device while V2xComm manages the communication between the Fog Node and the Vehicle. When a Vehicle makes a parking request to the Fog Node, it will send some metadata such as vehicle id and type (compact, regular or handicap) illustrated by Figure 6. In the Fog node, the v2xComm lambda function hosts a server that will process the parking request. A local SQLite database on the Fog Node stores the parking occupancy status of all the spots on the hardware simulation and is used to query for available parking spots based on the vehicle type. Once a suitable parking spot is found, the Fog Node will set the parking spot status to “reserved” in the local database and send the parking spot information to the Vehicle. In the event that the Fog Node receives information from the Edge Device that a parking spot’s occupancy status has changed, the sensor_update lambda function will trigger and it will update the occupancy status of that parking spot in the local

database. The Fog Node also handles the occupancy indicators and will change the color of the LEDs based on the parking occupancy status (green = empty, yellow = reserved, red = occupied).

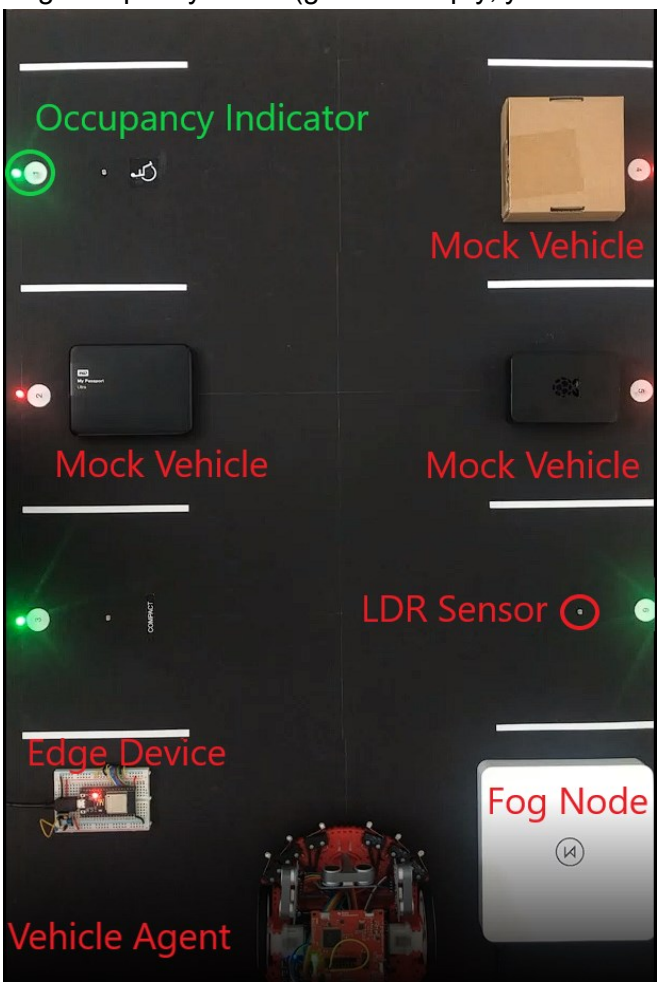

Figure 7: Hardware-in-the-loop simulation

The Edge Device utilizes the Amazon FreeRTOS kernel to establish a secure wireless connection with the Fog Node. When a Vehicle arrives at its reserved parking spot, the light-dependant resistor (LDR) sensor, shown in Figure 7, will sense a change in the light intensity value. If this change is greater than some preset threshold value, the ESP32 will detect that the parking spot is being occupied. It will then send this information to the Fog Node via MQTT publisher/subscriber protocol. The Fog Node will then reflect the occupancy status in the database and provide visual feedback through the occupancy indicators (LEDs). The code that communicates between the Edge Device and the Fog Node is based on our modification of the AWS IoT Greengrass discovery demo application [8]. We modified the code so that the ESP32 will continuously loop over the six LDR sensors to check if any of the sensors meet the threshold value.

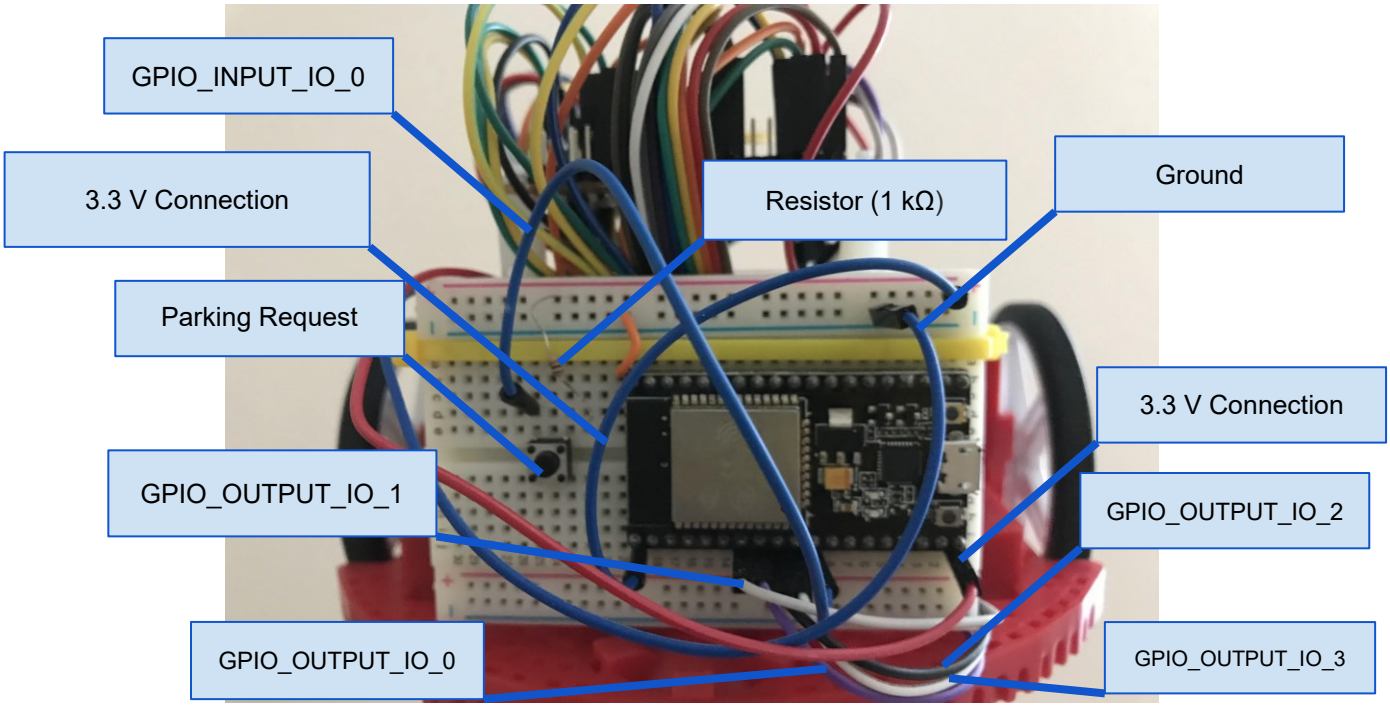

Figure 8: TI robotics vehicle attached with ESP32 for wireless capabilities

TI robotics learning kit is used as the test vehicle. Because the TI module does not come with a wireless module, we used a ESP32 microcontroller to handle wireless communications between the Vehicle and the Fog Node as shown in Figure 8. When the ESP32 boots up, it waits for a button press. When the button is pressed, it attempts to connect to the server created by the v2xComm lambda function on the Fog Node. When the connection is successful, it sends the Fog Node a message that includes its vehicle id and the type of vehicle (compact, regular or handicap). When the ESP32 successfully receives a parking spot number from the Fog Node, it will communicate this information to the Vehicle using 3 output pins (GPIO_OUTPUT_IO_0, GPIO_OUTPUT_IO_1, GPIO_OUTPUT_IO_2). These 3 pins are used to encode the parking spot numbers in binary format (ex. 001 = 1, 010 = 2, 011 = 3, etc.). Another pin (GPIO_OUTPUT_IO_3) is used to let the ESP32 know when to read the 3 pins for decoding. *Recall that these wired connections are within the demo vehicle between the TI kit and the ESP32 microcontroller, in which the ESP32 was added to incorporate wireless support for the demo vehicle.* Based on the decoded parking spot number, the Vehicle will then proceed to navigate itself towards the reserved parking spot via a pre-programmed path.

