**PROJECT REPORT**

**AI ENABLED CAR PARKING USING OPENCV**

**DEPARTMENT OF**

  Electronics and Communication Engineering

## Avanthi Institute of Engineering & Technology

Cherukupally (P),Bhogapuram (M), Near Tagarapuvalasa, NH16,

Kotabhogapuram, Andhra Pradesh 535006

***Submitted by***

* ***TEAM ID:*** ***LTVIP2023TMID04661***
* **KOLA LEELA KUMAR (SBAP0004661)**
* **VURITI VINAY KUMAR (SBAP0004705)**
* **DUDA DEEPAK (SBAP0004645)**
* **ANUPOJU SATYA AMAN (SBAP0004632)**
* **MATTA BHARAT KUMAR (SBAP0004671)**

CONTENTS:

* **INTRODUCTION**
  + Project Overview
  + Purpose
* **IDEATION & PROPOSED SOLUTION**
  + Problem Statement Definition
  + Empathy Map Canvas
  + Ideation & Brainstorming
  + Proposed Solution
* **REQUIREMENT ANALYSIS**
  + Functional requirement
  + Non-Functional requirements
* **PROJECT DESIGN**
  + Data Flow Diagrams
  + Solution & Technical Architecture
  + User Stories
* **CODING & SOLUTIONING (Explain the features added in the project along with code)**
  + Feature 1
  + Feature 2
  + Database Schema (if Applicable)
* **RESULTS**
  + Performance Metrics
* **ADVANTAGES & DISADVANTAGES**
* **CONCLUSION**
* **FUTURE SCOPE10. APPENDIX**
* Source Code
* GitHub & Project Video Demo Link

## INTRODUCTION

### Project Overview

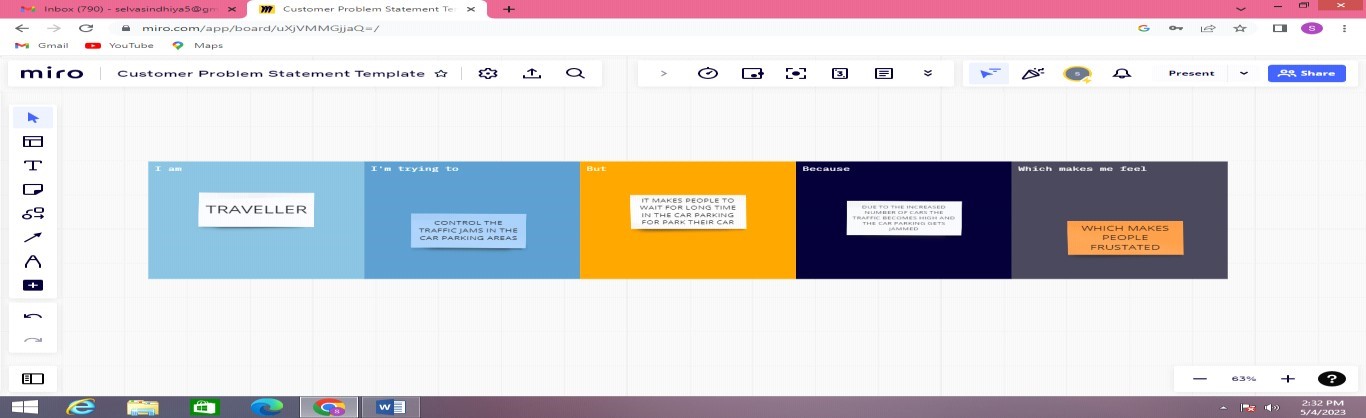
* Most AI-enabled car parking systems parking a car today are still managed by hand. There is no automated monitoring system in place to keep track of how much capacity each parking place contains. In order to find an empty spot, drivers often have to make a circuitous trip through the parking lot. Where there are more people than parking spots, such problems are especially common near hospitals, malls, schools, and other large gathering places.

