

Intelligent Parking Guidance System  
Using Edge Computing, Computer Vision & Internet Of Things

By

Pranav Rajesh

A Thesis Presented in Partial Fulfillment of the Requirements for the Degree  
Master of Science

Approved October 2024  
by the Graduate Supervisory Committee:

Junfeng Zhao, Chair  
Yan Chen  
Abdel Ra'ouf Mayyas

ARIZONA STATE UNIVERSITY

December 2024

## ABSTRACT

With the increasing population, the surge in vehicle owners has made parking challenges a pressing issue. Navigating a crowded parking lot in urban centers, commercial areas, and public places is often time-consuming, leading to wasted fuel and increased greenhouse gas emissions. Another concern in parking lots is the occurrence of traffic accidents, frequently caused by distracted drivers searching for parking spaces or navigating through congested lots. Many parking lots worldwide have implemented smart parking systems that provide information about the number of available parking spaces at each instant but do not guide drivers about where these parking spaces are in the parking lot. This paper introduces an intelligent parking system that monitors the parking lot using an infrastructure monocular camera to identify the available parking spaces using computer vision and edge computing to process the data and avoid latency. The processed video from the camera is streamed on a website interface for remote monitoring or viewing by drivers accessing the parking lot. The website also provides the driver with a navigation route to the nearest available parking spot among all the vacant parking spaces from the driver's location at each instance. An experimental evaluation was performed to assess the robustness of the developed prototype system. Additionally, vacant parking space detection using multiple cameras was performed and evaluated in a simulation environment to upgrade the current system. This can be seamlessly integrated with the developed system to mitigate the problems of occlusions and false positives/negatives. This innovative, intelligent parking guidance solution significantly simplifies the parking process for human drivers, making it safer by reducing the risk of accidents and making it more convenient

and environmentally friendly. The system's convenience will make the audience feel more comfortable, knowing that finding a parking spot will be a hassle-free experience and that they are less likely to be involved in a parking lot accident.

## ACKNOWLEDGEMENTS

I sincerely thank everyone who helped me with this thesis and gave vital support. I sincerely thank Professor Junfeng Zhao, my adviser, for his excellent advice, tolerance, and knowledge. His mentoring greatly influenced my thesis, career path, and research interests. I appreciate that he allowed me to join the BELIV Lab in December 2022.

I express my deepest gratitude to the distinguished members of my committee, Dr. Yan Chen and Dr. Abdel Ra'ouf Mayyas, for their devoted time and careful reading of my thesis. Their perceptive criticism has improved the caliber of my work.

I appreciate Jingxong Meng, a PhD candidate in our lab, for his encouragement and support. He gave me insightful advice and assistance during my project.

I must thank my family, especially my parents, for their everlasting love and support. Their consistent encouragement and belief in my talents propelled me through this journey.

Throughout this thesis, every single one of the people listed above has had an enormous influence on my education and personal development. I will always be appreciative of this.

## TABLE OF CONTENTS

	Page
LIST OF TABLES .....	viii
LIST OF FIGURES .....	viii
SINGLE INFRASTRUCTURE CAMERA-BASED INTELLIGENT PARKING	
GUIDANCE SYSTEM.....	1
Introduction.....	1
Problem statement.....	1
Thesis Project Objective: .....	2
Thesis Summary: .....	2
LITERATURE REVIEW .....	4
Intelligent Parking Guidance System Design & Architecture .....	9
Hardware Components.....	11
GoPro Hero 10 .....	11
Nvidia Jetson AGX Orin (Edge Computer) .....	11
Gimbal.....	12
Cradle point R1900 Router .....	13
Vacant Parking Space Detection .....	14
Parking Space definition using Bresenham's Algorithm .....	15

	Page
Parking Space Detection .....	18
Vacant Parking Space Localization.....	21
Navigation Route Guidance Generation .....	27
Website Interface & Network Communication.....	30
Centralized Database .....	35
RESULTS & DISCUSSION – SINGLE INFRASTRUCTURE CAMERA-BASED INTELLIGENT PARKING GUIDANCE SYSTEM.....	38
MULTIPLE INFRASTRUCTURE CAMERAS – BASED INTELLIGENT PARKING GUIDANCE SYSTEM.....	42
Introduction.....	42
Problem Statement .....	42
Objective .....	42
Simulation Environment Setup .....	44
Multiple Cameras-based Parking Space Detection.....	52
Methodology for fusing data from multiple infrastructure cameras.....	59
Mathematical Formulae .....	60
RESULTS & DISCUSSION – MULTIPLE INFRASTRUCTURE CAMERAS – BASED INTELLIGENT PARKING GUIDANCE SYSTEM.....	63
Occlusion .....	63

	Page
False Positives/Negatives.....	65
CONCLUSION & FUTURE WORK.....	70
REFERENCES .....	72
APPENDIX	
A PARKING SPACES LOCATION .....	76
B GITHUB REPOSITORY & FEEDBACK FORM.....	80

## LIST OF TABLES

Table	Page
Table I: Association Table containing Parking Spots Information.....	24
Table II: Comparison of different Yolo Models .....	41
Table III: Multiple Infrastructure Cameras Spawn Locations .....	49
Table IV: List of parking spots and their corresponding location coordinates .....	77



## LIST OF FIGURES

Figures	Page
1. System Architecture .....	10
2. Go Pro Hero 10 Black.....	11
3. Nvidia Jetson Nano Orin.....	12
4. Gimbal for Camera Stabilization .....	13
5. Cradle Point Router.....	14
6. Snapshot of Simulator Building Parking lot .....	16
7. Conda Environment Activation.....	17
8. Executing Parking Space Definition command .....	17
9. Defined Parking Spaces .....	17
10. IoU-Based occupancy status calculation .....	20
11. Executing Vacant Parking Space Detection Algorithm.....	20
12. Depiction of Vacant Parking Space Detection .....	21
13. Zoomed Google Maps View of the Parking lot sector.....	23
14. Collecting Location Coordinates of the parking space .....	23
15. Naming Layer of the Parking Lot .....	24
16. Executing Localization Algorithm.....	27
17. Generated Custom Map .....	29
18. Executing Navigation Route Generation Algorithm.....	29
19. Generated Navigation Route Guidance .....	30

20. Video Upload using Gstreamer .....	32
21. Data communication in the backend.....	33
22. Website Interface Architecture .....	33
23. WEBUI. Find parking spot page.....	34
24. WEBUI. Navigation Page .....	35
25. MongoDB database structure.....	37
26. Hardware Setup.....	38
27. Testing & Validation of the prototype system.....	39
28. Conda Environment Activation.....	45
29. Carla Environment Initialization.....	45
30. Carla Simulator .....	46
31. Initializing Town05 Map.....	47
32. Spawning and Manually Controlling a vehicle.....	48
33. Collecting Infrastructure Camera Spawn Location Coordinates .....	49
34. Carla Scenario to spawn vehicles in the parking lot .....	51
35. Parking lot Environment Setup with Spawned Vehicles.....	51
36. Collecting live feed from the Infrastructure Cameras.....	53
37. Vehicle Detection Algorithm Initialization .....	53
38. Detected Vehicles by Camera 1 .....	54
39. Detected Vehicles by Camera 2 .....	54
40. Detected Vehicles by Camera 3 .....	55

41. Detected Vehicles by Camera 4 .....	55
42. Vacant Parking Space Detection Algorithm Initialization .....	56
43. Initial Parking Space Occupancy Status from Camera 1 live feed .....	57
44. Initial Parking Space Occupancy Status from Camera 2 live feed .....	57
45. Initial Parking Space Occupancy Status from Camera 3 live feed .....	58
46. Initial Parking Space Occupancy Status from Camera 4 live feed .....	58
47. Corrected Final Occupancy Status of P2 in Camera 4 feed .....	67
48. Corrected Final Occupancy Status of P9 in Camera 1 feed .....	67
49. Corrected Final Occupancy Status of P21 in Camera 2 feed .....	68

# SINGLE INFRASTRUCTURE CAMERA-BASED INTELLIGENT PARKING GUIDANCE SYSTEM

## Introduction

### Problem statement

Urban areas' constant growth and the number of cars accompanying them have created various difficulties, including parking, which is becoming a more pressing problem. The demand for parking in public locations is overwhelming, causing traffic jams inside the parking lots. Vehicle owners often find it extremely difficult to find a vacant parking spot. To solve this problem, technology was used in the past to identify if there are any vacant parking spaces in a particular parking lot. Further, an electronic board would display the number of available parking spaces in that lot at each instant. However, in this case, the technological system only tells the user if there are any available spots to park their vehicle. Still, it doesn't precisely notify the user about the whereabouts of these open spaces. In this scenario, the user is motivated to enter a parking lot to park his vehicle, hoping to find a parking lot, but would still have to search for the vacant space by usually circling the parking lot. First, it leads to elevated fuel consumption of the vehicles circling in the parking lot, leading to increased greenhouse gas emissions and environmental pollution. In these crowded lots, while searching for parking spaces, the users often get inattentive to their surroundings and end up in a traffic accident. A creative parking solution is required to make parking more accessible for those who drive and safer, more convenient, and ecologically friendly. By reducing the time spent in parking lots and the associated fuel consumption, the

intelligent parking system significantly reduces greenhouse gas emissions, contributing to a cleaner environment.

#### Thesis Project Objective:

The main objective of this research project is to make parking more accessible to the public by developing an innovative, intelligent parking solution that uses technological tools to not only notify the user about the number of available vacant parking spots but also identify the exact location of these vacant spots, provide them to the user, and further aid the user by providing them with navigation route guidance to these spots. This would help the users by reducing their waste of time and fuel and would help them avoid traffic accidents.

#### Thesis Summary:

This thesis initially presents the prototype of a novel intelligent parking system that utilizes edge computing and computer vision with a single monocular camera infrastructure to monitor the parking lot and identify the vacant parking spots. This system additionally identifies the location coordinates of the vacant parking spaces. This system guarantees real-time operation, essential for efficient navigation and space management, by reducing latency through on-site data processing using edge computing. The system's primary function is to send processed video data to a driver-accessible online interface. Parking lot statuses can be remotely monitored thanks to this configuration. It incorporates a dynamic navigation system that uses the driver's current location in the parking lot to direct them to the closest parking spot. By cutting down on the amount of time spent looking for parking, raising the possibility of accidents, and lessening the effect that vehicle emissions have on the environment, such a system has the potential to transform the parking experience

completely. This thesis also explores using multiple infrastructure cameras for the same solution. It presents simulation-based results for parking lot monitoring using various cameras, which can be easily incorporated into the above-developed intelligent parking system to monitor parking lots with larger parking spaces and provide navigation guidance.

The proposed solution thus addresses both the operational inefficiencies in contemporary parking lots and the environmental and safety concerns exacerbated by current parking practices. This system significantly enhances the parking experience, aligning with broader environmental and safety goals by offering a seamless, user-friendly interface that provides real-time updates and navigational assistance.

## LITERATURE REVIEW

In [1], a cloud-based Internet of Things-based smart parking reservation system is presented. The suggested method uses ultrasonic sensors to detect the presence of the cars and determine whether a parking space is available. The Arduino Uno board receives the data collected from the sensor. The data collected is sent to the Relational Database Service (RDS) of Amazon Web Services (AWS) via the Ethernet card linked to the Arduino Uno. The driver receives a notification through the mobile app on the availability of a free slot, and the parking slot's LED lights reflect this information. To offer reservation services as part of a user-targeted service and broadcast real-time parking information to vehicles, [2] built and implemented a working model of a reservation-based smart parking system (RSPS). Using cutting-edge detecting and mobile communication technologies, RSPS analyzes time-stamped sensing data from the parking lot's sensor network and disseminates information about available parking. Through Wi-Fi or the Internet, drivers can get parking information and book the desired open places. [3] addresses the pressing parking problem by putting forth a novel concept known as ROSAP—reservation-based multi-objective smart parking. Using a simulated annealing-based meta-heuristic, this method optimizes the parking space assignment issue, expressed as a multi-objective integer linear program (ILP). Within the boundaries of their designated areas of interest and with the assistance of ROSAP, vehicles can locate the best parking space. [4] suggests a novel, optimally navigated, reservation-based method for locating open parking spaces in smart cities. The proposed approach aims to determine the shortest path to the open parking space. To achieve this, the issue is framed as an optimization problem that locates the closest open

parking spot using a genetic algorithm. The driver's location is obtained via the GPS on the smartphone, and a genetic algorithm locates the nearest open parking space. [5] explains a technique that finds any empty place outside the range of human eyesight. The imagery of the parking area is obtained using a camera. The edges of each parking place are then carefully extracted using clever edge recognition and LUV-based color variation detection techniques. The model used to categorize parking spaces as occupied or unoccupied is created by training the retrieved characteristics with a Random Forest classifier. [6] explains a novel Smart Parking System (SPS) that has been suggested to help vehicles locate open spots in a parking lot more quickly. The proposed system uses ultrasonic (ultrasound) sensors to identify improper parking activities and car park occupancy. [7] suggests developing an Internet of Things-based vehicle parking management system showing open or available parking spaces. It incorporates an LCD screen, an IR sensor, and a nodeMCU microcontroller. When an automobile reaches a parking space, the IR sensor determines whether it is there. The LCD screen then shows the driver the available parking space. The data on the LCD panel is updated regularly, and the parking spaces are constantly being observed. The Internet of Things (IoT) technology is being used to construct the suggested parking system as cheaply, effectively, and efficiently as possible. In [8], this study presents the eco-parking system driven by IoT to lower emissions, traffic congestion, and inefficiencies associated with parking in urban surroundings. The Eco-Parking system combines RFID technology with mobile applications to authenticate vehicles and assign parking spaces based on real-time availability and the vehicle's emission class. The principal aim of the system is to augment the sustainability of urban



transportation by reducing carbon footprints and maximizing the available parking spots. IoT sensors keep an eye on the parking lot all the time, giving data on how each spot is doing in real-time. By giving lower-emission automobiles priority parking, this strategy shortens commute times and promotes environmentally friendly vehicles' usage. The study also explores how conventional parking arrangements affect the environment and how this method provides a more sustainable option. Technology enhances consumers' entire parking experience while supporting global environmental goals by using smart city concepts. [9] suggests Eco Park as an affordable and environmentally friendly way to manage parking spots in cities. By providing real-time data on parking availability, the system makes use of Internet of Things technology to lessen traffic congestion and improve customer experience when parking. It provides an effective and clever parking management system by utilizing sensors, gateways, cloud-based platforms, and mobile applications. The authors draw attention to the growing difficulties in controlling parking in urban environments, particularly considering the expanding size of luxury cars and the dearth of parking spots. The system employs a cloud-based platform to process and store data, allowing for scalability and connection with other smart city efforts. It also gives customers real-time information on available parking places. The study highlights how IoT may improve conventional parking systems to become more environmentally friendly, intelligent, and energy-efficient solutions that support sustainable urban transportation. To address parking issues in metropolitan areas, this paper [10] addresses the construction of a smart parking system that makes use of edge computing, camera networks, and deep learning. The system's cost-effective design eliminates the need for pricey sensors in each

parking space by using a network of cameras to monitor broad regions. The system uses wide-angle cameras to detect parking space occupancy and zoom lens cameras to record license plate data. To provide real-time inference of parking spot availability, the study presents a specially designed neural network tailored for edge devices, such as Raspberry Pi and Nvidia Jetson Tx2. Tiny YOLO is an additional enhancement to the technology that allows for tracking and vehicle detection. A cloud server receives data from the cameras over a WiFi mesh, processes and stores the parking status and vehicle tracking information there. The use of LIDAR in interior parking lots, where camera coverage may be limited, is also explored in this research. In [11], This study addresses the rising issue of poorly managed parking spaces in urban environments, both off-street and on-street, by presenting a camera-based smart parking solution. The suggested solution provides real-time information about open parking spaces using a Raspberry Pi and Pi cameras. Using real-time video feeds, a deep learning model based on YOLO v2 is utilized to identify cars and provide real-time parking availability updates. Every five seconds, the system updates the parking slot status in a Firebase database by processing camera data locally on the Raspberry Pi. Through an Android application that shows the number of parking spaces that are accessible in real-time, users can get this information. A form of the system that can reliably identify cars and update parking statuses without requiring a manual refresh is lightweight and efficient. The system's scalability, possibilities for both stilt and open parking lots, and upcoming enhancements like Google Maps integration are all covered in the paper. In [12], This work focuses on integrating license plate recognition and real-time slot computation with Tiny Machine Learning (TinyML) into smart parking systems to

maximize parking space use. The system uses edge computing and IoT cameras and is geared for situations with limited resources. It uses TinyML models to reduce dependency on cloud infrastructure by performing dynamic parking space calculations and license plate recognition directly on edge devices. The construction of two major datasets—one for Arabic license plates from Tunisia and the other for parking spot occupancy—is presented in this study. These datasets are used to train the TinyML models, which guarantee great accuracy in identifying license plates and open slots. The system makes use of a network of smart cameras that are linked to Raspberry Pi nodes locally so that the video feeds may be processed instantly. The solution is appropriate for both indoor and outdoor parking spaces since it fosters energy efficiency, scalability, and privacy. The paper's conclusion discusses potential improvements and emphasizes how this approach can be applied globally.

