Ai Enable Car Parking With Open CV

TEAM ID:592031

Done By:

Thakur Akshath Singh
Shaik Mansoor Vali
Pavan
Manichandan

Project Report Format

1. INTRODUCTION

1.1 Project Overview

Nowadays, the majority of AI-enabled parking systems are still operated manually.There isn't an automatic monitoring mechanism in place to track the amount of capacity that each parking space has. Drivers frequently need to take a roundabout route through the parking lot in search of an empty place. These issues are particularly prevalent in areas with a high population density, such as those close to schools, malls, hospitals, and other major gathering centers.

1.2 Purpose

The purpose of using AI-enabled car parking systems with OpenCV (Open Source Computer Vision) is to automate and streamline the process of parking cars in garages or parking lots. Through the use of computer vision techniques and artificial intelligence algorithms, these systems are able to effectively track and identify cars, determine when parking spaces are available, and direct drivers to open places.

LITERATURE SURVEY

2.1 Existing problem

1. **Accuracy and Reliability:**
    - **Detection Accuracy:** AI systems need to accurately detect and recognize objects, including cars and obstacles, to ensure reliable parking assistance. Inaccuracies in object detection can lead to collisions or improper parking.
2. **Real-time Processing:**
    - **Latency:** Real-time processing is crucial for car parking systems. Any delay in processing information could lead to accidents or difficulties in parking.

3. **Adaptability:**
    - **Varied Environments:** Parking lots can have diverse environments with different lighting conditions, weather, and types of vehicles. AI systems must be adaptable to these varying conditions.
4. **Human Interactions:**
    - **Interaction with Pedestrians:** Ensuring the safety of pedestrians and providing clear communication between the AI system and humans in the parking area is vital.
5. **Integration with Infrastructure:**
    - **Compatibility:** Integration with existing infrastructure, such as cameras, sensors, and communication systems, can be challenging. Compatibility issues may arise when implementing AI in older parking facilities.
6. **Security and Privacy:**
    - **Data Security:** Handling and storing sensitive information, such as license plate numbers or user data, requires robust security measures to prevent unauthorized access or misuse.
7. **Cost:**
    - **Implementation Costs:** The cost of deploying and maintaining AI-enabled parking systems may be a barrier for widespread adoption, especially for smaller parking facilities.
8. **Regulatory and Legal Challenges:**
    - **Compliance:** Meeting regulatory standards and ensuring that the AI system complies with local laws can be a complex process.
9. **User Experience:**
    - **User Interface:** The user interface of the AI system should be intuitive and user-friendly to encourage adoption and ensure a positive experience for users.
10. **Maintenance and Upgrades:**
    - **Software Updates:** Regular updates and maintenance are essential to address bugs, improve performance, and incorporate new features. However, implementing updates without disrupting the parking operations can be a challenge

2.2 References

https://medium.com/swlh/build-a-simple-smart-parking-project-using-python-and-opencv-2bd891d05199

https://www.hackster.io/timothy_malche/aispark-ai-based-smart-parkin

g-system-f0ee8f

https://www.mural.co/templates/empathy-map-canvas

https://www.mural.co/templates/ideate

https://aws.amazon.com/blogs/industries/voice-applications-in-clinical-research-powered-by-ai-on-aws-part-1-architecture-and-design-considerations/

