INTRODUCTION

1.1 Project Overview

An aim of the project is to decide a vehicle's fitness for parking by considering the length and height of the vehicle and relating the vehicle with its classes to check the exactness of the outcome. These days,there are many methods used in identifying the parking vehicles in parking lots as listed in references. A camera is used as a sensor for video image recognition. This is due to its competence and realization cost. The related project that used camera for video image recognition have presented . This project applies the edge detection with boundary condition technique for image detecting module which is used as point detection with canny operator method. In that we can use moving vehicles as a reference image to spot the parking lot. This proposed project is smart car parking using deep learning. This project is very useful to time save and know the parking space availabilities, using camera we can accurately monitor while compared to sensor based project.

1.2 Purpose

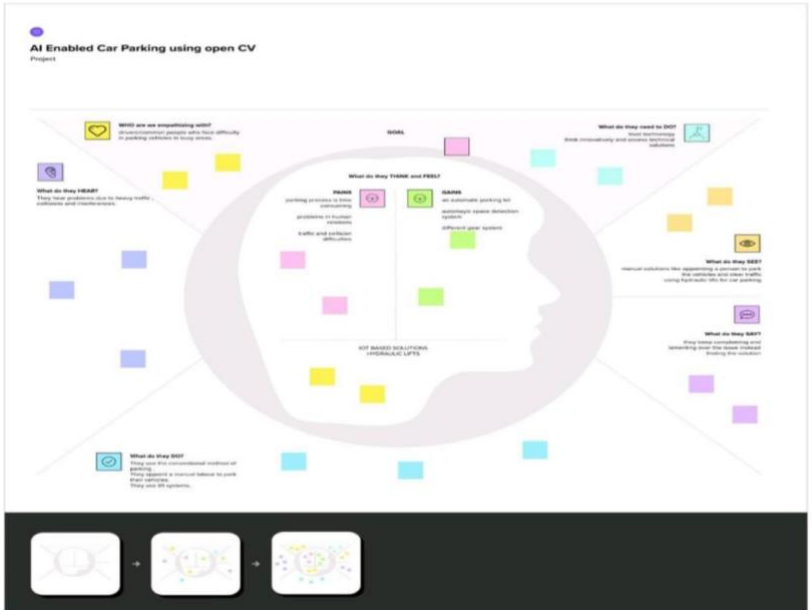
System designed to minimize the area and/or volume required for parking cars. Like a multi-story parking garage, an APS provides parking for cars on multiple levels stacked vertically to maximize the number of parking spaces while minimizing land usage . Artificial Intelligence prevents several drivers from being guided to one available parking space at the same time.

2. IDEATION & PROPOSED SOLUTION

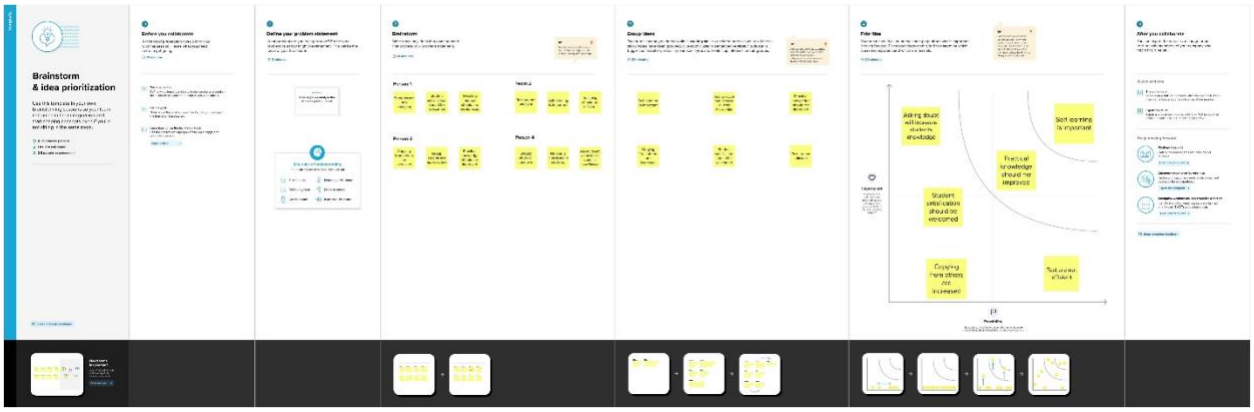
2.1 Problem Statement Definition

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.

