Smart Parking System

A Report Presented to the
Computer Engineering Department in Partial
Fulfillment of Requirements for Feedback and Control Systems

Submitted by:
Jao, Jaxie H.

CPE 3144 5-432

Submitted to:
Dr. Eleonor V. Palconit

December 2024

Introduction

Background of the study

The rapid growth of urban areas has led to increased vehicle usage, creating a demand for more efficient and user-friendly parking solutions. Traditional parking systems often result in time-consuming searches for vacant spots, leading to traffic congestion, wasted fuel, and increased carbon emissions. In fact, according to Mark Braibanti [1], drivers waste over 55 hours annually searching for parking, with up to 30 percent of traffic in urban areas resulting from vehicles circling the block in search of parking places. Smart Parking Systems have become a viable answer to these problems since they use modern technology to maximize the process of locating and controlling parking spaces.

A Smart Parking System uses sensors, cameras, and digital platforms to provide real-time information on the availability of parking spaces. These systems enable drivers to quickly locate vacant spots, improving the overall parking experience while reducing the time and energy spent searching. Additionally, it offers potential benefits to city planners and parking operators by providing valuable data on parking patterns and utilization. It also directly impacts climate change by reducing the time cars spend searching for parking, contributing to a sustainable future on this planet [2].

This paper aims to understand the feedback and control systems in implementing a Smart Parking System. It will study the inputs from various sensors, the processing of this data, and the final outputs generated by the system. By analyzing its effectiveness, user experience, and practical challenges, this paper can recommend and provide further improvements in developing parking solutions that are both more efficient and sustainable.

Objective:

This study aims to design and simulate a smart parking system that can monitor and detect vehicles to determine the availability of parking spaces using image or camera-based vehicle detection and occupancy feedback mechanisms. The study seeks:

1. To simulate a parking layout and implement a system to detect vehicle locations and occupancy status.
2. To develop a feedback mechanism that updates parking availability in response to changes in parking space occupancy.
3. To provide a user-friendly and visual output display of any available and occupied parking spaces for the user's convenience.

This objective supports both the technical goals of creating a reliable parking system simulation and the practical aim of improving real-time data accuracy for parking availability.

Related Research Literature

With rising traffic and high demand for parking, cities have turned to new parking management solutions. Smart Parking Systems offer an innovative way to tackle these issues by using sensors, data analysis, and mobile apps to improve parking convenience and efficiency. This section reviews existing research to understand the current state of Smart Parking, looking at core technologies, user experiences, environmental benefits, data processing, and the main challenges and opportunities. The studies discussed here provide a basis for examining the feedback and control systems that make Smart Parking effective.

Some studies with smart parking technology have focused on systems that connect users with real-time parking information. For instance, a study by Elsonbaty and Shams [3] explored using Arduino sensors to link users with the parking area, sending live updates to a mobile app for easy access, which introduced the Internet of Things (IoT) system. According to their study, finding a parking spot in their Smart Parking Systems involves three stages. First, the parking area uses Arduino devices and sensors to check space availability. In the second stage, a WiFi module connects cloud services to the parking area and processes data for the user. Finally, users receive notifications about available spaces through a mobile app for real-time updates.

Another similar study by Chaurasia K., Shahare M., et al. [4] also discusses using Arduino sensors, such as infrared sensors, alongside a mobile application for the Smart Parking System. However, they implemented a QR code reader for parking spaces to enable reservations, easy vehicle entry, and exit, and reduced the need for tickets or paperwork, especially on a paid parking system.

In another approach, camera sensors detect cars in parking spaces by collecting data through inverse perspective mapping (IPM), which allows cameras to capture an aerial view of the parking area. For object detection, the YOLOv5 model (a computer vision model designed for identifying objects) is applied to analyze camera images, identifying details such as colors, locations, and so on. This model processes each camera frame to detect bounding boxes around vehicles in specific locations. The information gathered then feeds into a guidance system that directs users to available parking spaces or notifies them when parking is full [5].

Camera sensors are also used for data collection on car detection in parking spaces. YOLO, a single-shot object recognition algorithm, quickly identifies objects without losing accuracy. It can handle tasks like detecting vehicles, pedestrians, license plates, and traffic signs. YOLO operates as an end-to-end object detection network that processes camera frames to detect bounding boxes for vehicles in specific locations. The information gathered from the camera is sent to a guidance system, which accurately identifies parking spaces occupied by vehicles and highlights available spots for users [6].

Overall, the studies reviewed show that Smart Parking Systems can be used with sensors, mobile apps, and camera-based detection, which can significantly improve parking efficiency and user experience. Real-time data and guidance systems are provided via Arduino sensors, IoT, QR codes, and image processing models like YOLOv5 to locate open parking spots. By reducing traffic from finding free spots and its environmental impact, these technologies can meet the increasing demand for convenient parking.

