

AI ENABLED CAR PARKING USING OPENCV

PROJECT REPORT

Submitted by:-

1. P N J D M Prakash(21bce8942)
2. Charan Adimalla (21bce9482)
3. C. Naga Vardhan(21bce7773)
4. T. Harsha Vardhana Reddy(21bce2249)

Project Report Format

1. INTRODUCTION

- 1.1 Project Overview
- 1.2 Purpose

2. IDEATION & PROPOSED SOLUTION

- 2.1 Problem Statement Definition
- 2.2 Empathy Map Canvas
- 2.3 Ideation & Brainstorming
- 2.4 Proposed Solution

3. REQUIREMENT ANALYSIS

- 3.1 Functional requirement
- 3.2 Non-Functional requirements

4. PROJECT DESIGN

- 4.1 Data Flow Diagrams
- 4.2 Solution & Technical Architecture
- 4.3 User Stories

5. CODING & SOLUTIONING (Explain the features added in the project along with code)

- 5.1 Feature 1
- 5.2 Feature 2
- 5.3 Database Schema (if Applicable)

6. RESULTS

- 6.1 Performance Metrics

7. ADVANTAGES & DISADVANTAGES

8. CONCLUSION

9. FUTURE SCOPE

10. APPENDIX

- Source Code
- GitHub & Project Video Demo Link

1. INTRODUCTION

1.1 Project Overview

Currently, the majority of AI-enhanced car parking systems rely on manual management, lacking an automated monitoring system to assess the capacity of each parking space. The absence of real-time tracking often forces drivers to navigate through parking lots extensively to locate an available spot. This issue is particularly prevalent in areas with a high demand for parking, such as near hospitals, malls, schools, and other densely populated locations. The need for a more efficient and automated approach to parking management is evident in these scenarios.

1.2 Purpose

The purpose of using AI-enabled car parking systems with OpenCV (Open Source Computer Vision) is to automate and optimize the process of parking vehicles in a parking lot or garage. By employing computer vision techniques and AI algorithms, these systems can efficiently detect and track vehicles, analyze parking space availability, and guide drivers to vacant spots.

2. IDEATION & PROPOSED SOLUTION

2.1 Problem Statement Definition

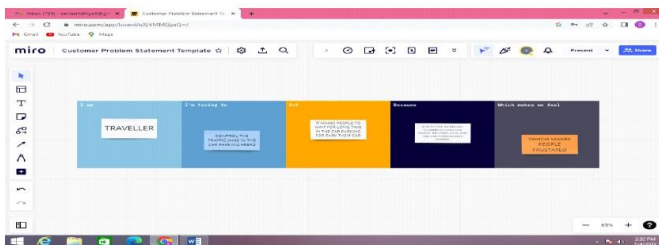
Customer Problem Statement template:

Problem statement 1 :

Consequently, once a car enters a parking garage followed by a parking space, a ping ultrasonic sensor will then be able to determine if a car is parked in the space or not. This information would then be relayed to update the network.

Problem statement 2 :

Avoid excessive parking supply. Use Parking Management to encourage more efficient use of existing parking facilities and address any spill over problems that result from pricing. Develop Transportation Management Associations to provide parking management and brokerage services in a particular area.



Problem Statement (PS)	I am (Customer)	I'm trying to	But	Because	Which makes me feel
PS-1	Traveller	Control the traffic in car parking	It makes more time	Due to the increased number of cars	Frustrated
PS-2	Traveller	Make the traveller to feel automated	It makes more time	Increased traffic	Frustrated

2.2 Empathy Map Canvas

Empathy Map Canvas:

An empathy map is a simple, easy-to-digest visual that captures knowledge about a user's behaviours and attitudes.

It is a useful tool to helps teams better understand their users.

Creating an effective solution requires understanding the true problem and the person who is experiencing it. The exercise of creating the map helps participants consider things from the user's perspective along with his or her goals and challenges.

EXAMPLE:




2.3 Ideation & Brainstorm

Brainstorm & Idea Prioritization Template:

Brainstorming provides a free and open environment that encourages everyone within a team to participate in the creative thinking process that leads to problem solving. Prioritizing volume over value, out-of-the-box ideas are welcome and built upon, and all participants are encouraged to collaborate, helping each other develop a rich amount of creative solutions.

Use this template in your own brainstorming sessions so your team can unleash their imagination and start shaping concepts even if you're not sitting in the same room.

Step-1: Team Gathering, Collaboration and Select the Problem Statement



Brainstorm & idea prioritization

Use this template in your own brainstorming sessions so your team can unleash their imagination and start shaping concepts even if you're not sitting in the same room.

