Parking control system using computer vision and convolutional neural network models for object detection.

November 20, 2025

Abstract

The mismatch between the supply and demand for parking spots is a recurring problem for public street parking in places with high vehicle densities. Traditional parking meter systems fail to accurately align the paid time with the actual occupancy. To address this issue, we propose a computer vision–based management approach that estimates zone-level occupancy and enables charging based on actual usage time, without the need for intrusive hardware that could contribute to urban visual pollution. Commercial RTSP cameras provide visual data that are transmitted to edge processing devices responsible for real-time detection and tracking. This allows inference of individual space status, duration of presence, and anonymous usage records. The pilot design cover one zone and evaluates operational, financial, and urban impact outcomes.
Keywords— Yolo, parking, vision computing, camera, hough line.

1 Introduction

Over the years, there has been a quest to bring technology not only to factories, businesses, and medicine, but also to homes and streets. Industry 4.0 has brought with it an idea where technology lives among us and is also connected to transmit data, which ends up being processed in large computer complexes where, based on that process, necessary decisions are made and policies are implemented that can change the way we behave as a species living in large urban centers. The problems caused by excessive and unplanned urbanization and the economic growth of large metropolises, as well as their demographics since the Industrial Revolution in England, have been noted in recent years, with cities such as New York, Mexico City, and London, to name a few from the G20, ranking among the cities with the most traffic congestion. This is a problem that affects both the quality of life of citizens and the viability of the city itself. The existence of this problem has brought with it other problems, such as contributing to high levels of greenhouse gasses, as indicated by CO2 ppm indicators.

During the early 2000s, the rapid acceleration of machine learning models, such as advances in the field of perceptrons and what we now know as deep learning, and in the field of embedded systems focused on robotics and public use, has made the proposals of Industry 4.0 increasingly possible, bringing us closer to the idea of introducing them into people's everyday lives. The need for computational models that allow us to process large volumes of data and subsequently take action based on these data has grown, either due to specific problems, such as the aforementioned problem of excessive vehicle concentration. The demand for parking spaces exceeds the available supply in many cases in densely populated cities. It can be inferred that the relationship between availability and waiting time for a

vehicle can indirectly affect the increase in vehicle density on the streets instead of finding an available space.

The implementation of a system that allows drivers to know the availability of these spaces would reduce the number of cars circulating in search of a parking space and make this task more efficient. Integration of embedded systems that do not affect visual pollution in the streets is necessary because when looking for a solution that affects quality of life to some extent, aspects such as visual pollution must be taken into account.

Current object detection and computer vision models allow us to carry out several of the tasks necessary to implement a solution that seeks to be minimally invasive to the environment on a visual level, as it is a solution where many of the processes are carried out centrally and can use things already present on the streets themselves without the need to implement additional hardware in the field where it is to be deployed. In this work, we explore a computer vision-based solution that enables efficient and non-invasive urban parking management, using cameras already present in the environment to reduce congestion and improve the citizen experience.

2 Related Work

Various projects have been carried out that seek to address the aforementioned issues using multiple computer vision systems, sensors, and deep learning. [1] One example is the "A Distributed Markovian Parking Assist System" approach, which is based on vehicle and sensor networks, although it stands out for its distributed approach and relies on costly infrastructure that presents scalability challenges. [2] One of the most recent is the one developed in 2025 called On-Street "Parking Space Detection Using YOLO Models and Recommendations Based on KD-Tree Suitability Search," which uses Yolo to detect free parking spaces. [3] "Resource-Efficient Design and Implementation of Real-Time Parking Monitoring System with Edge Device" (2025) proposes a solution for integrating computer vision with vehicle networks to improve occupancy detection. Although its approach is more holistic, it emphasizes the need for joint integration between sensors and cameras. [4] "Parking Lot Occupancy Detection with Improved MobileNetV3" it proposes an improvement to the MobileNetV3 model for detecting parking space occupancy, where it is intended to be used in real time and with high accuracy in urban environments. The publication seeks to modify the architecture by incorporating a CBAM attention module and using separable convolutions of the blueprint type.
These studies agree on the need for a computer vision solution that can be implemented in urban areas. Although many models take different approaches, there is a clear way to implement the integration of computer vision models. It is important to note that several of these solutions seek to detect parking areas, which may be changed or affected by uncontrollable factors. Therefore, many of these models would lack the robustness to consider each of the scenarios or any type of changing scenario, such as a road closure or cut, and therefore do not adapt to the changing environment independent of them.