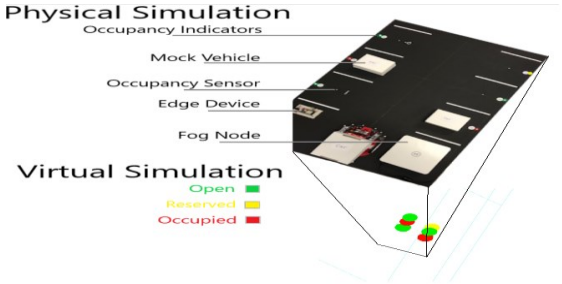

Figure 9: Hardware-in-the-loop and software simulation layout

C. Parking Occupancy Simulation

In order to test the compatibility of our hardware simulation with real parking occupancy monitoring systems, we integrated a parking occupancy software that mimics a parking lot environment. The software simulation is based on the Parking Occupancy Simulator from Geospatial Technologies Research Group [9]. We made modifications to the code for it to be compatible with our hardware simulation. The simulation utilizes Geographic Information Systems (GIS) data to map the parking environment and uses model drivers as agents to interact with the parking environment. Six spots on the software simulation are reserved to reflect the physical parking spots on our hardware simulation. As shown in Figure 9, there are visible parking spots on the map differentiated by colors of green, yellow, and red which indicate empty, reserved and occupied, respectively. As the simulation runs, the driver agents will begin to occupy parking spaces throughout the map. The software simulation communicates with the hardware simulation (Fog Node) through a tcp server. When a parking spot reservation request is detected by v2xComm or when a change in parking occupancy data (empty or occupied) is sent to sensor_update, the Fog Node will send the parking spot id of the affected parking spot to the simulation through the server. The server has the mapping of the physical parking location (local spot id) to six corresponding parking spaces (global spot id) on the simulation. Therefore, any changes to the parking occupancy status in the physical simulation will be reflected in the software simulation in real-time.

One caveat for the simulation software is that the original ArcGIS server that hosts the geographic data for the map is currently no longer in service. Therefore, we devised a workaround with copies of the GIS data stored on several json files and have a python flask web server that runs in the background to service the GIS data. Therefore, it is necessary to run the python web server code prior to launching the simulation in order for it to work correctly.

2.3 DSRC Integration

A. Overview

Integrating the Dedicated Short-Range Communication (DSRC) with the Parking Tracker Fog System (PTFS) is to enable communication between the PTFS and Intelligent Vehicles (IV). DSRC is a variation of the IEEE 802.11 short-range wireless communication protocol called IEEE 802.11p. DSRC operates in the 5.9 GHz band with a bandwidth of 75 MHz. Just like the WiFi 802.11a/g protocols, DSRC also follows the Open Systems Interconnection model (OSI model) for implementing various layers of this protocol. The fundamental layers of the OSI model are the physical layer (PHY layer) and the data link layer which consists of the Logical Link Control (LLC) and Medium Access Control (MAC) sub-layers. Above the data link layer are the high level

abstraction layers like the application layer and various other sublayers in between. To enable communication using DSRC, we need to at least have the PHY layer and LLC/MAC layers.

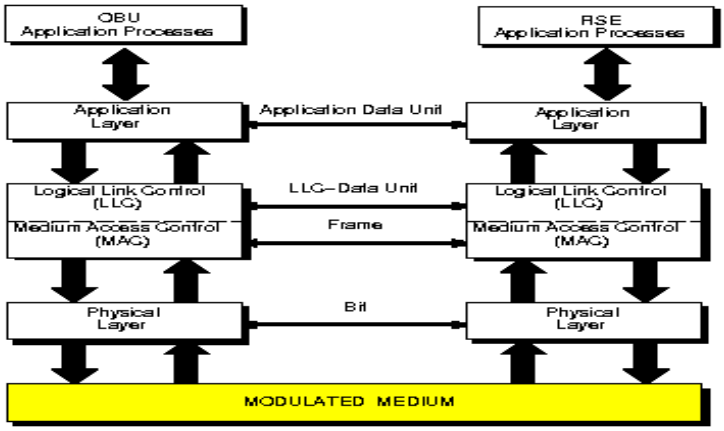

Figure 10: DSRC architecture [1]

B. Implementation Details

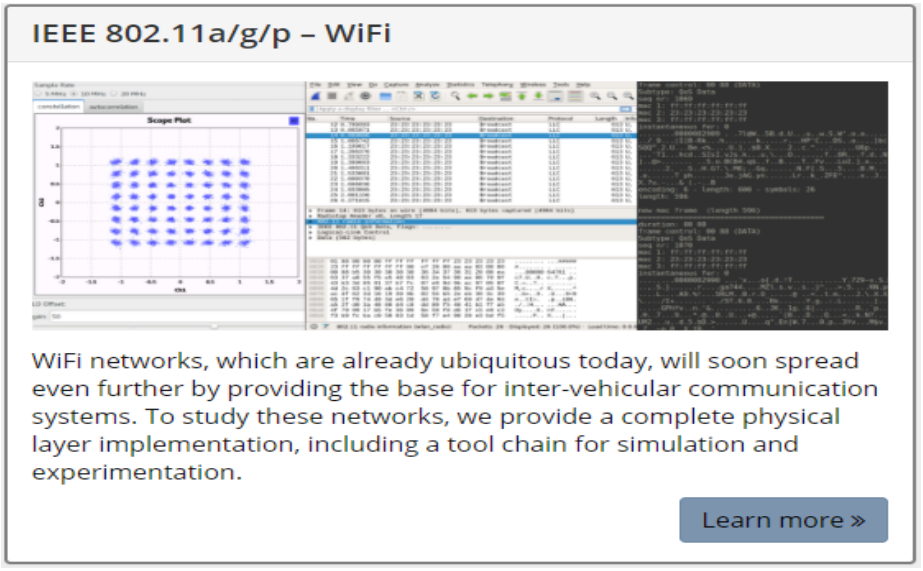

Figure 11: Wime-project

To build the PHY and MAC layers, we looked into various resources that have implemented some kind of PHY and MAC layers for the DSRC standard. We found that the Wime-project [11] was the best option in the open source area for wireless communication modules. The Wime-project is an open-source collaboration project between various college institutions. The creators of the project also published a paper detailing the performance assessment of their WiFi module in [12]. They created a DSRC Software Defined Radio (SDR) prototype that is compatible with the Ettus B210 & N210 boards, which are boards that are capable of operating in the 5.9 GHz band for DSRC communication. We integrated their IEEE 802.11a/g/p – WiFi module into our system [13]. The module provides a complete physical layer implementation for a WiFi transceiver that works

with the IEEE 802.11p protocol. The module was created using GNU Radio, a well-known real-time signal processing framework for use in Software Defined Radio (SDR) systems.

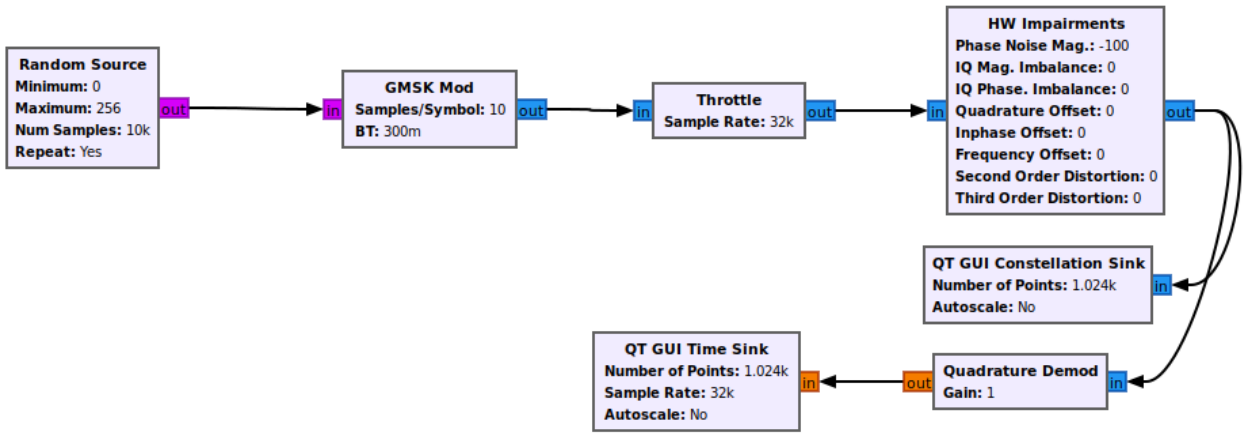

Figure 12: Example of GNU Radio Flowgraph

Just like Simulink and LabView, GNU Radio is a graph-based application that enables users to design software radios to be used with hardware to create software-defined radios (SDR). Within each flowgraph, for example Figure 12, blocks are contained that perform simple mathematical operations to complex signal processing. The flow of data is represented by the arrows that connect each block, and the type of input/output data is determined by the color of the input/output ports of each block. GNU Radio comes with an extensive library of blocks that perform standard signal processing tasks. Just by connecting blocks together, we can create more complex blocks that can model or perform custom signal processing functions.