Methodology

In this study, the Smart Parking System is designed to monitor the occupancy of parking spaces using cameras placed above the parking area. The cameras can detect vehicles and free spaces through real-time footage or captured images. An object detection model such as YOLO (You Only Look Once) can be applied to the captured images to identify vehicles. YOLO works by dividing an image into a grid, with each grid cell responsible for detecting objects within its area. This approach enables real-time detection of objects, allowing the system to track which parking spots are occupied or free [7].

To simulate the system, MATLAB coding was utilized to replicate the process of inserting and removing vehicles within a parking lot. This simulation also allowed the monitoring of free spaces and the corresponding LED indicators. When a vehicle is inserted in the simulation, its coordinates are registered, and the parking space is marked as occupied. Conversely, when a vehicle is removed, the system updates the status of that parking space to free.

The system continuously updates the occupancy status of each parking space as the cameras receive new data. So, when a vehicle is detected, the system registers the parking spot as occupied. The space is marked as a free space if no vehicle is detected. This real-time data will go to the guidance system that will inform the users of any available parking spaces. The guidance system uses LED displays that change colors to indicate whether a space is available on the lanes (green) or are all occupied (red).

Additionally, the system also includes a feedback mechanism to monitor and update parking spaces' status continuously. When a vehicle enters or leaves its parking space, the system adjusts the occupancy information accordingly. This helps the users to always have access to accurate, real-time details about available parking spaces, making the process more efficient and easier to navigate.

The MATLAB code used for the simulation, which includes the insertion and removal of vehicles, monitoring of free spaces, and controlling the LEDs, is included below to provide a detailed view of the implementation.