3 Technical Background

For the technical background, it is important to mention the main components of the solution. These are necessary for the efficient and effective implementation of the system.

3.1 CCN (Convolutional Neural Network)

CCN (Convolutional Neural Network) is a type of neural network designed for the analysis of data such as visual resources. They are useful for identifying patterns in images in order to recognize objects, classes, and categories. They are inspired by the visual cortex. Convolution applies a set of convolutional filters to the input images, and each filter activates different characteristics of the images. Unlike traditional neural networks, in CNN, in each convolutional layer, each filter has a unique set of weights and a bias per filter, shared spatially. Many object identification models, such as YOLO, are based on CNN in their structure.

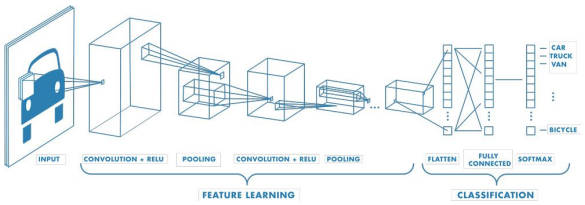

Figure 1: Architecture of a Convolutional Neural Network (CNN) [6]

3.2 Yolo (you only look once)

YOLO (You Only Look Once) is a real-time CNN-based object detection system that identifies objects in entire images. Instead of making multiple passes, it does it only once. This unified approach allows it to be significantly faster than other object detection models.

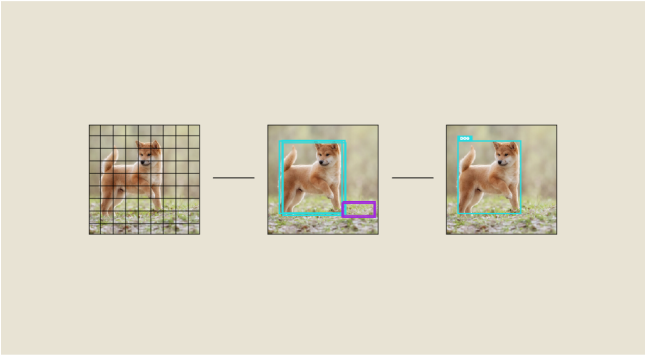

Figure 2: Illustration of the YOLO object detection pipeline.[8]

3.3 Hugh line transform

Hugh line transform is a technique for detecting parametric shapes that proceeds to find alignments in space in the same way as locating maxima in a parameter space. In the case of lines, each edge pixel proceeds to a “vote for all the lines that could pass through it,” and the aggregation of votes reveals dominant parameters.

Figure 3: Visual representation of Hough Transform applied to a highway scene.[7]

4 Method

4.1 Overview

The proposed system for parking occupancy detection is composed of four main modules:

- **Mapping Module** (`map_parking_zones.py`) – used to manually define and store parking zones as polygons within a video frame.
- **Detection Module** (`vehicle_detector.py`) – performs real-time detection of vehicles using a trained YOLO model and checks whether vehicles overlap with mapped parking zones.
- **Main Application** (`main.py`) – coordinates input video processing, loads zone configurations (JSON), and displays final results.
- **Editing Module** (`edit_parking_zones.py`) – allows modifications to previously saved zone mappings.

4.2 Algorithmic Methodology

Process	Description
Video Frame Capture	The video is read frame-by-frame using OpenCV.
Object Detection (YOLO)	Each frame is processed through the YOLO model to detect vehicles.
Geometric Intersection	The detected vehicle bounding boxes are checked for intersection with parking zone polygons (using OpenCV polygon functions).
Occupancy Decision	If a vehicle overlaps a polygon, the zone is marked as "occupied"; otherwise, "available".
Visualization & Export	Results are displayed in real-time with colored overlays and can be saved for further analysis.

Table 1: Algorithmic methodology for parking occupancy detection.