All the above works have explored intelligent parking solutions, primarily focusing on detecting and identifying vacant parking spaces and providing that information to the user. In all these cases, the user would still have to circle in the parking lot to find those parking spaces. Therefore, a parking system equipped with navigation is indispensable for optimizing the parking experience. Firstly, it enhances efficiency by swiftly guiding drivers to available parking spots, reducing congestion and idle time. This saves drivers time and contributes to alleviating traffic congestion in urban areas. Secondly, it enhances user experience by providing real-time information about parking availability, helping to minimize frustration and stress associated with finding parking. Thirdly, it promotes sustainability by reducing fuel consumption and emissions from circling vehicles.

Integrating navigation into a parking system streamlines parking operations and enhances urban mobility and environmental sustainability.

### Intelligent Parking Guidance System Design & Architecture

This section provides a comprehensive overview of the proposed system design and architecture. The intelligent parking guidance system is meticulously designed to efficiently guide the human driver to the nearest vacant parking space in a parking lot, simplifying the parking process and enhancing safety and convenience. Figure 1 depicts the system architecture comprising four key modules: Detection, Localization, Navigation, and Website Interface. Each module is pivotal in the system's functionality, ensuring an efficient and user-friendly parking experience. In real-time, the parking space detection algorithm is applied to the live camera feed frame to detect all the parking lots' unoccupied or vacant spaces. The information regarding the parking spaces, i.e., name and occupancy status, is sent to the centralized database of the intelligent parking system. The localization module then retrieves the vacant spaces list from the centralized database. It manually searches each available parking space's global coordinates (Latitude, Longitude) in Google Maps. Furthermore, the localization module identifies the distance between each vacant parking space and the user's location/entry point of the parking lot. These distances are sorted to find the shortest distance, which enables the system to identify the nearest vacant parking space from the user's location as well as the entry point of the parking lot. A list of all vacant parking spaces with their corresponding location, along with the identified nearest parking space and its location, is sent to a centralized database. The website interface module acquires this information from the database and displays it on the website,

which will be available to the users accessing the website. The user entering the parking lot could choose the suggested parking space solution (nearest vacant parking space) or any available parking space from the list. After the user selects the parking space, the Navigation module collects the location data of the chosen parking spaces from the backend of the website interface. It develops the navigation route to the selected parking space from the user's location/entry point of the parking lot. This navigation route is further displayed in the website interface and made available for the user to follow the navigation guidance and park the vehicle.

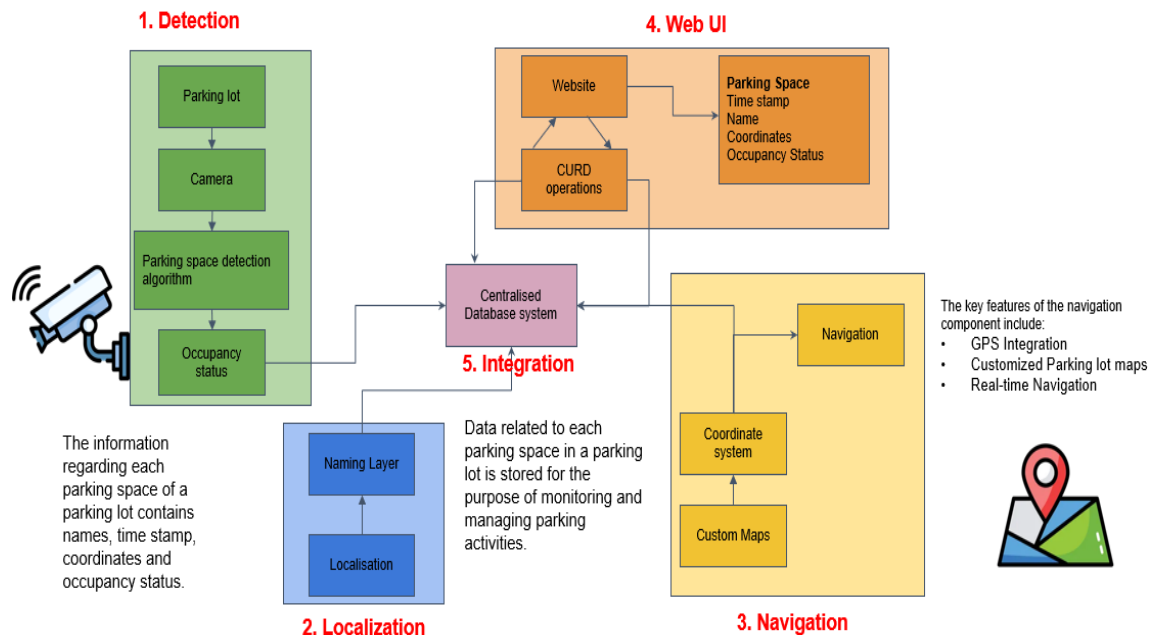


Figure 1: System Architecture

## Hardware Components

### GoPro Hero 10

A versatile action camera with a reputation for excellent performance and a sturdy build, the GoPro Hero 10 Black, as shown in Figure 2, is perfect for outdoor location monitoring. Its powerful GP2 CPU can record 5.3K videos at 60 frames per second and take 23MP pictures, producing clear images. Because of the camera's HyperSmooth 4.0 stabilization, fluid video is produced even in dynamic environments. The Hero 10 Black is an excellent option for high-quality video recording and flexible outdoor use because of its front-facing screen and enhanced low-light performance. These key desired attributes were considered while choosing the appropriate outdoor parking lot monitoring camera.



Figure 2: Go Pro Hero 10 Black

### Nvidia Jetson AGX Orin (Edge Computer)

For robotics, Artificial Intelligence (AI) applications, and autonomous systems, NVIDIA has created the Jetson Orin Nano, as depicted in Figure 3, a potent edge AI computing platform. It can be used for computer vision applications and deploying sophisticated AI

models because it has a small and energy-efficient design and can achieve up to 40 TOPS of AI performance. It enables deep learning, computer vision, and high-performance computing applications thanks to its 8-core ARM Cortex-A78AE CPU and 512-core NVIDIA Ampere GPU. Projects including autonomous navigation, object detection, and intelligent automation benefit greatly from the Jetson Orin Nano's ability to interpret real-time data from sensors and its suitability for AI inference at the edge.



Figure 3: Nvidia Jetson Nano Orin

## Gimbal

A gimbal, as portrayed in *Figure 4* Gimbals are stabilization devices designed to keep a camera steady during movement, ensuring smooth and professional-quality footage. They utilize motors and sensors to counteract unwanted camera shake, allowing for seamless panning, tilting, and rotation. Gimbals are particularly useful for filming in dynamic environments, providing smooth motion even when the operator is in motion. They are compatible with various camera types, including GoPros. With features like multiple

shooting modes, joystick controls, and customizable settings, a gimbal is essential for high-quality, stabilized video output.



Figure 4: Gimbal for Camera Stabilization

#### Cradle point R1900 Router

A dependable 5G and LTE mobile networking solution, the Cradle Point R1900 Router as shown in Figure 5 is made for in-car and Internet of Things applications. It is perfect for field services, public safety, and transportation because it is made to withstand challenging environments. With its potent quad-core processor, sophisticated Wi-Fi 6 features, and all-encompassing security, the R1900 guarantees secure connections and smooth data transfer. It may be remotely configured and monitored using Cradlepoint's NetCloud platform, and it supports GPS for real-time location tracking. The R1900's durable construction and flexible connectivity options guarantee constant, fast network access even in the most demanding settings.





Figure 5: Cradle Point Router

### Vacant Parking Space Detection

The intelligent parking system's input source is a GoPro Hero 10, which captures video frames for the parking space detection module. The module's two main functions are detecting available parking spaces and updating each space's occupancy status in the real-time centralized database. It then produces an output that visualizes the occupancy status of the parking spaces, indicating whether they are empty or occupied.

The first primary function, detecting available parking spaces, can be achieved in various ways. The effectiveness of this function largely depends on the position of the cameras, the identification of the region of interest (specific parking spots), and the continuous monitoring of these spots. As mentioned in [13], the initial goal is strategically positioning cameras to achieve complete visibility of all parking spaces. This involves identifying the region of interest, i.e., the specific parking spots, and continuously monitoring them.

Once the camera positions are finalized, various image-processing techniques [14], such as edge detection [15] combined with the Hough Transform [16], are used to extract the coordinates of the parking spaces and define the regions of interest for monitoring. For ongoing monitoring, computer vision algorithms powered by machine learning [17] or deep learning [18] can process images or video feeds from the cameras. These algorithms

can detect vehicles, identify empty spaces, and even classify different types of cars, showcasing the system's advanced capabilities. The identification process may involve training the system with a custom dataset developed internally or provided by the company to detect and accurately classify free and occupied parking spots.

Our system is designed to be adaptable, employing a combination of computer vision, machine learning, and image processing techniques. The primary function of detecting available parking spaces is defining and detecting parking spots. This adaptability allows the system to be fine-tuned and updated according to the parking space's changing requirements and conditions.

#### Parking Space definition using Bresenham's Algorithm

This system is designed to be adaptable, employing a combination of computer vision, machine learning, and image processing techniques. The primary step of detecting available parking spaces is defining and detecting parking spots. This adaptability allows the system to be fine-tuned and updated according to the parking space's changing requirements and conditions.

This component outlines the procedure for defining parking spots using Bresenham's Algorithm [19]. This line-drawing algorithm determines which points on a grid-based display must be plotted to form a straight line between them.

Initially, a single snapshot of the parking area is captured, encompassing all spots, which is the input for the parking space definition algorithm. This prototype system was built for a parking lot sector opposite the Simulator Building, Arizona State University, Polytechnic

Campus. Figure 6 shows the sector of the parking lot for which the prototype system is built.



Figure 6: Snapshot of Simulator Building Parking lot

Then, this snapshot is inputted to the algorithm to highlight the regions of interest within this frame manually. The regions of interest are the 20 parking spaces in focus. The following commands are run in the Windows command prompt shell to execute the parking space definition algorithm.

Step 1: Open the command prompt shell and activate the Conda environment, as shown in Figure 7. This environment contains all the necessary packages to execute the code.

```
conda activate psdetect
```

```
C:\Users\prana>conda activate psdetect  
(psdetect) C:\Users\prana>|
```

Figure 7: Conda Environment Activation

Step 2: Run the “setregions.py” Python code, as shown in Figure 8, to run the parking space definition algorithm in the same terminal.

```
python3 setregions.py
```

```
(psdetect) C:\Users\prana>python3 setregions.py
```

Figure 8: Executing Parking Space Definition command

This command allows you to define the parking spaces (regions of interest) manually, highlighted as purple boxes, as shown in Figure 9. This command also stores the x and y pixel coordinates of all four corners of the defined parking spaces.

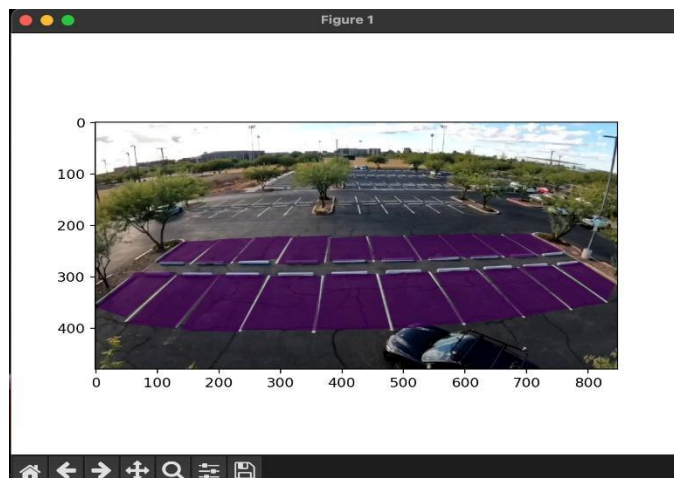


Figure 9: Defined Parking Spaces

## Parking Space Detection

A car-driven method was used to detect parking spots where parking spot occupancy is determined based on the detected vehicle coordinates. This scalable and efficient approach minimizes the need for extensive labeling and training at each facility by marking parking spaces and boundaries once. Focusing on car detection through various available algorithms simplifies debugging with encapsulated open-source code and mitigates concerns related to space complexity. This component operates in two phases. The first involves obtaining the coordinates of the parking spaces and the vehicles. The second phase calculates the Intersection over the Union (IoU) value using these coordinates. The coordinates of the parking spaces are already obtained during the process of parking space definition. Vehicle detection needs to be performed to identify the coordinates of the vehicles in the parking lot.

In the first phase, we use an open-source, custom-built object detection model based on the Yolov5x6 architecture to perform vehicle detection. YOLO [20] is part of the You Only Look Once (YOLO) family of computer vision models, widely used for object detection and, in our case, for detecting vehicles. YOLOv5, which is trained on the COCO dataset, comes in four versions—small (s), medium (m), large (l), and extra-large (x)—with each offering progressively higher accuracy. The YOLO algorithm works by dividing an image into a grid system, where each grid is responsible for detecting objects within its area. This architecture is trained explicitly for car detection and achieves a 94% accuracy rate in identifying cars of various colors, shapes, and sizes, as observed by the camera used for detection. The vehicle detection algorithm forms bounding boxes over the detected

vehicles and calculates and stores the image pixel coordinates of the centroid of the bounding box of the car.

The vacant parking space detection algorithm now takes the coordinates of the parking spaces and detected vehicles as input. Internally, the parking space detection algorithm contains an Intersection over Union (IoU) algorithm to which these coordinates are fed.

The IoU Ray Casting Algorithm [21] uses a polygon-point method to calculate the IoU value in the second phase. This algorithm calculates the Intersection over Union (IoU) between two geometric shapes, often for object detection in computer vision tasks. The IoU metric measures the overlap between two shapes, such as bounding boxes. Ray Casting is usually used to calculate this overlap between two polygon shapes. This technique involves shooting rays from a point (typically from the vertices or centroids of the shapes) and counting how often the ray intersects with the edges. This helps the algorithm determine the overlapping region between two shapes. The points inside the overlapping regions are used to calculate the intersection area. The union area is the combined area of both shapes minus the intersection. The IoU between the two overlapping shapes is computed by dividing the Intersection Area by the Union Area as shown in (1).

$$\text{Intersection over Union (IoU)} = (\text{Area of Intersection}) / (\text{Area of Union}) \quad (1)$$

Whether the centroid of the detected vehicle lies inside, outside, or on the polygon defined by the parking space coordinates is determined using the Ray Casting technique. Based on the vehicle's centroid's position inside a specific defined parking spot, the IoU value is

computed. Suppose the IoU value is greater than or equal to 50 %, then the occupancy status of that specific parking lot is identified as occupied. Otherwise, it is recognized as a vacant parking space. A depiction of the IoU-based occupancy status calculation is given in Figure 10.

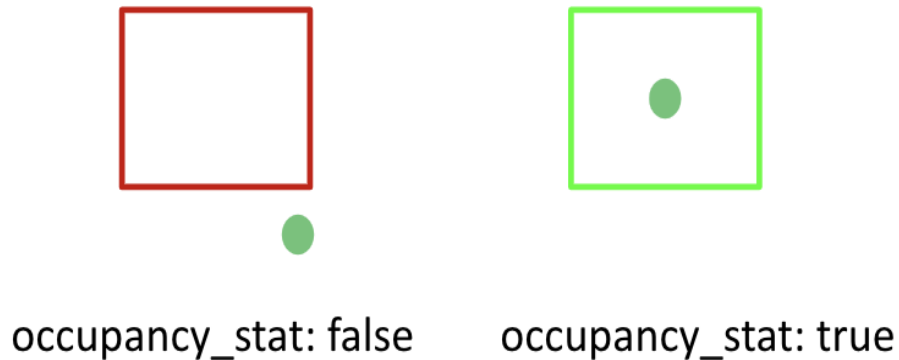


Figure 10: IoU-Based occupancy status calculation

The steps to run the vacant parking space detection algorithm are given below:

Step 1: Open a new command prompt shell and activate the conda environment using the same command used during the parking space definition algorithm.

Step 2: Once the conda environment is activated, run the `parking_space_detection.py` file as shown in Figure 11.

```
(psdetect) C:\Users\prana>python3 parking_space_detection.py
```

Figure 11: Executing Vacant Parking Space Detection Algorithm

Once the algorithm runs, the live feed from the camera is processed using the discussed computer vision techniques. It displays the live results of the vacant parking space detection, as depicted in Figure 12. The green dots over the vehicles symbolize the vehicle detection process and the detected vehicle's centroids. The defined parking space region turns green when the parking space's occupancy status is vacant, and it turns red when the occupancy status is occupied, as shown in Figure 12.