“Main Code”

```
scenario = drivingScenario;
scenario.SampleTime = 0.2;
% Define the roads and parking lot as before
roadCenters = [69.2 11.7 0; -1.1 11.5 0];
marking = [laneMarking("Solid")
        laneMarking("DoubleSolid", Color=[1 0.9 0])
        laneMarking("Solid")];
laneSpecification = lanespec(2, Width=5.925, Marking=marking);
road(scenario, roadCenters, Lanes=laneSpecification, Name="Road");
roadCenters = [12.4 7.7 0; 12.4 -15.8 0];
road(scenario, roadCenters, Name="Road1");
roadCenters = [50 7.7 0; 50 -15.8 0];
road(scenario, roadCenters, Name="Road2");
lot = parkingLot(scenario, [3 -5; 60 -5; 60 -45; 3 -45]);
% Create the parking spaces
cars = parkingSpace;
accessible = parkingSpace(Type="Accessible");
accessibleLane = parkingSpace(Type="NoParking", MarkingColor=[1
1 1], Width=1.5);
fireLane = parkingSpace(Type="NoParking", Length=2, Width=40);
% Insert the parking spaces and keep track of the indices
insertParkingSpaces(lot, cars, 13, Edge=2, Offset=3); % Top edge
insertParkingSpaces(lot, cars, 13, Edge=4, Offset=3); % Bottom edge
insertParkingSpaces(lot, cars ,9,Rows=2,Position=[42 -12]);
insertParkingSpaces(lot, cars ,9,Rows=2,Position=[23 -12]);
insertParkingSpaces(lot, fireLane, 1, Edge=3, Offset=8); % Right edge
% Define positions of parking spaces for each section
parkingSpacePositions = struct( ...
  'top', [55.5, -9.6, 0; 55.5, -12.3, 0; 55.5, -15, 0; 55.5, -17.7, 0; 55.5, -
20.4, 0; 55.5, -23.2, 0; 55.5, -25.9, 0; 55.5, -28.7, 0; 55.5, -31.5, 0;
55.5, -34.4, 0; 55.5, -37, 0; 55.5, -39.9, 0; 55.5, -42.5, 0], ...
  'midupup', [43.5, -13.4, 0; 43.5, -16.2, 0; 43.5, -19, 0; 43.5, -21.8, 0;
43.5, -24.4, 0; 43.5, -27.2, 0; 43.5, -30, 0; 43.5, -32.7, 0; 43.5, -35.5,
0], ...
  'midupdown', [38, -13.4, 0; 38, -16.2, 0; 38, -19, 0; 38, -21.8, 0; 38, -
24.4, 0; 38, -27.2, 0; 38, -30, 0; 38, -32.7, 0; 38, -35.5, 0], ...
  'middownup', [24.5, -13.4, 0; 24.5, -16.2, 0; 24.5, -19, 0; 24.5, -21.8,
0; 24.5, -24.4, 0; 24.5, -27.2, 0; 24.5, -30, 0; 24.5, -32.7, 0; 24.5, -35.5,
0], ...
  'middowndown', [19, -13.4, 0; 19, -16.2, 0; 19, -19, 0; 19, -21.8, 0;
19, -24.4, 0; 19, -27.2, 0; 19, -30, 0; 19, -32.7, 0; 19, -35.5, 0], ...
  'down', [4.7, -7.5, 0; 4.7, -10.5, 0;4.7, -13.2, 0; 4.7, -15.9, 0; 4.7, -
18.6, 0; 4.7, -21.4, 0; 4.7, -24.1, 0; 4.7, -26.9, 0; 4.7, -29.7, 0; 4.7, -
32.4, 0; 4.7, -35.1, 0; 4.7, -37.9, 0; 4.7, -40.6, 0 ] ...
);
```

```
occupancy = struct( ...
  'top', zeros(size(parkingSpacePositions.top, 1), 1), ...
  'midupup', zeros(size(parkingSpacePositions.midupup, 1), 1), ...
  'midupdown', zeros(size(parkingSpacePositions.midupdown, 1), 1), ...
  'middownup', zeros(size(parkingSpacePositions.middownup, 1), 1), ...
  'middowndown', zeros(size(parkingSpacePositions.middowndown, 1),
1), ...
  'down', zeros(size(parkingSpacePositions.down, 1), 1) ...
);

function occupancy = insertVehicle(scenario, position,
parkingSpacePositions, occupancy)
  ego = vehicle(scenario, 'ClassID', 1, 'Position', position);

  % Loop through each section of parking spaces
  fields = fieldnames(parkingSpacePositions);
  for i = 1:length(fields)
    section = fields{i};  % Get the section name (e.g., 'top',
'midupdown', etc.)
    % Loop through each parking space in the section
    for j = 1:size(parkingSpacePositions.(section), 1)
      % Use a tolerance to compare the position with the parking
space
      tolerance = 1e-4;  % Adjust as needed
      if all(abs(parkingSpacePositions.(section)(j, :) - position) <
tolerance)
        % Check if the parking space is vacant
        if occupancy.(section)(j) == 0
          % Mark as occupied
          occupancy.(section)(j) = 1;
          return;  % Exit function after inserting the vehicle.
        end
      end
    end
  end
end
 function vehicleObj = getVehicleByPosition(scenario, position)
  tolerance = 1e-4;
  vehicles = scenario.Actors;  % Get all actors in the scenario

  vehicleObj = [];  % Initialize as empty

  % Loop through all vehicles in the scenario
  for i = 1:length(vehicles)
    if all(abs(vehicles(i).Position - position) < tolerance)
      vehicleObj = vehicles(i);  % Found the vehicle
      break;
    end
  end
end
end
```

```
function updatedOccupancy = insertVehicle(scenario, position,
parkingSpacePositions, occupancy)
   % Function to insert a vehicle into a parking space
   tolerance = 0.5;
   found = false;
   for i = 1:length(parkingSpacePositions)
       if occupancy(i)
           % Skip already occupied spaces
           continue;
       end
       if norm(parkingSpacePositions(i, :) - position) < tolerance
           % Insert a vehicle at the matched parking space
           vehicleObj = vehicle(scenario, 'ClassID', 1); % Create a vehicle
           vehicleObj.Position = parkingSpacePositions(i, :);
           occupancy(i) = true;
           found = true;
           break; % Exit the loop once a match is found
       end
   end
   if ~found
       error('No available parking space near the specified position.');
   end
   % Return the updated occupancy
   updatedOccupancy = occupancy;
end

function occupancy = exitVehicle(vehicleObj, parkingSpacePositions,
occupancy)
   % Find the index of the parking space the vehicle is occupying
   found = false;
   sections = {'top', 'midupup', 'midupdown', 'middownup', 'middowndown',
'down'};
   for i = 1:length(sections)
       section = sections{i};
       % Loop through each parking space in the section
       for j = 1:size(parkingSpacePositions.(section), 1)
           % Compare the position of the vehicle to the parking space position
           tolerance = 1e-4;
           if all(abs(parkingSpacePositions.(section)(j, :) - vehicleObj.Position) <
tolerance)
               % Mark the parking space as empty
               occupancy.(section)(j) = 0;
               disp(['Vehicle removed from ', section, ' section, spot ', num2str(j)]);
               vehicleObj.Position = [0 8 0];
               found = true;
               break;
           end end
       if found
           break;
       end  end
   if ~found
       disp('Vehicle not found at the specified position.');
   end
end

while true
   % Prompt the user for input
   disp('Enter the command as "insert: [x, y, z]" or "remove: [x, y, z]", or type
"exit" to stop:');
   userInput = input('Command: ', 's');

   % Check if the user wants to exit
   if strcmpi(userInput, 'exit')
       disp('Exiting the interactive vehicle management process.');
       break;
   end

try
   if startsWith(userInput, 'insert:', 'IgnoreCase', true)
       % Extract position for insertion
       positionStr = strtrim(extractAfter(userInput, 'insert:'));
       position = str2num(positionStr); %#ok<ST2NM>

       if numel(position) ~= 3
           error('Position must be a 3-element vector [x, y, z].');
       end
```

```
           % Insert the vehicle
    occupancy = insertVehicle(scenario, position, parkingSpacePositions,
occupancy);
       disp('Vehicle inserted successfully.');

   elseif startsWith(userInput, 'remove:', 'IgnoreCase', true)
   % Extract position for removal
   positionStr = strtrim(extractAfter(userInput, 'remove:'));
   position = str2num(positionStr);

   % Check if the position is a 3-element vector
   if numel(position) ~= 3
       error('Position must be a 3-element vector [x, y, z].');
   end
   vehicleObj = getVehicleByPosition(scenario, position);  % Custom
function to get vehicle based on position

   % Check if a vehicle was found
   if ~isempty(vehicleObj)
       % Call the exitVehicle function to remove the vehicle
       occupancy = exitVehicle(vehicleObj, parkingSpacePositions,
occupancy);  % Exit the vehicle from parking space
       disp('Vehicle removed successfully.');
   else
       disp('No vehicle found at the specified position.');
   end
else
   error('Invalid command. Use "insert:", "remove:", or "exit".');
end
  % Display status
       freeTop = sum(occupancy.top == 0);
       freeMidupup = sum(occupancy.midupup == 0);
       freeMidupdown = sum(occupancy.midupdown == 0);
       freeMiddownup = sum(occupancy.middownup == 0);
       freeMiddowndown = sum(occupancy.middowndown == 0);
       freeDown = sum(occupancy.down == 0);

       freeTopSection = freeTop + freeMidupup;
       freeMiddleSection = freeMidupdown + freeMiddownup;
       freeBottomSection = freeMiddowndown + freeDown;

       topLEDStatus = 'OFF';
       middleLEDStatus = 'OFF';
       bottomLEDStatus = 'OFF';

       if freeTopSection == 0
           topLEDStatus = 'RED LIGHT';
       else
           topLEDStatus = 'GREEN LIGHT';
       end

       if freeMiddleSection == 0
           middleLEDStatus = 'RED LIGHT';
       else
           middleLEDStatus = 'GREEN LIGHT';
       end
       if freeBottomSection == 0
           bottomLEDStatus = 'RED LIGHT';
       else
           bottomLEDStatus = 'GREEN LIGHT';
       end

       disp(['Total free (Top Section): ', num2str(freeTopSection)]);
       disp(['Total free (Middle Section): ', num2str(freeMiddleSection)]);
       disp(['Total free (Bottom Section): ', num2str(freeBottomSection)]);
       disp(['Top LED : ', topLEDStatus]);
       disp(['Middle LED : ', middleLEDStatus]);
       disp(['Bottom LED : ', bottomLEDStatus]);
       plot(scenario);

   catch ME
       disp(['Error: ', ME.message]);
   end
end
plot(scenario);
```