4.3 Multi-Zone Detection System Components

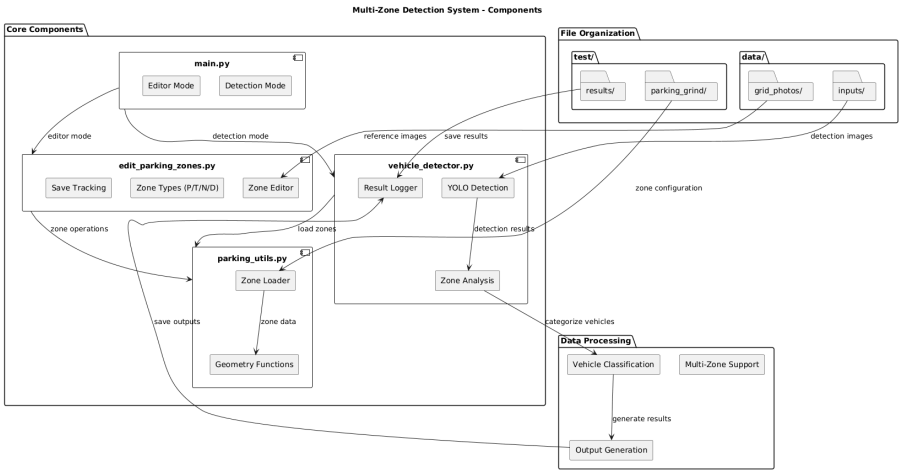

Figure 4: UML component diagram showing the structural elements and relationships of the multi-zone parking detection system.

4.4 Multi-Zone Parking Detection DataFlow

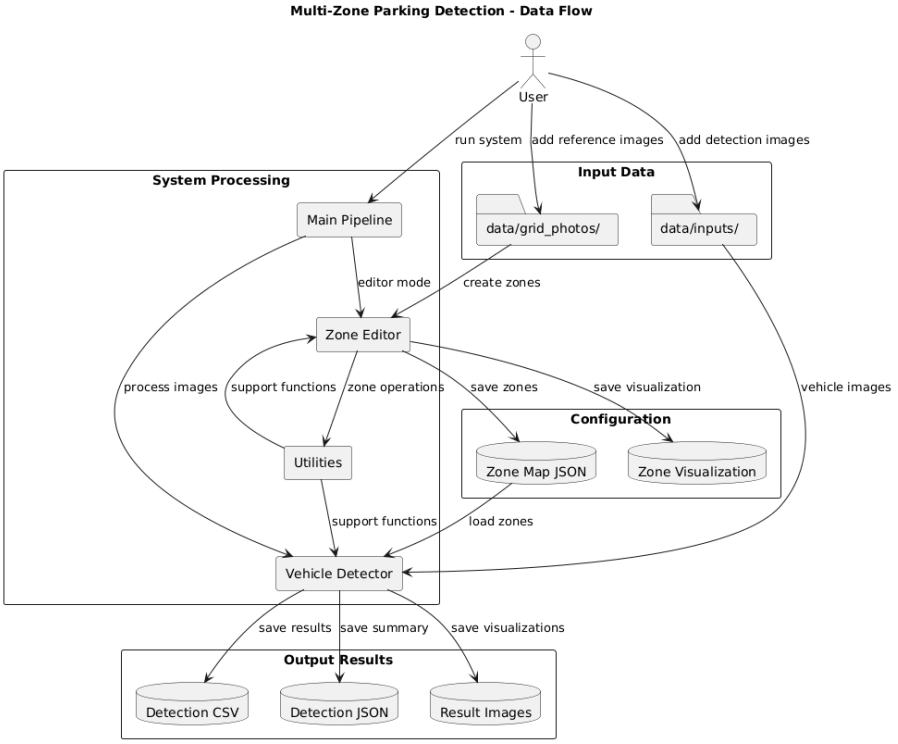

Figure 5: Data flow diagram illustrating the processing pipeline and information exchange within the multi-zone parking detection system.