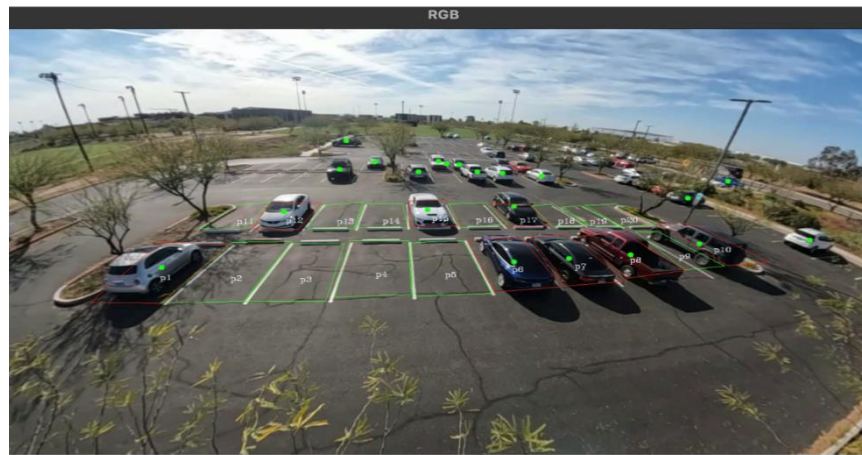


Figure 12: Depiction of Vacant Parking Space Detection

### Vacant Parking Space Localization

Once the vacant parking spaces are detected, identifying their location coordinates is a critical step, as these coordinates are the input for the navigation module to develop a navigation route to the selected vacant parking spaces.

The location coordinates can be local (relative to the camera) or global (concerning a fixed reference point, like GPS coordinates). Regarding the local coordinates, parking spaces are mapped to coordinates relative to the camera's position and orientation. This approach



is functional when you want to know where parking spaces are for the camera, which can be beneficial for navigation within the camera's view. Mapping parking spaces to global coordinates means referencing them to a fixed global reference point, like latitude and longitude. This approach is practical when integrating parking information with other location-based services or maps and was therefore used as the system leveraged Google Maps API services.

The steps to gather the location coordinates of all the desired parking spaces in the parking lot is given below:

Step 1: Open Google Maps and enter the location address of the Simulator Building, ASU Polytechnic Campus, in the search toolbar. Once the simulator building opens, zoom into the parking lot sector for which the prototype system is built, as shown in Figure 13.

Step 2: Use the mouse to manually click on each parking space, note the global location coordinates, as shown in Figure 14, and store them.

Step 3: Similarly, collect the location coordinates of the entry point of the parking lot.

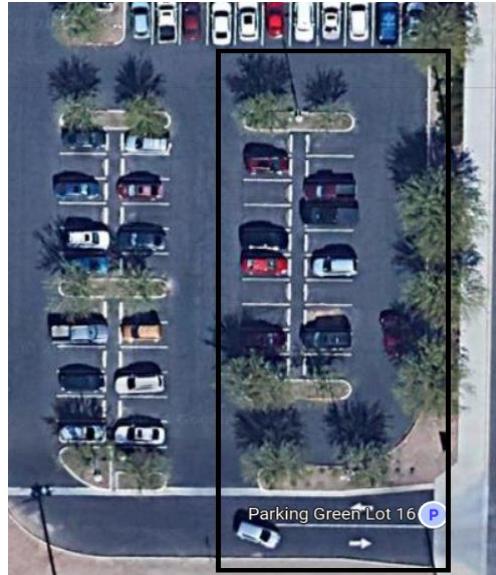


Figure 13: Zoomed Google Maps View of the Parking lot sector

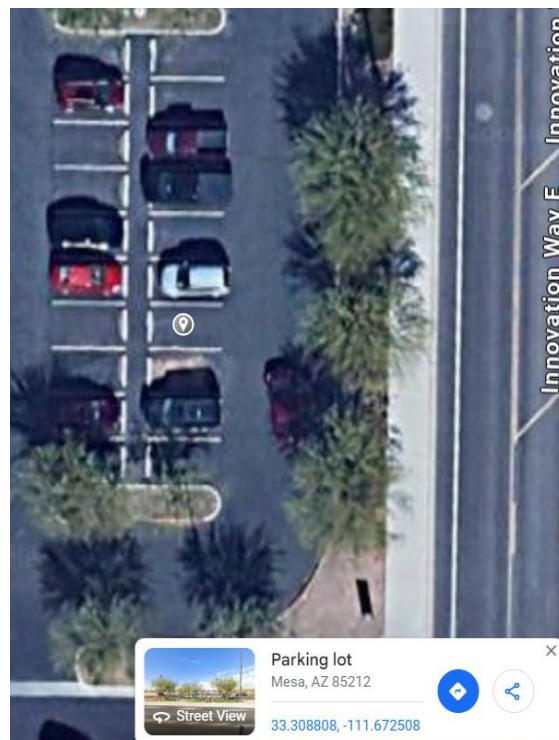


Figure 14: Collecting Location Coordinates of the parking space

A naming layer is created for the chosen parking lot to label each parking space. This prototype system was tested on a parking lot containing 20 spaces. Therefore, the parking spaces were labeled P1, P2, P3, and so on until P20. The global coordinates obtained using Google Maps were mapped to their corresponding parking space labels. Figure 15 portrays the naming layer of the chosen parking lot. Table I contains each parking space, its corresponding location coordinates, and its respective distances from the parking lot's entry point.



Figure 15: Naming Layer of the Parking Lot

Table I: Association Table containing Parking Spots Information

Parking Spot	Location (GPS Coordinates)	Distance from the entry point
P1	(33.308732, -111.672509)	170 ft
P2	(33.308761, -111.672508)	180 ft

P3	(33.308786, -111.672509)	190 ft
P4	(33.308811, -111.672508)	200 ft
P5	(33.308835, -111.672510)	210 ft
P6	(33.308858, -111.672511)	220 ft
P7	(33.308883, -111.672510)	230 ft
P8	(33.308907, -111.672507)	240 ft
P9	(33.308933, -111.672507)	250 ft
P10	(33.308736, -111.672507)	260 ft
P11	(33.308761, -111.672577)	120 ft
P12	(33.308787, -111.672574)	130 ft
P13	(33.308811, -111.672577)	140 ft
P14	(33.308837, -111.672578)	150 ft
P15	(33.308860, -111.672572)	160 ft
P16	(33.308885, -111.672574)	170 ft
P17	(33.308909, -111.672577)	180 ft
P18	(33.308909, -111.672578)	190 ft
P19	(33.308933, -111.672574)	197 ft
P20	(33.308960, -111.672575)	210 ft

In the case of parking space detection, the algorithm updates the database regarding the occupancy status of each parking space in real time. The localization module obtains the occupancy status from the database. The user's location (GPS coordinates) is also acquired.

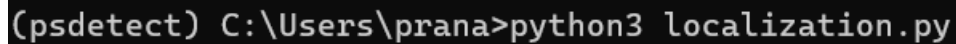
The localization algorithm identifies the distance between the user's location and each vacant parking space's global coordinates using the Distance Matrix API services from Google Maps. Many platforms, like Google Maps, offer a function called the Distance Matrix API that computes the time and distance of travel between multiple origins and destinations. This API is frequently used to calculate the distances between different places. To access the distance matrix API and get distances between the user and the available parking as input into the localization algorithm, a Google Maps API key needs to be created in the Google Cloud console website. Once the API key is created, the Google Maps distance matrix API can be easily integrated with the localization code. Now, the localization code will contain the location coordinates of all the available parking spaces and the user's entry point location. The distances between the user/entry point and the vacant parking spaces are computed and stored with the help of the API.

These distances are sorted from the shortest to the largest using a sorting algorithm inside the localization module. The output of the sorting algorithm, i.e., the shortest distance, associates the parking space with the nearest vacant parking space to the user accessing the parking lot or the entry point of the parking lot, forming the optimal parking space solution. The user accessing the parking lot can choose any parking space from the list of vacant parking spaces or choose the optimal parking space solution recommended by the system. While navigating to a parking space, either his choice or the recommended solution, if the user fails to follow the navigation route guidance and ends up entering a different aisle in the parking lot, the system will still reroute the user to a new nearest parking spot based on his new current location. The steps to run the localization algorithm are given below:

Step 1: Open a new command prompt shell and activate the conda environment using the same command used during the parking space definition algorithm.

Step 2: Once the conda environment is activated, run the localization.py file as shown in Figure 16.

Command to run: python3 localization.py



```
(psdetect) C:\Users\prana>python3 localization.py
```

Figure 16: Executing Localization Algorithm

### Navigation Route Guidance Generation

The most crucial step in creating a custom map is the utilization of Google Maps API, a robust toolset offering comprehensive functionalities for crafting interactive maps and integrating navigation features. The initial step in this endeavor is securing an API key from Google, serving as the authentication mechanism for accessing Google's mapping services. After the API key is obtained, the map is initialized on a web page using JavaScript, finely tuning parameters such as the map's center coordinates and zoom level to cater to your specific requirements [22]. The initialization process involves several technical steps. The HTML container, typically a <div> element, acts as the canvas for the map. The Google Maps API is loaded using a script tag with the API key. Following this, a new map object is instantiated with specified options such as 'center,' 'zoom,' and 'mapTypeId.' The 'center' option requires precise latitude and longitude coordinates, while 'zoom' defines the map's initial zoom level. This step ensures the map loads correctly and

aligns with user requirements. Overlaying the parking lot layout onto the map is a pivotal milestone in customization, providing users with a vivid representation of available parking spaces. This intricate task entails precisely defining the coordinates of parking spaces and marking them with bespoke markers or polygons. We can be customized with different icons and labels, enhancing visual clarity and user interaction [23]. Polygons are defined using a series of latitude and longitude coordinates to outline parking areas, driveways, entrances, exits, and pedestrian pathways. This detailed overlay contributes to a holistic user experience by integrating critical spatial information into the map. Figure 17 depicts the generated custom map of the parking lot for which the prototype system is built.

Generating a real-time route map to the parking lot involves obtaining the starting point and destination—the user's current location coordinates and the nearest available parking spot. The user's location information is acquired using the Google Maps Geolocation API, which leverages the device's GPS to fetch accurate coordinates. The data on available parking spots can be gathered from the interface, possibly stored in a database, and retrieved via API calls [24]. Leveraging the Google Maps Directions Service, the system calculates the shortest route from the user's current location to the chosen parking spot. This service provides detailed directions. This encompasses visual and textual navigation instructions, ensuring users receive step-by-step guidance [25]. The real-time navigation functionality is enhanced by integrating dynamic route adjustments based on traffic conditions and user feedback, optimizing the overall parking experience and mitigating congestion [23]. Steps to run the navigation route guidance algorithm is given below:

Step 1: Open a new command prompt shell and activate the conda environment using the same command used during the parking space definition algorithm.

Step 2: Once the conda environment is activated, run the navigation.py file as shown in Figure 18.

Command to run: `python3 navigation.py`

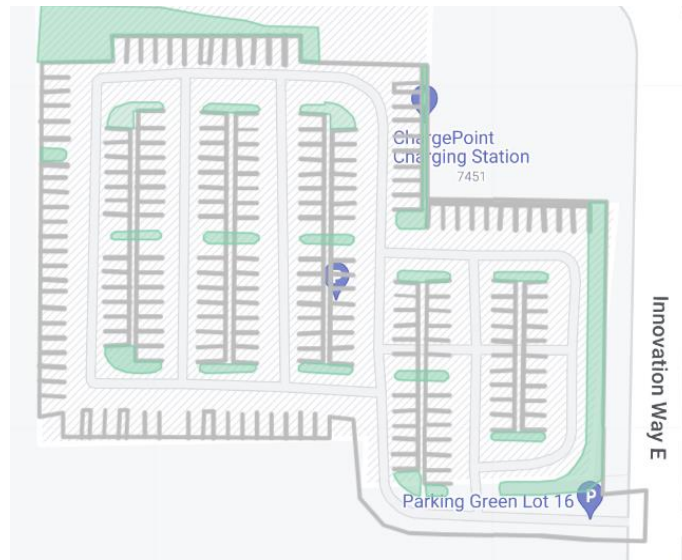


Figure 17: Generated Custom Map

```
(psdetect) C:\Users\prana>python3 navigation.py
```

Figure 18: Executing Navigation Route Generation Algorithm

Figure 19 represents the real-time route to the empty parking spot, where point A represents the user's current location and point B represents the parking spot's location.



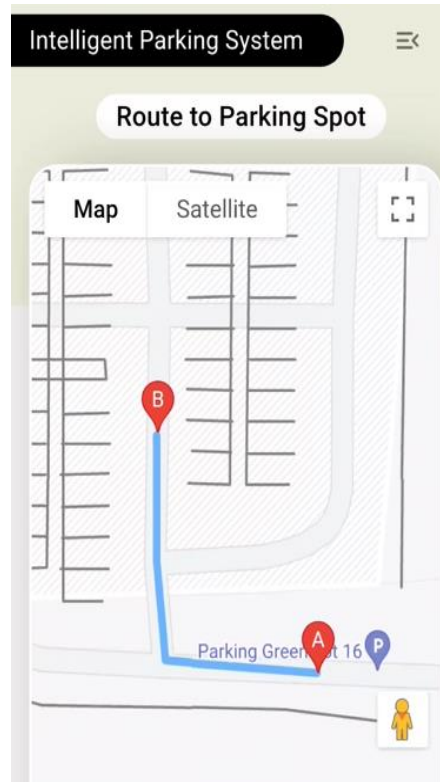


Figure 19: Generated Navigation Route Guidance

### Website Interface & Network Communication

System communication and web interface design had to be hosted wholly in the cloud to reduce network latency, throughput, and cost. Once the detection module publishes its results to MongoDB, those are made available to users with a lambda microservice and React website. Two separate data pipelines were built to reduce congestion and latency for detection module results and video streaming. This helps decouple data streams and decrease data updates and retrieval latency. Initially, the choice was to move the video streaming to the cloud and employ a detection module, making it easier to publish data to the website and fault-resistant and scalable. Though it introduced new latency problems on slower connections, it reduced the hardware required on-site at the parking system. This

mode's problem was maintaining a reliable connection and losing network data. The detection module was moved on-site using an edge computer to eliminate this. By doing this, the main result from the detection module (Vacancy status of parking spots) is readily available at the edge, even without an internet connection. This is the basis for further exploring network communication and web interface implementation details.

Once the video reaches the edge device (through wire), it must be optimized for video upload to the cloud, which gets published to the user interface for reference. This video doesn't necessarily have to be high-quality and can be fine-tuned based on the network performance. For this reason, a GStreamer component is used to process the video before sending it to the cloud. GStreamer acts like a pipeline to process the input video in different stages and optimize it before sending it to the cloud. GStreamer with video convert and h264enc components is a must for optimizing and converting video to publish it to the cloud. The component's video conversion optimizes the video resolution to 640X480 and sets the framerate to 30FPS. This reduced the average network latency by 4 seconds for HD video recorded with a GoPro camera. Once optimized and encoded, it is inserted into kvssink queue to get published into kinesis video streams. Hosted streams provide excellent reliability and minimal maintenance. The minimum configuration required for any video stream is to hold the video for at least 24 hours. Also, Kinesis video streams provide decoupled functions to consume content from streams and use it in further stages. Figure 20 depicts three elements that capture, convert, and compress the feed for upload to Kinesis Video Streams. The v4l2src element captures video from the camera source connected to the edge device, while the video conversion and h264enc elements scale and encode the

video. Finally, the feed is dumped to kvssink and built from the CPP plugin for uploading AWS Kinesis Video Streams.

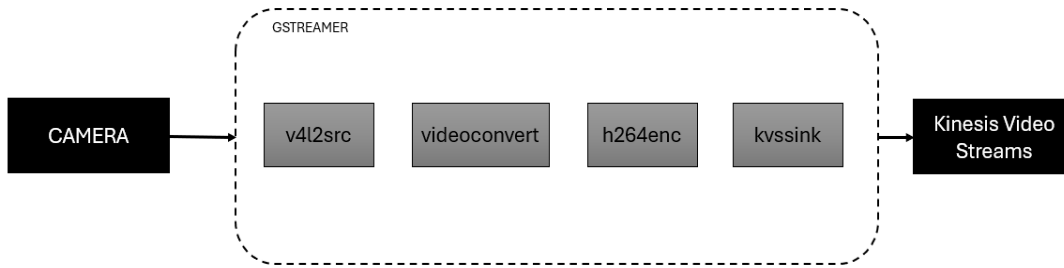


Figure 20: Video Upload using Gstreamer

This pipeline published data to the cloud with 2-4 seconds latency. REST API drives the data that fuels the web interface calls from the React website. The detection module tracks the parking spot vacancy status in real-time, and the web interface sends out user updates tailored to their requirements. Implementing web sockets for real-time view updates trades system resources for seamless information access and using REST API gives efficient cost management. After experimenting with GraphQL and REST, we observed reduced calls to middleware (AWS Lambda API) with REST and continued to build interfaces with REST API. The UI doesn't depend on what type of API is being used; hence, both can be swapped. The reduced calls directly reflect the cost associated with middleware. As shown in Figure 21, the backend for the application sits in AWS lambda and is connected to the react frontend using API Gateway. Primary computing that enhances the user experience is moved to the client machine.