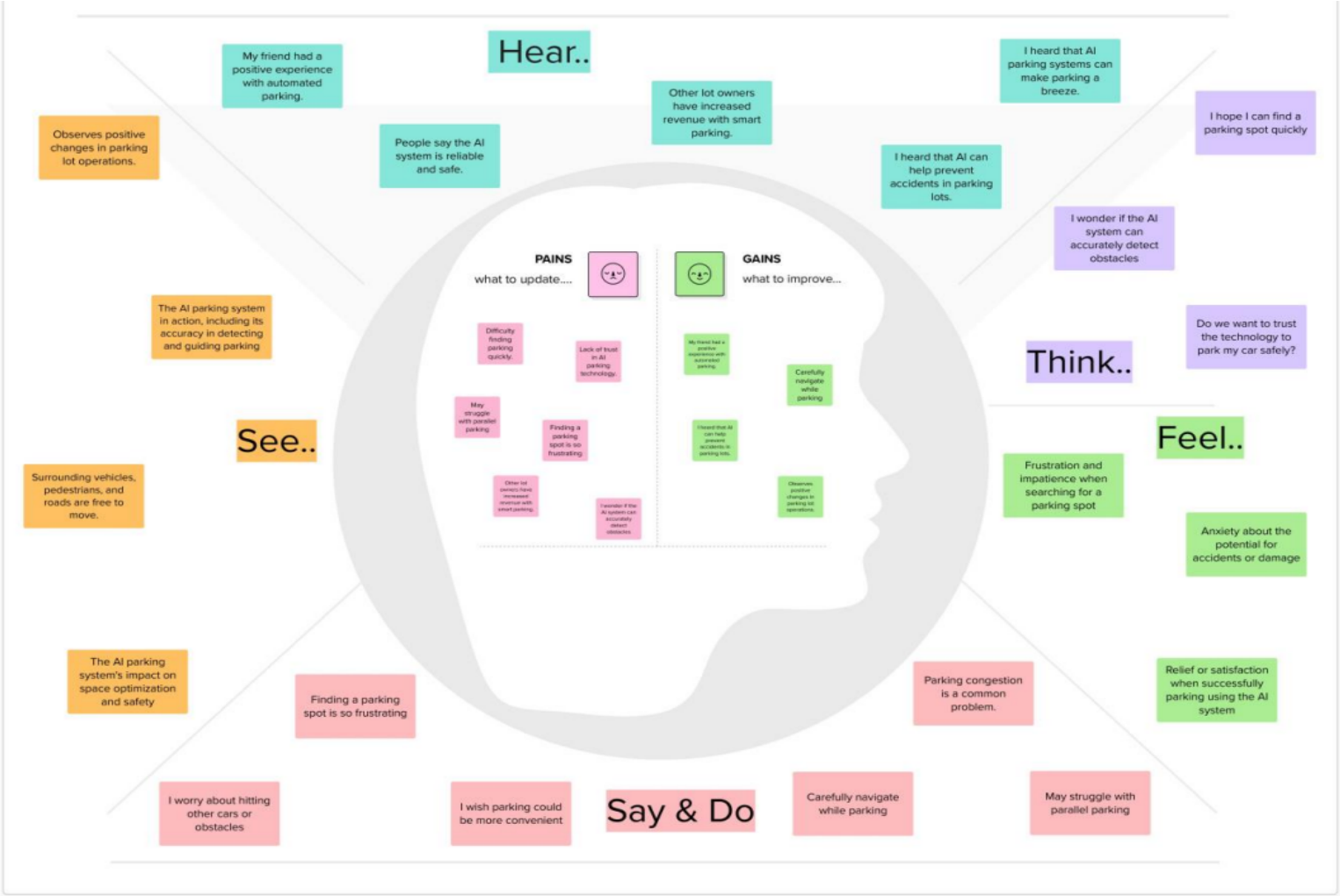
2.3 Problem Statement Definition

Developing an AI-enabled car parking system using OpenCV poses challenges in achieving accurate and real-time object detection and recognition for efficient parking assistance. The system must adapt to diverse environmental conditions, including varying lighting, weather, and types of vehicles in parking lots. Ensuring seamless integration with existing infrastructure, such as cameras and sensors, is critical, along with addressing security concerns related to data handling. Human interactions within the parking area, including pedestrians, necessitate clear communication interfaces. The high implementation costs and potential compatibility issues in older parking facilities must be considered. Regulatory compliance, user-friendly interfaces, and the need for continuous maintenance and updates add complexity to the successful deployment of AI-based car parking systems.

3.IDEATION & PROPOSED SOLUTION

3.1 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



3.2 Ideation & Brainstorming

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.

Brainstorming:

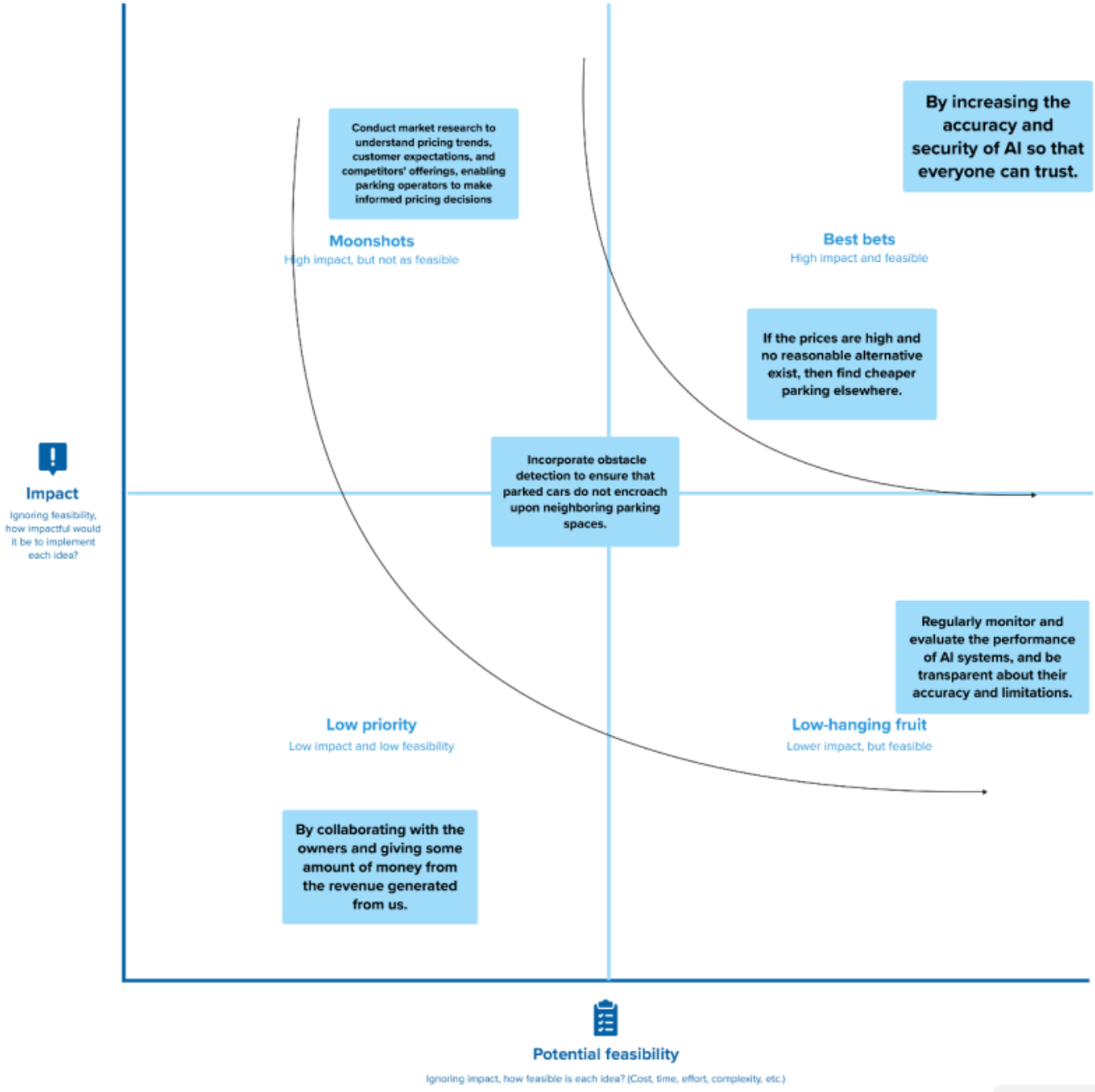
Shaik Mansoor Vali			
By collaborating with the owners and giving some amount of money from the revenue generated from us.	By making all the available parking areas visible to the customer who wants to park their vehicles.	By increasing the accuracy and security of AI so that everyone can trust.	By adding an alignment algorithm in the code to navigate the perfect lining to park without getting crashed by other cars
By researching the ongoing projects and making a robust technology that makes the customer believe in AI.	By having some awareness, pushing our product to the masses and advertising our product.	By creating an application which navigates the user activities in parking and giving rewards for successful parking.	By dividing the vehicles by their size and model and providing a perfect parking to pare their vehicle.

Mani Chandan			
Negotiate with Lot Owners: Reach out to the other lot owners and attempt to negotiate them.	Form a Parking Association: Consider forming a parking association with the other car owners in your lot. A collective approach can give you more negotiating power when dealing with the lot owners.	Document Everything: Keep records of all communications and actions related to the price increase.	Enhance Training Data: AI systems heavily rely on the quality and quantity of training data.