- 10 minutes to prepare
- 1 hour to collaborate
- 2-8 people recommended

Before you collaborate

A little bit of preparation goes a long way with this session. Here's what you need to do to get going.

10 minutes

- Team gathering**
Define who should participate in the session and send an invite. Share relevant information or pre-work ahead.
- Set the goal**
Think about the problem you'll be focusing on solving in the brainstorming session.
- Learn how to use the facilitation tools**
Use the facilitator statements to run a happy and productive session.

[Open article](#)

Define your problem statement

What problem are you trying to solve? Frame your problem as a how might we statement. This will be the focus of your brainstorm.

5 minutes

review

How might we [your problem statement]?

Key rules of brainstorming

To run a smooth and productive session

- Stay in topic.
- Encourage wild ideas.
- Defer judgment.
- Listen to others.
- Go for volume.
- If possible, be visual.

Step-2: Brainstorm, Idea Listing and Grouping

2

Brainstorm

Write down any ideas that come to mind that address your problem statement.

🕒 10 minutes

Person 1

Develop AI model to solve parking problem in complex areas

Using cameras and sensors to detect the place accurately

Making the vehicle flow easier

Updating and tracking the moments

Person 3

Finding space quicker in hurry situations

Create identity for every car in parking space

Tracking the moments of vehicles

Analyzing the patterns and rush in the area

Improve user experience and using area efficiently

Person 2

Studying the traffic behaviour

Ensuring safety measure to avoid congestion

Accurately checking parameters

Less fuel consumption in indoor parking areas

Person 4

Less fuel consumption ways

Support and feedback from people and implementing them

Updating of places accurately without errors

Optimized tools to reduce cost

Step-3: Group Ideas

3

Group ideas

Take turns sharing your ideas while clustering similar or related notes as you go. Once all sticky notes have been grouped, give each cluster a sentence-like label. If a cluster is bigger than six sticky notes, try and see if you and break it up into smaller sub-groups.

20 minutes

TIP

Add customizable tags to sticky notes to make it easier to find, browse, organize, and categorize important ideas as themes within your mural.



Step-4: Prioritization

4

Prioritize

Your team should all be on the same page about what's important moving forward. Place your ideas on this grid to determine which ideas are important and which are feasible.

20 minutes

Tip

Participants can use their centers to point at where ideas were placed on the grid. The facilitator can confirm the point by using the rear camera holding the iPad on the keyboard.

Importance

If one of these tasks could get done without any difficulty or cost, which would have the most positive impact?

Feasibility

Regardless of their importance, which tasks are more feasible than others? Cost, time, effort, complexity, etc.

Optimize AI models to solve parking problem by computer vision

Tracking the movements of vehicles

Freeing space quicker in busy situations

Updated plans accurately reflect errors

Using computer vision to detect the park accurately

Accurately choosing parameters

Create identity for every car in parking space

Making the vehicle flow easier

Optimized roads to reduce costs

5

After you collaborate

You can export the mural as an image or pdf to share with members of your company who might find it helpful.

Quick add-ons

Share the mural

Share a view link to the mural with stakeholders to keep them in the loop about the outcomes of the session.

Export the mural

Export a copy of the mural as a PNG or PDF to attach to emails, include in slides, or save in your drive.

Keep moving forward

Strategy blueprint

Define the components of a new idea or strategy.

Open the template

Customer experience journey map

Understand customer needs, motivations, and obstacles for an experience.

Open the template

Strengths, weaknesses, opportunities & threats

Identify strengths, weaknesses, opportunities, and threats (SWOT) to develop a plan.

Open the template

about:blank

9/43

2.4 Proposed Solution

S.No.	Parameter	Description
1.	Problem Statement (Problem to be solved)	Parking systems have long been a source of frustration for drivers due to their inefficiencies and time-consuming nature. Fortunately, the emergence of smart parking solutions has revolutionized the traditional parking experience. These advanced systems integrate cutting-edge technologies to streamline the process, enhance user convenience, and alleviate common challenges associated with parking.
2.	Idea / Solution description	The concept of an AI-powered car parking system built on OpenCV revolves around harnessing the capabilities of computer vision to streamline the parking experience. This innovative approach employs camera technology and sophisticated image processing algorithms to identify the presence of vehicles within the parking area, subsequently assisting drivers in locating vacant parking spots. Moreover, the system is designed to automate ticketing and payment procedures, effectively expediting the entire process and enhancing convenience for drivers.
3.	Novelty / Uniqueness	Utilizing computer vision technology in car parking systems represents a groundbreaking strategy with the potential to notably enhance the effectiveness and user-friendliness of parking facilities. With the integration of OpenCV, the system can precisely identify and