2.2 Empathy Map Canvas



2.3 Ideation & brainstorming



2:4 Proposed Solution

Project Design Phase-I
Proposed Solution Template

Date	09 May 2023
Team ID	NM2023TMID01151
Project Name	AI Enabled car parking using open CV

Proposed Solution Template:

S.No.	Parameter	Description
1.	Problem Statement	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.
2.	Idea / Solution description	AI-based smart parking is an innovative parking solution that leverages data from different devices like sensors and cameras to form an AI-driven parking management system. These devices are either embedded into the parking lot or placed in close proximity to the parking lot to detect the availability of parking spots.
3.	Novelty / Uniqueness	Occupancy detection : Whether a particular parking place is occupied or not (in Real-Time) Statistics on your fingertips : Customized stats as per your requirements on the go. Automatic car license plates recognition : Automatic Car's license plate recognition using efficient OCR integration.
4.	Social Impact / Customer Satisfaction	Controls pollution and vehicle congestion. Convenient, cost-effective and faster mode of parking system. Increased safety and security
5.	Business Model (Revenue Model)	Revenue will be generated from the users of the application when they park their cars. Revenue will be collected from the vendors who deploy our system.
6.	Scalability of the Solution	This could be supplied to various local and international outlets also to governmental organization's for environment monitoring and control. This increases revenue utilisation as well as exposure.

3. REQUIREMENT ANALYSIS

Components required:

- NodeMCU (ESP-8266)
- ARDUINO UNO.
- Ultrasonic Sensor SRF- 04
- Servo Motor SG90
- Infrared Sensor (IR)

Methods used :

- Sensor based Recognition System
- Circle Hough Transformation Method
- Deep Recognition Method

3.4 Functional requirements

- Backend Management System

Authenticate users and admins before modifying any sensitive data and Accept reservation of parking lots based on availability. It also generates a Session ID for each reservation and send it to the user allow modification of parking lot status by operators and Auto-cancel reservation if user fails to reach within the window period.

- Mobile Application for end users

Register for the service and enter personal and vehicle details. To find a parking area from the list of areas, registered by parking admins. To view the details of a selected parking area such as the name, price per minute, number of total available lots.

- Mobile Application for parking operator

Send vehicle plate number and reservation password (Session ID) to central server for verification when users check in.

- Web Application for parking admin

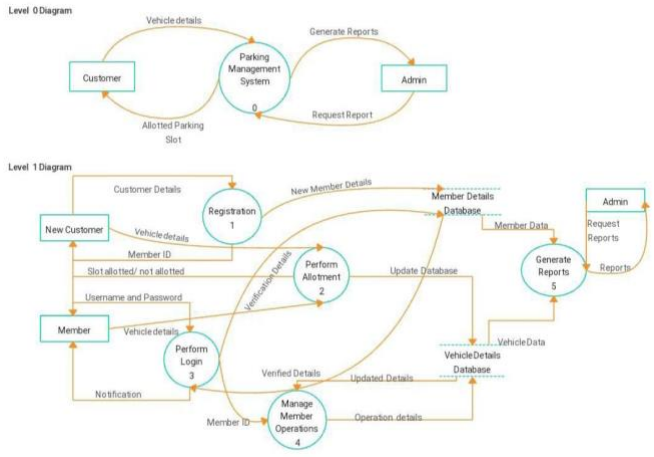
Defines new parking areas, specify number of parking lots, the parking cost per minute/hour and other details. It Modifies data of existing parking areas and to View the data of all registered parking areas.