Pavan Kumar			
Open communication with the lot owners may be helpful in decreasing the prices. One more option is car-pooling, which might reduce the parking cost as well as diesel cost.	***If the prices are high and no reasonable alternative exist, then find cheaper parking elsewhere.***	***Improve accuracy by investing in better data collection and preprocessing to have high training data.***	***Implement security and privacy measures to protect sensitive information, ensuring that users' data is handled responsibly.***
Clear Parking Lot Design: Design parking lots with clear and well-marked parking spaces. Use painted lines, numbers, and other markers to designate individual parking spots.	***In cases where parking lot space is limited, consider implementing valet services to maximize the efficient use of the available parking spots.***	***For those who repeatedly fail to park correctly, impose escalating penalties, which may include towing the vehicle.***	***Regularly monitor and evaluate the performance of AI systems, and be transparent about their accuracy and limitations.***

Thakur Akshath Singh			
Market Research: Conduct market research to understand pricing trends, customer expectations, and competitors' offerings, enabling parking operators to make informed pricing decisions	Flexible Pricing Models: Offer parking operators the flexibility to choose from various pricing models, such as hourly, daily, or monthly rates, depending on their specific needs and competition in their area.	Negotiation and Partnerships: Engage in negotiations with other lot owners to explore partnerships, consortiums, or cooperative pricing strategies that benefit all stakeholders	Collaborate with third-party organizations
Advanced Object Detection Models: Invest in more accurate and robust object detection models, like the latest versions of YOLO or improved custom models, to increase the accuracy of vehicle and object recognition.	License Plate Recognition: Utilize license plate recognition to verify the vehicle's position within a parking space, providing an additional layer of confirmation.	Object detection: Incorporate obstacle detection to ensure that parked cars do not encroach upon neighboring parking spaces.	Parking Space Design: Work with parking facility operators to optimize parking space design, including the size and layout of parking spots to facilitate proper alignment.

Ideation:


Impact
Ignoring feasibility, how impactful would it be to implement each idea?
Conduct market research to understand pricing trends, customer expectations, and competitors' offerings, enabling parking operators to make informed pricing decisions
Moonshots
High impact, but not as feasible
By increasing the accuracy and security of AI so that everyone can trust.
Best bets
High impact and feasible
If the prices are high and no reasonable alternative exist, then find cheaper parking elsewhere.
Incorporate obstacle detection to ensure that parked cars do not encroach upon neighboring parking spaces.
Regularly monitor and evaluate the performance of AI systems, and be transparent about their accuracy and limitations.
Low priority
Low impact and low feasibility
Low-hanging fruit
Lower impact, but feasible
By collaborating with the owners and giving some amount of money from the revenue generated from us.
Potential feasibility
Ignoring impact, how feasible is each idea? (Cost, time, effort, complexity, etc.)

4. REQUIREMENT ANALYSIS

4.1 Functional requirement

No.	Functional Requirement	Sub Requirement (Story / Sub-Task)
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
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
3	Payment and billing	Calculate parking fee based on parking duration Allow payment via various modes like credit card,mobile wallet
4	Security and surveillance	Monitor parking lot using cameras for suspicious activities Alert security personnel in case of any security breach
5	System maintenance and support	Provide regular maintenance and updates to the system Offer customer support for any issues faced by users

4.2 Non-Functional requirements

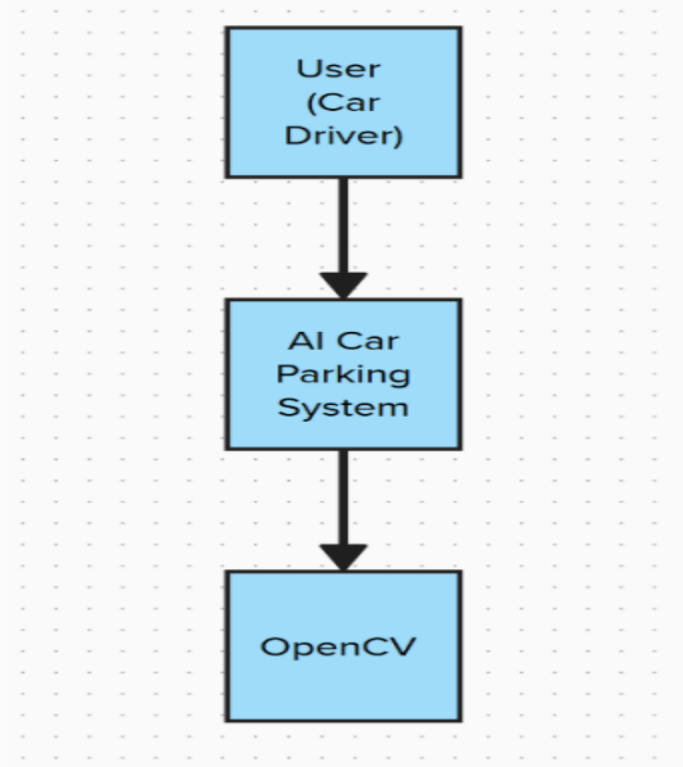
No	Non-Functional Requirement	Description
1	Usability	The system should be easy to use and user friendly, providing clear instructions to drivers on where to park and displaying available spots in an intuitive manner
2	Security	The system should be secure and able to prevent unauthorized access to the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

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.