AI Enabled CAR Parking Using OPEN CV

4.	Social Impact / Customer Satisfaction	Implementing an AI-driven car parking system with OpenCV can yield substantial societal benefits, notably in the realm of alleviating traffic congestion and enhancing the quality of the driving experience. Through its capacity to direct drivers to available parking spots, the system effectively curtails the duration spent searching for parking, consequently mitigating congestion within parking lots. Furthermore, by automating ticketing and payment operations, the system streamlines the parking process, fostering enhanced convenience and operational efficiency for users.
5.	Business Model (Revenue Model)	The business strategy for an AI-fueled car parking system can revolve around a pay-per-use framework, necessitating drivers to remit fees commensurate with their parking duration. Seamlessly fused with a payment gateway, the system facilitates swift and hassle-free online transactions, heightening convenience for users. Moreover, the system can augment its revenue streams by compiling and analyzing data on parking behaviors and occupancy levels. This valuable data can then be leveraged by parking lot operators to refine and optimize the overall parking experience.
6.	Scalability of the Solution	Expanding the scope of an AI-driven car parking system leveraging OpenCV is seamlessly achievable, catering to extensive parking lots and diverse geographical locations. This scalability is facilitated through the implementation of a centralized database and cloud-based infrastructure, enabling the system to efficiently manage substantial data volumes and serve numerous concurrent users. Furthermore, the system's adaptability is exemplified by its capacity for customization, allowing it to be tailored to the unique demands of each parking facility. This flexibility ensures its seamless integration into various environments and facilitates its alignment with specific use cases.

3. REQUIREMENT ANALYSIS

3.1 Functional requirement

Functional Requirements:

Following are the functional requirements of the proposed solution.

FR No.	Functional Requirement (Epic)	Sub Requirement (Story / Sub-Task)
FR-1	User authentication	Capture and store user's face image Authenticate user's identity based on face recognition Allow access to parking lot upon successful authentication

FR-2	Parking spot management	Detect availability of parking spots using camera feed Assign parking spot to the user Update parking spot availability in real-time
FR-3	Payment and billing	Calculate parking fee based on parking duration Allow payment via various modes like credit card, mobile wallet
FR-4	Security and surveillance	Monitor parking lot using cameras for suspicious activities Alert security personnel in case of any security breach
FR-5	System maintenance and support	Provide regular maintenance and updates to the system Offer customer support for any issues faced by users

3.2 Non-Functional requirements

Non-functional Requirements:

Following are the non-functional requirements of the proposed solution.

FR No.	Non-Functional Requirement	Description
NFR-1	Usability	The system should be easy to use and user-friendly, providing clear instructions to drivers on where to park and displaying available spots in an intuitive manner.
NFR-2	Security	The system should be secure and able to prevent unauthorized access to the parking lot. It should also protect the data collected by the system, ensuring that it cannot be accessed by unauthorized parties.

NFR-3	Reliability	The system must be reliable and accurate in detecting and identifying available parking spots. It should also be able to identify and prevent unauthorized access to the parking lot.
NFR-4	Performance	The system must be fast enough to detect and identify available parking spots in real-time, allowing drivers to quickly find a parking spot. It should also be able to handle multiple vehicles entering and exiting the parking lot simultaneously.
NFR-5	Availability	The system should be compatible with different types of vehicles, including cars, trucks, and motorcycles, and be able to accommodate different sizes of vehicles.
NFR-6	Scalability	The system should be able to handle a large number of parking spaces, and be easily scalable to accommodate additional spaces in the future.

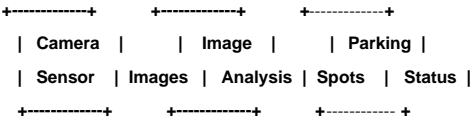
4. PROJECT DESIGN

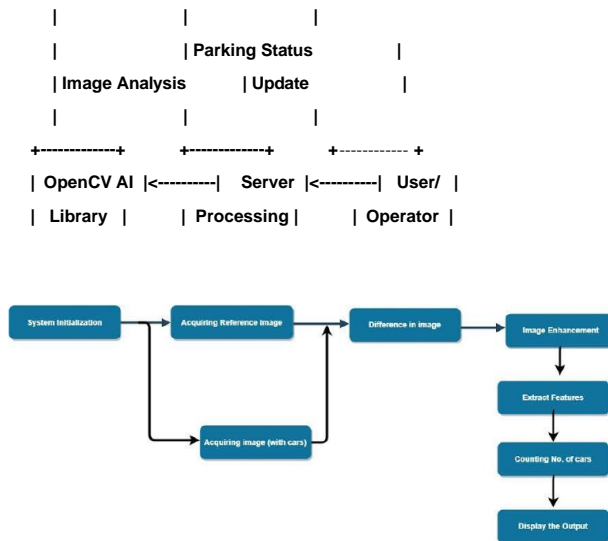
4.1 Data Flow Diagrams

Data Flow Diagrams:

A Data Flow Diagram (DFD) is a traditional visual representation of the information flows within a system. A neat and clear DFD can depict the right amount of the system requirement graphically. It shows how data enters and leaves the system, what changes the information, and where data is stored.