4.5 Multi-Zone Parking Detection System Architecture

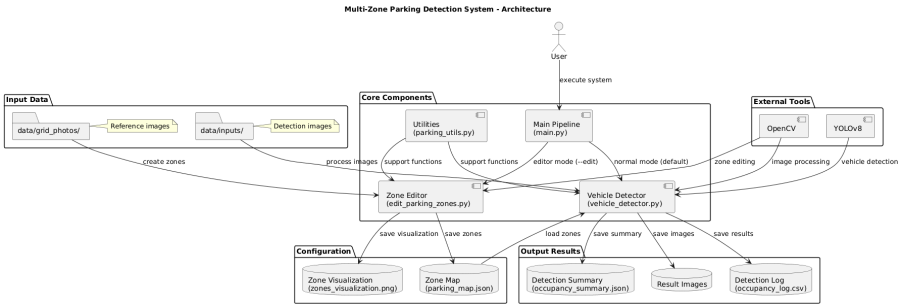

Figure 6: System architecture diagram depicting the high-level structure and component interactions of the multi-zone parking detection system.

5 Implementation

The vehicle detection system in parking lots was created using Python 3.10 or higher and various libraries, which are listed below.

1. `opencv-python>=4.8.0` – For image and video processing operations
2. `numpy>=1.24.0` – For numerical computations and array operations
3. `pandas>=2.0.0` – For data manipulation and CSV file handling
4. `ultralytics>=8.0.0` – For YOLO object detection
5. `torch>=2.0.0` – Required by Ultralytics for deep learning inference
6. `pillow>=10.0.0` – For image processing and manipulation
7. `matplotlib>=3.7.0` – For visualization and debugging tools
8. `scikit-learn>=1.3.0` – For evaluation metrics (confusion matrix, F1-score, precision, recall)
9. `seaborn>=0.12.0` – For enhanced confusion matrix heatmaps and statistical visualizations

Python was chosen for its ease of developing solutions related to artificial intelligence and image processing, as well as for its multiple libraries.

5.1 Core Technologies

The overall functioning of the project is based on a relatively small number of key libraries. Opencv is mainly responsible for image and/or video processing tasks such as frame reading, overlay drawing, and output management. For vehicle detection, ultralytics is used with its yolov11 version, which provides speed and accuracy. Numpy handles the numerical calculations required when it is necessary to calculate the overlaps between detected vehicles and parking areas. The parking area configurations are stored in JSON files, making it easy to edit them manually if required and to exchange them between different camera configurations.

5.2 System Architecture

The code has been organized into several modules to make it easier to add or correct different functions in the future. The `main.py` file is the main file where command line arguments are managed and mainly coordinates the overall workflow of the entire system. When it is necessary to configure a parking zone for the first time, `map_parking_zones.py` is used to generate a map of which areas are parking zones in the frame related to the camera's field of view. This ultimately generates a JSON file with all the coordinates of the zones.

For vehicle detection operations, `vehicle_detector.py` contains the functions that perform the work. Video frames, live camera feeds, or images are loaded, and YOLO is run on each frame to find vehicles, checking which parking zones the detected vehicles overlap with. If it is necessary to adjust the zones at any time due to a change in policy or an external event, they can be edited using `edit_parking_zones.py`, which provides an interactive editor where the zones can be drawn and modified visually.

The system processes files from two directories. Visual files with vehicles are stored in `data/inputs/`, while reference photos of the empty parking lot are stored in `data/grid_photos/`. When the main script is run, it automatically processes everything inside these folders and displays the results in `test/results/`.

5.3 How Detection Works

To steer each frame, Yolo "scans" the image and returns boxes delimited around any vehicle it detects. In this case, pre-trained models are being used that can recognize cars, trucks, motorcycles, and buses, which means that no customized training for vehicle steering has been necessary.

Once we have the bounding boxes for the vehicles, we compare them with the polygons assigned to the parking areas. For each area, we calculate the percentage of it that is covered by the detected vehicles. If the percentage exceeds the threshold of 60 percentage (set by default), the space is marked as occupied. If it is not marked as occupied, it remains free.

Different types of zones have been implemented to manage various parking scenarios. Normal parking spaces are marked as "P," traffic zones as "T," which are areas where vehicles are in transit, thus indicating that the vehicle is in transit and not parked; no-parking zones are marked as "N," and disabled/accessible zones as "D." Each type is shown in a different color on the display:

1. Green for free parking spaces.
2. Red for occupied spaces.
3. Orange for traffic zones.
4. Blue for accessible spaces.