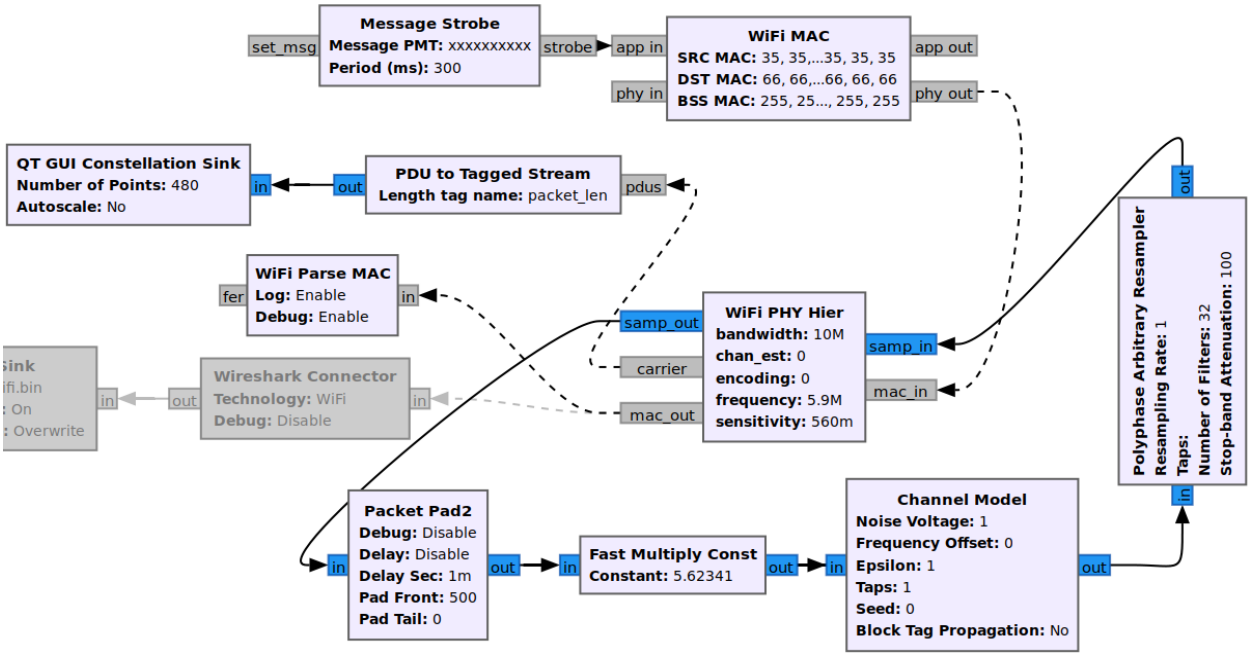

Figure 13: WiFi Loopback Flowgraph

Having installed GNU Radio following the official GNU Radio webpage [13] as well as installing the custom blocks created by the Wime-project, we ran a WiFi loopback test that assess the functionality of the PHY and MAC layers of the custom blocks. Figure 13 shows the wifi loopback

flowgraph and the blocks that construct it. The main goal of this loopback test is to verify the correct functionality of the WiFi PHY Hier block. In order to do so, a message strobe with 10 x's is sent through the WiFi MAC and into the WiFi PHY Hier's mac_in input port. The WiFi PHY Hier is a complex block that contains a multitude of complex signal processing blocks inside. For more information on the details and implementations of the WiFi PHY Hier block with signal processing, please refer to the paper in [12]. To give a general explanation of the functionality of the WiFi PHY Hier block, it basically applies the signal processing to the data and outputs it through the PDU to Tagged Stream block and into a graphical user interface (GUI) constellation sink. At the same time, the processed signal was extracted and sent out from the samp_out output port to be reconfigured by the Packet Pad2, Fast Multiply Const, Channel Model, and Polyphase Arbitrary Resampler, so that it can be sent back to the WiFi PHY Hier block as the data for the test.

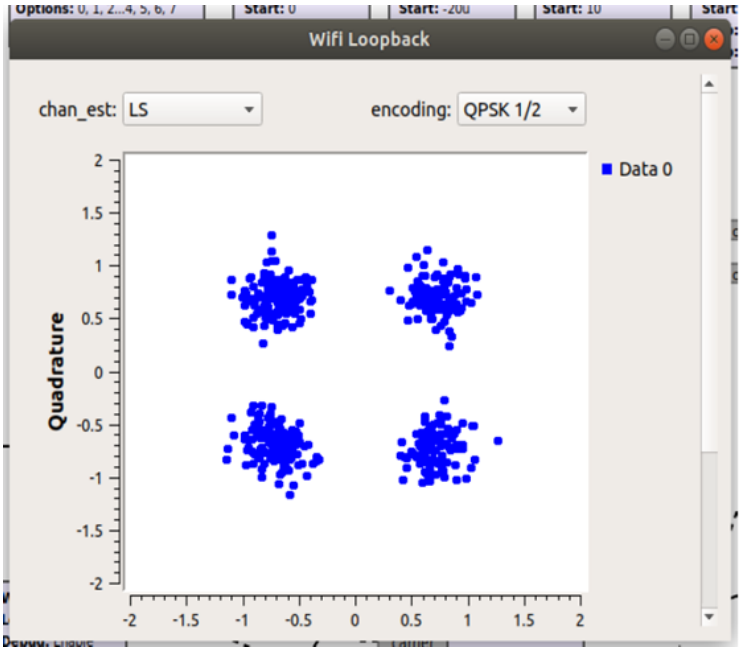

Figure 14: WiFi loopback Quadrature

The results of the loopback test are shown in Figure 14 through the constellation diagram. The number of constellation points in a diagram depends on the encoding type and it specifies the size of the symbols that can be transmitted by each sample of data. In this case, the encoding type is QPSK ½ with 4 points which represents a modulation scheme that can separately encode 4 combinations of two bits: 00, 01, 10, and 11, and so can transmit two bits per sample. The clear separation in the constellation shows that the samples are being interpreted correctly and there is no misrepresentation of the samples.

C. Hardware and Software Specifications

For the hardware we decided to use the Ettus Board B210 from Ettus Research. The Wime-project has shown successful integration of their IEEE802.11p module with the USRP N210 as well as the B210. The USRP B210 is a fully integrated, single board Universal Software Radio Peripheral (USRP) platform with continuous frequency coverage from 70 MHz - 6 GHz. It has full support for development in GNU Radio with its USRP Hardware Driver (UHD). Once we

received the USRP B210 boards and installed the necessary drivers for the device from Ettus Research, we ran the benchmark software and verified that the B210s are operating correctly without faults.

2.4 End-to-end Integration

A. Overview

The integration of the Dedicated Short-Range Communication (DSRC) with the Parking Tracker Fog System (PTFS) can be broken down into three parts shown in Figure 15: the vehicle agent, the parking tracker fog node, and the parking simulation. Each component consists of subcomponents that allow the communication flow to work correctly. In the following sections, we will be breaking down the three main parts alongside their subcomponents in detail.