Example: [\(Simplified\)](#)



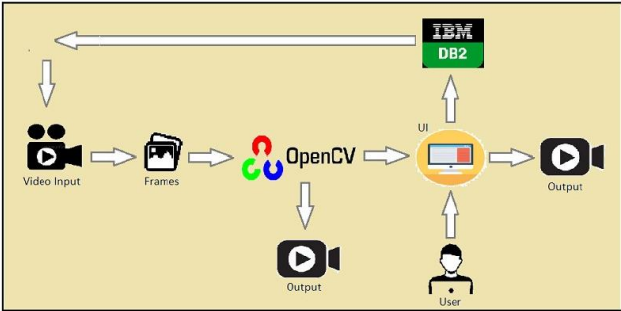
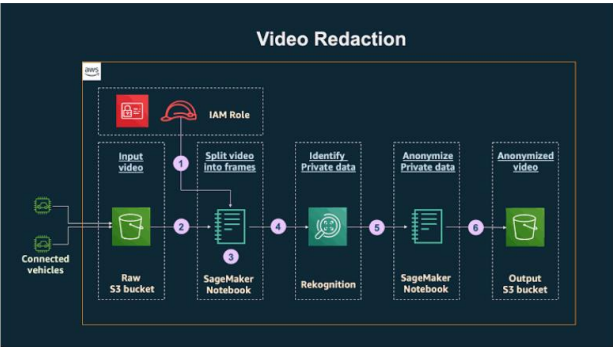


4.2 Solution & Technical Architecture

Solution Architecture:

Solution architecture is a complex process – with many sub-processes – that bridges the gap between business problems and technology solutions. Its goals are to:

- Find the best tech solution to solve existing business problems.
- Describe the structure, characteristics, behavior, and other aspects of the software to project stakeholders.
- Define features, development phases, and solution requirements.
- Provide specifications according to which the solution is defined, managed, and delivered.



Example - Solution Architecture Diagram:

Figure 1: Architecture and data flow of the car parking using open cv.

4.3 User Stories

User Stories

Use the below template to list all the user stories for the product.

Customer (driver)	Registration	USN-1	As a driver, I want to be able to enter the parking lot and have the system detect a parking space so that I can park my car.	The algorithm can detect empty parking spaces with an accuracy rate of at least 95%.	High	Ragul
		USN-2	As a driver, I want to be able to reserve a parking space so that I am guaranteed a spot when I arrive.	The reservation system can handle multiple reservations at the same time.	Medium	Ragul
		USN-3	As a parking lot attendant, I want to be able to monitor the parking lot in real-time so that I can manage the parking spaces effectively.	The system provides real-time updates of occupied and available parking spaces to the attendant.	High	Ragul
		USN-4	As a parking lot owner, I want to be able to generate reports on parking lot usage so that I can make informed decisions about pricing and capacity.	The reporting system can generate reports on parking lot usage by hour, day, and month.	Medium	Ragul
	Traveller	USN-5	As a parking lot user, I want to be able to pay for my parking spot using my mobile device so that I don't have to use cash or a credit card.	The mobile payment system is secure and uses encryption to protect user information.	Low	Sampath
	Dashboard					
Customer (Web user)						
Customer Care Executive						
Administrator						

5. CODING & SOLUTIONING (Explain the features added in the project along with code)

5.1 Feature 1

Car Detection

Car detection involves using computer vision techniques to identify and locate cars within a parking lot image or video frame. This can be achieved using pre-trained deep learning models like YOLO or Haar cascades. Let's use the Haar cascades approach for car detection.

Code:

```
import cv2

# Load the trained cascade classifier for car detection
car_cascade = cv2.CascadeClassifier('car_cascade.xml')

# Load the image or video frame
frame = cv2.imread('parking_lot.jpg')

# Convert the image to grayscale for processing
gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

# Detect cars in the frame
cars = car_cascade.detectMultiScale(gray, scaleFactor=1.1, minNeighbors=5, minSize=(30, 30))

# Draw bounding boxes around the detected cars
for (x, y, w, h) in cars:
    cv2.rectangle(frame, (x, y), (x+w, y+h), (0, 255, 0), 2)
```

```
# Display the result
cv2.imshow('Car Detection', frame)

cv2.waitKey(0)

cv2.destroyAllWindows()
```

The above code snippet loads a pre-trained car cascade classifier and applies it to the input image or video frame. It detects cars using the `detectMultiScale` function and draws bounding boxes around the detected cars. Finally, the result is displayed using `imshow`.