1.2 Non-Functional requirements

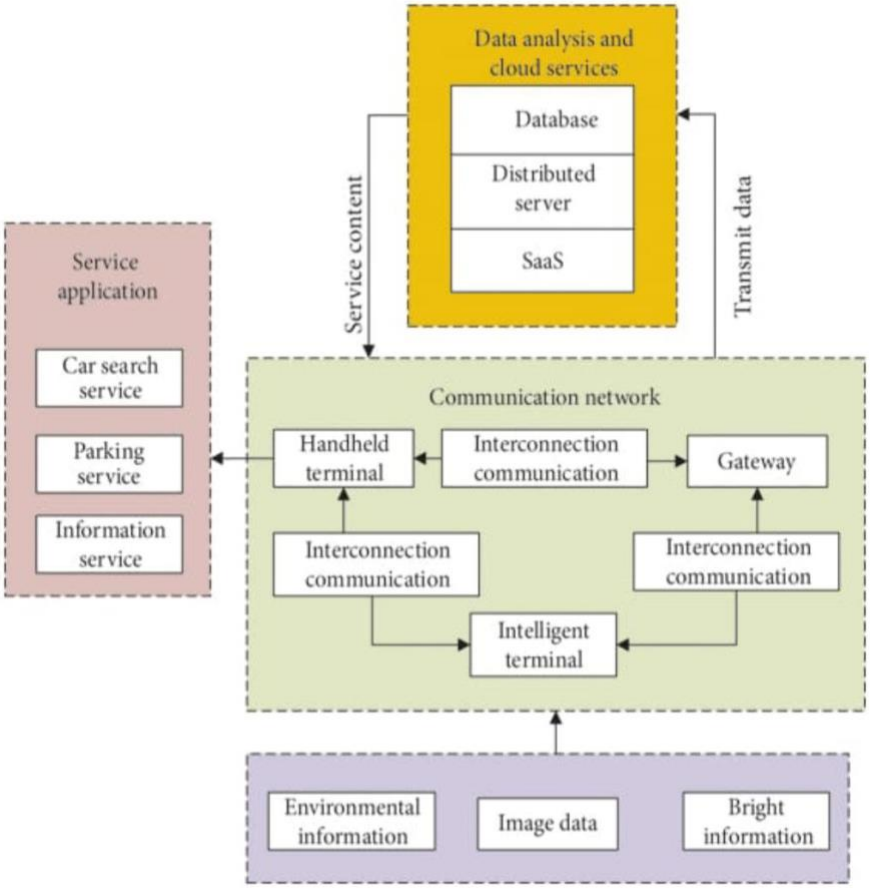
- The central repository should be platform independent so that it can be accessible and store application data via the web application and the mobile app.
- The server should be able to handle concurrent requests from different users.
- The system should provide confidentiality for user data using database encryption and local encryption to protect data in the event of device theft (laptop/handheld device).
- The android application provides high availability and high accuracy in finding the locations.

4. PROJECT DESIGN

4.1 Data Flow Diagrams



4.2 Solution & Technical Architecture



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

5.1 Feature 1

Color

```
COLOR_BLACK = (0, 0, 0)
COLOR_BLUE = (0, 0, 255)
COLOR_GREEN = (0, 255, 0)
COLOR_RED = (255, 0, 0)
COLOR_WHITE = (255, 255, 255)
```