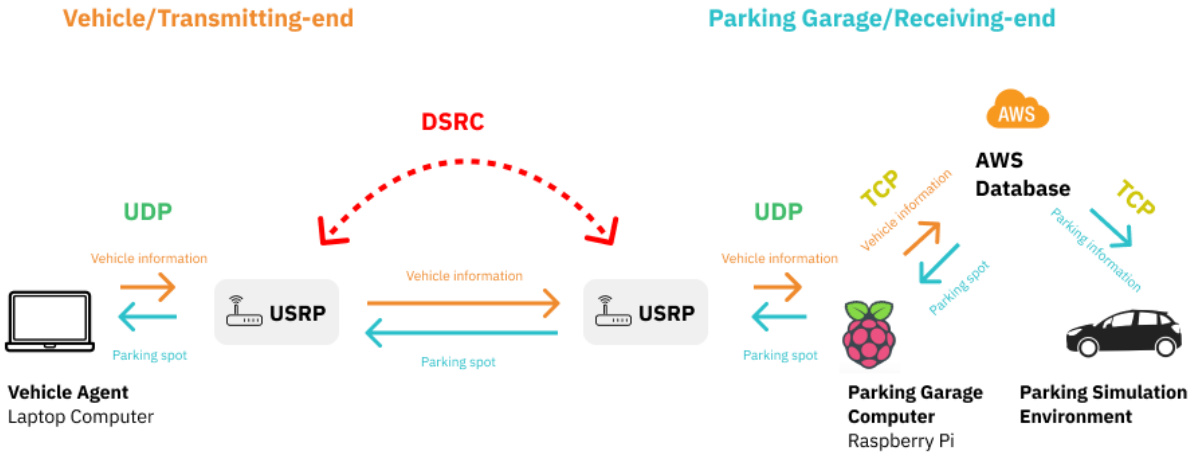

Figure 15: Hardware Diagram

B. Implementation Details

Vehicle Agent: the communication between DSRC equipped vehicle agents and the smart parking fog node starts from the generation of a parking request message. In our implementation, the vehicle agent is represented by a laptop computer that is equipped with a Universal Software Radio Peripheral (USRP). *Note that the DSRC-based agent is orthogonal to the one implemented using the TI Robotics kit, as each agent reflects different operational scenarios (i.e., a vehicle that supports DSRC communication and another which does not).* The USRP that we are using is the Ettus Research B210 board. The B210 boards allow the transmission and reception of signals in the 5.9 Ghz band, but the processing of the signals is accomplished through GNU Radio, a software development toolkit for signal processing in.

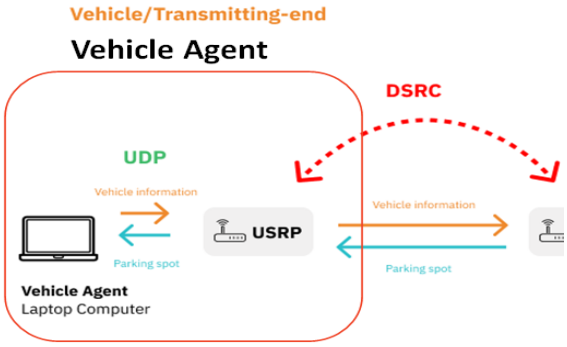

Figure 16: Hardware Diagram (Vehicle Agent)

As shown in Figure 16, when a parking request message is prepared by the vehicle agent, it will send the message to the USRP to be transmitted over-the-air to the fog node. After sending the parking request message, the vehicle agent will wait to receive a parking spot reservation message from the fog node. How the USRP sends and receives data over-the-air is shown by the expanded view of the USRP in Figure 17, which shows the GNU Radio blocks that performs signal processing in the background.

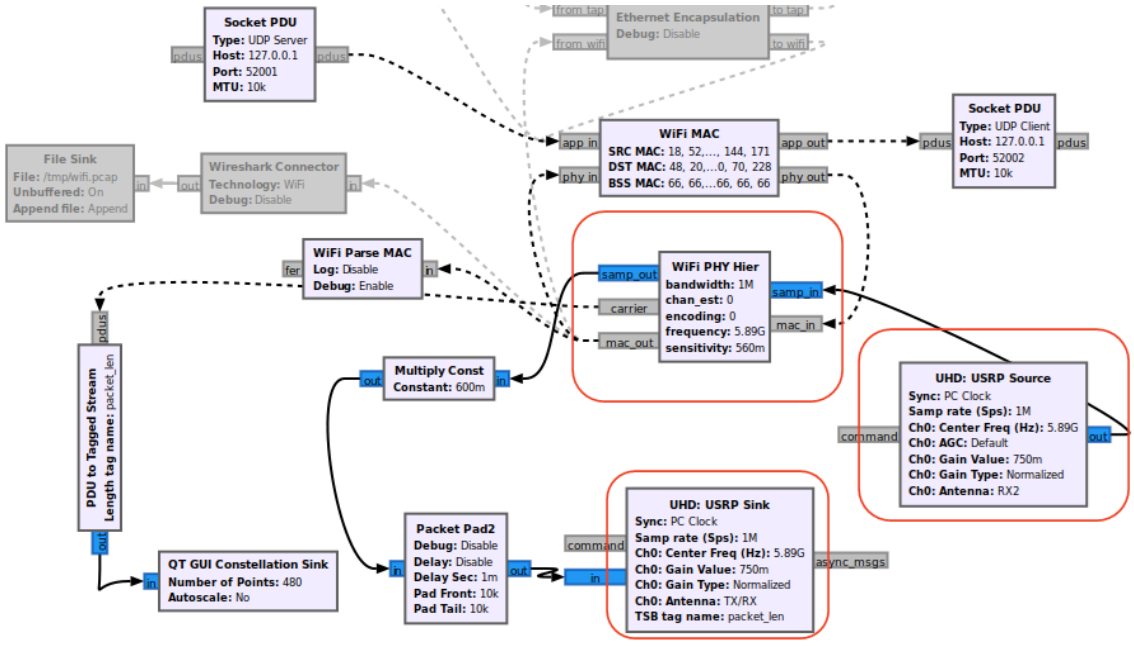

Figure 17: USRP Expanded View (GNU Radio Blocks)

The USRP gets the parking request message from the vehicle agent through the socket PDU. The message goes through the WiFi MAC and WiFi PHY Hier blocks where the corresponding MAC and PHY header information is added to the original message. The WiFi PHY Hier block models the IEEE 802.11p physical layer protocol. Finally, the padded message with the header information goes to the UHD: USRP Sink block which transmits the padded message over-the-air in the 5.9 Ghz band. The UHD: USRP Source block receives messages over-the-air from the

5.9 Ghz band. Both UHD blocks are device driver modules that allow the GNU Radio software to interact with the B210 board's transmit (TX) and receive (RX) channels.

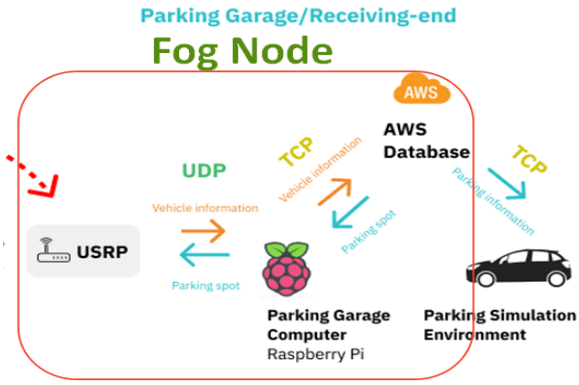

Figure 18: Hardware Diagram (Fog Node)

Fog Node: The fog node in our implementation (Fig. 18) is built on a Raspberry Pi 3 B+. Just like the vehicle agent, the fog node is also equipped with a USRP device that allows it to communicate through DSRC using a B210 and the GNU Radio software. Alongside the GNU Radio software for signal processing, the fog node also runs the Amazon Web Service (AWS) greengrass service which handles the smart parking system's reservation functions using a local database of parking spots. We will refer to the AWS greengrass node as the AWS database. When the fog node receives the parking request message from the USRP device, which was explained in the vehicle agent part, it will relay this information to the AWS database. The AWS database will parse the message to extract relevant information and decide which parking spot to reserve for the vehicle. When it finds a suitable parking spot, the AWS database will send the reserved parking spot message back to the USRP to send it over-the-air to the requesting vehicle agent. It should be noted that installing all the necessary packages on Raspberry Pi 3 B+ was challenging and time consuming due to the limited memory resources of the device. We recommend an upgrade to Raspberry Pi 4 devices if the same brand of devices is to be used for the fog nodes.

To explain the inner workings of the communication that goes on inside the fog node, we provide a communication diagram from the point of view of the fog node in Figure 19.