5.PROJECT DESIGN

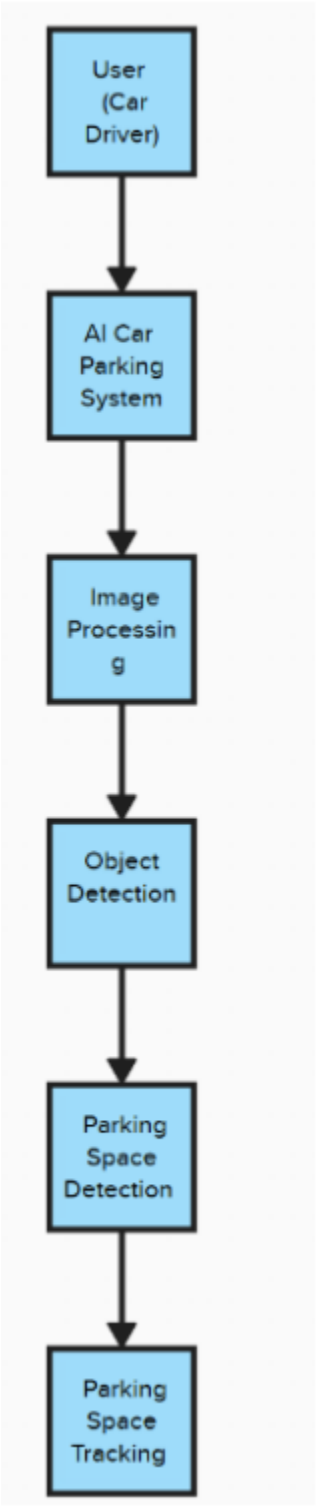
5.1 Data Flow Diagrams & User Stories

Level 0 DFD:



In this DFD: The "User (Car Driver)" interacts with the "AI Car Parking System" to park their car. The "AI Car Parking System" uses OpenCV for various computer vision tasks, such as object detection, image processing, and tracking. OpenCV processes the camera feed data, identifies available parking spaces, and manages the parking process.

Level 1 DFD: Now, let's break down the "AI Car Parking System" further to illustrate the internal data flow



In this Level 1 DFD:

1)The "AI Car Parking System" contains various subprocesses, such as "Image Processing," "Object Detection," "Parking Space Detection," and "Parking Space Tracking."

2)"Image Processing" involves preprocessing the camera feed data to

enhance it for better analysis.

3)"Object Detection" identifies vehicles and objects in the camera feed. 4)"Parking Space Detection" identifies available parking spaces. 5)"Parking Space Tracking" tracks the movement of vehicles within the parking area

5.2 Solution Architecture

Step 1: The parking lot cameras capture images and videos of the parking lot.

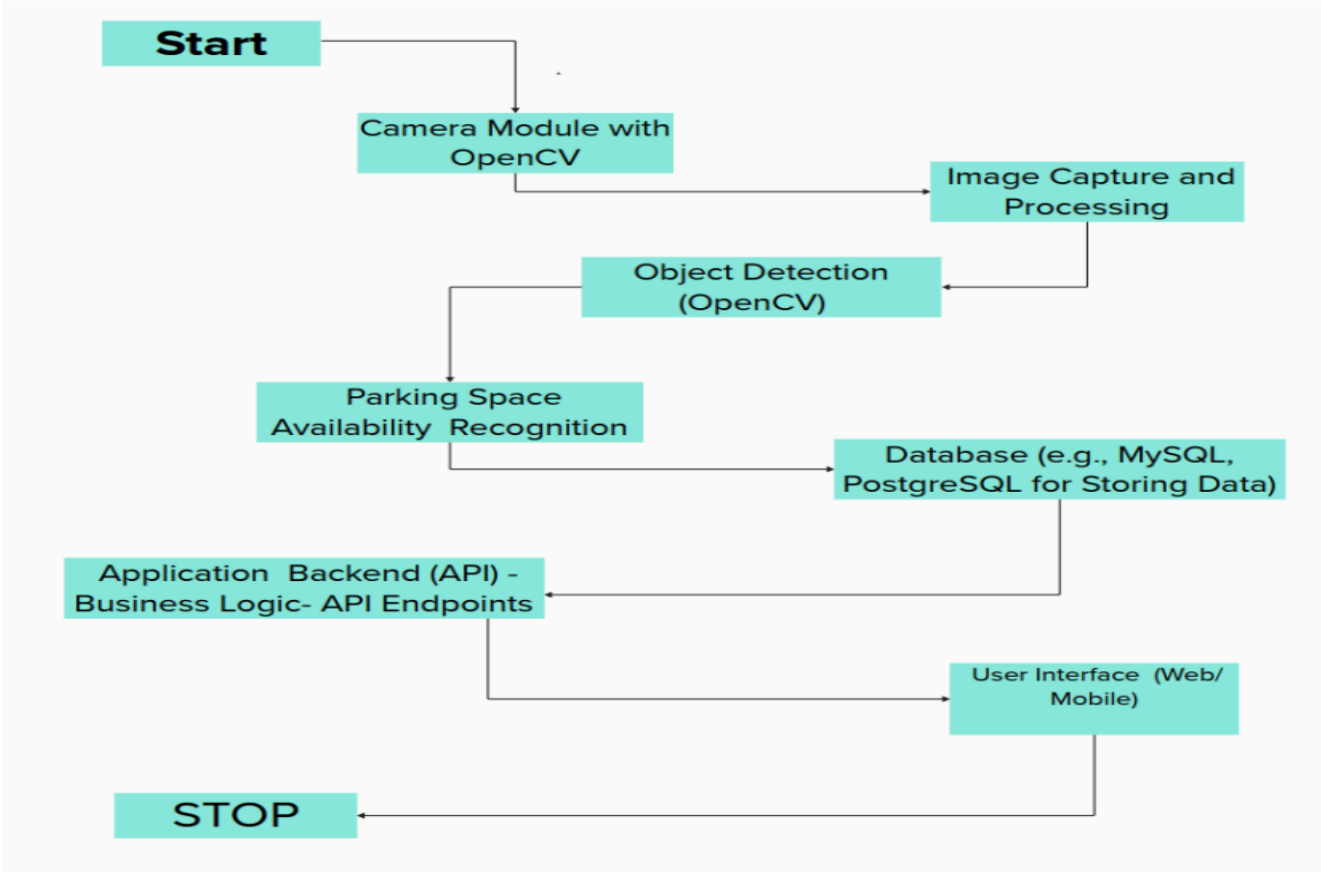
Step 2: Upload the captured images and videos to the S3 bucket.

Step 3: The Amazon Recognition endpoint analyzes the images and videos to detect vehicles and track them.

Step 4: The parking lot occupancy is determined and updated in Amazon DynamoDB.

Step 5: If a vehicle enters or exits the parking lot, it will trigger a notification.