### 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.

## IDEATION & PROPOSED SOLUTION

### 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**

Traveller

PS-1

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

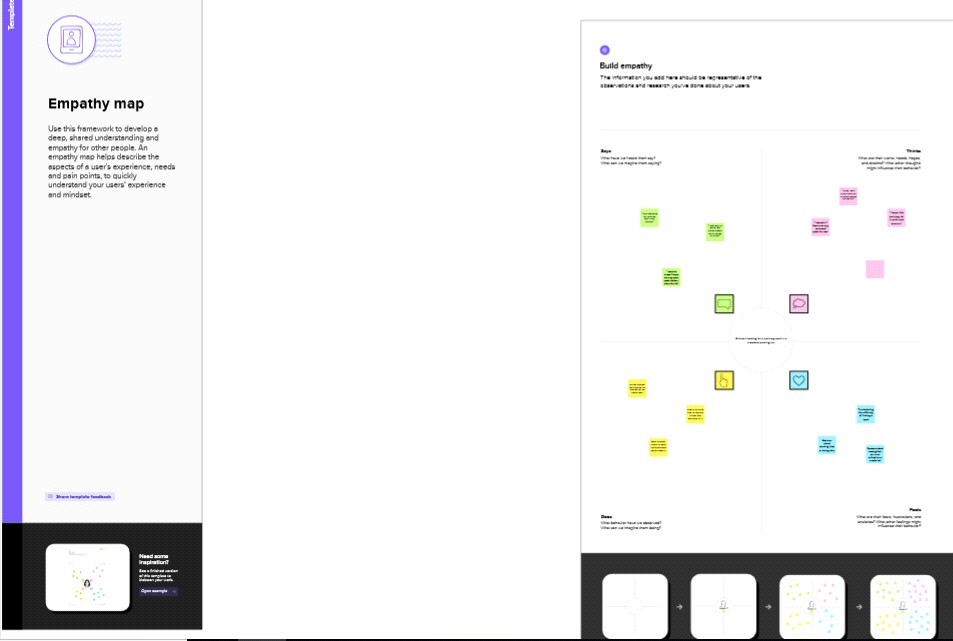
more time

Increased

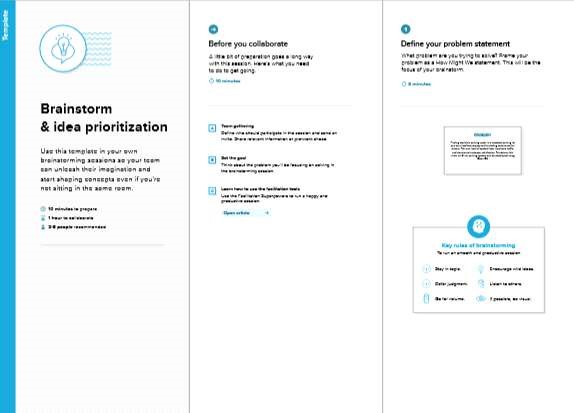
traffic

Frustrated

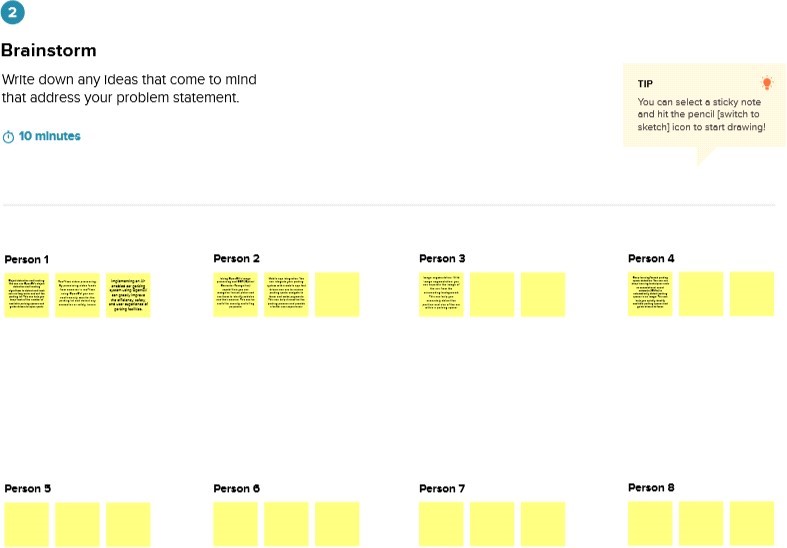