5.4 Testing and Validation

For system testing, a controlled environment is required in the first instance in which the system can be systematically evaluated. Recording images or obtaining visual material would take time to obtain

for various reasons, as it would require requesting images from authorities, which are only permitted to be released in exceptional cases.

Instead, Unity Engine will be used to create a parking scenario. This would allow us to build a 3D parking environment and place and arrange vehicles in specific positions to test different scenarios that we might encounter in real images. In Unity, you can adjust camera angles, change lighting to simulate different times of day, and control exactly how many vehicles there would be and where they would be located.

Twenty different test scenarios will be generated. Some will have empty spaces, while others will be occupied and have different vehicles in different positions. Each scenario will be saved as an image that will be placed in `data/inputs/`. When executed, `main.py` will automatically process the 20 images through the address channel. For each test image, a visualization will be generated showing the detected vehicles and the status of the parking area. The records will be saved in csv with detailed detection data and the summary in JSON format.

5.5 Privacy

In different regions and countries, privacy laws can be quite strict regarding data handling and taking photographs, such as of users or even their vehicles, for example. Given the growing concern about data handling and privacy, a post-processing step is carried out when processing the information in which vehicles are detected and a blur is applied to censor the detected vehicle. In this way, characteristics such as the model and even the make of the vehicle remain unknown. An example applied to privacy laws is in the European Union. The GDPR (General Data Protection Regulation) [9] came into force in May 2018, which establishes that images captured by video surveillance security cameras are considered personal data if they allow individuals to be identified directly or indirectly.

6 Testing

6.1 Test area

Validation strategy: metrics, experiments, results. As mentioned in the previous point, the tests will be carried out in Unity engine. As shown in the following image, a parking space with 30 available spaces is used, which can be modified to see how it works and evaluate the system's capabilities. The 30-space parking lot can be considered a 5x6 matrix in which we can arrange the available spaces and thus organize them for evaluation. We obtain an n31 space, which is the transit zone. This zone is considered separately for the evaluation, as it is important to consider several additional aspects of the context. The area was manually prepared using the tools prepared for the solution.

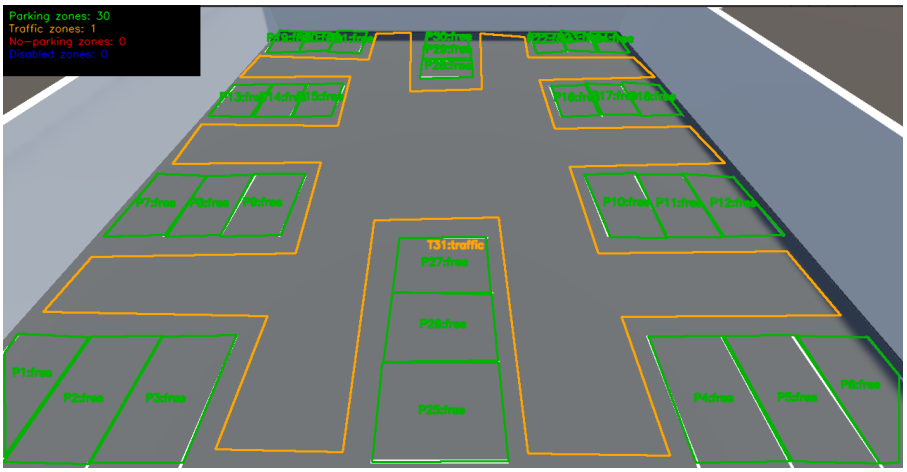

Figure 7: Visual distribution of parking spaces in Unity engine

The parking configuration is represented by the set λ, which contains all available parking spaces in the parking area. Each element $p_i \in \lambda$ represents the specific location of a parking space. For a conceptual example in this work, the parking area consists of $N = 30$ parking spaces.

$$\lambda = \{p_1, p_2, \ldots, p_{30}\}. \tag{1}$$

In addition, parking spaces also take into account existing traffic zones. These zones are grouped together in the set of traffic zones.

$$T = \{t_1, t_2, \ldots, t_k\}, \tag{2}$$

where each element $t_i \in T$ represents a different transit zone. Although transit zones are not parking spaces, they are included in the analysis due to their relevance to the behavior of the system itself.