Step 6: The notification will be sent to the users via SNS



6. PROJECT PLANNING & SCHEDULING

6.1 Technical Architecture

a) Technical Architecture

Component	Description
Sensors and Input Devices	Cameras, additional sensors (e.g., ultrasonic sensors)
Hardware Platform	Computer, embedded system (e.g., Raspberry Pi), specialized AI hardware
Image/Video Processing Layer	OpenCV for video capture, analysis, object detection, and transformations
Object Detection	Pre-trained/custom models for car, pedestrian, obstacle detection
Control and Path Planning	Control algorithms (e.g., PID controllers), path planning
User Interface	Optional UI for monitoring and control
Integration Layer	Manages data flow and communication between components
Cloud Connectivity (Optional)	Cloud services for remote features (e.g., monitoring, updates)

b) Open Source Frameworks

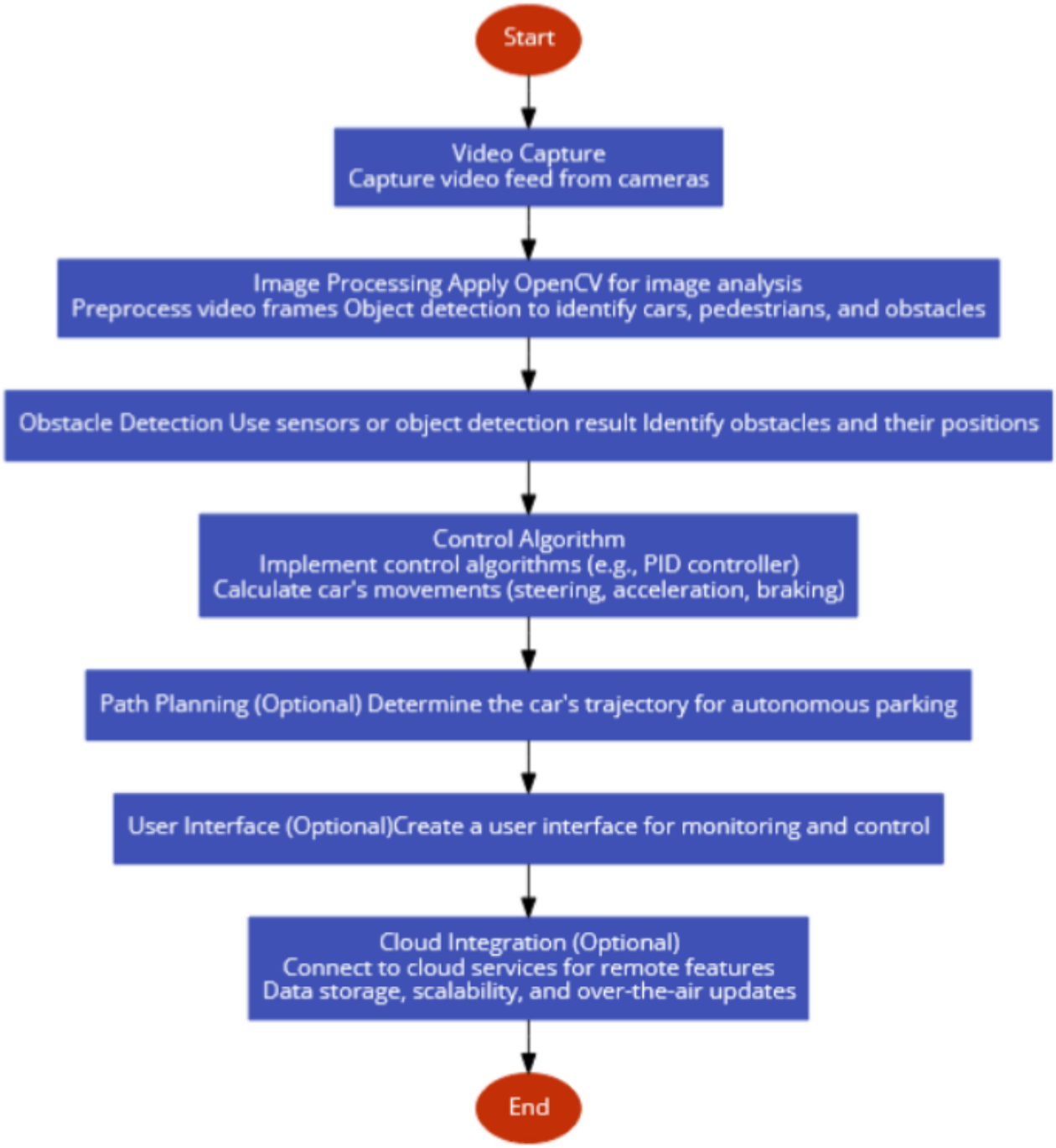
Framework/Library	Description
OpenCV	Image and video processing, object detection
TensorFlow	Machine learning framework (object detection training)
PyTorch	Deep learning framework (object detection training)
ROS (Robot Operating System)	Robotic control and communication
NumPy	Numerical operations and data manipulation
Matplotlib	Data visualization (plotting)
OpenCV-Python	Image display and manipulation
Pandas	Data manipulation and analysis

c) Third-party APIs

API	Use Case
Maps and Geolocation APIs	Finding nearest parking space, geolocation
Speech Recognition and NLP APIs	Voice commands, natural language interactions
IoT and Sensor APIs	Sensor data collection and management
Cloud Services APIs (for cloud deployment)	Data storage, remote monitoring, updates