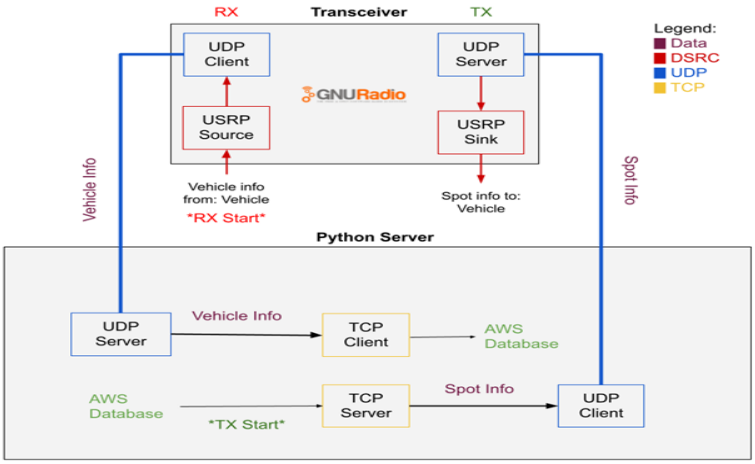

Figure 19: Communication Diagram (Fog Node)

Because there is no direct communication between the GNU Radio software and the AWS Database, since they are two independent programs running on the fog node, we had to create a python program that acts as a bridge between the two components. When GNU Radio receives a parking request message through USRP Source, it sends this information to the UDP port of the python server. The python server will relay the information to the AWS Database through a TCP port. When the AWS database returns a parking spot reservation, the python server will relay the information back to GNU Radio and the information will be transmitted over-the-air to the vehicle agent through the USRP Sink. The reason we don't use a UDP/TCP port to communicate directly from GNU Radio to the AWS database is because the UDP port in GNU radio is different from the standard UDP port and the python program imports the GNU radio libraries to use the special UDP port for communication. Since it's a custom python program, we can also use standard TCP socket libraries to communicate with the AWS database.

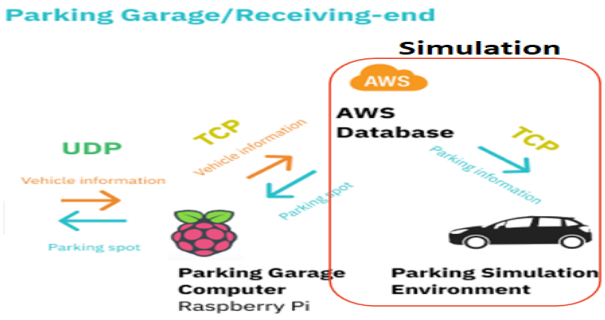

Figure 20: Hardware Diagram (Simulation)

Simulation: The last component for the DSRC integration is the simulation, shown in Figure 20. When the fog node receives the parking request message and before it parses the message, it also sends this message to the parking simulation software. The parking simulation software also parses the message for relevant information and will generate a parking agent to park at a reserved spot in the simulation. This component exists for DSRC to work with our simulation software. It is not a necessary component for implementing PTFS with DSRC capabilities, only the vehicle agent and fog node components are necessary.

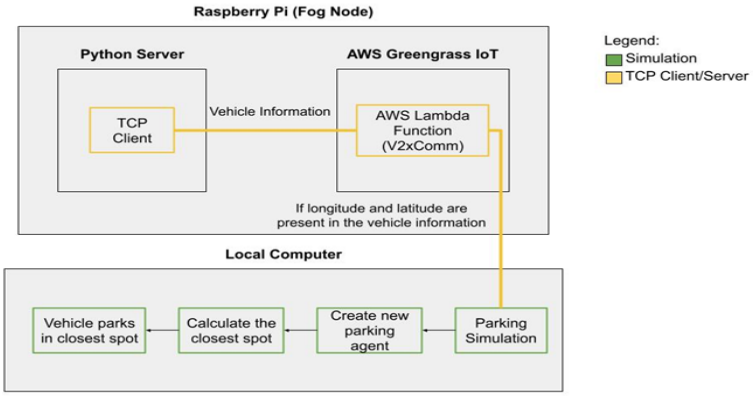

Figure 21: Communication Diagram (Simulation)

We use the communication diagram in Figure 21 to explain the communication between the fog node and the simulation. The TCP client inside the Python Server shown here is the same TCP client that was shown in Figure 19. This diagram shows that the information is also sent from the AWS database to the simulation and that the simulation creates a corresponding parking agent that will park at a reserved spot.

C. Message Protocols

We have been talking about sending and receiving parking request messages and reserved parking spot messages between the vehicle agent, the fog node, and the simulation, but we have not explained what these messages contain. The exact messages that we are sending and receiving is our variant of the Basic Safety Message (BSM).

The BSM format was carefully designed to minimize the message size. Smaller messages can help reduce DSRC channel congestion. To keep BSM sizes small, their content is structured into two parts. Part I – known as Basic Vehicle State – is mandatory and contains those data elements and data frames that must always be included in a BSM. BSM Part I has a fixed size of 39 bytes.

Table 3. BSM Part I Data Elements and Data Frames.

BSM Data Item	Sequence	BSM Part	Type	Bytes
Message ID		I	Data Element	1
Message Count		I	Data Element	1
Temporary ID		I	Data Element	4
Time		I	Data Element	2
Latitude	PositionLocal3D	I	Data Element	4
Longitude		I	Data Element	4
Elevation		I	Data Element	2
Positioning Accuracy		I	Data Frame	4
Transmission & Speed	Motion	I	Data Frame	2
Heading		I	Data Element	2
Steering Wheel Angle		I	Data Element	1
Accelerations		I	Data Frame	7
Brake System Status	Control	I	Data Frame	2
Vehicle Size	VehicleBasics	I	Data Frame	3

Part II, which includes the Vehicle Safety Extensions and Vehicle Status data frames, is optional. Typically, vehicles periodically broadcast BSM Part I only: specific events, such as emergency braking and control loss, can be described by setting the corresponding event flag in BSM Part II.

Figure 22: Basic Safety Message Part I Elements [14]

BSM is a message protocol that is used to communicate information between vehicles equipped with DSRC capabilities, and was designed to be used in safety applications. BSM consists of two parts, Part I - known as Basic Vehicle State is mandatory and must always be included in a BSM (shown in Figure 22), while Part II, which includes the Vehicle Safety Extensions and Vehicle Status is optional. Typically, vehicles periodically broadcast BSM Part I only. Therefore, we only consider BSM Part I and its corresponding data elements and data frames for our implementation. In our case, we utilize the Latitude and Longitude data elements which are part of the PositionLocal3D sequence. Elevation is currently not being used but it could be useful in the case of multilevel parking garages such that the elevation position of the vehicle can be taken into account when reserving a parking spot that is on the same floor as the vehicle.

It is important to note that due to the fact that BSM is used solely for safety communication, it is not designed to work with a smart parking application, therefore we use a variant of the BSM to convey the necessary information for our system to work.

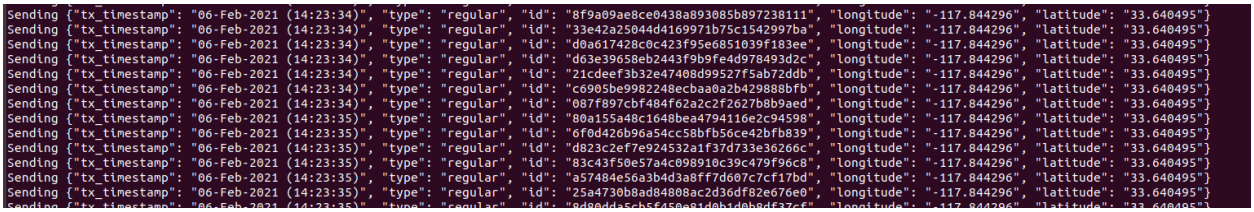

Figure 23: Parking Request Message (Our BSM Variant)

Our BSM variant for parking request message, shown in Figure 23, sent from the vehicle agents to the fog node consists of the following: Tx_timestamp (the time the message is transmitted), Type (the type of vehicle i.e. compact, sedan, etc.) Id (a unique vehicle identifier), longitude (current GPS longitude), and latitude (current GPS latitude). For the sake of demonstration, we did not include all the data elements and data frames that are part of the BSM but only the ones useful to us. However, once can modify and include the full BSM as they desire.

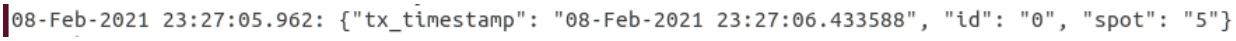

Figure 24: Reserved Parking Spot Message (Our BSM Variant)

On the other hand, the other reserved parking spot BSM (Figure 24.) variant message sent from the fog node to the vehicle agents contains the following: Tx_timestamp (the time the message is transmitted), Id (a unique fog node identifier in case there are multiple parking garages) and spot (the reserved parking spot). The spot data shows the parking spot number that is reserved, this is so that it is compatible with our simulation, however in a real smart parking system, the spot data will be replaced by the longitude and latitude of the physical spot so that the vehicle can navigate to that position.