Since parking spaces and transit zones are different types of entities, no matrix operation is defined between them. Instead, the system operates on the underlying sets and defines the general configurations of the study area as a conceptual union.

$$\Lambda = \lambda \cup T. \tag{3}$$

6.2 Data collected

Column	Type	Description
ID	String	Test identifier from filename (e.g., `test_001`); matches with test configuration
source_file	String	Input filename (e.g., `test_001.png`)
ts_utc	ISO 8601	UTC timestamp in ISO 8601 format with Z suffix
frame	Integer	Frame number (always 1 for images)
t_ms	Integer	Milliseconds from start (always 0 for images)
spot_id	String	Parking spot ID (P1–Pn) or `UNASSIGNED`
status	String	Smoothed occupancy: `free`, `occupied`, `in_traffic_zone`, `in_no_parking_zone`, `partially_in_parking_zone`, `illegally_parked`
raw_status	String	Immediate status before temporal smoothing
overlap	Float	Parking zone coverage by vehicle (0.0–1.0)
traffic_overlap	Float	Traffic zone overlap (0.0–1.0); ≥ 0.1 indicates blocking
vehicle	String	Vehicle type: `car`, `truck`, `bus`, `motorcycle`; empty if free
vehicle_id	Float	Unique vehicle identifier; empty if free
det_conf	Float	YOLOv11 confidence score (0.0–1.0); empty if free
type	String	`parking_spot` or `unassigned_vehicle`

Table 2: CSV Log File Column Definitions

6.3 Metrics

The main objective is to evaluate the functioning of the system through testing, not the YOLO object detection model. Using metrics such as precision, recall, or F1 score would not allow us to fully understand the functioning of the system, since the testing area is a virtual environment with certain realistic characteristics that are not present at the image level. In this case, heat maps will be used to detect where there are errors in detecting certain expected vehicles vs. detected vehicles in different areas.

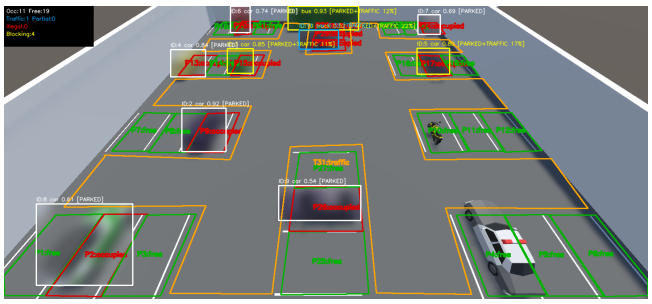

Figure 8: Demonstration of system deployment.

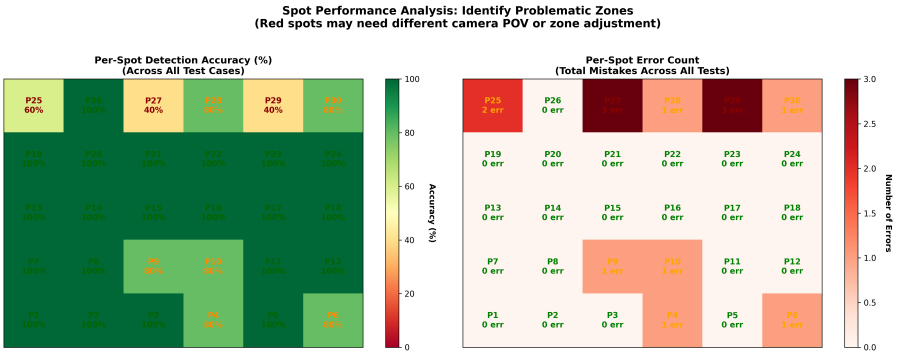

Figure 9: Performance analysis heat map.

6.4 Analysis

Five photos were taken in which the vehicles were placed in a concentrated manner, as on the right, left, scattered, and diagonal. Although the problem may lie in the detection of the vehicles and the model itself applied in the simulation environment, the results show that the vehicles located in the central area based on the viewpoint present a detection problem, especially those at the back, which are represented as p29 and p30. This would indicate that, as they are located in a central area and a central viewpoint with a perspective towards the center, the objects tend to be smaller and their detectability decreases.