5.2 Feature 2

Coordinates

```
Import cv2 as open_cv
Import numpy as np
From colors import COLOR_WHITE
From drawing_utils import draw_contours
Class CoordinatesGenerator:
   KEY_RESET = ord(“r”)
   KEY_QUIT = ord(“q”)
   Def __init__(self, image, output, color):
      Self.output = output
      Self.caption = image
      Self.color = color
      Self.image = open_cv.imread(image).copy()
      Self.click_count = 0
      Self.ids = 0
      Self.coordinates = []
      Open_cv.namedWindow(self.caption, open_cv.WINDOW_GUI_EXPANDED)
      Open_cv.setMouseCallback(self.caption, self.__mouse_callback)
   Def generate(self):
      While True:
         Open_cv.imshow(self.caption, self.image)
         Key = open_cv.waitKey(0)
         If key == CoordinatesGenerator.KEY_RESET:
            Self.image = self.image.copy()
         Elif key == CoordinatesGenerator.KEY_QUIT:
            Break
      Open_cv.destroyWindow(self.caption)
```

```
Def __mouse_callback(self, event, x, y, flags, params):
   If event == open_cv.EVENT_LBUTTONDOWN:
      Self.coordinates.append((x, y))
      Self.click_count += 1
      If self.click_count >= 4:
         Self.__handle_done()
      Elif self.click_count > 1:
         Self.__handle_click_progress()
   Open_cv.imshow(self.caption, self.image)
Def __handle_click_progress(self):
   Open_cv.line(self.image, self.coordinates[-2], self.coordinates[-1], (255, 0, 0), 1)
Def __handle_done(self):
   Open_cv.line(self.image,
            Self.coordinates[2],
            Self.coordinates[3],
            Self.color,
            1)
   Open_cv.line(self.image,
            Self.coordinates[3],
            Self.coordinates[0],
            Self.color,
            1)
   Self.click_count = 0
   Coordinates = np.array(self.coordinates)
   Self.output.write("-\n        id: " + str(self.ids) + "\n        coordinates: [" +
               "[" + str(self.coordinates[0][0]) + "," + str(self.coordinates[0][1]) + "]," +
               "[" + str(self.coordinates[1][0]) + "," + str(self.coordinates[1][1]) + "]," +
               "[" + str(self.coordinates[2][0]) + "," + str(self.coordinates[2][1]) + "]," +
               "[" + str(self.coordinates[3][0]) + "," + str(self.coordinates[3][1]) + "]]\n")
```

Draw_contours(self.image, coordinates, str(self.ids + 1), COLOR_WHITE)

For I in range(0, 4):

Self.coordinates.pop()

Self.ids += 1

5.3 Feature 3

Drawing utils

```
import cv2 as open_cv
from colors import COLOR_RED


def draw_contours(image,
                  coordinates,
                  label,
                  font_color,
                  border_color=COLOR_RED,
                  line_thickness=1,
                  font=open_cv.FONT_HERSHEY_SIMPLEX,
                  font_scale=0.5):
    open_cv.drawContours(image,
                         [coordinates],
                         contourIdx=-1,
                         color=border_color,
                         thickness=2,
                         lineType=open_cv.LINE_8)
    moments = open_cv.moments(coordinates)

    center = (int(moments["m10"] / moments["m00"]) - 3,
              int(moments["m01"] / moments["m00"]) + 3)

    open_cv.putText(image,
                    label,
                    center,
                    font,
                    font_scale,
                    font_color,
                    line_thickness,
                    open_cv.LINE_AA)
```