Conceptual Framework

Various components work together in the Smart Parking System to create an efficient parking management process. The system begins with camera sensors that detect vehicles and identify their positions in the parking area. This data is sent to a simulation developed in MATLAB, where it is processed to calculate the occupancy of each parking lane. The system determines the number of available parking slots by comparing the total spaces with the occupied ones. A feedback mechanism then updates this information in real-time and sends it to a guidance system. The guidance system, represented by LEDs, will light up a green color when there are free spaces available in each lane and turn red if not, providing users with a clear visual for easy navigation. This interaction between vehicle detection, data processing, feedback, and guidance enables the system to optimize parking operations and improve user convenience.

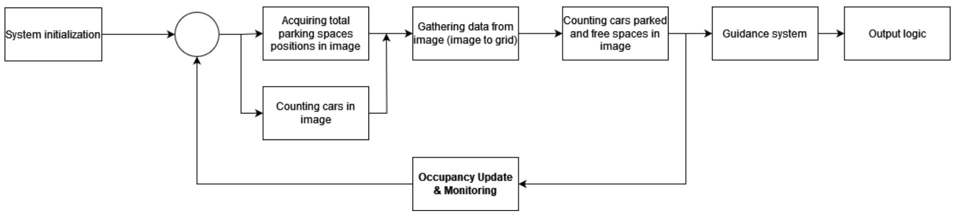

System initialization
Acquiring total parking spaces positions in image
Counting cars in image
Gathering data from image (image to grid)
Counting cars parked and free spaces in image
Guidance system
Output logic
Occupancy Update & Monitoring

Figure 1. Block Diagram of the smart parking system

Figure 1 shows the block diagram of the Smart Parking System, outlining the key components and their interactions. It illustrates how the system detects vehicle occupancy in parking spaces and provides real-time updates to users for better parking management.