As a final note to the message protocol, we wanted to mention that currently there is no standard message set that contains data elements specific to smart parking system applications (i.e., parking spot availability, routing information, etc.). BSM was developed for safety message exchange within V2x applications and so our BSM variant would not be recognized by vehicles and they would not know what to do with these messages. To develop a working smart parking system that accommodates smart vehicles equipped with DSRC capabilities would require a protocol standard agreement amongst the vehicle manufacturers and the protocol standard creation companies. Our preliminary research suggests smart parking system integrators to look into the SAE J2945/x series that is currently in development by SAE International which specify application specific message sets for DSRC systems, however most of the standards are listed as work-in-progress and none have been created for smart parking applications.

3. Experiments

3.1 Assessments and Experiments

A. Hardware Setup

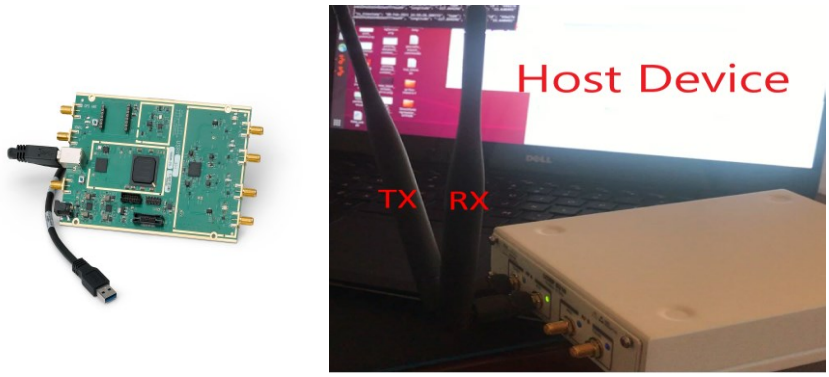

Figure 25: Ettus Research B210

The Ettus Research B210 boards [15] are used as our USRP device. In order for the B210s to communicate with GNU Radio we need to install the USRP Hardware Driver (UHD) onto the host device, a comprehensive installation guide for installing UHD and GNU Radio is shown in [16]. As can be seen from the image, the B210 boards connect to the host device through USB 3.0 Type B. Once UHD drivers are properly installed on the host device, it is simply plug & play by connecting the device directly to the host device's USB port (recommended is USB 3.0 for best data rates). To verify the operation of the USRP device, we followed the instructions in [17] which has connection tests, transmission and reception benchmarks for us to test our device. Note that most if not all of the support for both UHD and GNU Radio is for linux-based Operating Systems (OS) like Ubuntu so it is recommended to use the supported Operating System for the host device.

B. DSRC Latency

We conducted a simple latency test to determine the time it takes to send and receive our custom messages over-the-air to two devices equipped with the B210s. These two devices are a laptop and a desktop computer both running the Ubuntu 18.04 LTS OS with GNU Radio version 3.8 and UHD driver version 4.0. The test was conducted with the two devices at separate ends of a room around 20-25 feet apart from each other. From figure 26 we can see that the latency is between 43-78 milliseconds. Note that this is a simple test of transmission and receive which does not account for environmental factors that would interfere with the signal.

C. DSRC Integration Tests

Once we confirmed the correct operability of our B210 devices, we integrated the USRP into our existing smart parking system. Figure 27 shows the transmission of the parking request message from the vehicle agent which is represented by the python program running on a local computer. The bottom portion of the image is GNU Radio running in the background and transmitting the message over-the-air. This is the demonstration of the hardware diagram (vehicle agent) portion.

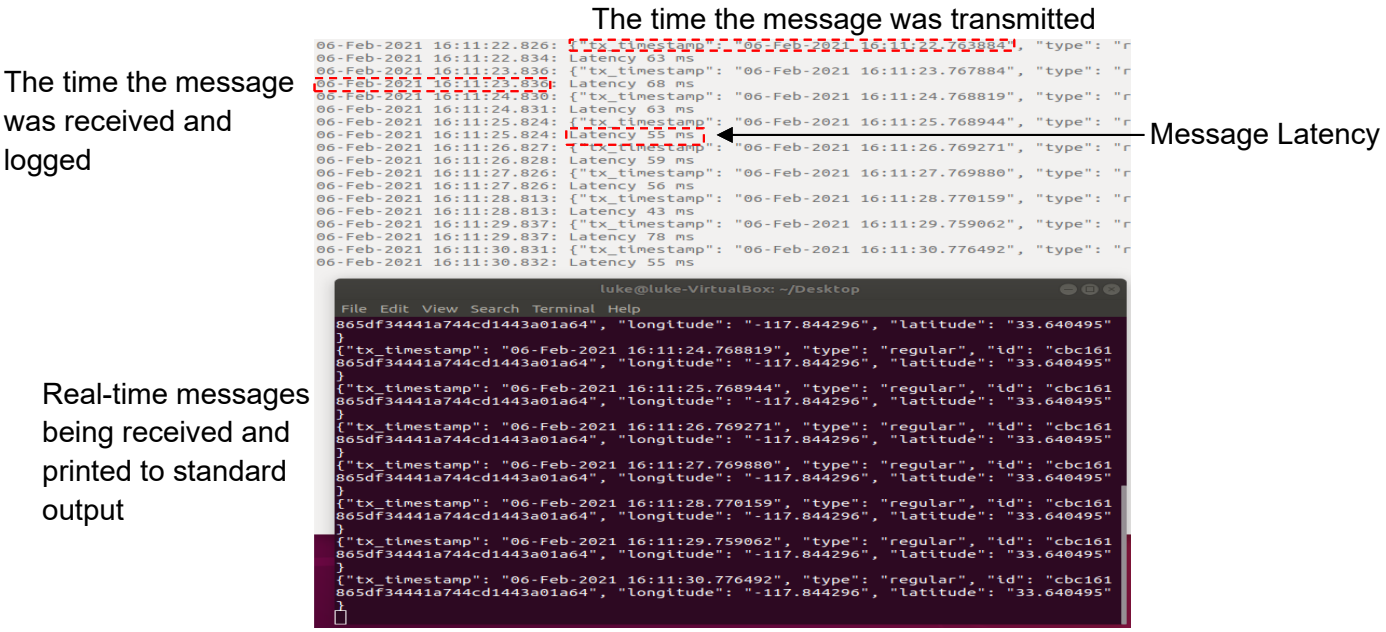

Figure 26: DSRC Latency Test

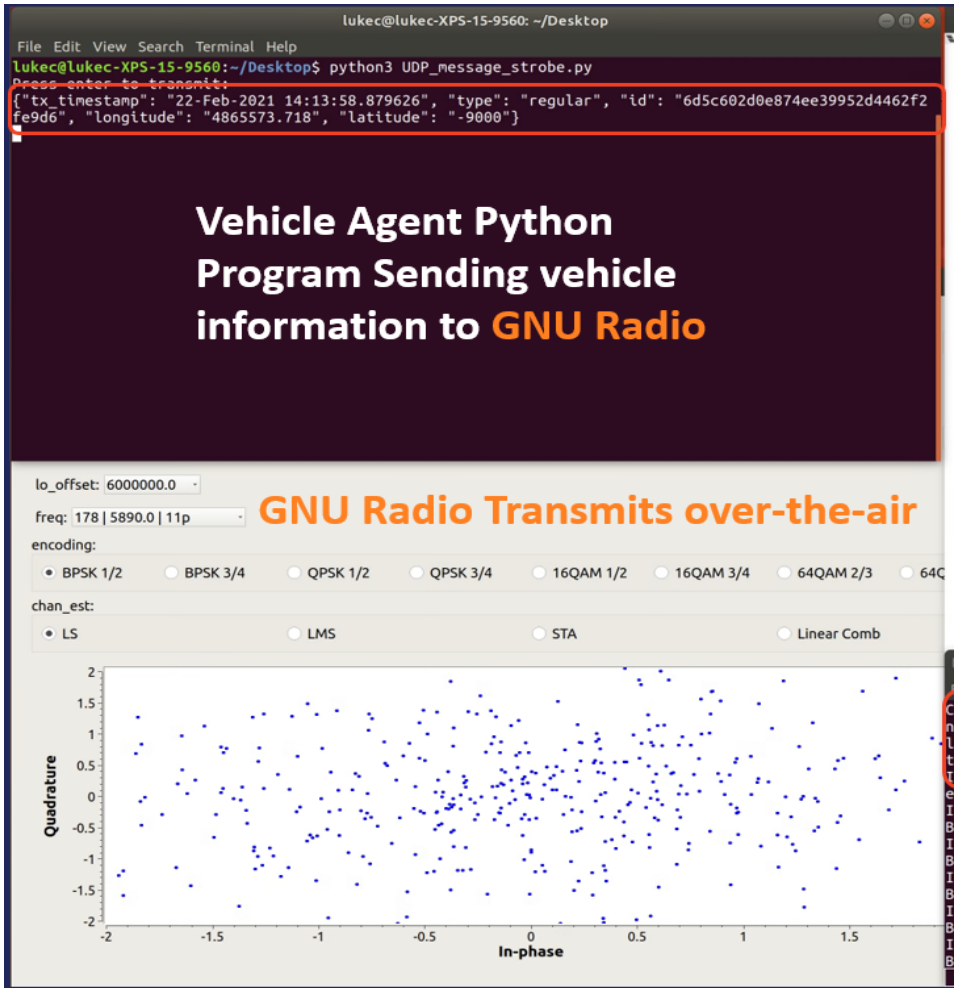

Figure 27: DSRC Integration (Vehicle Agent)