5.4 Feature 4

Motion Detection

```
Import cv2 as open_cv
Import numpy as np
Import logging
From drawing_utils import draw_contours
From colors import COLOR_GREEN, COLOR_WHITE, COLOR_BLUE
Class MotionDetector:
    LAPLACIAN = 1.4
    DETECT_DELAY = 1
    Def __init__(self, video, coordinates, start_frame):
        Self.video = video
        Self.coordinates_data = coordinates
        Self.start_frame = start_frame
        Self.contours = []
        Self.bounds = []
        Self.mask = []
    Def detect_motion(self):
        Capture = open_cv.VideoCapture(self.video)
        Capture.set(open_cv.CAP_PROP_POS_FRAMES, self.start_frame)
        Coordinates_data = self.coordinates_data
        Logging.debug("coordinates data: %s", coordinates_data)
        For p in coordinates_data:
            Coordinates = self._coordinates(p)
            Logging.debug("coordinates: %s", coordinates)
            Rect = open_cv.boundingRect(coordinates)
            Logging.debug("rect: %s", rect)
            New_coordinates = coordinates.copy()
            New_coordinates[:, 0] = coordinates[:, 0] – rect[0]
```

```
    New_coordinates[:, 1] = coordinates[:, 1] – rect[1]
    Logging.debug(“new_coordinates: %s”, new_coordinates)
    Self.contours.append(coordinates)
    Self.bounds.append(rect)
    Mask = open_cv.drawContours(
        Np.zeros((rect[3], rect[2]), dtype=np.uint8),
        [new_coordinates],
        contourIdx=-1,
        color=255,
        thickness=-1,
        lineType=open_cv.LINE_8)
    mask = mask == 255
    self.mask.append(mask)
    logging.debug(“mask: %s”, self.mask)
statuses = [False] * len(coordinates_data)
times = [None] * len(coordinates_data)
while capture.isOpened():
    result, frame = capture.read()
    if frame is None:
        break
    if not result:
        raise CaptureReadError(“Error reading video capture on frame %s” % str(frame))
    blurred = open_cv.GaussianBlur(frame.copy(), (5, 5), 3)
    grayed = open_cv.cvtColor(blurred, open_cv.COLOR_BGR2GRAY)
    new_frame = frame.copy()
    logging.debug(“new_frame: %s”, new_frame)
    position_in_seconds = capture.get(open_cv.CAP_PROP_POS_MSEC) / 1000.0
    for index, c in enumerate(coordinates_data):
        status = self.__apply(grayed, index, c)
```

```
            if times[index] is not None and self.same_status(statuses, index, status):
                times[index] = None
                continue
            if times[index] is not None and self.status_changed(statuses, index, status):
                if position_in_seconds – times[index] >= MotionDetector.DETECT_DELAY:
                    statuses[index] = status
                    times[index] = None
                continue
            if times[index] is None and self.status_changed(statuses, index, status):
                times[index] = position_in_seconds
        for index, p in enumerate(coordinates_data):
            coordinates = self._coordinates(p)
            color = COLOR_GREEN if statuses[index] else COLOR_BLUE
            draw_contours(new_frame, coordinates, str(p["id"] + 1), COLOR_WHITE, color)
        open_cv.imshow(str(self.video), new_frame)
        k = open_cv.waitKey(1)
        if k == ord("q"):
            break
    capture.release()
    open_cv.destroyAllWindows()
def __apply(self, grayed, index, p):
    coordinates = self._coordinates(p)
    logging.debug("points: %s", coordinates)
    rect = self.bounds[index]
    logging.debug("rect: %s", rect)
    roi_gray = grayed[rect[1]☹rect[1] + rect[3]), rect[0]☹rect[0] + rect[2])]
    laplacian = open_cv.Laplacian(roi_gray, open_cv.CV_64F)
    logging.debug("laplacian: %s", laplacian)
```

```
    coordinates[:, 0] = coordinates[:, 0] – rect[0]
    coordinates[:, 1] = coordinates[:, 1] – rect[1]
    status = np.mean(np.abs(laplacian * self.mask[index])) < MotionDetector.LAPLACIAN
    logging.debug("status: %s", status)
    return status
  @staticmethod
  Def _coordinates(p):
    Return np.array(p["coordinates"])
  @staticmethod
  Def same_status(coordinates_status, index, status):
    Return status == coordinates_status[index]
  @staticmethod
  Def status_changed(coordinates_status, index, status):
    Return status != coordinates_status[index]
Class CaptureReadError(Exception)
```

6. RESULTS

6.1 Performance Metrics

S.No	Methods	Advantages	Disadvanta-ges	Tools used	Data set	Layer	Accuracy
1	Segmentation [3]	Responsiveness to changes in the target output.	Limited time and less resources	OpenCV	8M Segments	Video level	80%
2	Feature extraction [4][5]	Capable of dealing with different kinds of noise	Need to improve flexibility	Tensor Flow	Divere face images	9-layer DNN	75%
3	Image restoration [10][11]	Similar to customized impression	Needs further optimization	PyTorch	Sketch Transfer	Ada -FM	75%
4	Image analysis [12]	Easy for image comparison	Original image can be tampered	EmguCV	ImageNet -A	WCS layer	70%
5	Image compression [18][19]	High compression ratio	Needs further optimization	VXL	Waymo Open Dataset	Raster layer	80%