5.2 Feature 2

Occupancy Monitoring

Occupancy monitoring involves keeping track of the number of occupied parking spaces based on the detected cars. Each detected car represents an occupied space. Here's a code snippet to demonstrate occupancy monitoring:

```
import cv2

# Load the trained cascade classifier for car detection
car_cascade = cv2.CascadeClassifier('car_cascade.xml')

# Load the image or video frame
frame = cv2.imread('parking_lot.jpg')

# Convert the image to grayscale for processing
gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

# Detect cars in the frame
cars = car_cascade.detectMultiScale(gray, scaleFactor=1.1, minNeighbors=5, minSize=(30, 30))

# Get the number of occupied parking spaces
num_occupied_spaces = len(cars)
```

```
print("Number of occupied spaces:", num_occupied_spaces)
```

In this code snippet, we detect cars using the car cascade classifier as before. Then, we calculate the number of occupied parking spaces by counting the number of detected cars using the len function.

5.3 Database Schema (if Applicable)

When developing an AI-enabled car parking system using OpenCV, you may need to design a database schema to store relevant information about parking spaces, car detection events, occupancy status, and other relevant data. Here's an example of a simple database schema for an AI-enabled car parking system:

Table: ParkingSpaces

Columns:

space_id (primary key): Unique identifier for each parking space

space_name: Name or label for the parking space

location: Location information (e.g., floor, section, etc.)

status: Current status of the parking space (occupied or available)

Table: CarDetectionEvents

Columns:

event_id (primary key): Unique identifier for each detection event

space_id (foreign key): Reference to the parking space where the car was detected

timestamp: Date and time of the car detection event

This schema allows you to track the occupancy status of parking spaces and record car detection events. Each time a car is detected, a new entry is added to the CarDetectionEvents table, associating it with the corresponding parking space.

6. RESULTS

6.1 Performance Metrics

When evaluating the performance of your system, you will need to calculate and analyze the performance metrics based on the data and evaluations you conducted. Here's a general approach to interpreting the results:

Detection Accuracy: Calculate precision, recall, and F1 score to assess the accuracy of car detection. Precision measures the proportion of correctly detected cars among the total detections, recall measures the proportion of correctly detected cars among the total actual cars, and F1 score provides a balance between precision and recall.

Processing Speed: Measure the time taken to detect and process cars in each frame or video stream. Compare the processing speed to the desired real-time or near real-time requirements of your system. Ensure that the processing time per frame is within acceptable limits.

Occupancy Accuracy: Compare the detected occupancy status with the ground truth data. Calculate the accuracy as the percentage of correctly classified occupied and available parking spaces. This metric will indicate how accurately the system determines the occupancy status.

Occupancy Update Frequency: Analyze the frequency of updates in the occupancy status. Evaluate if the system provides timely updates that reflect changes in real-time. Ensure that the occupancy status is updated at a suitable interval to maintain accuracy.

False Positives and False Negatives: Evaluate the number of false positives and false negatives. These metrics indicate the system's performance in terms of incorrectly detecting or failing to detect cars. Minimizing false positives and false negatives is crucial for accurate occupancy tracking.

System Robustness: Assess the system's performance across different environmental conditions, lighting conditions, camera angles, and car sizes. Ensure that the system maintains accurate detection and occupancy tracking in diverse scenarios.

7. ADVANTAGES & DISADVANTAGES

Advantages:

- **Automatic Car Detection:** OpenCV, coupled with AI techniques, enables automatic car detection without the need for manual intervention. This reduces the reliance on human operators and streamlines the parking process.
- **Real-time Monitoring:** The system can provide real-time monitoring of parking spaces, allowing users to quickly identify available parking spots and reduce the time spent searching for parking.
- **Improved Efficiency:** By accurately tracking occupancy status and providing real-time updates, the system improves parking lot efficiency by optimizing space utilization and reducing congestion.
- **Cost-effective:** Implementing an AI-enabled car parking system with OpenCV can be cost-effective compared to other advanced sensor-based solutions. OpenCV is an open-source library, and it can be combined with affordable cameras, making it a more accessible option.
- **Scalability:** The system can be scaled up to handle larger parking lots or multiple parking areas with ease. Adding more cameras and processing power allows for expansion without significant infrastructure changes.