### Empathy Map Canvas

* **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:**
* 

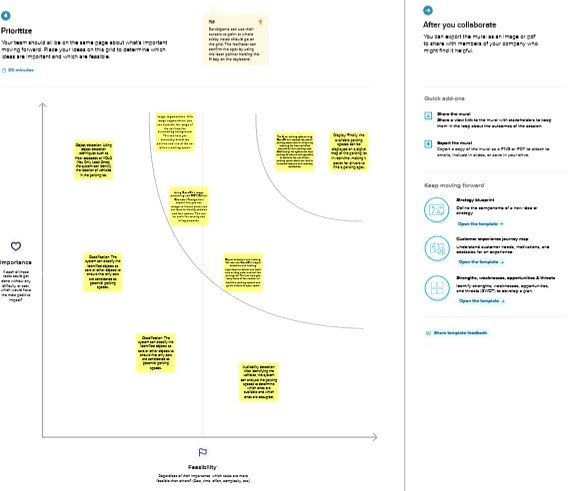
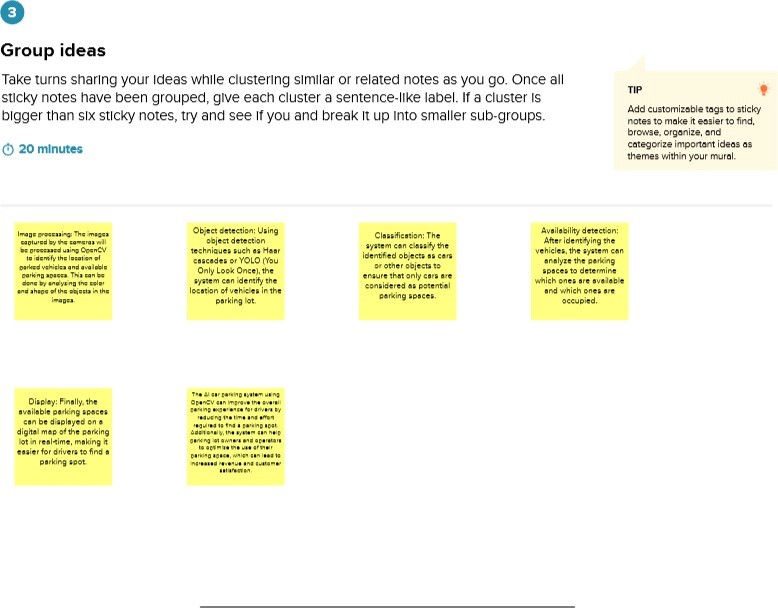
### Ideation & Brainstorming

* **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.
* Reference: <https://www.mural.co/templates/empathy-map-canvas>**Step-1: Team Gathering, Collaboration and Select the Problem Statement**
* 