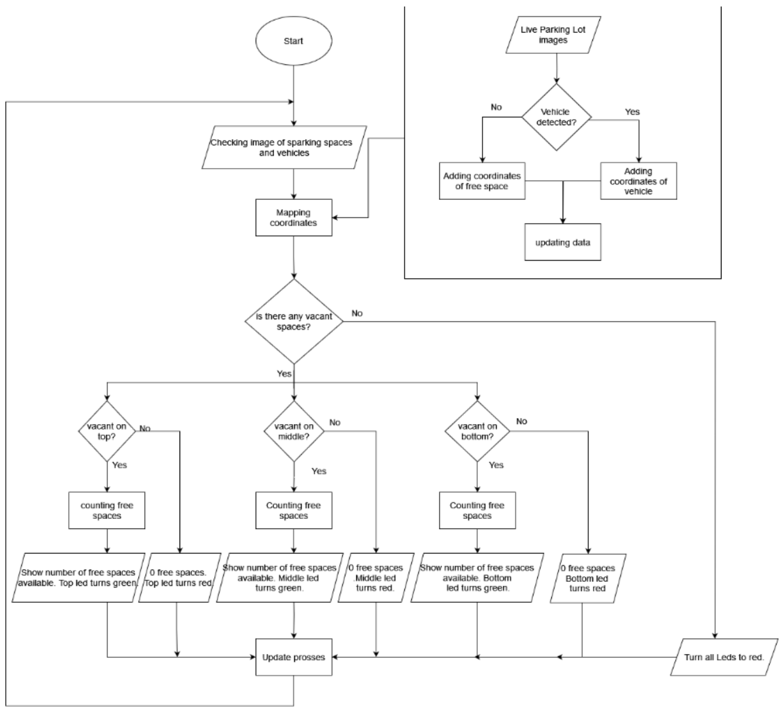

Start
Checking image of sparking spaces and vehicles
Mapping coordinates
Live Parking Lot images
Vehicle detected?
No
Yes
Adding coordinates of free space
Adding coordinates of vehicle
updating data
Is there any vacant spaces?
No
Yes
vacant on top?
No
Yes
counting free spaces
Show number of free spaces available. Top led turns green.
0 free spaces. Top led turns red.
vacant on middle?
No
Yes
Counting free spaces
Show number of free spaces available. Middle led turns green.
0 free spaces Middle led turns red.
vacant on bottom?
No
Yes
Counting free spaces
Show number of free spaces available. Bottom led turns green.
0 free spaces Bottom led turns red
Update prosses
Turn all Leds to red.

Figure 2. Flowchart of the smart parking system

Figure 2 shows the flowchart of each step of the Smart Parking System and how the feedback check enables continuous updates to the occupancy information by rechecking occupancy each time a change in vehicle presence is detected.

Mathematical Solutions

The mathematical solution in this study is based on the block diagram of the Smart Parking System. Each step and feedback process in the diagram is converted into a function to represent how the system works. The block diagram is then turned into a signal flow graph to show the flow of data and relationships within the system.

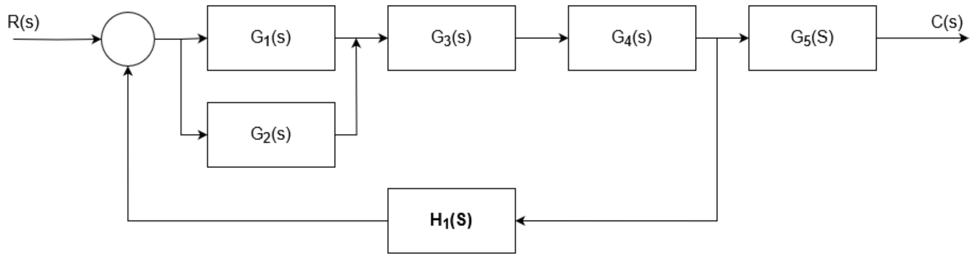


Figure 3. Block Diagram function of the smart parking system

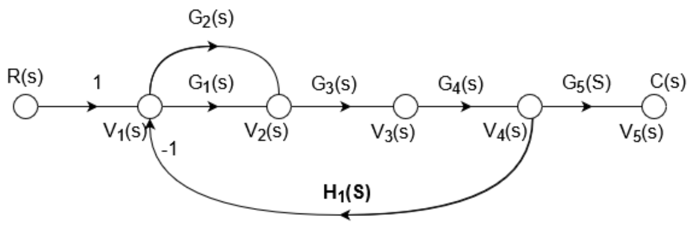
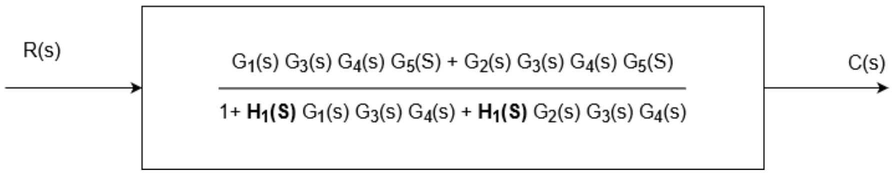


Figure 4. Signal flow of the smart parking system

The signal flow graph is analyzed using Mason's Gain Rule [8] to find the system's transfer function. This transfer function is a simple way to describe how the system processes inputs, like detecting vehicles, and produces outputs, such as updating parking availability and guiding users.