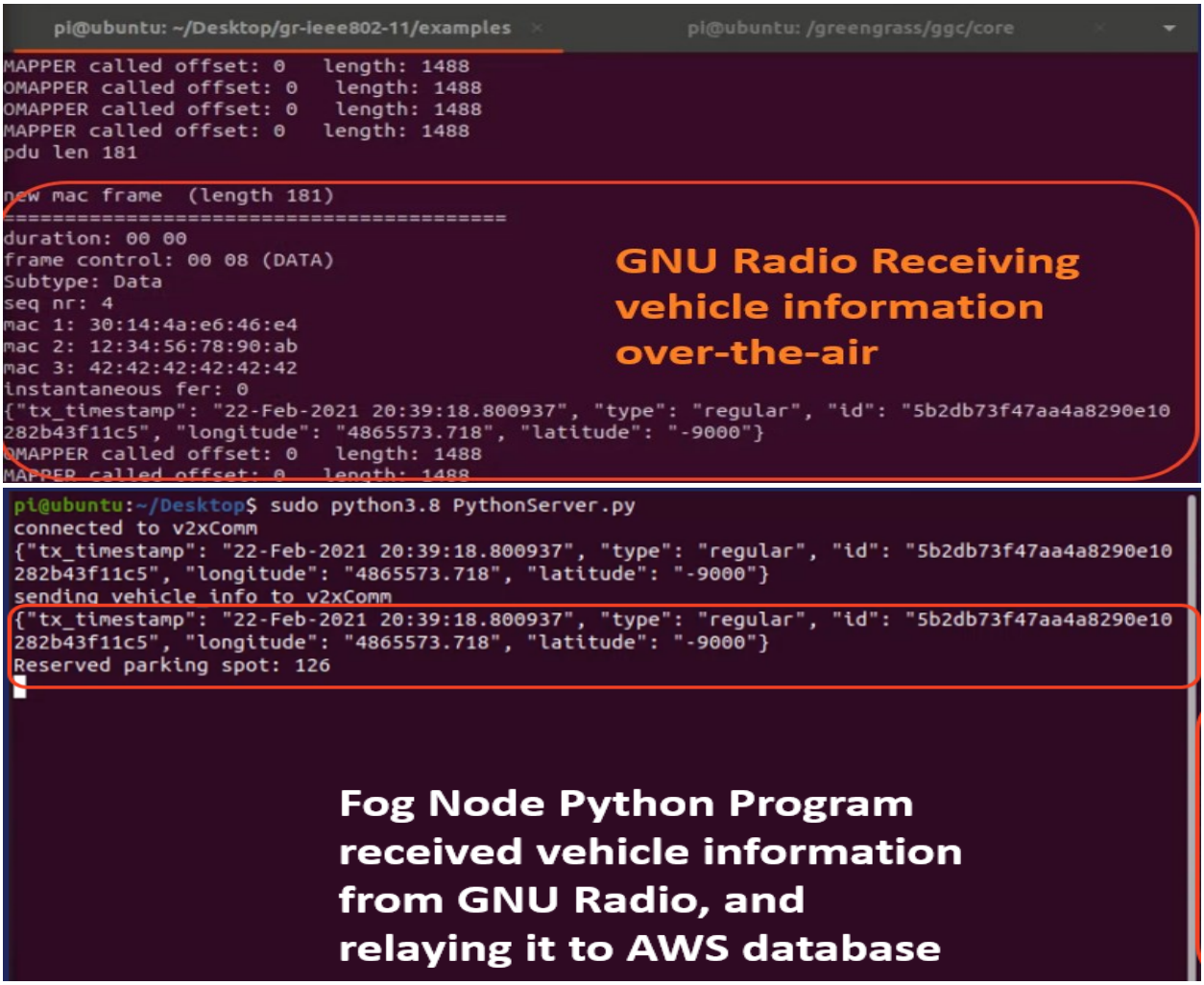

Figure 28: DSRC Integration (Fog Node)

The top image of figure 28 shows the parking request message being received by GNU Radio for the fog node. The bottom image shows the python program relaying the message to v2xComm which is the AWS database that handles the smart parking reservation. Once the AWS database queries and finds a parking spot, the python program receives the reserved parking spot and will relay the reserved spot message back to GNU Radio to transmit over-the-air to the vehicle agent.

Figure 29 shows the parking request message being received by our simulation software from the AWS database. Remember that the AWS database first receives the message over-the-air through the USRP and before it parses the message it also relays the message to the simulation through TCP. When the simulation receives the parking request message, it will parse the longitude and latitude information from the parking request message. The simulation will generate a BSM agent at the given longitude and latitude location and reserve a parking spot that is closest to the longitude and latitude of the vehicle. Finally, the vehicle will proceed to park at its reserved parking spot.

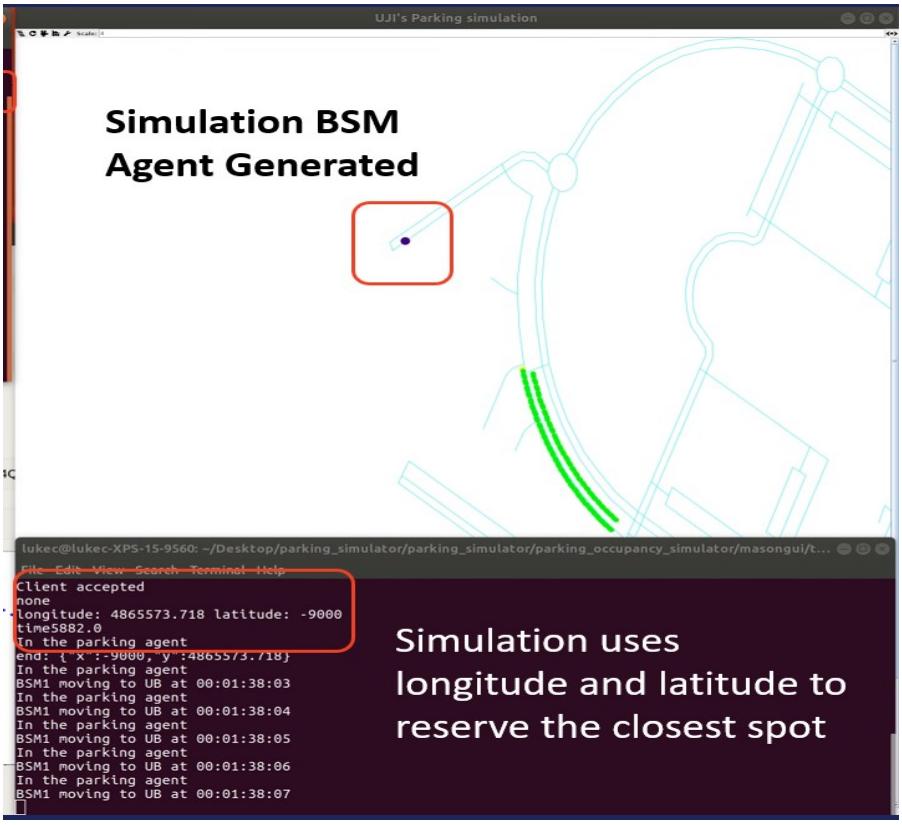

Figure 29: DSRC Integration (Simulation)

3.2 Simulation Scenarios

One of the tasks for us was to consider real life parking scenarios in our smart parking system. This section addresses the concerns of Multi-entry parking garages, parking in unassigned spots, and reservation timeout.