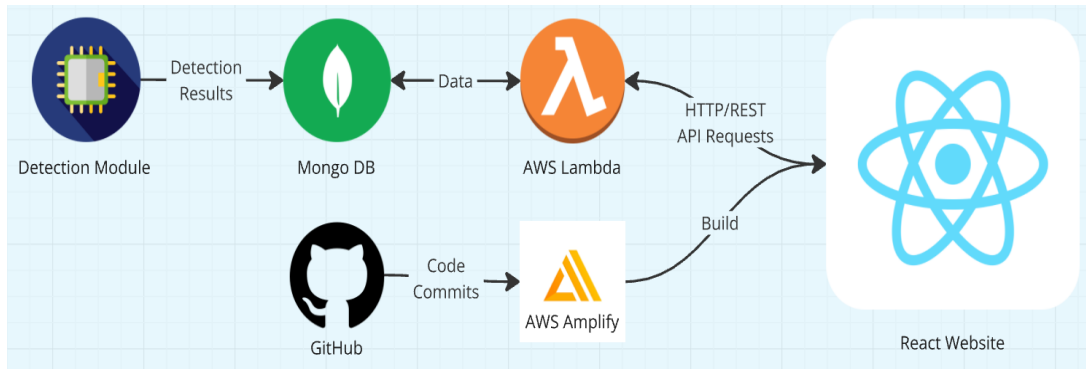


Figure 21: Data communication in the backend

The React website, marked as Web Interface in Figure 22, connects to the backend system, Kinesis Video streams, and Google Location services. The user interface is divided into three pages, each dealing with a separate module but connected with a single react hook that makes the results from the detection module available to the entire website.

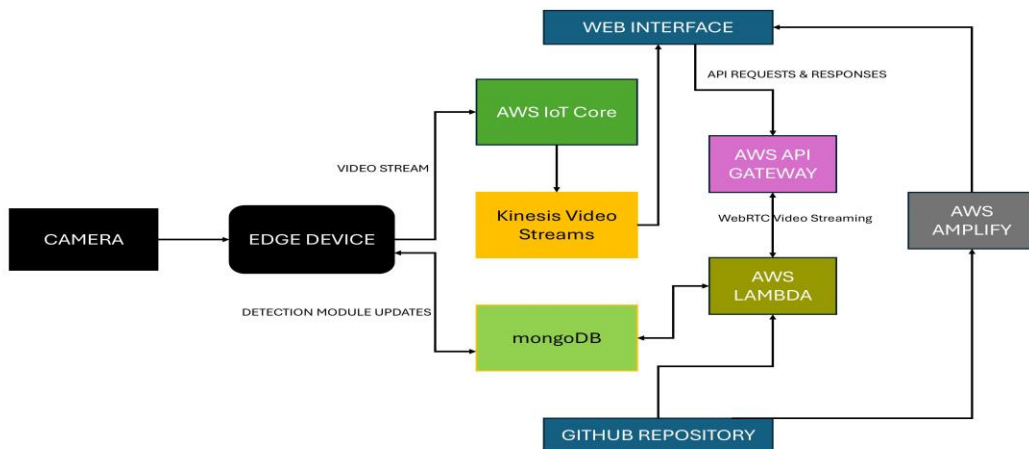


Figure 22: Website Interface Architecture

The listing page from Figure 23 shows how the results from the detection module are visualized on the website. The first entry of the listing is marked to indicate prioritization, determined based on the distance between the user and the parking spots, denoting the

closest parking spot in the parking lot. The distance parameter here is not calculated on the client machine to reduce computing on this page. The filters on the page work entirely locally on the client machine, reducing network calls.

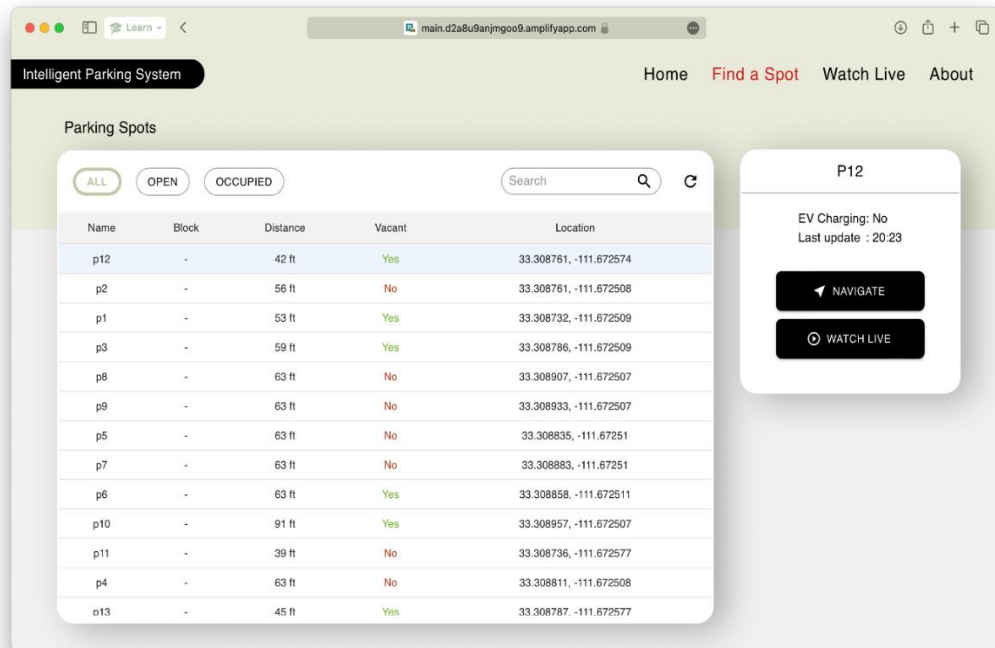


Figure 23: WEBUI: Find parking spot page

Figure 24 shows the navigation from the user location to the parking spot. We integrated Google Maps API into this page to dynamically update the destination location based on the vacancy status, distance, and user location. Initially, our approach was to determine the directions once and not change the destination, as it would become increasingly challenging to track vacancy statuses when the direction interface is in use. We optimized the network calls so that the react website queries the Google distance API in a single call and determines the nearest parking spot locally at the client location.

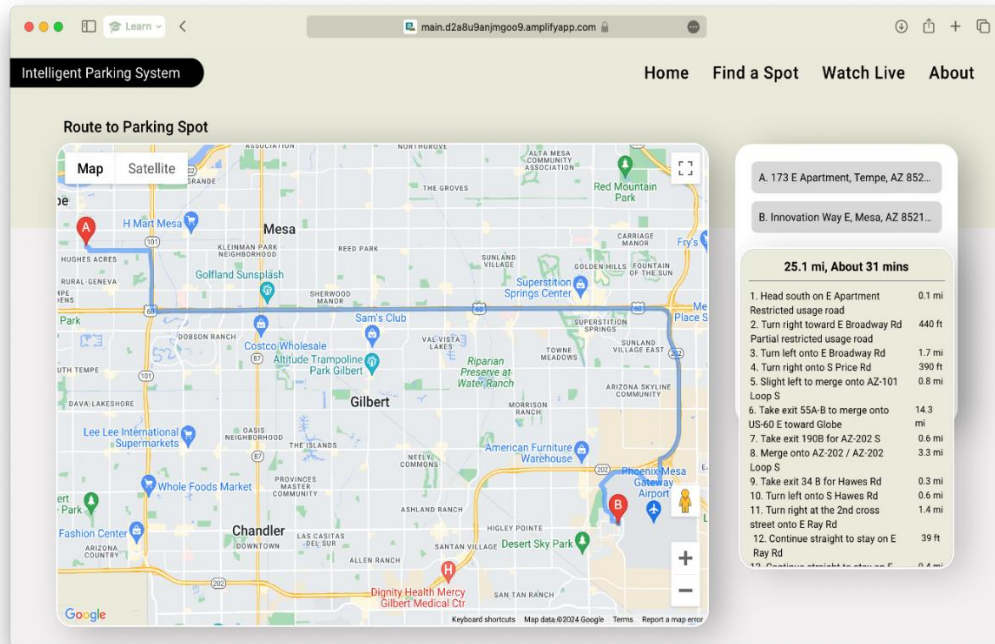


Figure 24: WEBUI: Navigation Page

### Centralized Database

In a centralized parking management system, essential data of each parking spot within a parking lot is stored to facilitate monitoring and management of parking activities. Each parking spot is associated with several key fields of information that collectively offer a comprehensive understanding of its state and history. Figure 25 depicts the centralized database structure used for the intelligent parking system.

**Name (Character):** The name field is a unique identifier for each parking space. It helps distinguish between different parking spots and enables the correlation of data with specific locations within the parking lot. For example, names like "P1" or "P11" could represent individual parking spots.

GPS Coordinates (Float): The GPS coordinates field provides precise geographical location information for each parking spot. This data is crucial for visualizing the exact position of the parking space on maps and assisting users in locating available parking. By integrating GPS coordinates into mapping applications, users can navigate directly to vacant parking spots within the parking lot.

Distance (Integer): The distance field denotes the distance of each parking space from the parking lot's entry point. It helps users gauge the proximity of available parking spaces to the entrance, aiding in decision-making when selecting a parking spot.

Occupancy Status (Boolean): The occupancy status field indicates whether a parking spot is occupied or vacant. It is updated in real time. When a parking spot is occupied, the status is marked as "true," whereas when it is vacant, it is recorded as "false."

The collective data stored for each parking space enables real-time occupancy monitoring, historical analysis of parking trends, and the development of intelligent parking solutions. Using this information, some data analysis can be implemented to improve parking efficiency, reduce congestion, and enhance the overall user experience. Additionally, insights derived from parking usage patterns can inform decision-making processes related to infrastructure development and resource allocation within the parking facility.

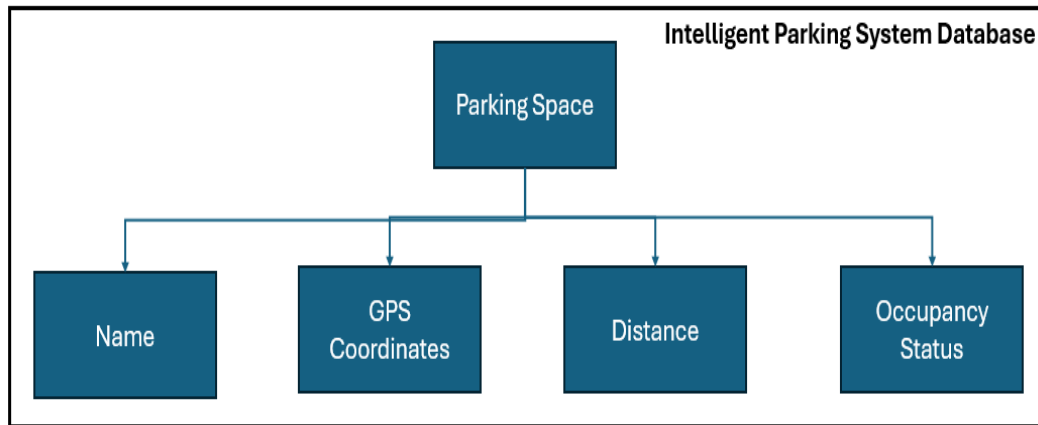


Figure 25: MongoDB database structure



## RESULTS & DISCUSSION – SINGLE INFRASTRUCTURE CAMERA-BASED INTELLIGENT PARKING GUIDANCE SYSTEM

Figure 26 showcases a monocular GoPro Hero 10 camera mounted on a tripod at 30 feet at an optimal angle to acquire the perfect view of an outdoor parking lot containing 20 parking spaces. The camera is seamlessly connected to the NVIDIA Jetson AGX Orin, the system's edge computer. The live feed from the camera is processed in the edge computer to monitor the parking lot in focus.

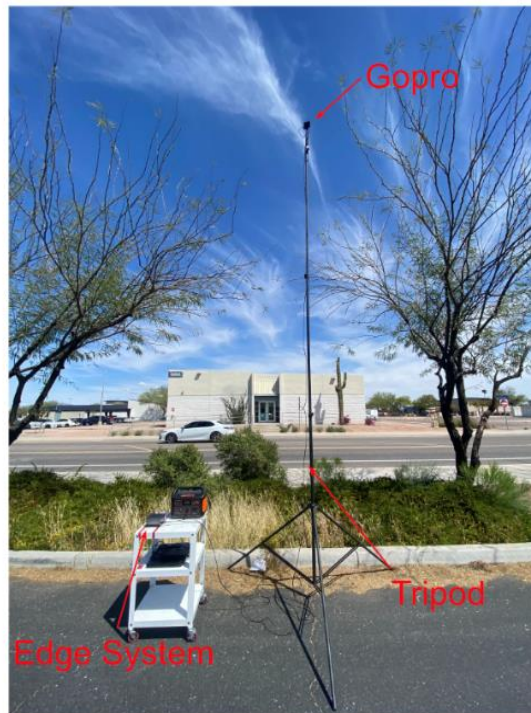


Figure 26: Hardware Setup

Multiple rounds of real-time testing were performed, during which the robustness and accuracy of the intelligent parking system were analyzed. The individual software modules and the overall system integration were evaluated. Figure 27 shows the computer vision

processed frame of the live GOPRO feed, navigation route guidance provided to the driver, and the view from inside the vehicle during the testing and validation.

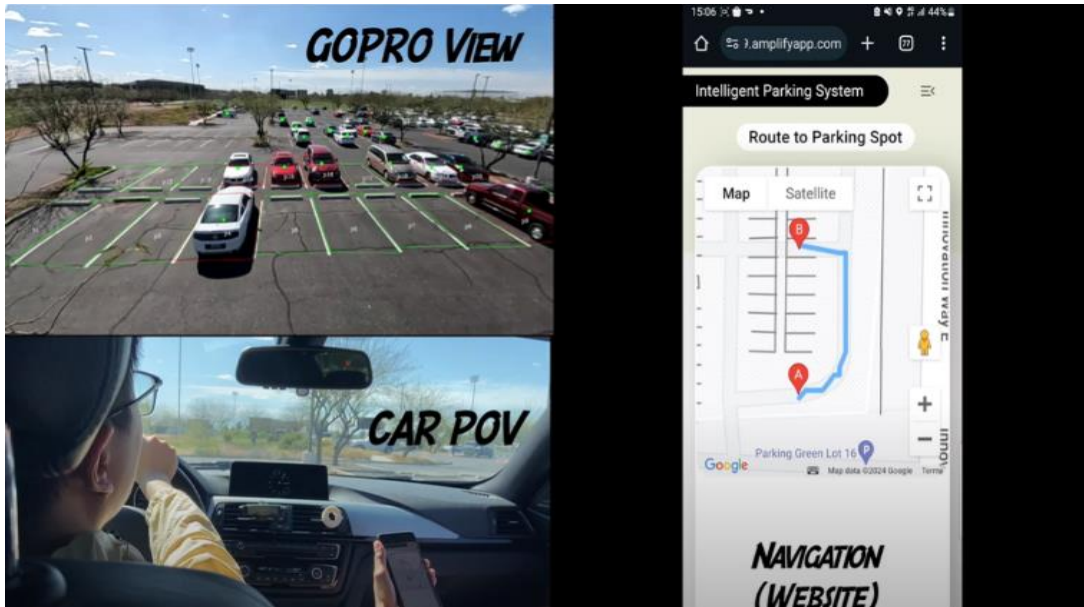


Figure 27: Testing & Validation of the prototype system

The intelligent parking system website would ask for the user's location access when they open it. Once they permit the system to access their location, the users can enter the “find a spot” page to find vacant parking spaces. The system would recommend a parking space solution for the user, usually the vacant spot closest to the user’s location. The name of the parking space, along with its location coordinates, would be listed at the top of the list of all the vacant parking spaces. The user can choose either the recommended parking spot or one of the other vacant ones on the list.