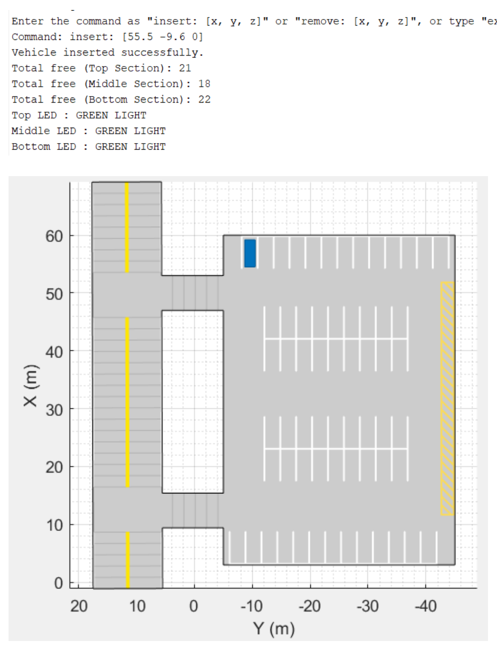


$$\frac{G_1(s)\,G_3(s)\,G_4(s)\,G_5(S) + G_2(s)\,G_3(s)\,G_4(s)\,G_5(S)}{1+ \mathbf{H_1(S)}\,G_1(s)\,G_3(s)\,G_4(s) + \mathbf{H_1(S)}\,G_2(s)\,G_3(s)\,G_4(s)}$$

Figure 5. Transfer function of the smart parking system

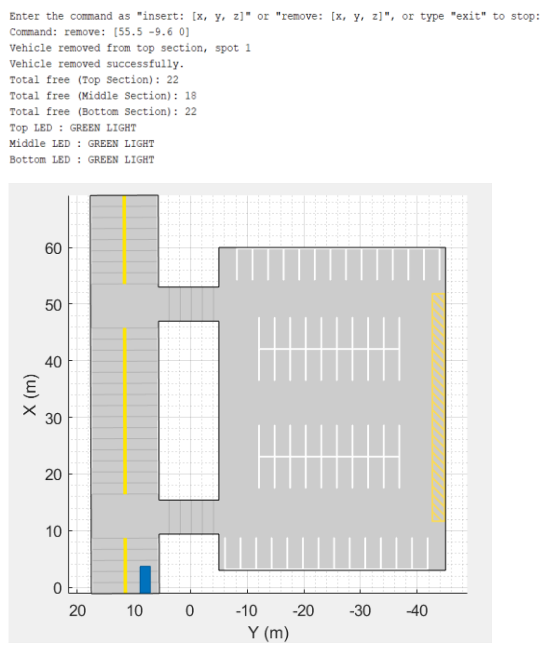
Illustration (Simulation)

The simulation in this study uses MATLAB's drivingScenario [8] plot to model the parking lot. In this plot, different colored boxes represent vehicles, while the entire plot space represents the parking spaces. The driving scenario plot uses a bird's-eye view of the parking lot, allowing for a clear visualization of the vehicle occupancy and free parking spaces.



```
Enter the command as "insert: [x, y, z]" or "remove: [x, y, z]", or type "exit" to stop:
Command: insert: [55.5 -9.6 0]
Vehicle inserted successfully.
Total free (Top Section): 21
Total free (Middle Section): 18
Total free (Bottom Section): 22
Top LED : GREEN LIGHT
Middle LED : GREEN LIGHT
Bottom LED : GREEN LIGHT
```

Figure 6. inserting vehicles



```
Enter the command as "insert: [x, y, z]" or "remove: [x, y, z]", or type "exit" to stop:
Command: remove: [55.5 -9.6 0]
Vehicle removed from top section, spot 1
Vehicle removed successfully.
Total free (Top Section): 22
Total free (Middle Section): 18
Total free (Bottom Section): 22
Top LED : GREEN LIGHT
Middle LED : GREEN LIGHT
Bottom LED : GREEN LIGHT
```

Figure 7. removing vehicles

The camera system is assumed to insert and remove vehicles from the parking view dynamically. This capability updates the top, middle, and bottom lanes' free space statuses in real-time. When a vehicle is added or removed, the camera system provides the location of the parking space occupied or emptied. For example, when a vehicle is inserted, the system may register its coordinates as [55.5, -9.6, 0]. These updates are integral to the feedback mechanism of the system, ensuring accurate monitoring of parking space availability.

```
Total number of free parking spaces (Top Section): 0
Total number of free parking spaces (Middle Section): 0
Total number of free parking spaces (Bottom Section): 0

Top LED : RED LIGHT
Middle LED : RED LIGHT
Bottom LED : RED LIGHT
```