d) Cloud Deployment

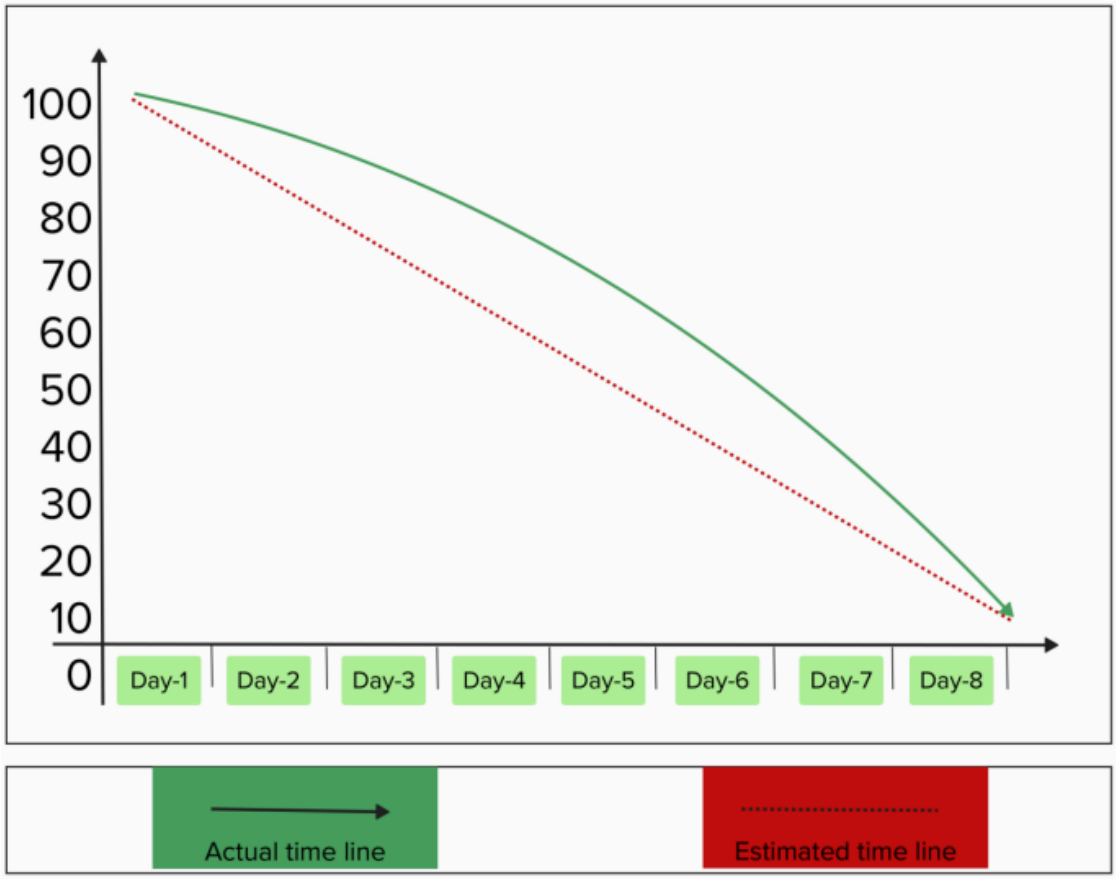
Cloud Deployment Element	Description
Data Storage	Cloud storage for video footage, logs, data
Scalability	Scalable infrastructure for more cameras, parking spaces
Remote Monitoring	Dashboard or remote monitoring application
Over-the-Air Updates	Simplified process for software updates
Security	Cloud security features to protect data and system
Cost Management	Cost-effective resource usage


Start
Video Capture
Capture video feed from cameras
Image Processing Apply OpenCV for image analysis
Preprocess video frames Object detection to identify cars, pedestrians, and obstacles
Obstacle Detection Use sensors or object detection result Identify obstacles and their positions
Control Algorithm
Implement control algorithms (e.g., PID controller)
Calculate car's movements (steering, acceleration, braking)
Path Planning (Optional) Determine the car's trajectory for autonomous parking
User Interface (Optional)Create a user interface for monitoring and control
Cloud Integration (Optional)
Connect to cloud services for remote features
Data storage, scalability, and over-the-air updates
End

6.2 Sprint Planning & Estimation

Phase	Number	Requirement (Epic)	User Story	Story Points	Priority
Phase-2	1	AI Car Parking System	As a driver, I want to detect empty parking spots.	8	High
Phase-2	2	AI Car Parking System	As a driver, I want the AI to guide my car into a parking spot.	13	Medium
Phase-2	3	AI Car Parking System	As a parking lot owner, I want to monitor available parking spaces in real-time.	5	High
Phase-3	4	AI Car Parking System	As a driver, I want to receive notifications when a parking spot becomes available.	3	Low
Phase-1	5	AI Car Parking System	As a driver, I want a clean interface to track my movement easily without any delay.		

6.3 Sprint Delivery Schedule



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

7.1 Feature 1

```
import cv2
import pickle
```

```
#Define the width and height of ROI
width, height = 107, 48
# Creating an empty file and loading to a variable & Creating an empty list
try:
    with open('parkingSlotPosition', 'rb') as f:
        posList = pickle.load(f)
except:
    posList = []
```

```
def mouseClick(events, x, y, flags, params):
    if events == cv2.EVENT_LBUTTONDOWN:
        posList.append((x, y))
# Removing unwanted ROI from posList
    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)
# Saving the postist values to parkingSlotPosition file
    with open('parkingSlotPosition', 'wb') as f:
        pickle.dump(posList, f)
```

```
while True:
    img = cv2.imread("D:\programs\Python programs\smart internz\carParkImg.png")
    for pos in posList:
        cv2.rectangle(img, pos, (pos[0] + width, pos [1] + height), (255, 255, 255), 2)
    cv2.imshow("Image", img)
    cv2.setMouseCallback("Image", mouseClick)
    cv2.waitKey(1)
```

7.2 Feature 2

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

```
#Video feed
cap= cv2.VideoCapture("D:\programs\Python programs\smart internz\carParkingInput.mp4")
#Loading the ROI from parkingSlotPosition file
with open('parkingSlotPosition', 'rb') as f:
    posList = pickle.load(f)
#Define width and height
width, height = 107, 48
```

```
def checkParkingSpace (imgPro):
    spaceCounter = 0
    for pos in posList:
        x, y = pos
        # Crop the image based on ROI
        imgCrop= imgPro[y:y + height, x:x + width]
        # Counting the pixel values from cropped image
        count = cv2.countNonZero (imgCrop)
        if count < 900:
            color = (0, 255, 0)
            thickness = 5
            spaceCounter += 1
        else:
            color= (0, 0, 255)
            thickness = 2
        # Draw the rectangle based on the condition defined above
        cv2.rectangle(img, pos, (pos[0]+ width, pos[1]+ height), color, thickness)
    # Display the available parking slot count / total parking slot count
    cvzone.putTextRect(img, f'Free: {spaceCounter}/{len(posList)}', (100, 50), scale=3, thickness= 5, offset=20, color=(0,200,0))
```

```
while True:
# Looping the video
if cap.get(cv2.CAP_PROP_POS_FRAMES) == cap.get(cv2.CAP_PROP_FRAME_COUNT):
    cap.set(cv2.CAP_PROP_POS_FRAMES, 0)
```

```
while True:
    # Looping the video
    if cap.get(cv2.CAP_PROP_POS_FRAMES) == cap.get(cv2.CAP_PROP_FRAME_COUNT):
        cap.set(cv2.CAP_PROP_POS_FRAMES, 0)
    # Reading frame by frame from video
    success, img = cap.read()
    # Converting to gray scale image
    imgGray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    imgBlur = cv2.GaussianBlur (imgGray, (3, 3), 1) # Applying blur to image
    # Applying threshold to the image
    imgThreshold = cv2.adaptiveThreshold(imgBlur, 255, cv2.ADAPTIVE_THRESH_GAUSSIAN_C, cv2.THRESH_BINARY_INV, 25, 16)
    imgMedian = cv2.medianBlur (imgThreshold, 5) # Applying blur to image
    kernel = np.ones((3, 3), np.uint8)
    imgDilate = cv2.dilate (imgMedian, kernel, iterations=1)
    # Passing dilate image to the function
    checkParkingSpace (imgDilate)
    cv2.imshow("Image", img)
    cv2.waitKey(10)
```