In the simulation software, green dots are open spots, and black dots are vehicle agents that have parked. Yellow dots are parking spots that are reserved, and blue dots are guided agents that communicate with the smart parking system and receive reserved parking. The brown dots are explorer agents that do not communicate with the smart parking system and park on their own, and lastly red dots are guided agents that have their reserved spots taken by an explorer agent. In the simulation snippet in figure 30, vehicles were spawn at different entry points of the simulated parking space which took into account the multi-gate/multi-entry scenario of a parking structure. Moreover, the parking in the unassigned spot can be seen in Figure 30 by the red dotted vehicle agent due to its reserved spot being taken by an explorer agent.

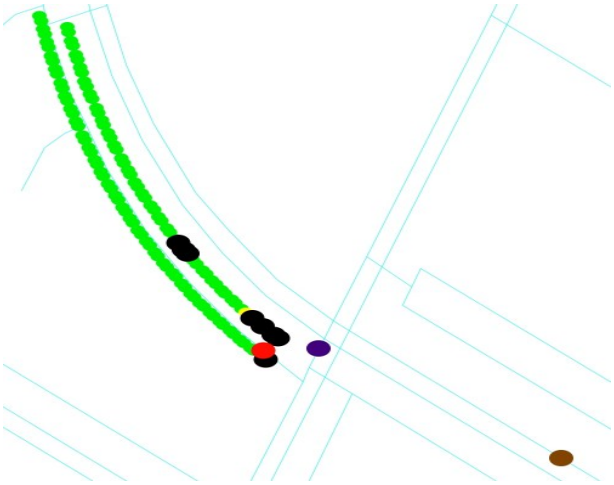

Figure 30: Multi-entry & Parking in unassigned spots

```
Explorer Agent 51 : simulation.common.globals.SlotData@75383096simulation.common.globals.Slot@4a990ef7 parking: 2, slot: 155,
4865034.7485 -8099.2155
Guided Agent 621 : simulation.common.globals.SlotData@75383096simulation.common.globals.Slot@4a990ef7 parking: 2, slot: 155, t
4865034.7485 -8099.2155
Guided Agent 773 : simulation.common.globals.SlotData@7cdbfe73simulation.common.globals.Slot@18a60b20 parking: 2, slot: 154, t
4865036.5477 -8101.8613
Guided Agent 818 : simulation.common.globals.SlotData@42b82b3bsimulation.common.globals.Slot@6aefc747 parking: 2, slot: 153, t
4865039.4052 -8105.6713
Guided Agent 981 : simulation.common.globals.SlotData@4caddec9simulation.common.globals.Slot@76a7bf25 parking: 2, slot: 151, t
4865043.9292 -8112.0333
Explorer Agent 260 : simulation.common.globals.SlotData@92e7d18simulation.common.globals.Slot@2b57a162 parking: 2, slot: 137,
4865081.0476 -8152.6711
Explorer Agent 292 : simulation.common.globals.SlotData@7cdbfe73simulation.common.globals.Slot@18a60b20 parking: 2, slot: 154,
4865036.5477 -8101.8613
Guided Agent 621 requests new slot
Guided Agent 621 : simulation.common.globals.SlotData@154f4086simulation.common.globals.Slot@2d5cebd0 parking: 2, slot: 150, t
4865045.8474 -8114.4146
Guided Agent 1433 : simulation.common.globals.SlotData@6fa40fesimulation.common.globals.Slot@4cb2399a parking: 2, slot: 149, t
4865048.9563 -8118.2511
Guided Agent 773 requests new slot
```

Figure 31: Simulation Software Console Output

If we look at the console output of the simulation software in Figure 31, we can see that explorer agent 51 wanted to park at slot 155, however at the same time guided agent 621 was assigned to slot 155 since it was not yet taken. Due to explorer agent 51 arriving at slot 155 before guided agent 621, when guided agent 621 arrives at its reserved spot location, it sees that its spot has been taken and it requests for a new slot and eventually parks at slot 150. The same case can be seen between explorer agent 292 and guided agent 773.

The last scenario is to show a parking reservation timeout system. We demonstrate this through our hardware-in-the-loop setup where the green lights are open parking spaces, red lights are occupied spaces, and blue lights are reserved spaces. It can be seen from Figure 16 and 17 that the reserved parking spot eventually clears to become an open parking spot after a set amount of time. This is accomplished with a simple timer that is set when a parking spot is reserved.

Figure 32: Parking Spot reserved (Blue)

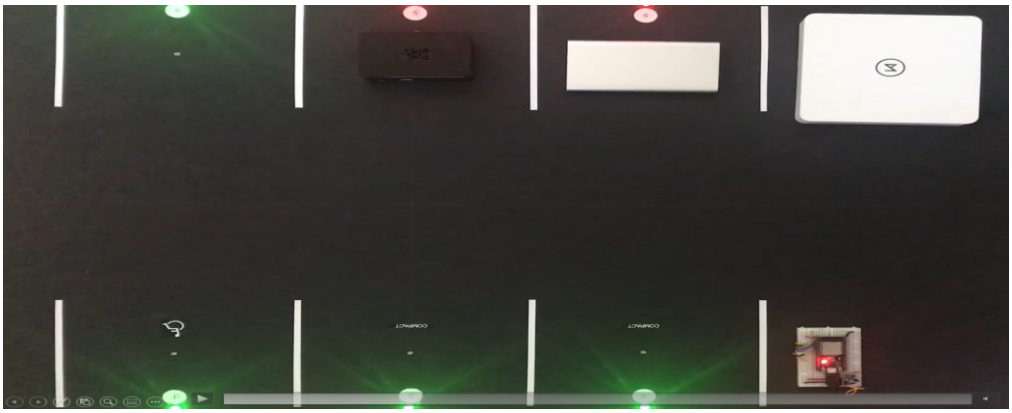

Figure 33: Reservation Timeout (Back to green)

4. Conclusion

The work throughout this project showcased hardware and software implementation of a smart parking system that is capable of communicating with both traditional and connected vehicles alike. This comes as a step towards evolving the existing parking infrastructures to accommodate the estimated wide-scale adoption of autonomous vehicles and connected vehicles, which would require real-time information about available parking services. In this regard, we developed a Parking Tracker Fog System (PTFS) that is capable of establishing communication through traditional wireless communication and DSRC, which is the expected communication technology for connected vehicles. We conducted several experiments through simulations and Hardware setup demos to assess the reliability of the parking information exchange in real-time for both traditional and DSRC-based communications. Finally, we foresee our work being scalable to multiple parking spaces, each represented through its own PTFS, and all PTFSs are managed through a central cloud that can alleviate the smart parking services to a city-wide level and complement it with parking occupancy predictions.

5. References

[1] [Online] https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf

[2] [Online] https://aws.amazon.com/freertos/

[3] [Online] https://aws.amazon.com/greengrass/

[4] [Online] https://docs.aws.amazon.com/freertos/latest/userguide/getting_started_espressif.html

[5] [Online] https://docs.aws.amazon.com/greengrass/latest/developerguide/setup-filter.rpi.html

[6] [Online] https://www.raspberrypi.org/products/raspberry-pi-3-model-b-plus/

[7] [Online] https://training.ti.com/ti-robotics-system-learning-kit

[8] [Online] https://docs.aws.amazon.com/freertos/latest/userguide/gg-demo.html

[9] [Online] https://github.com/GeoTecINIT/parking_occupancy_simulator

[10] [Online] http://www.wirelesscommunication.nl/reference/chaptr01/dtmmsyst/dsrc/dssr4.htm

[11] [Online] https://www.wime-project.net/

[12] [Online] Bastian Bloessl, Michele Segata, Christoph Sommer and Falko Dressler, "Performance Assessment of IEEE 802.11p with an Open Source SDR-based Prototype," IEEE Transactions on Mobile Computing, vol. 17 (5), pp. 1162-1175, May 2018.

[13] [Online] https://github.com/bastibl/gr-ieee802-11

[14] [Online] https://www-esv.nhtsa.dot.gov/Proceedings/24/files/24ESV-000379.PDF

[15] [Online] https://files.ettus.com/manual/page_usrp_b200.html

[16] [Online] https://kb.ettus.com/Building_and_Installing_the_USRP_Open-Source_Toolchain_(UHD_and_GNU_Radio)_on_Linux

[17] [Online] https://kb.ettus.com/Verifying_the_Operation_of_the_USRP_Using_UHD_and_GNU_Radio