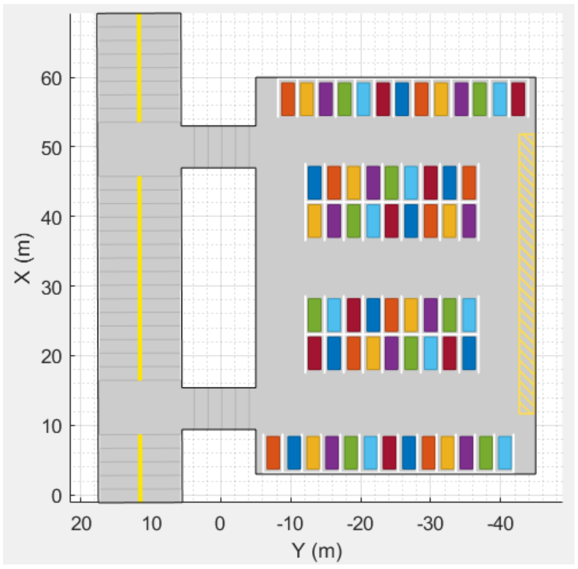


Figure 8. First simulation

```
Total number of free parking spaces (Top Section): 4
Total number of free parking spaces (Middle Section): 0
Total number of free parking spaces (Bottom Section): 0

Top LED : GREEN LIGHT
Middle LED : RED LIGHT
Bottom LED : RED LIGHT
```

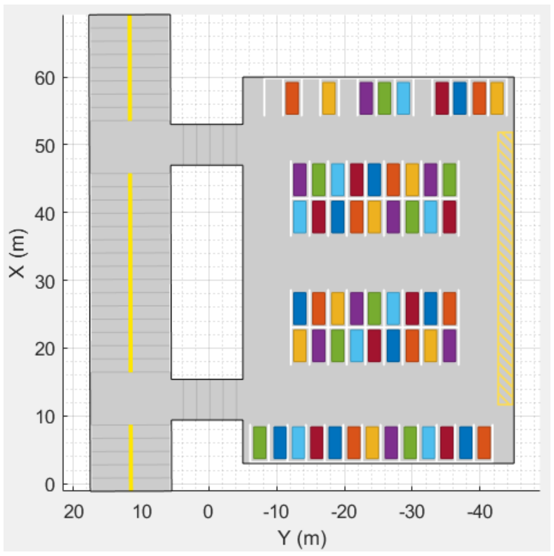


Figure 9. Second simulation

```
Total number of free parking spaces (Top Section): 22
Total number of free parking spaces (Middle Section): 18
Total number of free parking spaces (Bottom Section): 22

Top LED : GREEN LIGHT
Middle LED : GREEN LIGHT
Bottom LED : GREEN LIGHT
```

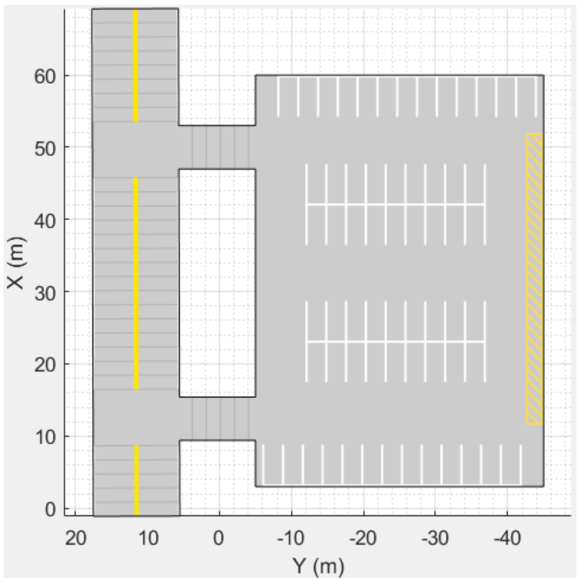


Figure 10. Third simulation

```
Total number of free parking spaces (Top Section): 4
Total number of free parking spaces (Middle Section): 5
Total number of free parking spaces (Bottom Section): 6

Top LED : GREEN LIGHT
Middle LED : GREEN LIGHT
Bottom LED : GREEN LIGHT
```

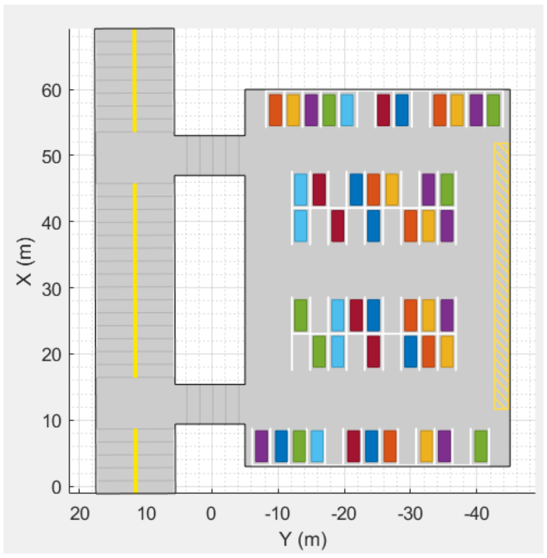


Figure 11. Forth simulation

In the first simulation, all parking spaces are occupied, with a text display indicating 0 free parking spaces in all lanes. All the LEDs turn red, signaling that no parking spaces are available. The second simulation shows that only the top parking lane has free spaces, while the middle and bottom lanes remain fully occupied. In this scenario, the LED for the top lane turns green, this shows that there are available spaces, while the LEDs for the middle and bottom lanes remain red to indicate full occupancy. The text display shows the number of free spaces in the top lane and 0 free spaces in the other two lanes. In the third simulation, all parking spaces are empty, and the LEDs for all lanes turn green to show that all parking spaces are available. The text display also indicates that all spaces are free. The fourth simulation features a mix of random vehicle occupancy across the lanes. Some spaces are occupied while others are free, with the text display showing the number of free spaces in each lane. In this case, the LEDs for the occupied lanes remain red, and the LEDs for the free spaces turn green.