During each round of testing, two main functionalities were tested. At first, we chose the parking spot recommended by the system (p11) and clicked the navigate button. The system showed navigation route guidance to parking spot p11, and we could comfortably

maneuver the vehicle to the spot. Next, we tested the second functionality, which allowed the user to choose any open parking spot. We chose parking spot p6 and clicked the navigate button, and the system showed us the route guidance to parking spot p6, as shown in Fig 10. We also tested out a third functionality of the system, which can reroute the user to the nearest parking spot from their current location if the user gets lost in the parking lot trying to navigate a parking space. The system recommended p11 as the optimal solution, so we chose p11 and clicked the navigate button. But we deliberately went into another aisle in the parking lot without following the navigation route guidance given by the system to p11. Immediately, the system rerouted us to the new nearest vacant parking space in our aisle. This proved the system's ability to provide dynamic navigation route guidance.

We have used the NoSQL database to store the data. This type of database often supports a schematic design, which means that each record in the database can have its own structure. The request to update the database using the pool requests concept drastically reduced the lag of the on-the-edge system. Initially, the navigation system computed the nearest parking spots on the edge device, causing a notable delay of 6 seconds in the navigation module. Real-time routing was implemented to improve performance by shifting Google API calls to the user's device, eliminating the significant lag. The distance calculation from the current location to the parking spots was optimized to determine the nearest parking spaces by minimizing Google API calls. This also proved that the localization and navigation modules worked individually without lag.

The detection model used to detect the vehicle model's accuracy is evaluated using the mAP50 metric, which stands for mean Average Precision at IoU threshold 0.5. This metric

is widely used in object detection tasks and calculates the average precision across various object classes. The detection of cars with the yolov5s model was about 57%. Further, a custom dataset containing 200 images of vehicles was added to the COCO dataset, and the yolov5s model was trained on the new COCO dataset on the ROBOFLOW website. This drastically improved the vehicle detection accuracy to 93.6%, which directly improved the accuracy of the vacant parking space detection algorithm. Table II depicts the difference in detection accuracy percentage between different Yolo models.

Table II: Comparison of different Yolo Models

<b>Model</b>	<b>Yolov5s</b>	<b>Yolov5x6</b>	<b>Yolov5x6 Custom</b>
<b>mAP50</b>	56.8%	72.7%	93.6%
<b>mAP50-95</b>	37%	55%	57%

Overall, this intelligent parking guidance system with one infrastructure camera works robustly, making parking and finding a spot much smoother. The following section discusses how the same system can be implemented using multiple cameras.

# MULTIPLE INFRASTRUCTURE CAMERAS – BASED INTELLIGENT PARKING GUIDANCE SYSTEM

## Introduction

### Problem Statement

As discussed in the previous section, there is an increasing demand for an innovative, intelligent parking solution that can make parking more accessible while reducing traffic accidents, wasted time & greenhouse gas emissions. This thesis in the previous sections showcased the prototype of an intelligent parking guidance system that can monitor the parking lot using a monocular camera, identify the nearest vacant parking space, and provide navigation route guidance to the user to maneuver to the vacant parking space. However, one drawback of this system is that it can monitor only a small number of parking spaces as it uses only one infrastructure camera. Another problem this system faces is its struggle to perform robust vehicle detection in the parking lot when the vehicles are occluded by other cars. This also affects the detection of vacant parking spaces. False positives/negatives form another major concern with vacant parking space detection using a single infrastructure camera. To tackle these problems, upgrading this prototype system by replacing a single infrastructure camera with multiple cameras would make the system more robust.

### Objective

The main objective was to upgrade the prototype system developed by adding multiple cameras to monitor the parking lot and mitigate the problems of occlusion and false

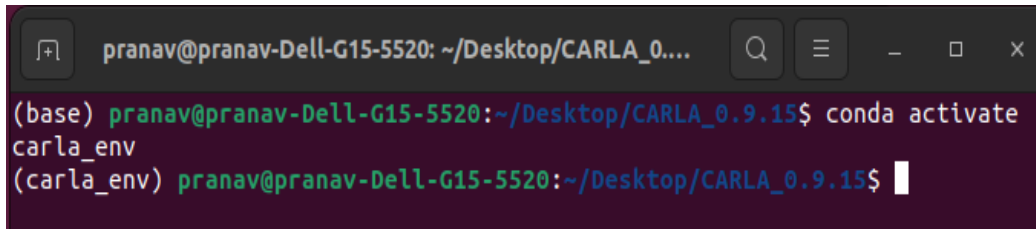
positives/negatives during vacant parking space detection and occupancy status calculation. The upcoming sections propose and discuss multiple infrastructure cameras–based intelligent parking guidance system. Vacant parking space detection was performed using multiple cameras in a simulated parking lot environment instead of a single infrastructure camera, and the new results of the new parking space detection system can then be seamlessly integrated with the remaining components of the developed prototype system, i.e., Parking Space Localization, Navigation Route Generation & Website Interface.

### Simulation Environment Setup

CARLA [26] was the simulation software chosen to simulate a large parking lot with vehicles and infrastructure cameras. The open-source simulator CARLA (Car Learning to Act) was created especially to study and advance autonomous driving. It offers a high-fidelity virtual environment to train, test, and validate self-driving models. This allows researchers to develop and test autonomous vehicle systems in a realistic, controlled setting. It supports a wide range of sensors, such as RGB cameras, LiDAR, RADAR, GPS, and depth cameras, which can be virtually placed in parking lots as monitoring cameras or on roads as roadside units. Comprehensive 3D maps of actual urban environments, complete with streets, buildings, stop signs, and traffic lights, are included with the simulator. Town 1, Town 2, and Town 5 are popular maps offering different difficulty levels and testing scenarios. It mimics various weather scenarios, such as bright, cloudy, and rainy, as well as changes in the time of day. Different in-built Python APIs enable developers to control simulation parameters, spawn vehicles, retrieve sensor data, and define complex behaviors for autonomous agents. The CARLA software was installed in a Ubuntu 22.04 operating system-based computer. The steps to start and run the CARLA simulation software are given below:

Step 1: Activate the Conda Environment, as shown in Figure 28.

Command to run: `conda activate Carla_env`

A terminal window with a dark background. The title bar shows the user 'pranav' on a 'pranav-Dell-G15-5520' machine, with the current directory being '~/Desktop/CARLA\_0....'. The prompt is '(base) pranav@pranav-Dell-G15-5520:~/Desktop/CARLA\_0.9.15\$'. The user has entered the command 'conda activate carla\_env'. The prompt has changed to '(carla\_env) pranav@pranav-Dell-G15-5520:~/Desktop/CARLA\_0.9.15\$' and a cursor is visible at the end of the line.

```
(base) pranav@pranav-Dell-G15-5520:~/Desktop/CARLA_0.9.15$ conda activate carla_env
(carla_env) pranav@pranav-Dell-G15-5520:~/Desktop/CARLA_0.9.15$
```

Figure 28: Conda Environment Activation

Step 2: Run the command to initialize the CARLA software, as shown in Figure 29.

Command to run: `./CarlaUE4.sh`

A terminal window with a dark background. The title bar is the same as in Figure 28. The prompt is '(base) pranav@pranav-Dell-G15-5520:~/Desktop/CARLA\_0.9.15\$'. The user has entered 'conda activate carla\_env'. The prompt has changed to '(carla\_env) pranav@pranav-Dell-G15-5520:~/Desktop/CARLA\_0.9.15\$'. The user has entered './CarlaUE4.sh'. The terminal output shows '4.26.2-0+++UE4+Release-4.26 522 0' and 'Disabling core dumps.' followed by a blank line and a cursor.

```
(base) pranav@pranav-Dell-G15-5520:~/Desktop/CARLA_0.9.15$ conda activate carla_env
(carla_env) pranav@pranav-Dell-G15-5520:~/Desktop/CARLA_0.9.15$ ./CarlaUE4.sh
4.26.2-0+++UE4+Release-4.26 522 0
Disabling core dumps.

```

Figure 29: Carla Environment Initialization

The CARLA simulator opens with the built-in Town10 map, as shown in Figure 30.



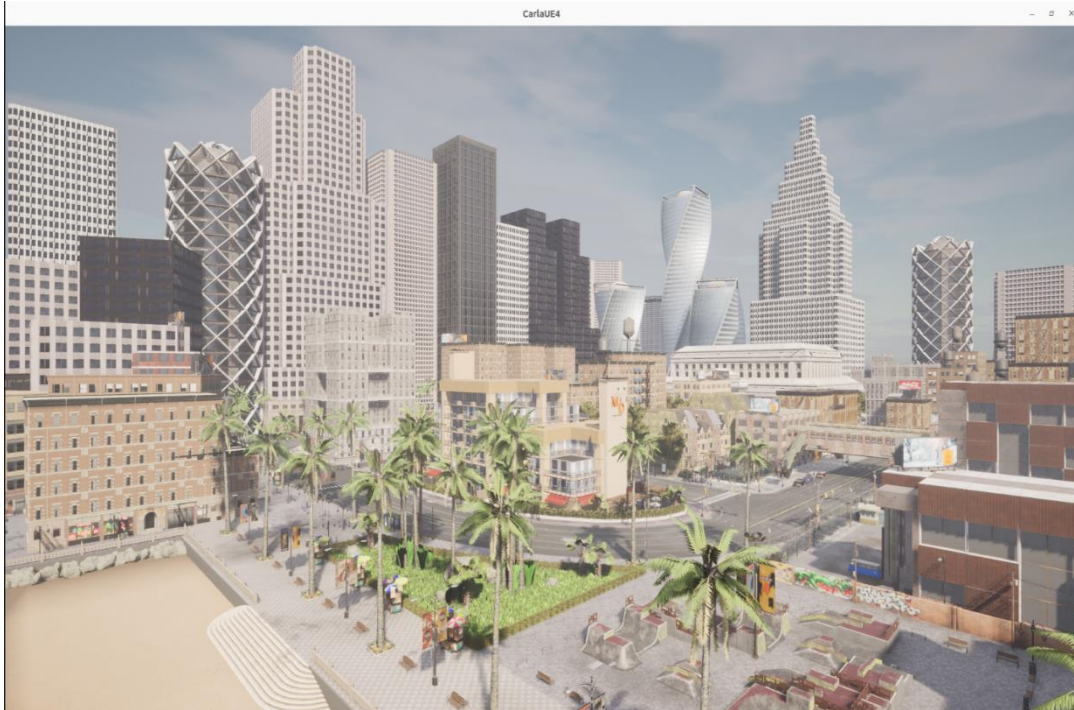


Figure 30: Carla Simulator

To simulate a multiple infrastructure camera setup, a parking lot with 60 parking spaces spread over three aisles was chosen from one of CARLA's built-in maps (Town05). The steps to open the Town 5 map in CARLA and view the parking lot are given below:

Step 1: Activate the conda environment and run the command to initialize the CARLA software

Step 2: Open a new terminal and run the command to open the town5 map in CARLA, as shown in Figure 31.

Command to run: `python3 config.py --map Town05`

```
(carla_env) pranav@pranav-Dell-G15-5520:~/Desktop/CARLA_0.9.15/PythonAPI/util$ python3 config.py --map Town05  
load map 'Town05'.  
(carla_env) pranav@pranav-Dell-G15-5520:~/Desktop/CARLA_0.9.15/PythonAPI/util$
```

Figure 31: Initializing Town05 Map

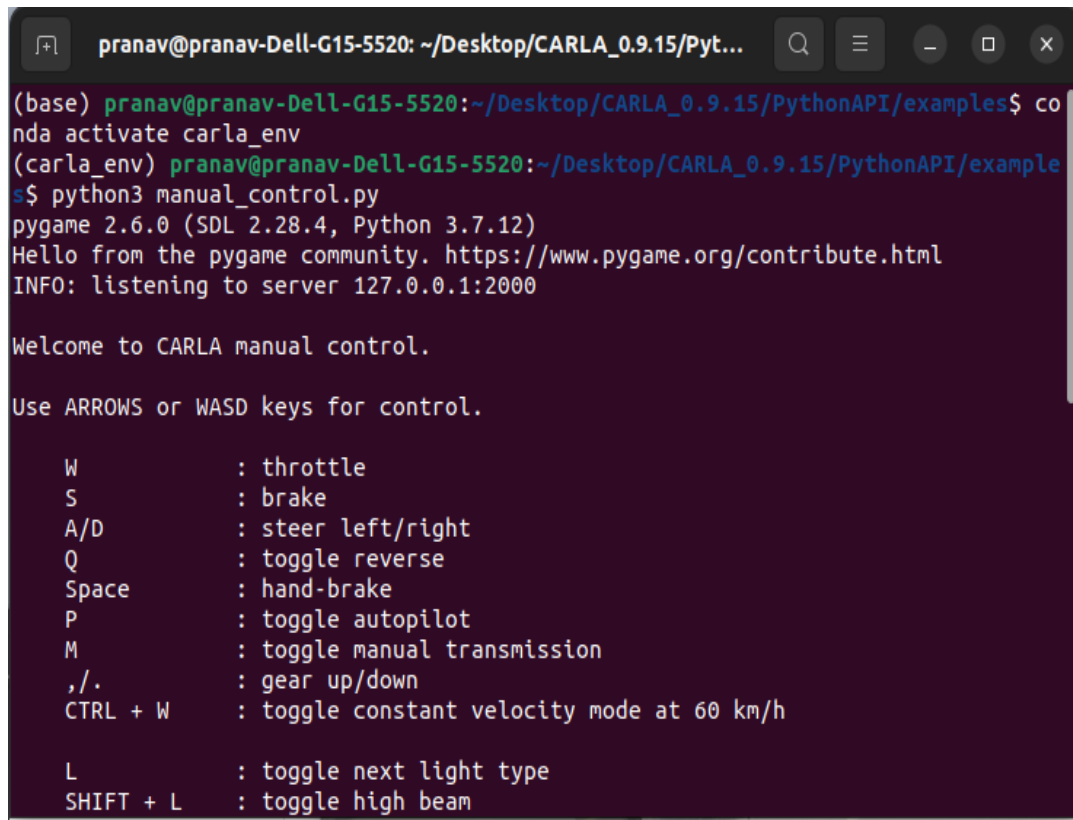
Step 3: Zoom into the map and navigate to the parking lot manually using the keyboard keys W, A, S, and D to get the top view of the map with 60 parking spaces.

The next crucial step is to spawn four cameras in the four corners of the parking lot to monitor all 60 parking spaces collectively. The location coordinates at which the four cameras will spawn must be gathered. A vehicle was spawned near the parking lot to collect the coordinates. The car was manually maneuvered inside the parking lot to the four different corners using the keyboard keys W, A, S, and D, and the x, y, and z coordinates at which the four cameras needed to be spawned were collected. The steps to spawn the vehicle are given below:

Step 1: Activate the Anaconda environment and initialize the CARLA simulator with the Town5 map

Step 2: Enter a new terminal, activate the Anaconda environment, navigate to the directory containing all the CARLA Python APIs, and run the “manual\_control.py” file, as shown in Figure 32.

Command to run: `python3 manual_control.py`

A terminal window with a dark background and light-colored text. The window title is 'pranav@pranav-Dell-G15-5520: ~/Desktop/CARLA\_0.9.15/Pyt...'. The terminal shows the following commands and output:

```
(base) pranav@pranav-Dell-G15-5520:~/Desktop/CARLA_0.9.15/PythonAPI/examples$ conda activate carla_env
(carla_env) pranav@pranav-Dell-G15-5520:~/Desktop/CARLA_0.9.15/PythonAPI/examples$ python3 manual_control.py
pygame 2.6.0 (SDL 2.28.4, Python 3.7.12)
Hello from the pygame community. https://www.pygame.org/contribute.html
INFO: listening to server 127.0.0.1:2000

Welcome to CARLA manual control.

Use ARROWS or WASD keys for control.

      W      : throttle
      S      : brake
    A/D      : steer left/right
      Q      : toggle reverse
    Space    : hand-brake
      P      : toggle autopilot
      M      : toggle manual transmission
    ,/.      : gear up/down
    CTRL + W : toggle constant velocity mode at 60 km/h

      L      : toggle next light type
    SHIFT + L : toggle high beam
```

Figure 32: Spawning and Manually Controlling a vehicle

An example of the spawned vehicle collecting the location coordinates of one of the camera's spawn locations by maneuvering to one of the corners of the parking lot is depicted in Figure 33.



Figure 33: Collecting Infrastructure Camera Spawn Location Coordinates

Table III contains the CARLA location coordinates of all four infrastructure cameras collected during the abovementioned process.

Table III: Multiple Infrastructure Cameras Spawn Locations

Infrastructure Camera	CARLA Location Coordinates
Camera 1	(X1, Y1, Z1)
Camera 2	(X2, Y2, Z2)
Camera 3	(X3, Y3, Z3)
Camera 4	(X4, Y4, Z4)

Each camera was used to monitor 30 parking spaces. The field of view for a specific set of cameras overlapped, resulting in some of the parking spaces being observed by more than

one camera. All the cameras were placed at a height of 7 meters, with a field of view of 90 degrees and a tilt angle of 40 degrees.