## Step-2: Brainstorm, Idea Listing and Grouping

* 

## Step-3: Idea Prioritization



### Proposed Solution

* Proposed Solution Template:
* Project team shall fill the following information in proposed solution template.
* S.No. Parameter Description
* Problem Statement (Problem to be solved)
* Traditional car parking systems are inefficient and time-consuming. Drivers often have to spend a lot of time searching for an empty parking space, leading to congestion in the parking lot. Additionally, traditional parking systems require manual intervention for ticketing and payment processing, which can lead to long queues and delays.
* Idea / Solution description The idea behind an AI-enabled car parking system using OpenCV is to leverage the power of computer vision to automate the parking process. By using cameras and image processing algorithms, the system can detect the presence of cars in the parking lot and guide drivers to empty parking spaces. The system can also automate the ticketing and payment processing, making the process faster and more convenient for drivers.
* Novelty / Uniqueness The use of computer vision technology for car parking systems isa novel approach that can significantly improve the efficiency and convenience of parking. By using OpenCV, the system can accurately detect and track cars, even in crowded parking lots. Additionally, the use of AI algorithms can optimize the parking process by analyzing parking patterns and occupancy rates
* Social Impact / Customer Satisfaction An AI-enabled car parking system usingOpenCV can have a significant social impact by reducing traffic congestion and improving the overall driving experience. By guiding drivers to empty parking spaces, the system can reduce the time spent searching for parking and minimize traffic congestion in the parking lot. Additionally, by automating the ticketing and payment process, the system can improve the convenience and efficiency of the parking process.
* Business Model (Revenue Model) The business model for an AI-enabled car parkingsystem can be based on a pay-per-use model, where drivers pay a fee based on the duration of the parking. The system can be integrated with a payment gateway to enable online payments, making the process faster and more convenient for drivers. Additionally, the system can generate revenue by collecting data on parking patterns and occupancy rates, which can be used by parking lot operators to optimize the parking process.
* Scalability of the Solution An AI-enabled car parking system using OpenCV can be easily scaled to accommodate large parking lots and multiple parking locations. By using a centralized database and cloud-based architecture, the system can easily handle large volumes of data and support multiple users. Additionally, the system can be customized based on the specific requirements of the parking lot, making it adaptable to different environments and use cases.

## REQUIREMENT ANALYSIS 3.1Functional 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

Calculate parking fee based on parking duration

FR-3

Payment and billing

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

* **3.2Non-Functional requirements Non-functional Requirements:**
* Following are the non-functional requirements of the proposed solution.
* **FR Non-Functional Requirement Description No.**
* 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.

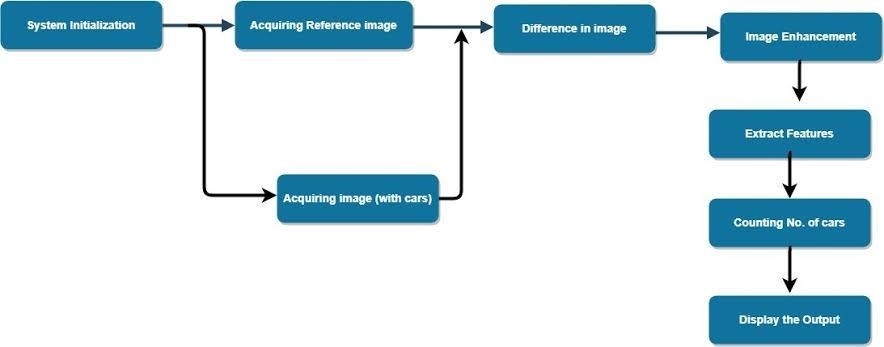
* NFR-5 **Availability** The system should be compatible with different

types of vehicles, including cars, trucks, and motorcycles, and be able to accommodate different sizes of vehicles.