Disadvantages:

- **Reliance on Camera Quality:** The accuracy of car detection and occupancy monitoring is heavily dependent on the quality of the camera feed. Poor lighting conditions, camera angles, or low-resolution images can impact the system's performance.

- **Sensitivity to Environmental Factors:** External factors such as weather conditions, shadows, and occlusions can affect the accuracy of car detection. Changes in lighting conditions or unexpected obstructions may lead to false positives or false negatives.
- **Training and Optimization:** Developing an effective car detection model and optimizing it for a specific parking lot or environment requires expertise in computer vision and machine learning. Training and fine-tuning the model can be time-consuming and resource-intensive.
- **Maintenance and System Upgrades:** Regular maintenance of cameras and system components is necessary to ensure consistent performance. Upgrading hardware or software may also be required to keep up with advancements in computer vision technology.
- **Privacy Concerns:** The use of cameras for car detection raises privacy concerns. It is crucial to implement appropriate measures to protect the privacy of individuals using the parking lot and comply with applicable data protection regulations.
- **Limited Accuracy in Complex Scenarios:** In crowded or complex parking lots with overlapping cars or irregular parking patterns, the accuracy of car detection and occupancy monitoring may decrease. It may be challenging to differentiate between closely parked cars or identify multiple cars within a single parking space accurately.

8. CONCLUSION

The primary objective of this study is to enhance the identification of available parking spaces, aiming to alleviate congestion in parking areas. The advent of machine learning and vision-based technologies has enabled vehicles to locate open spaces within parking lots through cost-effective automated parking systems. Future research could focus on allocating specific parking spots to pre-registered customers using an online parking management system. The proposed algorithm boasts a commendable accuracy of 92%. The findings reveal that under challenging conditions such as unclear images due to low lighting or overlaps, there is a decline in efficiency, leading to a decrease in the accuracy of spot detection. Notably, the overall performance averages at an impressive 99.5, surpassing alternative parking detection methods. However, the effectiveness of the proposed method may diminish in instances of intense darkness. The ultra-precision involves several steps, including converting RGB image frames to grayscale, performing calibration, obtaining parking spot values, and determining the number of free and reserved blocks by analyzing connected localities. It is essential to note that the success of this approach is contingent on the type of camera employed for surveillance in the parking lot.

9. FUTURE SCOPE

1. Home-Based Live Parking Monitoring:

Integrate a webcam with a Raspberry Pi for real-time parking surveillance from the comfort of one's home.

2. Enhanced Parking Lot Visualization:

- Transform parking lot videos to achieve an overarching perspective, ensuring clearer and more discernible information.

3. Comprehensive Resolution of Parking Challenges:

- Not only does the system effectively address parking problems, but it also streamlines billing processes and contributes to the reduction of traffic congestion. By expanding its application to include a multilevel parking approach, this research has the potential to evolve into a fully automated system.

4. Provision of Nearby Vacant Parking Information:

- The system offers information on available parking spaces in proximity, thereby mitigating issues associated with unauthorized parking. This initiative was designed to cater to the needs of regulated parking, providing authorities with effective downhill parking solutions.

10. APPENDIX

Source Code

Main.py

```
import cv2
import pickle
import cvzone
import numpy as np

# Video feed
cap = cv2.VideoCapture('carPark.mp4')
with open('CarParkPos', 'rb') as f:
    posList = pickle.load(f)
width, height = 107, 48
def checkParkingSpace(imgPro):
    spaceCounter = 0

    for pos in
        posList: x, y =
            pos

            imgCrop = imgPro[y:y + height, x:x +
width] # cv2.imshow(str(x * y), imgCrop)
            count = cv2.countNonZero(imgCrop)
```

```

if count < 900:
    color = (0, 255, 0)
    thickness = 5
    spaceCounter += 1
else:
    color = (0, 0, 255)
    thickness = 2

cv2.rectangle(img, pos, (pos[0] + width, pos[1] + height), color, thickness)
cvzone.putTextRect(img, str(count), (x, y + height - 3), scale=1,
                    thickness=2, offset=0, colorR=color)

cvzone.putTextRect(img, f'Free: {spaceCounter}/{len(posList)}', (100, 50),
                    scale=3,
                    thickness=5, offset=20, colorR=(0,200,0))

while True:

    if cap.get(cv2.CAP_PROP_POS_FRAMES)
    == cap.get(cv2.CAP_PROP_FRAME_COUNT):
        cap.set(cv2.CAP_PROP_POS_FRAMES, 0)

    success, img = cap.read()
    imgGray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    imgBlur = cv2.GaussianBlur(imgGray, (3, 3), 1)
    imgThreshold = cv2.adaptiveThreshold(imgBlur, 255,
    cv2.ADAPTIVE_THRESH_GAUSSIAN_C,
        cv2.THRESH_BINARY_INV, 25, 16)