The location coordinates (X, Y, Z) of all the parking spaces were collected by manually driving the spawned vehicle inside each simulated parking space, similar to collecting camera spawn locations.

Table IV depicts the list of parking spaces and their corresponding location coordinates. Using the parking spaces' location coordinates, a few vehicles were spawned in some of the spaces using a Python API, and a CARLA scenario was generated, in which the four spawned infrastructure cameras monitored the parking lot with cars parked in a few spots. The steps to spawn the vehicles and develop the CARLA scenario are given below:

Step 1: Activate the Conda environment and run the command to initialize the CARLA software.

Step 2: Open a new terminal, activate the Anaconda environment, navigate to the directory containing all the CARLA Python APIs, and run the “spawn\_vehicles.py” file as shown in Figure 34.

Command to run: `python3 spawn_vehicles.py`

```
pranav@pranav-Dell-G15-5520: ~/Desktop/CARLA_0.9.15/Pyt...  
(base) pranav@pranav-Dell-G15-5520:~/Desktop/CARLA_0.9.15/PythonAPI/examples$ co  
nda activate carla_env  
(carla_env) pranav@pranav-Dell-G15-5520:~/Desktop/CARLA_0.9.15/PythonAPI/example  
s$ python3 spawn_vehicles.py  
ERROR: Spawn failed because of collision at spawn position  
ERROR: Spawn failed because of collision at spawn position  
ERROR: Spawn failed because of collision at spawn position  
spawned 30 vehicles and 7 walkers, press Ctrl+C to exit.
```

Figure 34: Carla Scenario to spawn vehicles in the parking lot

Figure 35 depicts the setup of the cameras and vehicles in the parking lot and the parking space coverage for each camera.

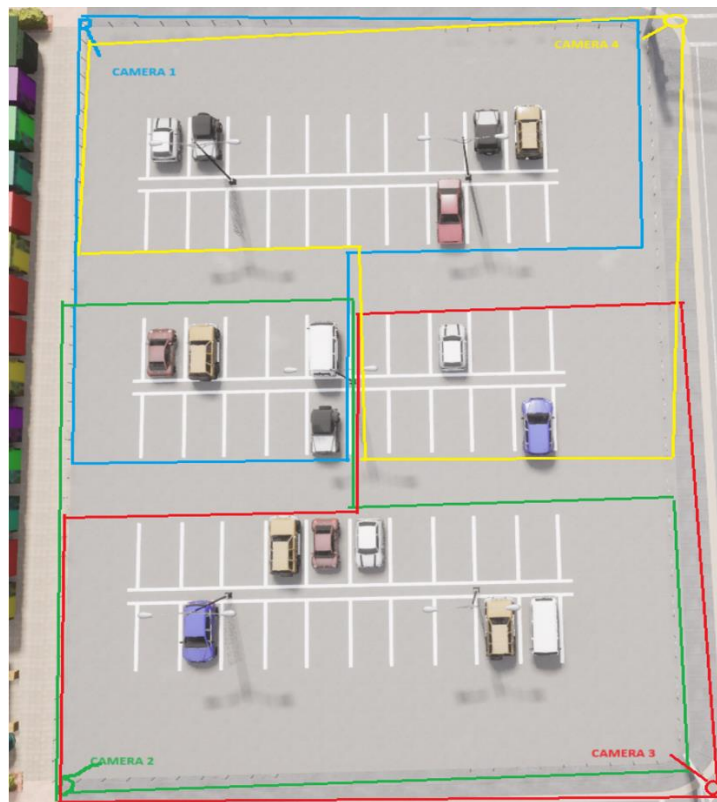


Figure 35: Parking lot Environment Setup with Spawned Vehicles

### Multiple Cameras-based Parking Space Detection

The parking space detection algorithm used in the case of a single infrastructure camera-based prototype system was also used in the case of multiple cameras-based detection system. The main idea behind achieving multiple camera-based vacant parking space detection was that the live feed from each infrastructure would be processed using the vacant parking space detection algorithm. Since there is an overlap in the parking space coverage between the cameras, the detection results of one camera would be validated by another. In this case, the same set of 20 parking spaces were monitored by both Camera 1 and Camera 4. A set of 10 parking spaces were monitored by Camera 1 and Camera 2. Similarly, another set of 20 parking spaces was monitored by both Camera 2 and Camera 4, while a different set of 10 parking spaces was monitored by both Camera 3 and Camera 4. Finally, the occupancy status results from each camera's vacant parking detection algorithm were fused using an innovative methodology to produce the occupancy status results for the overall parking lot at each instant.

The live feed from each infrastructure camera was captured using a Python API. The steps to run the Python API to get the live feed from each camera are given below:

Step 1: Activate the Anaconda environment and initialize the CARLA simulator with the Town5 map.

Step 2: Open a new terminal, activate the conda environment, navigate to the CARLA python API directory, and run the “camera\_test.py” file as shown in Figure 36

Command to run: `python3 camera_test.py`

```
connected by user: user
(carla_env) pranav@pranav-Dell-G15-5520:~/Desktop/CARLA_0.9.15/PythonAPI/example
s$ python3 camera_test.py
The CARLA server is in asynchronous mode.
```

Figure 36: Collecting live feed from the Infrastructure Cameras

Since the vacant parking space detection algorithm involves vehicle detection, the live feed from the cameras was captured, and vehicle detection was performed on each of these feeds as the first step in validating the robustness of the object detection algorithm on the simulation data. The steps to perform vehicle detection are as follows:

Step 1: Open the Windows command prompt shell, activate the conda environment, and navigate to the “Intelligent Parking” directory.

Step 2: Run the “Vehicle\_Detection.py” file as shown in Figure 37.

```
C:\Users\prana>conda activate psdetect
(psdetect) C:\Users\prana>cd C:\Users\prana\OneDrive - Arizona State University\Documents\IntelligentParking
(psdetect) C:\Users\prana\OneDrive - Arizona State University\Documents\IntelligentParking>python3 vehicle_detection.py
```

Figure 37: Vehicle Detection Algorithm Initialization

Figure 38, Figure 39, Figure 40 & Figure 41 show the live feed from Camera 1, Camera 2, Camera 3, and Camera 4, respectively, and portray the detected vehicle in each camera frame.



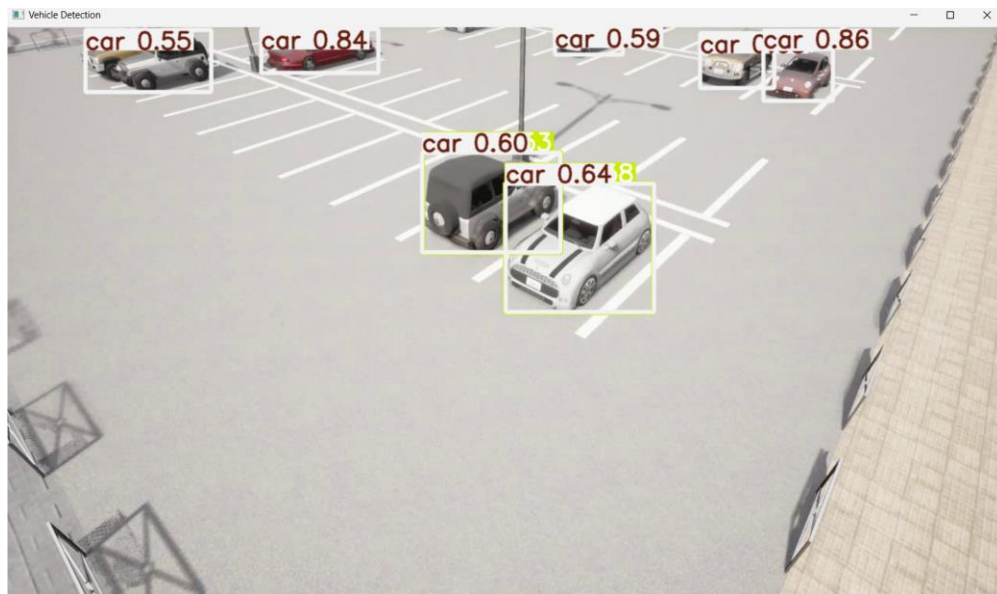


Figure 38: Detected Vehicles by Camera 1



Figure 39: Detected Vehicles by Camera 2

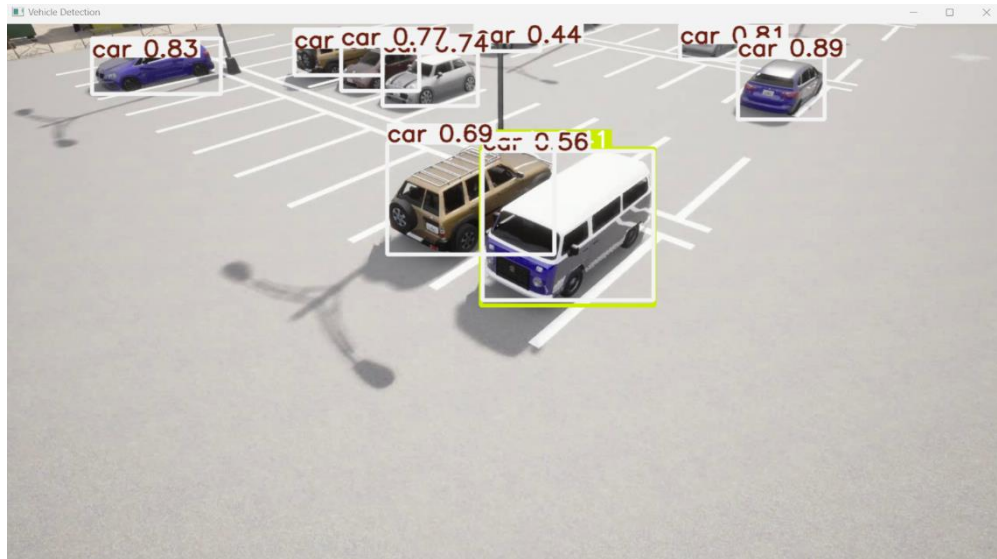


Figure 40: Detected Vehicles by Camera 3



Figure 41: Detected Vehicles by Camera 4

Like single infrastructure parking space detection, the Bresenham algorithm defined the regions of interest, i.e., the parking spaces in focus in each camera frame. The centroids of the bounding boxes of all the detected vehicles in each frame were also identified. The IoU Ray Casting technique determined the overlap region between the parking space bounding and the detected vehicle parked in the spots. Finally, the IoU for every parking space was computed. The vacant parking space detection algorithm was used to do this process and calculate the occupancy status for each parking spot in all four camera frames. The steps to perform vacant parking space detection using the four infrastructure cameras separately are given below:

Step 1: Go to the Intelligent Parking Directory, Activate the conda environment, and run the file detection.py to perform detection for Camera 1, Camera 2, Camera 3, and Camera 4 frames, as shown in Figure 42.

Command to run: python3 detection.py

```
C:\Users\prana>conda activate psdetect
(psdetect) C:\Users\prana>cd C:\Users\prana\OneDrive - Arizona State University\Documents\IntelligentParking
(psdetect) C:\Users\prana\OneDrive - Arizona State University\Documents\IntelligentParking>python3 detection.py
```

Figure 42: Vacant Parking Space Detection Algorithm Initialization

Figure 43, Figure 44, Figure 45, and Figure 46 show the initial occupancy status results of Vacant parking Space Detection using Cameras 1, 2, 3, and 4, respectively.