7.3 Database Schema

```
from flask import Flask, render_template, request, session
import cv2
import pickle
import cvzone
import numpy as np
import ibm_db
import re
```

```
app = Flask(__name__)
app.secret_key = 'your_secret_key'
```

```
conn= ibm_db.connect("DATABASE=vali3;HOST=localhost;PORT=25000;PROTOCOL=TCPIP;UID=
print("connected")
```

```
@app.route('/')
def project():
        return render_template('Login.html')
@app.route('/login')
def login():
        return render_template('Login.html')
@app.route('/signup')
def signup1():
        return render_template('Signup.html')
@app.route('/index')
def index():
    return render_template('Index.html')
@app.route('/home')
def Home():
    return render_template('Index.html')
@app.route('/model')
def model():
        return render_template('Model.html')
@app.route('/about')
def about():
        return render_template('About.html')
@app.route('/contact')
def contact():
        return render_template('Contact.html')
```

```
def signup():
    msg = ' '
    if request.method == 'POST':
        name= request.form["name"]
        email = request.form["email"]
        password = request.form["password"]
        sql = "SELECT * FROM REGISTER WHERE name = ?"
        stmt = ibm_db.prepare(conn, sql)
        ibm_db.bind_param(stmt, 1, name)
        ibm_db.execute(stmt)
        account = ibm_db.fetch_assoc(stmt)
        print (account)
        if account:
            return render_template('login.html', error=True)
        elif not re.match(r'[^@]+@[^@]+\.[^@]+', email):
            msg = "Invalid Email Address!"
        else:
            insert_sql = "INSERT INTO REGISTER VALUES (?,?,?)"
            prep_stmt = ibm_db.prepare(conn, insert_sql)
            # this username & password should be same as db-2 details & order also
            ibm_db.bind_param(prep_stmt, 1, name)
            ibm_db.bind_param(prep_stmt, 2, email)
            ibm_db.bind_param(prep_stmt, 3, password)
            ibm_db.execute(prep_stmt)
            msg = "You have successfully registered !"
            return render_template('login.html', msg=msg)
```

```
@app.route("/log", methods=['POST', 'GET'])
def login1():
    if request.method == "POST":
        email = request.form["email"]
        password = request.form["password"]
        sql = "SELECT * FROM REGISTER WHERE EMAIL=? AND PASSWORD=?" # from db2 sql table
        stmt = ibm_db.prepare(conn, sql)
        # this username & password should be same as db-2 details & order also
        ibm_db.bind_param(stmt, 1, email)
        ibm_db.bind_param(stmt, 2, password)
        ibm_db.execute(stmt)
        account = ibm_db.fetch_assoc(stmt)
        print (account)
        if account:
            session['Loggedin'] = True
            session['id'] = account['EMAIL']
            session['email'] = account['EMAIL']
            return render_template('model.html')
        else:
            msg = "Incorrect Email/password"
            return render_template('login.html', msg=msg)
    else:
            return render_template('login.html')
```