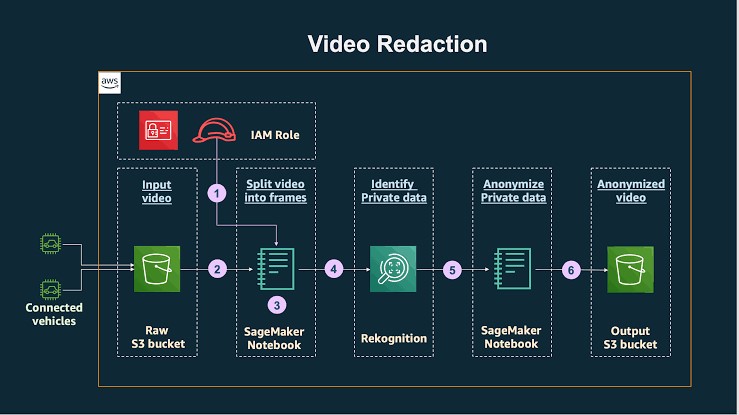
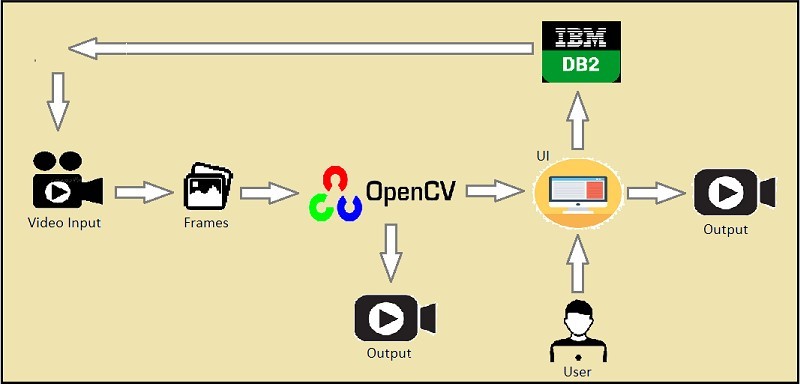
* NFR-6 **Scalability** The system should be able to handle a large

number of parking spaces, and be easily scalable to accommodate additional spaces in the future.