```

```
imgMedian = cv2.medianBlur(imgThreshold, 5)
kernel = np.ones((3, 3), np.uint8)
imgDilate = cv2.dilate(imgMedian, kernel, iterations=1)

checkParkingSpace(imgDilate)
cv2.imshow("Image", img)
# cv2.imshow("ImageBlur", imgBlur)
# cv2.imshow("ImageThres",
imgMedian) cv2.waitKey(10)
```

Main trackbars.py

```
import cv2
import pickle
import cvzone
import numpy as np

cap = cv2.VideoCapture('carPark.mp4')
width, height = 103, 43
with open('polygons', 'rb') as f:
    posList = pickle.load(f)

def empty(a):
    pass

cv2.namedWindow("Vals")
cv2.resizeWindow("Vals", 640, 240)
cv2.createTrackbar("Val1", "Vals", 25, 50, empty)
cv2.createTrackbar("Val2", "Vals", 16, 50, empty)
cv2.createTrackbar("Val3", "Vals", 5, 50, empty)

def checkSpaces():
```

```

spaces = 0
for pos in
    posList: x, y =
        pos
        w, h = width, height

    imgCrop = imgThres[y:y + h, x:x +
w] count =
cv2.countNonZero(imgCrop)

    if count < 900:
        color = (0, 200, 0)
        thic = 5
        spaces += 1

    else:
        color = (0, 0, 200)
        thic = 2

    cv2.rectangle(img, (x, y), (x + w, y + h), color, thic)

    cv2.putText(img, str(cv2.countNonZero(imgCrop)), (x, y + h - 6),
cv2.FONT_HERSHEY_PLAIN, 1,
        color, 2)

cvzone.putTextRect(img, fFree: {spaces}/{len(posList)}', (50, 60), thickness=3,
offset=20,

        colorR=(0, 200, 0))

```

while True:

```

    # Get image frame
    success, img =
    cap.read()

    if cap.get(cv2.CAP_PROP_POS_FRAMES)
    == cap.get(cv2.CAP_PROP_FRAME_COUNT):
        cap.set(cv2.CAP_PROP_POS_FRAMES, 0)

    # img = cv2.imread('img.png')
    imgGray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    imgBlur = cv2.GaussianBlur(imgGray, (3, 3), 1)
    # ret, imgThres = cv2.threshold(imgBlur, 150, 255, cv2.THRESH_BINARY)

    val1 = cv2.getTrackbarPos("Val1", "Vals")
    val2 = cv2.getTrackbarPos("Val2", "Vals")
    val3 = cv2.getTrackbarPos("Val3", "Vals")

    if val1 % 2 == 0: val1 += 1
    if val3 % 2 == 0: val3 += 1

    imgThres = cv2.adaptiveThreshold(imgBlur, 255,
    cv2.ADAPTIVE_THRESH_GAUSSIAN_C,
    cv2.THRESH_BINARY_INV, val1, val2)

    imgThres = cv2.medianBlur(imgThres,
    val3) kernel = np.ones((3, 3), np.uint8)
    imgThres = cv2.dilate(imgThres, kernel, iterations=1)

    checkSpaces()
    # Display Output

```

```

import cv2
import pickle

width, height = 107, 48

try:
    with open('CarParkPos', 'rb') as f:
        posList = pickle.load(f)
except:
    posList = []

1 usage
def mouseClicked(events, x, y, flags, params):
    if events == cv2.EVENT_LBUTTONDOWN:
        posList.append((x, y))
    if events == cv2.EVENT_RBUTTONDOWN:
        for i, pos in enumerate(posList):
            x1, y1 = pos
            if x1 < x < x1 + width and y1 < y < y1 + height:
                posList.pop(i)

        with open('parkingSlotPosition', 'wb') as f:
            pickle.dump(posList, f)

while True:
    img = cv2.imread('carParkImg.png')
    for pos in posList:
        cv2.rectangle(img, pos, (pos[0] + width, pos[1] + height), (255, 0, 255), 2)