7. ADVANTAGES & DISADVANTAGES

Advantages:

- Cost effective
- Sensors and other cost intensive hardware are not employed.
- No infrastructure maintenance involved.
- Environment Friendly
- The cloud uses energy in a more streamlined and efficient way than traditional, in-house data centers
- Uses multi-tenant architecture and this tends to be more Efficient than the typical, single-tenant, statically-allocated data centers
- Highly available -The system will be available to the end users provided they have an internet connection and the smart parking mobile Application
- Easy to use
- Highly scalable
- Robust

Disadvantages

- Errors in providing accuracy of locations and occupancy
- Not a centralized system.
- This system is considerably expensive.
- Availability
- Stable internet connection

8. CONCLUSION

Due to advancement in technology, drivers are demanding easier and less time-consuming parking facilities. There are various methodologies of smart parking that have been implemented to provided better services to the end users and improve the overall management of the existing parking system. The real time monitoring of available parking lots and Allotment of the suitable parking area by advanced reservation.

10. REFERENCES

[1] Z. Pala and N. Inanc, “Smart parking applications using RFID technology” in 1st Annual Eurasia RFID Conference, September 2007.

[2] Wand and W. He, “A reservation based smart parking System” in 1st Int.‟l Workshop on Cyber-Physical networking systems, April 2011.

[3] N.H.H.M. Hanif, M.H. Badiozaman and H. Daud, “Smart parking reservation system using short message services (SMS)”, in 2010 International Conference on Intelligent and Advanced Systems (ICIAS), June 2010.

9. FUTURE SCOPE

- The global smart parking market size is expected to grow from USD 24,329.6 million in 2020 to USD 95,059.9 million by 2027, at a CAGR of 25.5% from 2021 to 2027. The growth of the global smart parking market is majorly driven by increase in energy demand due to the rising population.
- The proposed Smart Parking System in this research is an addition on existing smart parking systems by introducing actual smart features, such as preventing drivers to hit-and-run, occupying multiple parking spaces and parking in spaces for people with special needs.

10. APPENDIX

Source code

```
Import argparse

Import yaml

From coordinates_generator import CoordinatesGenerator

From motion_detector import MotionDetector

From colors import *

Import logging

Def main():
   Logging.basicConfig(level=logging.INFO)
   Args = parse_args()
   Image_file = args.image_file
   Data_file = args.data_file
   Start_frame = args.start_frame
   If image_file is not None:
      With open(data_file, “w+”) as points:
         Generator = CoordinatesGenerator(image_file, points, COLOR_RED)
         Generator.generate()
   With open(data_file, “r”) as data:
      Points = yaml.load(data)
      Detector = MotionDetector(args.video_file, points, int(start_frame))
      Detector.detect_motion()
Def parse_args():
```

```
    Parser = argparse.ArgumentParser(description='Generates Coordinates File')
    Parser.add_argument("—image",
                        Dest="image_file",
                        Required=False,
                        Help="Image file to generate coordinates on")
    Parser.add_argument("—video",
                        Dest="video_file",
                        Required=True,
                        Help="Video file to detect motion on")
    Parser.add_argument("—data",
                        Dest="data_file",
                        Required=True,
                        Help="Data file to be used with OpenCV")
    Parser.add_argument("—start-frame",
                        Dest="start_frame",
                        Required=False,
                        Default=1,
                        Help="Starting frame on the video")
    Return parser.parse_args()
If __name__ == '__main__':
    Main()
```

GitHub Link

https://github.com/olgarose/ParkingLot/tree/master/parking_lot/videos