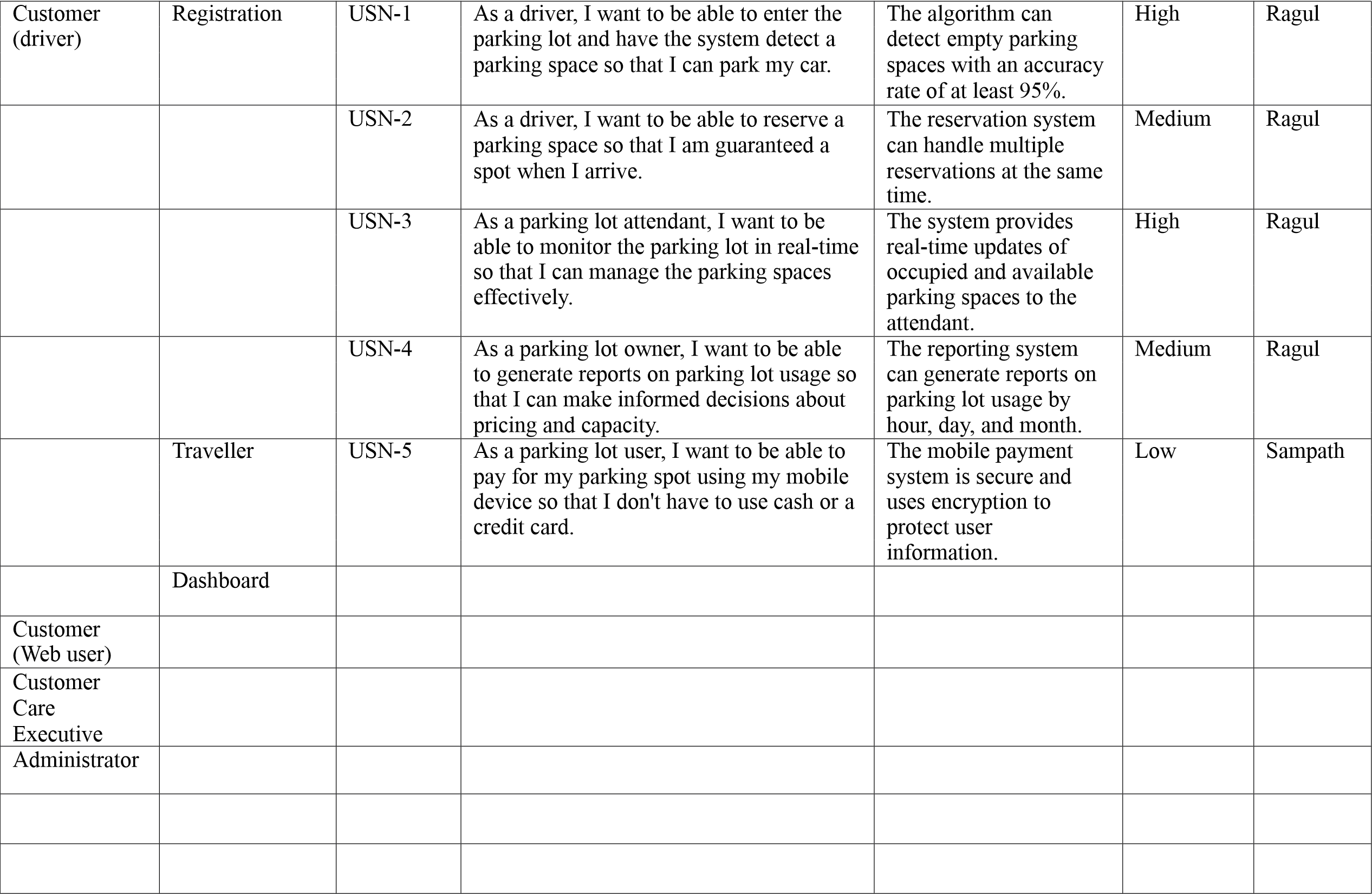
## PROJECT DESIGN 4.1Data Flow Diagrams

* **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)**
* **+-------------+ +-------------+ +-------------+**
* **| Camera | | Image | | Parking |**
* **| Sensor | Images | Analysis | Spots | Status |**
* **+-------------+ +-------------+ +-------------+**
  + **| | |**
  + **| | Parking Status |**
  + **| Image Analysis | Update |**
  + **| | |**
* **+-------------+ +-------------+ +-------------+**
* **| OpenCV AI |<----------| Server |<----------| User/ |**
* **| Library | | Processing | | Operator |**
* 

## 4.2Solution & 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.**

## 4.3User Stories

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

## CODING & SOLUTIONING (Explain the features added in the project along with code) 5.1Feature 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.

## 5.2Feature 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.
* **RESULTS**

## 6.1Performance 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 realtime. 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.

## 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.

## 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 automaticbilling, as well as eliminating traffic congestion. Utilising 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

## APPENDIX:

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, f'Free: {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

cv2.imshow("Image", img)

# cv2.imshow("Imagery", imgThres)

# cv2.imshow("ImageBlur",

imgBlur) key = cv2.waitKey(1)

if key ==

ord('r'): pass

Parking space picker py

import cv2

import pickle

width, height = 107,

48 try:

with open('CarParkPos', '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))

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('CarParkPos', '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)

cv2.imshow("Image", img)

cv2.setMouseCallback("Image", mouseClick)

cv2.waitKey(1)

## CONCLUSION

* This study's main beneficence is to perfect the unearthing of open parking spaces in an expenditure to ease parking arena slowdown. The development of machine learnedness and vision- grounded technology has made it possible for motorcars to find open spaces at parking lots using affordable automatic parking systems. unborn studies can concentrate on assigning specific emplacements to customers who have afore registered with an online parking management system. The precision about the proposal algorithm is inaugurated to be 92.The outcomes demonstrates that, when the captured photos of the parking lot aren`t clear due to low lighting or overlaps, the productivity drops and the exactitude for spotting decreases. It’s noticed that the average performance is 99.5 and is remarkably high as contrasted with other parking lot finding out procedures. The effectiveness of the proposed method in some cases drops down due to the strong darkness. The ultra precision of Get image frames RGB to Gray image Do Calibration Get equals of parking spot Get fellows of car Parking spot divided into Blocks Convert Block to inverse binary Get value of connected locality to determine autos number of free and Reserved Blocks Input Live stream recording 1313 the proposed task additionally relies on the kind of camera utilized for covering the parking lot.

**OUTPUT:**





**GitHub & Project Video Demo Link**

https://github.com/Abhijustin441/AI\_enabled\_carparking\_using\_opencv