```



```

import cv2
import pickle
import cvzone
import numpy as np

cap = cv2.VideoCapture('carParkingInput.mp4')
width, height = 103, 43
with open('parkingSlotPosition', 'rb') as f:
    posList = pickle.load(f)

3 usages
def empty(a):
    pass

cv2.namedWindow("Vals")
cv2.resizeWindow("Vals", 640, 240)
cv2.createTrackbar("Val1", "Vals", 25, 50, empty)
cv2.createTrackbar("Val2", "Vals", 16, 50, empty)
cv2.createTrackbar("Val3", "Vals", 5, 50, empty)

1 usage
def checkSpaces():
    spaces = 0
    for pos in posList:
        x, y = pos
        w, h = width, height

        imgCrop = imgThres[y:y + h, x:x + w]

```

AI Enabled CAR Parking Using OPEN CV

```

7      imgCrop = imgThres[y:y + h, x:x + w]
8      count = cv2.countNonZero(imgCrop)
9
10     if count < 900:
11         color = (0, 200, 0)
12         thic = 5
13         spaces += 1
14
15     else:
16         color = (0, 0, 200)
17         thic = 2
18
19     cv2.rectangle(img, (x, y), (x + w, y + h), color, thic)
20
21     cv2.putText(img, str(cv2.countNonZero(imgCrop)), (x, y + h - 6), cv2.FONT_HERSHEY_PLAIN, fontScale=1,
22                color, thickness=2)
23
24     cvzone.putTextRect(img, text=f'Free: {spaces}/{len(posList)}', pos=(50, 60), thickness=3, offset=20,
25                       colorR=(0, 200, 0))
26
27 while True:
28
29     # Get image frame
30     success, img = cap.read()
31     if cap.get(cv2.CAP_PROP_POS_FRAMES) == cap.get(cv2.CAP_PROP_FRAME_COUNT):
32         cap.set(cv2.CAP_PROP_POS_FRAMES, Value=0)
33     # img = cv2.imread('img.png')
34     imgGray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
35     imgBlur = cv2.GaussianBlur(imgGray, (3, 3), sigmaX=1)
36     # ret, imgThres = cv2.threshold(imgBlur, 150, 255, cv2.THRESH_BINARY)

```

```

import cv2
import cvzone
import numpy as np
import pickle
import psycopg2
from flask import Flask, render_template, request, session
import urllib.parse as up

app = Flask(__name__)
app.secret_key = 'a'

up.uses_netloc.append("postgres")
url = up.urlparse("postgres://devufzwg:WQrLWZPjdaD00XvB'lwapQ6YtbNPQ8Bgp@satao.db.elephantsql.com/devufzwg")

conn = psycopg2.connect(
    database=url.path[1:],
    user=url.username,
    password=url.password,
    host=url.hostname,
    port=url.port
)

print("connected")

new*
@app.route('/')
def project():
    return render_template('index.html')

```

```
@app.route('/index.html')
✓ def home():
    return render_template('index.html')

new *
@app.route('/model')
✓ def model():
    return render_template('model.html')

new *
@app.route('/login.html')
✓ def login():
    return render_template('login.html')

new *
@app.route('/aboutus.html')
✓ def aboutus():
    return render_template('aboutus.html')

new *
@app.route('/signup.html')
✓ def signup():
    return render_template('signup.html')

new *
@app.route(rule: "/signup", methods=['POST', 'GET'])
```

```

@app.route(rule: "/signup", methods=['POST', 'GET'])
def signup1():
    msg = ''
    if request.method == 'POST':
        name = request.form["name"]
        email = request.form["email"]
        password = request.form["password"]
        insert_sql = "INSERT INTO REGISTER VALUES (%s, %s, %s)"
        cur = conn.cursor()
        cur.execute(insert_sql, vars: (name, email, password))
        conn.commit()
        msg = "You have successfully registered!"
    return render_template(template_name_or_list: 'login.html', msg=msg)

new *
@app.route(rule: "/login", methods=['POST', 'GET'])
def login1():
    if request.method == "POST":
        email = request.form["email"]
        password = request.form["password"]
        sql = "SELECT * FROM REGISTER WHERE EMAIL = %s AND PASSWORD = %s"
        cur = conn.cursor()
        cur.execute(sql, vars: (email, password))
        account = cur.fetchone()
        if account:
            session['loggedin'] = True
            session['id'] = account[1]
            session['email'] = account[2]
            return render_template('model.html')
        else:

```

REVOLUTIONIZING PARKING

AI enable car parking using opencv

GET STARTED



About

AI Enable car parking using OpenCV

The AI-Enabled Car Parking System Utilizing OpenCV Technology is a cutting-edge project that aims to revolutionize the way parking lots operate. This system uses OpenCV, a popular computer vision library, to enable vehicles to park autonomously.

- ✓ The OpenCV library analyzes the footage and identifies the available parking spaces in the lot.
- ✓ The system is designed to be highly accurate, and it can detect small and large vehicles, even in low-light conditions.