Results & Discussion

The simulation of the Smart Parking System successfully shows an efficient way to manage parking spaces. Vehicles are automatically identified through a camera-based detection system, and their positions are mapped within the parking lot. The system updates the status of parking spaces in real-time, marking occupied spaces when a vehicle is detected and labeling unoccupied spaces as available. A continuous feedback mechanism also plays a role in maintaining up-to-date parking records when a vehicle enters or leaves the lot, providing a clear view of parking occupancy. This information integrates with a guidance system that uses LED indicators to visually display available spaces so that users can locate free spaces more quickly.

To validate the theoretical design, the system was modeled mathematically. Using a block diagram as a foundation, the simulation was transformed into a signal flow graph, and Mason's Gain Rule was applied to derive the system's transfer function. Although the simulation assumed vehicle detection through technologies like YOLO, the study focused on the system's capability to update parking status.

Conclusion

The Smart Parking System simulation demonstrates an effective solution for managing parking availability. The study showcases how real-time feedback and guidance mechanisms can be used by parking operations by simulating the process of detecting vehicles and updating parking space statuses. Using an LED-based guidance system makes use of easy navigation for drivers, while the mathematical modeling validates the system's theoretical performance.

While the simulation assumes the camera detection methods, it is more focused on the practical application of real-time updates and user-friendly outputs. This approach not only improves parking efficiency but also offers insights for scaling the system to accommodate larger or more complex parking areas.

Recommendation/Outlook/Future Studies

The Smart Parking System works well for open parking lots like airports or supermarkets. However, it could face challenges in indoor or basement parking lots, especially with navigation, due to limited camera views or signal interference. To solve this, adding more cameras or using physical sensors to cover blind spots could improve the system.

A mobile app could also be helpful, allowing drivers to check parking availability before arriving. However, an app might only work for one parking lot at a time. To improve this, the app could be updated to show multiple parking lots, making it easier for drivers to choose a spot. Overall, expanding this system to include app integration and multi-camera setups could further improve parking efficiency and user experience.

References

[1] M. Braibanti, "Driving the parking industry into the 21st century.," inrix, 2017. [Online]. Available: https://www2.inrix.com/parking/SmarterParking-2/article. [Accessed 9 November 2024].

[2] Cleverciti, "Cleverciti," Cleverciti, 2024. [Online]. Available: Smart parking. https://www.cleverciti.com/en/smart-parking. [Accessed 16 November 2024].

[3] A. A. &. S. M. Y. Elsonbaty, "The Smart Parking Management System. researchgate.," researchgate, September 2020. [Online]. Available: https://www.researchgate.net/publication/344411337_The_Smart_Parking_Management_System. [Accessed 16 November 2024].

[4] K. e. Chaurasia, "Review on Smart Parking Systems," irejournals, February 2021. [Online]. Available: https://www.irejournals.com/formatedpaper/1702598.pdf. [Accessed 16 November 2024].

[5] B. e. a. Liu, "Camera-Based Smart Parking System Using Perspective Transformation," researchgate, April 2023. [Online]. Available: https://www.researchgate.net/publication/370126214_Camera-Based_Smart_Parking_System_Using_Perspective_Transformation. [Accessed 16 November 2024].

[6] G. e. a. Peng, "Video-Based Parking Occupancy Detection for Smart Control System," Researchgate, February 2020. [Online]. Available: https://www.researchgate.net/publication/339096892_Video-Based_Parking_Occupancy_Detection_for_Smart_Control_System. [Accessed 16 November 2024].

[7] A. e. a. Pawar, "Elaborative Study of Smart Parking Systems. IJERT – International Journal of Engineering Research & Technology," ijert, 10 October 2021. [Online]. Available: https://www.ijert.org/research/an-elaborative-study-of-smart-parking-systems-IJERTV10IS100056.pdf. [Accessed 16 November 2024].

[8] N. S. Nise, "," in *Control Systems Engineering Sixth Edition*, California State Polytechnic University, Pomona, John Wiley & Sons, Inc., 2010, pp. 235 - 300.

[9] Mathworks, "Simulate vehicle parking maneuver in driving scenario. MathWorks - Maker of MATLAB and Simulink - MATLAB & Simulink," Mathworks, 2024. [Online]. Available: https://www.mathworks.com/help/driving/ug/simulate-vehicle-parking-maneuver-in-driving-scenario.html. [Accessed 16 November 2024].