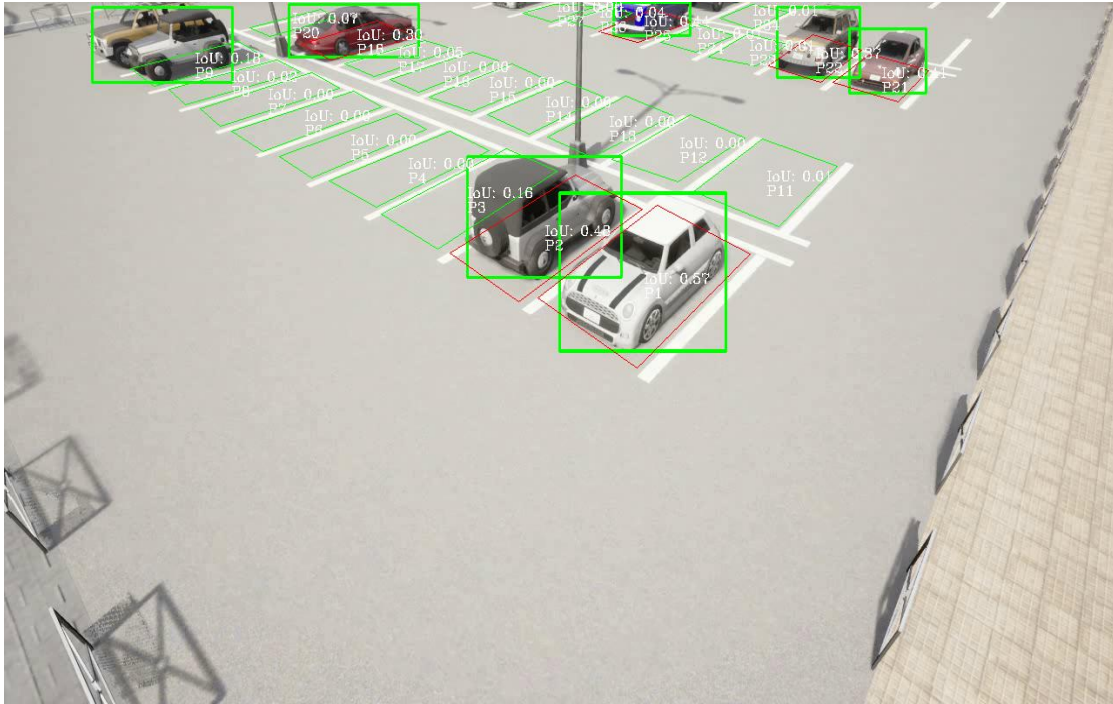


Figure 43: Initial Parking Space Occupancy Status from Camera 1 live feed

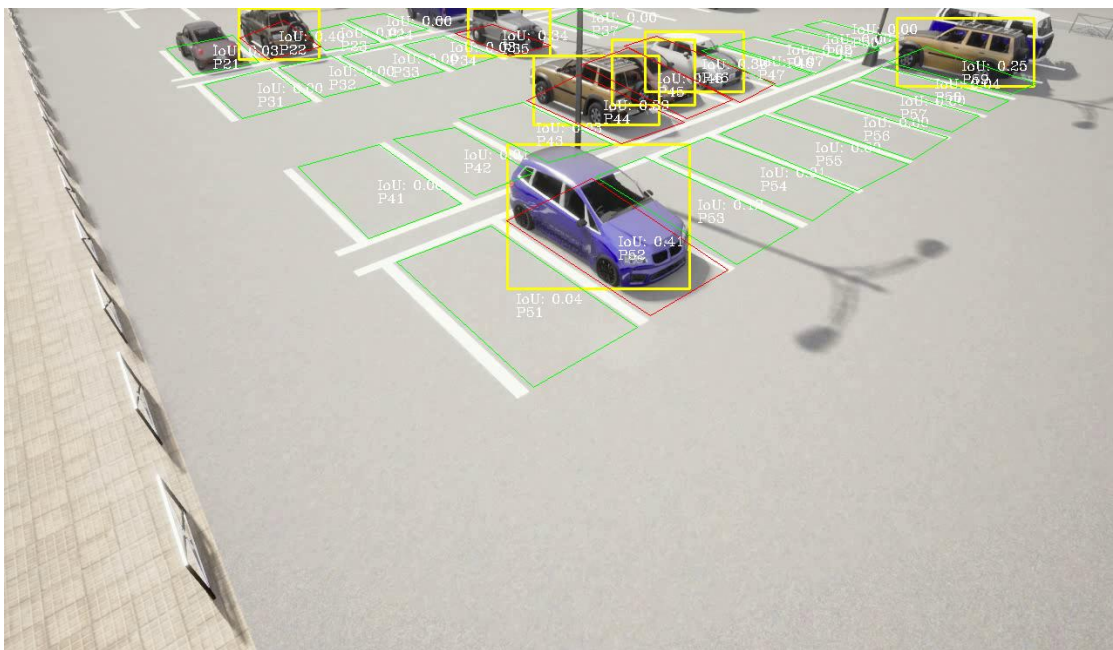


Figure 44: Initial Parking Space Occupancy Status from Camera 2 live feed



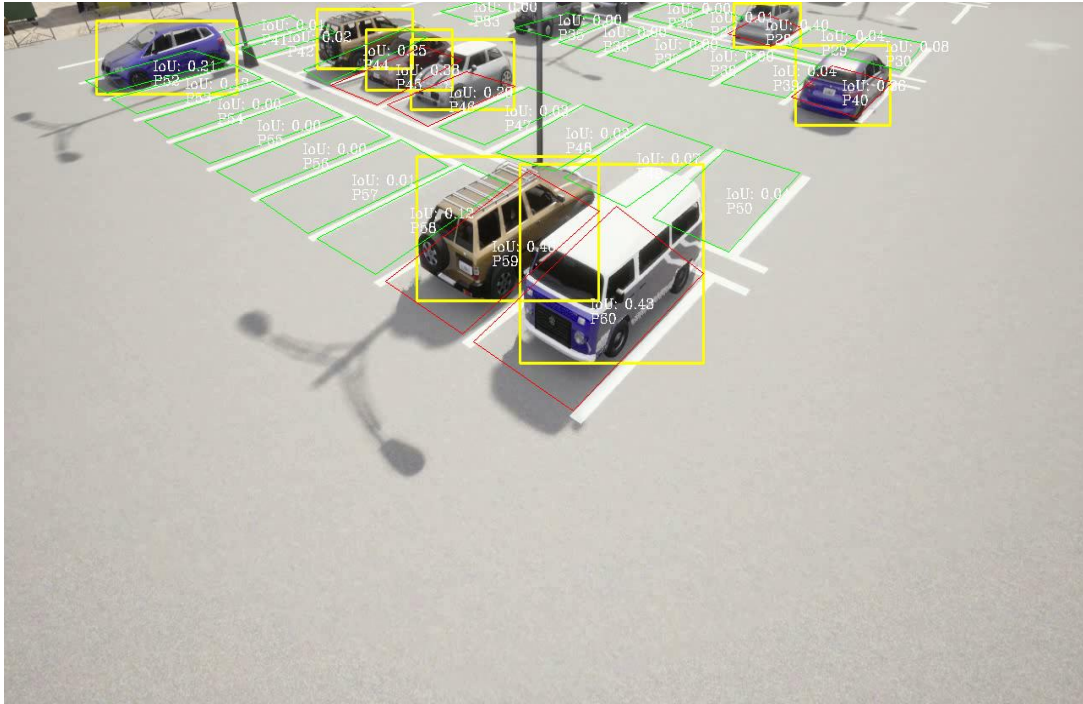


Figure 45: Initial Parking Space Occupancy Status from Camera 3 live feed

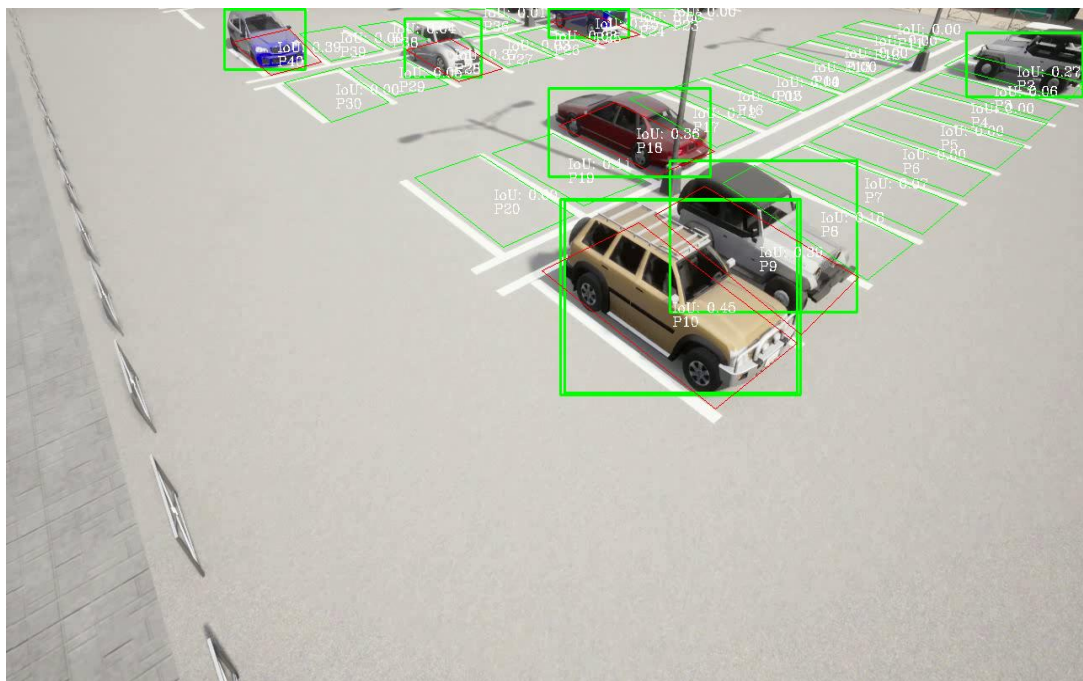


Figure 46: Initial Parking Space Occupancy Status from Camera 4 live feed

There are several cases of parking spaces with or without vehicles being occluded. In some camera frames, those parking spaces' occupancy status cannot be determined. However, the multiple infrastructure cameras set in the parking lot help to navigate through this problem of occlusion as a parking space or vehicle that is being occluded in one camera can be detected in another camera frame as all these have common overlapping view ranges. Another primary concern while using a single camera for parking space detection is the inability to deal with the issues caused by false positives/negatives. The only solution in this case is to use a well-trained, validated, and tested machine learning model to avoid this issue. While that can be done, using multiple cameras also helps as there is more than one camera to monitor the same parking lot, which resolves the problem of false positives/negatives. The following subsection discusses our approach to dealing with false positives/negatives using multiple cameras. The results of occlusion cases being dealt with using multiple infrastructure camera setups are discussed in the results & discussion section.

#### Methodology for fusing data from multiple infrastructure cameras

This subsection discusses the methodology for mitigating the false positives/negatives caused during parking space detection using multiple cameras. In the case of multiple cameras, the issue of false positives/negatives can be recognized when one of the cameras detects a parking space as occupied and other cameras monitoring that space identify it as vacant. In this case of conflicting detection results from different cameras, it isn't apparent which camera's result is the correct detection result. In such situations, weights were assigned to each camera, monitoring the parking spot's occupancy status. If two cameras

monitor a parking spot, the distance between each camera and the parking spot is calculated. The weight of that camera was calculated based on the distance between the parking spot and the camera. The lower the distance between the camera and the parking spot, the larger the weight assigned to the camera and vice versa. The idea is to give more importance (higher weight) to the closer cameras in the case of conflicting detection results between two or more cameras focusing on a parking space, as they are likely to have a better view of the parking space. Next, the Intersection over Union (IoU), i.e., the overlap between the detected bounding box of a vehicle and the defined region of a parking space from each camera's perspective, is calculated. It quantifies how well the detected object matches the actual parking space. When multiple cameras monitor the same parking space, the weighted average IoU between all the cameras focusing on the parking space is calculated to combine their inputs into a single value that reflects both the IoU measurements and the reliability of each camera (based on proximity). The weighted average IoU provides a more balanced view by giving more importance to cameras closer to the parking space, thus likely having better accuracy. The parking space is marked as occupied if the weighted average IoU is 0.45 or higher. This threshold represents a confidence level in the detection accuracy. The mathematical formulae to calculate the distance between the camera and the parking space, the weight calculation, the IoU calculation, and the weighted average IoU calculation are discussed in the next subsection.

#### Mathematical Formulae

The system calculates weights for each camera monitoring a parking space based on the Euclidean distance between the camera and the parking space using Equation 1

$$D=(X2-X1)^2+(Y2-Y1)^2+(Z2-Z1)^2 \quad (1)$$

where (X1, Y1, Z1) and (X2, Y2, Z2) are the camera and parking space location coordinates collected in the CARLA simulation environment.

The system calculates the inverse of each distance. Inverse distances are used because a smaller distance should yield a larger weight. The inverse distances are normalized to sum to 1 to calculate the weight. This is done by dividing each inverse distance by the total sum of inverse distances in question for conflicting detection scenarios, as shown in Equation 2.

$$W_i = [(1/D_i) / (\sum_{j=1}^n 1/D_j)] \quad (2)$$

Where n is the number of cameras covering the parking space.

The intersection over the union for each parking spot is calculated using the equation 3

$$(I_A / U_A) \quad (3)$$

Where  $I_A$  is the intersection area, i.e., the overlap area between the detected vehicle's bounding box and the parking space polygon, and  $U_A$  is the total area covered by both the detected bounding box and the parking space polygon minus the intersection area.

Once the distances, weights, and IoU are calculated, the weighted average IoU is calculated using the equation 4

$$\text{Weighted Average IoU} = \sum_{i=1}^n (W_i \times \text{IoU}_i) \quad (4)$$

Where  $W_i$  is the normalized weight of camera i,  $\text{IoU}_i$  is the IoU value from camera i, and n is the number of cameras covering that parking space.

By combining these calculations, the system effectively integrates data from multiple cameras, resolves conflicts, and makes an informed decision about each parking space's



occupancy status. The data integration results from multiple cameras and the parking space occupancy detection of the overall parking lot will be discussed in the next section.

## RESULTS & DISCUSSION – MULTIPLE INFRASTRUCTURE CAMERAS – BASED INTELLIGENT PARKING GUIDANCE SYSTEM

The main aim of using a multiple infrastructure camera setup in the parking lot as an upgrade to the single infrastructure camera setup is to solve the issues of occlusions and false positives/negatives that might occur in the case of single infrastructure camera systems. These issues were mitigated using the setup simulated in the previous sections. The next two subsections will individually discuss the cases of occlusions and false negatives encountered during the simulation and will present the results of them being resolved.

### Occlusion

As depicted in Figure 43 in the live feed frame of Infrastructure Camera 1, the parking space adjacent to the parking space “P9” which is “P10”, and the vehicle parked in “P10” are both partially occluded by the vehicle parked in the parking space “P9”. Similarly, the parking space adjacent to “P18” i.e., “P19” is partially occluded by the vehicle parked in “P18”. However, both these cases of occlusion are resolved using Infrastructure Camera 4. As shown in Figure 46 in the live feed of infrastructure camera 4, the parking space “P10” is visible, and the vehicle parked in “P10” is also detected because of which the final occupancy status of the parking space “P10” is determined as “Occupied”. Similarly, the parking space “P19” is also visible in the live feed of camera 4 and no vehicle was detected in “P19” therefore the final occupancy status of “P19” is determined as “Vacant”.

Similarly, as depicted in Figure 44 in the live feed frame of infrastructure camera 2, the parking space adjacent to the parking space “P59” i.e., “P60” is partially occluded by the

vehicle parked in the parking space “P59”. This was resolved by infrastructure camera 3, as “P60” was visible in the live feed frame of camera 3 as shown in Figure 45 in which the vehicle parked in “P60” was also detected, and the final occupancy status of the parking space “P60” was determined as “Occupied”.

Likewise, as depicted in Figure 45, the parking space “P51” was occluded by the vehicle parked in “P52” in the live feed frame of infrastructure camera 3. However, the same parking space (P51) was visible in the live feed frame of camera 2 as depicted in Figure 44 and no vehicle was detected by camera 2 in “P51” because of which the final occupancy status of the parking space “P51” was identified as “Vacant”.

Another case of occlusion is depicted in Figure 46 which portrays the live feed frame of Infrastructure Camera 4. The parking space “P37” was occluded by the vehicle parked in “P28” but the same parking space was visible in the live feed frame of camera 3 as portrayed in Figure 45 and no vehicle was detected by camera 3 in “P37” because of which the final occupancy status of the parking space “P37” was determined as “Vacant”. Similarly, the parking space “P1” was occluded by the vehicle parked in “P2” in the live feed frame of camera 4 as shown in Figure 46 which was resolved by camera 1. In the live feed frame of camera 1 as depicted in Figure 43, “P1” was visible, and the vehicle parked was also detected because of which the final occupancy status of the parking space “P1” was identified as “Occupied”.

### False Positives/Negatives

Four cases of False Negatives were observed during the simulation i.e., the parking spaces “P2”, “P9”, “P21” and “P35” had conflicting occupancy statuses from different cameras monitoring them. Each of these cases will be discussed in this subsection.

As depicted in Figure 43, the occupancy status of “P2” was observed to be “Occupied” from the live feed frame of camera 1 and the parking space were visually represented as red bounding boxes to portray occupied status. However, the occupancy status of the same parking spaces in the live feed of camera 4 as shown in Figure 46 was determined as “Vacant” and was visually represented with green bounding boxes to portray vacancy status. The IoU of the parking space P2 in Camera 1 and Camera 4 frames are 0.48 and 0.26 respectively as portrayed in Figure 43 and Figure 46. The distances between the “P2” and Camera 1 & “P2” and Camera 4 were 13.6 meters and 31.3 meters respectively as computed. The weights assigned to “Camera 1-P2” and “Camera 4-P2” were 0.69 and 0.3 respectively. The weighted average IoU was computed as “0.41”. If the weighted average  $\text{IoU} \geq 0.30$ , the final occupancy status was determined as “Occupied” and therefore “P2” parking space’s final occupancy status was identified as occupied. This final occupancy status was visually represented back in the live feed frame of Camera 4 and the parking space bounding box of “P2” turned to red portraying occupied status as shown in Figure 47.

Similarly, the occupancy status of “P9” was observed to be “Occupied” from the live feed frame of camera 4, and the parking space was visually represented as red bounding boxes to portray occupied status. However, the occupancy status of the same parking space in the

live feed of camera 1 as shown in Figure 43 was determined as “Vacant” and was visually represented with green bounding boxes to portray vacancy status. The IoU of the parking space P9 in Camera 1 and Camera 4 frames are 0.18 and 0.39 respectively as portrayed in Figure 43 and Figure 46. The distances between the “P9” and Camera 1 & “P9” and Camera 4 were 30 meters and 13.5 meters respectively as computed. The weights assigned to “Camera 1-P9” and “Camera 4-P9” were 0.3 and 0.7 respectively. The weighted average IoU was computed as “0.32”. If the weighted average  $\text{IoU} \geq 0.30$ , the final occupancy status was determined as “Occupied” and therefore “P9” parking space’s final occupancy status was identified as occupied. This final occupancy status was visually represented back in the live feed frame of Camera 1 and the parking space bounding box of “P9” turned to red portraying occupied status as shown in Figure 47. IoUs, distances & weights for Camera 1 - “P9” & Camera 4 - “P9” were identified and the weighted average IoU was calculated to be 0.3 which equals the threshold value as a result, the final occupancy status of “P9” was determined as Occupied and the same was visually represented in the live feed frame of Camera 4 as shown in Figure 48.

Similarly in the case of the parking space “P21”, the live feed from Camera 1 showed “P1” as occupied and visually represented the parking space in red as shown in Figure 43, while the same parking space was identified as vacant and was visually represented as green in the live feed frame of Camera 2 as shown in Figure 44. The weighted average IoU was calculated as 0.37 based on the calculated IoUs, distances, and weights between Camera 1 – P21 & Camera 2 – P21. Since the weighted average IoU was greater than the threshold limit, the final occupancy status of the parking space “P21” was determined as “Occupied”,

and the same was reflected in the live feed frame of camera 2 in which the parking space box color turned to red as shown in Figure 49.