The system fulfills the main idea of developing a system that detects vehicles, knows what type of vehicles are there, and additionally indicates which ones are obstructing traffic. It is important to consider the model that will be implemented in future versions. Additionally, the incorporation of second observation points, such as other cameras, would allow for the reevaluation of detections and thus make the system more reliable.

7 Conclusion

7.1 Limitations

The limitations that the system may present are varied and all relate to the model's detection capabilities. Although models such as YOLO could be considered robust enough to carry out detection in a virtual environment that can be considered to be of lower graphic quality than more realistic simulations, over the years it may be necessary to retrain the models as car designs become obsolete and concepts change over time. Another limitation of the system is in the hypothetical case of finding only a single point of view, which would not allow vehicles hidden by other objects or even other vehicles to be observed.

7.2 Future work

The substantial improvement is the incorporation of an evaluation method where additional observation points can be added, thereby reducing the number of undetected objects that are actually present. This would make the system more robust to external factors such as backlighting, blocking, or even malfunction. By maintaining an already implemented structured data system, the only requirement is the consideration of additional observation points. Another improvement could be the incorporation of segmentation detection, which would reduce errors in relation to parking spaces that are not actually caused by the observation point. Finally, the last improvement is the consideration of existing systems, such as data obtained by sensors already incorporated in the parking area. This would allow the system to be more accurate in obtaining the final data required, such as whether a space is occupied or not, when external factors are taken into account.

7.3 Final conclusion

Although the project has been implemented for a parking lot, the conceptual idea is clear and allows it to be applied in different environments, such as public avenues and parking lots. Although it does not seek to solve the larger problem, which is the high demand for parking spaces, the impact can be positive as it seeks to contribute to significantly reducing the problem. It also seeks to be a non-invasive solution, using the means already established in cities.

References

[1] Liu, M., Naoum-Sawaya, J., Gu, Y., Lecue, F., & Shorten, R. (2019). A distributed Markovian parking assist system. *IEEE Transactions on Intelligent Transportation Systems, 20*(6), 2230–2240. https://doi.org/10.1109/TITS.2018.2865648

[2] Kim, J., Jeong, I., Jung, J., & Cho, J. (2025). Resource-efficient design and implementation of real-time parking monitoring system with edge device. *Sensors, 25*(7), 2181. https://doi.org/10.3390/s25072181

[3] Garta, I. Y., Manongga, W. E., Huang, S.-W., & Chen, R.-C. (2025). On-street parking space detection using YOLO models and recommendations based on KD-tree suitability search. *Computers, Materials & Continua, 85*(3), 4458–4471. https://doi.org/10.32604/cmc.2025.067149

[4] Yuldashev, Y., Mukhiddinov, M., Abdusalomov, A. B., Nasimov, R., & Cho, J. (2023). Parking lot occupancy detection with improved MobileNetV3. *Sensors, 23*, 7642. https://doi.org/10.3390/s23177642

[5] Microsoft Azure. (n.d.). ¿Qué es Computer Vision? *Microsoft Azure — Cloud Computing Dictionary*. Retrieved November 6, 2025, from https://azure.microsoft.com/es-es/resources/cloud-computing-dictionary/what-is-computer-vision/

[6] MathWorks. (n.d.). ¿Qué son las redes neuronales convolucionales? *MATLAB & Simulink — Discovery*. Retrieved November 6, 2025, from https://es.mathworks.com/discovery/convolutional-neural-network.html

[7] OpenCV. (n.d.). Hough Line Transform. *OpenCV 4.x Documentation*. Retrieved November 6, 2025, from https://docs.opencv.org/4.x/d6/d10/tutorialpyhoughlines.html

[8] DataScientest. (2024, March 28). You Only Look Once (YOLO): What is it? *DataScientest*. Retrieved from https://datascientest.com/en/you-only-look-once-yolo-what-is-it

[9] European Parliament. (2022). Video protection policy of the European Parliament. *Directorate-General for Security and Safety*. https://www.europarl.europa.eu/at-your-service/files/stay-informed/security-and-access/es-cctv-ep-video-protection-policy.pdf

[10] Barredez, C. (2025). *Barredez Carlos 4251122 – Computer Science Project* [GitHub repository]. GitHub. https://github.com/Cbarredeziu/Barredez-Carlos$_4$251122$_{project-Computer-Science-Project}$