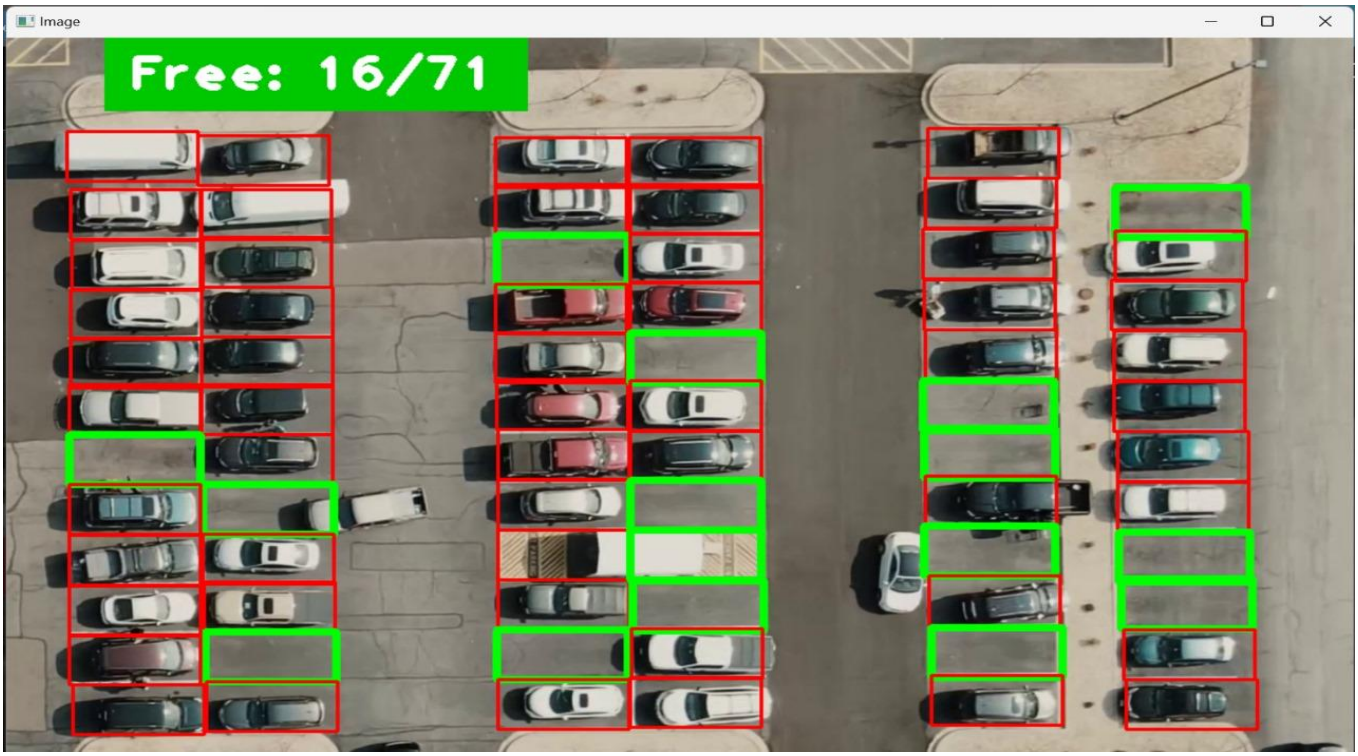
8. PERFORMANCE TESTING

8.1 Performance Metrics

S.No.	Parameter	Values
1.	Model Summary	**http://127.0.0.1:5000/**
2.	Accuracy	Training Accuracy - 93.4893
3.	Confidence Score (Only Yolo Projects)	Class Detected -NA Confidence Score - NA

9. RESULTS

9.1 Output Screenshots



10. 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.

11. CONCLUSION

The primary goal of this study is to optimise the process of locating available parking spots in order to reduce parking lot congestion. Cars may now use reasonably priced automated parking systems to locate available spaces in parking lots thanks to advancements in machine learning and vision-based technologies. Future research might focus on allocating certain parking spaces to clients who have already registered with an online parking management system.It is announced that the proposed algorithm's accuracy is 92.The results show that

productivity reduces and spotting accuracy declines when images of the parking lot are taken in poor illumination or with overlapping objects. It is observed that 99.5 is the average performance.The extreme accuracy of Convert RGB picture frames to grayscale image Execute Adjustment Obtain a parking space equivalent to Obtain parking spaces that are separated into blocks. Block conversion to inverted binary To calculate the automatic number of free and reserved blocks, get the value of the associated locality. Entry Real-time video recording 1313 The suggested work also depends on the type of camera that is used to monitor the parking lot.

12. FUTURE SCOPE

● Hook up a webcam to a snort Pi and have live parking monitoring at home

● Alchemize parking lot video to have overview perspective(for clearer globules)

● It's effective at resolving parking issues. In addition, it provides automatic billing, as well as eliminating traffic congestion. Utilizing a multilevel parking technique, this work can be further developed into a fully automated system.

● The system presents the details of vacant parking areas nearby, and reduces the market problems related to illegal parking in the area. It was intended to meet the requirements of controlled parking that offers downhill parking techniques to the authorities

13. APPENDIX

Source Code

```
import os
os.add_dll_directory(r"D:\programs\Python programs\smart internz\AI car Parking\venv\Lib\site-packages\clidriver\bin")


from flask import Flask, render_template, request, session
import cv2
```

```
import pickle
import cvzone
import numpy as np
import re
import ibm_db


app = Flask(_name_)
app.secret_key = 'your_secret_key'


conn=
ibm_db.connect("DATABASE=vali3;HOST=localhost;PORT=2500
0;PROTOCOL=TCPIP;
print("connected")


@app.route('/')
def project():
        return render_template('Login.html')
@app.route('/login')
def login():
        return render_template('Login.html')
@app.route('/signup')
def signup1():
        return render_template('Signup.html')
@app.route('/index')
def index():
        return render_template('Index.html')
@app.route('/home')
def Home():
        return render_template('Index.html')
@app.route('/model')
def model():
```

```
        return render_template('Model.html')
@app.route('/about')
def about():
        return render_template('About.html')
@app.route('/contact')
def contact():
        return render_template('Contact.html')



@app.route("/reg", methods=["POST","GET"])
def signup():
        msg = ' '
        if request.method == 'POST':
        name = request.form["name"]
        email = request.form["email"]
        password = request.form["password"]
        sql = "SELECT * FROM vali3.register WHERE name = ?"
        stmt = ibm_db.prepare(conn, sql)
        ibm_db.bind_param(stmt, 1, name)
        ibm_db.execute(stmt)
        account = ibm_db.fetch_assoc(stmt)
        print (account)
        if account:
        return render_template('Signup.html', error=True)
        elif not re.match(r'[^@]+@[^@]+\.[^@]+', email):
        msg = "Invalid Email Address!"
        else:
        insert_sql = "INSERT INTO vali3.register VALUES(?,?,?)"
        prep_stmt = ibm_db.prepare(conn, insert_sql)
        # this username & password should be same as db-2
details & order also
        ibm_db.bind_param(prep_stmt, 1, name)
        ibm_db.bind_param(prep_stmt, 2, email)
```

```
        ibm_db.bind_param(prep_stmt, 3, password)
        ibm_db.execute(prep_stmt)
        msg = "You have successfully registered !"
        return render_template("Login.html", msg=msg)


@app.route("/log", methods=['POST', 'GET'])
def login1():
        if request.method == "POST":
        email = request.form["email"]
        password = request.form["password"]
        sql = "SELECT * FROM vali3.register WHERE email=? AND password=?" # from db2 sql table
        stmt = ibm_db.prepare(conn, sql)
        # this username & password should be same as db-2 details & order also
        ibm_db.bind_param(stmt, 1, email)
        ibm_db.bind_param(stmt, 2, password)
        ibm_db.execute(stmt)
        account = ibm_db.fetch_assoc(stmt)
        print (account)
        if account:
        session['Loggedin'] = True
        session['id'] = account['EMAIL']
        session['email'] = account['EMAIL']
        return render_template('Index.html')
        else:
        msg = "Incorrect username/password"
        return render_template('Login.html', msg=msg)
        else:
        return render_template('Login.html')


@app.route('/predict_live')
```

```
def liv_pred():
    # Video feed
    cap = cv2.VideoCapture('carParkingInput.mp4')
    with open('parkingSlotPosition', '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, 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)
        if cv2.waitKey(10) & 0xFF== ord('q'):
        break

if _name=="__main_":
        app.run(debug=True)
```

GitHub & Project Demo Link

https://github.com/smartinternz02/SI-GuidedProject-612310-1698643569

https://drive.google.com/file/d/1nTMGCd5esNnVkkzpVIue_DCvdvEBJbo2/view?usp=sharing