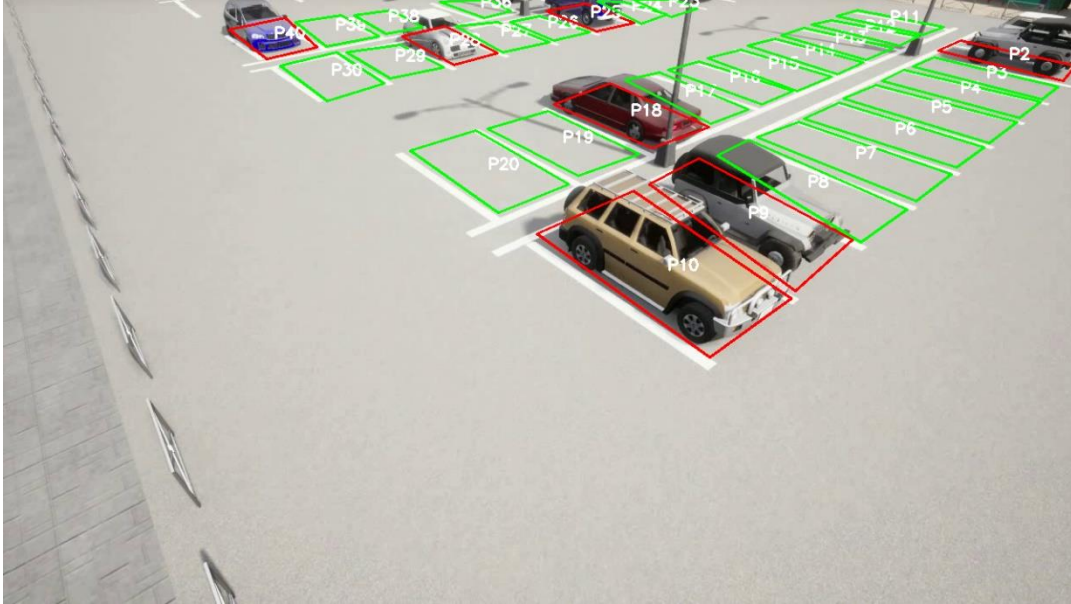


Figure 47: Corrected Final Occupancy Status of P2 in Camera 4 feed

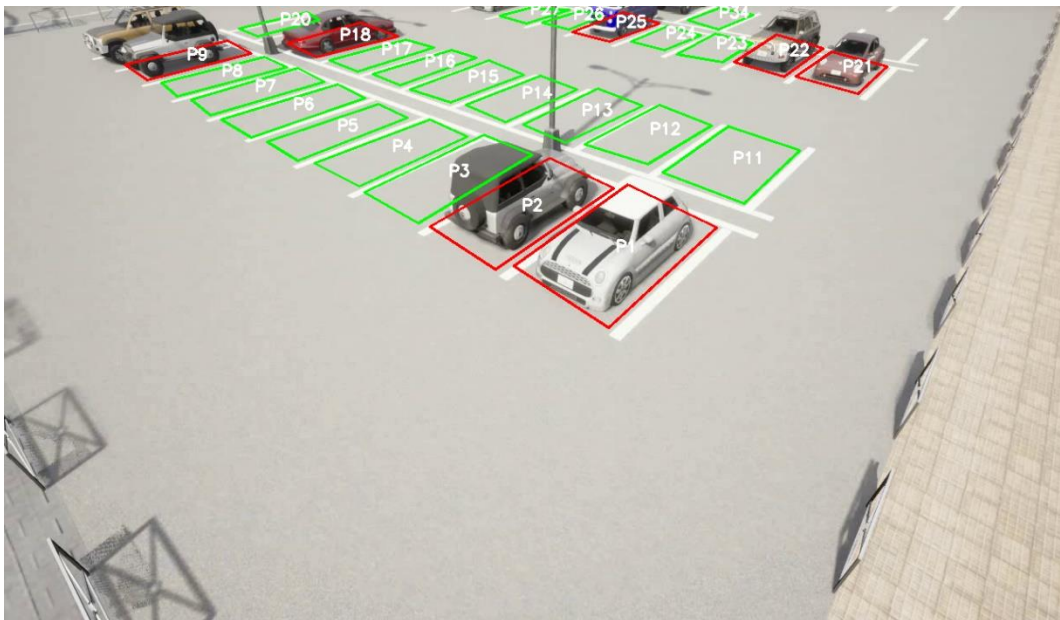


Figure 48: Corrected Final Occupancy Status of P9 in Camera 1 feed

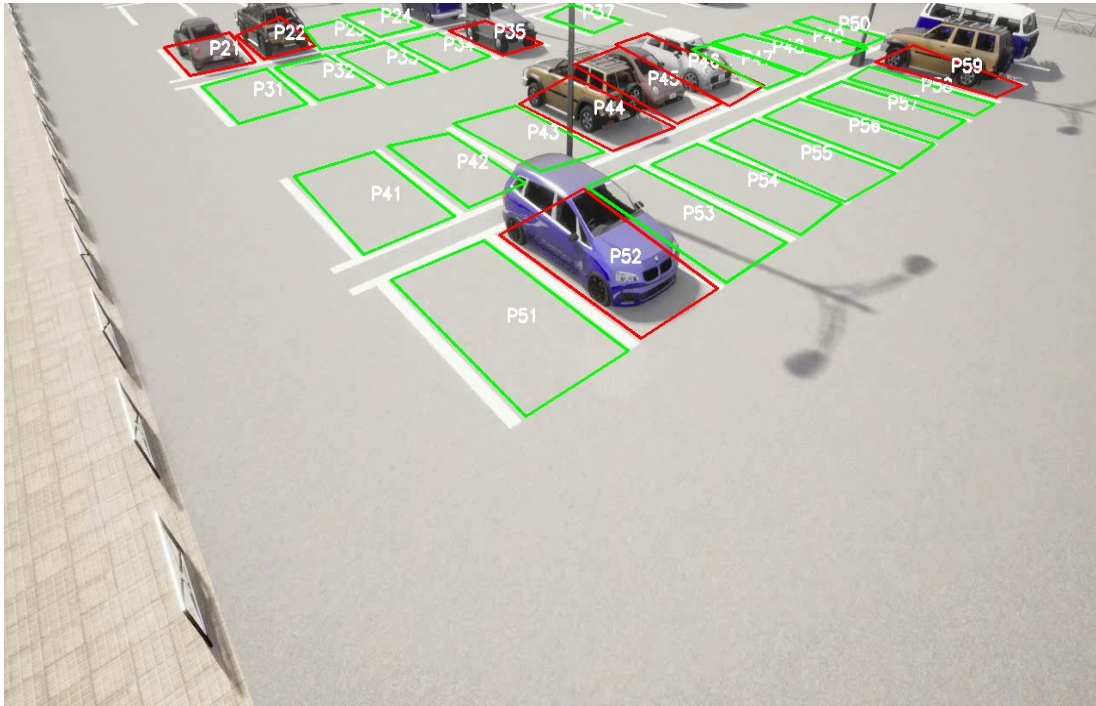


Figure 49: Corrected Final Occupancy Status of P21 in Camera 2 feed

The final occupancy status decider algorithm runs the IoU, distances, weights, and average weighted IoU calculations for all the parking spaces in the background. If a parking space is covered/monitored by more than 1 infrastructure camera, the average weighted IoU is determined and if the calculated value is greater than or equal to 0.30, the parking space's final occupancy status is determined as occupied. In the case of parking spaces covered or monitored by only 1 parking space, the IoU is calculated, if it is greater than or equal to 0.45, the parking space's final occupancy status is determined as occupied.

The final occupancy status decider algorithm runs in the background in real time and sends a list of the final occupancy status of all the parking spaces to the MongoDB database in real-time. This data is further fetched and displayed on the website interface for the users

who can make use of the navigation module to navigate to the parking spot recommended by the system or navigate to a spot of their choice.



## CONCLUSION & FUTURE WORK

This thesis project started with the main goal of tackling the pressing issue of parking which has become a constant source of frustration among drivers. Drivers often end up roaming endlessly in the parking lot trying to find a spot which leads to fuel wastage, increased harmful emissions, and wasted time. Distracted drivers in the parking lot searching for a spot also tend to get into traffic accidents in the parking lot. There is a great demand for a technological parking solution to eradicate all these issues and make parking more accessible for users. The literature review was carried out on the existing smart parking solutions. Most of the previous works/existing solutions focused on identifying whether the parking spaces in the parking lot were occupied/vacant and provided information about the number of available parking spaces to the users before entering the parking lot. However, this would still require the user to maneuver/circle the parking lot to identify the vacant spot. To overcome this, a novel Intelligent Parking Guidance System was proposed, and a prototype system was developed and validated. This prototype system uses a single infrastructure monocular camera to monitor a segment of the parking lot containing 20 parking spaces and identify the vacant & occupied parking spaces. Further, it identifies the location of all available parking spaces and provides a suggestion to the driver about the nearest vacant parking space amongst the list of all the available parking spaces. It also provides the user with the choice to choose a vacant parking space of their choice. Finally, the system provides the user with accurate navigation route guidance to the selected parking space with the option of rerouting to a new nearest open spot in case the user fails to follow the navigation route guidance. The system also provides the drivers with a

website interface using which the driver can monitor the occupancy status of the parking lot and see the live feed of the infrastructure camera. However, certain problems like occlusions and false positives/negatives exist in the case of the single infrastructure camera-based vacant parking space detection which reduces the overall efficiency of the intelligent parking guidance system. To upgrade the developed system to mitigate the above-mentioned problems, the concept of the multiple-infrastructure camera-based vacant parking space detection system was discussed and simulated in a simulation environment. This system increases the accuracy of parking space detection and improves the overall robustness of the proposed intelligent parking guidance system. Furthermore, future work could include developing a similar system for multi-level parking lot buildings. The same system can be implemented in harsh weather conditions and the robustness of the system can be validated under extreme rain/snow conditions.

## REFERENCES

- [1] M. Karthi and P. Harris, "Smart Parking with Reservation in Cloud Based Environment," in *IEEE International Conference on Cloud Computing in Emerging Markets (CCEM)*, Bangalore, 2016.
- [2] H. Wang and W. He, "A Reservation-based Smart Parking System," in *IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, Nebraska, 2011.
- [3] N. Mejri, M. Ayari, R. Langar and A. S. Leila, "Reservation-based Multi-Objective Smart Parking Approach for Smart Cities," in *IEEE International Smart Cities Conference (ISC2)*, Trento, 2020.
- [4] I. Aydin, M. Karakose and E. Karakose, "A navigation and reservation-based smart parking platform using genetic optimization for smart cities," in *5th International Istanbul Smart Grid and Cities Congress and Fair (ICSG)*, Istanbul, 2017.
- [5] S. U. Raj, M. V. Manikanta, P. S. S. Harsitha and M. J. Leo, "Vacant Parking Lot Detection System Using Random Forest Classification," in *3rd International Conference on Computing Methodologies and Communication (ICCMC)*, Erode, India, 2019.
- [6] A. Kianpisheh, "Smart Parking System (SPS) Architecture Using Ultrasonic Detector," *International Journal of Software Engineering and Its Applications*, 2012.
- [7] A. Kadir, M. M. Osman and M. N. Othman, "IoT based Car Parking Management System using IR Sensor," *Journal of Computing Research and Innovation*, p. 75–84, 2020.

- [8] T. Ibrahim, W. Shen and D. Soufiene, "An IoT-based Eco-Parking System for Smart Cities," in *2020 IEEE International Smart Cities Conference (ISC2)*, Piscataway, NJ, USA, 2020.
- [9] M. D. Yash, R. K. R. D and A. K. U, "Eco Park: A Low-Cost and Sustainable Approach for Smart Parking Systems," in *2023 World Conference on Communication & Computing (WCONF)*, Raipur, India, 2023.
- [10] B. Harshitha, L. Nathan, K. Naveen, M. Sangram, N. Sushma and L. Kaikai, "An Edge Based Smart Parking Solution Using Camera Networks and Deep Learning," in *2018 IEEE International Conference on Cognitive Computing (ICCC)*, San Francisco, CA, 2018.
- [11] S. Kunal, S. Aditya, S. Prasheel, S. Payal and S. R. Surendra, "Smart Parking Solutions for On-Street and Off-Street Parking," in *2021 International Conference on Communication information and Computing Technology (ICCICT)*, Mumbai, India, 2021.
- [12] H. Amira, B. E. Hadda and C. F. Lamia, "TinyML enabled smart parking dynamic slots computing and license plate recognition," in *2024 International Wireless Communications and Mobile Computing (IWCMC)*, Ayia Napa, Cyprus, 2024.
- [13] L. Wang, X. Zhang, W. Zeng, W. Liu, L. Yang and J. Li, "Global Perception-Based Robust Parking Space Detection Using a Low-Cost Camera," *IEEE Transactions on Intelligent Vehicles*, vol. 8, no. 2, pp. 1439 - 1448, 2023.
- [14] A. Rawat, Manisha, M. S, R. M. C and S. S, "Parking Space Detection Using Image Processing in MATLAB," *International Research Journal of Engineering and Technology (IRJET)* , vol. 5, no. 4, pp. 1289 - 1291, 2018.
- [15] J. Canny, "A Computational Approach to Edge Detection," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vols. PAMI-8, no. 6, pp. 679 - 698, 1986.

- [16] P. E. H. Richard O. Duda, "Use of the Hough transformation to detect lines and curves in pictures," *Communications of the ACM*, vol. 15, no. 1, pp. 11-15, 1972.
- [17] R. Patel and P. Meduri, "Car Detection Based Algorithm For Automatic Parking Space Detection," in *2020 19th IEEE International Conference on Machine Learning and Applications (ICMLA)*, Miami, FL, USA, 2020.
- [18] K. He, G. Gkioxari, P. Dollár and R. Girshick, "Mask R-CNN," in *2017 IEEE International Conference on Computer Vision (ICCV)*, Venice, Italy, 2017.
- [19] P. Koopman, "Bresenham line drawing," in *Forth Dimensions*, Redwoodcity, CA, M&T Books, 1987, pp. 12-16.
- [20] J. Redmon, S. Divvala, R. Girshick and A. Farhadi, "You Only Look Once: Unified, Real-Time Object Detection," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Las Vegas, NV, USA, 2016.
- [21] F. G. Y. Ye and O. Shiqi, "An Algorithm for Judging Points Inside or Outside a Polygon," in *2013 Seventh International Conference on Image and Graphics*, Qingdao, China, 2013.
- [22] A. Dincer and U. Balkan, *Google Maps Api Cookbook*, Packt Pub Ltd, 2013.
- [23] H. Viddit, "The Working of Google Maps, and the Commercial," *INTERNATIONAL JOURNAL OF INNOVATIVE RESEARCH IN TECHNOLOGY*, vol. 6, no. 5, pp. 185 - 191, 2019.
- [24] B. G. S. Grepon, J. P. Margallo, J. B. Maserin and R. A.-D. A. Dompol, "RUI: A Web-based Road Updates Information System using," *International Journal of Computing Sciences Research*, vol. 7, pp. 2253-2271, 2023.

- [25] H. Li and L. Zhijian, "The study and implementation of mobile GPS navigation system based on Google Maps," in *2010 International Conference on Computer and Information Application*, Tianjin, China, 2010.
- [26] A. Dosovitskiy, "CARLA: An Open Urban Driving Simulator," in *1st Conference on Robot Learning (CoRL 2017)*, Mountain View, CA, 2017.

## APPENDIX A

### PARKING SPACE LOCATION

Table IV: List of parking spots and their corresponding location coordinates

Parking Space	CARLA Location Coordinates
P1	(10.4, -43.9, 0)
P2	(10.4, -41.1, 0)
P3	(10.4, -38.3, 0)
P4	(10.4, -35.5, 0)
P5	(10.4, -32.8, 0)
P6	(10.4, -29.8, 0)
P7	(10.4, -27.3, 0)
P8	(10.4, -24.4, 0)
P9	(10.4, -21.5, 0)
P10	(10.4, -18.8, 0)
P11	(4, -43.9, 0)
P12	(4, -41.1, 0)
P13	(4, -38.3, 0)
P14	(4, -35.5, 0)
P15	(4, -32.8, 0)
P16	(4, -29.8, 0)
P17	(4, -27.3, 0)
P18	(4, -24.4, 0)
P19	(4, -21.5, 0)



P20	(4, -18.8, 0)
P21	(-6.3, -43.9, 0)
P22	(-6.3, -41.1, 0)
P23	(-6.3, -38.3, 0)
P24	(-6.3, -35.5, 0)
P25	(-6.3, -32.8, 0)
P26	(-6.3, -29.8, 0)
P27	(-6.3, -27.3, 0)
P28	(-6.3, -24.4, 0)
P29	(-6.3, -21.5, 0)
P30	(-6.3, -18.8, 0)
P31	(-12.6, -43.9, 0)
P32	(-12.6, -41.1, 0)
P33	(-12.6, -38.3, 0)
P34	(-12.6, -35.5, 0)
P35	(-12.6, -32.8, 0)
P36	(-12.6, -29.8, 0)
P37	(-12.6, -27.3, 0)
P38	(-12.6, -24.4, 0)
P39	(-12.6, -21.5, 0)
P40	(-12.6, -18.8, 0)
P41	(-22.4, -43.9, 0)

P42	$(-22.4, -41.1, 0)$
P43	$(-22.4, -38.3, 0)$
P44	$(-22.4, -35.5, 0)$
P45	$(-22.4, -18.8, 0)$
P46	$(-22.4, -29.8, 0)$
P47	$(-22.4, -27.3, 0)$
P48	$(-22.4, -24.4, 0)$
P49	$(-22.4, -21.5, 0)$
P50	$(-22.4, -18.8, 0)$
P51	$(-29.1, -43.9, 0)$
P52	$(-29.1, -41.1, 0)$
P53	$(-29.1, -38.3, 0)$
P54	$(-29.1, -35.5, 0)$
P55	$(-29.1, -32.8, 0)$
P56	$(-29.1, -29.8, 0)$
P57	$(-29.1, -27.3, 0)$
P58	$(-29.1, -24.4, 0)$
P59	$(-29.1, -21.5, 0)$
P60	$(-29.1, -18.8, 0)$

## APPENDIX B

### GITHUB REPOSITORY & FEEDBACK FORM

Source code on Github: <https://github.com/BELIV-ASU/Intelligent-Parking-System>

Final demo video and survey: <https://forms.gle/g1dfgib9h38JbwXT6>